COMPUTATIONAL PHYSICS 430

INSTRUCTOR MANUAL

Ross L. Spencer and Michael Ware

Department of Physics and Astronomy

Brigham Young University

This manual contains Matlab scripts that solve the assigned exercises in Physics 430. Please contact us if you find errors or if you have suggestions for improving the course or these solutions.

Chapter 0

Review

0.1 How to teach this lab

Many students have difficulty writing loops that work. This lab will quickly expose the gap between those who already know how to program and those who need to learn it. The ones who struggle with this lab will need extra help throughout the semester.

This lab used to be the main part of the first lab. I (Michael Ware) took these problems out of the main sequence because I found that reviewing everything in a block like this didn't seem to help down the road in the labs where they needed these skills. I try to review these things as they come up in the course of the labs. Others may still like the big review, so I've left it in the manual.

0.2 Solutions

Problem 0.1(a)

Listing 1 (lab0p1a.m)

```
% Lab Problem 0.1a, Physics 430

for n=2:3:101
    if mod(n,5)==0
        fprintf(' fiver: %g \n',n);
    end
end
```

Problem 0.1(b)

Listing 2 (lab0p1b.m)

```
% Lab Problem 0.1b, Physics 430

N=input(' Enter N - ')
```

```
sum=0;
for n=1:N
    sum=sum+n;
end

fprintf(' Sum = %g  Formula = %g \n',sum,N*(N+1)/2);
```

Problem 0.1(c)

Listing 3 (lab0p1c.m)

```
% Lab Problem 0.1c, Physics 430

x=input(' Enter x - ')

sum=0;

for n=1:1000
    sum=sum + n*x^n;
end

fprintf(' Sum = %g  Formula = %g \n',sum,x/(1-x)^2);
```

Problem 0.1(d)

Listing 4 (lab0p1d.m)

```
% Lab Problem 0.1d, Physics 430

sum=0; n=0;

term=1;

while term > 1e-8
    n=n+1;
    term=n*x^n;
    sum=sum+term;
end
fprintf(' Sum = %g  Formula = %g \n',sum,x/(1-x)^2);
```

Problem 0.1(e)

Listing 5 (lab0p1e.m)

```
% Lab Problem 0.1e, Physics 430

% initialize the counter n, the term to be added, and the sum
n=1; term=1.; sum=term;
```

```
while term>1e-6
    n=n+1;
    term=1/n^2;
    sum=sum+term ;
end

fprintf(' Sum/(pi^2/6) = %g \n',sum*6/pi^2);
```

Problem 0.1(f)

Listing 6 (lab0p1f.m)

```
% Lab Problem 0.1f, Physics 430

prod=1;

for n=1:100000
    term=1+1/n^2;
    prod=prod*term;
    if term-1 < 1e-8
        break
    end
end

fprintf(' Product = %g  Formula = %g \n',prod,sinh(pi)/pi);
```

Problem 0.1(g)

Listing 7 (lab0p1g.m)

```
% Lab Problem 0.1g, Physics 430

x1=1;x2=1;x3=1;

chk=1;

while chk > 1e-8

    x1new=exp(-x1);
    x2new=cos(x2);
    x3new=sin(2*x3);

    chk=max([abs(x1new-x1),abs(x2-x2new),abs(x3-x3new)]);

    x1=x1new;
    x2=x2new;
    x3=x3new;

end
```

```
fprintf(' x1 = %g x2 = %g  x3 = %g \n',x1,x2,x3);
```

Chapter 1

Grids and Numerical Derivatives

1.1 How to teach this lab

This lab has a lot of reading, but the concepts aren't extremely difficult. If you can email the students and ask them to read this lab before class, it helps keep things moving.

The students will be unfamiliar with grids, so a mini-lecture at the beginning would be helpful. The students may not understand roundoff, so it may be helpful to do a long-hand subtraction on the board to emphasize the point. Some discussion of the error figure is also helpful.

1.2 Solutions

Problem 1.1(a)

Listing 1.1 (lab1p1a.m)

```
% Lab Problem 1.1(a), Physics 430
clear; close all;

N=100;
a=0;
b=pi;
h=(b-a)/(N-1);
x=a:h:b;

y=sin(x).*sinh(x);

plot(x,y);
```

Problem 1.1(b)

Listing 1.2 (lab1p1b.m)

```
% Lab Problem 1.1(b), Physics 430
clear; close all;

N = 5000;
a = 0;
b = 2;
h = (b-a)/N;
x = a+h/2:h:b-h/2;
y = cos(x);

integral = sum(y)*h

% exact integral is sin(2)
error = integral - sin(2)
```

Problem 1.1(c)

Listing 1.3 (lab1p1c.m)

```
% Lab Problem 1.1(b), Physics 430
clear; close all;

N = 5000;
a = 0;
b = pi/2;
h = (b-a)/(N-2);
x = a-h/2:h:b+h/2;
f = sin(x);

firstpts = f(1:2)
lastpts = f(end-1:end)
```

Problem 1.2(a)

Halfway between, linear interpolation predicts

$$\frac{y_1 + y_2}{2},$$

i.e., the average.

Problem 1.2(b)

3/4 of the way from x_1 to x_2 we get

$$\frac{1}{4}y_1 + \frac{3}{4}y_2$$

Use 3/4 for the nearest point and 1-3/4 for the other point.

Problem 1.2(c)

$$y(x_2+h) = 2y_2 - y_1$$

and

$$y(x_2+h/2) = 3y_2/2 - y_1/2.$$

Problem 1.3(a)

$$\frac{3}{8}y_1 + \frac{3}{4}y_2 - \frac{1}{8}y_3$$

and

$$-\frac{1}{8}y_1 + \frac{3}{4}y_2 + \frac{3}{8}y_3$$

Use 3/4 in the middle, 3/8 for the other near point and -1/8 for the more distant point.

Problem 1.3(b)

$$y(x_3+h) = 3y_3 - 3y_2 + y_1$$

and

$$y(x_3+h/2) = 15y_3/8 - 5y_2/4 + 3y_1/8.$$

Problem 1.4

Listing 1.4 (lab1p4.m)

```
% Lab Problem 1.4, Physics 430
clear; close all;

xmin=0;
xmax=5;
N=100;
h=(xmax-xmin)/(N-1);
x=xmin:h:xmax;

f=besselj(0,x);

fp(2:N-1)=(f(3:N)-f(1:N-2))/(2*h);
fpp(2:N-1)=(f(3:N)-2*f(2:N-1)+f(1:N-2))/h^2;

% fp(1)=2*fp(2)-fp(3);
% fp(N)=2*fp(N-1)-fp(N-2);
% fpp(1)=2*fpp(2)-fpp(3);
% fpp(N)=2*fpp(N-1)-fpp(N-2);

fp(1)=3*fp(2)-3*fp(3)+fp(4);
fp(N)=3*fp(N-1)-3*fp(N-2)+fp(N-3);
fpp(1)=3*fpp(2)-3*fpp(3)+fpp(4);
fpp(N)=3*fpp(N-1)-3*fpp(N-2)+fpp(N-3);
```

```
figure
subplot(2,1,1)
plot(x,fp,'g*',x,-besselj(1,x),'b-');
title(' J_0''(x)')
subplot(2,1,2);
plot(x,fpp,'g*',x,-besselj(0,x)+besselj(1,x)./x,'b-');
title(' J_0''''(x)')
```

Problem 1.5

(a) $f''''(x_0)h^2/12$
(b) $f''''''(x_0)h^2/8$

Problem 1.6

Listing 1.5 (lab1p6.m)

```
% Lab Problem 1.6, Physics 430
clear; close all;

h = [0.1 0.01 0.001];

forward = (exp(h)-1) ./ h;
center = (exp(h) - exp(-h)) ./ (2*h);
second = (exp(h) - 2 + exp(-h)) ./ h.^2;

ferr = abs(forward - 1)
cerr = abs(center - 1)
serr = abs(second - 1)
```

Output:

```
ferr = 5.1709e-002  5.0167e-003  5.0017e-004

cerr = 1.6675e-003  1.6667e-005  1.6667e-007

serr = 8.3361e-004  8.3334e-006  8.3407e-008
```

For the forward derivative with our h, we predict errors of

```
h/2 =   5.0000e-002  5.0000e-003  5.0000e-004
```

For the centered derivative with our h, we predict errors of

```
h.^2/6  =  1.6667e-003  1.6667e-005  1.6667e-007
```

which agrees very nicely with those found in the problem.

Problem 1.7

Listing 1.6 (lab1p7.m)

```
% Lab Problem 1.7, Physics 430
clear;close all;

xmin=0;
xmax=5;
N=1000;
h=(xmax-xmin)/(N-1);
x=xmin:h:xmax;

f=cos(x)+.001*rand(1,length(x));

fp(2:N-1)=(f(3:N)-f(1:N-2))/(2*h);
fpp(2:N-1)=(f(3:N)-2*f(2:N-1)+f(1:N-2))/h^2;

fp(1)=3*fp(2)-3*fp(3)+fp(4);
fp(N)=3*fp(N-1)-3*fp(N-2)+fp(N-3);
fpp(1)=3*fpp(2)-3*fpp(3)+fpp(4);
fpp(N)=3*fpp(N-1)-3*fpp(N-2)+fpp(N-3);

figure
subplot(2,1,1)
plot(x,fp,x,-sin(x));
title('First Derivative with Random Error');
subplot(2,1,2);
plot(x,fpp,x,-cos(x))
title('Second Derivative with Random Error');
```

Chapter 2

Differential Equations with Boundary Conditions

2.1 How to teach this lab

The linear algebra in this lab will be a challenge to the students. I begin with a careful review on the board of how to interpret a linear differential equation as an operator equation, then turn the differential operator into a matrix. I then discuss how to load the top and bottom rows of the matrix with boundary conditions. The example at the beginning of 2.1 is simple: $y(0) = 0$ and $y(2) = 1$, so the operator is simple. They won't really learn how boundary conditions are coded in the top and bottom rows until Problem 2.3. It is very important that they spend the effort to understand the idea of translating a differential equation into a linear algebra problem because this problem comes up repeatedly in this course.

2.2 Solutions

Problem 2.1

Problem 2.1(c)

Listing 2.1 (lab2p1c.m)

```
% Lab problems 2.1(c), Physics 430
clear;close all;
% define the grid, making x a column vector

N=30;
xmin=0;
xmax=2;
h=(xmax-xmin)/(N-1);
x=xmin:h:xmax;
x=x';

% load the matrix representing the differential operator
```

```
% first load it with zeros, then load the non-zero elements
A=zeros(N,N);
% define the right-hand side b as a column vector
b=zeros(N,1);

% load the non-zero elements of A and b
for j=2:N-1
    A(j,j)=-2/h^2+9;
    A(j,j-1)=1/h^2;
    A(j,j+1)=1/h^2;
    b(j)=sin(x(j));
end

% load the top and bottom rows with boundary conditions
A(1,1)=1;b(1)=0;
A(N,N)=1;b(N)=1;
% do the solve and display it together with the exact solution
y=A\b;

yexact = -1/8*sin(3*x)*(-8+sin(2))/sin(6)+1/8*sin(x);

plot(x,yexact,'b-',x,y,'r.')
```

Problem 2.2(a)

Listing 2.2 (lab2p2a.m)

```
% Lab problems 2.2(a), Physics 430
clear;close all;

% define the grid, making x a column vector
N=30;
xmax=5;
h=xmax/(N-1);
x=0:h:xmax;
x=x';

% load the matrix representing the differential operator
% first load it with zeros, then load the non-zero elements
A=zeros(N,N);
% define the right-hand side b as a column vector
b=zeros(N,1);

% load the non-zero elements of A and b
for j=2:N-1
    A(j,j)=-2/h^2+(1-1/x(j)^2);
    A(j,j-1)=1/h^2-.5/h/x(j);
    A(j,j+1)=1/h^2+.5/h/x(j);
    b(j)=x(j);
end
% load the top and bottom rows with boundary conditions
A(1,1)=1;b(1)=0;
A(N,N)=1;b(N)=1;
```

```
% do the solve and display it together with the exact solution
y=A\b;
yexact=-4*besselj(1,x)/besselj(1,5)+x;
plot(x,y,'b-',x,yexact,'r*')
```

Problem 2.2(b)

Listing 2.3 (lab2p2b.m)

```
% Lab problems 2.2(b), Physics 430
clear;close all;

% define the grid, making x a column vector
N=200;
xmax=5;
h=xmax/(N-1);
x=0:h:xmax;
x=x';

% load the matrix representing the differential operator
% first load it with zeros, then load the non-zero elements
A=zeros(N,N);
% define the right-hand side b as a column vector
b=zeros(N,1);

% load the non-zero elements of A and b
for j=2:N-1
    A(j,j)=-2/h^2+exp(x(j));
    A(j,j-1)=1/h^2-.5/h*sin(x(j));
    A(j,j+1)=1/h^2+.5/h*sin(x(j));
    b(j)=x(j)^2;
end
% load the top and bottom rows with boundary conditions
A(1,1)=1;b(1)=0;
A(N,N)=1;b(N)=3;
% do the solve and display it
y=A\b;
plot(x,y,'b-')
```

Problem 2.3

Solution: The Matlab script comes first, followed by the Maple worksheet. When I compare the simple rule in (a) ($-y_{N-1}+y_N=0$) with the more complicated one in (b) ($y_{N-2}/(2h)-2y_{N-1}/h+3y_N/(2h)=0$) using 500 grid points the maximum difference between the computed result and the exact result in (a) is 7×10^{-5} while in (b) it is 3×10^{-6}, 25 times more accurate.

Listing 2.4 (lab2p3.m)

```
% Lab problem 2.3, Physics 430
```

```
% (a)
clear;close all;

% define the grid, making x a column vector
N=input(' Enter N - ');
xmax=2;
h=xmax/(N-1);
x=0:h:xmax;
x=x';

% load the matrix representing the differential operator
% first load it with zeros, then load the non-zero elements
A=zeros(N,N);
% define the right-hand side b as a column vector
b=zeros(N,1);

% load the non-zero elements of A and b
for j=2:N-1
    A(j,j)=-2/h^2+9;
    A(j,j-1)=1/h^2;
    A(j,j+1)=1/h^2;
    b(j)=x(j);
end
% load the top and bottom rows with boundary conditions
A(1,1)=1;b(1)=0;
% just force y(N-1)-y(N)=0 to approximately get the first
% derivative condition right
A(N,N)=1;A(N,N-1)=-1;b(N)=0;
% do the solve and display it together with the exact solution
y=A\b;
yexact=-sin(3*x)/27/cos(6)+x/9;
plot(x,y,'b-',x,yexact,'r*')
title('Part a')
% print the maximum error
erra=max(abs(y-yexact));
fprintf(' Maximum error in part (a): %g \n',erra)

% (b)

% Do the better method for the BC
A(N,N)=1.5/h;A(N,N-1)=-2/h;A(N,N-2)=.5/h;b(N)=0;

% do the solve and display it together with the exact solution
y=A\b;
yexact=-sin(3*x)/27/cos(6)+x/9;
figure
plot(x,y,'b-',x,yexact,'r*')
title('Part b')
% print the maximum error
errb=max(abs(y-yexact));
fprintf(' Maximum error in part (b): %g \n',errb)
```

Problem 2.4(b)

Listing 2.5 (lab2p4b.m)

```
% Lab problems 2.4(b), Physics 430
clear; close all;

% rewrite the equation in the form y''=1-sin(y) and iterate by
% using the previous y on the right and solving the modified ode
% above each iteration step to get the next approximate y(x)

N=501;h=3/(N-1);
x=0:h:3;

y=0*x';
for m=1:200

A=zeros(N,N);
b=zeros(N,1);

for j=2:N-1
    A(j,j)=-2/h^2;
    A(j,j-1)=1/h^2;
    A(j,j+1)=1/h^2;
    b(j)=1-sin(y(j));
end
A(1,1)=1;b(1)=0;
A(N,N)=1;b(N)=0;
yt=A\b;
y=1*yt+.0*y;
max(A*y-1+sin(y))
plot(x,y)

end
```

Chapter 3

The Wave Equation: Steady State and Resonance

3.1 How to teach this lab

They will have forgotten what they have been taught about the eigenvalue problem, and they will have no clue about what eigenvalues mean, so a mini-lecture on eigenvalues and eigenvectors at the start of class will help. The lab is designed to convince them that eigenvalues are parameter values that make amplitudes go to infinity, so make sure they understand that 3.1 and 3.2 are linked problems. You will also need to discuss the tricky business about the boundary conditions.

3.2 Solutions

Problem 3.1

Listing 3.1 (lab3p1.m)

```
% Lab Problem 3.1, Physics 430
clear;close all;

xmin=0;xmax=1.2;
N=input(' Enter N - ');
h=(xmax-xmin)/(N-1);
x=xmin:h:xmax;
x=x';

% define the driving force

for n=1:N
    if x(n) >= 0.8 & x(n) <= 1
        f(n)=0.73;
    else
        f(n)=0.;
    end
end
```

```
T=127;mu=.003;

w1=400;w2=1200;Nw=100;
dw=(w2-w1)/Nw;
wscan=w1:dw:w2;
Nw=length(wscan);

for m=1:Nw
    w=wscan(m);

% define the matrix
A=zeros(N,N);

% load the operator
r=zeros(N,1);
for n=2:N-1
   A(n,n-1)=T/h^2;
   A(n,n+1)=T/h^2;
   A(n,n)=-2*T/h^2+mu*w^2;
   r(n)=-f(n);
end
A(1,1)=1;r(1)=0;
A(N,N)=1;r(N)=0;

g=A\r;

plot(x,g,'LineWidth',2.5)
pause(.1)

Amp(m)=max(abs(g));
end

plot(wscan,Amp,'LineWidth',2.5);
xlabel('\omega');
ylabel('Amplitude');
title('Steady Amplitude vs. Driving Frequency')
```

Problem 3.2

Listing 3.2 (lab3p2.m)

```
% Lab Problem 3.2, Physics 430
clear;close all;

xmin=0;xmax=1.2;
N=input(' Enter N - ');
h=(xmax-xmin)/(N-1);
x=xmin:h:xmax;
x=x';

T=127;mu=.003;
```

```
% define the matrices A and B
A=zeros(N,N);
B=eye(N,N);

% load the A operator at interior points
for n=2:N-1
   A(n,n-1)=1/h^2;
   A(n,n+1)=1/h^2;
   A(n,n)=-2/h^2;
end
% now implement the boundary condition y(0)=0 and
% y(L)=0 in both A and B
A(1,1)=1;B(1,1)=0;
A(N,N)=1;B(N,N)=0;

% find the eigenvalues
[V,D]=eig(A,B);

% convert them to squared frequencies
w2raw=-(T/mu)*diag(D);

[w2,k]=sort(w2raw);

for n=1:N
    t=sprintf(' w^2 = %g w = %g w(exact) = %g ',...
      w2(n),sqrt(abs(w2(n))),(n-2)*pi*sqrt(T/mu)/xmax );
    gn=V(:,k(n));
    plot(x,gn,'b-')
    title(t);xlabel('x');ylabel('g(x)');
    pause(0.3)
end
```

Problem 3.3

Listing 3.3 (lab3p3.m)

```
% Lab Problem 3.3, Physics 430
clear;close all;

xmin=0;xmax=1.2;
N=input(' Enter N - ');
h=(xmax-xmin)/(N-1);
x=xmin:h:xmax;
x=x';

T=127;mu=.003;

% define the matrices A and B
A=zeros(N,N);
B=eye(N,N);

% load the A operator at interior points
for n=2:N-1
```

```
   A(n,n-1)=1/h^2;
   A(n,n+1)=1/h^2;
   A(n,n)=-2/h^2;
end
% now implement the boundary condition y(0)=0 and
% the two boundary conditions asked for in parts (a)
% and (b)

A(1,1)=1;B(1,1)=0;

% (a) y'(L)=0
%A(N,N)=1.5/h;A(N,N-1)=-2/h;A(N,N-2)=.5/h;B(N,N)=0;

% (b) y'(L)=2*y(L)
A(N,N)=1.5/h-2; % Change this line to go from (a) to (b)
A(N,N-1)=-2/h;
A(N,N-2)=.5/h;
B(N,N)=0;

% find the eigenvalues
[V,D]=eig(A,B);

% convert them to squared frequencies
w2raw=-(T/mu)*diag(D);

[w2,k]=sort(w2raw);

for n=1:N
    t=sprintf(' w^2 = %g w = %g w(exact) = %g ',...
        w2((n)),sqrt(abs(w2(n))),(2*n-5)/2*pi*sqrt(T/mu)/xmax );
    gn=V(:,k(n));
    plot(x,gn,'b-')
    title(t);xlabel('x');ylabel('g(x)');
    pause(0.3)
end
```

Chapter 4

The Hanging Chain and Quantum Bound States

4.1 How to teach this lab

Start the period by hanging a chain from the ceiling, measuring the frequencies of its first three modes with a stopwatch, and measuring its length with a meter stick. (Wayne has a semi-official chain that you can use.) Then estimate the frequency of the lowest mode by approximating it as a swinging rod (you will come close.) Then modify the operator and the boundary conditions in the eigenvalue code from 4.2 to do this problem and show that you get a better value for the lowest mode frequency, and check your results against the other measured modes as well. Do all of this as a class project with the instructor at the board and two of the best students in the class doing the coding. Make sure the other students understand what these two are doing. Then solve the differential equation with Maple, discard the singular solution, and show how requiring zero amplitude at the ceiling gives a formula for the mode frequencies in terms of the zeros of J_0. Also show that these mode frequencies agree with the Matlab eigenvalues. Try to move through this part as quickly as is reasonable because the students have 4.2 to do.

