UNIVERSITY OF NEW HAMPSHIRE

Numeric Analysis of One Dimensional Korteweg-de-Vries Equation

IAM851 – Project 1

Anthony Edmonds

4/14/2014

A libaray was writen to numerically solve the Korteweg-de-Vries equation using a 4th order Runge-Kutta. The equation was discretized using finite differences and then solved using the Runge-Kutta integrator. The basic theory, methods, results and documentation are outlined in the following report.

Contents

1 Introduction

The Korteweg-de-Vries (KdV) equation is a one dimensional partial differential equation (PDE) that describes the behavior of shallow water waves. The equation can be solved analytically or numerically. For the numeric solution the equation must be discretized using finite difference approximations and then solved numerically using an ordinary differential equation (ODE) integrator. A 4^{th} order Runge-Kutta (RK4) was used to numerical integrate the discretized equation.

A library containing all of the necessary functions and tools needed to solve the KdV equation was created, along with ten test examples. The library contains basic vector operation functions needed to solve the equation, an RK4 function, and the discretized KdV equation. The library has been written with the use of OpenMP parallelization.

2 Theory

2.1 Korteweg-de-Vries Equation

The KdV equation is a shallow water wave equation describing the motion of a wave. The equation is:

$$\partial_t u + \partial_{xxx} u + 6u\partial_x u = 0 \tag{1}$$

Where u is the height of the wave above the seafloor, the first time is the time rate of change of that height, the second term is the dispersion of the wave, and the third term is an advection term. To use the RK4 or any other numeric time integrator the equation must be rearranged into the desired slope, or time rate of change, of the function.

$$\partial_t u = -\partial_{xxx} u - 6u\partial_x u \tag{2}$$

In order to solve the equation numerically the equation must be first discretized using finite differences. First and third order finite difference formulas are required and are derived using a Taylor series expansion around the desired point with a panel of size Δx. The centered first ordered finite difference is:

$$f'(x) \approx \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x} \tag{3}$$

When written in code:

$$\partial_x u_i \approx \frac{1}{2\Delta x} * (u_{i+1} - u_{i-1}) \tag{4}$$

The third order requires a second order derivative, so the following second order centered derivative was used:

$$f''(x) \approx \frac{f(x+\Delta x) - 2f(x) + f(x-\Delta x)}{\Delta x^2} \tag{5}$$

The third derivative is:

$$f'''(x) \approx \frac{f''(x+\Delta x) - f''(x-\Delta x)}{2\Delta x} \tag{6}$$

Substituting in equation 5 and rewriting in code:

$$\partial_{xxx} u_i = \frac{\partial_{xx} u_{i+1} - \partial_{xx} u_{i-1}}{2\Delta x} \tag{7}$$

$$\partial_{xxx} u_i = \frac{\frac{u_{i+2} - 2u_{i+1} + u_i}{\Delta x^2} - \frac{u_i - 2u_{i-1} + u_{i-2}}{\Delta x^2}}{2\Delta x} \tag{8}$$

$$\partial_{xxx} u_i = \frac{u_{i+2} - 2u_{i+1} + 2u_{i-1} - u_{i-2}}{2\Delta x^3} \tag{9}$$

Since accessing i-1 and i-2 at indices 0 and 1, and accessing i+2 and i+1 at indices N-2 and N-1, where N is the number of discretization points, will be outside the bounds of the array, and will cause issues, periodic boundary conditions are used such at:

$$u_{-1} = u_{N-1} \tag{10}$$

$$u_{-2} = u_{N-2} \tag{11}$$

$$u_N = u_0 \tag{12}$$

$$u_{N+1} = u_1 \tag{13}$$

2.2 4th Order Runge-Kutta

The ODE integration scheme is implemented with the following equations:

$$s_1(t_j) = \partial_t u_i\big(t_j, u_j\big) \tag{14}$$

$$s_2(t_j) = \partial_t u_i\left(t_j + \frac{\Delta t}{2}, u_j + \frac{\Delta t}{2} s_1\right) \tag{15}$$

$$s_3(t_j) = \partial_t u_i\left(t_j + \frac{\Delta t}{2}, u_j + \frac{\Delta t}{2} s_2\right) \tag{16}$$

$$s_4(t_j) = \partial_t u_i\big(t_j + \Delta t, u_j + \Delta t\, s_3\big) \tag{17}$$

