Manual Namics

Frans Leermakers

November 2023

Abstract

This is the manual of `namics`.[1] The `namics` program is replacing `sfbox`. The latter program has been the working horse for SF-SCF calculations for many years. The code however was hard to maintain because of the simula 'history' as this appears in, e.g., the use of the home-made vector class (emulating simula vectors that start not from zero but from unity). `namics` is by construction designed to work with graphical cards (only appropriate in 3-gradient calculations, and currently the use of graphical cards is not guaranteed yet). The `namics` program is in development and at all times one should be on the outlook of 'issues'. If you find any 'issues' please report these to frans.leermakers@wur.nl. You can find the `namics` project in git. If you want to contribute to the coding, you can contact the author as well.

In due time the `namics` program will be renamed to `sfbox`. The program works very similarly as sfbox. One of the main changes is the use template files to control the quantities that are sent to the output-files. In `sfbox` this was done using 'template' files. In `namics` you need to specify the output quantities inside the input file. For the remainder we have tried to keep 'consistency' between the programs. You may find example inputfiles for your inspiration in the git repository of `namics directory example_inputs`.

The program has been developed with the idea that it can be targeted to some application when needed. If the features you want are not yet available, there is a chance that it can be built in. Send requests to frans.leermakers@wur.nl.

I wish you the best of luck in using `namics`.

[1]This is a preliminary manual (version 0). There are still some omissions and many possible improvements. If small or big mistakes, either in the `namics` program or in this manual are detected, you kindly are invited to report these at frans.leermakers@wur.nl, Please submit your inputfile and a report of your issue with it.

Contents

1 What `namics` does

The target of `namics` calculations is to find the equilibrium distribution of an arbitrary number of different (chain) molecules in the interfacial regions of the system compared to their bulk counter parts. The approach to find this is by using the Self Consistent Field (SCF) theory. Following the original choice of Scheutjens and Fleer, space is divided in lattice sites, the average volume fraction of (fractions of) the molecules is calculated in each lattice site. The molecules, either monomers, homopolymers, copolymers, dendrimers, combs, or arbitrarily branched molecules (no loops) are modeled as linear or correspondingly branched strings of spherically symmetric segments with the same size of a lattice site. The chain statistics is accounted for by the freely jointed chain approximation (default). Semi-flexibility is implemented for 1-gradient systems, e.g. for linear chain-fragments. The molecules may be in equilibrium with a bulk solution (grand canonical), grafted at arbitrary coordinate, or fixed with a given amount in the system (canonical). The molecules consist of segments which may contain charges. The segments can have different internal states, to model chemical reactions (e.g. deprotonation). The mean field approximation may be applied in one gradient, using a flat, cylindrical or spherical geometry, it may be applied in two gradients, using a flat or cylindrical geometry or may have three gradient directions (in simple cubic or (in one-and two-gradient systems) hexagonally configured lattice sites). Boundary conditions may be adsorbing, reflecting or periodic. Surfaces may in one- or two-gradient problems be presented as a boundary condition, but in general solid objects may be defined at an arbitrary place on the lattice. Interaction parameters for the solid phases must be chosen on the segment basis, similarly to segments that form the molecules.

2 Big fat WARNING

The `namics` program is under continuous development. Not everything is implemented for all possible scenario's and if this text states that it is, this by no means guarantees that it works perfectly. Check your output! All of it.

Below you will find for several input-options the remark that the option is rarely used, or that the option is in some 'experimental' stage of investigations. This also illustrates that the program is a being subject to changes/modifications etcetera. We constantly try to generate solutions to existing issues and unfortunately there are still plenty of settings that may lead to crashes or malfunctions. If you find such issue for your application, you are advised to ask for help. (frans.leermakers@wur.nl) Helping with the coding is also very much welcomed.

2.1 Restrictions on use

`namics` is free to use for non-profit academic work. When publishing your obtained results you are kindly requested to acknowledge our work. A reference should mostly do fine. Contact Frans Leermakers (frans.leermakers@wur.nl) for details.

2.2 Starting the program

You have an input-file with name with an '.in' extension on a given directory. In this directory there is an directory 'output' to which the output will be directed. You should navigate to this directory. The program is then started on-line by typing (e.g. in Linux command window):

```
\namics\ input-filename.in //the .in extension is optional
```

When necessary you can pipe the output to a log file

```
namics input-filename.in > logfile&
```

and the program will run in the background. You can follow the progress by typing

```
tail -f log file
```

which will print new entries to the log file as they occur. You can quit the tail comment by the cntl C comment.

Some on line help is available by typing `namics -h`. Debugging type of information is spit to the output when you use `namics -d`. This is helpful when an unexpected crash is found.

3 The input file

3.1 General layout

The input file consists of a number of lines. There are a few exceptions, but the general idea is that each line is looking as follows:

```
type : type title : parameter name : parameter value
```

The main type entries are:

- `sys` for choosing different kind of calculations and global parameters
- `lat` for lattice parameters
- `mon` for segment parameters
- `state` for state parameters
- `mol` for molecule parameters
- `newton` for parameters influencing the convergence of the numerical equations

- `reaction` for defining how states are in mutual equilibrium.
- `output` for setting parameters influencing the resulting output file(s)
- `kal` specifies entries what goes in the so-called 'kal'-file
- `pro` specifies entries what goes in the so-called 'pro'-files
- `var` for automatic variation of variables. see section 14
- `alias` is of use to define and vary the composition of the molecules. see section 15
- `mesodyn` Mesodyn entries. These will not be covered in this manual. If you want to use mesodyn (dynamic density functional calculations) contact frans.leermakers@wur.nl.
- `cleng` cleng entries. Special module which uses fragments pinned at both locations. With such a tool you can model, e.g., gells. Cleng entries will not be covered in this manual.
- `teng` teng entries. Special module which uses monte carlo simulations in combination with SCF. Teng entries will not be covered in this manual.

Multiple calculations can be separated in the input file by placing the keyword `start` on a separate line. `start` may be set at the end of the last calculation but this is not necessary. Having multiple `start` statements will execute consecutive calculations wherefore the previous calculation is by default the guess for the next calculation.

The order in which the input lines are given for a separate calculation is irrelevant. Of course the position of the `start` statement, which separates individual calculations is of key importance.

The `namics` inputfile is user friendly in the sense that when you make a typo (either by accident or willfully) in the input file, you will get context related feedback when you try to run it. In many cases you may learn from this feedback what the correct entry is that is targeted. When the programs has many 'start' statements, only the context related error checking is done upto the start that is relevant. This means that a 'typo' at the end of the file may only be noticed after a relatively large 'run'. This may be considered as a programming error and future versions of 'namics' are likely to improve on this.

3.2 Entering numbers

When an integer value is expected in the input one can add and exponent. A few examples of integers:

```
3
5736745
5e8
```

If the input requires a 'real' value, an integer is also accepted, as well as scientific notation. A few examples:

```
45
45.67
0.345e4
12.4e-3
.34E5
```

Note that both 'e' and 'E' are accepted as a separator between mantissa and exponent. The value `12.4e-3` means 12.4×10^{-3}.

3.3 Adding comment

To add comment to the input file or to temporary exclude a line for reading, place `//` in front of the line that you don't want to include in the parsing. The comment `//` can also be set half way a line, this excludes the remainder of the line from interpretation by `namics`.

3.4 Error messages and warnings

The input file is read in a fully safe way. This means a full syntax check on every input line is done. In context parsing is done per calculation. Messages are generated for unrecognised commands, logical errors, and omissions in the input (if one parameter is set, another should sometimes also be set). Since most errors are rare, inconsistencies and errors may appear in the error messages: not everything that can go wrong has been tried... Please report any inconsistencies and errors in the error handling. Redundant statements, parameter settings that are not relevant in the local context may remain unnoticed and not always generate 'warnings' or may not trigger 'errors' as these may simply be ignored.

4 Some tips and tricks

4.1 Obtaining information about default values

When you ask for so-called 'ana' file output, with the statement: `output : ana : append : false`, you will, after the calculations have finished, find a file `input-filename.ana` (the `.in` extension is replace by `ana`) where 'input-filename' is the name of your `inputfile` (minus the extension). In this file you find all values that the program can output. It thus also specifies the implicitly assumed default values that are used. If you are interested in some typical quantity that (by accident) has not yet been directed to the output, and you like to have access to this parameter, you can ask Frans to make this quantity available for your disposal.

4.2 How to get the output you want

In general you may only have a guess which keywords to use for getting the wanted output in your outputfiles. Please use the `ana`-file option mentioned in the previous section to generate an outputfile with name `input-filename.ana`. This file will contain the keywords to be used in your `kal` or `pro` statements. See also section 16.1 for the meaning of some of the outputted quantities.

