Qs

Version 1.3

Nicholas M Glykos, 2005

If you ever need to reference this program in your publications *(but note that you are not required —at least by me— to do so)*, please use the following citation : Glykos, N.M. & Kokkinidis M. (2000), "A Stochastic Approach to Molecular Replacement", *Acta Cryst.*, D**56**, 169–174.

Please send comments, suggestions and bug reports to `glykos@mbg.duth.gr`
This document was typeset with LaTeX2e (TeX 3.14159) and the PDF file prepared with *pdfLaTeX*.

The latest version of the program plus electronic reprints of program-related papers can be obtained from the following www address : `http://www.mbg.duth.gr/∼glykos/`

Contents

1 Recent additions

1.1 Version 1.3

- A fine-grained parallelisation has been implemented *via* MPI.
- The automatic temperature determination procedure now uses multiple runs to estimate the temperature range.
- Updated documentation.

1.2 Version 1.2

- Support for multiple different search models added.
- Corrected a bug with the application of a model's overall temperature factor.
- It is now possible to prematurely end a minimisation (without loosing the results) by sending a TSTP signal to the running process.
- Some marginal speed improvements.
- The default mode of the program now uses Boltzmann annealing, a constant move size, and a correlation target.
- Updated documentation (mostly by throwing away the theory and outdated examples).
- A new little program in extras/ has been added to allow a real time graphical representation of how the minimisations are going.

1.3 Version 1.1

- Fixed a $P1$-specific bug.
- Updated documentation.
- Tidy-up code.

Queen of Spades

A stochastic approach to molecular replacement

2 Introduction

The program 'Queen of Spades' (QS from now on), is an attempt to write a multi-dimensional, multi-model, space-group general, molecular (re)placement program. The classical approach to the problem of placing n copies of m different search models in the asymmetric unit of a target structure, is to divide this $6n$-dimensional problem into a succession of 3-dimensional searches (rotational search followed by translational search for the first model, followed by a rotational search and a translational search for the second model etc). QS will attempt to solve this problem by a direct minimisation of a suitably chosen statistic (like the R-factor, or the correlation coefficient) in the $6n$-dimensional space defined by the rotational and translational parameters of the n molecules. The minimisation algorithm is based on a modified reverse Monte Carlo procedure[1,2,3,4] which is made (possibly) practical through (i) the calculation and in-core storage of the molecular transforms of the search models, and (ii) by keeping a table containing the contribution of each molecule to each reflection (thus making the CPU time per step independent of the number of molecules).

I should add here, that treating the molecular replacement problems as $6n$-dimensional, is like shooting a sparrow with a cannon. In reality (and for n molecules per asymmetric unit), the molecular replacement problems can be divided into two $3n$-dimensional problems : a $3n$ dimensional cross rotation function (which will simultaneously determine the orientations of all molecules present in the asymmetric unit), and a $3n$-dimensional translation function (which will determine their positions). The $6n$-dimensional approach would only be valid if the self and cross vectors were not topologically segregated in the Patterson function.

3 A word of caution

QS is *not* the program to use as a first attempt at a molecular replacement problem. It is rather "yet another program to try" when you are desperate enough and all else had failed. There are several reasons for that :

1. QS *is slow.* Although for a typical problem —and on a modern workstation— the program can examine several hundred configurations per second of CPU time, the dimensionality of the configurational space guarantees that you will probably need tens of million moves if you want to have any chance of finding the global minimum of the target function (which, after all, may not coincide with the true crystal structure).

2. QS, being based on a Monte Carlo minimisation, is by nature stochastic, and so there is no guarantee that even if a pronounced global minimum exists, it will be found during your

[1] Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H. & Teller, E. (1953), "Equation of State Calculations by Fast Computing Machines", *J. Chem. Phys.*, **21**, 1087–1092.

[2] Kirkpatrick, S., Gelatt, C.D. Jr. & Vecchi, M.P. (1983), "Optimization by Simulated Annealing", *Science*, **220**, 671–680.

[3] McGreevy, R.L. & Pusztai, L. (1988), "Reverse Monte Carlo Simulation: a new technique for the determination of disordered structures", *Mol. Simulation*, **1**, 359–367.

[4] Keen, D.A. & McGreevy, R.L. (1990), "Structural modelling of glasses using reverse Monte Carlo simulation", *Nature*, **344**, 423–425

lifetime. The good news (if you are an optimist) is that if a pronounced global minimum does exist, there is a slim chance that your next minimisation will fall inside it three hundred moves after you submitted your batch job.

3. The CPU time required per move is proportional to the product $\{\{$ Number of unique reflections $\} \times \{$ Number of symmetry operators $\}\}$. The good news is that the CPU time per step is independent of the number of atoms in the search model and of the number of molecules in the asymmetric unit. The bad news, is that if you insist on using all your reflections to 2Å for the minimisation, you better start the batch job just before your summer vacations (and I bet you won't be very popular upon your return).

In other words, the CPU time per step is proportional to the volume of the unit cell (if you double the length of all cell edges, the CPU time per step will go up by a factor of eight).

4. QS is very demanding on hardware resources, especially physical memory. If your workstation has less than 64 MBytes of RAM, you won't be able to use QS with molecules bigger than (approximately) the size of a lysozyme molecule (and not at a resolution higher than about 4Å). You also need a fast CPU with a floating point co-processor if you want your simulations to finish this week (needless to say that you need a batch queue without time limitations).

4 Installation

QS is distributed both as pre-build executables (for Linux, SGI, windows & VMS machines), and as source code for the other machines.

4.1 Using the stand-alone executables

- **Unix :** Just find the correct executable (in the `bin/` subdirectory of the QS distribution), move it somewhere in the user's path, and give the name `Qs`. If you do not have root privileges and want to try Qs from your area, find the correct executable and define a symbol `Qs` pointing to it. A typical sequence of commands would be :

Qs can output VT100-compatible control characters to help making its output more readable (mainly by highlighting errors and warnings). Because these control characters are a nuisance when incorporated in a log file, the feature is off by default. To switch it on, you must create a copy of the executable (or, better still, make a link with `ln -s`) to a file with the name `Qsi` (`i` for "interactive"). When QS is called as `Qsi`, it will emit control characters understandable by the great majority of terminal emulators.

```
$ cd Qs/bin/SGI/
$ mv IRIX64_R10000_IP27 Qs
$ ln -s ./Qs ./Qsi
$ pwd
/usr/people/something/Qs/bin/SGI
$ alias Qs /usr/people/something/Qs/bin/SGI/Qs
$ alias Qsi /usr/people/something/Qs/bin/SGI/Qsi
```

- **VMS & Windows :** The executables that I distribute are at the current release with the exception that the FFT step is still based on a radix-2 algorithm. The reason for this is that I couldn't even try to build FFTW on these two architectures. To use the program from VMS you will need something like `$Qs :== $dsk$12:[my.directory]QS.EXE` in your login.com file. For windows, you will probably have to run QS *via* MS-DOS.

4.2 Building from source

The minimum requirement for building QS is that you have a C compiler and the the FFTW libraries compiled in the single precision mode (FFTW can be obtained from `http://www.fftw.org/`. You need version 2.1.3 or 2.1.5. The newer v.3.x libraries will not do (as yet)). Assuming that you built FFTW with the `--enable-float --enable-type-prefix` options, and that the libraries and include files are in the `cc`'s and `ld`'s search path, the following should suffice

```
cd Qs/src/
./configure
make
make install
```

Building a usable executable on non-Unix machines is expected to be rather more challenging, with the first challenge being to build the FFTW library. If you manage to build FFTW, then you might as well give it a try with Qs by giving from the command line (if one exists) :

IRIX users : if you decide that you would like to prepare an executable optimised for your machine (with options like -Ofast=??? -r???? -TARG:platform=???:-processor=r????), you must take care to prepare the FFTW libraries using the same ABI (-32, -n32, -64), or you won't be able to link the program.

```
cc <my_optimisation_flags> Qs.c -o Qs -lsrfftw -lsfftw -lm
```

where `<my_optimisation_flags>` correspond to the highest safe level of optimisation supported by your compilers (you can always check that everything is OK by comparing the results from the test file included in the `example` directory).

If you are getting error messages during compilation, try adding a `-DVMS` flag in the `cc` step above. If the problems persist, please do send me a mail message, but do not expect too much.

It is still possible to make an executable that does not depend on FFTW, but this requires that you have the Numerical Recipes library (or code) for their radix-2 FFT. I am afraid that because the Numerical Recipes code is copyrighted, I can not distribute it with Qsś source code. If you do have the NR code, all you need to do is to edit the source code and change the line `#define FFTW 1` to `#undef FFTW` and to add the NR library in the `cc` line (with something like `-lrecipes`).

4.3 Parallel Qs

Building and using the parallel version of Qs requires some familiarity both with some details of your computing environment and with using Qs *via* a script (and not in its automatic mode). I suspect that non-adventurous first-time users would rather prefer skipping this section and use the uniprocessor version of the program (in its automatic mode).

Version 1.3 of Qs contains an implementation of a fine-grained parallelisation of the program based on MPI. Because this is a new feature and because you will have to built Qs from source anyway, I have kept everything related to parallelisation in this section. The rest of the documentation should apply identically to the parallel version with the exception of the command line (or script) needed to run a parallel program on your cluster. This section also contains results from some tests that I performed on two Beowulf clusters. To cut a long story short (and to save you from possibly unnecessary efforts) :

1. You need a high-speed interconnect (Gigabit ethernet or better). You may get an acceptable scale-up even from a cluster based on a fast (100 Mbps) ethernet, but only for problems for which the target structure belongs to a high symmetry spacegroup.
2. Even with a Gigabit ethernet interconnect the scale-up will be far from ideal for low symmetry problems.
3. For most problems the optimum number of processors will be around 4 to 8 processors per minimisation (and so, you should not expect to submit a Qs job to your 256-node brand-new-cluster with something like `mpirun -np 256`).

The bright side of these findings are :

- Finding 1 has no bright side.
- Finding's 2 bright side is that it is exactly these high-symmetry problems that make the uniprocessor version of Qs unbearably slow.
- Finding 3 is not as bad as it sounds : even with 4 processors per minimisation (and assuming that you want to perform 5 minimisations) you can easily keep 20 processors busy for a week or two.

