Notes on the package *PT*

Introduction

The *PT* software library is a set of Fortran 90 routines for implementing Parallel Tempering of a user supplied Markov chain. The software here was used in the paper *Sambridge* (2014) where further details can be found. For an example of an application of Parallel Tempering in a geophysical setting see *Dosso et al.* (2012). The package includes a scalar and parallel version which are implemented through calls to a single routine from a user supplied driver program.

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 makes use of the MPI protocol when compiled in parallel to before message passing between processes. 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 McMC algorithm. The library is organised in such a way that all details of the parameter space, its dimension, the target PDF as well as the nature of the user supplied McMC algorithm are unseen by the PT algorithm. All calls to the MPI library are also unseen. The intention is that running in serial or in parallel can be carried out with the same driver program.

Software packGage

The package consists of a single Fortran 90 library of routines, only one of which *pt(...)* must be called from the user's driver program. Source code for the library is in the file `./PT-demo/src/pt.F90`. An example driver program which calls this routine is in `./PT-demo/Examples/Ex1/pt_ex1.F90`. This example program implements sampling of a bi-modal test PDF. See the examples section below for details.

Installation

To compile the library:

- Edit the file `./PT-demo/compile-options` to suit your platform. This file contains information about where serial and parallel compilers reside on your machine. In the default case locations for an MPI linked compiler `mpif90` and a serial compiler `gfortran` are specified. If you have to change these locations in the top directory then do the same in the `Examples` directory. To compile then

  > cd ./PT-demo
  > make all

  This will create an object file `./PT-demo/lib/pt.o`. To remove object and .mod files

> make clean

- In some systems the first call to make will also descend into the `Examples` directory and compile each test problem, but it may be necessary to do this manually. For example

  > cd ./PT-demo/Examples
  > make all.

  This will create executables for all examples, e.g. `Ex1/run/pt_ex1.serial.x` and `Ex1/run/pt_ex1.mpi.x` and similar for other examples. To remove executables type

  > make clean

Running examples

If all goes well the executables can be run and will produce results as described below. In `Ex1/run` a *pbs.sh* batch script is included which I use to run the example `pt_ex1.mpi.x` on 24 cores of the Terrawulff III cluster (http://rses.anu.edu.au/terrawulf/). On a serial platform execution is simply

```
> ./pt_ex1.serial.x
```

and on Mac OSX I was able to run the openmpi compiled version on 7 cores using

> mpirun -np 7 `./pt_ex1.mpi.x`,

The procedure for running other examples, e.g. in `Ex2/run`, follows a similar format. This code has been developed and tested using openMPI and gfortran, with successful compilation and execution on a linux cluster and on Mac OSX 10.7.5.

Example 1: Sampling the twin peaks function

The first test example distributed with the package is a simply defined bi-modal function

$$\pi(x) = 2^{-x} + 2^{-(100-x)}, \tag{1}$$

and here x is an integer variable, $1 \leq x \leq 100$. Using the driver code `pt_ex1.F90` this problem may be sampled with a standard McMC sampler and no parallel tempering (by setting parameter `swaprate = 0`.) The result is a single McMC chain which is unable to escape from the peak at $x = 0$. As shown in Figure 2. Changing the swap rate to unity means that exchange swaps are proposed once for every along chain step in the x direction. The change in the Markov chain's ability to jump across the low probability region is dramatic (see Figure 2). See *Sambridge* (2014) for further details.

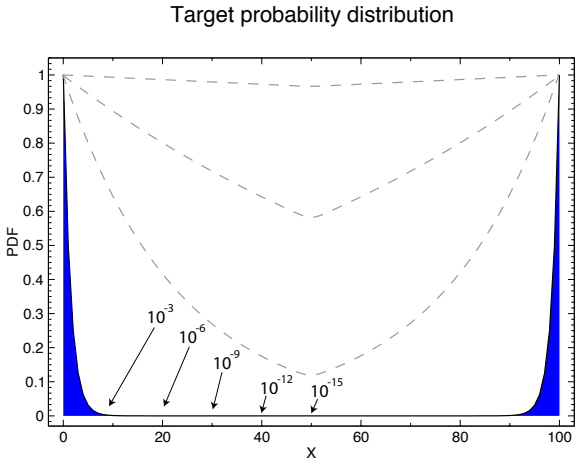

Figure 1: Bi-modal probability density function sampled in the test problem. All chains begin in the peak at $x = 0$ and must pass through a region of low probability in order to reach the right peak at $x = 100$. After *Atchadé et al.* (2011).