It is possible to generate multiple output files for one calculation. (see also section 12) For example a file that can be read in a spreadsheet (a `kal` file), contains columns for various variables. The rows denote the different calculations that are specified in the input file. However this file format is not suitable for profiles. So for profiles it is common to generate another file (a `pro` file). For three-gradient calculations it is of use to use the `vtk` file format. This file can then be read by Paraview. Note that in a `vtk`-file you can have just a single 'profile' entry. Suppose you want to output in the `kal`-file the quantity `phibulk` of the molecule `mol` with name 'mypol'. You may check that this quantity exist as an output option first. In the `ana`-file (see above) you find the statement `mol : mypol : phibulk : xxx`, with 'xxx' some numerical value and therefore you can use:

```
kal : mol : mypol : phibulk
```

This makes only sense if you have explicitly asked for such file

```
output : kal : append : false.
```

In this output statement you tell that you are not going to append previous existing file, but overwrite it with the new file. If the file does not yet exist, it will generate such a file. You will find in your output directory the file `input-filename.kal`

Similarly, you may be interested in the profile of the volume fractions of the monomer with name 'A' (you may guess that the keyword 'phi' is used. However, again you may check this and find in the `ana`-file the line: `mon : A : phi : profile`. This means that you can put the `phi`-info a `pro`-file. You can do this via:

```
pro : mon : A : phi
```

Of course you should ask explicitly for such type of profile using the statement `output : pro : append : false`. You will find in your output directory a file `input-filename.pro`. Note that when you have more than one consecutive calculation, a unique (set of numbers) number is appended to the `input-filename`: e.g. `input-filename_`*xx*`_`*yy*`.pro`, where *xx* refers to the *xx*-th 'start', and *yy* to the repeat number in the `var` statement (if appropriate)

Note that you can not mix quantities that are labeled as 'profile' in a `kal`-file and quantities that are not labeled as 'profile' in a `pro`-file.

If you do a calculations and you have not yet asked for any output, you will obtain a warning to help you remind to define your output.

4.3 How to speed up (or achieve) convergence

At the basis of SCF calculations there is a free energy functional. Typically a saddle point solution is needed and there are no strong minimisation packages which solves for arbitrary saddle point. On top of this, the set of SCF equations is highly non-linear, and convergence depends from case to case. A lot of work has been done to ensure that as many calculations converge to a solution as possible. However, the default tricks that are used may not work for the particular problem at hand. Initial guesses (see section 11.2 are essential for computing self-assembling structures but might also help convergence for other problems. One and small two-gradient calculations usually are solved by the default method (pseudohessian). However, this default method can not be used for large two-gradient and three-gradient problems, simply because this method takes too much memory. Note that when you try to take too much memory the program may simply stop without any warning or error message. Hessian free search methods are used in those cases, but these are generally less rigorous in solving the equations. Finding search directions is only part of the problem. A good step length control (linesearch) is another. That is why some methods work in combination with specific linesearches and others are less picky on this. More information on this is given in section 13.

4.4 How to prevent overflows

During the iterations, the weighting factors may have values that result in computed volume fractions that exceed the maximum value of the compiler or computer (e.g.: 1.0×10^{400}). Normally you will notice the occurrence of an overflow by a stop of the iterations, Newton will report something like:

```
NaN f=NaN
```

and it will claim to have found a solution. Your output file will also contain `NaN` as values for parameters. `namics` tries to catch these overflows and reports them. An overflow is likely to occur when a calculation is done for long chains (1000 segments or more) and/or with charged systems which contain very little salt (this gives very high potential fields).

When you ask for `overflow_protection` in `namics` you have to combine this with a setting in the file 'namics.h'. In this header file you will find the variable 'Real'. This quantity can be set as 'float' (single precision) 'double' (typical choice) 'double double' (in combination with `sys : NN : overflow_protection : true`. See section 11.

You could also decrease the step size of the iteration by changing `deltamax` see section 13 or change the iteration method.

4.5 Checking the convergence of the calculations

This can be done in three different ways. One can check the log file (for a batch calculation) or read the output to screen (for an online calculation). However, another way is to check the output file itself.

You may ask for the `kal : newton : NN : residual` that was reached and compare this to the `tolerance` set in the inputfile (default value is 1e-7). Another way is to ask for the number of iterations used. `kal : newton : NN : iterations` and compare this to the `iterationlimit` set in the inputfile (default is 5000).

4.6 How to prevent running out of computer memory

The most effective way is to decrease the number of layers (coordinates) or number of gradients in the system. Another way is to select a special scheme for computing the densities that will save a lot of memory for long chain molecules (see section 7).

4.7 Lattice artefacts

A lattice artefact may occur due to the position of the interface on the lattice. This is most notable in the grand potential, when an accurate value of this parameter is desired, you may select for an advanced option to reduce the grit-size length compared to the segment size length (You can select for this approach using the setting `fjc_choices` in the lattice-settings. See section 8).

4.8 The program hangs or crashes

Well, that's not so nice. A typical workaround is to start with a parameter setting which does converge and then go via muliple 'starts' to the setting of interest. The idea is that the default initial guess for any new 'start' is the previous solution. Hence you will end up with a good initial guess for your problem of interest. Hopefully this will prevent the crash. If not, you can send your inputfile to frans.leermakers@wur.nl. Please mention what your 'planned' calculations.

5 `monomer` parameters

- `chi_monomername` (real) This defines the Flory-Huggins χ parameter between different segments.
- `chi_statename` (real) This defines the Flory-Huggins χ parameter between the monomer and a state of a different monomer.

When defining a `chi_` parameter for a segment that is not explicitly defined in the input file, you will get an error message. Define a segment by a line like `mon : a : freedom : free`, this is also needed when using a state name. This is done to catch typos in the segment/state names.

- `freedom` (`free/pinned/frozengrafted/tagged/clamp`) It is possible to restrict the freedom of `monomers`. The value `free` means that in principle the `monomers` can be anywhere on the lattice/system. The value `pinned` means that the `mono`mer is restricted to a certain area/sub-space, defined by setting `pinned_range`. The value `frozen` means that the segment is exclusively in a certain place on the lattice, defined by setting `frozen_range`. It will fully occupy the specified lattice sites and act as a solid, no iteration variables are generated, unless different internal states are defined (not implemented yet). The value of `tagged` is typically used in 'teng'-module (which we do not cover in this manual yet). The value of `clamp` is typically used in 'cling'-module which is not covered in the current manual yet.

- `pinned_range` Defines the range on the lattice where the segment is pinned. See section 9.
- `frozen_range` Defines the range on the lattice where the segment is frozen. See section 9.
- `tagged_range` Defines the range on the lattice where the segment is tagged. See section 9. Used by 'teng' which we do not cover here in this manual yet.
- `clamp_info` This is information on the 'clamp'-coordinates. This is used in the 'cling' module. We do not cover this in this manual.
- `pinned_filename` You can specify a file in which you specify the pinned_ranges. Use this option when you have strange pinned_range conditions. (not often used)
- `frozen_filename` You can specify a file in which you specify the frozen_ranges. Use this option when you have strange pinned_range conditions. (not often used)
- `tagged_filename` You can specify a file in which you specify the tagged_ranges. Use this option when you have strange pinned_range conditions. (not often used)
- `clamp_filename` You can specify a file in which you specify the clamp_ranges. Use this option when you have strange pinned_range conditions. (not often used)
- `sub_box_size` This is used in the 'cleng'-module. (we do not cover this here.
- `valence` (real) The electrostatic charge of a monomer (default: 0).
- `epsilon` (real) The relative dielectric constant of a pure solution of this monomer. A physically realistic value for epsilon is greater than or equal to 1 by definition (normally 5 - 80). Smaller, or even negative values are allowed (default: 80).

- `e.psi0/kT` This sets the dimensionless potential on a frozen segment type. For this segment you then can not set any valence. The idea is to use this ti fix the surface potential instead of surface charge. (This option has not yet be used often; test your answers carefully).
- `fluctuation_potentials` Used to set external potentials. We do not cover this in this manual.
- `fluctuation_amplitude` Used to set external potentials. We do not cover this in this manual.
- `fluctuation_wavelength` Used to set external potentials. We do not cover this in this manual.
- `seeds` Used to set external potentials. We do not cover this in this manual.
- `var_pos` Used to set external potentials. We do not cover this in this manual.
- `phi` Only in 1-gradient computations. Method to fix a particular density profile (kind of frozen) Expect for the argument of 'phi' a sequence of z_values and corresponding volume fractions:(z_value_1,phi_value_1)(z_value_2,phi_value_2) ....(z_value_last,phi_value_last). These values are put as some external constraint for this segment type during the calculations. (not frequently used; test your results)
- `n` used in combination with 'size', and 'pos" to set a number of spherical particles (in 3-gradient problems). In two-gradient problems you set one (then n=1 is needed) particle on the axis of the coordinate system. (not frequently used; test your results)
- `size` used in combination with 'n' and 'pos' to set the size (in lattice units) of the spherical particle.
- `pos` used in combination with 'n' and 'size' set the coordinates of the centers of the particle(s). (not frequently used; test your results)
- `set_equal_to` method to make chi-values equal to the ones for another segment type. The argument of this quantity should therefor the segment-name of another segment type. This option is given to reduce the number of χ-parameters that the user needs to set, but still wants segments to have 'unique' names so that the 'output' of this 'unique' segment name can be asked for. (check your results, because this option is rarely used).

6 `state` parameters

The monomers can consist in different states with different physical properties. When using states, reactions (section 10) need to be defined.

- `mon` specifies the monomer-name to which this state belongs to. Without this statement the state is 'homeless' and the program can not continue.
- `chi_monomername` (real) This defines the Flory-Huggins χ parameter between this state and a monomer.
- `chi_statename` (real) This defines the Flory-Huggins χ parameter between the different states.
- `valence` (real) see Section 5
- `epsilon` (real) see Section 5
- `alphabulk` (real) can be set to enforce the bulk dissociation. This can be used to set for example the pH to a certain value.

6.1 Redundancy between `state` and `monomer` parameters

When setting a variable one can define it either at the `state` or at the `mon`omer level. Setting a variable at the `mon`omer level is equal to setting the same variable with the same value for all states belonging to the `mon`omer. Therefore it is not possible to set for example `epsilon` to 10 for a `mon`omer and define `epsilon` again for some (or all) states of the same `mon`omer.

