October 16, 2001

UPI— Commands[1]

Contents

This file describes the keywords (lines) which may appear in the input file (`.sy` file) of UPI , the Universal Path Integral code[2]. Order of keywords in this file *is* sometimes important. Accordingly, we try to list them here in the appropriate order. An alphabetical index is included at the end.

Comments in the input file start with an exclamation mark and end with the end of line.

1 Keywords

We use the following typographic conventions:

- Keywords appear **boldface**, parameters in `typewriter style`
- variables are enclosed in ⟨brackets⟩
- optional parts are enclosed in [square brackets]
- (parentheses) group specifications together; they are also used in implicit for loops like `(ell(i),i=1,ndim)` which expands to `ell(1) ...ell(ndim)`
- alternative specifications are separated by a vertical bar |

[1]David Ceperley (`ceperley@uiuc.edu`)

[2]The current version is: upi10l

- triple dots (. . .) allow for (multiple) repetition of last parameter/parameter group

None of these specifications must appear in the actual input file. Parameters are parsed from the input file in the subroutine `pimc/spara.f`.

We divide the parameters into several categories: physical parameters, sampling parameters, commands to compute expectations, and computer related commands. Finally we list the drivers: those commands that cause the execution of the random walk or related actions. The parameters are divided from the commands by the `RESTART` parameter.

A successful setup of the system file for PUPI not only requires putting in the necessary keywords, they must also appear in the correct order. The relative sequence of keywords is not important, sometimes, however, common sense is needed: for example, `LABEL` should be used after the corresponding `SYSTEM`, `POT` should be used after `TYPE`, etc.

1.1 Physical Parameters

This section describes the "physics" parameters; those that describe the physical system to be simulated.

NSLICES <`int nslices`> – required
sets the number of time slices into which the path is divided along the τ-direction.

UNITS <`string*8 eunit`> <`string*8 lunit`> – optional
specifies energy and length units used for output labelling; These parameters do not change any numbers.

ENORM <`float enorm`>
specifies normalization constant for energy; the total energy will be divided by this constant for output; is usually equal to the number of particles or molecules.
default value: `enorm`= 1. This corresponds to the total energy being output (instead of energy/particle.)

BOXSIZE (⟨`float ell(i)`⟩,`i=1,`⟨`ndim`⟩) – required
specifies the size of the periodic box in the ⟨ndim⟩ dimensions; If you do not want periodic boundary conditions then you can either make the box quite large (but that will have implications for the g(r) function), or you can comment out the periodic boundary conditions in the file pbc.cm. ⟨ndim⟩ is set in `upi.p`;

BETA <`float beta`> – required
sets the value of the inverse temperature $\beta = \frac{1}{k_B T}$ (units specified by `UNITS` statement). One step in imaginary time is then: $\tau = \beta/\langle \text{nslices} \rangle$

NREF <`int nrefpt`>
specifies number of reference points (used only for fermions):

0 ground state nodes
1 one reference point
2 two symmetrically positioned points in imaginary time
default value: `nrefpt`= 2

TYPE ⟨`string name`⟩ ⟨`int nspins`⟩ ⟨`int number`⟩ ⟨`float hbs2m`⟩ (⟨`int nppss`⟩`(i),i=1,`⟨`nspins`⟩) ⟨`string fname`⟩ [⟨`float charge`⟩] – required
Defines a new type of particle, named `name`. One must have a `TYPE` command for each species of particles.
`number` is the number of particles of this type.
`nspins` gives the number of spin degrees of freedom:

`nspins`= −1	distinguishable particles
`nspins`= 0	bosons
`nspins`> 0	fermions with spin degeneracy of nspins the number of particles with given spin will be `nppss`

`hbs2m` defines $\hbar^2/2m$ for this particle; `hbs2m`= 0 denotes a fixed particle.
The file `fname` contains the initial starting positions. If `fname` starts with a '+', aged configurations will be assumed.
`charge` gives the optional charge of the given species.
Default value: `charge`= 0

LINE ⟨`name`⟩ ⟨`x1`⟩ ⟨`y1`⟩ ⟨`z1`⟩ [(⟨`x2`⟩ ⟨`y2`⟩ ⟨`z2`⟩) `...`] – optional
defines line particles of a specific type `name` (like vortex lines).
The input are triplets (in 3d) or pairs (in 2d). The coordinates of the particles are given as x y z. The number corresponding to the axis parallel to the orientation of the line particle is replaced by the symbol '|' (the UNIX pipe). Example: `LINE vortex 0. 0. |` represents a 'vortex' particle which is a line passing through the origin in the z direction.
The line particle must be parallel to one of the boxes axes.

