7 Markov-Chain Monte Carlo (MCMC) Generation of the Posterior

The final step in Bayesian analyses is to generate MCMC traces through parameter space. *Smooth Emulator* software is designed for the MCMC programs to be run from within the `${MY_PROJECT}` directory. Once the emulator is tuned and before it is applied to a Markov Chain investigation of the likelihood, the software needs know the experimental measurement and uncertainty. That information must be entered in the `Info/experimental_info.txt` file. The file should have the format:

```
obsname1  -12.93    0.95   0.5
obsname2  159.3     3.0    2.4
obsname3  -61.2.    1.52   0.9
obsname4  -1.875    0.075  0.03
   ⋮
```

The first number is the measured value and the second is the experimentally reported uncertainty. The third number is the uncertainty inherent to the theory, due to missing physics. For example, even if a model has all the parameters set to the exact value, e.g. some parameter of the standard value, the full-model can't be expected to exactly reproduce a correct measurement given that some physics is likely missing from the full model. For the MCMC software, the relevant uncertainty incorporates both, and only the combination of both, added in quadrature, affects the outcome. We emphasize that this last file is not needed to train and tune the emulator. It is needed once one performs the MCMC search of parameter space.

The log-likelihood, $\boldsymbol{LL}$, for the MCMC generation is assumed to be of a simple form. Summing over the observables $\boldsymbol{I}$,

$$\begin{aligned} \boldsymbol{\sigma^2_{I,\mathrm{tot}}} &= \boldsymbol{\sigma^2_{I,\mathrm{exp}}} + \boldsymbol{\sigma^2_{I,\mathrm{theory}}} + \boldsymbol{\sigma^2_{I,\mathrm{emulator}}}, \\ \boldsymbol{LL} &= -\sum_{\boldsymbol{I}} \frac{(\boldsymbol{Y_{I,\mathrm{exp}}} - \boldsymbol{Y_{I,\mathrm{emu}}})^2}{2\boldsymbol{\sigma^2_{I,\mathrm{tot}}}} + \log(\boldsymbol{\sigma_{I,\mathrm{tot}}}). \end{aligned}$$

The main program that runs the mcmc code is `${MY_LOCAL}/main_programs/mcmc_main.cc`. To compile the program:

```
${MY_LOCAL}/main_programs% make mcmc
```

Next, one needs to edit the parameter file `${MY_PROJECT}/parameters/mcmc_parameters.txt`. An example file is:

```
MCMC_LANGEVIN false
MCMC_METROPOLIS_STEPSIZE 0.04
MCMC_LANGEVIN_STEPSIZE 0.5
MCMC_NBURN  100000
MCMC_NTRACE 100000
MCMC_NSKIP  5
RANDY_SEED  12345
```

The program can run either a Metropolis algorithm or a Langevin algorithm. If `MCMC_LANGEVIN` is `false`, the Metropolis algorithm is invoked. The Langevin algorithm is faster when one has large numbers of parameters, but it ignores the emulator uncertainty (because it has a $\boldsymbol{\theta}$ dependence). Thus, if the emulator uncertainty is significant, the Metropolis method is recommended. For the Metropolis method, the stepsize of the random steps should be adjusted so that the Metropolis success probability is on the order of one half. If the probability is only a few percent, or if it is very close to 100% the efficiency of the method suffers. When one runs `mcmc` it will report the success percentage. If the percentage is worrisome, one should reset `MCMC_METROPOLIS_STEPSIZE` and re-run. The parameter `MCMC_NBURN` is the number of Metropolis steps the program attempts during the burn-in stage, i.e. before the trace is reported when the trace might be outside the likely region. The trace then records `MCMC_NTRACE` steps, taking `MCMC_NSKIP` steps between writing down the information about the point in model parameter space. Thus, after burn-in, the chain performs `MCMC_NSKIP`$\times$`MCMC_NTRACE` samplings, recording `MCMC_NTRACE` values of $\vec{\boldsymbol{\theta}}$ and $\vec{\boldsymbol{X}}$. At the current time, the Langevin method is not fully tested and should be avoided.

Then, from the project directory, run the program. Output should look something like this:

```
${MY_PROJECT}% ${MY_LOCAL}/bin/mcmc
 At beginning of Trace, LL=-15.526125
 At end of trace, best LL=-0.131612
 Metropolis success percentage=72.030000
 finished burn in
 At beginning of Trace, LL=-2.904881
 finished 10%
 finished 20%
 finished 30%
 finished 40%
 finished 50%
 finished 60%
 finished 70%
 finished 80%
 finished 90%
 finished 100%
 At end of trace, best LL=-0.075831
 Met This is list of $\vec{\theta}$ for each point in the trace.
\item {\tt XbarThetabar.txt}: Same as the output above
\item {\tt sigma2.txt}: This gives the $N_{\rm par}\times N_{\rm par}$ covariance matrix $\
\item {\tt sigma2\_eigenvalues.txt}: That eigenvalues of that matrix
\item {\tt sigma2\_eigenvectors.txt}: The eigenvectors
\end{itemize}
The information for $\langle\delta \theta_i\delta\theta_{b}\rangle$ might be useful later o

Finally, we review the source code in {\tt \$\{MY\_LOCAL\}/main\_programs/mcmc\_main.cc}:
{\tt
\begin{verbatim}
int main(){
```

```
  CparameterMap *parmap=new CparameterMap();
  parmap->ReadParsFromFile(string("parameters/emulator_parameters.txt"));
  parmap->ReadParsFromFile(string("parameters/mcmc_parameters.txt"));
  CSmoothMaster master(parmap);
  CMCMC mcmc(&master);
  master.ReadCoefficientsAllY();
  unsigned int Nburn=parmap->getI("MCMC_NBURN",1000);  // Steps for burn in
  unsigned int Ntrace=parmap->getI("MCMC_NTRACE",1000); // Record this many points
  unsigned int Nskip=parmap->getI("MCMC_NSKIP",5); // Only record every Nskip^th point
  mcmc.PerformTrace(1,Nburn);
  CLog::Info("finished burn in\n");
  mcmc.PruneTrace(); // Throws away all but last point
  mcmc.PerformTrace(Ntrace,Nskip);
  mcmc.EvaluateTrace();
  mcmc.WriteTrace();
  return 0;
}
```

This is mostly self-explanatory. If one wishes to avoid writing out the trace, the line `mcmc.WriteTrace` can be deleted. If the User wishes to run the code in batch mode, the output can be directed to a file, `mcmc_log.txt`, by adding the line

```
SmoothEmulator_LogFileName mcmc_log.txt
```

to the parameter file `parameters/mcmc_parameters.txt`.