These results are produced by plotting the output ensemble x positions at each step in the cold chain ($T = 1$). In the example these are written to the file `xsamples_k.txt` where `k` is the process number. Other output files are `pt.log` which is a short summary text file of the various settings used; `Tlevels` which is a list of the randomly generated temperature levels across all chains; `Tbins` which is a list of temperatures which are the central values of bins between which diagnostic information is calculated in terms of the number of successful exchange swaps; and `./log/log_k.txt` which contains statistics of the successful exchange swaps between temperature bins. The subdirectory `log` must be present in the directory where the code is executed. The statistics of exchange swaps can be a useful diagnostic to ensure that the temperature ladder (determined by `Tlow, Thigh`) is distributed appropriately. See Figure 11 of *Sambridge* (2014) for an example and further details.

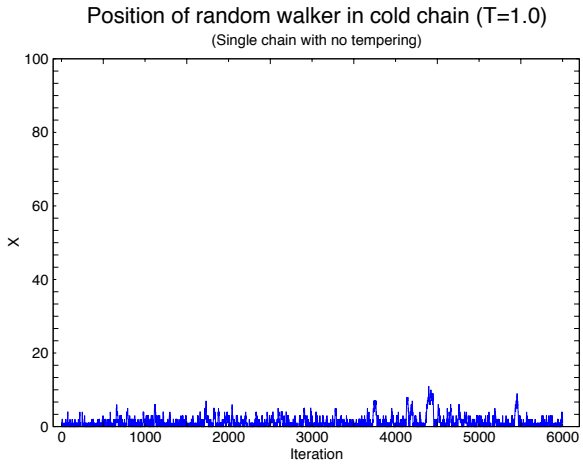

Figure 2: Samples from an McMC chain (x) as a function of step for a single chain with no Parallel Tempering. Despite minor excursions the chain is unable to escape from the peak at $x = 0$ due to the infinitesimal probability region in the centre of the range, which creates a barrier to the Markov chain.

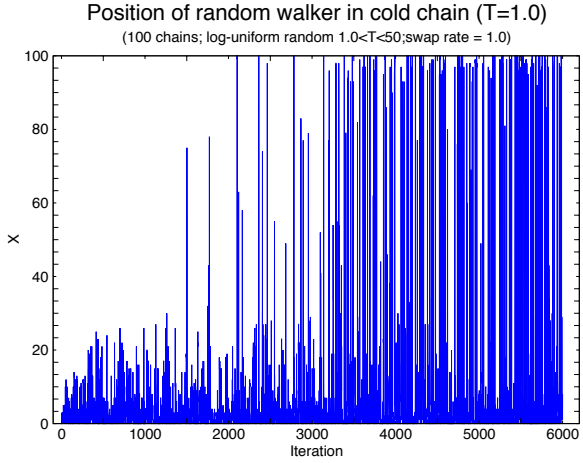

Figure 3: Samples from an McMC chain (x) as a function of step for the cold chain ($T = 1$) with exchange swaps allowed between 100 randomly distributed chains with temperature ranging between $1 \leq T \leq 50$. In this case the cold chain is able to traverse the low probability region in the centre with ease.

```
ialg = 0          :  Algorithm type
                  :  ialg=0, T swap between all levels
                  :  ialg=2, T swap only allowed between
                  :  neighbouring temperature levels
nchains = 100     :  Define number of chains
swaprate = 1.0d0  :  Rate at which exchange swaps are proposed relative
                  :  chain steps.  1.0 = one exchage swap proposed for
                  :  Set this value to zero to turn off Parallel Temper
nsteps = 5000     :  Number of chain steps per temperature
iburn = 1000      :  Number of burn in samples
tlow = 1.d0       :  Lowest temperature of chains
thigh = 50.d0     :  Highest temperature of chains
nbins = 10        :  Number of temperature bins for diagnostics
```

Control parameters set in the driver program `pt_ex1.F90`.