6.2 Note about thermodynamics of states

Thermodynamic properties like the free energy and the chemical potential are defined relative to a reference of pure unmixed components. When defining for example a weak acid one could argue about the composition of the corresponding reference phase, it could be partially dissociated (to the same extent as the bulk solution) or not dissociated at all. Here, the reference phase was chosen to contain only one state, only `state1` is present in the reference phase. This choice gives the responsibility of defining the reference phase to the user of `namics`.

In practice one will almost never need to worry about this, since the interest is mostly in differences between the thermodynamic quantities of two calculations. There is however one situation where one should take care. If one wants to compare thermodynamic quantities between two different calculations using states then one should not make the mistake of interchanging states (e.g. switch `state1` and `state2`) between two calculations, because the absolute values of the thermodynamical quantities are affected by this.

One could worry about reference phases containing charges, for example a strong electrolyte. It should be clear that a reference phase containing net charge can not be realised in practice, in theory this is simply solved by ignoring any charges in a reference phase. (this section is copied from sfbox info. Test your thermodynamic results at any time. Please contact frans.leermakers@wur.nl when inconsistencies are found.

7 `molecule` parameters

- `freedom` (`free` /`restricted` /`solvent` /`neutralizer` / `second_generation` /`range_restricted`).
  - `free` the molecule is in complete equilibrium with the bulk solution, the parameter `phibulk` must also be set. When all components have freedom 'free' (or 'solvent') system is open and the calculation are said to be grand-canonical.
  - `restricted` defines molecules which are not in full equilibrium with the bulk solution. This means molecules that contain segments that are `pinned` or molecules with `free` segments but a specified amount in the system. Set `theta` (number of segments)) (or `n` (number of chains) to define the amount of the molecule present in the system. When all molecules are 'restricted' (or 'solvent') the system is said to be closed and the calculations are said to be 'canonical'.
  - `solvent` In every calculation the freedom of one molecule should be defined as `solvent`, the phibulk value will be adjusted until the sum of all phibulk values equals one. Hence the solvent makes the reference 'bulk' 'incompressible'.
  - `neutralizer` In most calculations which contain charges you must define the `freedom` of one molecule as `neutralizer`, the phibulk value of this molecule will be adjusted to make the **bulk** electroneutral.
  - `gradient` This boundary condition is set in steady-state calculations. Now, `phibulk` nor `theta` nor `n` values are needed, because the volume fractions at the lower and upper bounds are replacing these entries. See section 17.
  - `ange\_`estricted will impose a special restriction on the molecule, the density is normalised so that a set density is reached in a certain layer. This restriction can also be used to set a *sum* of densities in a specified range. Also set the parameters `esticted`_range .

  default: `free`

- `phibulk` (real) defines the volume fraction of the molecule in the bulk, the value should be between 0 and 1 (mostly no default present)

- `theta` (real) defines the total amount of molecules present in the system, it is equal to the sum over all layers of all densities multiplied by the number of lattice sites in a layer. Values should be between 0 and the volume (amount of lattice sites) of the system.

- `n` (real) defines the total number of molecules present in the system ($n = theta/N$, with N the chain length/total number of segments in the molecule as specified by /tt composition). The value should be not larger than the volume divided by N.

- `composition`: a composition is defined by a segment name between brackets followed by a number, e.g.:

  `(a)1` : composition is 1 segment `a`
  `(a)100` : composition is 100 segments `a`
  `(a1)10(a2)10` : block copolymer
  `((a)10(b)10)4` : sequential block copolymer
  `( ( (a)1(a1)1 )4 (a2)5 )5(a3)4( (a4)20(a5)4 )3(a6)5` : also these bizarre compositions are valid. Here a1 a2 a3 a4 a5 and a6 are all monomer names!

  The language for branched molecules involves the use of square brackets $[\cdots]$, which specifies the sequence that branches off from the main chain. Valid notations are:
  `(a)1((a)1[(a)5](a)1)10` : comb polymer with backbone of 21 segments having 10 arms of length 5.
  `(a)1((a)1[(a)5][(a)5](a)1)10` : comb polymer with backbone of 21 segments having 10 branch points each having two side chains of length 5.
  `(a)1((a)1[(a)5[(a)3][(a)3](a)3](a)1)10` : comb polymer with backbone of 21 segments having 10 side chains that are branched. The side chains have a 'backbone with length 8 and two side chains with length 3 attached on the fifth segment.

  Segment names may contain numbers, spaces are allowed in the composition.

  Special features: @dend(...), comb(...) and water(?) These special features are introduced to generate some molecular composition in such a way that the program can take advantage of the 'symmetry' of the composition to do calculations more efficiently. You can use @dend(?) or @comb(?) to get help.

  `dendrimers`: Take inspiration from examples:

  - example 1: consider segment name 'A' which serves as the 'root'. the 'root' is a single segment name. Let there be spacers of length 2 and segment type B: (B)2 with functionality 3 (all spacers connected to the branch points are of same length) and 4 generations. We specify each generation. We refer to this as `symmetric dendrimers`. `@dend(A,(B)2,3; A,(B)2,3; A,(B)2,3; A,(B)2,3)` hence generations are separated by ';', branch points are 'monomer_names' without the brackets '( )'. Each generation can have unique functionality and (linear) fragment structures.
  - example 2: `asymmetric dendrimers`, with asymmetry in branches of first generation. `@dend(A,(B)2,3,(B)4,1; A,(B)2,3,(B)2,2; A,(B)2,3; A,(B)2,3)` Again the generations are separated by ';', the first generation has 4 arms, 3 of them have length 2 and the other has length 4 (all segment are of type B). etc. Each generation can has its unique features.

  - example 3: Note that a 1-generation dendrimer has the star architecture: Let's do a star with 10 arms `@dend(A,(B)100,10)` Note that in the 'standard' dendrimer the branches are symmetric -all are equal-
  - example 4: 10 arm star end-grafted by one arm to a surface by way of segment 'X' (of course then the freedom of X should be specified accordingly). `@dend(A,(B)100,9,(B)99(X)1,1)`

  `combs`: The idea is that there is a starting fragment, a fragment that is repeated many times containing the teeth, and a trailing fragment. Again the branching segment is by definition a single segment and is not surrounded by the brackets ( ).

  generic example: `@comb((A)5; A,(B)3,(A)6,4; (A)7 )` Backbone has a first part of 5 A-segments, then spacers with 6 A units and a trailing of 7 A-segments. Teeth (first argument after the branch) are composed of 3 B-segments. The second argument is the part which specifies the backbone between the teeth. Number of repeats (last argument of the middle block) is equal to 4. Number of segments in molecule is 5+(1+3+6)x4+7 = 52 segments backbone length is 5+(1+6)x4+7 = 40 segments

  `water`: This special feature is used to define the cluster model for water. This feature is rarely used. Used as `@water(monomer-name)`, where the monomer name should be a valid name of a monomer. This choice for composition should be combined with `Kw`, the value for the association constant.

- `ring` : When the first and last segment of the composition is the same segment type, then you can join these by setting the 'ring' option to 'true'. Note that the chain length is reduced by one in this case. Make sure to start the main chain and end the main chain by a block with length 1: for example '(B)1(B)99(B)1' will be a ring of 100 segments. This option is rarely used and test your results as usual.

- `save_memory` (`true`/`false`) When calculating extremely long chains (say over 5000 segments long) your computer may run out of memory. Setting this variable to `true` triggers a different method to calculate the densities, it's slower (by a factor of about 2) but uses only a fraction of the memory that the standard method consumes. Note: a big memory user is newton in the standard mode, the library that solves the SCF equations, making a smaller system (less layers) saves a lot of memory too . The save memory is done for each fragment separately. Hence it is advised to define long blocks to use `save_memory` effectively.(default: `false`)

- `es`ticted_range Sets the range where the molecules will need to have a given sum of densities for molecules with freedom `ange\_`estricted .

- `Kw` Association constant in the water model that is used in combination with `composition : @water(monomer-name)`. Let the monomer name be 'W', then the water model considers the reactions W+W=W2; W2+W=W3; W3+W=W4; etc. all with reaction constants Kw. Note that these equilibria are all applied within a given lattice layer (coordinate). Sometimes this model is called the cluster model for water.

- `Markov` (default 1) When `Markov : 1` then the chain model is the freely jointed chain (lattice variant of course). When `Markov : 2`, direct backfolding is forbidden in the chain statistics. Use this in combination with `k_stiff`.

- `k_stiff` This parameter set's the stiffness of the chain. In the current implementation the chain stiffness is homogeneous along the chain. The higher this value the stiffer is the chain. The probabilities that are taken are printed to the screen. statistical weight $g(\alpha) = \exp(-\frac{1}{2}k_{\text{stiff}}\alpha)^2$. with α is the angle in radials between two consecutive bonds in the chain.

- `phi_LB_x` Value between 0..1 of the volume fraction just below the lowerbound of the x-coordinate. This is used in steady state calculations. Use in combination with mol-parameters `phi_UB_x` and `B`. Note that steady state calculations are currently only implemented for one-gradient calculations. This value becomes relevant when in system parameters the `calculation_type : steady_state` is set. See also sec 17

- `phi_UB_x` Value between 0..1 of the volume fraction just above the upperbound of the x-coordinate. This is used in steady state calculations. Use in combination with `phi_LB_x` and `B`. Note that steady state calculations are currently only implemented for one-gradient calculations. This value becomes relevant when in system parameters the `calculation_type : steady_state` is set. See also sec 17

- `B` Value larger than 0 of the mobility constant used in steady state calculations. Use in combination with `phi_LB_x` and `phi_UB_x` . This value becomes relevant when in system parameters the `calculation_type : steady_state` is set. See also sec 17

7.1 Special molecules

Namics is not limited to linear chains. Arbitrary branched molecules can be defined using the square bracket notation. (see `composition`). Using this notation implies that the system evaluates the end-point distribution functions brute force. That is, there is no account for possible symmetries. For linear chains the program does not attempt to recognise and use inversion symmetry. If you want to use the symmetry you should use the special molecule entries such as `@dend(...)` or `@comb(...)`.