The students will have trouble with the idea of putting Schroedinger's equation in dimensionless form, so you will need to sketch for them how this works. Make sure nobody gets so hung up on this part that they can't get to the numerical computation.

4.2 Solutions

Problem 4.1

The vertical forces on a given bit of chain located at position x are balanced by definition when the chain is in vertical equilibrium. Thus the tension must be equal to the weight of the chain below point x. The mass of the chain below x is

$m = \mu x$, and the weight of the chain is $mg = \mu g x$.

Problem 4.2

Listing 4.1 (lab4p2.m)

```
% Lab Problem 4.2, Physics 430
clear;close all;

L=2;
xmin=0;xmax=L;
N=500;%input(' Enter N - ');
h=(xmax-xmin)/N;
x=-h/2:h:xmax+h/2;

g=9.8;

% define the matrices A and B
A=zeros(N+2,N+2);
B=A;
% load the operator
for n=2:N+1
   A(n,n-1)=x(n)/h^2-.5/h;
   A(n,n+1)=x(n)/h^2+.5/h;
   A(n,n)=-2*x(n)/h^2;
   B(n,n)=1;
end
A(1,1)=-1/h;A(1,2)=1/h;
B(1,1)=.5;B(1,2)=.5;

A(N+2,N+2)=.5;A(N+2,N+1)=.5;
B(N+2,N+2)=0 ;

[V,D]=eig(A,B);

w2raw=-g*diag(D);

[w2,k]=sort(w2raw);

% here is the exact frequency for the lowest mode
 exact=.5*2.4048*sqrt(g/L);
 fprintf(' Exact lowest mode frequency: %g \n',exact)

for n=1:5

    t=sprintf(' w^2 = %g w = %g  ',w2(n),sqrt(abs(w2(n))) );
    fn=V(:,k(n));
    plot(x,fn,'b-')
    title(t);xlabel('x');ylabel('g(x)');
    pause(0.3)
end
```

The first five eigenvalues from Matlab were: 2.66166, 6.10962, 9.57795, 130509, and 16.5257.

Problem 4.3

Here are the Maple pages for rescaling, including the rescaling in 4.5 (sol5_2.mws).

Scaling Schroedinger's Equation, Lab 5.3 and 5.5, Physics 430

```
> restart;
```

(5.3)

```
> eq:=-hbar^2/2/m*diff(psi(x),x$2)+1/2*k*x^2*psi(x)=E*psi(x);
```

$$-\frac{1}{2}\frac{\mathrm{hbar}^2\left(\frac{d}{dx}\left(\frac{d}{dx}\psi(x)\right)\right)}{m}+\frac{1}{2}k\,x^2\,\psi(x)=E\,\psi(x)$$

Rescale it with $x = a\,\xi$.

```
> eqnew:=-hbar^2/2/m*diff(psi(xi),xi$2)/a^2+1/2*k*a^2*xi^2*psi(x)=E*
  psi(x);
```

$$-\frac{1}{2}\frac{\mathrm{hbar}^2\left(\frac{d}{d\xi}\left(\frac{d}{d\xi}\psi(\xi)\right)\right)}{m\,a^2}+\frac{1}{2}k\,a^2\,\xi^2\,\psi(x)=E\,\psi(x)$$

Divide by $k\,a^2$ so that the potential term is dimensionless

```
> eqnew:=expand(lhs(eqnew)/k/a^2=rhs(eqnew)/k/a^2);
```

$$-\frac{1}{2}\frac{\mathrm{hbar}^2\left(\frac{d}{d\xi}\left(\frac{d}{d\xi}\psi(\xi)\right)\right)}{k\,a^4\,m}+\frac{1}{2}\xi^2\,\psi(x)=\frac{E\,\psi(x)}{k\,a^2}$$

We now choose a so that

```
> eqa:=hbar^2/k/a^4/m=1;
```

$$\frac{\mathrm{hbar}^2}{k\,a^4\,m}=1$$

```
> s:=solve(eqa,a);
```

$$\frac{(\mathrm{hbar}^2\,k^3\,m^3)^{(1/4)}}{k\,m},\ \frac{(\mathrm{hbar}^2\,k^3\,m^3)^{(1/4)}\,I}{k\,m},\ -\frac{(\mathrm{hbar}^2\,k^3\,m^3)^{(1/4)}}{k\,m},\ \frac{-I\,(\mathrm{hbar}^2\,k^3\,m^3)^{(1/4)}}{k\,m}$$

```
> a:=s[1];
```

$$\frac{(\mathrm{hbar}^2\,k^3\,m^3)^{(1/4)}}{k\,m}$$

```
> a:=simplify(a,symbolic);
```

$$\frac{\sqrt{\mathrm{hbar}}}{k^{(1/4)}\,m^{(1/4)}}$$

```
> eqnew;
```

$$-\frac{1}{2}\left(\frac{d}{d\xi}\left(\frac{d}{d\xi}\psi(\xi)\right)\right)+\frac{1}{2}\xi^2\,\psi(x)=\frac{E\,\psi(x)\sqrt{m}}{\sqrt{k}\ \text{hbar}}$$

And now we can make the right side dimensionless by making the scaled energy be $\varepsilon=\frac{E\sqrt{\frac{m}{k}}}{\text{hbar}}$, or E = ε Ebar with Ebar = hbar $\sqrt{\frac{k}{m}}$.

```
> E:=epsilon*hbar*sqrt(k/m);
```

$$\varepsilon\ \text{hbar}\sqrt{\frac{k}{m}}$$

```
> simplify(eqnew,symbolic);
```

$$-\frac{1}{2}\left(\frac{d}{d\xi}\left(\frac{d}{d\xi}\psi(\xi)\right)\right)+\frac{1}{2}\xi^2\,\psi(x)=\varepsilon\,\psi(x)$$

(5.5) Now do the same thing for the 4th power potential

```
> restart;
> eq:=-hbar^2/2/m*diff(psi(x),x$2)+mu*x^4*psi(x)=E*psi(x);
```

$$-\frac{1}{2}\frac{\text{hbar}^2\left(\frac{d}{dx}\left(\frac{d}{dx}\psi(x)\right)\right)}{m}+\mu\,x^4\,\psi(x)=E\,\psi(x)$$

Rescale it with x = a ξ.

```
> eqnew:=-hbar^2/2/m*diff(psi(xi),xi$2)/a^2+mu*a^4*xi^4*psi(x)=E*psi
  (x);
```

$$-\frac{1}{2}\frac{\text{hbar}^2\left(\frac{d}{d\xi}\left(\frac{d}{d\xi}\psi(\xi)\right)\right)}{m\,a^2}+\mu\,a^4\,\xi^4\,\psi(x)=E\,\psi(x)$$

Divide by $\mu\,a^4$ so that the potential term is dimensionless

```
> eqnew:=expand(lhs(eqnew)/mu/a^4=rhs(eqnew)/mu/a^4);
```

$$-\frac{1}{2}\frac{\text{hbar}^2\left(\frac{d}{d\xi}\left(\frac{d}{d\xi}\psi(\xi)\right)\right)}{\mu\,a^6\,m}+\xi^4\,\psi(x)=\frac{E\,\psi(x)}{\mu\,a^4}$$

We now choose a so that

```
> eqa:=hbar^2/mu/a^6/m=1;
```

$$\frac{\text{hbar}^2}{\mu\,a^6\,m}=1$$

```
> s:=solve(eqa,a);
```

$$\frac{(\mathrm{hbar}^2\,\mu^5\,m^5)^{(1/6)}}{\mu\,m},\ \frac{\left(\frac{1}{2}+\frac{1}{2}I\sqrt{3}\right)(\mathrm{hbar}^2\,\mu^5\,m^5)^{(1/6)}}{\mu\,m},$$
$$\frac{\left(-\frac{1}{2}+\frac{1}{2}I\sqrt{3}\right)(\mathrm{hbar}^2\,\mu^5\,m^5)^{(1/6)}}{\mu\,m},\ -\frac{(\mathrm{hbar}^2\,\mu^5\,m^5)^{(1/6)}}{\mu\,m},$$
$$\frac{\left(-\frac{1}{2}-\frac{1}{2}I\sqrt{3}\right)(\mathrm{hbar}^2\,\mu^5\,m^5)^{(1/6)}}{\mu\,m},\ \frac{\left(\frac{1}{2}-\frac{1}{2}I\sqrt{3}\right)(\mathrm{hbar}^2\,\mu^5\,m^5)^{(1/6)}}{\mu\,m}$$

```
> a:=simplify(s[1],symbolic);
```

$$\frac{\mathrm{hbar}^{(1/3)}}{\mu^{(1/6)}\,m^{(1/6)}}$$

```
> eqnew;
```

$$-\frac{1}{2}\left(\frac{d}{d\xi}\left(\frac{d}{d\xi}\psi(\xi)\right)\right)+\xi^4\,\psi(x)=\frac{E\,\psi(x)\,m^{(2/3)}}{\mu^{(1/3)}\,\mathrm{hbar}^{(4/3)}}$$

And now we can make the right side dimensionless by making the scaled energy be

$\varepsilon = E\left(\frac{m^2}{\mu\,\mathrm{hbar}^4}\right)^{\left(\frac{1}{3}\right)}$ which is in the form $E = \varepsilon\,\mathrm{Ebar}$ with $\mathrm{Ebar} = \left(\frac{\mathrm{hbar}^4\,\mu}{m^2}\right)^{\left(\frac{1}{3}\right)}$.

```
> E:=epsilon*(mu*hbar^4/m^2)^(1/3);
```

$$\varepsilon\left(\frac{\mu\,\mathrm{hbar}^4}{m^2}\right)^{(1/3)}$$

```
> simplify(eqnew,symbolic);
```

$$-\frac{1}{2}\left(\frac{d}{d\xi}\left(\frac{d}{d\xi}\psi(\xi)\right)\right)+\xi^4\,\psi(x)=\varepsilon\,\psi(x)$$

which is the dimensionless form we want

Problem 4.4

Listing 4.2 (lab4p4.m)

```
% Lab Problem 4.4, Physics 430
clear;close all;

L=8;
xmin=-L;xmax=L;
N=500;%input(' Enter N - ');
h=(xmax-xmin)/(N-1);
x=xmin:h:xmax;
x=x';

% set the physical constants to 1 since we are using
% a dimensionless form
hbar=1;m=1;k=1;

% define the matrices A and B
A=zeros(N,N);
B=A;
% load the operator
for n=2:N-1
   A(n,n-1)=-hbar^2/2/m/h^2;
   A(n,n+1)=-hbar^2/2/m/h^2;
   A(n,n)=hbar^2/m/h^2+.5*x(n)^2;
   B(n,n)=1; % just make B the identity in this problem
end

% apply zero boundary conditions
A(1,1)=1;B(1,1)=0;
A(N,N)=1;B(N,N)=0;

[V,D]=eig(A,B);
Eraw=diag(D);

[E,k]=sort(Eraw);

for n=1:N

    t=sprintf(' E = %g  ',E(n) );
    plot(x,V(:,k(n)),'b-','LineWidth',2)
    title(t);xlabel('x');ylabel('\psi(x)');
    pause(0.3)
end
```

Problem 4.5

The scaled energies are: 0.668, 2.39, 4.70, 7.33, ,10.2.

Listing 4.3 (lab4p5.m)

```
% Lab Problem 4.5, Physics 430
clear;close all;
```

```
L=8;
xmin=-L;xmax=L;
N=input(' Enter N - ');
h=(xmax-xmin)/(N-1);
x=xmin:h:xmax;
x=x';

% set the physical constants to 1 since we are using
% a dimensionless form
hbar=1;m=1;k=1;


% define the matrices A and B
A=zeros(N,N);
B=A;
% load the operator
for n=2:N-1
   A(n,n-1)=-hbar^2/2/m/h^2;
   A(n,n+1)=-hbar^2/2/m/h^2;
   A(n,n)=hbar^2/m/h^2+x(n)^4;
   B(n,n)=1; % just make B the identity in this problem
end

% apply zero boundary conditions
A(1,1)=1;B(1,1)=0;
A(N,N)=1;B(N,N)=0;

[V,D]=eig(A,B);
Eraw=diag(D);

[E,k]=sort(Eraw);401

for n=1:10

    t=sprintf(' E = %g  ',E(n) );
    plot(x,V(:,k(n)),'b-','LineWidth',2)
    title(t);xlabel('x');ylabel('\psi(x)');
    pause(0.3)
end
```

Chapter 5

Animating the Wave Equation: Staggered Leapfrog

5.1 How to teach this lab

This lab is really fun because the students will be making animations, so just give them a short introduction and then let them start. You should, however, give a mini-lecture on boundary conditions and the initial conditions. The initial condition algorithm is a little tricky, but they are to derive it themselves, so just review the basic idea. You will need to do significant coaching along the way. In the middle somewhere, after a few students have finished part (e), get a slinky and observe the rather odd wave behavior when the slinky is struck sharply in the middle.

5.2 Solutions

Maple for Problems 5.1, 5.2, 5.3(d)-(e)

Here are printouts of the maple. The file is sol6.mws.

Lab Problem 6.1, 6.2, 6.3(d), 6.3(e), Physics 430

```
> restart;
```

(6.1) Use variables named y(j,n) to make the answer come out close to what we need to code the algorithm

Second-order time derivative

```
> dyt2:=(y(j,n-1)-2*y(j,n)+y(j,n+1))/tau^2;
```

$$dyt2 := \frac{y(j, n-1) - 2\,y(j, n) + y(j, n+1)}{\tau^2}$$

Second-order space derivative

```
> dyx2:=(y(j-1,n)-2*y(j,n)+y(j+1,n))/h^2;
```

$$dyx2 := \frac{y(j-1, n) - 2\,y(j, n) + y(j+1, n)}{h^2}$$

Here is the wave equation tranlated into finite-difference form

```
> wave:=dyt2-c^2*dyx2=0;
```

$$wave := \frac{y(j, n-1) - 2\,y(j, n) + y(j, n+1)}{\tau^2} - \frac{c^2\,(y(j-1, n) - 2\,y(j, n) + y(j+1, n))}{h^2} = 0$$

Now solve for y(j,n+1), the value of y at each grid point one time step into the future

```
> expand(solve(wave,y(j,n+1)));
```

$$-y(j, n-1) + 2\,y(j, n) + \frac{c^2\,\tau^2\,y(j-1, n)}{h^2} - \frac{2\,c^2\,\tau^2\,y(j, n)}{h^2} + \frac{c^2\,\tau^2\,y(j+1, n)}{h^2}$$

(6.2) Write down the two equations and and solve for y(j) one time step into the past

```
> restart;
> eq1:=(y(j,1)-y(j,-1))/2/tau=vj;
```

$$eq1 := \frac{1}{2}\,\frac{y(j, 1) - y(j, -1)}{\tau} = vj$$

```
> eq2:=y(j,1)=2*y(j,0)-y(j,-1)+c^2*tau^2/h^2*(y(j+1,0)-2*y(j,0)+y(j-
  1,0));
```

$$eq2 := y(j, 1) = 2\,y(j, 0) - y(j, -1) + \frac{c^2\,\tau^2\,(y(j+1, 0) - 2\,y(j, 0) + y(j-1, 0))}{h^2}$$

Solve for both y(j,-1) and y(j,1)

```
> solve({eq1,eq2},{y(j,1),y(j,-1)});
```

$$\{y(j, -1) = -\frac{1}{2}\,\frac{-2\,y(j, 0)\,h^2 + 2\,h^2\,vj\,\tau - c^2\,\tau^2\,y(j+1, 0) + 2\,c^2\,\tau^2\,y(j, 0) - c^2\,\tau^2\,y(j-1, 0)}{h^2},$$

$$y(j, 1) = -\frac{1}{2}\,\frac{-2\,y(j, 0)\,h^2 - 2\,h^2\,vj\,\tau - c^2\,\tau^2\,y(j+1, 0) + 2\,c^2\,\tau^2\,y(j, 0) - c^2\,\tau^2\,y(j-1, 0)}{h^2}\}$$

```
> assign(%);
```

Display the final form

```
> y(j,-1);
```

$$-\frac{1}{2}\frac{-2\,y(j,0)\,h^2+2\,h^2\,vj\,\tau-c^2\,\tau^2\,y(j+1,0)+2\,c^2\,\tau^2\,y(j,0)-c^2\,\tau^2\,y(j-1,0)}{h^2}$$

```
> expand(y(j,-1));
```

$$y(j,0)-vj\,\tau+\frac{\frac{1}{2}c^2\,\tau^2\,y(j+1,0)}{h^2}-\frac{c^2\,\tau^2\,y(j,0)}{h^2}+\frac{\frac{1}{2}c^2\,\tau^2\,y(j-1,0)}{h^2}$$

