Documentation for Equation Free Code

Matt Williams

8/31/21

The following is a short document on the Equation Free code I wrote up during my summer internship. The code is intended for my own personal use, so is almost definitely lacking in comments in a number of locations. Please feel free to contact me at `mowill@uw.edu` if need be.

1 MATLAB Code (CMS)

The MATLAB code for the implementation of Equation Free Modeling for the Eichner-Dietz problem is consists of object oriented MATLAB code. The objects required to run the code are as follows:

- CMSWrapper.m - Wrapper code to initialize and read from the CMS
- dataSet.m - Matlab class that contains data from the micro scale. The constructor deals with different possible input formats (cell array, large matrix, etc)
- EQFScales.m - Matlab class that implements the lifting and restriction operators
- macroState.m - Matlab class that contains the time and state of the macro scale system. Overloads the multiplication, addition, and subtraction operators
- ProcessResults.m - Extract statistics from the macro scale
- Stepper.m - Implements the coarse grained projective integrator
- runParameters.m - Contains parameters like the size of the projective step and number of healing steps
- executeRun.m - executes these files in the correct order

It also requires the following parameter files. These names can be changed, but the defaults are:

- eqfObserve - list of observables
- eqfParams - list of parameters and their values
- eqfReactions - list of reactions (in the .emodl format)
- config - config file for CMS
- ICData.mat - initial condition data (micro scale level)

These files must be in the same directory (or the path must include these files) in order for the EQF to run. The files that will change most frequently are EQFScales.m, ProcessResults.m, Stepper.m, and runParameters.m, in order to select the projective step size and the number of healing steps.

For an example of how to run this code, see the file executeRun.m in the Example subdirectory.

2 Python Code (GPU based SSA)

The Python Equation Free code requires numpy[1], scipy[2], pyopencl[3] (version 2012.2), and pylab[4]. The latter, pylab, is only needed for plotting. Some of the EQF code uses it, but could be safely commented out with the only loss being a lack of plots. The necessary files are in two folders:

- SSA
  - emodlParse.py - parses a subset of .emodl files (doesn't implement all the commands) and generates openCL code
  - SSAOpenCL.py - Interface between the CPU and GPU to run SSA and gather results
  - SSATemplates.py - Template files (contains strings) where the reactions are to be filled in the appropriate places
- EQF
  - EQF.py - top level EQF class
  - EQFTimesteppers.py - File containing projective integrators (similar to Stepper.m)
  - EQFWrapper - Declares the micro scales and macro scales

With these six files and the associated .emodl and config files, we are able to parse and run simple .emodl files with good results. With larger file sizes, the amount of memory each thread requires is the main bottleneck.

Running on a NVIDIA 570 GTX[5], it takes around 5 minutes of computational time for the Eichner-Dietz model with 100,000 realizations. The computational cost scales linearly, as expected, with some granularity due to that fact that the system is most efficient when every computational core is being exploited.

To run this code, see the sample script below:

```
from EQF import *

GetN.NRuns = 100000 # number of iterations to run
NRuns = GetN()

ICFull = np.tile(np.array([16680, 74, 74, 320, 320,0,0,0,0,202000],"float64"),NRuns) # set initial condi
macroIC = microMeanCov(0.0, ICFull, 0.0).restrict() # Generate initial macroscale


Stp = EulerStep(2./12, 0, 0.0, 'EDVaccinate.emodl', 'config.json', 'EDModel/') # initialize timestepper
Wrap = EQF(Stp, macroIC,0.0,10.0) # initialize EQF class

# Perform and time the coarse projective integration
t1 = time.time()
output = Wrap.CPI()
t2 = time.time()

# Save and write the results in an external numpy array
Perad = np.zeros((len(output),3))
```

[1]http://numpy.scipy.org
[2]http://www.scipy.org
[3]http://mathema.tician.de/software/pyopencl/
[4]http://www.scipy.org/PyLab
[5]http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-570

```
for ii in xrange(0, len(output)):
    Perad[ii,2] = output[ii].perad

Perad[:,0] = 0.0
Perad[:,1] = t2-t1

np.savetxt("Data/EulerStep_microMeanCov_0_0.0",np.array(Perad))
```