7.2 Deleting a molecule

A molecule can be deleted for calculation by setting `phibulk` or `theta` or `n` to 0. This might be useful for some series of calculations. Currently this gives problems in the thermodynamics quantities that are computed.

8 `lattice` parameters

- `gradients` (integer) Specifies the number of gradients for the calculation. The value should be 1, 2, or 3(default: 1).
- `geometry` (`flat/cylindrical/spherical`) Alternative for `flat` is `planar`. `geometry` defines the geometry of the lattice. A lattice with 2 gradients cannot be spherical (if you know how to implement this, contact the authors). A cylindrical lattice with two gradients has the curvature in the x-direction, the y-direction specifies the height of the cylinder (default: `flat`). When `gradients = 3`, the geometry is always `flat` (equivalent with `planar`.
- `n_layers` (integer) Specifies the number of layers in a lattice with 1 gradient (no default).
- `n_layers_x` (integer) Specifies the number of lattice layers in the x-direction for a lattice with 2 gradients (no default) or for `gradients : 3`.
- `n_layers_y` (integer) Specifies the number of layers in the y-direction for a lattice with 2 gradients (no default) or for `gradients : 3`.
- `n_layers_z` (integer) Specifies the number of layers in the z-direction for a lattice with 3 gradients (no default).
- `lattice_type` (simple_cubic, hexagonal) In the case of simple_cubic the lattice lambda parameter is set to 1/6. In the case of hexagonal the lambda parameter is set to 1/4. Other choices are possible but in practise these two values are results for these lattice types are more or less accepted. That's why other values are not accepted.
- `bondlength` (real) SCF calculations are typically done in dimensional units. However, when electrostatic interactions are included in the system, we need to specify the size of a lattice cell. In fact the classical choice made by Scheutjens and Fleer is to take the bondlength between two segments along the chain equal to the lattice cell size. A typical value is 3e-10 (units in meters). Another frequent choice is 7e-10 (Bjurrum length in water at room temperature). When you include electrostatic calculations and the value of `bond_length` is not specified, the calculations will stop with a corresponding error message.

- `offset_first_layer` (real) This parameter can only be set in curved (`cylindrical/spherical`) lattices (`gradients : 1`, `gradients : 2`). It defines the distance from the centre of the curvature to the first layer. One might for example use it to calculate adsorption on a curved surface. (default: 0)

- `FJC_choices` (default 3) (3, 5, 7, 9, $\cdots$) In the classical SCF theory the value is 3. With a grit-refinement the number of choices can be increased to 5, 7, etcetera. When `FJC_choices : 3` then `b/l : 1`; `FJC_choices : 5` then `b/l : 2`; `FJC_choices : 7` then `b/l : 3`; etc. The lattice-refinement strategy can also be asked for by setting `b/l`. Make sure that the lattice type is set to hexagonal.

- `b/l` (default 1) (1, 2, 3, 4, $\cdots$) In the classical SCF theory the value is 1. With a lattice-refinement requested by `b/l > 1` the size of the bondlength and lattice-size length can be increased to 2, 3, $\cdots$. Again `FJC_choices : 3` equivalent with `b/l : 1`; `FJC_choices : 5` equivalent with b/l : 2 ; `FJC_choices : 7` equivalent `b/l : 3`; etc. The lattice-refinement strategy can also be asked for by setting `FJC_choices`. Lattice type needs to be hexagonal.

- `ignore_site_fraction`. (true/false) In the segment potentials we can ignore the site-fractions in the FH-term. Then the potential is truly local. (Default false) The value 'true' is not recommended unless you want to test the influence of the use of site-fractions.

- `fcc_site_fraction` (true/false) (default false). Experimental. Not recommended to set this to true.

- `stencil_full` (true/false) (default false) Experimental. Not recommended to set this to true.

8.1 Setting the boundary conditions

The following boundary conditions are build in:

- `surface` means that a surface is placed just outside the system. One should position a segment with freedom `frozen` at that place (see sections 5 and 9). Note that this is not implemented in `gradients : 3`. When you need a surface somewhere in the system with `gradient : 3`, you should introduce a monomer with `freedom frozen` and specify the `frozen_range` . Sometimes it is necessary to specify a thick solid phase, e.g. to control the image charges in cases where the charge distribution on top of a surface is laterally inhomogeneous.

- `mirror` positions a mirror plane exactly outside the outer most layer.

- A `periodic` boundary condition means that the lattice is periodic in the direction where it is applied. A conformation of a molecule walking out of the system at one periodic boundary walks back into the system at the opposite periodic boundary. Typically this is useful in gradients : 3 cases.

To define the boundary conditions, set the following variables:

- `lowerbound` (`surface/mirror/periodic`) specifies the boundary condition near layer 1 in a lattice with 1 gradient (default: `mirror`).
- `upperbound` (`surface/mirror/periodic`) specifies the boundary condition near the last layer (`n_layers`) in a lattice with 1 gradient (default: `mirror`).
- `lowerbound_x` , `upperbound_x` , `lowerbound_y` , `upperbound_y` , (`surface/ mirror/ / periodic`) specify the boundary conditions in a lattice with 2 gradients (default: `mirror`).
- `lowerbound_x` , `upperbound_x` , `lowerbound_y` , `upperbound_y` , `lowerbound_z` , `upperbound_z` (`mirror/ periodic`) specify the boundary conditions in a lattice with 3 gradients (default: `periodic`). If you want to set a surface in systems with 3 gradients, you have to place a molecule with freedom frozen inside the box. You should then use the frozen_range to specify the dimensions of your substrate. This option can also be used in gradients : 1 and gradients : 2, but is obviously less effective.

9 Defining ranges

At some point a range in the lattice needs to be specified, for example to graft segments or molecules. The syntax for a lattice with 1 gradient is as follows:

```
\layernr; layernr
```

The first layernumber is the lower boundary of the range, the second layernumber is the higher boundary of the range. Instead of specifying a number the keywords `lowerbound`, `upperbound` and `lastlayer` can be used.

The following combination of input lines would graft the molecule called `brush` with a length of 100 segments at a grafting density (molecules per unit area) of 1%, with the segment pp pinned in lattice layer 5.

```
mon : p : freedom : free
mon : pp : freedom : pinned
mon : pp : pinned_range : 5;5
mol : brush : composition : (pp)1(p)99
mol : brush : theta : 1
mol : brush : freedom : restricted.
```

A surface at the upperbound/lowerbound (in `gradients : 1` could be made using these 'keywords'. Putting a surface in the upperbound can be done as follows:

```
mon : s : freedom : frozen
mon : s : frozen_range : upperbound
```

It is also possible to graft molecules in 'firstlayer' or 'lastlayer' values. Grafting a chain in, e.g., the 'firstlayer' can be done as follows:

```
mon : p : freedom : free
mon : pp : freedom : pinned
mon : pp : pinned_range : firstlayer
mol : brush : composition : (pp)1(p)99
mol : brush : theta : 1
mol : brush : freedom : restricted.
```

For a lattice with two gradients a similar syntax exists:

```
layernr_x, layernr_y; layernr_x, layernr_y
```

The first two layernumbers give the co-ordinates for the lower boundary of the range, the last two give the upper boundary of the range. In this way one can define 'squares' in the lattice to be a range. Complex ranges should be build up from defining multiple 'squares'. For example: to define a solid (freedom `frozen`) sphere it is necessary to define multiple `frozen` segments and give them a different `frozen_range` .

For a lattice with three gradients a similar syntax exists:

```
layernr_x, layernr_y, layernr_z; layernr_x, layernr_y, layernr_z
```

The first three layernumbers give the co-ordinates for the lower boundary of the range, the last three give the upper boundary of the range. In this way one can define 'cubes' in the lattice to be a range. Complex ranges can be build up from defining multiple 'rectangular'-regions. For example: to define a solid (freedom `frozen`) sphere it is possible to define multiple `frozen` segments and give them a different `frozen_range` .

For `gradients : 3` an extends set of options exist to specify the range of frozen or pinned segments. Below we choose to illustrate this using the `frozen_range` property; the same applies for `pinned_range` .

The most general one is to read the restriction range via a file which contains the range information. The most important ones are:

- `pinned_filename` When `pinned_range` is needed but not specified it is expected that this information is given via an inputfile. This filename is provided via `pinned_filename` entry. See below for the file-content in this case.

- `frozen_filename` When `frozen_range` is needed but not specified it is expected that this information is given via an inputfile. This filename is provided via `frozen_filename` entry. See below for the file-content in this case.

Such file with pinned and frozen range info, should exists in the same directory as the inputfile that is used. In this file for each coordinate in the system, either a unity (1) or a zero (0) entry is given. The unity says that the range is active (part of the range) and zero is inactive (not part of the range). The unities and zeroes are in separate lines. There are as many lines as there are coordinates. The coordinates will be read in the following way: for ($x = 0$; $x <= upperbound_x + 1$; $x++$) for ($y = 0$; $y <= upperbound_y + 1$; $y++$) for ($z = 0$; $z <= upperbound_z + 1$; $z++$). Write your own python code if you need a special topology, e.g., for your surface.

