MATH 450, HOMEWORK 5

DUE WEDNESDAY, APRIL 1, 2015

Part I. Theory

Problem 1. Given $N \in \mathbb{N}$, let $h = \frac{1}{N}$ and $x_k = kh$ for $k = 0, \ldots, N$. Find the finite-difference coefficients $a_0, a_1, a_2, a_3 \in \mathbb{R}$ such that

$$\frac{1}{h^2}\big(a_0 u(x_k) + a_1 u(x_{k+1}) + a_2 u(x_{k+2}) + a_3 u(x_{k+3})\big) = u''(x_k) + \mathcal{O}(h^2),$$

whenever $u \in C^4\big([0,1]\big)$ and $k = 0, \ldots, N-3$.

Problem 2. Given sequences of real numbers $v_0, \ldots, v_N$ and $w_0, \ldots, w_N$, prove the "summation by parts" formula,

$$\sum_{k=1}^{N} v_k (w_k - w_{k-1}) = (v_N w_N - v_0 w_0) - \sum_{k=0}^{N-1} (v_{k+1} - v_k) w_k.$$

Hint: Split up the sum on the left-hand side and reindex.

Problem 3. Given any $v_1, \ldots, v_k \in \mathbb{R}$, prove the inequality

$$\left(\sum_{j=1}^{k} v_j\right)^2 \le k \sum_{j=1}^{k} v_j^2.$$

Hint: Write $\sum_{j=1}^{k} v_j = (v_1, \ldots, v_k) \cdot (1, \ldots, 1)$, and apply Cauchy–Schwarz.

Part II. Programming

Problem 4. In this problem, you're going to learn a bit about SciPy's tools for computing efficiently with SPD banded matrices. Begin by inputting (or adding to the beginning of your code) the following import command:

```
from scipy.linalg import solveh_banded
```

(The `h` in `solveh` stands for "Hermitian," which means the same thing as "symmetric" for matrices with real entries.)

As a motivating example, suppose we have a SPD tridiagonal matrix

$$A = \begin{pmatrix} \alpha_0 & \beta_1 & & \\ \beta_1 & \alpha_1 & \ddots & \\ & \ddots & \ddots & \beta_{999} \\ & & \beta_{999} & \alpha_{999} \end{pmatrix}.$$

This 1000×1000 matrix has one million entries, but we don't want to store them all, since only 2998 of them are nonzero! In fact, since the matrix is symmetric, we only need to store 1999 pieces of information: the entries

$\alpha_0, \ldots, \alpha_{999}$ and $\beta_1, \ldots, \beta_{999}$ along the bands. The bands can be stored conveniently in a 2×1000 matrix A_{b}, defined by

$$A_{\mathrm{b}} = \begin{pmatrix} * & \beta_1 & \cdots & \beta_{999} \\ \alpha_0 & \alpha_1 & \cdots & \alpha_{999} \end{pmatrix}.$$

(The entry $*$ here can be any number; since there is no β_0, it's just ignored.) The system $Ax = b$ can be solved using the command `solveh_banded(Ab,b)`, where `Ab` corresponds to the 2×1000 matrix A_{b}.

- **a.** Use `solveh_banded` to solve the 1000×1000 linear system $Ax = b$, where

$$A = \begin{pmatrix} 2 & -1 & & \\ -1 & 2 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{pmatrix}, \qquad b = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}.$$

  Print out the first 10 entries `x[:10]`.
- **b.** Compare the running time of `solveh_banded` and the standard non-sparse solver `solve` by running the following commands in IPython:

```
%timeit solveh_banded(Ab,b)
%timeit solve(A,b)
```

  where `A`, `Ab`, and `b` correspond to the linear system in part a.
- **c.** Repeat part b—but this time, with a 10000×10000 system. (The non-sparse computation may take a few minutes, so this could be a good opportunity to get a cup of coffee.)

Problem 5. Create a function `solveBVP1(f,N)` that approximates the solution u to the two-point BVP

$$-u''(x) = f(x) \text{ for } 0 < x < 1, \qquad u(0) = u(1) = 0,$$

using the centered, second-order finite difference method with $h = \frac{1}{N}$. Your method should use the function `solveh_banded`, as in Problem 4—not `solve`!

- **a.** Let $f(x) = 1$. Plot the approximate solutions for $N = 4, 16, 256$.
- **b.** Let $f(x) = x^2$. Plot the approximate solutions for $N = 4, 16, 256$.

Problem 6. Create another function `solveBVP2(f,N)` that approximates the solution u to the (slightly different) two-point BVP

$$-u''(x) + u(x) = f(x) \text{ for } 0 < x < 1, \qquad u(0) = u(1) = 0.$$

Again, use `solveh_banded`—not `solve`!

- **a.** Let $f(x) = 1$. Plot the approximate solutions for $N = 4, 16, 256$.
- **b.** Let $f(x) = x^2$. Plot the approximate solutions for $N = 4, 16, 256$.