18.085, PROBLEM SET 1 SOLUTIONS

Question 1. (20 pts.)

Using odd values of n, the error is on the order of *machine precision*, about 10^{-16}, for smaller values of n. We expect this because we are doing finite differences on a quadratic (the true solution is a parabola), and we know finite differences are exact on quadratics. We do not get exqctly a 0 error because of how decimals are truncated inside the computer.

You should have noticed as well that using even values of n was not appropriate here. Indeed, when we are comparing our approximate solution to the true solution, we are comparing at the specific point $x_{\text{test}} = 1/2$. Of course, we can evaluate our true solution there. However, we can only compare that to the approximate solution at $x = 1/2$ if that point $x = 1/2$ is one of our mesh points. In other words, we need $1/2$ (or x_{test} in general) to be an integer multiple of $h = 1/(n+1)$. Then it is a fair comparison to look at the difference between the true solution at $x_{\text{test}} = 1/2$ and the approximate solution at $x_{\text{test}} = \frac{n+1}{2}h = 1/2$ (or entry "`xtest/h`" of vector u in Matlab).

Optional: If you tried larger values of n, say bigger than 1000, you might have seen the error become a bit larger, say on the order of 10^{-13} (still very small). This is because the matrix K we are "inverting" (or solving the system with) is somewhat *ill-conditioned.* As we increase n, this K amplifies more and more the errors already caused by truncation into something worse. We will come back to ill-conditioning when we discuss singular values later.

Question 2. (20 pts.) This time we start with the solution $u(x) = \sin(\pi x)$, see in a) which boundary conditions it satisfies, and in b) what would be the f in $f = -u''$ such that the given $u(x)$ is a solution of that equation. We can then use this $u(x)$ as our true solution.

a) Since $u(x) = \sin(\pi x)$, we have $u(0) = \sin(\pi 0) = 0$ and $u(1) = \sin(\pi 1) = 0$. This is again the fixed-fixed case.
b) $f(x) = -u'' = -\frac{d}{dx}(\pi \cos(\pi x)) = -(-1)\pi^2 \sin(\pi x) = \pi^2 \sin(\pi x)$.
c) Here is file "`pset1_2.m`". Lines 9 and 13 were modified to account, respectively, for a new right-hand side "`f`" and a new true solution "`ut`".

```
% pset1_2.m: I encourage you to try all these commands in the command line,
% without the semi-colons, to see what they print out

n=3;
h=1/(n+1); % step size
xtest=1/2; % value of x from 0 to 1 where we test the error

K=toeplitz([2 -1 zeros(1,n-2)]); % fixed-fixed
x=h:h:1-h;f=(pi^2)*sin(pi*x'); % right-hand side

u=(1/h^2)*K\f; % solution

ut=sin(pi*xtest); % true solution at test point

e=abs(u(round(xtest/h))-ut) % error at test point
```

