



# **COSMO-RS Manual**

*Amsterdam Modeling Suite 2026.1*

**[www.scm.com](http://www.scm.com)**

**Apr 02, 2026**



# CONTENTS

<b>1</b>	<b>General</b>	<b>1</b>
1.1	Introduction . . . . .	1
1.2	What's new in COSMO-RS 2026.1 . . . . .	2
1.3	What's new in COSMO-RS 2025.1 . . . . .	3
1.4	What's new in COSMO-RS 2024.1 . . . . .	3
1.5	What's new in COSMO-RS 2023.1 . . . . .	3
1.6	What's new in COSMO-RS 2022.1 . . . . .	3
1.7	What's new in COSMO-RS 2021.1 . . . . .	3
1.8	What's new in COSMO-RS 2020.1 . . . . .	3
<b>2</b>	<b>COSMO-RS, COSMO-SAC, UNIFAC</b>	<b>5</b>
2.1	COSMO-RS theory . . . . .	5
2.1.1	COSMO-RS combinatorial term . . . . .	7
2.1.2	Fast approximation for COSMO-RS calculations . . . . .	8
2.1.3	Temperature dependent hydrogen bond interaction . . . . .	8
2.2	COSMO-SAC 2013-ADF, 2016-ADF, DHB-ADF . . . . .	8
2.2.1	COSMO-SAC 2013-ADF . . . . .	9
2.2.2	COSMO-SAC 2016-ADF . . . . .	9
2.2.3	COSMO-SAC DHB-ADF . . . . .	9
2.2.4	COSMO-SAC DHB-MESP . . . . .	9
2.2.5	ADF Implementation of the COSMO-SAC Model . . . . .	10
2.3	UNIFAC theory . . . . .	10
2.3.1	Residual term . . . . .	11
2.3.2	Combinatorial term . . . . .	11
2.4	Calculation of properties . . . . .	12
2.4.1	Properties derived from pseudochemical potential in solution . . . . .	14
2.4.2	Liquid-Liquid equilibrium properties . . . . .	14
2.4.3	Vapor-Liquid equilibrium properties . . . . .	14
2.4.4	Solid-Liquid equilibrium properties . . . . .	15
<b>3</b>	<b>COSMO result files</b>	<b>19</b>
3.1	ADF COSMO calculation . . . . .	19
3.1.1	ADF COSMO settings . . . . .	19
3.1.2	Atomic cation or anion . . . . .	21
3.1.3	Accuracy . . . . .	21
3.1.4	Cavity construction . . . . .	22
3.2	ADFCRS-2018 Database . . . . .	24
3.2.1	ADFCRS-2010 . . . . .	25
3.2.2	ADFCRS-IL-2014 . . . . .	26
3.3	ADFCRS-POLYMERS-2019 Database . . . . .	26

3.4	FastSigma: a QSPR method to estimate COSMO sigma-profiles	27
3.4.1	Introduction	27
3.4.2	Input options	27
3.4.3	GUI Input	28
3.4.4	Examples	28
3.5	MOPAC COSMO calculation	33
<b>4</b>	<b>The COSMO-RS program</b>	<b>35</b>
4.1	Running the COSMO-RS program	35
4.2	COSMO-RS and COSMO-SAC parameters	35
4.2.1	COSMO-RS general parameters	35
4.2.2	COSMO-RS element specific parameters	37
4.2.3	COSMO-SAC general parameters	37
4.2.4	COSMO-SAC element specific parameters	40
4.2.5	Technical and accuracy parameters	41
4.3	Compounds	42
4.4	Temperature	44
4.5	Pressure	44
4.6	Molar fractions and mass fraction	44
4.7	Properties	44
4.7.1	Vapor pressure	45
4.7.2	Boiling point	45
4.7.3	Solvent flash point	46
4.7.4	Partition coefficients (LogP)	46
4.7.5	Activity coefficients solvent and solute	47
4.7.6	Solubility	47
4.7.7	Phase Stability Test	48
	Advanced algorithm settings	48
	Example	50
4.7.8	Liquid-Liquid Equilibrium (LLE)	51
	Advanced algorithm settings	51
	Example	55
4.7.9	Binary mixture (VLE/LLE)	55
4.7.10	Ternary mixture (VLE/LLE)	56
4.7.11	Solvents s1 - s2 Composition Line	56
4.8	Analysis	57
4.8.1	Sigma profile	57
4.8.2	Sigma potential	57
4.9	Other inputs	58
4.9.1	Pitzer-Debye-Hückel long-range electrostatic correction	58
4.9.2	Elbro Combinatorial term for polymers	58
4.9.3	Reuse generated sigma profile	59
4.9.4	Update generated sigma profile	59
<b>5</b>	<b>Polymers with COSMO-RS(-SAC)</b>	<b>61</b>
5.1	Additional properties/units for polymer systems	62
5.1.1	Average molecular weight	62
5.1.2	Mole fractions of the monomers and polymers	63
	x(monomer)	63
	x(polymer)	63
5.1.3	Weight- and volume-fraction activity coefficients	63
5.1.4	Flory-Huggins parameter	64
5.2	Modified combinatorial term	64
5.3	General application guidelines/warnings	65

5.4	Downloading the ADFCRS-POLYMER-2019 database . . . . .	65
5.5	Tutorial on polymer calculations . . . . .	65
<b>6</b>	<b>Pitzer-Debye-Hückel long-range electrostatic correction</b>	<b>67</b>
6.1	Mixing rules and required property inputs for the PDH term . . . . .	67
6.1.1	Molecular weight . . . . .	67
6.1.2	Density . . . . .	67
6.1.3	Dielectric constant . . . . .	68
6.2	Derivation of the PDH term for general mixtures . . . . .	68
6.3	Tutorial on using the PDH correction . . . . .	69
<b>7</b>	<b>The UNIFAC program</b>	<b>71</b>
7.1	Compound Input . . . . .	71
7.1.1	Basic Input . . . . .	71
7.1.2	Physical Property Input . . . . .	72
7.2	Program Input . . . . .	72
7.2.1	List of possible input flags . . . . .	72
7.2.2	Examples of general program flags . . . . .	73
	-t . . . . .	73
	-temperature . . . . .	73
	-steps . . . . .	73
	-preset . . . . .	74
7.3	Templates . . . . .	74
7.3.1	ACTIVITYCOEF . . . . .	74
7.3.2	LOGP . . . . .	74
7.3.3	PURESOLUBILITY . . . . .	75
7.3.4	SOLUBILITY . . . . .	75
7.3.5	PUREVAPORPRESSURE . . . . .	76
7.3.6	VAPORPRESSURE . . . . .	76
7.3.7	BINMIXCOEF . . . . .	77
7.3.8	TERNARYMIX . . . . .	77
<b>8</b>	<b>Pure compound property prediction</b>	<b>79</b>
8.1	Introduction . . . . .	79
8.2	Available properties . . . . .	80
8.3	Running the Property Prediction program . . . . .	81
8.4	Index of property keys . . . . .	83
8.5	General warnings . . . . .	84
8.6	Equations for temperature-dependent properties . . . . .	85
8.6.1	<b>VPM1</b> : liquid vapor pressure . . . . .	85
8.6.2	<b>empirical-VIS1</b> : Liquid Viscosity . . . . .	85
8.7	References . . . . .	85
<b>9</b>	<b>Solvent Optimization</b>	<b>87</b>
9.1	Introduction . . . . .	87
9.2	Problem types . . . . .	87
9.3	Running the Solvent Optimization program . . . . .	88
9.4	Examples . . . . .	90
9.4.1	Solubility . . . . .	90
9.4.2	Liquid-liquid extraction . . . . .	91
9.5	Guidelines for difficult problems . . . . .	93
9.6	Differences from standard implementations . . . . .	94
<b>10</b>	<b>Python Scripting with COSMO-RS</b>	<b>95</b>
10.1	pyCRS: a python wrapper for thermodynamic calculation . . . . .	95

10.1.1	General Information	95
10.1.2	Overview	95
	Database	96
	CRSManager	98
10.1.3	API	103
	Database	103
	CRSManager	117
	PropPred	123
	FastSigma	125
	Input	126
	Output	126
	Molecule	127
10.2	Python scripting for COSMO-RS with PLAMS	129
10.2.1	General Information	129
10.2.2	Generating COSMO result file	129
10.2.3	Executing the code from the command line	129
10.2.4	Specifying a problem type	130
10.2.5	Inputting Compounds	130
	User-provided compound files	131
10.2.6	Specifying mole fractions, temperatures, and pressures	131
10.2.7	Running jobs	132
10.2.8	Reading the results of a job	132
10.2.9	Plotting results	134
10.3	Examples for pyCRS/PLAMS	135
10.3.1	pyCRS : Basic usage for Database and CRSManager	135
	Adding compound to a pyCRS database	135
	Activity coefficient calculation	137
	Solid Solubility calculation	137
	Establishment of a pyCRS database from the ADFCRS-2018 database	139
10.3.2	pyCRS : Conformer usage for Database and CRSManager	139
	Generating multiple conformers for a compound	139
	Adding conformers to the database	140
	Activity coefficient calculation considering conformers	141
10.3.3	pyCRS : Basic usage for PropPred and FastSigma	143
	Basic usage	143
	Temperature-dependent properties	144
	Estimating multiple properties	145
	Calculating sigma profiles with all models	147
10.3.4	Examples using PLAMS	148
	Partition coefficient	148
	Binary mixture	150
	Solid solubility	152
10.3.5	Advanced scripting examples with PLAMS/pyCRS	153
	Changing the default parameters or re-parameterizing the COSMO-RS/-SAC methods	153
	Solubility screening for solid solute	157
	Screening for cocrystals	162
	Regression of NRTL parameters	167
	Automated screening of ionic liquids	176
	Calculating and estimating sigma profiles	183
	Sigma Moments	186
	Eutectic systems	192
	Binodal and Spinodal Curves	195
	Distribution of species in multispecies calculations	203
	Automated pKa calculation	205

<b>11</b>	<b>Command Line Scripting with COSMO-RS</b>	<b>209</b>
11.1	AMSprep: construct an ADF COSMO results file . . . . .	209
11.2	CRSprep: generate (multiple) COSMO-RS jobs . . . . .	209
11.3	AMSreport: generate report . . . . .	211
11.4	KF utilities for COSMO-RS . . . . .	213
11.4.1	KF browser . . . . .	213
11.4.2	kf2cosmo and cosmo2kf . . . . .	213
11.4.3	pkf, cpkf, dmpkf, udmpkf . . . . .	213
11.5	Scripting Examples . . . . .	214
11.5.1	Example: COSMO result files . . . . .	214
11.5.2	Example: COSMO-RS parameters and analysis . . . . .	215
11.5.3	Example: COSMO-RS properties . . . . .	216
11.5.4	Example: The COSMO-RS compound database . . . . .	224
11.5.5	Example: pKa values . . . . .	228
11.5.6	Example: Polymers . . . . .	229
<b>12</b>	<b>Required Citations</b>	<b>231</b>
12.1	General References . . . . .	231
12.2	Solvent Optimizations . . . . .	231
12.3	External programs and Libraries . . . . .	231
<b>13</b>	<b>Keywords</b>	<b>233</b>
<b>14</b>	<b>FAQ</b>	<b>235</b>
14.1	I want to include solvent effects, do I need COSMO-RS? . . . . .	235
14.2	Do I need to use the ADF module with COSMO-RS? . . . . .	235
14.3	Can I use a .cosmo file from COSMOlogic, Dmol3, or Gaussian with ADF COSMO-RS? . . . . .	235
14.4	For which compounds do you have sigma profiles in the database? . . . . .	236
14.5	I have COSMO-RS download permissions but I can not automatically download the COSMO-RS databases? . . . . .	236
	<b>Python Module Index</b>	<b>237</b>
	<b>Index</b>	<b>239</b>



## 1.1 Introduction

The ADF COSMO-RS (COnductor like Screening MOdel for Realistic Solvents) program is a program that can be used for calculating thermodynamic properties of (mixed) fluids. The COSMO-RS method was developed by Klamt and coworkers<sup>1,2,3</sup>. On the basis of the framework of COSMO-RS, Lin and Sandler<sup>6</sup> suggested a variation, the COSMO-SAC (where SAC denotes segment activity coefficient) model. There are different implementations of COSMO-RS and COSMO-SAC or derivatives, and different parametrizations. The implementation of COSMO-RS in ADF is described in Ref.<sup>4</sup>, which is based on the COSMO-RS method developed by Klamt et al.<sup>2</sup>. The implementation of COSMO-SAC 2013-ADF in ADF is based on the COSMO-SAC model developed by Xiong et al.<sup>7</sup>. The implementation of COSMO-SAC 2016-ADF in ADF is based on the COSMO-SAC model developed by Hsieh et al.<sup>5</sup>, but the parameters in COSMO-SAC 2016-ADF were optimized by Chen et al., like in<sup>8</sup>, for use with ADF. The implementation of COSMO-SAC DHB-ADF in ADF is based on the COSMO-SAC-DHB model developed by Chen et al.<sup>9</sup>, but the parameters were reoptimized by Chen et al. for use with ADF. The implementation of COSMO-SAC DHB-ADF MESP is based on the model developed by Chang et al.<sup>11</sup>.

An alternative to COSMO-SAC or COSMO-RS based methods is the UNIFAC (UNIQUAC Functional-group Activity Coefficients) method, which was developed by Fredenslund, Jones, and Prausnitz, see<sup>10</sup>. The empirical UNIFAC method is a group contribution based method to predict activity coefficients and other thermodynamic properties, in which the group specific parameters are parametrized against a large data base. The implementation in ADF uses so called original UNIFAC parameters.

Our COSMO-RS capabilities are summarized on the product page (<https://www.scm.com/product/cosmo-rs>).

<sup>1</sup> A. Klamt, *Conductor-like Screening Model for Real Solvents: A New Approach to the Quantitative Calculation of Solvation Phenomena*. J. Phys. Chem. 99, 2224 (1995) (<https://doi.org/10.1021/j100007a062>)

<sup>2</sup> A. Klamt, V. Jonas, T. Bürger and J.C. Lohrenz, *Refinement and Parametrization of COSMO-RS*. J. Phys. Chem. A 102, 5074 (1998) (<https://doi.org/10.1021/jp980017s>)

<sup>3</sup> A. Klamt, *COSMO-RS From Quantum Chemistry to Fluid Phase Thermodynamics and Drug Design*, Elsevier. Amsterdam (2005), ISBN 0-444-51994-7.

<sup>6</sup> S.T. Lin and S.I. Sandler, *A Priori Phase Equilibrium Prediction from a Segment Contribution Solvation Model*, Ind. Eng. Chem. Res. 41, 899 (2002) (<https://doi.org/10.1021/ie001047w>)

<sup>4</sup> C.C. Pye, T. Ziegler, E. van Lenthe, J.N. Louwen, *An implementation of the conductor-like screening model of solvation within the Amsterdam density functional package. Part II. COSMO for real solvents*. Can. J. Chem. 87, 790 (2009) (<https://doi.org/10.1139/V09-008>)

<sup>7</sup> R. Xiong, S.I. Sandler, R.I. Burnett, *An improvement to COSMO-SAC for predicting thermodynamic properties*, Ind. Eng. Chem. Res. 53, 8265 (2014) (<https://doi.org/10.1021/ie404410v>)

<sup>5</sup> C.M. Hsieh, S.I. Sandler, S.T. Lin, *Improvements of COSMO-SAC for vapor-liquid and liquid-liquid equilibrium predictions*, Fluid Phase Equilib. 297, 90 (2010) (<https://doi.org/10.1016/j.fluid.2010.06.011>)

<sup>8</sup> W.L. Chen, C.M. Hsieh, L. Yang, C.C. Hsu, S.T. Lin, *A Critical Evaluation on the Performance of COSMO-SAC Models for Vapor-Liquid and Liquid-Liquid Equilibrium Predictions Based on Different Quantum Chemical Calculations*, Ind. Eng. Chem. Res. 55, 9312 (2016) (<https://doi.org/10.1021/acs.iecr.6b02345>)

<sup>9</sup> W.L. Chen, S.T. Lin, *Explicit consideration of spatial hydrogen bonding direction for activity coefficient prediction based on implicit solvation calculations*, Phys.Chem.Chem.Phys. 19, 20367 (2017) (<https://doi.org/10.1039/c7cp02317k>)

<sup>11</sup> C.C. Chang, W.L. Chen, D.T. Wu, S.T. Lin, *Improved directional hydrogen bonding interactions for the prediction of activity coefficients with COSMO-SAC*, Ind. Eng. Chem. Res. 57, 11229 (2018) (<https://doi.org/10.1021/acs.iecr.8b02493>)

<sup>10</sup> A. Fredenslund, R.L. Jones, and J.M. Prausnitz, *Group-contribution estimation of activity coefficients in nonideal liquid mixtures*, AIChE Journal 21, 1086 (1975) (<https://doi.org/10.1002/aic.690210607>)

With COSMO-RS it is possible to use a thermodynamically consistent combinatorial contribution to the chemical potential as is used in Ref.<sup>3</sup>, and a temperature dependent hydrogen bond interaction, also described in Ref.<sup>Page 1, 3</sup>. The parameters in the paper<sup>Page 1, 2</sup> were reparametrized for ADF, see Ref.<sup>Page 1, 4</sup> for details.

The parameters in COSMO-SAC 2013-ADF, COSMO-SAC 2016-ADF, COSMO-SAC DHB-ADF and COSMO-SAC DHB-ADF MESP were optimized for use with ADF COSMO result files. Other COSMO-SAC parameter sets exist that were optimized for different QM packages.

The ADF COSMO-RS (and COSMO-SAC) command line program is called *crs*. The main authors of this program are Cory Pye (Saint Mary's University, Halifax NS Canada) and Jaap Louwen (Albemarle Corporation). COSMO-SAC 2013-ADF was implemented in collaboration with R. Xiong and R.I. Burnett (Sandler group, University of Delaware, Newark, USA). Previous COSMO-SAC methods were implemented by Erin McGarrity (TU Delft, the Netherlands). The COSMO-RS GUI *AMScrs* contains an input builder for COSMO-RS and can visualize results, see the [COSMO-RS GUI tutorials](#) and the [COSMO-RS GUI reference manual](#).

COSMO-RS (and COSMO-SAC) use the intermediate results from quantum mechanical (QM) calculations on individual molecules to predict thermodynamic properties of mixtures of these molecules, for example, solubility. There are a fair number of reports of accurate prediction by COSMO-RS of thermodynamic properties in general in the literature. Many of these have been written by Klamt and co-workers, see Ref.<sup>Page 1, 3</sup> and references therein. Instead of a relatively expensive QM calculation one can use a fast Quantitative Structure-Property Relationship (QSPR) method to estimate the so called COSMO sigma-profile of a molecule that is needed in COSMO-RS (and COSMO-SAC) calculations.

There are also empirical methods like UNIFAC that can predict thermodynamic properties (including the activity coefficients). These methods contain group specific parameters and are parametrized against a large data base. They will often do better than COSMO-RS or COSMO-SAC methods (especially, of course, if the system of interest was part of the data base used for parameter estimation). An advantage of these methods is that they require no QM calculations to be done in order to provide an estimate of thermodynamic properties. However, these methods cannot handle every type of molecule. In particular when unusual combinations of functional groups occur (such as in drug molecules), no parametrization is available. COSMO-RS and COSMO-SAC methods, on the other hand, only feature general parameters not specific to chemical groups or functionalities. All that is required is that a quantum mechanical calculation can be done on the molecule. Therefore, COSMO-RS or COSMO-SAC can be a valuable tool for the prediction of chemical engineering thermodynamical properties, like, for example, partial vapor pressures, solubilities, and partition coefficients. An additional advantage of COSMO-RS and COSMO-SAC over empirical methods is that the molecules dissolved may in fact be transition states of a chemical reaction. This follows from the fact that all that is required is that one can do a QM calculation on the solute and QM on a transition state has become standard in the last two decades. This affords a unique opportunity to predict the thermodynamics of a reaction including, for instance, the balance between kinetically and thermodynamically favored reaction pathways as a function of the solvent used.

## 1.2 What's new in COSMO-RS 2026.1

- *Michelsen Tangent Plane Distance (TPD) stability test* (page 48) is implemented for multicomponent systems.
- *Liquid-Liquid Equilibrium* (page 50) is implemented for multicomponent systems.
- *Reuse Sigma Profile* (page 59) is implemented to reduce computational cost.

## 1.3 What's new in COSMO-RS 2025.1

- *COSMO-SAC DHB MESP* (page 9) is implemented, with the hydrogen bond center determined from the local minima of the molecular electrostatic potential

## 1.4 What's new in COSMO-RS 2024.1

- A new *Database* (page 96) module in pyCRS designed for the management of coskf files
- A new *CRSManager* (page 98) module in pyCRS facilitates a streamlined scripting process for the creation and execution of CRSJob in conjunction with the Database module

## 1.5 What's new in COSMO-RS 2023.1

- *pyCRS* (page 95) is a new python module that provides an interface to the pure compound property prediction and sigma profile prediction tools
- SG1: a new *sigma profile prediction* (page 27) method that relies on a database of pre-computed molecular sub-graphs
- A new, temperature-dependent pure compound liquid viscosity model
- A synthetic accessibility estimation method – see the *Property prediction documentation* (page 77) for more details.
- Improved convergence criteria that make multispecies calculations more stable

## 1.6 What's new in COSMO-RS 2022.1

- Pitzer-Debye-Hückel long-range electrostatic correction

## 1.7 What's new in COSMO-RS 2021.1

- Improved handling of compounds with multi-species components

## 1.8 What's new in COSMO-RS 2020.1

- Support for compounds with multi-species components
  - conformers
  - dimers, trimers, ...
  - dissociation
  - association



## COSMO-RS, COSMO-SAC, UNIFAC

### 2.1 COSMO-RS theory

Below some of the COSMO-RS theory is explained, but a more complete description can be found in Refs.<sup>1</sup> and<sup>2</sup>.

Although in principle all of chemistry can be predicted by appropriate solutions of the Schrödinger Equation, in practice due to the extreme mathematical complexity of doing so only the smallest systems can be computed at an accuracy rivaling that of the most accurate experiments. However, with suitable approximations, for isolated molecules of up to a few hundred atoms these days quite reasonable results can be obtained. Of course, this means that direct computation of thermodynamic properties is out of reach. Thermodynamic properties can only be computed as an average over a large number of configurations of a large number of molecules. To address this, people have typically resorted to so-called Molecular Dynamics (MD) or Monte Carlo (MC) methods where configurations are generated either by numerically simulating the atomic motions over discrete time steps or by random generation, in either case using empirical molecular models parametrized against quantum mechanical calculations and experimental data to compute energies. However, even these approaches often fall short in generating sufficiently large ensembles, and there is little chance of that situation improving dramatically in the near future.

Around 1995, Andreas Klamt, then working for Bayer, hit upon an approach that made it possible to compute the details of molecules quantum mechanically and subsequently use these details in an approximate statistical mechanics procedure<sup>3</sup>. This approach is called COSMO-RS (CONductor like Screening MOdel for Realistic Solvents) and has proven to be quite powerful. It may currently be the best link between the world of chemical quantum mechanics and engineering thermodynamics.

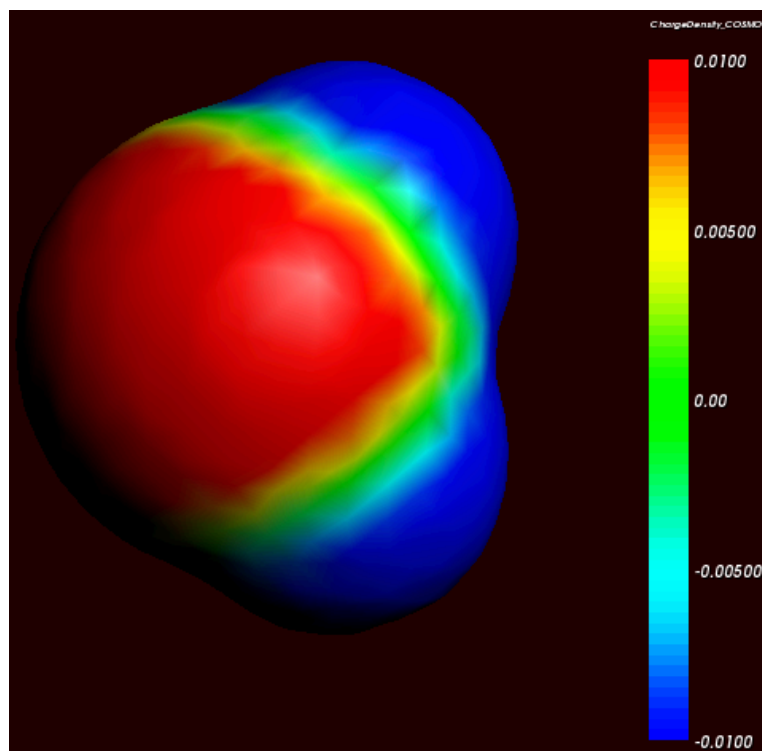
Thermodynamic reference states can be chosen arbitrarily. They do not even have to be physically realizable, as long as it is consistently used. We are at liberty to choose as reference state a molecule embedded in a perfect conductor, that is a material with an infinitely large dielectric constant ('the perfectly screened state'). Suppose a molecule A resides in a molecule shaped cavity. Everywhere outside of this cavity is conductor material. Although it would be hard to realize this in practice, it is relatively easy to do quantum mechanical calculations on this hypothetical state. Since the molecule will in general have a charge distribution and therefore possess an electric field, it will polarize the embedding medium. That will result in another electric field, given by a charge distribution on the surface of the molecule shaped cavity. This charge distribution is generated by the quantum mechanical calculations, for example with ADF if one uses COSMO. From now on the surface of the molecule shaped cavity will be called molecular surface, and the volume of the molecule shaped cavity will be called molecular volume.

---

<sup>1</sup> A. Klamt, V. Jonas, T. Bürger and J.C. Lohrenz, *Refinement and Parametrization of COSMO-RS*. *J. Phys. Chem. A* 102, 5074 (1998) (<https://doi.org/10.1021/jp980017s>)

<sup>2</sup> A. Klamt, *COSMO-RS From Quantum Chemistry to Fluid Phase Thermodynamics and Drug Design*, Elsevier. Amsterdam (2005), ISBN 0-444-51994-7.

<sup>3</sup> A. Klamt, *Conductor-like Screening Model for Real Solvents: A New Approach to the Quantitative Calculation of Solvation Phenomena*. *J. Phys. Chem.* 99, 2224 (1995) (<https://doi.org/10.1021/j100007a062>)



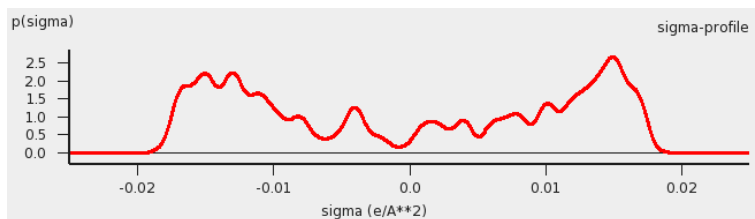
Cosmo charge density on the COSMO surface of water (picture made with AMSview).

Although the actual charge distribution on the molecular surface will be highly detailed, let us for the moment consider the molecular surface as consisting of segments with a constant charge density (i.e. the detailed charge distribution averaged over segments). Now instead of the single molecule A consider, as an arbitrary example, a fluid consisting of three types of molecules: A, B and C. In a fluid not too close to the critical temperature, the molecular surfaces present in the fluid will all be in close contact. That means that the segments of constant density introduced above are in close contact.

We now compare our molecule A in the fluid with our chosen reference state. Any segment of the molecular surface with a charge density of  $\sigma_i$  will be aligned with a segment with charge density  $\sigma_j$  of another molecule. If the two charge densities happen to be opposite (i.e.  $\sigma_i + \sigma_j = 0$ ) the charges required for achieving the perfectly screened state will vanish. However, this will not happen too often and in general an excess charge density is left of  $\sigma_i + \sigma_j$  between the two segments. From electrostatic theory it follows that this introduces an energy penalty proportional to the segment size and  $(\sigma_i + \sigma_j)^2$ . In principle this gives a way to compute the chemical potential of component A, by going over all possible conformations of a large number of molecules A, B and C (in their proper molar fractions) and do computations on the statistical ensemble. However, in practice that would be similar to doing Molecular Dynamics calculations using empirical structure models and about as computationally prohibitive. Instead, an approximation can be made that is not easily justified a priori and must be judged by the results of subsequent simulations. This assumption is that all segments in the fluid are able to make contact independent of one another. In a way it can be said, that the segments are cut loose from the original (rigid) molecular surfaces.

As one would guess, the approximation of independent segments makes the mathematics of computing ensemble properties quite tractable. In fact, computing the chemical potential of component A (or B or C) in the mixture by means of the COSMO-RS and related methods takes in the order of seconds on a normal PC (given the results of quantum mechanical calculations that may have taken days, of course). Note that the molecular surface around the molecule is divided rather arbitrarily in segments and that the assumption was that the segment of one molecule will overlap perfectly with that of another. How can this be true? The answer is that one can split up the molecular surface into segments in an infinite number of ways. However, the molecules in a fluid are always in contact with another. At any given time, molecule A will be in contact with a number of other molecules and share patches of, for example, 7 square Angstroms of its surface with each of the surrounding molecules. At that particular time, the segments will be those patches. A split second later, of course, there will be a different set of segments. That is not a problem. One needs to do statistical

mechanics with charged segments for which one needs to know how many 7 square Angstrom segments a particular molecule brings into the fluid and the probability of any segment having an average charge density  $\sigma$  (for all values of  $\sigma$ ). Both can be computed from the results of the quantum mechanical calculation on the molecule in the perfect conductor. Just to get a flavor, in the figure below the so-called  $\sigma$ -profile of water is given. These are the statistical distributions of possible segments over charge densities multiplied by the surface area of the molecular volume. The  $\sigma$ -profile relates to the detailed charge distribution on the molecular surface.



$\sigma$ -profile of water (picture made with the CRS-GUI), smoothed curve, Delley COSMO surface construction

In principle vapor pressures of pure liquids can be computed directly with COSMO-RS. COSMO-RS calculations yield the chemical potential of a component in a liquid with respect to the perfectly screened reference state. It is easy to compute the energy difference between the reference state and the gas phase by doing an additional quantum mechanical calculation (of the isolated molecule). However, often experimental vapor pressures for the pure liquid are known. Using such experimental data for pure liquids can help in predicting the correct partial vapor pressures in a mixture.

### 2.1.1 COSMO-RS combinatorial term

In Ref.<sup>Page 5, 1</sup> a thermodynamically inconsistent combinatorial contribution  $\mu_i^{comb}$  to the chemical potential was used:

$$\mu_i^{comb} = -\lambda RT \ln(q_{av}/\text{\AA}^2)$$

$$q_{av} = \sum_i x_i q_i$$

In this equation  $q_i$  is the surface area of the molecular volume of compound  $i$ ,  $x_i$  is the molar fraction of compound  $i$  in the solution, and  $\lambda$  is a COSMO-RS parameter.

The importance of using a thermodynamically consistent combinatorial contribution is discussed in Ref.<sup>Page 5, 2</sup>. In the ADF COSMO-RS program it is possible to use a thermodynamically consistent combinatorial contribution of the form (Equation C.4 of Ref.<sup>Page 5, 2</sup>, with  $\lambda_0 = \lambda_1 = \lambda_2 = \lambda$ ):

$$\mu_i^{comb} = \lambda RT (1 - r_i/r_{av} + \ln(r_i/r_{av}) + 1 - q_i/q_{av} - \ln(q_{av}/\text{\AA}^2))$$

$$r_{av} = \sum_i x_i r_i$$

In this equation  $r_i$  is the molecular volume of compound  $i$ . In the ADF COSMO-RS program this combinatorial term is used by default, see also Ref.<sup>4</sup>.

<sup>4</sup> C.C. Pye, T. Ziegler, E. van Lenthe, J.N. Louwen, *An implementation of the conductor-like screening model of solvation within the Amsterdam density functional package. Part II. COSMO for real solvents.* *Can. J. Chem.* 87, 790 (2009) (<https://doi.org/10.1139/V09-008>)

## 2.1.2 Fast approximation for COSMO-RS calculations

In the 1998 COSMO-RS model each segment of the molecular surface has a charge density of  $\sigma_v$ , but also a second charge density  $\sigma_v^\perp$ , which is a descriptor for the correlation between the charge density on the segment with its surrounding. In the original ADF COSMO-RS implementation this was treated as a 2-dimensional problem, in the fast approximation this is effectively reduced to 1-dimension by averaging the  $\sigma_v^\perp$  for each  $\sigma_v$ . Starting from COSMO-RS 2010 this fast approximation is adopted. This approximation reduces the computation time, especially in cases of more than 1 compound.

## 2.1.3 Temperature dependent hydrogen bond interaction

In Ref.<sup>Page 5, 2</sup> a temperature dependent hydrogen bond interaction is suggested, which is used by default in the ADF COSMO-RS program. The temperature dependence (Equation 6.2 of Ref.<sup>Page 5, 2</sup>) is of the form:

$$\begin{aligned} term(T) &= T \ln[1 + \exp(20 \text{ kJ/mol} / RT) / 200] \\ f_{hb}(T) &= term(T) / term(298.15 \text{ K}) \end{aligned}$$

Note that here the correct formula is used with a plus sign before 20 kJ/mol (there is a sign error in Equation 6.2 of Ref.<sup>Page 5, 2</sup>, see online ‘List of Errata in the COSMO-RS book’ by Andreas Klamt), such that this factor goes to zero for large T. In this equation R is the gas constant and T the temperature (in Kelvin). In the ADF COSMO-RS program the hydrogen bond interaction of Ref.<sup>Page 5, 1</sup> is multiplied by this factor  $f_{hb}(T)$  to make the hydrogen bond interaction temperature dependent.

The complete set of parameters used in COSMO-RS model can be found at the *COSMO-RS program* (page 35) section.

The governing equations for various property calculations are summarized in the *Calculation of Properties* (page 12) section.

## 2.2 COSMO-SAC 2013-ADF, 2016-ADF, DHB-ADF

On the basis of the framework of COSMO-RS, Lin and Sandler<sup>1</sup> suggested a variation, the COSMO-SAC (where SAC denotes segment activity coefficient) model by invoking a necessary thermodynamic consistency criterion. Although there are differences, COSMO-RS and COSMO-SAC share some similarities. Over time, several improvements have been made to the COSMO-SAC model, particularly in the treatment of electrostatic interaction and hydrogen bonding interaction:

- In Ref.<sup>2</sup>, a Gaussian-type probability for determining the hydrogen bonding  $\sigma$ -profile ( $P^{hb}(\sigma)$ ) is introduced. This version is called here the COSMO-SAC 2007 method.
- In Ref.<sup>3</sup>, temperature dependence is introduced into the electrostatic interaction, and  $P^{hb}(\sigma)$  is differentiated into  $P^{OH}(\sigma)$  and  $P^{OT}(\sigma)$ . This revised COSMO-SAC model is called here the COSMO-SAC 2010 method.

Note: The parameters of the COSMO-SAC 2007 and COSMO-SAC 2010 methods have not been optimized for use with ADF COSMO result files.

<sup>1</sup> S.T. Lin and S.I. Sandler, *A Priori Phase Equilibrium Prediction from a Segment Contribution Solvation Model*, Ind. Eng. Chem. Res. 41, 899 (2002) (<https://doi.org/10.1021/ie001047w>)

<sup>2</sup> S. Wang, S.I. Sandler, C.C. Chen, *Refinement of COSMO-SAC and the Applications*, Ind. Eng. Chem. Res. 46, 7275 (2007) (<https://doi.org/10.1021/ie070465z>)

<sup>3</sup> C.M. Hsieh, S.I. Sandler, S.T. Lin, *Improvements of COSMO-SAC for vapor-liquid and liquid-liquid equilibrium predictions*, Fluid Phase Equilib. 297, 90 (2010) (<https://doi.org/10.1016/j.fluid.2010.06.011>)

## 2.2.1 COSMO-SAC 2013-ADF

The COSMO-SAC 2013-ADF method used in ADF is the one developed by Xiong et al., which is described in detail in Ref.<sup>4</sup>. The COSMO-SAC 2013-ADF parameters in Ref.<sup>Page 9, 4</sup> were optimized for use with ADF COSMO result files. COSMO-SAC 2013-ADF is an improved COSMO-SAC method compatible to ADF and different than previous COSMO-SAC methods. The main difference compared to previous COSMO-SAC methods is that the COSMO-SAC 2013 model includes a dispersion contribution in the mixture interaction.

## 2.2.2 COSMO-SAC 2016-ADF

In Ref.<sup>5</sup> COSMO-SAC model parameters were optimized by Chen et al. for different quantum mechanical calculations. The authors of Ref.<sup>5</sup> also reoptimized the revised COSMO-SAC model<sup>Page 8, 3</sup> parameters for quantum mechanical calculations with ADF, which will be called here the COSMO-SAC 2016-ADF method.

## 2.2.3 COSMO-SAC DHB-ADF

Instead of differentiating the  $P^{hb}(\sigma)$  into  $P^{OH}(\sigma)$  and  $P^{OT}(\sigma)$ , a directional hydrogen bond (DHB) approach was introduced in Ref.<sup>6</sup>, leading to the development of the COSMO-SAC(DHB) model. This model constrains hydrogen bond segments within a cutoff radius around the hydrogen bond center (hbc), which is determined using VSEPR theory and molecular geometry. The parameters were reoptimized by Chen et al. for use with ADF, which will be called here the COSMO-SAC DHB-ADF method.

## 2.2.4 COSMO-SAC DHB-MESP

In Ref.<sup>7</sup> an improved directional hydrogen bond approach was developed. The new scheme determines the hbc through the local minima of the molecular electrostatic potential (MESP), computed from the Densf calculation. Unlike the VSEPR-based approach, which is limited to chemical species with distinct hybridization characteristics, this new method is applicable to all types of chemical species. This improved approach will be referred to as the COSMO-SAC DHB-ADF MESP method.

---

**Note: The COSMO-SAC DHB-MESP method requires the hydrogen bond center derived from the Densf calculation. To use this model, ensure that:**

- The *ADF-CRS-2018 database* (page 24) is updated.
- The Densf calculation is enable when generating the ADF COSMO result file using the *COSMO-RS Compound Task*.

---

<sup>4</sup> R. Xiong, S.I. Sandler, R.I. Burnett, *An improvement to COSMO-SAC for predicting thermodynamic properties*, Ind. Eng. Chem. Res. 53, 8265 (2014) (<https://doi.org/10.1021/ie404410v>)

<sup>5</sup> W.L. Chen, C.M. Hsieh, L. Yang, C.C. Hsu, S.T. Lin, *A Critical Evaluation on the Performance of COSMO-SAC Models for Vapor-Liquid and Liquid-Liquid Equilibrium Predictions Based on Different Quantum Chemical Calculations*, Ind. Eng. Chem. Res. 55, 9312 (2016) (<https://doi.org/10.1021/acs.iecr.6b02345>)

<sup>6</sup> W.L. Chen, S.T. Lin, *Explicit consideration of spatial hydrogen bonding direction for activity coefficient prediction based on implicit solvation calculations*, Phys.Chem.Chem.Phys. 19, 20367 (2017) (<https://doi.org/10.1039/c7cp02317k>)

<sup>7</sup> C.C. Chang, W.L. Chen, D.T. Wu, S.T. Lin, *Improved directional hydrogen bonding interactions for the prediction of activity coefficients with COSMO-SAC*, Ind. Eng. Chem. Res. 57, 11229 (2018) (<https://doi.org/10.1021/acs.iecr.8b02493>)

## 2.2.5 ADF Implementation of the COSMO-SAC Model

The ADF COSMO-RS program can calculate activity coefficients using any of the COSMO-SAC model version described above. Like in the COSMO-RS method, pure compound vapor pressures can be given as input, for example, if experimental values are available. In case of the COSMO-SAC 2013-ADF model, if these values are not specified then the pure compound vapor pressure will be calculated according to the COSMO-SAC 2013-ADF model. However, for all other COSMO-SAC models, if these values are not specified then the pure compound vapor pressures will be approximated using a method similar as in the COSMO-RS method.

The COSMO-SAC 2013 model includes a dispersion contribution in the mixture interaction. This dispersion contribution is a complicated expression which also depends on the liquid molar volume of the pure compounds and on the molar volume of the mixture. The molar volume of the mixture is calculated from the pure compound liquid molar volumes assuming ideal mixing. In the input for the ADF COSMO-RS program one include for each compound the experimental pure compound liquid density (kg/L), from which the program can calculate the pure compound liquid molar volumes. If this density is not given the pure compound liquid molar volume will be calculated from its COSMO volume. Note that in the calculations with the COSMO-SAC 2013-ADF model in Ref.<sup>4</sup> often experimental pure compound liquid molar volumes were used.

The complete set of parameters used in different COSMO-SAC models can be found at the *COSMO-RS program* (page 37) section.

The governing equations for various property calculations are summarized in the *Calculation of Properties* (page 12) section.

## 2.3 UNIFAC theory

Below some of the UNIFAC method is explained, but a more complete description can be found in Ref.<sup>1</sup>.

The UNIFAC method is an activity coefficient model derived from the UNIQUAC model. Both UNIFAC and UNIQUAC are thermodynamic models based on local composition theory, which holds that the local environment of a molecule in solution can be used to calculate the probabilities of molecular configurations in the bulk solution. While UNIQUAC requires parameters for every compound in solution and interaction parameters for every pair of compounds, UNIFAC estimates these parameters as functions of the number of occurrences of various molecular substructures, or groups, in a molecule. This means that UNIFAC can be applied to estimate activity coefficients for arbitrary systems, so long as every group is defined and an interaction parameter exists for every pair of groups in the composition.

The UNIFAC method calculates the activity coefficient as a function of two contributions: (1) the residual contribution, meant to account for the interactions of groups in the mixture; and (2) the combinatorial contribution, meant to account for entropic effects due to differences in molecular shape. Using these two components, the activity coefficient for each compound  $i$  is calculated as follows:

$$\ln \gamma_i = \ln \gamma_i^R + \ln \gamma_i^C$$

where  $\ln \gamma_i^R$  corresponds to the residual contribution to the activity coefficient and  $\ln \gamma_i^C$  to the combinatorial contribution.

---

<sup>1</sup> A. Fredenslund, R.L. Jones, and J.M. Prausnitz, *Group-contribution estimation of activity coefficients in nonideal liquid mixtures*, *AIChE Journal* 21, 1086 (1975) (<https://doi.org/10.1002/aic.690210607>)

### 2.3.1 Residual term

In the UNIFAC method, we first define  $n_i^k$  to be the number of times group  $k$  occurs in molecule  $i$ . Using this with the mole fraction  $x_i$  of each compound  $i$ , we calculate the group mole fraction,  $X_k$ , or the amount of groups of type  $k$  as a fraction of the total groups:

$$X_k = \frac{\sum_i x_i n_i^k}{\sum_i \sum_j x_i n_i^j}$$

We use the  $X_k$  values to then determine the relative surface area each group represents in the mixture. This is done by taking an average of the  $X_k$  values weighted with the surface area contributions,  $Q_k$ , of each group  $k$ . We define the result as the area fraction of group  $k$ :

$$\Theta_k = \frac{X_k Q_k}{\sum_m x_m Q_m}$$

the surface area and volume of a molecule is estimated as a linear function of the number and types of groups that are present. Each group  $k$  occurs a  $n_i^k$  times in molecule  $i$ .

Additionally, between every pair of groups,  $k$  and  $m$  an interaction energy is defined as  $a_{km}$ . This energy is given in Kelvin and is used to calculate the group interaction parameter  $\Psi_{km}$ :

$$\Psi_{km} = \exp\left(\frac{-a_{km}}{T}\right)$$

where  $T$  is the temperature of the system in Kelvin. Note that  $\Psi_{km} \neq \Psi_{mk}$ .

These parameters are then used to calculate the residual contribution to the activity coefficient for each group  $k$ :

$$\ln \Gamma_k = Q_k \left( 1 - \ln \sum_m \Theta_m \Psi_{mk} - \sum_m \frac{\Theta_m \Psi_{km}}{\sum_n \Theta_n \Psi_{nm}} \right)$$

Additionally, one must also calculate  $\ln \Gamma_k^{(i)}$ , which follows the same procedure as above for each compound assuming it exists in a pure form, i.e.,  $x_i = 1$ . Then, the residual contribution to the activity coefficient is calculated as follows:

$$\ln \gamma_i^R = \sum_k n_i^k \left( \ln \Gamma_k - \ln \Gamma_k^{(i)} \right)$$

### 2.3.2 Combinatorial term

To estimate the combinatorial contribution to the activity coefficient, first the surface area,  $q_i$ , and volume,  $r_i$ , of molecule  $i$  are estimated as follows:

$$q_i = \sum_k Q_k n_i^k \quad r_i = \sum_k R_k n_i^k$$

where  $Q_k$  is the surface area contribution of group  $k$  and  $R_k$  is the volume contribution. Using these parameters we can define the relative surface area and relative volume (also called fractional surface area and fractional volume) corresponding to molecule  $i$  in solution. This is simply an average of the surface areas/volumes of each compound weighted by the mole fraction of that compound in solution:

$$\theta_i = \frac{x_i q_i}{\sum_j x_j q_j} \quad \phi_i = \frac{x_i r_i}{\sum_j x_j r_j}$$

Additionally, we calculate the parameter  $L_i$ :

$$L_i = \frac{z}{2}(r_i - q_i) - (r_i - 1)$$

where  $z$  is the coordination number and is usually taken to be equal to 10. All of these parameters are then used to calculate the combinatorial contribution to the activity coefficient:

$$\ln \gamma_i^C = \ln \frac{\phi_i}{x_i} + \frac{z}{2} q_i \ln \frac{\theta_i}{\phi_i} + L_i - \frac{\phi_i}{x_i} \sum_j x_j L_j$$

## 2.4 Calculation of properties

The COSMO-RS method allows to calculate the pseudochemical potential of a compound in the liquid phase ( $\mu_i(T, \mathbf{x})$ ), as well as in the gas phase ( $\mu_i^{gas}(T)$ ), see the previous *COSMO-RS theory* (page 5) section and Ref.<sup>1</sup>. In the ADF COSMO-RS implementation the following equations were used to calculate properties using these pseudochemical potential where  $\mu_i^{pure}$  and  $\mu_i^{solv}$  refers to the  $\mu_i(T, \mathbf{x})$  evaluated at the mole fraction of unity and the specified solvent composition, respectively. The detailed explanation of the variables can be found *here* (page 15).

The following sections will elaborate on the application of these equations, categorized according to the thermodynamic principles:

1. *Properties derived from pseudochemical potential in solution* (page 14)
2. *Liquid-Liquid equilibrium properties* (page 14)
3. *Vapor-Liquid equilibrium properties* (page 14)
4. *Solid-Liquid equilibrium properties* (page 15)

---

<sup>1</sup> Klamt, A.; Jonas, V.; Bürger, T.; Lohrenz, J. C. W. *Refinement and Parametrization of COSMO-RS*. J. Phys. Chem. A 1998, 102 (26), 5074–5085 (<https://doi.org/10.1021/jp980017s>)

Equation	No. [Tutorial]
$\mu_i(T, \mathbf{x}) = \mu_i^{res}(T, \mathbf{x}) + \mu_i^{comb}(T, \mathbf{x})$	(1)
$\mu_i^{gas}(T) = E_i^{gas} - E_i^{COSMO} + \Delta E_i^{diel} - \sum_k \gamma_k A_i^k - \omega n_i^{ring} + \eta RT$	(2)
$\sum_i x_i = \sum_i y_i = \sum_i w_i = 1$	(3)
$w_i = \frac{x_i M_i^{pure}}{\sum_j x_j M_j^{pure}} = \frac{x_i M_i^{pure}}{M^{ave}}$	(4)
$M^{ave} = \sum_i x_i M_i^{pure}$	(5)
$P_i^{vap} = \exp\left(\frac{\mu_i^{pure} - \mu_i^{gas}}{RT}\right)$	(6)
$P_i = x_i \exp\left(\frac{\mu_i^{solv} - \mu_i^{gas}}{RT}\right)$	(7)
$P = \sum_i P_i$	(8)
$y_i = P_i/P$	(9)
$\gamma_i = \exp\left(\frac{\mu_i^{solv} - \mu_i^{pure}}{RT}\right)$	(10)
$a_i = \gamma_i x_i$	(11)
$G^E = H^E - TS^E = \sum_i x_i (\mu_i^{solv} - \mu_i^{pure}) = \sum_i x_i RT \ln(\gamma_i)$	(12)
$H^E = -RT^2 \frac{\partial}{\partial T} \left(\frac{G^E}{RT}\right)$	(13)
$G^{mix} = G^E + RT \sum_i x_i \ln(x_i) = \sum_i x_i RT \ln(x_i \gamma_i)$	(14)
$\Delta_{vap} H = \frac{RT^2}{P} \frac{\partial P}{\partial T}$	(15)
$k_H = \frac{1}{V_{solvent}} \exp\left(\frac{\mu_i^{gas} - \mu_i^{solv}}{RT}\right)$	(16)
$k_H^{cc} = k_H RT$	(17)
$k_H^{px} = \frac{1}{k_H V_{solvent}} = \gamma_i P_i^{vap}$	(18)
$x_i^{SOL(liq)} = \frac{1}{\gamma_i}, \quad T > T_m \quad (\text{insoluble liquid})$	(19)
$x_i^{SOL(solid)} = \frac{1}{\gamma_i} \exp\left\{\frac{\Delta_{fus} H_i}{R} \left(\frac{1}{T_{m,i}} - \frac{1}{T}\right) - \frac{\Delta C_{p,i}}{R} \left(\ln \frac{T_{m,i}}{T} - \frac{T_{m,i}}{T} + 1\right)\right\}, \quad T < T_{m,i}$	(20)
$x_i^{SOL(gas)} = \frac{P_i^{gas}}{x_i \gamma_i P_i^{vap}}$	(21)
$\Delta G_{solv}^{liq-solv} = \mu_i^{solv} - \mu_i^{pure}$	(22)
$\Delta G_{solv}^{gas-solv} = \mu_i^{solv} - \mu_i^{gas} + RT \ln\left(\frac{V_{solvent}}{V_{gas}}\right)$	(23)
$\log_{10} P_{solv1/solv2} = \frac{1}{\ln(10)} \frac{\mu_i^{solv2} - \mu_i^{solv1}}{RT} + \log_{10} \left(\frac{V_{solv1}}{V_{solv2}}\right)$	(24)
$\frac{1}{LFL_{mix}} = \sum_i \frac{y_i}{LFL_i}$	(25)

The above equations are not always exact, for example the vapor-liquid equilibrium properties assume perfect gas behavior.

### 2.4.1 Properties derived from pseudochemical potential in solution

The  $\mu_i^{res}$  represents the electrostatic interaction contribution and the  $\mu_i^{comb}$  accounts for the molecular size and shape difference between species. This pseudochemical potential is essential for predicting various properties in a liquid solution, including the activity coefficient (Eq. 10) and excess properties (Eqs. 12 and 13).

In the COSMO-RS method the volume of 1 molecule in the liquid phase is approximated with the volume of the molecule shaped cavity, that is used in the COSMO calculations. In this way it is possible to calculate the volume of 1 mole of solvent molecules in the liquid phase. However, for properties that depend on such volumes, one can also use (related) experimental data as input data for the calculation.

Starting from ADF2012 the Gibbs-Helmholtz equation is used to calculate the excess enthalpy of a mixture. Previously it was estimated using the misfit and hydrogen bonding energy of the mixture and its pure compounds.

### 2.4.2 Liquid-Liquid equilibrium properties

Please note the pseudochemical potential  $\mu_i$  do not include the  $RT \ln(x_i)$  term in a standard chemical potential,  $\left(\frac{\partial G}{\partial N_i}\right)_{T,P,N_{j \neq i}}$ . As a result, the equation for the partition coefficient (Eq. 24) should be derived from the equality of the standard chemical potential between the two liquid phase, as expressed in Eq. 26, as referenced in Ref.<sup>2</sup>:

$$\mu_i(T, x_i^I) + RT \ln(x_i^I) = \mu_i(T, x_i^{II}) + RT \ln(x_i^{II}) \quad (26)$$

$$\mu_i^{pure} = \mu_i(T, x_i) + RT \ln(x_i) \quad (27)$$

In addition, the solubility of an insoluble liquid in a solvent can be further simplified using Eq. (27) and determined by solving Eq. (19). However in many cases, this assumption is invalid. A more accurate approach is solving the miscibility gap directly from Eq. (26). An example is given for the calculation miscibility gap in the [binary mixture of Methanol and Hexane](#).

### 2.4.3 Vapor-Liquid equilibrium properties

The COSMO-RS and COSMO-SAC 2013-ADF have been parameterized for vapor pressure prediction. COSMO-RS provides a reliable approximation of the pseudochemical potential for a compound in the gas phase (perfect gas with a reference state of 1 bar) as expressed in Eq. (2)<sup>Page 12, 1</sup>. In this semi-empirical equation,  $E_i^{gas}$  is the gas phase bond energy,  $E_i^{COSMO}$  is the bond energy in a perfect conductor,  $\Delta E_i^{diel}$  is the dielectric energy correction, the rest terms are proposed as the dispersion energy, ring correction energy and the energy correction on reference state. The  $\gamma_k$  is the dispersive parameter of atom  $k$ ,  $A_i^k$  is the exposed surface area of atom  $k$ ,  $n_i^{ring}$  is the numbers of ring atoms,  $\omega$  and  $\eta$  are both model parameters.

The vapor pressure can then be estimated using Eq. (6), based on the ideal gas assumption and the equality of the standard chemical potential between the gas and liquid phases (Eq. 28 and Eq. 29). Alternatively, Eq. (29) can be reformulated in terms of the fugacity, as shown in Eq. (30). From this relation, various vapor-liquid equilibrium properties can be determined, such as Henry's constant (Eqs 16, 17 and 18), gas solubility (Eq. 21) and solvation energy (Eq 23).

$$\mu_i^{gas}(T, P) = \mu_i^{gas}(T) + RT \ln(P) \quad (28)$$

$$\mu_i^{gas}(T) + RT \ln(P) + RT \ln(y_i) = \mu_i(T, \mathbf{x}) + RT \ln(x_i) \quad (29)$$

$$y_i P = x_i \gamma_i(T, \mathbf{x}) P_i^{vap}(T) \quad (30)$$

The boiling temperature of a solvent is determined using an iterative method, where the temperature is adjusted until the calculated vapor pressure falls within a certain threshold of the desired pressure.

<sup>2</sup> Hsieh, C.-M.; Wang, S.; Lin, S.-T.; Sandler, S. I. *A Predictive Model for the Solubility and Octanol–Water Partition Coefficient of Pharmaceuticals*. *J. Chem. Eng. Data* 2011, 56 (4), 936–945 (<https://doi.org/10.1021/je1008872>)

Also experimental vapor pressure data for pure compounds can be incorporated as input to enhance the accuracy of these calculations. This approach will corrects  $\mu_i^{gas}$  through Eq (6) and improve the prediction accuracy on vapor-liquid equilibrium properties.

## 2.4.4 Solid-Liquid equilibrium properties

The solid solubility can be calculated from the solid-liquid equilibrium, as expressed in Eq. (31). It can be solved using Eq. (32) and Eq. (33), ultimately leading to Eq. (20) similar to Ref.<sup>2</sup>. Then the solid solubility can be calculation with an iterative method, since the activity coefficient  $\gamma_i$  depends on the molar fraction of this compound. The COSMO-RS method does not predict  $\Delta_{fus}H_i$ ,  $\Delta C_{p,i}$ , or  $T_{m,i}$ . These need to be provide as input data for the solid solubility calculation. When experimental data is unavailable, the *property prediction tool* (page 77) can be used to estimate  $\Delta_{fus}H_i$  and  $T_{m,i}$  while assume  $\Delta C_{p,i} = 0$ .

$$\mu_i(T, x_i) + RT \ln(x_i) = \mu_i^{solid}(T) \quad (31)$$

$$\mu_i^{solid}(T) = \mu_i^{pure}(T) - \Delta_{fus}G_i(T) \quad (32)$$

$$\Delta_{fus}G_i(T) \approx \Delta_{fus}H_i \left(1 - \frac{T}{T_{m,i}}\right) + \Delta C_{p,i} \left\{T - T_{m,i} - T \ln\left(\frac{T}{T_{m,i}}\right)\right\} \quad (33)$$

Table 2.1: Definition of Variables

Quantity	Meaning
$R$	Gas constant
$T$	Temperature
$x_i$	The molar fraction of compound i in a liquid solution
$y_i$	The molar fraction of compound i in the gas phase
$w_i$	The mass fraction of compound i in a liquid solution
$M_i^{pure}$	The molar mass of the pure compound i
$M^{ave}$	The average molar mass of the mixture
$\gamma_i$	Activity coefficient of compound i in a liquid solution
$a_i$	Activity of compound i in a liquid solution
$P_i^{vap}$	The vapor pressure of the pure compound i
$P_i$	The partial vapor pressure of compound i
$P$	The total vapor pressure
$P_i^{gas}$	The specified partial pressure of gaseous solute compound i used in gas solubility calculations
$\mu_i^{gas}$	The pseudochemical potential of the pure compound i in the gas phase
$\mu_i^{pure}$	The pseudochemical potential of the pure compound i in the liquid phase, also as $\mu_i(T, x_i = 1)$
$\mu_i^{solv}$	The pseudochemical potential of compound i in a liquid solution, also as $\mu_i(T, x_i = x^{solv})$
$G^E$	The excess Gibbs free energy
$H^E$	The excess enthalpy, Gibbs-Helmholtz equation
$G^{mix}$	The Gibbs energy of mixing
$\Delta_{vap}H$	The enthalpy of vaporization, Clausius-Clapeyron equation
$E_i^{HB \text{ pure}}$	The hydrogen bond energy of the pure compound i in the liquid phase, see Ref. <a href="#">Page 12, 1</a>
$E_i^{HB}$	The partial hydrogen bond energy of compound i in a liquid solution
$E_i^{misfit \text{ pure}}$	The misfit energy of the pure compound i in the liquid phase, see Ref. <a href="#">Page 12, 1</a>
$E_i^{misfit}$	The partial misfit energy of compound i in a liquid solution
$\Delta G_{solv}^{liq-solv}$	The solvation Gibbs free energy from the pure compound liquid phase to the solvated phase

continues on next page

Table 2.1 – continued from previous page

Quantity	Meaning
$\Delta C_{solv}^{gas-solv}$	The solvation Gibbs free energy from the pure compound gas phase to the solvated phase, with a reference state of 1 mol/L in both phases
$k_H$	Henry's law constant: ratio between the liquid phase concentration of a compound and its partial vapor pressure in the gas phase
$k_H^{cc}$	dimensionless Henry's law constant: ratio between the liquid phase concentration of a compound and its gas phase concentration
$k_H^{px}$ <sub>inv</sub>	Henry's law constant, representing the volatility instead of the solubility, ratio between the partial vapor pressure of a compound in the gas phase and the molar fraction in the liquid phase
$V_{solvent}$	Volume of 1 mole of solvent molecules in the liquid phase
$V_{gas}$	Volume of 1 mole of molecules in the gas phase (at 1 atm, perfect gas)
$x_i^{SOL(state)}$	Solubility of solute compound i in a solvent (molar fraction) where state refers to the physical state of the solute (gas, liquid, or solid)
$\Delta_{fus}H_i$	The enthalpy of fusion of compound i
$\Delta C_{p,i}$	The $\Delta$ heat capacity of fusion of compound i, $C_p^{liq} - C_p^{solid}$
$T_m$	The melting temperature of compound i
$\log_{10}P_{solv1/solv2}$	The logarithm of the partition coefficient P, which is the ratio of the concentrations of a compound in two immiscible solvents, solvent 1 and solvent 2
$LFL_i$	The flash point (lower flammable limit, LFL) of compound i
$LFL_{mix}$	The flash point (lower flammable limit, LFL) of a mixture, Le Chatelier's mixing rule

See also the COSMO-RS GUI tutorial for the calculation of the following properties:

- boiling point of a solvent [a]
- $pK_a$  values [a]
- vapor-liquid diagram binary mixture (VLE/LLE) [a, b]

### Ionic liquids in COSMO-RS 2020

The activity coefficient of a compound i solvated in an ionic liquid is an important thermodynamic property. In COSMO-RS 2020 one can treat the ionic liquid as one compound, which means that the value of the activity coefficient is calculated in the standard way most applications report them. In particular, in COSMO-RS 2020 one can treat the ionic liquid as one compound, which only has the dissociated form.

- [ionic liquids tutorial](#)

### Ionic liquids in COSMO-RS <=2019

The activity coefficient of a compound i solvated in an ionic liquid is an important thermodynamic property. The cation and anion, which have been treated separately, will be used in equal amounts to ensure an electroneutral mixture in the COSMO-RS calculation.

In other applications cation-anion pair have been treated as one molecule, however, below we will treat the cation and anion as two separate molecules, which is needed in older versions of COSMO-RS <=2019. This has consequences for the value of the activity coefficient.

For example, for a 1:1 IL (i.e.,  $[A]^+ [B]^-$ ), the activity coefficient at a finite concentration of solute i in the binary mixture (IL + solute) can be calculated by

$$\gamma_i^{bin} = (\gamma_i^{tern} x_i^{tern}) / x_i^{bin} = \gamma_i^{tern} / (1 + x_{IL}^{bin})$$

where the superscript "tern" represents the hypothetical ternary system comprising cation, anion and solute i, with

$$x_{\text{cation}}^{\text{tern}} = x_{\text{anion}}^{\text{tern}}$$

$$x_{\text{cation}}^{\text{tern}} + x_{\text{anion}}^{\text{tern}} + x_i^{\text{tern}} = 1$$

and the superscript “bin” represents the binary mixture comprising solute and IL, with

$$x_{\text{IL}}^{\text{bin}} + x_i^{\text{bin}} = 1$$

Accordingly, the activity coefficient of a solute i in the binary mixture (IL + solute) at infinite dilution is simplified as

$$\gamma_i^{\text{bin}} = 0.5 \gamma_i^{\text{tern}} \quad (\text{at infinite dilution})$$

Thus in this case we should scale the activity coefficient at infinite dilution  $\gamma_i^{\text{tern}}$ , which is directly obtained from the COSMO-RS calculation, with a factor of 0.5.

Similarly, for a ternary system comprising component i, component j and an ionic liquid, the activity coefficient at finite concentration of component i can be calculated by

$$\gamma_i^{\text{tern}} = \gamma_i^{\text{quart}} / (1 + x_{\text{IL}}^{\text{tern}})$$

where the superscript “quart” represents the hypothetical quaternary system comprised of cation, anion, solute i and solute j, with:

$$x_{\text{cation}}^{\text{quart}} = x_{\text{anion}}^{\text{quart}}$$

$$x_{\text{cation}}^{\text{quart}} + x_{\text{anion}}^{\text{quart}} + x_i^{\text{quart}} + x_j^{\text{quart}} = 1$$

and the superscript “tern” represents the ternary mixture comprising solute i, j, and IL, with

$$x_{\text{IL}}^{\text{tern}} + x_i^{\text{tern}} + x_j^{\text{tern}} = 1$$



## COSMO RESULT FILES

COSMO-RS needs as input for the calculation so called COSMO result files for each compound, which are results of quantum mechanical calculation using COSMO. In ADF such a COSMO result file is called an adf.rkf file (previously ADF<=2019 known as TAPE21 or as a .t21 file) or a COSKF file (.coskf). With Fast Sigma such a COSMO result file is a COMPKF file (.compkf). With MOPAC such a COSMO result file is a .cos file, which can be converted to a COSKF file. In other programs such a file could be called a .cosmo file. ADF has databases of .coskf files, the COSMO-RS compound database *ADFCRS-2018* (page 24) (including ionic liquids) and the COSMO-RS polymer database *ADFCRS-2019* (page 26). At <https://design.che.vt.edu> one may find a database of .cosmo files, which were made with a different program. Note that the optimal COSMO-RS parameters may depend on the program chosen.

### 3.1 ADF COSMO calculation

#### 3.1.1 ADF COSMO settings

Here it is described briefly how to make COSMO result files consistent with the way they were made for the ADF parametrization of COSMO-RS to ensure full parameter applicability. First a gas phase geometry optimization should be performed with ADF, with a small core TZP basis set, the Becke-Perdew functional (BP86), the relativistic scalar ZORA method (which is the default in ADF2020), and good numerical integration quality:

```
AMS_JOBNAME=GASPHASE "$AMSBIN/ams" << eor
Task GeometryOptimization
System
  Atoms
  ...
  End
End
Engine ADF
  Basis
  Type TZP
  Core Small
  PerAtomType Symbol=I File=ZORA/TZ2P/I.4p
  End
  XC
  GGA Becke Perdew
  End
  BeckeGrid
  Quality Good
  End
  Relativity
  Level Scalar
  Formalism ZORA
```

(continues on next page)

(continued from previous page)

```
End
EndEngine
eor
```

For heavier elements than krypton ( $Z>36$ ), like iodine, a small core TZ2P basis set is required. The resulting adf.rkf (previously ADF<=2019 TAPE21 file or .t21 file) of the molecule is used as a restart file in the COSMO calculation, and the system is loaded using the resulting ams.rkf. The ADF COSMO calculation is performed with the following settings:

```
AMS_JOBNAME=COSMO "$AMSBIN/ams" << eor
Task SinglePoint
LoadSystem
  File GASPHASE/ams.rkf
End
EngineRestart GASPHASE/adf.rkf
Engine ADF
  Basis
    Type TZP
    Core Small
    PerAtomType Symbol=I File=ZORA/TZ2P/I.4p
  End
  XC
    GGA Becke Perdew
  End
  BeckeGrid
    Quality Good
  End
  Relativity
    Level Scalar
    Formalism ZORA
  End
  Symmetry NOSYM
  SOLVATION
    Surf Delley
    Solvent name=CRS emp=0.0 cav0=0.0 cav1=0.0
    Charged method=CONJ corr
    C-Mat EXACT
    SCF VAR ALL
    RADII
      H 1.30
      C 2.00
      N 1.83
      O 1.72
      F 1.72
      Si 2.48
      P 2.13
      S 2.16
      Cl 2.05
      Br 2.16
      I 2.32
    SubEnd
  END
EndEngine
eor
```

In this COSMO calculation the Delley type of cavity construction is chosen (See Ref.<sup>1</sup> for details on the Delley surface

<sup>1</sup> B. Delley, *The conductor-like screening model for polymers and surfaces*. *Molecular Simulation* 32, 117 (2006) (<https://doi.org/10.1080/08927020600589684>)

construction). The name of the solvent is CRS, which sets the dielectric constant to infinite and sets the radius of the probing sphere to determine the solvent excluded part of the surface to 1.3 Angstrom.

In case of a cation or an anion, in both the gas phase calculations as well as in the COSMO calculation one should include the charge with the subkey CHARGE of the key SYSTEM in the AMS part of the input.

In the Radii subblock key the Klamt atomic cavity radii are chosen. The parameters emp, cav0, and cav1 are zero. The corr option to the CHARGED subkey constrains the computed solvent surface charges to add up to the negative of the molecular charge. Specifying exact for the C-MAT subkey causes ADF to compute straightforwardly the Coulomb potential due to the charge  $q$  in each point of the molecular numerical integration grid and integrate against the electronic charge density. This is, in principle, exact but may have inaccuracies when the numerical integration points are very close to the positions of a charge  $q$ . To remedy this, starting from ADF2010 the electrostatic potential is damped in case of (very) close lying numerical integration points and COSMO surface points. The numerical stability of the results compare to those of ADF2009 was increased as a result of this. Specifying exact for the C-MAT subkey also requires that the ADF calculation uses SYMMETRY NOSYM.

The resulting adf.rkf (previously ADF<=2019 TAPE21 file or .t21 file) of the COSMO calculation is a COSMO result file.

In a COSMO-RS calculation only the 'COSMO' part of this file is needed. One can make a kf file compound.coskf, which only consists of the section 'COSMO' if one does:

```
$AMSBIN/cpkf adf.rkf compound.coskf "COSMO"
```

The file compound.coskf should not exist before this command is given. Note that such a .coskf file is not a complete adf.rkf anymore. For example, only the COSMO surface can be viewed with AMSview. It is useful mostly for COSMO-RS calculations.