Building it : I have tried to keep the parallelisation transparent with respect to the uniprocessor version of the program which means that in the absence of a suitable `define` (passed to the compiler) there should be no difference from the uniprocessor version. Compiling and linking the parallel version of QS depends on your cluster set-up, but ideally (and if you have a properly set-up LAM or MPICH implementation) you could simply say something like :

```
mpicc -DMPI -o Qs_MPI Qs.c -lsrfftw -lsfftw -lm
```

(which assumes that `mpicc` already known which compiler and optimisation flags are best for your cluster). The line shown above also implies that you already have the FFTW libraries ready to go (note that you do *not* need a parallel version of the FFTW libraries). Once you have an executable you can quickly test it using the files in the `example/` directory of the distribution with something in the spirit of :

```
mpirun -np 4 Qs_MPI example.in
```

If all goes as planned, just copy the parallel version of the program somewhere in the users path with a suitable name and you're done (installation-wise, the rest of the steps are problem-specific).

Determination of 'best' number of processors : The best number of processors for any given problem depends mainly on two parameters : the number of unique reflections and the number of crystallographic symmetry operators (for the given space group). In general, the program will perform better as the number of unique reflections decreases and the symmetry increases. To get an idea of what to expect for your problem, I wasted some CPU time on a dual Athlon Beowulf with a Gigabit Ethernet to run a few tests with problems of different sizes. In all cases, I used short runs (200,000 steps each) and I assumed that the machines were otherwise idle. The results are shown below (wallclock times in seconds) :

P422, 4692 refl.

Proc	Wall	Scale-up
1	4909	1.0000
2	2427	1.9636
3	1850	2.6535
4	1474	3.3303
5	1236	3.9716
6	1069	4.5921
7	959	5.1188
8	867	5.6620
9	792	6.1982
10	743	6.6069
12	661	7.4266
14	598	8.2090
16	558	8.7974
18	526	9.3326
24	469	10.4669
40	399	12.3032

P422, 9383 refl.

Proc	Wall	Scale-up
1	8421	1.0000
2	4733	1.7792
3	3464	2.4310
4	2723	3.0925
5	2315	3.6375
6	2021	4.1667
7	1792	4.6992
8	1644	5.1222
9	1516	5.5547
10	1466	5.7442
12	1553	5.4224

P422, 18698 refl.

Proc	Wall	Scale-up
1	15396	1.0000
2	8888	1.7322
3	6311	2.4395
4	4967	3.0996
5	4297	3.5829
6	3696	4.1655
7	3466	4.4420
8	3418	4.5043
9	3625	4.2471
10	3656	4.2111

P21, 9383 refl.

Proc	Wall	Scale-up
1	2386	1.0000
2	1510	1.5801
3	1218	1.9589
4	1039	2.2964
5	922	2.5878
6	844	2.8270
7	2330	1.0240

To determine the best number of processors for your problem proceed as follows :

- Run the program on a stand-alone machine using the automatic mode and stop it after in starts the actual minimisation (as described in the next sections of this document). The program will create a file named `Qs_auto.in` in its current directory.
- Edit the file `Qs_auto.in` and change the lines saying

```
CYCLES              5
STEPS               10000000
```

to

```
CYCLES              1
STEPS               200000
```

(the actual number of steps you will find in `Qs_auto.in` depends on the number of molecules per asymmetric unit and may be different from the one shown above).

- Copy the files `Qs_auto.in`, `data.hkl` and `model.pdb` (or `model1.pdb`, `model2.pdb`, etc) to suitably named directories (like `1proc/`, `2proc/`, etc). The idea is that you will be successively submitting otherwise identical QS jobs to increasing number of processors.
- After the jobs finish, find the wallclock time recorded for each of the runs. This is written out by the program after each minimisation finishes (do a `tail -50 <log file>` and you should see it). Error messages of the type `"FFTW failed to read the wisdom file ..."` can safely be ignored.
- Decide how many processors per minimisation you will use.

Production runs : This is easy :

- Prepare directories like `minim1/`, `minim2/`, ... `minim5/` and copy the `Qs_auto.in`, `data.hkl` and `model.pdb` (or `model1.pdb`, `model2.pdb`, etc) files in them.
- Edit the `Qs_auto.in` file, change the number of steps to its original value (say, 10,000,000), keep the number of cycles to the value of 1, find the line saying '`SEED 147579`' and change it to a different (integer) number. The values for `SEED` should be different for each of your minimisations, otherwise you will be performing a large number of identical minimisations.
- Submit the jobs using the already determined 'best number of processors'.

4.4 Testing the executable

Assuming that the executable is in your PATH (or that you have a symbol defined for it), and that its name is `Qs`, go to the `/my_dirs/Qs/example/` directory, make your window at least 80 characters wide, and type `Qs example.in`. What you should see on your terminal should be similar to this :

```
host# Qs example.in

            QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
             QQQQQQQQQQQQQQQQQQQQQQ QQQQQQ QQQQQQQQQQQQQQQQQQQQQQ
             QQQQQQQQQQQQQQQQQQQQQ   QQQQ   QQQQQQQQQQQQQQQQQQQQQ
              QQQQQQQQQQQQQQQQQQQ    QQQQ    QQQQQQQQQQQQQQQQQQQ
               QQQQQQQQQQQQQQQQQ     QQQQ     QQQQQQQQQQQQQQQQQ
                 QQQQQQQQQQQQQ      QQQQQQ      QQQQQQQQQQQQQ
                    QQQQQQ          QQQQQQ          QQQQQQ
                                   QQQQQQQQ
                                  QQQQQQQQQQ
                                 QQQQQQQQQQQQ
                                 QQQQQQQQQQQQ
                                  QQQQQQQQQQ
                                    QQQQQQ
```

```
                              Queen of Spades
                                Version 1.3

--------------------------------------------------------------------------------

#
# Example script for testing Qs
#

TARGET          R-FACTOR
CYCLES          3
STEPS           100000
STARTING_T      0.01500
FINAL_T         0.00500
INFO            1000

NOISE_ADDED     0.20

RESOLUTION      150.0 4.0
AMPLIT_CUTOFF   500.0
SIGMA_CUTOFF    0.0
RANDOM_SELECT   1.0
FREE            0.20

MODEL           example.pdb
DATA            example.hkl
GLOBAL_B        20.0
MOLECULES       1

SEED            357539
SCALECELL       4.0
MAXGRIDSPACING  1.0
SCMODE          wilson
INTERPOLATION   linear
POSTSCRIPT      colour

CELL            103.900   38.700   34.000   90.000  100.600   90.000
GROUP           5


--------------------------------------------------------------------------------

Minimisation performed against the R-factor.
Will perform 3 independent minimisations.
Number of steps for minimisation : 100000
Information about the minimisation will be printed every 1000 moves.
Noise added to Fs : max fraction  0.20000
Resolution limits set to  150.00 -  4.00 Angstrom.
Amplitude cutoff set to  500.00
Reflections with F/sigma(F) less than  0.00, will be rejected.
Only a fraction  1.000 of the available reflections will be used.
Free value will be calculated over a fraction 0.200 of the data set.
Model PDB file name set to example.pdb
Reflection file name set to example.hkl
Global temperature factor for model set to 20.000000
Number of molecules in asymmetric unit : 1
Random number generator reset to 357539
Initial scale value set to  4.00
```

```
Maximum grid spacing (in Angstrom) is 1.000000
Wilson-like scaling will be used.
Will be using linear interpolation.
Colour postscript output requested.
Cell parameters  103.90  38.70  34.00  90.00  100.60  90.00
Space group    5 with   2 symmetry operators
Symmetry operator :
                 -1 +0 +0
                 +0 +1 +0
                 +0 +0 -1
with translation vector : +0.0000000 +0.0000000 +0.0000000
2 symmetry operators read in.
Space group given. Lattice type is C (only used for the packing diagrams).

Target function for minimisation : R-factor
No bulk-solvent correction requested.

Temperature control :
         Temperature is linearly dependent on time, and will be
         decreased from To at t=0, to Tt at t=100000.
Move size control :
         The modulus of the moves attempted is linearly dependent on the current
         R-factor (or correlation) and time, with maximum possible values of
         180.00 degrees for the kappa angle, and 0.5000 fractional units
         for any of the translations.

-------------------------------------------------------------------------------

Reading atoms ...                       994 atoms read (0 unknown)
Centre of mass at                         -3.272   30.256   33.689
Box dimensions (A) :                      31.760   43.723   33.092 along x,y,z
Translating/rotating ...                done.
Centre of mass at                         -0.000    0.000   -0.000
Box dimensions (A) :                      45.112   31.867   28.035 along x,y,z
Reading reflections ...                 414 read.
Reflections for free value              81
Excluded reflections                    2351
Lowest resolution reflection :          51.1 Angstrom
Highest resolution reflection :         4.0 Angstrom
Big cell :                               180.448  127.467  112.140
Grid                                     192  128  120
 with spacing                              0.940    0.996    0.934
Allocate memory ...                     done.
FFTW is learning how to do FFTs ...     done.
Saving FFTW's wisdom file ...           done.
Atomic density profiles (B=20.0) ...    done.
Make electron density map ...           done.
Write out projections ...               done.
Calculate molecular transform ...       done in 1 seconds.
Rearranging transform  ...              done.
Write out central sections ...          done.
Initialisations ...                     done.
Ready to roll after ...                 4 seconds.

-------------------------------------------------------------------------------


Starting minimisation   1.
Initial R-factor  0.64758
```

```
Starting free value   0.52872
$TABLE: Qs simulation 1:
$GRAPHS
:R      vs time:A:1,2:
:Rfree  vs time:A:1,3:
:Temp   vs time:A:1,4:
:R & Rf vs time:A:1,2,3:
$$


           TIME       R-FACTOR      FREE        TEMP
$$
$$
            1000      0.558826    0.576907    0.014900
            2000      0.574403    0.564562    0.014800
            3000      0.558421    0.565640    0.014700
           ..........................................
          100000      0.321598    0.384552    0.005000
$$
Best solution had a R-factor (or 1-Corr) of  0.30767
Few cycles down the gradient starting from best solution ...
Starting R-factor (or 1-Corr)    0.30767
Starting free value ...          0.36779
           91000      0.308809    0.369162    0.000900
           92000      0.309151    0.368926    0.000800
           93000      0.308709    0.369571    0.000700
           94000      0.308692    0.368035    0.000600
           95000      0.308307    0.365925    0.000500
           96000      0.308363    0.365070    0.000400
           97000      0.308948    0.364910    0.000300
           98000      0.308106    0.364235    0.000200
           99000      0.307692    0.365615    0.000100
          100000      0.307527    0.365723    0.000000
Done the minimisation in ...            145 seconds.
Best solution had a R-factor (or 1-Corr) of  0.30730


Starting minimisation   2.
Initial R-factor  0.61194
Starting free value   0.62831
$TABLE: Qs simulation 2:
$GRAPHS
:R      vs time:A:1,2:
:Rfree  vs time:A:1,3:
:Temp   vs time:A:1,4:
:R & Rf vs time:A:1,2,3:
$$


           TIME       R-FACTOR      FREE        TEMP
$$
$$
            1000      0.566813    0.567553    0.014900
            .........................................
```

Normally, at least one of the minimisations is expected to converge to the correct solution (with an *R*-factor of about 30%).