Example 2: Polynomial regression

This test example deals with independently sampling four different polynomial regression functions for fitting the test data set $y_i^{obs}, (i = 1, \ldots, 20)$ shown in Figure 4. The problem is one of Bayesian sampling over the model parameters of each polynomial separately. For this example the likelihood function is

$$p(\mathbf{d}|\mathbf{m}) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{ -\sum_{i=1}^{20} \frac{(y_i^{obs} - y_i^{pred}(\mathbf{m}))^2}{2\sigma^2} \right\}. \tag{2}$$

where $\mathbf{m}$ represents the set of coefficients of the polynomial, and t_i^{pred} the corresponding y values of each curve. The prior in each case is uniform and takes the form

$$p(\mathbf{m}) = \prod_{j=1}^{k} \frac{1}{\Delta m_j}, \tag{3}$$

where k is the number of coefficients in the polynomial and Δm_j the prior range for the j coefficient. The data set shown in Figure 4 is the same as that used by *Sambridge et al.* (2006) to which the reader is referred for more details. The source code for example is in `Ex2/src/pt_ex2.F90`, which is where all details of the chain and PDFs are set. The user is advised to example this example driver program.

The number of chains in the default example is 20 and temperatures are log-uniformly distributed between $1.0 \leq T \leq 50$. The first four chains are all set to $T = 1$ and correspond to cases $k = 1, \ldots, 4$, i.e. sampling with a polynomial containing k coefficients. The k values of each chain are arranged sequentially with every fourth chain performing the same k value.

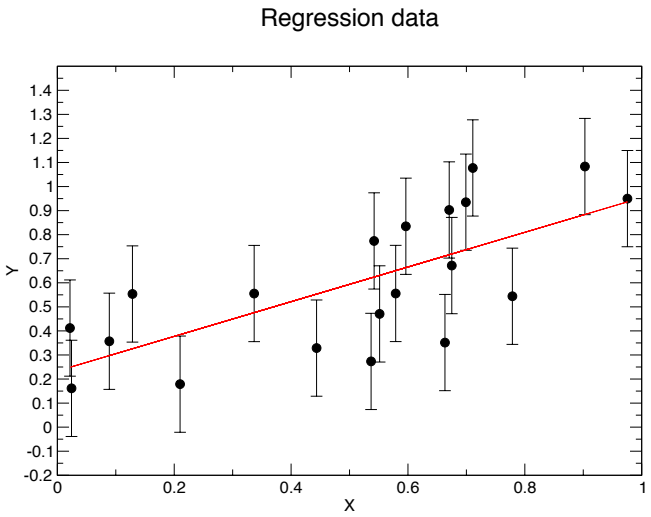

Figure 4: Regression data set used in example 2. Red line is the true linear model and each y value contain Gaussian noise $N(0, \sigma^2)$, with $\sigma = 0.2$.

Results of running all 20 chains with exchange swapping between temperatures are shown in Figures 5, 6 and 7. These plots are produced by the python script `Ex2/run/plot/plot.py` which reads the chain information written to file `Ex2/run/xsamples_0.txt` which contains all details of chains at $T = 1$ for this example. This file can get very large for long runs.

Evidence calculations

Note that this example does not perform trans-dimensional sampling because each chain has a fixed dimension, and so the state spaces for $k = 1, \ldots, 4$ do not mix. However by calculating the conditional evidence it is possible to determine the relative support provided by the data for each polynomial. In the source routine `Ex2/src/pt_ex2.F90` the posterior PDF is calculated by routine `target` for any value of k in the range $1 \leq k \leq 4$. By editing this source and changing the parameter `ityp` one can control whether the routine returns the posterior (`typ = 0`), the prior (`typ = 1`), or the likelihood function (`typ = 2`). As described in *Sambridge et al.* (2006), by sampling the prior and collecting the average value of the likelihood one gets an estimate of the evidence $p(k|\mathbf{d})$ for each chain. The user can experiment with this by setting the variable `typ = 1` in subroutine `target`. In which case the output file `Ex2/run/pk_hist0.txt` will contain estimates of the evidence for each state, $k = 1, \ldots, 4$. These values should be in agreement with those determined by *Sambridge et al.* (2006) for the same data set, which are approximately $0.04\%, 88.5\%, 10.5\%$ and 0.9% for cases $k = 1, \ldots, 4$ respectively. In this case the data strongly support the $k = 2$ case which is indeed the correct one for this data set, as seen in Figure 4.