PLANE ⟨`string name`⟩ ⟨`int d1`⟩ ⟨`float x1`⟩ [⟨`int dn`⟩ ⟨`float xn`⟩ `...`] – optional
defines plane-like particles of a specific type `name`.
The input are pairs d x which specify the planes' position in the following way: The normals are in the d direction and the coordinate is x, where d is an integer $1 \leq d \leq \langle ndim \rangle$. The plane particle must be parallel to one of the boxes axes.

POT `PAIR` <`part1`> <`part2`> <`string filename`> [<`ntermact`> [<`ntermeng`>]] – optional
This uses a spherical interaction between the two given particle types; the file, which contains potential and action tables, should have been created by `squarer`. Without a `POT PAIR` command there will be no pair interaction.
The code saves space by using pointers to the data, thus if 'filename' appears a second time, the data is not read in but the second pointer points to the first location. This implies that the computed pair correlations are also grouped together. The potential and action are assumed to be constant outside the

range of the table.`ntermact` and `ntermeng` are the number of off-diagonal elements used in the evaluation of action and energy. Both are greater or equal to 1.
Default values: `ntermact`= 3, `ntermeng`= 6.

POT `EWALD <cutkn> <string filename>`
(`pimc/addpot.f`; Adds a k-space pair potential. This command must be after all TYPE commands. PUPI stops if more than one such command appears in an input file (for a given system). cutkn is the k-space cutoff of the interaction. If CUTK has already been called, this command over-rides it. A 5 dimensional array is read from the given file. The indices are are respectively: k-vectors, typea, typeb, levels and the last index is 1 for the potential, 2 for the action and 3 for the beta derivative of the action.

NOLEAK turns off the leakage term in the action (ifleak=0.) With NOLEAK paths are rejected only if they actually cross into the negative region of the density matrix. Otherwise they are rejected also if they are close to a node. NOLEAK will save time and be more robust but have a larger time-step error. default is ifleak=1.

PTAIL
Shifts the pressure to account for the potential outside the cutoff. See also `VTAIL`.

VTAIL
Shifts the potential (and hence the energy) to account for the potential outside the cutoff and any other constant. See also `PTAIL`.

IFDIAG reads in the parameter ifdiag (default value 1) used in dmeval.f ifdiag is the level at which the off-diagonal terms are used in the bisection method to accept. ifdiag ≤ 0 means that offdiagonal terms are never used and hence this overrides ntermact on the TYPE PAIR command. ifdiag=0 will be faster but with a larger time step error. ifdiag 1 makes the time/step slower but in principle could give higher acceptance ratios at the lower levels.

FIXNODE passes the fixed-node parameter ifixnd as the second argument. 1 indicates fixed-node, otherwise one uses exact fermions with signs. default is ifixnd=1 .

1.2 Sampling Parameters

These parameters modify how the Metropolis random walk goes through path space. Tuning the default parameters may result in a dramatic speed-up in convergence.

FREESAMP `<string type1> <string type2> ...` – optional
If present, use free particle sampling for the given particle types, otherwise

UPI/PUPI will use the sampling table. Free particle sampling is faster (per step) and more reliable but correlated sampling can have a higher acceptance ratio. Default is correlated sampling.

GAMMA ⟨`int nbpart`⟩ ⟨`string type`⟩ ⟨`float gamma(nbpart,itype)`⟩
sets probability values for the given particle type for commands which move particles or do permutations (see section 1.6) The effect of gamma depends on the particular driver (OMOVE, SELECT or PERMUTE). For details see those sections. Default values for all particle types and numbers is 1. except for fermions, in whose case permutations with even number of particles are zeroed

DISPLACE ⟨`string type`⟩ ⟨`float distance`⟩ [⟨`float gammad`⟩] – optional
sets parameter for global path moves performed by `OMOVE`, `PERMUTE` etc. The entire chain is moved uniformly in a cube of side `distance` – not for fermions or exchanging bosons.
`gammad` is the probability of attempting individual move For example is gammad=0.1, on the average 1/10 of the particles will have a displace move attempted after each pass. Displace moves are NSLICES/2**LEVELS more expensive than OMOVES so one should be cautious in making too many DISPLACE moves. However Displace moves are very good at shaking up an initial configuration (say melting them from an initial configuration.) Default is not to displace.

