Homework 4

Due: February 19, 2021

1. (spectral methods, `chebfun`) Download the package `chebfun` from `www.chebfun.org`. This package works with functions that are represented (to machine precision) as sums of Chebyshev polynomials. It can solve 2-point boundary value problems using spectral methods. Use `chebfun` to solve the same problem that you solved in HW3+; i.e.,

$$-\frac{d}{dx}\left((1+x^2)\frac{du}{dx}\right) = f(x), \quad 0 \le x \le 1,$$

$$u(0) = 0, \quad u(1) = 0,$$

where $f(x) = 2(3x^2 - x + 1)$, so that the exact solution is $u(x) = x(1-x)$. Print out the L_2-norm and the ∞-norm of the error.

Solution. We implement `chebfun` as follows

```
addpath(fullfile(cd,'chebfun-master'))
L=chebop(@(x,u) -diff((1+x.^2)*diff(u)),[0,1],0,0);
f=chebfun(@(x) 2*(3*x.^2-x+1),[0,1]);
u=L\f;
u_true=chebfun(@(x) x.*(1-x),[0,1]);
norm(u-u_true)
norm(u-u_true,'inf')
```

The L_2-norm is 2.9211e-15 and the ∞-norm is 3.9993e-15.

2. Write a code to solve Poisson's equation on the unit square with Dirichlet boundary conditions:

$$u_{xx} + u_{yy} = f(x, y), \quad 0 < x, y < 1$$
$$u(x, 0) = u(x, 1) = u(0, y) = u(1, y) = 1.$$

Take $f(x, y) = x^2 + y^2$, and demonstrate numerically that your code achieves second order accuracy. [Note: If you do not know an analytic solution to a problem, one way to check the code is to solve the problem on a fine grid and pretend that the result is the exact solution, then solve on coarser grids and compare your answers to the fine grid solution. However, you must be sure to compare solution values corresponding to the same points in the domain.]

Solution. We implement the 5-point Laplacian as `laplace_five`

```
function [u]=laplace_five(h)
m=1/h-1; x=0:h:1; y=x';
u=zeros(m+2,m+2);
u(:,1)=1; u(:,m+2)=1; u(1,:)=1; u(1,:)=1; u(m+2,:)=1; % BC
f=@(x,y) x.^2+y.^2;
F=f(x(2:m+1),y(2:m+1));
F=F-(u(1:m,2:m+1)+u(3:m+2,2:m+1)+u(2:m+1,1:m)+u(2:m+1,3:m+2))/h^2;
% Sparse Laplacian matrix
I=speye(m,m);
E=sparse(2:m,1:m-1,1,m,m);
D=E+E'-2*I;
A=kron(D,I)+kron(I,D);
A=A/h^2;
F=reshape(F,m^2,1);
u_int=A\F;
u(2:m+1,2:m+1)=reshape(u_int,m,m);
```

We set $h = 0.001$ as our exact solution and the errors can be computed by comparing the values of $\min u_{ij}$. We test a range of h (0.0025, 0.05, 0.01, 0.5). Then we fit a line to the log(error)-log(h) plot and the slope will be the order of accuracy.

```
h=[0.0025 0.005 0.01 0.05];
[u0]=laplace_five(0.001); % exact solution
for i=1:4
    [u]=laplace_five(h(i));
    log_err(i)=log(min(min(u))-min(min(u0)));
    log_h(i)=log(h(i));
end
slope=polyfit(log_h,log_err,1);
```

We get a slope of 2.0562, thus we conclude that our algorithm achieves second order accuracy.

3. Now use the 9-point formula with the correction term described in Sec. 3.5 to solve the same problem as in the previous exercise. Again take $f(x, y) = x^2 + y^2$, and numerically test the order of accuracy of your code by solving on a fine grid, pretending that is the exact solution, and comparing coarser grid approximations to the corresponding values of the fine grid solution. [Note: You probably will not see the 4th-order accuracy described in the text. Can you explain why?]