**Links** COSMO-RS GUI tutorial: COSMO result files [1]

### 3.1.2 Atomic cation or anion

In case of an atomic calculation one should of course not perform a geometry optimization. In case of a cation or an anion, in both the gas phase calculations as well as in the COSMO calculation one should include the charge with the key CHARGE. Only for atomic calculations one should include the argument method=atom to the subkey Charged of the key SOLVATION:

```
SOLVATION
  Surf Delley
  Solvent name=CRS cav0=0.0 cav1=0.0
  Charged method=atom corr
  C-Mat EXACT
  SCF VAR ALL
END
```

### 3.1.3 Accuracy

Several parameters in the COSMO calculation can influence the accuracy of the result of the quantum mechanical calculation. Some of these parameters will be discussed. Note that if one chooses different parameters in the COSMO calculation one may also have to reparametrize the ADF COSMO-RS parameters. A list of some of the ADF COSMO parameters.

- XC functional
- basis set
- fit set

- atomic cavity radii and radius of the probing sphere
- cavity construction
- geometry

The atomic cavity radii and the radius of the probing sphere are the same as in Ref.<sup>2</sup>, which describes the COSMO-RS method developed by Klamt et al., which is implemented in ADF. The Becke Perdew functional is relatively good for weakly bound systems, but may not be so good in other cases. The basis set TZP is a compromise basis set. For heavier elements than krypton ( $Z > 36$ ), like iodine, a TZ2P basis set is required, including the relativistic scalar ZORA method. Since the relativistic method hardly cost extra time compared to a non-relativistic method, the scalar relativistic scalar ZORA method is recommended to be used also for light elements. The Delley type of cavity construction in ADF can give a large number of COSMO points. The XC functional, basis set, and cavity construction chosen in the ADF COSMO calculation have a similar accuracy as those that were used in Ref.<sup>Page 22, 2</sup>. Note that they are not exactly the same as were used in Ref.<sup>2</sup>, since in that paper a different quantum mechanical program was used.

In the parametrization for ADF the same geometry was used for the gas phase and the COSMO calculation, which is different than in Ref.<sup>2</sup>. It depends on the actual solvent if reoptimizing the molecule in the COSMO calculation may give better results. Note that the dielectric medium used in the COSMO model has an infinite dielectric constant in the COSMO-RS model. Thus a geometry optimization of the molecule in the COSMO calculation might be more appropriate for a molecule dissolved in water than for a molecule dissolved in n-hexane.

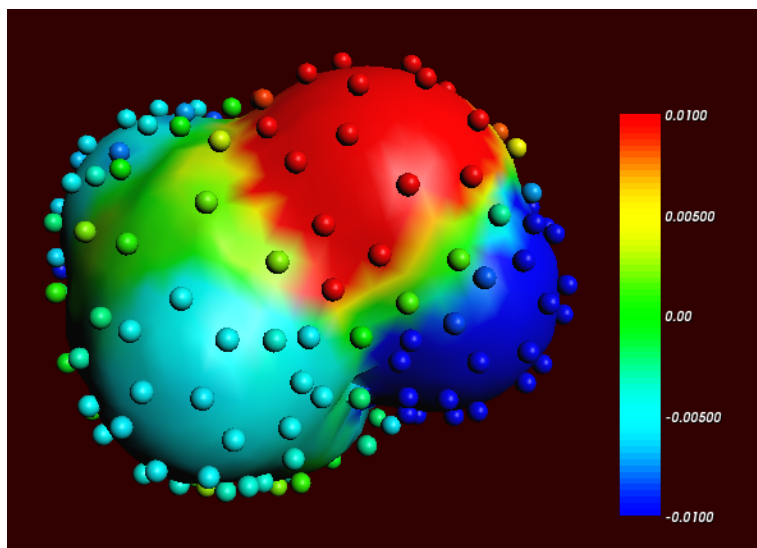
The fit set in ADF is not always able to describe the Coulomb potential accurately at each of the COSMO surface points. In regular ADF calculations this problem is not apparent since the numerical errors in the integrals computed in the vicinity of the COSMO surface have little impact. However, in COSMO calculations this may have some effect. This is why the option C-Mat exact was selected above, instead of the default C-Mat pot option. Another possibility is to add more fit functions, for example, using 'FitQuality Good' for the key ZlmFit in the input for the adf calculation.

### 3.1.4 Cavity construction

The Esurf type of cavity construction in ADF with default settings does not give a large number of COSMO points. Therefore it is recommended to use the so called Delley type of cavity construction (Ref.<sup>Page 20, 1</sup>), which allows one to construct a surface which has many more points. The Esurf type of cavity construction also allows many more points if one sets the option NFDiv of the subkey DIV of the key SOLVENT to a larger value than the default value of 1. This will not be discussed here further. In ADF2010 the numerical stability of the Delley surface has been improved, by merging close lying COSMO surface points, and removing COSMO surface points with a small surface area. A figure of a COSMO surface with the Esurf type of cavity construction with default settings is given below. In this figure the small spheres represent the COSMO surface points that are used for the construction of the COSMO surface. The red part represents positive COSMO charge density, the blue part negative COSMO charge density (the coloring scheme is chosen to match the one by Klamt):

---

<sup>2</sup> A. Klamt, V. Jonas, T. Bürger and J.C. Lohrenz, *Refinement and Parametrization of COSMO-RS*. *J. Phys. Chem. A* 102, 5074 (1998) (<https://doi.org/10.1021/jp980017s>)

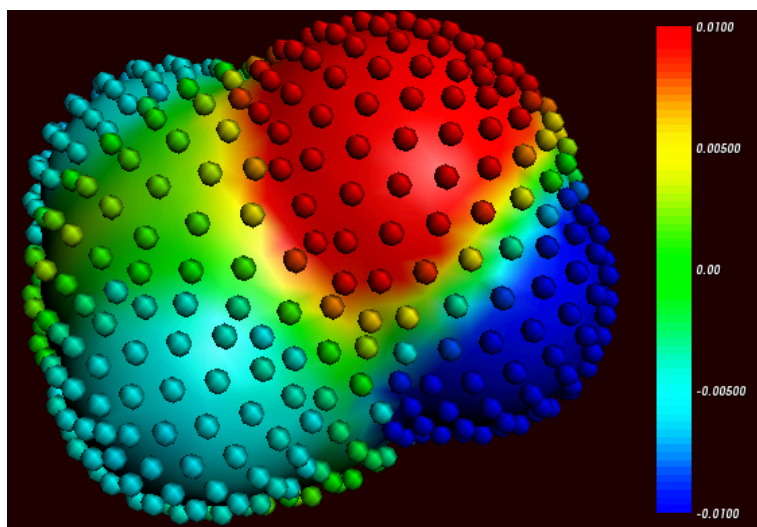


Cosmo charge density on the COSMO surface of methanol, Esurf surface (picture made with AMSview).

One can construct a surface which has many more points using a so called Delley surface. For the subkey SURF of the key SOLVENT one can choose delley. The subkey DIV of the key SOLVENT has extra options leb1 (default value 23), leb2 (default value 29), and rleb (default value 1.5 Angstrom). If the cavity radius of an atom is lower than rleb use leb1, otherwise use leb2. These values can be changed: using a higher value for leb1 and leb2 gives more surface points (maximal value leb1, leb2 is 29). A value of 23 means 194 surface points in case of a single atom, and 29 means 302 surface points in case of a single atom. Typically one could use leb1 for the surface point of H, and leb2 for the surface points of other elements.

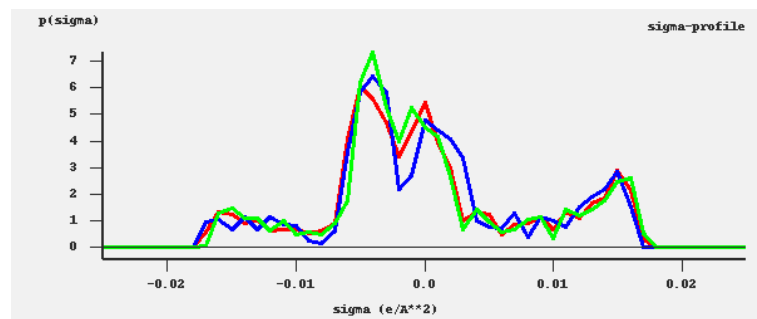
The next figure is made with the following (default for the Delley surface) settings:

```
SOLVATION
SURF Delley
DIV leb1=23 leb2=29 rleb=1.5
END
```



Cosmo charge density on the COSMO surface of methanol, Delley surface (picture made with AMSview).

The different ways of constructing the cavity has some consequences for the  $\sigma$ -profile of methanol, see the figure below:



$\sigma$ -profiles of methanol (picture made with the CRS-GUI). In this picture the blue line is the  $\sigma$ -profile with the Esurf type of construction, the red line is that with the Delley type of construction with many surface points. For comparison, the green line is the  $\sigma$ -profile of methanol if a large QZ4P basis set is used, again with the Delley type of construction with many surface points.

## 3.2 ADFCRS-2018 Database

The COSMO-RS Database ADFCRS-2018 contains 2560 compounds. This database combines the COSMO-RS database ADFCRS-2010, the ionic liquid database ADFCRS-IL-2014, and some extra compounds.

Follow the [COSMO-RS GUI Tutorial on the COSMO-RS compound database](#) for more information on how to download and install the zipped COSMO-RS Database ADFCRS-2018 <https://downloads.scm.com/Downloads/packages/ADFCRS/ADFCRS-2018.5.zip>, and how to use it.

The database contains CAS Registry Numbers. CAS Registry Number is a Registered Trademark of the American Chemical Society.

---

**Note: Update on ADFCRS-2018-5** Compounds containing hydrogen bond atoms have been updated to include the hydrogen bond center (hbc). The hbc is determined from the local minima of the molecular electrostatic potential (MESP) computed from the Densf calculation. This update is necessary for utilizing the COSMO-SAC DHB-ADF MESP method.

---

---

**Note: Update on ADFCRS-2018-4** Some compounds have been updated with the correct CAS number and SMILES. Upon updating the SMILES, the coskf file has been rerun.

---



*vent Mixtures for Organic Pharmacological Compounds with COSMO-Based Thermodynamic Methods.* *Ind. Eng. Chem. Res.* 2008, 47 (5), 1707-1725. (<https://doi.org/10.1021/ie0711022>)

Phillips, K.L.; Sandler, S.I.; Greene, R.W.; Di Toro, D.M. *Quantum Mechanical Predictions of the Henry's Law Constants and Their Temperature Dependence for the 209 Polychlorinated Biphenyl Congeners.* *Environ. Sci. Technol.* 2008, 42 (22), 8412-8418. (<https://doi.org/10.1021/es800876w>)

All structures in the database ADFCRS-2010 differ from those in the papers above, as they are the result of ADF geometry optimizations.

### 3.2.2 ADFCRS-IL-2014

Ionic liquids (ILs), usually consisting of a large organic cation and a small inorganic polyatomic anion, have attracted considerable attention due to their unique thermophysical properties. The low vapor pressure and high conductivity of these molten salts combined with highly tunable properties, have resulted in highly diverse applications across many different fields in chemistry, materials science (battery electrolytes), chemical engineering (gas sorption and purification), and many more.

The COSMO-RS ionic liquid database ADFCRS-IL-2014 contains 80 cations and 56 anions. This ADFCRS-IL-2014 database consists of ADF COSMO result (.coskf) files, from standard ADF quantum mechanical calculations, as described in [COSMO-RS GUI Tutorial: COSMO result files](#).

SCM gratefully acknowledges Prof. Zhigang Lei's research group (State Key Laboratory of Chemical Resource Engineering, Beijing University of Chemical Technology, China) for providing the ionic liquid database as well as the corresponding tutorial.

The work of Zhigang Lei group based on the COSMO-RS model using the ADF software is listed as follows:

Z. Lei, C. Dai, J. Zhu, B. Chen, *Extractive distillation with ionic liquids: A review*, *AIChE Journal* 60, 3312 (2014) (<https://doi.org/10.1002/aic.14537>)

Z. Lei, C. Dai, B. Chen, *Gas solubility in ionic liquids*, *Chemical Reviews* 14, 1289 (2014) (<https://doi.org/10.1021/cr300497a>)

Z. Lei, J. Han, Q. Li, and B. Chen, *Process Intensification on the Supercritical Carbon Dioxide Extraction of Low-Concentration Ethanol from Aqueous Solutions*, *Industrial Engineering Chemistry research* 51, 2730 (2012) (<https://doi.org/10.1021/ie2021027>)

Z. Lei, J. Han, B. Zhang, Q. Li, J. Zhu, and B. Chen, *Solubility of CO<sub>2</sub> in Binary Mixtures of Room-Temperature Ionic Liquids at High Pressures*, *Journal of Chemical Engineering data* 57, 2153 (2012) (<https://doi.org/10.1021/je300016q>)

Z. Lei, C. Dai, X. Liu, L. Xiao, and B. Chen, *Extension of the UNIFAC Model for Ionic Liquids*, *Industrial Engineering Chemistry research* 51, 12135 (2012) (<https://doi.org/10.1021/ie301159v>)

Z. Lei, C. Dai, Q. Yang, J. Zhu, and B. Chen, *UNIFAC model for ionic liquid-CO (H<sub>2</sub>) systems: An experimental and modeling study on gas solubility*, *AIChE Journal* (2014), DOI: 10.1002/aic.14606 (<https://doi.org/10.1002/aic.14606>)

## 3.3 ADFCRS-POLYMERS-2019 Database

The COSMO-RS database ADFCRS-POLYMERS-2019 contains data for more than 200 polymers. The zipped COSMO-RS database ADFCRS-POLYMERS-2019 can be downloaded from <https://downloads.scm.com/Downloads/crs/ADFCRS-POLYMERS-2019.zip>. The database needs to be unzipped.

## 3.4 FastSigma: a QSPR method to estimate COSMO sigma-profiles

### 3.4.1 Introduction

The traditional workflow for performing a COSMO-RS/-SAC calculation first involves a very expensive DFT geometry optimization and single point calculation in the COSMO phase to generate  $\sigma$ -profiles and other necessary parameters for COSMO-RS/-SAC. Once these parameters are known, the COSMO-RS/-SAC calculation can be performed extremely efficiently, often in only a matter of milliseconds. This imbalance of computational expense means there is a significant opportunity in circumventing the expensive DFT steps.

The `FastSigma` program reads a molecule in several possible formats (SMILES, .mol, .sdf) and estimates all of the properties required for a COSMO-RS/-SAC calculation: the HB-/Non-HB-/OT-/OH-  $\sigma$ -profiles, COSMO surface area, and COSMO volume as well as bond energies that can be important for vapor phase or multispecies calculations. This code incorporates two distinct methods that are able to estimate these important COSMO-RS/-SAC properties. The first uses QSPR techniques similar to those applied in our [Property Prediction program](#) which shares the same accepted atom types. The second uses a database of  $\sigma$ -profiles and a custom molecular graph hashing algorithm to build  $\sigma$ -profiles for query molecules using a set of the  $\sigma$ -profiles from molecules in the database containing similar substructures.

Both of these techniques are extremely efficient and are capable of providing estimates for these essential COSMO-RS/-SAC properties in milliseconds. This allows for quick thermodynamic calculations to be done for a new molecule of interest as well as drastically expedites searches through screening databases of molecular candidates as compared to the traditional, full-fledged COSMO-RS/-SAC workflow.

---

**Important:** To use the `SG1` method, users will have to first download the **Subgraph Sigma Profile Estimation (SG1) Database** (`molsg_sg1.db`) using the [AMS Package Manager](#).

---

**Note:** `pyCRS` (page 95) can be used for python scripting with `FastSigma`. Several python examples are given in the `pyCRS` documentation.

---

### 3.4.2 Input options

A list of the input options and examples of their usage is given below.

Flag	Purpose	Example
-h [-help]	Produces help message	<code>\$AMSBIN/fast_sigma -help</code>
-s [-smiles]	Input molecule as SMILES sting	<code>\$AMSBIN/fast_sigma -smiles &lt;SMILES&gt; ...</code>
-m [-mol]	Input molecule as .mol file	<code>\$AMSBIN/fast_sigma -mol &lt;mol file&gt; ...</code>
-sdf	Input molecule as an .sdf file	<code>\$AMSBIN/fast_sigma -sdf &lt;file.sdf&gt; ...</code>
-model	Choose from 2 possible techniques	<code>\$AMSBIN/fast_sigma -model FS1 ...</code>
-d [-display]	Display problem results	<code>\$AMSBIN/fast_sigma -d ...</code>
-o [-output]	Write output to file	<code>\$AMSBIN/fast_sigma -o &lt;output.compkf&gt; ...</code>
-method	Chose a COSMO-RS/-SAC method	<code>\$AMSBIN/fast_sigma -method COSMO-RS ...</code>

**--model FS1**

The FS1 model is a QSPR model. It currently has two supported methods: **COSMO-RS** and **COSMOSAC2016**. One of these method names must be entered after the `-method` flag. The default method is COSMO-RS.

**--model SG1**

The SG1 model is based on substructure hashing and database searching. It currently has two supported methods: **COSMO-RS** and **COSMOSAC2013**.

---

**Note:** This model can take a few seconds to load the required database. If the user would like to use this method to estimate multiple compounds, it is recommended to use pyCRS. With pyCRS, the database will only be loaded during the first calculation and then stay in memory.

---

**-o <output.compkf>**

The fast sigma program writes the output results to a file in .compkf format. The chosen output filename should generally end with .compkf. This suffix helps other parts of the code (COSMO-RS/-SAC/-UNIFAC/Solvent Optimization) recognize the format and use the file accordingly. If no filename is supplied the program writes to a file called CRSKF.compkf.

**-s <SMILES\_string or .mol file>**

Though COSMO-RS/-SAC can make estimates for many types of molecular species, the fast sigma program currently only supports organic, neutral, closed shell molecules.

### 3.4.3 GUI Input

The simplest way to use the Fast Sigma program is through the COSMO-RS GUI. There are two ways to do this:

- SMILES string: **Compounds** → **List of Compounds** → **Add Compound using FastSigma** → **SMILES** and select Add.
- .xyz file: **Compounds** → **List of Compounds** → **Add Compound using FastSigma** → **.xyz**, and select Add.

A .compkf file will be saved that can be used as input in COSMO-RS calculations.

### 3.4.4 Examples

This example calculates COSMO-RS (the default) parameters for phenol:

```
$AMSBIN/fast_sigma --smiles "c1ccccc1(O)" -d
```

sigma value	Total profile	HB profile
-0.025	0.000	0.000
-0.024	0.000	0.000
-0.023	0.000	0.000
-0.022	0.002	0.002
-0.021	0.054	0.054
-0.020	0.263	0.263
-0.019	0.523	0.523
-0.018	0.684	0.684
-0.017	0.828	0.828
-0.016	0.801	0.801
-0.015	0.732	0.716
-0.014	0.642	0.597
-0.013	0.653	0.519
-0.012	0.678	0.487

(continues on next page)

(continued from previous page)

-0.011	0.607	0.423
-0.010	0.567	0.382
-0.009	0.646	0.245
-0.008	4.183	0.023
-0.007	7.405	0.000
-0.006	7.912	0.000
-0.005	6.701	0.000
-0.004	5.544	0.000
-0.003	4.658	0.000
-0.002	3.899	0.000
-0.001	4.097	0.000
0.000	6.109	0.000
0.001	7.854	0.000
0.002	8.640	0.000
0.003	9.726	0.000
0.004	11.175	0.000
0.005	12.524	0.000
0.006	8.673	0.000
0.007	2.255	0.000
0.008	1.174	0.161
0.009	1.279	1.159
0.010	1.442	1.442
0.011	1.759	1.751
0.012	1.795	1.788
0.013	0.838	0.829
0.014	0.095	0.093
0.015	0.054	0.054
0.016	0.030	0.030
0.017	0.000	0.000
0.018	0.000	0.000
0.019	0.000	0.000
0.020	0.000	0.000
0.021	0.000	0.000
0.022	0.000	0.000
0.023	0.000	0.000
0.024	0.000	0.000
0.025	0.000	0.000
Molecular Mass =	94.0418648120 g/mol	
COSMO Area =	127.5012207186 Angstrom**2	
COSMO Volume =	122.0791950835 Angstrom**3	
Gas Phase Bond Energy =	-2.9875007647 Hartree	
Bond Energy =	-2.9968155744 Hartree	
Dispersion =	-4.5319123638 kcal/mol	
Deltaediel =	0.0000000000 Hartree	
Nring =	6	
Chemical Formula =	C6H6O	
SMILES =	c1ccccc1(O)	

Additionally, we calculate the COSMOSAC2016 parameters for Ibuprofen as a mol file:

```
$AMSBIN/fast_sigma --mol Ibuprofen.mol --method COSMOSAC2016 -d
```

sigma value	Total profile	OH profile	OT profile
-0.025	0.000	0.000	0.000
-0.024	0.000	0.000	0.000
-0.023	0.000	0.000	0.000

(continues on next page)

(continued from previous page)

-0.022	0.000	0.000	0.000
-0.021	0.009	0.009	0.000
-0.020	0.062	0.061	0.000
-0.019	0.395	0.385	0.000
-0.018	0.914	0.881	0.000
-0.017	0.925	0.879	0.000
-0.016	0.840	0.781	0.000
-0.015	0.652	0.590	0.000
-0.014	0.697	0.606	0.000
-0.013	0.604	0.499	0.000
-0.012	0.561	0.398	0.000
-0.011	0.725	0.418	0.000
-0.010	0.833	0.350	0.000
-0.009	1.282	0.230	0.000
-0.008	2.141	0.158	0.000
-0.007	5.133	0.085	0.000
-0.006	10.428	0.048	0.000
-0.005	14.386	0.000	0.000
-0.004	23.816	0.000	0.000
-0.003	26.081	0.000	0.000
-0.002	23.295	0.000	0.000
-0.001	21.443	0.000	0.000
0.000	22.124	0.000	0.000
0.001	20.652	0.000	0.000
0.002	24.315	0.036	0.000
0.003	15.722	0.086	0.035
0.004	11.878	0.171	0.092
0.005	13.670	0.288	0.197
0.006	10.405	0.381	0.307
0.007	5.479	0.561	0.413
0.008	3.525	0.713	0.613
0.009	3.358	0.823	1.055
0.010	3.879	0.639	1.840
0.011	4.503	0.180	3.025
0.012	2.708	0.083	2.006
0.013	0.930	0.020	0.745
0.014	0.061	0.000	0.104
0.015	0.000	0.000	0.000
0.016	0.000	0.000	0.000
0.017	0.000	0.000	0.000
0.018	0.000	0.000	0.000
0.019	0.000	0.000	0.000
0.020	0.000	0.000	0.000
0.021	0.000	0.000	0.000
0.022	0.000	0.000	0.000
0.023	0.000	0.000	0.000
0.024	0.000	0.000	0.000
0.025	0.000	0.000	0.000
Molecular Mass =	206.1306798160	g/mol	
COSMO Area =	278.4276940312	Angstrom**2	
COSMO Volume =	279.3341044098	Angstrom**3	
Gas Phase Bond Energy =	-7.1463537624	Hartree	
Bond Energy =	-7.1619486814	Hartree	
Dispersion =	-9.7153055452	kcal/mol	
Deltaediel =	0.0007518662	Hartree	
Nring =	6		
Chemical Formula =	C13H18O2		

(continues on next page)

(continued from previous page)

```
SMILES = CC(C)Cc1ccc(C(C)C(=O)O)cc1
```

We can also use the SG1 model for phenol.

```
$AMSBIN/fast_sigma --smiles "c1ccccc1(O)" --model SG1 -d
```

sigma value	Total profile	HB profile
-0.025	0.000	0.000
-0.024	0.000	0.000
-0.023	0.000	0.000
-0.022	0.003	0.003
-0.021	0.067	0.067
-0.020	0.434	0.434
-0.019	0.878	0.878
-0.018	0.995	0.995
-0.017	0.996	0.996
-0.016	0.942	0.940
-0.015	0.771	0.766
-0.014	0.684	0.635
-0.013	0.610	0.549
-0.012	0.693	0.486
-0.011	0.671	0.397
-0.010	0.755	0.350
-0.009	1.344	0.255
-0.008	4.312	0.026
-0.007	7.751	0.000
-0.006	7.855	0.000
-0.005	6.819	0.000
-0.004	6.226	0.000
-0.003	5.612	0.000
-0.002	4.654	0.000
-0.001	4.679	0.000
0.000	4.969	0.000
0.001	5.814	0.000
0.002	7.672	0.000
0.003	10.711	0.000
0.004	12.231	0.000
0.005	12.061	0.000
0.006	8.394	0.000
0.007	3.355	0.000
0.008	1.677	0.153
0.009	1.434	1.226
0.010	1.566	1.566
0.011	1.972	1.972
0.012	2.133	2.133
0.013	0.966	0.966
0.014	0.062	0.062
0.015	0.000	0.000
0.016	0.000	0.000
0.017	0.000	0.000
0.018	0.000	0.000
0.019	0.000	0.000
0.020	0.000	0.000
0.021	0.000	0.000
0.022	0.000	0.000
0.023	0.000	0.000

(continues on next page)

(continued from previous page)

```

0.024          0.000          0.000
0.025          0.000          0.000
Molecular Mass =      94.0418648120 g/mol
COSMO Area =      133.1606910587 Angstrom**2
COSMO Volume =      122.0268006780 Angstrom**3
Gas Phase Bond Energy =      -2.9830476046 Hartree
Bond Energy =      -2.9928087890 Hartree
Dispersion =      0.0000000000 kcal/mol
Deltaediel =      0.0000000000 Hartree
Nring =      6
Chemical Formula =      C6H6O
SMILES =      c1ccccc1(O)

```

The warning message will be displayed if a molecule contains atoms or substructures that are not listed in the [accepted atom types table](#). For example, in the compound C1=CC=[Ge]C=C1, the atom 'Ge' is not available in the QSPR method. As a result, the property prediction tool will yield incorrect sigma profile.

```
$AMSBIN/fast_sigma --smiles "C1=CC=[Ge]C=C1" -d
```

```
WARNING: there are atoms and/or substructures in the molecule which cannot be
←estimated.
```

```
This will affect the accuracy of the results.
```

```
Atoms which cannot be estimated:
```

```
Ge
```

sigma value	Total profile	HB profile
-0.025	0.000	0.000
-0.024	0.000	0.000
-0.023	0.000	0.000
-0.022	0.000	0.000
-0.021	0.000	0.000
-0.020	0.000	0.000
-0.019	0.000	0.000
-0.018	0.000	0.000
-0.017	0.000	0.000
-0.016	0.000	0.000
-0.015	0.000	0.000
-0.014	0.000	0.000
-0.013	0.000	0.000
-0.012	0.000	0.000
-0.011	0.000	0.000
-0.010	0.000	0.000
-0.009	0.000	0.000
-0.008	0.896	0.000
-0.007	2.280	0.000
-0.006	5.170	0.000
-0.005	9.078	0.000
-0.004	9.044	0.000
-0.003	4.854	0.000
-0.002	4.211	0.000
-0.001	4.505	0.000
-0.000	4.415	0.000
0.001	4.824	0.000
0.002	4.750	0.000
0.003	5.745	0.000

(continues on next page)

(continued from previous page)

0.004	3.006	0.000
0.005	4.904	0.000
0.006	5.411	0.000
0.007	4.222	0.000
0.008	2.623	0.000
0.009	0.000	0.000
0.010	0.000	0.000
0.011	0.000	0.000
0.012	0.000	0.000
0.013	0.000	0.000
0.014	0.000	0.000
0.015	0.000	0.000
0.016	0.000	0.000
0.017	0.000	0.000
0.018	0.000	0.000
0.019	0.000	0.000
0.020	0.000	0.000
0.021	0.000	0.000
0.022	0.000	0.000
0.023	0.000	0.000
0.024	0.000	0.000
0.025	0.000	0.000
Molecular Mass =	138.9603029600 g/mol	
COSMO Area =	79.9378826526 Angstrom**2	
COSMO Volume =	88.0443110798 Angstrom**3	
Gas Phase Bond Energy =	-2.2538026599 Hartree	
Bond Energy =	-2.2571102789 Hartree	
Dispersion =	-2.9625031363 kcal/mol	
Deltaediel =	0.0000000000 Hartree	
Nring =	6	
Chemical Formula =	C5H5Ge	
SMILES =	C1=CC=[Ge]C=C1	

### 3.5 MOPAC COSMO calculation

Here it is described briefly how to make MOPAC COSMO result files.

The simplest way is to use AMSinput. Draw the molecule using AMSinput, and save the .ams file. Select **Right Panel** → **MOPAC** → **Solvation method** → **COSMO-CRS**. Select 362 for NSPA. Press Run to run the MOPAC calculation. A .coskf file will be saved that can be used as input in COSMO-RS calculations.

In AMS2019 AMSinput uses the MOPAC engine, which is part of the AMS driver. Note that this is different than in AMS2018 and before. In the Atoms block key in the AMS driver part of the input one puts the coordinates of the molecule. The main input keys for the AMS driver and the MOPAC engine are:

```
$AMSBIN/ams << eor
Task GeometryOptimization
System
  Atoms
  ....
  End
End
Engine MOPAC
```

(continues on next page)

(continued from previous page)

```
Solvation
  Enabled Yes
  NSPA 362
  Solvent
    Name CRS
  End
End
EndEngine

eor
```

The use of the solvent CRS makes the MOPAC engine to create a .cos file, which is converted to a .coskf file by \$AMSBIN/cosmo2kf

```
cosmo2kf file.cos file.coskf
```

Note that this is automatically done if one uses AMSinput.

Compared to the default ADF COSMO-RS values a few *COSMO-RS parameters* (page 35) were reoptimized for MOPAC PM6 COSMO result files to improve the calculation of a number of partition coefficients, when compared to experimental values. Note that MOPAC is a semi-empirical quantum chemistry program, whereas ADF is based on density functional theory (DFT). Thus the MOPAC COSMO result files will not be of the same quality as the ADF COSMO result files.

## THE COSMO-RS PROGRAM

The ADF COSMO-RS command line program *crs* is described here, including all input options.

### 4.1 Running the COSMO-RS program

Running the COSMO-RS program involves the following steps:

- Construct an ASCII input file, say *in*.
- Run the program by typing (under UNIX): `$AMSBIN/crs < in > out`
- Move / copy relevant result files (in particular *CRSKF*) to the directory where you want to save them, and give them appropriate names.
- Inspect the standard output file *out* to verify that all has gone well.

Note that in the one can also put the call to `$AMSBIN/crs` inside a script, which could be named, for example, `example.run`. Such shell script `example.run` needs to be executable, if it isn't you will need to make it executable, e.g. `chmod u+x example.run`. The `example.run` file needs to be executed as a shell script, not as input to `$AMSBIN/crs`.

### 4.2 COSMO-RS and COSMO-SAC parameters

The COSMO-RS model has general parameters and element specific parameters. ADF's COSMO-SAC 2013-ADF model has general parameters, but also uses some of the COSMO-RS parameters, such as the element specific parameters. There are also technical and accuracy parameters, such as convergence criteria. This section explains how to set these parameters, and shows the default values for these parameters. By default the COSMO-RS method is chosen.

#### 4.2.1 COSMO-RS general parameters

```
CRSPARAMETERS
{RAV          rav}
{APRIME       aprime}
{FCORR        fcorr}
{CHB          chb}
{SIGMAHBOND   sigmahbond}
{AEFF         aeff}
{LAMBDA       lambda}
{OMEGA        omega}
{ETA          eta}
{CHORTF       chortf}
```

(continues on next page)

(continued from previous page)

```
{combi1998 | combi2005}
{hb_all | hb_hnof}
{hb_temp | hb_notemp}
End
```

The ADF default values are optimized parameters for ADF calculations. The Klamt values can be found in Ref.<sup>1</sup>. See also Ref.<sup>Page 36, 1</sup> for the meaning of the parameters.

Table 4.1: COSMO-RS Parameters

symbol	ADF Default	ADF combi1998	Klamt	MOPAC PM6
	Ref. <sup>2</sup>	Ref. <sup>2</sup>	Ref. <sup>1</sup>	
rav ( $r_{av}$ )	0.400	0.415	0.5	0.400
aprime ( $a'$ )	1510.0	1515.0	1288.0	1550.0
fcorr ( $f_{corr}$ )	2.802	2.812	2.4	2.802
chb ( $c_{hb}$ )	8850.0	8850.0	7400.0	8400.0
sigmahbond ( $\sigma_{hb}$ )	0.00854	0.00849	0.0082	0.00978
aeff ( $a_{eff}$ )	6.94	7.62	7.1	5.96
lambda ( $\lambda$ )	0.130	0.129	0.14	0.135
omega ( $\omega$ )	-0.212	-0.217	-0.21	-0.212
eta ( $\eta$ )	-9.65	-9.91	-9.15	-9.65
chortf ( $c^\perp$ )	0.816	0.816	0.816	0.816
combi1998   combi2005	combi2005	combi1998	combi1998	combi2005
hb_all   hb_hnof	hb_hnof	hb_hnof	hb_hnof	hb_hnof
hb_temp   hb_notemp	hb_temp	hb_notemp	hb_notemp	hb_temp

**chortf**

See Ref.<sup>1</sup> for the definitions:  $\sigma_v^\perp = \sigma_v^0 - c^\perp \sigma_v$ . Refer to *the section on the fast approximation for COSMO-RS calculations* (page 7)

**combi1998 | combi2005**

If the subkey combi1998 is included a thermodynamically inconsistent combinatorial contribution to the chemical potential  $\mu_i^{comb}$  of Ref.<sup>1</sup> is used. If the subkey combi2005 is included (default) a thermodynamically consistent combinatorial contribution of Ref.<sup>3</sup> is used. See *the section on the combinatorial term* (page 7) and Ref.<sup>3</sup>.

**hb\_all | hb\_hnof**

If the subkey hb\_all is included hydrogen bond interaction can be included between segments that belong to H atoms and all other segments. If the subkey hb\_hnof is included (default) hydrogen bond interaction can be included only between segments that belong to H atoms that are bonded to N, O, or, F, and segments that belong to N, O, or F atoms.

**hb\_temp | hb\_notemp**

If the subkey hb\_notemp is included the hydrogen bond interaction is not temperature dependent, as in Ref.<sup>1</sup>. If the subkey hb\_temp is included (default) the hydrogen bond interaction is temperature dependent, as in Ref.<sup>3</sup>. See *the section on the temperature dependent hydrogen bond interaction* (page 8) and Ref.<sup>3</sup>.

**Links** COSMO-RS GUI tutorial: set COSMO-RS parameters [1]

<sup>1</sup> A. Klamt, V. Jonas, T. Bürger and J.C. Lohrenz, *Refinement and Parametrization of COSMO-RS*. J. Phys. Chem. A 102, 5074 (1998) (<https://doi.org/10.1021/jp980017s>)

<sup>2</sup> C.C. Pye, T. Ziegler, E. van Lenthe, J.N. Louwen, *An implementation of the conductor-like screening model of solvation within the Amsterdam density functional package. Part II. COSMO for real solvents*. Can. J. Chem. 87, 790 (2009) (<https://doi.org/10.1139/V09-008>)

<sup>3</sup> A. Klamt, *COSMO-RS From Quantum Chemistry to Fluid Phase Thermodynamics and Drug Design*, Elsevier. Amsterdam (2005), ISBN 0-444-51994-7.

## 4.2.2 COSMO-RS element specific parameters

```
DISPERSION
  {H  dispH}
  {C  dispC}
  {N  dispN}
  { ... }
End
```

The following table gives the element specific dispersion constants. The ADF default values are optimized parameters for ADF calculations. The Klamt values can again be found in Ref.<sup>1</sup>. The constants for F, Si, P, S, Br, and I in the ADF defaults were only fitted to a small number of experimental values or taken from Ref.<sup>Page 36, 3</sup>.

Table 4.2: COSMO-RS Element Dispersion Parameters

element	ADF Default	ADF combi1998	Klamt
			Ref. <sup>Page 36, 1</sup>
H	-0.0340	-0.0346	-0.041
C	-0.0356	-0.0356	-0.037
N	-0.0224	-0.0225	-0.027
O	-0.0333	-0.0322	-0.042
Cl	-0.0485	-0.0487	-0.052
F	-0.026		
Si	-0.04		
P	-0.045		
S	-0.052		
Br	-0.055		
I	-0.062		

Note that not for all elements in the periodic system COSMO-RS parameters were fitted.

**Links** COSMO-RS GUI tutorial: set COSMO-RS parameters [1]

## 4.2.3 COSMO-SAC general parameters

The ADF COSMO-RS program can calculate activity coefficients using the available COSMO-SAC model in the *COSMO-SAC method table* (page 38). Like in the COSMO-RS method, pure compound vapor pressures can be given as input, for example, if experimental values are available.

- For the COSMO-SAC 2013-ADF model, if vapor pressure values are not specified, they will be calculated directly using the COSMO-SAC 2013-ADF itself.
- For all other COSMO-SAC models, if vapor pressure values are not specified, they will be approximated using a method similar to the COSMO-RS approach.

Note that the earlier COSMO-SAC papers<sup>75</sup> do not include parameters that were optimized for use with ADF COSMO result files. A brief introduction to these models is available at [here](#) (page 8). For further details, please refer to the original references.

<sup>7</sup> S. Wang, S.I. Sandler, C.C. Chen, *Refinement of COSMO-SAC and the Applications*, Ind. Eng. Chem. Res. 46, 7275 (2007) (<https://doi.org/10.1021/ie070465z>)

<sup>5</sup> C.M. Hsieh, S.I. Sandler, S.T. Lin, *Improvements of COSMO-SAC for vapor-liquid and liquid-liquid equilibrium predictions*, Fluid Phase Equilib. 297, 90 (2010) (<https://doi.org/10.1016/j.fluid.2010.06.011>)

Table 4.3: COSMO-SAC Method Key

Method Key	GUI Label	Optimized with ADF Result Files	Reference
COSMOSAC   COSMOSAC2013	2013-ADF Xiong	Y	Ref. <sup>4</sup>
COSMOSAC2016	2016-ADF Chen	Y	Ref. <sup>6</sup>
COSMOSACDHB	DHB-ADF Chen	Y	Ref. <sup>8</sup>
COSMOSACDHB-MESP	DHB-ADF MESP	Y	Ref. <sup>9</sup>
COSMOSAC2010	2010 Hsieh	N	Ref. <sup>5</sup>
COSMOSAC2007	2007 Wang	N	Ref. <sup>Page 37, 7</sup>

To use a COSMO-SAC model, specify one of the method keys listed in the *COSMO-SAC method table* (page 38).

```
Method {COSMOSAC | COSMOSAC2013 | COSMOSAC2016 |
        COSMOSACDHB | COSMOSACDHB-MESP |
        COSMOSAC2010 | COSMOSAC2007}
```

You can provide the specific parameters in the SACPARAMETERS section, as shown below. If the SACPARAMETERS section is not included, the default parameters from the *COSMO-SAC Parameters Table* (page 39) will be used.

```
SACPARAMETERS
{AEFF      aeff}
{FDECAY    fdecay}
{SIGMA0    sigma0}
{RN        rn}
{QN        qn}
{AES       aes}
{BES       bes}
{COHOH     cohoh}
{COTOT     cotot}
{COHOT     cohot}
{RAV       rav}
{QS        qs}
{rhbcut    rhbcut}
{hb_temp   | hb_notemp}
End
```

<sup>4</sup> R. Xiong, S.I. Sandler, R.I. Burnett, *An improvement to COSMO-SAC for predicting thermodynamic properties*, *Ind. Eng. Chem. Res.* 53, 8265 (2014) (<https://doi.org/10.1021/ie404410v>)

<sup>6</sup> W.L. Chen, C.M. Hsieh, L. Yang, C.C. Hsu, S.T. Lin, *A Critical Evaluation on the Performance of COSMO-SAC Models for Vapor-Liquid and Liquid-Liquid Equilibrium Predictions Based on Different Quantum Chemical Calculations*, *Ind. Eng. Chem. Res.* 55, 9312 (2016) (<https://doi.org/10.1021/acs.iecr.6b02345>)

<sup>8</sup> W.L. Chen, S.T. Lin, *Explicit consideration of spatial hydrogen bonding direction for activity coefficient prediction based on implicit solvation calculations*, *Phys.Chem.Chem.Phys.* 19, 20367 (2017) (<https://doi.org/10.1039/c7cp02317k>)

<sup>9</sup> C.C. Chang, W.L. Chen, D.T. Wu, S.T. Lin, *Improved directional hydrogen bonding interactions for the prediction of activity coefficients with COSMO-SAC*, *Ind. Eng. Chem. Res.* 57, 11229 (2018) (<https://doi.org/10.1021/acs.iecr.8b02493>)

Table 4.4: COSMO-SAC Parameters

symbol	2013-ADF Xiong	2016-ADF Chen	DHB-ADF Chen	DHB-ADF MESP	2010 Hsieh	2007 Wang
	Ref. <sup>4</sup>	Ref. <sup>Page 38, 6</sup>	Ref. <sup>Page 38, 8</sup>	Ref. <sup>Page 38, 9</sup>	Ref. <sup>Page 37, 5</sup>	Ref. <sup>Page 37, 7</sup>
a <sub>eff</sub> (a <sub>eff</sub> )	6.4813	5.8447	5.8447	5.8447	7.25	7.25
f <sub>decay</sub> (f <sub>decay</sub> )		3.57	3.57	3.57	3.57	3.57
σ <sub>0</sub> (σ <sub>0</sub> )	0.01233	0.007	0.0063	0.0063	0.007	0.007
r <sub>n</sub> (r)		66.69	66.69	66.69	66.69	66.69
q <sub>n</sub> (q)	79.352	79.53	79.53	79.53	79.53	79.53
A <sub>ES</sub> (A <sub>ES</sub> )	7877.13	5920.84	5920.84	5920.84	6525.69	8451.77
B <sub>ES</sub> (B <sub>ES</sub> )	0.0	1.3950 10 <sup>8</sup>	1.3950 10 <sup>8</sup>	1.3950 10 <sup>8</sup>	1.4859 10 <sup>8</sup>	0.0
c <sub>OH-OH</sub> (c <sub>OH-OH</sub> )	5786.72	3551.10	33306.83	34234.17	4013.78	3484.42
c <sub>OT-OT</sub> (c <sub>OT-OT</sub> )	2739.58	1077.26	33306.83	34234.17	932.31	3484.42
c <sub>OH-OT</sub> (c <sub>OH-OT</sub> )	4707.75	3099.31	33306.83	34234.17	3016.43	3484.42
r <sub>av</sub> (r <sub>av</sub> )	0.51					
q <sub>s</sub> (q <sub>s</sub> )	0.57					
rhbcut			1.4432	1.3871		
hb_temp   hb_notemp	hb_notemp	hb_notemp	hb_notemp	hb_notemp	hb_notemp	hb_notemp

See also Refs.<sup>Page 38, 4Page 37, 5</sup> for the meaning of the parameters a<sub>eff</sub>, f<sub>decay</sub>, σ<sub>0</sub>, r, q, A<sub>ES</sub>, B<sub>ES</sub>, c<sub>OH-OH</sub>, c<sub>OT-OT</sub>, c<sub>OH-OT</sub>, r<sub>av</sub>, q<sub>s</sub>. The parameter names in<sup>Page 37, 7</sup> have been translated into parameter names used in Ref.<sup>Page 37, 5</sup>, by calculating A<sub>ES</sub> from 0.3 f<sub>pol</sub> a<sub>eff</sub><sup>3/2</sup> / (2ε<sub>0</sub>), using B<sub>ES</sub> = 0, and using c<sub>OH-OH</sub> = c<sub>OT-OT</sub> = c<sub>OH-OT</sub> = c<sub>hb</sub>. The parameters f<sub>decay</sub> and r are not used in COSMO-SAC 2013-ADF<sup>Page 38, 4</sup>. The parameters r<sub>av</sub> and q<sub>s</sub> are only used in COSMO-SAC 2013-ADF. The element specific COSMO-SAC 2013-ADF epsilon constants can be set with the block key *EPSILON* (page 40). These element specific epsilon constants can not be used in ADF's implementation of earlier COSMO-SAC methods. The parameter rhbcut is only used in COSMO-SAC DHB-ADF<sup>Page 38, 8</sup> and COSMO-SAC DHB-ADF MESP<sup>Page 38, 9</sup>. Note that the parameters for COSMO-SAC DHB-ADF were reoptimized by Chen et al., and are different than in Ref.<sup>Page 38, 8</sup>.

#### hb\_temp | hb\_notemp

If the subkey hb\_notemp is included (default) the hydrogen bond interaction is not temperature dependent, as in Refs.<sup>Page 37, 7Page 37, 5Page 38, 4</sup>. If the subkey hb\_temp is included the temperature dependence of the hydrogen bond interaction f<sub>hb</sub>(T) is the same as is described in *the section on the temperature dependent hydrogen bond interaction* (page 8).

Except for COSMO-SAC 2013-ADF, some COSMO-RS specific parameters are used in the next COSMO-SAC methods:

```
Method COSMOSAC
SACPARAMETERS
...
{OMEGA      omega}
{ETA        eta}
End
```

symbol	2013-ADF Xiong	2016-ADF Chen, DHB-ADF Chen, DHB-ADF MESP, 2010 Hsieh, 2007 Wang
omega (ω)		-0.212
eta (η)		-9.00

In ADF2016 these parameters are not used in the COSMO-SAC 2013-ADF method, only in the ADF implementation of the other COSMO-SAC methods. The parameters ω, η and the element specific COSMO-RS dispersion constants are

taken from the COSMO-RS model. The element specific COSMO-RS dispersion constants can be set with the block key DISPERSION.  $\omega$ ,  $\eta$ , and the element specific COSMO-RS dispersion constants are used in a COSMO-RS like method for the calculation of pure compound vapor pressures.

#### 4.2.4 COSMO-SAC element specific parameters

```
Method COSMOSAC2013
EPSILON
  {H epsH}
  {C epsC}
  {N epsN}
  { ... }
End
```

The following table gives the element specific epsilon constants in case of COSMO-SAC 2013-ADF, see Ref.<sup>Page 38, 4</sup>. Like in the COSMO-RS method, pure compound vapor pressures can be given as input, for example, if experimental values are available. In these values are not given, in ADF2016 the pure compound vapor pressure will be approximated using the the COSMO-SAC 2013-ADF method, which depend on these element specific epsilon constants. These constants will also have an effect on the calculated activity coefficients in case of a mixture. Note that these only have an effect in the ADF's COSMO-SAC 2013-ADF implementation.

Table 4.5: COSMO-SAC Element Dispersion Parameters

element	2013-ADF Xiong
	Ref. <sup>Page 38, 4</sup>
H	338.13
C.sp3	29160.92
C.sp2	30951.83
C.sp	20685.98
N.sp3	23488.54
N.sp2	22663.34
N.sp	6390.40
O.sp3-H	8527.06
O.sp3	8484.38
O.sp2	6736.85
O.sp2-N	12145.28
Cl	8435.13
F	82512.21
P	56067.81
S	45065.19
Br	62947.83
I	105910.88

Note that not for all elements in the periodic system COSMO-SAC 2013-ADF parameters were fitted.

If one leaves the EPSILON block keyword empty the contribution of the mixture dispersion to the activity coefficient will be zero.

```
EPSILON
End
```

**Links** COSMO-RS GUI tutorial: Expert option: set COSMO-SAC 2013-ADF parameters [1]

## 4.2.5 Technical and accuracy parameters

```

TECHNICAL
  {RSCONV rsconv}
  {SACCONV sacconv}
  {MAXITER maxiter}
  {BPCONV bpconv}
  {BPMAXITER bpmaxiter}
  {SOLCONV solconv}
  {SOLMAXITER solmaxiter}
  {SOLXILARGE solxilarge}
  {EHDELTA ehdeltaT}
End

```

Table 4.6: Technical Parameters

symbol	Default values
rsconv	$10^{-7}$ kcal/mol
sacconv	$10^{-7}$
maxiter	10000
bpconv	$10^{-6}$ bar
bpmaxiter	40
solconv	$10^{-5}$ molar fraction
solmaxiter	40
solxilarge	0.99 molar fraction
ehdeltaT	1.0 Kelvin

### **rsconv**

Convergence criterion in kcal/mol in chemical potential calculation, not used in COSMO-SAC 2013-ADF. Default value  $1e-7$  kcal/mol.

### **sacconv**

Convergence criterion in activity coefficient calculation, only used in COSMO-SAC 2013-ADF. Default value  $1e-7$ .

### **maxiter**

Maximum number of cycles in chemical potential or activity coefficients calculation. Default value 10000.

### **bpconv**

Convergence criterion (bar) for isobar or solvent boiling point calculation. Default value  $1e-6$  bar.

### **bpmaxiter**

Maximum number of cycles in isobar or solvent boiling point calculation. Default value 40.

### **solconv**

Convergence criterion (molar fraction) used in solubility calculations. Default value  $1e-5$  molar fraction.

### **solmaxiter**

Maximum number of cycles in solubility calculation. Default value 40.

### **solxilarge**

Threshold for (im-)miscibility (molar fraction) in solubility calculations. Above this value the mixture is considered to be fully miscible. Default value 0.99.

### **ehdeltaT**

$\Delta T$  (Kelvin) used in the calculation of the excess enthalpy using the Gibbs-Helmholtz equation and in the calculation of the enthalpy of vaporization using the Clausius-Clapeyron equation using a numerical derivative with respect to  $T$ . Default value 1.0 Kelvin.

Links COSMO-RS GUI tutorial: set COSMO-RS or COSMO-SAC 2013-ADF parameters [1]

## 4.3 Compounds

For each compound one has to add the keyword COMPOUND

```
COMPOUND filename
{cosmofile}
{drophbond}
{NRING nring}
{FRAC1 frac1}
{FRAC2 frac2}
{PVAP pvap}
{TVAP tvap}
{vp_equation equation_name}
{vp_params A B C D E}
{MELTINGPOINT meltingpoint}
{HFUSION hfusion}
{CPFUSION cpfusion}
{FLASHPOINT flashpoint}
{DENSITY density}
{SCALEAREA scalearea}
End
```

### filename

The filename (can be a full path, otherwise relative path is assumed) should be a COSMO result file. How to make an ADF COSMO result file can be found [here](#) (page 17).

### cosmofile

If the subkey cosmofile is included the file should be an ASCII COSMO file (.cosmo). If not specified (default) the file should be a kf file, either an ADF COSMO result file adf.rkf (previously ADF<=2019 TAPE21 file or .t21 file) or a COSKF file (.coskf).

### drophbond

If the subkey drophbond is included no hydrogen-bond terms will be included for this compound. If not specified (default) the hydrogen-bond terms are included for this compound.

### nring

The number of ring atoms. This is a COSMO-RS parameter. It should be 6 for benzene, for example. Default value is 0.

### frac1

The molar fraction of the compound in the solvent (mass fraction if the key MASSFRACTION is used). This is solvent 1 in case of the calculation of partition coefficients (Log P) or in case of a composition line.

### frac2

The molar fraction of solvent 2 (mass fraction if the key MASSFRACTION is used), only used in case of the calculation of partition coefficients (Log P) or in case of a composition line.

### pvap, tvap

Pure compound vapor pressure pvap (bar) at temperature tvap (Kelvin). Used only if both pvap and tvap are specified, and then will have an effect on the calculated vapor pressures or boiling points. Alternative is to set the vp\_equation and vp\_params. If both are not specified the pure compound vapor pressure is approximated using the COSMO-RS method.

### vp\_equation, vp\_params

Specify the vapor pressure correlation to be used for the compound. The equation is chosen via vp\_equation,

and the parameters A, B, C, D, and E are coefficients corresponding to the selected form. The supported equations are:

**Antoine :**

$$\log_{10}(P) = A - \frac{B}{T+C}$$

**VPM1 :**

$$\ln(P) = \frac{A}{T} + B \ln(T) + CT + D$$

**DIPPR101 :**

$$\ln(P) = A + \frac{B}{T} + C \ln(T) + DT^E$$

**DIPPR115 :**

$$\ln(P) = A + \frac{B}{T} + C \ln(T) + D + \frac{E}{T^2}$$

**KDB :**

$$\ln(P) = A \ln(T) + \frac{B}{T} + C + DT^2$$

The units used in the equations are Kelvin (K) for temperature T and bar for pressure P. The provided equation will have an effect on the calculated vapor pressures or boiling points. Alternative is to give input values for the pure compound vapor pressure at a fixed temperature ( $p_{\text{vap}}$  and  $t_{\text{vap}}$ ). If both are not specified the pure compound vapor pressure is approximated using the COSMO-RS method.

For example, to use the Antoine equation for water, define the following:

```
vp_equation Antoine
vp_params 3.55959 643.748 -198.043 0.0 0.0
```

**meltingpoint, hfusion, cpfusion**

Pure compound melting point meltingpoint (Kelvin), pure compound enthalpy of fusion hfusion (kcal/mol), and pure compound heat capacity of fusion cpfusion (kcal/(mol K)). Only used if both meltingpoint and hfusion are specified (cpfusion optional), and will then have an effect in solubility calculations if the temperature of the solvent is below the melting point.

**flashpoint**

Pure compound flash point flashpoint (Kelvin).

**density**

Pure compound density density (kg/L). Used for calculating the volume of a solvent molecule.

**dielectric\_const**

The dielectric constant of the solvent.

**scalearea**

Input scaling of COSMO surface area for a given compound. Default value 1.0 means the COSMO surface area is not scaled. Changing this value is an expert option, for example, to fit to experiment.

**Links** COSMO-RS GUI tutorial: set pure compound parameters [1]

## 4.4 Temperature

```
TEMPERATURE temperature {temperature_high ntemp}
```

### **temperature**

Temperature (Kelvin) at which temperature the COSMO-RS calculation should take place. Default room temperature 298.15. The first temperature in case of a range of temperatures.

### **temperature\_high**

The last temperature (Kelvin) in case of a range of temperatures. Only used in case of solvent vapor pressure calculations or solubility calculations.

### **ntemp**

The number of temperatures in case of a range of temperatures.

## 4.5 Pressure

```
PRESSURE pressure {pressure_high npress}
```

### **pressure**

Pressure (bar) at which pressure the COSMO-RS calculation should take place. Default 1.01325 bar (1 atm). The first pressure in case of a range of pressures.

### **pressure\_high**

The last pressure (bar) in case of a range of pressures. Only used in case of solvent boiling point calculations.

### **npress**

The number of pressures in case of a range of pressures.

## 4.6 Molar fractions and mass fraction

By default the ADF COSMO-RS program assumes molar fractions as input. This can be changed into mass fractions if one includes:

```
MASSFRACTION
```

## 4.7 Properties

Here, we illustrate the required inputs for calculating different properties. Refer to the governing equation at *calculation of properties* (page 12).

### 4.7.1 Vapor pressure

The vapor pressure of a mixture can be calculated with:

```
PROPERTY vaporpressure
End
```

In case of a mixture the mole fraction of each compound of the solvent should be given with the subkey FRAC1 of the key COMPOUND for each compound. In case of a mixture also activity coefficients, and excess energies are calculated.

To calculate pure compound vapor pressures for more than one compound use:

```
PROPERTY purevaporpressure
End
```

It is possible to calculate the vapor pressure for a temperature range, see key TEMPERATURE.

The input pure compound vapor pressure will be used in the calculation of the vapor pressure of this compound if it is supplied with the key COMPOUND for this compound. If it is not specified then it will be approximated using the COSMO-RS method.

**Links** COSMO-RS GUI tutorial: solvent vapor pressure [1, 2]

### 4.7.2 Boiling point

The boiling point of a mixture can be calculated with the block key:

```
PROPERTY boilingpoint
End
```

In case of a mixture the mole fraction of each compound of the solvent should be given with the subkey FRAC1 of the key COMPOUND for each compound. In case of a mixture also activity coefficients, and excess energies are calculated.

To calculate pure compound boiling points for more than one compound use:

```
PROPERTY pureboilingpoint
End
```

It is possible to calculate the boiling temperature for a pressure range, see key PRESSURE.

The input pure compound vapor pressure will be used in the calculation of the vapor pressure of this compound in the mixture if it is supplied with the key COMPOUND for this compound. If it is not specified then it will be approximated using the COSMO-RS method.

The COSMO-RS calculation of the boiling temperature of a solvent is performed with an iterative method. The temperature is varied until the calculated vapor pressure is within a certain threshold of the desired pressure.

**Links** COSMO-RS GUI tutorial: boiling point of a solvent [1]

### 4.7.3 Solvent flash point

The flash point (lower flammable limit) of a compound is the lowest temperature at which the vapor of the compound forms an ignitable mixture in air. The COSMO-RS module can calculate the flash point of a mixture. The COSMO-RS module, however, does not calculate or predict the flash point of pure compounds. The COSMO-RS method is used to calculate the partial vapor pressures of each compound in the mixture, and it uses Le Chatelier's mixing rule to calculate the flash point of this mixture in the gas phase. Input pure compound flash points should be provided by the user, with the subkey FLASHPOINT flashpoint of the key COMPOUND.

```
PROPERTY flashpoint
End
```

The mole fraction of each compound of the solvent should be given with the subkey FRAC1 of the key COMPOUND for each compound.

### 4.7.4 Partition coefficients (LogP)

The partition coefficient of a compound in a mixture of two immiscible solvents, can be calculated with:

```
PROPERTY logp
{VOLUMEQUOTIENT volumequotient}
End
```

#### **volumequotient**

If the subkey VOLUMEQUOTIENT is included the volumequotient will be used for quotient of the molar volumes of solvent 1 and solvent 2 instead of calculated values.

The mole fraction of each compound of the solvent 1 and solvent 2 should be given with the subkey FRAC1 and subkey FRAC2 of the key COMPOUND for each compound, respectively. In case of partly miscible liquids, like, for example, the Octanol-rich phase of Octanol and Water, both components have nonzero mole fractions. The compounds that are included without a given mole fraction are considered to be infinite diluted solutes. The partition coefficients are calculated for all compounds.

One can use some compounds that are present in \$AMSHOME/atomicdata/ADFCRS (Water, 1-Octanol, Benzene, Ethoxyethane, Hexane), or one can use compounds from the ADFCRS-2010 database. For example, for Octanol/Water partition coefficients one can use:

```
Property logp
  VolumeQuotient 4.93
End
Compound "$AMSHOME/atomicdata/ADFCRS/1-Octanol.coskf"
  frac1 0.725
End
Compound "$AMSHOME/atomicdata/ADFCRS/Water.coskf"
  frac1 0.275
  frac2 1.0
End
```

**Links** COSMO-RS GUI tutorial: partition coefficients (log P) [1, 2], Octanol-Water partition coefficients (log P<sub>OW</sub>) [1]

### 4.7.5 Activity coefficients solvent and solute

The mole fraction of each compound of the solvent should be given with the subkey `FRAC1` of the key `COMPOUND` for each compound. The compounds that are included without a given mole fraction are considered to be infinite diluted solutes. The activity coefficients are calculated for all compounds.

```
PROPERTY activitycoef
{DENSITYSOLVENT densitysolvent}
End
```

#### **densitysolvent**

If the subkey `DENSITYSOLVENT` is included the `densitysolvent` will be used for the density of the solvent (kg/L) instead of calculated values. Relevant for the calculation of the Henry's law constant.

The input pure compound vapor pressure will be used in the calculation of the partial vapor pressure of this compound in the mixture if it is supplied with the key `COMPOUND` for this compound. If it is not specified then it will be approximated using the COSMO-RS method. Relevant for the calculation of the Henry's law constant.

The Henry's law constants are calculated in 2 units. The Henry's law constant  $k_H$  is the ratio between the liquid phase concentration of a compound and its partial vapor pressure in the gas phase. The dimensionless Henry's law constant  $k_H^{cc}$  is the ratio between the liquid phase concentration of a compound and its gas phase concentration.

Also calculated is  $\Delta G_{sol}^{liq-solv}$ , which is the solvation Gibbs free energy from the pure compound liquid phase to the solvated phase, and  $\Delta G_{sol}^{gas-solv}$ , which is the solvation Gibbs free energy from the pure compound gas phase to the solvated phase, with a reference state of 1 mol/L in both phases. In addition a Gibbs free energy is calculated which is the free energy of the solvated compound with respect to the gas phase energy of the spin restricted spherical averaged neutral atoms, the compound consist of. Note that zero-point vibrational energies are not taken into account in the calculation of this free energy. This energy could be used in the calculation of  $pK_a$  values.

**Links** COSMO-RS GUI tutorial: activity coefficients [1, 2], solvation free energies [1], Henry's law constants [1],  $pK_a$  values [1]

### 4.7.6 Solubility

The solubility of solutes in 1 mixture can be calculated with:

```
PROPERTY solubility
End
```

The mole fraction of each compound of the solvent should be given with the subkey `FRAC1` of the key `COMPOUND` for each compound, and should add up to 1.0. The solutes should have zero molar fraction in the solvent. The solubility of 1 solute in pure solvents can be calculated with

```
PROPERTY puresolubility
End
```

The mole fraction of each pure solvent should be 1.0, and should be set the subkey `FRAC1` of the key `COMPOUND` for each compound. The solute should have zero molar fraction in the solvent. It is possible to calculate the solubility of a solute at a temperature range, see key `TEMPERATURE`.

For solubility calculations of a solid compound one should add the pure compound melting point  $T_m$ , pure compound enthalpy of fusion  $\Delta H_{fus}$ , and optionally the pure compound heat capacity of fusion  $\Delta C_p$  using the subkeys `melting-point`, `hfusion`, and `cpfusion`, respectively, of the key `COMPOUND` for this compound. The COSMO-RS method does not predict these  $\Delta H_{fus}$ ,  $\Delta C_p$ , or  $T_m$ .

The assumption made in the solubility calculation may be invalid in case of a solubility of a liquid in a solvent, especially if the solubility of the solvent in the liquid is high. For binary systems one may check this by calculating the miscibility gap

in the binary mixture of the two liquids. It is possible to calculate the solubility of a gas in a solvent, if one adds the subkey isobar and adds the partial vapor pressure partialvaporpressure (bar) of the gas as argument for the key PRESSURE:

```
PROPERTY solubility
  isobar
End

PRESSURE partialvaporpressure
```

The solubility of a gas in a solvent can also be calculated using Henry's law, which is valid for ideal dilute solutions, see see the key PROPERTY activitycoef. The COSMO-RS calculation of the solubility of a compound is performed with an iterative method, since the activity coefficient of the compound depends on the molar fraction of this compound.

**Links** COSMO-RS GUI tutorial: solubility [1, 2]

### 4.7.7 Phase Stability Test

The Michelsen Tangent Plane Distance (TPD) stability test can be calculated with:

```
PROPERTY stability
End
COMPOUND "Water.coskf"
  frac1 0.5
End
COMPOUND "1-Octanol.coskf"
  frac1 0.5
End
```

The feed composition is given with the subkey FRAC1 of the key COMPOUND for each compound. The values of FRAC1 should sum to 1.0 over all compounds.

#### Advanced algorithm settings

The Michelsen TPD stability test<sup>1</sup> determines phase stability by searching for stationary points of the tangent-plane distance function  $TPD(w)$  in composition space. If a stationary point is found such that  $TPD(w^*) < 0$ , the mixture is thermodynamically unstable and phase splitting is detected.

The governing equations and the algorithmic workflow are summarized in the flowchart below.

#### Algorithm outline

#### Governing equations

$$TPD(w) = \sum_i w_i (\ln w_i + \ln \gamma_i(w) - \ln z_i - \ln \gamma_i(z)) \quad (S1)$$

$$d_i = \ln z_i + \ln \gamma_i(z) \quad (S2)$$

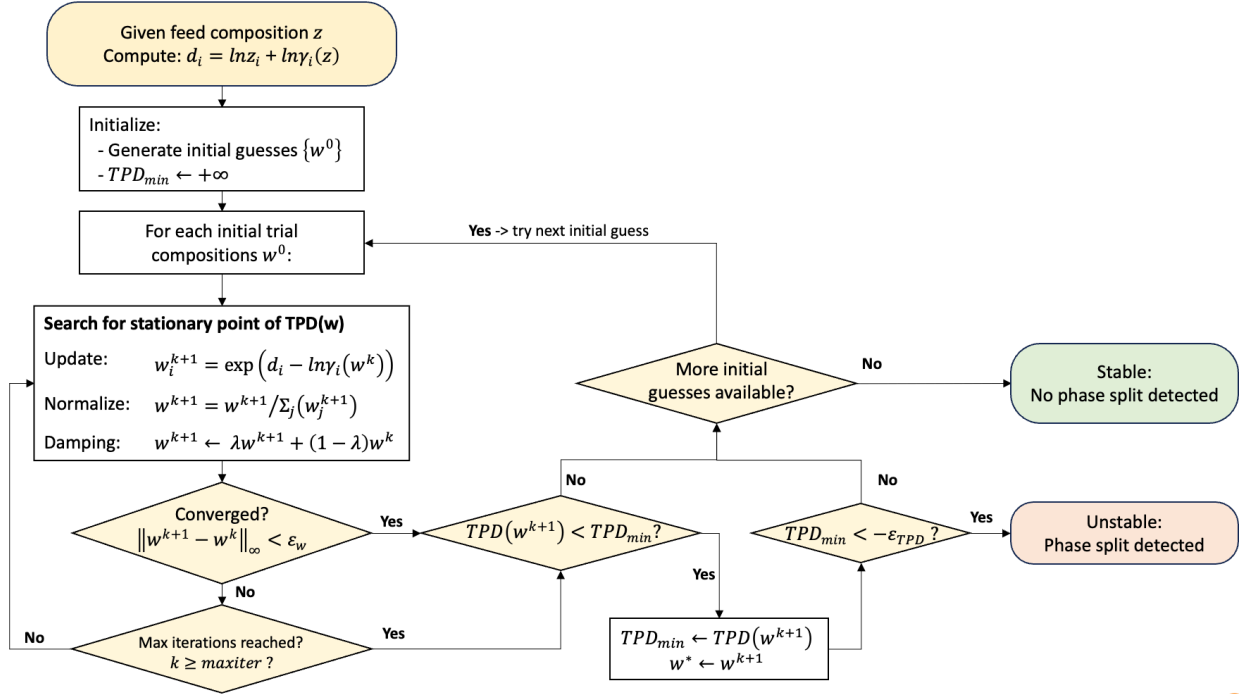
$$w_i^{k+1} = \exp(d_i - \ln \gamma_i(w^k)) \quad (S3)$$

The TPD function (Eq. S1) is evaluated for trial compositions  $w$  relative to the feed composition  $z$ . For convenience,  $d_i$  is defined as in Eq. S2.

<sup>1</sup> Michelsen, M. L.; Mollerup, J. M. Thermodynamic Models: Fundamentals & Computational Aspects; Tie-Line Publications, 2007.

The stationary condition is solved by a fixed-point iteration (Eq. S3), followed by normalization ( $\sum_i w_i = 1$ ) and optional damping with factor  $\lambda$  as shown in the flowchart below.

### Flowchart



Given a feed composition  $z$ , the algorithm first evaluates  $d_i$  using Eq. S2, then generates a set of initial trial compositions  $\{w^0\}$  and initializes  $TPD_{\min} = +\infty$ .

For each initial guess  $w^0$ , it searches for a stationary point of  $TPD(w)$  using the update-normalize-damp cycle shown in the flowchart, until either convergence

$$\|w^{k+1} - w^k\|_{\infty} < \varepsilon_w$$

or the iteration limit is reached ( $k \geq \text{maxiter}$ ).

After each trial, if a lower value is found, the best-so-far minimum is updated:

$$TPD_{\min} \leftarrow TPD(w^{k+1}), \quad w^* \leftarrow w^{k+1}.$$

After all initial guesses are processed, the mixture is classified as unstable if

$$TPD_{\min} < -\varepsilon_{tpd}.$$

Otherwise, it is classified as stable (no phase split detected).

### Technical Parameters

## TPD solver parameters

**eps\_tpd**

Instability tolerance  $\varepsilon$ . The mixture is reported as unstable if  $TPD(w^*) < -\varepsilon$ . (default: 1e-5)

**eps\_w**

Convergence tolerance  $\varepsilon_w$  for the fixed-point iteration: stop when  $\|w^{k+1} - w^k\|_\infty < \varepsilon_w$ . (default: 1e-8)

**maxiter**

Maximum number of iterations per initial trial composition. (default: 500)

**lam**

Damping factor  $\lambda$  used in the relaxed update of  $w$ . (default: 0.7)

**xmin / gammin**

Lower bound for compositions/activity coefficients to avoid  $\log(0)$ , division by zero, and numerical instability; values below this threshold are clamped and renormalized. (default: xmin = 1e-8, gammin = 1e-8)

## Optional initial guess

**tpd\_w**

User-supplied trial composition  $w_i$  of component  $i$  for the Michelsen TPD stability test (molar or mass fraction). (default: 0.0, meaning no user-supplied initial guess)

If `tpd_w` values are provided for all compounds, they should represent a valid composition (sum to 1.0). If no `tpd_w` values are provided, a built-in default set of trial-phase compositions is used.