1.3 Expectations

What follows are commands that create additional averages, or in the case of `WRITEPC` dump out path variables.

WRITEPC <`int freq`>
specifies the number of passes between writes to the compressed path file (`.pc`-file). (variable is `iwrc`) This file can be read in by `READPC` or a movie made by viewer. Each coordinate is packed into 2 bytes; hence the accuracy is 2**(-17)*ell(l). Danger: enormous files can be quickly generated if `iwrc` is small. Also the permutation is written. Default is to never write.

ADDKV ⟨`n`⟩ (⟨`rkadd(i)`⟩`,i=1,n`)
select additional k-vectors for which the structure factor $S(k)$ is calculated. This is similar to `READKSHELLS`. Useful for computing reciprocal lattice vectors in a crystal.

AREA ⟨`string type`⟩
Computes the mean squared area of a path projected on the plane in the direction given by the LINE projection. `type` is the type of particle for which the area is computed. `radius` is used to find the rotating partition function for the case of a vortex line. The mean squared area is needed to determine the response of the system to an infinitesimal rotation about the axis (given by the direction of the line particle.) The output are the scalars 'area^2' and 'moi'. See Eq. (3.26) in RMP.

CM ⟨`string type`⟩
Calculates diffusion constant of the center of mass for particles of the given `type`. This creates a file ⟨`qid`⟩`.dcm` which gives the center of mass distribution around the origin. Only appropriate for a cluster centered at the origin.

CUTK <`float cutk`>
specifies cutoff in k-space for $S(k)$; it's a good idea to use a relatively small cutoff, as the computation is quite slow, and to obtain values of S(K) for large K by a Fourier transformation of $g(r)$
This keyword is required for Ewald sums
`CUTK` makes any previously issued `READKSHELL` ineffective and vice versa
By default $S(k)$ is not computed.

EFFMASS
calculates the single particle diffusion. A file ⟨`qid`⟩`.em` will be created. It contains in the first column the imaginary time and in the following columns the effective mass in the x,y,z and average for each type of particle.

FKT
Calculates the intermediate scattering function,

$$f(k,t) = \frac{1}{N}\langle \rho_k(0)\rho_{-k}(t)\rangle \tag{1}$$

Make sure that `CUTK`, resp. `READKSHELLS` precede this (as well as most other commands). A file ⟨`qid`⟩`.fkt` will be created. It contains in the first column the imaginary time and in the following columns the function for each shell of k-vectors. The binary file ⟨`quid`⟩`.bfkt` contains the block averages only. One uses the covariance program to calculate from the block average the covariance in $f(k,t)$ and do the maximum-entropy analysis.

READKSHELL ⟨`string filename`⟩
add vectors in k-space for which $S(k)$ is calculated; the vectors/shells are read in from a data file which can be generated by `genshells.f`; the format of this data file is the following:
⟨nshlls⟩ ⟨nvects⟩
and for each shell (⟨ks⟩≤⟨nshlls⟩):
⟨mult⟩
and then for each vector (⟨i⟩≤⟨mult⟩):
⟨kcomp(1)⟩ ... ⟨kcomp(ndim)⟩
`READKSHELL` makes any previously issued `CUTK` ineffective and vice versa

LINDEM ⟨`string type`⟩`,` ⟨`string filename`⟩ computes the mean squared deviation of particles from lattice sites The file can be the same as the .ic file on the TYPE line, however then that file must be a perfect lattice. Note we use centering vector to correct for any overall drift away from the lattice sites. This assume translation invariance and no permutations. Also physical exchange of paths will cause a slow increase in rsq. The mean squared displacement is the average: 'deltar^2'.

VWINDOW ⟨`integer virwin`⟩ [⟨`integer maxvirwin`⟩ [⟨t_w⟩]] This command is followed by 1 or 2 integers and possibly a real. The first (`virwin`) puts the virwin'th value of the virial estimator onto the scalar averages for the total energy (it appears as E_{virial}). The second integer (`maxvirwin`) (if present) cuts off the computation after maxvirwin time values (to save time in anal.f). If a third argument (t_w) is present it will be the imaginary time and overwrite `virwin` as with $virwin = \beta/t_w$. (You can use the first if you change number of timeslices but beware of truncation errors.)

