Notes on the package *Hyper-sweep*

Introduction

The *Hyper-sweep* software library is a set of Fortran routines for parallelized search of a n-dimensional hyper cube or hyper sphere. It includes an option to perform numerical integration of user supplied functions through parallelized Monte Carlo integration. In all cases the search and integration operations can be nested over multiple grids which are re-positioned and refined according to prescribed criteria (See Figure 1). Parallelization is invisible to the user and can be performed over an arbitrary number of cores in a cluster with approximately linear scaling in performance. The software is a user callable source library. Included in the packages are some example driver programs which can be modified or written in other languages. Overall computation cost scales with the time required to evaluate the user supplied optimisation (or integration) function; the number of dimensions, and the grid density of the hyper-cube, or number of samples in the hyper-sphere.

The algorithms described in these routines will be inefficient as the number of dimensions grows due to the *Curse of dimensionality*. They are not expected to be practical for search problems in more than 7 or 8 dimensions. The nested refinement scheme is also quite simple and may miss local or global minima if the grids not sufficiently dense. The overall computation time will scale linearly with the time required to evaluate the user supplied objective function (see Figure 6).

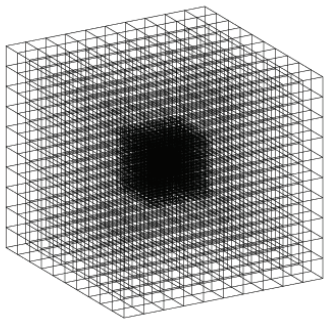

Figure 1: An illustration of a nested grid in three dimensions. The program *pgs_mpi* visits each point in the lattice and finds either the maximum or minimum of a user supplied function. The workload is distributed approximately evenly among an arbitrary number of cores in a cluster.

Programs

The package consists of four separate programs:

- **pgs_mpi** - Program for exhaustive grid search through an n-dimensional hyper-cube to find extrema of a user supplied function. Options are included for i) repeated search over a user specified number of nested grids centred on the extremum of the previous grid, and ii) Monte Carlo integration of a separate user supplied function over the same grids.

  Driver source code: *PGSex.F90*. Libraries used: *hyper-g.F90* and *hilbert_lib.c*.

  Compilation: > mpif90 *PGSex.F90* -DMPI=1 *hyper-g.F90* *hilbert_lib.o* -o ./bin/pgs_mpi

  Output: Results to standard out. Files `planes.dat` - data for 2-D contour plots of planar slices through optimum point found (see **plotting** section below); `hgrid-details.dat` - contains details of nested grids produced during search.

- **pgs_serial** - Same as *pgs_mpi* without MPI calls. Used to test or run code on a single processor. Output should be identical to *pgs_mpi* for search and statistically identical for numerical integration.

  Driver source code: *PGSex.F90*. Libraries used: *hyper-g.F90* and *hilbert_lib.c*.

  Compilation: > gfortran *PGSex.F90* *hyper-g.F90* *hilbert_lib.o* -o ./bin/pgs_serial

- **pss_mpi** - Program for randomised search through an n-dimensional hyper-sphere to find extrema of a user supplied function. Options are included for i) repeated search over a user specified number of nested spheres centred on the extremum of the previous sphere and ii) either uniform random or Gaussian distributed random sampling within the hyper sphere (see Figure 3.) 4.

  Driver source code: *PSSex.F90*. Libraries used: *hyper-s.F90*.

  Compilation: > mpif90 *PSSex.F90* -DMPI=1 *hyper-s.F90* -o ./bin/pss_mpi

- **pss_serial** - Same as *pss_mpi* without MPI calls. Used to test or run code on a single processor. Output should be statistically identical to *pss_mpi*.

- **PGStime** - Utility program to perform some timing calculations and estimate the length of time for program *pgs_mpi* or *pgs_serial* to complete. Output information written to standard out.