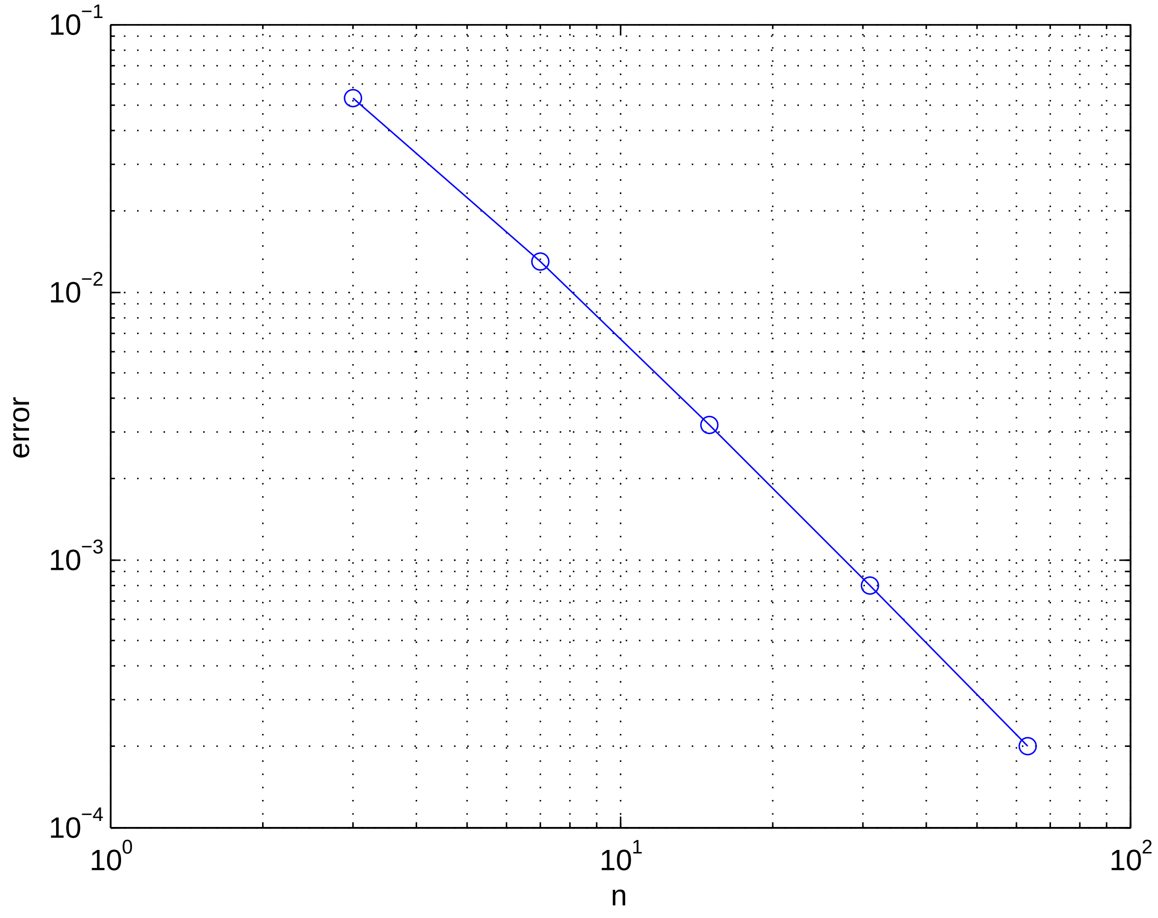
d) We expect the error to behave like h^2. This means that if I double n, the error should go down by a factor of 4. Trying values of n of 3, 7, 15, 31 and 63, we obtain the following errors, which clearly are divided by about 4 each time: 0.0530, 0.0130, 0.0032, $8.0358e-04$, $2.0082e-04$.

Optional: you did not have to do this, but we will be using these plots in later problem sets, so you might as well read this now!

The better way of checking that we get second order convergence is to use a loglog plot. In Matlab, do the following:

```
ns=[3 7 15 31 63];
es=[ 0.0530 0.0130 0.0032 8.0358e-04  2.0082e-04];
loglog(ns,es,'-o');grid on;
xlabel('n');ylabel('error');
```

You should obtain something similar to this figure, where we can see a slope of about -2, confirming that the order of convergence of the method is 2.



Question 3. (20 pts.)

Here is "`pset1_3.m`", using the first-order approximation to the free boundary condition:

```
% pset1_3.m: I encourage you to try all these commands in the command line,
% without the semi-colons, to see what they print out

n=3;
h=1/(n+1); % step size
xtest=0; % value of x from 0 to 1 where we test the error

T=toeplitz([2 -1 zeros(1,n-2)]); T(1,1)=1;% free-fixed
f=ones(n,1); % right-hand side

u=(1/h^2)*T\f; % solution

ut=-(1/2)*(xtest^2)+1/2; % true solution at test point

e=abs(u(1)-ut) % error at test point, note this is only valid when xtest=0 or h
```

a) Line 6 has changed to reflect now "`xtest=0`", and line 13 to reflect the new true solution, "`ut=-(1/2)*(xtest^2)+1/2`".
b) Line 8 has changed, we have T instead of K, and so we need T instead of K in line 11 as well. If you did not understand why T instead of K, come and see me and I'll happily explain (typing this up would be quite hard, but you can look at the book pages 19-20 too, or see OpenCourseWare (OCW) lecture video 2 online).
c) Line 15 now has $u(1)$ (which in Matlab means the first entry of vector u, or u_1 as we used in class, the approximation to the solution at $x = h$). This is correct because we have that $u_0 = u_1$ by the first-order approcimation to the free boundary condition. Other variations on line 15 which would be equivalent are ok as well.
d) Repeating the same error analysis as before: trying values of n of 3, 7, 15, 31 and 63, we obtain the following errors, which clearly are divided by ONLY 2 each time: 0.1250, 0.0625, 0.0312, 0.0156, 0.0078. Hence we obtain only first order convergence, as expected from our poor treatment of the boundary condition. The loglog graph (optional, see next page) shows this as well, with a slope of only -1 this time.