1.4 Debugging Support

DEBUG <`int idebug`>
sets debug level (0 means no tests, 1 do some tests, 2 do many tests which slows down execution considerably) The testing is primarly to see if the fermion matrices and inverses are being correctly updated and shuffled around as permutations are done.
The default value is `idebug`= 0

SEED ⟨`int iseed(1)`⟩ [⟨`int iseed(2)`⟩ [⟨`int iseed(3)`⟩]]
used to set the random number seed; as the generator uses 48 bit integers two arguments are possible; the third argument, if given, overrides the default prime adder
Default is to use the clock and possibly the processor number. The number actually used is written out and the job can be rerun by using that as the seed. (We need to replace by the sprng library).

TIMER <`float tcheck`>
specifies maximal time (in seconds) per block in `PERMUTE` etc. OMOVE and SELECT. TIMER overrides the number of steps/block (actually the block is over which ever occurs first.
default is to wait until all passes are done The second argument is the total time of the run. f Unless there is enough time to finish another block it will stop. The time is estimated either from tcheck or the time of the last block in case tcheck is zero.

STOP sets the parameter nstop so that on the nstop entry into fdminit the code will stop. Can be used in conjugtion with `DEBUG` to stop the code at a particular step.

1.5 Restart files

Restart or check-pointing files are produced at the end of each block. The Restart key-word separates the setup commands from the drivers.

RESTART [<`string restarttype`> [`string filename`]] – required
determines the restart properties (after interruption); `NONE` starts at the beginning (first set, first block) with all averages set to zero; `PARTIAL` also restarts at the beginning but reads in the path from the restart file instead of from the initial coordinates on the file referenced by TYPE. Previous averages are

inte the restart file but are not used in PUPI. `FULL` continues with the block following the last completed one, keeping the averages. The run after restart should be exactly the same as if the run had not been stopped. further output will be appended to the restart file (`.rs`) so that results obtained so far are not lost
One can take a completed run and edit the input files either by adding additional blocks to the final block or by adding additional command lines.
Default value is `NONE`

1.6 Drivers

Each driver command starts a new *set* of *blocks*. Each block consists of several *steps*. Expectation values are calculated in each block. They are then dumped together with a cumulated sum over all blocks so far of this set and the corresponding average per block to the output files. Comparing the results of different sets allows for studies of relaxation to equilibrium or estimation of correlation.

The following arguments are common to the most important commands:

<`nbblocks`> gives the number of blocks per set; averages are reset at the beginning of each set and after a `RESTART PARTIAL`; after each block a number of scalar quantities is dumped onto the `.out`–file;

<`steps/block`> specifies the number of steps per block; the operations executed in one *step* vary by command

<`levels`> determines the number of levels which are used for the multilevel rejection; the path is divided into 2^{levels} segments in τ-space (1 would, for example, use midpoint bisection only); 2^{levels} should be smaller than the number of time slices (`NSLICES` line)

OMOVE <`int nbblocks`> <`int steps/block`> <`int levels`>
samples over one–particle moves; for each step the following operations are executed: first, an origin in τ space is picked out at random; then for each particle a random number is drawn; if this number is less than the value of `GAMMA` for 1-particle moves (of this particle type), a move for this particle is attempted; then averages of the quantities of interest are determined A pass consists an attempted move of all particles; after a pass we do analysis, reference move, write out configurations and insertion, and a displacement.

SELECT ⟨`int nbblocks`⟩ ⟨`int steps/block`⟩ ⟨`int levels`⟩ ⟨`int nhits`⟩
samples over permutations: a *pass* consists of `nhits` attempted moves; after a pass we do analysis, reference move, write out configurations and insertion, and a displacement At the beginning of a pass, from the set of all possible 1,2 and 3 particle moves is constructing a table of their probabilities based on the free particle density matrix. These probabilites are multiplied by gamma so the user can emphasis say 3 particle moves versus other moves.

PERMUTE ⟨`int nblocks`⟩ ⟨`int nsteps`⟩ ⟨`int levels`⟩ ⟨`int nhits`⟩
The input is rather similar to `SELECT` but `nhits` and `gamma` must be chosen differently. A *pass* consists of first randomly selecting a time interval,

constructing the table of pair distances, then making `nhits` attempted constructions of cyclic permutations. After a pass we do analysis, reference move, write out configurations and insertion, and a displacement. If the parameter `TIMER` is set, the block is finished when a certain amount of CPU time has been used up.