  Compilation: > gfortran *PGStime.F90* -o ./bin/pgstime

  Run with: > *./bin/pgstime* < pgstime.in

- **clot-p** - A plot program to plot contour values of the user supplied objective function for planar slices through the optimum solution found by the grid search programs *pgs_mpi* or *pgs_serial*. This program reads in ASCII information in files `planes.dat` and writes a postscript file.

  Compilation: > cd plot; make all

  Run with: > cd plot; *./cplot-p* < planes.dat (See section **plotting** below.)

The codes *pgs_mpi* and *pgs_serial* make use of the Hilbert curve C library of John Skilling (*Skilling*, 2004) (see also *Lawder*, 2000). The serial versions are supplied for convenience. They produce the same results as the MPI versions when random perturbations to grid points are not included. MPI must be installed on your system and accessible through an MPI compiler such as *mpif90*. The coded has been developed and tested using openMPI.

Installation

To compile the entire package:

- Edit the file pgs-compile-options to suit your platform. This file contains information about where certain programs reside on your machine. Templates for several systems are in the subdirectory 'compile'. If a suitable template exists you could

  > cp compile/pgs-compile-options_*your-platform* pgs-compile-options

  and then edit pgs-compile-options if need be.

- to compile and install the whole package type

  > make all.

  To remove executables type

  > make clean

If all goes well you will get a message confirming that installation was successful and the four programs will appear in ./bin. Plot programs are compiled separately with '> cd plot; make all' which produces an executable *cplot-p* in subdirectory plot.

Running examples

The default search problem is to find the minimum of a quadratic log-Likelihood objective function centred on $\mathbf{x} = 0$. This is implemented in the example driver program *PGSex.F90* by calling a user-supplied subroutine `objval`. The default integration problem is to integrate the exponent of the quadratic log-Likelihood objective function centred on $\mathbf{x} = 0$. This is implemented in the example user-supplied subroutine `Integrand` in *PGSex.F90*. (See section **integration** below for more details.) All example programs may be run from the main directory. Some ways to run the default example would be

> ./bin/pgs_serial

to run the serial version for the default example in PGSex.F90. This puts simple output to the screen.

> mpirun -np 7 ./bin/pgs_mpi

to run the parallel version over 7 processes for the default example in PGSex.F90 and output info to the screen.

> ./bin/pgstime < pgstime.in

to run the utility program pgstime. This evaluates expected run time of a grid search with parameters in the file `pgstime.in` and writes result to std out. See `pgstime.in` for details. Similar example problem is implemented in *PSSex.F90* which can be edited.

Running your own grid search problems

To change the examples you should edit the parameters in the driver programs PGSex.F90 or PSSex.F90 and supply subroutines for evaluation of the objective function or integrand as described below. Specific user adjustable control parameters for program PGSex.F90 are as follows:

```
n           :  number of dimensions of hypercube lattice.
b           :  number of bits in discretization of each axis.
xo          :  Vector.  Co-ordinates of grid origin (see Figure 2).
xlen        :  Vector.  Length of each axis (see Figure 2).
nest        :  number of nested loops to be performed.
sfactor     :  shrink factor for size of grid.
            :  If factor < 0.0 then nested grid is centred on the
            :  maximum of the previous grid, otherwise the minimum.
random      :  Logical.  If true then a uniform random perturbation is
            :  added to the grid points within the interval ±dx/2.
integrate   :  Logical.  If true then MC integration is performed.
```

Note that the hyper cube is divided into a series of 2^b points along each axis, giving a total of 2^{nb} points in the n dimensional lattice. The unit interval is `dx` $=$ `xlen`$/2^b$. The user can edit the subroutine `objval` in *PGSex.F90* to insert a different objective function for optimisation. The user can edit the subroutine `integrand` in *PGSex.F90* to insert a different function for integration. For further details on the nature of the nested grids and numerical integration see below.