Solution. We implement the 9-point Laplacian as `laplace_nine`. Since we know that $\nabla^2 f = 4$, we can hopefully achieve fourth order accuracy by removing the dominant error term.

```
function [u]=laplace_nine(h)
m=1/h-1; x=0:h:1; y=x';
u=zeros(m+2,m+2);
u(:,1)=1; u(:,m+2)=1; u(1,:)=1; u(1,:)=1; u(m+2,:)=1; % BC
f=@(x,y) x.^2+y.^2;
F=f(x(2:m+1),y(2:m+1));
F=F-4*(u(1:m,2:m+1)+u(3:m+2,2:m+1)+u(2:m+1,1:m)+u(2:m+1,3:m+2))/h^2;
F=F-(u(1:m,1:m)+u(1:m,3:m+2)+u(3:m+2,1:m)+u(3:m+2,3:m+2))/h^2;
F=F+(h^2/12)*4; % dominant error
% Sparse Laplacian matrix
E=ones(m,1);
S=spdiags([E 10*E E],[-1 0 1],m, m);
I=spdiags([-1/2*E E -1/2*E],[-1 0 1],m,m);
A=kron(I,S)+kron(S,I);
A=-A/h^2;
F=reshape(F,m^2,1);
u_int=A\F;
u(2:m+1,2:m+1)=reshape(u_int,m,m);
```

Using the same method in Q2

```
h=[0.005 0.0025 0.01 0.05];
[u0]=laplace_nine(0.001); % exact solution
for i=1:4
    [u]=laplace_nine(h(i));
    log_err(i)=log(min(min(u))-min(min(u0)));
    log_h(i)=log(h(i));
end
slope=polyfit(log_h,log_err,1);
```

We get a slope of 2.0709, thus it seems that the 9-point laplacian is second order accurate. For a fourth order accuracy, we require u to be sufficiently smooth such that $\nabla^4 u = \nabla^2(\nabla^2 u)$ exists. This is not true for this problem. We have $\nabla^2 u = x^2 + y^2 = x^2$ along the $(x, 0)$ boundary, but the boundary condition requires that $\nabla^2 u = u_{xx} + u_{yy} = 0$. Thus $\nabla^2 u$ is discontinuous.