## Example

```
PROPERTY stability
End
COMPOUND "Water.coskf"
  frac1 0.5
  tpd_w 0.1
End
COMPOUND "1-Octanol.coskf"
  frac1 0.5
  tpd_w 0.9
End

TECHNICAL
  TPD
    eps_tpd 1e-5
    eps_w 1e-8
    maxiter 200
    lam 0.7
  END
END
```

**Links** COSMO-RS GUI tutorial: phase stability test [1]

## 4.7.8 Liquid-Liquid Equilibrium (LLE)

The liquid-liquid equilibrium (LLE) of a multicomponent mixture can be calculated with:

```
PROPERTY lle
End
COMPOUND "Water.coskf"
  frac1 0.5
End
COMPOUND "1-Octanol.coskf"
  frac1 0.5
End
```

The feed composition is given with the subkey FRAC1 of the key COMPOUND for each compound. The values of FRAC1 should sum to 1.0 over all compounds.

### Optional initial guess

The initial guesses for the phase split composition can be supplied:

#### lle\_xI0

Initial guess for phase-I composition  $x_i^I$  for component  $i$  (molar or mass fraction). (default: 0.0 [no user-supplied initial guess])

If no initial guesses are provided, a stability test is used to generate a starting split. If the feed is stable (single phase), the calculation reports “no phase split detected”.

---

**Note:** The current implementation solves only **two-phase liquid–liquid equilibrium (LLE)**.

For systems exhibiting **three-liquid-phase equilibrium (LLLE)**, the solver does not perform a global multi-phase stability analysis. While a two-phase solution may still be obtained, the presence of an additional liquid phase cannot be automatically identified by the current algorithm.

---

### Advanced algorithm settings

The LLE solver iterates on the distribution coefficients  $K$  to satisfy the isoactivity condition<sup>Page 48, 1</sup>. The governing equations and the algorithmic workflow are summarized in the flowchart below.

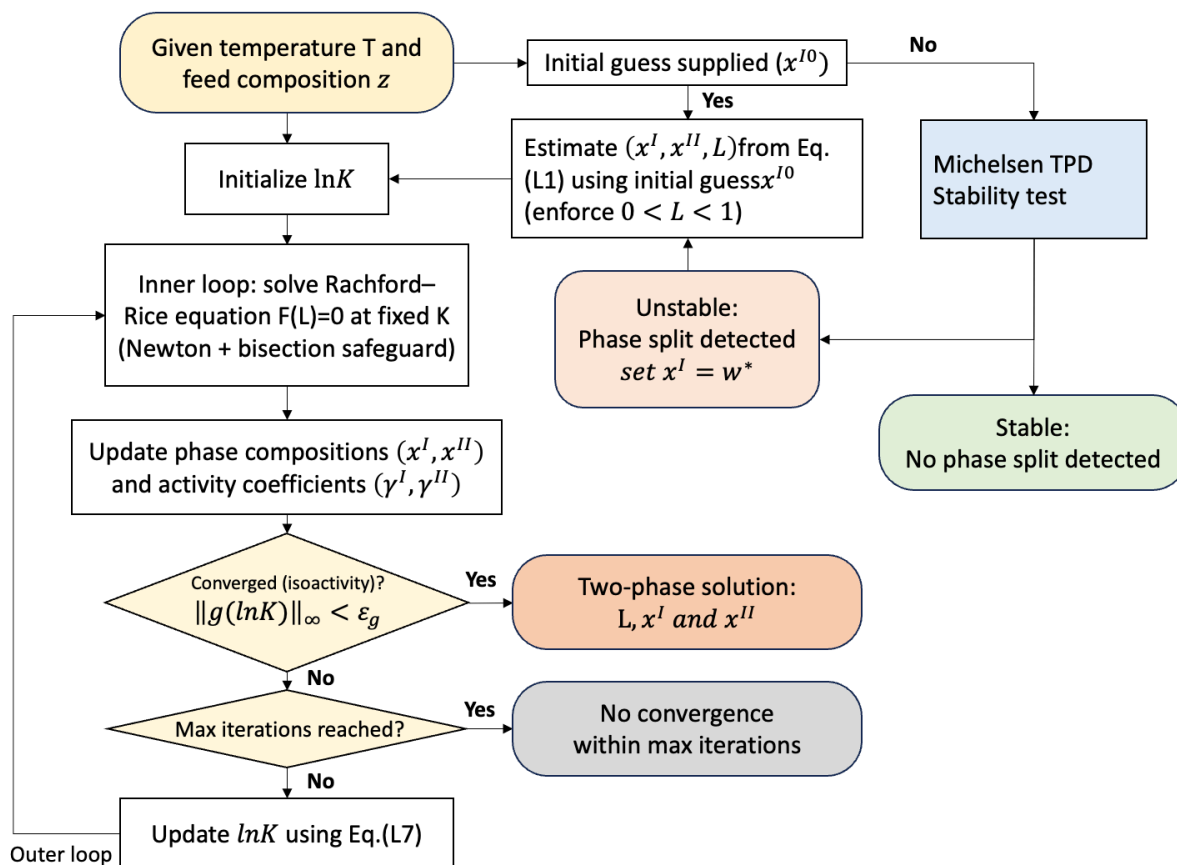
#### Algorithm outline

## Governing equations

<b>Convention</b>	
$z_i = (1 - L)x_i^I + Lx_i^{II}$	(L1)
$K_i = x_i^{II}/x_i^I$	(L2)
<b>Rachford–Rice equation (<math>\sum x_i^I = 1</math>):</b>	
$F(L) = \sum \frac{z_i(K_i - 1)}{(1 + L(K_i - 1))} = 0$	(L3)
$x_i^I = \frac{z_i}{(1 + L(K_i - 1))}$	(L4)
$x_i^{II} = K_i x_i^I$	(L5)
<b>Isoactivity criterion</b>	
$g_i(\ln K) = \ln(x_i^{II} \gamma_i^{II}) - \ln(x_i^I \gamma_i^I)$ $\mathbf{g}(\ln K) = 0$	(L6)
<b>lnK update</b>	
$\ln K^{new} = \ln K^{old} - \lambda * \mathbf{g}(\ln K)$	(L7)
<b>Step-size strategy (<math>\lambda</math>)</b>	
(1) $\lambda = \text{Constant}$	
(2) $\lambda = \frac{\langle \Delta \ln K, \Delta \mathbf{g} \rangle_z}{\langle \Delta \mathbf{g}, \Delta \mathbf{g} \rangle_z}$	
(3) $\lambda^{new} = \lambda^{old} \times \frac{\langle \mathbf{g}^{old}, \mathbf{g}^{old} \rangle_z}{\langle \mathbf{g}^{old}, (\mathbf{g}^{old} - \mathbf{g}^{new}) \rangle_z}$	
Safeguard :	
$\lambda \leftarrow \min \left( \lambda, \frac{d \ln K_{max}}{\max_i  g_i } \right)$	
$\lambda \leftarrow \text{clip}(\lambda, \lambda^{min}, \lambda^{max}) \text{ if } use\_step\_bounds$	

The solver uses the convention in Eqs. L1-L2, and solves the Rachford–Rice equation  $F(L) = 0$  at fixed  $K$  (Eq. L3) to obtain phase compositions  $x^I$  and  $x^{II}$  (Eqs. L4–L5). The outer iteration updates  $\ln K$  using the isoactivity residual  $g$  (Eq. L6) via the step-controlled update in Eq. L7.

### Flowchart



An initial guess can be supplied by the user via `lle_xI0`. If no guess is provided,  $w^{\min}$  is obtained from the TPD stability test and used as the initial phase-split composition.

At each outer iteration, convergence is checked using the log-isoactivity residual  $g_i = \ln(x_i^{II}\gamma_i^{II}) - \ln(x_i^I\gamma_i^I)$  (Eq. L6). The solver reports convergence when  $\max_i |g_i| < \epsilon_g$ .

### Technical Parameters

#### Outer-iteration convergence

##### `eps_g`

Convergence tolerance  $\epsilon_g$  on the log-isoactivity residual (Eq. L6): the solver checks  $\max_i |g_i|$ . (default:  $1e-6$ )

##### `maxiter`

Maximum number of outer LLE iterations (lnK updates). (default: 2000)

## Step-size control for the lnK update

### lnK\_step\_scheme

Step-size scheme for the outer update  $\ln K \leftarrow \ln K - \lambda g$ . Default: 3.

- 1: fixed step (use `step_lnK`)
- 2: BB-short (Barzilai–Borwein short step)
- 3: Mehra Algorithm III (adaptive; see Ref.<sup>2</sup>)

### step\_lnK

Step size  $\lambda$  for scheme 1. For other schemes it is used as the initial value. (default: 1.0)

### dlnK\_max

Limiter on the maximum  $\ln K$  change per iteration. The limiter follows Mehra et al. (1983, Eq. 40)<sup>Page 54, 2</sup>. Default: 6.0.

$$\max_i |\Delta \ln K_i| \leq d\ln K_{\max}$$

### use\_step\_bounds

Enable clipping of the step size  $\lambda$  to the interval `[step_min, step_max]`. Default: `false`.

### step\_min / step\_max

Bounds for the step size  $\lambda$  (clipping) to avoid stagnation (too small) and overly aggressive updates (too large). (default: `step_min = 0.002, step_max = 500.0`)

## Inner solve for phase fraction L (Rachford–Rice)

### tolF

Residual tolerance for the Rachford–Rice equation  $F(L)$ . Convergence is accepted when  $|F(L)| < \text{tolF}$ . (default: `1e-8`)

### tolL

Tolerance for phase-fraction  $L$  uncertainty (root-bracketing interval width); convergence is accepted when it is below `tolL`. (default: `1e-6`)

### itLmax

Maximum number of  $L$  root-finding iterations per LLE step. (default: `200`)

### Lmin

Minimum allowed phase fraction (physical/model bound). (default: `1e-6`)

## Robustness bounds

### xmin

Numerical floor for LLE phase mole fractions (used in clipping, normalization, and log terms). (default: `1e-8`)

### gammin

Numerical floor for activity coefficients used in log terms. (default: `1e-8`)

### margin

Numerical safety margin in the  $L$  solver to keep denominators positive. (default: `1e-12`)

### Lnum\_min

Numerical lower bound for phase-fraction safety; protects against an overly small `Lmin`. (default: `1e-12`)

<sup>2</sup> Mehra, R. K.; Heidemann, R. A.; Aziz, K. An Accelerated Successive Substitution Algorithm. *The Canadian Journal of Chemical Engineering* 1983, 61 (4), 590–596. (<https://doi.org/10.1002/cjce.5450610414>)

## Notes

- `lnK_step_scheme = 3` often improves convergence speed, but it may still fail for some cases.
- If the outer iteration fails to converge, first reduce `dlnK_max` (and, if needed, increase `maxiter`), then try a different `lnK_step_scheme`.
- If the inner L-solver fails to converge, try increasing `itLmax` or relaxing `tolF/tolL`.

## Example

```
PROPERTY lle
End
COMPOUND "Water.coskf"
  frac1 0.5
  lle_xI0 0.275
End
COMPOUND "1-Octanol.coskf"
  frac1 0.5
  lle_xI0 0.725
End

TECHNICAL
  LLE
    eps_g 1e-6
    maxiter 2000
    lnK_step_scheme 3
    step_lnK 1.0
    dlnK_max 6.0
    tolF 1e-8
    itLmax 200
  END
END
```

`lle_xI0` is an optional initial guess.

**Links** COSMO-RS GUI tutorial: liquid-liquid equilibrium [1, 2]

### 4.7.9 Binary mixture (VLE/LLE)

The COSMO-RS module can automatically calculate properties of a binary mixture, by calculating these properties for a number of different compositions.

Exactly two compounds should be given in the input file.

```
PROPERTY binmixcoef
{Nfrac nfrac}
{isotherm | isobar | flashpoint}
End
```

#### **nfrac**

Number of different mixtures for which the binary mixture is calculated will be `nfrac+5`. Default value for `nfrac` is 10, which means 15 different mixtures.

#### **isotherm | isobar | flashpoint**

If the subkey `isotherm` is included (default) the binary mixture will be calculated at a fixed temperature. If the

subkey isobar is included the binary mixture will be calculated at a fixed vapor pressure. If the subkey flashpoint is included the flash point of the binary mixture will be calculated.

The input pure compound vapor pressure will be used in the calculation of the partial vapor pressure of this compound in the mixture if it is supplied with the key COMPOUND for this compound. If it is not specified then it will be approximated using the COSMO-RS method.

In case of a miscibility gap (LLE) data of the 2 immiscible liquid phases will be calculated. Also information about possible azeotropes will be calculated. With the COSMO-RS GUI, activity coefficients, (partial) vapor pressures, and excess energies can be viewed.

**Links** COSMO-RS GUI tutorial: vapor-liquid diagram binary mixture (VLE/LLE) [1, 2]

#### 4.7.10 Ternary mixture (VLE/LLE)

The COSMO-RS module can automatically calculate properties of a ternary mixture, by calculating these properties for a number of different compositions. Exactly three compounds should be given in the input file.

```
PROPERTY ternarymix
  {Nfrac nfrac}
  {isotherm | isobar | flashpoint}
End
```

##### **nfrac**

Number of different mixtures for which the ternary mixture is calculated will be  $(nfrac+1)*(nfrac+2)/2$ . Default value for nfrac is 10, which means 55 different mixtures.

##### **isotherm | isobar | flashpoint**

If the subkey isotherm is included (default) the ternary mixture will be calculated at a fixed temperature. If the subkey isobar is included the ternary mixture will be calculated at a fixed vapor pressure. If the subkey flashpoint is included the flash point of the ternary mixture will be calculated.

The input pure compound vapor pressure will be used in the calculation of the partial vapor pressure of this compound in the mixture if it is supplied with the key COMPOUND for this compound. If it is not specified then it will be approximated using the COSMO-RS method.

In case of a miscibility gap liquid-liquid equilibrium (LLE) data such as tie lines and an approximate phase diagram, are calculated. With the COSMO-RS GUI, activity coefficients, (partial) vapor pressures, and excess energies can be viewed as a colormap in a 2-dimensional plot with 2 of the liquid compositions on the axes.

**Links** COSMO-RS GUI tutorial: ternary mixtures (VLE/LLE) [1]

#### 4.7.11 Solvents s1 - s2 Composition Line

The COSMO-RS module can linear interpolate between the compositions of solvent 1 and solvent 2, which both could be mixtures, and calculate properties, like activity coefficients, and excess energies. This property calculation does not calculate a possible miscibility gap. The mole fraction of each compound of the solvent 1 and solvent 2 should be given with the subkey FRAC1 and subkey FRAC2 of the key COMPOUND for each compound, respectively.

```
PROPERTY compositionline
  {Nfrac nfrac}
  {isotherm | isobar | flashpoint}
End
```

##### **nfrac**

Number of different mixtures of the 2 solvents is calculated will be  $(nfrac+1)$ . Default value for nfrac is 10, which means 11 different mixtures.

**isotherm | isobar | flashpoint**

If the subkey isotherm is included (default) a fixed temperature will be used. If the subkey isobar is included a fixed vapor pressure will be used. If the subkey flashpoint is included the flashpoint will be calculated.

**Links** COSMO-RS GUI tutorial: A composition line between solvents s1 and s2 [1]

## 4.8 Analysis

### 4.8.1 Sigma profile

The sigma profile of a mixture can be calculated with:

```
PROPERTY sigmaprofile
{Nprofile nprofile}
{SigmaMax sigmamax}
End
```

In case of a mixture the mole fraction of each compound in the mixture should be given with the subkey FRAC1 of the key COMPOUND for this compound.

The sigma profile pure compounds can be calculated with:

```
PROPERTY puresigmaprofile
{Nprofile nprofile}
{SigmaMax sigmamax}
End
```

**nprofile**

Number of data points for which to calculate the sigma profile. default value 50.

**sigmamax**

The sigma profile is calculated between -sigmamax and sigmamax. Default value 0.025.

The hydrogen bonding part (HB) of the sigma profile(s) will also be calculated. In case of a COSMO-SAC 2013-ADF calculation also the OH component of hydrogen bonding (HB-OH) is calculated, and the other type component of hydrogen bonding (HB-OT) is calculated.

**Links** COSMO-RS GUI tutorial: sigma profile [1]

### 4.8.2 Sigma potential

The sigma potential of a mixture can be calculated with:

```
PROPERTY sigmapotential
{Nprofile nprofile}
{SigmaMax sigmamax}
End
```

In case of a mixture the mole fraction of each compound in the mixture should be given with the subkey FRAC1 of the key COMPOUND for this compound.

The sigma profile pure compounds can be calculated with:

```
PROPERTY puresigmapotential
{Nprofile nprofile}
{SigmaMax sigmamax}
End
```

**nprofile**

Number of data points for which to calculate the sigma potential. default value 50.

**sigmamax**

The sigma potential is calculated between -sigmamax and sigmamax. Default value 0.025.

**Links** COSMO-RS GUI tutorial: sigma potential [1]

## 4.9 Other inputs

### 4.9.1 Pitzer-Debye-Hückel long-range electrostatic correction

```
PDH_CORRECTION Yes/No
```

**PDH\_CORRECTION**

**Type**

Bool

**Default value**

No

**Description**

Logical to use Pitzer Debye-Hueckel electrostatic correction

### 4.9.2 Elbro Combinatorial term for polymers

```
USEPOLYCOMBIFORPOLYMER Yes/No
```

**USEPOLYCOMBIFORPOLYMER**

**Type**

Bool

**Default value**

Yes

**Description**

Logical to use polymer-specific combinatorial term

### 4.9.3 Reuse generated sigma profile

REUSE\_SIGMA\_PROFILE Yes/No

#### REUSE\_SIGMA\_PROFILE

**Type**

Bool

**Default value**

Yes

**Description**

Logical to enable reuse the generated sigma profile stored in the COSKF file.

### 4.9.4 Update generated sigma profile

UPDATE\_SIGMA\_PROFILE Yes/No

#### UPDATE\_SIGMA\_PROFILE

**Type**

Bool

**Default value**

No

**Description**

Logical to update (overwrite) the generated sigma profile in the COSKF file if model parameters have changed.

---

**Note:** In the AMScrs GUI, these settings can be changed via Method → Parameters.

---



## POLYMERS WITH COSMO-RS(-SAC)

Many types of polymers can be described with COSMO-RS theory. However, the typical procedure for calculating  $\sigma$ -profiles (a geometry optimization with ADF, a single point COSMO calculation, processing the surface points to make a  $\sigma$ -profile) is far too expensive for generating the  $\sigma$ -profiles of polymers. Instead, the  $\sigma$ -profile of a polymer can be represented as the  $\sigma$ -profile of a monomer scaled to the size of the polymer. This means that given a  $\sigma$ -profile of an “average monomer,” we can generate a  $\sigma$ -profile for the polymer by multiplying the monomer  $\sigma$ -profile by a factor equal to the number of repeat units in the polymer. This procedure, though not without some shortcomings, makes the treatment of polymers with COSMO-RS computationally tractable.

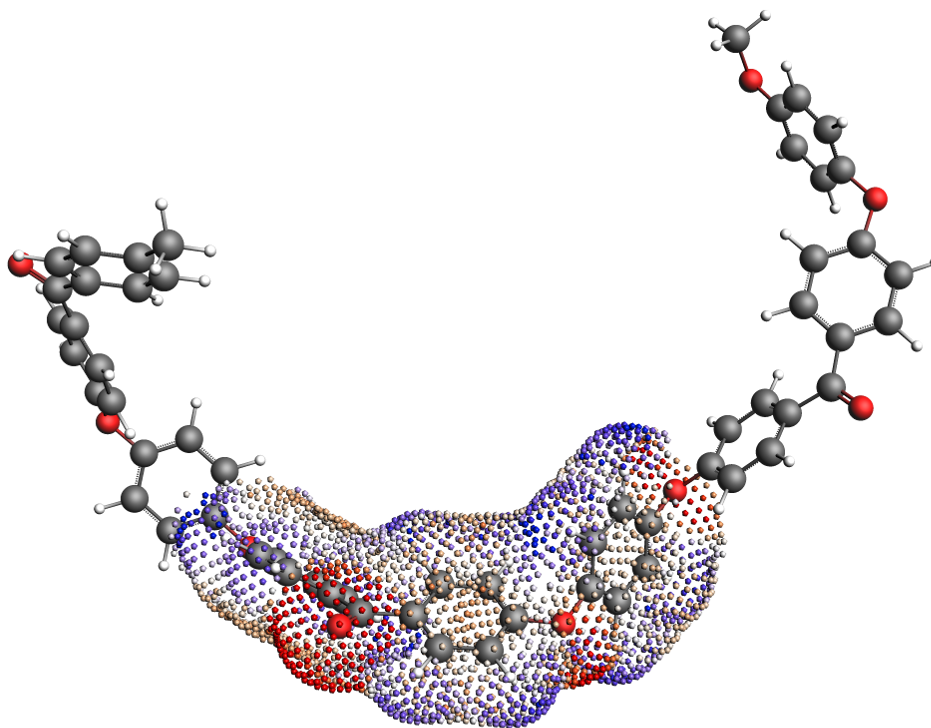


Fig. 5.1: COSMO surface of the Polyether ether ketone (PEEK) monomer within a trimer. The “average monomer”  $\sigma$ -profile is calculated from the central monomer unit of the trimer.

There are many possible approaches to generating the  $\sigma$ -profile for an “average monomer.” The approach used to generate the monomers in the 2019 ADFCRS-Polymer Database is the following:

- (1) Construct a trimer from 3 units of the monomer

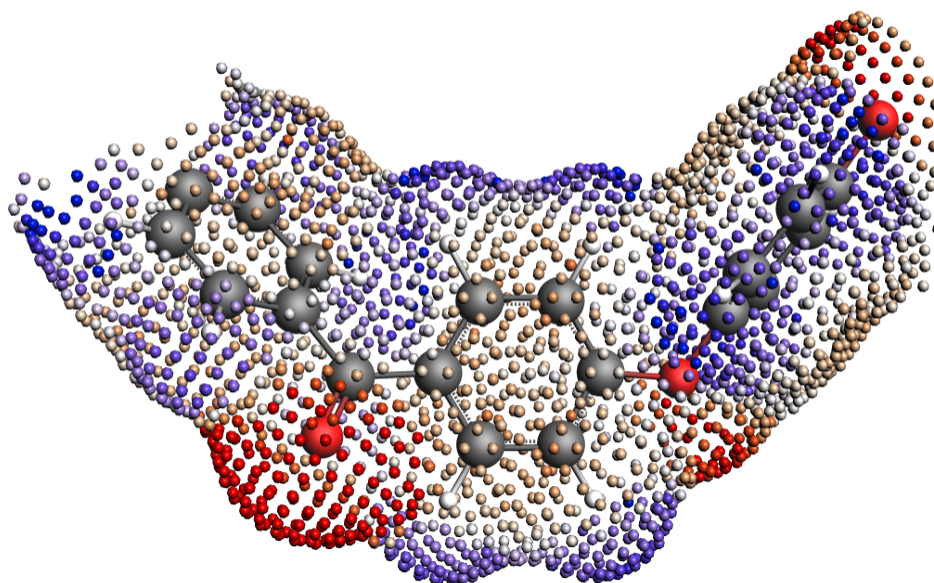


Fig. 5.2: COSMO surface of the isolated Polyether ether ketone (PEEK) monomer.

- (2) Cap the ends of the outer two monomers with methyl groups
- (3) Optimize the geometry of the trimer with ADF
- (4) Do a single point COSMO calculation
- (5) Generate the  $\sigma$ -profile for *only the central unit* of the trimer

This method provides the  $\sigma$ -profile of a monomer surrounded by two copies of itself on either side. Though perhaps longer polymer chains could be considered in the determination of the “average monomer,” the trimer representation was found to be sufficiently accurate for calculating various thermodynamic properties while not being exorbitantly expensive in the ADF step. In cases where this monomer representation fails to capture the behavior of the polymer, the following alternative procedures are possible:

- Generating a structure from a longer polymer chain and taking the “average monomer”  $\sigma$ -profile as an average of the  $\sigma$ -profiles of all of the interior monomers
- Sampling the conformational space of the polymer and using different structures for different problems (e.g., a polymer with a possible intramolecular H-bond may exhibit this internal H-bonding in a neutral solvent but not in a more polar one)

## 5.1 Additional properties/units for polymer systems

### 5.1.1 Average molecular weight

As mentioned above, polymers in COSMO-RS are calculated using a scaled version of a monomer’s  $\sigma$ -profile. However, because polymers come in many different lengths, we must be able to adjust the length of the polymer and accordingly adjust the scaling of the monomer  $\sigma$ -profile. This is done with an average molecular weight parameter, which represents

the Number Average Molecular Weight  $M^N$ , defined as the following:

$$M^N = \frac{\sum_j n_j w_j}{\sum_j n_j}$$

where  $j$  is in index over all different molecules (different-length polymers are different molecules) in solution,  $w_j$  is the weight of each molecule, and  $n_j$  is the number of molecules of type  $j$  in solution. Because this parameter is adjustable, the same monomer  $\sigma$ -profile can be used for calculations with polymers of different lengths. Note that a default value of 10000 g/mol is used if no input is given.

## 5.1.2 Mole fractions of the monomers and polymers

Because a typical polymer consists of many polymer chains of different lengths, the mole fraction of the polymer is well-defined. In the COSMO-RS program, there are two mole fractions defined:

### x(monomer)

x(monomer) is simply the mole fraction of a monomer in the solution, treating all polymeric species as a number of monomers equal to the length of the polymer chain. For non-polymeric components, x(monomer) is the mole fraction in relation to a solution of monomers.

### x(polymer)

x(polymer) defines a mol of each polymer chain using the average molecular weight parameter to determine the average chain length. Specifically, we can define a term  $R_i$  as the number of repeat units in the average polymer of species  $i$ . For non-polymeric components,  $R_i = 1$ .  $R_i$  is shown below:

$$R_i = \frac{M_i^N}{M(\text{monomer})_i}$$

where  $M(\text{monomer})_i$  refers to the molar mass of the monomer. Using this  $R_i$ , we can define  $x(\text{polymer})_i$ :

$$x(\text{polymer})_i = \frac{x(\text{monomer})_i / R_i}{\sum_j x(\text{monomer})_j / R_j}$$

where now  $j$  is an index over all distinct components. All polymers of the same type are assumed to be of length  $R_i$ .

## 5.1.3 Weight- and volume-fraction activity coefficients

As a further consequence of the ill-defined mole fraction for polymer solutions, activity coefficients are often reported in the literature in terms of weight- or volume-fraction. These are also reported for calculations in the COSMO-RS program in which at least one species is a polymer. They are defined from the activity of each species:

$$a_i = \gamma_i x(\text{polymer})_i$$

where  $\gamma_i$  is the activity coefficient in relation to the polymer mole fractions. Note: this is the default value for  $\gamma_i$  in polymer calculations. The weight-fraction activity coefficient ( $\Omega_i$ ) can then be calculated as:

$$\Omega_i = \frac{a_i}{w_i}$$

where  $w_i$  is the weight fraction of component  $i$ . Finally, the volume fraction activity coefficient ( $\gamma_i^\phi$ ) can be defined as follows:

$$\gamma_i^\phi = \frac{a_i}{\phi_i}$$

where  $\phi_i$  is the volume fraction of component  $i$ .

### 5.1.4 Flory-Huggins parameter

The Flory-Huggins model is used extensively for binary polymer/solvent and polymer/polymer mixtures. In the Flory-Huggins model, the  $\chi$  parameter is the single system-specific parameter and is intended to quantify the enthalpic interactions between system components. Users of the Flory-Huggins model can deduce important system behavior (phase stability, solubility, etc.) from the value of the  $\chi$  parameter.

The  $\chi$  parameter was originally intended to be composition- and temperature- independent, although it is now known to vary significantly across compositions and temperatures for many systems. Fortunately, system changes related to composition and temperature are captured in the COSMO-RS program and are reflected in the calculation of the Flory-Huggins  $\chi$  parameter. This is due to the calculation of the  $\chi$  parameter from the free energy of mixing of two species:

$$\frac{G_{mix}}{RTV} = \frac{\phi_1}{v_1} \ln(\phi_1) + \frac{\phi_2}{v_2} \ln(\phi_2) + \frac{\phi_1 \phi_2 \chi_{12}}{v_r}$$

where  $G_{mix}$  is the free energy of mixing,  $R$  is the gas constant,  $T$  is the absolute temperature,  $V$  is the system volume,  $\phi_i$  is the volume fraction of component  $i$ ,  $v_i$  is the molar volume of species  $i$ , and  $v_r$  is a reference volume. Note that the difference in  $v_1$  and  $v_2$  can be significant in the case of a polymer/solvent mixture. Since  $G_{mix}$  does vary with composition and temperature, the  $\chi$  parameter will also exhibit this variance.

---

#### Important: Reference volumes ( $v_r$ ) used in calculating the Flory-Huggins parameter

- polymer/solvent systems:  $v_r$  is equal to the molar volume of the solvent
  - polymer/polymer systems:  $v_r$  is set to a value of 0.6022140857 L/mol, which corresponds to a site size of 1 nm<sup>3</sup>
- 

The COSMO-RS program can calculate thermodynamic properties for multi-component systems, including polymer-containing systems. However, the Flory-Huggins  $\chi$  parameter is traditionally defined for binary mixtures (although some extensions to multi-component mixtures exist). In the case of multi-component mixtures containing polymers, we calculate a  $\chi$  parameter for each species. This is summarized in the following:

---

#### Note: Calculating the Flory-Huggins parameter for systems with more than 2 components

For systems with more than 2 components, a  $\chi$  parameter is defined for each component. This  $\chi_i$  parameter is calculated as the  $\chi$  parameter for a species  $i$  as if all the other species in solution were combined into one component. This means that for a ternary mixture of solvent/polymer A/polymer B, the  $\chi$  parameter of the solvent would correspond to the  $\chi$  parameter of the system defined by the solvent and the *co-polymer* of A and B. Similarly, the  $\chi$  parameter of polymer A would correspond to the  $\chi$  parameter of the binary system of polymer A and the meta-solvent of a combination of polymer B and the solvent.

---

## 5.2 Modified combinatorial term

The entropy of polymer solutions cannot be calculated in the same way as the entropy of small molecule solutions. For this reason, the normal combinatorial term in COSMO-RS is replaced with a polymer-specific combinatorial term. Specifically, the combinatorial term used is that of Elbro (1990)<sup>1</sup>, which has previously been applied successfully in the context of COSMO-RS<sup>2</sup>. The Elbro combinatorial requires the free volume fraction ( $\phi_i^{fv}$ ) of each component:

$$\phi_i^{fv} = \frac{x_i(v_i - v_i^*)}{\sum_j x_j(v_j - v_j^*)}$$

<sup>1</sup> H.S. Elbro, A. Fredenslund, and P. Rasmussen, *A new simple equation for the prediction of solvent activities in polymer solutions*, *Macromolecules* 23, 4707 (1990) (<https://doi.org/10.1039/c7cp02317k>)

<sup>2</sup> C. Loschen and A. Klamt, *Prediction of solubilities and partition coefficients in polymers using COSMO-RS*, *Industrial & Engineering Chemistry Research* 53, 11478 (2014) (<https://doi.org/10.1021/ie501669z>)

where  $x_i$  is the (polymer) mole fraction of compound  $i$ ,  $v_i$  is the molar volume of compound  $i$ ,  $v_i^*$  is the molar hard-core volume of compound  $i$ , and  $j$  is also an index over the compounds. Using the values of  $\phi_i^{fv}$  for each component, the combinatorial contribution to the activity coefficient can be defined as follows:

$$\ln(\gamma_i^{fv}) = \ln\left(\frac{\phi_i^{fv}}{x_i}\right) + 1 - \frac{\phi_i^{fv}}{x_i}$$

Finally, this combinatorial term is used to calculate the final activity coefficient:

$$\ln(\gamma_i) = \ln(\gamma_i^{crs}) + \ln(\gamma_i^{fv})$$

where  $\gamma_i^{crs}$  is the activity coefficient calculated with COSMO-RS without any combinatorial corrections.

### 5.3 General application guidelines/warnings

At present, the quality of the predictions of the COSMO-RS program is *unreliable* or *untested* for the following types of systems:

- **Cross-linked polymers.** The structures of cross-linked polymers are not perfectly compatible with the assumptions in the COSMO-RS model for polymers. These types of systems are not recommended to be used with COSMO-RS.
- **Polymers with significant swelling behavior.** Polymer swelling is not easily captured with COSMO-RS. It is likely that some of the error due to swelling can be offset by using correct molar volume values for the polymers in different solvents/at different temperatures.

### 5.4 Downloading the ADFCRS-POLYMER-2019 database

See the *section on the ADFCRS-POLYMER-2019 database* (page 26) on how to download this database.

### 5.5 Tutorial on polymer calculations

There is a [tutorial on COSMO-RS calculations with polymers](#) that demonstrates basic the basic functionality of the COSMO-RS GUI with polymers. This will cover calculations using the polymer database as well as *inputting your own polymer structures*.



## PITZER-DEBYE-HÜCKEL LONG-RANGE ELECTROSTATIC CORRECTION

Systems with charged species often demonstrate behavior that is not easily captured with standard thermodynamic models meant for neutral molecules. A major reason for this that the chemical potential requires an additional contribution due to the electrostatic potential. The electrostatic potential decays with  $r^{-1}$ , where  $r$  is the distance between two charges. This inverse dependence on  $r$  makes the electrostatic potential a long-range term, especially as compared to other intermolecular energetic contributions, which are typically only relevant at short distances. One successful model for adding necessary electrostatic corrections for systems with charged species is the Pitzer-Debye-Hückel (PDH)<sup>1</sup> model. In the COSMO-RS package, we use a modification of the PDH term that allows for mixed solvents.

### 6.1 Mixing rules and required property inputs for the PDH term

In the original publication, the PDH model is used only for pure solvents. To generalize to mixed solvent systems, we have included mixing rules to estimate required parameters for the PDH model. These are discussed below in addition to required parameters.

#### 6.1.1 Molecular weight

No additional input is needed for this property as this is directly determined by the atomic composition of a molecule. The average molecular weight for the entire system,  $M$ , is given as a function of all the individual molecular weights  $m_i$  :

$$M = \frac{\sum_i n_i m_i}{\sum_i n_i}$$

where  $n_i$  is the number of moles of species  $i$ .

#### 6.1.2 Density

The density of each compound is also required for the calculation of the PDH term. For ionic species, one may simply use the density of the combined salt. The average density for the entire system,  $D$ , is given as a function of all the individual densities  $d_i$  :

$$D = \frac{\sum_i n_i}{\sum_i \frac{n_i}{d_i}}$$

<sup>1</sup> Pitzer, Kenneth S., and John M. Simonson. *Thermodynamics of multicomponent, miscible, ionic systems: theory and equations*. *The Journal of Physical Chemistry* 90 (1986): 3005-3009. (<https://doi.org/10.1021/j100404a042>)

### 6.1.3 Dielectric constant

Additionally, the dielectric constant of each compound is required for the PDH term. These terms do not need to be experimentally accurate numbers, but should be approximate. The dielectric constant of a combined salt can be used for the individual ionic species. The average dielectric constant for the entire system,  $\mathcal{E}$ , is given as a function of all the individual dielectric constants  $\epsilon_i$  :

$$\mathcal{E} = \frac{\sum_i n_i \frac{m_i \epsilon_i}{d_i}}{\sum_i n_i \frac{m_i}{d_i}}$$

## 6.2 Derivation of the PDH term for general mixtures

In<sup>Page 67, 1</sup>, the Debye-Hückel model is adapted to account for higher concentrations of ions (the Debye-Hückel model works well for low concentrations). The authors choose from a variety of possible models, ultimately deciding on one that is deemed most accurate empirically. The expression for the PDH excess energy (long range electrostatic correction) is given below.

$$\frac{G^{PDH}}{RT} = -4 \left( \frac{1000}{M} \right)^{1/2} \frac{N A_x I_x}{\rho} \ln(1 + \rho I_x^{1/2})$$

where  $I_x$  is the ionic strength of the system, which is a function of the charge  $z_i$  of each species:

$$I_x = \frac{1}{2} \frac{\sum_i n_i z_i^2}{\sum_i n_i}$$

and  $A_x$  is the Debye-Hückel parameter, defined as:

$$A_x = \frac{1}{3} \frac{(2\pi N_A D e^6)^{1/2}}{(4\pi \epsilon_0 \mathcal{E} k T)^{3/2}}$$

where  $e$  is the charge of the electron,  $N_A$  is Avogadro's number,  $k$  is the Boltzmann constant, and  $\epsilon_0$  is the permittivity of free space.

Finally,  $\rho$  is the closest approach parameter. The specifics of this parameter are still under development, and for now the following simple formula from<sup>Page 67, 1</sup> is used.

$$\rho = 2150 \left( \frac{D}{\mathcal{E} T} \right)^{1/2}$$

Finally, we can calculate the PDH contribution to the activity coefficient as follows:

$$\begin{aligned} \ln(\gamma_i^{PDH}) &= \frac{\delta}{\delta n_i} \left( \frac{G^{PDH}}{RT} \right)_{T,P} \\ &= -4 \left( \frac{1000}{M} \right)^{1/2} \frac{A_x I_x}{\rho} \left[ \ln(1 + \rho I_x^{1/2}) \left( \frac{M - m_i}{2M} + \frac{z_i^2}{2I} \right) \right. \\ &\quad \left. + \frac{1}{2} \left( 1 - \frac{D}{d_i} \right) + \frac{3N}{2} \frac{m_i}{d_i} \frac{m_j}{\sum_j n_j \frac{m_j}{d_j}} \left( 1 - \frac{\epsilon_i}{\mathcal{E}} \right) - \frac{N}{\rho} \frac{\delta \rho}{\delta n_i} \right] \\ &\quad \left. + \frac{1}{1 + \rho I_x^{1/2}} \left( \frac{\rho I_x^{1/2}}{2N} \left( \frac{z_i^2}{2I} - 1 \right) + I_x^{1/2} \frac{\delta \rho}{\delta n_i} \right) \right] \end{aligned}$$

Note that in the above, the analytical form of the term  $\frac{\delta \rho}{\delta n_i}$  is not used to simplify the expression.

## 6.3 Tutorial on using the PDH correction

There is a tutorial on using the PDH correction that demonstrates the use of this correction with the GUI.

### References



## THE UNIFAC PROGRAM

Using the UNIFAC program from the command line is documented in the following:

### 7.1 Compound Input

#### 7.1.1 Basic Input

In the UNIFAC program, compounds are expected to be input as SMILES strings, and their ratios are expected as mole fractions. A summary of basic compound input is given below:

Flag	Purpose	Example
-h	Produces help message	\$AMSBIN/unifac -help
-smiles	Input molecule as SMILES sting	\$AMSBIN/unifac -smiles <SMILES1> <SMILES2> ...
-frac	Input the mole fractions	\$AMSBIN/unifac -frac <mol frac- tion 1> <mol fraction 2> ...
-solute	Specify a molecule as a solute	\$AMSBIN/unifac -smiles CCC - solute -smiles ...
-o	Write output to file	\$AMSBIN/unifac -o <output file> ...

Note that the -smiles and -frac flags are only specified one time and all information ( SMILES strings and mole fractions ) comes after these flags. It is assumed that the order of the mole fractions after the -frac corresponds to the order of the SMILES strings after the -smiles flag. A simple example demonstrating an activity coefficient calculation for a mixture of three compounds is given below:

```
$AMSBIN/unifac -smiles CCCCC CCCO CCCBr -solute -frac 0.2 0.3 0.5 -t ACTIVITYCOEF
```

The -solute flag is used to specify which compounds should be treated as solutes for the PURESOLUBILITY template. See the PURESOLUBILITY section for more information.

## 7.1.2 Physical Property Input

A number of problem templates require physical property information to be input. Physical property information should be input directly after a compound's SMILES representation. A list of the physical property flags and examples of usage are given below:

Flag	Purpose	Example
-pvap	Vapor pressure (bar)	\$AMSBIN/unifac -smiles <SMILES> -pvap 0.43 ...
-tvap	Temperature (K) corresponding to pvap	\$AMSBIN/unifac -smiles <SMILES> -tvap 320.1 ...
-antoine	Antoine coefficients for compound	\$AMSBIN/unifac -smiles <SMILES> -antoine 7.23 1504.2 246.87 ...
-hfusion	Enthalpy of fusion in kJ/mol	\$AMSBIN/unifac -smiles <SMILES> -hfusion 6.4
-meltingpoint	Melting point of compound (K)	\$AMSBIN/unifac -smiles <SMILES> -meltingpoint 421.12

Below is an example (with synthetic antoine parameters) demonstrating the command line input for a binary mixture calculation (BINMIXCOEF) using vapor pressure estimated from the antoine parameters.

```
$AMSBIN/unifac -smiles "CCCCOCC" -antoine 5 1500 30 \
"CCCCOCC" -antoine 6 1234 10 -t BINMIXCOEF
```

Additionally, we present an example for calculating the solubility of DDT in ethanol. Since DDT is a solid at room temperature, this requires us to input Enthalpy of Fusion and Melting Point data.

```
$AMSBIN/unifac -smiles \
"C1=CC(=CC=C1C(C2=CC=C(C=C2)C1)C(C1)(C1)C1)C1" -hfusion 26.28 -meltingpoint 383 \
"CCO" -frac 0.0 1.0 -t SOLUBILITY
```

## 7.2 Program Input

### 7.2.1 List of possible input flags

The UNIFAC program accepts a few additional flags to specify system conditions, choose a template for the calculation, or set the number of points taken over a provided range (e.g., a temperature range). A summary of these input options is provided below. In the following section, examples are given for each flag.

Flag	Meaning
-t	One of a number of template strings to indicate the problem type
-temperature	One/two values (K) to specify the system temperature/temperature range
-steps	the number of steps taken if a range is specified or for a BINMIXCOEF or TERNARYMIX calculation
-preset	an integer to indicate which solvent system is chosen for logP calculations

## 7.2.2 Examples of general program flags

In this section, we provide a brief example of each of the above flags.

### -t

The -t flag has been shown in previous examples and will be described in more detail in the following section. The -t flag must be followed by one of the following template names:

```
LOGP
ACTIVITYCOEF
PURESOLUBILITY
SOLUBILITY
VAPOREPRESSURE
PUREVAPOREPRESSURE
BINMIXCOEF
TERNARYMIX
```

A brief example of this for the ACTIVITYCOEF template is given below:

```
$AMSBIN/unifac -smiles "CCCCO" "CCCOCC" -frac 0.5 0.5 -t SOLUBILITY
```

### -temperature

In this example, we take the previous DDT solubility calculation and perform the calculation at a temperature of 310 K. This is shown below:

```
$AMSBIN/unifac -smiles \
"C1=CC(=CC=C1C(C2=CC=C(C=C2)C1)C(C1)(C1)C1)C1" -hfusion 26.28 -meltingpoint 383 \
"CCO" -frac 0.0 1.0 -temperature 310 -t SOLUBILITY
```

If we want to calculate the solubility over a temperature range (say, 310-350 K), we need to specify the temperature flag twice and also include a number of steps to take between the two temperatures with the -steps flag. This looks like the following:

```
$AMSBIN/unifac -smiles \
"C1=CC(=CC=C1C(C2=CC=C(C=C2)C1)C(C1)(C1)C1)C1" -hfusion 26.28 -meltingpoint 383 \
"CCO" -frac 0.0 1.0 -temperature 310 -temperature 350 -steps 10 -t SOLUBILITY
```

### -steps

The -steps flag specifies a number of steps to take between a temperature range or the number of steps to take along each mole fraction axis for a BINMIXCOEF or a TERNARYMIX calculation. For an input value of N for the -steps flag, the BINMIXCOEF and TERNARYMIX templates consider the following number of distinct mole fraction combinations:

Template	Number of distinct systems considered
BINMIXCOEF	N+5
TERNARYMIX	(N+1)(N+2)/2

If we wanted to calculate the thermodynamic properties of a binary mixture with a very small step size, we could input a N value of 1000 to take 1005 samples of the mole fraction space:

```
$AMSBIN/unifac -smiles \  
"CCCCOCC" -antoine 5 1500 30 \  
"CCCCCC" -antoine 6 1234 10 \  
-t BINMIXCOEF -steps 1000
```

### -preset

The preset flag is used for a logP calculation. A preset of 2 (default) indicates that that we do a logP calculation on the traditional Octanol/Water system. This looks like the following:

```
$AMSBIN/unifac -smiles "CCCCOCC" -t LOGP -preset 2
```

More information on the preset flag options for the LOGP template will be given in the templates section.

## 7.3 Templates

The -t flag indicates which of several problem types, or templates, should be used. Different templates expect different input options and produce different results. A summary of the different templates is given below.

### 7.3.1 ACTIVITYCOEF

Number of compounds required	>= 2
Mole fraction values (-frac) required	Yes
-solute flag used	No
-antoine or pvap/tvap required	No
-hfusion/-meltingpoint required	No

The activity coefficient template calculates activity coefficients from a set of mole fraction values. In the following example, we calculate the activity coefficients of the water/propanol system over a temperature range of 230-298.15 K.

```
$AMSBIN/unifac -smiles "O" "CCCO" -frac 0.2 0.8 \  
-t ACTIVITYCOEF -temperature 230 -temperature 298.15 -steps 20
```

### 7.3.2 LOGP

Number of compounds required	>= 1
Mole fraction values (-frac) required	No
-solute flag used	No
-antoine or pvap/tvap required	No
-hfusion/-meltingpoint required	No

The logP template calculate the partition coefficient (P) of input structures between a variety of common liquid/liquid systems. The specific set of liquids can be chosen with the -preset flag. A summary of the -preset options is given below:

-preset value	Liquid phases
2	Octanol/Water
3	Benzene/Water
4	Diethyl ether/Water
5	Hexane/Water

In the following example, we calculate the logP of Sertraline in the Octanol/Water system.

```
$AMSBIN/unifac -smiles "CNC1CCC(C2=CC=CC=C12)C3=CC(=C(C=C3)C1)C1" -t LOGP -preset 2
```

### 7.3.3 PURESOLUBILITY

Number of compounds required	>= 2
Mole fraction values (-frac) required	No
-solute flag used	Yes
-antoine or pvap/tvap required	No
-hfusion/-meltingpoint required	if solute is a solid

The pure solubility template calculates the solubility of a solute (designated by the -solute flag) in a variety of pure solutes. More specifically, the solute's solubility is calculated in every one of the other input molecules.

In the following example we calculate the solubility of Undecanedioic acid (a solid at 298.15 K) in n-Hexane, Benzene, Water, and Ethanol.

```
$AMSBIN/unifac -smiles \
"C(CCCCC(=O)O)CCCCC(=O)O" -hfusion 39.65 -meltingpoint 385 -solute \
"CCCCC" "c1ccccc1" "O" "CCO" -t PURESOLUBILITY
```

### 7.3.4 SOLUBILITY

Number of compounds required	>= 2
Mole fraction values (-frac) required	Yes
-solute flag used	No
-antoine or pvap/tvap required	No
-hfusion/-meltingpoint required	if solute is a solid

The solubility template calculates the solubility of every input molecule in the system defined by the remaining molecules. For example, assume we input a system with molecules A, B, and C with mole fractions 0.2, 0.6, and 0.2. The solubility of molecule A is then calculated in a mixture of B/C where the mole fraction ratio is fixed to 3/1 (from the 0.6/0.2 in the input). The solubility of A may very well be 0.6, but this would mean the remaining mole fractions would be 0.3/0.1 (B/C). The same calculation is then also done for B and C.

In the following example we calculate the solubilities of Benzene, n-Hexane, 1-Hexanol, and Acetic acid. Note that the mole fraction of Benzene is set to 0.0. This means that Benzene's solubility will still be calculated, but it will not be part of the solvent system when the other molecules' solubilities are being calculated.

```
$AMSBIN/unifac \
-smiles "c1ccccc1" "CCCCC" "CCCCCO" "CC(=O)O" \
-frac 0.0 0.2 0.3 0.5 -t SOLUBILITY
```

### 7.3.5 PUREVAPORPRESSURE

Number of compounds required	>= 1
Mole fraction values (-frac) required	No
-solute flag used	No
-antoine or pvap/tvap required	Yes
-hfusion/-meltingpoint required	No

The pure vapor pressure template simply calculates the vapor pressure of a pure component. Because this requires the antoine parameters as input, this template simply evaluates the antoine equation, possibly over a temperature range.

The following example calculates the pure vapor pressure (again using synthetic antoine coefficients) for two molecules over a temperature range:

```
$AMSBIN/unifac \
-smiles "c1ccccc1" -antoine 4 1245 123 \
"CCCCC" -antoine 5 1241 242 \
-t PUREVAPORPRESSURE -temperature 320 -temperature 350 -steps 10
```

### 7.3.6 VAPORPRESSURE

Number of compounds required	>= 1
Mole fraction values (-frac) required	Yes
-solute flag used	No
-antoine or pvap/tvap required	Yes
-hfusion/-meltingpoint required	No

The vapor pressure template calculates the vapor pressure of a mixture (or a pure component if only one compound is entered). This again requires the antoine parameters for each compound as input.

We repeat the previous example, now calculating the vapor pressure of the 0.2/0.8 mole fraction mixture.

```
$AMSBIN/unifac \
-smiles "c1ccccc1" -antoine 4 1245 123 \
"CCCCC" -antoine 5 1241 242 \
-frac 0.2 0.8 \
-t VAPORPRESSURE -temperature 320 -temperature 350 -steps 10
```

### 7.3.7 BINMIXCOEF

Number of compounds required	2
Mole fraction values (-frac) required	No
-solute flag used	No
-antoine or pvap/tvap required	Yes
-hfusion/-meltingpoint required	No

The binary mixture template takes exactly 2 compounds as input. Unlike other templates where thermodynamic properties are calculated over a range of temperatures, the binary mixture template calculates properties over a range of mole fractions. In other words, it takes a number of samples of the mole fraction space. If no antoine coefficients are given, then no gas phase thermodynamic properties are reported.

In this example we calculate binary mixture properties for the Water/Ethanol system (again with synthetic antoine parameters).

```
$AMSBIN/unifac -smiles "O" -antoine 4 1245 123 "CCO" \
-antoine 5 1241 242 -t BINMIXCOEF -steps 10
```

### 7.3.8 TERNARYMIX

Number of compounds required	3
Mole fraction values (-frac) required	No
-solute flag used	No
-antoine or pvap/tvap required	Yes
-hfusion/-meltingpoint required	No

The ternary mixture template takes exactly 3 molecules as input and performs similar calculations to those done in the binary mixture template. Note that tie lines are not calculated like they are in the COSMO-RS/-SAC programs.

In this example we add a Acetone to our previous two compounds and change the temperature to 330 K.

```
$AMSBIN/unifac -smiles "O" -antoine 4 1245 123
"CCO" -antoine 5 1241 242 \
"CC(=O)C" -antoine 6 2414 221 \
-t TERNARYMIX -steps 20 -temperature 330
```



## PURE COMPOUND PROPERTY PREDICTION

## 8.1 Introduction

The Property Prediction program in ADF provides quick, accurate estimates for many important pure component physical properties. At its core, the Property Prediction program maps various QSPR descriptors of an input molecule onto a single numerical value, the property estimate. Many of these property models rely on easy-to-evaluate QSPR descriptors and numerically straightforward computations, meaning that an estimate can be provided for every property in  $\ll$  1s per molecule. The general expression for the models used in the Property Prediction program is as follows:

$$f(P) = C + g\left(\sum_i c_i n_i\right) + h\left(\sum_i d_i n_i, T\right)$$

where  $f$  is a function that transforms the property value space,  $g$  is a function that maps QSPR descriptors onto a numerical value, and  $h$  is a function which also captures temperature-dependence of certain properties by including temperature,  $T$ , as an input. Additionally,  $C$  is a constant,  $n_i$  refers to QSPR values of QSPR descriptor  $i$ , and  $c_i$  and  $d_i$  are fitted coefficients corresponding to each QSPR descriptor  $i$ .

The accuracy of the property estimates depends on the nature/complexity of the input molecular structure. For many common organic structures, the property estimates should be reasonably accurate. However, as is always the case with QSPR models, the Property Prediction program will likely lose accuracy for molecules outside its training domain, i.e., for molecules that are very “dissimilar” to compounds which occur in the training set. In general, the program can be used for molecules with the following atom types:

Accepted atom types	Example functional groups which can be used with atom type
H	Alkanes, Alkenes, Alkynes, Aldehydes, Amides, Amines, Aromatics, Carboxylic Acids, Hydroxides, Sulfides, Thiols
C	Acid chlorides, Alkanes, Alkenes, Alkynes, Aldehydes, Amides, Aromatics, Carboxylic Acids, Esters, Ethers, Ketones, C-X (halogens)
N	Amides, Amines, Aromatics, Cyanides, Imines, Nitro groups
O	Acid chlorides, Aldehydes, Amides, Aromatics, Carboxylic Acids, Esters, Ethers, Ketones, Nitro groups
F	-CF, -CF <sub>2</sub> , -CF <sub>3</sub>
S	Sulfides, Thiols
Cl	Acid chlorides, -CCl, -CCl <sub>2</sub> , -CCl <sub>3</sub>
Br	-CBr
I	-CI

A brief description of molecule types for which this method will not work well is given in the [General warnings section](#). Common molecules for which this method will fail are: (1) those that contain only one non-hydrogen atom, e.g., Methane or Water; (2) those that contain atoms not listed in the table above.

## 8.2 Available properties

The Property Prediction program can predict the values of various pure component physical properties. These properties can be of interest themselves or can be used in conjunction with other COSMO-RS property calculations (e.g., to calculate the solubility of a solid in a liquid, we must know the enthalpy of fusion and melting point of the solid – both of these properties can be estimated with the Property Prediction program). The available properties and their units are listed below:

Property Name	Units	Additional Information	Typical error
Boiling point	K	at 1 atm	15 K
Critical Pressure	bar		1.5 bar
Critical Temperature	K		30 K
Critical Volume	L/mol		0.02 L/mol
Dielectric Constant			3
Ideal Gas Entropy	J/(mol K)	at 298.15 K and 1 bar	20 J/(mol K)
Flash point	K		15 K
Gibbs Energy Ideal Gas	kJ/mol	at 298.15 K and 1 bar	25 kJ/mol
Enthalpy of Combustion	kJ/mol	at 298.15 K	50 kJ/mol
Std. Enthalpy of Formation	kJ/mol	at 298.15 K and 1 bar	30 kJ/mol
Enthalpy of Fusion	kJ/mol	at Normal Melting Point	4 kJ/mol
Enthalpy of Form. Ideal Gas	kJ/mol	at 298.15 K and 1 bar	25 kJ/mol
Enthalpy of Sublimation	kJ/mol		5 kJ/mol
Melting point	K	at 1 atm	35 K
Liquid Molar volume (Liquid Density)	L/mol kg/L	at 298.15 K	0.005 L/mol uses Liquid Molar Volume
Liquid Vapor Pressure	bar		10-30%
Liquid Viscosity ( $\mu$ )	Pa-s		20-50%
Parachor			7
Solubility Parameter	(cal/cm <sup>3</sup> ) <sup>1/2</sup>	at 298.15 K	0.7
Synthetic accessibility <sup>1</sup>	scored from 1 (easy) to 10 (hard)		
Triple point temperature	K		35 K
Van der Waals Area	Å <sup>2</sup>		6 Å <sup>2</sup>
Van der Waals Volume	Å <sup>3</sup>		3 Å <sup>3</sup>

<sup>1</sup>

P. Ertl and A. Schuffenhauer. *Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions* Journal of cheminformatics 1 (2009): 1-11. (<https://doi.org/10.1186/1758-2946-1-8>)

## 8.3 Running the Property Prediction program

The Property Prediction program can be run from the command line. The following general flags are used by the program:

Flag	Purpose	Example
-h [--help]	Produces help message	\$AMSBIN/prop_prediction --help
-s [--smiles]	Input molecule as SMILES sting	\$AMSBIN/prop_prediction --smiles <SMILES> ...
-m [--mol]	Input molecule as .mol file	\$AMSBIN/prop_prediction --mol <mol file> ...
--sdf	Input molecule as an .sdf file	\$AMSBIN/prop_prediction --sdf <file.sdf> ...
--temperature	Set temperature/range (K)	\$AMSBIN/prop_prediction --temperature 298.15 ...
-n	number of steps for temperature range	\$AMSBIN/prop_prediction --n 20 ...
-d [--display]	Display calculated properties	\$AMSBIN/prop_prediction --d ...
-o [--output]	Write output to file	\$AMSBIN/prop_prediction --o <output file> ...

Note, if no output flag is supplied, then the results are written to a file called CRSKF by default. Additionally, the user may enter as many compounds as desired on the command line in either of the two available input formats.

The program can be run in 2 ways:

- Estimating all available properties for every molecule
- Estimating specific properties for every molecule

To estimate all properties for every input compound, simply execute the program with all molecules specified on the command line. Don't forget that the -d flag is required to display the results in the terminal. An example of this is below.

```
$AMSBIN/prop_prediction --smiles CCCCCO -o example.crskf -temperature 298.15 -  
↪temperature 398.15 -n 20 -d
```

```
Boiling point at standard pressure :  
CCCCCO          435.777 K  
Critical pressure :  
CCCCCO          34.3493 bar  
Critical temperature :  
CCCCCO          576.466 K  
Critical volume :  
CCCCCO          0.404124 L/mol  
Liquid density :  
CCCCCO          0.79182 kg/L  
Dielectric constant :  
CCCCCO          10.9512  
Absolute entropy of an ideal gas at 298.15 K and 1 bar :  
CCCCCO          439.885 J/(mol K)  
Flash point :  
CCCCCO          342.271 K  
Gibbs energy of formation for an ideal gas at 298.15 K and 1 bar :
```

(continues on next page)

(continued from previous page)

```

CCCCCO          -131.869  kJ/mol
Net enthalpy of combustion at 298.15 K  :
CCCCCO          -3678.12  kJ/mol
Standard state enthalpy of formation at 298.15 K and 1 bar :
CCCCCO          -384.388  kJ/mol
Enthalpy of fusion at normal melting point :
CCCCCO          18.5054  kJ/mol
Enthalpy of formation for an ideal gas 298.15 K :
CCCCCO          -316.821  kJ/mol
Enthalpy of sublimation :
CCCCCO          80.9799  kJ/mol
Melting point at 1 atm :
CCCCCO          231.141  K
Liquid molar volume :
CCCCCO          0.128949  L/mol
Parachor :
CCCCCO          289.059
Solubility parameter :
CCCCCO          10.1294  (cal/cm^3)^0.5
Triple point temperature :
CCCCCO          230.404  K
Van der Waals area :
CCCCCO          171.059  Å^2
Van der Waals volume :
CCCCCO          120.519  Å^3

```

Liquid viscosity :

Molecule: CCCCCO

Temperature (K)	Liquid viscosity (Pa-s)
298.15	0.004465
303.15	0.003865
308.15	0.003364
313.15	0.002942
318.15	0.002584
323.15	0.002281
328.15	0.002021
333.15	0.001798
338.15	0.001606
343.15	0.001439
348.15	0.001294
353.15	0.001168
358.15	0.001057
363.15	0.000959
368.15	0.000873
373.15	0.000797
378.15	0.000729
383.15	0.000669
388.15	0.000615
393.15	0.000567
398.15	0.000523

Liquid vapor pressure :

Molecule: CCCCCO

Temperature (K)	Vapor pressure (bar)
298.15	0.001229
303.15	0.001809
308.15	0.002623
313.15	0.003750
318.15	0.005289

(continues on next page)

(continued from previous page)

323.15	0.007362
328.15	0.010123
333.15	0.013757
338.15	0.018487
343.15	0.024582
348.15	0.032357
353.15	0.042182
358.15	0.054484
363.15	0.069757
368.15	0.088563
373.15	0.111537
378.15	0.139394
383.15	0.172929
388.15	0.213022
393.15	0.260644
398.15	0.316852

For most applications, it is not necessary to calculate all of the available physical properties (although doing so is practically just as fast). In these cases, additional property flags need to be specified on the command line to restrict the program to calculating only certain physical properties. For example, if we were doing solid/liquid solubility calculations on Ibuprofen and Paracetamol, we would require the Enthalpy of Fusion and the Melting Point of both compounds. To calculate only these two properties, we simply have to add the two property flags “-hfusion” and “-meltingpoint” to the command line. Using the .mol file for Ibuprofen and the SMILES string for Paracetamol, we execute the following:

```
$AMSBIN/prop_prediction -d -m Ibuprofen.mol -s 'CC(=O)NC1=CC=C(C=C1)O' -hfusion -
↪meltingpoint
```

```
Enthalpy of fusion at normal melting point :
CC(=O)NC1=CC=C(C=C1)O      33.0298  kJ/mol
Ibuprofen.mol              24.0336  kJ/mol

Melting point at 1 atm :
CC(=O)NC1=CC=C(C=C1)O      469.282  K
Ibuprofen.mol              331.887  K
```

## 8.4 Index of property keys

The available properties and their corresponding property flags are listed below:

Property Name	Property Flag
Boiling point	--boilingpoint
Critical Pressure	--criticalpressure
Critical Temperature	--criticaltemp
Critical Volume	--criticalvol
Dielectric Constant	--dielectricconstant
Ideal Gas Entropy	--entropygas
Flash point	--flashpoint
Gibbs Energy Ideal Gas	--gidealgas
Enthalpy of Combustion	--hcombust
Std. Enthalpy of Formation	--hformstd
Enthalpy of Fusion	--hfusion
Enthalpy of Form. Ideal Gas	--hidealgas
Enthalpy of Sublimation	--hsublimation
Melting point	--meltingpoint
Liquid Molar volume	--molarvol
(Liquid Density)	--molarvol
Liquid Vapor Pressure	--vaporpressure
Liquid Viscosity	--liquidviscosity
Parachor	--parachor
Solubility Parameter	--solubilityparam
Synthetic accessibility	--synacc
Triple point temperature	--tpt
Van der Waals Area	--vdwarea
Van der Waals Volume	--vdwvol

## 8.5 General warnings

This method will **fail** for the following types of molecules:

- Those that contain *only one non-hydrogen atom* (e.g., Methane or Water). However, experimental data is ample for these small molecules. The vapor pressure model is the exception in that it *can* represent such small structures.
- Those that contain atoms or substructures that are not listed in the *Accepted atom types table* (page 79) above.
- Polymers and Ionic Liquids

This method will **lose accuracy** for some properties in the following domains:

- Molecules with *many different types of functional groups*
- Molecules that are *extremely light* (< 3 non-Hydrogen atoms) or *heavy* (> 30 non-Hydrogen atoms)

## 8.6 Equations for temperature-dependent properties

### 8.6.1 VPM1: liquid vapor pressure

$$\ln(P) = A/T + B \ln(T) + CT + D$$

Symbol	Meaning
P	vapor pressure
T	absolute temperature
A,B,C,D	estimated constants

### 8.6.2 empirical-VIS1: Liquid Viscosity

$$\ln(\mu) = A + B/T + C \ln(T) + (D/T) \exp(E/T)$$

Symbol	Meaning
$\mu$	liquid viscosity
T	absolute temperature
A,B,C,D,E	estimated constants

## 8.7 References



## SOLVENT OPTIMIZATION

### 9.1 Introduction

The choice of solvent or solvent system can have a dramatic impact on the solubility of solutes, the performance of the solvent system for liquid-liquid extraction, or reaction rates/selectivities for many chemical reactions. The solvent is also one of the most accessible variables in formulation/reaction/extraction design as many of the other species in these problems (active ingredients, co-solvents, reactants, catalysts, extracts, raffinates, etc. ) are often fixed or tightly constrained. However, the solvent selection problem is combinatorially complex and non-trivial. As an example of this complexity, consider the problem of choosing up to 4 solvents for a process from a set of 100 possible solvents. The number of combinations alone is over 4 million, a number that makes laboratory- or pilot-scale testing for every system untenable. An additional complication is that the identities of the solvents alone do not determine a solvent system: we must also know the mole fractions. Of course, there are an infinite number of possible mole fraction values for a solvent system ( a value of  $x_1 = 0.3$ ,  $x_2 = 0.7$  is just as valid as  $x_1 = 0.29999$ ,  $x_2 = 0.70001$  ), meaning a high-throughput approach would still require multiple mole fraction values for each solvent system to effectively sample the mole fraction space.

Luckily, the complexity of this problem can be addressed with modern mathematical optimization approaches. Following the approach of<sup>1</sup>, we re-structure the COSMO-RS/-SAC parameters and equations and incorporate them into a Mixed Integer Nonlinear Programming (MINLP) formulation. Using this generic formulation, we can apply a number of optimization solvers and solution techniques to the problem of determining an optimal solvent system. We note that the optimization methods currently in use only guarantee local solutions, but the formulation should be robust enough to provide high-quality solutions for many types of problems. In fact, for most of our example problems, our optimization approach was able to find the globally optimal solution (as determined by an exhaustive enumeration of the solvent space and dense sampling of the mole fractions space). There are additionally some features in the program (e.g., the -multistart flag) which help to provide a diversity of starting conditions so the solvers can find high-quality solutions.

### 9.2 Problem types

At present, the program has two problem templates:

- The **SOLUBILITY** template: this selects a solvent system and mole fractions in order to maximize or minimize the mole fraction solubility of a solid solute in the liquid mixture. Note that currently only solid/liquid equilibria calculations are supported.
- The **LLEXTRACTION** template: this selects a two-phase solvent system and mole fractions in order to maximize (or minimize) the distribution ratio (D) of two solutes between the two liquid phases. The distribution ratio for these problems is defined in terms of mole fractions rather than concentrations. The formula for this is given in an equation below. Note that LLEXTRACTION problems will fail if all possible solvents are miscible.

---

<sup>1</sup> N.D. Austin, N.V. Sahinidis, D.W. Trahan, *COSMO-based computer-aided molecular/mixture design: A focus on reaction solvents*, *AIChE Journal* 64, 104 (2018) (<https://doi.org/10.1002/aic.15871>)

Because liquid densities are not always known, we calculate the distribution ratio (D) in terms of mole fractions. The liquid-liquid phase equilibrium condition provides an equivalent expression for D. More precisely, this means that D can be calculated as follows:

$$D = \max \left( \frac{\gamma_1^I \gamma_2^{II}}{\gamma_1^{II} \gamma_2^I}, \frac{\gamma_2^I \gamma_1^{II}}{\gamma_2^{II} \gamma_1^I} \right)$$

where  $\gamma_i^j$  represents the activity coefficient of solute  $i$  in phase  $j$ . Here, we assume that the two solutes to be separated are indexed with  $i \in \{1, 2\}$ . The max operator, though not used in the optimization problem itself, allows us to express the correct value of D in the equation above. In other words, the max operator removes the dependence of the D value on arbitrary indexing of solutes and phases.

**Note:** For LLEXTRACTION problems, the mole fractions of the solutes are fixed to 0 unless they are also specified as solvents. This means that the distribution coefficient is calculated using the infinite dilution activity coefficients.

A brief summary of what to expect for solvent optimization problems using the two templates is given below:

	SOLUBILITY	LLEXTRACTION
Minimum number of solvents	1	2
Preferred number of solvents	>1	>4
Typical solution times	<2s	1-30s
Recommended multistarts	<5 if any	5-10
Warmstart recommended	No	Problem-dependent

The warmstart and multistart options will be explained in a later section.

### 9.3 Running the Solvent Optimization program

The Solvent Optimization program can be run from the command line. The following general flags are used by the program:

Flag	Purpose	Example
-h [-help]	Produces help message	\$AMSBIN/solvent_opt -help
-s [-smiles]	Input molecule as SMILES sting	\$AMSBIN/solvent_opt -smiles <SMILES> ...
-m [-mol]	Input molecule as .mol file	\$AMSBIN/solvent_opt -mol <mol file> ...
-c [-coskf]	Input molecule as .coskf file	\$AMSBIN/solvent_opt -coskf <.coskf file> ...
-d [-display]	Display problem results	\$AMSBIN/solvent_opt -d ...
-o [-output]	Write output to file	\$AMSBIN/solvent_opt -o <output file> ...

Additionally, physical properties required for the calculation can be input on the command line. Presently, only two physical properties (Enthalpy of Fusion and Melting Point) are required for certain calculations. These flags must follow a molecule input and be followed by the property value. Some examples are given below:

Flag	Property	Example
-hfusion	H of Fusion (kJ/mol)	\$AMSBIN/solvent_opt -c Ibuprofen.coskf -hfusion 26.6 ...
-meltingpoint	Melting point (K)	\$AMSBIN/solvent_opt -c Ibuprofen.coskf -meltingpoint 349.2 ...

Optimization problem specifications and method options can be input with the following flags:

Flag	Meaning	Example
-method	Choice of method (COSMO-RS COSMOSAC2016)	... -method COSMO-RS ...
-max	maximize the Solubility/Extraction ratio	
-min	minimize the Solubility/Extraction ratio	
-solute	specify which molecules are solutes	... -s CCCO -solute ...
-t [-template]	choose a problem template	... -t LLEXTRACTION ...
-temperature	input 1 or 2 temperatures (K)	... -temperature 298.15 ...

