OCE680 Assignment 1, Instructor's notes

1: First derivative matrix [10]

(a,b)

$$f_1{}' = \frac{-3f_1 + 4f_2 - f_3}{2\Delta} + \frac{1}{3}\Delta^2 f_1{}''' + ..., \tag{14.1.13}$$

and similar for $f_N{}'$. Everyone got the coefficients right and showed that the error is $\sim \Delta^2$.

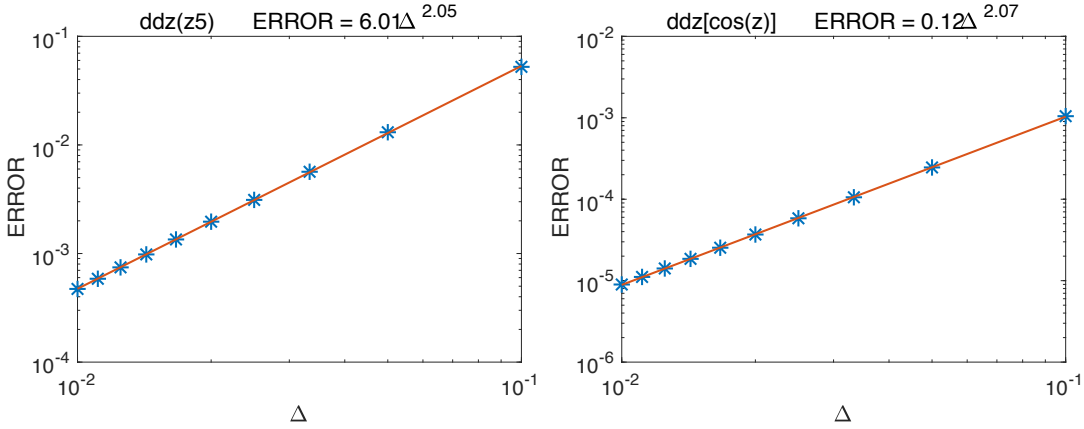

Figure 14.2: Approximating the first derivative of $f = z^5$ and $f = \cos z$. The error is the absolute difference between the approximation and the true value. Orange line = linear fit.

(c,d) Numerical results: The error test shows that the error $\propto \Delta^2$ in every case except $f = z^2$ (figures 14.2,14.2).

These calculations have two sources of error: *truncation error* and *roundoff error*. Truncation error results from the fact that we *truncate* the Taylor series expansions when designing our finite difference formulas. If we could somehow use the whole, infinite Taylor series there would be no truncation error. Roundoff error occurs because the computer can only store real numbers up to a certain number of significant digits, i.e., it must round them off. In Matlab, try calculating $2^{1/2} - 4^{1/4}$. You should get zero, but you don't. (I get 2.2204e-16.) That's roundoff error.

Usually, truncation error is much larger than roundoff error. But when we approximate the derivative a low-order polynomial like z^2, the terms that we neglected (truncated) are all zero; hence, so is the truncation error. In that case only roundoff error remains. Roundoff error is basically a stream of very small random numbers

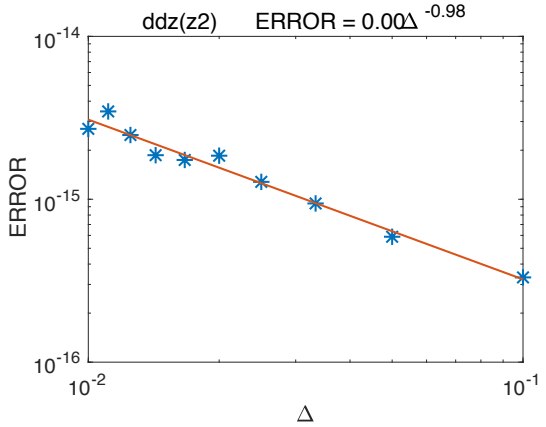

Figure 14.3: Approximating the first derivative of $f = z^2$.

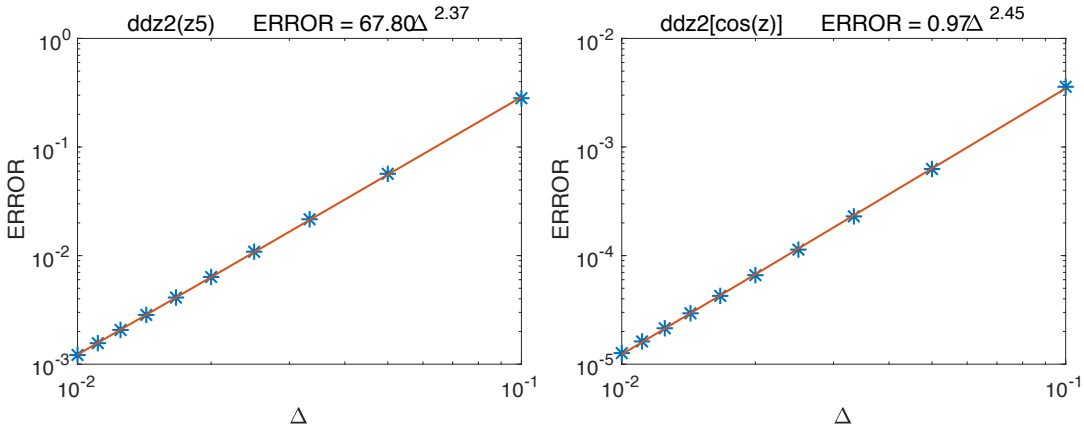

Figure 14.4: Approximating the second derivative of $f = z^5$ and $f = \cos z$. The error is the absolute difference between the approximation and the true value. Orange line = linear fit.

that all get divided by Δ in the process of computing $D_{ij}f_i$. The result is, on average, proportional to $1/\Delta$. That's what you see in figure 14.3.