`PERMUTE` differs from `SELECT` in that the permutation step is accepted or rejected *before* the call to the subroutine `weave` (where the path is constructed) and that the cyclic permutation is constructed from a random walk through label space instead of explicitly looking at *all* possible cyclic permutations. This means that one can do longer permutation moves simply by increasing the parameter `mnmovers` in `upi.p`. Currently `PERMUTE` is less robust than `SELECT` in that the `GAMMA` parameters need to be chosen more carefully. Also it uses many random numbers. One should continue to use `SELECT`, if the permutation acceptance ratio is greater than .5.

`NHITS` should be set to a large value (of the order of (number of particles)^2) because many permutations are tried out before a successful one is found. One can look at "fraction of time in weave" (printed after each block in the output file) to get some idea of the time spent constructing the path.

`GAMMA` parameters have a different meaning in `PERMUTE` and need to be adjusted by hand.
`GAMMA(1,itype)` is used to select the *type* of particle to be moved: `GAMMA`$(1, i)$ is the probability that type i is moved. It is only used if $\langle ntypes \rangle > 1$, when we must have $\sum_i$ `GAMMA`$(1, i) = 1$..

`GAMMA`(j, i) for $j > 1$ is the probability that we continue to construct permutation cycles of length j or longer. The construction is cut off at cycle length m, if `GAMMA`$(m + 1, itype) = 0$ or if m=`mnmovers` (set in `upi.p`). Hence $0 \leq$ `GAMMA`$(j, i) \leq 1$. It is important to allow for one-particle moves by specifying `GAMMA`$(2, i) < 1.0$ so that even in the case that permutation moves are rarely accepted the system undergoes some changes.

Here is a typical setup for generating cycles of length 6 and less: (`mnmovers`= 6)

```
GAMMA 1 4HE 1.
GAMMA 2 4HE .98
GAMMA 3 4HE .95
GAMMA 4 4HE  .8
GAMMA 5 4HE  .8
GAMMA 6 4HE  .8
```

Here is a fermion setup where only cycles of length 1, 3 and 5 are generated:

```
GAMMA 1 3HE 1.    ! not really needed in one-component problems.
GAMMA 2 3HE  .95  ! 5% of time try 1 particle moves.
GAMMA 3 3HE  1.   !  no pair exchanges.
GAMMA 4 3HE  .8   ! 20% of time try 3 particle moves.
```

```
GAMMA 5 3HE  1.   ! no 4 particle exchanges.
GAMMA 6 3HE  0.   ! no 6 or higher exchanges.
```

READPC ⟨`int nblock`⟩ [⟨`int nspb`⟩ [⟨`int levels`⟩ [⟨`string filename`⟩]]]
This is a driver. It reads packed file `filename` and compute energy
`nblock` gives the number of statistical blocks, `nspb` the number of steps/block, and `levels` the time step level. One can use it to compute how the energies depend on the time step, the number of levels of the action etc. with the same paths. The systematic error will be evident even when the statistical errors are larger.

Default values are: `nspb`= 1, `levels`= 1 and `filename`= run identification with extension `.pc`

TESTER ⟨`int islices`⟩ ⟨`float dx`⟩ ⟨`int tol`⟩
is a driver which calculates the gradients to make sure that they are consistent with the action. It compares the analytic gradients with those determined from finite differences. `islices` gives the number of slices tested,
`dx` specifies how far the coordinates are displaced (recommended value: .0001), and
`tol` controls the printout (only errors greater than `tol` are printed)

Printouts are now on ⟨`qid`⟩`.ts` `TESTER` is slow! But it should be done for a new project, after changes in underlying codes, on new machines etc. Make sure that tester is given a typical configuration to analyze, not the perfect crystal with the identity permutation.

If `TESTER` fails, one should break the action into short range long-range and ewald and try to trace where the error is coming from. `TESTER` only determines, if the action is consistent with the gradient both computed within `pairact`. It does not know whether the action in `pairact` is the same as that in `weave`. But this is what the correctness of the virial energy estimator depends on.

2 File Naming convention:

- .cm common block (a file included in many routines)
- .dm is a density matrix file containing everything having to do with a pair interaction
- .f fortran source routine
- .ic data file containing input coordinates (just coordinates at one time slice)
- .os diagnostics generated by setup program
- .p parameters to dimension variables (included in fortran source)
- .pc packed configuration file = unit 90
- .sc scalar averages (time history) =unit 62 *Source files:*

- fnlib.f : 'library' files (do not depend on .p files)
- *set.f : sources which produce .sy input for various problems

  Fixed files produced or required by PUPI:
- qid.sy : system file:sets up system and wavefunction
- qid.out: output of PUPI (unit 6)
- qid.dcm: center of mass distribution (see `CM`)
- qid.sc : scalar averages
- qid.mm : maximum memory used (unit 77)
- qid.gv : ρ(k)
- Error messages are written to unit * (not true in PUPI?)