5 A do-er's guide

5.1 One or more copies of the same model

1. Prepare an ASCII file containing $h, k, l, Fo, \sigma(Fo)$. Give it the name `data.hkl`
2. Get a copy of the PDB file containing the coordinates of your search model. Edit the `CRYST` card, and give unit cell dimensions and space group **number** of the **target** structure. Give it the name `model.pdb`
3. Do `Qs -reso <low> <high> -auto <xxx>`, where `<low>` and `<high>` are resolution limits, and `<xxx>` is the number of copies of your search model in the asymmetric unit of the target structure (eg. `Qs -reso 20 4 -auto 2`). Check that the program goes through the initialisation phase and starts the actual minimisation.
4. Start a proper batch job (and forget about it for a few days ? a few weeks ?).

5.2 One or more copies of several (≥ 2) models

1. Prepare an ASCII file containing $h, k, l, Fo, \sigma(Fo)$. Give it the name `data.hkl`
2. Get a copy of the each of the PDB files containing the coordinates of your search models. Edit the `CRYST` cards, and give unit cell dimensions and space group **number** of the **target** structure. Give them the names `model1.pdb`, `model2.pdb`, ...
   NOTE WELL : You will need to have separate files for each and every model present in the asymmetric unit of the target crystal structure, even if these models correspond to the same molecule. To make this clear : suppose that the target crystal structure contains two copies of molecule A and two copies of molecule B. Then you need four PDB files : `model1.pdb` and `model2.pdb` are identical and contain the coordinates of molecule A. `model3.pdb` and `model4.pdb` are identical and contain the coordinates of molecule B.
3. Do `Qs -reso <low> <high> -auto <xxx>`, where `<low>` and `<high>` are resolution limits, and `<xxx>` is the total number of models (including multiple copies of the same molecule) that are present in the asymmetric unit of the target structure (for the example above, `Qs -reso 20 4 -auto 4`). Check that the program goes through the initialisation phase and starts the actual minimisation.
4. Start a proper batch job (and forget about it for a few days ? weeks ?).

5.3 In more detail : The `data.hkl` file

If you have a .mtz file containing just $h, k, l, F, \sigma(F)$, mtzdump is the easiest way to prepare the required file :

```
mtzdump hklin my.mtz\
> data.hkl
reso 20.0 4.0
nref 100000
go
```

Then, edit the file data.hkl with your favourite editor to remove the lines above and below the list of reflections.

The file `data.hkl` is a free-format ASCII file containing $h, k, l, F, \sigma(F)$ for your unknown (target) crystal structure. Use all data to the resolution limit you want to work at (eg. 4Å). Do not use all your data to, say, 2Å with a large protein, unless you have plenty of physical memory and a year of CPU time to spare. Alternatively, you can place all your data in the `data.hkl` file, and then run the program with something like `Qs -reso 20 4 -auto 3`, where the `-reso 20 4` flag defines resolution limits. If you work with the CCP4 suite of programs, the program you need is `mtz2various` (but `mtzdump` will also do). Your file should look like this :

```
 -23   1   9      329.80      3.55
 -23   1  11      190.39      4.39
 ................................
```

5.4 In more detail : The PDB files

QS knows nothing about most of the PDB identifiers, and can not understand anisotropic B factors. Your PDB file must be as simple as it can be : it should contain a `CRYST` card (followed by unit cell dimensions and space group number for the *target* structure) and a series of `ATOM` cards on the following lines. QS will ignore all lines that do not start with `CRYST` or `ATOM`. This means that if you want to include `HETATM`s in the calculation, you must change `HETATM` to `ATOM` in your PDB file. Also note that QS doesn't check the occupancy column, so if your PDB file contains discretely disordered residues, QS will include them in the calculation at full occupancies. Additionally, QS will assign to all atoms of a model the same isotropic temperature factor. If some parts of your models have Bs in the 60Å^2 range, you may want to exclude them from the calculation by deleting them from the PDB file. Finally, please note the QS's universe comprise only carbon, nitrogen, oxygen and sulphor atoms. Everything else will be treated as carbon atoms with the exception of (i) phosphorus atoms which will be treated as sulphor atoms, and, (ii) hydrogen atoms which will be ignored. A minimal but functional PDB file could be :

```
CRYST1   38.070   33.200   46.120  90.00 110.06  90.00     4
ATOM      1  N   LYS     1      -7.237  14.795 -10.161  1.00 58.48
ATOM      2  CA  LYS     1      -6.682  13.509  -9.562  1.00 57.62
ATOM      3  C   LYS     1      -6.904  13.768  -8.083  1.00 49.27
ATOM      4  O   LYS     1      -6.436  14.842  -7.821  1.00 46.31
ATOM      5  CB  LYS     1      -5.227  13.264  -9.843  1.00 47.34
ATOM      6  CG  LYS     1      -4.509  11.995  -9.589  1.00 40.83
ATOM      7  CD  LYS     1      -3.472  11.949  -8.511  1.00 42.51
ATOM      8  CE  LYS     1      -3.142  10.659  -7.735  1.00 29.40
.............................................................
ATOM    994  OXT LEU   129       9.815   9.083   4.618  1.00 98.99
```

In the risk of becoming repetitious : The `CRYST` card should be followed by $a, b, c, \alpha, \beta, \gamma$ and space group number of the **target** structure.

5.5 In more detail : Running the program

Plenty of memory ?
If you have loads of physical memory, you can make use of this extra memory to speed-up the program by a factor of approximately 25%. This you do as follows : start the program as discussed in the main text, stop it with CTRL-C, edit the file `Qs_auto.in`, remove the line that says `GRAUTO`, add a line `SCALE 8.0`, add a line `MAXGRID 1.0`, and finally, change the line `INTERPOLATION linear` to `INTERPOLATION none`. Save the modified `Qs_auto.in` file and give `Qsi Qs_auto.in`. The program will probably stop with an error message about how much memory is required with this set of parameters. If you have enough memory, use `Qsi -force Qs_auto.in` to run the program (the `-force` flag is needed both for interactive and batch jobs).

In the directory where the `data.hkl` and `model.pdb` (or `model?.pdb`) files are —and assuming that the symbol `Qs` has been defined (as discussed in the installation section), give : `Qs -auto <xxx>` where `<xxx>` is the total number of search models per asymmetric unit of the target structure, or `Qs -reso <low> <high> -auto <xxx>` if you want to explicitly define resolution limits (for example, `Qs -auto 1` or `Qs -reso 20 4 -auto 2`, etc.).[5] The program will do its best to determine parameters suitable for the given problem, and if it succeeds, it will start the calculation. The name of the game in this procedure is memory : if the physical memory of your system is adequate for storing the whole molecular transform (or transforms in the multi-model case), then you can stop the program and submit it as a batch job. If, on the other hand, QS allocated 128 MBytes of memory when only 64 MBytes of it is physical memory, the program will never start : it will endlessly be reading and writing to the swap space on the disk with no chance of even finishing the FFT step. If that's the case, you will probably have to reduce the resolution of your data and try again. What I suggest you do, is to run the program interactively till it gets past the FFT stage. For medium-sized proteins and with adequate physical memory, this should only take a few minutes (or even seconds). If on the other hand the physical memory is not enough, the program will get stuck in the electron density calculation and FFT steps. What you should see (in the case of a multi-model problem) should look similar to this :