6.3(d) Rederive staggered leapfrog with damping included

```
> restart;
```

Here is the damping term

```
> damp:=gamma*(y(j,n+1)-y(j,n-1))/(2*tau);
```

$$damp := \frac{1}{2}\frac{\gamma\,(y(j,n+1)-y(j,n-1))}{\tau}$$

Second-order time derivative

```
> dyt2:=(y(j,n-1)-2*y(j,n)+y(j,n+1))/tau^2;
```

$$dyt2 := \frac{y(j,n-1)-2\,y(j,n)+y(j,n+1)}{\tau^2}$$

Second-order space derivative

```
> dyx2:=(y(j-1,n)-2*y(j,n)+y(j+1,n))/h^2;
```

$$dyx2 := \frac{y(j-1,n)-2\,y(j,n)+y(j+1,n)}{h^2}$$

```
> wave2:=dyt2+damp-c^2*dyx2=0;
```

$$wave2 := \frac{y(j,n-1)-2\,y(j,n)+y(j,n+1)}{\tau^2}+\frac{\frac{1}{2}\gamma\,(y(j,n+1)-y(j,n-1))}{\tau}$$
$$-\frac{c^2\,(y(j-1,n)-2\,y(j,n)+y(j+1,n))}{h^2}=0$$

```
> y(j,n+1):=solve(wave2,y(j,n+1));
```

$$y(j,n+1) := -\left(2\,h^2\,y(j,n-1)-4\,h^2\,y(j,n)-\gamma\,\tau\,h^2\,y(j,n-1)-2\,c^2\,\tau^2\,y(j-1,n)\right.$$
$$\left.+4\,c^2\,\tau^2\,y(j,n)-2\,c^2\,\tau^2\,y(j+1,n)\right)\Big/\left(h^2\,(2+\gamma\,\tau)\right)$$

```
> expand(y(j,n+1));
```

$$-2\frac{y(j,n-1)}{2+\gamma\,\tau}+\frac{4\,y(j,n)}{2+\gamma\,\tau}+\frac{\gamma\,\tau\,y(j,n-1)}{2+\gamma\,\tau}+\frac{2\,c^2\,\tau^2\,y(j-1,n)}{h^2\,(2+\gamma\,\tau)}-\frac{4\,c^2\,\tau^2\,y(j,n)}{h^2\,(2+\gamma\,\tau)}$$

$$+\frac{2\,c^2\,\tau^2\,y(j+1,n)}{h^2\,(2+\gamma\,\tau)}$$

This is the damping-modified algorithm. Now find the new starting value of yold

```
> restart;
```

First comes the velocity initial condition

```
> eq1:=(y(j,1)-y(j,-1))/2/tau=vj;
```

$$eq1 := \frac{1}{2}\,\frac{y(j,1)-y(j,-1)}{\tau} = vj$$

Then we have the damped leapfrog algorithm

```
> eq2:=y(j,1)=-2/(2+gamma*tau)*y(j,-1)+4/(2+gamma*tau)*y(j,0)+1/(2+g
  amma*tau)*gamma*tau*y(j,-1)+2/h^2/(2+gamma*tau)*c^2*tau^2*y(j-1,0)
  -4/h^2/(2+gamma*tau)*c^2*tau^2*y(j,0)+2/h^2/(2+gamma*tau)*c^2*tau^
  2*y(j+1,0);
```

$$eq2 := y(j,1) = -2\,\frac{y(j,-1)}{2+\gamma\,\tau} + \frac{4\,y(j,0)}{2+\gamma\,\tau} + \frac{\gamma\,\tau\,y(j,-1)}{2+\gamma\,\tau} + \frac{2\,c^2\,\tau^2\,y(j-1,0)}{h^2\,(2+\gamma\,\tau)} - \frac{4\,c^2\,\tau^2\,y(j,0)}{h^2\,(2+\gamma\,\tau)} + \frac{2\,c^2\,\tau^2\,y(j+1,0)}{h^2\,(2+\gamma\,\tau)}$$

```
> solve({eq1,eq2},{y(j,1),y(j,-1)});
```

$$\{y(j,1) = -\frac{1}{2}\,\frac{-2\,h^2\,vj\,\tau + h^2\,\gamma\,\tau^2\,vj - 2\,y(j,0)\,h^2 - c^2\,\tau^2\,y(j-1,0) + 2\,c^2\,\tau^2\,y(j,0) - c^2\,\tau^2\,y(j+1,0)}{h^2},$$

$$y(j,-1) = -\frac{1}{2}\,\frac{2\,h^2\,vj\,\tau + h^2\,\gamma\,\tau^2\,vj - 2\,y(j,0)\,h^2 - c^2\,\tau^2\,y(j-1,0) + 2\,c^2\,\tau^2\,y(j,0) - c^2\,\tau^2\,y(j+1,0)}{h^2}\}$$

```
> assign(%);
```

Here is the final form for y(j,-1)

```
> expand(y(j,-1));
```

$$-vj\,\tau - \frac{1}{2}\,\gamma\,\tau^2\,vj + y(j,0) + \frac{\frac{1}{2}\,c^2\,\tau^2\,y(j-1,0)}{h^2} - \frac{c^2\,\tau^2\,y(j,0)}{h^2} + \frac{\frac{1}{2}\,c^2\,\tau^2\,y(j+1,0)}{h^2}$$

6.3(e) Rederive damped staggered leapfrog with a driving force. Recall that we found the wave-equation form containing $\frac{\partial^2}{\partial t^2}y$ by dividing by the mass density μ, so the driving force must be divided by μ as well.

```
> restart;
```

Here is the damping term

```
> damp:=gamma*(y(j,n+1)-y(j,n-1))/(2*tau);
```

$$damp := \frac{1}{2}\,\frac{\gamma\,(y(j,n+1)-y(j,n-1))}{\tau}$$

Second-order time derivative

```
> dyt2:=(y(j,n-1)-2*y(j,n)+y(j,n+1))/tau^2;
```

$$dyt2 := \frac{y(j, n-1) - 2\,y(j, n) + y(j, n+1)}{\tau^2}$$

Second-order space derivative

```
> dyx2:=(y(j-1,n)-2*y(j,n)+y(j+1,n))/h^2;
```

$$dyx2 := \frac{y(j-1, n) - 2\,y(j, n) + y(j+1, n)}{h^2}$$

```
> wave3:=dyt2+damp-c^2*dyx2=f/mu;
```

$$wave3 := \frac{y(j, n-1) - 2\,y(j, n) + y(j, n+1)}{\tau^2} + \frac{\frac{1}{2}\gamma\,(y(j, n+1) - y(j, n-1))}{\tau}$$
$$- \frac{c^2\,(y(j-1, n) - 2\,y(j, n) + y(j+1, n))}{h^2} = \frac{f}{\mu}$$

```
> y(j,n+1):=solve(wave3,y(j,n+1));
```

$$y(j, n+1) := (-2\,h^2\,\mu\,y(j, n-1) + 4\,h^2\,\mu\,y(j, n) + \gamma\,\tau\,h^2\,\mu\,y(j, n-1) + 2\,c^2\,\tau^2\,\mu\,y(j-1, n)$$
$$- 4\,c^2\,\tau^2\,\mu\,y(j, n) + 2\,c^2\,\tau^2\,\mu\,y(j+1, n) + 2\,f\,\tau^2\,h^2) \,/\, (h^2\,\mu\,(2 + \gamma\,\tau))$$

```
> expand(y(j,n+1));
```

$$-2\frac{y(j, n-1)}{2+\gamma\,\tau} + \frac{4\,y(j, n)}{2+\gamma\,\tau} + \frac{\gamma\,\tau\,y(j, n-1)}{2+\gamma\,\tau} + \frac{2\,c^2\,\tau^2\,y(j-1, n)}{h^2\,(2+\gamma\,\tau)} - \frac{4\,c^2\,\tau^2\,y(j, n)}{h^2\,(2+\gamma\,\tau)}$$
$$+ \frac{2\,c^2\,\tau^2\,y(j+1, n)}{h^2\,(2+\gamma\,\tau)} + \frac{2\,f\,\tau^2}{\mu\,(2+\gamma\,\tau)}$$

The term at the end is the new piece due to the driving force. Now find the new starting value of yold

```
> restart;
```

First comes the velocity initial condition

```
> eq1:=(y(j,1)-y(j,-1))/2/tau=vj;
```

$$eq1 := \frac{1}{2}\frac{y(j, 1) - y(j, -1)}{\tau} = vj$$

Then we have the damped leapfrog algorithm with the force term added at the back end

```
> eq2:=y(j,1)=-2/(2+gamma*tau)*y(j,-1)+4/(2+gamma*tau)*y(j,0)+1/(2+g
  amma*tau)*gamma*tau*y(j,-1)+2/h^2/(2+gamma*tau)*c^2*tau^2*y(j-1,0)
  -4/h^2/(2+gamma*tau)*c^2*tau^2*y(j,0)+2/h^2/(2+gamma*tau)*c^2*tau^
  2*y(j+1,0)+2/(2+gamma*tau)*f*tau^2/mu;
```

$$eq2 := y(j, 1) = -2\frac{y(j, -1)}{2+\gamma\,\tau} + \frac{4\,y(j, 0)}{2+\gamma\,\tau} + \frac{\gamma\,\tau\,y(j, -1)}{2+\gamma\,\tau} + \frac{2\,c^2\,\tau^2\,y(j-1, 0)}{h^2\,(2+\gamma\,\tau)} - \frac{4\,c^2\,\tau^2\,y(j, 0)}{h^2\,(2+\gamma\,\tau)}$$
$$+ \frac{2\,c^2\,\tau^2\,y(j+1, 0)}{h^2\,(2+\gamma\,\tau)} + \frac{2\,f\,\tau^2}{(2+\gamma\,\tau)\,\mu}$$

```
> solve({eq1,eq2},{y(j,1),y(j,-1)});
```

$$\{y(j,1) = -\frac{1}{2}(-2h^2\mu vj\tau + h^2\mu\gamma\tau^2 vj - 2y(j,0)h^2\mu - c^2\tau^2 y(j-1,0)\mu + 2c^2\tau^2 y(j,0)\mu - c^2\tau^2 y(j+1,0)\mu - f\tau^2 h^2) \Big/ (h^2\mu),\ y(j,-1) = -\frac{1}{2}(2h^2\mu vj\tau + h^2\mu\gamma\tau^2 vj - 2y(j,0)h^2\mu - c^2\tau^2 y(j-1,0)\mu + 2c^2\tau^2 y(j,0)\mu - c^2\tau^2 y(j+1,0)\mu - f\tau^2 h^2) \Big/ (h^2\mu)\}$$

```
> assign(%);
```

Here is the final form for y(j,-1)

```
> y(j,-1):=expand(y(j,-1));
```

$$y(j,-1) := -vj\tau - \frac{1}{2}\gamma\tau^2 vj + y(j,0) + \frac{\frac{1}{2}c^2\tau^2 y(j-1,0)}{h^2} - \frac{c^2\tau^2 y(j,0)}{h^2} + \frac{\frac{1}{2}c^2\tau^2 y(j+1,0)}{h^2} + \frac{\frac{1}{2}f\tau^2}{\mu}$$

$$-vj\tau - \frac{1}{2}\gamma\tau^2 vj + y(j,0) + \frac{\frac{1}{2}c^2\tau^2 y(j-1,0)}{h^2} - \frac{c^2\tau^2 y(j,0)}{h^2} + \frac{\frac{1}{2}c^2\tau^2 y(j+1,0)}{h^2}$$

But in the problem at hand all of the y(j)'s as well as the initial velocity are zero. So in this simple case y(j,-1) reduces to

```
> subs(vj=0,y(j,0)=0,y(j+1,0)=0,y(j-1,0)=0,y(j,-1));
```

$$\frac{1}{2}\frac{f\tau^2}{\mu}$$

which is just our old friend from freshman physics, $\frac{1}{2}at^2$.

```
>
>
```

Problem 5.3(a)

Listing 5.1 (lab5p3a.m)

```
% Lab Problem 5.3(a), Physics 430
clear; close all;

% Staggered Leapfrog Script

% build a cell-centered grid with N=200 and L=1;
L=1;N=200;h=L/N;
x=-h/2:h:L+h/2;

% Define the initial displacement and velocity
y = exp(-(x-L/2).^2*40/L^2)-exp(-(0-L/2).^2*40/L^2);
vy = 0*x;

subplot(2,1,1)
plot(x,y) % plot the initial conditions
xlabel('x');ylabel('y(x,0)');title('Initial Displacment')
subplot(2,1,2)
plot(x,vy) % plot the initial conditions
xlabel('x');ylabel('v_y(x,0)');title('Initial Velocity')

% Set the wave speed;
c=2; % wave speed

% Suggest to the user that a time step no smaller than taulim be used
taulim=h/c;
fprintf(' Courant time step limit %g \n',taulim)
tau=input(' Enter the time step - ')

% Get the initial value of yold from the initial y and vy
yold(2:N+1)=y(2:N+1)-tau*vy(2:N+1)+ ...
c^2*tau^2/2/h^2*(y(3:N+2)-2*y(2:N+1)+y(1:N));

% load the ghost point bounday conditions in yold(1) and yold(N+2)
yold(1)=-yold(2);
yold(N+2)=-yold(N+1);

tfinal=input(' Enter tfinal - ')
skip=input(' Enter # of steps to skip between plots (speeds the animation up - ')
nsteps=tfinal/tau;

figure
% here is the loop that steps the solution along
for n=1:nsteps
    time=n*tau; % compute the time
   % Use leapfrog and the boundary conditions to load ynew with y at the next
   % time step using y and yold, i.e., ynew(2:N+1)=...
   % Be sure to use colon commands so it will run fast.
  ynew(2:N+1)=2*y(2:N+1)-yold(2:N+1)+ ...
  c^2*tau^2/h^2*(y(3:N+2)-2*y(2:N+1)+y(1:N));
  ynew(1)=-ynew(2);ynew(N+2)=-ynew(N+1);
   %update yold and y
```

```
   yold=y;y=ynew;

% make plots every skip time steps
   if mod(n,skip)==0
      plot(x,y,'b-')
      xlabel('x');ylabel('y');
% Print a top label with the current time included
      topline=sprintf('Staggered Leapfrog Wave: time= %g',time)
      title(topline);
      axis([min(x) max(x) -2 2]);
      pause(.1)
   end
end
```

Problem 5.3(b)

Listing 5.2 (lab5p3b.m)

```
% Lab Problem 5.3(b), Physics 430
clear; close all;

% Staggered Leapfrog Script


% build a cell-centered grid with N=200 and L=1;
L=1;N=200;h=L/N;
x=-h/2:h:L+h/2;

% Define the initial displacement and velocity
y = exp(-(x-L/2).^2*40/L^2)-exp(-(0-L/2).^2*40/L^2);
vy = 0*x;

subplot(2,1,1)
plot(x,y) % plot the initial conditions
xlabel('x');ylabel('y(x,0)');title('Initial Displacment')
subplot(2,1,2)
plot(x,vy) % plot the initial conditions
xlabel('x');ylabel('v_y(x,0)');title('Initial Velocity')

% Set the wave speed;
c=2; % wave speed

% Suggest to the user that a time step no smaller than taulim be used
taulim=h/c;
fprintf(' Courant time step limit %g \n',taulim)
tau=input(' Enter the time step - ')

% Get the initial value of yold from the initial y and vy
yold(2:N+1)=y(2:N+1)-tau*vy(2:N+1)+ ...
c^2*tau^2/2/h^2*(y(3:N+2)-2*y(2:N+1)+y(1:N));


% load the ghost point bounday conditions in yold(1) and yold(N+2)
```

```
yold(1)=yold(2);
yold(N+2)=yold(N+1);

tfinal=input(' Enter tfinal - ')
skip=input(' Enter # of steps to skip between plots (speeds the animation up - ')
nsteps=tfinal/tau;

figure
% here is the loop that steps the solution along
for n=1:nsteps
   % Use leapfrog and the boundary conditions to load ynew with y at the next
   % time step using y and yold, i.e., ynew(2:N+1)=...
   % Be sure to use colon commands so it will run fast.
  ynew(2:N+1)=2*y(2:N+1)-yold(2:N+1)+ ...
  c^2*tau^2/h^2*(y(3:N+2)-2*y(2:N+1)+y(1:N));
  ynew(1)=ynew(2);ynew(N+2)=ynew(N+1);
   %update yold and y
   yold=y;y=ynew;

% make plots every skip time steps
   if mod(n,skip)==0
      plot(x,y,'b-')
      xlabel('x');ylabel('y');title('Staggered Leapfrog Wave Equation');
      axis([min(x) max(x) -2 2]);
      pause(.1)
   end
end
```