Note that, like the -hfusion and -meltingpoint flags, the -solute flag comes *after* a compound identifier (SMILES string/filename).

Finally, there are two more optimization problem flags which can be altered for problems that do not converge. The first is the -multistart flag. This flag takes an integer  $N$  as input and instructs the algorithm to begin from  $N$  randomly-generated starting points. This can be useful for difficult problems because not only will the algorithm begin from more starting points, but it will also adjust internal parameters every time a problem fails. The -warmstart flag instructs the main algorithm to attempt to make the convert the initial starting point to a high-quality, feasible starting point which can then be given to the optimization algorithm. This option can be helpful for many problems, especially those with small numbers of solvents or LLEXTRACTION problems where the solvents are extremely immiscible (e.g., Water and n-Hexane). A summary of these options is presented below:

Flag	Meaning	Example
-multistart	Start from a number of random starting points	... -multistart 5 ...
-warmstart	Use the warmstart strategy	\$AMSBIN/solvent_opt -warmstart ...

## 9.4 Examples

In this section, we provide a few example problems to demonstrate a few of the features available in the Solvent Optimization program. We first do a sample problem with the SOLUBILITY template, and then we provide an example of the usage of the LLEXTRACTION template.

### 9.4.1 Solubility

For a first example, we determine a mixture of solvents to maximize the solubility of Paracetamol. For the purposes of illustrating features, we assume that we do not have an available .coskf file for Paracetamol and must use its SMILES string. We can use a few common solvents from the ADFCRS-2018 database:

```
$AMSBIN/solvent_opt -t SOLUBILITY -d -max \
-s "CC(=O)NC1=CC=C(C=C1)O" -solute -meltingpoint 443.1 \
-c $AMSHOME/atomicdata/ADFCRS-2018/Acetic_acid.coskf \
-c $AMSHOME/atomicdata/ADFCRS-2018/Hexane.coskf \
-c $AMSHOME/atomicdata/ADFCRS-2018/Toluene.coskf \
-c $AMSHOME/atomicdata/ADFCRS-2018/Butanoic_acid.coskf \
-c $AMSHOME/atomicdata/ADFCRS-2018/Ethanol.coskf
```

```
=====
                Estimating missing property values
=====
                                Estimated values
-----
Molecule                                Missing Property    Estimated Value
-----
CC(=O)NC1=CC=C(C=C1)O                   Hfusion              7.89433
-----

=====
                Beginning solvent optimization problem
=====

*****
                Iteration 1
*****

Initial guess x1:    0.0235183
Initial guess x2:    0.0758477
Initial guess x3:    0.220089
Initial guess x4:    0.283974
Initial guess x5:    0.314382
Initial guess x6:    0.082189
-----> Solver Status: CONVERGED
Objective value: 0.159729

-----
                                Variable values
-----
                x1:    0.159729    CC(=O)NC1=CC=C(C=C1)O
                x2:    0           Acetic_acid.coskf
                x3:    0           Hexane.coskf
                x4:    0           Toluene.coskf
```

(continues on next page)

(continued from previous page)

```
x5: 0          Butanoic_acid.coskf
x6: 0.840271  Ethanol.coskf
```

The problem correctly selects Ethanol as the solvent in which Paracetamol is most soluble. Single solvent solutions are common in SOLUBILITY problems as often no mixed solvent system outperforms single solvents. Notice that in this example any required property values for solid/liquid equilibria that are missing are estimated based on the input SMILES string. Because the -meltingpoint flag provided a value for the Melting Point, only the Enthalpy of Fusion is estimated.

## 9.4.2 Liquid-liquid extraction

Our next example focuses on a classic liquid-liquid extraction problem: separating Acetic acid and Water. In this example, we assume that we want to replace a standard solvent for this extraction (n-Hexane) with something more environmentally friendly. Consulting GSK's Solvent Selection Guide<sup>2</sup>, we restrict our problem to the solvents with the fewest issues: Water, 1-Butanol, 2-Butanol, t-Butyl acetate, Isopropyl acetate, Propyl acetate, and Dimethyl carbonate.

```
$AMSBIN/solvent_opt -d -t LLEXTRACTION -max \
-c $AMSHOME/atomicdata/ADFCRS-2018/Acetic_acid.coskf -solute \
-c $AMSHOME/atomicdata/ADFCRS-2018/Water.coskf -solute \
-c $AMSHOME/atomicdata/ADFCRS-2018/Water.coskf \
-c $AMSHOME/atomicdata/ADFCRS-2018/1-Butanol.coskf \
-c $AMSHOME/atomicdata/ADFCRS-2018/2-Butanol.coskf \
-c $AMSHOME/atomicdata/ADFCRS-2018/tert-Butyl_acetate.coskf \
-c $AMSHOME/atomicdata/ADFCRS-2018/Isopropyl_acetate.coskf \
-c $AMSHOME/atomicdata/ADFCRS-2018/Propyl_acetate.coskf \
-c $AMSHOME/atomicdata/ADFCRS-2018/Dimethyl_carbonate.coskf
```

Notice that water is repeated because it is both a solute and a solvent in the solvent space. The output produced is the following:

```
Removing duplicate entry: /home/austin/amshome/atomicdata/ADFCRS-2018/Water.coskf
=====
Beginning solvent optimization problem
=====
*****
Iteration 1
*****
Phase I          Phase II
Initial guess x1: 0.0818796    0.241048
Initial guess x2: 0.162378    0.185238
Initial guess x3: 0.198892    0.0429676
Initial guess x4: 0.114387    0.152842
Initial guess x5: 0.0267187    0.0437898
Initial guess x6: 0.232349    0.042073
Initial guess x7: 0.112232    0.182333
Initial guess x8: 0.071164    0.109708
-----> Solver Status: CONVERGED
Objective value: 232.779
-----
Variable values
-----
x1: 0          0          Acetic_acid.coskf
x2: 0.994592  0.0358616  Water.coskf
```

(continues on next page)

<sup>2</sup> GSK Solvent Selection Guide. Accessed 1/9/18. <https://www.rsc.org/suppdata/gc/c0/c0gc00918k/c0gc00918k.pdf>

(continued from previous page)

x3:	0	0	1-Butanol.coskf
x4:	0	0	2-Butanol.coskf
x5:	0.000100098	0.186012	tert-Butyl_acetate.coskf
x6:	0	0	Isopropyl_acetate.coskf
x7:	0	0	Propyl_acetate.coskf
x8:	0.00530747	0.778127	Dimethyl_carbonate.coskf
-----			
Extraction values			
-----			
Distribution			
coefficient (D)		log10 (D)	
-----			
232.779		2.36694	
Solute 1 ID:	Acetic_acid.coskf		
Solute 2 ID:	Water.coskf		
-----			
	Partition ratio		
	Phase I	Phase II	
			Partition
			coefficient (P)
			log10 (P)
-----			
Solute 1:	1	8.39321	0.119144
Solute 2:	27.7342	1	-0.923928
			1.44302

In this problem, we obtain a mostly aqueous phase and a dimethyl carbonate/tert-butyl acetate phase as the solution. This solvent system provides a distribution coefficient (D) of 232.779. This is a good value for a separation, but it is still worse than the distribution coefficient of the water/hexane solvent system (D = 1372.14) by roughly a factor of 6.

We then increase our solvent search space to include the solvents deemed to have “some issues” by GSK and are also present in our database: Ethanol, 1-Propanol, 2-Propanol, Methanol, Ethyl acetate, Methyl acetate, Methyl isobutyl ketone, Acetone, p-xylene, Toluene, Isooctane, Cyclohexane, Heptane, and DMSO.

```

$AMSBIN/solvent_opt -d -t LLEXTRACTION -max \
-c $AMSHOME/atomicdata/ADFCRS-2018/Acetic_acid.coskf -solute \
-c $AMSHOME/atomicdata/ADFCRS-2018/Water.coskf -solute \
-c $AMSHOME/atomicdata/ADFCRS-2018/Water.coskf \
-c $AMSHOME/atomicdata/ADFCRS-2018/1-Butanol.coskf \
-c $AMSHOME/atomicdata/ADFCRS-2018/2-Butanol.coskf \
-c $AMSHOME/atomicdata/ADFCRS-2018/tert-Butyl_acetate.coskf \
-c $AMSHOME/atomicdata/ADFCRS-2018/Isopropyl_acetate.coskf \
-c $AMSHOME/atomicdata/ADFCRS-2018/Propyl_acetate.coskf \
-c $AMSHOME/atomicdata/ADFCRS-2018/Dimethyl_carbonate.coskf
-c $AMSHOME/atomicdata/ADFCRS-2018/Ethanol.coskf \
-c $AMSHOME/atomicdata/ADFCRS-2018/1-Propanol.coskf \
-c $AMSHOME/atomicdata/ADFCRS-2018/2-Propanol.coskf \
-c $AMSHOME/atomicdata/ADFCRS-2018/Methanol.coskf \
-c $AMSHOME/atomicdata/ADFCRS-2018/Ethyl_acetate.coskf \
-c $AMSHOME/atomicdata/ADFCRS-2018/Methyl_acetate.coskf \
-c $AMSHOME/atomicdata/ADFCRS-2018/Methyl_isobutyl_ketone.coskf \
-c $AMSHOME/atomicdata/ADFCRS-2018/Acetone.coskf \
-c $AMSHOME/atomicdata/ADFCRS-2018/p-Xylene.coskf \
-c $AMSHOME/atomicdata/ADFCRS-2018/Toluene.coskf \
-c $AMSHOME/atomicdata/ADFCRS-2018/2,2,4-Trimethylpentane.coskf \
-c $AMSHOME/atomicdata/ADFCRS-2018/Cyclohexane.coskf \
-c $AMSHOME/atomicdata/ADFCRS-2018/Heptane.coskf \
-c $AMSHOME/atomicdata/ADFCRS-2018/Dimethyl_sulfoxide.coskf

```

This produces the following:

```

...
-----
                                Variable values
-----
...
x2:  0.999972      7.50215e-05      Water.coskf
...
x15:  5.58242e-06  0.00252496      Methyl_isobutyl_ketone.coskf
...
x20:  2.27012e-05  0.9974           Cyclohexane.coskf
...
-----
                                Extraction values
-----
Distribution
coefficient (D)      log10 (D)
-----
1892.42              3.27702
Solute 1 ID:         Acetic_acid.coskf
Solute 2 ID:         Water.coskf
-----
                                Partition ratio
                                Phase I      Phase II      Partition
                                -----
                                coefficient (P)  log10 (P)
-----
Solute 1:            7.04343      1             7.04343      0.847784
Solute 2:            13329.1      1             13329.1      4.1248

```

As shown, this solvent system has a D value of 1892.42, better than that of the hexane/water system. Removing cyclohexane from the possible solvents, we still obtain a solution with a D value of 1891.09. Successively removing the best non-aqueous solvents from the solvent list, we obtain solutions with D values of: 1864.41, 1645.38, 1597.42, and finally 232.779 again. The number of good solutions for this problem lends credence to the idea of using such a solvent selection algorithm in general extraction design.

Additionally, there is a [tutorial on solvent optimization](#) which demonstrates running the program from the COSMO-RS GUI.

## 9.5 Guidelines for difficult problems

The Solvent Optimization program should produce high-quality solutions for many problems. However, there may be examples where the algorithm struggles to produce solutions at all. Below we list troubleshooting guidelines to help solve problematic solvent optimizations:

### (1) For LLEXTRACTION problems, ensure that there are at least 2 immiscible solvents

Because the LLEXTRACTION template requires that both the liquid-liquid phase equilibria condition is met and that there are two distinct liquid phases, the Solvent Optimization program will fail if all of the available solvents are miscible in all mole fractions (no phase separation is possible).

### (2) Re-execute the program several times

The Solvent Optimization program is not entirely deterministic. In particular, starting points are selected at random for every iteration. These starting points affect the convergence of the problem and in some cases can have a large impact on the optimization. This means that if one execution of the Solvent Optimization program fails to produce a solution,

then it is possible that a subsequent execution could succeed. If the program continues to fail after multiple attempts at re-execution, consider using the *multistart* or *warmstart* flags.

### (3) Use the **-multistart** flag

This flag executes the program multiple times from multiple starting points. If problems fail, the program uses information from these problems and updates internal optimization parameters to aid in the convergence of successive problems. Because each iteration takes a relatively short amount of time, the multistart flag can be used with high numbers of different starting points. It is useful to first try a smaller number of multistarts (5-15). If this produces no solution, try using the *-warmstart* flag in addition to the multistarts. If the problems continue to fail, use gradually higher numbers of multistarts (20,40,60,80,100+).

### (4) Use the **-warmstart** flag

If this flag is present, the program attempts to find a good starting point for LLEXTRACTION problems rather than simply using the randomly-generated starting point. This can be useful with or without the *-multistart* flag and is very problem-dependent. In our experience, LLEXTRACTION problems sometimes have difficulty converging if there are a small number of solvents and/or if the solution contains two highly-immiscible liquids (e.g., Hexane/Water). We reiterate that this is very problem-dependent.

## 9.6 Differences from standard implementations

The COSMO-RS method of the Solvent Optimization program is nearly identical to the ADF combi2005 implementation (the default COSMO-RS method). The single difference is that there is no *f\_corr* parameter in the Solvent Optimization implementation. This parameter is used for the perpendicular component of the sigma values and has only a small effect on the results. Removing it from the Solvent Optimization program was done to improve solution times and robustness. Though the calculated values will be similar, results from the Solvent Optimization program can easily be input to ADFCRS and checked against the full ADF combi2005 method if desired.

The COSMOSAC2016 implementation in Solvent Optimization is identical to the 2016-ADF Chen implementation in ADFCRS.

To reproduce the results from the Solvent Optimization program to within tolerance, parameters must be changed in the GUI or from the command line. The following parameter changes are required:

Method	Required parameter changes
COSMO-RS	set <i>f_corr</i> to 0
COSMOSAC2016	none

In the current version of the program, the COSMO-RS/-SAC parameters *cannot* be changed/customized.

## PYTHON SCRIPTING WITH COSMO-RS

### 10.1 pyCRS: a python wrapper for thermodynamic calculation

#### 10.1.1 General Information

**Attention:** Windows and Mac users may find it helpful to first read the *Getting Started* guides for scripting: [scripting with Windows](#) | [scripting with MacOS](#)

It is recommended to use the version of python that is shipped with AMS. This version ensures that all the necessary libraries (e.g., **PLAMS**) are properly imported and are mutually compatible. The best way to do this is to run the `am-python` program. That can be executed from the command line as follows:

```
$AMSBIN/amspython <your_program.py>
```

Furthermore, interactive usage of the `amspython` program can be beneficial. To do so, execute the following command in your terminal:

```
$AMSBIN/amspython # in terminal
```

```
run <your_program.py> # in interactive python
```

where `<your_program.py>` should of course be replaced by the name of your program.

#### 10.1.2 Overview

The `pyCRS` Python library offers a convenient wrapper for various thermodynamic calculations, providing the following utilities:

Modules	A	Description
<i>Database</i> (page 96)	A	Provides an interface to a SQL database for managing COSKF files with conformers.
<i>CRS-Manager</i> (page 98)	A	It's a python wrapper based on PLAMS library for facilitating the COSMO-RS calculation. It is designed to be used in conjunction with Database.
Prop-Pred	A	Predicts physical properties based on QSPR descriptors derived from SMILES.
Fast-Sigma	A	Predicts sigma profiles based on QSPR descriptors derived from SMILES.

Future releases will incorporate more functionality.

In addition to introducing the Database and CRSManager in the following section, the *examples section* (page 135) contains various template scripts and helpful examples. Exploring the example section is one of the simplest ways to get started with the module.

## Database

### Introduction

The sql database contains the following tables.

Table name	Description
Compound	contains unique compounds along with their COSKF file based on either CAS number or any preferred identifier
Conformer	contains multiple conformers along with their COSKF file
Physical-Property	contains the physical properties input by user
Prop-Pred	contains the estimated physical properties using QSPR methods based on SMILES

These tables can be visualized using open-source tools like [DB Browser](https://sqlitebrowser.org) (https://sqlitebrowser.org) or accessed directly through methods within the COSKFDatabase class.

**Attention:** This module currently does not support charged species, polymers, or multiple-species with dissociation and association. While it's feasible to add these compounds to the database, it's worth noting that the SMILES generated from the coordinates using OpenBabel might be inaccurate. Additionally, the PropPred tool should not be utilized for charged species and polymers.

**Note:** Currently, the `.coskf` files themselves are stored as raw `.coskf` files in a directory called `SCM_PYCRS_COSKF_DB`, which will be created in the parent directory of the `path` variable. This database will

not be overwritten or deleted by this `COSKFDatabase` class. This directory exists for the convenience of the user, but it is best to access all files via the API below as the exact format of the database may evolve over time.

## Basic Usage

In this brief example, we'll demonstrate how to interact with the database. This includes adding a compound, searching for a compound, specifying experimental physical properties, and utilizing PropPred to estimate physical properties.

There are a few important points to note when using the `add_compound` method:

1. If the `coskf_path` parameter is not provided by the user, the method will attempt to locate the ADFCRS-2018 database path if it's available. Please ensure to install the ADFCRS-2018 database via `ampackages` (`SCM->Packages`) before running the below script.
2. If the `cas` parameter is not provided by the user, the method will attempt to use the value stored in the 'Compound Data' section within the `.coskf` file if it's available.
3. If the `name` parameter is not provided by the user, the method will prioritize using the IUPAC name, CAS number, identifier, or the name of the `.coskf` file. The IUPAC name will be attempted to be retrieved from the 'Compound Data' section within the `.coskf` file.

```
from pyCRS.Database import COSKFDatabase

#Please ensure to install the ADFCRS-2018 database via ampackages (SCM->Packages)
->before running the below script #

db = COSKFDatabase("my_coskf_db.db")
db.add_compound("Water.coskf", name="Water", cas="7732-18-5")
db.add_compound("Methanol.coskf")
db.add_compound("Ethanol.coskf")
db.add_compound("Benzene.coskf")
db.add_compound("Ibuprofen.coskf")

rows = db.get_compounds(["7732-18-5", "Ibuprofen"])
for key, row in rows.items():
    print(row[0])

db.add_physical_property("Ibuprofen", "meltingpoint", 347.6)
db.add_physical_property("Ibuprofen", "hfusion", 27.94, unit="kJ/mol")
db.estimate_physical_property("Ibuprofen")

EXP = db.get_physical_properties("Ibuprofen")[0]
QSPR = db.get_physical_properties("Ibuprofen", source="PropPred")[0]

print("experimental value = ", EXP.hfusion, " estimated value = ", QSPR.hfusion)
```

The output produced is the following:

```
CompoundRow(compound_id=1, conformer_id=1, name='water', cas='7732-18-5',
->identifier=None, smiles='O', resolved_smiles='O', coskf='Water_1.coskf', Egas=-323.
->935, Ecosmo=-330.386, nring=0)
CompoundRow(compound_id=4, conformer_id=4, name='ibuprofen', cas='15687-27-1',
->identifier=None, smiles='CC(Cc1ccc(cc1)[C@@H](C(=O)O)C)C', resolved_smiles=
->'CC(Cc1ccc(cc1)[C@@H](C(=O)O)C)C', coskf='Ibuprofen_1.coskf', Egas=-4486.784,
```

(continues on next page)

(continued from previous page)

```
↪Ecosmo=-4494.859, nring=6)
experimental value = 6.678 estimated value = 5.744
```

## Adding multiple conformers

When adding multiple conformers (i.e. multiple COSKF files) of a compound to the database using `add_compound` method, three confirmation steps are attempted:

1. Initially, the set of conformers in the database is retrieved by matching the CAS number or identifier. If it's not found the compound will be added to the database as a new compound.
2. Subsequently, it attempts to generate the canonical SMILES from its coordinates to confirm that the compound matches the one in the database. If this resolution fails, one can bypass this check by setting `ignore_smiles_check=True`.
3. Lastly, it conducts duplicate recognition using `UniqueConformersCrest` in `AMSC conformer` tool to ensure that it's not a duplicate in the database. If this recognition fails, one can skip this step by setting `ignore_duplicates=True`.

By default, the conformer with the lowest gas phase bond energy is stored in the COMPOUND TABLE. However, a specific conformer can be chosen using its conformer ID through the `update_compound_by_conformer_id` method.

For detailed information on these methods, please refer to the [Database API documentation](#) (page 103).

## CRSManager

### Introduction

The module provides a user-friendly approach for configuring standard `crsJob` setups via Python scripting.

For instance, here is an example how we set up the solid solubility calculation using `CRSManager` once the database has been set up in advance. This approach provides a more convenient method for conducting high-throughput calculations without the need for specific handling, such as [solubility screening](#) (page 157).

*Python code using `pyCRS`*

```
from pyCRS.CRSManager import CRSSystem
cal = CRSSystem()
mixture = {}
mixture["Water"] = 1.0
mixture["Ibuprofen"] = 0
cal.add_Mixture( mixture,
                 database="my_coskf_db.db",
                 temperature=298.15,
                 problem_type="solubility",
                 solute="solid")
print(cal.get_input(0))
```

Alternatively, the same calculation can be set up through `PLAMS`, as demonstrated below. This approach requires users to input correct settings but provides greater flexibility in managing complex workflows. For example, the [advanced scripting example section](#) (page 153) demonstrates various tasks, such as eutectic and ionic liquid calculations, and more.

*Python code using `PLAMS`*

```

from scm.plams import Settings, init, finish, CRSJob
import os

##### Note: Please ensure to install the ADFCRS-2018 database via amspackages_
↳ (SCM->Packages) before running the below script #####

coskf_path = os.path.join(os.environ["SCM_PKG_ADFCRSDIR"] , "ADFCRS-2018")

settings = Settings()
settings.input.property._h = "SOLUBILITY"
settings.input.method = "COSMORS"

num_compounds = 2
compounds = [Settings() for i in range(num_compounds)]
compounds[0]._h = os.path.join(coskf_path, "Water.coskf" )
compounds[0].frac1 = 1.0
compounds[1]._h = os.path.join(coskf_path, "Ibuprofen.coskf" )
compounds[1].frac1 = 0.0
compounds[1].nring = 6
compounds[1].hfusion = 27.94/4.184
compounds[1].meltingpoint = 347.6

settings.input.temperature = 298.15

# specify the compounds as the compounds to be used in the calculation
settings.input.compound = compounds
# create a job that can be run by COSMO-RS
my_job = CRSJob(settings=settings)
print(my_job.get_input())

```

**Attention:** This module currently does not support charged species, polymers, or multiple-species with dissociation and association.

## Basic Usage

In this example, we'll create a database containing ethanol and benzene from the ADFCRS-2018 database. Then, we'll instantiate a CRSSystem and generate several CRSMixture instances using the add\_Mixture method. After executing the calculation with the runCRSJob method, the corresponding CRSResults instances will be generated and stored in the outputs attribute. Finally, we can retrieve the .crskf file and the activity coefficient using methods provided by CRSResults.

```

from scm.plams import Settings, init, finish, CRSJob, config
from pyCRS.CRSManger import CRSSystem
from pyCRS.Database import COSKFDatabase
import numpy as np
import os, sys

init()
config.log.stdout = 0 # suppress plams output default=3

```

(continues on next page)

(continued from previous page)

```

#create a database containing ethanol and benzene. please ensure install the ADFCRS-
↳2018 database first via amspackages (SCM -> Packages)
db = COSKFDatabase("my_coskf_db.db")
db.add_compound("Ethanol.coskf")
db.add_compound("Benzene.coskf",cas="71-43-2")

#create a CRSSystem instance
cal = CRSSystem()

#generate several CRSMixture instance
for i, x in enumerate(np.linspace(0, 1, 11)):
    mixture = {}
    mixture["ethanol"] = x
    mixture["71-43-2"] = 1-x
    cal.add_Mixture(mixture,database="my_coskf_db.db",temperature=298.15,problem_type=
↳"ACTIVITYCOEF",method="COSMORS")

#print out the first calculation setting in the CRSSystem class
#print(cal.get_input(0))

#execution calculation for all CRSMixture
cal.runCRSJob()

gamma_ethanol = []
gamma_benzene = []
#retrieve result using methods provided by CRSResults
for out in cal.outputs:
    crskf = out.kfpath()
    res = out.get_results()
    gamma_ethanol.append(res["gamma"][0][0])
    gamma_benzene.append(res["gamma"][1][0])

print(f"the last crskf file = {os.path.basename(crskf)}")
print(f"activity coefficient of ethanol = {gamma_ethanol}")
print(f"activity coefficient of benzene = {gamma_benzene}")

finish()

```

This generates the following output:

```

the last crskf file = crsJob_10.crskf
activity coefficient of ethanol = [41.59625151549539, 3.539543443573337, 2.
↳234237060852824, 1.712495443157569, 1.4321494593043322, 1.2612087014707123, 1.
↳1507589807729406, 1.0784886266786704, 1.0329501218595156, 1.0079137778145097, 1.0]
activity coefficient of benzene = [1.0, 1.0632078891234995, 1.1497425466225775, 1.
↳2547602114699132, 1.3804970769948137, 1.5309168719748845, 1.7115353244885652, 1.
↳9297200775482874, 2.1952374886461046, 2.521062652267439, 2.9247536001030006]

```

## Additional Keywords

Additional keywords can be used in conjunction with certain problem types:

Key-words name	Description
vp_corr	When set to True, it attempts to use the vapor pressure of pure compounds to adjust the gas phase chemical potential, potentially increasing vapor pressures in a mixture. The default value is False.
density_corr	When set to True, it attempts to utilize the volume of a pure compound derived from its density instead of the cavity volume used in the COSMO calculation. The default value is False.
solubility	This parameter allows you to specify 'solid', 'gas', or 'liquid,' applicable to SOLUBILITY and PURESOLUBILITY problem types. The default value is 'solid'.
isotherm	This parameter offers the choice between 'isotherm', 'isobar' or 'flashpoint', applicable to BINMIXCOEF, TERNARYMIX, and COMPOSITIONLINE. The default value is 'isotherm'.
adiabatic	This parameter provides the PLAMS approach to set up various parameters such as the VolumeQuotient in LOP, DensitySolvent in SOLUBILITY, Nfrac in BINMIXCOEF/TERNARYMIX/COMPOSITIONLINE, Nprofile/SigmaMax in SIGMAPROFILE/SIGMAPOTENTIAL.

For certain problem types, necessary physical properties are retrieved from the PhysicalProperty TABLE and PropPred TABLE in the database. The value from the PhysicalProperty TABLE is prioritized; if unavailable, the system attempts to use the value from the PropPred TABLE instead. Failure to obtain either value may result in calculation failure.

For instance, you can find the following examples.

```
# Solid solubility calculation
db.add_physical_property("Benzene", "hfusion", 2.37, unit="kcal/mol")
db.add_physical_property("Benzene", "meltingpoint", 278.7)
mixture = {}
mixture["Water"] = 1
mixture["Benzene"] = 0
cal.add_Mixture( mixture,
                 database="my_coskf_db.db",
                 temperature="273.15 373.15 10",
                 problem_type="SOLUBILITY",
                 solute="solid")

# Binary isobar VLE using the vapor pressure from PropPred
mixture = {}
mixture["methanol"] = 0.5
mixture["ethanol"] = 0.5
db.estimate_physical_property(["methanol", "ethanol"])
additional_sett = Settings()
additional_sett.input.property.Nfrac = 10
cal.add_Mixture( mixture,
                 database="my_coskf_db.db",
                 pressure=1.01325,
                 problem_type="BINMIXCOEF",
                 iso="isobar",
                 additional_sett=additional_sett,
                 vp_corr=True)

# Partition Coefficient for benzene/water
mixture = {}
mixture["benzene"] = [1, 0]  #[mole fraction in phase1, mole fraction in phase2]
```

(continues on next page)

(continued from previous page)

```

mixture["water"] = [0, 1] #[mole fraction in phase1, mole fraction in phase2]
mixture["ethanol"] = [0, 0] #[0, 0] for solute
additional_sett = Settings()
additional_sett.input.property.VolumeQuotient = 4.93
cal.add_Mixture( mixture,
                 database="my_coskf_db.db",
                 problem_type="LOGP",
                 additional_sett=additional_sett)

# Sigma profile
mixture = {}
mixture["benzene"] = 1
additional_sett = Settings()
additional_sett.input.property.Nprofile = 100
additional_sett.input.property.SigmaMax = 0.05
cal.add_Mixture( mixture,
                 database = "my_coskf_db.db",
                 problem_type="PURESIGMAPROFILE",
                 method="COSMORS",
                 additional_sett=additional_sett)

```

The available problem types is listed below.

problem_type value	Problem type
ACTIVITYCOEF	Activity Coefficient
BINMIXCOEF	Binary mixture LLE/VLE
TERNARYMIX	Ternary mixture LLE/VLE
COMPOSITIONLINE	Solvent composition line interpolation
SOLUBILITY	Solubility calculation in a mixed solvent
PURESOLUBILITY	Solubility calculation in a pure solvent
LOGP	Partition coefficient calculation
VAPORPRESSURE	Vapor pressure calculation for a mixed solvent
PUREVAPORPRES- SURE	Vapor pressure calculation for a pure solvent
BOILINGPOINT	Boiling point calculation for a mixture
PUREBOILINGPOINT	Boiling point calculation for a pure solvent(s)
FLASHPOINT	Flashpoint calculation for a mixture
SIGMAPROFILE	Sigma profile calculation for a mixture
PURESIGMAPROFILE	Sigma profile calculation for a pure component(s)
SIGMAPOTENTIAL	Sigma potential calculation for a mixture
PURESIGMAPOTEN- TIAL	Sigma potential calculation for a pure component(s)

For detailed information on these methods, please refer to the *CRSManager API documentation* (page 117).

### 10.1.3 API

#### Database

The submodule contain several class for providing an interface to a sql database for managing COSKF files and physical properties.

**class** `pyCRS.Database.COSKFDatabase` (*path*: `str` (<https://docs.python.org/3.8/library/stdtypes.html#str>))

A class provide an interface to a sql database containing the following tables.

Table name	Description
<b>Compound</b>	Unique compounds with COSKF files by CAS number or identifier.
<b>Conformer</b>	Multiple conformers with corresponding COSKF files.
<b>Physical-Property</b>	User-defined physical properties.
<b>Prop-Pred</b>	Estimated properties using QSPR methods from SMILES.

#### Parameters

**path** (`str` (<https://docs.python.org/3.8/library/stdtypes.html#str>)) – Path to the database file. Created if it doesn't exist.

Example:

```
db = COSKFDatabase("my_coskf_db.db")
db.add_compound("Water.coskf")
db.add_compound("Benzene.coskf", cas="71-43-2")
db.add_physical_property("Benzene", "meltingpoint", 278.7)
db.add_physical_property("Benzene", "hfusion", 9.91, unit="kJ/mol")
db.estimate_physical_property("Benzene")
```

**add\_compound** (*coskf\_file*: `str` (<https://docs.python.org/3.8/library/stdtypes.html#str>), *name*: `str` (<https://docs.python.org/3.8/library/stdtypes.html#str>) | `None` (<https://docs.python.org/3.8/library/constants.html#None>) = `None`, *cas*: `str` (<https://docs.python.org/3.8/library/stdtypes.html#str>) | `None` (<https://docs.python.org/3.8/library/constants.html#None>) = `None`, *identifier*: `str` (<https://docs.python.org/3.8/library/stdtypes.html#str>) | `None` (<https://docs.python.org/3.8/library/constants.html#None>) = `None`, *coskf\_path*: `str` (<https://docs.python.org/3.8/library/stdtypes.html#str>) | `None` (<https://docs.python.org/3.8/library/constants.html#None>) = `None`, *smiles*: `str` (<https://docs.python.org/3.8/library/stdtypes.html#str>) | `None` (<https://docs.python.org/3.8/library/constants.html#None>) = `None`, *nring*: `int` (<https://docs.python.org/3.8/library/functions.html#int>) | `None` (<https://docs.python.org/3.8/library/constants.html#None>) = `None`, *ignore\_smiles\_check*: `bool` (<https://docs.python.org/3.8/library/functions.html#bool>) = `False`, *ignore\_duplicates*: `bool` (<https://docs.python.org/3.8/library/functions.html#bool>) = `False`)

Adds a new **.coskf** file to the database.

**Parameters**

**coskf\_file** (*str* (<https://docs.python.org/3.8/library/stdtypes.html#str>)) – a path to the **.coskf** file, or alternatively, the file name of the **.coskf** file if the **coskf\_path** is provided.

**Keyword Arguments**

- **name** (*str*, optional) – Compound name. Default to IUPAC name, CAS number, identifier, or **.coskf** file name if not specified. Can be set via keyword argument or read from the **.coskf** file.
- **cas** (*str*, optional) – CAS number. If not provided, it will attempt to use the value from the **.coskf** file if available.
- **identifier** (*str*, optional) – Chemical identifier of the compound.
- **coskf\_path** (*str*, optional) – Directory containing the **.coskf** file. Defaults to ADFCRS-2018 database path.
- **smiles** (*str*, optional) – SMILES string. Defaults to the value in the **.coskf** file if available.
- **nring** (*int*, optional) – Numbr of ring atoms. Defaults to the value from the **.coskf** file.
- **ignore\_smiles\_check** (*bool*, optional) – If True, skips identity check via SMILES generation. Defaults to False.
- **ignore\_duplicates** (*bool*, optional) – If True, skips duplicate recognition using UniqueConformersCrest in AMSConformer tool. Default to False.

**Note:**

- Each compound must have a unique CAS number or identifier.
- During **add\_compound**, CAS and identifier are checked for uniqueness in the database.
- An error is raised if multiple compounds share the same CAS number and identifier.
- The example below is invalid because both compounds use the same identifier, CRS0001.

```
db.add_compound("Benzene.coskf", cas="71-43-2", identifier="CRS0001")
db.add_compound("Ethanol.coskf", cas="64-17-5", identifier="CRS0001")
```

**add\_physical\_property** (*identifier: str* (<https://docs.python.org/3.8/library/stdtypes.html#str>), *attribute: str* (<https://docs.python.org/3.8/library/stdtypes.html#str>), *value: float* (<https://docs.python.org/3.8/library/functions.html#float>) | *str* (<https://docs.python.org/3.8/library/stdtypes.html#str>), *unit: str* (<https://docs.python.org/3.8/library/stdtypes.html#str>) | *None* (<https://docs.python.org/3.8/library/constants.html#None>) = *None*)

Add a value of a physical property to the PhysicalProperty TABLE in the database using compound's identifier

**Parameters**

- **identifier** (*str* (<https://docs.python.org/3.8/library/stdtypes.html#str>)) – CAS number, identifier or compound name.
- **attribute** (*str* (<https://docs.python.org/3.8/library/stdtypes.html#str>)) – Name of the physical property (eg. meltingpoint or hfusion).
- **value** (*float* or *str*) – Value of the physical property.

- **unit** (*str*, optional) – the unit of the input value. The default units are **K**, **kcal/mol** and **kcal/mol-K**. The following units are accepted and will be automatically converted to the default units: - Temperature: K, C - Enthalpy: kcal/mol, kJ/mol, cal/g, J/g - Heat capacity: kcal/mol-K, kJ/mol-K, cal/g-K, J/g-K - pvap: bar, atm, Pa, mmHg

**Note:** The **vp\_equation** accepts only parameters for pressure in **bar** and temperature in **Kelvin**.

```
db.add_physical_property("Benzene", "meltingpoint", 278.7)
db.add_physical_property("Benzene", "hfusion", 9.91, unit="kJ/mol")
db.add_physical_property("Benzene", "vp_equation", "Antoine")
db.add_physical_property("Benzene", "vp_params", "4.72583, 1660.652, -1.461")
db.add_physical_property("Benzene", "flashpoint", -11.63, unit="C")
#Vapor pressure at 353.25K is 1.01325 bar
db.add_physical_property("Benzene", "tvap", 353.25)
db.add_physical_property("Benzene", "pvap", 1.01325)
```

**clear\_physical\_property** (*identifier: str* (<https://docs.python.org/3.8/library/stdtypes.html#str>) | *List* (<https://docs.python.org/3.8/library/typing.html#typing.List>)[*str* (<https://docs.python.org/3.8/library/stdtypes.html#str>) | *None* (<https://docs.python.org/3.8/library/constants.html#None>) = *None*, *attribute: str* (<https://docs.python.org/3.8/library/stdtypes.html#str>) | *List* (<https://docs.python.org/3.8/library/typing.html#typing.List>)[*str* (<https://docs.python.org/3.8/library/stdtypes.html#str>) | *None* (<https://docs.python.org/3.8/library/constants.html#None>) = *None*)

Clears the value of a physical property in PhysicalProperty TABLE in the database by compound's identifier

#### Parameters

- **identifier** (*str* or *List[str]*, optional) – CAS number, chemical identifier or compound name as a string or a list of strings. If *None*, all compound are selected.
- **attribute** (*str* or *List[str]*, optional) – Specific property to clear as a string or a list of strings. If *None*, all properties are cleared.

```
db.clear_physical_property(["water", "benzene"])
```

**del\_row** (*dbrow: CompoundRow* (*page 110*) | *Dict* (<https://docs.python.org/3.8/library/typing.html#typing.Dict>)[*str* (<https://docs.python.org/3.8/library/stdtypes.html#str>), *List* (<https://docs.python.org/3.8/library/typing.html#typing.List>)[*CompoundRow* (*page 110*)]])

Remove a compound from the database and delete the corresponding **.coskf** file.

#### Parameters

**dbrow** (*CompoundRow* or *Dict[str, List[CompoundRow]]*) – the row to remove from the database

**del\_row\_by\_conformer\_id** (*conformer\_id: int* (<https://docs.python.org/3.8/library/functions.html#int>))

Remove the conformer from the database.

#### Parameters

**conformer\_id** (*int* (<https://docs.python.org/3.8/library/functions.html#int>)) – A integer of intergers representing the conformer in the CONFORMER TABLE.

```
db.del_row_by_conformer_id(1)
```

**del\_rows** (*dbrows*: *List* (<https://docs.python.org/3.8/library/typing.html#typing.List>)[*CompoundRow* (page 110)] | *Dict* (<https://docs.python.org/3.8/library/typing.html#typing.Dict>)[*str* (<https://docs.python.org/3.8/library/stdtypes.html#str>), *List* (<https://docs.python.org/3.8/library/typing.html#typing.List>)[*CompoundRow* (page 110)]]])

Remove multiple compounds from the database and delete the corresponding **.coskf** files.

#### Parameters

**dbrows** (*List*[*CompoundRow*] or *Dict*[*str*, *List*[*CompoundRow*]]) – the rows to remove from the database.

```
db.del_rows(db.get_compounds('benzene'))
```

**estimate\_physical\_property** (*identifier*: *str* (<https://docs.python.org/3.8/library/stdtypes.html#str>) | *List* (<https://docs.python.org/3.8/library/typing.html#typing.List>)[*str* (<https://docs.python.org/3.8/library/stdtypes.html#str>)] | *None* (<https://docs.python.org/3.8/library/constants.html#None>) = *None*, *compound\_id*: *int* (<https://docs.python.org/3.8/library/functions.html#int>) | *List* (<https://docs.python.org/3.8/library/typing.html#typing.List>)[*int* (<https://docs.python.org/3.8/library/functions.html#int>)] | *None* (<https://docs.python.org/3.8/library/constants.html#None>) = *None*)

Estimate the physical properties using the property prediction tool and add the values to the PropPred TABLE in the database

#### Keyword Arguments

- **identifier** (*str* or *List*[*str*], *optional*) – CAS number, chemical identifier or compound name as a string or a list of strings.
- **compound\_id** (*int* or *List*[*int*], *optional*) – an integer or a list representing the compound ID(s).

---

**Note:** The QSPR descriptor used in the property prediction tool is determined from the SMILES string. The selection priority of SMILES is as follows: (1) User-provided SMILES via the `add_compound()` method. (2) SMILES read from the **.coskf** file. (3) SMILES generated by OpenBabel using the compound's coordinates in the **.coskf** file. Please note that the automatically resolved SMILES may be incorrect for some molecules, for instance when bond orders cannot be automatically determined and species with charges.

---

```
db.estimate_physical_property("Benzene")
```

**get\_all\_compounds** () → *List* (<https://docs.python.org/3.8/library/typing.html#typing.List>)[*CompoundRow* (page 110)]

Retrieve all compounds in the database

#### Returns

The full list of *CompoundRow* instances in the database

#### Return type

*List*[*CompoundRow* (page 110)]

**get\_all\_conformers** () → *List* (<https://docs.python.org/3.8/library/typing.html#typing.List>)[*ConformerRow* (page 112)]

Retrieves all conformers in the database

**Returns**

The full list of ConformerRow instances in the database.

**Return type**

List[*ConformerRow* (page 112)]

**get\_all\_physical\_properties** (*source*: str (<https://docs.python.org/3.8/library/stdtypes.html#str>) = 'PhysicalProperty') → List (<https://docs.python.org/3.8/library/typing.html#typing.List>)[*PhysicalPropertyRow* (page 113)] | List (<https://docs.python.org/3.8/library/typing.html#typing.List>)[*PropPredRow* (page 115)]

Retrieve all physical properties in the database

**Parameters**

**source** (str, optional) – Source of the properties. - 'PhysicalProperty' (default): Returns properties from the **PhysicalProperty** table. - 'PropPred': Returns estimated properties from the **PropPred** table.

**Returns**

A list of PhysicalPropertyRow instances or PropPredRow instances in the database.

**Return type**

List(*PhysicalPropertyRow* (page 113)) or List(*PropPredRow* (page 115))

**get\_attribute\_by\_compound\_id** (*attributes*: str (<https://docs.python.org/3.8/library/stdtypes.html#str>) | List (<https://docs.python.org/3.8/library/typing.html#typing.List>)[str (<https://docs.python.org/3.8/library/stdtypes.html#str>)], *compound\_id*: int (<https://docs.python.org/3.8/library/functions.html#int>) | List (<https://docs.python.org/3.8/library/typing.html#typing.List>)[int (<https://docs.python.org/3.8/library/functions.html#int>)], *source*: str (<https://docs.python.org/3.8/library/stdtypes.html#str>) | List (<https://docs.python.org/3.8/library/typing.html#typing.List>)[str (<https://docs.python.org/3.8/library/stdtypes.html#str>)] | None (<https://docs.python.org/3.8/library/constants.html#None>) = None)

Retrieve the list of values for compounds with specified compound\_id(s) in the database

**Parameters**

- **attributes** (str or List[str]) – Attribute(s) to be retrieved.
- **compound\_id** (int or List[int]) – A integer or a list of intergers used to search for compounds in the COMPOUND TABLE.
- **source** (str or List[str], optional) – The table used in the search. Default is COMPOUND TABLE and PhysicalProperty TABLE

**Returns**

A list of tuples containing the values of the specified attributes for the compounds.

**Return type**

list (<https://docs.python.org/3.8/library/stdtypes.html#list>) of attributes

```
db.get_attribute_by_compound_id("name", 1)
db.get_attribute_by_compound_id(["name", "cas", "hfusion"] 1)
db.get_attribute_by_compound_id(["name", "hfusion"], 1, source=["COMPOUND",
↪"PropPred"])
```

**get\_compounds** (*identifier*: *str* (<https://docs.python.org/3.8/library/stdtypes.html#str>) | *List* (<https://docs.python.org/3.8/library/typing.html#typing.List>)[*str* (<https://docs.python.org/3.8/library/stdtypes.html#str>)]) → *Dict* (<https://docs.python.org/3.8/library/typing.html#typing.Dict>)[*str* (<https://docs.python.org/3.8/library/stdtypes.html#str>), *CompoundRow* (page 110)]

Retrieves compounds from the COMPOUND TABLE in the database by matching CAS number, chemical identifier, or name.

#### Parameters

**identifier** (*str* or *List*[*str*]) – CAS number, chemical identifier or compound name as a string or a list of strings.

#### Returns

A dictionary where each key is an input identifier and its corresponding value is the **CompoundRow** instances.

#### Return type

*Dict*[*str* (<https://docs.python.org/3.8/library/stdtypes.html#str>), *CompoundRow* (page 110)]

**get\_compounds\_id** (*identifier*: *str* (<https://docs.python.org/3.8/library/stdtypes.html#str>) | *List* (<https://docs.python.org/3.8/library/typing.html#typing.List>)[*str* (<https://docs.python.org/3.8/library/stdtypes.html#str>)]) → *List* (<https://docs.python.org/3.8/library/typing.html#typing.List>)[*int* (<https://docs.python.org/3.8/library/functions.html#int>) | *None* (<https://docs.python.org/3.8/library/constants.html#None>)]

Retrieves compound id from the COMPOUND TABLE in the database by matching CAS number, chemical identifier, or name.

#### Parameters

**identifier** (*str* or *List*[*str*]) – CAS number, chemical identifier or compound name as a string or a list of strings.

#### Returns

A list of compound IDs corresponding to the input identifier. If a name is not found, *None* is returned at the corresponding position.

#### Return type

*List*[*Optional*[*int* (<https://docs.python.org/3.8/library/functions.html#int>)]]

**get\_conformers** (*identifier*: *str* (<https://docs.python.org/3.8/library/stdtypes.html#str>) | *List* (<https://docs.python.org/3.8/library/typing.html#typing.List>)[*str* (<https://docs.python.org/3.8/library/stdtypes.html#str>)]) → *Dict* (<https://docs.python.org/3.8/library/typing.html#typing.Dict>)[*str* (<https://docs.python.org/3.8/library/stdtypes.html#str>), *ConformerRow* (page 112)]

Retrieves conformers from the CONFORMER TABLE in the database by matching CAS number, chemical identifier, or name.

#### Parameters

**identifier** (*str* or *list*) – CAS number, chemical identifier or compound name as a string or a list of strings.

#### Returns

A list of *ConformerRow* instances that match the search criteria.

#### Return type

*Dict*[*str* (<https://docs.python.org/3.8/library/stdtypes.html#str>), *ConformerRow* (page 112)]

**get\_physical\_properties** (*identifier*: *str* (<https://docs.python.org/3.8/library/stdtypes.html#str>) | *List* (<https://docs.python.org/3.8/library/typing.html#typing.List>)[*str* (<https://docs.python.org/3.8/library/stdtypes.html#str>)] | *None* (<https://docs.python.org/3.8/library/constants.html#None>) = *None*, *compound\_id*: *int* (<https://docs.python.org/3.8/library/functions.html#int>) | *List* (<https://docs.python.org/3.8/library/typing.html#typing.List>)[*int* (<https://docs.python.org/3.8/library/functions.html#int>)] | *None* (<https://docs.python.org/3.8/library/constants.html#None>) = *None*, *source*: *str* (<https://docs.python.org/3.8/library/stdtypes.html#str>) = 'PhysicalProperty') → *List* (<https://docs.python.org/3.8/library/typing.html#typing.List>)[*PhysicalPropertyRow* (page 113)] | *List* (<https://docs.python.org/3.8/library/typing.html#typing.List>)[*PropPredRow* (page 115)]

Retrieve physical properties in the database by matching CAS number, chemical identifier, name or compound id.

#### Parameters

- **identifier** (*str* or *List*[*str*], optional) – CAS number, chemical identifier or compound name as a string or a list of strings. If *None*, *compound\_id* must be provided.
- **compound\_id** (*int* or *List*[*int*], optional) – Compound ID as an integer or a list of integers. If *None*, *identifier* must be provided.
- **source** (*str*, optional) – Source of the properties. - 'PhysicalProperty' (default): Returns properties from the **PhysicalProperty** table. - 'PropPred': Returns estimated properties from the **PropPred** table.

#### Returns

A list of **PhysicalPropertyRow** or **PropPredRow** instances, depending on the source.

#### Return type

*List*[*PhysicalPropertyRow* (page 113)] or *List*[*PropPredRow* (page 115)]

**modify\_attribute\_by\_compound\_id** (*attribute*: *str* (<https://docs.python.org/3.8/library/stdtypes.html#str>), *value*: *str* (<https://docs.python.org/3.8/library/stdtypes.html#str>) | *int* (<https://docs.python.org/3.8/library/functions.html#int>), *compound\_id*: *int* (<https://docs.python.org/3.8/library/functions.html#int>))

Modifies the value of a specified attribute for a given compound ID.

#### Parameters

- **attribute** (*str* (<https://docs.python.org/3.8/library/stdtypes.html#str>)) – Attribute to modify. It can be one of the following: 'name', 'cas', 'identifier', 'smiles', 'nring'.
- **value** (*str* or *int*) – the new value of the specified attribute.
- **compound\_id** (*int* (<https://docs.python.org/3.8/library/functions.html#int>)) – an integer representing the compound ID.

```
db.modify_attribute_by_compound_id("identifier", "InChI=1S/C6H6/c1-2-4-6-5-3-1/↵h1-6H", 0)
```

**update\_compound\_by\_conformer\_id** (*compound\_id*: *int* (<https://docs.python.org/3.8/library/functions.html#int>), *conformer\_id*: *int* (<https://docs.python.org/3.8/library/functions.html#int>))

Update the data for a compound ID row in the COMPOUND TABLE using the data from a conformer ID row in the CONFORMER TABLE.

#### Parameters

- **compound\_id** (*int* (<https://docs.python.org/3.8/library/functions.html#int>)) – A integer representing compound id corresponding to a specific row in the COMPOUND TABLE of the database
- **conformer\_id** (*int* (<https://docs.python.org/3.8/library/functions.html#int>)) – A integer representing conformer id corresponding to a specific row in the CONFORMER TABLE of the database

**update\_compound\_by\_lowestE** (*compound\_id: int* (<https://docs.python.org/3.8/library/functions.html#int>) | *List* (<https://docs.python.org/3.8/library/typing.html#typing.List>) [*int* (<https://docs.python.org/3.8/library/functions.html#int>)] | *None* (<https://docs.python.org/3.8/library/constants.html#None>) = *None*)

Update the data for a compound ID row in the COMPOUND TABLE using the data from a conformer ID row with the lowest energy having the same compound ID in the CONFORMER TABLE.

#### Keyword Arguments

**compound\_id** (*int* or *List[int]*, optional) – Compound ID as an integer or a list of integers. If *None*, updates all compounds in the database.

**visualize\_conformers** (*compound\_id: int* (<https://docs.python.org/3.8/library/functions.html#int>) | *None* (<https://docs.python.org/3.8/library/constants.html#None>) = *None*, *identifier: str* (<https://docs.python.org/3.8/library/stdtypes.html#str>) | *None* (<https://docs.python.org/3.8/library/constants.html#None>) = *None*)

Visualize conformers in ascending order of conformers IDs.

#### Parameters

- **compound\_id** (*int*, optional) – Compound ID for which conformers are visualized.
- **identifier** (*str*, optional) – CAS number, chemical identifier or compound name.

```
class pyCRS.Database.CompoundRow (compound_id: int
    (https://docs.python.org/3.8/library/functions.html#int), conformer_id:
    int (https://docs.python.org/3.8/library/functions.html#int), name: str
    (https://docs.python.org/3.8/library/stdtypes.html#str), cas: str
    (https://docs.python.org/3.8/library/stdtypes.html#str), identifier: str
    (https://docs.python.org/3.8/library/stdtypes.html#str), smiles: str
    (https://docs.python.org/3.8/library/stdtypes.html#str),
    resolved_smiles: str
    (https://docs.python.org/3.8/library/stdtypes.html#str), coskf: str
    (https://docs.python.org/3.8/library/stdtypes.html#str), Egas: float
    (https://docs.python.org/3.8/library/functions.html#float), Ecosmo:
    float (https://docs.python.org/3.8/library/functions.html#float), nring:
    int (https://docs.python.org/3.8/library/functions.html#int))
```

A data class to represent the contents of a row in a COMPOUND TABLE in COSKFDATABASE

#### **compound\_id**

A unique identifier for a specific row in the COMPOUND TABLE of the database

#### Type

*int* (<https://docs.python.org/3.8/library/functions.html#int>)

**conformer\_id**

A unique identifier for a specific row in the CONFORMER TABLE of the database

**Type**

`int` (<https://docs.python.org/3.8/library/functions.html#int>)

**name**

The name associated with the row in the COMPOUND TABLE

**Type**

`str` (<https://docs.python.org/3.8/library/stdtypes.html#str>)

**cas**

The CAS number associated with the row, i.e., the compound

**Type**

`str` (<https://docs.python.org/3.8/library/stdtypes.html#str>)

**identifier**

The chemical identifier associated with the row, i.e., the compound

**Type**

`str` (<https://docs.python.org/3.8/library/stdtypes.html#str>)

**smiles**

The SMILES string provided by user

**Type**

`str` (<https://docs.python.org/3.8/library/stdtypes.html#str>)

**resolved\_smiles**

The derived SMILES string obtained using OpenBabel from the coordinates in the COSKF file.

**Type**

`str` (<https://docs.python.org/3.8/library/stdtypes.html#str>)

**coskf**

The filename of the `.coskf` file stored in the local `SCM_PYCRS_COSKF_DB` directory

**Type**

`str` (<https://docs.python.org/3.8/library/stdtypes.html#str>)

**Egas**

The gas phase bond energy rounded to 3 decimal places in kcal/mol

**Type**

`float` (<https://docs.python.org/3.8/library/functions.html#float>)

**Ecosmo**

The bond energy in a perfect conductor rounded to 3 decimal places in kcal/mol

**Type**

`float` (<https://docs.python.org/3.8/library/functions.html#float>)

**nring**

The number of ring atoms

**Type**

`int` (<https://docs.python.org/3.8/library/functions.html#int>)

**db\_path**

The path to the `.coskf` file directory

**Type**

`str` (<https://docs.python.org/3.8/library/stdtypes.html#str>)

**get\_full\_coskf\_path()**

Returns the full path of the corresponding `.coskf` file

**read\_coskf()**

Opens the `.coskf` file corresponding to the database entry and returns a `scm.plams.KFFFile` instance

```
class pyCRS.Database.ConformerRow (conformer_id: int
                                   (https://docs.python.org/3.8/library/functions.html#int),
                                   compound_id: int
                                   (https://docs.python.org/3.8/library/functions.html#int), name: str
                                   (https://docs.python.org/3.8/library/stdtypes.html#str), cas: str
                                   (https://docs.python.org/3.8/library/stdtypes.html#str), identifier: str
                                   (https://docs.python.org/3.8/library/stdtypes.html#str), smiles: str
                                   (https://docs.python.org/3.8/library/stdtypes.html#str),
                                   resolved_smiles: str
                                   (https://docs.python.org/3.8/library/stdtypes.html#str), coskf: str
                                   (https://docs.python.org/3.8/library/stdtypes.html#str), Egas: float
                                   (https://docs.python.org/3.8/library/functions.html#float), Ecosmo:
                                   float (https://docs.python.org/3.8/library/functions.html#float),
                                   nring: int (https://docs.python.org/3.8/library/functions.html#int))
```

A data class to represent the contents of a row in a CONFORMER TABLE in `COSKFDatabase`

**conformer\_id**

A unique identifier for a specific row in the CONFORMER TABLE of the database

**Type**

`int` (<https://docs.python.org/3.8/library/functions.html#int>)

**compound\_id**

A unique identifier for a specific row in the COMPOUND TABLE of the database

**Type**

`int` (<https://docs.python.org/3.8/library/functions.html#int>)

**name**

The name associated with the row in the CONFORMER TABLE

**Type**

`str` (<https://docs.python.org/3.8/library/stdtypes.html#str>)

**cas**

The CAS number associated with the row, i.e., the compound

**Type**

`str` (<https://docs.python.org/3.8/library/stdtypes.html#str>)

**identifier**

The chemical identifier associated with the row, i.e., the compound

**Type**

`str` (<https://docs.python.org/3.8/library/stdtypes.html#str>)

**smiles**

The SMILES string provided by user

**Type**

`str` (<https://docs.python.org/3.8/library/stdtypes.html#str>)

**resolved\_smiles**

The derived SMILES string obtained using OpenBabel from the coordinates in the COSKF file

**Type**

`str` (<https://docs.python.org/3.8/library/stdtypes.html#str>)

**coskf**

The filename of the `.coskf` file stored in the local `SCM_PYCRS_COSKF_DB` directory

**Type**

`str` (<https://docs.python.org/3.8/library/stdtypes.html#str>)

**Egas**

The gas phase bond energy rounded to 3 decimal places in kcal/mol

**Type**

`float` (<https://docs.python.org/3.8/library/functions.html#float>)

**Ecosmo**

The bond energy in a perfect conductor rounded to 3 decimal places in kcal/mol

**Type**

`float` (<https://docs.python.org/3.8/library/functions.html#float>)

**nring**

The number of ring atoms

**Type**

`int` (<https://docs.python.org/3.8/library/functions.html#int>)

**db\_path**

The path to the `.coskf` file directory

**Type**

`str` (<https://docs.python.org/3.8/library/stdtypes.html#str>)

**get\_full\_coskf\_path()**

Returns the full path of the corresponding `.coskf` file

**read\_coskf()**

Opens the `.coskf` file corresponding to the database entry and returns a `scm.plams.KFFFile` instance

```

class pyCRS.Database.PhysicalPropertyRow (compound_id: int
                                         (https://docs.python.org/3.8/library/functions.html#int),
                                         meltingpoint: float
                                         (https://docs.python.org/3.8/library/functions.html#float),
                                         hfusion: float
                                         (https://docs.python.org/3.8/library/functions.html#float),
                                         cpfusion: float
                                         (https://docs.python.org/3.8/library/functions.html#float),
                                         boilingpoint: float
                                         (https://docs.python.org/3.8/library/functions.html#float),
                                         density: float
                                         (https://docs.python.org/3.8/library/functions.html#float),
                                         flashpoint: float
                                         (https://docs.python.org/3.8/library/functions.html#float),
                                         dielectricconstant: float
                                         (https://docs.python.org/3.8/library/functions.html#float),
                                         vp_equation: str
                                         (https://docs.python.org/3.8/library/stdtypes.html#str),
                                         vp_params: str
                                         (https://docs.python.org/3.8/library/stdtypes.html#str),
                                         tvap: float
                                         (https://docs.python.org/3.8/library/functions.html#float),
                                         pvap: float
                                         (https://docs.python.org/3.8/library/functions.html#float),
                                         Mn: float
                                         (https://docs.python.org/3.8/library/functions.html#float))

```

A data class to represent the contents of a row in a PhysicalProperty TABLE in COSKFDATABASE

#### **compound\_id**

A unique identifier for a specific row in the COMPOUND TABLE of the database

##### **Type**

`int` (<https://docs.python.org/3.8/library/functions.html#int>)

#### **meltingpoint**

melting temperature (K)

##### **Type**

`float` (<https://docs.python.org/3.8/library/functions.html#float>)

#### **hfusion**

enthalpy of fusion (kcal/mol)

##### **Type**

`float` (<https://docs.python.org/3.8/library/functions.html#float>)

#### **cpfusion**

heat capacity of fusion (kcal/mol-K) calculated as the difference between the heat capacity in the liquid state and the heat capacity in the solid state.

##### **Type**

`float` (<https://docs.python.org/3.8/library/functions.html#float>)

#### **boilingpoint**

boiling pointK (K)

##### **Type**

`float` (<https://docs.python.org/3.8/library/functions.html#float>)

**density**

liquid density (kg/L)

**Type**`float` (<https://docs.python.org/3.8/library/functions.html#float>)**flashpoint**

flash point (K)

**Type**`float` (<https://docs.python.org/3.8/library/functions.html#float>)**dielectricconstant**

dielectric constant

**Type**`float`**vp\_equation**

The vapor pressure equation to use. Unit in bar. Options include: ANTOINE, VPM1 and DIPPR101

**Type**`str` (<https://docs.python.org/3.8/library/stdtypes.html#str>)**vp\_params**

Parameters for the vp\_equation, expressed as "A, B, C, D, E"

**Type**`str` (<https://docs.python.org/3.8/library/stdtypes.html#str>)**tvap**

Temperature(K) at pvap

**Type**`float` (<https://docs.python.org/3.8/library/functions.html#float>)**pvap**

Pressure(bar) at tvap

**Type**`float` (<https://docs.python.org/3.8/library/functions.html#float>)**Mn**

polymer average molecular weight (g/mol)

**Type**`float` (<https://docs.python.org/3.8/library/functions.html#float>)**Vapor Pressure Equations:****ANTOINE:**

$$\log_{10}(P) = A - B/(C+T)$$

**DIPPR101:**

$$\ln(P) = A + B/T + C*\ln(T) + D*T**E$$

**VPM1:**

$$\ln(P) = A/T + B*\ln(T) + C*T + D$$

```

class pyCRS.Database.PropPredRow (compound_id: int
                                   (https://docs.python.org/3.8/library/functions.html#int), adopt_smiles:
                                   str (https://docs.python.org/3.8/library/stdtypes.html#str),
                                   meltingpoint: float
                                   (https://docs.python.org/3.8/library/functions.html#float), hfusion:
                                   float (https://docs.python.org/3.8/library/functions.html#float),
                                   boilingpoint: float
                                   (https://docs.python.org/3.8/library/functions.html#float), density:
                                   float (https://docs.python.org/3.8/library/functions.html#float),
                                   flashpoint: float
                                   (https://docs.python.org/3.8/library/functions.html#float),
                                   dielectricconstant: float
                                   (https://docs.python.org/3.8/library/functions.html#float),
                                   vp_equation: str
                                   (https://docs.python.org/3.8/library/stdtypes.html#str), vp_params: str
                                   (https://docs.python.org/3.8/library/stdtypes.html#str))

```

A data class to represent the contents of a row in a PropPred TABLE in COSKFDatabase

#### **compound\_id**

A unique identifier for a specific row in the COMPOUND TABLE of the database

##### **Type**

`int` (<https://docs.python.org/3.8/library/functions.html#int>)

#### **adopt\_smiles**

The SMILES used for QSPR method

##### **Type**

`str` (<https://docs.python.org/3.8/library/stdtypes.html#str>)

#### **meltingpoint**

melting temperature (K)

##### **Type**

`float` (<https://docs.python.org/3.8/library/functions.html#float>)

#### **hfusion**

enthalpy of fusion (kcal/mol)

##### **Type**

`float` (<https://docs.python.org/3.8/library/functions.html#float>)

#### **boilingpoint**

boiling point (K)

##### **Type**

`float` (<https://docs.python.org/3.8/library/functions.html#float>)

#### **density**

liquid density (kg/L)

##### **Type**

`float` (<https://docs.python.org/3.8/library/functions.html#float>)

#### **flashpoint**

flash point (K)

##### **Type**

`float` (<https://docs.python.org/3.8/library/functions.html#float>)

**dielectricconstant**

dielectric constant

**Type**

flash

**vp\_equation**

The vapor pressure equation to use. Unit in bar. VPM1

**Type**

str (<https://docs.python.org/3.8/library/stdtypes.html#str>)

**vp\_params**

Parameters for the vp\_equation, expressed as “A, B, C, D, E”

**Type**

str (<https://docs.python.org/3.8/library/stdtypes.html#str>)

**CRSManager**

This submodule facilitate the creation, execution, and output processes for mutiple `crsJob` and is intended to be used in conjunction with the `pyCRS.database` module.

**class** `pyCRS.CRSManager.CRSSystem` (*iteration=False*)

This class manages multiple `CRSMixture` instances, handling their creation, execution, and result collection. Use `add_Mixture()` to append `CRSMixture` objects to the mixture list; results are stored in outputs with the same order. Requires `pyCRS.database` for COSKF access.

**iteration**

Enable iteration mode; mixtures will manage `CRSIteration` instead of a single `CRSJob`.

**Type**

bool (<https://docs.python.org/3.8/library/functions.html#bool>)

**num\_mix**

Number of mixtures managed by this `CRSSystem`.

**Type**

int (<https://docs.python.org/3.8/library/functions.html#int>)

**mixture**

Ordered list of mixtures in this system.

**Type**

list[`CRSMixture`]

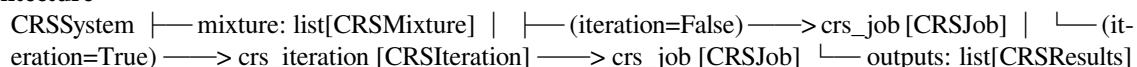
**outputs**

Results aligned to mixture order.

**Type**

list[`CRSResults`]

**Architecture**



## Notes

Use `runCRSJob()` when iteration is False; use `runCRSIteration()` when iteration is True.

### Example

```
db = COSKFDatabase("my_coskf_db.db")
crs = CRSSystem()
crs.add_Mixture(mixture={"ethanol": 0.0, "water":1.0}, database=db)
crs.runCRSJob()
res = crs.outputs[0].get_results()
print(f"{res['temperature']} : {res['gamma']}")
```

```
db = COSKFDatabase("my_coskf_db.db")
crs = CRSSystem(iteration=True)
crs.add_Mixture(mixture={"ethanol": 0.0, "water":1.0}, database=db)
crs.runCRSIteration()
act = cal.get_activity_coefficients()
print(f"activity coefficient of mixture0: comp0: point0={act[0][0][0]:.4f}")

parameter_sett = Settings()
parameter_sett.input.CRSParameters.aeff = 6.5
cal.runCRSIteration(parameter_sett=parameter_sett)
act = cal.get_activity_coefficients()
print(f"activity coefficient of mixture0: comp0: point0={act[0][0][0]:.4f}")
```

**add\_Mixture** (*mixture*: *dict* (<https://docs.python.org/3.8/library/stdtypes.html#dict>), *temperature*: *float* (<https://docs.python.org/3.8/library/functions.html#float>) | *str* (<https://docs.python.org/3.8/library/stdtypes.html#str>) = 298.15, *problem\_type*: *str* (<https://docs.python.org/3.8/library/stdtypes.html#str>) = 'activitycoef', *database*: *str* (<https://docs.python.org/3.8/library/stdtypes.html#str>) | **COSKFDatabase** (page 103) = 'my\_coskf\_db.db', *method*: *str* (<https://docs.python.org/3.8/library/stdtypes.html#str>) = 'COSMORS', *pressure*: *float* (<https://docs.python.org/3.8/library/functions.html#float>) | *str* (<https://docs.python.org/3.8/library/stdtypes.html#str>) = 1.01325, *jobname*: *str* (<https://docs.python.org/3.8/library/stdtypes.html#str>) | *None* (<https://docs.python.org/3.8/library/constants.html#None>) = *None*, *conformer*: *bool* (<https://docs.python.org/3.8/library/functions.html#bool>) = *False*, *massfraction*: *bool* (<https://docs.python.org/3.8/library/functions.html#bool>) = *False*, *density\_corr*: *bool* (<https://docs.python.org/3.8/library/functions.html#bool>) = *False*, *vp\_corr*: *bool* (<https://docs.python.org/3.8/library/functions.html#bool>) = *False*, *solute*: *str* (<https://docs.python.org/3.8/library/stdtypes.html#str>) = 'solid', *iso*: *str* (<https://docs.python.org/3.8/library/stdtypes.html#str>) = 'isotherm', *additional\_sett*: *Settings* | *None* (<https://docs.python.org/3.8/library/constants.html#None>) = *None*, *multi\_species*: *dict* (<https://docs.python.org/3.8/library/stdtypes.html#dict>) | *None* (<https://docs.python.org/3.8/library/constants.html#None>) = *None*)

Adds a new CRSMixture to the **mixture** attribute of the CRSSystem.

#### Parameters

**mixture** (*dict* (<https://docs.python.org/3.8/library/stdtypes.html#dict>)) – Composition as {Identifier : mole/mass fraction}

#### Keyword Arguments

- **temperature** (*float* or *str*) – Temp in Kelvin (default 298.15). If str, format can be “T1” or “T1 T2 ntemp”. Where T1 is the starting temperature, T2 is the ending temperature, and ntemp is the number of intervals between T1 and T2.

- **problem\_type** (`str` (<https://docs.python.org/3.8/library/stdtypes.html#str>)) – Calculation type (default “activitycoef”).
- **database** (`str` or `COSKFDatabase`) – Path or instance of `COSKFDatabase` (default “my\_coskf\_db.db”).
- **method** (`str` (<https://docs.python.org/3.8/library/stdtypes.html#str>)) – Version of COSMORS or COSMOSAC (default “COSMORS”).
- **pressure** (`float` or `str`) – Pressure in bar (default 1.01325). If `str`, format can be “P1” or “P1 P2 npress”. Where P1 is the starting pressure, P2 is the ending pressure, and npress is the number of intervals between P1 and P2.
- **jobname** (`str` (<https://docs.python.org/3.8/library/stdtypes.html#str>)) – Job name in `plam_workdir` (defaults “crsJob\_n”).
- **conformer** (`bool` (<https://docs.python.org/3.8/library/functions.html#bool>)) – Use multiple conformers in COSMO-RS (default False).
- **massfraction** (`bool` (<https://docs.python.org/3.8/library/functions.html#bool>)) – Input fraction as mass (default False).
- **density\_corr** (`bool` (<https://docs.python.org/3.8/library/functions.html#bool>)) – Use pure compound density for volume. Default to False (uses COSMO volume).
- **vp\_corr** (`bool` (<https://docs.python.org/3.8/library/functions.html#bool>)) – Correct gas phase chemical potential with vapor pressure (default False).
- **solute** (`str` (<https://docs.python.org/3.8/library/stdtypes.html#str>)) – Solute state (“solid”, “gas”, “liquid”) in `SOLUBILITY` and `PURESOLUBILITY`. Default is “solid”.
- **iso** (`str` (<https://docs.python.org/3.8/library/stdtypes.html#str>)) – Condition (“isotherm”, “isobar”, “flashpoint”) for `BINMIXCOEF`, `TERNARYMIX`, and `COMPOSITIONLINE`. Default is “isotherm”.
- **additional\_sett** (`Settings`) – Optional custom settings (default None).
- **multi\_species** (`dict` (<https://docs.python.org/3.8/library/stdtypes.html#dict>)) – Multi-species settings as {Identifier: Settings} (default None).