[5] If you have defined a symbol `Qsi` pointing to the QS executable (as discussed in the installation section), try running the program interactively using the `Qsi` instead of the `Qs` symbol. The reason is that possible error messages will be shown in bold, and are thus significantly easier to spot.

```
host# Qs -reso 50 4 -auto 3

            QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
            QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
            QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
             QQQQQQQQQQQQQQQQQQQQQQ QQQQQQ QQQQQQQQQQQQQQQQQQQQQQ
             QQQQQQQQQQQQQQQQQQQQQ   QQQQ   QQQQQQQQQQQQQQQQQQQQQ
              QQQQQQQQQQQQQQQQQQQ    QQQQ    QQQQQQQQQQQQQQQQQQQ
               QQQQQQQQQQQQQQQQQ     QQQQ     QQQQQQQQQQQQQQQQQ
                 QQQQQQQQQQQQQ      QQQQQQ      QQQQQQQQQQQQQ
                    QQQQQQ          QQQQQQ          QQQQQQ
                                   QQQQQQQQ
                                  QQQQQQQQQQ
                                 QQQQQQQQQQQQ
                                 QQQQQQQQQQQQ
                                  QQQQQQQQQQ
                                    QQQQQQ


                                Queen of Spades
                                  Version 1.3


-------------------------------------------------------------------------------

############################################################
##                                                        ##
##              Automatically created by Qs               ##
##                                                        ##
############################################################

############################################################
#
# Target function (can be R-FACTOR, CORR-1 or CORR-2) and
# number of minimisations and steps.
#
TARGET          CORR-1
CYCLES          5
STEPS           30000000

############################################################
#
# Annealing schedule & move size control.
# This is for a Boltzmann annealing schedule with a constant
# move-size and a fixed starting temperature.
#
BOLTZMANN
STARTING_TEMP   0.0700

############################################################
#
# Reflection selection.
#
KEEP            0.70
AMPLIT_CUTOFF   1.0
SIGMA_CUTOFF    2.0
RESOLUTION      50.00 4.00
RANDOM_SELECT   1.0
```

```
############################################################
#
# Fraction of reflections for free set.
#
FREE            0.10

############################################################
#
# Files to use for reading model(s) and data.
#
MOD1            model1.pdb
MOD2            model2.pdb
MOD3            model3.pdb
DATA            data.hkl

############################################################
#
# Scales, grid, random number generator seed, B-factors etc.
#
GRAUTO
SCMODE          wilson
INTERPOLATION   linear
SEED            47579
B_M1            20.00
B_M2            20.00
B_M3            20.00

############################################################
#
# Log file and postscript-related things.
#
INFO            1000
POSTSCRIPT      colour

############################################################
#
# Finally, cell, space group and molecules per a.u.
#
CELL             64.660  85.460  83.370 90.000 112.030 90.000
GROUP           4
MOLECULES       3
#
#
############################################################


------------------------------------------------------------------------------

Minimisation performed against the 1-Corr(Fo,Fc) target.
Will perform 5 independent minimisations.
Number of steps for minimisation : 30000000
Fraction of strongest reflections to keep : 0.700000
Amplitude cutoff set to    1.00
Reflections with F/sigma(F) less than  2.00, will be rejected.
Resolution limits set to   50.00 -  4.00 Angstrom.
Only a fraction  1.000 of the available reflections will be used.
Free value will be calculated over a fraction 0.100 of the data set.
Model number 1 : PDB file name set to model1.pdb
Model number 2 : PDB file name set to model2.pdb
Model number 3 : PDB file name set to model3.pdb
Reflection file name set to data.hkl
Will try to determine SCALE and MAXGRIDSPACING automatically.
Wilson-like scaling will be used.
Will be using linear interpolation.
Random number generator reset to 47579
Model number 1 : overall B-factor set to 20.000000
Model number 2 : overall B-factor set to 20.000000
Model number 3 : overall B-factor set to 20.000000
Information about the minimisation will be printed every 1000 moves.
Colour postscript output requested.
Cell parameters  64.66  85.46  83.37  90.00  112.03  90.00
Space group    4 with   2 symmetry operators
Symmetry operator :
                 -1 +0 +0
                 +0 +1 +0
                 +0 +0 -1
with translation vector : +0.0000000 +0.5000000 +0.0000000
Number of molecules in asymmetric unit : 3
2 symmetry operators read in.
Space group given. Lattice type is P (only used for the packing diagrams).

Target function for minimisation : 1.0-Corr(Fo,Fc)
No bulk-solvent correction requested.

Temperature control :
         Boltzmann scaling selected : the temperature at each step (k)
         will be given by T=T0/ln(k). T0 is set to the value defined by
         the keyword STARTing_temp.
Move size control :
         The modulus of the moves attempted is constant and independent of
         either time or R-factor (or correlation). Its maximum value only
         depends on the resolution of the input data.

--------------------------------------------------------------------------------


Reading atoms (Model 1)                 2240 atoms read (0 unknown)
Centre of mass at                         19.749   23.683   64.488
Box dimensions (A) :                      57.433   58.288   56.652 along x,y,z
Translating/rotating ...                done.
Centre of mass at                          0.000    0.000   -0.000
Box dimensions (A) :                      65.001   51.063   50.244 along x,y,z

Reading atoms (Model 2)                 869 atoms read (0 unknown)
Centre of mass at                         39.417   33.807   65.919
Box dimensions (A) :                      49.416   32.517   81.659 along x,y,z
Translating/rotating ...                done.
Centre of mass at                          0.000    0.000   -0.000
Box dimensions (A) :                      85.850   41.229   27.084 along x,y,z

Reading atoms (Model 3)                 814 atoms read(0 unknown,38 P->S)
Few or no CA atoms. Packing diagrams will include all atoms (keyword PACKall).
Centre of mass at                         37.756   36.302   41.899
Box dimensions (A) :                      75.250   22.857   24.875 along x,y,z
Translating/rotating ...                done.
Centre of mass at                         -0.000   -0.000    0.000
Box dimensions (A) :                      75.564   22.832   25.470 along x,y,z
```

```
Calculating |F| cutoff ...                112.70
Reading reflections ...                 4943 read.
Reflections for free value              525
Excluded reflections                    17136
Lowest resolution reflection :          19.5 Angstrom
Highest resolution reflection :         4.0 Angstrom
Scale set to                             4.000
Big cell :                               343.400  204.253  200.977
Grid                                     384  210  210
 with spacing                              0.894    0.973    0.957
Physical memory required                193 MBytes
Allocate memory ...                     done.
FFTW is learning how to do FFTs ...     done.
Saving FFTW's wisdom file ...           done.
Atomic density profiles (B=20.0) ...    done.
Make electron density map (Model 1) ... done.
Atomic density profiles (B=20.0) ...    done.
Make electron density map (Model 2) ... done.
Atomic density profiles (B=20.0) ...    done.
Make electron density map (Model 3) ... done.
Write out projections ...               done.
Calculate molecular transform (1) ...   done in 5 seconds.
Calculate molecular transform (2) ...   done in 5 seconds.
Calculate molecular transform (3) ...   done in 5 seconds.
Rearranging transforms  ...             done.
Write out central sections ...          done.
Initialisations ...                     done.
Ready to roll after ...                 72 seconds.

-------------------------------------------------------------------------------

Starting minimisation   1.
Initial 1-Corr(Fo,Fc)  0.94847
Starting free value   0.88592
$TABLE: Qs simulation 1:
$GRAPHS
:1-C(Fo,Fc)        vs time:A:1,2:
:Free 1-C(Fo,Fc)   vs time:A:1,3:
:Temp              vs time:A:1,4:
:1-C(Fo,Fc) & Free vs time:A:1,2,3:
$$


           TIME      1-C(Fo,Fc)     FREE        TEMP
$$
$$
            1000      0.931011    0.921275    0.010132
            2000      0.921983    0.927676    0.009209
            3000      0.924896    0.916017    0.008743
            4000      0.922867    0.919347    0.008440
    .........................................
```

You can now stop the program, and submit a proper batch job : for unix something like

```
batch
/usr/local/bin/Qs -reso 50 4 -auto 3 > LOG
<CTRL-D>
```

and for VMS :

```
submit/noprint/notify/queue=for_ever/log=dsk$23:[my.dir]QS.LOG Qs.com
```

where the `Qs.com` looks like

```
$ set def dsk$21:[my.directory]
$
$ Qs :== $dsk$56:[Qs.directory]Qs-dec4000-vms6.exe
$ Qs -auto 1
$
$ exit
```

6 Interacting with the program

QS is definitely a non-interactive program. But there are two cases for which some interaction would have been useful.

The first is being able to watch the (slow) progress of the minimisations. This is easy : if you are saving a log file (as shown above), then you have everything you need for watching QS's progress. By far the easiest way for doing this is to use the program `loggraph` that comes as part of recent versions of the CCP4 suite of programs. If you have loggraph, then doing

If you are on the ball, you must have noticed that redirecting the standard output definitely does not involve any interaction whatsoever with the program. But there you go.

```
loggraph LOG
```

should allow you to view graphs similar to those shown in all QS-related papers. A second (playful) way (for unix machines only) can be found in the `Qs/extras/QsLOG` directory of the QS distribution.

The second case that calls for some interaction with the program is best illustrated with an example : suppose that two hours after you've started your 18-dimensional minimisation the *R*-factor drops to 30%. But, alas, there are still 48,900,000 moves to go, which means that you will have to seat and wait for a couple of days (or weeks). Don't even think of killing the batch job and restarting the program with fewer steps : for the majority of annealing schedules this will lead to a different minimisation path, and the solution will be lost. So what do you do ? Here it goes :

NOTE WELL : This command assumes that you run one and only one QS job. `killall` will send the signal to all running QS processes that you own. If you are running more than one QS job, all of them will receive the signal, and all of them will end their current minimisation. In case of multiple QS processes, you have to use `kill` with a PID specification.

```
killall -s TSTP Qs
```

What this will do is to instruct the program to (i) end the current minimisation, (ii) refine and save the best solution found up to the point that the signal was received, and, (iii) move-on to the next minimisation. If by any chance you want to do the same thing while running QS interactively (!), press CTRL-Z (which emits the TSTP signal).

If you do not have a unix machine, have patience.

7 Program output

The basic program output is a series of PDB files containing the final coordinates of your search model(s) from each minimisation. The program will also create a PDB file containing the coordinates of all symmetry equivalent copies of all search models, which can be viewed directly with a molecular graphics program (to check for bad contacts).

QS will create quite a number of files. A directory devoted to the program will look like this after the end of a typical single-model run :

```
-rw-r--r--    1 glykos   user      200182 Apr  1 19:16 001.ps
-rw-r--r--    1 glykos   user      200182 Apr  1 19:16 010.ps
-rw-r--r--    1 glykos   user      100240 Apr  1 19:16 100.ps
-rw-r--r--    1 glykos   user      532644 Apr  1 23:42 LOG
-rw-r--r--    1 glykos   user       66661 Apr  1 19:16 Qs.pdb
-rw-r--r--    1 glykos   user       66708 Apr  1 19:26 Qs001_mol_1.pdb
-rw-r--r--    1 glykos   user      133306 Apr  1 19:26 Qs001_packing.pdb
-rw-r--r--    1 glykos   user       66708 Apr  1 19:35 Qs002_mol_1.pdb
-rw-r--r--    1 glykos   user      133306 Apr  1 19:35 Qs002_packing.pdb
-rw-r--r--    1 glykos   user       66708 Apr  1 19:44 Qs003_mol_1.pdb
-rw-r--r--    1 glykos   user      133306 Apr  1 19:44 Qs003_packing.pdb
-rw-r--r--    1 glykos   user       66708 Apr  1 19:53 Qs004_mol_1.pdb
-rw-r--r--    1 glykos   user      133306 Apr  1 19:53 Qs004_packing.pdb
-rw-r--r--    1 glykos   user       66708 Apr  1 20:02 Qs005_mol_1.pdb
-rw-r--r--    1 glykos   user      133306 Apr  1 20:02 Qs005_packing.pdb
-rw-r--r--    1 glykos   user         392 Apr  1 09:59 Qs_auto.in
-rw-r--r--    1 glykos   user       44892 Apr  1 09:59 data.hkl
-rw-r--r--    1 glykos   user      196808 Apr  1 19:18 mod_001.ps
-rw-r--r--    1 glykos   user      201390 Apr  1 19:18 mod_010.ps
-rw-r--r--    1 glykos   user      101407 Apr  1 19:18 mod_100.ps
-rw-r--r--    1 glykos   user       80578 Apr  1 09:59 model.pdb
```