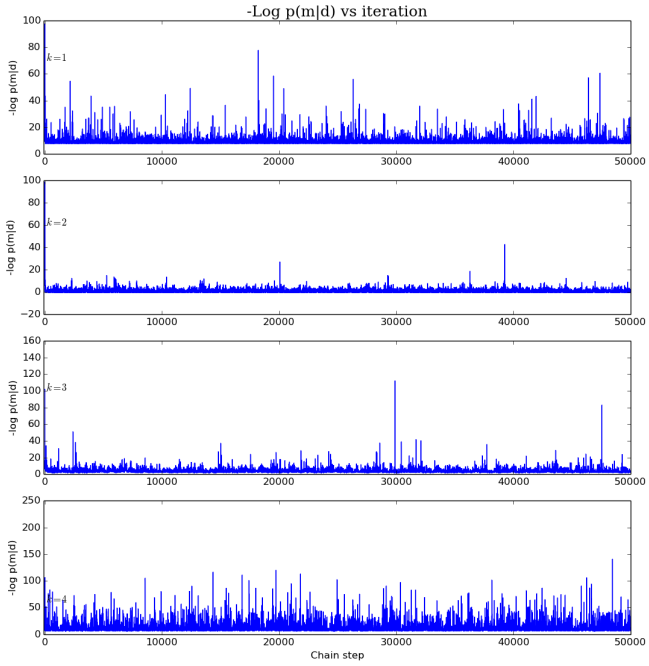

Figure 5: Results of Bayesian sampling with Parallel Tempering for the regression data set in Figure 4, showing -log-likelihood as a function of chain step for the four cold chains at $T = 1$, corresponding to cases $k = 1, \ldots, 4$. The first variable is the constant in a linear regression and the second is the gradient. Red bars show the true values which lie near the peak of the marginal PDFs.

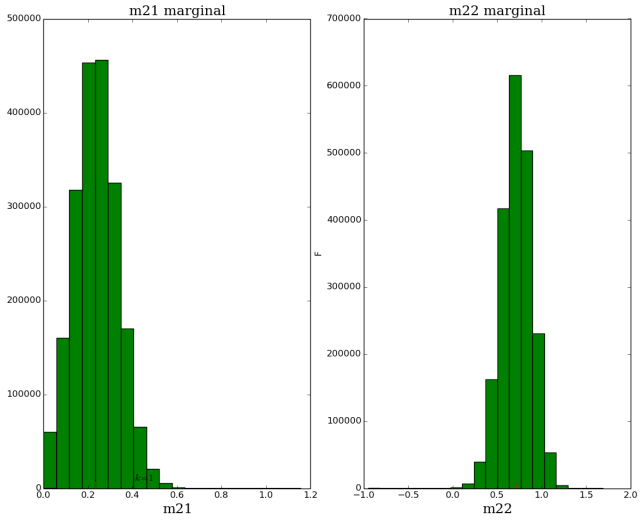

Figure 6: Results of Bayesian sampling with Parallel Tempering for the regression data set in Figure 4, showing marginal PDFs for the two polynomial coefficients in case $k = 2$ in cold chains at $T = 1$. Each variable is a coefficient in the polynomial regression. Red bars show the true values which lie near the peak of the marginal PDFs.

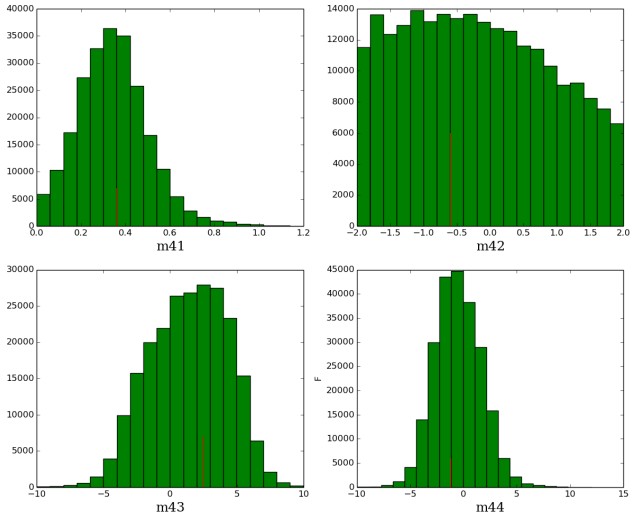

Figure 7: Results of Bayesian sampling with Parallel Tempering for the regression data set in Figure 4, showing marginal PDFs for the four polynomial coefficients in case $k = 4$ in cold chains at $T = 1$.