---

**Note:** The identifiers for a compound can be the name, CAS number, or identifier in the `pyCRS.database`. When searching in the database, the input string will be converted to lowercase, except when it starts with ‘InChI’.

---

The available problem types are listed below.

problem_type value	Problem type
ACTIVITYCOEF	Activity Coefficient
SOLUBILITY	Solubility calculation in a mixed solvent
PURESOLUBILITY	Solubility calculation in a pure solvent
LOGP	Partition coefficient calculation
VAPORPRESSURE	Vapor pressure calculation for a mixed solvent
PUREVAPORPRES- SURE	Vapor pressure calculation for a pure solvent
BOILINGPOINT	Boiling point calculation for a mixture
PUREBOILING- POINT	Boiling point calculation for a pure solvent(s)
FLASHPOINT	Flashpoint calculation for a mixture
LLE	Liquid-Liquid Equilibrium calculation
STABILITY	Phase stability test
BINMIXCOEF	Binary mixture LLE/VLE
TERNARYMIX	Ternary mixture LLE/VLE
COMPOSITION- LINE	Solvent composition line interpolation
SIGMAPROFILE	Sigma profile calculation for a mixture
PURESIGMAPRO- FILE	Sigma profile calculation for a pure component(s)
SIGMAPOTENTIAL	Sigma potential calculation for a mixture
PURESIGMAPO- TENTIAL	Sigma potential calculation for a pure component(s)

**get\_activity\_coefficients** (*idx*: *int* (<https://docs.python.org/3.8/library/functions.html#int>) | *List* (<https://docs.python.org/3.8/library/typing.html#typing.List>)[*int* (<https://docs.python.org/3.8/library/functions.html#int>)] | *None* (<https://docs.python.org/3.8/library/constants.html#None>) = *None*) → *List* (<https://docs.python.org/3.8/library/typing.html#typing.List>)[*ndarray*]

Returns activity coefficients for all `CRSMixture` or the specified ones.

#### Keyword Arguments

**idx** (*int* or *List*[*int*], optional) – Index or list of indices of `CRSMixture` to return. If *None*, returns for all mixtures. Default is *None*.

#### Returns

Activity coefficients as a list of numpy arrays.

#### Return type

*List*[*np.ndarray*])

**runCRSIteration** (*parameter\_sett*=*None*, *history*=*False*)

Run CRS iterations for all `CRSMixture` objects when `iteration=True`.

Each mixture executes `CRSIteration.do_iteration` with the base settings plus any overrides in `parameter_sett`. When `history` is `True`, each iteration is kept as a distinct `CRSJob`; otherwise only the last iteration is retained.

#### Keyword Arguments

- **parameter\_sett** (*Settings*, optional) – Settings to merge into the base CRS job settings for this iteration.

- **history** (`bool` (<https://docs.python.org/3.8/library/functions.html#bool>)) – Keep all iterations as `children["iteration_N"]` when `True`; keep only `children["iteration_last"]` when `False`.

**runCRSJob()**

Run all CRSJob in each CRSMixture

**class** `pyCRS.CRSManager.CRSMixture` (*\*\*kwargs*)

Build a CRSJob (or CRSIteration when enabled) from mixture input and settings.

This class builds a CRSJob from mixture composition/conditions, thermodynamic settings, and database inputs. When iteration is `True`, it creates a CRSIteration instead of a single CRSJob.

**crs\_job**

Created when iteration is `False`.

**Type**

`CRSJob`

**crs\_iteration**

Created when iteration is `True`.

**Type**

`CRSIteration`

**iteration**

Enable iterative runs by applying different settings across iterations (default `False`).

**Type**

`bool` (<https://docs.python.org/3.8/library/functions.html#bool>)

**mixture**

Composition as {Identifier : mole/mass fraction}

**Type**

`dict` (<https://docs.python.org/3.8/library/stdtypes.html#dict>)

**temperature**

Temp in Kelvin (default 298.15). If str, format can be “T1” or “T1 T2 ntemp”. Where T1 is the starting temperature, T2 is the ending temperature, and ntemp is the number of intervals between T1 and T2.

**Type**

`float` or `str`

**problem\_type**

Calculation type (default “activitycoef”).

**Type**

`str` (<https://docs.python.org/3.8/library/stdtypes.html#str>)

**database**

Path or instance of COSKFDatabase (default “my\_coskf\_db.db”).

**Type**

`str` or `COSKFDatabase`

**method**

Version of COSMORS or COSMOSAC (default “COSMORS”).

**Type**

`str` (<https://docs.python.org/3.8/library/stdtypes.html#str>)

**pressure**

Pressure in bar (default 1.01325). If str, format can be “P1” or “P1 P2 npress”. Where P1 is the starting pressure, P2 is the ending pressure, and npress is the number of intervals between P1 and P2.

**Type**

float or str

**jobname**

Job name in plam\_workdir (defaults “crsJob\_n”).

**Type**

str (<https://docs.python.org/3.8/library/stdtypes.html#str>)

**conformer**

Use multiple conformers in COSMO-RS (default False).

**Type**

bool (<https://docs.python.org/3.8/library/functions.html#bool>)

**massfraction**

Input fraction as mass (default False).

**Type**

bool (<https://docs.python.org/3.8/library/functions.html#bool>)

**density\_corr**

Use pure compound density for volume. Default to False (uses COSMO volume).

**Type**

bool (<https://docs.python.org/3.8/library/functions.html#bool>)

**vp\_corr**

Correct gas phase chemical potential with vapor pressure (default False).

**Type**

bool (<https://docs.python.org/3.8/library/functions.html#bool>)

**solute**

Solute state (“solid”, “gas”, “liquid”) in SOLUBILITY and PURESOLUBILITY. Default is “solid”.

**Type**

str (<https://docs.python.org/3.8/library/stdtypes.html#str>)

**iso**

Condition (“isotherm”, “isobar”, “flashpoint”) for BINMIXCOEF, TERNARYMIX, and COMPOSITION-LINE. Default is “isotherm”.

**Type**

str (<https://docs.python.org/3.8/library/stdtypes.html#str>)

**additional\_sett**

Optional custom settings (default None).

**Type**

Settings

**multi\_species**

Multi-species settings as {Identifier: Settings} (default None).

**Type**

dict (<https://docs.python.org/3.8/library/stdtypes.html#dict>)

## Reference

*Settings examples with COSMO-RS* (page 130)

Settings API

CRSJob API

CRSResults API

## PropPred

An interface to the PropPred program for the prediction of physical properties.

`pyCRS.PropPred.units`

a dictionary which maps property names to strings representing the units

### Type

`dict` (<https://docs.python.org/3.8/library/stdtypes.html#dict>)

`pyCRS.PropPred.available_properties`

a list of the property names available in PropPrediction

### Type

`list` (<https://docs.python.org/3.8/library/stdtypes.html#list>)

`PropPred.estimate` (*molecule*: `pyCRS_internal.Molecule` (page 127), *properties*: `str` (<https://docs.python.org/3.8/library/stdtypes.html#str>) | `list` (<https://docs.python.org/3.8/library/stdtypes.html#list>)[`str` (<https://docs.python.org/3.8/library/stdtypes.html#str>) = 'all', *temperatures*: `list` (<https://docs.python.org/3.8/library/stdtypes.html#list>)[`float` (<https://docs.python.org/3.8/library/functions.html#float>)] = [298.1499938964844], *show\_errors*: `bool` (<https://docs.python.org/3.8/library/functions.html#bool>) = `False`) → `None` (<https://docs.python.org/3.8/library/constants.html#None>)

Estimates one or more properties with PropPrediction. The results are written to the `pyCRS.Molecule` object provided to the function.

### Parameters

- **molecule** (`pyCRS.Molecule` (page 127)) – A `pyCRS.Molecule` object
- **properties** (`str` or `list(str)`) – a string naming a property for calculating a single property value or a list of strings naming all the desired properties to calculate
- **temperatures** (`list(float)`) – a list of temperature values with which to calculate the temperature-dependent properties
- **show\_errors** (`bool` (<https://docs.python.org/3.8/library/functions.html#bool>)) – indicate whether errors should be shown in std out

## Example

```

import pyCRS
mol = pyCRS.Input.read_smiles("c1ccccc1(OCC)")
# estimate all properties by default
pyCRS.PropPred.estimate(mol, temperatures=[290,295,300,305])

for prop, value in mol.properties.items():
    unit = pyCRS.PropPred.units[prop]
    print(f'{prop:<20s}: {value:.3f} {unit}')

for prop, value in mol.properties_tdep.items():
    print(f'{prop:<20s}:')
    unit = pyCRS.PropPred.units[prop]
    propunit = f'{prop} ({unit})'
    print("T (K)".rjust(30)+f'{propunit:>30s}')
    for t,v in value:
        print(f'{t:>30.3f}{v:>30.8g}')

```

```

boilingpoint      : 432.028 K
criticalpressure  : 34.205 bar
criticaltemp      : 569.012 K
criticalvol       : 0.376 L/mol
density           : 0.958 kg/L (298.15 K)
dielectricconstant : -3.824
entropygas        : 366.424 J/(mol K)
flashpoint        : 319.734 K
gidealgas         : 29.448 kJ/mol
hcombust          : -4173.723 kJ/mol
hformstd          : -165.241 kJ/mol
hfusion           : 10.622 kJ/mol
hidealgas         : -115.682 kJ/mol
hsublimation      : 66.720 kJ/mol
meltingpoint      : 202.182 K
molarvol          : 0.127 L/mol
parachor          : 305.809
solubilityparam   : 9.356  $\sqrt{(\text{cal}/\text{cm}^3)}$ 
synacc            : 1.042
tpt               : 201.439 K
vdwarea           : 161.566  $\text{\AA}^2$ 
vdwvol            : 122.256  $\text{\AA}^3$ 
liquidviscosity   :
                    T (K)          liquidviscosity (Pa-s)
                    290.000        0.0014622948
                    295.000        0.0013171559
                    300.000        0.0011921719
                    305.000        0.0010839762
vaporpressure     :
                    T (K)          vaporpressure (bar)
                    290.000        0.0010699197
                    295.000        0.0015221786
                    300.000        0.0021369822
                    305.000        0.0029624981

```

## FastSigma

This submodule provides an interface to the FastSigma program.

```
pyCRS.FastSigma.estimate (molecule: pyCRS\_internal.Molecule (page 127), method: str
    (https://docs.python.org/3.8/library/stdtypes.html#str) = 'COSMO-RS', model: str
    (https://docs.python.org/3.8/library/stdtypes.html#str) = 'FS1', display: bool
    (https://docs.python.org/3.8/library/functions.html#bool) = False) → None
    (https://docs.python.org/3.8/library/constants.html#None)
```

Uses the FastSigma program to estimate a sigma profile. The results are written to the `pyCRS.Molecule` object provided to the function.

### Parameters

- **molecule** (`pyCRS.Molecule` (page 127)) – the molecule to estimate
- **method** (`str` (<https://docs.python.org/3.8/library/stdtypes.html#str>)) – the COSMO-RS/-SAC method to use. Available options are (COSMO-RS, COSMOSAC2013, COSMOSAC2016). Not every method is available for every model.
- **model** (`str` (<https://docs.python.org/3.8/library/stdtypes.html#str>)) – the estimation technique for the sigma profile. Available options are (FS1, SG1). FS1 is a QSPR approach and SG1 uses a database of sigma profiles and matches the query molecule's substructures against substructures in that database.
- **display** (`bool` (<https://docs.python.org/3.8/library/functions.html#bool>)) – whether to display the results to standard out

### Example

```
import pyCRS
mol = pyCRS.Input.read_smiles("c1ccccc1(OCC)")
pyCRS.FastSigma.estimate(mol, method='COSMO-RS', model='SG1', display=True)
```

sigma value	Total profile	HB profile
...	...	...
-0.002	10.689	0.000
-0.001	9.453	0.000
0.000	9.216	0.000
0.001	10.264	0.000
0.002	12.356	0.000
0.003	12.007	0.000
0.004	13.050	0.000
0.005	11.818	0.000
0.006	7.250	0.000
0.007	2.643	0.000
0.008	1.218	0.101
0.009	1.131	0.890
0.010	1.248	1.185
0.011	1.209	1.181
0.012	1.251	1.239
0.013	1.233	1.228
0.014	0.422	0.420
0.015	0.025	0.024
...	...	...

(continues on next page)

(continued from previous page)

```

Molecular Mass =      122.0731649400 g/mol
  COSMO Area =      174.8609437511 Angstrom**2
  COSMO Volume =     165.2344104876 Angstrom**3
Gas Phase Bond Energy =    -4.1647328138 Hartree
  Bond Energy =    -4.1720958091 Hartree
  ...

```

## Input

This submodule provides functions used to initialize a `pyCRS.Molecule`.

`Input.read_smiles` (*smiles*: *str* (<https://docs.python.org/3.8/library/stdtypes.html#str>)) → *pyCRS\_internal.Molecule* (page 127)

### Parameters

**smiles** (string) – the SMILES string to be used

### Example

```

import pyCRS
mol = pyCRS.Input.read_smiles("c1ccccc1(OCC)")
mol = pyCRS.Input.read_smiles("c1ccccc1(OCC)fg1(OCC)")
print("Check if the molecule is valid:", mol.is_valid)

```

`Input.read_sdf` (*filename*: *str* (<https://docs.python.org/3.8/library/stdtypes.html#str>)) → *pyCRS\_internal.Molecule* (page 127)

### Parameters

**filename** (string) – the .sdf file containing the input molecule

### Example

```

import pyCRS
mol = pyCRS.Input.read_sdf("molecule.sdf")

```

## Output

This submodule provides functions to write output in kf format.

`pyCRS.Output.write_kf` (*molecule*: *pyCRS\_internal.Molecule* (page 127), *filename*: *str* (<https://docs.python.org/3.8/library/stdtypes.html#str>)) → `None` (<https://docs.python.org/3.8/library/constants.html#None>)

writes a molecule to a kf file with the name <filename>. The saved file can then be used directly with AMS COSMO-RS/-SAC.

### Parameters

- **molecule** (*pyCRS.Molecule* (page 127)) – the molecule to write to kf
- **filename** (*str* (<https://docs.python.org/3.8/library/stdtypes.html#str>)) – the filename for the output file. If the filename does not end with `.compkf`, then this extension is appended

## Example

```
import pyCRS
mol = pyCRS.Input.read_smiles("c1ccccc1(OCC) ")
pyCRS.FastSigma.estimate(mol)
pyCRS.Output.write_kf(mol, "example.compkf")
```

## Molecule

### class pyCRS.Molecule

This class stores information about a molecule and serves as the main interface for accessing estimated properties.

### Attributes

area	The COSMO surface area of the molecule
bond_energy	The Bond Energy (in Hartrees) for the molecule in the COSMO conductor phase.
dispersion	The dispersion energy
formula	The molecular formula
gas_phase_bond_energy	The Bond Energy (in Hartrees) for the molecule in the gas phase
is_valid	A bool that indicates whether the molecule is valid or if there were errors in processing it (e.g., invalid SMILES)
method	The method (COSMO-RS, COSMO-SAC, etc.) for which the estimate is made.
molar_mass	The molar mass in g/mol
nring	The nring parameter used in COSMO-RS
parameters	A dictionary of parameters that can be used for certain models (e.g., vapor pressure model parameters)
properties	A dictionary of property values for temperature-independent properties
properties_tdep	A dictionary of (temperature,property) pairs for temperature-dependent properties
smiles	The SMILES string of a molecule, if available
volume	The COSMO volume of the molecule

### Methods

**get\_sigma\_profile** (*self*: pyCRS\_internal.Molecule (page 127)) → dict  
 (<https://docs.python.org/3.8/library/stdtypes.html#dict>)[str  
 (<https://docs.python.org/3.8/library/stdtypes.html#str>), list  
 (<https://docs.python.org/3.8/library/stdtypes.html#list>)[float  
 (<https://docs.python.org/3.8/library/functions.html#float>)]

Returns a dictionary of sigma profiles values. For COSMO-RS, the dictionary has two entries (Total Profile and H-Bonding Profile). For COSMO-SAC, the dictionary has Total Profile, OH Profile, and OT Profile entries.

### Example

```
import pyCRS
mol = pyCRS.Input.read_smiles("CCCCCN")
pyCRS.FastSigma.estimate(mol, method='COSMO-RS', model = 'FS1', display=False)
for k,v in mol.get_sigma_profile().items():
    print(k, v)
```

```
H-Bonding Profile [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.051576871269907, 0.
↪093772696234738, ... , 0.36159288378918, 0.14796979638662, 0.0]
Total Profile [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.051576871269907, 0.
↪094440478268766, ... , 0.36159288378918, 0.147976661591423, 0.0]
```

**get\_tdep\_values** (*self*: [pyCRS\\_internal.Molecule](#) (page 127), *arg0*: *str* (<https://docs.python.org/3.8/library/stdtypes.html#str>)) → tuple (<https://docs.python.org/3.8/library/stdtypes.html#tuple>)[list (<https://docs.python.org/3.8/library/stdtypes.html#list>)[float (<https://docs.python.org/3.8/library/functions.html#float>)], list (<https://docs.python.org/3.8/library/stdtypes.html#list>)[float (<https://docs.python.org/3.8/library/functions.html#float>)]]]

Returns a tuple of (temperatures, values) for a temperature-dependent property given as an argument to the function.

### Example

```
import pyCRS
mol = pyCRS.Input.read_smiles("OCCCCC")
pyCRS.PropPred.estimate(mol, 'vaporpressure', temperatures=list(range(270,330,
↪10)))
unit = pyCRS.PropPred.units['vaporpressure']
print("Temperature (K)".rjust(15)+f'Vapor pressure ({unit})'.rjust(25) )
for temp, val in zip(*mol.get_tdep_values('vaporpressure')):
    print(f'{temp:>15.3f}{val:>25.3f}')
```

Temperature (K)	Vapor pressure (bar)
270.000	0.000
280.000	0.000
290.000	0.001
300.000	0.001
310.000	0.003
320.000	0.006

**has\_missing\_atoms** (*self*: [pyCRS\\_internal.Molecule](#) (page 127), *arg0*: *str* (<https://docs.python.org/3.8/library/stdtypes.html#str>)) → bool (<https://docs.python.org/3.8/library/functions.html#bool>)

**missing\_atoms** (*self*: [pyCRS\\_internal.Molecule](#) (page 127), *arg0*: *str* (<https://docs.python.org/3.8/library/stdtypes.html#str>)) → list (<https://docs.python.org/3.8/library/stdtypes.html#list>)[str (<https://docs.python.org/3.8/library/stdtypes.html#str>)]

A list of atom symbols which could not be mapped to a descriptor

## 10.2 Python scripting for COSMO-RS with PLAMS

### 10.2.1 General Information

**Attention:** Windows and Mac users may find it helpful to first read the *Getting Started* guides for scripting: [scripting with Windows](#) | [scripting with MacOS](#)

Python and the [PLAMS library](#) can be used for scripting with COSMO-RS. Due to the speed of COSMO-RS calculations, these jobs can be run interactively from the python interpreter. Larger numbers of jobs or high-throughput calculations can also easily be automated with python scripting. All results are returned as a python object, meaning the properties calculated with COSMO-RS can immediately be post-processed or used directly in other python functions.

**Note:** COSMO-RS calculations require the COSMO result file, either a `.coskf` or `.compkf` file, for each compound in the system. These files only need to be calculated once and then can be stored in a database for reuse in any further calculation involving the same compound.

### 10.2.2 Generating COSMO result file

The generation of a `.coskf` file involves calculating the COSMO surface with ADF, which requires a relatively expensive DFT calculation. In contrast, a `.compkf` file provide an estimated sigma profile based on the group contribution method. However, the use of a `.coskf` file is generally recommended for better accuracy.

Before performing a COSMO-RS calculation, ensure the COSMO result files are available for all species involved. If any required files are missing, they can be generated via the GUI or using the following tools:

- **.coskf** : Use the PLAMS recipe, `adfcosmorscompound` or the command-line tool `amsprep` (page 214)
- **.coskf for conformers** : Use the PLAMS recipe, `adfcosmorsconformers`
- **.compkf** : Generating using the command-line tool `fastsigma` (page 214)

### 10.2.3 Executing the code from the command line

It is recommended to use the version of python that is shipped with AMS. This version ensures that all the necessary libraries (e.g., [PLAMS](#)) are properly imported and are mutually compatible. The best way to do this is to run the `am-python` program. That can be executed from the command line as follows:

```
$AMSBIN/ampython <your_program.py>
```

where `<your_program.py>` should of course be replaced by the name of your program.

## 10.2.4 Specifying a problem type

To run COSMO-RS, the user must first provide a problem type for the calculation. This can be done by first creating a `Settings` object and then specifying the `.input.property._h` attribute. For example, to set up an activity coefficient calculation, we do the following:

```
from scm.plams import Settings, init, finish, CRSJob
import os

settings = Settings()
settings.input.property._h = 'ACTIVITYCOEF'
```

For other problem types, the `.input.property._h` attribute must be set to other values. The other options for this value are summarized below:

<code>._h</code> value	Problem type
ACTIVITYCOEF	Activity Coefficient
BINMIXCOEF	Binary mixture LLE/VLE
TERNARYMIX	Ternary mixture LLE/VLE
COMPOSITIONLINE	Solvent composition line interpolation
SOLUBILITY	Solubility calculation in a mixed solvent
PURESOLUBILITY	Solubility calculation in a pure solvent
LOGP	Partition coefficient calculation
VAPORPRESSURE	Vapor pressure calculation for a mixed solvent
PUREVAPORPRES- SURE	Vapor pressure calculation for a pure solvent
BOILINGPOINT	Boiling point calculation for a mixture
PUREBOILINGPOINT	Boiling point calculation for a pure solvent(s)
FLASHPOINT	Flashpoint calculation for a mixture
SIGMAPROFILE	Sigma profile calculation for a mixture
PURESIGMAPROFILE	Sigma profile calculation for a pure component(s)
SIGMAPOTENTIAL	Sigma potential calculation for a mixture
PURESIGMAPOTEN- TIAL	Sigma potential calculation for a pure component(s)

## 10.2.5 Inputting Compounds

In PLAMS, each compound is also input as a `Settings` object. Additional information about the compounds required for the calculation (e.g., mole fraction) can be specified as an attribute of the compound's `Settings` object. Ensure the *ADFCRS-2018 Database* (page 24) is installed beforehand. An example for a calculation with two compounds is given below.

```
##### Note: Ensure to configure the database path to either the installed ADFCRS-
->2018 directory or your own specified directory #####

database_path = os.path.join(os.environ["SCM_PKG_ADFCRSDIR"], "ADFCRS-2018")
if not os.path.exists(database_path):
    raise OSError(f"The provided path does not exist. Exiting.")

# set the number of compounds
num_compounds = 2
```

(continues on next page)

(continued from previous page)

```
compounds = [Settings() for i in range(num_compounds)]
compounds[0]._h = os.path.join(database_path, "Water.coskf")
compounds[1]._h = os.path.join(database_path, "1-Hexanol.coskf")
```

### User-provided compound files

In addition to the default `.coskf` files, user-provided compound files can also be used.

To use a `.compkf` file generated with *FASTSIGMA* (page 27)

```
compounds[1]._h = os.path.join(compkf_path, "1-Hexanol.compkf")
compounds[1].compkffile = True
```

To use a `.cosmo` file:

```
compounds[1]._h = os.path.join(cosmo_path, "1-Hexanol.cosmo")
compounds[1].cosmofile = True
```

---

**Note:** For the best accuracy, we recommend using an ADF-generated `.coskf` file whenever possible.

---

## 10.2.6 Specifying mole fractions, temperatures, and pressures

Mole fractions are attributes of the compound `Settings` object. There are two types of mole fractions used in COSMO-RS. `frac1` is for standard specification of mole fractions in most problem types. `frac2` is used when the problem type requires two distinct liquid phases (COMPOSITIONLINE or LOGP). Additionally, the temperature can be specified using the input `.temperature` attribute of the `Settings` object. An example of this is shown below:

```
#set compound mole fractions
compounds[0].frac1 = 0.3
compounds[1].frac1 = 0.7

#set temperature (range)
#to specify a range, use 3 numbers: (1) the lowest temperature,
#(2) the highest temperature, and (3) the steps taken between these temperatures
settings.input.temperature = "298.15"
```

To specify a temperature range, set the input `.temperature` object equal to a python `str` which contains the lower temperature, upper temperature, and number of steps taken between the temperatures. These values should simply be separated by spaces. For example, to specify that a calculation should go over the temperature range 298.15K to 398.15K with 10 temperature steps, do the following:

Note: Temperature range is only used for calculation of vapor pressure and solubility.

```
settings.input.temperature = "298.15 398.15 10"
```

Pressure works in much the same way. To input the system pressure (in bar), do the following:

```
settings.input.pressure = "1.5"
```

Note: Pressure is only used in calculation of boiling temperature, gas solubility, isobar binary mixture and isobar ternary mixture.

## 10.2.7 Running jobs

To run a job with COSMO-RS, first assign the `input.compound` attribute to the list of compound Settings objects used previously. Then, simply create the job using `CRSJob(settings=<your previously defined Settings object>)`. Once a job is created, you can run it with the `.run()` function. An example of this is given below:

```
# specify the compounds as the compounds to be used in the calculation
settings.input.compound = compounds
# create a job that can be run by COSMO-RS
my_job = CRSJob(settings=settings)
# run the job
init()
out = my_job.run()
finish()
```

## 10.2.8 Reading the results of a job

Once a job has finished running, we can access the results directly in python. First, we can check to see which properties are available. We can do this using the `get_prop_names()` function on the output. For example, adding the line:

```
# check for the available properties
print( "Available properties:", out.get_prop_names() )
```

gives us the available properties as a python set for our calculation type (“ACTIVITYCOEF” in this case). The result of the print statement is the following:

```
Available properties: {'henrycnodim', 'property', 'deltag', 'henryc', 'nitems', 'gamma
→',
'ncomp', 'filename', 'temperature', 'frac1', 'G solute', 'mu gas', 'molmass', 'E gas',
'mu', 'usepolyunits', 'mu pure', 'method'}
```

We can also convert all of the calculation results to a python dict using the `get_results()` function. For example, to collect all of the results and then print the activity coefficient values (“gamma”), we write the following code:

```
# convert all the results into a python dict
res = out.get_results()
print( "Activity coef values:\n", res["gamma"] )
```

This results in the following program output:

```
Activity coef values:
[[ 3.71486   ]
 [ 1.04484607]]
```

Here the two activity coefficient values are returned as elements in a `numpy.ndarray`. Properties with multiple values are always stored as a numpy array.

**Note:** For properties with multiple values, the dictionary values are stored as a `numpy.ndarray`. If applicable to the calculation, the rows of the array represent different compounds and the columns represent different steps of the calculation (e.g., different temperatures/pressures or different mole fractions for a binary/ternary mixture calculation).

Putting all the previous code together, we have the following working example for calculating activity coefficients for 2 components:

```
from scm.plams import Settings, init, finish, CRSJob
import os

##### Note: Ensure to configure the database path to either the installed ADFCRS-
↳2018 directory or your own specified directory #####

database_path = os.path.join(os.environ["SCM_PKG_ADFCRSDIR"], "ADFCRS-2018")
if not os.path.exists(database_path):
    raise OSError(f"The provided path does not exist. Exiting.")

# initialize settings object
settings = Settings()
settings.input.property._h = "ACTIVITYCOEF"

# set the number of compounds
num_compounds = 2

compounds = [Settings() for i in range(num_compounds)]
compounds[0]._h = os.path.join(database_path, "Water.coskf")
compounds[1]._h = os.path.join(database_path, "1-Hexanol.coskf")

# set compound mole fractions
compounds[0].frac1 = 0.3
compounds[1].frac1 = 0.7

# set temperature (range)
# to specify a range, use 3 numbers: (1) the lowest temperature,
# (2) the highest temperature, and (3) the steps taken between these temperatures
settings.input.temperature = "298.15"

# specify the compounds as the compounds to be used in the calculation
settings.input.compound = compounds
# create a job that can be run by COSMO-RS
my_job = CRSJob(settings=settings)
# run the job
init()
out = my_job.run()
finish()

# check for the available properties
print("Available properties:", out.get_prop_names())

# convert all the results into a python dict
res = out.get_results()
print("Activity coef values:\n", res["gamma"])
```

## 10.2.9 Plotting results

2D graphs can also be generated to visualize the results with the `plot` function. The `plot` function takes as a first argument any (or multiple) of the following:

- a `numpy.ndarray` object. This can be passed to the function as a dictionary value after calling the `get_results()` function.
- the name of a property. This property is read from the results and plotted. For a list of available properties, use the `get_prop_names()` function.

Additionally, the `plot` function takes the following keyword arguments:

- `x_axis`. This can be the name of a property or a `numpy.ndarray` object. This represents the independent variable in the plot. This value must be one dimensional, meaning it cannot be indexed over both compounds and temperatures.
- `x_label`. This can be used to label the x axis in the plot.
- `y_label`. This can be used to label the y axis in the plot.
- `plot_fig`. This is set to `True/False` to indicate whether a plotted figure should be displayed. The default is `True`.

The results of `plot` are returned as a `matplotlib.pyplot` object and can be further modified.

To demonstrate the use of `plot`, we do an example in which we calculate the solubility of methane gas in 1-Octanol and Ethanol across the temperature range from 298.15K to 398.15K. We also include the vapor pressure of methane using the VPM1 model. The code is shown below:

```
from scm.plams import Settings, init, finish, CRSJob
import os

##### Note: Ensure to configure the database path to either the installed ADFCRS-
->2018 directory or your own specified directory #####

database_path = os.path.join(os.environ["SCM_PKG_ADFCRSDIR"], "ADFCRS-2018")
if not os.path.exists(database_path):
    raise OSError(f"The provided path does not exist. Exiting.")

# initialize settings object
settings = Settings()
settings.input.property._h = "PURESOLUBILITY"

# this indicates we're calculating gas solubility
settings.input.property.isobar = ""

# set the number of compounds
num_compounds = 3

compounds = [Settings() for i in range(num_compounds)]
compounds[0]._h = os.path.join(database_path, "Methane.coskf")
compounds[1]._h = os.path.join(database_path, "1-Octanol.coskf")
compounds[2]._h = os.path.join(database_path, "Ethanol.coskf")

# set compound mole fractions
# for pure solubility the solvent gets a mole fraction of 1
# and the solute does not have the frac1 attribute
compounds[1].frac1 = 1
compounds[2].frac1 = 1
```

(continues on next page)

(continued from previous page)

```

# specify the vapor pressure equation for methane
compounds[0].vp_equation = "VPM1"
compounds[0].vp_params = "-1039.67755001 -0.183945615995 0.00061368649128 10.
↪1113503603315 0.0"

# set temperature (range)
# to specify a range, use 3 numbers: (1) the lowest temperature,
# (2) the highest temperature, and (3) the steps taken between these temperatures
settings.input.temperature = "298.15 398.15 10"

# 1 atm = 1.01325 bar
settings.input.pressure = "1.01325"

# specify the compounds as the compounds to be used in the calculation
settings.input.compound = compounds
# create a job that can be run by COSMO-RS
my_job = CRSJob(settings=settings)
# run the job
init()
out = my_job.run()
finish()

# convert all the results into a python dict
res = out.get_results()

# plot the solubilities in g/L solution
# the [1:] indicates that we're not plotting the values for methane (these are
↪automatically set to 0)
plt = out.plot(
    res["solubility g_per_L_solution"][1:],
    x_axis="temperature",
    x_label="Temperature",
    y_label="solubility g/L solution",
)
# plt.savefig("./PLAMS_gas_solubility.png")

```

This code generates the following plot:

## 10.3 Examples for pyCRS/PLAMS

### 10.3.1 pyCRS : Basic usage for Database and CRSManager

#### Adding compound to a pyCRS database

First, we'll add ethanol and water to the database and search for a list of compounds. When adding a compound, along with the COSKF file, it is necessary to provide a compound name and an unique chemical identifier. If only the COSKF file is provided, the method will attempt to utilize the IUPAC as the compound name and the CAS number as the chemical identifier, extracted from the Compound Data section within the COSKF file. In case such data is unavailable in the COSKF file, the user can specify the name parameter to define the compound name and either cas or identifier as the chemical identifier.

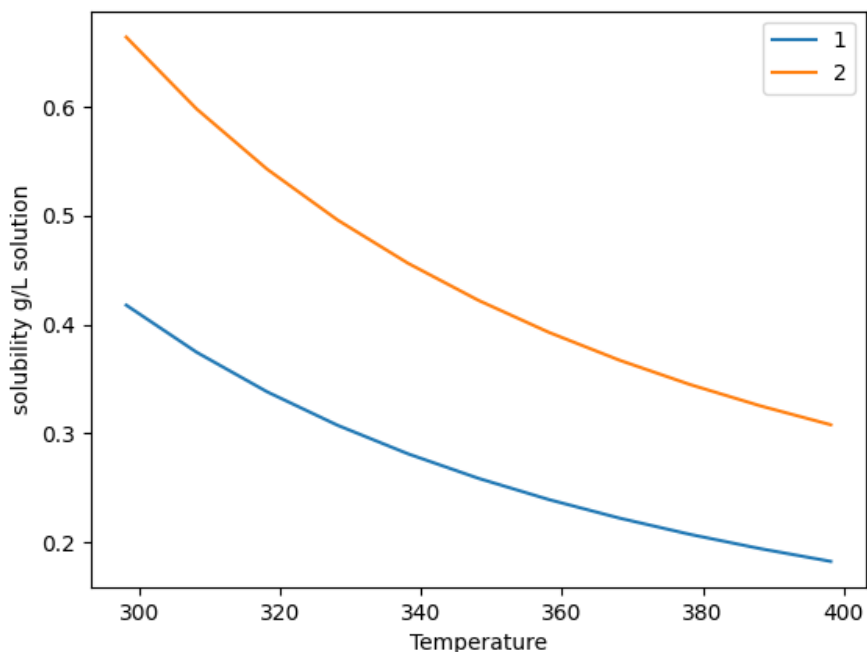


Fig. 10.1: The output of the `plot` function for a gas solubility calculation.

```

from pyCRS.Database import COSKFDatabase

##### Note: Please ensure to install the ADFCRS-2018 database via amspackages_
↳ (SCM->Packages) before running the below script #####

db = COSKFDatabase("my_coskf_db.db")
db.add_compound("1-Hexanol.coskf", cas="111-27-3")
db.add_compound("Water.coskf", identifier="InChI=1S/H2O/h1H2")

compounds = db.get_compounds(["111-27-3", "Water"])
for key, rows in compounds.items():
    for row in rows:
        print(f"{key}: {row}")

```

The output produced is the following

```

Successfully add new COMPOUND :compound_id=1 conformer_id=1 name=1-hexanol
Successfully add new COMPOUND :compound_id=2 conformer_id=2 name=water
111-27-3: CompoundRow(compound_id=1, conformer_id=1, name='1-hexanol', cas='111-27-3',
↳ identifier=None, smiles='CCCCCCO', resolved_smiles='CCCCCCO', coskf='1-Hexanol.
↳ coskf', Egas=-2562.042, Ecosmo=-2567.322, nring=0)
Water: CompoundRow(compound_id=2, conformer_id=2, name='water', cas='7732-18-5',
↳ identifier='InChI=1S/H2O/h1H2', smiles='O', resolved_smiles='O', coskf='Water.coskf
↳ ', Egas=-323.939, Ecosmo=-330.389, nring=0)

```

### Activity coefficient calculation

```

from scm.plams import Settings, init, finish
from pyCRS.Database import COSKFDatabase
from pyCRS.CRSManger import CRSSystem

db = COSKFDatabase("my_coskf_db.db")
cal = CRSSystem()

mixture = {}
mixture["7732-18-5"] = 0.3
mixture["111-27-3"] = 0.7

cal.add_Mixture(mixture, database="my_coskf_db.db", temperature=298.15, problem_type=
↳"activitycoef")

init()
cal.runCRSJob()
finish()

for out in cal.outputs:
    print(f"Property = {out.section}")
    res = out.get_results()
    for name, x, gamma in zip(res["name"], res["frac1"], res["gamma"]):
        print(name, x, gamma)

```

The output produced is the following

```

Property = ACTIVITYCOEF
water [0.3] [3.71486029]
1-hexanol [0.7] [1.04484603]

```

### Solid Solubility calculation

Next, we can add the melting point and heat of fusion if experimental data is available or estimate these values using PropPred and then perform the solid solubility calculation. In addition, it also shows the way to either use the pure compound density or solvent density in the calculation which will only affect the volume dependent properties.

```

from scm.plams import Settings, init, finish
from pyCRS.Database import COSKFDatabase
from pyCRS.CRSManger import CRSSystem

##### Note: Please ensure to install the ADFCRS-2018 database via amspackages_
↳(SCM->Packages) before running the below script #####

db = COSKFDatabase("my_coskf_db.db")

db.add_compound("Ibuprofen.coskf")
db.add_compound("Water.coskf")
db.add_physical_property("Ibuprofen", "hfusion", 27.94, unit="kJ/mol")
db.add_physical_property("Ibuprofen", "meltingpoint", 347.6)
db.estimate_physical_property("Ibuprofen")

EXP = db.get_physical_properties("Ibuprofen")[0]
QSPR = db.get_physical_properties("Ibuprofen", source="PropPred")[0]

```

(continues on next page)

(continued from previous page)

```

print("experimental value = ", EXP.hfusion, " estimated value = ", QSPR.hfusion)

mixture = {}
mixture["Water"] = 1.0
mixture["Ibuprofen"] = 0

init()

cal = CRSSystem()
cal.add_Mixture(
    mixture, database="my_coskf_db.db", temperature=293.15, problem_type="solubility",
    ↪ solute="solid", jobname="default"
)

db.add_physical_property("Water", "density", 0.997)
cal.add_Mixture(
    mixture,
    database="my_coskf_db.db",
    temperature=293.15,
    problem_type="solubility",
    solute="solid",
    density_corr=True,
    jobname="compound_density",
)

solvent_density = 0.9982067
additional_sett = Settings()
additional_sett.input.property.DensitySolvent = solvent_density
cal.add_Mixture(
    mixture,
    database="my_coskf_db.db",
    temperature=293.15,
    problem_type="solubility",
    solute="solid",
    additional_sett=additional_sett,
    jobname="solvent_density",
)

cal.runCRSJob()

for out in cal.outputs:
    res = out.get_results()
    print(f"Jobname = {out.job.name}; Property = {out.section}")
    print(
        f"{res['name'][1]} in {res['name'][0]} x= {res['molar fraction'][1][0]:.3e}; ↪
    ↪ (g/L)= {res['solubility g_per_L_solvent'][1][0]:.5f}"
    )

finish()

```

The output produced is the following

```

experimental value = 6.678 estimated value = 5.744
Jobname = default; Property = SOLUBILITY
ibuprofen in water x= 1.647e-06; (g/L)= 0.02207
Jobname = compound_density; Property = SOLUBILITY
ibuprofen in water x= 1.647e-06; (g/L)= 0.01880

```

(continues on next page)

(continued from previous page)

```
Jobname = solvent_density; Property = SOLUBILITY
ibuprofen in water x= 1.647e-06; (g/L)= 0.01882
```

## Establishment of a pyCRS database from the ADFCRS-2018 database

The script facilitates the automatic creation of a sql database utilizing the COSKF file from the *ADFCRS-2018 database* (page 24).

```
from pyCRS.Database import COSKFDatabase
from pathlib import Path

##### Note: Please ensure to install the ADFCRS-2018 database via amspackages_
↳ (SCM->Packages) before running the below script #####

db = COSKFDatabase("ADFCRS-2018.db")
path = Path(db.ADFCRS2018_path)

for x in path.glob("*.coskf"):
    if not x.name.startswith("IL_"):
        db.add_compound(x)
```

### 10.3.2 pyCRS : Conformer usage for Database and CRSManager

#### Generating multiple conformers for a compound

Different conformers of a molecule can have significantly different sigma profiles, which can lead to big differences in predicted properties with COSMO-RS. For this reason, it's important for COSMO-RS calculations to use geometries corresponding to the lowest-energy conformer or a set of low-energy conformers when it's possible several conformers may exist in significant amounts. The script shows how to use the *adfcosmorsconformers* recipe based on the *ConformerTools* functionality in AMS to generate a set of low-energy conformers, refine the geometries with a semi-empirical method and finally perform the ADF and COSMO calculations necessary to produce *.coskf* files.

```
from scm.plams import Settings, init, finish, from_smiles
from scm.plams.recipes.adfcosmorsconformers import ADFCOSMORSConfJob,
↳ADFCOSMORSConfFilter
from scm.conformers import ConformersJob

init()

mol = from_smiles("CC(=O)O")

conf_sett = Settings()
conf_sett.input.AMS.Generator.RDKit
conf_sett.input.AMS.Generator.RDKit.InitialNConformers = 50
conf_job = ConformersJob(name="conformers_uff", molecule=mol, settings=conf_sett)

dftb_sett = Settings()
dftb_sett.input.AMS.Task = "Optimize"
dftb_sett.input.DFTB

# ADFCOSMORSConfFilter(max number of conformers, max energy range)
fil1 = ADFCOSMORSConfFilter(20, 22) # applied to UFF
```

(continues on next page)

(continued from previous page)

```

fil2 = ADFCOSMORSConfFilter(10, 12) # applied to DFTB
fil3 = ADFCOSMORSConfFilter(5, 7) # applied to ADF gas phase

mol_info = {"CAS": "64-19-7", "IUPAC": "Acetic acid", "SMILES": "CC(=O)O"}

job = ADFCOSMORSConfJob(
    mol,
    conf_gen=conf_job,
    first_filter=fil1,
    additional=[(dftb_sett, fil2)],
    final_filter=fil3,
    coskf_name="acetic_acid",
    coskf_dir="coskf_acetic_acid",
    mol_info=mol_info,
)
job.run()

finish()

```

### Adding conformers to the database

This script shows how to add multiple conformers into the database, retrieve the `ConformerRow` and visualize conformers for a given compound. Please generate the conformers for acetic acid using above script or download the files from the below link before running the below script.

Download relevant coskf file

```

from pyCRS.Database import COSKFDatabase
import os

##### Note: Ensure to download the coskf_acetic_acid or generating_
↳ conformers for acetic acid before running the script #####

conformer_path = os.path.join(os.getcwd(), "coskf_acetic_acid")

db = COSKFDatabase("my_coskf_db.db")

db.add_compound("acetic_acid_0.coskf", coskf_path=conformer_path)
db.add_compound("acetic_acid_1.coskf", coskf_path=conformer_path)

print("Displaying all ConformerRow of acetic acid")
for key, rows in db.get_conformers("Acetic acid").items():
    for row in rows:
        print(f" {row}")

print("Displaying the CompoundRow of acetic acid with the lowest energy structure_
↳ (default)")
for key, rows in db.get_compounds("Acetic acid").items():
    for row in rows:
        print(f" {row}")

db.visualize_conformers(identifier="acetic acid")

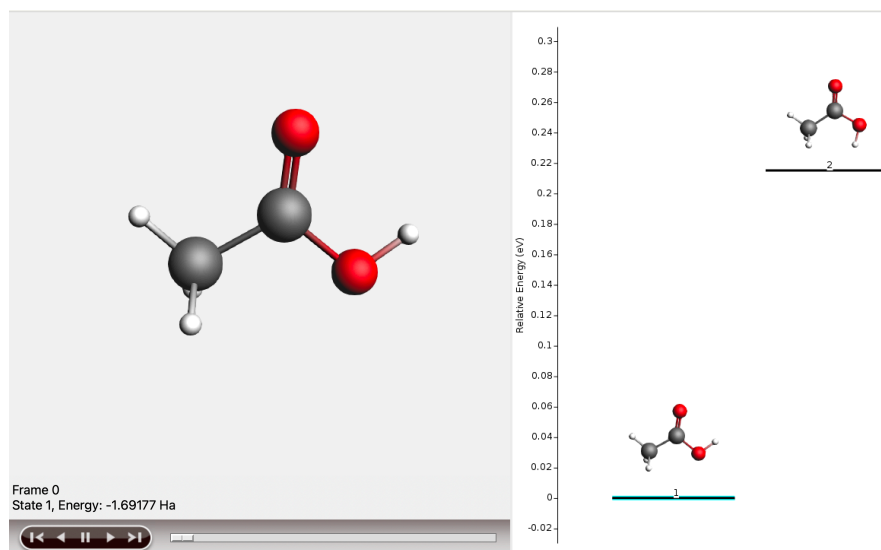
```

The output produced is the following

```

Displaying all ConformerRow of acetic acid
ConformerRow(conformer_id=16, compound_id=16, name='acetic acid', cas='64-19-7',
↪identifier=None, smiles='CC(=O)O', resolved_smiles='CC(=O)O', coskf='acetic_acid_0.
↪coskf', Egas=-1061.604, Ecosmo=-1068.355, nring=0)
ConformerRow(conformer_id=17, compound_id=16, name='acetic acid', cas='64-19-7',
↪identifier=None, smiles='CC(=O)O', resolved_smiles='CC(=O)O', coskf='acetic_acid_1.
↪coskf', Egas=-1056.642, Ecosmo=-1066.721, nring=0)
Displaying the CompoundRow of acetic acid with the lowest energy structure (default)
CompoundRow(compound_id=16, conformer_id=16, name='acetic acid', cas='64-19-7',
↪identifier=None, smiles='CC(=O)O', resolved_smiles='CC(=O)O', coskf='acetic_acid_0.
↪coskf', Egas=-1061.604, Ecosmo=-1068.355, nring=0)
name=acetic acid; cas=64-19-7; identifier=None
conformer_id=16, coskf=acetic_acid_0.coskf, Egas=-1061.604, Ecosmo=-1068.355
conformer_id=17, coskf=acetic_acid_1.coskf, Egas=-1056.642, Ecosmo=-1066.721

```



### Activity coefficient calculation considering conformers

This calculation models acetic acid as a mixture of conformers and plots activity coefficients and the conformer distribution over the mole fraction range. Please generate the conformers for acetic acid using above script or download the files from the below link before running the below script.

Download relevant coskf file

```

from scm.plams import Settings, init, finish
from pyCRS.Database import COSKFDatabase
from pyCRS.CRSManger import CRSSystem
import matplotlib.pyplot as plt
import numpy as np
import os

##### Note: Please ensure to install the ADFCRS-2018 database via amspackages_
↪(SCM->Packages) before running the below script #####

db = COSKFDatabase("my_coskf_db.db")

db.add_compound("Water.coskf")

```

(continues on next page)

(continued from previous page)

```

cal = CRSSystem()

mixture = {}
x_range = np.linspace(0, 1, 11)
for x in x_range:
    mixture["Acetic acid"] = x
    mixture["Water"] = 1 - x
    cal.add_Mixture(mixture, database="my_coskf_db.db", temperature=298.15, problem_
    ↪type="activitycoef", conformer=True)

init()
cal.runCRSJob()
finish()

gamma_water = []
gamma_acid = []
comp_acetic = {}
for out in cal.outputs:
    res = out.get_results()
    gamma_acid.append(res["gamma"][0])
    gamma_water.append(res["gamma"][1])
    if comp_acetic == {}:
        comp_acetic = out.get_multispecies_dist()[0]
    else:
        tmp_comp_acetic = out.get_multispecies_dist()[0]
        for key in comp_acetic:
            comp_acetic[key].extend(tmp_comp_acetic[key])

fig, axs = plt.subplots(2)
axs[0].plot(x_range, gamma_acid, label="$\gamma_1$ (acetic)")
axs[0].plot(x_range, gamma_water, label="$\gamma_2$ (water)")

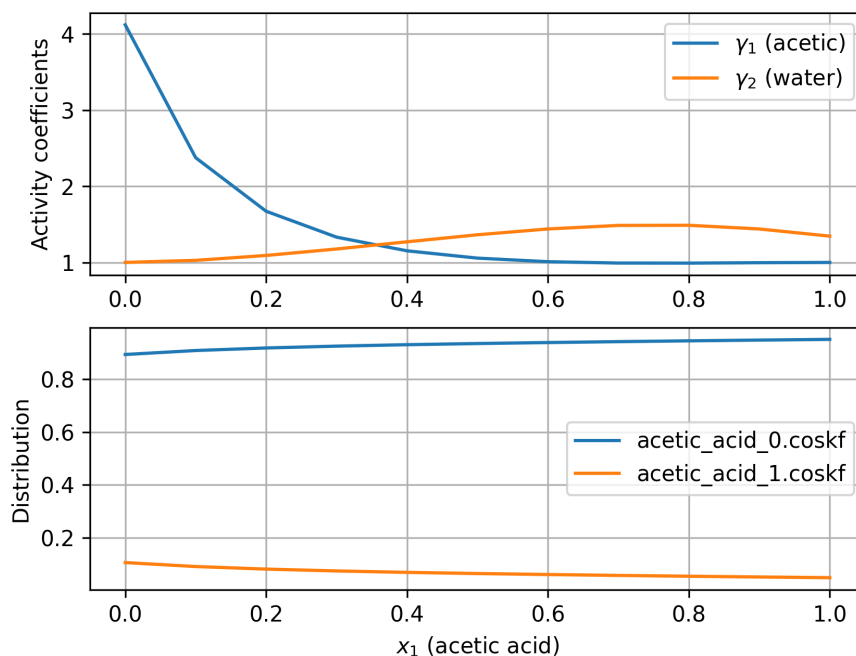
for struct, val in comp_acetic.items():
    axs[1].plot(x_range, val, label=os.path.basename(struct))

plt.setp(axs[0], ylabel="Activity coefficients")
plt.setp(axs[1], ylabel="Distribution")
for i in range(2):
    axs[i].legend()
    axs[i].grid()

plt.xlabel("$x_1$ (acetic acid)")
# plt.savefig('./pyCRS_activitycoef_conformer', dpi=300)
plt.show()

```

This code produces the following figure which plots activity coefficients and the distribution of two conformers.



### 10.3.3 pyCRS : Basic usage for PropPred and FastSigma

#### Basic usage

This basic example provides a minimal script input a molecule (paracetamol) as a smiles string, calculate a single property, and then estimate the default (COSMO-RS) sigma profile.

```
import pyCRS
import numpy as np
import matplotlib.pyplot as plt

mol = pyCRS.Input.read_smiles("CC(=O)Nc1ccc(O)cc1") # paracetamol

print("available properties:", pyCRS.PropPred.available_properties)

pyCRS.PropPred.estimate(mol, "hfusion")
print("hfusion value:", mol.properties["hfusion"], pyCRS.PropPred.units["hfusion"])

pyCRS.FastSigma.estimate(mol, method="COSMO-RS", display=False)

sigma_profiles = mol.get_sigma_profile()
print("Total sigma profile:")
print(sigma_profiles["Total Profile"])
print("H-Bonding:")
print(sigma_profiles["H-Bonding Profile"])

chgden = np.linspace(-0.025, 0.025, 51)
plt.plot(chgden, sigma_profiles["Total Profile"], label="total profile")
plt.plot(chgden, sigma_profiles["H-Bonding Profile"], "--", label="hb profile")
plt.xlabel("σ (e/Å**2)")
```

(continues on next page)

(continued from previous page)

```
plt.ylabel("p( $\sigma$ )")
plt.legend()
plt.show()
```

The output produced is the following:

```
available properties: ['acentricfactor', 'autoignitiontemp', 'boilingpoint',
↪ 'critcompress', 'criticalpressure', 'criticaltemp', 'criticalvol', 'density',
↪ 'dielectricconstant', 'dipolemoment', 'entropygas', 'entropystd', 'flashpoint',
↪ 'gformstd', 'gidealgas', 'hcombust', 'hformstd', 'hfusion', 'hidealgas',
↪ 'hsublimation', 'liquidviscosity', 'lowflamlimper', 'meltingpoint', 'molarvol',
↪ 'parachor', 'radgyration', 'refractiveindex', 'solubilityparam', 'synacc', 'tpp',
↪ 'tpt', 'upflamlimper', 'vaporpressure', 'vdwarea', 'vdwvol']
hfusion value: 28.084352493286133 kJ/mol
Total sigma profile:
[0.0, 0.0, 0.0, 0.002353191375733, 0.050697326660157, 0.24713134765625, 0.
↪ 5523872375488279, 0.840805053710938, 1.301651000976562, 1.316818237304688, 1.
↪ 408416748046875, 1.173583984375, 1.027912139892578, 1.160888671875, 0.
↪ 979301452636719, 0.8482666015625, 0.5888519287109371, 5.276319718325397, 11.
↪ 539728505193898, 13.300330108724362, 12.906459205297642, 10.834775504652777, 8.
↪ 539289410607651, 7.653812772468701, 7.964459592741903, 6.8641550757498555, 9.
↪ 304299127480718, 7.66864582217209, 11.452185796196918, 12.639082653211547, 13.
↪ 748599090727417, 11.745518829550344, 3.4796992518586065, 3.1053283341103315, 2.
↪ 864667892456055, 3.6309814453125, 4.3218994140625, 3.9415283203125, 2.786376953125, ↪
↪ 0.968017578125, 0.48760986328125, 0.319122314453125, 0.047515869140625, 0.0, 0.0, 0.
↪ 0, 0.0, 0.0, 0.0, 0.0, 0.0]
H-Bonding:
[0.0, 0.0, 0.0, 0.002353191375733, 0.050697326660157, 0.246719360351562, 0.
↪ 552310943603516, 0.840805053710938, 1.301651000976562, 1.313278198242188, 1.
↪ 3948974609375, 1.173583984375, 1.027912139892578, 1.15216064453125, 0.
↪ 979301452636719, 0.8482666015625, 0.5888519287109371, 0.065513610839843, 0.0, 0.0, ↪
↪ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.277877807617188, ↪
↪ 2.451416015625, 3.6309814453125, 4.3218994140625, 3.9354248046875, 2.78125, 0.
↪ 968017578125, 0.48760986328125, 0.319122314453125, 0.047515869140625, 0.0, 0.0, 0.0,
↪ 0.0, 0.0, 0.0, 0.0, 0.0]
```

## Temperature-dependent properties

This example calculates the vapor pressure and produces a plot of vapor pressure against temperature.

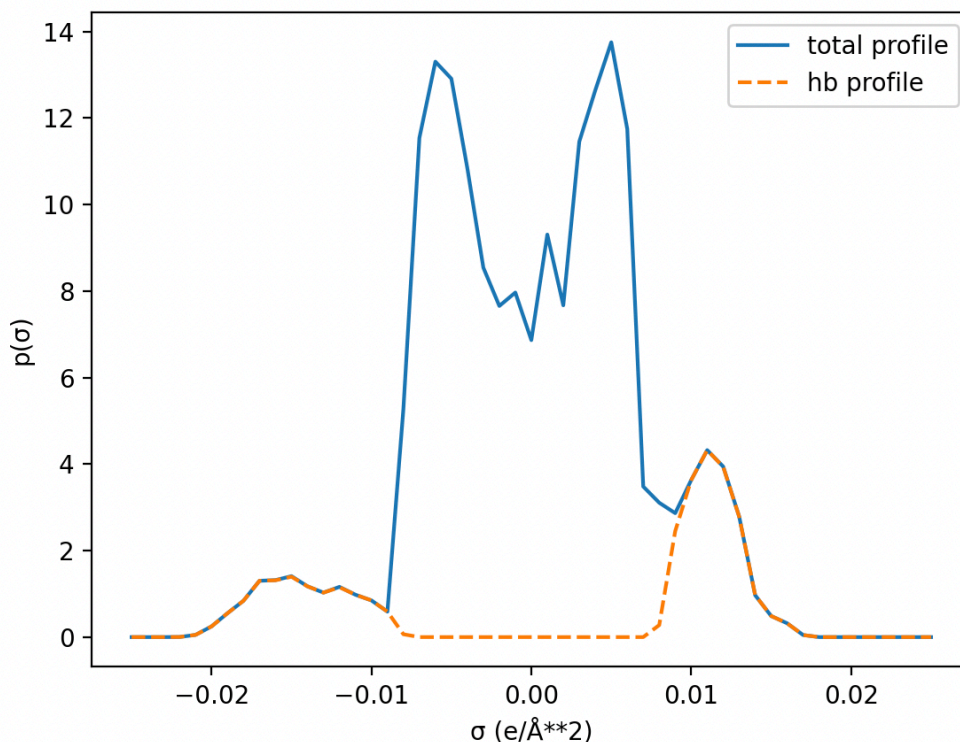
```
import pyCRS
import matplotlib.pyplot as plt

mol = pyCRS.Input.read_smiles("CCCCCO")

prop_name = "vaporpressure"
pyCRS.PropPred.estimate(mol, temperatures=[290, 295, 300, 305, 310, 315])
print("Results:", mol.properties_tdep[prop_name])

x, y = mol.get_tdep_values(prop_name)
unit = pyCRS.PropPred.units[prop_name]
plt.plot(x, y, "-o")
plt.ylabel(f"vapor pressure ({unit})")
plt.xlabel("Temperature (K)")
```

(continues on next page)



(continued from previous page)

```
# plt.savefig('./pyCRS_PropPred_Tdep.png')
plt.show()
```

The output shows the format of the results: (temperature, vapor pressure) pairs

```
Results: [(290.0, 0.0006316340979498092), (295.0, 0.0009549864170162676), (300.0, 0.
→ 0014201952225525484), (305.0, 0.002079200184963613), (310.0, 0.0029991100667307634),
→ (315.0, 0.004265490872465904)]
```

Finally, the plot produced is the following:

### Estimating multiple properties

Estimating multiple properties is as simple as supplying a list of property names to the PropPred interface. All properties are estimated by default if no property argument is supplied. In this example, we first estimate a few properties, and then estimate all properties.

```
import pyCRS

def print_props(mol):
    for prop, value in mol.properties.items():
        unit = pyCRS.PropPred.units[prop]
        print(f"{prop:<20s}: {value:.3f} {unit}")

    for prop, value in mol.properties_tdep.items():
        print(f"{prop:<20s}:")
```

(continues on next page)

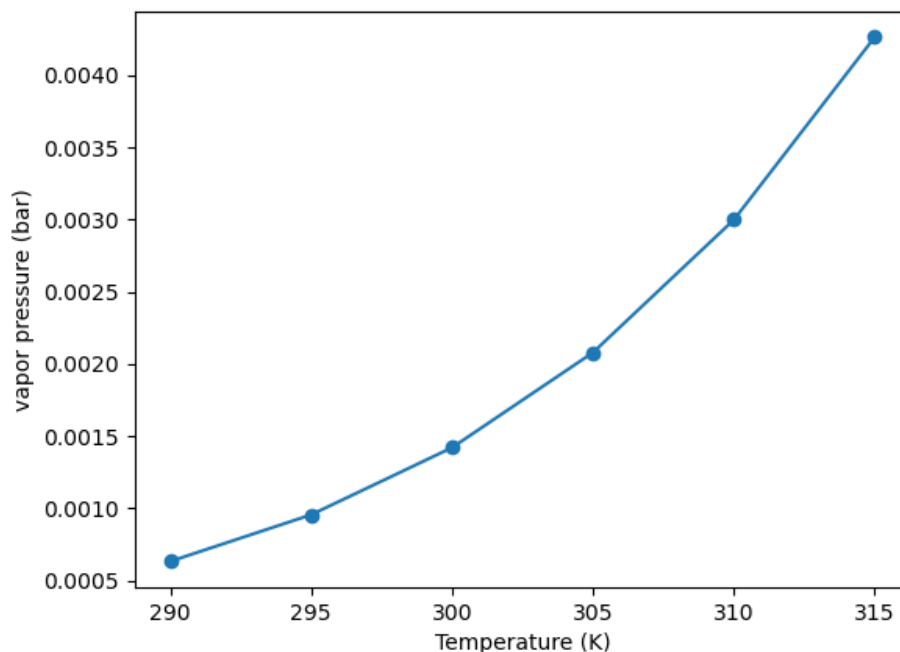


Fig. 10.2: The estimated vapor pressure versus temperature for 1-Hexanol

(continued from previous page)

```

unit = pyCRS.PropPred.units[prop]
propunit = f"{prop} ({unit})"
print("T (K)".rjust(30) + f"{propunit:>30s}")

for t, v in value:
    print(f"{t:>30.3f}{v:>30.8g}")

mol = pyCRS.Input.read_smiles("CCCCCCO")

props = ["meltingpoint", "boilingpoint", "density", "flashpoint", "vaporpressure"]
pyCRS.PropPred.estimate(mol, props, temperatures=[298.15, 308.15, 318.15, 328.15])
print("Results (temp-independent) :", mol.properties)
print("Results (temp-dependent)   :", mol.properties_tdep)

# we can also estimate all properties by supplying the property name 'all' or simply
→ omitting this argument
pyCRS.PropPred.estimate(mol, temperatures=[298.15, 308.15, 318.15, 328.15])
print_props(mol)

```

The output produced is the following:

```

Results (temp-independent) : {'boilingpoint': 435.7771752780941, 'density': 0.
→ 7918196941677842, 'flashpoint': 342.2705857793571, 'meltingpoint': 231.
→ 1412353515625, 'molarvol': 0.1289491355419159}
Results (temp-dependent)   : {'vaporpressure': [(298.1499938964844, 0.
→ 00122854727137622), (308.1499938964844, 0.0026233569814824815), (318.1499938964844,

```

(continues on next page)

(continued from previous page)

```

↪0.005288582928457778), (328.1499938964844, 0.010122673317257832)]]}
boilingpoint      : 435.777 K
criticalpressure   : 34.349 bar
criticaltemp       : 878.101 K
criticalvol        : 0.404 L/mol
density            : 0.792 kg/L (298.15 K)
dielectricconstant : 10.951
entropygas         : 439.885 J/(mol K)
flashpoint         : 342.271 K
gidealgas          : -131.869 kJ/mol
hcombust           : -3678.121 kJ/mol
hformstd           : -384.388 kJ/mol
hfusion            : 18.505 kJ/mol
hidealgas          : -316.821 kJ/mol
hsublimation       : 80.980 kJ/mol
meltingpoint       : 231.141 K
molarvol           : 0.129 L/mol
parachor           : 289.059
solubilityparam    : 10.129  $\sqrt{(\text{cal}/\text{cm}^3)}$ 
synacc             : 6.747
tpt                : 230.404 K
vdwarea            : 171.059  $\text{\AA}^2$ 
vdwvol             : 120.519  $\text{\AA}^3$ 
liquidviscosity    :
      T (K)          liquidviscosity (Pa-s)
      298.150        0.0044653385
      308.150        0.003363708
      318.150        0.0025843814
      328.150        0.0020210327
vaporpressure      :
      T (K)          vaporpressure (bar)
      298.150        0.0012285473
      308.150        0.002623357
      318.150        0.0052885829
      328.150        0.010122673

```

### Calculating sigma profiles with all models

This example demonstrates how to calculate COSMO-RS sigma profiles with both available models (SG1 and FS1). We'll use 4-Methylphenol for this example. The variable `ref_sp` in the script is the  $\sigma$ -profile calculated in the AMS COSMO-RS.

```

import pyCRS
import matplotlib.pyplot as plt
from pyCRS.Database import COSKFDatabase
from pyCRS.CRSManger import CRSSystem

db = COSKFDatabase("my_coskf_db.db")
db.add_compound("4-Methylphenol.coskf")

cal = CRSSystem()
mixture = {"4-Methylphenol": 1.0}

cal.add_Mixture(mixture, problem_type="PURESIGMAPROFILE")
cal.runCRSJob()

```

(continues on next page)

(continued from previous page)

```

out = cal.outputs[0]
res = out.get_results()

chdens = res["chdval"][0]
ref_sp = res["profil"][0]

# chdens = [-0.025 + 0.001 * x for x in range(51)]
# ref_sp = [0.0, 0.0, 0.0, 0.0, 0.0, 0.170502089, 0.735172882, 0.983838864, 0.
↳86397332, 1.15571158, 0.670512863, 0.793806705, 0.714228251, 0.688547843, 0.
↳72126747, 0.724141667, 0.832938008, 2.33617486, 5.42197092, 8.58230745, 9.84294559,
↳8.81993395, 8.8093211, 6.79578176, 6.57764296, 6.0780249, 9.19651976, 11.0472172,
↳12.8025552, 11.1767254, 12.2878801, 10.7212122, 2.77829981, 1.11136819, 1.58235813,
↳1.13444125, 1.8131698, 2.4446856, 2.02558727, 0.0124393296, 0.0, 0.0, 0.0, 0.0, 0.0,
↳ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

mol = pyCRS.Input.read_smiles("Cc1ccc(O)cc1")

pyCRS.FastSigma.estimate(mol, method="COSMO-RS", model="FS1")
sp_fs1 = mol.get_sigma_profile()["Total Profile"]

pyCRS.FastSigma.estimate(mol, method="COSMO-RS", model="SG1")
sp_sg1 = mol.get_sigma_profile()["Total Profile"]

plt.plot(chdens, ref_sp, "--", label="Reference  $\sigma$ -profile")
plt.plot(chdens, sp_fs1, label="FS1  $\sigma$ -profile")
plt.plot(chdens, sp_sg1, label="SG1  $\sigma$ -profile")
plt.ylabel("Area ( $\text{\AA}^2$ )")
plt.xlabel(" $\sigma$ ")
plt.grid()
plt.legend()
# plt.savefig('./pyCRS_PropPred_SigmaProfile.png')
plt.show()

```

Finally, the plot produced shows the various  $\sigma$ -profiles produced.

