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

7.1 Preparing for MCMC using *Smooth Emulator*

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 `smooth_data/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 $\boldsymbol{I}$,

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

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

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

One should tune the emulator before running `mcmc`. This could involve running the emulator and saving the Taylor coefficients, or inserting the tuning into the main source code for `mcmc` mentioned above.

7.2 MCMC Parameter File

Next, one needs to edit the parameter file `${MY_PROJECT}/smooth_data/smooth_parameters/mcmc_parameter`
An example file is:

```
#LogFileName mcmc_log.txt
MCMC_LANGEVIN false
MCMC_METROPOLIS_STEPSIZE 0.04
MCMC_LANGEVIN_STEPSIZE 0.5
MCMC_NBURN  100000
MCMC_NTRACE 100000
MCMC_NSKIP  5
MCMC_IGNORE_EMULATOR_ERROR false
RANDY_SEED  12345
```

As was the case with *Smooth Emulator*, each parameter has a default value. Summarizing the parameters:

- `LogFileName`
  To run interactively, leave this line commented out. Otherwise, output will be directed to the designated file.
- `MCMC_LANGEVIN`
  The Langevin method can replace the Metropolis method, but it is not fully tested. If the default, `false`, is set, the Metropolis method will be invoked. The Langevin method currently ignores the emulator uncertainty. At this time it is not clear that once it is invoked, it will significantly improve performance.
- `MCMC_METROPOLIS_STEPSIZE`
  Metropolis algorithms require taking random steps in $\vec{\boldsymbol{\theta}}$ space. If the steps are small, it takes longer to explore the space, but if the steps are very long, the success rate of the Metropolis steps becomes low. Maximum efficiency occurs when the Metropolis success rate, which is provided during running, is near 50%. Rates of 20% or 80% are also fine, but if the rate is only a few percent of if it becomes close to 100%, the User should adjust the parameter.
- `MCMC_LANGEVIN_STEPSIZE`
  For the Langevin procedure, all steps are successful, but the accuracy can suffer if the steps are too large. (The Langevin option is still in development)
- `MCMC_NBURN`
  A certain number of Metropolis or Langevin steps should be taken before the trace is recorded so that the trace is not biased by the starting value.
- `MCMC_NTRACE`
  This is the number of points in the trace that are recorded for subsequent analysis. More points provides a more accurate representation of the posterior.
- `MCMC_NSKIP`
  Because succesive points in the trace are correlated, it makes sense to skip several points before skipping. For example, in the Metropolis procedure if the success rate is 50%, the neighboring points have a 50% chance of being the same. The values of `MCMC_NBURN`, `MCMC_NTRACE` and `MCMC_NSKIP` are stored by the `CMCMC` objects, but are only set when the object calls the `trace` function. Thus, default values are set at the time the call is made to the `PerformTrace` funtion. See Sec. 7.4 below for an example of how this is used.

- MCMC_IGNORE_EMULATOR_ERROR
  If this flag is set to true the emulator error will be ignored. This significantly increases the speed of the MCMC procedure, but should not be done if the emulator error is significant.

7.3 Running the MCMC Program

To run the provided MCMC program, to the project directory, and run the program mcmc. Output should look something like this:

```
${MY_PROJECT}% ${MY_LOCAL}/bin/mcmc
 At beginning of Trace, LL=-68.764478
At end of trace, best LL=1.563806
Best Theta=
0.249554  0.153237  0.190531  0.210907  0.058929  0.230998
Metropolis success percentage=54.090000
finished burn in
At beginning of Trace, LL=-5.477040
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=1.678685
Best Theta=
0.269108  0.099867  0.185094  0.204939  0.037417  0.207398
Metropolis success percentage=53.609600
writing, ntrace = 100001
writing, ntrace = 100001
```

For the sake of numerical efficiency ‘‘Metropolis success percentage’’ should be in the range of 50%. If the efficiency is very near 100%, it suggests the step sizes might be too small to best explore the entire model-parameter space. If the efficiency is near zero, the step size might be too large and too few successful Metropolis steps will be taken. This affects only the efficiency, not the validity, so any success percentage between 10% and 90% should suffice. In the output, ‘‘LL’’ refers to the log-likelihood. At the end of the burn-in, one hopes that best value of LL is not much lower than the best value found from the entire trace. If not, one should probably increase the value of MCMC_NBURN. The values of ‘‘Best Theta’’ refer to the point in the trace with the highest LL.

Information about the trace is found in the files located in the directory smooth_data/mcmc_trace/:

- Xtrace.txt: A list of the model-parameter values from the posterior sampling.

- `trace.txt`: A list of the scaled model-parameter values ($\vec{\boldsymbol{\theta}}$) from the posterior sampling.
- `xbar_thetabar.txt`: The average of the parameter values ($\vec{\boldsymbol{X}}$), and the scaled values ($\vec{\boldsymbol{\theta}}$) from the posterior.
- `CovThetaTheta.txt`: This gives the $\boldsymbol{N}_{\mathbf{par}} \times \boldsymbol{N}_{\mathbf{par}}$ covariance matrix $\langle \boldsymbol{\delta\theta_i \delta\theta_j} \langle$, describing the size and shape of the points in the trace.
- `CovThetaTheta_eigenvalues.txt`: That eigenvalues of that matrix
- `CovThetaTheta_eigenvecs.txt`: The eigenvectors
- `ResolvingPower.txt`: An $\boldsymbol{N}_{\mathbf{pars}} \times \boldsymbol{N}_{\mathbf{obs}}$ matrix describing the influence of each observable in resolving each model parameter.

7.4 Reviewing the MCMC Source Code

Finally, we review the source code in `${MY_LOCAL}/software/main_programs/mcmc_main.cc`:

```
int main(){
  CparameterMap *parmap=new CparameterMap();
  CSmoothMaster master(parmap);
  CMCMC mcmc(&master);
  //master.ReadCoefficientsAllY();
  master.TuneAllY();
  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

```
LogFileName mcmc_log.txt
```

to the parameter file `parameters/mcmc_parameters.txt`. The line `mcmc.EvaluateTrace()` will evaluate the trace and calculate the resolving power and covariances.