5.2.1 Problem 5.3(c)

Listing 5.3 (lab5p3c.m)

```
% Lab Problem 5.3(c), Physics 430
clear; close all;

% Staggered Leapfrog Script

% build a cell-centered grid with N=200 and L=1;
L=1;N=200;h=L/N;
x=-h/2:h:L+h/2;

% Define the initial displacement and velocity
vy = exp(-(x-L*.7).^2*160/L^2)-exp(-(0-L*.7).^2*160/L^2);
y = 0*x;

subplot(2,1,1)
plot(x,y) % plot the initial conditions
xlabel('x');ylabel('y(x,0)');title('Initial Displacment')
subplot(2,1,2)
plot(x,vy) % plot the initial conditions
xlabel('x');ylabel('v_y(x,0)');title('Initial Velocity')

% Set the wave speed;
```

```
c=2; % wave speed

% Suggest to the user that a time step no smaller than taulim be used
taulim=h/max(c);
fprintf(' Courant time step limit %g \n',taulim)
tau=input(' Enter the time step - ')

% Get the initial value of yold from the initial y and vy
yold(2:N+1)=y(2:N+1)-tau*vy(2:N+1)+ ...
c^2*tau^2/2/h^2*(y(3:N+2)-2*y(2:N+1)+y(1:N));

% load the ghost point bounday conditions in yold(1) and yold(N+2)
yold(1)=-yold(2);
yold(N+2)=-yold(N+1);

tfinal=input(' Enter tfinal - ')
skip=input(' Enter # of steps to skip between plots (speeds the animation up - ')
nsteps=tfinal/tau;

figure
% here is the loop that steps the solution along
for n=1:nsteps
   % Use leapfrog and the boundary conditions to load ynew with y at the next
   % time step using y and yold, i.e., ynew(2:N+1)=...
   % Be sure to use colon commands so it will run fast.
  ynew(2:N+1)=2*y(2:N+1)-yold(2:N+1)+ ...
  c^2*tau^2/h^2*(y(3:N+2)-2*y(2:N+1)+y(1:N));
  ynew(1)=-ynew(2);ynew(N+2)=-ynew(N+1);
   %update yold and y
   yold=y;y=ynew;

% make plots every skip time steps
   if mod(n,skip)==0
      plot(x,y,'b-')
      xlabel('x');ylabel('y');title('Staggered Leapfrog Wave Equation');
      axis([min(x) max(x) -.2 .2]);
      pause(.1)
   end
end
```

sol6_1f.m

Problem 5.4

Listing 5.4 (lab5p4.m)

```
% Lab Problem 5.4, Physics 430
clear; close all;

% Staggered Leapfrog Script

% build a cell-centered grid with N=200 and L=1;
L=1;N=200;h=L/N;
x=-h/2:h:L+h/2;
```

```
% Define the initial displacement and velocity
vy = exp(-(x-L*.7).^2*160/L^2)-exp(-(0-L*.7).^2*160/L^2);
y = 0*x;

subplot(2,1,1)
plot(x,y) % plot the initial conditions
xlabel('x');ylabel('y(x,0)');title('Initial Displacment')
subplot(2,1,2)
plot(x,vy) % plot the initial conditions
xlabel('x');ylabel('v_y(x,0)');title('Initial Velocity')

% Set the wave speed;
c=2; % wave speed
gamma=.2; % damping constant

% Suggest to the user that a time step no smaller than taulim be used
taulim=h/max(c);
fprintf(' Courant time step limit %g \n',taulim)
tau=input(' Enter the time step - ')

% Get the initial value of yold from the initial y and vy

yold(2:N+1)=-1/2*(2*h^2*vy(2:N+1)*tau+gamma*tau^2*h^2*vy(2:N+1)- ...
    2*h^2*y(2:N+1)-c^2*tau^2*y(3:N+2)+2*c^2*tau^2*y(2:N+1)-c^2*tau^2*y(1:N))/h^2;

% load the ghost point bounday conditions in yold(1) and yold(N+2)
yold(1)=-yold(2);
yold(N+2)=-yold(N+1);

tfinal=input(' Enter tfinal - ')
skip=input(' Enter # of steps to skip between plots (speeds the animation up - ')
nsteps=tfinal/tau;

figure
% here is the loop that steps the solution along
for n=1:nsteps
   % Use leapfrog and the boundary conditions to load ynew with y at the next
   % time step using y and yold, i.e., ynew(2:N+1)=...
   % Be sure to use colon commands so it will run fast.

  ynew(2:N+1)=(4*y(2:N+1)-2*yold(2:N+1))/(2+gamma*tau) + ...
      gamma*tau*yold(2:N+1)/(2+gamma*tau) + ...
      2*c^2*tau^2/h^2/(2+gamma*tau)*(y(3:N+2)-2*y(2:N+1)+y(1:N));

  ynew(1)=-ynew(2);ynew(N+2)=-ynew(N+1);
   %update yold and y
   yold=y;y=ynew;

% make plots every skip time steps
   if mod(n,skip)==0
      plot(x,y,'b-')
      xlabel('x');ylabel('y');title('Staggered Leapfrog Wave Equation');
      axis([min(x) max(x) -0.2 0.2]);
      pause(.1)
```

```
   end
   amp(n)=max(abs(y));
   t(n)=tau*n;
end

% plot the amplitude and exp(-gamma*t/2) decay together
% I cheated and looked at the data and found that Amp was
% maximum at t=0.94
plot(t,amp,t,max(amp)*exp(-gamma*(t-.094)/2))
title('Max Amplitude vs. Time')
```

Problem 5.5

Listing 5.5 (lab5p5.m)

```
% Lab Problem 5.5, Physics 430
clear; close all;

% Staggered Leapfrog Script

% build a cell-centered grid with N=200 and L=1;
L=1.2;N=200;h=L/N;
x=-h/2:h:L+h/2;

% Define the initial velocity and displacement
vy = exp(-(x-L*.7).^2*160/L^2)-exp(-(0-L*.7).^2*160/L^2);
y = 0*x;

subplot(2,1,1)
plot(x,y) % plot the initial conditions
xlabel('x');ylabel('y(x,0)');title('Initial Displacment')
subplot(2,1,2)
plot(x,vy) % plot the initial conditions
xlabel('x');ylabel('v_y(x,0)');title('Initial Velocity')

% Set the wave speed;
T=127;mu=.003;
c=sqrt(T/mu); % wave speed
gamma=60; % damping constant

% load the forcing function
for n=1:N+2
    if x(n) > 0.8 & x(n) < 1
        f(n)=.73;
    else
        f(n)=0;
    end
end

% Suggest to the user that a time step no smaller than taulim be used
taulim=h/max(c);

fprintf(' Courant time step limit %g \n',taulim)
```

```
tau=input(' Enter the time step - ')
omega=1080;
% Get the initial value of yold from the initial y and vy

yold(2:N+1)=-1/2*(2*h^2*vy(2:N+1)*tau+gamma*tau^2*h^2*vy(2:N+1)- ...
    2*h^2*y(2:N+1)-c^2*tau^2*y(3:N+2)+2*c^2*tau^2*y(2:N+1)-c^2*tau^2*y(1:N))/h^2;

% load the ghost point bounday conditions in yold(1) and yold(N+2)
yold(1)=-yold(2);
yold(N+2)=-yold(N+1);

tfinal=input(' Enter tfinal - ')
skip=input(' Enter # of steps to skip between plots (speeds the animation up - ')
nsteps=tfinal/tau;

figure
% here is the loop that steps the solution along
for n=1:nsteps
    time=(n-1)*tau;
   % Use leapfrog and the boundary conditions to load ynew with y at the next
   % time step using y and yold, i.e., ynew(2:N+1)=...
   % Be sure to use colon commands so it will run fast.

  ynew(2:N+1)=(4*y(2:N+1)-2*yold(2:N+1))/(2+gamma*tau) + ...
      gamma*tau*yold(2:N+1)/(2+gamma*tau) + ...
      2*c^2*tau^2/h^2/(2+gamma*tau)*(y(3:N+2)-2*y(2:N+1)+y(1:N)) + ...
      f(2:N+1)*tau^2/mu/(1+gamma*tau/2)*cos(omega*time) ;
  ynew(1)=-ynew(2);ynew(N+2)=-ynew(N+1);
   %update yold and y
   yold=y;y=ynew;

% make plots every skip time steps
   if mod(n,skip)==0
      plot(x,y,'b-')
      xlabel('x');ylabel('y');title('Staggered Leapfrog Wave Equation');
      axis([min(x) max(x) -1e-2 1e-2]);
      time
      pause(.1)
   end
   amp(n)=max(abs(y));
   t(n)=time+tau;
end

plot(t,amp)
title('Max Amplitude vs. Time')
```

Challenge Problem

With $\omega = 1080$ and $\gamma = 20$ the maximum amplitude is about 3.5 mm. With $\gamma = 60$ the maximum amplitude is about 1.2 mm.

Chapter 6

The 2-D Wave Equation With Staggered Leapfrog

6.1 How to teach this lab

I'd suggest a reading quiz on elliptic, hyperbolic, and parabolic partial differential equations.

Many students will not have finished Lab 5 yet, but that's OK because this lab is short. They will need some coaching to code staggered leapfrog in two-dimensions using : commands, but once that's done the lab is easy. You should point out to them that the 2-d wave equation is rather odd. In 1 and 3 dimensions a pulse just propagates out leaving an undisturbed medium behind it. But in 2 dimensions an outgoing pulse leaves a slowly decaying amplitude behind it, and this is what they are supposed to see by plotting the displacement in the center as a function of time. If you don't point it out to them, they won't notice it, so make sure everybody sees it. It is an interesting bit of mathematical arcana that the wave equation in an odd number of dimensions allows outgoing pulses that leave an undisturbed medium behind, but in an even number of dimensions a slowly decaying amplitude is left behind.

6.2 Solutions

Problem 6.1

Listing 6.1 (lab1p1d.m)

```
% Lab Problem 1.1(d), Physics 430
clear;close all;

Nx=30;
Ny=50;
xmin=0;xmax=2;
ymin=-1;ymax=3;
dx=(xmax-xmin)/(Nx-1);
```

```
dy=(ymax-ymin)/(Ny-1);
x=xmin:dx:xmax;
y=ymin:dx:ymax;
[X,Y]=ndgrid(x,y);

Z=exp(-(X.^2+Y.^2)).*cos(5*sqrt(X.^2+Y.^2));
figure
surf(X,Y,Z)
xlabel('x');ylabel('y');
title('Problem 1.1(d): f(x,y)')
```

Problem 6.2

Listing 6.2 (lab6p2.m)

```
% Lab Problem 6.2, Physics 430
clear;close all;

% Staggered Leapfrog Script

% define sigma and mu
sigma=2;mu=0.3;
% define the wave speed
c=sqrt(sigma/mu);

% build a cell-edge grid with N=50 and L=5;
L=5;M=50;h=L/M;
x=-L:h:L;
y=x;
N=length(x);
[X,Y]=ndgrid(x,y);
% get the initial velocity and displacement

z=exp(-5*(X.^2+Y.^2));
vz=0*z;

surf(X,Y,z);
xlabel('x');ylabel('y)');title('Initial Displacement')

% set taulim, the Courant limit.  It is not h/c in 2-dimensions,
% but h/c/sqrt(2) instead

taulim=h/c/sqrt(2);
fprintf(' CFL time step limit: %g \n',taulim)
tau=input(' Enter tau - ')
fac=c^2*tau^2/h^2;

% Get the initial value of zold from the initial z and vz
   zold(2:N-1,2:N-1)=z(2:N-1,2:N-1)-vz(2:N-1,2:N-1)*tau + ...
       .5*fac*(-4*z(2:N-1,2:N-1)+ z(2:N-1,1:N-2)+z(2:N-1,3:N) ...
       +z(3:N,2:N-1)+z(1:N-2,2:N-1));
```

```
% load the fixed edge boundary conditions
zold(1,:)=0;zold(N,:)=0;zold(:,1)=0;zold(:,N)=0;

tfinal=input(' Enter tfinal - ')
skip=input(' Enter # of steps to skip between plots (speeds up animation - ')
nsteps=tfinal/tau;

figure
% here is the loop that steps the solution along
for m=1:nsteps
    time=(m-1)*tau;
   % Use leapfrog and the boundary conditions to load znew with z at the next
   % time step using z and zold, i.e., znew(2:N+1)=...
   % Be sure to use colon commands so it will run fast.

  %for n=2:N-1
  %znew(n,2:N-1)=2*z(n,2:N-1)-zold(n,2:N-1)+fac*(-4*z(n,2:N-1)+ ...
  %    z(n,1:N-2)+z(n,3:N)+z(n+1,2:N-1)+z(n-1,2:N-1));
  %end

  znew(2:N-1,2:N-1)=2*z(2:N-1,2:N-1)-zold(2:N-1,2:N-1)+ ...
      fac*(-4*z(2:N-1,2:N-1) + z(2:N-1,1:N-2)+z(2:N-1,3:N)+ ...
      z(3:N,2:N-1)+z(1:N-2,2:N-1));

  znew(1,:)=0;znew(N,:)=0;znew(:,1)=0;znew(:,N)=0;

   %update zold and z
   zold=z;z=znew;

% make plots every skip time steps
   if mod(m,skip)==0
      surf(X,Y,z)
      xlabel('x');ylabel('y');title('2-d Wave Equation');
      axis([min(x) max(x) min(y) max(y) -.5 .5]);
      pause(.1)
   end

   k=floor((N+1)/2);
   amp(m)=z(k,k);
   t(m)=time+tau;
end
figure
plot(t,amp)
```

When the initial conditions are changed to a velocity pulse the center of the sheet rises, but does not stay up as it does in one dimension. Instead it falls, gradually decaying toward zero until the reflection returns.

Chapter 7

The Diffusion, or Heat, Equation

7.1 How to teach this lab

Most students don't know how to estimate how long diffusion takes in time, or how far it goes in space, by making simple dimensional arguments based on the units of D. This makes a nice classroom exercise at the beginning, or a good reading quiz question since the problem is discussed in 8.1(c). Once they understand this they can start, and the lab is straightforward since explicit algorithm they will use involves a simple modification of their staggered leapfrog scripts from Lab 6.

7.2 Solutions

Problem 7.1(a)–(c)

Solution: Here is a Maple worksheet for (a)-(c). sol8_1abc.mw

Lab Problem 8.1(a), (b), (c) Physics 430

(a)

```
> restart;
```

Here is the proposed solution

```
> T:=To/sqrt(1+4*D*t/sigma^2)*exp(-(x-L/2)^2/(sigma^2+4*D*t));
```

$$T := \frac{To\, \mathbf{e}^{\left(-\frac{(x-1/2\,L)^2}{\sigma^2+4\,D\,t}\right)}}{\sqrt{1+\frac{4\,D\,t}{\sigma^2}}}$$

Calculate the time derivative on the left side of the diffusion equation

```
> left:=diff(T,t);
```

$$left := -2\,\frac{To\, \mathbf{e}^{\left(-\frac{(x-1/2\,L)^2}{\sigma^2+4\,D\,t}\right)} D}{\left(1+\frac{4\,D\,t}{\sigma^2}\right)^{(3/2)}\sigma^2} + \frac{4\,To\left(x-\frac{1}{2}L\right)^2 D\, \mathbf{e}^{\left(-\frac{(x-1/2\,L)^2}{\sigma^2+4\,D\,t}\right)}}{\sqrt{1+\frac{4\,D\,t}{\sigma^2}}\,(\sigma^2+4\,D\,t)^2}$$

Calculate the spatial derivative on the right

```
> right:=D*diff(T,x$2);
```

$$right := D\left(-2\,\frac{To\, \mathbf{e}^{\left(-\frac{(x-1/2\,L)^2}{\sigma^2+4\,D\,t}\right)}}{\sqrt{1+\frac{4\,D\,t}{\sigma^2}}\,(\sigma^2+4\,D\,t)} + \frac{4\,To\left(x-\frac{1}{2}L\right)^2 \mathbf{e}^{\left(-\frac{(x-1/2\,L)^2}{\sigma^2+4\,D\,t}\right)}}{\sqrt{1+\frac{4\,D\,t}{\sigma^2}}\,(\sigma^2+4\,D\,t)^2}\right)$$

Check to see if the two sides match

```
> simplify(expand(left-right));
```

$$0$$

(b)

```
> restart;
```

Here is the trial form of the solution

```
> T:=sin(n*Pi*x/L)*f(t);
```

$$T := \sin\left(\frac{n\,\pi\,x}{L}\right) f(t)$$

Substitute it into the diffusion equation to get an equation for f(t)

```
> eq:=diff(T,t)=D*diff(T,x$2);
```

$$eq := \sin\left(\frac{n\,\pi\,x}{L}\right)\left(\frac{d}{dt}\,\mathrm{f}(t)\right) = -\frac{\mathrm{D}\sin\left(\frac{n\,\pi\,x}{L}\right) n^2\,\pi^2\,\mathrm{f}(t)}{L^2}$$

```
> eq:=simplify(eq/sin(n*Pi*x/L));
```

$$eq := \frac{d}{dt}\,\mathrm{f}(t) = -\frac{\mathrm{D}\,n^2\,\pi^2\,\mathrm{f}(t)}{L^2}$$

Solve for f(t)

```
> dsolve(eq,f(t));
```

$$\mathrm{f}(t) = \_C1\ \mathbf{e}^{\left(-\frac{\mathrm{D}\,n^2\,\pi^2\,t}{L^2}\right)}$$

So it is a decaying exponential with the damping rate given by

```
> gn:=n^2*Pi^2*D/L^2;
```

$$gn := \frac{\mathrm{D}\,n^2\,\pi^2}{L^2}$$

:

(c)

```
> restart;
```

Here is the expression for the width of the distribution

```
> width:=sqrt(sigma^2+4*D*t);
```

$$width := \sqrt{\sigma^2 + 4\,\mathrm{D}\,t}$$

Build the equation that determines the time when the width has doubled

```
> eq:=subs(t=0,width)*2=width;
```

$$eq := 2\sqrt{\sigma^2} = \sqrt{\sigma^2 + 4\,\mathrm{D}\,t}$$

Solve for the doubling time

```
> tdouble:=solve(eq,t);
```

$$tdouble := \frac{3}{4}\,\frac{\sigma^2}{\mathrm{D}}$$

```
>
```

Problem 7.2(a)

Listing 7.1 (lab7p2a.m)

```
% Lab Problem 7.2(a), Physics 430

% FTCS script, diffusion

clear; close all;

% build a cell-centered grid with N=20 and L=3;
L=3;N=20;h=L/N;
x=-h/2:h:L+h/2;

% Define the initial temperature
y = sin(pi*x/L);

% Set the diffusion coefficient;
D=2;

% Suggest to the user that a time step no smaller than taulim be used
% note that there is a factor of 2 in the denominator, which the students
% are supposed to find by numerical experimentation

taulim=h^2/max(D)/2;
fprintf(' Stability time step limit %g \n',taulim);
tau=input(' Enter the time step - ')

tfinal=input(' Enter tfinal - ')
skip=input(' Enter # of steps to skip between plots (speeds the animation up - ')
nsteps=tfinal/tau;

% here is the loop that steps the solution along
for n=1:nsteps
   % Use FTCS and the boundary conditions to load ynew with y at the next
   % time step using the current temperature
  ynew(2:N+1)=y(2:N+1)+ ...
  D*tau/h^2*(y(3:N+2)-2*y(2:N+1)+y(1:N));
  ynew(1)=-ynew(2);ynew(N+2)=-ynew(N+1);
   %update y
   y=ynew;

% make plots every skip time steps
   if mod(n,skip)==0
      subplot(2,1,1)
       time=n*tau;
       yexact=sin(pi*x/L)*exp(-D*pi^2/L^2*time);
      plot(x,yexact,'g*','MarkerSize',10,'LineWidth',3.5)

      hold on
      plot(x,y,'b-','LineWidth',2.5)
      xlabel('x');ylabel('y');title('FTCS Diffusion: numerical and exact (*)');
      axis([min(x) max(x) -2 2]);

      hold off
```

```
      subplot(2,1,2)
      plot(x,y-yexact,'b-');
      title(' Error: y(numerical)-y(exact)')
      pause(.1)
   end
end
```

Problem 7.2(b)

Listing 7.2 (lab7p2b.m)

```
% Lab Problem 7.1(b), Physics 430

% FTCS script, diffusion

clear; close all;

% build a cell-centered grid with N=50 and L=3;
L=3;N=50;h=L/N;
x=-h/2:h:L+h/2;

% Define the initial temperature
y0= exp(-40*(x/L-.5).^2);
y=y0;

% Set the diffusion coefficient;
D=2;

% Suggest to the user that a time step no smaller than taulim be used
% note that there is a factor of 2 in the denominator, which the students
% are supposed to find by numerical experimentation

taulim=h^2/max(D)/2;
fprintf(' Stability time step limit %g \n',taulim);
tau=input(' Enter the time step - ')


tfinal=input(' Enter tfinal - ')
skip=input(' Enter # of steps to skip between plots (speeds the animation up - ')
nsteps=tfinal/tau;

% here is the loop that steps the solution along
for n=1:nsteps
   % Use FTCS and the boundary conditions to load ynew with y at the next
   % time step using the current temperature
  ynew(2:N+1)=y(2:N+1)+ ...
  D*tau/h^2*(y(3:N+2)-2*y(2:N+1)+y(1:N));

% zero boundary conditions
 % ynew(1)=-ynew(2);ynew(N+2)=-ynew(N+1);
% zero slope (no heat flow out of the system)
  ynew(1)=ynew(2);ynew(N+2)=ynew(N+1);
```

```
   %update y
   y=ynew;

% make plots every skip time steps
   if mod(n,skip)==0
      hold off

      plot(x,y0,'r-','LineWidth',2.5)
      hold on

      plot(x,y,'b-','LineWidth',2.5)
      xlabel('x');ylabel('y');title('FTCS Diffusion: numerical and initial');
      axis([min(x) max(x) -2 2]);


      pause(.1)
   end
end
```

Problem 7.2(c)

Listing 7.3 (lab7p2c.m)

```
% Lab Problem 7.2(c), Physics 430

% FTCS script, diffusion

clear; close all;

% build a cell-centered grid with N=50 and L=3;
L=3;N=50;h=L/N;
x=-h/2:h:L+h/2;

% Define inline functions that specify the initial temperature
yinit=inline('exp(-40*(x/L-.5).^2)','x','L') % zero velocity initial condition
% load y(x,0)
y=yinit(x,L);



% Set the diffusion coefficient and its spatial derivative
D=2/(L/5)*(x.^2+L/5);
Dp=2/(L/5)*2*x;

% Suggest to the user that a time step no smaller than taulim be used
taulim=h^2/max(D)/2;
fprintf(' Stability time step limit %g \n',taulim);
tau=input(' Enter the time step - ')


tfinal=input(' Enter tfinal - ')
skip=input(' Enter # of steps to skip between plots (speeds the animation up - ')
```

```
nsteps=tfinal/tau;

% here is the loop that steps the solution along
for n=1:nsteps
   % Use FTCS and the boundary conditions to load ynew with y at the next
   % time step using the current temperature
  ynew(2:N+1)=y(2:N+1)+ ...
  D(2:N+1)*tau/h^2.*(y(3:N+2)-2*y(2:N+1)+y(1:N))+Dp(2:N+1)*tau/(2*h).* ...
  (y(3:N+2)-y(1:N));

% no heat loss, zero derivative
% ynew(1)=ynew(2);ynew(N+2)=ynew(N+1);
  % cold edges, y=0
  ynew(1)=-ynew(2);ynew(N+2)=-ynew(N+1);

   %update y
   y=ynew;

% make plots every skip time steps
   if mod(n,skip)==0

      plot(x,y,'b-','LineWidth',2.5)
      xlabel('x');ylabel('y');title('FTCS Diffusion, D=D_0 (x^2+L/5)/(L/5)');
      axis([min(x) max(x) -2 2]);

      pause(.1)
   end
end
```

Chapter 8

Implicit Methods: the Crank-Nicholson Algorithm

8.1 How to teach this lab

They will have read about implicit vs. explicit, but they will not understand it completely, so I give them a reading quiz where I ask them to define explicit and implicit, then finite difference a simple equation like

$$y' = -2y \tag{8.1}$$

as we do with Crank-Nicholson. Most can't do it, but it forces them to think about it again, and then they are ready to understand it when I do it at the board. Then I review the Crank-Nicholson finite-differencing of the diffusion equation at the board, show them how to set up the matrices A and B, and turn them loose. With a little coaching they can do this lab without any trouble, but they will have trouble again when they try to apply it to Schroedinger's equation in the next lab.