In the case of a multi-model run of the program, you will see something similar to this :

```
-rw-rw-r--    1 glykos   glykos    2046649 Jul 23 15:28 LOG
-rw-rw-r--    1 glykos   glykos     492202 Jul 19 20:59 Model1_001.ps
-rw-rw-r--    1 glykos   glykos     492202 Jul 19 20:59 Model1_010.ps
-rw-rw-r--    1 glykos   glykos     269308 Jul 19 20:59 Model1_100.ps
-rw-rw-r--    1 glykos   glykos     492202 Jul 19 20:59 Model2_001.ps
-rw-rw-r--    1 glykos   glykos     492202 Jul 19 20:59 Model2_010.ps
-rw-rw-r--    1 glykos   glykos     269308 Jul 19 20:59 Model2_100.ps
-rw-rw-r--    1 glykos   glykos     492202 Jul 19 20:59 Model3_001.ps
-rw-rw-r--    1 glykos   glykos     492202 Jul 19 20:59 Model3_010.ps
-rw-rw-r--    1 glykos   glykos     269308 Jul 19 20:59 Model3_100.ps
-rw-rw-r--    1 glykos   glykos     150202 Jul 22 22:23 Qs001_mol_1.pdb
-rw-rw-r--    1 glykos   glykos      58345 Jul 22 22:23 Qs001_mol_2.pdb
-rw-rw-r--    1 glykos   glykos      54660 Jul 22 22:23 Qs001_mol_3.pdb
-rw-rw-r--    1 glykos   glykos    4205578 Jul 22 22:23 Qs001_packing.pdb
-rw-rw-r--    1 glykos   glykos     150202 Jul 25 03:17 Qs002_mol_1.pdb
-rw-rw-r--    1 glykos   glykos      58345 Jul 25 03:17 Qs002_mol_2.pdb
-rw-rw-r--    1 glykos   glykos      54660 Jul 25 03:17 Qs002_mol_3.pdb
-rw-rw-r--    1 glykos   glykos    4205578 Jul 25 03:17 Qs002_packing.pdb
-rw-rw-r--    1 glykos   glykos     150143 Jul 19 20:59 Qs_1.pdb
-rw-rw-r--    1 glykos   glykos      58286 Jul 19 20:59 Qs_2.pdb
-rw-rw-r--    1 glykos   glykos      54601 Jul 19 20:59 Qs_3.pdb
-rw-rw-r--    1 glykos   glykos       1953 Jul 19 20:59 Qs_auto.in
-rw-rw-r--    1 glykos   glykos     486034 Jul 19 20:59 Transform_Model1_001.ps
-rw-rw-r--    1 glykos   glykos     486034 Jul 19 20:59 Transform_Model1_010.ps
-rw-rw-r--    1 glykos   glykos     266752 Jul 19 20:59 Transform_Model1_100.ps
-rw-rw-r--    1 glykos   glykos     486034 Jul 19 20:59 Transform_Model2_001.ps
-rw-rw-r--    1 glykos   glykos     486034 Jul 19 20:59 Transform_Model2_010.ps
-rw-rw-r--    1 glykos   glykos     266752 Jul 19 20:59 Transform_Model2_100.ps
-rw-rw-r--    1 glykos   glykos     486034 Jul 19 20:59 Transform_Model3_001.ps
-rw-rw-r--    1 glykos   glykos     486034 Jul 19 20:59 Transform_Model3_010.ps
-rw-rw-r--    1 glykos   glykos     266752 Jul 19 20:59 Transform_Model3_100.ps
-rw-r--r--    1 glykos   glykos     927318 Jul 19 18:46 data.hkl
-rw-r--r--    1 glykos   glykos     172542 Jul 19 20:57 model1.pdb
-rw-r--r--    1 glykos   glykos      66975 Jul 19 20:57 model2.pdb
-rw-r--r--    1 glykos   glykos      62740 Jul 19 20:58 model3.pdb
```

The files `data.hkl` and `model.pdb` (or `model1.pdb`, `model2.pdb`, ..., in the case of the multi-model mode) is where you started from. File `Qs_auto.in` is the parameter file generated by QS when you choose the automatic mode (by giving `Qs -auto <xxx>`). The `LOG` file is exactly what you think it is. The `.ps` files are colour postscript files containing projections of the model unit cell and central sections through the molecular transform(s), and are completely useless from the structure determination point of view (they are discussed in more detail under the `POSTscript` keyword). The file `Qs.pdb` (or `Qs_1.pdb`, `Qs_2.pdb`, ..., in multi-model mode) contain your search model(s) after QS applied a translation and a rotation.

The real results are contained in the files of the form `Qs###_mol_#.pdb`. These contain the final atomic coordinates of your search model(s) from each of the independent minimisations performed by the program. They are minimal PDB files that should look like this :

```
REMARK  R-factor  0.0278  Free R-value -1.0000
CRYST1   38.070   33.200   46.120  90.00 110.06  90.00 P 1
ATOM      1  N   LYS     1       8.677  16.604  37.021  1.00 20.00
ATOM      2  CA  LYS     1       7.975  15.873  38.045  1.00 20.00
ATOM      3  C   LYS     1       8.898  14.827  38.643  1.00 20.00
ATOM      4  O   LYS     1       9.843  14.369  37.973  1.00 20.00
.....................................................................
ATOM    994  OXT LEU   129       2.565  -5.064  34.429  1.00 20.00
END
```

(the R_{free} value was not calculated in this case, which explains the `-1.00` value. When the `-auto` scheme is used, the R_{free} is calculated). If you had two molecules in the asymmetric unit, then for every `Qs###_mol_1.pdb` file, you would also be getting a `Qs###_mol_2.pdb` file as well (containing the coordinates of the second model). To save you some time, for every finished minimisation QS writes out a PDB file (of the form `Qs###_packing.pdb`) containing the coordinates of all molecules in the asymmetric unit plus their crystallographically related ones (all reseted to lie within a volume of 2×2×2 unit cells). With this file at hand, it is quite easy to check a putative solution for the presence (or otherwise) of bad contacts.

If your eyes are trained for side-by-side stereo viewing, then the fastest way to check for bad contacts is *via* RasMol : just do `rasmol Qs003_packing.pdb` and then type `set unitcell on` followed by `set stereo on` and `colour chain`. You should now have a coloured side-by-side stereo image of the contents of 2×2×2 unit cells corresponding to the 3rd minimisation.

NOTE WELL : If you decided to edit the script file and to define the crystallographic symmetry operators explicitly (through the `SYMMetry` keyword, QS will know nothing about the lattice type of your space group, and so, all copies of your model(s) that are related by non-primitive symmetry operators will be missing from the `Qs###_packing.pdb` files. A solution would be to define even the non-primitive symmetry operators with the `SYMMetry` keyword, but this would at least double the CPU time requirements.

8 Identifying promising solutions

If in a graph of the type *time vs. target function* you see a simultaneous sudden drop of both the target function and its free-value counterpart, you may be in business. Wait until the minimisation finishes, and examine the corresponding packing PDB file produced by QS. If this also looks good, go ahead and try some rigid-body refinement with your favorite program. For examples of what ”a simultaneous sudden drop” is, please refer to the QS-related papers (they are full of such graphs).

9 What if automation fails ?

I wouldn't be surprised.

Seriously now. If the fully automatic run fails to give a promising solution and you still have CPU time to waste, there is one thing that I would definitely recommend that you try. Because this is to use the *R*-factor as a target function, and because the *R*-factor is a politically incorrect word to use these days in crystallography, this section contains some hand-waving. But the doer's bit first :

Create a clean directory and copy there the files `data.hkl`, `model.pdb` (or `model1.pdb`, etc), *and the file* `Qs_auto.in` (which was produced by QS in the fully automated run). Go to this new directory, edit the file `Qs_auto.in`, and change the line saying : `TARGET            CORR-1`, so that it reads `TARGET            R-FACTOR` and multiply by 0.75 the starting temperature (the number shown next to `STARTING_TEMP`).

Save the modified `Qs_auto.in` file, and try to run QS interactively by typing `Qs Qs_auto.in` (you should not use the `-reso` or `-auto` flags in this case). If the program stops with a message about memory requirements, check that you have enough memory, and then give `Qs -force Qs_auto.in` to run it again. Once it reaches to the stage of actually starting the first minimisation, stop it with CTRL-C and submit a proper batch job with :

```
batch
/usr/local/bin/Qs -force Qs_auto.in > LOG
<CTRL-D>
```

Here comes the hand-waving bit.

The correlation-based targets have repeatedly been shown to perform much better than the *R*-factor, and are considered to be the next best thing after a maximum-likelihood target. Indeed, the linear correlation coefficient is a beautiful statistic to have and to share, but under certain circumstances it may be cursed with deep false minima that are not present with the good-old *R*-factor. How could that be ? It is not uncommon to find macromolecular crystals which give data sets with uneven distribution of intensity in their various reflection classes (some types of reflections are systematically strong, some other weak). Now, *any* structure that can reproduce this strong-weak pattern will end-up having a beautiful value of the linear-correlation coefficient. Unfortunately, the great majority of the possible arrangements that can reproduce these intensity patterns are wrong solutions. Let me give an example : lets say that your crystals contain two molecules per asymmetric unit which are related by a simple translation of 1/2,0,0 along the crystallographic axes. Then, all reflections with *h* even will be strong, and all reflection with *h* odd will be weak (and there will be a strong pseudo-origin peak in the native Patterson function at 1/2,0,0). As it happens, all trial structures for which these two molecules have the same orientation (*but any orientation*) and the correct position in the cell, will reproduce this strong-weak pattern, and will give you a very nice value of the target function. QS will enter this false minimum and may happily spend the rest of its time in there. The possibility that the two molecules will slowly and in concert turn around to find their correct orientation is an improbability. So, you got it : you are in deep nice-looking minimum, changing either the orientation or the position of any of the two molecules gives a much-much-worse value of the target function (because you loose the weak-strong pattern), and the scene is set for QS to waste your time and effort.

Enter *R*-factor : The *R*-factor is a precision indicator. It couldn't care less whether all strong reflections are predicted strong, and the weak reflections weak. What matters to the *R*-factor is how precise our individual predictions are. An example will convince you. Suppose that for six reflections we have F_os=$\{5, 15, 10, 1600, 1000, 2000\}$ and a trial structure that gives F_cs=$\{10, 2, 1, 3000, 700, 900\}$ (notice that the two data sets are practically on the same scale). The linear correlation coefficient between the two sets is 0.708, but the *R*-factor stands at a hefty 62.4% (somewhat worse than what we consider "random" for non-centrosymmetric space groups).