`tagged_filename` and `clamp_filename` entries are not discussed in this manual.

10 `reaction` parameters

It is possible to define reactions between segments and states. The states may differ in charge and χ parameters. Instead of calculating dissociation of acids and bases using the standard approach (a multi state-approach), where water and, e.g., protons are assumed to be implicitly present. It is not yet possible to define reactions between states that are `frozen` (surfaces).

- `pK` : $-\log$(reaction constant) (no default).

  The relation is as follows

$$\ln K = -\frac{1}{k_B T}\sum_k \nu_k \mu_k^{\ominus}$$

where ν_k is the stocheometry of the state in the reaction, it is negative for reactants and positive for products.

- `equation` This parameter defines the reaction equation. For example to model the reaction

$$H_3O^+ + OH^- = 2H_2O$$

  consider the following input lines:

```
state : H3O : mon : W
state : H2O : mon : W
state : OH : mon : W

state : H3O : valence : 1
state : H2O : valence : 0
state : OH : valence : -1

reaction : auto : equation : 2(H2O)=1(H3O)1(OH)
reaction : auto : pK : 17.5
```

So the number before the state or segment defines the stocheometry, the equal sign `=` separates the left hand and right hand arguments of the reaction. Best strategy is to find an example input file that uses reactions and use such file as inspiration for your own system.

11 system parameters

The default name of the system is 'NN' (a short for no name). Parameters set in the 'sys'-block affect the entire calculation. Currently the setting for the temperature is not implemented. This means that the standard temperature is set to 298.15. Change your chi-parameters to emulate a change in temperature.

- `overflow_protection` (`true/false`) When set to `true`, you should also make sure that the namics.cpp program is properly prepared for it. In the top of the namics.h header you must activate `#define LongReal`, which defines the 'Real' to be a 'long double'. In this setting the computer can take higher numbers than usual. Calculations will go slower and will take more memory.

- `initial_guess` (`previous_result` / `none` / `file` / `polymer_adsorption` / `membrane/micelle`) (default: `previous_result`)

  `previous_result`: `namics` will do its best to find proper SCF solutions. However, sometimes there are more than one solution and you will need to guide the program to the proper solution. Sometimes it is hard to find the solution if you fail to give the program a reasonable guess. One way to go is to do a series of calculations (see section below on this topic), and work from one solution to the next. The default setting for this strategy `initial_guess : previous_result` will use the solution of the previous calculation as the guess for the next one.

  `file` It is possible that the road towards a good initial guess for the problem at hand is long and tedious. In this case it is good to know that you can store the iteration variables that correspond to a given SCF solution to a file. You can trigger this with `final_guess : file`. At any time, e.g. in the first calculation of a subsequent 'run' of `namics` you can read the initial guess from this file again, triggered by `initial_guess : file`. See section 11.2 for more details. Do not forget to reset the `initial_guess to previous_result` in a subsequent calculation, or else the program will continue reading the initial guess from the same guess file.

`none` In this case `initial_guess` is not active. Calculations start with all iteration variables set to zero.

`polymer_adsorption` Even though polymer adsorption was one of the first topic covered by Scheutjens and Fleer, the convergence of such systems is far from trivial. You can try to help the system by using this setting. It works only in one-gradient calculations in the classical setting (surface on the lower bound). Do not forget to reset the `initial_guess` to `previous_result` for subsequent calculations. (No strong guarantee that this guess is useful for you)

`membrane` When you have the plan to link the bilayer to the lowerbound of your system (classical way to go), you can try this setting. Make sure that you have amphiphiles in your system or else the program can not decide on it's guess. Do not forget to reset the `initial_guess` to `previous_result` for subsequent calculations. (No strong guarantee that this guess is useful for you) `micelle` See previous item on `membrane`. Make sure you have spherical or cylindrical lattice geometry selected. (No strong guarantee that this guess is useful for you)

- `constraint delta` Lagrange method to control for example the position of the interface independently from the normalisation of the density profiles (grand canonical or canonical). Currently only `delta` is implemented, but other types of methods are envisioned triggered by the value coupled to this keyword. When `constraint : delta` is found the program expects to find `delta_molecules`, `delta_range`, `phi_ratio` and in some cases also the `delta_range_units` entries. See section 11.3 for some details.

- `delta_molecules moleculenameA;moleculenameB` This entry is needed when `constraint : delta` is set. It requires two molecule names separated by a ';'. Note that the order of these names should be the same as the order in which the molecules are introduced first in the inputfile. (this may be considered as a programming error, and may be solve in the near future).

- `delta_range (x1, y1, z1)(x2, y2, z2)` $\cdots$ or `(x1, y1)(x2, y2)`$\cdots$ or `(x1)(x2)`$\cdots$ for 3, 2 or one gradient calculations, respectively. This entry is needed when `constraint : delta` and the goal is to communicate the coordinates where the delta constraint is applied to the system. There is an option to provide the coordinates by means of a file. (see `delta_inputfile`)

- `delta_inputfile delta_inputfilename` the inputfile, which the specified name, contains the coordinates of where the `constraint : delta` is implemented. File format: put on each line in the file the (x,y,z) info; see `delta_range`. You either use `delta_inputfile` or `delta_range` but not both simultaneously. This option is rarely used.

- `phi_ratio` (real) This entry is needed when `constraint : delta` is set. Typically you want to use a value ratio here such that the interface can be located. Typically the `phi_ratio` is given the the ratio of the critical densities. So suppose mol A has composition (A)N, and the mol B has composition (B)M then a suitable ratio is given by `phi_ratio :` $\sqrt{M}/\sqrt{N}$. Here we expect `mol : A` to be defined before `mol : B`.

- `delta_range_units` `bondlength/gritsize` The natural choice here is `bondlength` because n_layers, etc are also given in bondlength units. If `FJC_choices > 3` (see section 8) the grit as a smaller size than the bondlength and the value specified with `delta_range` will bring your interface to a lower x-value (by a factor of 2, 3, $\cdots$).

- `GPU true/false` (default `false`. `GPU : true` will trigger the use of a GPU. This is only relevant for 3 gradient calculations, as the use of GPU is only effective when the number of computations per iteration is huge. There are only a few settings in which this setting will work properly. For example only linear chains are currently supported. Mesodyn applications my also work smoothly using the GPU. With some work other applications should also work in the GPU mode, but currently we can not guarantee this. Please contact frans.leermakers@wur.nl when you experience severe issues. Currently the use of GPU is not recommended.

- `find_local_solution truefalse` (default `false`). Turn this option on in 3 gradient calculations when the convergence is very poor (e.g when capillary bridges are generated between a random collection of particles. (this is a highly specialized flag. The advice is not to use this option at all.) It is used in combination with `split`.

- `split` (integer) (default 2). This quantity is only needed when `find_local_solution : true`. The total 3d volume is subdivided by some integer; that is, you have a number of sub-boxes given by the 'spit'-entry. Then the amount that is found in each part of the system is fixed and the SCF solution is searched per sub-box. This may help the system to find a solution when many potential solutions compete. You are not adviced to use this option. These developments are at best in some 'experimental' stage.

- `X` With X we refer to the characteristic function in your system. It is hard for the program to decide which thermodynamic potentials it will compute and make available for output. The `X` will allow you to define your own thermodynamic potential. See section 11.4 for more details.

11.1 Steady state calculations

- `calculation_type` [`equilibrium`, `steady_state`] (default: `equilibrium`). `steady_state` So called steady state calculations are called for. Use this in combine with `phi_LB_x`, and `phi_UB_x` and some value for the mobility constant `B`. (Can be set of one or more molecules in the system. This option is available in one-gradient calculations. The option is not used frequently and might need some fine tuning. See also sec 17.

`equilibrium`) (default) triggers the classical SCF machinery.

11.2 Storing guess to file and read guess from file

For general information of initial guess strategies, see `initial_guess` in section 11. To store an guess to file (here named myfinalguess.outi' you need to do the following:

```
sys : NN : final_guess : file
sys : NN : guess_outputfile : myfinalguess.outi
```

Then you will find in the output directory your file myfinalguess_xx_yy.outi, possibly modified by the 'start'-numbers `xx_yy`. When you copy this file back to the input directory and name it, for example, 'myinitialguess.ini', you can read this file in the following way:

```
sys : NN : initial_guess : file
sys : NN : guess_inputfile : myinitialguess.ini
```

Again, do not forget to set the `initial_guess : previous_result` in the subsequent calculation or else the program will continue reading its guess from the file.

11.3 Add constraint on SCF solution through a Lagrange parameter

Consider, e.g. nucleation of a droplet at some super-saturation of the system. This problem calls for a imposed chemical potential of the droplet forming component. You want to find grand potentials of drops as a function of the radius of the droplet. In classical calculations where the SCF solution is found by imposing `theta` of the droplet forming component, we only find the solution close to the top of the nucleation curve: for a given radius, a finite but relatively fixed interfacial tension, the Laplace pressure depends on the particle size. The Laplace in turn is linked to the chemical potential of the droplet forming component.