Example 3: Transdimensional Polynomial regression

Example 3 is identical to example 2 only each chain is now sampled with a trans-dimensional Markov chain. This means that the number of unknowns in each chain is not fixed but undergoes birth and death steps in addition to adjusting the polynomial coefficients. The within chain *move* steps are identical to example 2 and involve a Gaussian perturbation of coefficients using a standard deviation equal to 1/20 of the prior range for each coefficient. The birth step proposal is to add a new coefficient generated from the uniform prior PDF for that variable. During a birth all existing coefficients are unchanged. Similarly, a death step is to delete the highest order coefficient. The theory of trans-D inference on regression problems can be found in *Sambridge et al.* (2006); *Gallagher et al.* (2011).

The source code implementing these additional steps can be found in `Ex3/src/pt_ex3.F90` which must be examine for further details. A run of the example produces an additional file called `pk_histN.txt` where `N` is the processor rank. This contains the percentage of visits to each model state collected over the $T = 1$ chains. These frequencies reflect the marginal posterior on model dimension, $p(k|\mathbf{d})$ and are a direct estimate of the relative evidence for each value of k given the data. As in example 2 these values are known, from *Sambridge et al.* (2006), to be $0.04\%, 88.5\%, 10.5\%$ and 0.9% for cases $k = 1, \ldots, 4$ respectively. Table 1 shows some results obtained with the source code for example 3.

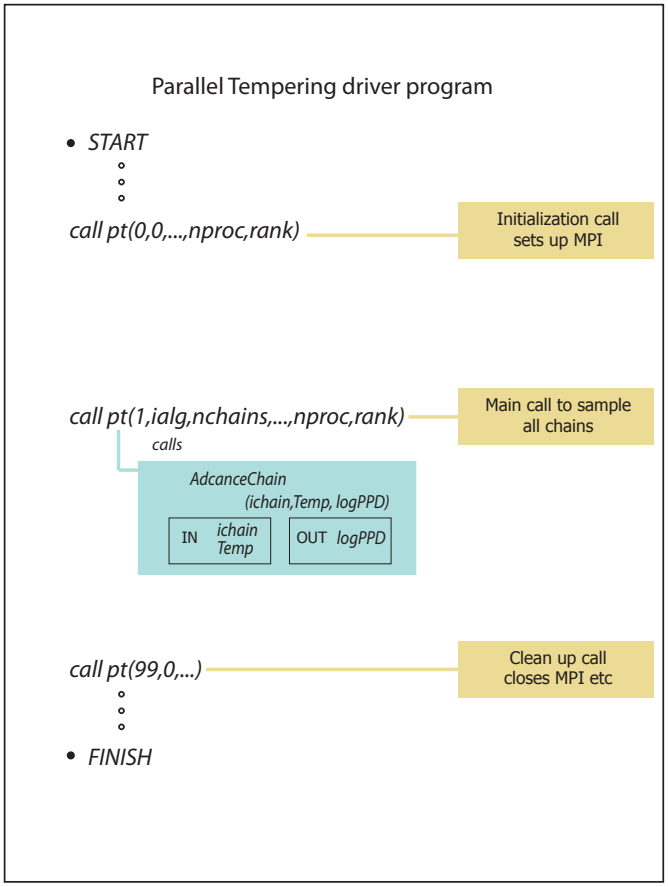

Figure 8: Basic structure of a program making use of the PT library. The PT routine communicates with the users program through calls to the routine *AdvanceChain* which carries out one step of the users McMC algorithm for the chain `chain` at temperature `Temp` and returns the value of the log of the target probability distribution, `logPPD`. Routine AdvanceChain contains all details of the Markov chain and must be present.