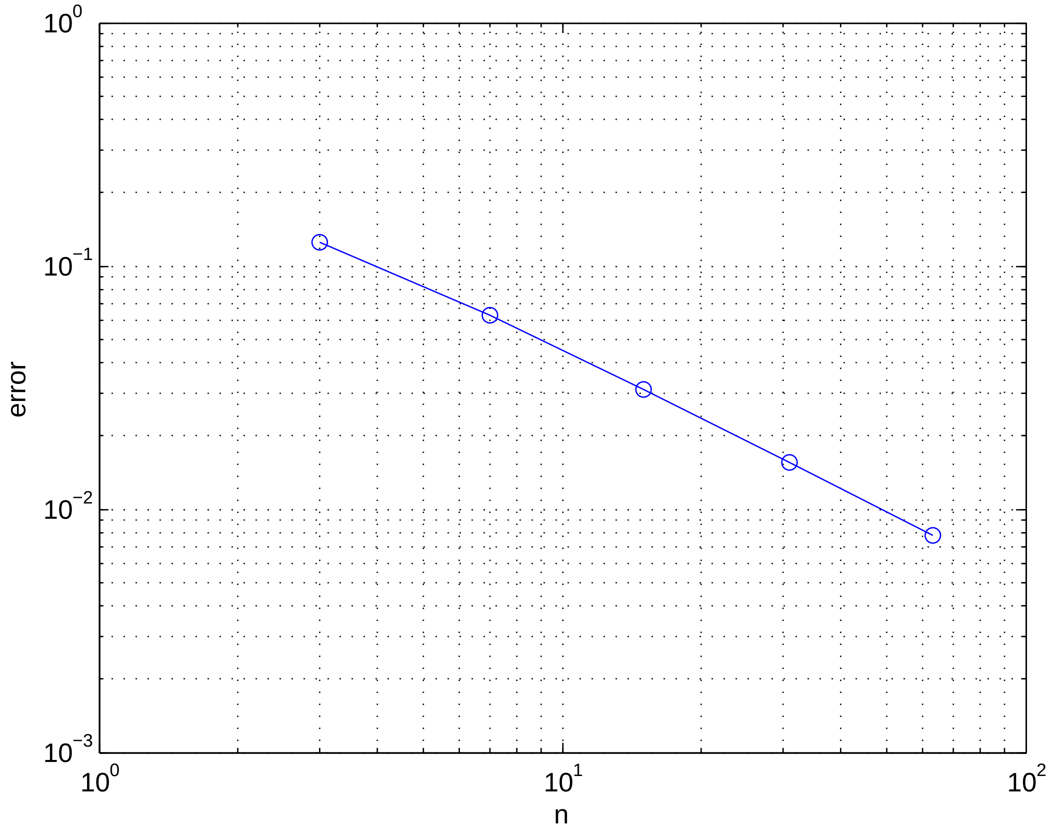
Question 4. (20 pts.)

Here is "`pset1_4.m`", using the second-order approximation to the free boundary condition:

```
% pset1_4.m I encourage you to try all these commands in the command line,
% without the semi-colons, to see what they print out

n=3;
h=1/(n+1); % step size
xtest=0; % value of x from 0 to 1 where we test the error

T=toeplitz([2 -1 zeros(1,n-1)]); T(1,1)=1; % free-fixed, ONE MORE UNKNOWN
f=ones(n+1,1); % right-hand side, ONE MORE UNKNOWN
```



```
f(1)=f(1)/2; % fix right-hand side for second order accuracy at boundary
u=(1/h^2)*T\f; % solution

ut=-(1/2)*(xtest^2)+1/2; % true solution at test point

e=abs(u(round(xtest/h)+1)-ut) % error at test point
```

a) Vector f has one more entry in it, because there is one more unknown in vector u, that is, the value at $x = 0$.
b) T is now $(n+1)$ by $(n+1)$, to reflect the additional unknown we have.
c) The first entry of f needs to be divided by 2. Again, if you did not understand the procedure, ask me, or see in the book pages 21-22 and lecture 2 in OCW.
d) Now we can use "`u(round(xtest/h)+1)`" to index the vector u in all generality (so you can change "`xtest`" and the code will still work).
e) Now we are back to the situation of Problem 1 where finite differences are exact, up to machine precision, on quadratics. Much better than the first order convergence of Problem 3!