To solve this problem one needs to decouple the droplet size, the interfacial tension and the Laplace pressure. In other words, you need a way to impose some chemical potential/that is you have to find a specified Laplace pressure for any position of the interface. Consider the following input lines:

```
sys : sysname : constraint : delta
sys : sysname : delta_molecules : A;B
sys : sysname : delta_range : (100)
sys : sysname : phi_ratio : 1
sys : sysname : delta_range_units : bondlength
```

This strategy requires a lattice with at least 100 lattice layers. It further assumes two molecules with names 'A' and 'B' respectively. We assume that the A and B molecules have a solubility gap because the χ parameter is sufficiently high. When `FJC_choices > 3` see section 8, you need to specify which units you use. Here we choose `bondlength`. This strategy assumes that some initial guess is present which has its interface approximately located at $x = 100$ (one-gradient system). What the program is doing is the following. A Lagrange parameter (called `delta`) is introduced at coordinate $x = 100$. This Lagrange parameter is modified until the ratio of the densities of A and B are equal to unity (in this case; it can be set to other values of course). When B is the solvent and A is normalised by a specified `phibulk` value, we exactly combine our two goals. Fix the chemical potentials (though phibulk), and specify the location of the interface. You are advised to check out a working example if your interest is in this type of calculations.

11.4 Compose your own thermodynamic potential

The Helmholtz energy F and the grand potential Ω are computed for each calculation. Moreover for each molecular component the program computes the chemical potential (from the composition of the bulk in which the system is in equilibrium with). The relation between grand potential and Helmholtz energy is simple $F = \Omega + \sum_i \mu_i n_i$ where the index i runs over all molecular components (all 'mol's), and n is the number of molecules of the corresponding species. Sometimes the interest is in a partial open thermodynamic potential: $X = F - \sum_j mu_j n_j$, where j runs over only those components that have a specified bulk concentration (so excluding the pinned molecules). We can ask `namics` to compute X in the following way.

```
sys : sysname : X : F-molname1-molname2
```

which now means that $X = F - \mu_1 n_1 - \mu_2 n_2$ where 1 refers to molecule with molecule name 'molname1' en 2 refers to molecule with molecule name 'molname2'. This X is of use when mol-name1 and molname2 are both 'mobile' components in the system (e.g. with freedom free and specified phibulk). When there are only two molecules in the system, X will be the same as the grand potential of course. This become useful, e.g., when a third molecular component is grafted to a surface.

When the system has segments with internal states, we can use:

```
sys : sysname : X : F - molname_1-(statename_i,2) - ...
```

As the state-name is uniquely connected to a segment type, and for each internal state of the segment the chemical potential is computed and available, we can thus extract from the Helmholtz energy the contributions of the specified states (through the number of segments of that type (given by theta). In this case $X = F - \mu_1 n_1 - 2\mu_i \theta_i$. The computation of X in this way is not often used, and as always carefully check whether your desired thermodynamic potential is computed correctly by the program. You can write

```
sys : sysname : X : ?
```

to get help on how to define your thermodynamic potential. Of course X will be made available to be outputted. You should explicitly ask for this. See section 12

12 Output files and parameters

To keep track of the relation between inputfile of the program and potentially several outputfiles, we choose to name the output file, similarly as the input file, and only change the 'extension' of the file. Typically the second argument is a 'name' of some soft which is only used to distinguish one item from the other. In the output statements, the second argument has a distinct function. You can choose from just a limited set of keywords. So the output statement reads. Let the inputfilename be 'myinput.in'. The third and fourth argument of the output statement allows you to pass some additional information to the output module concerning this file. These may contain keywords such as `write_bounds` (true/false)/`use_output_folder` (folder-name) / `write` (true/false) / `clear` (true/false) / `header_separator` (default tab) Third and corresponding fourth arguments can be varied when appropriate of course. You may put for the same file multiple output statements (all having the same second argument) of course.

```
output : ana : write : false //'myinput.ana'
output : kal : write : true    //generates 'myinput.kal'
output : pro : append : false  //generates 'myinput_xx_yy.pro'
output : vtk : write_bounds : false   //generates 'myinput_xx_yy,vtk'
output : pos : clear : true   //generates 'myinput.pos'
output : vec : use_output_folder : myfolder //generates 'myinput.vec'
```

- `second_argument ana` file, may be read in an editor directly. The use of the 'ana' program which could be used to extract data from it, has been discontinued. However, there are database like programs that can be configured to read the 'ana' file. This file dumps all the values that are standard being evaluated plus the parameters that are generated by the `extra_output module` that can be asked for. It you cannot remember, or do not know what 'keyword' is used to put a result to the outputfile, you can inspect an ana-file and you will be able to find exactly your information. Ana-files are rarely used to systematically generate results. So typically you can set the 'write' to 'false' after the first calculation.

- `second_argument kal`eidagraph file, which can be read into almost any spreadsheet or plotting routine. You can add many quantities to this file; however profiles, or vectors are not allowed. You can echo input quantities in the file. You can add computed quantities in the file etc. Consider the following examples.

```
output : kal : write : true    //generates 'myinput.kal'
kal : mon : A : chi-B  //A is monname, B is other monname
kal : mon : B : epsilon //
kal : mol : polA : composition //polA is moleculename
kal : mol : polC : theta
kal : mol : water : phibulk
kal : sys : mysys : X  //mysys is systemname
kal : lat : flat : n_layers //flat is name of lattice
kal : sys : mysys : grand_potential
kal : mol : polB : Mu  //chemical potential of polB
kal : newton : isaac : residual //find error at termination of iterations
kal : newton : isaac : iterations //find number of iterations used
kal : alias : myalias : value //puts the alias value
kal : mol : polD : phi(5)
```

Note that in the kal file you can not put profiles. However you can ask for any of the profiles the value at a specified coordinate. (This works for one-gradient calculations).

The order in which the quantity is put in the kal-file depends on the order in which it is put in the inputfile. When the kal file already exists in the output directory, you can overwrite it, or append to it. You control this by the statement `output : kal : append : false`. When `append` is set to `true`, it adds extra line(s) to the kal-file.

- `second_argument pro`files file that only lists profiles and is suitable for reading into a spreadsheet. A separate file will be generated for each calculation! The file name will be appended with a number denoting the number of the calculation (starts). When automatically varying parameters (see section 14) `namics` will also append the actual var-number. The size of a profile is specified by the number of lattice coordinates in the system. The files become a bit messy when two or three gradient profiles are asked for. For examples consider:

```
output : pro : append : false // true is not an option.
pro : lat : mylat : L //number of sites per layer
pro : mon : A : phi  //A is monname, phi is the volume fraction
pro : mol : PolA : phi // PolA is molname
pro : mol : Copol_A : phi // phi for mon A of molecule Copol
pro : sys : NN : grand_potential_density
pro : sys : NN : q // dimensionless charge/e
pro : sys : NN : psi //electrostatic potential (V)
pro : mon : water : alpha-H3O //H3O is a internal state of water.
```

- `second_argumentvtk` file can only lists one given profiles and is suitable for plotting by Paraview. A separate file will be generated for each calculation!

The file name will be appended with a number denoting the number of the calculation (starts) (vtk can read the numbered file in one statement and then make a 'movie'. When automatically varying parameters (see section 14) `namics` will also append the actual var-number. The size of a profile is specified by the number of lattice coordinates in the system. The header of the vtk file makes it less practical to be inspected by a spreadsheet.

```
output : vtk : append : false // true is not an option.
vtk : sys : NN : free_energy_density
```

Any profile that is accepted by 'pro' can be put in the vtk file. Currently we have just one vtk file per calculations. This might change in the future.

- `second_argumentpos` Option to use coordinates of tagged or clamped molecules. We do not cover examples in this case.
- `second_argumentvec` Sometimes there are arrays in vector format that can be put in an output file (you can find these in the ana file). When several vectors are combined in one output file, they need be be of the same length.

13 `newton` parameters

13.1 Introduction

The C++ class 'Newton' is used to numerically solve the equations that are generated from the input that is given. A lot of settings can be made to influence convergence of the equations and/or change the additional information given during the iterations. Many variables that can be set by a programmer using the class Newton are also available to the user of `namics`. In this manual we will only cover the most frequently used ones.

Currently `namics` uses a linesearch algorithm originally developed by Jan Scheutjens. It is hard to pinpoint what strategy is exactly used, but it is not one of the standard ones used in minimisation algorithms. The classical linesearches use a function evaluation and test whether the search is 'down hill'. In SCF we need to find a saddle point of our free energy and that is why we can not use a free energy as a function in the linesearch. The alternative implemented by Scheutjens is to use a 'trust-region' for the steplength α. One of the consequences is that there is no guarantee that the program in fact does solve the equations correctly. Fortunately, the line search has all kind of 'tricks' to monitor and to guide the iteration process and often the results are satisfactory. In the future we may include alternative linesearches

- `method` (`hessian`, `pseudohessian`, `LBFGS`, `DIIS`, `BRR` (default = pseudohessian)

`-hessian` When you have only a few iteration variables and still the convergence is poor, you may opt for the 'hessian' option. In this case the full hessian is computed numerically every iteration. Below we will discuss options to do a full hessian analysis only once in a while as an intermediate option to improve convergence for systems that have only few degrees of freedom. This method is extremely time consuming and often not necessary. The method is used in combination with the Scheutjens linesearch.

`-pseudohessian` together with the default linesearch is doing the best job for small systems, that is for one-gradient and not too large two-gradient systems. However, as in this case, the approximate- and continuously improved variants of the 'hessian' is stored in memory, this method may take too much memory for large two-gradient and most three-gradient systems. The inverse of the hessian is computed to find a search direction. The Scheutjens linesearch is implemented to find a guess for the steplength. The method is affected by `deltamax` (see below). For high molecular weight systems, you should consider using relatively small values for this quantity (deltamax : 0.01), but usually the default for deltamax : 0.5 will be perfect. Less frequently the value of `deltamin` should be modified beyond the default value (see below). The value of m is not used in this method and values set are simply ignored.