Target PDF	$p(1)$	$p(2)$	$p(3)$	$p(4)$	T	Exchange swaps	No. chains
Prior	24.95	25.08	25.12	24.85	1.0	No	2
Prior	25.02	25.01	25.06	24.90	$1.0 \leq T \leq 2.0$	Yes	20
Posterior	0.0415	87.46	11.97	0.53	1.0	No	20
Posterior	0.0377	87.45	11.46	1.05	$1.0 \leq T \leq 2.0$	Yes	20
Posterior	0.0408	87.07	11.85	1.04	$1.0 \leq T \leq 50.0$	Yes	20

Table 1. Some results of experiments using example 3. Table shows percentage values of marginal on dimension, $p(k|\mathbf{d}, T)$ for the cold chain, $T = 1$, simulated with tempered trans-D sampling.

Modifying the driver program

The source code for both the driver programs `Ex1/src/pt_ex1.F90`, `Ex2/src/pt_ex2.F90` and the library `pt.F90` contain notes on the calling sequence and the purpose of various sub-routines. With careful examination of these it will be possible to write your own code to utilise

Parallel Tempering combined with your own McMC or optimisation algorithm. Usually this is best achieved by modifying the example program. Here we give some pointers to the basic structure and set up of the code.

The primary structure of the driver program is to call the *PT(...)* routine three times (see Figure 8). Once for setup (where MPI is initialised if present), once to do all of the work and finally to clean up the chains and finished the MPI (if present). The inclusion of MPI calls is determined by a flag at compilation time. See the `Makefile` in `./PT-demo/Examples`. The main *PT(...)* routine communicates with the user's program through calls to the routine *AdvanceChain* which carries out one step of the McMC algorithm for the chain `chain` at temperature `Temp` and returns the value of the log of the target probability distribution, `logPPD`. Routine AdvanceChain contains all details of the Markov chain and must be present. Any data required by the Markov chain to perform the *within-chain* step must be passed with a module directly to the routine *AdvanceChain*, i.e. without being passed through the PT library (to maintain independence). For example if this were a Bayesian sampling problem then the data and any non varying parameters associated with the model would have to be passed this way.

In the example in `Ex1` AdvanceChain is an interface to a routine called *McMC* which advances the chain of the simple bi-modal test problem. An important constraint is that the temperature passed in through the calling sequence to AdvanceChain **must** be used to temper the current Markov chain in order to implement Parallel Tempering. This means that the temperature should be used to determine the acceptance criterion of the *within-chain* steps performed by *AdvanceChain*. For example, if the target PDF for sampling is $p(\mathbf{m}|\mathbf{d})$ then it is assumed that the target PDF is raised to the power of T^{-1} in the Metropolis-Hastings criterion, i.e.

$$\alpha(\mathbf{x}'|\mathbf{x}) = \min\left[1, \left(\frac{\pi(\mathbf{x}')}{\pi(\mathbf{x})}\right)^{1/T} \frac{q(\mathbf{x}|\mathbf{x}')}{q(\mathbf{x}'|\mathbf{x})}\right], \tag{4}$$

During an exchange swap the algorithms swaps values of the temperature parameter between chains rather than swapping the model between chains. In this way the details of the Markov chain are independent of the PT library and only the passing of a single scalar parameter is required between processors, which reduces communication overhead in MPI mode. For further details on the calling arguments to each PT routine see the source code for the driver program `./Examples/Ex1/src/pt_ex2.F90`

Optional routines

Optional routines used in the bi-modal example are *PT_diagnostics(...)* which can be used to collect statistics of performance of exchange swaps and *PT_McMC_accept(...)* which implements the tempered Metropolis-Hastings acceptance criterion for the *within-chain* step, (4). Both of these are used in the example driver program to demonstrate use.

Acknowledgments

PT 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 *PT* in work that leads to scientific presentations or publications, please consider citing the papers below, acknowledging the author for use of the code and including a URL of Inversion Laboratory (*http://www.iearth.org.au*) [1].

M. Sambridge,
Canberra,
8/05/2014.

References

Atchadé, Y. F., G. O. Roberts, and J. S. Rosenthal (2011), Towards optimal scaling of metropolis-coupled Markov chain Monte Carlo, *Statistics and Computing*, *21*(4), 555–568, doi:10.1007/s11222-010-9192-1.

Dosso, S. E., C. W. Holland, and M. Sambridge (2012), Parallel tempering in strongly nonlinear geoacoustic inversion, *J. Acoust. soc. Am.*, *132*(5), 3030–3040.