One last thing worth mentioning is why nobody else discussed this before (to my knowledge). I believe the answer is because previous authors didn't have the chance to see this behaviour : In traditional molecular replacement you never "see" (in your calculations) all molecules before you are actually finished with molecular replacement. Worse, in traditional molecular replacement calculations it is highly unlikely to have all your molecules placed correctly but oriented incorrectly. Traditional methods due to their divide-and-conquer approach miss some combinations of parameters that can even make the *R*-factor look better than the correlation coefficient. Quoting Aaron Levenstein, "*Statistics are like bikinis : what they reveal is suggestive, but what they conceal is vital*".

10 Beyond automation

Successive versions of this document contain decreasing amounts of documentation on the internal workings of the program. The reason for this inverse relationship is that I can see of no real reason duplicating the effort that I'm putting in writing the QS-related papers. Electronic reprints of these papers can be freely downloaded from my homepage and its mirrors (generously provided by CCP14). The following paragraphs have been retained because they contain some technical details not available from the papers.

It is impossible to discuss the input to the program without reference to how the program works. Table 1 outlines the basic steps that QS will follow under normal circumstances.

The essential ingredients of the reverse Monte Carlo minimisation are (i) that the next configuration to be tested is chosen randomly, and, (ii) that the probability of accepting a move against the gradient (ie. accepting a new configuration although it gives a higher R-factor) is in general non-zero (so that moves against the gradient are possible). As is always the case with this type of minimisation algorithms, the difficult points are : (i) to select the annealing protocol, ie to decide how the temperature[6] varies with respect to time, and, (ii) to select the displacement vector $\Delta\mathbf{x}$ which will take us from the current configuration $\mathbf{x}$ to the new configuration $\mathbf{x} + \Delta\mathbf{x}$. [7]

QS supports four different minimisation strategies. These different annealing protocols are not of cosmetic value : choosing one or the other might make the difference between solving the structure and wasting CPU time. Unfortunately, there are no hard and fast rules defining the best minimisation strategy, and this is one of the basic problems with QS. These three protocols are discussed below.

10.1 Constant temperature run

In this mode you define a temperature T which is kept constant for the whole length of the calculation. The maximum move sizes (how much the parameters that we will test in the next move may deviate from the current parameters) are also kept constant for the whole run and equal to : $\max(\Delta_\mathrm{t}) = 2d_{min}/\max(a, b, c)$, $\max(\Delta_\kappa) = 2d_{min}$, where d_{min} is the minimum Bragg spacing of the input data, a, b, c are the unit cell translations of the target structure (in Å), $\max(\Delta_\mathrm{t})$ is the maximum possible offset for any of the molecular translations (in fractional units) and $\max(\Delta_\kappa)$ is the maximum possible offset (in degrees) for the κ angle of any of the molecular orientations[8].

The constant temperature protocol can be quite efficient if you know how to choose a suitable temperature (so that the system is neither thermally disordered, nor as cold as to get stuck inside a shallow local minimum). I have recently started trusting the automatic temperature determination performed by QS, and so, an (automatically determined) constant temperature run is well-worth trying if the default runs fails to find a convincing solution.

10.2 The slow-cooling mode

In this mode you define a starting temperature T_0 and a finishing temperature T_f, with $T_0 > T_f$. At any time step t during the minimisation, the temperature T will be given by $T = T_0 - t(T_0 - T_f)/t_\mathrm{total}$, where t_total is the total number of time steps for the minimisation. In the slow cooling mode the

[6]No physical meaning should be attributed to the word 'temperature". It is only a control parameter which adjusts the probability of accepting a move against the gradient. The higher the temperature, the higher the probability of accepting a move that makes the R-factor worse.

[7]Practically speaking (and in the case of QS), $\Delta\mathbf{x}$ corresponds to the differences between the current set of orientation angles and positions of the molecules, and the set that will be tested for the next move.

[8] QS stores the molecular orientations in terms of the polar angles ω, ϕ, κ. A new molecular orientation is obtained from the previous one by rotating the molecule by κ degrees about an axis defined by the ω and ϕ angles. The orientation of the new rotation axis is always chosen randomly in the interval $0 \leq \omega \leq \pi/2$ and $0 \leq \phi < 2\pi$, ie the full half sphere. We obtain the new orientation by rotating about this axis by $\pm\xi\Delta_\kappa$ degrees, where $\xi, 0 \leq \xi \leq 1$ is a random number.

1. Enter 'Queen of Spades".
2. Read in data, coordinates, parameters.
3. Translate molecule to 0,0,0, and rotate it so that the axes of inertia are parallel to the orthogonal frame.
4. Generate the electron density map of the rotated/translated molecule in an orthogonal cell that is at least four times as big as the molecular dimensions.
5. Calculate FFT to obtain molecular transform. From now on, to calculate the contribution of any molecule to any reflection we only have to find the coordinates of the reflection in the (orthogonal) frame containing the transform and pick-up the values (real and imaginary) of the transform at that point. No more FFTs, but interpolation required.
6. Assign random orientations and translations to all molecules in the asymmetric unit.
7. Calculate and store in memory the contribution of each molecule in the asymmetric unit (and its symmetry-related ones) to each reflection. This allows us to make the CPU time per step independent of the number of molecules (At each step of the minimisation we only move one molecule, and, so, the contribution from all other molecules remains the same and we do not have to re-calculate them). Calculate initial *R*-factor, set starting temperature T.
8. Begin the basic iteration for a given number of time steps :
   - Choose a molecule, a rotation and a translation randomly. Apply the rotation and translation to the given molecule, and calculate the new *R*-factor.
     - If $R_{\text{new}} \leq R_{\text{old}}$, the move is accepted and saved.
     - If $R_{\text{new}} > R_{\text{old}}$, the move is accepted with probability $\exp(\Delta R/T)$, ie. if $\exp((R_{\text{old}} - R_{\text{new}})/T) > \xi$, where ξ is a random number between 0.0 and 1.0. If that's case, the move is accepted and saved, otherwise we return to the previous conformation.
   - The temperature is updated accordingly to a given protocol. The program supports : (i) a constant temperature simulation, (ii) a slow-cooling protocol with the current temperature linearly dependent on time, (iii) a Boltzmann annealing mode, and, (iv) a variable-temperature heating bath : at regular intervals the program calculates the average number of moves against the gradient that have been made in that interval. If this number is less than a preset value (which may happen when the system is inside a relatively deep local or global minimum), the temperature is slowly increased until the specified fraction of moves against the gradient is reached again. This guarantees that the system will wonder freely from minimum to minimum for the whole run (not even the global minimum can stop it). This method can only do the job because at each step the program examines the *R*-factor, and if it is the lowest one encountered so far, it saves the parameters of all molecules.
9. Using the parameters that gave the lowest *R*-factor, calculate orientations and positions of the molecules and write the .pdb files corresponding to each and every of the molecules in the asymmetric unit. Generate a .pdb file with all molecules in the unit cell (to easily check for bad contacts).
10. Exit 'Queen of Spades".

Table 1 : **Flow diagram for** Qs**.**

maximum move sizes (as defined above) are linearly dependent on both time and the current *R*-factor, with $\max(\Delta_{\text{t}}) = 0.5Rt/t_{\text{total}}$ and $\max(\Delta_{\kappa}) = \pi Rt/t_{\text{total}}$, where R is the current *R*-factor, t is the current time step and t_{total} is the total number of time steps for the minimisation. The dependence on R is justified on the grounds that as we approach a minimum of the target function, we should be sampling the configurational space on a finer grid[9]. The time dependence follows from a similar argument.

10.3 The Boltzmann annealing mode

This is the default mode for Qs (ie. when the `-auto` flag is defined). For this mode, the temperature T at time step t of the minimisation is given by $T = T_0/\log(t)$ where T_0 is the starting temperature for the simulation (can be determined automatically with the keyword `DETErmine_bath`). This mode implies a constant move-size control as described for the constant temperature run.

[9]The word "grid" is used here metaphorically. For all practical purposes the values returned by the random number generator in the 0.0–1.0 range are continuous (if the generator returns values in the range 0 to $2^{31} - 1$, then the 'grid size' on Δ_{κ} for example, is only 0.00000008 degrees)

10.4 The heating-bath mode

This is a primitive attempt to incorporate an automatic procedure for adjusting the temperature of the system. The essence of the method is to periodically adjust the temperature of the system in such a way as to keep the fraction of moves against the gradient constant and equal to a user-defined value. This means that as the system is closing-in towards a minimum of the R-factor (which will lead to a reduction of the moves made against the gradient) the temperature will be increased, and inversely, when the system is thermally disordered and wonders freely across the parameter space, the temperature will be decreased. In this mode the maximum move sizes can either be constant (as for the constant temperature run) with $\max(\Delta_\mathrm{t}) = d_{min}/\max(a, b, c)$ and $\max(\Delta_\kappa) = d_{min}$, or can be made linearly dependent on both time and the current R-factor, with $\max(\Delta_\mathrm{t}) = 0.5Rt/t_\mathrm{total}$ and $\max(\Delta_\kappa) = \pi Rt/t_\mathrm{total}$ (see keyword `CONStant`).

11 The keywords

11.1 Command line options

There are two ways to run the program. The first is `Qs -auto <number of molecules in a.u.>` which will start a fully automatic run (discussed on section 5, page 12), and the second is `Qs <input file name>` in which case the program will read the parameters from the defined file. In some cases you will have to give `Qs -force <input file name>` to tell QS to bypass some of the tests that it makes concerning the size of the array for FFT, or the molecular dimensions, etc.

Note that when QS is run in the automatic mode, it simply creates a parameter file with the name `Qs_auto.in` which is then treated as if supplied by the user. If this first attempt fails to find a convincing solution, you can modify this `Qs_auto.in` file to try some other annealing schedules.

11.2 Input file keywords

A typical input file would look like this :

```
############################################################
##                                                        ##
##                 Example script for Qs                  ##
##                                                        ##
############################################################

############################################################
#
# Target function definition and
# number of minimisations and steps.
#
TARGET          R-FACTOR
CYCLES          5
STEPS           10000000

############################################################
#
# Annealing schedule & move size control.
# This is for an automatically determined constant
# temperature run.
#
DETERMINE_BATH
CONSTANT
```

```
############################################################
#
# Reflection selection.
#
KEEP            0.70
AMPLIT_CUTOFF   1.0
SIGMA_CUTOFF    2.0
RESOLUTION      15.0 4.0
RANDOM_SELECT   1.0

############################################################
#
# Fraction of reflections for free set.
#
FREE            0.10

############################################################
#
# Files to use for reading model and data.
#
MODEL           model.pdb
DATA            data.hkl

############################################################
#
# Scales, grid, random number generator seed, etc.
#
GRAUTO
SCMODE          wilson
INTERPOLATION   linear
SEED            47579
GLOBAL_B        20.00

############################################################
#
# Log file and postscript-related things.
#
INFO            1000
POSTSCRIPT      colour

############################################################
#
# Finally, cell, space group and molecules per a.u.
#
CELL              27.23   63.66   59.12  90.000  92.900  90.000
GROUP           4
MOLECULES       2
#
#
############################################################
```

The characters `#`, `!` and space, when they occur at the beginning of a line inform QS to ignore the rest of the line. A complete list of keywords understood by the program follows.