## 10.3.4 Examples using PLAMS

### Partition coefficient

In this example, we calculate the logP of Ibuprofen. We use the standard octanol/water system. The code is as follows:

```

from scm.plams import Settings, init, finish, CRSJob
import os

##### Note: Ensure to configure the database path to either the installed ADFCRS-
↳2018 directory or your own specified directory #####

database_path = os.path.join(os.environ["SCM_PKG_ADFCRSDIR"], "ADFCRS-2018")
if not os.path.exists(database_path):
    raise OSError(f"The provided path does not exist. Exiting.")

# initialize settings object
settings = Settings()
settings.input.property._h = "LOGP"

```

(continues on next page)

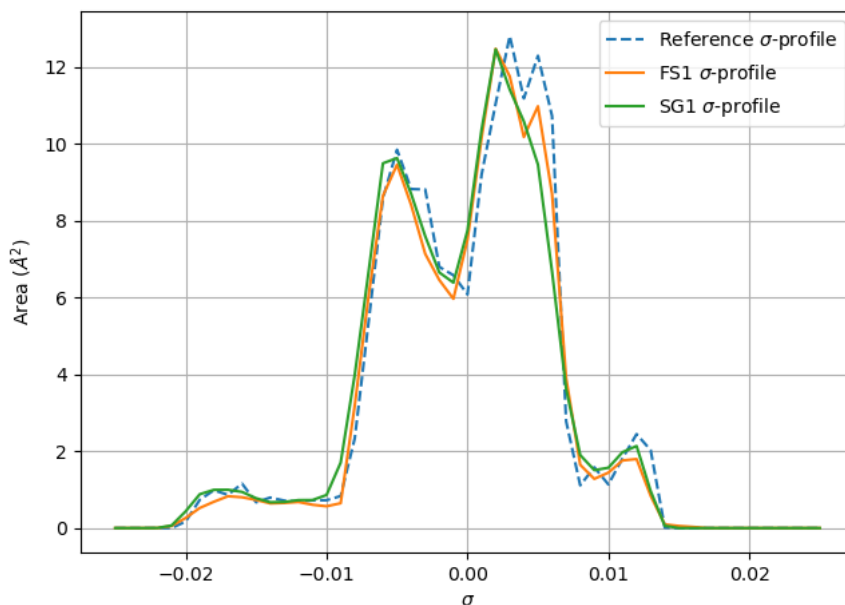


Fig. 10.3: The sigma profile of 4-Methylphenol

(continued from previous page)

```

# set the number of compounds
num_compounds = 3

compounds = [Settings() for i in range(num_compounds)]
compounds[0]._h = os.path.join(database_path, "1-Octanol.coskf")
compounds[1]._h = os.path.join(database_path, "Water.coskf")
compounds[2]._h = os.path.join(database_path, "Ibuprofen.coskf")

# phase1 (octanol phase)
compounds[0].frac1 = 0.725
compounds[1].frac1 = 0.275

# phase2 (water phase)
compounds[0].frac2 = 0
compounds[1].frac2 = 1

# set temperature (range)
# to specify a range, use 3 numbers: (1) the lowest temperature,
# (2) the highest temperature, and (3) the steps taken between these temperatures
settings.input.temperature = "298.15"

# specify the compounds as the compounds to be used in the calculation
settings.input.compound = compounds
# create a job that can be run by COSMO-RS
my_job = CRSJob(settings=settings)
# run the job
init()
out = my_job.run()

```

(continues on next page)

(continued from previous page)

```

finish()

# convert all the results into a python dict
res = out.get_results()

# print the logP of Ibuprofen
print("logP of Ibuprofen:", res["logp"][2])

```

This generates the following output:

```
logP of Ibuprofen: [ 4.67381309]
```

## Binary mixture

In this example, we calculate a binary mixture of water and 2-Hexanone and plot the vapor pressures as a function of composition. We also show how to change the method and calculate the binary mixture with the COSMO-SAC2013-Xiong model.

```

from scm.plams import Settings, init, finish, CRSJob
import os

##### Note: Ensure to configure the database path to either the installed ADFCRS-
↳2018 directory or your own specified directory #####

database_path = os.path.join(os.environ["SCM_PKG_ADFCRSDIR"], "ADFCRS-2018")
if not os.path.exists(database_path):
    raise OSError(f"The provided path does not exist. Exiting.")

# initialize settings object
settings = Settings()
settings.input.property._h = "BINMIXCOEF"

# let's also change to the COSMOSAC2013 method
settings.input.method = "COSMOSAC2013"

# set the number of compounds
num_compounds = 2

compounds = [Settings() for i in range(num_compounds)]
compounds[0]._h = os.path.join(database_path, "Water.coskf")
compounds[1]._h = os.path.join(database_path, "2-Hexanone.coskf")

# use the vapor pressures from the VPM1 model
compounds[0].vp_equation = "VPM1"
compounds[0].vp_params = "-6093.40215895 -3.09584608667 0.000498622924643 34.
↳47450247140318 0.0"
compounds[1].vp_equation = "VPM1"
compounds[1].vp_params = "-6474.348470271438 -6.057589837807771 0.003390587477679571
↳51.07134238467479 0.0"

# set temperature (range)
# to specify a range, use 3 numbers: (1) the lowest temperature,
# (2) the highest temperature, and (3) the steps taken between these temperatures
settings.input.temperature = "298.15"

```

(continues on next page)

(continued from previous page)

```

# specify the compounds as the compounds to be used in the calculation
settings.input.compound = compounds
# create a job that can be run by COSMO-RS
my_job = CRSJob(settings=settings)
# run the job
init()
out = my_job.run()
finish()

# convert all the results into a python dict
res = out.get_results()

# plot all the pressures as a function of mole fraction of water
out.plot(
    "vapor pressure",
    "pressure",
    x_axis=res["molar fraction"][0],
    x_label="mole fraction water",
    y_label="Pressure (bar)",
)

```

The code generates the following plot:

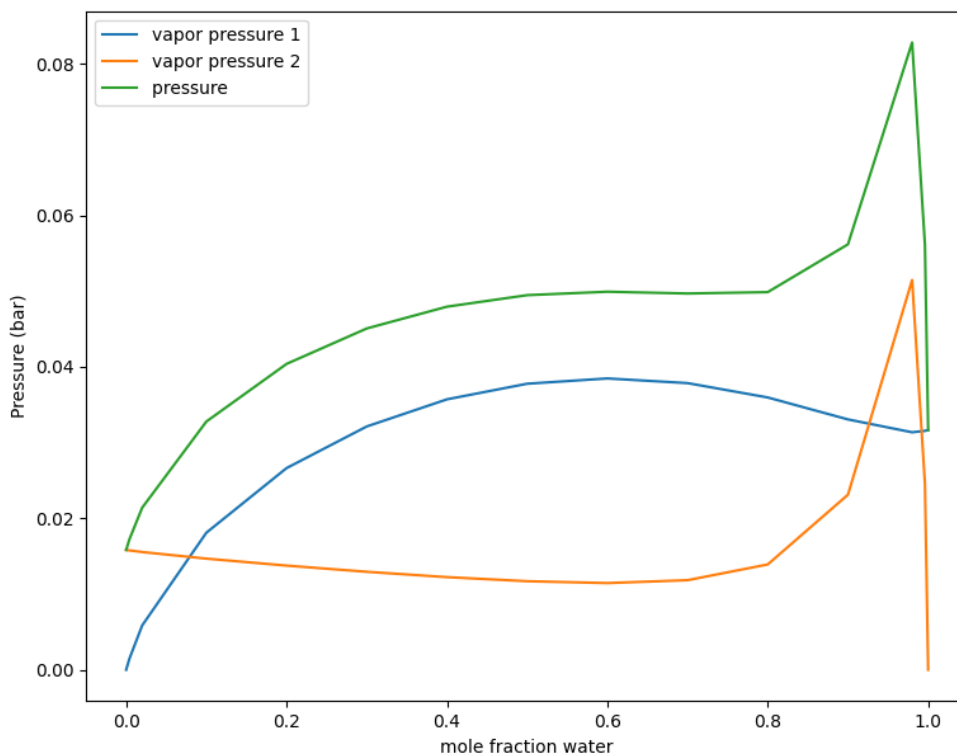


Fig. 10.4: A plot showing the total and partial vapor pressures for the water/2-Hexanone system.

Water and 2-Hexanone do not mix so well, thus there will be a miscibility gap, which is not taken into account in the graph. COSMO-SAC calculates the miscibility gap for the water/2-Hexanone system for molar fraction of water between around 0.29 and around 0.998. Within the miscibility gap the results shown in the graph use the unphysical condition that the two liquids are forced to mix.

## Solid solubility

In this example, we calculate the solubility of ibuprofen in water at 298.15K. When considering a compound in its solid state, it's essential to account for the energy change of a compound from the subcooled liquid state to the ordered solid state. The energy change can be estimated using its heat of fusion and melting point. The heat capacity of fusion, while beneficial, is often not readily available.

```

from scm.plams import Settings, init, finish, CRSJob
import os

##### Note: Ensure to configure the database path to either the installed ADFCRS-
↳2018 directory or your own specified directory #####

database_path = os.path.join(os.environ["SCM_PKG_ADFCRSDIR"], "ADFCRS-2018")
if not os.path.exists(database_path):
    raise OSError(f"The provided path does not exist. Exiting.")

settings = Settings()
settings.input.property._h = "SOLUBILITY"
settings.input.method = "COSMORS"

num_compounds = 2
compounds = [Settings() for i in range(num_compounds)]
compounds[0]._h = os.path.join(database_path, "Water.coskf")
compounds[0].frac1 = 1.0
compounds[1]._h = os.path.join(database_path, "Ibuprofen.coskf")
compounds[1].frac1 = 0.0
compounds[1].nring = 6
compounds[1].hfusion = 27.94 / 4.184 # kcal/mol
compounds[1].meltingpoint = 347.6 # K
# compounds[1].cpfusion = #kcal/mol-K

settings.input.temperature = 298.15

# specify the compounds as the compounds to be used in the calculation
settings.input.compound = compounds
# create a job that can be run by COSMO-RS
my_job = CRSJob(settings=settings)
# print out the input file
# print(my_job.get_input())

# run the job
init()
out = my_job.run()
finish()

# convert all the results into a python dict
res = out.get_results()

# print the solubility of Ibuprofen in water
print("Solubility of Ibuprofen in water [g/L]:", res["solubility g_per_L_solvent"][1])

```

This generates the following output:

```
Solubility of Ibuprofen in water [g/L]: [0.02799089]
```

### 10.3.5 Advanced scripting examples with PLAMS/pyCRS

COSMO-RS is capable of calculating a wide variety of important thermodynamic properties, but not all of those properties are easily available from the GUI. In this section, several scripts are provided that calculate more complex properties and demonstrate additional useful workflows. These scripts can be used as they are or used as a template to develop additional or more specific functionality.

**Note:** To ensure compatibility among the necessary python modules, it is recommended to use SCM's python distribution, `amspython`. Executing the scripts in this section from the command line can be done simply as follows: `amspython <script_name.py>`. Those users with more python experience or who prefer to use their own version of python are welcome to use alternatives to `amspython`.

#### Table of Contents:

#### Changing the default parameters or re-parameterizing the COSMO-RS/-SAC methods

There are many situations for which changing the default COSMO-RS/-SAC parameters may be useful. Most commonly, users may wish to try a certain parameterization that is not available from the program. Alternatively, some users may have a customized or proprietary dataset which they would like to use to re-fit the main model parameters. All of these tasks are straightforward via python scripting.

The following scripts will demonstrate how to alter parameters for COSMO-RS and COSMO-SAC.

#### Python code (COSMO-RS parameters)

```
import os
from scm.plams import Settings, init, finish, CRSJob, config

##### Note: Ensure to configure the database path to either the installed ADFCRS-
↳2018 directory or your own specified directory #####

database_path = os.path.join(os.environ["SCM_PKG_ADFCRSDIR"], "ADFCRS-2018")
if not os.path.exists(database_path):
    raise OSError(f"The provided path does not exist. Exiting.")

init()
# suppress plams output
config.log.stdout = 0

# our system of interest
files = ["Acetone.coskf", "Water.coskf"]
fracs = [0.3, 0.7]

# initialize settings object
settings = Settings()
settings.input.property._h = "ACTIVITYCOEF"
settings.input.temperature = 298.15
```

(continues on next page)

(continued from previous page)

```
compounds = [Settings() for i in range(len(files))]
for i, (name, frac) in enumerate(zip(files, fracs)):
    compounds[i]._h = os.path.join(database_path, name)
    compounds[i].frac1 = frac

settings.input.compound = compounds

# Here, we will change the method parameters specific to COSMO-RS
# Main CRS parameters
settings.input.CRSParameters.rav = 0.400
settings.input.CRSParameters.aprime = 1510.0
settings.input.CRSParameters.fcorr = 2.802
settings.input.CRSParameters.chb = 8850.0
settings.input.CRSParameters.sigmahbond = 0.00854
settings.input.CRSParameters.aeff = 6.94
settings.input.CRSParameters.Lambda = 0.130
settings.input.CRSParameters.omega = -0.212
settings.input.CRSParameters.eta = -9.65
settings.input.CRSParameters.chortf = 0.816
settings.input.CRSParameters.HB_HNOF = "" # hb for only H,N,O,F
# settings.input.CRSParameters.HB_ALL = "" # hb for all elements
settings.input.CRSParameters.HB_TEMP = "" # temperature-dependent H-bond
# settings.input.CRSParameters.HB_NOTEMP = "" # non-temperature-dependent H-bond
settings.input.CRSParameters.COMBI2005 = "" # default combinatorial term
# settings.input.CRSParameters.COMBI1998 = ""

# Dispersion parameters
settings.input.Dispersion.H = -0.0340
settings.input.Dispersion.C = -0.0356
settings.input.Dispersion.N = -0.0224
settings.input.Dispersion.O = -0.0333
settings.input.Dispersion.F = -0.026
settings.input.Dispersion.Si = -0.04
settings.input.Dispersion.P = -0.045
settings.input.Dispersion.S = -0.052
settings.input.Dispersion.Cl = -0.0485
settings.input.Dispersion.Br = -0.055
settings.input.Dispersion.I = -0.062

# Technical and accuracy parameters
settings.input.Technical.rsconv = 1e-7
settings.input.Technical.maxiter = 10000
settings.input.Technical.bpconv = 1e-6
settings.input.Technical.bpmaxiter = 40
settings.input.Technical.solconv = 1e-5
settings.input.Technical.solmaxiter = 40
settings.input.Technical.solxilarge = 0.99
settings.input.Technical.ehdeltaT = 1.0

# We will vary the chb parameter (default value 8850.0)
# and observe the effect on activity coefficients

print("Resulting Activity Coefficients:")
print("chb value".ljust(15), "activity coefficients".ljust(20))
hbvals = [8700.0 + 50 * i for i in range(7)]
for hbval in hbvals:
```

(continues on next page)

(continued from previous page)

```

settings.input.CRSPParameters.chb = hbval
# create a job that can be run by COSMO-RS
my_job = CRSJob(settings=settings)
# run the job
out = my_job.run()
# convert all the results into a python dict
res = out.get_results()

print(str(hbval).ljust(15), str(res["gamma"].flatten()).ljust(20))

finish()

```

### Python code (COSMO-SAC parameters)

```

import os
from scm.plams import Settings, init, finish, CRSJob, config

##### Note: Ensure to configure the database path to either the installed ADFCRS-
↳2018 directory or your own specified directory #####

database_path = os.path.join(os.environ["SCM_PKG_ADFCRSDIR"], "ADFCRS-2018")
if not os.path.exists(database_path):
    raise OSError(f"The provided path does not exist. Exiting.")

init()
# suppress plams output
config.log.stdout = 0

# our system of interest
files = ["Acetone.coskf", "Water.coskf"]
fracs = [0.3, 0.7]

# initialize settings object
settings = Settings()
settings.input.property._h = "ACTIVITYCOEF"
settings.input.temperature = 298.15

compounds = [Settings() for i in range(len(files))]
for i, (name, frac) in enumerate(zip(files, fracs)):
    compounds[i]._h = os.path.join(database_path, name)
    compounds[i].frac1 = frac

settings.input.compound = compounds

# Here, we will change the method parameters specific to COSMO-SAC
# First, change the method to COSMOSAC2013
settings.input.method = "COSMOSAC2013"

# Main SAC parameters
settings.input.SACParameters.aeff = 6.4813
settings.input.SACParameters.fdecay = 0.0
settings.input.SACParameters.sigma0 = 0.01233
settings.input.SACParameters.rn = 0.0
settings.input.SACParameters.qn = 79.532

```

(continues on next page)

(continued from previous page)

```

settings.input.SACParameters.aes = 7877.13
settings.input.SACParameters.bes = 0.0
settings.input.SACParameters.cohoh = 5786.72
settings.input.SACParameters.cotot = 2739.58
settings.input.SACParameters.cohot = 4707.75
settings.input.SACParameters.rav = 0.51
settings.input.SACParameters.qs = 0.57
settings.input.SACParameters.rhbcut = 0.0
settings.input.SACParameters.omega = 0.0
settings.input.SACParameters.eta = 0.0
settings.input.SACParameters.HB_NOTEMP = "" # non-temperature-dependent H-bonding_
↪ (default)
# settings.input.SACParameters.HB_TEMP = "" # temperature-dependent H-bonding

# Epsilon Constants
settings.input.Epsilon.H = 338.13
settings.input.Epsilon["C.sp3"] = 29160.92
settings.input.Epsilon["C.sp2"] = 30951.83
settings.input.Epsilon["C.sp"] = 20685.98
settings.input.Epsilon["N.sp3"] = 23488.54
settings.input.Epsilon["N.sp2"] = 22663.38
settings.input.Epsilon["N.sp"] = 6390.40
settings.input.Epsilon["O.sp3-H"] = 8527.06
settings.input.Epsilon["O.sp3"] = 8484.38
settings.input.Epsilon["O.sp2"] = 6736.85
settings.input.Epsilon["O.sp2-N"] = 12145.28
settings.input.Epsilon.F = 8435.13
settings.input.Epsilon.P = 82512.21
settings.input.Epsilon.S = 56067.81
settings.input.Epsilon.Cl = 45065.19
settings.input.Epsilon.Br = 62947.83
settings.input.Epsilon.I = 105910.88

# Technical and accuracy parameters
settings.input.Technical.sacconv = 1e-7
settings.input.Technical.maxiter = 10000
settings.input.Technical.bpconv = 1e-6
settings.input.Technical.bpmaxiter = 40
settings.input.Technical.solconv = 1e-5
settings.input.Technical.solmaxiter = 40
settings.input.Technical.solxilarge = 0.99
settings.input.Technical.ehdeltaT = 1.0

# We will vary the cohot parameter (default value 4707.75)
# and observe the effect on activity coefficients
print("Resulting Activity Coefficients:")
print("cohot value".ljust(15), "activity coefficients".ljust(20))
cohot_vals = [4707.75 + 50 * i for i in range(-3, 4)]
for cohot in cohot_vals:

    settings.input.SACParameters.cohot = cohot
    # create a job that can be run by COSMO-RS
    my_job = CRSJob(settings=settings)
    # run the job
    out = my_job.run()
    # convert all the results into a python dict
    res = out.get_results()

```

(continues on next page)

(continued from previous page)

```

print(str(cohot).ljust(15), str(res["gamma"].flatten()).ljust(20))

finish()

```

## Solubility screening for solid solute

Download relevant coskf file

In this example, we explore the solubility of avobenzene in several solvents and compare the predicted solubility with experimental data<sup>1</sup>. When conducting solid solubility calculations, it's crucial to provide the melting point and melting enthalpy of the solid solute. If experimental melting properties are not available, it's possible to estimate these values using the pure properties prediction tool, which is based on the group contribution method. Alternatively, the ranking of solubility can also be compared using the infinite dilution activity coefficient.

In the first script example, it utilizes the *pyCRS.Database* (page 96) and *pyCRS.CRSManger* (page 98) modules to systematically set up solubility calculations and infinite dilution activity coefficients calculation. These modules significantly simplify the process, providing users with a direct and efficient method to perform high-throughput screening tasks.

## Python code using pyCRS

```

from scm.plams import Settings, init, finish, config, JobRunner
from pyCRS.Database import COSKFDatabase
from pyCRS.CRSManger import CRSSystem
from pathlib import Path
import matplotlib.pyplot as plt
import numpy as np

init()

##### Step 1 : add compounds to a pyCRS.databse. Ensure coskf_solubility is_
↳downloaded #####

##### Note: Ensure to download the coskf_solubility before running the script ####
↳#####

db = COSKFDatabase("my_coskf_db.db")

coskf_path = Path("./coskf_solubility")

if not coskf_path.exists():
    raise OSError(f"The provided path does not exist. Exiting.")

for file in coskf_path.glob("*.coskf"):

    if file.name == "avobenzene.coskf":
        db.add_compound(file.name, name="avobenzene", cas="70356-09-1", coskf_
↳path=coskf_path)
        db.add_physical_property("avobenzene", "meltingpoint", 347.6)
        db.add_physical_property("avobenzene", "hfusion", 5.565, unit="kcal/mol")
        db.estimate_physical_property("avobenzene")

```

(continues on next page)

<sup>1</sup> Benazzouz, Adrien, et al. "Hansen approach versus COSMO-RS for predicting the solubility of an organic UV filter in cosmetic solvents." *Colloids and Surfaces A: Physicochemical and Engineering Aspects* 458 (2014): 101-109.

(continued from previous page)

```

else:
    db.add_compound(file.name, coskf_path=coskf_path)

# Experimental solubility data in %w/w
exp_data = {
    "Glycerol": 0.1,
    "1,2-Propylene glycol": 0.2,
    "Hexadecane": 1,
    "Ethanol": 2,
    "Di-n-octyl ether": 5,
    "beta-Pinene": 7.7,
    "Isopropyl myristate": 10,
    "Propylene carbonate": 10.7,
    "Di-2-ethylhexyl-adipate": 12,
    "Dimethyl-isosorbide": 38.2,
    "Dimethoxymethane": 73,
}

##### Step2 : Iterately set up calculation for solubility and activitycoef #####

## Set up for parallel run ##
if True:
    config.default_jobrunner = JobRunner(
        parallel=True, maxjobs=8
    ) # Set jobrunner to be parallel and specify the numbers of jobs run_
    ↪simultaneously (eg. multiprocessing.cpu_count())
    config.default_jobmanager.settings.hashing = None # Disable rerun prevention
    config.job.runscript.nproc = 1 # Number of cores for each job
    config.log.stdout = 1 # suppress plams output default=3

##### Step2 : Iterately set up calculation for solubility and activitycoef #####
cal = CRSSystem()
cal2 = CRSSystem()

for solvent in exp_data.keys():
    mixture = {}
    mixture[solvent] = 1.0
    mixture["avobenzene"] = 0.0

    cal.add_Mixture(
        mixture,
        database="my_coskf_db.db",
        temperature=298.15,
        problem_type="solubility",
        solute="solid",
        jobname="solubility",
    )
    cal2.add_Mixture(
        mixture, database="my_coskf_db.db", temperature=298.15, problem_type=
    ↪"activitycoef", jobname="IDAC"
    )

cal.runCRSJob()
cal2.runCRSJob()

##### Step3 : Output processing to retrieve results for plotting #####

```

(continues on next page)

(continued from previous page)

```

solubility = []
lnIDAC = []
for out, out2 in zip(cal.outputs, cal2.outputs):
    res = out.get_results()
    res2 = out2.get_results()

    solubility.append(res["solubility massfrac"][1][0] * 100)
    lnIDAC.append(np.log(res2["gamma"][1][0]))

plt.rcParams["figure.figsize"] = (9, 4)
fig, axs = plt.subplots(1, 2)
axs[0].plot(solubility, exp_data.values(), "o", color="Red", markerfacecolor="none")
axs[0].plot([-5, 80], [-5, 80], color="gray")
axs[0].set_xlim([-5, 80])
axs[0].set_ylim([-5, 80])
axs[0].set_xlabel("predicted solubility(%w/w)")
axs[0].set_ylabel("experimental solubility(%w/w)")

axs[1].plot(lnIDAC, exp_data.values(), "o", color="Blue", markerfacecolor="none")
axs[1].set_xlabel("ln(infinite dilution activitycoef) ")
axs[1].set_ylabel("experimental solubility(%w/w)")
plt.tight_layout()
# plt.savefig('./images/pyCRS_solubility_screening.png', dpi=300)
plt.show()

finish()

```

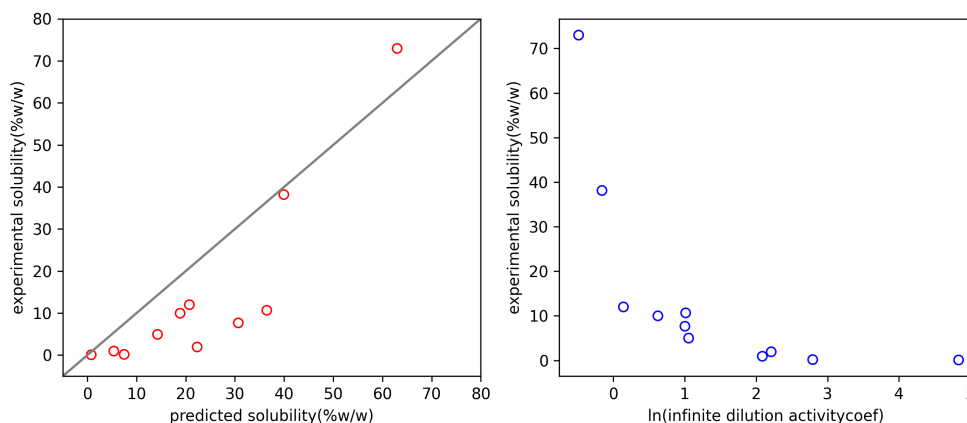


Fig. 10.5: The comparison of experimental solubility with (1) predicted solubility and (2) infinite dilution activity coefficient.

In the second script example, the solubility screening is configured using the `set_CRSSJob_solubility` function within the script. While this approach requires appropriately setting up the *PLAMS input setting* (page 130) within the function, it provides greater flexibility for configuring unconventional calculations, such as *ionic liquids screening* (page 176).

## Python code using PLAMS

```

import os, time
import multiprocessing
import numpy as np
import matplotlib.pyplot as plt
from scm.plams import Settings, init, finish, CRSJob, config, JobRunner

init()

##### Note: Ensure to download the coskf_solubility before running the script #####
->###

def set_CRSJob_solubility(index, ncomp, coskf, database_path, cal_type, method,
->temperature, comp_input={}):

    s = Settings() # initialize a settings object
    s.input.property._h = cal_type # specify problem type
    s.input.method = method # specify method
    s.input.temperature = temperature # specify temperature

    compounds = [Settings() for i in range(ncomp)] # initialization of compounds
    for i in range(ncomp):
        compounds[i]._h = os.path.join(database_path, coskf[i]) # specify absolute_
->directory of coskf file
        for column, value in comp_input.items(): # specify compound's information_
->through comp_input, for example
            if value[i] != None: # column: frac1, meltingpoint, hfusion
                compounds[i][column] = value[i]
    s.input.compound = compounds

    my_job = CRSJob(settings=s) # create jobs
    my_job.name = cal_type + "_" + str(index) # specify job name

    return my_job

Parallel_run = True
Plot_option = True

if Parallel_run:
    config.default_jobrunner = JobRunner(
        parallel=True, maxjobs=8
    ) # Set jobrunner to be parallel and specify the numbers of jobs run_
->simultaneously (eg. multiprocessing.cpu_count())
    config.default_jobmanager.settings.hashing = None # Disable rerun prevention
    config.job.runscript.nproc = 1 # Number of cores for each job
    config.log.stdout = 1 # suppress plams output default=3

###----INPUT YOUR DATA HERE----###

database_path = os.path.join(os.getcwd(), "coskf_solubility")
cal_type = "solubility"
method = "COSMORS"
ncomp = 2
solvents = [

```

(continues on next page)

(continued from previous page)

```

"Glycerol.coskf",
"1,2-Propylene_glycol.coskf",
"Hexadecane.coskf",
"Ethanol.coskf",
"Di-n-octyl_ether.coskf",
"beta-Pinene.coskf",
"Isopropyl_myristate.coskf",
"Propylene_carbonate.coskf",
"Di-2-ethylhexyl-adipate.coskf",
"Dimethyl-isosorbide.coskf",
"Dimethoxymethane.coskf",
]
solute = "avobenzone.coskf"

temperature = 298.15 # K

# store input data along with the necessary thermal property in comp_input dictionary
comp_input = {}
comp_input["name"] = ["solvent", "solute"]
comp_input["frac1"] = [1, 0] # mole fraction
comp_input["meltingpoint"] = [None, 355] # melting point(K)
comp_input["hfusion"] = [None, 5.565] # heat of fusion(Kcal/mol)

# Experimental solubility data in %w/w
exp_sol = [0.1, 0.2, 1, 2, 5, 7.7, 10, 10.7, 12, 38.2, 73]

###----INPUT END----###

index = 0
outputs = []
for solv in solvents:
    coskf = [solv, solute]
    job = set_CRSSJob_solubility(index, ncomp, coskf, database_path, cal_type, method,
    ↪temperature, comp_input)
    index = index + 1
    outputs.append(job.run())

print(job.get_input())

# In a parallel run, the get_results function will wait for the completion of the
    ↪corresponding jobs.
results = []
for out in outputs:
    results.append(out.get_results())

if Plot_option:
    if cal_type == "solubility":

        cal_sol = [res["solubility massfrac"][1][0] * 100 for res in results]
        cal_lfact = [np.log(res["gamma"][1][0]) for res in results]

        plt.figure(figsize=(5, 4))
        plt.plot(cal_sol, exp_sol, "o", color="Red", markerfacecolor="none",
    ↪label=method)
        plt.plot([-5, 80], [-5, 80], color="gray")
        plt.xlim([-5, 80])

```

(continues on next page)

(continued from previous page)

```

plt.ylim([-5, 80])
plt.xlabel("predicted solubility(%w/w)")
plt.ylabel("experimental solubility(%w/w)")

plt.tight_layout()
# plt.savefig('./PLAMS_solubility_screening.png',dpi=300)
plt.show()

finish()

```

## References

### Screening for cocrystals

In this section, we provide an example application that can be used as a template for many high-throughput screening scripts. Cocrystals are crystals formed from two or more compounds in a defined stoichiometry. There are many uses for cocrystals, especially for pharmaceutical applications where one compound is an active pharmaceutical ingredient (API). This example problem screens multiple compounds for their potential as components of a (1:2) cocrystal with Itraconazole. This problem uses the excess enthalpy for a hypothetical supercooled liquid phase as a proxy for cocrystallization affinity. The rankings of the solvents are in good agreement with model and experimental results for this problem given in<sup>1</sup>.

Download relevant coskf file: `coskf_Hex.zip`

### Python code using pyCRS

```

from scm.plams import Settings, init, finish, CRSJob, config, JobRunner, KFFile
from pyCRS.Database import COSKFDatabase
from pyCRS.CRSManager import CRSSystem
from pathlib import Path
import matplotlib.pyplot as plt
import numpy as np

init()

##### Step 1 : add compounds to a pyCRS.database. Ensure coskf_Hex is downloaded ####
↪###

##### Note: Ensure to download the coskf_Hex before running the script #####
↪###

db = COSKFDatabase("my_coskf_db.db")

coskf_path = Path("./coskf_Hex")

if not coskf_path.exists():
    raise OSError(f"The provided path does not exist. Exiting.")

CAS_ref = {
    "tartaric_acid": "526-83-0",

```

(continues on next page)

<sup>1</sup> Abramov, Yuriy A., Christoph Loschen, and Andreas Klamt. "Rational cofomer or solvent selection for pharmaceutical cocrystallization or desolvation." *Journal of pharmaceutical sciences* 101.10 (2012): 3687-3697. (<https://doi.org/10.1002/jps.23227>)

(continued from previous page)

```

    "fumaric_acid": "110-17-8",
    "succinic_acid": "110-15-6",
    "malic_acid": "6915-15-7",
    "glutaric_acid": "110-94-1",
    "malonic_acid": "141-82-2",
    "adipic_acid": "124-04-9",
    "maleic_acid": "110-16-7",
    "itz": "84625-61-6",
}

for file in coskf_path.glob("*.coskf"):
    for key, val in CAS_ref.items():
        if key in file.name:
            CAS = val
            name = key.replace("_", " ")
            if name == "itz":
                name = "itraconazole"

            db.add_compound(file.name, cas=CAS, name=name, coskf_path=coskf_path)

##### Step2 : Iterately set up calculation for solubility and activitycoef #####

## Set up for parallel run ##
if True:
    config.default_jobrunner = JobRunner(
        parallel=True, maxjobs=8
    ) # Set jobrunner to be parallel and specify the numbers of jobs run_
    ↪simultaneously (eg. multiprocessing.cpu_count())
    config.default_jobmanager.settings.hashing = None # Disable rerun prevention
    config.job.runscript.nproc = 1 # Number of cores for each job
    config.log.stdout = 0 # suppress plams output default=3

solv_name = [x.replace("_", " ") for x in list(CAS_ref.keys())[0:-1]]

cal = CRSSystem()
cal2 = CRSSystem()

for co_solvent in solv_name:
    mixture = {}
    mixture[co_solvent] = 0.33333
    mixture["itraconazole"] = 0.66667

    cal.add_Mixture(
        mixture,
        database="my_coskf_db.db",
        temperature=298.15,
        problem_type="VAPORPRESSURE",
        conformer=True,
        jobname="conf",
    )
    cal2.add_Mixture(
        mixture,
        database="my_coskf_db.db",
        temperature=298.15,
        problem_type="VAPORPRESSURE",
        conformer=False,
        jobname="Emin",
    )

```

(continues on next page)

```

)

cal.runCRSJob()
cal2.runCRSJob()

##### Step3 : Output processing to retrieve results for plotting #####

print("Solvent".ljust(14), "Hex_Emin", "Hex_conf", "Population of solvent's conformers
→")
Hex_conf = []
Hex_Emin = []
for out, out2, name in zip(cal.outputs, cal2.outputs, solv_name):
    res = out.get_results()
    res2 = out2.get_results()

    Hex = res["excess H"]
    Hex2 = res2["excess H"]

    Hex_conf.append(Hex)
    Hex_Emin.append(Hex2)

    compositions = out.get_multispecies_dist()[0]

    print(name.ljust(15), end="")
    print(f"{Hex2:.3f}", end=" ")
    print(f"{Hex:.3f}", end=" ")
    for conf, frac in compositions.items():
        print(f"{frac[0]:.5f}", end=" ")
    print("")

if True:
    plt_index = [i for i in range(len(Hex_conf))]
    plt.xlabel("Excess enthalpy (kcal/mol)")
    plt.barh(plt_index, Hex_conf, zorder=3)
    plt.yticks(plt_index, solv_name)
    plt.grid(axis="x", ls="--", zorder=0)
    plt.gca().invert_xaxis()
    # plt.savefig('./Cocrystal_screening.png', dpi=300)
    plt.title("Hex with conformers")
    plt.tight_layout()

    plt.show()

if True:
    plt_index = [i for i in range(len(Hex_Emin))]
    plt.xlabel("Excess enthalpy (kcal/mol)")
    plt.barh(plt_index, Hex_Emin, zorder=3)
    plt.yticks(plt_index, solv_name)
    plt.grid(axis="x", ls="--", zorder=0)
    plt.gca().invert_xaxis()
    # plt.savefig('./Cocrystal_screening_Emin.png', dpi=300)
    plt.title("Hex with Lowest energy conformer")
    plt.tight_layout()

    plt.show()

```

## Python code using PLAMS

```

import os, time
import multiprocessing
import numpy as np
import matplotlib.pyplot as plt
from scm.plams import Settings, init, finish, CRSJob, config, JobRunner

def set_CRSJob_Hex_conf(index, ncomp, coskf, database_path, cal_type, method,
    ↪temperature, comp_input={}, comp_type=[]):

    s = Settings() # initialize a settings object
    s.input.property._h = cal_type # specify problem type
    s.input.method = method # specify method
    s.input.temperature = temperature # specify temperature

    compounds = [Settings() for i in range(ncomp)] # initialization of compounds
    for i in range(ncomp):
        if len(comp_type) > i:
            if "conf" in comp_type[i]:
                form = [Settings() for i in range(len(coskf[i]))] # initialize_
    ↪compound in multiple form
                for j in range(len(coskf[i])):
                    form[j]._h = os.path.join(
                        database_path, coskf[i][j]
                    ) # specify conformer's coskf file for each form
                compounds[i].form = form
            else:
                compounds[i]._h = os.path.join(database_path, coskf[i]) # specify_
    ↪absolute directory of coskf file
            else:
                compounds[i]._h = os.path.join(database_path, coskf[i]) # specify_
    ↪absolute directory of coskf file

            for column, value in comp_input.items(): # specify compound's information_
    ↪through comp_input, for example
                if value[i] != None: # column: frac1, meltingpoint, hfusion
                    compounds[i][column] = value[i]

    s.input.compound = compounds

    my_job = CRSJob(settings=s) # create jobs
    my_job.name = cal_type + "_" + str(index) # specify job name

    return my_job

init()

Parallel_run = True
Plot_option = True

if Parallel_run:
    config.default_jobrunner = JobRunner(
        parallel=True, maxjobs=8
    ) # Set jobrunner to be parallel and specify the numbers of jobs run_

```

(continues on next page)

(continued from previous page)

```

↳simultaneously (eg. multiprocessing.cpu_count())
    config.default_jobmanager.settings.hashing = None # Disable rerun prevention
    config.job.runscript.nproc = 1 # Number of cores for each job
    config.log.stdout = 1 # suppress plams output default=3

###----INPUT YOUR DATA HERE----###

database_path = os.path.join(os.getcwd(), "coskf_Hex")
cal_type = "VAPORPRESSURE"
method = "COSMORS"
ncomp = 2
solute = "itz_c1.coskf"

# a list of different conformers for the screening each line contains 3 conformers of
↳the same molecule
solvents = [
    ["tartaric_acid_c1.coskf", "tartaric_acid_c2.coskf", "tartaric_acid_c3.coskf"],
    ["fumaric_acid_c1.coskf", "fumaric_acid_c2.coskf", "fumaric_acid_c3.coskf"],
    ["succinic_acid_c1.coskf", "succinic_acid_c2.coskf", "succinic_acid_c3.coskf"],
    ["malic_acid_c1.coskf", "malic_acid_c2.coskf", "malic_acid_c3.coskf"],
    ["glutaric_acid_c1.coskf", "glutaric_acid_c2.coskf", "glutaric_acid_c3.coskf"],
    ["malonic_acid_c1.coskf", "malonic_acid_c2.coskf", "malonic_acid_c3.coskf"],
    ["adipic_acid_c1.coskf", "adipic_acid_c2.coskf", "adipic_acid_c3.coskf"],
    ["maleic_acid_c1.coskf", "maleic_acid_c2.coskf", "maleic_acid_c3.coskf"],
]

temperature = 298.15 # K

# store input data along with the necessary thermal property in comp_input dictionary
comp_input = {}
comp_input["frac1"] = [0.33333, 0.66667] # the stoichiometric ratio of the co-
↳crystal (solvent:solute)

# enter 'conf' for compound with multiple conformers; '' for compound with single
↳structure
comp_type = ["conf", ""]

###----INPUT END----###

outputs = []
solv_name = []
for index, solv in enumerate(solvents):
    solv_name.append(solv[0].replace("_c1.coskf", ""))
    coskf = [solv, solute]
    comp_input["name"] = [solv[0].replace("_c1.coskf", ""), solute.replace("_c1.coskf
↳", "")]

    job = set_CRSSJob_Hex_conf(index, ncomp, coskf, database_path, cal_type, method,
↳temperature, comp_input, comp_type)
    outputs.append(job.run())

finish()

# In a parallel run, the get_results function will wait for the completion of the
↳corresponding jobs.
results = []
excess_h = []

```

(continues on next page)

(continued from previous page)

```

print("")
print("Solvent".ljust(14), "Population of solvent's conformers")
for out, name in zip(outputs, solv_name):
    res = out.get_results()
    results.append(res)
    excess_h.append(res["excess H"])

    compositions = out.get_multispecies_dist()[0]
    print(name.ljust(15), end="")
    for conf, frac in compositions.items():
        print(f"{frac[0]:.5f}", end=" ")
    print("")

print("")
print("Solvent".ljust(15), "Excess enthalpy (kcal/mol)")
for (
    name,
    Hex,
) in zip(solv_name, excess_h):
    print(name.ljust(15), round(Hex, 5))

if Plot_option:
    plt_index = [i for i in range(len(excess_h))]
    plt.xlabel("Excess enthalpy (kcal/mol)")
    plt.barh(plt_index, excess_h, zorder=3)
    plt.yticks(plt_index, solv_name)
    plt.grid(axis="x", ls="--", zorder=0)
    plt.gca().invert_xaxis()
    plt.tight_layout()
    # plt.savefig('./Cocrystal_screening.png', dpi=300)
    plt.show()

```

This figure (produced by the code) shows the excess enthalpy values of all solvents in a supercooled liquid mixture with Itraconazole. The lowest 4 excess enthalpy values correspond to 4 solvent for which a stable co-crystal with Itraconazole is known<sup>Page 162, 1</sup>.

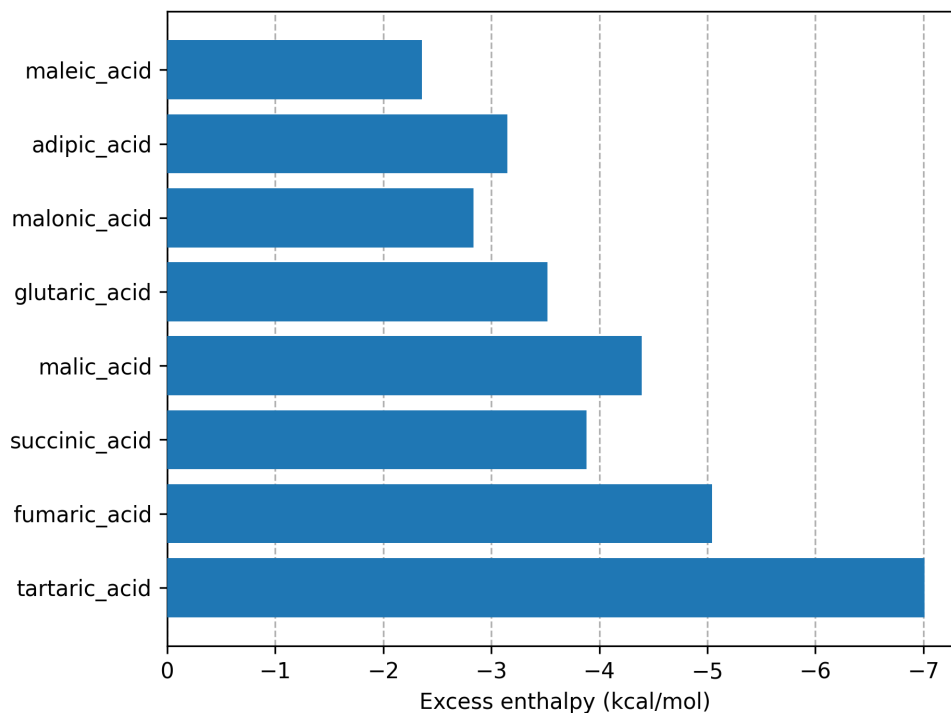
## References

### Regression of NRTL parameters

In many cases, semi-empirical activity coefficient models, such as the NRTL (Non-random two-liquid) model, are more commonly available in engineering software, including process simulation tools. When experimental data is unavailable for a given binary mixture, it can be useful to derive NRTL parameters using activity coefficients from COSMO-RS/-SAC as reference data. This approach allows user to perform simulation by simply importing the fitted NRTL parameters.

The following script demonstrates how to generate a set of activity coefficients for a given mixture and utilize the optimization tools from the scipy library to obtain the optimal parameters for the NRTL model.

In this example, we use the following equation for a binary mixture, adopting the expressions for binary interaction parameters and the non-randomness factor as used in Aspen Plus. The equation can be easily adjusted if a different form



is preferred.

$$\ln \gamma_1 = x_2^2 \left[ \tau_{21} \left( \frac{G_{21}}{x_1 + x_2 G_{21}} \right)^2 + \frac{\tau_{12} G_{12}}{(x_2 + x_1 G_{12})^2} \right]$$

$$\ln \gamma_2 = x_1^2 \left[ \tau_{12} \left( \frac{G_{12}}{x_2 + x_1 G_{12}} \right)^2 + \frac{\tau_{21} G_{21}}{(x_1 + x_2 G_{21})^2} \right]$$

Where :

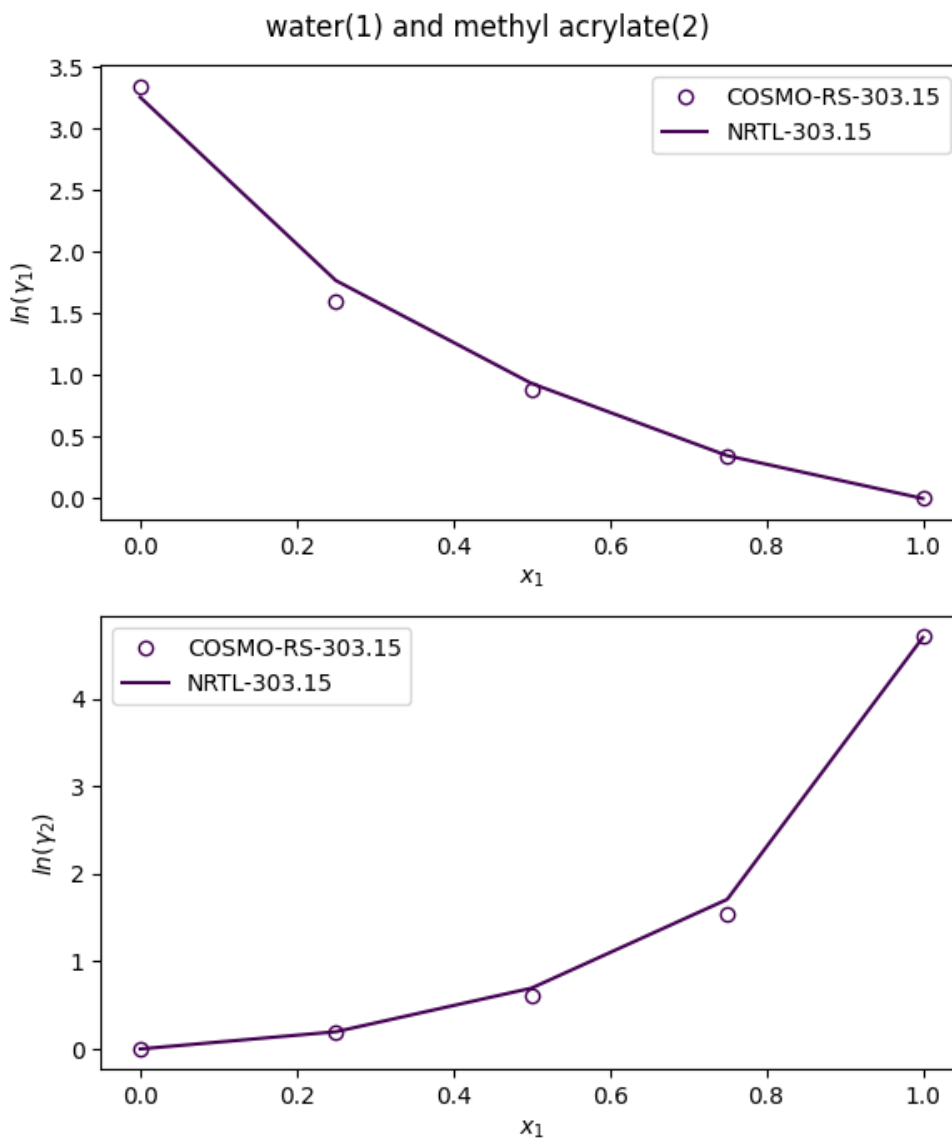
$$G_{ij} = \exp(-\alpha_{ij} \tau_{ij})$$

$$\tau_{ij} = a_{ij} + b_{ij}/T + e_{ij} \ln(T) + f_{ij} T$$

$$\alpha_{ij} = c_{ij} + d_{ij}(T - 273.15)$$

The figure below illustrate the regression for the mixture of water and methyl acrylate at 303K, using a fixed value of non-randomness factor at 0.3. The optimal interaction parameter obtained are 3.595 and 2.030, which are close to the value of 3.464 and 3.563 derived from experimental data regression reported in the literature<sup>1</sup>.

<sup>1</sup> Ravichandran, A., Khare, R., & Chen, C. C. (2018). Predicting NRTL binary interaction parameters from molecular simulations. *AIChE Journal*, 64(7), 2758-2769.



$A_{ij} = 3.5954$ ,  $A_{ji} = 2.0300$   
 $B_{ij} = B_{ji} = \text{Not fitted}$   
 $C_{ij} = C_{ji}$  is fixed to 0.3000  
 $D_{ij} = D_{ji} = \text{Not fitted}$   
 $E_{ij} = E_{ji} = \text{Not fitted}$   
 $F_{ij} = F_{ji} = \text{Not fitted}$   
 objective = 0.0745

## Python code

```

from pyCRS.CRSManger import CRSSystem
from pyCRS.Database import COSKFDatabase
from scm.plams import config, JobRunner
import numpy as np
import random
import sys
from scipy import optimize
import matplotlib.pyplot as plt
import matplotlib.cm as cm

def objective(params, ref_data, coef_fitted=None, coef_asymmetric=None, Cij=None, inf_
↳weight=1.0):
    """
    Evaluate the error between calculated and reference activity coefficients.
    Parameters :
        params (list of float) : Parameters used for model calculations.
        ref_data (dict) : Reference data containing reference activity coefficients.
        coef_fitted (dict, optional) : Flags indicating which coefficients to include_
↳in the fitting.
        coef_asymmetric (dict, optional) : Flags indicating asymmetric of a_
↳coefficient.
        Cij (float, optional): Fixed Cij paramter in model calculation.
        inf_weight (float, optional) : weighting apply to infinite dilute activity_
↳coefficient.

    Returns:
        float : The total weighted error between calculated and reference activity_
↳coefficients.

    """
    if coef_fitted is None:
        coef_fitted = {"A": True, "B": False, "C": True, "D": False, "E": False, "F":_
↳False}

    if coef_asymmetric is None:
        coef_asymmetric = {"A": True, "B": True, "C": False, "D": False, "E": True, "F"
↳": True}

    lng1_calc, lng2_calc = compute_gammas_binaries(params, ref_data, coef_fitted,_
↳coef_asymmetric, Cij)

    weight_lng1 = np.ones(len(ref_data["temperature"]))
    weight_lng2 = np.ones(len(ref_data["temperature"]))

    if inf_weight != 1.0:
        inf1_mask = ref_data["x1"] == 0.0
        weight_lng1[inf1_mask] = inf_weight

        inf2_mask = ref_data["x1"] == 1.0
        weight_lng2[inf2_mask] = inf_weight

    errors = np.sum(
        weight_lng1 * np.power(lng1_calc - ref_data["lngamma1"], 2.0)
        + weight_lng2 * np.power(lng2_calc - ref_data["lngamma2"], 2.0)

```

(continues on next page)

(continued from previous page)

```

)

return errors

def compute_gammas_binaries(params, ref_data, coef_fitted, coef_asymmetric, Cij=None):
    lng1_calc = np.zeros(len(ref_data["temperature"]))
    lng2_calc = np.zeros(len(ref_data["temperature"]))

    temperature_old = None
    for idx, (temperature, x1) in enumerate(zip(ref_data["temperature"], ref_data["x1
↪"])):
        if temperature == temperature_old:
            pass
        else:
            tau_12, tau_21, alpha_12, alpha_21 = compute_interaction_parameters_
↪binaries(
                params, coef_fitted, coef_asymmetric, Cij, temperature
            )
            temperature_old = temperature

            lng1_calc[idx], lng2_calc[idx] = NRTL_lngamma_binaries(x1, tau_12, tau_21,
↪alpha_12, alpha_21)

    return lng1_calc, lng2_calc

def compute_interaction_parameters_binaries(params, coef_fitted, coef_asymmetric, Cij,
↪ temperature):
    idx = 0
    tau_12 = 0.0
    tau_21 = 0.0
    alpha_12 = 0.0
    alpha_21 = 0.0

    for key in coef_fitted.keys():
        if coef_fitted[key]:
            v1 = params[idx]
            if coef_asymmetric[key]:
                idx += 1
                v2 = params[idx]
            else:
                v2 = v1
            idx += 1

            if key == "A":
                tau_12 += v1
                tau_21 += v2
            elif key == "B":
                tau_12 += v1 / temperature
                tau_21 += v2 / temperature
            elif key == "E":
                tau_12 += v1 * np.log(temperature)
                tau_21 += v2 * np.log(temperature)
            elif key == "F":
                tau_12 += v1 * temperature
                tau_21 += v2 * temperature

```

(continues on next page)

(continued from previous page)

```

    if key == "C":
        alpha_12 += v1
        alpha_21 += v2
    elif key == "D":
        alpha_12 += v1 * (temperature - 273.15)
        alpha_21 += v2 * (temperature - 273.15)

    if key == "C" and not coef_fitted["C"]:
        if Cij is not None:
            alpha_12 += Cij
            alpha_21 += Cij
        else:
            print(f"Regression stopped: Default Cij parameter is not provided.")
            sys.exit()

    return tau_12, tau_21, alpha_12, alpha_21

def NRTL_lngamma_binaries(x1, tau_12, tau_21, alpha_12, alpha_21):
    x2 = 1.0 - x1
    G_12 = np.exp(-alpha_12 * tau_12)
    G_21 = np.exp(-alpha_21 * tau_21)

    # Calculate activity coefficients
    lng1 = x2**2 * (tau_21 * (G_21 / (x1 + x2 * G_21)) ** 2 + tau_12 * G_12 / (x2 +
↪x1 * G_12) ** 2)
    lng2 = x1**2 * (tau_12 * (G_12 / (x2 + x1 * G_12)) ** 2 + tau_21 * G_21 / (x1 +
↪x2 * G_21) ** 2)

    return lng1, lng2

def summarize_coefficients(opt_res, coef_fitted, coef_asymmetric, Cij, display=False):
    params = opt_res.x
    errors = opt_res.fun

    idx = 0
    summary = []
    for key in coef_fitted.keys():
        if coef_fitted[key]:
            v1 = params[idx]
            idx += 1
            if coef_asymmetric[key]:
                v2 = params[idx]
                idx += 1
                summary.append(f"${key}_{ij} = {v1:.4f},\ {key}_{ji} = {v2:.4f}$")
            else:
                v2 = p1
                summary.append(f"${key}_{ij} = {key}_{ji} = {v1:.4f}$")

        if display:
            print(f"{key}ij = {v1:.4f}; {key}ji = {v2:.4f}")
    else: # If the coefficient is not fitted
        if key == "C":
            summary.append(f"${key}_{ij} = {key}_{ji} is fixed to {Cij:.4f}$")
        if display:

```

(continues on next page)

(continued from previous page)

```

        print(f"{key}ij = {key}ji = {Cij:.4f}")
    else:
        summary.append(f"${key}_{ij} = {key}_{ji} \ =\ Not\ fitted$")

summary.append(f"objective = {errors:.4f}")

return "\n".join(summary)

def show_regression(
    opt_res, ref_data, coef_fitted, coef_asymmetric, Cij=None, figname="NRTL_
    ↪regression.png", save=True
):
    params = opt_res.x
    errors = opt_res.fun
    lng1_calc, lng2_calc = compute_gammas_binaries(params, ref_data, coef_fitted, ↪
    ↪coef_asymmetric, Cij)

    unique_temperature = np.unique(ref_data["temperature"])
    colormap = cm.get_cmap("viridis", len(unique_temperature)) # 'viridis' is just ↪
    ↪an example

    fig = plt.figure(figsize=(6, 10))
    ax1 = fig.add_subplot(311)
    ax2 = fig.add_subplot(312)
    ax3 = fig.add_subplot(313)

    for idx, temperature in enumerate(unique_temperature):
        used_mask = ref_data["temperature"] == temperature
        color = colormap(idx)

        ax1.plot(
            ref_data["x1"][used_mask],
            ref_data["lngamma1"][used_mask],
            "o",
            color=color,
            markerfacecolor="none",
            label=f"{ref_data['method']}-{str(temperature)}",
        )
        ax1.plot(ref_data["x1"][used_mask], lng1_calc[used_mask], "-", color=color, ↪
    ↪label=f"NRTL-{str(temperature)}")

        ax2.plot(
            ref_data["x1"][used_mask],
            ref_data["lngamma2"][used_mask],
            "o",
            color=color,
            markerfacecolor="none",
            label=f"{ref_data['method']}-{str(temperature)}",
        )
        ax2.plot(ref_data["x1"][used_mask], lng2_calc[used_mask], "-", color=color, ↪
    ↪label=f"NRTL-{str(temperature)}")

    ax1.set_xlabel(f"$x_{{1}}$")
    ax1.set_ylabel(f"$\ln(\gamma_{{1}})$")
    ax1.legend()
    ax2.set_xlabel(f"$x_{{1}}$")

```

(continues on next page)

(continued from previous page)

```

ax2.set_ylabel(f"$\ln(\gamma_{{2}})$")
ax2.legend()

summary = summarize_coefficients(opt_res, coef_fitted, coef_asymmetric, Cij)

ax3.text(0.5, 0.5, summary, ha="center", va="center", fontsize=12)
ax3.axis("off")

plt.suptitle(f"{ref_data['compounds'][0]}(1) and {ref_data['compounds'][1]}(2)")
plt.tight_layout()

if save:
    if ".png" != figname[-4::]:
        figname = figname + ".png"
    plt.savefig(f"{figname}")
plt.show()

def get_bounds(coef_fitted, coef_asymmetric, coef_bounds):
    bounds = []
    for key in coef_fitted.keys():
        if coef_fitted[key]:
            bounds.append(coef_bounds[key])
        if coef_asymmetric[key]:
            bounds.append(coef_bounds[key])

    return bounds

def get_initial_guess(bounds, seed=1):
    initial_guess = []

    np.random.seed(seed)

    for lower, upper in bounds:
        if lower is not None and upper is not None:
            random_number = random.uniform(lower, upper)
        else:
            random_number = random.random()

        initial_guess.append(random_number)

    return initial_guess

# create a COSKFDatabase
db = COSKFDatabase("my_db.db")
db.add_compound("Water.coskf")
db.add_compound("Methyl_acrylate.coskf")

# set up for parallel run
config.default_jobrunner = JobRunner(
    parallel=True, maxjobs=8
) # Set jobrunner to be parallel and specify the numbers of jobs run simultaneously
↪(eg. multiprocessing.cpu_count())
config.default_jobmanager.settings.hashing = None # Disable rerun prevention
config.job.runscript.nproc = 1 # Number of cores for each job

```

(continues on next page)

(continued from previous page)

```

config.log.stdout = 1 # suppress plams output default=3

# create a CRSystem for generating reference activity coefficient data
cal = CRSSystem()

mixture = {}
c1 = "Water"
c2 = "Methyl acrylate"

mixture[c1] = 0.0
mixture[c2] = 1.0

temperature = 303.15
for x in np.linspace(0, 1, 5):
    mixture[c1] = x
    mixture[c2] = 1.0 - x
    cal.add_Mixture(mixture=mixture, temperature=temperature, database="my_db.db")

# execute calculation
cal.runCRSJob()

# retrieve result and create the reference data
CRS_results = [out.get_results() for out in cal.outputs]

ref_data = {}
ref_data["compounds"] = np.array(CRS_results[0]["name"])
ref_data["temperature"] = np.array([res["temperature"] for res in CRS_results])
ref_data["x1"] = np.array([res["frac1"][0][0] for res in CRS_results])
ref_data["lngamma1"] = np.array([np.log(res["gamma"][0][0]) for res in CRS_results])
ref_data["lngamma2"] = np.array([np.log(res["gamma"][1][0]) for res in CRS_results])
ref_data["method"] = CRS_results[0]["method"]

# Specify the regression settings for the NRTL model
# Define whether to fit the coefficients in the equations for tau_ij and alpha_ij
# tau_ij = Aij + Bij / T + Eij * ln(T) + Fij * T
# alpha_ij = Cij + Dij * (T - 273.15)
coef_fitted = {"A": True, "B": False, "C": False, "D": False, "E": False, "F": False}
# Specify whether the coefficients are asymmetric (i.e., Aij ≠ Aji)
coef_asymmetric = {"A": True, "B": True, "C": False, "D": False, "E": True, "F": True}
# Set bounds for each coefficient during regression
coef_bounds = {
    "A": (-100, 100),
    "B": (-30000, 30000),
    "C": (0.0, 1.0),
    "D": (-0.02, 0.02),
    "E": (-10, 10),
    "F": (-10, 10),
}

# Default value for Cij, if required
Cij = 0.3
# Cij = None

# Create bounds and initial guess for the parameters used in the regression
bounds = get_bounds(coef_fitted, coef_asymmetric, coef_bounds)
initial_guess = get_initial_guess(bounds, seed=1)

```

(continues on next page)

(continued from previous page)

```

# regression using the method build-in scipy.optimize
opt_method = "L-BFGS-B"
opt_method = "differential_evolution"

if opt_method == "differential_evolution":
    opt_res = optimize.differential_evolution(
        objective, bounds=bounds, x0=initial_guess, args=(ref_data, coef_fitted, coef_
        ↪asymmetric, Cij), maxiter=1000
    )
else:
    opt_res = optimize.minimize(
        objective,
        method=opt_method,
        bounds=bounds,
        x0=initial_guess,
        args=(ref_data, coef_fitted, coef_asymmetric, Cij),
        options={"maxiter": 100},
    )

summarize_coefficients(opt_res, coef_fitted, coef_asymmetric, Cij, display=True)
figname = f"fig_NRTL_{opt_method}.png"
show_regression(opt_res, ref_data, coef_fitted, coef_asymmetric, Cij, figname)

```

## References

### Automated screening of ionic liquids

Download python script and relevant file

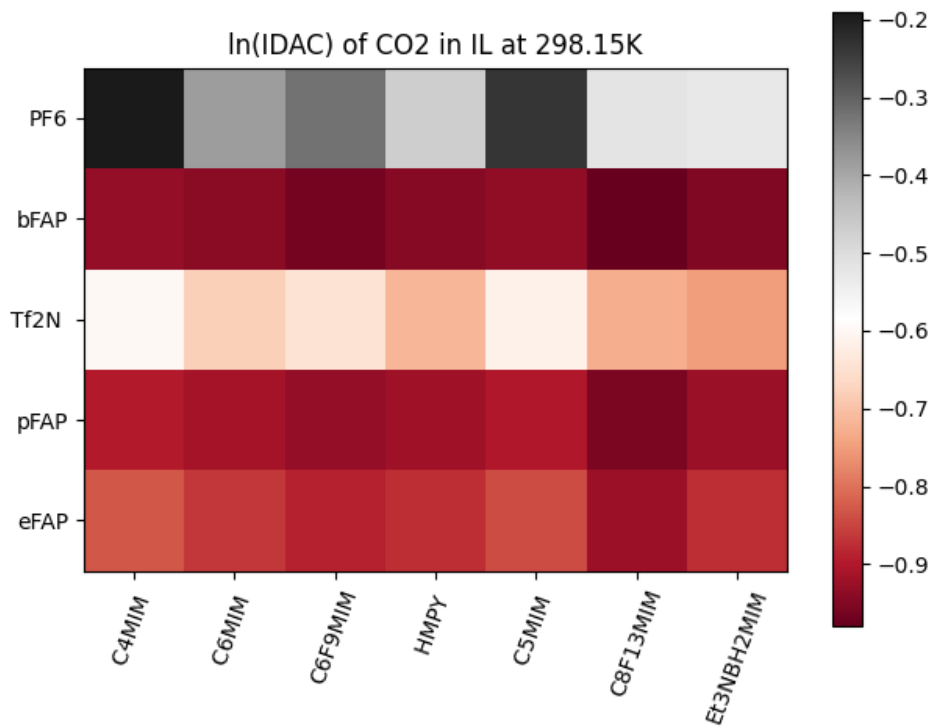
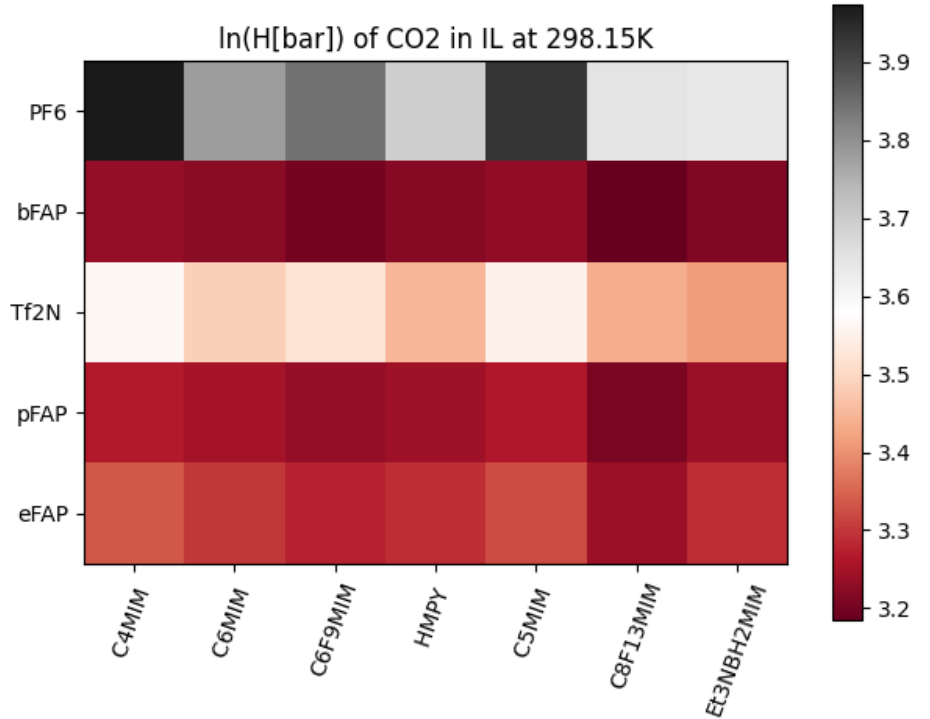
In this example, we provide a template for the high-throughput screening with ionic liquid. The script will automates the screening of the infinite dilution activity coefficient (IDAC) of a solute among all combinations of available ionic liquids in the directory. The *set\_CRSJob\_IL* in this script adopt the same structure as *set\_CRSJob\_Hex\_conf* in previous example but have an additional input variable, *comp\_stoichiometric*, to specify the stoichiometric number of cation and anion. The stoichiometric number will be determined from their charge, and it is crucial to ensure that the **IL\_list.csv** file contains the minimum required information for each ion, such as abbreviation, charge and coskf file name. Furthermore, if the vapor pressure of the solute (Pvap) is provided, you can calculate the Henry's constant and solute solubility at 1 bar under the assumption of ideal gas by setting *cal\_Henry=True*.

$$H(\text{bar}) = \text{IDAC} * P_{\text{vap}}$$

$$x(1\text{bar}) = 1\text{bar}/H(\text{bar})$$

This figure (produced by the code) illustrates the screening outcomes for carbon dioxide involving 7 cations and 5 anions which have proven to be useful tool for searching good candidates<sup>1</sup>.

<sup>1</sup> Zhang, Xiaochun, Zhiping Liu, and Wenchuan Wang. "Screening of ionic liquids to capture CO2 by COSMO-RS and experiments." *AIChE journal* 56.10 (2008): 5617-5628.