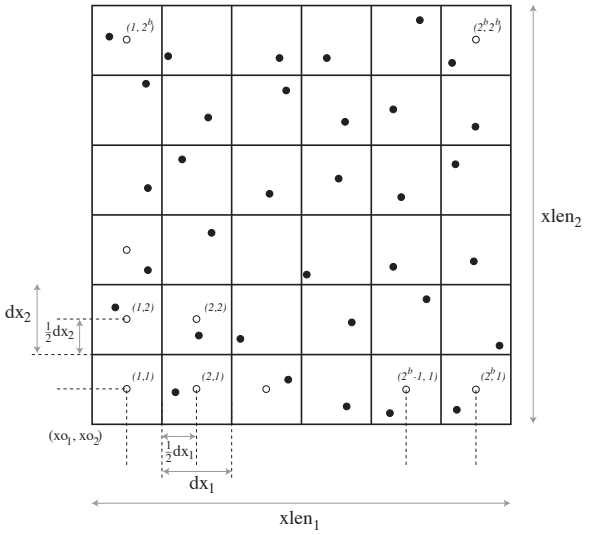

Figure 2: Definition of the search grid in terms of the parameters `xo`, `xlen` and `b`. White nodes show the lattice points for a 2-D grid with 2^b points in each direction. Black nodes represent a uniform random perturbation of the grid position within the unit box of side dx. b controls the total number of nodes in each nest and is kept constant. For each new nested grid the parameters `xo` and `xlen` are updated to centre the new grid on the optimum value from the previous grid. Grid lengths `xlen` are reduced by a factor `sfactor` for each successive grid.

Method: parallelizing grid search through the Hilbert curve

The purpose of the grid search is to evaluate the objective function at each point on the lattice (white or black nodes in Figure 2.

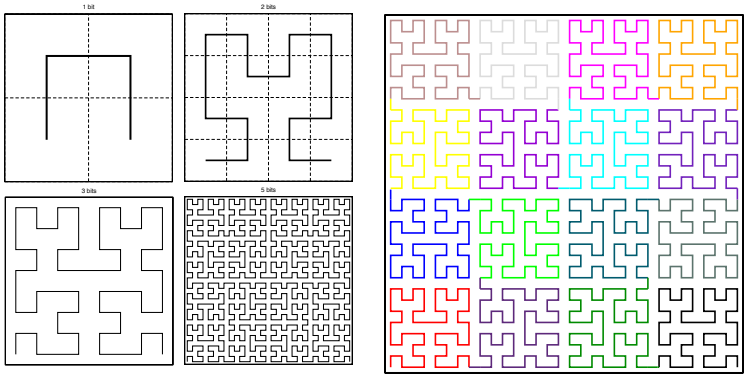

Figure 3: Left. Hilbert curves for bit values $b = 1, 2, 3$ and 5. The Hilbert curve is a recursively defined path through an n-dimensional grid where each side has 2^b lattice points. Right. Distribution of the $b = 5$ Hilbert curve across 16 processors. The position of each grid node on the Hilbert curve is used to determine which processor it gets mapped to.

The Hilbert curve provides a convenient unique mapping of all grid points of an n-dimensional hyper-cube onto a 1-D line. Mapping onto a single index allows convenient distribution onto an arbitrary number of processors. The Hilbert curve is the most compact mapping of this type, in that points which are close along the Hilbert curve are, on average, close in Euclidean space in the n-dimensional grid. This feature can be seen through the distribution of colours in Figure 3 which are consecutively plotted along the Hilbert curve. Use of the Hilbert curve makes it straightforward to distribute the computation across an arbitrary number of processors, with the potentially useful property that all grid nodes in the same processor are in close proximity in Euclidean space.

Random sampling of the hyper-sphere

The program *PSSex.F90* which produces the executables *pss_mpi* and *pss_serial* performs randomised sampling of a sphere with a user determined radius, either in parallel or serial. In this case parallelization is straightforward because the random deviates are independent and can be generated equally on all processors and then results combined. This can also be nested any number of times with the centre of the next hyper-sphere on the previous maximum or minimum.

