Numerical Analysis (CS 450)

Homework Set 6

Due: May 7, 2014 · Out: April 24, 2014

Problem 1: Numerical Methods for ODEs and Dissipation (20 points)

Consider the initial value problem for the harmonic oscillator

$$u'' = -u, \qquad u(0) = 1, \qquad u'(0) = 0. \tag{1}$$

(a) Set up a system of first-order ODEs corresponding to this problem. What is the exact solution of (1)?

(b) Write code to solve the system using

  (i) the fourth-order Runge-Kutta scheme, and

  (ii) the so-called *leapfrog* method:
  The centered difference approximation

$$y' \approx \frac{y_{k+1} - y_{k-1}}{2h}$$

  leads to the explicit integrator

$$y_{k+1} = y_{k-1} + 2hf(t_k, y_k)$$

  for solving the ODE $y' = f(t, y)$. Start the system by using one step of Heun's method. This method is also known as the Störmer-Verlet method[1].

Compute the empirical order of convergence of each method by using different time step sizes and using the maximum norm as your measure of error against the exact solution.

(c) The total energy of a harmonic oscillator consists of two contributions, kinetic energy and potential energy. Taking the mass and the spring constant to be 1, the total energy can be written as

$$E = \frac{1}{2}u'^2 + \frac{1}{2}u^2$$

Using this expression, plot the total energy of the oscillator over a long period of time (at least 400 periods of the oscillator) for both methods.

What do you observe about the evolution of the energy over time in either method? Which method would describe as 'dissipative', i.e. as losing ('*dissipating*') energy over time? Why could this be problematic in applications involving orbital dynamics? More concretely, if you use a dissipative method to simulate the trajectory of a satellite orbiting the earth over a long period of time, what would happen to the satellite?

[1] `https://en.wikipedia.org/wiki/Verlet_integration`

(continued on next page...)

Harmonic oscillators are an example of a Hamiltonian system[2]. In such a system, the total energy (the so-called *Hamiltonian*) is conserved. Symplectic integrators[3] are numerical schemes that aim to, up to some oscillation, preserve the Hamiltonian.

Your writeup should include the first-order form, the exact solution, your convergence estimates, one long-term plot each of u vs t for the two methods, and one plot showing the total energy for both the methods, along with responses to the text questions on dissipation.

Problem 2: BVP theory (20 points)

Consider the two-point BVP for the second-order scalar ODE

$$u'' = u, \quad 0 < t < b, \tag{2}$$

with boundary conditions

$$u(0) = \alpha, \quad u(b) = \beta.$$

(a) Rewrite the problem as a first-order system of ODEs with separate boundary conditions.

(b) Show that the fundamental solution matrix for the resulting linear system of ODEs is given by

$$\boldsymbol{Y}(t) = \begin{bmatrix} \cosh(t) & \sinh(t) \\ \sinh(t) & \cosh(t) \end{bmatrix}$$

(c) Are the solutions to this ODE stable?

(d) Determine the matrix $\boldsymbol{Q} \equiv \boldsymbol{B}_0\boldsymbol{Y}(0) + \boldsymbol{B}_b\boldsymbol{Y}(b)$ for this problem.

(e) Determine the rescaled solution matrix $\boldsymbol{\Phi}(t) = \boldsymbol{Y}(t)\boldsymbol{Q}^{-1}$.

(f) What can you say about the conditioning of $\boldsymbol{Q}$, the norm of $\boldsymbol{\Phi}(t)$, and the stability of solutions to this BVP as the right endpoint b grows?

[2]https://en.wikipedia.org/wiki/Hamiltonian_system
[3]http://en.wikipedia.org/wiki/Symplectic_integrator

(continued on next page...)

Problem 3: BVPs and the method of manufactured solutions (20 points)

Consider the linear boundary value problem (BVP) of the form:

$$u''(x) + p(x)u'(x) + q(x)u(x) = f(x) \text{ on } (a, b), \qquad u(a) = g, \quad u(b) = h. \tag{3}$$

(a) Write a function to solve this BVP using a second-order centered finite difference approximation.

Your function should have a call signature as below:

```
def bvp_solve(p, q, f, x, g, h):
    # add code here
```

In writing this code, proceed as follows:

(1) Create sparse matrices D and E that implement first and second centred derivatives respectively. Test those to make sure they do the right thing. (no turned-in result required for this)

Hint: Look up and use `scipy.sparse.csr_matrix` and `scipy.sparse.diags` for this.

(2) Next, obtain diagonal matrices that implement multiplication by p and q.

