November 16, 2016

Tools and Techniques of Computational Science – Fall 2016

Canned Project Proposal

The task for this project is to design and implement a well-written, organized, and verified scientific application that numerically approximates the solution to some differential equations. The first example is a simple problem that will be solved with a forward euler method. The second is a system of ODEs that simulate charged particle motion in a vertical electric field. You will solve this system using a variety of approaches.

You are free to implement your application in either C, C++, or F90. Some items for consideration and requirements when coding are summarized below:

- adopt a modular coding style - I do not want to see one big monolithic main function. Instead, organize your code into manageable chunks with functions/subroutines tasked with focused units of work. Your main driver routine should be a fairly short function.
- include comments in your code
- use dynamic memory allocations for variables that derive sizing information from runtime input options
- use git frequently: begin by creating a directory named "project" in your repository and commit often. Do not wait till you are done to commit everything.
- provide a mechanism to have multiple levels of output. At a minimum, I want to see two modes: standard and debug. The debug mode will include more verbose (debug) information when chosen.
- include performance timing measurements for key kernels within the code
- have a verification mode that solves one of the equations and compares the numerical to the analytical solution
- perform asymptotic convergence analysis and provide a log-log plot of the results for all methods
- compare timings of your forward euler to the tpl's forward euler implementation

Simple ODE

Solve a simple first order differential equation of your chosing using forward euler. You are to solve this problem two ways: 1) Explicitly implement forward euler yourself and 2) use a third party library (such as GSL) to solve the problem.

Show convergence to the analytical solution as a function of h. Also compare the performance of your implementation to that of the tpl implementation.

Charged particle motion in a vertical electric field

The trajectory $\mathbf{x}(t)$ of the particle shown below is governed by a set of three second order differential equations:

$$\ddot{x} = \omega\dot{y} - \frac{\dot{x}}{\tau} \tag{1a}$$

$$\ddot{y} = -\omega\dot{x} - \frac{\dot{y}}{\tau} \tag{1b}$$

$$\ddot{z} = -\frac{\dot{z}}{\tau}, \tag{1c}$$

where ω and τ are constants. An initial value problem of this type needs $3 \times 2 = 6$ initial conditions. These are given as the initial position $\mathbf{x}_0 = (x_0, y_0, z_0)$ and the initial velocity $\dot{\mathbf{x}}_0 = (\dot{x}_0, \dot{y}_0, \dot{z}_0) = (u_0, v_0, w_0) = \mathbf{v}_0$ of the particle.

In order to solve the equations above numerically, they have to be rewritten as a set of six first order differential equations:

$$\dot{x} = u \tag{2a}$$

$$\dot{y} = v \tag{2b}$$

$$\dot{z} = w \tag{2c}$$

$$\dot{u} = \omega v - \frac{u}{\tau} \tag{2d}$$

$$\dot{v} = -\omega u - \frac{v}{\tau} \tag{2e}$$

$$\dot{w} = -\frac{w}{\tau}. \tag{2f}$$

For this problem, you can assume ω and τ are both set to the value of 5, the initial position of the particle is defined as $\mathbf{x}_0 = (0, 0, 0)$ and its initial velocity vector as $\mathbf{v}_0 = (20, 0, 2)$.

For this problem, use a third party library to find the trajectory $\mathbf{x}(t)$ using three different Runge–Kutta methods with one of them being the classical Runge–Kutta method aka RK4. Analyze the performance and convergence of each of the methods. Plot the results of the trajectory in 3D using either gnuplot or matplotlib. Also provide the scripts necessary to produce your plot output.

Input Options

Your application must derive necessary runtime options from an input file. You can specify the name of the input file as a command-line argument, or you can assume that your application will always read from the same filename. If you choose the latter, assume an input file named "input.dat". The exact format of the input file is up to you, and you may use the GRVY library installed on Stampede if you want to simplify the process (recall that API documentation is available online).

Regardless of the format, you need to support a minimum number of runtime options including the following:

- choice between problems
- choice between solution methods
- choice of step sizes (h)
- option to run in a verification mode
- option to control standard output mode (e.g. standard vs debug)

Build System

Your application must include a build system using either autotools or make that works on Stampede. This build system must have configuration options to link in all 3rd party libraries that your application requires. It must also have an option to run a suite of regression tests via the "make check" target. It also should support a "make coverage" target that uses lcov/gcov to process the results from the test suite.

Code Verification

Provide a "verification" mode option in your input file which will enable your code to compute the difference between your numerical solution and the analytic solution. Your code should output this error norm within the standard output when running in verification mode.