2: Second derivative matrix [10]

(a) $f_i'' = \frac{f_{i-1}-2f_i+f_{i+1}}{\Delta^2} + \frac{1}{12}\Delta^2 f_1'''' + \ldots$

$f_1'' = \frac{2f_1-5f_2+4f_3-f_4}{\Delta^2} + \frac{11}{12}\Delta^2 f_1'''' + \ldots$ and similar for f_N''.

(b) My version of ddz2 is below. Be sure to include comments! Your ddz and ddz2 routines are valuable; you'll be using them constantly. It's worthwhile to keep them organized and to upgrade them if you see an opportunity.

```
function d=ddz2(z)
% Second derivative matrix for independent variable z.
% 2nd order centered differences, with 1-sided derivatives at the boundaries.
% z is assumed to be equally spaced.

% check for equal spacing
if abs(std(diff(z))/mean(diff(z)))>.000001
    disp(['ddz2: values not evenly spaced!'])
    d=NaN;
    return
end

del=z(2)-z(1);N=length(z);

d=zeros(N,N);
for n=2:N-1
    d(n,n-1)=1.;
    d(n,n)=-2.;
    d(n,n+1)=1.;
end
d(1,1)=2;d(1,2)=-5;d(1,3)=4;d(1,4)=-1;
d(N,N)=2;d(N,N-1)=-5;d(N,N-2)=4;d(N,N-3)=-1;
d=d/del^2;
```

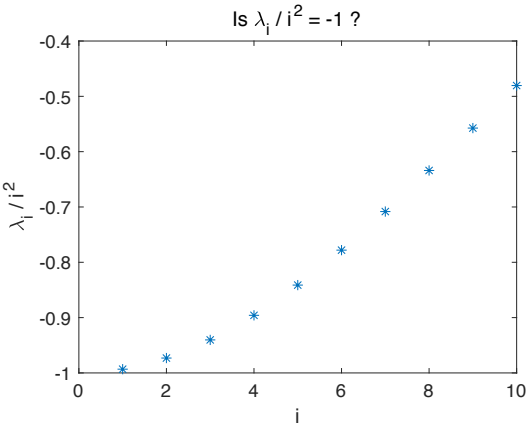

Figure 14.5: Eigenvalues of the differential eigenvalue problem (A.0.1), scaled by i^2 so that the analytical result is -1.

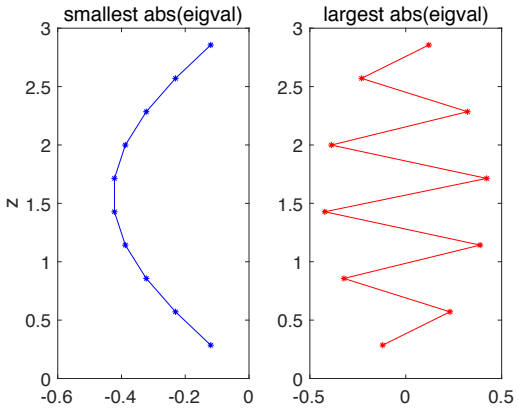

Figure 14.6: Eigenvectors corresponding to the smallest and largest (absolute) eigenvalues.

```
return
end
```

For the cases shown in figure 14.4, accuracy is actually a bit better than 2nd order; i.e. the exponent is >2.

3: Differential eigenvalue problem [10]

(a) Eigenvalues of the differential equation are $-i^2; i = 1,2,...,N$.

(b) When you solve the matrix eigenvalue problem, the first few eigenvalues are close to -1, -4, âĂę, but things degenerate as you go to higher eigenvalues. This is because the approximate derivative in ddz2 is only accurate for functions that are smooth enough to be well-resolved by the number of z values you've used. To be more specific, the truncation error is approximately proportional to $\Delta^2 f^{(4)}$. If there's a lot of small-scale structure, the fourth derivative can make this error large even when Δ is small. Also, higher-order terms can contribute to the error.

For N=10, I get the eigenvalues shown in figure 14.5. These eigenvalues have been sorted and normalized by i^2 so that each would equal -1 if the solution were perfectly accurate. The first few eigenvalues are fairly accurate, but things degenerate fast after that. The final eigenvalue is wrong by a factor of two!

The eigenvector corresponding to $i = 1$ ($\sigma = -0.993$) is shown on the left in figure 14.5. It is clearly well resolved by the ten z values. Note that it is headed for zero at $z = 0$ and $z = \pi$, i.e. it obeys the boundary conditions.