8.2 Solutions

Problem 8.1

Listing 8.1 (lab8p1a.m)

```
% Lab Problem 8.1, Physics 430

clear;close

gamma=3.;

tau=input(' Enter the time step tau - ')

tfinal=100/gamma;

Nsteps=ceil(tfinal/tau);
```

```
y(1)=1;
t(1)=0;
for n=2:Nsteps
   % Euler's method
   % y(n)=y(n-1)-gamma*tau*y(n-1);
   % Crank-Nicholson
   % y(n)=y(n-1)*(1-.5*gamma*tau)/(1+.5*gamma*tau);
   % Fully implicit
   y(n)=y(n-1)/(1+gamma*tau);

   t(n)=(n-1)*tau;
end

plot(t,y,'r-',t,exp(-gamma*t),'b-')
```

Problem 8.2(a)

Listing 8.2 (lab8p2a.m)

```
% Lab Problem 8_2(a), Physics 430

clear; close all;

% Set the number of grid points and build the cell-center
% grid.

N=input(' Enter N, cell number - ')
L=10;
h=L/N;

x=-.5*h:h:L+.5*h;
% Turn x into a column vector.
x=x';

%  Load the diffusion coefficient array
%  Make it a column vector.

D=2*ones(N+2,1);

% Load Dm with average values D(j-1/2) and Dp with D(j+1/2)
% Make them column vectors.

Dm=zeros(N+2,1);Dp=zeros(N+2,1);
Dm(2:N+1)=.5*(D(2:N+1)+D(1:N)); % average j and j-1
Dp(2:N+1)=.5*(D(2:N+1)+D(3:N+2)); % average j and j+1

% Initialize the temperature with a sine function.
T=sin(pi*x/L);

% Find the maximum of T for setting the plot frame.
Tmax=max(T);Tmin=min(T);
```

```
% Suggest a timestep by giving the time step for
% explicit stability, then ask for the time step.

fprintf(' Maximum explicit time step: %g \n',h^2/max(D))
tau = input(' Enter the time step - ')

tfinal=input(' Enter the total run time - ')

% Set the number of time steps to take.
nsteps=tfinal/tau;

% Define a useful constant.
coeff=tau/h^2 ;

% Define the matrices A and B by loading them with zeros
A=zeros(N+2);
B=zeros(N+2);

% load A and B at interior points using
% colon commands

for j=2:N+1
   A(j,j-1)= -0.5*Dm(j)*coeff;
   A(j,j)  = 1+.5*(Dm(j)+Dp(j))*coeff;
   A(j,j+1)= -0.5*Dp(j)*coeff;

   B(j,j-1)=  0.5*Dm(j)*coeff;
   B(j,j)  = 1-.5*(Dm(j)+Dp(j))*coeff;
   B(j,j+1)=  0.5*Dp(j)*coeff;
end

% now load the boundary conditions for the
% case T(0)=0 and T(L)=0

A(1,1)=0.5;A(1,2)=0.5;B(1,1)=0.;
A(N+2,N+1)=0.5;A(N+2,N+2)=0.5;B(N+2,N+2)=0;


% This is the time advance loop.
for mtime=1:nsteps

t=mtime*tau; % define the time

r=B*T;
r(1)=0;r(N+2)=0; % set the right side vector r for T(0)=0 and T(L)=0
T=A\r; % do the linear solve

% Make a plot of the radial T(r) profile every once in a while.
   if(rem(mtime,5) == 0)
   Texact=sin(pi*x/L)*exp(-pi^2*2*t/L^2);
   plot(x,T,'b-',x,Texact,'r-')
   axis([0 L Tmin Tmax])
   err=max(T-Texact)
   pause(.1)
   end
```

```
end
```

Problem 8.2(b)

Listing 8.3 (lab8p2b.m)

```
% Lab Problem 8_2(b), Physics 430

clear; close all;

% Set the number of grid points and build the cell-center
% grid.

N=input(' Enter N, cell number - ')
L=10;
h=L/N;

x=-.5*h:h:L+.5*h;
% Turn x into a column vector.
x=x';

%  Load the diffusion coefficient array
%  Make it a column vector.

D=2*ones(N+2,1);

% Load Dm with average values D(j-1/2) and Dp with D(j+1/2)
% Make them column vectors.

Dm=zeros(N+2,1);Dp=zeros(N+2,1);
Dm(2:N+1)=.5*(D(2:N+1)+D(1:N)); % average j and j-1
Dp(2:N+1)=.5*(D(2:N+1)+D(3:N+2)); % average j and j+1

% Initialize the temperature with a sine function.
T=sin(pi*x/L);

% Find the maximum of T for setting the plot frame.
Tmax=max(T);Tmin=min(T);

% Suggest a timestep by giving the time step for
% explicit stability, then ask for the time step.

fprintf(' Maximum explicit time step: %g \n',h^2/max(D))
tau = input(' Enter the time step - ')

tfinal=input(' Enter the total run time - ')
iskip=input(' Enter the plot skip factor - ')

% Set the number of time steps to take.
nsteps=tfinal/tau;

% Define a useful constant.
```

```
coeff=tau/h^2 ;

% Define the matrices A and B by loading them with zeros
A=zeros(N+2);
B=zeros(N+2);

% load A and B at interior points using
% colon commands

for j=2:N+1
   A(j,j-1)= -0.5*Dm(j)*coeff;
   A(j,j)  = 1+.5*(Dm(j)+Dp(j))*coeff;
   A(j,j+1)= -0.5*Dp(j)*coeff;

   B(j,j-1)=  0.5*Dm(j)*coeff;
   B(j,j)  = 1-.5*(Dm(j)+Dp(j))*coeff;
   B(j,j+1)=  0.5*Dp(j)*coeff;
end

% now load the boundary conditions for the
% case T'(0)=0 and T'(L)=0

A(1,1)=-1/h;A(1,2)=1/h;B(1,1)=0.;
A(N+2,N+1)=-1/h;A(N+2,N+2)=1/h;B(N+2,N+2)=0;


% This is the time advance loop.
for mtime=1:nsteps

t=mtime*tau; % define the time

r=B*T;
r(1)=0;r(N+2)=0; % set the right side vector r for T(0)=0 and T(L)=0
T=A\r; % do the linear solve

% Make a plot of the radial T(r) profile every once in a while.
   if(rem(mtime,iskip) == 0)
   plot(x,T,'b-')
   axis([0 L Tmin Tmax])
   pause
   end

end
```

Problem 8.2(c)

Listing 8.4 (lab8p2c.m)

```
% Lab Problem 8_2(c), Physics 430

clear; close all;

% Set the number of grid points and build the cell-center
```

```
% grid.

N=input(' Enter N, cell number - ')
L=10;
h=L/N;

x=-.5*h:h:L+.5*h;
% Turn x into a column vector.
x=x';

%  Load the diffusion coefficient array
%  Make it a column vector.

% load the variable diffusion coefficient

for n=1:N+2
   if x(n) < L/2
      D(n)=1;
   else
      D(n)=5;
   end
end

% Load Dm with average values D(j-1/2) and Dp with D(j+1/2)
% Make them column vectors.

Dm=zeros(N+2,1);Dp=zeros(N+2,1);
Dm(2:N+1)=.5*(D(2:N+1)+D(1:N)); % average j and j-1
Dp(2:N+1)=.5*(D(2:N+1)+D(3:N+2)); % average j and j+1

% Initialize the temperature with a sine function.
T=sin(pi*x/L);

% Find the maximum of T for setting the plot frame.
Tmax=max(T);Tmin=min(T);

% Suggest a timestep by giving the time step for
% explicit stability, then ask for the time step.

fprintf(' Maximum explicit time step: %g \n',h^2/max(D))
tau = input(' Enter the time step - ')

tfinal=input(' Enter the total run time - ')
iskip=input(' Enter the plot skip factor - ')

% Set the number of time steps to take.
nsteps=tfinal/tau;

% Define a useful constant.
coeff=tau/h^2 ;

% Define the matrices A and B by loading them with zeros
A=zeros(N+2);
B=zeros(N+2);
```

```
% load A and B at interior points using
% colon commands

for j=2:N+1
   A(j,j-1)= -0.5*Dm(j)*coeff;
   A(j,j)  = 1+.5*(Dm(j)+Dp(j))*coeff;
   A(j,j+1)= -0.5*Dp(j)*coeff;

   B(j,j-1)=  0.5*Dm(j)*coeff;
   B(j,j)  = 1-.5*(Dm(j)+Dp(j))*coeff;
   B(j,j+1)=  0.5*Dp(j)*coeff;
end

% now load the boundary conditions for the
% case T'(0)=0 and T'(L)=0

A(1,1)=-1/h;A(1,2)=1/h;B(1,1)=0.;
A(N+2,N+1)=-1/h;A(N+2,N+2)=1/h;B(N+2,N+2)=0;


% This is the time advance loop.
for mtime=1:nsteps

t=mtime*tau; % define the time

r=B*T;
r(1)=0;r(N+2)=0; % set the right side vector r for T(0)=0 and T(L)=0
T=A\r; % do the linear solve

% Make a plot of the radial T(r) profile every once in a while.
   if(rem(mtime,iskip) == 0)
   plot(x,T,'b-')
   axis([0 L Tmin Tmax])
   pause(.1)
   end

end
```

Chapter 9

Schrödinger's Equation

9.1 How to teach this lab

Some discussion at the beginning about how Schrödinger's equation is just like the diffusion equation, but an imaginary diffusion coefficient is helpful. They will also need help reviewing the Crank-Nicholson derivation to see how the potential enters. Then coach them through the changes they will make to their Lab 9 scripts. It always surprises me how long it takes for the students to debug their codes in this part, so watch them closely so they can finish the lab.

9.2 Solutions

Problem 9.1

First do the space derivatives and define $D = i\frac{\hbar}{2m}$

$$\frac{\partial \psi_j}{\partial t} = D\frac{\psi_{j+1} - 2\psi_j + \psi_{j-1}}{h^2} - \frac{i}{\hbar}V(x_j)\psi_j \tag{9.1}$$

Then take a forward time derivative and replace things on the right with the average:

$$\frac{\psi_j^{n+1} - \psi_j^n}{\tau} = D\frac{(\psi_{j+1}^{n+1} + \psi_{j+1}^n) - 2(\psi_j^{n+1} + \psi_j^n) + (\psi_{j-1}^{n+1} + \psi_{j-1}^n)}{2h^2} - \frac{i}{\hbar}V(x_j)\frac{(\psi_j^{n+1} + \psi_j^n)}{2} \tag{9.2}$$

Now put everything at $n+1$ on the left.

$$-D\psi_{j-1}^{n+1} + \left(\frac{2h^2}{\tau} + 2D + \frac{ih^2}{\hbar}V(x_j)\right)\psi_j^{n+1} - D\psi_{j+1}^{n+1} = D\psi_{j-1}^n + \left(\frac{2h^2}{\tau} - 2D - \frac{ih^2}{\hbar}V(x_j)\right)\psi_j^n + D\psi_{j+1}^n \tag{9.3}$$

From here, you just read off the coefficients for **A** and **B**.

Problem 9.2

Listing 9.1 (lab9p2.m)

```
% Lab Problem 9.2, Physics 430
% this code solves Schrodingers equation with Crank-Nicholson
% hbar and m are taken to be 1
clear; close all;

% Set the number of grid points and build the cell-edge grid.
N=input(' Enter N, cell number - ')
L=10;
xmin=-L;xmax=L;
h=(xmax-xmin)/(N-1);
x=xmin:h:xmax;

% initialize psi
p=2*pi;
sigma=2;
psi=1/sqrt(sigma*sqrt(pi))*exp(i*p*x).*exp(-x.^2*.5/sigma^2);
psi = psi-psi(1); % Boundary condition

% turn psi into a column vector.  Note that we use .' (transpose
% without complex conjugate) instead of ' so that if we
% are doing quantum mechanics we don't reverse the direction
% a wave packet moves.
psi=psi.';

% get the maximum of psi for setting the plot frame
psimax=2;psimin=-2;

% Suggest a timestep by giving the time step for
% explicit stability, then ask for the time step
fprintf(' Maximum explicit time step: %g \n',h^2/0.5)
tau = input(' Enter the time step - ')

% Set the potential
V=zeros(1,N);

% load the operator matrices A and B
A=zeros(N,N);
B=A;

% put the boundary conditions in: zero value at each end
A(1,1)=1; % psi(1)=0
A(N,N)=1; % psi(N)=0

D = i/2;
const = 2*h^2 / tau ;
const2 = i*2*h^2;
for j=2:N-1
   A(j,j-1) = -D;
   A(j,j) = const + 2*D + V(j)*const2;
   A(j,j+1) = -D;

   B(j,j-1)= D;
   B(j,j)= const - 2*D - const2*V(j);
```

```
   B(j,j+1)=D;
end

% set the number of time steps to take
tfinal=input(' Enter the total run time - ')
nsteps=tfinal/tau;

% this is the time advance loop
for n=1:nsteps
   t(n)=n*tau; % set the time

   r=B*psi;
   r(1)=0;r(N)=0;
   psi=A\r;

   % check the normalization
   prob(n)=trapz(x,psi.*conj(psi));

   % check the expectation value of the position

   xbar(n)=trapz(x',x'.*psi.*conj(psi));

   % check the energy, using both y(j+1)-y(j-1) (cell edge)
   % and y(j+1)-y(j) (cell center)
   yp(2:N-1)=(psi(3:N)-psi(1:N-2))/(2*h);
   yp(1)=2*yp(2)-yp(3);
   yp(N)=2*yp(N-1)-yp(N-2);
   E(n)=trapz(x,yp.*conj(yp)*.5);
   dy=diff(psi)/h;
   E2(n)=.5*sum(dy.*conj(dy))*h;

% make a plot of the psi(x) once in a while
   if(rem(n,5) == 0)

   % psi(x)
   plot(x,abs(psi).^2)
   axis([-L L 0 1])
   pause(.1)
   end

end

figure
plot(t,prob)
axis([0 max(t) 0 1.2])

figure
plot(t,real(xbar))

figure
plot(t,E)
```

Problem 9.3

Listing 9.2 (lab9p3.m)

```
% Lab Problem 9.3, Physics 430
% this code solves Schrodingers equation with Crank-Nicholson
% hbar and m are taken to be 1
clear; close all;

% Set the number of grid points and build the cell-edge grid.
N=input(' Enter N, cell number - ')
L=10;
xmin=-2*L;xmax=2*L;
h=(xmax-xmin)/(N-1);
x=xmin:h:xmax;

% initialize psi
p=2*pi;
sigma=2;
psi=1/sqrt(sigma*sqrt(pi))*exp(i*p*x).*exp(-x.^2*.5/sigma^2);
psi = psi-psi(1); % Boundary condition

% turn psi into a column vector.  Note that we use .' (transpose
% without complex conjugate) instead of ' so that if we
% are doing quantum mechanics we don't reverse the direction
% a wave packet moves.
psi=psi.';

% get the maximum of psi for setting the plot frame
psimax=2;psimin=-2;

% Suggest a timestep by giving the time step for
% explicit stability, then ask for the time step
fprintf(' Maximum explicit time step: %g \n',h^2/0.5)
tau = input(' Enter the time step - ')

% Set the potential
V=zeros(1,N);
vmax=input(' Enter V of barrier - ')

left=0.98*L;right=L;
for n=1:N
    if x(n) > left & x(n) < right
        V(n)=vmax;
    else
        V(n)=0;
    end
end

% load the operator matrices A and B
A=zeros(N,N);
B=A;

% put the boundary conditions in: zero value at each end
A(1,1)=1; % psi(1)=0
```

```
A(N,N)=1; % psi(N)=0

D = i/2;
const = 2*h^2 / tau ;
const2 = i*2*h^2;
for j=2:N-1
   A(j,j-1) = -D;
   A(j,j) = const + 2*D + V(j)*const2;
   A(j,j+1) = -D;

   B(j,j-1)= D;
   B(j,j)= const - 2*D - const2*V(j);
   B(j,j+1)=D;
end

% set the number of time steps to take
tfinal=input(' Enter the total run time - ')
nsteps=tfinal/tau;

% this is the time advance loop
for n=1:nsteps
   t(n)=n*tau; % set the time

   r=B*psi;
   r(1)=0;r(N)=0;
   psi=A\r;

   % check the normalization
   prob(n)=trapz(x,psi.*conj(psi));

   % check the expectation value of the position

   xbar(n)=trapz(x',x'.*psi.*conj(psi));

   % check the energy, using both y(j+1)-y(j-1) (cell edge)
   % and y(j+1)-y(j) (cell center)
   yp(2:N-1)=(psi(3:N)-psi(1:N-2))/(2*h);
   yp(1)=2*yp(2)-yp(3);
   yp(N)=2*yp(N-1)-yp(N-2);
   E(n)=trapz(x,yp.*conj(yp)*.5);
   dy=diff(psi)/h;
   E2(n)=.5*sum(dy.*conj(dy))*h;

% make a plot of the psi(x) once in a while
   if(rem(n,5) == 0)

   % psi(x)
   plot(x,abs(psi).^2)
   axis([-2*L 2*L 0 1])
   pause(.1)
   end

end

figure
```

```
plot(t,prob)
axis([0 max(t) 0 1.2])

figure
plot(t,real(xbar))

figure
plot(t,E)
```

Chapter 10

Poisson's Equation I

10.1 How to teach this lab

The students will need a mini-lecture on solving equations by iterating and how over-relaxation works. They will also be confused about which index in $V(i, j)$ goes with x and which one goes with y. I have written `sor.m` in "physics mode", $V(x, y)$, instead of "matrix mode" in which the first index moves vertically and the second one moves horizontally. They will have trouble with this when they set the boundary conditions. Watch carefully how they build the boundary conditions so this problem doesn't come back to haunt them next week.