The user supplies routine *objsphere* which evaluates an objective function for minimisation or maximisation. Parameters that may be set and adjusted are similar to *PGSex.F90* and include `n, xo, ntot, nest, factor`. Here `xo` is the centre of the sphere of radius given by `radius`. `ntot` is the total number of points to draw. In the example program a routine (*Nsph*) is provided which calculates the number of random points inside the N-sphere with the same density as the enclosing hyper-sphere on n dimensions with axis discretization 2^b. By setting `ntot` in this way ensures that the hypersphere and hyper-cube have the same average density of points, although there will be far fewer points in the sphere due to the much smaller volume[1] .

[1]The ratio of volumes of the hyper-sphere to the hyper-cube is $\frac{2\pi^{n/2}}{2^n\Gamma(n/2+1)}$, which tends toward zero as n increases.

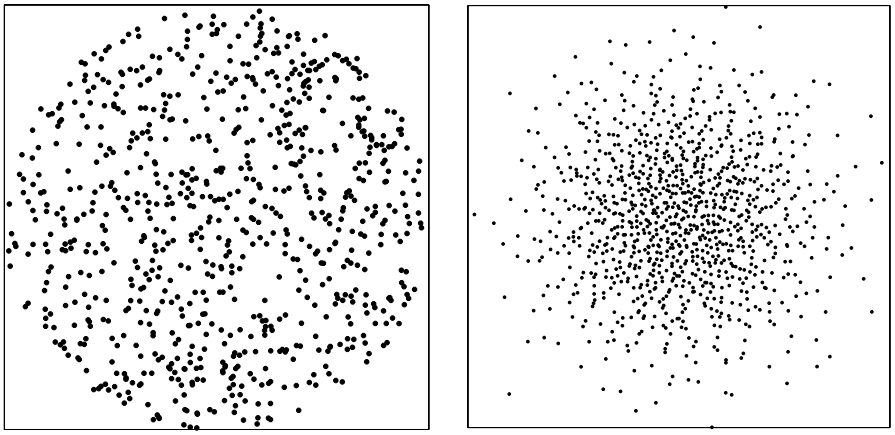

Figure 4: Two types of randomised sampling of the hypersphere produced by the program PSSex.F90. On the left are uniform random draws within the hyper-sphere of dimension 2 (a circle) and on the right are draws from a Normal distribution. The normal distribution is isotropic but more concentrated near the centre. The density ratio of sampling at the centre of the sphere increases significantly with dimension.

Running time of hyper-sweep

If n_p is the number of processors used for calculation and $\bar{t}$ is the average compute time to evaluate the user supplied function `objval` then the parallelized grid search compute time will be

$$\tilde{T}_{n_p} = \bar{t}\,\frac{2^{bn}}{n_p} \tag{1}$$

which indicates a uniform increase in speed with n_p. Figure 5 shows results of computations of running time of *pgs_mpi* as a function of the number of processors, n_p. In this figure the y-axis is the speed up factor defined as $\frac{\tilde{T}_{n_p}}{\tilde{T}_1}$, where both terms are measured from experiments. These experiments were performed for a 6 dimensional grid search with $b = 6$, i.e. 64 points per axis, giving a total of 68×10^9 grid points.

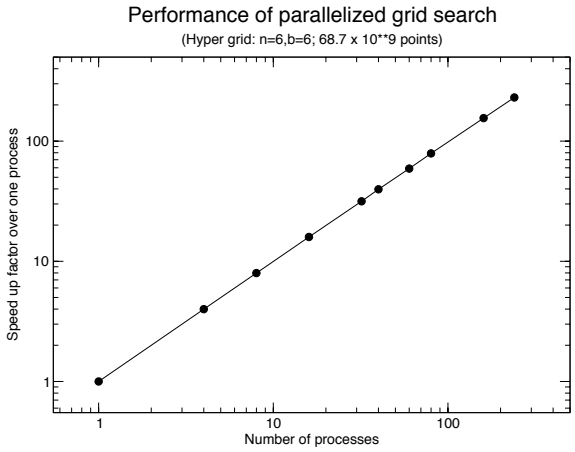

Figure 5: Scaling of compute times for parallelized grid search as a function of number of processors. The y-axis is the measured ratio of the time taken for n_p processors to the time taken on a single processor for the same enumeration problem.