Where s_1, s_2, s_3, and s_4 are evaluated at each node i in the grid, j is the time, and Δt is the time step. Equations 14-17 are then used in the RK4 integration:

$$u_{j+1} = u_j + \frac{\Delta t}{6}(s_1 + 2s_2 + 2s_3 + s_4) \tag{18}$$

3 Code Documentation

3.1 Installing

The library is packaged in a tarball, to unpack it navigate it to the desired directory and unpack it using the tar command as shown in Figure 1:

```
[user@server folder]$ tar zxvf kdv_solver-0.01.tar.gz
```

Figure 1 - Unpacking the tarball that contains the library. The following flags are enabled: -z to handle gzip files, -x extract, -v verbose mode, and –f force overwrite.

The tarball is now unpacked and ready to be installed. First run the reconfigure command as shown in Figure 2:

```
[user@server folder]$ autoreconf -i
```

Figure 2 - autoreconf command

Then the configure file can be ran, as in Figure 3:

```
[user@server folder]$ ./configure
```

Figure 3 – running configure

And lastly run make, as shown in Figure 4:

```
[user@server folder]$ make
```

Figure 4 - running make

The library is now installed. The headers can be used by including solver.h and kdv_equation.h.

3.2 Built-in Tests

The library includes ten tests each with their own corresponding output files; each test covers basic function uses and demonstrates the stability and scaling of the functions. Three simple tests involve solving three basic initial conditions for one hundred points over a space eight units long, with a time step of 5.2767E-4s. Three tests using a compound waves under the same conditions as the initial tests. One test demonstrating the effects of using a large number of points over a fixed simulation range. One test demonstrating the effects of changing the number of points on how long a simulation for a fixed time of four seconds takes. One test comparing a single threaded operation to a parallel threaded operation running a number of threads as determined by the user. Lastly, the library includes a test that demonstrates the profiling of the code with increasing number of points and increasing number of threads.

3.3 Sample Plots

Gifs will be posted on the Trac page for the project (https://fishercat.sr.unh.edu/trac/iam851_2014/wiki/Anthony/KdV_Solver) and will be hosted on IMGUR.

3.4 Profiling

When the code is ran on one thread for a simple input wave for a simulation time of one second for an increasing number of points from 64 to 2048 points results in the following profiling time:

Table 1 - Profiling experiment on one thread.

N	Trial 1 (s)	Trial 2 (s)
64	0.002942	
128	0.046989	
256	0.746099	
512	11.847038	
1024	188.813222	
2048	3020.770979	

Analyzing the results show in Table 1, the code is Order four scaling, that is if the number of points is double the solver will take sixteen times as long to solve the same simulation. The test was then run in parallel for the same simulation length and the following results were generated:

Table 2 - Results of Parallel Profiling

N	1 (s)	2 (s)	3 (s)	4 (s)
64	0.006722	0.014307	0.012926	0.012692
128	0.061015	0.113882	0.111424	0.108398
256	0.855917	0.867641	0.972169	0.974582
512	12.698554	8.287408	10.258021	9.390722
1024	196.035347	131.482480	111.370206	98.536929
2048	3077.525748	1863.235957	1407.280200	1166.687538

N	5 (s)	6 (s)	7 (s)	8 (s)
64	0.012367	0.014011	0.017601	0.021546
128	0.104066	0.117272	0.143911	0.134979
256	0.920049	1.016804	1.182457	1.178153
512	8.914179	9.165369	14.230247	11.895155
1024	90.293318	89.683614	97.169185	105.429503
2048	1027.308352	988.470930	1151.062747	1255.573654

Table 2 shows that the code takes longer for N less than about 512 points, as the overhead of running the threads takes longer than the code does to execute. Comparing the time it takes between one and two threads, and two and four threads shows about a 40% reduction in time the code takes to operate. Unfortunately, the results from threads six and above are not reflective the behavior, this is due to the server being used by another project during the profiling experiment. The project time for 2048 points on eight threads is about 700.01 seconds.

3.5 Function Use

3.6 Plot Generation

4 Conclusion

5 Appendix