18.303 Problem Set 2

Due Wednesday, 18 September 2013.

Note: For Julia homework problems in 18.303, turn in with your solutions a printout of any commands used and their results (please edit out extraneous/irrelevant stuff), and a printout of any graphs requested. **Always label** the axes of your graphs (with the `xlabel` and `ylabel` commands), add a title with the `title` command, and add a legend (if there are multiple curves) with the `legend` command. (Labelling graphs is a good habit to acquire.) Because IJulia notebooks let you combine code, plots, headings, and formatted text, it should be straighforward to turn in well-documented solutions.

Problem 1: Finite-difference approximations

For this question you may find it helpful to refer to the notes and readings from lecture 3. Suppose that we want to compute the operation

$$\hat{A}u = \frac{d}{dx}\left[c\frac{du}{dx}\right]$$

for some smooth function $c(x)$ (you can assume c has a convergent Taylor series everywhere). Now, we want to construct a finite-difference approximation for $\hat{A}$ with $u(x)$ on $\Omega = [0, L]$ and Dirichlet boundary conditions $u(0) = u(L) = 0$, similar to class, approximating $u(m\Delta x) \approx u_m$ for M equally spaced points $m = 1, 2, \ldots, M$, $u_0 = u_{M+1} = 0$, and $\Delta x = \frac{L}{M+1}$.

(a) Using center-difference operations, construct a finite-difference approximation $u''_m \approx u''(m\Delta x)$. (Hint: use a centered first-derivative evaluated at grid points $m + 0.5$, as in class, followed by multiplication by c, followed by another centered first derivative. Do *not* separate $\hat{A}u$ by the product rule into $c'u' + cu''$ first, as that will make the factorization in part (d) more difficult.)

(b) By plugging in the Taylor expansions of u and c, show that your approximation in (a) is second-order accurate (errors $\sim \Delta x^2$ for small Δx). (Hint: plug in each term of the Taylor series of c one by one, and you will see expressions from class re-appearing—then you can just quote the results from class. You should get $c'u' + cu''$ plus error terms from this expansion.)

(c) Check your answer to the previous part by numerically computing $\hat{A}u\Big|_1 = c'(1)u'(1) + c(1)u''(1)$, for $u(x) = \sin(x)$ and $c(x) = e^{3x}$, and plotting the errors as a function of Δx, similar to the finite-difference handout from class (refer to the handout posted on the web page for the relevant Julia commands and adapt them as needed). Verify from your log–log plot of the |errors| versus Δx that you obtained the expected rate of convergence.

(d) Show that your finite-difference expressions correspond to approximating $\hat{A}u$ by $A\mathbf{u}$ where $\mathbf{u}$ is the column vector of the M points u_m and A is a real-symmetric matrix of the form $A = -D^T C D$ (give C, and show that D is the same as the 1st-derivative matrix from lecture).

(e) Show that, if $c(x) > 0$, your matrix A from the previous part is negative-definite. (That is, show that $\mathbf{x}^* A\mathbf{x} < 0$ for all $\mathbf{x} \neq 0$.)

(f) In Julia, the `diagm(c)` command will create a diagonal matrix from a vector `c`. The function `diff1(M) = [ [1.0 zeros(1,M-1)]; diagm(ones(M-1),1) - eye(M) ]` will allow you to create the $(M+1) \times M$ matrix D from class via `D = diff1(M)` for any given value of M. Using these two commands, construct the matrix A from part (d) for $M = 100$ and $L = 1$ and $c(x) = e^{3x}$ via

```
L = 1
M = 100
D = diff1(M)
dx = L / (M+1)
x = dx*0.5:dx:L # sequence of x values from 0.5*dx to <= L in steps of dx
C = ....something from c(x)...
A = -D' * C * D / dx^2
```

You can now get the eigenvalues and eigenvectors by `λ, U = eig(A)`, where λ is an array of eigenvalues and U is a matrix whose columns are the corresponding eigenvectors (notice that all the λ are < 0 since A is negative-definite).

(i) Plot the eigenvectors for the smallest-magnitude four eigenvalues. Since the eigenvalues are negative and are sorted in increasing order, these are the *last* four columns of U. You can plot them with:

```
using PyPlot
plot(dx:dx:L-dx, U[:,end-3:end])
xlabel("x"); ylabel("eigenfunctions")
legend(["fourth", "third", "second", "first"])
```

(ii) Verify that the first two eigenfunctions are indeed orthogonal with `dot(U[:,end], U[:,end-1])` in Julia, which should be zero up to roundoff errors $\lesssim 10^{-15}$.

(iii) Verify that you are getting second-order convergence of the eigenvalues: compute the smallest-magnitude eigenvalue λ_M`[end]` for $M = 100, 200, 400, 800$ and check that the *differences* are decreasing by roughly a factor of 4 (i.e. $|\lambda_{100} - \lambda_{200}|$ should be about 4 times larger than $|\lambda_{200} - \lambda_{400}|$, and so on), since doubling the resolution should multiply errors by 1/4.

Problem 2: Inner products, adjoints, definiteness

Here, we consider inner products $\langle u, v\rangle$ on some vector space of complex-valued functions and the corresponding adjoint $\hat{A}^*$ of linear operators $\hat{A}$, where the adjoint is defined, as in class, by whatever satisfies $\langle u, \hat{A}v\rangle$ for all u and v. Usually, $\hat{A}^*$ is obtained from $\hat{A}$ by some kind of integration by parts. In particular, suppose V consists of functions $u(x)$ on $x \in [0, L]$ with the ("Robin") boundary conditions $u(0) = 0$ and $u'(L) = u(L)/L$ as in pset 1, and define the inner product $\langle u, v\rangle = \int_0^L \overline{u(x)}v(x)dx$ as in class.

(a) Show that $\hat{A} = -\frac{d^2}{dx^2}$ is self-adjoint ($\hat{A}^* = \hat{A}$) and negative definite ($\langle u, \hat{A}u\rangle < 0$ for $u \neq 0$). Be careful to account for the boundary terms in the integrals!

(b) You already found in pset 1 that $\hat{A}$ with these boundary conditions has negative real eigenvalues λ_n, consistent with (a). Now, verify numerically that the eigenfunctions are *orthogonal.* Number the eigenvalues in order of increasing magnitude ($|\lambda_1| < |\lambda_2| < \cdots$) and consider the first two eigenfunctions $u_1(x)$ and $u_2(x)$ from pset 1. Set $L = 1$ and numerically compute the integral $\langle u_1, u_2\rangle$ in Julia, and show that the integral is indeed zero (up to the accuracy of the computation). You can define two functions and integrate them over $[0, L]$ in Julia with:

```
L = 1
u1(x) = ...define function here...
u2(x) = ...define function here...
quadgk(x -> conj(u1(x)) * u2(x), 0, L, abstol=1e-13)
```

(Note that the `quadgk` numerical-integral function returns a pair (I, E) of values, where I is the estimated integral and E is an estimated error.)