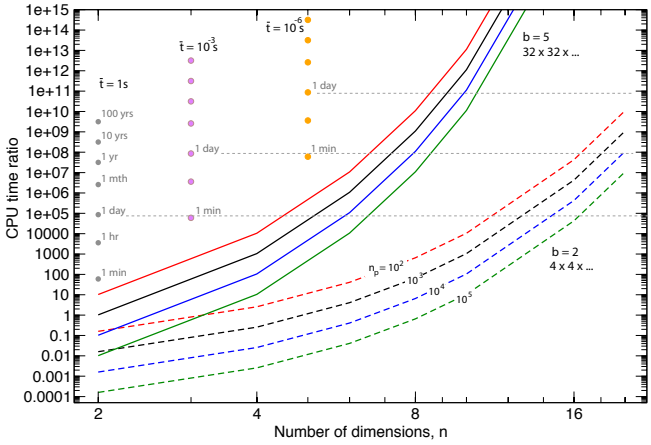

Figure 6: Scaling of CPU time of *pgs_mpi* with grid size, b, number of dimensions, n, and cost of objective function evaluation, $\bar{t}$. The y-axis is the ratio of total CPU time to $\bar{t}$, the cost of a single evaluation of `objval`. Coloured lines show calculations for four different numbers of processors, n_p. Coloured dots show how CPU time would scale for three different values of $\bar{t}$, namely $\bar{t} = 1s, 10^{-3}s$ and $10^{-6}s$. Grids with two different densities are shown, $b = 2$ and $b = 5$.

Figure 6 shows theoretical predictions on how the total compute time of different grid search scales as a function of the grid size and number of dimensions, as predicted by (1). From this figure it should be possible to get an estimate of how long a particular problem will take to solve. For example, for a problem where the objective function takes $\bar{t} = 10^{-6}s$ an $n = 8$ dimensional grid could be sampled at resolution $b = 5$, i.e. 32 points per axis, in about 3 hours on 100 processors; but if $\bar{t} = 1.0s$ the same calculation would be about 5.5 years. The calculations that produced Figure 6 are carried out by the utility program *PGStime.F90* and executable *./bin/PGStime* using parameters $b, n, \bar{t}$ and n_p in file `pgstime.in`.

Plotting

Some plotting software is included to plot slices through the optimum solution found in the grid search. The driver program *PGSex.F90* writes out a series of 2-D arrays of the objective function centred on the optimum point into file `planes.dat`. This has a simple ASCII format that can be seen by examining the source code. Plotting is achieved with program *cplot-p*.

Compilation of plotting software is separate to the steps above, and is achieved by changing directory to plot '> cd plot; make all'. After running *pgs_mpi* or *pgs_serial* postscript images of 2-D contour values of the function can be produced by

> cd plot; cp ../planes.dat .

> cplot-p 2 planes.dat

which will produce a postscript file called `contourplot.ps` with a contour image of the first 2 pairs of data in file `planes.dat`. Other possibilities are

> cplot-p 2 planes.dat -f

which creates the same plot with annotation information taken from the file `plot.cmd`.