10.2 Solutions

Problem 10.1

Solution is in sol11_1a.mws. Here is a printout:

Lab Problem 11.1(a), Physics 430

```
> restart;
```

Finite-difference Poisson's equation, then solve for the center value of V

```
> pois:=(V(i+1,j)-2*V(i,j)+V(i-1,j))/dx^2+(V(i,j+1)-2*V(i,j)+V(i,j-1
  ))/dy^2=-rho/e0;
```

$$pois := \frac{V(i+1,j)-2\,V(i,j)+V(i-1,j)}{dx^2}+\frac{V(i,j+1)-2\,V(i,j)+V(i,j-1)}{dy^2}=-\frac{\rho}{e0}$$

```
> Vij:=solve(pois,V(i,j));
```

$$Vij := \frac{1}{2}\,\frac{dy^2\,e0\,V(i+1,j)+dy^2\,e0\,V(i-1,j)+dx^2\,e0\,V(i,j+1)+dx^2\,e0\,V(i,j-1)+\rho\,dx^2\,dy^2}{e0\,(dy^2+dx^2)}$$

To put it in the form of the equation in Lab 12, multiply by the denominator of the equation and expand

```
> Vij*(2/dx^2+2/dy^2);
```

$$\frac{1}{2}\,(dy^2\,e0\,V(i+1,j)+dy^2\,e0\,V(i-1,j)+dx^2\,e0\,V(i,j+1)+dx^2\,e0\,V(i,j-1)+\rho\,dx^2\,dy^2)\left(2\,\frac{1}{dx^2}+\frac{2}{dy^2}\right)\Big/(e0\,(dy^2+dx^2))$$

```
> expand(simplify(%));
```

$$\frac{V(i+1,j)}{dx^2}+\frac{V(i-1,j)}{dx^2}+\frac{V(i,j+1)}{dy^2}+\frac{V(i,j-1)}{dy^2}+\frac{\rho}{e0}$$

This is the numerator of the equation in the lab, so all it needs is the denominator to be put back underneath.

```
>
```

Problem 10.2

Listing 10.1 (lab10p2.m)

```
% Lab Problems 10.2
close;clear all;

% simple iteration
x(1)=1;

for n=2:30
    x(n)=exp(-x(n-1));
end

x(30)

plot(x)
```

Problem 10.3

Problem 10.4

Listing 10.2 (lab10p5.m)

```
% Lab Problems 10.5
close;clear all;

% iteration with w
for n=1:50 % loop lots of times so you can choose
           % different values of w.  The best value
           % of w to use is about w=0.64

    w=input(' Enter a value for w - ')
x(1)=1;

for n=2:30
    x(n)=w*exp(-x(n-1))+(1-w)*x(n-1);
end

x(30)

figure
plot(x)

end
```

Problem 10.5(a)

Listing 10.3 (lab10p5a.m)

```
% Lab Problem 10.5(a)
```

```
% sor.m - Program to solve Laplace's equation using
% Successive-Overrelaxation on a square grid
clear;close all;
eps0=8.854e-12;  % set the permittivity of free space

Nx=input('Enter number of x-grid points - ');
Ny=input('Enter number of y-grid points - ');

Lx=4; % Length in x of the computation region
Ly=2; % Length in y of the computation region

% define the grids
dx=Lx/(Nx-1); % Grid spacing in x
dy=Ly/(Ny-1); % Grid spacing in y
x = (0:dx:Lx)-.5*Lx;  %x-grid, x=0 in the middle
y = 0:dy:Ly;  %y-grid

% estimate the best omega to use

r = (dy^2*cos(pi/Nx)+dx^2*cos(pi/Ny))/(dx^2+dy^2);
omega=2/(1+sqrt(1-r^2));
fprintf('May I suggest using omega = %g ? \n',omega);
omega=input('Enter omega for SOR - ');

% define the voltages
V0=1;  % Potential at x=0 and x=Lx
Vscale=V0; % set Vscale to the size of the potential in the problem
fprintf('Potential at ends equals %g \n',V0);
fprintf('Potential is zero on all other boundaries\n');

% set the error criterion
errortest=input(' Enter error criterion - say 1e-6 - ') ;

% Initial guess is zeroes
V = zeros(Nx,Ny);

% set the charge density on the grid
rho=zeros(Nx,Ny);

%%%%%% Set initial conditions and boundary conditions %%%%%

% in V(i,j), i is the x-index and j is the y-index
% set the boundary conditions here

% left and right edges are at V0
for j=1:Ny
   V(1,j) =V0 ;
   V(Nx,j) = V0;
end

% top and bottom are grounded
for i=1:Nx
   V(i,1)=0;
   V(i,Ny)=0;
end
```

```
%%%%%%% MAIN LOOP %%%%%%%%%
t1=cputime;

Niter = Nx*Ny*Nx;  %Set a maximum iteration count

%  set  factors used repeatedly in the algorithm
fac1 = 1/(2/dx^2+2/dy^2);
facx = 1/dx^2;
facy = 1/dy^2;

for n=1:Niter

   err(n)=0; % initialize the error at iteration n to zero

   for i=2:(Nx-1)   % Loop over interior points only
      for j=2:(Ny-1)

        % get the right-hand side
         rhs = fac1*((V(i+1,j)+V(i-1,j))*facx+...
                     (V(i,j-1)+V(i,j+1))*facy+...
                       rho(i,j)/eps0);

         % get the relative error, left side - right side
         err(n)=max(err(n),abs(V(i,j)-rhs)/Vscale);

         % SOR algorithm, update V(i,j)
         V(i,j) = omega*rhs + (1-omega)*V(i,j);
      end
   end

   % if err < errortest break out of the loop

   fprintf('After %g iterations, error= %g\n',n,err(n));

   if(err(n) < errortest)
      disp('Desired accuracy achieved; breaking out of loop');
      break;
   end

end
t2=cputime;
fprintf(' CPU time: %g \n',t2-t1)

% make a contour plot
   cnt=[0,.1,.2,.3,.4,.5,.6,.7,.8,.9,1]; % specify contours
   figure
   cs = contour(x,y,V',cnt);  % Contour plot with labels
   xlabel('x'); ylabel('y'); clabel(cs,[.2,.4,.6,.8])

% make a surface plot
   figure
   surf(x,y,V');  % Surface plot of the potential
   xlabel('x'); ylabel('y');
```

```
% make a plot of error vs. iteration number
   figure
   semilogy(err,'b*')
   xlabel('Iteration');
   ylabel('Relative Error')


% make an electric field plot
Ex=0*V;Ey=0*V;
for i=2:Nx-1
    for j=2:Ny-1

        Ex(i,j)=-(V(i+1,j)-V(i-1,j))/(2*dx);
        Ey(i,j)=-(V(i,j+1)-V(i,j-1))/(2*dy);

    end
end

for i=2:Nx-1
    Ey(i,1)=2*Ey(i,2)-Ey(i,3);
    Ey(i,Ny)=2*Ey(i,Ny-1)-Ey(i,Ny-2);
    Ex(i,1)=0;
    Ex(i,Ny)=0;
end
for j=2:Ny-1
    Ex(1,j)=2*Ex(2,j)-Ex(3,j);
    Ex(Nx,j)=2*Ex(Nx-1,j)-Ex(Nx-2,j);
    Ey(1,j)=0;
    Ey(Nx,j)=0;
end

Ex=min(Ex,1);
Ex=max(Ex,-1);
Ey=min(Ey,1);
Ey=max(Ey,-1);

[X,Y]=meshgrid(x,y);

E=sqrt(Ex.^2+Ey.^2);

figure
quiver(X',Y',Ex,Ey)
```

I ran the code 30x30 with the error criterion set at 1e-4 and found the following change in the number of iterations required to attain convergence as w was varied. The predicted optimum value is 1.8107

w	N-iterations
1	310
1.5	124
1.75	60
1.8	52
1.81	53

```
  1.85   62
  1.9    90
```

so around 1.81 is the best, but with some variation. The variation is caused by the way it starts up when phi is nowhere near its final shape.

Problem 10.5(b)

Listing 10.4 (lab10p5b.m)

```
% Lab Problem 10.5(b)
clear; close all;

% I ran sor.m for N=10, 20, 40, and 80
% and got the following run times
N=[10,20,40,80];
t=[.11,1.02,8.9,(73.6+74.28+73.7)/3];

plot(N,t)


for n=1:100
    p=input('Enter p - ')
    tt=(N/20).^p*t(2); % plot a t^p curve with the second point (N=20)
                       % set to be fit exactly

    plot(N,t,'b-',N,tt,'r*')
end

% p=3.08 is a good fit, so it goes about like 3
```

Chapter 11

Poisson's Equation II

11.1 How to teach this lab

All they have to do this time is modify the code they wrote last week to do the assigned problems in this one.

Don't let them waste all their time on 11.1. Show them how to use $\sigma = \epsilon_0 \mathbf{E} \cdot \mathbf{n}$, then help them implement the zero-derivative boundary condition. If you want to hurry through this you can forget about changing to ghost points or extrapolating and just have them use $V(i,1) = V(i,2)$, which puts the derivative to zero halfway between the grid points instead of at the edge, but the surface plots still look fine.

In 11.2 they will have trouble coming up with a simple way to just have SOR adjust the interior points of the triangle, so a mini-lecture and class discussion about how to do this will help.

And in 11.3 I usually show them how to use a mask array, with the mask set to 1 for SOR points and the mask set to zero for points that are supposed to be left alone. This is a useful programming tool that comes up all the time, so they ought to see it.

11.2 Solutions

Problem 11.1(a)

Listing 11.1 (lab11p1a.m)

```
% Lab 11.1(a)
clear; close all;
eps0=8.854e-12;  % set the permittivity of free space
Nx=40;Ny=40;

Lx=4; % Length in x of the computation region
Ly=2; % Length in y of the computation region

% define the grids
dx=Lx/(Nx-1); % Grid spacing in x
dy=Ly/(Ny-1); % Grid spacing in y
```

```
x = (0:dx:Lx)-.5*Lx;  %x-grid, x=0 in the middle
y = 0:dy:Ly;  %y-grid

% estimate the best omega to use

r = (dy^2*cos(pi/Nx)+dx^2*cos(pi/Ny))/(dx^2+dy^2);
omega=2/(1+sqrt(1-r^2));

% define the voltages
V0=1;  % Potential at x=0 and x=Lx
Vscale=V0; % set Vscale to the size of the potential in the problem
fprintf('Potential at ends equals %g \n',V0);
fprintf('Potential is zero on all other boundaries\n');

% set the error criterion
%errortest=input(' Enter error criterion - say 1e-6 - ') ;
errortest=1e-4;
% Initial guess is zeroes
V = zeros(Nx,Ny);

% set the charge density on the grid
rho=zeros(Nx,Ny);


% left edge at V0, right edge at -V0
for j=1:Ny
   V(1,j) =-V0 ;
   V(Nx,j) = V0;
end

% top and bottom are grounded
for i=1:Nx
   V(i,1)=0;
   V(i,Ny)=0;
end

%%%%%%% MAIN LOOP %%%%%%%%
t1=cputime;

Niter = Nx*Ny*Nx;  %Set a maximum iteration count

%  set  factors used repeatedly in the algorithm
fac1 = 1/(2/dx^2+2/dy^2);
facx = 1/dx^2;
facy = 1/dy^2;

for n=1:Niter

   err(n)=0; % initialize the error at iteration n to zero

   for i=2:(Nx-1)   % Loop over interior points only
      for j=2:(Ny-1)

        % get the right-hand side
         rhs = fac1*((V(i+1,j)+V(i-1,j))*facx+...
```

```
                    (V(i,j-1)+V(i,j+1))*facy+...
                      rho(i,j)/eps0);

          % get the relative error, left side - right side
          err(n)=max(err(n),abs(V(i,j)-rhs)/Vscale);

          % SOR algorithm, update V(i,j)
          V(i,j) = omega*rhs + (1-omega)*V(i,j);
       end
    end

    % if err < errortest break out of the loop

    fprintf('After %g iterations, error= %g\n',n,err(n));

    if(err(n) < errortest)
       disp('Desired accuracy achieved; breaking out of loop');
       break;
    end

end
t2=cputime;
fprintf(' CPU time: %g \n',t2-t1)

% make a contour plot
    cnt=-1:.1:1; % specify contours
    cs = contour(x,y,V',cnt);  % Contour plot with labels
    xlabel('x'); ylabel('y'); clabel(cs,[.2,.4,.6,.8])

% make a surface plot
    figure
    surf(x,y,V');  % Surface plot of the potential
    xlabel('x'); ylabel('y');

% Find the surface charge density along the bottom using
% sigma=dot(E,nhat)*epsilon0, where nhat is the unit normal
% pointing away from the surfaced, yhat in this case

for i=2:Nx-1
     Ey1=-(V(i,2)-V(i,1))/dy;
     Ey2=-(V(i,3)-V(i,2))/dy;
     Ey(i)=1.5*Ey1-.5*Ey2;
end

sigma=eps0*Ey;
figure
plot(x(2:Nx-1),sigma(2:Nx-1))
title(' \sigma along the bottom')
xlabel('x');ylabel('\sigma(x)')
```

Problem 11.1(b)

Listing 11.2 (lab11p1b.m)

```
% Lab 11.1(b)
clear; close all;
eps0=8.854e-12;  % set the permittivity of free space
Nx=40;Ny=40;

Lx=4; % Length in x of the computation region
Ly=2; % Length in y of the computation region

% define the grids
dx=Lx/(Nx-1); % Grid spacing in x
dy=Ly/(Ny-1); % Grid spacing in y
x = (0:dx:Lx)-.5*Lx;  %x-grid, x=0 in the middle
y = 0:dy:Ly;  %y-grid

% estimate the best omega to use

r = (dy^2*cos(pi/Nx)+dx^2*cos(pi/Ny))/(dx^2+dy^2);
omega=2/(1+sqrt(1-r^2));
fprintf('May I suggest using omega = %g ? \n',omega);
omega=input('Enter omega for SOR - ');

% define the voltages
V0=1;  % Potential at x=0 and x=Lx
Vscale=V0; % set Vscale to the size of the potential in the problem
fprintf('Potential at ends equals %g \n',V0);
fprintf('Potential is zero on all other boundaries\n');

errortest=1e-4;
% Initial guess is zeroes
V = zeros(Nx,Ny);

% set the charge density on the grid
rho=zeros(Nx,Ny);


%
for j=1:Ny
   V(1,j) =0 ;
   V(Nx,j) = V0; % ground it to start with
end


for i=1:Nx
   V(i,1)=0;
   V(i,Ny)=0;
end

%%%%%%% MAIN LOOP %%%%%%%%%
t1=cputime;

Niter = Nx*Ny*Nx;  %Set a maximum iteration count

%  set  factors used repeatedly in the algorithm
```

```
fac1 = 1/(2/dx^2+2/dy^2);
facx = 1/dx^2;
facy = 1/dy^2;

for n=1:Niter

   err(n)=0; % initialize the error at iteration n to zero

   for i=2:(Nx-1)   % Loop over interior points only
      for j=2:(Ny-1)

        % get the right-hand side
         rhs = fac1*((V(i+1,j)+V(i-1,j))*facx+...
                     (V(i,j-1)+V(i,j+1))*facy+...
                       rho(i,j)/eps0);

         % get the relative error, left side - right side
         err(n)=max(err(n),abs(V(i,j)-rhs)/Vscale);

         % SOR algorithm, update V(i,j)
         V(i,j) = omega*rhs + (1-omega)*V(i,j);
      end
   end

   % modify the edge values of V to get zero slope, extrapolated to the edge
   for i=1:Nx
       V(i,1)=4/3*V(i,2)-1/3*V(i,3);
   end
   for j=1:Ny
       V(Nx,j)=4/3*V(Nx-1,j)-1/3*V(Nx-2,j);
   end

   % if err < errortest break out of the loop

   fprintf('After %g iterations, error= %g\n',n,err(n));

   if(err(n) < errortest)
      disp('Desired accuracy achieved; breaking out of loop');
      break;
   end

end
t2=cputime;
fprintf(' CPU time: %g \n',t2-t1)

% make a contour plot
   cnt=-1:.1:1; % specify contours
   cs = contour(x,y,V',cnt);  % Contour plot with labels
   xlabel('x'); ylabel('y'); clabel(cs,[.2,.4,.6,.8])

% make a surface plot
   figure
   surf(x,y,V');  % Surface plot of the potential
   xlabel('x'); ylabel('y');
```

```
% Find the surface charge density along the bottom

for i=2:Nx-1
    Ey1=-(V(i,2)-V(i,1))/dx;
    Ey2=-(V(i,3)-V(i,2))/dx;
    Ey(i)=1.5*Ey1-.5*Ey2;
end

sigma=eps0*Ey;
figure
plot(x(2:Nx-1),sigma(2:Nx-1))
title(' \sigma along the bottom')
xlabel('x');ylabel('\sigma(x)')
```