11.2.1 *GROUP 1 : Target structure related keywords*

`MOLEcules` *n* : Where *n* is the (integer) number of search models in the crystallographic asymmetric unit of the target structure.

`MODEl <string>` : Where `<string>` is a valid file name corresponding to the file that contains the atomic coordinates of the search model in the single-model mode. For the multi-model mode, see below. The form of this PDB file was discussed in section 5. Note that when you run the program without the `-auto` flag, the unit cell dimensions and space group number must be defined in the input file (the `CRYST` card in the PDB file will be ignored).

`MOD`*N* `<string>` : Where `<string>` is a valid file name corresponding to the file that contains the atomic coordinates of the *N*th search model in the multi-model mode (for example, `MOD1 DNA.pdb` will instruct the program to read the coordinates of the first model from a PDB file named `DNA.pdb`). The form of these PDB file was discussed in section 5. Note that when you run the program without the `-auto` flag, the unit cell dimensions and space group number must be defined in the input file (the `CRYST` card in the PDB files will be ignored). Please also note that the keywords `MOD`*N* and `MODEl` are mutually exclusive.

`DATA <string>` : Where `<string>` is a valid file name pointing to the ASCII file that contains the crystallographic data $(h, k, l, F, \sigma(F))$ for the unknown structure.

`CELL` $a\ b\ c\ \alpha\ \beta\ \gamma$: Unit cell dimensions for the target (unknown) structure. All six (floating point) parameters must be present.

`GROUp` *n* : Where *n* is the space group number of the target structure. QS does not understand space group symbols.

`SYMM` : This is an alternative to the `GROUp` command. The `SYMM` keyword should be followed by three lines containing the rotational and translational part of a valid symmetry operator as shown below for the operator $1/2 - x, 1/2 + y, \overline{z}$

```
SYMM
 -1.0  0.0  0.0       0.50
  0.0  1.0  0.0       0.50
  0.0  0.0 -1.0       0.00
```

The identity operator is defined internally and should *not* be given. QS does not understand fractions, so translational parts like 1/6 must be converted to floats with a sufficient number of decimal digits after the period (`0.33` will be converted by QS to 0.3300 which for most problems is a bad approximation to 1/3. You are better off giving something like `0.33333333`).

11.2.2 *GROUP 2 : Time related keywords*

`SLEEp` *n* : This is a very polite keyword that will take your system administrator off your back. You can use `SLEEp` to make QS run only for a specified time interval of the day (usually overnight). This you do as follows : suppose that most users go home at about 1900 hours and come back in again at 0830 hours next morning. If you submit your job using the unix command `at 2000` and you give `SLEEP 12` in your input file, QS will run for $24 - n = 12$ hours (from 8:00 in the evening until 8:00 next morning) and then will go to sleep for $n = 12$ hours (till 20:00 in the evening). Note that the time at which the job is started defines QS's sense of time (so it is critical that the job gets started at the right time). VMS users must use the command line option `/AFTER=...` in their `submit` command (but have a look at the help page on submit/after because there are some timing peculiarities with VAX clusters).

`CYCLes` *n* : Where *n* is the total number of minimisations to be performed (integer). You definitely need more than just a couple of runs.

`STEPs` *n* : Where *n* is the total number of moves for each minimisation (integer). For a medium-sized protein, in a low symmetry space group and with just one molecule per asymmetric unit, 5 million moves per minimisation might as well do the job.

11.2.3 *GROUP 3 : Annealing schedule and move size control*

For a constant temperature run with a user-defined temperature you need the keywords `CONStant` and `STARting_t`.

For a constant temperature run with an automatic determination of the temperature you need the keywords `CONStant` and `DETErmine_bath`.

For a Boltzmann annealing run with a user-defined starting temperature you need the keywords `STARting_t` and `BOLTzmann`.

For a Boltzmann annealing run with an automatic determination of the temperature you need the keywords `DETErmine_bath` and `BOLTzmann`.

For a slow-cooling run with a user-defined temperature range, you need `STARting_t` and `FINAl_t`.

For a slow-cooling run with an automatic determination of the temperature range define `DETErmine_bath`.

For a heating-bath run with user-defined temperature limits, you need `AUTO`, `STARting_t`, `FINAl_t`, `DELTatemp` and `UPDAte`.

For a heating-bath run with user-defined temperature limits and constant move size, you need `AUTO`, `CONStant`, `STARting_t`, `FINAl_t`, `DELTatemp` and `UPDAte`.

For a heating-bath run with an automatic determination of the temperature limits define `AUTO`, `DETErmine_bath`, `DELTatemp` and `UPDAte`.

For a heating-bath run with an automatic determination of the temperature limits and constant move size, define `AUTO`, `CONStant`, `DETErmine_bath`, `DELTatemp` and `UPDAte`.

`STARting_temp` *T* : Depending on the annealing mode, *T* is either the temperature for the simulation (constant temperature run), the initial temperature for a slow-cooling run, or the upper temperature limit for a heating-bath simulation.

`FINAl_temp` *T* : Depending on the annealing mode, *T* is either the final temperature for a slow-cooling run, or the lower temperature limit for a heating-bath simulation.

`DETErmine_bath` n : This flag starts n identical copies of a procedure aiming to determine a suitable temperature (or temperature range) for all four annealing modes. This we do as follows : the temperature is set to a sufficiently remote (very high) value[10] of $T_0 = 0.6250$. The system is equilibrated at this temperature for Δt=10000 time steps and the mean value of the resulting R-factors (say, $\bar{R}_0$) is saved. Then the temperature is set to $T_1 = T_0/2 = 0.31250$ and the procedure repeated up to and including $T_{13} = 0.0000763$ which will give us $\bar{R}_{13}$. For every pair of average R-factors $\bar{R}_n$, $\bar{R}_{n+1}$, we calculate $\Delta\bar{R}_n = \bar{R}_n - \bar{R}_{n+1}$ and then we do a cubic spline interpolation on the resulting $\Delta\bar{R}_n$'s and from that we determine the temperature T_0 for which $\Delta\bar{R}$ is maximum[11]. A graphical representation of the results from this calculation are written in the file `temp.ps` (a postscript file, only produced when the keyword `POSTSCRIPT` is on).

`CONStant` : (No arguments) Depending on the presence or absence of the `AUTO` keyword, this flag either sets the annealing mode to a constant temperature run, or —in the presence of the `AUTO` keyword— sets the maximum move sizes to a constant value (as described in section 10).

`BOLTzmann` : (No arguments) This flag sets the annealing mode to the Boltzmann annealing mode.

`AUTO` f : This keyword, together with the `DELTa_temp` and `UPDAte` keywords define a heating-bath simulation as described on page 24. The argument f is the fraction of moves made against the gradient that the program will aim for (floating point).

`UPDAte` n : Where n is interval (in time steps) over which the average number of moves made against the gradient will be calculated. Based on the result of this calculation (and the value defined in the `AUTO` keyword), the temperature will be increased or decreased by the value defined in the `DELTa_temp` keyword. A value in the range 2000–5000 will be appropriate for most problems.

`DELTa_temp` ΔT : Where ΔT is the increment or decrement applied to the current temperature (every `UPDAte` steps) during a heating-bath simulation. If too big a value is given, the system will abruptly jump from hot ($T_{steady} + \Delta T$) to cold ($T_{steady} - \Delta T$) every `UPDAte` time steps. If too small a value is given, the system will be approaching the equilibrium state too slowly. A value of about 0.00020 units should do for most problems.

`MAXKappa` f : This keyword allows you to explicitly define the maximum possible offset for the κ angle ($\max(\Delta_\kappa)$ as discussed on page 22). It will not have any effect if you have also given the keyword `CONStant` (in which case, $\max(\Delta_\kappa) = d_{min}$ as discussed in 10.1).

`MAXFraction` f : As for the keyword `MAXKappa` but defining the maximum possible translational offset ($\max(\Delta_t)$ on page 22).

[10]The probability of accepting a move against the gradient is $\exp(-|\Delta R|/T)$. Now, $|\Delta R|$ is the absolute value of the difference between two values of the conventional R factor and, so, takes values of the order of 10^{-1}. For, say, $T = 10$, the probability of accepting a move against the gradient is 0.990, ie practically all moves are accepted and so the system is thermally disordered.

[11]The quantity $\Delta\bar{R}/\Delta T$ is analogous to the specific heat from statistical mechanics. A large value of $\Delta\bar{R}/\Delta T$ signifies that a "phase" transition is occurring at the corresponding temperature, and so the cooling rate should be as low as possible at this temperature range.

11.2.4 *GROUP 4 : Reflection selection*

`RESOlution` *low high* : Low and high resolution cutoff (two floating point numbers). Default 500.0–1.0Å.

`SIGMa_cutoff` cut_σ : All reflections with $F/\sigma(F) < cut_\sigma$ will be excluded from the minimisation. Default 0.0.

`AMPLitude_cutoff` cut_{ampl} : All reflections with $F < cut_{ampl}$ will be excluded from the minimisation. Default 1.0.

`KEEP` f : This is another way to select a set of reflections for the minimisation. Instead of trying different amplitude cutoffs (until you have as many reflections as you want), you can ask QS to use the strongest $(100 \cdot f)\%$ of the input data for the simulation. Note that because this is essentially an amplitude (and not an $|E|$-value) cutoff, you may end-up with a lower effective resolution for your minimisation (which may not be a bad idea, especially if you have indeed measured your low resolution data). This is now the default for the automatic mode (with `KEEP 0.50`).

`RANDom_select` f : From the set of reflections that satisfy all other selection criteria, chose randomly only a fraction f for the minimisation. Although this might sound like an unbiased way to reduce the number of reflections for the minimisation, in practice it didn't prove much of a help. Default 1.0 (ie use all selected reflections).