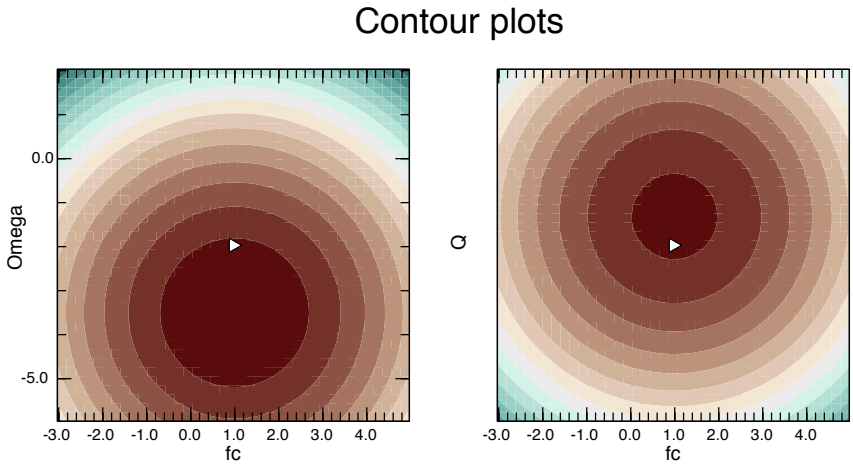

Figure 7: Example contour plot produced by program *cplot-p* showing two planes of a quadratic objective function which was minimised by the nested grid search. This plot was produced with the command 'clot-p 2 planes_quad.dat -f'

> cplot-p 6 planes.dat -f

which plots the first 6 pairs of variables in `planes.dat` with annotation information taken from the file `plot.cmd`. NOte this only makes sense if at least 6 pairs of contour data were actually written to the file.

> cplot-p 6 planes.dat -f -r

Plots the same plot and normalises all contour images to the same colour scale, rather than the default which is to normalise per slice. To see examples of the type of plot without running *pgs_mpi* an example data file `planes_quad.dat` is included for reference. All plots are subject to adjustments in position of axis annotation and label which can be controlled from file `finetune`. Command files which influence *cplot-p* are

```
plotcmd        :  controls axis labels, plot order and data range.
finetune       :  controls minor plot positioning
contourvalues  :  contains value of special contours to plot on top of figure
```

See each file for further details. Figure 7 and 8 shows examples using the test data in `planes_quad.dat`.

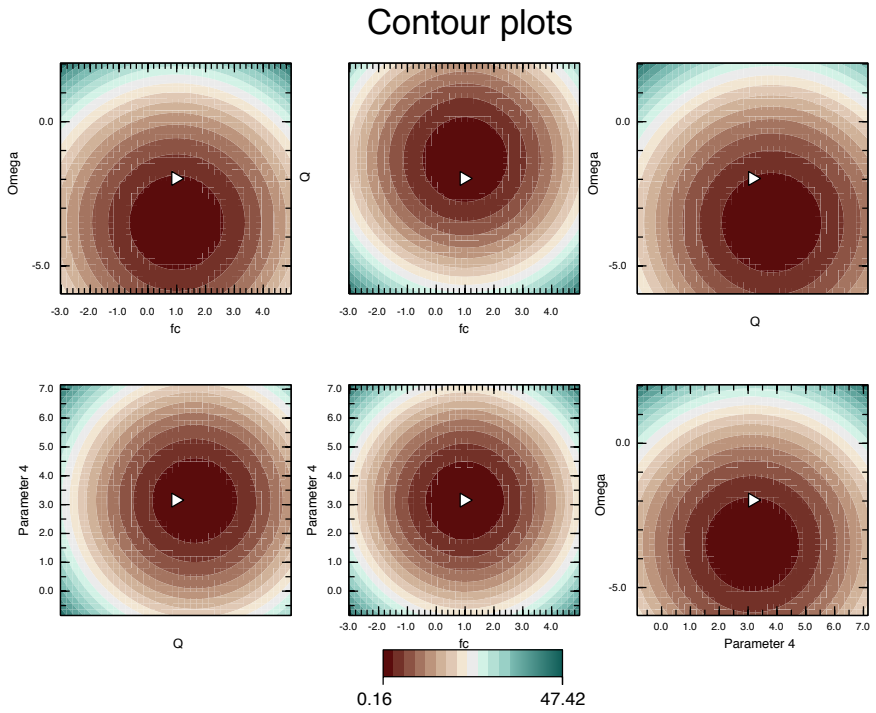

Figure 8: Example contour plot produced by program *cplot-p* showing six planes of a quadratic objective function which was minimised by the nested grid search. Here the colour map is common across all plots. This was produced with the command 'clot-p 6 planes_quad.dat -f -r'

Numerical integration over nested grids

Numerical Monte Carlo integration may be performed of a user supplied function over the hyper-cube. For low dimension problems this will provide an estimate of the integral of the function over the domain.

$$I = \int_V f(\mathbf{x}) d\mathbf{x} \tag{2}$$

In *PGSex.F90* the function for integration is supplied by the subroutine `Integrand` written by the user. This function may be different from the objective function used in the optimisation stage. To perform integration the parameter `integrate` is set to true and a second sweep over a set of nested grids is carried out.