## Python code

```

import os, sys, time, glob
import multiprocessing
import numpy as np
import matplotlib.pyplot as plt
from scm.plams import Settings, init, finish, CRSJob, config, JobRunner
import pandas as pd
from math import gcd

##### Note: Ensure to download the coskf_IL before running the script #####

"""
The Python script automates the screening of the infinite dilution activity_
↳coefficient (IDAC)
of a solute among all combinations of available ionic liquids in the corresponding_
↳directory.

To run the script, ensure that the coskf file of ions and solute is stored in the_
↳database_path
and the coskf file of ions follows a specific naming convention, such as IL_*cation*_
↳coskf or IL_*anion*_coskf.
For example, you should have files like IL_cation_1-butyl-3-methyl-imidazolium.coskf,
IL_anion_hexafluorophosphate.coskf and Carbon_dioxide.coskf in ./coskf-IL/

Additionally, you need to provide the ion's information in the IL_list.csv file as_
↳input including:
    name, [abbreviation], type, charge, coskf, smiles(optional)
    1-butyl-3-methyl-imidazolium, [C4MIM ; C4C1im ; BMIM], cation, 1.0, IL_cation_1-
↳butyl-3-methyl-imidazolium.coskf, CCCN1cc[n+](C)c1
The IL_list.csv file contain the information of 80 cations and 56 anions in the_
↳ADFCRS-IL-2014 database.

Furthermore, if the vapor pressure of the solute (Pvap) is provided, you can_
↳calculate the Henry's constant and
solute solubility at 1 bar under the assumption of ideal gas by setting cal_
↳Henry=True.
    H(bar) = IDAC*Pvap
    x(1bar)= 1bar/H

The parallel calculation can be enabled by setting the Parallel_run=True and_
↳maxjobs=numbers_of_processes.

The calculated result will be saved in a pandas dataframe (df) and a csv file (result_
↳csv='IL_screening.csv').

The visualization of the result using a contour plot can be enable by setting Plot_
↳option=True.

AMS2026 and later include the Pandas library by default. On older AMS versions,_
↳install it through AMSpackages.
    "${AMSBIN}/amspackages" install pandas
"""

init()

```

(continues on next page)

(continued from previous page)

```

def set_CRSSJob_IL(
    index,
    ncomp,
    coskf,
    database_path,
    cal_type,
    method,
    temperature,
    comp_input={},
    comp_type=[],
    comp_stoichiometric=[],
):
    s = Settings() # initialize a settings object
    s.input.property._h = cal_type # specify problem type
    s.input.method = method # specify method
    s.input.temperature = temperature # specify temperature

    compounds = [Settings() for i in range(ncomp)] # initialization of compounds
    for i in range(ncomp):
        if len(comp_type) > i:
            if "conf" in comp_type[i]:
                form = [Settings() for i in range(len(coskf[i]))] # initialize_
↪compound in multiple form
                for j in range(len(coskf[i])):
                    form[j]._h = os.path.join(
                        database_path, coskf[i][j]
                    ) # specify conformer's coskf file for each form
                compounds[i].form = form
            elif "IL" in comp_type[i]:
                form = [Settings() for j in range(1)] # initialize IL in one form:_
↪dissociated IL
                species = [Settings() for j in range(2)] # initialize dissociated IL_
↪in two species: cation and anion
                for j in range(2):
                    struct = [Settings() for k in range(1)] # initialize cation and_
↪anion in one structure
                    struct[0]._h = os.path.join(database_path, coskf[i][j])
                    struct[0].count = comp_stoichiometric[i][j] # specify the_
↪Stoichiometric number of cation and anion
                    species[j] = struct
                form[0].species = species
                compounds[i].form = form
            else:
                compounds[i]._h = os.path.join(database_path, coskf[i]) # specify_
↪absolute directory of coskf file
            else:
                compounds[i]._h = os.path.join(database_path, coskf[i]) # specify_
↪absolute directory of coskf file

            for column, value in comp_input.items(): # specify compound's information_
↪through comp_input, for example
                if value[i] != None: # column: frac1, meltingpoint, hfusion
                    compounds[i][column] = value[i]

    s.input.compound = compounds

```

(continues on next page)

(continued from previous page)

```

my_job = CRSJob(settings=s) # create jobs
my_job.name = cal_type + "_" + str(index) # specify job name

return my_job

Parallel_run = True # If True -> parallel calculation
cal_Henry = True # If True -> calculation of Henry's constant using IDAC*Pvap
Plot_option = True # If True -> visualization of the results with a contour plot

if Parallel_run:
    config.default_jobrunner = JobRunner(
        parallel=True, maxjobs=8
    ) # Set jobrunner to be parallel and specify the numbers of jobs run_
    ↪simultaneously (eg. multiprocessing.cpu_count())
    config.default_jobmanager.settings.hashing = None # Disable rerun prevention
    config.job.runscript.nproc = 1 # Number of cores for each job
    config.log.stdout = 1 # suppress plams output default=3

###----INPUT YOUR DATA HERE----###

database_path = os.path.join(os.getcwd(), "coskf_IL")
cal_type = "activitycoef"
method = "COSMORS"
ncomp = 2
solute = "Carbon_dioxide.coskf"
solute_abb = "CO2"
temperature = 298.15

# Retrive all coskf file for cation and anion in database_path. Note the file must be_
↪named as: IL_*cation*.coskf and IL_*anion*.coskf
cations_coskf = [os.path.basename(x) for x in glob.glob(os.path.join(database_path,
↪"IL_*cation*.coskf"))]
anions_coskf = [os.path.basename(x) for x in glob.glob(os.path.join(database_path,
↪"IL_*anion*.coskf"))]

# store input data along with the necessary thermal property in comp_input dictionary
comp_input = {}
comp_input["frac1"] = [1.0, 0.0]

# enter 'IL' for ionic liquid; 'conf' for compound with multiple conformers; '' for_
↪compound with single structure
comp_type = ["IL", ""]

# Read ion's information from IL_list.csv where contain name, [abbreviation], type,_
↪charge, coskf, smiles(optional)
df_IL = pd.read_csv("IL_list.csv")

# Write the screening result
result_csv = "result_IL_screening.csv"

###----INPUT END----###

index = 0
outputs = []
df = pd.DataFrame()
for cation in cations_coskf:

```

(continues on next page)

(continued from previous page)

```

for anion in anions_coskf:

    cur_df = df_IL.loc[df_IL["coskf"] == cation]
    cation_abb = cur_df["abbreviation"].values[0].split(";")[0].rstrip() #_
    ↪abbreviation
    cation_charge = int(cur_df["charge"].values[0]) # charge

    cur_df = df_IL.loc[df_IL["coskf"] == anion]
    anion_abb = cur_df["abbreviation"].values[0].split(";")[0].rstrip() #_
    ↪abbreviation
    anion_charge = int(cur_df["charge"].values[0]) # charge

    # find the Least Common Multiple of cation charge and anion charge
    IL_lcm = cation_charge * anion_charge / gcd(cation_charge, anion_charge)
    cation_v = -IL_lcm / cation_charge # stoichiometric number of cation
    anion_v = IL_lcm / anion_charge # stoichiometric number of anion

    IL_abb = [cation_abb, anion_abb] # abbreviation
    IL_coskf = [cation, anion]
    IL_v = [cation_v, anion_v] # stoichiometric number

    coskf = [IL_coskf, solute]
    comp_stoichiometric = [IL_v, 1.0] # stoichiometric number used for multi-
    ↪species

    comp_input["name"] = [cation_abb + "_" + anion_abb, solute_abb]

    df.loc[index, "cation"] = cation_abb
    df.loc[index, "anion"] = anion_abb
    df.loc[index, "solute"] = solute.replace(".coskf", "")
    df.loc[index, "charge_c"] = int(cation_charge)
    df.loc[index, "charge_a"] = int(anion_charge)

    job = set_CRSSJob_IL(
        index,
        ncomp,
        coskf,
        database_path,
        cal_type,
        method,
        temperature,
        comp_input,
        comp_type,
        comp_stoichiometric,
    )
    outputs.append(job.run())

    index = index + 1

# In a parallel run, the get_results function will wait for the completion of the_
    ↪corresponding jobs.
results = []
for index, out in enumerate(outputs):
    res = out.get_results()
    results.append(res)
    df.loc[index, "IDAC"] = res["gamma"][-1][0]

```

(continues on next page)

(continued from previous page)

```

if cal_Henry:
    if solute == "Carbon_dioxide.coskf":
        Pvap = np.power(10, 6.35537 - 2067.0 / (temperature + 156.462))
        # antonie equation(unit in bar), parameters are fitted by SCM in temperature.
        ↪range 260-305K
    else:
        Pvap = np.nan
        print("The vapor pressure of the solute is not defined")
    if not np.isnan(Pvap):
        df["H(bar)"] = df["IDAC"] * Pvap
        df["x(1bar)"] = 1 / (df["IDAC"] * Pvap)

df.to_csv(result_csv, index=None)

if Plot_option:
    # contour visulization
    nx = len(cations_coskf)
    ny = len(anions_coskf)

    # Extract the 1st abbreviation of the ions. For instance, [C4MIM ; C4Clim ; BMIM].
    ↪-> C4MIM
    cation_name = [df_IL.loc[df_IL["coskf"] == x]["abbreviation"].values[0].split(";
    ↪") [0] for x in cations_coskf]
    anion_name = [df_IL.loc[df_IL["coskf"] == x]["abbreviation"].values[0].split(";
    ↪") [0] for x in anions_coskf]

    x = [i for i in range(nx)]
    y = [i for i in range(ny)]

    if cal_Henry and not np.isnan(Pvap):
        cal_data = df["H(bar)"].values
        sub_title = "ln(H[bar]) of " + solute_abb + " in IL at " + str(temperature) +
        ↪"K"
        fig_title = "IL_screening_lnH.png"
    else:
        cal_data = df["IDAC"].values
        sub_title = "ln(IDAC) of " + solute_abb.replace("_", " ") + " in IL at " +
        ↪str(temperature) + "K"
        fig_title = "IL_screening_lnIDAC.png"

    plt_data = np.zeros((ny, nx))
    for i in range(ny):
        for j in range(nx):
            plt_data[i][j] = np.log(cal_data[j * ny + i])

    fig, ax = plt.subplots()

    plt.imshow(plt_data, cmap="RdGy", interpolation="nearest")
    if len(x) > 10:
        x = [0 + 5 * n for n in range((len(x) // 5) + 1)]
        plt.xticks(x, x, rotation=70)
    else:
        plt.xticks(x, cation_name, rotation=70)
    if len(y) > 10:
        y = [0 + 5 * n for n in range((len(y) // 5) + 1)]

```

(continues on next page)

(continued from previous page)

```

        plt.yticks(y, y)
    else:
        plt.yticks(y, anion_name)
plt.colorbar()
plt.title(sub_title)
plt.tight_layout()
# plt.savefig(fig_title)
plt.show()

finish()

```

## References

### Calculating and estimating sigma profiles

Sigma profiles are one of the fundamental pieces of a COSMO-RS/-SAC calculation. They are also widely used as an important empirical descriptor for a molecule's behavior in a solution as well as for a molecule's properties in a number of applications. In the standard COSMO-RS/-SAC workflow, sigma profiles are generated after a sequence of DFT calculations which – for large molecular systems – can take considerable time to complete. For computationally expensive systems or high-throughput screening applications, it is sometimes advantageous to approximate sigma profiles using tools like `fast_sigma` from AMS.

In the following python script, we generate sigma profiles for n-Hexanoic acid using the two approaches discussed above. The function `calc_sigma_profile` will generate sigma profiles from `.coskf` files, and the function `fast_sigma` will generate sigma profiles from SMILES strings using the `fast_sigma` tool.

### Python code

```

import os
import numpy as np
import matplotlib.pyplot as plt

from scm.utils.runsubprocess import RunSubprocess
from scm.plams import Settings, init, finish, CRSJob, config, KFFile
import subprocess

##### Note: Ensure to configure the database path to either the installed ADFCRS-
↪2018 directory or your own specified directory #####

database_path = os.path.join(os.environ["SCM_PKG_ADFCRSDIR"], "ADFCRS-2018")
if not os.path.exists(database_path):
    raise OSError(f"The provided path does not exist. Exiting.")

init()
# suppress plams output
config.log.stdout = 0

class SigmaProfile:
    def __init__(self, chdens, profiles, profile_names):
        if len(profiles) != len(profile_names):
            print("Error: profiles_names and profiles of different sizes")

```

(continues on next page)

(continued from previous page)

```

self.chdens = chdens.flatten() if isinstance(chdens, np.ndarray) else chdens
self.profiles = {
    name: prof.flatten() if isinstance(prof, np.ndarray) else prof
    for name, prof in zip(profile_names, profiles)
}

def __str__(self):
    line = "-" * (15 * (1 + len(self.profiles)))
    ret = (
        line
        + "\n"
        + "".join(["Charge Dens.".ljust(15)] + [name.ljust(15) for name in self.
↪profiles])
        + "\n"
        + line
        + "\n"
    )
    for i in range(len(self.chdens)):
        ret += "{0:.5g}".format(self.chdens[i]).ljust(15) + "".join(
            ["{0:.5g}".format(v[i]).ljust(15) for k, v in self.profiles.items()])
        ret += "\n"
    return ret

def fast_sigma(smiles):

    results_file = "tmp_results18954.compkf"
    subprocess_string = " --smiles '" + smiles + "'"

    if not os.path.isfile(os.path.join(os.path.expandvars("$AMSBIN"), "fast_sigma")):
        raise OSError("ERROR: cannot find fast_sigma ... has amsbashrc been executed?
↪")

    fs = os.path.join(os.path.expandvars("$AMSBIN"), "fast_sigma")
    scm_sp = RunSubprocess(fs + subprocess_string + " -o " + results_file)

    if os.path.isfile(results_file):
        crskf = KFFile(results_file)
        res = crskf.read_section("PURESIGMAPROFILE")
        sp = SigmaProfile(
            chdens=res["chdval"],
            profiles=[res["profil"], res["hbprofil"]],
            profile_names=["total_profile", "HB_profile"],
        )
        os.remove(results_file)
        return sp, scm_sp
    else:
        return None, scm_sp

def calc_sigma_profile(coskf_file, cosmosac=False):

    # initialize settings object
    settings = Settings()
    settings.input.property._h = "SIGMAPROFILE"

```

(continues on next page)

(continued from previous page)

```

# set the number of compounds
compounds = [Settings()]
compounds[0]._h = os.path.join(database_path, coskf_file)
compounds[0].frac1 = 1

# to change to the COSMOSAC2013 method
if cosmosac:
    settings.input.method = "COSMOSAC2013"

# specify the compounds as the compounds to be used in the calculation
settings.input.compound = compounds
# create a job that can be run by COSMO-RS
my_job = CRSJob(settings=settings)

out = my_job.run()
res = out.get_results()

if cosmosac:
    prof_len = len(res["hbprofil"]) // 3
    sp = SigmaProfile(
        chdens=res["chdval"],
        profiles=[res["profil"]] + [res["hbprofil"][i * prof_len : (i + 1) * prof_
↪len] for i in range(3)],
        profile_names=["total_profile", "HB", "HB-OH", "HB-OT"],
    )
else:
    sp = SigmaProfile(
        chdens=res["chdval"],
        profiles=[res["profil"], res["hbprofil"]],
        profile_names=["total_profile", "HB_profile"],
    )
return sp

# regular way to generate a sigma profile from a .coskf file
filename = "n-Hexanoic_acid.coskf"
sp = calc_sigma_profile(filename, cosmosac=False)

# way using the fast_sigma estimation method
fs_sp, err = fast_sigma("CCCCCC(=O)O")
if fs_sp is None or len(err[1]) > 0:
    print("fast_sigma generated the following output:\n" + err[1])

plt.xlabel("σ value (e/Å²)")
plt.ylabel("p(σ)")

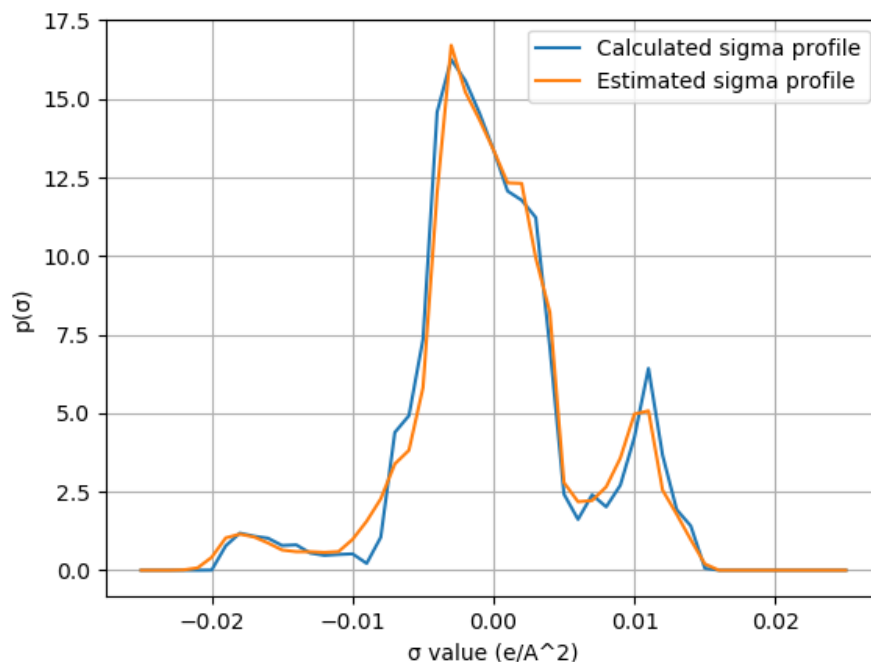
plt.plot(sp.chdens, sp.profiles["total_profile"], label="Calculated sigma profile")
if fs_sp is not None:
    plt.plot(fs_sp.chdens, fs_sp.profiles["total_profile"], label="Estimated sigma_
↪profile")

plt.legend(loc="upper right")
plt.grid()
# plt.savefig(f"./Sigma_profile.png")
plt.show()

finish()

```

This code produces the following output:



## Sigma Moments

$\sigma$  moment are useful chemical descriptors derived from the  $\sigma$ -profile. They are analogous to moments of a statistical distribution and can be thought of as a way to reduce the high-dimensional information present in a  $\sigma$ -profile to a smaller number of descriptors that characterize that  $\sigma$ -profile.  $\sigma$ -moments are known to be valuable descriptors in QSPR and are thought to represent the solvent space well<sup>1</sup>.

$\sigma$  moment can be calculated by the following equation through the  $\sigma$ -profile,  $P(\sigma)$ , and the  $\sigma^{hb}$ -profile,  $P^{HB}(\sigma)$ , of a compound. The zero-order  $\sigma$  moment ( $MOM_0$ ) is the molecular area of a compound. The first-order  $\sigma$  moment ( $MOM_1$ ) is the negative of total charge of a compound. The second-order sigma moment ( $MOM_2$ ) represents the polarity of a compound. The third-order  $\sigma$  moment ( $MOM_3$ ) represents the asymmetry of the  $\sigma$ -profile of a compound. The other  $\sigma$  moment,  $MOM_4$ ,  $MOM_5$  and  $MOM_6$ , do not have well-established physical interpretations. Last, the  $MOM_{acc\_l}^{hb}$  and  $MOM_{don\_l}^{hb}$  quantify the strength of a molecule's ability to form hydrogen-bond interactions as an acceptor and a donor, respectively. The different values of  $l$  correspond to varying threshold values,  $\sigma_{cutoff\_l}$ , used in the equation.

$$MOM_i = \int P(\sigma) \sigma^i d\sigma \quad \text{where } i = 0, 1, 2, 3, 4, 5, 6$$

$$MOM_{acc\_l}^{hb} = \int P^{HB}(\sigma) \max(0, +\sigma - \sigma_{cutoff\_l}) \quad \text{where } l = 1, 2, 3, 4$$

$$MOM_{don\_l}^{hb} = \int P^{HB}(\sigma) \max(0, -\sigma - \sigma_{cutoff\_l}) \quad \text{where } l = 1, 2, 3, 4$$

when  $l = 1$ ,  $\sigma_{cutoff\_l} = \sigma_{cutoff}^{HB} = 0.00854$  (default)

when  $l = 2, 3, 4$ ,  $\sigma_{cutoff\_l} = 0.006 + 0.002 * l$

<sup>1</sup> A. Klamt, *COSMO-RS From Quantum Chemistry to Fluid Phase Thermodynamics and Drug Design*, Elsevier. Amsterdam (2005), ISBN 0-444-51994-7.

where the unit are  $e^2$  and  $e/2$  for  $P(\sigma)$  and  $\sigma$  respectively.

The following script will calculate the  $\sigma$  moment for few common compounds. By default, it computes five  $\sigma$  moment ( $MOM_0$ ,  $MOM_2$ ,  $MOM_3$ ,  $MOM_{acc,2}^{hb}$ ,  $MOM_{don,2}^{hb}$ ). To generate 15  $\sigma$  moment, you can specify the `moment_power` and `hb_moment_level` parameters in the `calculate_moments` function, as demonstrated in the script.

## Python code

```
import os
import numpy as np
from scm.plams import *
import matplotlib.pyplot as plt
from typing import List, Optional, Dict

class SigmaMoments:
    def __init__(self, filenames: List[str], database_path: str, hb_cutoff: float = 0.00854):
        """
        Initialize the SigmaMoments class and prepare the sigma profile for
        subsequent sigma moment calculations.

        Args:
            filenames : List[str]
                A list of `.coskf` files to be used for calculating sigma moments.
            database_path : str
                The directory path where the `.coskf` files are stored.
            hb_cutoff : float, optional
                The hydrogen bond cutoff value for the COSMO-RS model.
                Default is 0.00854 e/Å2.
        """
        self.filenames = filenames
        self.database_path = os.path.abspath(database_path)
        self.hb_cutoff = hb_cutoff
        self._calc_profiles_and_chdens()

    def calculate_moments(
        self, moment_power: Optional[List[int]] = None, hb_moment_level:
Optional[List[int]] = None
    ) -> dict:
        """
        Calculate the sigma moment.

        Args:
            moment_power : List[int], optional
                A list of integers specifying the powers for standard moment
                calculations.
                Default is [0, 2, 3]. The recommended maximum power is 6.
            hb_moment_level : List[int], optional
                A list of integers specifying the hydrogen bond moment levels to be
                calculated.
                For the 1st level, the `hb_cutoff` corresponds to the COSMO-RS
                parameter, defaulting to 0.00854 e/Å2.
                For the 2nd, 3rd, and 4th levels, the `hb_cutoff` is defined as 0.006
        """
```

(continues on next page)

(continued from previous page)

```

↪+ 0.002 * 1 e/Å2, where 1 represents the level (2, 3, or 4).
    """
    if moment_power is None:
        moment_power = [0, 2, 3]
    if hb_moment_level is None:
        hb_moment_level = [2]

    self.moments = {}
    self._calc_standard_moments(moment_power)
    self._calc_hb_moments(hb_moment_level)
    return self.moments

def __repr__(self):
    max_mom_len = max([len(m) for m in self.moments])
    lens = [len(fn) for fn in self.filenames]

    text = ""
    text += (" " * 5).join(["Moment".ljust(max_mom_len)] + self.filenames)
    text += "\n"
    for mom_name in self.moments:
        text += (" " * 5).join(
            [mom_name.ljust(max_mom_len)
             + [{"0:.5g}".format(m)].rjust(1) for m, l in zip(self.moments[mom_
↪name], lens)]
        )
        text += "\n"

    return f"{text}"

def to_dataframe(self):
    """
    Convert the calculated result into pandas DataFrame.
    """
    try:
        import pandas as pd
    except ImportError:
        print(
            "Pandas is included by default in AMS2026 and later. On older AMS_
↪versions, install it through AMSpackages via the GUI or with:\namspackages install_
↪pandas"
        )
        return None

    df = pd.DataFrame(self.moments)
    df.insert(0, "filename", self.filenames)

    return df

def _calc_profiles_and_chdens(self):
    # initialize settings object
    settings = Settings()
    settings.input.property._h = "PURESIGMAPROFILE"
    # set the cutoff value for h-bonding
    settings.input.CRSPParameters.sigmahbond = self.hb_cutoff
    compounds = [Settings() for i in range(len(self.filenames))]
    for i, filename in enumerate(self.filenames):
        compounds[i]._h = os.path.join(self.database_path, filename)

```

(continues on next page)

(continued from previous page)

```

settings.input.compound = compounds
# create a job that can be run by COSMO-RS
my_job = CRSJob(settings=settings)
# run the job
out = my_job.run()
# convert all the results into a python dict
res = out.get_results()
# retain profiles and charge density values
self.tot_profiles = res["profil"]
self.hb_profiles = res["hbprofil"]
self.chdens = res["chdval"]

def _calc_standard_moments(self, moment_power: Optional[List[int]] = None):
    if moment_power is None:
        moment_power = [0, 2, 3]
    for i in moment_power:
        tmp_moms = []
        for prof in self.tot_profiles:
            tmp_moms.append(np.sum(prof * np.power(self.chdens, i)))
        self.moments["MOM" + str(i)] = tmp_moms

def _calc_hb_moments(self, hb_moment_level: Optional[List[int]] = None):
    if hb_moment_level is None:
        hb_moment_level = [2]

    for l in hb_moment_level:
        self.moments["MOM_hb_acc" + str(l)] = []
    for l in hb_moment_level:
        self.moments["MOM_hb_don" + str(l)] = []

    zeros = np.zeros(len(self.chdens))
    for prof in self.hb_profiles:
        for l in hb_moment_level:
            if l == 1:
                hb_cutoff = self.hb_cutoff
            else:
                hb_cutoff = 0.006 + 0.002 * l

            self.moments["MOM_hb_acc" + str(l)].append(np.sum(prof * np.
↪maximum(zeros, self.chdens - hb_cutoff)))
            self.moments["MOM_hb_don" + str(l)].append(np.sum(prof * np.
↪maximum(zeros, -self.chdens - hb_cutoff)))

    def plot_sigmaprofile(self, compound_idx: int, show: bool = True, save: bool =_
↪False, dir_path: str = "."):
        """
        Plots the sigma profile for a specific compound.

        Args:
            compound_idx: int
                The index of the compound to visualize its sigma profile.
            show: bool, optional
                If True, displays the sigma profile plot. Default is True.
            save: bool, optional
                If True, saves the sigma profile plot to a file. Default is False.
            dir_path: str, optional

```

(continues on next page)

(continued from previous page)

```

        The directory path where the sigma profile plot will be saved, if
↪ `save` is True. Default is the current working directory.
    """
    chdens = self.chdens
    tot_profiles = self.tot_profiles[compound_idx]
    hb_profiles = self.hb_profiles[compound_idx]
    name = self.filenames[compound_idx]

    fig = plt.figure(figsize=(5, 6))
    ax1 = fig.add_subplot(211)
    ax2 = fig.add_subplot(212)

    ax1.plot(chdens, tot_profiles, label="total", color="blue")
    ax1.set_xlabel("$\u03C3(e/\u212B^{{2}})$")
    ax1.set_ylabel("P(\u03C3) $(\u212B^{{2}})$")
    ax1.set_title("$\u03C3$ profile (total)")
    ax1.set_xlim([-0.025, 0.025])
    ax1.legend()

    ax2.plot(chdens, hb_profiles, label="hb", color="red")
    ax2.set_xlabel("charge density $(e/\u212B^{{2}})$")
    ax2.set_ylabel("$\u03C3(e/\u212B^{{2}})$")
    ax2.set_ylabel("P(\u03C3) $(\u212B^{{2}})$")
    ax2.set_title("$\u03C3$ profile (hydrogen bond)")
    ax2.set_xlim([-0.025, 0.025])

    ax2.legend()
    fig.suptitle(f"{name}")
    fig.tight_layout()

    if save:
        dir_path = os.path.abspath(dir_path)
        if not os.path.exists(dir_path):
            os.makedirs(dir_path)
        figname = f"SP_{name.replace('.coskf', '')}.png"
        fig.savefig(os.path.join(dir_path, figname))

    if show:
        plt.show()

##### Note: Be sure to add the path to your own AMSCRS directory here #
↪#####
database_path = os.path.join(os.environ["SCM_PKG_ADFCRSDIR"], "ADFCRS-2018")
if not os.path.exists(database_path):
    raise OSError(f"The provided path does not exist. Exiting.")

init()
config.log.stdout = 0 # suppress plams output

# Define the files we want to use to calculate sigma moments
filenames = ["Water.coskf", "Hexane.coskf", "Ethanol.coskf", "Acetone.coskf", "Acetic_
↪acid.coskf"]

# Initialize SigmaMoments with the specified filenames and database path
sm = SigmaMoments(filenames, database_path)

```

(continues on next page)

(continued from previous page)

```

# Calculate sigma moments with default settings:
# - moment_power: [0, 2, 3]
# - hb_moment_level: [2]
sm.calculate_moments()
print(f"Default 5 sigma moments:\n{sm}")

# Calculate all 15 sigma moments by specifying:
# - moment_power: [0, 1, 2, 3, 4, 5, 6]
# - hb_moment_level: [1, 2, 3, 4]
sm.calculate_moments(moment_power=[0, 1, 2, 3, 4, 5, 6], hb_moment_level=[1, 2, 3, 4])
print(f"All 15 sigma moments:\n{sm}")

# Convert the sigma moments data to a DataFrame for analysis
# df = sm.to_dataframe()

# Visualize and save the sigma profile for the compound at index 0
# sm.plot_sigmaprofile(compound_idx=0, show=True, save=True, dir_path="./fig")

finish()

```

The output produced is the following

```

Default 5 sigma moments:
Moment      Water.coskf      Hexane.coskf      Ethanol.coskf      Acetone.coskf      ↵
↵Acetic_acid.coskf
MOM0          43.011          160.38          90.019          103.28      ↵
↵          94.302
MOM2          0.0062556          0.001061          0.0046302          0.004566      ↵
↵          0.0062206
MOM3          -3.8253e-07          1.1557e-07          1.5947e-05          2.883e-05      ↵
↵          -4.4743e-06
MOM_hb_acc2    0.056607          0          0.048228          0.044109      ↵
↵          0.027105
MOM_hb_don2    0.056566          0          0.024696          0          ↵
↵          0.042326

All 15 sigma moments:
Moment      Water.coskf      Hexane.coskf      Ethanol.coskf      Acetone.coskf      ↵
↵Acetic_acid.coskf
MOM0          43.011          160.38          90.019          103.28      ↵
↵          94.302
MOM1          0.00026666          0.002145          0.00089555          0.0011418      ↵
↵          0.00041249
MOM2          0.0062556          0.001061          0.0046302          0.004566      ↵
↵          0.0062206
MOM3          -3.8253e-07          1.1557e-07          1.5947e-05          2.883e-05      ↵
↵          -4.4743e-06
MOM4          1.2722e-06          1.3058e-08          8.23e-07          5.6827e-07      ↵
↵          9.6505e-07
MOM5          -1.5315e-10          2.6449e-12          4.1365e-09          6.4726e-09      ↵
↵          -4.6722e-09
MOM6          2.8941e-10          1.997e-13          1.8283e-10          9.9675e-11      ↵
↵          2.1072e-10
MOM_hb_acc1    0.078179          0          0.06562          0.068401      ↵
↵          0.055839
MOM_hb_acc2    0.056607          0          0.048228          0.044109      ↵
↵          0.027105

```

(continues on next page)

(continued from previous page)

MOM_hb_acc3	0.03146	0	0.027298	0.018472	↵
↵	0.0066449				
MOM_hb_acc4	0.012196	0	0.011404	0.0056525	↵
↵	0.00031588				
MOM_hb_don1	0.078272	0	0.034516	0	↵
↵	0.053599				
MOM_hb_don2	0.056566	0	0.024696	0	↵
↵	0.042326				
MOM_hb_don3	0.031173	0	0.01304	0	↵
↵	0.028124				
MOM_hb_don4	0.01279	0	0.0045683	0	↵
↵	0.015962				

## References

### Eutectic systems

A eutectic point of a chemical mixture defines the minimum melting composition of that system over the composition range. In other words, the eutectic point will have a lower melting point than the pure components making up the mixture as well any other possible mixture. In this example, we calculate the eutectic point of a binary mixture of ethanol and water as the intersection of the solid-liquid equilibrium curves of two systems: (1) Solid ethanol dissolved in water and (2) Solid water dissolved in ethanol. This script will output the mole fraction of ethanol at the eutectic point as well as the temperature. For comparison, Takaizumi and Wakabayashi<sup>1</sup> provide an experimental eutectic point with a mole fraction value of 0.86 for ethanol and a melting temperature of -124.3 °C.

### Python code

```
import os
import matplotlib.pyplot as plt
from scm.plams import Settings, init, finish, CRSJob, config

##### Note: Ensure to configure the database path to either the installed ADFCRS-
↵2018 directory or your own specified directory #####

database_path = os.path.join(os.environ["SCM_PKG_ADFCRSDIR"], "ADFCRS-2018")
if not os.path.exists(database_path):
    raise OSError(f"The provided path does not exist. Exiting.")

init()
# suppress plams output
config.log.stdout = 0

# the ethanol water system
# experimental numbers for this eutectic are 0.86 mole fraction ethanol at a
↵temperature of -124.3 degrees C
files = ["Ethanol.coskf", "Water.coskf"]
tm = [158.5, 273.15] # K
hfus = [1.2, 1.43] # kcal/mol
initial_t_range = [100, 300] # K -- the temperature range over which the eutectic
```

(continues on next page)

<sup>1</sup> Takaizumi, K., and T. Wakabayashi. "The freezing process in methanol-, ethanol-, and propanol-water systems as revealed by differential scanning calorimetry." Journal of solution chemistry 26.10 (1997): 927-939.

(continued from previous page)

```

↪search is done
steps = 20 # number of steps to take within the temperature range

# another eutectic system
# files = ["L-Menthol.coskf","Camphor.coskf"]
# tm     = [316.2,451.5] #K
# hfus  = [2.84,1.63] #kcal/mol
# initial_t_range = [100,460] #K
# steps          = 20

# if we know the eutectic temperature is bounded to within a range of <= estimate_
↪precision, we simply use a linear interpolation between two x,T pairs
estimate_precision = 1.0 # K

class Eutectic:

    def __init__(self, files, tm, hfus, steps=10):
        self.files = files
        self.tm = tm
        self.hfus = hfus
        self.steps = steps

        if not (len(self.files) == len(self.tm) == len(self.hfus)):
            print("Error. Inputs must be the same length.")

    def calc_xt_curves(self, t_range):

        # initialize settings object
        settings = Settings()
        settings.input.property._h = "SOLUBILITY"
        # optionally, change to the COSMOSAC2013 method
        settings.input.method = "COSMOSAC2013"

        # make compounds
        compounds = [Settings() for i in range(len(files))]
        for i, file in enumerate(self.files):
            compounds[i]._h = os.path.join(database_path, file)
            compounds[i].meltingpoint = self.tm[i]
            compounds[i].hfusion = self.hfus[i]

        comp1_fractions = []
        for i, frac1 in enumerate([0.0, 1.0]):
            compounds[0].frac1 = frac1
            compounds[1].frac1 = 1.0 - frac1

        settings.input.temperature = " ".join([str(t) for t in t_range] +
↪[str(self.steps)])

        # add the compounds to the settings object
        settings.input.compound = compounds
        # create a job that can be run by COSMO-RS
        my_job = CRSJob(settings=settings)
        # run the job
        out = my_job.run()
        # convert all the results into a python dict
        res = out.get_results()

```

(continues on next page)

(continued from previous page)

```

        if i == 0:
            comp1_fractions.append(res["molar fraction"][0])
        else:
            comp1_fractions.append(1.0 - res["molar fraction"][1])

    return comp1_fractions

def calc_eutectic(self, t_range, history=[[[]], [], []]):

    comp1_fractions = self.calc_xt_curves(t_range)

    # the temperatures used in the calculation
    temps = [t_range[0] + (t_range[1] - t_range[0]) / self.steps * i for i in
↳range(self.steps + 1)]

    history[0].extend(comp1_fractions[0])
    history[1].extend(comp1_fractions[1])
    history[2].extend(temps)
    # the difference between compound 1's mole fraction in the two calculations
    # when these mole fractions are the same, we've found the eutectic
    diffs = comp1_fractions[0] - comp1_fractions[1]
    # find where the sign changes (intersection of SLE lines)
    for i in range(self.steps):
        if diffs[i] * diffs[i + 1] < 0:

            if temps[i + 1] - temps[i] < estimate_precision:
                # use linear combination of t's
                tot = abs(diffs[i]) + abs(diffs[i + 1])
                w1 = tot - abs(diffs[i]) # same as abs(diffs[i+1])
                w2 = tot - abs(diffs[i + 1])
                return (
                    (
                        (w1 * comp1_fractions[0][i] + w2 * comp1_fractions[1][i + 1]) /
↳tot,
                        (w1 * temps[i] + w2 * temps[i + 1]) / tot,
                    )
                ), history
            else:
                return self.calc_eutectic([temps[i], temps[i + 1]], history)

    return None, None

eutectic_calc = Eutectic(files, tm, hfus, steps=steps)
eutectic, history = eutectic_calc.calc_eutectic(initial_t_range)

if not eutectic:
    print("No eutectic point found in the temperature range")
else:
    print("Found eutectic point:")
    print("x_1".rjust(10), "T (K)".rjust(10), "T (C)".rjust(10))
    x, t = eutectic
    print(str(x.round(5)).rjust(10), str(t.round(5)).rjust(10), str((-273.15 + t).
↳round(5)).rjust(10))

    # plot the solubility curves and eutectic point

```

(continues on next page)

(continued from previous page)

```

h_s1 = sorted(list(zip(history[0], history[2])), key=lambda x: x[0])
h_s2 = sorted(list(zip(history[1], history[2])), key=lambda x: x[0])

h_s1 = [x for x in h_s1 if x[0] >= eutectic[0] and x[1] >= eutectic[1]]
h_s2 = [x for x in h_s2 if x[0] <= eutectic[0] and x[1] >= eutectic[1]]

# adjust the melting point back to the correct value for high or low solubility
for i in range(len(h_s1)):
    if h_s1[i][0] > 0.9999:
        h_s1[i] = (h_s1[i][0], tm[0])
for i in range(len(h_s2)):
    if h_s2[i][0] < 0.0001:
        h_s2[i] = (h_s2[i][0], tm[1])

plt.plot([x[0] for x in h_s1], [x[1] for x in h_s1], label="x_1 (solvent compound
→1)")
plt.plot([x[0] for x in h_s2], [x[1] for x in h_s2], label="x_1 (solvent compound
→2)")
plt.plot(eutectic[0], eutectic[1], "o", label="Eutectic point")
if eutectic[0] < 0.5:
    plt.annotate(" " + str(tuple([xt.round(3) for xt in eutectic])), eutectic,
→va="center", ha="left")
else:
    plt.annotate(str(tuple([xt.round(3) for xt in eutectic])) + " ", eutectic,
→va="center", ha="right")

plt.xlabel("Mole fraction compound 1")
plt.ylabel("Melting point of mixture (K)")
plt.legend(loc="upper right")
plt.grid()
plt.show()

finish()

```

This figure (produced by the code) shows the two solubility curves calculated by the program.

```

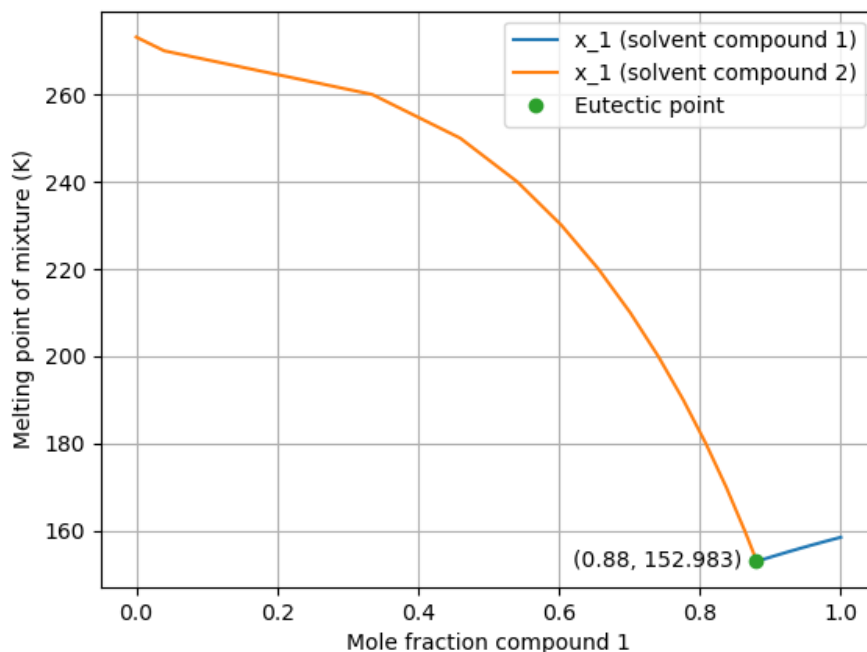
Found eutectic point:
    x_1      T (K)      T (C)
0.88028  152.98252 -120.16748

```

## References

### Binodal and Spinodal Curves

Binodal and spinodal curves are useful for understanding phase stability. The binodal curve (or coexistence curve) defines the temperatures and compositions at which phase separation is thermodynamically favorable. The spinodal curve is located within the binodal curve and indicates the limit of local phase stability. Compositions between the spinodal and binodal curves – while not thermodynamically stable – are robust against small fluctuations (i.e., the free energy surface is locally convex for points in this region).



### Python code (Binary mixture)

```
import os
import numpy as np
import matplotlib.pyplot as plt
from scm.plams import Settings, init, finish, CRSJob, config

##### Note: Ensure to configure the database path to either the installed ADFCRS-
↳2018 directory or your own specified directory #####

database_path = os.path.join(os.environ["SCM_PKG_ADFCRSDIR"], "ADFCRS-2018")
if not os.path.exists(database_path):
    raise OSError(f"The provided path does not exist. Exiting.")

init()
# suppress plams output
config.log.stdout = 0

files = ["Nitrobenzene.coskf", "Hexane.coskf"]
nring = [6, 0]
# problem paramters
temp_range = [203.15, 243.15] # K -- the temperature range over which the curves are
↳calculated
steps = 40 # number of steps to take within the temperature range

def binmix_at_T(files, temp):
    # initialize settings object
```

(continues on next page)

(continued from previous page)

```

settings = Settings()
settings.input.property._h = "BINMIXCOEF"

settings.input.property.nfrac = 100

# make compounds
compounds = [Settings() for i in range(len(files))]
for i, (file, nr) in enumerate(zip(files, nring)):
    compounds[i]._h = os.path.join(database_path, file)
    compounds[i].nring = nr

settings.input.temperature = temp

# optionally, change to the COSMOSAC2013 method
settings.input.method = "COSMOSAC2013"
# we'll also tighten the convergence threshold for better numerical accuracy
settings.input.Technical.sacconv = 1e-10

# add the compounds to the settings object
settings.input.compound = compounds
# create a job that can be run by COSMO-RS
my_job = CRSJob(settings=settings)
# run the job
out = my_job.run()
# convert all the results into a python dict
res = out.get_results()

return res

def calc_binodal_at_T(res):

    # this is the miscibility gap, so we can use the result calculated by the program
    if res["showmiscgap"]:
        return res["x11e"][:2]
    else:
        return None

def calc_spinodal_at_T(res):

    # here, we'll look for points with d^2(G_mix)/dx^2 = 0
    # we'll calculate a numerical second derivative for every point
    spinodal = []
    gmix = res["Gibbs energy of mixing"]
    frac1 = res["molar fraction"][0]
    second_deriv = np.zeros(len(gmix))

    # initial values for endpoints (assuming convexity close to pure compounds)
    second_deriv[0] = 0.0001
    second_deriv[-1] = 0.0001

    for i in range(1, len(gmix) - 1):
        delta1 = frac1[i] - frac1[i - 1]
        delta2 = frac1[i + 1] - frac1[i]
        d1 = (gmix[i] - gmix[i - 1]) / delta1
        d2 = (gmix[i + 1] - gmix[i]) / delta2

```

(continues on next page)

(continued from previous page)

```

second_deriv[i] = 2 * (d2 - d1) / (delta1 + delta2)

for i in range(len(second_deriv) - 1):
    if second_deriv[i] * second_deriv[i + 1] < 0:
        dist1 = abs(second_deriv[i])
        dist2 = abs(second_deriv[i + 1])
        tot = dist1 + dist2
        zero = (dist2 * frac1[i] + dist1 * frac1[i + 1]) / tot
        spinodal.append(zero)

return spinodal if spinodal else None

temps = [temp_range[0] + (temp_range[1] - temp_range[0]) / steps * i for i in
↪range(steps + 1)]
bin_left_points = []
bin_right_points = []
spin_left_points = []
spin_right_points = []
print("Temperature".ljust(15), "Binodal points".ljust(25), "Spinodal points")
for temp in temps:
    res = binmix_at_T(files, temp)
    binodal = calc_binodal_at_T(res)
    spinodal = calc_spinodal_at_T(res)

    bin_str = "(" + ",".join(["{0:<10.5g}".format(x) for x in binodal]) + ")" if
↪binodal is not None else "--"
    spin_str = "(" + ",".join(["{0:<10.5g}".format(x) for x in spinodal]) + ")" if
↪spinodal is not None else "--"
    print("{0:.5g}".format(temp).ljust(15), bin_str.ljust(25), spin_str)

    if binodal is not None:
        bin_left_points.append((binodal[0], temp))
        bin_right_points.append((binodal[1], temp))
    if spinodal is not None:
        spin_left_points.append((spinodal[0], temp))
        spin_right_points.append((spinodal[-1], temp))

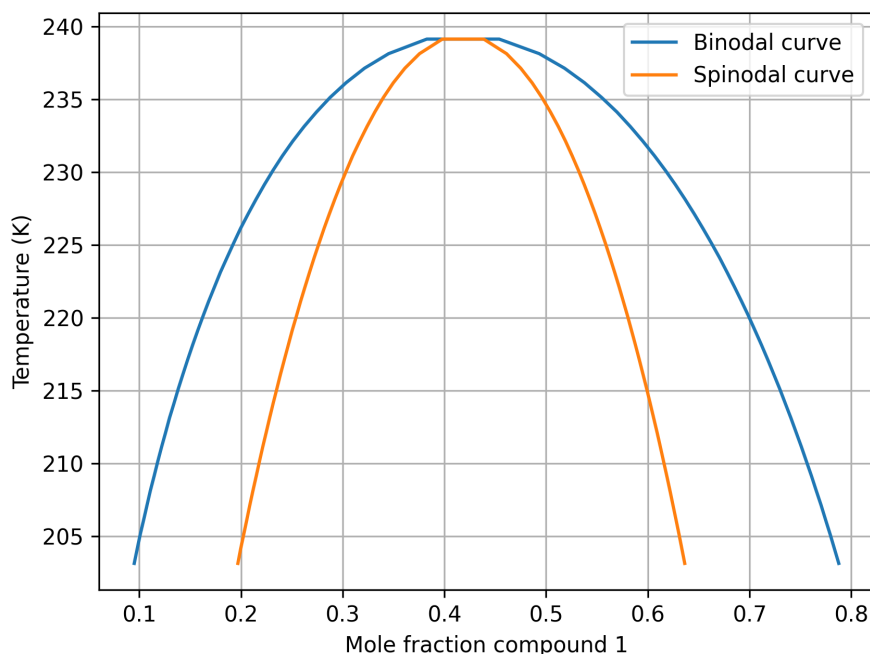
bin_points = bin_left_points + list(reversed(bin_right_points))
spin_points = spin_left_points + list(reversed(spin_right_points))

plt.plot([x[0] for x in bin_points], [y[1] for y in bin_points], label="Binodal curve
↪")
plt.plot([x[0] for x in spin_points], [y[1] for y in spin_points], label="Spinodal
↪curve")
plt.xlabel("Mole fraction compound 1")
plt.ylabel("Temperature (K)")
plt.legend(loc="upper right")
plt.grid()
# plt.savefig('./Binodalspinodal_binary.png', dpi=300)
plt.show()

finish()

```

This code produces the following output:



### Python code (Ternary mixture)

**Note:** This example uses the python package `ternary`, but this is *only required for plotting*. This external python package can be installed from the internet in `amspython` using `pip` as follows: `amspython -m pip install python-ternary`. Users may choose to remove the plotting features of the code and not install `ternary`.

```
import os
import numpy as np
import matplotlib.pyplot as plt
from scm.plams import *

try:
    import ternary
except ImportError:
    print("Cannot find ternary package.")
    print("Try to install with:")
    print("amspython -m pip install python-ternary")
    exit()

##### Note: Ensure to configure the database path to either the installed ADFCRS-
→2018 directory or your own specified directory #####

database_path = os.path.join(os.environ["SCM_PKG_ADFCRSDIR"], "ADFCRS-2018")
if not os.path.exists(database_path):
    raise OSError(f"The provided path does not exist. Exiting.")

init()
# suppress plams output
```

(continues on next page)

(continued from previous page)

```

config.log.stdout = 0

files = ["Water.coskf", "Chloroform.coskf", "Acetic_acid.coskf"]
nring = [0, 0, 0]
nfrac = 50 # the nfrac parameter for ternary mixtures
temp_range = [273.15, 273.15] # K -- the temperature range over which the curves are
↳calculated
steps = 0 # number of steps to take within the temperature range

def ternmix_at_T(files, temp, nfrac=20):

    # initialize settings object
    settings = Settings()
    settings.input.property._h = "TERNARYMIX"
    settings.input.property.nfrac = nfrac

    # make compounds
    compounds = [Settings() for i in range(len(files))]
    for i, (file, nr) in enumerate(zip(files, nring)):
        compounds[i]._h = os.path.join(database_path, file)
        compounds[i].nring = nr

    settings.input.temperature = temp

    # optionally, change to the COSMOSAC2013 method
    settings.input.method = "COSMOSAC2013"
    # we'll also tighten the convergence threshold for better numerical accuracy
    settings.input.Technical.saconv = 1e-10
    settings.input.Technical.rsconv = 1e-10

    # add the compounds to the settings object
    settings.input.compound = compounds
    # create a job that can be run by COSMO-RS
    my_job = CRSJob(settings=settings)
    # run the job
    out = my_job.run()
    # convert all the results into a python dict
    res = out.get_results()

    return res

def calc_binodal_at_T(res):

    # this is the miscibility gap, so we can use the result calculated by the program
    points_l = []
    points_r = []

    if res["nxll"] > 0:
        for i in range(len(res["xll"]) // 6):
            points_l.append(res["xll"][6 * i : 6 * i + 3])
            points_r.append(res["xll"][6 * i + 3 : 6 * i + 6])
        return points_l + list(reversed(points_r))
    else:
        return None

```

(continues on next page)

(continued from previous page)

```

def calc_spinodal_at_T(res):

    # in this function, we search for mole fraction values where the determinant of
    ↪the hessian of G_mix with respect to mole fractions is 0
    # using the first two mole fractions as degrees of freedom
    spinodal = []
    gmix_res = res["Gibbs energy of mixing"]
    fracs = res["molar fraction"][:2]

    step_size = abs(fracs[0][1] - fracs[0][0])
    tot_steps = round(1.0 / step_size) + 1
    gmix = np.empty((tot_steps, tot_steps))
    gmix.fill(None)
    det_mat = np.empty((tot_steps, tot_steps))
    det_mat.fill(None)

    for x1, x2, gmix_val in zip(fracs[0], fracs[1], gmix_res):
        idx1 = int(round(x1 / step_size, 0))
        idx2 = int(round(x2 / step_size, 0))
        gmix[idx1, idx2] = gmix_val

    # calculate second derivatives and determinants
    for i in range(tot_steps):
        for j in range(tot_steps - i):
            # these finite difference expressions work because the ternary mixture
            ↪always has a constant step size
            if 0 < i < tot_steps - 1 and 0 < j < tot_steps - 1:
                d_xx = (gmix[i + 1, j] - 2 * gmix[i, j] + gmix[i - 1, j]) / (step_
                ↪size**2)
                d_yy = (gmix[i, j + 1] - 2 * gmix[i, j] + gmix[i, j - 1]) / (step_
                ↪size**2)
                d_xy = (gmix[i + 1, j + 1] - gmix[i + 1, j - 1] - gmix[i - 1, j + 1]
                ↪+ gmix[i - 1, j - 1]) / (
                    4 * step_size**2
                )
                det_mat[i, j] = d_xx * d_yy - d_xy**2

    for i in range(tot_steps):
        for j in range(tot_steps - i):
            # compare right and below
            for (i1, j1), (i2, j2) in [(i, j), (i + 1, j)], [(i, j), (i, j + 1)]:
                if (
                    i2 < tot_steps
                    and j2 < tot_steps
                    and not np.isnan(det_mat[i1, j1])
                    and not np.isnan(det_mat[i2, j2])
                    and det_mat[i1, j1] * det_mat[i2, j2] < 0
                ):
                    x1 = step_size * np.array([i1, j1, tot_steps - 1 - i1 - j1])
                    x2 = step_size * np.array([i2, j2, tot_steps - 1 - i2 - j2])

                    spin = (abs(det_mat[i2, j2]) * x1 + abs(det_mat[i1, j1]) * x2) / (
                        abs(det_mat[i1, j1]) + abs(det_mat[i2, j2])
                    )
                    spinodal.append(spin)

```

(continues on next page)

```

return spinodal

## Make ternary figure
figure, tax = ternary.figure(scale=1.0)
tax.boundary(linewidth=2.0)
tax.gridlines(color="black", multiple=0.05)
# Set Axis labels and Title
fontsize = 10
tax.bottom_axis_label("$x_1$", fontsize=fontsize, offset=0.2)
tax.right_axis_label("$x_2$", fontsize=fontsize, offset=0.2)
tax.left_axis_label("$x_3$", fontsize=fontsize, offset=0.2)

tax.ticks(axis="lbr", multiple=0.1, linewidth=1, tick_formats="%0.1f", offset=0.02,
↳fontsize=fontsize)

# Remove default Matplotlib Axes
tax.clear_matplotlib_ticks()

temps = []
if steps > 0:
    temps = [temp_range[0] + (temp_range[1] - temp_range[0]) / steps * i for i in
↳range(steps + 1)]
else:
    temps = [temp_range[0]] if isinstance(temps, list) else [temp_range]

for temp in temps:
    res = termix_at_T(files, temp, nfrac)
    binodal = calc_binodal_at_T(res)
    spinodal = calc_spinodal_at_T(res)

    # the sorting index might need to be changed here if the curves are in a
↳different position
    binodal.sort(key=lambda x: x[1])
    spinodal.sort(key=lambda x: x[1])

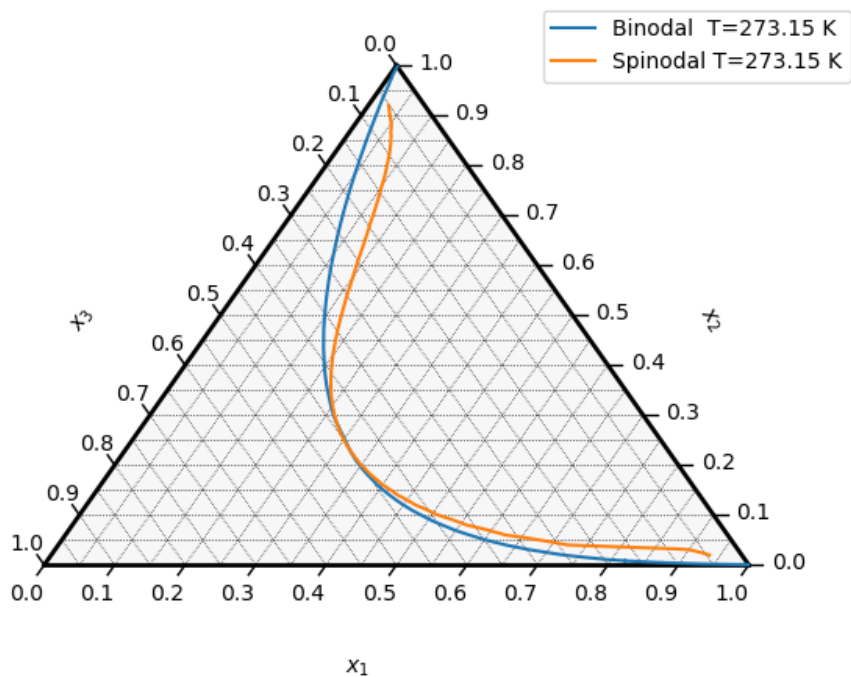
    tax.plot(binodal, label="Binodal T=" + str(temp) + " K")
    tax.plot(spinodal, label="Spinodal T=" + str(temp) + " K")

tax.get_axes().axis("off")
ternary.plt.legend(bbox_to_anchor=(1.1, 1.0), loc="right")
ternary.plt.show()

finish()

```

This code produces the following output:



### Distribution of species in multispecies calculations

COSMO-RS can be used with compounds which can be composed of multiple possible species. For these types of calculations, it is often desirable to know the distribution of the various possible forms/species that constitute a certain compound. In the following script, a binary mixture calculation is performed using benzene and an acetic acid compound which is capable of existing as either of 2 conformers or as a dimer. The distribution of these species is plotted as a function of mole fraction.

Download relevant coskf file

### Python code (Binary mixture)

```
import os
import matplotlib.pyplot as plt
from scm.plams import Settings, init, finish, CRSJob, config

##### Note: Ensure to download the coskf_acetic_acid before running the_
->script #####
database_path = os.path.abspath("./coskf_acetic_acid")

if not os.path.exists(database_path):
    raise OSError(f"The provided path does not exist. Exiting.")

def adjust_name(s):
    return os.path.basename(s)

init()
```

(continues on next page)

```
# suppress plams output
config.log.stdout = 0

# initialize settings object
settings = Settings()
settings.input.property._h = "BINMIXCOEF"
# optionally, change to the COSMOSAC2013 method
# settings.input.method = 'COSMOSAC2013'

# set the number of compounds
num_compounds = 2
compounds = [Settings() for i in range(num_compounds)]

compounds[0].name = "acetic_acid"
form = [Settings() for i in range(3)]

form[0]._h = os.path.join(database_path, "acetic_acid_0.coskf")

form[1]._h = os.path.join(database_path, "acetic_acid_1.coskf")

form[2]._h = os.path.join(database_path, "acetic_dimer.coskf")
form[2].count = 2
form[2].Hcorr = 9.25

compounds[0].form = form

compounds[1]._h = os.path.join(database_path, "Benzene.coskf")
compounds[1].name = "Benzene"

settings.input.temperature = 298.15

# specify the compounds as the compounds to be used in the calculation
settings.input.compound = compounds
# create a job that can be run by COSMO-RS
my_job = CRSJob(settings=settings)
# run the job
out = my_job.run()

# convert all the results into a python dict
res = out.get_results()

compositions = out.get_multispecies_dist()

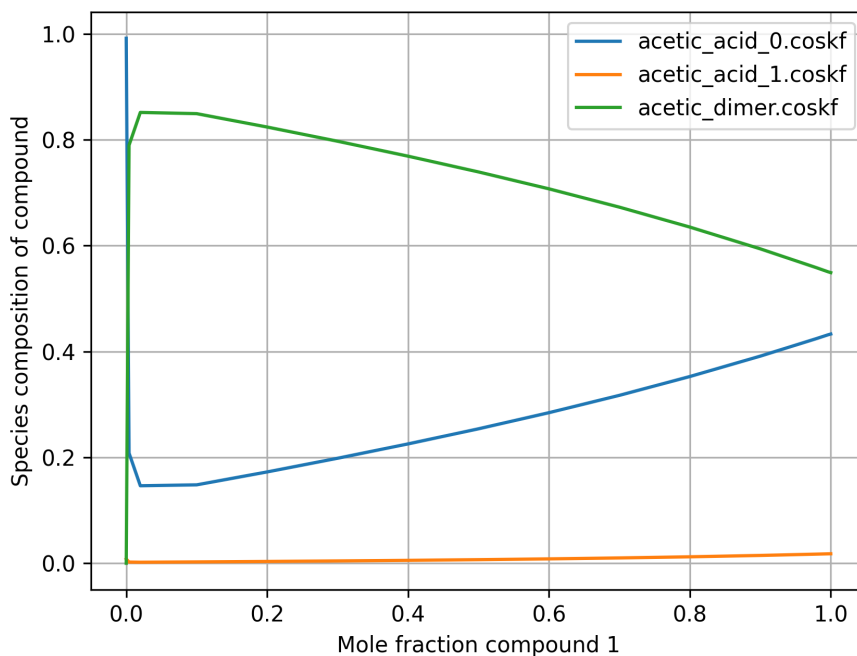
mf1 = res["molar fraction"][0]

plot_comp = 0 # we'll plot the first compound (acetic acid)
for struct, vals in compositions[plot_comp].items():
    plt.plot(mf1, vals, label=adjust_name(struct))

plt.xlabel("Mole fraction compound 1")
plt.ylabel("Species composition of compound")
plt.legend(loc="upper right")
plt.grid()
# plt.savefig("./ms_distribution.png", dpi=300)
plt.show()

finish()
```

This code produces the following output:



### Automated pKa calculation

Download relevant coskf file

### Python code

```
import os, math
from scm.plams import Settings, init, finish, CRSJob, config
import matplotlib.pyplot as plt

init()
# suppress plams output
config.log.stdout = 0

class PkaSystem:
    def __init__(
        self,
        comp_protonated: Settings = None,
        comp_deprotonated: Settings = None,
        systype="acid",
    ):
        self.comp_protonated = comp_protonated
        self.comp_deprotonated = comp_deprotonated
        self.type = systype.lower()
```

(continues on next page)

(continued from previous page)

```

self.g_cp = []
self.g_cd = []
self.pka = {}
self.error = self.check_type()

def check_type(self):
    if not all(isinstance(x, Settings) for x in [self.comp_protonated, self.comp_
↳deprotonated]):
        print("Incorrect input type for compounds in PkaSystem... must be a plams.
↳Settings instance")
        return False
    return True

class PkaCalculator:
    def __init__(
        self,
        systems=None,
        solv_protonated=None,
        solv_deprotonated=None,
        temp_range=[298.15, 298.15],
        steps=0,
        use_correction=True,
    ):

        self.systems = systems
        self.solv_protonated = solv_protonated
        self.solv_deprotonated = solv_deprotonated
        self.temp_range = temp_range
        self.steps = steps
        self.use_correction = use_correction
        self.R_const = 0.001987 # kcal/(mol K)

        self.g_sp = []
        self.g_sd = []

        if self.steps == 0:
            self.temp_vals = [temp_range[0]]
        else:
            self.temp_vals = [
                self.temp_range[0] * (1 - t_idx / self.steps) + self.temp_range[1] *
↳(t_idx / self.steps)
                for t_idx in range(self.steps + 1)
            ]

        self.calc_G_values()
        self.calc_pkas()

    def calc_G_values(self):

        for temp in self.temp_vals:
            settings = Settings()
            settings.input.property._h = "ACTIVITYCOEF"
            # optionally, change to the COSMOSAC2013 method
            # settings.input.method = 'COSMOSAC2013'
            settings.input.temperature = temp

```

(continues on next page)

(continued from previous page)

```

compounds = [Settings()] * (2 + 2 * len(systems))
compounds[0] = self.solv_deprotonated
compounds[0].frac1 = 1.0
compounds[1] = self.solv_protonated

for i, system in enumerate(systems):
    compounds[2 + 2 * i] = system.comp_deprotonated
    compounds[2 + 2 * i + 1] = system.comp_protonated

settings.input.compound = compounds
my_job = CRSJob(settings=settings)
out = my_job.run()
res = out.get_results()

g_vals = res["G solute"]
for i, system in enumerate(systems):
    system.g_cd.append(g_vals[2 + 2 * i])
    system.g_cp.append(g_vals[2 + 2 * i + 1])
self.g_sd.append(g_vals[0])
self.g_sp.append(g_vals[1])

def calc_pkas(self):
    for i, temp in enumerate(self.temp_vals):
        temp_key = round(temp, 3)
        for system in self.systems:
            g_diss = system.g_cd[i] - system.g_cp[i] + self.g_sp[i] - self.g_sd[i]
            if self.use_correction:
                pka = self.calc_corrected_pka(g_diss, system, temp)
            else:
                pka = g_diss / (self.R_const * temp * math.log(10)) - 1.74
            system.pka[temp_key] = pka

def calc_corrected_pka(self, g_diss, system, temp):
    if system.type == "acid":
        return 0.62 * g_diss / (self.R_const * temp * math.log(10)) + 2.1
    else:
        return 0.67 * g_diss / (self.R_const * temp * math.log(10)) - 2.0

if __name__ == "__main__":

    ##### Note: Ensure to download the coskf_pkas before running the_
    ↪script #####
    database_path = os.path.abspath("./coskf_pkas")

    if not os.path.exists(database_path):
        raise OSError(f"The provided path does not exist. Exiting.")

    systems = []

    benzoic_acid = Settings({"_h": os.path.join(database_path, "Benzoic_acid.coskf")})
    benzoic_acid_deprotonated = Settings({"_h": os.path.join(database_path,
    ↪"conjugate_base_Benzoic_acid.coskf")})
    systems.append(PkaSystem(comp_protonated=benzoic_acid, comp_deprotonated=benzoic_
    ↪acid_deprotonated, systype="acid"))

    pyridine = Settings({"_h": os.path.join(database_path, "Pyridine.coskf")})

```

(continues on next page)

(continued from previous page)

```
pyridineH = Settings({"_h": os.path.join(database_path, "conjugate_acid_Pyridine.  
↪coskf")})  
systems.append(PkaSystem(comp_protonated=pyridineH, comp_deprotonated=pyridine,  
↪systype="base"))  
  
water = Settings({"_h": os.path.join(database_path, "Water.coskf")})  
h3o = Settings({"_h": os.path.join(database_path, "conjugate_acid_Water.coskf")})  
  
PkaCalculator(systems, solv_protonated=h3o, solv_deprotonated=water, temp_  
↪range=[298.15, 348.15], steps=5)  
  
for system in systems:  
    print(system.pka)  
  
finish()
```

## COMMAND LINE SCRIPTING WITH COSMO-RS

### 11.1 AMSprep: construct an ADF COSMO results file

The module `amsprep` is intended to facilitate scripting. More details on `amsprep` can be found in the [AMSprep section in the Scripting manual](#). For COSMO-RS the most relevant is the ADFCRS template. The template ADFCRS will perform a gas phase geometry optimization, and next a COSMO calculation at the gas phase optimized geometry.

In the next example the result of the ADF calculation is a file called `adfwater.coskf`, which is an ADF COSMO result file, that can be used as input for a COSMO-RS calculation, see also the [Example: COSMO-RS Tutorial 1](#) (page 214).

```
cat << eor > water.xyz
3
H      0.00000000      0.77121000      0.18071000
O      0.00000000     -0.00000000     -0.36142000
H      0.00000000     -0.77121000      0.18071000
eor

"$AMSBIN/amsprep" -t ADFCRS -m water.xyz -j adfwater >job
chmod +x job
./job
```

### 11.2 CRSprep: generate (multiple) COSMO-RS jobs

The module `crsprep` is intended to facilitate scripting: it makes it easier to construct proper jobs, from within a script. This module can be used, for example, to run the same type of COSMO-RS job on various compounds, or to change input settings. This module can also be used to put pure compound data on an ADF COSMO result file.

In `$AMSHOME/examples/crs` one can find examples that follow the COSMO-RS GUI tutorials, which are also described in the section [Scripting Examples](#) (page 213).