`FREE` f : From the set of reflections that satisfy all selection criteria, chose randomly a fraction f and use it as a test set (ie for the calculation of the free R value). Because the number of reflections used for the minimisation has to be relatively small (to make the program practical), the free R value will be calculated over a very small number of reflections. This has two consequences : (i) the standard uncertainty (due to statistical fluctuations) of the free R value is large, and, (ii) because the number of reflections is small, we can not independently scale F_os to F_cs for the test set (which may bias the R_{free} ?). The take-home message is that the free R values produced by QS should not be taken at face value.

`USEValues` E_{limit} : This keyword instructs QS to calculate the normalised structure factor amplitudes corresponding to input Fs and —in addition to all other selection criteria— to select only those reflections for which $E_o > E_{limit}$. QS will also write out an ASCII file (with the name `Qs_Evalues.hkl`) containing $h, k, l, F, \sigma(F), E$ where E is the normalised structure factor amplitude corresponding to F. I should add that I still marvel at how I managed to make the E-value calculation so slow.

`RESFree` f : Reflections at a resolution *lower* than f will not be allowed to enter the free R-factor set (ie, all low resolution reflections will belong to the working set during minimisation). The reason for the presence of this keyword is that in some cases (see structures with strong periodicities, for example 4-α-bundles), there may be a few low resolution reflections of outstanding intensity. The presence of these reflections in the working set can significantly reduce the parameter space that we have to explore (because only a small part of the orientations and translations can account for these outstandingly strong reflections). For this reason, we would not want to risk "loosing" these reflections in the test set, and so here comes `RESF`. Please feel free to consider this as an outrageous R-free biasing keyword (and dangerous practice, etc.).

11.2.5 *GROUP 5 : Internal calculus related keywords*

`TARGet <R-FACTOR,CORR-1,CORR-2> :` Release 0.1 of QS only supported one target function, namely the *R*-factor. Following a suggestion by one of the referees of the QS-related paper, support has been added for two additional target functions (other than the *R*-factor). These are $1.0 - \mathrm{Corr}(F_o, F_c)$ and $1.0 - \mathrm{Corr}(F_o^2, F_c^2)$ where $\mathrm{Corr}()$ is the linear correlation coefficient between the respective sets. The target function for the minimisation is chosen with the `TARGet` keyword, with the meaning of the three possible arguments being obvious.

`SCALE` *s* `:` This keyword defines how finely the molecular transform will be sampled : If p, q, r are the dimensions (in Å) of the smallest orthogonal box that can fully enclose the search model, then QS will calculate the molecular transform by placing the model in an orthogonal cell with dimensions sp, sq, sr, followed by Fourier transformation of the corresponding electron density distribution. The higher the value of *s*, the finer the transform will be sampled, the more physical (computer) memory you need for storing the transform. On the other hand, the finer the sampling the higher the accuracy of the calculated F_c values. I think that you should aim for a value of `SCALE` of at least 4.0. If you have enough physical memory it may be a worth-while exercise to try to increase `SCALE` to about 8.0 and to give `INTErpolation_scheme NONE` (see below). The keyword `GRAUto` will attempt to automatically determine suitable values for both `SCALE` and `MAXGridspacing`, aiming for a value of `SCALE` of 4.0.

`MAXGridspacing` *d* `:` This keyword determines the extend (highest resolution) of the transform. It corresponds to the maximum acceptable (by the user) separation (in Å) between successive grid points on any of the directions of the cell that will be used for calculating the molecular transform. The value of *d* should be at least one-third of the maximum resolution of the data that you intend to use. In addition, you should not use too coarse a grid because serious errors will be introduced during the calculation of the electron density map. Values of *d* of the order of up to 1.0Å are possibly OK.

`GRAUto :` (No arguments) This flag instructs QS to attempt to determine the parameters for the electron density and molecular transform calculations automatically. QS will open the file containing the input reflections (`DATA` keyword) and will determine the maximum resolution of the data present in this file. It will then set the maximum acceptable grid spacing to 1/4th of data resolution (keyword `MAXGridspacing` above) and will try to determine a value for `SCALE` assuming that at least 48 MBytes of physical memory can be safely allocated and that the minimum acceptable value for `SCALE` is 4.0. The values determined by `GRAUto` are not necessarily optimal for your environment, so beware.

`INTErpolation_scheme <Linear,None,Polynomial> :` This keyword defines the molecular transform interpolation scheme used by QS. The program supports three modes : a polynomial 4th order interpolation mode, a linear interpolation mode, and a nearest grid point non-interpolating mode. Polynomial interpolation is the most accurate of the three but it slows down the program by a factor of two and the gain in accuracy is not sufficient to justify using it as the default. The nearest grid point mode is the fastest and least accurate. If you are sampling the molecular transform finely enough (with, say, `SCALE 8.0` or higher) you can use `INTErpolation_scheme NONE` safely. Having said these, I would suggest that you stick to `INTErpolation_scheme LINEAR` which is both fast and accurate for values of `SCALE` down to 4.0. Default is `LINEAR`.

`SCMOde <Sum,Wilson> :` This keyword defines the mode that QS will use for scaling F_c's to F_o's after each Monte Carlo step. The `sum` option defines a crude but fast scaling based on the quantity $\sum F_c / \sum F_o$, whereas `wilson` defines a Wilson-like scaling mode. Because of the sensitivity of the Monte Carlo algorithm to even small differences between the resulting *R* factors, I think that you should always use `SCMOde Wilson`.

`GLOBal_b` $B_{overall}$: where $B_{overall}$ is the global (isotropic) temperature factor that QS assigns to all atoms of the search model in the single-model mode. This only affects the electron density and molecular transform calculation, because after each Monte Carlo step the F_c's are scaled to F_o's using an overall scale and temperature factor. A value of about 20 to 30Å^2 would be OK for most protein work. Default 20.0Å^2.

`B_M`*N* $B_{overall}$: where $B_{overall}$ is the global (isotropic) temperature factor that QS assigns to all atoms of the *N*th search model (for example, `B_M1 20.0` will assign a temperature factor of 20.0Å^2 to all atoms of model number 1 (corresponding to the file defined by `MOD1`, see page 26)). This only affects the electron density and molecular transform calculation, because after each Monte Carlo step the F_c's are scaled to F_o's using an overall scale and temperature factor. A value of about 20 to 30Å^2 would be OK for most protein work. Default 20.0Å^2.

`BULK` <k_{sol} B_{sol}> : This is an attempt to incorporate a primitive bulk-solvent correction algorithm in QS. The idea is to use the exponential scaling model to down-scale the amplitudes of the low resolution data. Unfortunately, it is impossible to do a proper job since that would entail several rounds of non-linear scale refinement at each step (see review by Tronrud, D. E. (1997), *Methods Enzymol.*, **277**, 306–319.). So, what we do instead is to keep the values of the two solvent parameters (k_{sol}, B_{sol}) constant and equal to either the default values (when no argument is given next to the `BULK` keyword), or to the values given by the user. Now, about the values : the usual assumption is that k_{sol} is equal to the ratio of the mean electron densities of the protein and solvent components, and B_{sol} is related to the sharpness of the boundary between them. Unfortunately, things are not that simple : when I checked the PDB-deposited structures with respect to the distribution of the reported values for these two parameters the results were rather disappointing (you can see the distribution obtained from a paper abstract at my homepage `http://origin.imbb.forth.gr:8888/~glykos/`). I believe that down-scaling the low resolution terms with the default values can not make things worse than they are without any correction, so you may as well decide to include this keyword right from the beginning (but note that because it has not been extensively tested it is not the default).

11.2.6 *GROUP 6 : Miscellaneous keywords*

`PACKall` : When this keyword is present QS will write out `Qs###_packing.pdb` files that contain all atoms of the search model(s) instead of just the C_α atoms. Take care : QS now writes the contents of 8 unit cells in these files, so they can grow quite big (especially in high symmetry space groups).

`SEED` *n* : Initialise the random number generator using the (integer) argument *n* as a seed. I suggest that you stick to your phone number (unless you are prepared to take a note of the number, or, you are certain that you won't have to reproduce a run).

`INFO` *n* : Write out information about the minimisation every *n* time steps. This includes the current time step, the average *R*-factor for the last interval, the corresponding free *R* value (if calculated) and the current temperature. No need to use anything below, say, 1000 time steps.

`POSTscript <Monochrome,Colour> :` If present, QS will write out three postscript files (monochrome or colour, depending on the argument) containing the projections of the electron density of the orthogonal unit cell that will be used for calculating the molecular transform (files `001.ps, 010.ps, 100.ps`). After the transform calculation is finished, three more postscript files will be produced containing the modulus of the transform on the three orthogonal central sections that correspond to the electron density projections (files `mod_001.ps, mod_010.ps, mod_100.ps`). Useless (but the transforms are mystically beautiful).

`LATTice :` (No arguments) This flag instructs QS to plot the positions of the reflections that are located on the three orthogonal central sections of the transform on the corresponding postscript files (ie, the files `mod_001.ps, mod_010.ps, mod_100.ps`). This is neither useful, nor beautiful.

`NOISe` f `:` (for testing purposes) A random offset will be applied to the input F_os. The offset will range uniformly from $-fF_o$ to $+fF_o$.

NOTE WELL

QS will recognise the various keywords using only the first four characters starting from the first character on each line (No spaces or tabs before the keyword). You can append as many characters as you want to these four as long as you still have *one* string of characters (ie use _ instead of space, no tabs or other 'white' characters allowed). QS will ignore anything that follows the keyword and its arguments on a line of input. This means that you can freely append comments on each line (but if you want to have a line devoted to comments, then the first character of that line should be `#`, `!` or space).

VMS users : The keywords must be typed using capital letters.

12 Space group things

QS contains (hard-coded) the primitive symmetry operators corresponding to all 230 space groups as given in the International Tables. For those space groups for which the Tables give more than one (alternative) settings, only one is supported by the program. I have tried to stick to the most common ones (eg $C2$ instead of $B2$, hexagonal instead of rhombohedral axes, etc), but this doesn't mean that you should re-index your data ignoring the good reasons that made you chose a different setting. Instead, you can keep your favorite setting and define the corresponding symmetry operators explicitly using the `SYMMetry` keyword as described on page 26 (do not forget to remove the `GROUP` keyword from your new input file). Note that you can safely ignore all operators arising from the presence of a non-primitive lattice. Please also note that QS will not make any tests on the input matrices (not even a determinant calculation), so you have to type them carefully.

13 Bugs

None. It must be your hardware.