The Monte Carlo estimate of the integral in (2) is given by

$$I_{MC} = \sum_i \frac{f(\mathbf{x}_i)}{\rho(\mathbf{x}_i)}, \tag{3}$$

where $\mathbf{x}_i$ $(i = 1, \ldots, N)$ are the points generated with sampling density $\rho(\mathbf{x})$. For each nested grid produced by *hyper-sweep* its possible to obtain an estimate of the integral in (2) using (3). Figure 9 shows an example for three nested grids.

The sampling density, $\rho(\mathbf{x})$ must be calculated at each integration point, $\mathbf{x}_i$, and used to weight the MC integration (3). Each successive nested grid alters the overall sampling density function, $\rho(\mathbf{x})$. For the initial grid this is given by

$$\rho(\mathbf{x}_i) = \frac{N}{V} = \frac{2^{bn}}{L^n} \tag{4}$$

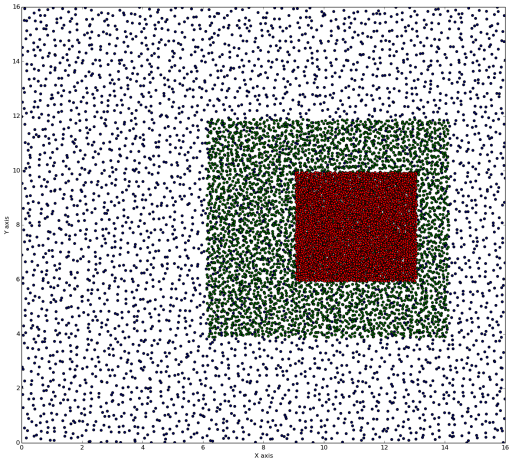

Figure 9: Example of sampling density of three nested grids produced by *hyper-sweep* for the case $n = 2, b = 6$ with parameter `random` set to true. Each point in these grids forms a Monte Carlo integration point.

where L is the length of each side of the hyper-cube. For cases where the sides are different in length, L^n is replaced by $\prod_j^n L_j$, where L_j is the length of the jth side, represented by the variable `xlen(j)` in *PGSex.F90*. In terms of the discretized interval $d\mathbf{x}_1$ of the first nested hyper-grid, we have

$$\rho(\mathbf{x}_i) = \frac{1}{d\mathbf{x}_1^n} \tag{5}$$

showing that the density is the inverse of the volume of the voxel of the first hyper-cube lattice. In Figure 9 we plot points from the first nested grid as blue, from the second as green, and the third as red. When two nested grids are used the density is unchanged for the blue points outside of the green region in Figure 9 but since there are both blue and green points in the 'green' region the combined density becomes

$$\rho(\mathbf{x}_i) = \frac{1}{d\mathbf{x}_1} + \frac{1}{d\mathbf{x}_2} \tag{6}$$

which is the sum of the densities. Similarly, with three nested grids the density in the 'red' region becomes the sum of the three densities, and so on for any number of nested grids. Note that for *hyper-sweep* the ratio of the lattice interval volumes is determined by the relative sizes of the hyper-grids, which is determined by parameter `sfactor`. As the algorithm creates nested grids it is possible that smaller grids are not fully enclosed by the larger grids. This situation is shown in Figure 10. In *hyper-sweep* the convention is adopted that the volume for integration is always the initial grid specified and so subsequent grids outside of the initial volume are ignored. In this situation the density calculation is unchanged.