3 Data file formats

A data file consists of a number of tensors. Two keywords must be present before each tensor: `RANK` and `BEGIN`.

1. `RANK` – required
   This command gives the rank of the next tensor and all of its dimensions.
   `RANK` ⟨`nrank`⟩ ⟨$\mathtt{n}_1$⟩ ... ⟨$\mathtt{n}_{\langle \mathtt{nrank} \rangle}$⟩
2. `BEGIN` [⟨informative labelling possible⟩]
   Data starts in the record following the `BEGIN`. There must be exactly $n_1 * n_2 * \ldots * n_{nrank}$ floating numbers.
3. `CGRID` ⟨integer dimension⟩ (string ⟨cvalues(i)⟩,i=1,$\mathtt{n}_{\langle \mathrm{dimension} \rangle}$)
   specifies character labeling of the values in the specified dimension
4. LABEL dimension value gives an overall label to a given dimension
5. SIZE SIZE indicies minimum maximum specifies that x(indices) are between min and max

4 Results

4.1 Observables

For each block a certain number of observables are calculated. Their values will be written onto the output file (`.out`) and in the restart file (`.rs`) (in binary format, not human-readable).

dfree_⟨type⟩_⟨dir⟩ Gives the free energy change in applying antiperiodic boundary conditions in the x,y and z directions. This can be used to estimate the boson lambda transition temperature as in the paper Pollock and Runge, PRB, **46**,3535 (1992). The average number of even windings minus the odd windings is printed out. One has to average the x,y,z components (for a cube). Then

$$\Delta F = -k_B T \log(\text{dfree}) \tag{2}$$

D the "diffusion constant" which is defined as $(\Delta r)^2/(\text{CPU time})$. This can be used to optimize levels or to turn on or off `FREESAMP`. It does not have physical significance but is only a measure of how quickly (in CPU time) the random walk is diffusing through path space. The definition allows one to compare different temperature and τ values as well but its value will depend on computer, compiler etc. Similarly D_P which measures the diffusion through permutation space. =total number of changes/time. a pair permutation counts 2, a triplet 3 etc.

4.2 Other result files

.bfkt Binary file containing information about the intermediate scattering function $f(k,t)$. (see `FKT`)

.dcm Distribution of center of mass around the origin. See `CM`.

.em Effective mass. See `EFFMASS`

.fkt Intermediate scattering function $f(k,t)$ (see `FKT`)

.grr Two-particle correlation function $g(r)$ in a format which can be read in by data plotting programs. First column: particle distance r, second, fourth, ... column $g(r)$ for given particle type combination.

.ev The virial estimate of the energy. The virial estimator changes the spring term to spread it out across the number of levels, not over the whole nslices. What comes out is the energy versus the time 'window', window=1 is the usual estimator. window=nslices is the maximal. the second the first column is window time, the second the energy, the third the error bars. The fact that it is constant with respect to the error bars is a good test that convergance has been achieved and that the derivatives are correctly computed.

.sk Structure function $S(k)$. You have to specify `CUTK` or similar in order to obtain this file.

5 Analysis tools

5.1 `gofr`

Computes simultaneous values of g(r) and s(k) (in 3d only) for extending g(r) and s(k).

5.2 coself

`coself` performs a maximum entropy analysis. It needs one or more binary `.bfkt` files as input. After giving the prefix the program reads the binary file and outputs the number of time slices over 2, `ntimes`, the number of k-shells, `nlamb`, τ, β, and the number of time slices, `nslices`.

You have to enter the number of energy levels whose positions are to be adjusted, `nvary`, the total number of energy levels, `ne`, and the minimum and maximum value of energy.

The program will then perform the maximum entropy analysis according to the model chosen (in the code) (see `http://www.ncsa.uiuc.edu/Apps/CMP/papers/bon96a/bon96a.html`). The main output file ist `ftn12` (fortran output unit 12 – actual name is machine dependent). It contains the vectors of the k-shells. 'Fixed spectral density ne=' gives the number of fixed energy levels, $\hbar\omega$. Then a list follows which gives $S(k,\omega)$ for every energy value (first column: energy, then $S(k,\omega)$ for every k.)