Hint: Again, `scipy.sparse.diags`.

(3) Assemble a matrix $\boldsymbol{A}$ to apply the left-hand-side of (3) to a vector $\boldsymbol{u}$ from the parts that you've built so far. (You will just need to add and multiply these matrices.)

(4) Deal with the boundary degrees of freedom $\boldsymbol{u}_0 = u(a)$ and $\boldsymbol{u}_{n-1} = u(b)$ by including the equations $\boldsymbol{u}_0 = g$ and $\boldsymbol{u}_{n-1} = h$ in your linear system. (I.e. the first row of your final operator matrix should contain nothing but a one on the diagonal of the first and last row, and the corresponding entries of the right-hand side should be g and h.)

Hint: `scipy.sparse.vstack`.

(5) Solve the resulting sparse linear system using `scipy.sparse.linalg.spsolve`.

(b) Use the so-called *method of manufactured solutions* to test your implementation.

This method works by picking an arbitrary u, plugging it into the left-hand side of (3), and observing what right-hand side (f above) is obtained. Using boundary conditions also obtained from the arbitrarily chosen u, the BVP is then solved from the obtained data (BCs and f), and the found solution compared to the u initially.

Implement the following test cases:

(i) $u(x) = {}^1\!/_3\, e^{-4x} + {}^2\!/_3\, e^{2x}$ on $(0, 1)$
with the coefficients $p(x) = 2$, $q(x) = -8$,

(ii) $u = \sin \ln x$ on $(1, 2)$
with $p(x) = 2/x$, $q(x) = -2/x^2$,

(iii) $u = -\sin x + 3 \cos x$ on $(0, {}^\pi\!/_2)$
with $p(x) = -1$, $q(x) = -2$.

(c) For each of the cases in part (b), compute the error and verify second-order convergence of your solution on an equispaced mesh of the form $a = x_0, x_1, \ldots, x_{n-1}, x_n = b$ where $n = 2^k, k = 3, 4, \ldots, 20$. You should demonstrate convergence by plotting n versus the ∞-norm of the error on a log-log plot and also drawing a slope 2 line on the plot for comparison for each of the cases separately. Does the error in each case decrease monotonically? Why or why not? Explain the observations in your plots.

(continued on next page...)

(d) For the first case in part (b), compute the 2-norm condition number of the resulting matrices for $k = 3, \ldots, 11$ and plot their values versus n on a loglog plot.

Hint: To compute the condition number, decompress your sparse matrix to a dense one (using `A.todense()`) and then use `np.linalg.cond`.

Observe that the sparse matrix will only take up $O(n)$ storage space, while the dense one will use $O(n^2)$. So this way of computing the condition number will only work for relatively small n.

Problem 4: Iterative methods (20 points)

(a) The Gauss-Seidel (GS) method to solve linear system of equations $\boldsymbol{Ax} = \boldsymbol{b}$ iteratively computes $\boldsymbol{x}$ as

$$x_i^{(k+1)} = \frac{b_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>i} a_{ij} x_j^{(k)}}{a_{ii}}, \quad i = 1, 2, \ldots, n$$

Successive Over-Relaxation (SOR) method accelerates the convergence of GS method by using the next step of GS iterate as a search direction with a fixed search parameter ω. Starting with $\boldsymbol{x}^{(k)}$, we first compute the next iterate that would be given by GS method $\boldsymbol{x}_{GS}^{(k+1)}$ and then take

$$\boldsymbol{x}^{(k+1)} = (1-\omega)\boldsymbol{x}^{(k)} + \omega \boldsymbol{x}_{GS}^{(k+1)}$$

ω is a fixed *relaxation parameter*, $\omega > 1$ gives *over*-relaxation, $\omega < 1$ gives *under*-relaxation and $\omega = 1$ gives exactly the GS method.

Prove that the SOR method diverges if ω does not lie in the interval $(0, 2)$.

(b) The Conjugate Gradient (CG) method for linear system of equations $\boldsymbol{Ax} = \boldsymbol{b}$ is given in Algorithm 1.