- `LBFGS` Limited memory BFGS (Broyden–Fletcher–Goldfarb–Shanno) method. In fact BFGS updates the inverse of the hessian, so this method does not need the hessian inversion. The limited memory variant is one of the more powerful hessian free search method and therefore it can be tried for large two-gradient and three-gradient systems. The method works well for some type of problems but may fail to converge in other problems. Important is the value of m. This value controls how much of the search history is included in the evaluation of the search method. The LBFGS method works surprisingly well with relatively low values of m. Try $m = 4$ and compare to $m = 8$, 16, 64 and one frequently finds that more is not necessarily better. The values of deltamax and deltamin are relevant as well. Select them similarly as in pseudohessian case. For small molecules surprisingly high values of deltamax will still work well. Note that the current LBFGS algorithm is not the classical one found in many optimisation packages. We have implemented it using the Scheutjens linesearch. Unfortunately this negatively impacts the potential of this method, as it relies relatively much on an 'exact' linesearch algorithm.

- `BRR` According to the literature, the Broyden Rank Reduction method should be competitive with the LBFGS method. This method also uses the value of m, and also frequently a relatively small value is working fine ($m \approx 10$). There is not much experience using this method and at this stage. We have modified standard BRR algorithms so that it works in combination with the Scheutjens linesearch.

- `DIIS` One of the most successful hessian free methods is DIIS (direct inversion in the iterative subspace). Especially for phase separation it seems to do a good job, and it often outperforms LBFGS method. This method also uses the value of m, and also frequently a relatively small value is working fine ($m \approx 10$). This is the method of choice for 3-gradient problems. This method is less reliant on the linesearch and therefore it works relatively well under the Scheutjens algorithm.

- `tolerance` (real) This parameter sets the condition where the iteration may stop. A lower value will lead to a more precise calculation. As a rule of thumb one may say that the number of significant digits in the density profiles is equal to -log(tolerance) (default: `10e-7`).
- `iterationlimit` (integer) maximum number of iterations (default: 000)
- `e_info` (`true/false`) `true` gives information about the residual error during iteration (default: `true`).
- `s_info` (`true/false`) `true` writes the residual error only after convergence (default: `false`).
- `i_info` (integer) writes the residual errors during iterations but does this after the specified number of iterations have been done.
- `deltamax` (real) `deltamax` specifies the largest step in the iteration variables that the iteration procedure can make while searching for the solution. If you get a floating point overflow, you could decrease `deltamax` but it is probably better to look at section 4.4. If you get very small values of alpha in the beginning of the iteration, increase `deltamax` (default: 0.5).
- `deltamin` (real) This parameter specifies the smallest step in the iteration variables that the iteration procedure can make while searching for the solution. If you get very small values of alpha halfway the iteration which make the iteration virtually come to a stop, you could try to increase `deltamin` slightly (e.g. 1e-6) but it is probably better to look at section 13.2. Large values of `deltamin` hinder convergence of the equations extremely (default: 0).
- `m` (positive integer) This parameter is used in several so-called hessian free iteration methods such as DIIS, LBFGS, BRR. it controls the amount of 'history' included in the calculations. Often these hessian free methods are used when the system is large (3-gradients) and the full hessian can not be stored in memory. The memory need of the program scales with m and therefore your interest should be on relatively low values of m. Try `m : 4` or 12 as small values give often good performance already. In BRR it may pay of to take larger values of m (as large as memory limits allow you to take).
- `n_iterations_for_hessian` (int) Specifies the number of iterations after which a explicit full Hessian will be computed. Also the accuracy (residual error) should have a certain minimum value see: `min_accuracy_for_hessian`.

This is a very powerful option for those problems that hardly converge and for large series of calculations that differ only slightly. In many cases convergence will be almost instantaneous after calculation of one full Hessian (default 0). Use this only in combination with `pseudohessian :true` .

- `min_accuracy_for_hessian` (real, 0 - 1) Specifies minimal accuracy (residual error) after which a explicit full Hessian will be computed. This option works in cooperation with `n_iterations_for_hessian` (default 0.1). Use this only in combination with `pseudohessian :true` .

- `max_fr_reverse_direction` (real, 0 - 1) Defines the maximum fraction of reverse directions the iteration may have for a long time (see `reverse_direction_range` below) before an explicit Hessian will be computed (default 0.4) Use this only in combination with `pseudohessian :true` .

13.2 Small alphas

The internal Newton variable alpha defines the relative step length of an iteration (1 is a full Newton step). It is dumped to screen for each iteration where alpha < 1 when `e_info` is set to `true` (the default). When alpha is (very) small for many iterations convergence gets slow or might even stop with a wrong solution. Therefore `namics` restarts the iteration automatically when this is the case. After the restart a new Hessian will be created, small alphas are usually due to a messed up Hessian. An alternative to restarting is to increase `deltamin`. Two variables can be set to influence the restart criterion: `small_alpha` and `max_n_small_alpha`.

- `small_alpha` (real, 0 - 1) Defines the maximum value that alpha must have for a long time to trigger a restart (default 1e-5) Use this only in combination with `linesearch :  Scheutjens` .

- `max_n_small_alpha` (int, 1 - iterationlimit) Defines what is meant by a long time. Time is measured in number of iterations. When alpha is smaller than `small_alpha` for `max_n_small_alpha` iterations in succession, the iteration will be restarted (default: 50). Use this only in combination with `linesearch :  Scheutjens` .

- `stop_criterion` `norm_of_g` / `max_of_element_of_|g|` (default `norm_of_g`) The alternative for the default stopping criterion is only used in mesodyn. We do not cover this module.

13.3 Residual functions

In classical calculations we need to solve the SCF equations. This means that the potential profiles should be consistent with the volume fraction profiles and vice versa. Moreover the system should obey to the incompressibility constraint (and in the case of other constraints) it obey these as well. In principle the newton algorithm is needing a residual function on which it decides to find the scf - solution. Only when all residuals are (close to) zero the algorithm is terminated with all error-flags down.

The `namics` program can solve for different types of residual functions. In particular this information can be selected using:

- `gradient_type classical/mesodyn` default `classical`

  `-classical` This approach is presented first by Evers et al. (Macromolecules 1990 23, 5221) and ingeniously combines all these requirements in a single set of residual functions. It basically uses the segment potentials as iteration variables. In steady state calculations, we still can choose this option, as the residual functions differ only little.

  `-mesodyn` In this case the system searches for the best segment potentials belonging to imposed volume fraction profiles. For the time being we do not cover mesodyn in this manual.

13.4 Super iterations

In sec 14.1 we will see that you can choose for a so-called 'search'. What happens behind the scene is that a super-iteration is started (usually a *regula falsi* method), to find the value which satisfies the search function. You can influence some of the properties and information output while doing these super iterations.

- `super_e_info` default `true`. while doing super iterations the `e_info` of the SCF equations is typically put to false.
- `super_s_info` default `false`
- `super_i_info` is bypassed.
- `super_tolerance` sets the tolerance of the super iteration. It is advised to set this tolerance a bit less high as the SCF tolerance.
- `super_iterationlimit` Set's the iteration limit
- `super_deltamax` Controls the step length.

14 varying parameters

It is possible to vary parameters in the input file by simply overriding a value from a previous calculation. For example:

```
mol : water : theta : 0.25
start
mol : water : theta : 0.5
start
```

It is also possible to add variables in between calculations, you can for example define an extra molecule or delete one (see section 7.2.

Mostly the interest is in varying a numeric value of one variable. When doing large series of calculations, entering the tail of your input file becomes rather boring. There is a shortcut for this. For example to change the variable `mol : water : theta` between 0.25 and 4 in steps of 0.25 enter the following.

```
mol : water : theta : 0.25
var : mol-water : scan : theta
var : mol-water : scale : linear //redundent: as it is the default
var : mol-water : step : 0.25
var : mol-water : end_value : 4
```

Various other quantities (phibulk, chi-values, valence, n_layers) can be 'scanned' similarly. You can choose between scanning linearly as illustrated above, but also an 'exponential' scan can be done. For example when we like to vary the theta over several decades, the following statements are useful.

```
mol : water : theta : 0.001
var : mol-water : scan : theta
var : mol-water : scale : exponential
var : mol-water : steps : 10
var : mol-water : end_value : 0.1
```

Note that in this case the `steps` option is chosen while in the first example the keyword 'step' was used. The integer number for `steps` specifies how many values are used per decade of variation. The default value for `scale` is `linear`

A few notes are in place. The second argument in the case of scanning a parameter needs to be the same for all var-statements. The second argument combines two pieces of information, separated by a dash. The first element is either sys, lat, mol, mon, the second element is the name referring to one of the 'incarnations' of the first element. So `mol-water` refers to the molecule with molecule-name 'water'. An error message will be issued if the 'name' does not exist. or when the first element is not one of the expected ones.

Note as well that when all steps specified by the var statements have been computed, the system will reset the quantity that is varied or searched back to its original/starting value.

14.1 varying parameters in a search mode

Consider the following 'var' statements, which will initiate a super-iteration in the calculations.

```
mol : pol : theta : 1e-3
var : mol-pol : search : theta
var : sys-NN : value : 0.1
```

Interestingly, in the 'search' mode the second argument of the two 'var' statements are not identical.

