ENGR 250 Numerical Methods in Matlab
Winter 2015

Take Home Final Exam

Due Friday, March 20th, 11:59 pm

1) Linear Systems

Numerical solutions of Poisson's equation

As discussed in class on 3/10-11, Solving the equation $\nabla^2\phi = \rho$ ultimately leaves us solving a linear system of the form $Ax = b$.

For this problem, solve this system instead on the line, which simplifies the equation to

$$\frac{\partial^2\phi(x)}{\partial x^2} = \rho(x)$$

Solve this equation for $x \in [0,10]$ with $\Delta x = 0.1$,

$$\rho(x) = \begin{cases} x & 0 \le x < 2.5 \\ 5 - x & 2.5 \le x < 7.5 \\ -10 + x & 7.5 \le x < 10 \end{cases}$$

And subject to boundary condition $\phi(0) = \phi(10) = 0$. This type of boundary condition, where the function value is specified on the boundary, is called a *Dirichlet* boundary condition.

Recall, this generates a large system of equations at each point on the line of the form

$$\phi_{i-1} - 2\phi_i + \phi_{i+1} = \rho_i \Delta x^2$$

Which is valid at each internal grid point. Building the Matrix A can be done in a fairly automated fashion with the "diag" function in MATLAB

Name your solution vector ϕ "Final1", and plot its value vs. $x = 0{:}\,0.1{:}\,10$

2) Interpolation and Numerical Differentiation

The value of splines

In class we discussed the value of splines and how they do more than make a graph look pretty. The purpose of this problem is to provide an example of the value of splines by combining several things we have now learned.

We are going to examine a function, $y = \sin(x)$ and its derivatives based on coarse data. Let $x_{coarse} = 0{:}\,0.5{:}\,5$, and $y_{coarse} = \sin(x_{coarse})$. Imagine, for the sake of this problem, that these two vectors are data that come from measurements, and you are unaware that they come from $\sin(x)$.

Your task is to calculate the first and second derivatives in 5 different ways. For many of these methods, we will need a finer x vector. Create a fine x vector, $x_{fine} = 0{:}\,0.01{:}\,5$. The five methods for the derivatives are:

1) **True Derivative**
   The analytical solution to the first and second derivative, evaluated on x_{fine}. Call the first derivative "dydxtrue", and second derivative "d2ydx2true".

2) **Coarse Finite difference**
   Use $O(\Delta x^2)$ accurate finite difference schemes on the coarse data. Call the derivatives "dydxcoarse" and "d2ydx2coarse".

3) **Finite difference on "nearest" interpolation**
   Using $O(\Delta x^2)$ accurate finite difference schemes on the coarse data interpolated on x_{fine}. Use the interp1 function with the 'nearest' option. Call the derivatives "dydxnearest" and "d2ydx2nearest"

4) **Finite difference on linear interpolation**
   Using $O(\Delta x^2)$ accurate finite difference schemes on the coarse data interpolated on x_{fine}. Use the interp1 function with the 'linear' option. Call the derivatives "dydxlinear" and "d2ydx2linear"

5) **Finite difference on spline interpolation**
   Using $O(\Delta x^2)$ accurate finite difference schemes on the coarse data interpolated on x_{fine}. Use the interp1 function with the 'spline option. Call the derivatives "dydxspline" and "d2ydx2spline"

Create three plots, one for the function itself, one for the first derivative, and one for the second derivative. For the plot of the function, plot all 5 methods. For the derivative, plot all except "nearest". For the second derivative, plot all except the linear and spline interpolation. Notice how the accuracy of the interpolation methods degrades as higher derivatives are taken. For each plot, make sure when you plot the coarse data you use a marker (like 'o') instead of a line, since this is only known at coarse locations.

To make the plots and lines easier to distinguish, I recommend using the "legend" and "title" commands in Matlab.

When option 1) above is unavailable (i.e., the function is not known), and a derivative is needed, (2) is the most obvious option, but gives the derivative based only on the coarse data. With interpolation, we can get finer data, but the accuracy degrades quickly when we take higher derivatives. This is an advantage of splines. With higher order polynomial interpolation, we can get even better results that remain accurate to higher derivatives.

Answer the following questions in comments in your m-file:

1) Regarding the plot of the function, rank the three interpolation methods from best to worst at:
   a) Fitting the data (x_{coarse}, y_{coarse})
   b) Fitting the parent function, $y = \sin(x)$

2) Regarding the calculation of the first derivative, rank the 4 methods from best to worst at fitting the parent function derivative, $\frac{dy}{dx} = cos(x)$. How would you rate the accuracy of the spline considering how coarse the data was it came from?

3) Regarding the calculation of the second derivative, rank the 4 methods from best to worst at fitting the parent function second derivative, $\frac{d^2y}{dx^2} = -\sin(x)$

Note: While the results here are specific to this problem, they generalize well so long as our data comes from a function which is infinitely differentiable (we can take its derivative as many times as we like), which is how most physical functions behave.

Hope you had a good time in the class, and good luck in future endeavors. Now, get some sleep.