Problem 11.2

Listing 11.3 (lab11p2.m)

```
% Lab 11.2(a)-(b)
clear; close all;

eps0=8.854e-12;  % set the permittivity of free space

Nx=60;Ny=60;

Lx=.2; % Length in x of the computation region
Ly=.4; % Length in y of the computation region

% define the grids
dx=Lx/(Nx-1); % Grid spacing in x
dy=Ly/(Ny-1); % Grid spacing in y
x = (0:dx:Lx);  %x-grid, x=0 in the middle
y = 0:dy:Ly;  %y-grid

% estimate the best omega to use

r = (dy^2*cos(pi/Nx)+dx^2*cos(pi/Ny))/(dx^2+dy^2);
omega=2/(1+sqrt(1-r^2));
%fprintf('May I suggest using omega = %g ? \n',omega);
%omega=input('Enter omega for SOR - ');

% define the voltages
V0=1;  % Potential at x=0 and x=Lx
Vscale=V0; % set Vscale to the size of the potential in the problem
fprintf('Potential at ends equals %g \n',V0);
fprintf('Potential is zero on all other boundaries\n');

% set the error criterion
%errortest=input(' Enter error criterion - say 1e-6 - ') ;
errortest=1e-4;
% Initial guess is zeroes
V = zeros(Nx,Ny);
```

```
% set the charge density on the grid
for i=1:Nx
    for j=1:i
        rho(i,j)=1e-10;
    end
end

% left and right edges are grounded
for j=1:Ny
   V(1,j) =0 ;
   V(Nx,j) = 0;
end

% top and bottom are grounded
for i=1:Nx
   V(i,1)=0;
   V(i,Ny)=0;
end

%%%%%%% MAIN LOOP %%%%%%%%%
t1=cputime;

Niter = Nx*Ny*Nx;  %Set a maximum iteration count

%  set  factors used repeatedly in the algorithm
fac1 = 1/(2/dx^2+2/dy^2);
facx = 1/dx^2;
facy = 1/dy^2;

for n=1:Niter

   err(n)=0; % initialize the error at iteration n to zero

   for i=2:(Nx-1)   % Loop over interior points only
      for j=2:i-1

        % get the right-hand side
         rhs = fac1*((V(i+1,j)+V(i-1,j))*facx+...
                    (V(i,j-1)+V(i,j+1))*facy+...
                      rho(i,j)/eps0);

         % get the relative error, left side - right side
         err(n)=max(err(n),abs(V(i,j)-rhs)/Vscale);

         % SOR algorithm, update V(i,j)
         V(i,j) = omega*rhs + (1-omega)*V(i,j);
      end
   end

   % if err < errortest break out of the loop

   fprintf('After %g iterations, error= %g\n',n,err(n));

   if(err(n) < errortest)
      disp('Desired accuracy achieved; breaking out of loop');
```

```
      break;
   end

end
t2=cputime;
fprintf(' CPU time: %g \n',t2-t1)

% make a contour plot
   cnt=-1:.1:1; % specify contours
   cs = contour(x,y,V');  % Contour plot with labels
   xlabel('x'); ylabel('y'); clabel(cs,[.2,.4,.6,.8])

% make a surface plot
   figure
   surf(x,y,V');  % Surface plot of the potential
   xlabel('x'); ylabel('y');

% make a field plot
[Ex,Ey]=gradient(-V',dx,dy);
figure
quiver(x,y,Ex,Ey);
axis equal
```

Problem 11.3

Listing 11.4 (lab11p3.m)

```
% Lab 11.1, Physics 430

clear; close all;
eps0=8.854e-12;  % set the permittivity of free space
Nx=41;Ny=41;

Lx=.2; % Length in x of the computation region
Ly=.4; % Length in y of the computation region

% define the grids
dx=Lx/(Nx-1); % Grid spacing in x
dy=Ly/(Ny-1); % Grid spacing in y
x = (0:dx:Lx)-.5*Lx;  %x-grid, x=0 in the middle
y = 0:dy:Ly;  %y-grid

% estimate the best omega to use

r = (dy^2*cos(pi/Nx)+dx^2*cos(pi/Ny))/(dx^2+dy^2);
omega=2/(1+sqrt(1-r^2));

% define the voltages
V0=1;  % Potential at x=0 and x=Lx
Vscale=V0; % set Vscale to the size of the potential in the problem
fprintf('Potential at ends equals %g \n',V0);
fprintf('Potential is zero on all other boundaries\n');
```

```
% set the error criterion
%errortest=input(' Enter error criterion - say 1e-6 - ') ;
errortest=1e-4;
% Initial guess is zeroes
V = zeros(Nx,Ny);

% set the charge density on the grid
rho=zeros(Nx,Ny);


% left edge at V0, right edge at -V0
for j=1:Ny
   V(1,j) =V0 ;
   V(Nx,j) = -V0;
end

% top and bottom are grounded
for i=1:Nx
   V(i,1)=0;
   V(i,Ny)=0;
end

% Define a mask array to define where the grounded points are.  The mask
% will be 1 at points that are to obey Poisson's equation and the mask will
% be 0 at points that are supposed to be grounded.
mask=ones(Nx,Ny);  % initially load it with ones

% now choose some points to ground.

% first locate the center

ic=ceil((Nx+1)/2);jc=(Ny+1)/2;

% choose the half-width of the cage in grid units
half=6;
j=jc+half;
mask(ic-half:3:ic+half,j)=0;
j=jc-half;
mask(ic-half:3:ic+half,j)=0;
i=ic+half;
mask(i,jc-half:3:jc+half)=0;
i=ic-half;
mask(i,jc-half:3:jc+half)=0;

Niter = Nx*Ny*Nx;  %Set a maximum iteration count

%  set  factors used repeatedly in the algorithm
fac1 = 1/(2/dx^2+2/dy^2);
facx = 1/dx^2;
facy = 1/dy^2;

for n=1:Niter

   err(n)=0; % initialize the error at iteration n to zero
```

```
   for i=2:(Nx-1)   % Loop over interior points only
      for j=2:(Ny-1)

        % get the right-hand side
         rhs = fac1*((V(i+1,j)+V(i-1,j))*facx+...
                     (V(i,j-1)+V(i,j+1))*facy+...
                       rho(i,j)/eps0);

         % get the relative error, left side - right side
         err(n)=max(err(n),abs((V(i,j)-rhs)*mask(i,j))/Vscale);

         % SOR algorithm, update V(i,j)
         V(i,j) = (omega*rhs + (1-omega)*V(i,j))*mask(i,j);
      end
   end

   % if err < errortest break out of the loop

   fprintf('After %g iterations, error= %g\n',n,err(n));

   if(err(n) < errortest)
      disp('Desired accuracy achieved; breaking out of loop');
      break;
   end

end

% make a contour plot
   cnt=-1:.1:1; % specify contours
   cs = contour(x,y,V',cnt);  % Contour plot with labels
   xlabel('x'); ylabel('y'); clabel(cs,[-.8,-.6,-.4,-.2,0.2,.4,.6,.8])

% make a surface plot
   figure
   surf(x,y,V');  % Surface plot of the potential
   xlabel('x'); ylabel('y');
```

Chapter 12

Gas Dynamics I

12.1 How to teach this lab

12.2 Solutions

Problem 12.1(a)

Solution:

$$\frac{\partial \rho(x-vt)}{\partial t} = -v\rho_0'(x-vt) \quad \frac{\partial v\rho(x-vt)}{\partial x} = v\rho_0'(x-vt) \tag{12.1}$$

so the two terms cancel in the conservation of mass equation.

Problem 12.1(b)

Solution:

$$\frac{\partial \rho}{\partial x} = 0 \text{ and } \frac{\partial v}{\partial x} = \frac{v_0}{a} \tag{12.2}$$

so the mass conservation equation becomes

$$\frac{\partial \rho}{\partial t} + \frac{v_0}{a}\rho = 0 \quad \Rightarrow \quad \rho = \rho_0 e^{-v_0 t/a} \quad \Rightarrow \quad \gamma = \frac{v_0}{a} \tag{12.3}$$

Problem 12.1(c)

Solution: it works the same, but with v_0 changing sign.

$$\frac{\partial \rho}{\partial x} = 0 \text{ and } \frac{\partial v}{\partial x} = -\frac{v_0}{a} \tag{12.4}$$

so the mass conservation equation becomes

$$\frac{\partial \rho}{\partial t} - \frac{v_0}{a}\rho = 0 \quad \Rightarrow \quad \rho = \rho_0 e^{v_0 t/a} \tag{12.5}$$

so now we have an exponential increase in density instead of a decrease

Problem 12.2

We have $p = nk_BT$ and $\rho = Mn$, so

$$p = \frac{\rho k_B T}{M} \tag{12.6}$$

Problem 12.3

Solution:

$$a = \frac{d}{dt}[v(x(t),t)] = \left(\frac{dx}{dt}\right)\frac{\partial v}{\partial x} + \frac{\partial v}{\partial t} = \frac{\partial v}{\partial t} + v\frac{\partial v}{\partial x} \tag{12.7}$$

Problem 12.4

Listing 12.1 (lab12p4.m)

```
% Lab Problem 12.4, Physics 430
clear;close all;

N=400;
L=10.;
h=L/N;
x=-.5*h:h:L+.5*h;
x=x'; % turn x into a column vector


n=zeros(N+2,1);
v=zeros(N+2,1);
nnew=zeros(N+2,1);

% initial pulse
n0=1;
n=n0*(1+exp(-200*(x/L-.5).^2));

v=ones(N+2,1);  % set the convection velocity

taulim=h/max(v);
fprintf(' Courant limit on time step: %g \n',taulim);
tau=input(' Enter the time step - ')


for m=1:500

%
nnew(2:N+1)=n(2:N+1)-tau/(2*h)*(n(3:N+2).*v(3:N+2)-n(1:N).*v(1:N));
nnew(1)=2*n0-nnew(2);
% bad boundary condition
% nnew(N+2)=2*n0-nnew(N+1)
% extrapolate to let it go
nnew(N+2)=2*nnew(N+1)-nnew(N);
n=nnew;
% watch the maximum value of the density
max(n)
```

```
if mod(m,5)==1
    plot(x,n);
    axis([0 L  0 2.*n0])

    pause
end
% check particle conservation
    t(m)=m*tau;
    nc(m)=sum(n(2:N+1))*h;
end

plot(t,nc)
```

Problem 12.5

Solution: To evaluate the time derivatives in the Taylor series use the mass conservation equation:

$$\frac{\partial \rho}{\partial t} = -\frac{\partial \rho v}{\partial x} \tag{12.8}$$

so the third term in the Taylor expansion becomes

$$\frac{\partial^2 \rho}{\partial t^2} = \frac{\partial}{\partial t}\left(-\frac{\partial \rho v}{\partial x}\right) = \frac{\partial}{\partial x}\left(v\frac{\partial \rho v}{\partial x}\right) \tag{12.9}$$

which leads directly to the Lax-Wendroff algorithm.

Listing 12.2 (lab12p5.m)

```
% Lab Problem 12.5, Physics 430
clear;close all;

N=102;
L=10.;
h=L/N;
x=-.5*h:h:L+.5*h;
x=x'; % turn z into a column vector


n=zeros(N+2,1);
v=zeros(N+2,1);
nnew=zeros(N+2,1);

% initial pulse
n0=1;
n=n0*(1+exp(-200*(x/L-.5).^2));

% step function initial condition
% j=ceil(N/2);
% n=ones(N+2,1);
% n(j:end) = 0;
```

```
v=ones(N+2,1);  % set the convection velocity

taulim=h/max(v);
fprintf(' Courant limit on time step: %g \n',taulim);
tau=input(' Enter the time step - ')

for m=1:5000

% Lax-Wendroff
 nnew(2:N+1)=n(2:N+1)-tau/(2*h)*(n(3:N+2).*v(3:N+2)-n(1:N).*v(1:N))...
  + tau^2/2/h^2*(.5*(v(3:N+2)+v(2:N+1)).*(n(3:N+2).*v(3:N+2)-n(2:N+1).*v(2:N+1))...
  -.5*(v(2:N+1)+v(1:N)).*(n(2:N+1).*v(2:N+1)-n(1:N).*v(1:N)));
 nnew(1)=2*n0-nnew(2);
 nnew(N+2)=2*nnew(N+1)-nnew(N);
 n=nnew;
 max(n)

if mod(m,5)==1
    plot(x,n);
    axis([0 L  0 2.5])

    pause(.1)
end
% check particle conservation
    t(m)=m*tau;
    nc(m)=sum(n(2:N-1))*h;
end

plot(t,nc)
```

Problem 12.6

Listing 12.3 (lab12p6.m)

```
% Lab Problem 12.6, Physics 430
clear;close all;

N=402;
L=10.;
h=L/N;
x=-.5*h:h:L+.5*h;
x=x'; % turn z into a column vector


n=zeros(N+2,1);
v=zeros(N+2,1);
nnew=zeros(N+2,1);

A=zeros(N+2,N+2);
B=A;

% initial pulse
n0=1;
```

```
n=n0*(1+exp(-200*(x/L-.5).^2));


% step function initial condition
% j=ceil(N/2);
% n=ones(N+2,1);
% n(j:end) = 0;

% v=ones(N+2,1);  % set the convection velocity to unity
 % v=1.2-x/L;  % set a variable convection velocity

taulim=h/max(v);
fprintf(' Courant limit on time step: %g \n',taulim);
tau=input(' Enter the time step - ')


% implicit method
    % load A and B for implicitly calculating the density
    for k=2:N+1
        A(k,k-1)=-v(k-1)/(4*h);
        A(k,k)=1/tau;
        A(k,k+1)=v(k+1)/(4*h);
        B(k,k-1)=v(k-1)/(4*h);
        B(k,k)=1/tau;
        B(k,k+1)=-v(k+1)/(4*h);
    end

    % left side boundary condition: n(1)+n(2)=2
    A(1,1)=1;A(1,2)=1; % the 2 goes in r(1) later
    % right side boundary condition:  n(N+2)=2*n(N+1)-n(N)
    A(N+2,N+2)=1;A(N+2,N+1)=-2;A(N+2,N)=1; % leave the bottom row of B
                                                                    % full of zeros
for m=1:500

   r=B*n;
   r(1)=2;
   n=A\r;

    max(n)

if mod(m,5)==1
    plot(x,n);
    %axis([0 L  0 2.*n0])

    pause(.01)
end
% check particle conservation
    t(m)=m*tau;
    nc(m)=sum(n(2:N+1))*h;
end

plot(t,nc)
```

Chapter 13

Gas Dynamics II

13.1 How to teach this lab

13.2 Solutions

Problem 13.1

Making the replacement $T^n \Rightarrow (T^n + T^{n+1})/2$, we have

$$\frac{T_j^{n+1} - T_j^n}{\tau} = \frac{D_1}{2}\left(T_{j-1}^n + T_{j-1}^{n+1}\right) + \frac{D_2}{2}\left(T_j^n + T_j^{n+1}\right) + \frac{D_3}{2}\left(T_{j+1}^n + T_{j+1}^{n+1}\right)$$

and then rearranging, we get a form where we can read off the matrix elements

$$-\frac{D_1}{2}T_{j-1}^{n+1} + \left(\frac{1}{\tau} - \frac{D_2}{2}\right)T_{j-1}^{n+1} - \frac{D_3}{2}T_{j+1}^{n+1} = \frac{D_1}{2}T_{j-1}^n + \left(\frac{1}{\tau} + \frac{D_2}{2}\right)T_{j-1}^n + \frac{D_3}{2}T_{j+1}^n$$

Then we substitute $\rho^n \Rightarrow \left(\rho^n + \rho^{n+1}\right)/2$ and $v^n \Rightarrow (v^n + v^{n+1})/2$ into the D terms:

$$D_1 = \frac{\left(v_j^n + v_j^{n+1}\right)}{4h} + \frac{2F}{\left(\rho_j^n + \rho_j^{n+1}\right)h^2}$$

$$D_2 = -(\gamma - 1)\frac{\left(v_{j+1}^n + v_{j+1}^{n+1}\right) - \left(v_{j-1}^n + v_{j-1}^{n+1}\right)}{4h} - \frac{4F}{\left(\rho_j^n + \rho_j^{n+1}\right)h^2}$$

$$D_3 = -\frac{\left(v_j^n + v_j^{n+1}\right)}{4h} + \frac{2F}{\left(\rho_j^n + \rho_j^{n+1}\right)h^2}$$

This gives us the Crank-Nicholson algorithm for the interior points.

Problem 13.2

First we do $v^n \Rightarrow \left(v^{n+1} + v^n\right)/2$

$$\frac{v_j^{n+1} - v_j^n}{\tau} = E_0 + \frac{E_1}{2}\left(v_{j-1}^{n+1} + v_{j-1}^n\right) + \frac{E_2}{2}\left(v_j^{n+1} + v_j^n\right) + \frac{E_3}{2}\left(v_{j+1}^{n+1} + v_{j+1}^n\right)$$

and then rearranging, we get a form where we can read off the matrix elements

$$-\frac{E_1}{2}v_{j-1}^{n+1} + \left(\frac{1}{\tau} - \frac{E_2}{2}\right)v_{j-1}^{n+1} - \frac{E_3}{2}v_{j+1}^{n+1} = E_0 + \frac{E_1}{2}v_{j-1}^n + \left(\frac{1}{\tau} + \frac{E_2}{2}\right)v_{j-1}^n + \frac{E_3}{2}v_{j+1}^n$$

Then we do $\rho^n \Rightarrow (\rho^n + \rho^{n+1})/2$ and $\check{v}^n \Rightarrow (\check{v}^n + \check{v}^{n+1})/2$ in the E terms:

$$E_0 = -\frac{k_B}{4hM\left(\rho_j^n + \rho_j^{n+1}\right)}\left[\left(\rho_{j+1}^n + \rho_{j+1}^{n+1}\right)\left(T_{j+1}^n + T_{j+1}^{n+1}\right) - \left(\rho_{j-1}^n + \rho_{j-1}^{n+1}\right)\left(T_{j-1}^n + T_{j-1}^{n+1}\right)\right]$$

$$E_1 = \frac{\left(\check{v}_j^n + \check{v}_j^{n+1}\right)}{4h} + \frac{8\mu}{3\left(\rho_j^n + \rho_j^{n+1}\right)h^2}$$

$$E_2 = -\frac{16\mu}{3\left(\rho_j^n + \rho_j^{n+1}\right)h^2}$$

$$E_3 = -\frac{\left(\check{v}_j^n + \check{v}_j^{n+1}\right)}{4h} + \frac{8\mu}{3\left(\rho_j^n + \rho_j^{n+1}\right)h^2}$$

Code

gas.m

```
% Gas Dynamics in a closed tube, Physics 430
clear;close all;

global gamma kB mu M F A B h tau N;

% Physical Constants
gamma = 1.4;      % Adiabatic Exponent
kappa = 0.024;    % Thermal conductivity
kB = 1.38e-23;    % Boltzman Constant
mu = 1.82e-5;     % Coefficient of viscosity
M = 29*1.67e-27; % Mass of air molecule (Average)
F = (gamma-1)*M*kappa/kB;   % a useful constant

% System Parameters
L=10.0;           % Length of tube
T0 = 293;         % Ambient temperature
rho0 = 1.3;       % static density (sea level)

% speed of sound
c=sqrt(gamma * kB * T0 /M);
```

```
% cell-center grid with ghost points
N=100;
h=L/N;
x=-.5*h:h:L+.5*h;
x=x'; % turn x into a column vector

% initial distributions
rho = rho0 * ones(N+2,1);
T = T0 * ones(N+2,1);
v = exp(-200*(x/L-0.5).^2) * c/100;

% taulim=...;
% fprintf(' Courant limit on time step: %g \n',taulim);
tau=1e-4;     %input(' Enter the time step - ')
tfinal = 0.1; %input(' Enter the run time - ')
nsteps = tfinal/tau;

skip = 5;  %input(' Steps to skip between plots - ')

A=zeros(N+2,N+2);
B=A;

for n=1:nsteps

    time = (n-1) * tau;

    % Plot the current values before stepping
    if mod((n-1),skip)==0
        subplot(3,1,1)
        plot(x,rho);
        ylabel('\rho');
        axis([0 L 1.28 1.32])
        title(['time=' num2str(time)])
        subplot(3,1,2)
        plot(x,T);
        ylabel('T')
        axis([0 L 292 294])
        subplot(3,1,3)
        plot(x,v);
        ylabel('v');
        axis([0 L -4 4])
        pause(0.1)
    end

% 1. Predictor step for rho
    rhop = S_rho(rho,v,v);

% 2. Predictor step for T
    Tp = S_T(T,v,v,rho,rhop);

% 3. Predictor step for v
    vp = S_v(v,v,v,rho,rhop,T,Tp);

% 4. Corrector step for rho
    rhop = S_rho(rho,v,vp);
```

```
% 5. Corrector step for T
    Tp = S_T(T,v,vp,rho,rhop);

% 6. Corrector step for v
    v = S_v(v,v,vp,rho,rhop,T,Tp);

    % Now put rho and T at the same time-level as v
    rho = rhop;
    T = Tp;
end
```