On the right is the eigenvector corresponding to $i = 10$. This function is very badly resolved, i.e. there is too much small-scale structure to be accurately represented by ten points, and the truncation error in ddz2 is

expected to be large. This is why the eigenvalue was inaccurate.

The moral of the story is, always keep in mind that the differential eigenvalue problem that you want to solve and the algebraic eigenvalue problem that you actually do solve are two different things. A well-designed algebraic equation should give good approximations to solutions of the corresponding differential equation, but you must always interpret the results with care. If the function you get is not well resolved by your spatial discretization (i.e. if the scale on which it varies is not much bigger than Δ, as in the red curve shown above), it is probably not a good approximate solution to the differential equation.

Here is the code I used:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% HMWK 1 Problem 3
%
clear
close all
direc='/Users/smyth/Dropbox/HOME/projects/Book_Instability/figures';
fs=20;

% define z values
N=10;
z=pi*[1:N]'/(N+1);

% compute derivative matrix
d=ddz2(z);
dz=z(2)-z(1);

fname='1SIDE'
%%%
% To use 1-sided derivatives, comment out the next three lines.
d(N,:)=0;d(N,N-1)=1/dz^2;d(N,N)=-2/dz^2;
d(1,:)=0;d(1,1)=-2/dz^2;d(1,2)=1/dz^2;
fname = 'DEP';
%%%

% compute eigvals & eigvecs
[v ee]=eig(d);e=diag(ee);

% sort
[~,ind] =sort(abs(e),'ascend');
e=e(ind);
v=v(:,ind);

% Plot eigenvalues /i^2
% If the eigfn is well-resolved, this will be close to -1.
figure
plot([1:N],e./[1:N].^2','*','markersize',10)
xlabel('i','fontsize',fs);
ylabel('\lambda_i / i^2','fontsize',fs)
title('Is \lambda_i / i^2 = -1 ?','fontsize',fs,'fontweight','normal')
set(gca,'fontsize',fs-2)

figname=[direc 'hmwk1' fname 'eigvals'];
print('-dpdf',figname)
figname=[direc '/hmwk1' fname 'eigvecs'];
```

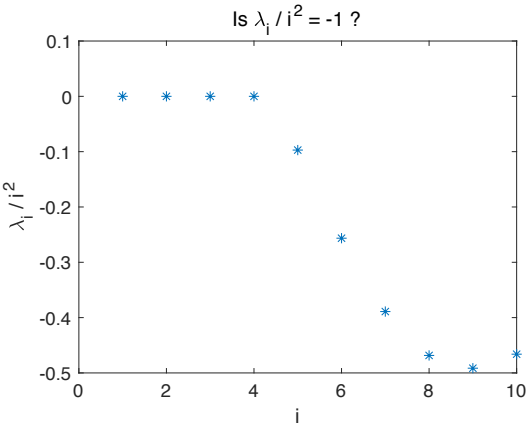

Figure 14.7: Eigenvalues using 1-sided derivatives at the boundaries.

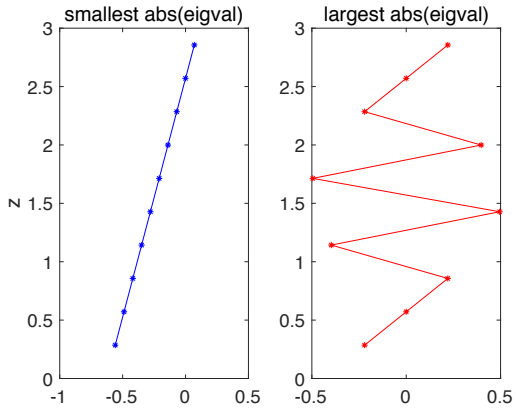

Figure 14.8: Eigenvectors using 1-sided derivatives at the boundaries.

```
print('-dpdf',figname)


% Plot first and last eigvecs.
% The first is well-resolved, and its eigval is close to -i^2.
% The last is poorly-resolved, and the eigval is not close to -i^2.
figure
subplot(1,2,1)
plot(v(:,1),z,'b*')
hold on
plot(v(:,1),z,'b')
ylabel('z','fontsize',fs)
title('smallest abs(eigval)','fontsize',fs,'fontweight','normal')
set(gca,'fontsize',fs-2)

subplot(1,2,2)
plot(v(:,end),z,'r*')
hold on
plot(v(:,end),z,'r')
title('largest abs(eigval)','fontsize',fs,'fontweight','normal')
set(gca,'fontsize',fs-2)
```

Replacing the boundary conditions with one-sided derivatives gives utter nonsense. The eigenvalues are nothing like $-i^2$. Looking at the eigenvectors, one can see why - they do not obey the boundary conditions.

Marking notes: I don't dock marks for errors I judge to be trivial, though I do point them out. I try to be generous if I suspect that the question was not phrased clearly enough. Assignments are due at the beginning of class, not the end. In case you're curious, this is how I assigned letter grades the last time I taught this course:
A = 93 or higher
A- = 90
B+ = 87
B = 80
B- = 77
Don't hesitate to speak to me if you have any concerns about your mark.