Algorithm 1 Conjugate Gradient method for linear systems

```
$\boldsymbol{x}_0$ = initial guess
$\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$
$\boldsymbol{s}_0 = \boldsymbol{r}_0$
for $k = 0, 1, 2, \ldots$ do
    $\alpha_k = \boldsymbol{r}_k^T \boldsymbol{r}_k / \boldsymbol{s}_k^T \boldsymbol{A} \boldsymbol{s}_k$
    $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{s}_k$
    $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k \boldsymbol{A} \boldsymbol{s}_k$
    $\beta_{k+1} = \boldsymbol{r}_{k+1}^T \boldsymbol{r}_{k+1} / \boldsymbol{r}_k^T \boldsymbol{r}_k$
    $\boldsymbol{s}_{k+1} = \boldsymbol{r}_{k+1} + \beta_{k+1} \boldsymbol{s}_k$
end for
```

Show that the subspace spanned by the first m search directions in the conjugate gradient method is the same as the Krylov subspace generated by the sequence $\boldsymbol{r}_0$, $\boldsymbol{A}\boldsymbol{r}_0$, $\boldsymbol{A}^2\boldsymbol{r}_0$, $\ldots$, $\boldsymbol{A}^{m-1}\boldsymbol{r}_0$.

(continued on next page...)

Problem 5: Stability vs. the advection equation (20 points)

The so-called *advection equation* is given by

$$\frac{\partial}{\partial t}u + c\frac{\partial}{\partial x}u = 0, \tag{$*$}$$

where c is a non-zero number indicating the 'speed'.

Observe that $u(x,t) = f(x - ct)$, i.e. transport (or 'advection') of some function f to the right with speed c solves $(*)$, for any f. Use $c = 1$ in this problem.

We consider $(*)$ with an initial condition of

$$u(0,x) = u_0(x), \quad x \in [0,1],$$

where u_0 is a given initial function (see below) and with periodic boundary conditions

$$u(0,t) = u(1,t).$$

Discretize the spatial (x) derivatives using

(i) centered differencing, and

(ii) one-sided 'upwind' differencing.

An upwind scheme is a forward/backward difference in space depending on the sign of c, i.e. the direction of advection. If $c > 0$, the 'wind' blows from the left (and the solution is transported to the right), so a left-facing finite difference is used. If $c < 0$, the reverse applies.

Observe that discretizing in space converts $(*)$ into a large, coupled, linear, '*semidiscrete*' system of ODEs

$$\boldsymbol{u}'(t) = \boldsymbol{A}\boldsymbol{u}(t). \tag{4}$$

Now discretize time (t) using

(i) Euler's method, and

(ii) the fourth-order Runge-Kutta scheme

for the initial conditions

(i) $u_0(x) = \sin(2\pi x)$,

(ii) $u_0(x) = |(4x) \bmod 2 - 1|$ (which should look like a triangle wave),

(iii) $u_0(x) = \begin{cases} 1 & 0 < x < 0.5 \\ 0, & \text{otherwise,} \end{cases}$

for times $t \in [0, 10]$. Use a mesh of 400 points for the spatial discretization. Time steps for PDE discretizations obey the so-called *Courant-Friedrichs-Lewy ('CFL') condition*, namely that if Δx is the spatial step size and Δt the time step, then:

$$\Delta t \le C\frac{\Delta x}{c},$$

where the constant C depends on the spatial differencing method and the time integrator in use.

(continued on next page...)

(a) Investigate the stability of all combinations of methods. (Note that stability does not depend on the initial condition for linear problems such as this one.)

Specifically, determine the size of the CFL constant C. For some methods, no stable time step may exist. A systematic way of studying stability involves finding the eigenvalues of the right-hand-side matrix $\boldsymbol{A}$ from (4) and choosing the time step so that all of its eigenvalues fall within the stability regions of the ODE integrator in use.

See
`http://andreask.cs.illinois.edu/cs450-s14/public/code/09-initial-value-problems/Stability%20regions.html`
for plots of the stability regions.

Hint: An inefficient (but quite effective) way to build the right-hand-side matrix is to find each column of the matrix one-by-one. If $\boldsymbol{e}_i$ is the first unit vector and $f(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x}$ is a function that performs the differencing operation, then $f(\boldsymbol{e}_i)$ will result in the ith column of $\boldsymbol{A}$.

(b) For those of the above combinations where a stable method is obtained, plot a three-dimensional wireframe graph of the solution, with one axis representing space, the other representing time, and the third representing the value of the solution.

Hint: You may use the following code to generate the 3D plot:

```
# mesh is the spatial mesh
# times are the times at which solution is computed
# solution is the computed advection solution

from mpl_toolkits.mplot3d import Axes3D
pt.subplot(111, projection="3d")
pt.gca().plot_wireframe(mesh, times, solution)
```

You may want to verify that your method achieves the order of accuracy you expect by varying the step size, but this is not required in this problem.

Hint: `numpy.roll` may be useful in implementing differencing with periodic boundary conditions.