Gallagher, K., T. Bodin, M. Sambridge, D. Weiss, M. Kylander, and D. Large (2011), Inference of abrupt changes in noisy geochemical records using transdimensional changepoint models, *Earth Planet. Sci. Lett.*, *311*, 182–194.

Sambridge, M. (2014), A parallel tempering algorithm for probabilistic sampling and multimodal optimization, *Geophys. J. Int.*, *196*, 357–374, doi: 10.1093/gji/ggt342.

Sambridge, M., K. Gallagher, A. Jackson, and P. Rickwood (2006), Trans-dimensional inverse problems, model comparison and the evidence, *Geophys. J. Int.*, *167*, 528–542.

[1] Please send any corrections or feedback to Malcolm.Sambridge@anu.edu.au.

Notes on Parallel Tempering algorithm implementation

Theory and scaling relatopnships

Background

The notes here provide a basic description of Parallel Tempering and provide details on some particular implementation issues associated with running on multiple multi-processors. The central problem is to draw simulations, $\mathbf{x}_j, (j = 1, \ldots, N)$ from an *unnormalized* target probability density function, $\pi(\mathbf{x})$, where $\mathbf{x}$ is a state space vector of D parameters. Typically this would be done with some form of Markov chain Monte Carlo (McMC) algorithm which generates a *chain* of vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots$, whose density, when converged, and possibly filtered, approximates the target distribution.

The basic structure of a Parallel Tempering (PT) algorithm is to have n_c such McMC chains running in parallel with each sampling a tempered version of the target distribution, i.e.

$$\pi(\mathbf{x}, T_i) = \pi(\mathbf{x})^{1/T_i}, \quad (i = 1, \ldots, n_c)$$

Tempering is seen as raising the target probability distribution to a power of $1/T$, its *temperature*. The effect of temperature is to flatten the peaks of the target distribution, making larger transitions across the space by the McMC algorithm more likely and hence a more efficient sampler. For $T = 1$ the target distribution is sampled, while as $T \to \infty$, $\pi(\mathbf{x})$ tends toward a uniform distribution.

It is evident that this formulation encompasses both Bayesian inference calculations, where draws are required from a posterior probability density function based on some data $\mathbf{d}$, i.e.

$$\pi(\mathbf{x}) = p(\mathbf{x}|\mathbf{d}),$$

and also global optimisation problems, where a global minimum is sought of some energy function $E(\mathbf{x})$, in which case we set

$$\pi(\mathbf{x}) = e^{-E(\mathbf{x})}.$$

In the latter case explicit optimisation is not the goal of a sampling algorithm, but draws from the target distribution will be attracted to the maximum of the target distribution $\pi(\mathbf{x})$, even if it is multi-modal since it is the point of maximum probability density, or equivalently minimum energy. This is done stochastically without the need to calculate derivatives of the energy function with respect to the model vector. For examples of applications of Parallel Tempering in a geophysical setting see *Dosso et al.* (2012); *Sambridge* (2014)

Basic relationships

If the n_c chains of the Parallel Tempering algorithm has n_g copies of each temperature and n_t temperatures, and this situation is repeated on each of n independent physical nodes or cpu cores, we have

$$n_c = n_t n_g n.$$

If within each chain the McMC random walker is run for n_l steps (the length of the chain) then the total number of samples generated n_s is

$$n_s = n_l n_t n_g n.$$

This ignores the samples generated in an initial 'burn in' period typically required for McMC chains to approximately converge. An update of the state vector within the ith Markov chain occurs according to probabilistic decisions using standard Metropolis-Hastings (M-H) rules. This is done by first perturbing the chain's state vector $\mathbf{x}_i$ to $\mathbf{x}_i'$ and then accepting or rejecting the change using the M-H acceptance criterion. This ensures the condition known as *detailed balance* which is required for the sequence of state vectors to converge to the desired local target distribution, $\pi(\mathbf{x})^{1/T_i}$. We call this *advancing the chain*.

In practice temperatures might be generated at random according to a particular probability distribution, e.g. log-uniform between two bounds (see parameter `tlow` and `thigh` above). This is the case for the example in the PT library, in which case $n_g = 1$.