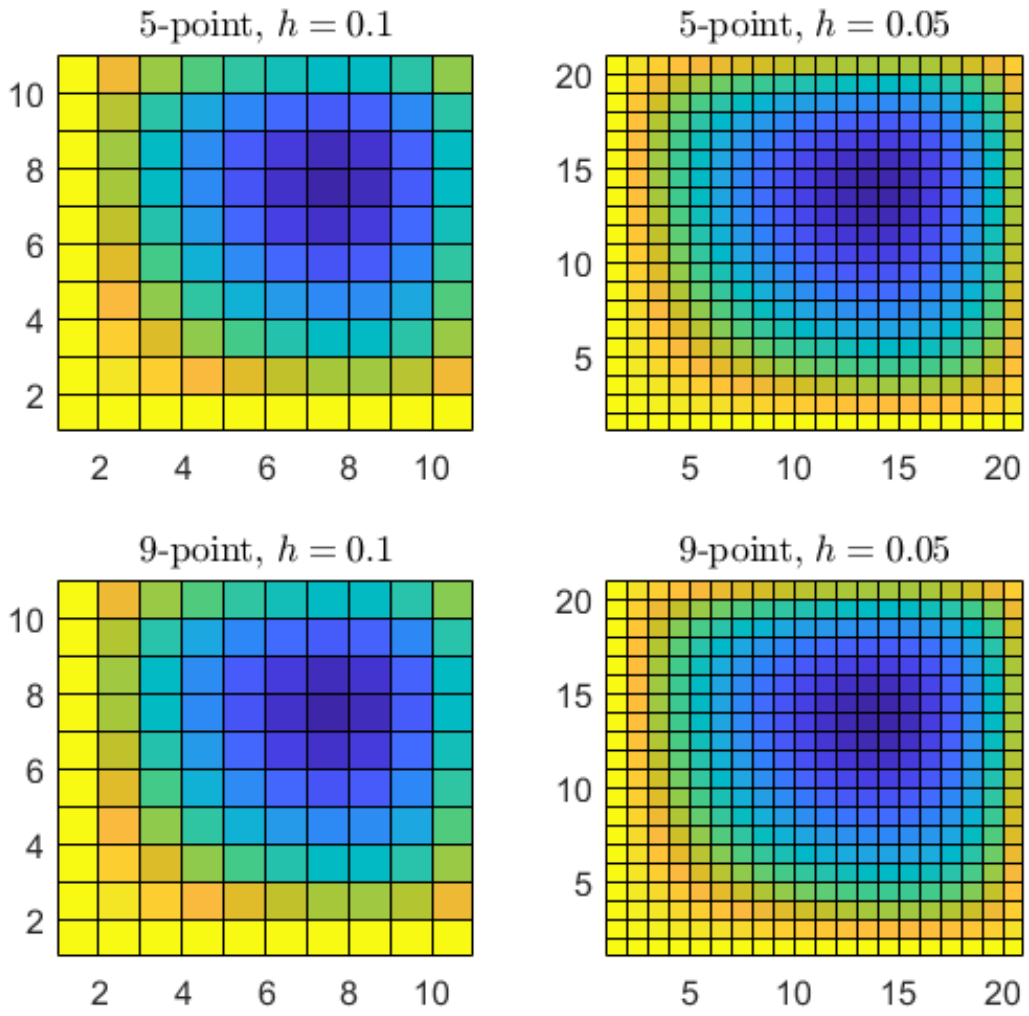

Figure 1: 5-point and 9-point stencil Laplacian for Poisson's equation

4. We have discussed using finite element methods to solve elliptic PDE's such as

$$\triangle u = f \quad \text{in } \Omega, \quad u = 0 \quad \text{on } \partial\Omega,$$

with *homogeneous* Dirichlet boundary conditions. How could you modify the procedure to solve the *inhomogeneous* Dirichlet problem:

$$\triangle u = f \quad \text{in } \Omega, \quad u = g \quad \text{on } \partial\Omega,$$

where g is some given function? Derive the equations that you would need to solve to compute, say, a continuous piecewise bilinear approximation for this problem when Ω is the unit square $(0,1) \times (0,1)$.

Solution. Finite element methods for homogeneous Dirichlet boundary conditions work as follows: we can approximate u by a function $\hat{u}$ from a space S, such that

$$\langle \mathcal{L}\hat{u}, \hat{v} \rangle = \langle f, \hat{v} \rangle \tag{1}$$

for all $\hat{v} \in S$. Here the inner prodcut $\langle \cdot, \cdot \rangle$ is defined by

$$\langle v, w \rangle \equiv \iint_\Omega v(x,y) w(x,y) dxdy$$

When evaluating $\langle \mathcal{L}\hat{u}, \hat{v} \rangle$, we use Green's theorem

$$\iint_\Omega \left(\hat{u}_{xx} + \hat{u}_{yy} \right) \hat{v} dxdy = - \iint_\Omega \left(\hat{u}_x \hat{v}_x + \hat{u}_y \hat{v}_y \right) dxdy + \int_{\partial\Omega} \hat{u}_{\mathbf{n}} \hat{v} d\gamma$$

Since the boundary conditions require that $\hat{v} = 0$ on $\partial\Omega$, the boundary term will vanish. We then approximate u as

$$\hat{u}(x,y) = \sum_{k=1}^{N} c_k \varphi_k(x,y)$$

where N is the number of interior nodes and $\varphi_1, \ldots, \varphi_N$ are continuous piecewise bilinear functions (as defined in the 'fem2d' notes), which form a basis of S. Then we can write (1) as

$$\langle \mathcal{L}\hat{u}, \varphi_\ell \rangle = \sum_{k=1}^{N} c_k \langle \mathcal{L}\varphi_k, \varphi_\ell \rangle = \langle f, \varphi_\ell \rangle, \quad \ell = 1, \ldots, N$$

which in matrix form is $A\mathbf{c} = \mathbf{f}$, where $A_{\ell k} = \langle \mathcal{L}\varphi_k, \varphi_\ell \rangle, \mathbf{c} = (c_1, \ldots, c_N)^T, \mathbf{f} = (\langle f, \varphi_1 \rangle, \ldots, \langle f, \varphi_N \rangle)^T$. If we use a uniform grid for both x and y directions, i.e. $h_x = 1/n_x$ and $h_y = 1/n_y$, then clearly $N = n_x n_y$. However, for the inhomogeneous case, the boundary term will not vanish and we need to increase the number of

our basis functions to include the boundary conditions. For a unit square , we need $M = (n_x + 2)(n_y + 2)$ functions. Then our new approximation to u is

$$\hat{u}(x, y) = \sum_{k=1}^{N} c_k \varphi_k(x, y) + \sum_{k=N+1}^{M} c_k \varphi_k(x, y)$$

where $c_k = g(x_k, y_k)$ for $k = N + 1, \ldots, M$ (note the basis functions for the boundary nodes will be defined in a different way to the interior nodes, since they only have 2 or 3 adjacent nodes instead of 4). We can keep A and $\mathbf{c}$ unchanged by subtracting the boundary term from $\mathbf{f}$, then the new equation has the form

$$\langle \mathcal{L}\hat{u}, \varphi_\ell \rangle = \langle f, \varphi_\ell \rangle - \sum_{k=N+1}^{M} c_k \langle \mathcal{L}\varphi_k, \varphi_\ell \rangle , \quad \ell = 1, \ldots, M$$