The target of the above piece of code is to find the situation that the grand potential equals to the value 0.1. (Note that grand potential is a system property hence the second argument is `sys-NN` -correct when system-name is 'NN'-). The variable that can be varied in the system is the theta of the molecule with molecule-name 'pol'. The initial guess for this quantity is provided by the first statement `mol : pol : theta : 1e-3`. To date there is a limited number of properties that can be used in such search strategy and there is only a limited number of properties that can be varied in order to reach the expected results. A list of searchable quantities include: molecule properties such as `mu, theta, n, phibulk`; and system properties such as `Laplace_pressure, grand_potential, free_energy`. Frequently this search option is used to find a zero of the Laplace pressure, or the grand potential.

If you are interested in using the search strategy, but your application has not yet been anticipated, you can contact encouraged to contact frans.leermakers@wur.nl

15 `alias` parameter

In some cases the molecular composition becomes rather complicated or lengthy. Then the use of the `alias` statements may become useful. Let's consider the following input which defines four polymer chains which have exactly the same `composition`.

```
...
mon : A : freedom : free
mon : B : freedom : free
mon : C : freedom : free
...
alias : Main1 : value : (A)10
alias : Side : value : (B)20
alias : Main2 : value : @Main1[(B)20](C)5
...
mol : pol1 : composition : (A)10[(B)20](C)5
mol : pol2 : composition : @Main1[(B)20](C)5
mol : pol3 : composition : (A)10[@Side](C)5
mol : pol4 : composition : @Main2
...
```

The three aliases with names `Main1`, `Main2`, `Side` have values which are fragments or even a completed chain. As shown, aliases may be nested. Of course care must be taken not to generate an 'infinite loop': So do not use something like:

```
...
mol : pol : composition : @Main1@Main2
```

```
....
alias : Main1 : value : (A)5@Main2
alias : Main2 : value : (B)5@Main1
...
```

Infinite loops will result in a fatal error. An alias by itself does not need to be a valid (sub)-composition. However, when substituted in the composition the resulting composition must be value. For example, the following input is perfectly okay, even though both alias are by themselves not valid compositions:

```
...
mon : A : freedom : free
...
mol : pol : composition : @A@N
....
alias : A : value : (A)
alias : N : value : 10
...
```

is equivalent to

```
...
mon : A : freedom : free
...
mol : pol : composition : (A)10
....
```

Note that when the value of the alias is an integer, that we can use the `var` statement to do calculations for a series of molecular weights. Consider the following example:

```
...
mon : A : freedom : free
mon : X : freedom : pinned
mon : X : pinned_range : 1;1
...
mol : pol : composition : (X)1(A)@N
mol : pol : freedom : restricted
mol : pol : n : 0.1
....
alias : N : value : 100
var : alias-N : scan : value
var : alias-N : step : 10
var : alias-N : end_value : 200
```

will results in 10 consecutive calculations for molecules with composition (X)1(A)100, (X)1(A)110, ... (X)1(A)200. As in this case the segment X is pinned, we thus have extended the length of the grafted chains (in a brush).

16 What does the output mean?

The output contains a lot of variables that can be set in the input. Here only those variables that are different from the input will be explained. This list is far from complete and will be extended soon.

16.1 Extra output procedures

Sometimes it is rather hard to evaluate particular quantities from output-file information, but it is not too difficult do do this locally in the molecule, lattice, monomer or system modules. For example, when there is a liquid/liquid interface, it is rather easy to find the width of this interface. Writing a code which automatically traces the width of the interface is hard to write, because it will fail for problems in which such an interface is not present. Also, a train-loop-tail analysis is relevant for a polymer adsorption layer, but less appropriate in a wetting problem, etc.

The idea that is followed in `namics` is to introduce extra-output statements which triggers the calculations of a set of observables. The idea is that the user will understand the in his/her problem this extra-output statement is relevant. Consider the following input lines:

```
mon : A : freedom : free
mon : B : freedom : free
mon : A : chi_B : 1
mol : polB : composition : (B)10
mol : polB : freedom  :solvent
...
mol : polA : composition : (A)10
mol : polA : compute_width_interface : true

output : kal : append : false
kal : mon : A : chi_B
kal : sys : NN : grand_potential
kal : mol : polA : width
kal : mol : polA : phi(1)
kal : mol : polA : phibulk
```

The property `width` is only available after setting `compute_width_interface : true`. The volume fraction of mol PolA in layer 1 and the bulk volume fraction are quantities that can be asked for at any time without the compute_width_interface : true statement. Using this data it is not difficult to find the density differs $\Delta\phi = \phi(1) - \phi^b$.

- `compute_width_interface` true/false (mol-property) example: see above. -width is computed

-Dphi is computed (phi(1)-phibulk)
-pos_interface is computed (gives position of the interface)

- `compute_Gibbs_excess` (sys-setting). It is assumed that the Gibbs excess is computed wrt the solvent. Hence for the solvent the Gibbs excess is per definition zero.
  -R_Gibbs is computed for the solvent
  -theta_Gibbs is computed per molecule
- `compute_kJ0` (sys-property)
  -kJ0 is computed in system from the fist moment of the grand-potential density profile. (infrequently used: make sure to test the result.)

16.2 `system` parameters

- `grand_potential`: surface tension times area (for simple systems) More generally it is the excess Gibbs (Helmholtz) energy that can be attributed to all the interfaces in the system. When multiple interfaces exists one gets the sum over all these interfaces of the excess free energies.
- `grand_potential_density profile`: The local excess tangential pressure
- `Laplace_pressure` dimensionless pressure difference across an interface. (difference between the grand potential density in the last and the first layer of the system).
- `psi profile`: The electric potential (in Volts)
- `q profile` The dimensionless charge, i.e. in units of e.
- `Sprod` Entropy production in case of steady state in dimensionless units.

16.3 `mol`ecule parameters

- `Mu`: Chemical potential from Flory-Huggins theory.
- `GN`: chain partition function.
- `norm`: normalisation constant used for the volume fraction profile computation for this molecule.
- `theta_exc`: Theta minus the phibulk times the number of lattice layers or more general theta minus the volume of the system times phibulk.

16.4 `newton` parameters

- `accuracy`: The resulting residual error, this value should be lower than `tolerance`.

17 Steady state approach

Consider the following input statements. (more full examples are available in example_inputs).

```
mon : A : freedom : free
mon : B : freedom : free

mol : water : composition : (B)1
mol : water : freedom : free
mol : water : B : 0.1

sys : NN : calculation_type : steady_state

mol : pol : composition : (A)10
mol : pol : freedom : gradient
mol : pol : phi_LB_x : 0.15
mol : pol : phi_UB_x : 0.21
mol : pol : B : 0.1
```

The method is triggered by `sys : NN : calculation_type : steady_state` The mobility constant B is given per molecule. The freedom of the molecule which has non-trivial boundary conditions must be specified as `gradient`. Then the program does not need a `phibulk` nor a `theta`. Instead you need to specify the volume fractions at the boundaries. Currently steady states are only implemented in one-gradient calculations and therefore the volume fractions are needed at the lowerbound and upperbound of the x-coordinate.

- `phi_LB_x` mol-property Value between 0..1 of the volume fraction just above the upper-bound of the x-coordinate.
- `phi_UB_x` mol-property Value between 0..1 of the volume fraction just above the upper-bound of the x-coordinate.
- `B` mol-property. Value larger than 0 of the mobility constant used in steady state calculations. Use in combination with `phi_LB_x` and `phi_UB_x` . This value becomes relevant when in system parameters the `calculation_type : steady_state` is set.

18 The include statement

Consider that you have a file with some collection of correct input-statements. For example a set of χ-parameters, with may serve as some force-field collection, or a set of alias statements referring to amino acids, etc. you can include this file in your inputfile. Consider the file containing the file *eps.inc* with the following items:

```
mon : C : epsilon : 2
mon : N : epsilon : 10
mon : O : epsilon : 30
mon : A : epsilon : 5
mon : B : epsilon : 35
alias : tooth : value : [(B)100](A)2
```

you can include this file in your inputfile:

```
include : eps.inc   //note that this line contains just 2 arguments...
mon : N : valence : 1
mon : O valence : -0.5
mol : comb : composition : (A)10(@tooth)10(A)8(C)1[(O)1](O)1
....
```

What happens (at a pre-processing step) is not difficult to understant. At the place of the include statement the file content of eps.inc is substituting the `include :  eps.inc` statement. The file eps.inc must be in the same director as your inputfile. You can use this idea to simplify the inputs that need to be generated and take the part that is the same in all your problems and put that in the such include-file. (For inspiration, see `peptide.in` and `aa.inc` in the example_inputs directory of the namics git repository)

19 Associated modules linked to `namics`

One of the ways to work with `namics` is to run the program from and analyse results with Jupyter notebooks. A few examples how to do this are available. Sometimes it is more efficient to write separate modules that work intimately together with the `namics` modules. Currently there are three associated modules linked to the `namics` program. When you have an idea to work on such a module, feel free to contact frans.leermakers@wur.nl, and join the `namics` team.

19.1 Mesodyn

Written by Daniel Emmery. The model aims to predict how a system which is off-equilibrium proceeds towards equilibrium using Fickian like transport equations. This method has been pioneered by Hans Fraaije (Groningen university/Leiden university) in the 80's-90's of previous century. Example files are available how to use this module.

19.2 Cleng: Monte Carlo routine sampling clamped fragments

Written by Alexander Kazakov. Example files are available on how to use this module.

19.3 Teng: Monte Carlo routine sampling tagged chains positions

Chain conformations are done on the SCF level. One segment per molecule is sampled using MC algorithm. Free energy of SCF is used in the Metropolis Algorithm. Written by Ramanatan Varadharajan.