Experimental results of numerical integration using *hyper-sweep* are shown in Table 1 for the function

$$f(\mathbf{x}) = e^{-\frac{1}{2}\mathbf{x}^T\mathbf{x}}. \tag{7}$$

The error in the Monte Carlo estimate decreases as the successive nested grids are used. However these results are dependent on both the initial range of the grid and the shrink factor, s_F. In effect the nested grids are improving the estimate of the integral only in a sub-domain of the full volume because of the increased number of samples there. However if the integral (7) has contributions outside the initial or sub-sequent ranges that are significant compared to the error in the new domains then the overall accuracy will not be improved by nesting, regardless of the density of sampling. In effect the overall error is a sum of the errors in each domain and in principle any one could be dominant.

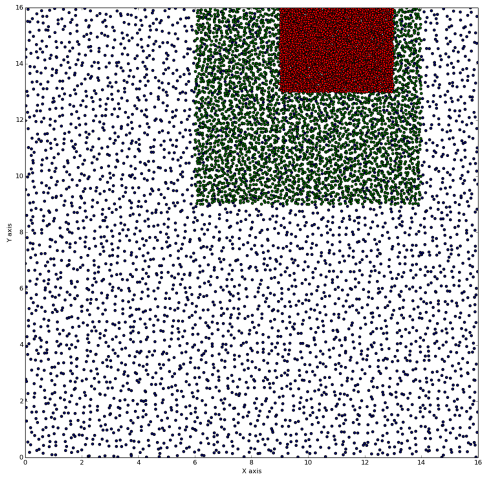

Figure 10: Example of three nested grids in *hyper-sweep* used for Monte Carlo integration where the second (green) and third (red) grids extend outside of the integration volume defined by the first grid and are ignored.

n	b	s_F	nnest	I_{MC}	I	Error	% Error
1	6	0.5	1	2.49188	$(2\pi)^{1/2}$	0.0147	0.59
1	6	0.5	2	2.51201	$(2\pi)^{1/2}$	-0.00539	0.22
2	6	0.5	1	6.262012	2π	0.02117	0.34
2	6	0.5	2	6.283235	2π	-4.963×10^{-5}	0.00079
3	6	0.5	1	15.72356	$(2\pi)^{3/2}$	0.026	0.17
3	6	0.5	2	15.74452	$(2\pi)^{3/2}$	0.0051	0.032
3	6	0.5	3	15.75309	$(2\pi)^{3/2}$	-0.0034	0.022
4	6	0.5	1	39.490745	$(2\pi)^2$	-0.01233	0.031
4	6	0.5	2	39.476245	$(2\pi)^2$	0.00217	0.0055

Table 1: Experimental results of the MC integration feature of *hyper-sweep* for integration of a Normal distribution over various numbers of dimensions. In all cases is the range of integration used in the first nested grid, is ± 16.0 for each dimension. s_F is the factor by which each side of the grid is multiplied during successive nests.

Acknowledgments

Hyper-sweep is distributed through the *Inversion Laboratory* which has support from the AuScope Australian Geophysical Observing System (AGOS, 2011-2014). AuScope is an organisation that manages construction of research infrastructure and is funded through the Australian Federal Government's NCRIS and EIF3 programs. If you make use of Hyper-sweep in work that leads to scientific presentations or publications, please acknowledge the author and AuScope[2].

M. Sambridge,
Canberra,
26/03/2014.

[2]Please send any corrections of these notes to Malcolm.Sambridge@anu.edu.au.

References

Lawder, J. K. (2000), Calculation of mappings between one and n-dimensional values using the hilbert space-filling curve, *Technical report no. JL1/00, August 15 2000*, School of Computer Science and Information Systems, Birkbeck College, University of London.

Skilling, J. (2004), Programming the Hilbert Curve, in *Bayesian inference and maximum entropy methods in science and engineering - 23rd International Workshop on Bayesian Inference and Maximum Entropy Methods in Science and Engineering*, edited by G. Erickson and Y. Zhai, pp. 381 – 387, AIP Conference Proceedings 707.