Temperature swaps

The central idea in Parallel Tempering is to occasionally exchange the model vector $\mathbf{x}$ between chains in a probabilistic fashion which also preserves detailed balance. Equivalently the temperature parameter can be swapped between chains. i.e.

$$(\mathbf{x}_i, T_i), (\mathbf{x}_j, T_j) \rightarrow (\mathbf{x}_i, T_j), (\mathbf{x}_j, T_i).$$

It can be shown that to preserve detailed balance this should be done with probability

$$\alpha_{i,j} = \min[1, \exp{(1/T_i - 1/T_j)(E(\mathbf{x}_i) - E(\mathbf{x}_j))}],$$

See *Sambridge* (2014) for details. If a proposal to swap temperatures occurs at a rate of r per chain step then the expected number of temperature swap proposals is, N_p, where

$$N_p = r \times n_l n_t n_g n.$$

If we allow temperature swap proposals uniform randomly between all possible $n_t \times n_t$ chain pairs then the expected number of exchange proposals for each pair of chains is N_P/n_t^2. Each chain index is chosen independently at random. If a represents the expected number of proposals between the chain i and itself $(i = j)$, and b between chain i and chain j $(i \neq j)$ then evidently

$$a = \frac{N_p}{n_t^2}, \qquad b = \frac{2N_p}{n_t^2}.$$

The second value is double the first because a swap between the pair (i, j) is identical to that between pair (j, i).

Parallelism

We want to implement the algorithm within a parallel computing environment in such a way that it behaves statistically identically whether there are n_c chains distributed across n nodes, or all n_c on a single node. Allowing for multiple chains on a single processor means that communication overhead can be minimised, and also the library can be used on a single cpu for testing.

The arrangement with the least communication volume is one where the state vectors remain on each node and temperatures are swapped between nodes. To do this the PT library uses a master-slave model, where each slave advances n chains independently and decides to propose an exchange swap with rate r per chain, and the master makes all decisions. When the ith chain is 'ready' for an exchange swap then its partner must be randomly chosen with equally likelihood from all chains. However only a subset of the chains reside on each node $i.e.$ $1/n$ (here n is the number of slaves). Hence selection for the partner of the ith chain is a two stage process. The first is to decide whether a swap occurs with chain on the same node or a different not, and once the node is known a chain must be selected. If a swap is to occur within the same node then partner chain j can be chosen at random from the N_p/n available. If the partner is on a different node then a request is made to the master to find a pair. In this case the ratio of internal to external swaps on each node needs to be determined in such a way to make the nodes invisible, and the likelihood equal for any pair of chains to be proposed for a swap.

Let s_1 be the expected number of proposals to be made between chain pairs on the same slave node, and s_2 be the expected number of requests by each slave to the master. We must have the total number of proposals equal to N_p, hence

$$N_p = ns_1 + n\frac{s_2}{2}$$

The factor of $1/2$ comes from the fact that the master receives two requests from slaves at random before a pair can be formed. Also the ratio of s_1 to s_2 must equal the ratio of pairs that can be form within the node and between this and all other nodes. Since each node has n_c/n chains, there are $(n_c/n)^2$ pairs within the node. The number of pairs that can be formed between this node and all other nodes is $2(n-1)(n_c/n)^2$, hence

$$\frac{s_1}{s_2} = \frac{(n_c/n)^2}{2(n-1)(n_c/n)^2} = \frac{1}{2(n-1)}$$

Solving for s_1 and s_2 we get

$$s_1 = \frac{N_p}{n^2}, \qquad s_2 = \frac{2N_p(n-1)}{n^2},$$

which is the portion of within node to between node pairings that must be achieved in order for parallelism to be invisible. Hence once an exchange swap proposal is decided upon, then a random choice is made for the first chain member pair, i, and a second random decision is made to either select the second pair within the node or request from the master. The second decision happens in the ratio s_1/s_2. Hence a random number $0 \leq r \leq 1$ is generated and if $r > 1/2(n-1)$ then an external swap is proposed and otherwise an internal one. In this way each the conditions are met and each has an equal likelihood of being pair with every other one.

A table of of examples.

n_t	n_g	n	n_l	r	N_p	a	b	s_1	s_2
2	1	1	2×10^4	0.02	800	200	400	800	0