Note. Some of you did not add another unknown. You might think this would not be such a big problem, since the structure of T and f remain the same. However, if you do not add an unknown when the method demands it, it is as if you are using too large an n. That is, your n in the code might be 5, but then T is 5 by 5, and T understands this as meaning "There are 4 unknowns inside the domain and 1 unknown at the boundary $x = 0$". Thus for this T we need to have a step size $h = 1/5$, corresponding to 4 unknowns inside the domain. But your n is 5, so your $h = 1/6$, which is smaller than what it should be when you multiply T by $1/h^2$. So instead, what we observe is that it is as if you were using the first-order approximation from Q3, but on an f which is one everywhere EXCEPT at $x = h$, where it is $1/2$. And so we have first-order convergence only, AND we converge to the solution coresponding to this wrong f. That's pretty bad! Again, boundary conditions are probably the hardest thing in finite differences to get, so we'll practice them more, but make sure you understand this.

The following questions test your understanding of linear algebra.

Question 5. (10 pts.) Elimination practice.

a) Carry out by hand the elimination of the circulant matrix C_4 to reach an upper triangular U and lower triangular L:

$$\begin{pmatrix} 2 & -1 & 0 & -1 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ -1 & 0 & -1 & 2 \end{pmatrix} \xrightarrow[R_4 \to R_4 - (-1)/(2)R_1]{R_2 \to R_2 - (-1)/(2)R_1} \begin{pmatrix} 2 & -1 & 0 & -1 \\ 0 & 3/2 & -1 & -1/2 \\ 0 & -1 & 2 & -1 \\ 0 & -1/2 & -1 & 3/2 \end{pmatrix}, \; l_{21} = l_{41} = -1/2, l_{31} = 0$$

$$\xrightarrow[R_4 \to R_4 - (-1/2)/(3/2)R_2]{R_3 \to R_3 - (-1)/(3/2)R_2} \begin{pmatrix} 2 & -1 & 0 & -1 \\ 0 & 3/2 & -1 & -1/2 \\ 0 & 0 & 4/3 & -4/3 \\ 0 & 0 & -4/3 & 8/6 \end{pmatrix}, \; l_{32} = -2/3, l_{42} = -1/3$$

$$\xrightarrow{R_4 \to R_4 - (-4/3)/(4/3)R_3} \begin{pmatrix} 2 & -1 & 0 & -1 \\ 0 & 3/2 & -1 & -1/2 \\ 0 & 0 & 4/3 & -4/3 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \; l_{43} = -1.$$

This means $L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -1/2 & 1 & 0 & 0 \\ 0 & -2/3 & 1 & 0 \\ -1/2 & -1/3 & -1 & 1 \end{pmatrix}$ and $U = \begin{pmatrix} 2 & -1 & 0 & -1 \\ 0 & 3/2 & -1 & -1/2 \\ 0 & 0 & 4/3 & -4/3 \\ 0 & 0 & 0 & 0 \end{pmatrix}$.

You should then make sure these work by multiplying them together! To confirm the result using Matlab, we can type "`C=toeplitz([2 -1 0 -1])`" to define matrix C_4. Then we do "`[L U]=lu(C)`" to compute and print out the factors L and U.

b) The last pivot of U is 0 because the matrix C is non-invertible.

c) The last column of U has new non-zeros. This is because, when we eliminate, we subtract multiples of rows above. In particular, the first row has a non-zero in entry $(1, 4)$ the top-right corner, which will force a non-zero in entry $(2, 4)$ as well. And the $(1, 3)$ entry of C_4 remains 0 in U (no fill-in there), because it is in the top row.

Question 6. (10 pts.) Multiplication practice.

By rows:

$$\begin{pmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} \text{inner prod. using row 1} \\ \text{inner prod. using row 2} \\ \text{inner prod. using row 3} \end{pmatrix} = \begin{pmatrix} 1 \times 1 + 1 \times (-1) + 1 \times 0 \\ 1 \times (-1) + 1 \times 2 + 1 \times (-1) \\ 1 \times 0 + 1 \times (-1) + 1 \times 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

By columns:

$$\begin{pmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \left(\text{comb. of columns}\right) = 1 \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix} + 1 \begin{pmatrix} -1 \\ 2 \\ -1 \end{pmatrix} + 1 \begin{pmatrix} 0 \\ -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$