The most convenient way to see the options of `crsprep` is to run the `crsprep` command without arguments. You will get output very much alike the following description, but probably more up-to-date.

```
% crsprep -h

CRSprepare (crsprep) saves pure compound data on an ADF COSMO result file
or generates a job script for COSMO-RS calculations,
with user specified changes to input options / method / system.
Required is at least 1 compound and -savecompound or -t template
```

(continues on next page)

(continued from previous page)

## Usage:

```

crsprep -savecompound -s compound.coskf
    [-nring nring] [-pvap pvap] [-tvap tvap] [-antoinc "A B C"]
    [-meltingpoint meltingpoint] [-hfusion hfusion] [-cpfusion cpfusion]
    [-flashpoint flashpoint] [-density density] [-scalearea scalearea]
crsprep -t template
    [-s compound.(coskf|compoundlist)] [-c compound.(coskf|compoundlist)]
    [-nring nring] [-pvap pvap] [-tvap tvap] [-antoinc "A B C"]
    [-meltingpoint meltingpoint] [-hfusion hfusion] [-cpfusion cpfusion]
    [-flashpoint flashpoint] [-density density] [-scalearea scalearea]
    [-frac1 frac1] [-frac2 frac2]
    [-densitysolvent densitysolvent] [-solphase solphase]
    [-volumequotient volumequotient] [-preset preset]
    [-method method] [-temperature temperature] [-pressure pressure]
    [-iso iso] [-n n] [-inputpurevap inputpurevap]
    [-inputpuredensity inputpuredensity]
    [-sigmax sigmax] [-massfraction] [-file filename] [-j jobname]

```

## SAVE PURE COMPOUND DATA

```

-savecompound
    use to save pure compound data on an existing ADF COSMO result file

```

## TEMPLATE

```

-t template
    use for COSMO-RS property calculation
    template should be one of:
    VAPORPRESSURE, PUREVAPORPRESSURE,
    BOILINGPOINT, PUREBOILINGPOINT, FLASHPOINT,
    LOGP, ACTIVITYCOEF, SOLUBILITY, PURESOLUBILITY,
    BINMIXCOEF, TERNARYMIX, COMPOSITIONLINE,
    SIGMAPROFILE, PURESIGMAPROFILE, SIGMAPOTENTIAL, PURESIGMAPOTENTIAL

```

## COMPOUNDS

```

    at least 1 compound is required
-s file: the special compound(s) to be used, should be a .coskf file,
    or a .compoundlist file. The -s key has to be repeated for each file
-c file: additional compound(s) to be used, should be a .coskf file,
    or a .compoundlist file. The -c key has to be repeated for each file
    the order of the compounds is: first all compound defined with -s,
    then those with -c
LOGP, ACTIVITYCOEF, SOLUBILITY: use -s for the solvent and -c for the solutes
PURESOLUBILITY: use -s for the solute and -c for the solvents

```

## PURE COMPOUND DATA

```

-nring: number of ring atoms
-pvap: pure compound vapor pressure (bar) at tvap
-tvap: at this temperature (Kelvin) pure compound has pressure pvap
-antoinc: Antoine coefficients A, B, and C
-meltingpoint: melting point (Kelvin)
-hfusion: enthalpy of fusion (kcal/mol)
-cpfusion: heat capacity of fusion (kcal/(mol K))
-flashpoint: flash point (Kelvin)
-density: liquid density (kg/L)
-scalearea: COSMO surface area scale factor
    these keys can be repeated for each compound,
    first appearance of the key will be for compound 1, second for compound 2, etc.
    note the order of the compounds

```

(continues on next page)

(continued from previous page)

```

SOLVENT
-frac1: define solvent
-frac2: define solvent 2 (LogP, composition line)
      the -frac1 and -frac2 key have to be repeated for
      each compound that should have a non-default value
      first appearance of the key will be for compound 1, second for compound 2, etc.
      note the order of the compounds
-densitysolvent: density solvent (kg/L)
-solphase: pure compound phase solute in solubility calculation
-volumequotient: molar volume phase 1/molar volume phase 2 (LogP)
-preset: LogP preset 0, 2, 3, 4, 5
        0: user defined; 2: Octanol/Water; 3: Benzene/Water; 4: Ether/Water;
        5: Hexane/Water

METHOD, SYSTEM
-method: COSMO-RS, COSMOSAC2013, COSMOSAC2016
-temperature: temperature (Kelvin)
      the -temperature key can be used twice to give a range
-pressure: pressure (bar)
      the -pressure key can be used twice to give a range
-iso: isotherm, isobar, flashpoint
-n: number of steps
-inputpurevap: if 1 use input pure compound pvap and tvap or Antoine equation
-inputpuredensity: if 1 use input pure compound liquid density
-sigmax: maximum value sigma (sigma profile, sigma potential)
-massfraction: use mass fractions

INPUT FILE
-file: content of the file will be added at the end of the input for the
      COSMO-RS calculation. The -file key has to be repeated for each file

OUTPUT
-j: produce a fully runnable job (as the .job files from AMSjobs),
   using the specified jobname. The job script produces files like jobname.out,
   jobname.crskf etc. Several job scripts can simply be concatenated,
   the results will be stored in different files using the jobname parameter
   the default is a simple run script

EXAMPLES
crsprep -s benzene.coskf -nring 6 -savecompound
crsprep -t VAPORPRESSURE -temperature 273.15 -temperature 373.15 -s methanol.coskf

```

## 11.3 AMSreport: generate report

The module `amsreport` is intended to facilitate scripting. More details on `amsreport` can be found in the [AMSreport section in the Scripting manual](#). It makes it very easy to get results calculated by COSMO-RS (or other programs in the ADF suite) in your own script. Compared to ADF2014 `AMSreport` has been extended to get easier results from COSMO-RS result files (.crskf files). It was already possible to report any proper KF variable from the .crskf file. Now a few predefined keys are added. See the `$AMSHOME/examples/crs` directory for use of `amsreport` in COSMO-RS calculations. Depending on the kind of calculation one can use:

Command line option	Property
---------------------	----------

(continues on next page)

(continued from previous page)

TOC	Table of contents
PROPERTY	General:Property
METHOD	Property:Method
NITEMS	Property:Number of Items
FRAC1	Property:Solvent: molar fraction
FRAC2	Property:Solvent 2: molar fraction
SOLVENT-FRACTION	Property:Solvent: solvent fraction
TEMPERATURE	Property:Temperature (in Kelvin)
PRESSURE	Property:Pressure (in bar)
GIBBS-ENERGY-MIXING	Property:Gibbs free energy of Mixing (in kcal/mol)
GIBBS-ENERGY-SOLVATION	Property:Gibbs free energy of Solvation (in kcal/mol)
GIBBS-ENERGY-SOLUTE	Property:Gibbs free energy solute (in kcal/mol)
EXCESS-G	Property:Excess Gibbs free energy (in kcal/mol)
EXCESS-H	Property:Excess Enthalpy (in kcal/mol)
ENTHALPY-VAPORIZATION	Property:Enthalpy of vaporization (in kcal/mol)
LOGP	Property:LogP
MOLAR-FRACTION	Property:Molar Fraction
ACTIVITY-COEFFICIENT	Property:Activity Coefficient
VAPOR-PRESSURE	Property:Vapor Pressure (in bar)
SOLUBILITY-X	Property:Solubility: molar fraction
SOLUBILITY-M	Property:Solubility: moles per liter (in mol/(L solution))
SOLUBILITY-G	Property:Solubility: gram per liter (in g/(L solution))
SOLUBILITY-MASS-FRACTION	Property:Solubility: mass fraction
HENRY	Property:Henry Constant (in mol/(L atm))
HENRY-NODIM	Property:Henry Constant dimensionless
MISCIBILITY-GAP	Property:Miscibility gap
MISCIBILITY-GAP-T	Property:Miscibility gap temperature (in Kelvin)
MISCIBILITY-GAP-P	Property:Miscibility gap pressure (in bar)
MISCIBILITY-GAP-X	Property:Miscibility gap molar fraction x1 x1'
MISCIBILITY-GAP-A	Property:Miscibility gap activities a1 a2
TIE-LINES-X	Property:Tie Lines molar fraction x1 x2 x3 x1' x2' x3'
TIE-LINES-A	Property:Tie Lines activities a1 a2 a3
CHEMICAL-POTENTIAL	Property:Chemical Potential
CHEMICAL-POTENTIAL-PURE	Property:Chemical Potential Pure Compounds Liquid
CHEMICAL-POTENTIAL-GAS	Property:Chemical Potential Pure Compounds Gas
SIGMA	Property:Sigma
SIGMA-PROFILE	Property:Sigma Profile
SIGMA-PROFILE-HB	Property:Sigma Profile Hydrogen Bonding part
SIGMA-PROFILE-TOTAL	Property:Total Sigma Profile
SIGMA-PROFILE-HB-TOTAL	Property:Total Sigma Profile Hydrogen Bonding part
SIGMA-POTENTIAL	Property:Sigma Potential
SIGMA-POTENTIAL-TOTAL	Property:Total Sigma Potential
NCOMP	Compounds:Number of Compounds
COMPOUNDS-FILENAME	Compounds:Filename
COMPOUNDS-NAME	Compounds:Name (from filename)
COMPOUNDS-MOLAR-MASS	Compounds:Molar Mass
Example	
"\$AMSBIN/amsreport" file.crskf TOC	
"\$AMSBIN/amsreport" file.crskf ncomp	
"\$AMSBIN/amsreport" file.crskf ncomp -plain	

## 11.4 KF utilities for COSMO-RS

### 11.4.1 KF browser

With the GUI module `kfbrowser` one can browse through the raw data on KF files (like the `.crskf` COSMO-RS result files).

```
$AMSBIN/kfbrowser file.crskf
```

### 11.4.2 `kf2cosmo` and `cosmo2kf`

The two COSMO-RS command line utility programs `kf2cosmo` and `cosmo2kf` convert COSMO kf files from binary to ASCII and vice versa.

```
kf2cosmo file.coskf file.cosmo
```

`kf2cosmo` reads from the kf file `file.coskf` (should exist) the section 'COSMO' and writes to the ASCII file `file.cosmo` (should not exist). Instead of a `.coskf` file, the file can also be a TAPE21 file which is a result file from an ADF COSMO calculation.

```
cosmo2kf file.cosmo file.coskf
```

`cosmo2kf` reads from the ASCII file `file.cosmo` (should exist) and writes a section 'COSMO' to the kf file `file.coskf` (should not exist). Note that only a section 'COSMO' is written to the kf file, which means that this file can not be used like an ordinary `adf.rkf` file (previously ADF<=2019 TAPE21 file or `.t21` file).

```
cosmo2kf file.cos file.coskf
```

`cosmo2kf` can also read a MOPAC COSMO result file `file.cos` (should exist and should have the file extension `.cos`) and writes a section 'COSMO' to the kf file `file.coskf` (should not exist).

### 11.4.3 `pkf`, `cpkf`, `dmpkf`, `udmpkf`

```
pkf file.crskf
```

`pkf` prints a summary of the contents of a kf file

```
cpkf adf.rkf file.coskf COSMO
```

With the ADF kf utility `cpkf` one can copy the section 'COSMO' from an `adf.rkf` (should exist) to a `file.coskf` (should not exist). The file `file.t21` should be a result of an ADF COSMO calculation. The file `file.coskf` is much smaller than `adf.rkf`. This file `file.coskf` can not be used like an ordinary `adf.rkf` file, but it contains all necessary information such that it can be used as input for a COSMO-RS calculations.

With the ADF kf utilities `dmpkf` and `udmpkf` one can also convert COSMO kf files from binary to ASCII and vice versa. Note that the ASCII files in this case are not so called `.cosmo` files.

```
dmpkf file.coskf > ascii_result
udmpkf < ascii_result newfile.coskf
```

## 11.5 Scripting Examples

### 11.5.1 Example: COSMO result files

Download COSMO\_files.run

```
#!/bin/sh

# This example tries to do the same as in the COSMO-RS GUI Tutorial: COSMO result_
↪files,
# using scripts.
#
# In the first part of the example you will find how to use amsprep to construct
# an ADF COSMO results file.
# The template ADFCRS will perform a gas phase geometry optimization (for water, in_
↪this case),
# and next a COSMO calculation at the gas phase optimized geometry.
# The result file adfwater.coskf, which is an ADF COSMO result file, can be used as_
↪input for
# a COSMO-RS calculation. amsreport is used to extract data from the resulting .coskf_
↪file.

echo "Results"

rm -f water.xyz
rm -f job.sh
cat << eor > water.xyz
3
H      0.00000000      0.77121000      0.18071000
O      0.00000000     -0.00000000     -0.36142000
H      0.00000000     -0.77121000      0.18071000
eor

"$AMSBIN/amsprep" -t GO -runtype "COSMO-RS Compound" -m water.xyz -j adfwater > job.
↪sh 2> job.err
chmod +x job.sh
./job.sh

echo "ADF"
"$AMSBIN/amsreport" adfwater.results geometry-a
cp adfwater.coskf adffile.coskf
"$AMSBIN/amsreport" adffile.coskf energies

# The template ADFCRS can also be used for generating an "average monomer" COSMO_
↪result file.
# The calculation could take up quite some time.

echo "POLYMER"
"$AMSBIN/amsprep" -t GO -runtype "COSMO-RS Compound" -smiles "C{-}C{n+}(c1ccccc1)" -j_
↪adfpolystyrene > job3.sh 2> job3.err
grep -i coskfatoms job3.sh |wc -w

# A MOPAC COSMO result file can also be constructed.
# Note that the syntax for MOPAC is not the same as in ADF2018.
# The template MOPAC-GO will perform a COSMO geometry optimization,
```

(continues on next page)

(continued from previous page)

```

# if -g "mopac.solvation.enabled 1" -g "mopac.solvation.solvent.name CRS" is included.
# If -g "mopac.solvation.nspa 362" is included the maximum number of COSMO surface
↳points
# is 362, which is more than the default 42.
# The result file mopacwaterresults/mopac.coskf is a MOPAC COSMO result file,
# which can be used as input for a COSMO-RS calculation.

"$AMSBIN/amsprep" -t MOPAC-GO -m water.xyz -g "mopac.solvation.enabled 1" \
                  -g "mopac.solvation.solvent.name CRS" -g "mopac.solvation.nspa 362"
↳\
                  -j mopacwater > job2.sh 2> job2.err
chmod +x job2.sh
./job2.sh

echo "MOPAC"
"$AMSBIN/amsreport" mopacwater.results geometry-a
"$AMSBIN/amsreport" mopacwater.results/mopac.coskf energies

# Fast Sigma can provide estimates of COSMO-RS sigma profiles in milliseconds.

echo "FAST SIGMA"
"$AMSBIN/fast_sigma" --smiles "CC(Cc1ccc(cc1)[C@@H](C(=O)O)C)C" -o ibuprofen.compkf
"$AMSBIN/amsreport" ibuprofen.compkf formula
"$AMSBIN/amsreport" ibuprofen.compkf energies

echo "Ready"

```

## 11.5.2 Example: COSMO-RS parameters and analysis

Download [Parameters\\_and\\_Analysis.run](#)

```

#!/bin/sh

# This example tries to part of the COSMO-RS GUI Tutorial: parameters and analysis
↳using scripts.
#
# First some .coskf files are copied to the location where the scripts are running.

cp $AMSHOME/examples/COSMO-RS/Parameters_and_Analysis/benzene.coskf .
cp $AMSHOME/examples/COSMO-RS/Parameters_and_Analysis/ethanol.coskf .
cp $AMSHOME/examples/COSMO-RS/Parameters_and_Analysis/methanol.coskf .
cp $AMSHOME/examples/COSMO-RS/Parameters_and_Analysis/water.coskf .

# Set pure compound parameters
# -----

# In the first part of the example you will find how to use crsprep to set pure
↳compound parameters,
# in this case to set the number of ring atoms for the benzene molecule, which has 6
↳ring atoms.
# The file benzene.coskf is changed, such that it contains the value "6" for the
↳number of ring compounds.

"$AMSBIN/crsprep" -c benzene.coskf -nring 6 -savecompound

```

(continues on next page)

(continued from previous page)

```

# The sigma profile and the sigma potential
# -----
#
# Next for convenience a file tutorial2.compoundlist is made that consists of 4 coskf_
↳files.

rm -f tutorial2.compoundlist
rm -f job
cat << eor > tutorial2.compoundlist
benzene.coskf
ethanol.coskf
methanol.coskf
water.coskf
eor

# In the second part of the example sigma profiles are calculated for the four_
↳compounds
# (result file step6.crskf), and next the sigma potentials of the four pure compounds
# (result file step7.crskf).
# With *amsreport* one can get the calculated sigma profile or sigma potential from
# the .crskf file on standard output.

"$AMSBIN/crsprep" -t PURESIGMAPROFILE -c tutorial2.compoundlist -j step6 > job.sh
"$AMSBIN/crsprep" -t PURESIGMAPOTENTIAL -c tutorial2.compoundlist -j step7 >> job.sh
chmod +x job.sh
./job.sh
echo "Results"
"$AMSBIN/amsreport" step6.crskf sigma
"$AMSBIN/amsreport" step6.crskf sigma-profile
"$AMSBIN/amsreport" step6.crskf sigma-profile-hb
"$AMSBIN/amsreport" step7.crskf sigma-potential
echo "Ready"

```

### 11.5.3 Example: COSMO-RS properties

Download Properties.run

```

#!/bin/sh

# This example tries to do the same as COSMO-RS GUI Tutorial: Properties, using_
↳scripts.

echo "Results"

# This example starts with copying 4 coskf files to the current directory,
# and modifying the benzene.coskf file such that the number of ring atoms is 6,
# like in the previous example.

rm -f job.sh
cp $AMSHOME/examples/COSMO-RS/Parameters_and_Analysis/benzene.coskf .
cp $AMSHOME/examples/COSMO-RS/Parameters_and_Analysis/ethanol.coskf .
cp $AMSHOME/examples/COSMO-RS/Parameters_and_Analysis/methanol.coskf .
cp $AMSHOME/examples/COSMO-RS/Parameters_and_Analysis/water.coskf .

```

(continues on next page)

(continued from previous page)

```

cp $AMSHOME/examples/COSMO-RS/Properties/2-hexanone.coskf .
cp $AMSHOME/examples/COSMO-RS/Properties/acetic_acid.coskf .

"$AMSBIN/crsprep" -c benzene.coskf -nring 6 -savecompound

# Step 2: Vapor pressure
# -----

# In step 2 of the example the vapor pressure of methanol is calculated at the
# default temperature of 298.15 K (result file step2a.crskf),
# and next for a series of temperatures ranging from 273.15 K to 373.15 K
# in 10 steps (result file step2b.crskf).

touch job.sh
chmod +x job.sh
"$AMSBIN/crsprep" -t PUREVAPORPRESSURE -j step2a \
                  -c methanol.coskf > job.sh
"$AMSBIN/crsprep" -t PUREVAPORPRESSURE -temperature 273.15 -temperature 373.15 -j_
->step2b \
                  -c methanol.coskf >> job.sh

./job.sh
echo "Step 2a"
"$AMSBIN/amsreport" step2a.crskf temperature
"$AMSBIN/amsreport" step2a.crskf pressure
"$AMSBIN/amsreport" step2a.crskf enthalpy-vaporization
echo "Step 2b"
"$AMSBIN/amsreport" step2b.crskf temperature
"$AMSBIN/amsreport" step2b.crskf pressure
"$AMSBIN/amsreport" step2b.crskf enthalpy-vaporization

# Step 3: Boiling point
# -----

# In step 3 of the example the boiling point of a mixture of methanol and ethanol
# is calculated, for a series of pressures ranging from 0.101325 to 1.01325 bar
# in 10 steps (result file step3.crskf).
# This mixture consist of 50% mole fraction methanol and 50% mole fraction ethanol.

"$AMSBIN/crsprep" -t BOILINGPOINT -pressure 0.101325 -pressure 1.01325 -j step3 \
                  -c methanol.coskf -frac1 0.5 \
                  -c ethanol.coskf -frac1 0.5 > job.sh

./job.sh
echo "Step 3"
"$AMSBIN/amsreport" step3.crskf temperature
"$AMSBIN/amsreport" step3.crskf pressure

# Step 4: Flash point
# -----

# In step 4 of the example the flash point of a mixture of ethanol and water
# is calculated (result file step4.crskf).
# This mixture consist of 44.2% mass fraction methanol and 55.8% mass fraction_
->ethanol.
# For a flash point calculation the pure compound flash points are needed as input,

```

(continues on next page)

(continued from previous page)

```

# since COSMO-RS does not predict pure compound flash points.
# For pure ethanol a flash point of 286 K is saved in the file ethanol.coskf.

"$AMSBIN/crsprep" -c ethanol.coskf -flashpoint 286 -savecompound

"$AMSBIN/crsprep" -t FLASHPOINT -massfraction -j step4 \
    -c ethanol.coskf -frac1 0.442 \
    -c water.coskf -frac1 0.558 > job.sh
./job.sh
echo "Step 4 flash point"
$AMSBIN/amsreport step4.crskf temperature

# Step 5: Activity coefficients, Henry coefficients, Solvation free energies
# -----

# In step 5 of the example the infinite diluted solutes benzene, methanol, and
# ethanol are calculated in the solvent water (result file step5.crskf).
# Activity coefficients, Henry coefficients and solvation free energies
# will be calculated.
# One should include -s flag for water, since it is the solvent,
# and considered here to be a special compound.

"$AMSBIN/crsprep" -t ACTIVITYCOEF -j step5 \
    -s water.coskf \
    -c benzene.coskf -c ethanol.coskf -c methanol.coskf > job.sh
./job.sh
echo "Step 5 Activity coefficients, Henry's law constants, Solvation energy"
"$AMSBIN/amsreport" step5.crskf Activity-Coefficient
"$AMSBIN/amsreport" step5.crskf Henry
"$AMSBIN/amsreport" step5.crskf Gibbs-energy-solvation

# Step 6: Partition coefficients (log P)
# -----

# In step 6 of the example the partition coefficients of infinitely diluted
# solutes in a mixture of two immiscible solvents are calculated.
# In step 6a the default Octanol/Water partition coefficients are calculated
# (default -preset 2) (result file step6a.crskf).
# In step 6b a user defined (-preset 0) Benzene/Water partition coefficients
# are calculated (result file step6b.crskf).
# In this case one should include -s flag for benzene and water, since these
# are the two immiscible solvents, and considered here to be the special compounds.
# The order of the compounds benzene and water is important, because the
# molar volume of phase 1 (benzene) divided by the the molar volume of phase 2
# (water) is given with the flag -volumequotient.

"$AMSBIN/crsprep" -t LOGP -j step6a \
    -c benzene.coskf -c ethanol.coskf -c methanol.coskf > job.sh
"$AMSBIN/crsprep" -t LOGP -preset 0 -volumequotient 4.93 -j step6b \
    -s benzene.coskf -s water.coskf \
    -c ethanol.coskf -c methanol.coskf >> job.sh
./job.sh
echo "Step 6a octanol/water"
"$AMSBIN/amsreport" step6a.crskf logp

```

(continues on next page)

(continued from previous page)

```

echo "Step 6b benzene/water"
"$AMSBIN/amsreport" step6b.crskf logp

# Step 7: Solubility
# -----

# In step 7 of the example the solubility of a compound is calculated.
# The solute can either be a liquid, solid, or gas.
# First some pure compound properties for benzene are set: the melting point,
# enthalpy of fusion, and the boiling point (file benzene.coskf).

"$AMSBIN/crsprep" -c benzene.coskf -meltingpoint 278.7 -hfusion 2.37 -savecompound
"$AMSBIN/crsprep" -c benzene.coskf -tvap 353.3 -pvap 1.01325 -savecompound

# In step 7a the solubility of benzene in water is calculated for a range of
# temperatures ranging from 273.15 K to 373.15 K in 10 steps (result file step7a.
→ crskf).
# If the template PURESOLUBILITY is used, the special compound is the solute,
# benzene in this case.
# Below 278.7 K, benzene is a solid. This will be taken into account, since the
# melting point and enthalpy of fusion are present on the file benzene.coskf.
# At higher temperatures benzene is assumed to be a liquid.
# Note that in this calculation above the normal boiling point of benzene
# the vapor pressure of benzene will be higher than 1.01325 bar.

"$AMSBIN/crsprep" -t PURESOLUBILITY -temperature 273.15 -temperature 373.15 -j step7a.
→ \
    -s benzene.coskf \
    -c water.coskf > job.sh

# In step 7b again the solubility of benzene in water is calculated for a (different)
# range of temperatures (result file step7b.crs) using the template SOLUBILITY.
# If the template SOLUBILITY is used, the special compound is the solvent,
# water in this case.
# For the density of the solvent water 1.0 kg/L is used.
# Below 278.7 K benzene is a solid. This will be taken into account, since the melting
# point and enthalpy of fusion are present on the file benzene.coskf.
# At higher temperatures benzene is assumed to be a liquid.

"$AMSBIN/crsprep" -t SOLUBILITY -temperature 273.15 -temperature 283.15 \
    -densitysolvent 1.0 -j step7b \
    -s water.coskf \
    -c benzene.coskf >> job.sh

# In step 7c again the solubility of benzene in water is calculated for a range
# of temperatures above the boiling point of benzene (result file step7c.crs)
# using the template SOLUBILITY.
# If the template SOLUBILITY is used, the special compound is the solvent,
# water in this case.
# For the density of water 1.0 kg/L is used.
# For the vapor pressure of benzene 1.01325 bar is used.

"$AMSBIN/crsprep" -t SOLUBILITY -temperature 353.3 -temperature 373.15 \
    -densitysolvent 1.0 -solphase Gas -pressure 1.01325 -j step7c \
    -s water.coskf \
    -c benzene.coskf >> job.sh

```

(continues on next page)

(continued from previous page)

```

# The solubility of a gas in a solvent can also be calculated using Henry's law,
# which is valid for ideal dilute solutions.
# Henry coefficients can be calculated with the template ACTIVITYCOEF.
# In step 7d of the example the infinite diluted solutes benzene is calculated
# in the solvent water (result file step7d.crskf)
# at a temperature of 363.15 K.
# If the template ACTIVITYCOEF is used, the special compound is the solvent,
# water in this case.
# For the density of water 1.0 kg/L is used.

"$AMSBIN/crsprep" -t ACTIVITYCOEF -temperature 363.15 -densitysolvent 1.0 -j step7d \
    -s water.coskf \
    -c benzene.coskf >> job.sh

# Next the job.sh is run which will produce the crskf files,
# and a report is made for all calculations in step 7.

./job.sh
echo "Step 7a"
"$AMSBIN/amsreport" step7a.crskf solubility-x
echo "Step 7b"
"$AMSBIN/amsreport" step7b.crskf solubility-g
echo "Step 7c"
"$AMSBIN/amsreport" step7c.crskf solubility-m
echo "Step 7d"
"$AMSBIN/amsreport" step7d.crskf henry

# Step 8: Binary mixtures VLE/LLE
# -----

# In step 8 phase diagrams of a mixture of two components are be calculated
# with the template BINMIXCOEF.
# Exactly two compound should be given.
# In step 8a of the example a binary mixture of water and methanol
# is calculated at 298.14 K (result file step8a.crskf).

"$AMSBIN/crsprep" -t BINMIXCOEF -temperature 298.14 -j step8a \
    -s water.coskf \
    -s methanol.coskf > job.sh

# In step 8b of the example a binary mixture of water and ethanol is calculated
# at 322.45 K (result file step8b.crskf).
# Pure compound vapor pressures are given with -tvap (temperature in K)
# and -pvap (vapor pressure in bar).
# Preferably both -tvap and -pvap should be included for both compounds.
# If only one -tvap and one -pvap is given, it is assumed to be for the first
↔ compound.
# Note that these pure compound values are not saved to water.coskf or
# ethanol.coskf in this case.

"$AMSBIN/crsprep" -t BINMIXCOEF -temperature 322.45 -j step8b \
    -s water.coskf -tvap 322.45 -pvap 0.123416 \
    -s ethanol.coskf -tvap 322.45 -pvap 0.294896 >> job.sh

# In step 8c of the example a binary mixture of water and benzene

```

(continues on next page)

(continued from previous page)

```

# is calculated at 323.15 K (result file step8c.crskf).
# Water and benzene do not mix very well. In this case a
# liquid-liquid equilibrium (LLE) will be calculated.
# The number of mixtures for which the binary mixture is calculated should be not too
→small,
# otherwise the properties of the 2 immiscible liquid phases will not be so accurate.
# In this case for the number of mixtures 100 is chosen.
# The actual number of mixtures is 5 more, thus 105 in this case.

"$AMSBIN/crsprep" -t BINMIXCOEF -temperature 323.15 -n 100 -j step8c \
    -s water.coskf \
    -s benzene.coskf >> job.sh

# In step 8d of the example a binary mixture of methanol and ethanol is calculated
# at a constant total vapor pressure (-iso isobar) of 1.01325 bar (result file step8d.
→crskf).
# Pure compound vapor pressures are given with -tvap (temperature in K)
# and -pvap (vapor pressure in bar).
# Preferably both -tvap and -pvap should be included for both compounds.
# If only one -tvap and one -pvap is given, it is assumed to be for the first
→compound.
# Note that these pure compound values are not saved to methanol.coskf
# or ethanol.coskf in this case.

"$AMSBIN/crsprep" -t BINMIXCOEF -iso isobar -pressure 1.01325 -j step8d \
    -s methanol.coskf -tvap 338 -pvap 1.01325 \
    -s ethanol.coskf -tvap 351 -pvap 1.01325 >> job.sh

# Next the job.sh is run which will produce the crskf files,
# and a report is made for all calculations in step 8.

./job.sh
echo "Step 8a"
"$AMSBIN/amsreport" step8a.crskf molar-fraction
"$AMSBIN/amsreport" step8a.crskf excess-g
"$AMSBIN/amsreport" step8a.crskf excess-h
echo "Step 8b"
"$AMSBIN/amsreport" step8b.crskf vapor-pressure
echo "Step 8c"
"$AMSBIN/amsreport" step8c.crskf miscibility-gap
"$AMSBIN/amsreport" step8c.crskf miscibility-gap-x
"$AMSBIN/amsreport" step8c.crskf miscibility-gap-a
echo "Step 8d"
"$AMSBIN/amsreport" step8d.crskf temperature

# Step 9: Ternary mixtures VLE/LLE
# -----
#
# In step 9 phase diagrams of a mixture of three components are be calculated
# with the template TERNARYMIX.
# Exactly three compound should be given.
# For convenience first some pure compound properties (normal boiling points)
# are saved to the .coskf files.

"$AMSBIN/crsprep" -c water.coskf -tvap 373.15 -pvap 1.01325 -savecompound

```

(continues on next page)

(continued from previous page)

```

"$AMSBIN/crsprep" -c methanol.coskf -tvap 338 -pvap 1.01325 -savecompound
"$AMSBIN/crsprep" -c ethanol.coskf -tvap 351 -pvap 1.01325 -savecompound
"$AMSBIN/crsprep" -c benzene.coskf -tvap 353.3 -pvap 1.01325 -savecompound

# In step 9a of the example a ternary mixture of methanol, ethanol, and benzene
# is calculated at 343.15 K (result file step9a.crskf).
# In step 9b of the example a ternary mixture of water, ethanol, and benzene is
↳calculated
# at a constant total vapor pressure (-iso isobar) of 1.01325 bar (result file step9b.
↳crskf).
# In step 9b a miscibility gap of the ternary mixture will be calculated.
# In this case, within the miscibility gap there are two immiscible phases of
# the liquid in equilibrium.
# The composition of the two phases, which are in equilibrium,
# can be found at the end points of the tie lines, that are calculated.

"$AMSBIN/crsprep" -t TERNARYMIX -temperature 343.15 -j step9a \
-s methanol.coskf -s ethanol.coskf -s benzene.coskf > job.sh
"$AMSBIN/crsprep" -t TERNARYMIX -iso isobar -pressure 1.01325 -j step9b \
-s water.coskf -s ethanol.coskf -s benzene.coskf >> job.sh
./job.sh
echo "Step 9a"
"$AMSBIN/amsreport" step9a.crskf molar-fraction
"$AMSBIN/amsreport" step9a.crskf pressure
"$AMSBIN/amsreport" step9a.crskf miscibility-gap
echo "Step 9b"
"$AMSBIN/amsreport" step9b.crskf miscibility-gap
echo "First 4 tie-lines"
"$AMSBIN/amsreport" step9b.crskf -r "TERNARYMIX%x11#1:24#12.4f##6"
"$AMSBIN/amsreport" step9b.crskf temperature

# Note that for printing all tie-lines that are calculated in step9b one can simply
↳use
# "$AMSBIN/amsreport" step9b.crskf tie-lines-x
# instead of the more complicated
# "$AMSBIN/amsreport" step9b.crskf -r "TERNARYMIX%x11#1:24#12.4f##6"
# which also only reports the first 4 tie-lines

# Step 10: A composition line between solvents s1 and s2
# -----

# In step 10 a phase diagram of a mixture of two solvents, which both are mixtures,
# is calculated with the template COMPOSITIONLINE.
# In this step one of the tie lines of the ternary mixture of water, ethanol, and
↳benzene
# of step 9b will be investigated in more detail.
# Note that here the .coskf files are used in which the normal boiling points were
↳saved to (in step 9).
# The mixture will be calculated for a list of molar (or mass) fractions of the
↳solvents
# between zero and one, and the compositions of solvent 1 and solvent 2 are linearly
↳interpolated.
# In this case solvent 1 consists of 0.3 molar fraction ethanol and 0.7 molar
↳fraction benzene, and
# solvent 2 consists of 0.9 molar fraction water and 0.1 molar fraction ethanol.
# In step 10a of the example this mixture is calculated at a constant total vapor
↳pressure (-iso isobar)

```

(continues on next page)

(continued from previous page)

```

# of 1.01325 bar (result file step10a.crskf).
# In step 10b of the example this mixture is calculated at 341.05 K (result file_
↪step10b.crskf).
# Why this temperature was chosen can be found in step 10 of COSMO-RS GUI Tutorial:↪
↪Properties.

"$AMSBIN/crsprep" -t COMPOSITIONLINE -iso isobar -pressure 1.01325 -n 100 -j step10a \
-s water.coskf -frac1 0.0 -frac2 0.9 \
-s ethanol.coskf -frac1 0.3 -frac2 0.1 \
-s benzene.coskf -frac1 0.7 -frac2 0.0 > job.sh
"$AMSBIN/crsprep" -t COMPOSITIONLINE -temperature 341.05 -n 100 -j step10b \
-s water.coskf -frac1 0.0 -frac2 0.9 \
-s ethanol.coskf -frac1 0.3 -frac2 0.1 \
-s benzene.coskf -frac1 0.7 -frac2 0.0 >> job.sh

./job.sh
echo "Step 10a"
$AMSBIN/amsreport step10a.crskf ncomp
$AMSBIN/amsreport step10a.crskf frac1
$AMSBIN/amsreport step10a.crskf frac2
$AMSBIN/amsreport step10a.crskf solvent-fraction
$AMSBIN/amsreport step10a.crskf molar-fraction
$AMSBIN/amsreport step10a.crskf activity-coefficient

echo "Step 10b"
$AMSBIN/amsreport step10b.crskf Gibbs-energy-mixing

# Step 11: Phase Stability Test and LLE
# -----

"$AMSBIN/crsprep" -t STABILITY -temperature 298.15 -j step11a \
-s water.coskf -frac1 0.4 -s ethanol.coskf -frac1 0.4 -s benzene.
↪coskf -frac1 0.2 > job.sh
"$AMSBIN/crsprep" -t LLE -temperature 298.15 -j step11b \
-s water.coskf -frac1 0.4 -s ethanol.coskf -frac1 0.4 -s benzene.
↪coskf -frac1 0.2 >> job.sh

./job.sh
echo "Step 11a"
$AMSBIN/amsreport step11a.crskf ncomp
$AMSBIN/amsreport step11a.crskf UNSTABLE

echo "Step 11b"
$AMSBIN/amsreport step11b.crskf ncomp
$AMSBIN/amsreport step11b.crskf UNSTABLE
$AMSBIN/amsreport step11b.crskf MOLAR-FRACTION-PHASE-I
$AMSBIN/amsreport step11b.crskf MOLAR-FRACTION-PHASE-II
$AMSBIN/amsreport step11b.crskf ACTIVITY-COEFFICIENT-PHASE-I
$AMSBIN/amsreport step11b.crskf ACTIVITY-COEFFICIENT-PHASE-II

# Step 12: Pure Compound Properties
# -----

# In step 11 a QSPR (Quantitative Structure-Property Relationship) method is used
# to estimate some pure compound properties.
# This QSPR method needs a SMILES string as input.

```

(continues on next page)

(continued from previous page)

```

echo "Step 12"
"$AMSBIN"/prop_prediction --smiles "c1ccccc1" --boilingpoint -d

# Step 13: Solvent Optimizations: Optimize Solubility
# -----

# In this step a solvent is optimized in order to maximize or minimize
# the mole fraction solubility of a solid solute in the liquid mixture.

echo "Step 13"
"$AMSBIN"/solvent_opt -t SOLUBILITY -method COSMO-RS -temperature 273.15 -max \
-c "benzene.coskf" -solute -meltingpoint 278.7 -hfusion 2.37 \
-c "ethanol.coskf" -c "methanol.coskf" -c "water.coskf" -d >_
↪max_solubility
grep benzene.coskf max_solubility
grep " ethanol.coskf" max_solubility
"$AMSBIN"/solvent_opt -t SOLUBILITY -method COSMO-RS -temperature 273.15 -min \
-c "benzene.coskf" -solute -meltingpoint 278.7 -hfusion 2.37 \
-c "ethanol.coskf" -c "methanol.coskf" -c "water.coskf" -d >_
↪min_solubility
grep benzene.coskf min_solubility
grep water.coskf min_solubility

# Step 14: Solvent Optimizations: Optimize Liquid-Liquid Extraction
# -----

# In this step a mixture of immiscible solvents is optimized in order to maximize or_
↪minimize
# the distribution ratio (D) of two solutes between the two liquid phases.

"$AMSBIN"/solvent_opt -t LLEXTRACTION -method COSMO-RS -multistart 10 -temperature_
↪298.15 -max \
-c "water.coskf" -solute -c "acetic_acid.coskf" -solute \
-c "benzene.coskf" -c "water.coskf" -c "2-hexanone.coskf" -c
↪"ethanol.coskf" -d > max_lle

echo "Step 14"
"$AMSBIN/amsreport" CRSKF -r "OPT_LLEXTRACTION%obj#8.3f"

echo "Ready"

```

### 11.5.4 Example: The COSMO-RS compound database

Download Database.run

```

#!/bin/sh

# This example tries to do the same as in the COSMO-RS GUI Tutorial: The COSMO-RS_
↪compound database,
# using scripts.
# In many of the next examples fewer compounds are included than in the COSMO-RS GUI_
↪Tutorial.

# First some .coskf files are copied to the location where the scripts are running.

```

(continues on next page)

(continued from previous page)

```

# These .coskf files are a sample of the full database.

echo "Results"

cp $AMSHOME/examples/COSMO-RS/Database/*coskf .

rm -f job.sh
touch job.sh
chmod +x job.sh

# 4.2 Octanol-Water partition coefficients (log P_OW)
# -----

"$AMSBIN/crsprep" -t LOGP -j t4.2 \
                  -c Methanol.coskf > job.sh
./job.sh
echo "4.2"
"$AMSBIN/amsreport" t4.2.crskf compounds-name
"$AMSBIN/amsreport" t4.2.crskf logp

# 4.3: Henry's law constants
# -----

"$AMSBIN/crsprep" -t ACTIVITYCOEF -temperature 293.15 -densitysolvent 0.998 -j t4.3a \
                  -s Water.coskf \
                  -c Acetone.coskf -c Benzene.coskf -c Ethanol.coskf -c Methanol.
↪coskf > job.sh
./job.sh
echo "4.3a"
"$AMSBIN/amsreport" t4.3a.crskf compounds-name
"$AMSBIN/amsreport" t4.3a.crskf henry

"$AMSBIN/crsprep" -s Acetone.coskf -pvap 0.246 -tvap 293.15 -savecompound
"$AMSBIN/crsprep" -s Benzene.coskf -pvap 0.100 -tvap 293.15 -savecompound
"$AMSBIN/crsprep" -s Ethanol.coskf -pvap 0.059 -tvap 293.15 -savecompound
"$AMSBIN/crsprep" -s Methanol.coskf -pvap 0.129 -tvap 293.15 -savecompound
"$AMSBIN/crsprep" -s Water.coskf -pvap 0.02536 -tvap 293.15 -savecompound

"$AMSBIN/crsprep" -t ACTIVITYCOEF -temperature 293.15 -densitysolvent 0.998 -j t4.3b \
                  -s Water.coskf \
                  -c Benzene.coskf -c Ethanol.coskf -c Methanol.coskf > job.sh
./job.sh
echo "4.3b"
"$AMSBIN/amsreport" t4.3b.crskf henry

# 4.4: Solubility of Vanillin in organic solvents
# -----

"$AMSBIN/crsprep" -s Vanillin.coskf -meltingpoint 355 -hfusion 5.35 -savecompound
"$AMSBIN/crsprep" -t PURESOLUBILITY -j t4.4 \
                  -s Vanillin.coskf \
                  -c Ethanol.coskf -c Methanol.coskf -c Water.coskf > job.sh
./job.sh
echo "4.4"

```

(continues on next page)

(continued from previous page)

```

"$AMSBIN/amsreport" t4.4.crskf compounds-name
"$AMSBIN/amsreport" t4.4.crskf solubility-x
"$AMSBIN/amsreport" t4.4.crskf solubility-m

# 4.5: Binary mixture of Methanol and Hexane
# -----

"$AMSBIN/crsprep" -s Methanol.coskf -pvap 0.845 -tvap 333.15 -savecompound
"$AMSBIN/crsprep" -s Hexane.coskf -pvap 0.77 -tvap 333.15 -savecompound
"$AMSBIN/crsprep" -t BINMIXCOEF -temperature 333.15 -n 100 -j t4.5a \
    -s Methanol.coskf \
    -s Hexane.coskf > job.sh

./job.sh
echo "4.5a"
"$AMSBIN/amsreport" t4.5a.crskf compounds-name
"$AMSBIN/amsreport" t4.5a.crskf molar-fraction
"$AMSBIN/amsreport" t4.5a.crskf pressure
"$AMSBIN/amsreport" t4.5a.crskf miscibility-gap
"$AMSBIN/amsreport" t4.5a.crskf miscibility-gap-x
"$AMSBIN/amsreport" t4.5a.crskf miscibility-gap-a

"$AMSBIN/crsprep" -s Methanol.coskf -pvap 1.01325 -tvap 337.8 -savecompound
"$AMSBIN/crsprep" -s Hexane.coskf -pvap 1.01325 -tvap 342 -savecompound
"$AMSBIN/crsprep" -t BINMIXCOEF -pressure 1.01325 -iso isobar -n 100 -j t4.5b \
    -s Methanol.coskf \
    -s Hexane.coskf > job.sh

./job.sh
echo "4.5b"
"$AMSBIN/amsreport" t4.5b.crskf molar-fraction
"$AMSBIN/amsreport" t4.5b.crskf temperature
"$AMSBIN/amsreport" t4.5b.crskf miscibility-gap
"$AMSBIN/amsreport" t4.5b.crskf miscibility-gap-x
"$AMSBIN/amsreport" t4.5b.crskf miscibility-gap-a

# 4.6: Large infinite dilution activity coefficients in Water
# -----

"$AMSBIN/crsprep" -t ACTIVITYCOEF -temperature 298.15 -j t4.6 \
    -s Water.coskf \
    -c Benzene.coskf > job.sh

./job.sh
echo "4.6"
"$AMSBIN/amsreport" t4.6.crskf compounds-name
"$AMSBIN/amsreport" t4.6.crskf activity-coefficient

# 4.7: Parametrization of ADF COSMO-RS: solvation energies,
# vapor pressures, partition coefficients
# -----

cat << eor > small.compoundlist
Hexane.coskf
Methanol.coskf
Ethanol.coskf
Acetone.coskf
Benzene.coskf
Water.coskf

```

(continues on next page)

(continued from previous page)

```

eor
"$AMSBIN/crsprep" -t ACTIVITYCOEF -temperature 298.15 -densitysolvent 0.997 -j t4.7a \
    -inputpurevap 0 \
    -s Water.coskf \
    -c small.compoundlist > job.sh
./job.sh
echo "4.7a"
"$AMSBIN/amsreport" t4.7a.crskf compounds-name
"$AMSBIN/amsreport" t4.7a.crskf gibbs-energy-solvation

"$AMSBIN/crsprep" -t PUREVAPORPRESSURE -temperature 298.15 -j t4.7b \
    -c small.compoundlist -inputpurevap 0 > job.sh
./job.sh
echo "4.7b"
"$AMSBIN/amsreport" t4.7b.crskf compounds-name
"$AMSBIN/amsreport" t4.7b.crskf vapor-pressure

"$AMSBIN/crsprep" -t LOGP -j t4.7c -c small.compoundlist > job.sh
./job.sh
echo "4.7c Octanol/Water"
"$AMSBIN/amsreport" t4.7c.crskf compounds-name
"$AMSBIN/amsreport" t4.7c.crskf logp

"$AMSBIN/crsprep" -t LOGP -preset 5 -j t4.7d -c small.compoundlist > job.sh
./job.sh
echo "4.7d Hexane/Water"
"$AMSBIN/amsreport" t4.7d.crskf logp

"$AMSBIN/crsprep" -t LOGP -preset 3 -j t4.7e -c small.compoundlist > job.sh
./job.sh
echo "4.7e Benzene/Water"
"$AMSBIN/amsreport" t4.7e.crskf logp

"$AMSBIN/crsprep" -t LOGP -preset 4 -j t4.7f -c small.compoundlist > job.sh
./job.sh
echo "4.7f Ether/Water"
"$AMSBIN/amsreport" t4.7f.crskf logp

# 4.8: COSMO-SAC 2013-ADF
# -----

"$AMSBIN/crsprep" -t PURESIGMAPROFILE -method COSMOSAC2013 -c Water.coskf \
    -c Ethanol.coskf -j t4.8a > job.sh
./job.sh
echo "4.8a"
"$AMSBIN/amsreport" t4.8a.crskf sigma
"$AMSBIN/amsreport" t4.8a.crskf sigma-profile
"$AMSBIN/amsreport" t4.8a.crskf sigma-profile-hb

"$AMSBIN/crsprep" -s Methanol.coskf -density 0.7918 -savecompound
"$AMSBIN/crsprep" -s Ethanol.coskf -density 0.789 -savecompound
"$AMSBIN/crsprep" -s Acetone.coskf -density 0.791 -savecompound

"$AMSBIN/crsprep" -t LOGP -method COSMOSAC2013 -j t4.8b \
    -c Methanol.coskf -c Ethanol.coskf -c Acetone.coskf > job.sh

```

(continues on next page)

(continued from previous page)

```

./job.sh
echo "4.8b Octanol/Water"
"$AMSBIN/amsreport" t4.8b.crskf compounds-name
"$AMSBIN/amsreport" t4.8b.crskf logp

"$AMSBIN/crsprep" -t LOGP -method COSMOSAC2013 -preset 5 -j t4.8c \
                  -c Methanol.coskf -c Ethanol.coskf -c Acetone.coskf > job.sh

./job.sh
echo "4.8c Hexane/Water"
"$AMSBIN/amsreport" t4.8c.crskf logp

"$AMSBIN/crsprep" -t LOGP -method COSMOSAC2013 -preset 3 -j t4.8d \
                  -c Methanol.coskf -c Ethanol.coskf -c Acetone.coskf > job.sh

./job.sh
echo "4.8d Benzene/Water"
"$AMSBIN/amsreport" t4.8d.crskf logp

"$AMSBIN/crsprep" -t LOGP -method COSMOSAC2013 -preset 4 -j t4.8e \
                  -c Methanol.coskf -c Ethanol.coskf -c Acetone.coskf > job.sh

./job.sh
echo "4.8e Ether/Water"
"$AMSBIN/amsreport" t4.8e.crskf logp

"$AMSBIN/crsprep" -s Acetone.coskf -density 0.791 -pvap 3.7      -tvap 373.15 -
↳savecompound
"$AMSBIN/crsprep" -s Water.coskf   -density 0.997 -pvap 1.01325 -tvap 373.15 -
↳savecompound

"$AMSBIN/crsprep" -t BINMIXCOEF -method COSMOSAC2013 -temperature 373.15 -n 20 -j t4.
↳8f \
                  -s Acetone.coskf \
                  -s Water.coskf > job.sh

./job.sh
echo "4.8f"
"$AMSBIN/amsreport" t4.8f.crskf molar-fraction
"$AMSBIN/amsreport" t4.8f.crskf vapor-pressure
"$AMSBIN/amsreport" t4.8f.crskf pressure

echo "Ready"

```

### 11.5.5 Example: pKa values

Download pKa.run

```

#!/bin/sh

# This example tries to do the same as in the COSMO-RS GUI Tutorial: pKa values,
# using scripts
# First some .coskf files are copied to the location where the scripts are running,
# next the jobs are prepared and the reports are made.
# Check the COSMO-RS GUI Tutorial: pKa values to see what to do with the results.

echo "Results"

rm -f job.sh

```

(continues on next page)

(continued from previous page)

```

cp $AMSHOME/examples/COSMO-RS/pKa/*.coskf .
cp $AMSHOME/examples/COSMO-RS/pKa/*compoundlist .

touch job.sh
chmod +x job.sh

echo "Acids"
"$AMSBIN/crsprep" -t ACTIVITYCOEF -j t5.1a \
                  -c tutorial5.1_acid.compoundlist -no_pdh > job.sh
./job.sh
"$AMSBIN/amsreport" t5.1a.crskf compounds-name
"$AMSBIN/amsreport" t5.1a.crskf Activity-Coefficient
"$AMSBIN/amsreport" t5.1a.crskf Gibbs-energy-solvation
"$AMSBIN/amsreport" t5.1a.crskf Gibbs-energy-solute

"$AMSBIN/crsprep" -t ACTIVITYCOEF -j t5.1b \
                  -c tutorial5.1_base.compoundlist -no_pdh > job.sh
./job.sh
"$AMSBIN/amsreport" t5.1b.crskf compounds-name
"$AMSBIN/amsreport" t5.1b.crskf Activity-Coefficient
"$AMSBIN/amsreport" t5.1b.crskf Gibbs-energy-solvation
"$AMSBIN/amsreport" t5.1b.crskf Gibbs-energy-solute

echo "Ready"

```

## 11.5.6 Example: Polymers

Download Polymers.run

```

#!/bin/sh

# This example tries to do the same as in the COSMO-RS GUI Tutorial: Polymers,
# using scripts.

# First some .coskf files are copied to the location where the scripts are running.
# Some of the .coskf files are a sample of the full polymer database ADFCRS-POLYMERS-
→2019.
# In the first part of the example crsprep is used to set pure compound parameters.
# Pure compound densities are required for every species in a calculation involving_
→any polymers.
# For some pure compounds the VPM1 vapor pressure equation is set including vapor_
→pressure coefficients.

echo "Results"

cp $AMSHOME/examples/COSMO-RS/Database/*.coskf .
cp $AMSHOME/examples/COSMO-RS/Polymers/*.coskf .

"$AMSBIN/crsprep" -c Benzene.coskf -density 0.876 -savecompound
"$AMSBIN/crsprep" -c Water.coskf -density 1.0 -savecompound
"$AMSBIN/crsprep" -c Methanol.coskf -density 0.792 -savecompound \
                  -vp_equation VPM1 -vp_params "-7057.597287 -9.936895562 0.
→00608530790 77.10002032 0"
"$AMSBIN/crsprep" -c Hexane.coskf -density 0.655 -savecompound \
                  -vp_equation VPM1 -vp_params "-5575.417318 -6.612402250 0.

```

(continues on next page)

(continued from previous page)

```
↪00437376138 53.36969532 0"

rm -f job.sh
touch job.sh
chmod +x job.sh

"$AMSBIN/crsprep" -t ACTIVITYCOEF -j Polymers.1 \
  -s "Poly(ethylene).coskf" -frac1 0.5 -s Benzene.coskf -frac1 0.5 \
  -massfraction > job.sh

./job.sh
echo "Polymers.1"
"$AMSBIN/amsreport" Polymers.1.crskf compounds-name Flory-Huggins WF-activity-
↪coefficient

"$AMSBIN/crsprep" -t VAPORPRESSURE -temperature 298.15 -temperature 398.15 -n 10 -j_
↪Polymers.2 \
  -s "Poly(dimethylsiloxane).coskf" -frac1 0.5 -s Methanol.coskf -
↪frac1 0.25 \
  -s Hexane.coskf -frac1 0.25 \
  -massfraction > job.sh

./job.sh
echo "Polymers.2"
"$AMSBIN/amsreport" Polymers.2.crskf compounds-name temperature pressure

"$AMSBIN/crsprep" -t LOGP -preset 0 -j Polymers.3 \
  -s "Poly(ethylene).coskf" -s Water.coskf \
  -c Methanol.coskf > job.sh

./job.sh
echo "Polymers.3"
"$AMSBIN/amsreport" Polymers.3.crskf logp

"$AMSBIN/crsprep" -t PURESOLUBILITY -temperature 398.15 -temperature 498.15 -n 10 -j_
↪Polymers.4 \
  -s Hexane.coskf -solphase Gas -pressure 1.01325 \
  -c "Poly(styrene).coskf" > job.sh

./job.sh
echo "Polymers.4"
"$AMSBIN/amsreport" Polymers.4.crskf solubility-g

"$AMSBIN/crsprep" -t BINMIXCOEF -method COSMOSAC2013 -j Polymers.5 \
  -s "Poly(ethyl_ethylene).coskf" -s Benzene.coskf \
  -massfraction > job.sh

./job.sh
echo "Polymers.5"
"$AMSBIN/amsreport" Polymers.5.crskf polymer-fraction Flory-Huggins

echo "Ready"
```

## REQUIRED CITATIONS

When you publish results in the scientific literature that were obtained with programs of the ADF package, you are required to include references to the program package with the appropriate release number, and a few key publications.

### 12.1 General References

For calculations with the COSMO-RS program, version 2023.1:

1. C.C. Pye, T. Ziegler, E. van Lenthe, J.N. Louwen, *An implementation of the conductor-like screening model of solvation within the Amsterdam density functional package. Part II. COSMO for real solvents*, *Can. J. Chem.* **87**, 790 (2009) (<https://doi.org/10.1139/V09-008>)
2. AMS 2026.1 COSMO-RS, SCM, Theoretical Chemistry, Vrije Universiteit, Amsterdam, The Netherlands, <https://www.scm.com> Optionally, you may add the following list of authors and contributors: J.N. Louwen, C.C. Pye, E. van Lenthe, N.D. Austin, E.S. McGarrity, R. Xiong, S.I. Sandler, R.I. Burnett

If you use COSMO-SAC 2013-ADF you must also add

3. R. Xiong, S.I. Sandler, R.I. Burnett, *An improvement to COSMO-SAC for predicting thermodynamic properties*, *Ind. Eng. Chem. Res.* **53**, 8265 (2014) (<https://doi.org/10.1021/ie404410v>)

### 12.2 Solvent Optimizations

For solvent optimizations:

- N.D. Austin, N.V. Sahinidis, D.W. Trahan, *COSMO-based computer-aided molecular/mixture design: A focus on reaction solvents*, *AIChE Journal* **64**, 104 (2018) (<https://doi.org/10.1002/aic.15871>)

### 12.3 External programs and Libraries

Third party software used in the 2026.1 version of the Amsterdam Modeling Suite can be found in the file titled “third-party-software.txt” in the root of your AMS installation.



## KEYWORDS

- *COMPOUND* (page 42)
- *COSMOSAC* (page 38)
- *COSMOSAC2013* (page 38)
- *COSMOSACDHB* (page 38)
- *CRSPARAMETERS* (page 35)
- *DISPERSION* (page 37)
- *EPSILON* (page 40)
- *MASSFRACTION* (page 44)
- *PRESSURE* (page 44)
- *PROPERTY activitycoef* (page 47)
- *PROPERTY binmixcoef* (page 55)
- *PROPERTY boilingpoint* (page 45)
- *PROPERTY compositionline* (page 56)
- *PROPERTY flashpoint* (page 46)
- *PROPERTY lle* (page 51)
- *PROPERTY logp* (page 46)
- *PROPERTY pureboilingpoint* (page 45)
- *PROPERTY puresigmapotential* (page 57)
- *PROPERTY puresigmaprofile* (page 57)
- *PROPERTY puresolubility* (page 47)
- *PROPERTY purevaporpressure* (page 45)
- *PROPERTY sigmapotential* (page 57)
- *PROPERTY sigmaprofile* (page 57)
- *PROPERTY solubility* (page 47)
- *PROPERTY stability* (page 48)
- *PROPERTY ternarymix* (page 56)
- *PROPERTY vaporpressure* (page 45)
- *TECHNICAL* (page 41)
- *TEMPERATURE* (page 44)
- *PDH\_CORRECTION* (page 58)
- *USEPOLYCOMBIFORPOLYMER* (page 58)
- *REUSE\_SIGMA\_PROFILE* (page 59)
- *UPDATE\_SIGMA\_PROFILE* (page 59)



## 14.1 I want to include solvent effects, do I need COSMO-RS?

You do not need COSMO-RS to do ADF calculations with continuum solvation (see [COSMO vs COSMO-RS FAQ](#)). If you want to calculate thermodynamic properties like partition coefficients, VLE, solubilities, etc. you do need COSMO-RS. COSMO-RS usually also yields more accurate solvation free energies and pKa values than COSMO.

## 14.2 Do I need to use the ADF module with COSMO-RS?

Many researchers and engineers use ADF together with COSMO-RS. If you want to add new compounds that are not yet available in our database, the recommended approach is to use the **COSMO-RS Compound task**, because COSMO-RS is sensitive to the sigma profile and its parameters are calibrated for profiles generated in this way.

For simpler cases, **FastSigma** can estimate the sigma profile directly from the SMILES representation of your compound. However, it may not support compounds containing uncommon elements or functional groups, and it does not support ionic compounds.

If **FastSigma** is not suitable for your compound, you may consider the **MOPAC** in our DFTB module, which is faster than ADF.

In practice, the expected accuracy is generally in the following order: **COSMO-RS Compound task** > **FastSigma** > **MOPAC**. For more details, please see [COSMO result files](#).

### In particular:

- Steps 1–3 describe the COSMO-RS Compound task
- Step 6 describes the MOPAC approach
- Step 7 describes FastSigma

## 14.3 Can I use a .cosmo file from COSMOlogic, Dmol3, or Gaussian with ADF COSMO-RS?

In principle, yes, but you are advised to rerun the COSMO calculation with ADF rather than using sigma profiles from Turbomole, Gaussian, DMol3 or another code. Since ADF uses a slightly different procedure (gas phase optimization followed by COSMO calculation with an infinite dielectric constant) as well as different technical settings (scalar relativistic, triple-zeta Slater-type basis set, integration, surface charge generation with a finer grid), our COSMO-RS implementation has been reparametrized specifically for use with sigma profiles generated in that way. Therefore, you are advised to follow the set procedure in ADF, easiest is to use the COSMO-RS preset in the ADF GUI. See for details: C.C. Pye, T. Ziegler, E. van Lenthe, J.N. Louwen, An implementation of the conductor-like screening model of

solvation within the Amsterdam density functional package. Part II. COSMO for real solvents, *Can. J. Chem.*87, 790 (2009) (<https://dx.doi.org/10.1139/V09-008>). If you do insist on using sigma profiles generated with the other programs with different defaults, you can use the *cosmo2kf* (page 213) utility to convert the plain ascii cosmo files to our binary file format. There are also *scripting tools for COSMO-RS* (page 209) to facilitate screening solvent (combinations) for activities, partition coefficients, VLE, etc.

## 14.4 For which compounds do you have sigma profiles in the database?

We have text files for the *regular compounds* ([https://www.scm.com/wp-content/uploads/COSMO-RS\\_compounds\\_database.txt](https://www.scm.com/wp-content/uploads/COSMO-RS_compounds_database.txt)) and *ionic liquids* ([https://www.scm.com/wp-content/uploads/COSMO-RS\\_ionic\\_liquids\\_database.txt](https://www.scm.com/wp-content/uploads/COSMO-RS_ionic_liquids_database.txt)). If you install the databases you can also search on elemental composition.

## 14.5 I have COSMO-RS download permissions but I can not automatically download the COSMO-RS databases?

Some firewalls prevent the automatic download of the COSMO-RS database. With your SCM User ID and password you can *download the compounds database manually* (page 24) (see also *video on how to install it on Windows* ([https://downloads.scm.com/distr/adfcrs\\_from\\_zip.mp4](https://downloads.scm.com/distr/adfcrs_from_zip.mp4)) for AMS2021).

## PYTHON MODULE INDEX

### p

`pyCRS.CRSManager`, 117  
`pyCRS.Database`, 103  
`pyCRS.FastSigma`, 125  
`pyCRS.Input`, 126  
`pyCRS.Output`, 126  
`pyCRS.PropPred`, 123



## A

activity coefficients, 46  
 add\_compound() (*pyCRS.Database.COSKFDatabase* method), 103  
 add\_Mixture() (*pyCRS.CRSManger.CRSSystem* method), 118  
 add\_physical\_property() (*pyCRS.Database.COSKFDatabase* method), 104  
 additional\_sett (*pyCRS.CRSManger.CRSMixture* attribute), 122  
 ADF COSMO calculation, 19  
 ADF COSMO settings, 19  
 ADFCRS-2010, 25  
 ADFCRS-2018, 24  
 ADFCRS-IL-2014, 26  
 ADFCRS-POLYMERS-2019, 26  
 adf.rkf file, 19  
 adopt\_smiles (*pyCRS.Database.PropPredRow* attribute), 116  
 amsprep module, 209  
 amsreport module, 211  
 available\_properties (*in module pyCRS.PropPred*), 123

## B

binary mixture, 55  
 boiling point, 45  
 boilingpoint (*pyCRS.Database.PhysicalPropertyRow* attribute), 114  
 boilingpoint (*pyCRS.Database.PropPredRow* attribute), 116

## C

calculation of properties, 12  
 cas (*pyCRS.Database.CompoundRow* attribute), 111  
 cas (*pyCRS.Database.ConformerRow* attribute), 112  
 cavity construction:, 22  
 clear\_physical\_property() (*pyCRS.Database.COSKFDatabase* method), 105  
 combinatorial term, 7

.compkf file, 28  
 composition line, 56  
 compound\_id (*pyCRS.Database.CompoundRow* attribute), 110  
 compound\_id (*pyCRS.Database.ConformerRow* attribute), 112  
 compound\_id (*pyCRS.Database.PhysicalPropertyRow* attribute), 114  
 compound\_id (*pyCRS.Database.PropPredRow* attribute), 116  
 CompoundRow (*class in pyCRS.Database*), 110  
 compounds, 42  
 conformer (*pyCRS.CRSManger.CRSMixture* attribute), 122  
 conformer\_id (*pyCRS.Database.CompoundRow* attribute), 110  
 conformer\_id (*pyCRS.Database.ConformerRow* attribute), 112  
 ConformerRow (*class in pyCRS.Database*), 112  
 .cos file, 34  
 coskf (*pyCRS.Database.CompoundRow* attribute), 111  
 coskf (*pyCRS.Database.ConformerRow* attribute), 113  
 COSKF file, 19  
 .coskf file, 19  
 COSKFDatabase (*class in pyCRS.Database*), 103  
 COSMO accuracy:, 21  
 COSMO cavity construction:, 22  
 COSMO file, 19  
 .cosmo file, 19  
 cosmo2kf, 213  
 COSMO-RS parameters, 35  
 COSMO-RS program crs, 35  
 COSMO-RS theory, 5  
 COSMO-SAC, 8  
 COSMO-SAC 2013-ADF, 8  
 COSMO-SAC 2013-ADF parameters, 37  
 COSMO-SAC 2016-ADF, 8  
 COSMO-SAC DHB-ADF, 8  
 COSMO-SAC parameters, 37  
 COSMO-SAC theory, 8  
 cpfusion (*pyCRS.Database.PhysicalPropertyRow* attribute), 114

- crs\_iteration (*pyCRS.CRSManager.CRSMixture attribute*), 121
- crs\_job (*pyCRS.CRSManager.CRSMixture attribute*), 121
- CRSMixture (*class in pyCRS.CRSManager*), 121
- crsprep module, 209
- CRSSystem (*class in pyCRS.CRSManager*), 117
- ## D
- database (*pyCRS.CRSManager.CRSMixture attribute*), 121
- db\_path (*pyCRS.Database.CompoundRow attribute*), 111
- db\_path (*pyCRS.Database.ConformerRow attribute*), 113
- del\_row() (*pyCRS.Database.COSKFDDatabase method*), 105
- del\_row\_by\_conformer\_id() (*pyCRS.Database.COSKFDDatabase method*), 105
- del\_rows() (*pyCRS.Database.COSKFDDatabase method*), 105
- density (*pyCRS.Database.PhysicalPropertyRow attribute*), 115
- density (*pyCRS.Database.PropPredRow attribute*), 116
- density\_corr (*pyCRS.CRSManager.CRSMixture attribute*), 122
- dielectricconstant (*pyCRS.Database.PhysicalPropertyRow attribute*), 115
- dielectricconstant (*pyCRS.Database.PropPredRow attribute*), 116
- ## E
- Ecosmo (*pyCRS.Database.CompoundRow attribute*), 111
- Ecosmo (*pyCRS.Database.ConformerRow attribute*), 113
- Egas (*pyCRS.Database.CompoundRow attribute*), 111
- Egas (*pyCRS.Database.ConformerRow attribute*), 113
- element specific parameters, 36
- estimate() (*in module pyCRS.FastSigma*), 125
- estimate() (*pyCRS.PropPred method*), 123
- estimate\_physical\_property() (*pyCRS.Database.COSKFDDatabase method*), 106
- examples, 213
- excess energies, 55
- execution of COSMO-RS, 35
- execution of UNIFAC, 71, 72, 74
- ## F
- fast approximation, 7
- Fast Sigma COSMO file, 26
- Fast Sigma COSMO settings, 26
- flash point, 45
- flashpoint (*pyCRS.Database.PhysicalPropertyRow attribute*), 115
- flashpoint (*pyCRS.Database.PropPredRow attribute*), 116
- Flory-Huggins parameter, 63
- ## G
- get\_activity\_coefficients() (*pyCRS.CRSManager.CRSSystem method*), 120
- get\_all\_compounds() (*pyCRS.Database.COSKFDDatabase method*), 106
- get\_all\_conformers() (*pyCRS.Database.COSKFDDatabase method*), 106
- get\_all\_physical\_properties() (*pyCRS.Database.COSKFDDatabase method*), 107
- get\_attribute\_by\_compound\_id() (*pyCRS.Database.COSKFDDatabase method*), 107
- get\_compounds() (*pyCRS.Database.COSKFDDatabase method*), 107
- get\_compounds\_id() (*pyCRS.Database.COSKFDDatabase method*), 108
- get\_conformers() (*pyCRS.Database.COSKFDDatabase method*), 108
- get\_full\_coskf\_path() (*pyCRS.Database.CompoundRow method*), 112
- get\_full\_coskf\_path() (*pyCRS.Database.ConformerRow method*), 113
- get\_physical\_properties() (*pyCRS.Database.COSKFDDatabase method*), 108
- get\_sigma\_profile() (*pyCRS.Molecule method*), 127
- get\_tdep\_values() (*pyCRS.Molecule method*), 128
- ## H
- has\_missing\_atoms() (*pyCRS.Molecule method*), 128
- Henry's law constants, 46
- hfusion (*pyCRS.Database.PhysicalPropertyRow attribute*), 114
- hfusion (*pyCRS.Database.PropPredRow attribute*), 116
- hydrogen bond interaction, 8
- ## I
- identifier (*pyCRS.Database.CompoundRow attribute*), 111
- identifier (*pyCRS.Database.ConformerRow attribute*), 112
- infinite dilute, 46
- iso (*pyCRS.CRSManager.CRSMixture attribute*), 122

- iteration (*pyCRS.CRSManger.CRSMixture attribute*), 121
- iteration (*pyCRS.CRSManger.CRSSystem attribute*), 117
- ## J
- jobname (*pyCRS.CRSManger.CRSMixture attribute*), 122
- ## K
- kf2cosmo, 213
- ## L
- liquid-liquid equilibrium, 50
- liquid-liquid extraction, 87
- LLE, 50
- LLE (*multi-component*), 50
- LLE binary mixture, 55
- LLE diagram, 55
- LLE ternary mixture, 56
- log P, 46
- ## M
- mass fractions, 44
- massfraction (*pyCRS.CRSManger.CRSMixture attribute*), 122
- meltingpoint (*pyCRS.Database.PhysicalPropertyRow attribute*), 114
- meltingpoint (*pyCRS.Database.PropPredRow attribute*), 116
- method (*pyCRS.CRSManger.CRSMixture attribute*), 121
- missing\_atoms() (*pyCRS.Molecule method*), 128
- mixture (*pyCRS.CRSManger.CRSMixture attribute*), 121
- mixture (*pyCRS.CRSManger.CRSSystem attribute*), 117
- Mn (*pyCRS.Database.PhysicalPropertyRow attribute*), 115
- modify\_attribute\_by\_compound\_id() (*pyCRS.Database.COSKFDatabase method*), 109
- module
- pyCRS.CRSManger, 117
  - pyCRS.Database, 103
  - pyCRS.FastSigma, 125
  - pyCRS.Input, 126
  - pyCRS.Output, 126
  - pyCRS.PropPred, 123
- molar fractions, 44
- Molecule (*class in pyCRS*), 127
- MOPAC COSMO file, 33
- MOPAC COSMO settings, 33
- multi\_species (*pyCRS.CRSManger.CRSMixture attribute*), 122
- ## N
- name (*pyCRS.Database.CompoundRow attribute*), 111
- name (*pyCRS.Database.ConformerRow attribute*), 112
- nring (*pyCRS.Database.CompoundRow attribute*), 111
- nring (*pyCRS.Database.ConformerRow attribute*), 113
- num\_mix (*pyCRS.CRSManger.CRSSystem attribute*), 117
- ## O
- Octanol/Water partition coefficients, 46
- optimizing solubility, 87
- optimizing the solvent, 87
- outputs (*pyCRS.CRSManger.CRSSystem attribute*), 117
- ## P
- partition coefficients, 46
- phase stability test, 48
- PhysicalPropertyRow (*class in pyCRS.Database*), 113
- pKa values, 46
- polymer sigma-profile, 61
- pressure (*pyCRS.CRSManger.CRSMixture attribute*), 121
- problem\_type (*pyCRS.CRSManger.CRSMixture attribute*), 121
- property prediction, 79
- PropPredRow (*class in pyCRS.Database*), 115
- pvap (*pyCRS.Database.PhysicalPropertyRow attribute*), 115
- pyCRS.CRSManger module, 117
- pyCRS.Database module, 103
- pyCRS.FastSigma module, 125
- pyCRS.Input module, 126
- pyCRS.Output module, 126
- pyCRS.PropPred module, 123
- ## Q
- QSPR COSMO file, 26
- ## R
- read\_coskf() (*pyCRS.Database.CompoundRow method*), 112
- read\_coskf() (*pyCRS.Database.ConformerRow method*), 113
- read\_sdf() (*pyCRS.Input method*), 126
- read\_smiles() (*pyCRS.Input method*), 126
- resolved\_smiles (*pyCRS.Database.CompoundRow attribute*), 111
- resolved\_smiles (*pyCRS.Database.ConformerRow attribute*), 113

runCRSIteration() (pyCRS.CRSManger.CRSSystem method), 120  
 runCRSJob() (pyCRS.CRSManger.CRSSystem method), 121

## S

scripting examples, 213  
 sigma potential:, 57  
 sigma profile, 57  
 smiles (pyCRS.Database.CompoundRow attribute), 111  
 smiles (pyCRS.Database.ConformerRow attribute), 112  
 solubility, 47  
 solute (pyCRS.CRSManger.CRSMixture attribute), 122  
 solvation energies, 46  
 solvent boiling point, 45  
 solvent flash point, 45  
 solvent vapor pressure, 44

## T

.t21 file, 19  
 tangent plane distance, 48  
 temperature (pyCRS.CRSManger.CRSMixture attribute), 121  
 ternary mixture, 56  
 theory COSMO-RS, 5  
 theory COSMO-SAC, 8  
 theory UNIFAC, 10  
 TPD stability test, 48  
 Tuorial Polymers, 229  
 Tutorial COSMO files, 214  
 Tutorial parameters and analysis, 215  
 Tutorial pKa, 228  
 Tutorial Properties, 216  
 tvap (pyCRS.Database.PhysicalPropertyRow attribute), 115

## U

UNIFAC, 10  
 UNIFAC program unifac, 71  
 units (in module pyCRS.PropPred), 123  
 update\_compound\_by\_conformer\_id() (pyCRS.Database.COSKFDDatabase method), 109  
 update\_compound\_by\_lowestE() (pyCRS.Database.COSKFDDatabase method), 110

## V

vapor pressure, 44  
 visualize\_conformers() (pyCRS.Database.COSKFDDatabase method), 110  
 VLE binary mixture, 55  
 VLE diagram, 55  
 VLE ternary mixture, 56

vp\_corr (pyCRS.CRSManger.CRSMixture attribute), 122  
 vp\_equation (pyCRS.Database.PhysicalPropertyRow attribute), 115  
 vp\_equation (pyCRS.Database.PropPredRow attribute), 117  
 vp\_params (pyCRS.Database.PhysicalPropertyRow attribute), 115  
 vp\_params (pyCRS.Database.PropPredRow attribute), 117

## W

write\_kf() (in module pyCRS.Output), 126