S_rho.m

```
function rho = S_rho(rho,v,vp)
% Step rho forward in time by using Crank-Nicolson
% on the continuity equation

global A B h tau N;

% Clear A and B (pre-allocated in main program)
A=A*0;
B=A;

% Load interior points
for j=2:N+1
    C1 = -tau * (v(j+1) + vp(j+1)) / (8*h);
    C2 = -tau * (v(j-1) + vp(j-1)) / (8*h);
    A(j,j-1)=C2;
    A(j,j)=1;
    A(j,j+1)=-C1;
    B(j,j-1)=-C2;
    B(j,j)=1;
    B(j,j+1)=C1;
end

% Apply boundary condition
fact = -tau/4/h*(vp(2)+v(2)-vp(1)-v(1));
C = (1+fact) / (1-fact);
A(1,1) = 1; A(1,2) = 1;
B(1,1) = C; B(1,2) = C;
fact = -tau/4/h*(vp(N+2)+v(N+2)-vp(N+1)-v(N+1));
C = (1+fact) / (1-fact);
A(N+2,N+1) = 1; A(N+2,N+2) = 1;
B(N+2,N+1) = C; B(N+2,N+2) = C;

% Crank Nicolson solve to step rho in time
r = B*rho;
rho = A\r;

end
```

S_T.m

```
function T = S_T(T,v,vp,rho,rhop)
```

```
global gamma F A B h tau N;

% Clear A and B (pre-allocated in main program)
A=A*0;
B=A;

% Load interior points
for j=2:N+1
    D1 = (v(j) + vp(j))/(4*h) + 2*F/(rho(j)+rhop(j))/h^2;
    D2 = -(gamma-1) * (v(j+1) + vp(j+1) - v(j-1) - vp(j-1) )/(4*h) ...
         - 4*F/(rho(j) + rho(j+1))/h^2;
    D3 = -(v(j) + vp(j))/(4*h) + 2*F/(rho(j)+rhop(j))/h^2;
    A(j,j-1)=-0.5*D1;
    A(j,j)=1/tau - 0.5*D2;
    A(j,j+1)=-0.5*D3;
    B(j,j-1)=0.5*D1;
    B(j,j)=1/tau + 0.5*D2;
    B(j,j+1)=0.5*D3;
end

% Apply boundary condition
% Insulating: dt/dx = 0
A(1,1) = 1; A(1,2) = -1;
A(N+2,N+1) = 1;  A(N+2,N+2) = -1;

% Crank Nicolson solve to step rho in time
r = B*T;
T = A\r;

end
```

S_v.m

```
function v = S_v(v,vbar,vbarp,rho,rhop,T,Tp)

global kB mu M A B h tau N;

% Clear A and B (pre-allocated in main program)
A=A*0;
B=A;

E0 = v * 0; % create the constant term the right size

% Load interior points
for j=2:N+1
    E0(j) = -kB/4/h/M/(rho(j)+rhop(j)) * ...
        ( (rho(j+1) + rhop(j+1)) * (T(j+1) + Tp(j+1)) ...
        - (rho(j-1) + rhop(j-1)) * (T(j-1) + Tp(j-1)));
    E1 = (vbar(j) + vbarp(j))/(4*h) ...
        +8*mu/3/h^2/(rho(j)+rhop(j));
    E2 =-16*mu/3/h^2/(rho(j)+rhop(j));
    E3 =-(vbar(j) + vbarp(j))/(4*h) ...
        +8*mu/3/h^2/(rho(j)+rhop(j));
    A(j,j-1)=-0.5*E1;
    A(j,j)=1/tau - 0.5*E2;
```

```
    A(j,j+1)=-0.5*E3;
    B(j,j-1)=0.5*E1;
    B(j,j)=1/tau + 0.5*E2;
    B(j,j+1)=0.5*E3;
end

% Apply boundary condition
% Fixed: v = 0
A(1,1) = 1; A(1,2) = 1;
A(N+2,N+1) = 1;  A(N+2,N+2) = 1;

% Crank Nicolson solve to step rho in time
r = B*v + E0;
v = A\r;

end
```

Chapter 14

Solitons: Korteweg-deVries Equation

14.1 How to teach this lab

14.2 Solutions

Problem 14.1

Solution: see following Maple page sol14_1.mws

Lab Problem 14.1, Physics 430

```
> restart;
> one:=1/2/tau*(vnew(j)+vnew(j+1)-v(j)-v(j+1));
```

$$one := \frac{1}{2}\,\frac{\mathrm{vnew}(j)+\mathrm{vnew}(j+1)-\mathrm{v}(j)-\mathrm{v}(j+1)}{\tau}$$

```
> two:=(vstar(j+1)+vstar(j))/2/2/h*(vnew(j+1)-vnew(j)+v(j+1)-v(j));
```

$$two := \frac{1}{4}\,\frac{(\mathrm{vstar}(j+1)+\mathrm{vstar}(j))\,(\mathrm{vnew}(j+1)-\mathrm{vnew}(j)+\mathrm{v}(j+1)-\mathrm{v}(j))}{h}$$

```
> three:=alpha/2/h^3*(vnew(j+2)-3*vnew(j+1)+3*vnew(j)-vnew(j-1)+v(j+
  2)-3*v(j+1)+3*v(j)-v(j-1));
```

$$three := \frac{1}{2}\,\alpha\,(\mathrm{vnew}(j+2)-3\,\mathrm{vnew}(j+1)+3\,\mathrm{vnew}(j)-\mathrm{vnew}(j-1)+\mathrm{v}(j+2)-3\,\mathrm{v}(j+1)$$
$$+3\,\mathrm{v}(j)-\mathrm{v}(j-1))/h^3$$

```
> eq:=one+two+three=0;
```

$$eq := \frac{1}{2}\,\frac{\mathrm{vnew}(j)+\mathrm{vnew}(j+1)-\mathrm{v}(j)-\mathrm{v}(j+1)}{\tau}$$
$$+\frac{\frac{1}{4}\,(\mathrm{vstar}(j+1)+\mathrm{vstar}(j))\,(\mathrm{vnew}(j+1)-\mathrm{vnew}(j)+\mathrm{v}(j+1)-\mathrm{v}(j))}{h}+\frac{1}{2}\,\alpha\,(\mathrm{vnew}(j+2)$$
$$-3\,\mathrm{vnew}(j+1)+3\,\mathrm{vnew}(j)-\mathrm{vnew}(j-1)+\mathrm{v}(j+2)-3\,\mathrm{v}(j+1)+3\,\mathrm{v}(j)-\mathrm{v}(j-1))\Big/h^3=0$$

```
> collect(eq,{vnew(j+2),vnew(j+1),vnew(j),vnew(j-1),v(j+2),v(j+1),v(
  j),v(j-1)});
```

$$\left(\frac{1}{2}\,\frac{1}{\tau}+\frac{\frac{1}{4}\,(\mathrm{vstar}(j+1)+\mathrm{vstar}(j))}{h}-\frac{3}{2}\,\frac{\alpha}{h^3}\right)\mathrm{vnew}(j+1)$$
$$+\left(-\frac{1}{2}\,\frac{1}{\tau}-\frac{1}{4}\,\frac{\mathrm{vstar}(j+1)+\mathrm{vstar}(j)}{h}+\frac{\frac{3}{2}\,\alpha}{h^3}\right)\mathrm{v}(j)$$
$$+\left(-\frac{1}{2}\,\frac{1}{\tau}+\frac{\frac{1}{4}\,(\mathrm{vstar}(j+1)+\mathrm{vstar}(j))}{h}-\frac{3}{2}\,\frac{\alpha}{h^3}\right)\mathrm{v}(j+1)+\frac{\frac{1}{2}\,\alpha\,\mathrm{vnew}(j+2)}{h^3}-\frac{1}{2}\,\frac{\alpha\,\mathrm{vnew}(j-1)}{h^3}$$
$$+\frac{\frac{1}{2}\,\alpha\,\mathrm{v}(j+2)}{h^3}-\frac{1}{2}\,\frac{\alpha\,\mathrm{v}(j-1)}{h^3}+\left(\frac{1}{2}\,\frac{1}{\tau}-\frac{1}{4}\,\frac{\mathrm{vstar}(j+1)+\mathrm{vstar}(j)}{h}+\frac{\frac{3}{2}\,\alpha}{h^3}\right)\mathrm{vnew}(j)=0$$

```
>
```

Problem 14.2

Just have them explain the matrices.

Problem 14.3

Solution: just run `kdv.m` as instructed.

Listing 14.1 (kdv.m)

```
% Korteweg-deVries equation on a periodic cell-centered grid using
% Crank-Nicholson

clear; close all;

N=500;
L=10;
h=L/N;
x=h/2:h:L-h/2;
x=x'; % turn x into a column vector

alpha=input(' Enter alpha - ')

vmax=input(' Enter initial amplitude of v - ')

% load an initial Gaussian centered on the computing region
v=vmax*exp(-(x-.5*L).^2);

% choose a time step
tau=input(' Enter the time step - ')

% select the time to run
tfinal=input(' Enter the time to run - ')
Nsteps=ceil(tfinal/tau);

iskip=input(' Enter the plot skip factor - ')

% Initialize the parts of the A and B matrices that
% do not depend on vstar and load them into At and Bt.
% Make them be sparse so the code will run fast.

At=sparse(N,N);
Bt=At;

% Build integer arrays for handling the wrapped points at the ends
% (periodic system)

jm=1:N;
jp=mod(jm,N)+1;
jpp=mod(jm+1,N)+1;
jmm=mod(jm-2,N)+1;

% load the matrices with the terms that don't depend on vstar
for j=1:N
    At(j,jmm(j))=-0.5*alpha/h^3;
```

```
    At(j,jm(j))=0.5/tau+3/2*alpha/h^3;
    At(j,jp(j))=0.5/tau-3/2*alpha/h^3;
    At(j,jpp(j))=0.5*alpha/h^3;
    Bt(j,jmm(j))=0.5*alpha/h^3;;
    Bt(j,j)=0.5/tau-3/2*alpha/h^3;
    Bt(j,jp(j))=0.5/tau+3/2*alpha/h^3;
    Bt(j,jpp(j))=-0.5*alpha/h^3;
end

for n=1:Nsteps

    % do the predictor step
    A=At;B=Bt;
    % load vstar, then add its terms to A and B
    vstar=v;
    for j=1:N
        tmp=0.25*(vstar(jp(j))+vstar(jm(j)))/h;
        A(j,jm(j))=A(j,jm(j))-tmp;
        A(j,jp(j))=A(j,jp(j))+tmp;
        B(j,jm(j))=B(j,jm(j))+tmp;
        B(j,jp(j))=B(j,jp(j))-tmp;

    end

    % do the predictor solve
    r=B*v;
    vp=A\r;

    % corrector step
     A=At;B=Bt;
     % average current and predicted v's to correct vstar
     vstar=.5*(v+vp);
     for j=1:N
        tmp=0.25*(vstar(jp(j))+vstar(jm(j)))/h;
        A(j,jm(j))=A(j,jm(j))-tmp;
        A(j,jp(j))=A(j,jp(j))+tmp;
        B(j,jm(j))=B(j,jm(j))+tmp;
        B(j,jp(j))=B(j,jp(j))-tmp;

    end

    % do the final corrected solve
    r=B*v;
    v=A\r;

    if rem(n-1,iskip)==0
      plot(x,v,'LineWidth',2.5)
      xlabel('x');ylabel('v')
      pause(.1)
    end


end
```

Problem 14.4

Solution: just run `kdv.m` as instructed.

Problem 14.5(a)

Solution: see the following Maple pages sol14_5a.mw

Lab Problem 14.5(a), Physics 430

```
> restart;
```

Write down the Korteweg-deVries equation

```
> kdv:=diff(v(x,t),t)+v(x,t)*diff(v(x,t),x)+alpha*diff(v(x,t),x$3)=0
  ;
```

$$kdv := \left(\frac{\partial}{\partial t} v(x,t)\right) + v(x,t)\left(\frac{\partial}{\partial x} v(x,t)\right) + \alpha\left(\frac{\partial^3}{\partial x^3} v(x,t)\right) = 0$$

Write down the soliton solution

```
> soliton:=12*k^2*alpha/cosh(k*(x-x0-4*alpha*k^2*t))^2;
```

$$soliton := 12\,\frac{k^2\,\alpha}{\cosh(k\,(x - x0 - 4\,\alpha\,k^2\,t))^2}$$

Substitute the soliton solution into the Korteweg-deVries equation

```
> subs(v(x,t)=soliton,kdv);
```

$$\left(\frac{\partial}{\partial t}\left(12\,\frac{k^2\,\alpha}{\cosh(k\,(x - x0 - 4\,\alpha\,k^2\,t))^2}\right)\right) + \frac{12\,k^2\,\alpha\left(\frac{\partial}{\partial x}\left(12\,\frac{k^2\,\alpha}{\cosh(k\,(x - x0 - 4\,\alpha\,k^2\,t))^2}\right)\right)}{\cosh(k\,(x - x0 - 4\,\alpha\,k^2\,t))^2} + \alpha\left(\frac{\partial^3}{\partial x^3}\left(12\,\frac{k^2\,\alpha}{\cosh(k\,(x - x0 - 4\,\alpha\,k^2\,t))^2}\right)\right) = 0$$

Evaluate the derivatives

```
> value(%);
```

$$96\,\frac{k^5\,\alpha^2\sinh(k\,(x - x0 - 4\,\alpha\,k^2\,t))}{\cosh(k\,(x - x0 - 4\,\alpha\,k^2\,t))^3} - \frac{288\,k^5\,\alpha^2\sinh(k\,(x - x0 - 4\,\alpha\,k^2\,t))}{\cosh(k\,(x - x0 - 4\,\alpha\,k^2\,t))^5} + \alpha\left(-288\,\frac{k^5\,\alpha\sinh(k\,(x - x0 - 4\,\alpha\,k^2\,t))^3}{\cosh(k\,(x - x0 - 4\,\alpha\,k^2\,t))^5} + \frac{192\,k^5\,\alpha\sinh(k\,(x - x0 - 4\,\alpha\,k^2\,t))}{\cosh(k\,(x - x0 - 4\,\alpha\,k^2\,t))^3}\right) = 0$$

And simplify to see if the left side vanishes, as it should for a solution

```
> simplify(%);
```

$$0 = 0$$

```
>
```

Problem 14.5(b)-(c)

Listing 14.2 (lab14p5.m)

```
% Lab Problem 14.5, Physics 430

% Korteweg-deVries equation on a periodic cell-centered grid using
% Crank-Nicholson

clear; close all;

N=500;
L=10;
h=L/N;
x=h/2:h:L-h/2;
x=x'; % turn x into a column vector

alpha=input(' Enter alpha - ')

% vmax=input(' Enter initial amplitude of v - ')

k=1.1;
x0=.5*L;
% Build a soliton pulse from alpha, k, and x0
v = 12*k^2*alpha./cosh(k*(x-x0)).^2;

% choose a time step
tau=input(' Enter the time step - ')

% run until t=10, just before the peak comes back in on the left
tfinal=10;
Nsteps=ceil(tfinal/tau);

iskip=input(' Enter the plot skip factor - ')

% Initialize the parts of the A and B matrices that
% do not depend on vstar and load them into At and Bt.
% Make them be sparse so the code will run fast.

At=sparse(N,N);
Bt=At;

% Build integer arrays for handling the wrapped points at the ends
% (periodic system)

jm=1:N;
jp=mod(jm,N)+1;
jpp=mod(jm+1,N)+1;
jmm=mod(jm-2,N)+1;

% load the matrices with the terms that don't depend on vstar
for j=1:N
    At(j,jmm(j))=-0.5*alpha/h^3;
    At(j,jm(j))=0.5/tau+3/2*alpha/h^3;
    At(j,jp(j))=0.5/tau-3/2*alpha/h^3;
```

```
    At(j,jpp(j))=0.5*alpha/h^3;
    Bt(j,jmm(j))=0.5*alpha/h^3;;
    Bt(j,j)=0.5/tau-3/2*alpha/h^3;
    Bt(j,jp(j))=0.5/tau+3/2*alpha/h^3;
    Bt(j,jpp(j))=-0.5*alpha/h^3;
end

for n=1:Nsteps

    % do the predictor step
    A=At;B=Bt;
    % load vstar, then add its terms to A and B
    vstar=v;
    for j=1:N
        tmp=0.25*(vstar(jp(j))+vstar(jm(j)))/h;
        A(j,jm(j))=A(j,jm(j))-tmp;
        A(j,jp(j))=A(j,jp(j))+tmp;
        B(j,jm(j))=B(j,jm(j))+tmp;
        B(j,jp(j))=B(j,jp(j))-tmp;

    end

    % do the predictor solve
    r=B*v;
    vp=A\r;

    % corrector step
     A=At;B=Bt;
     % average current and predicted v's to correct vstar
     vstar=.5*(v+vp);
     for j=1:N
        tmp=0.25*(vstar(jp(j))+vstar(jm(j)))/h;
        A(j,jm(j))=A(j,jm(j))-tmp;
        A(j,jp(j))=A(j,jp(j))+tmp;
        B(j,jm(j))=B(j,jm(j))+tmp;
        B(j,jp(j))=B(j,jp(j))-tmp;

    end

    % do the final corrected solve
    r=B*v;
    v=A\r;

    [peak,j]=max(v);
    xpeak(n)=(j-1)*h;
    t(n)=n*tau;

    % track the position of the peak so that we can measure the velocity


    if rem(n-1,iskip)==0
      plot(x,v)
      pause(.1)
    end
```

```
end

% use polyfit to fit a line to the position of the peak so that we can find
% the speed

a=polyfit(t,xpeak,1);
speed=a(1);
fprintf(' Speed formula: %g  Measured speed: %g \n',4*alpha*k^2,speed)
```

Problem 14.6

Listing 14.3 (lab14p6.m)

```
% Lab Problem 14.6, Physics 430
clear; close all;

% Korteweg-deVries equation on a periodic cell-centered grid using
% Crank-Nicholson


N=500;
L=10;
h=L/N;
x=h/2:h:L-h/2;
x=x'; % turn x into a column vector

alpha=input(' Enter alpha - ')

% vmax=input(' Enter initial amplitude of v - ')

% make an inline function that builds
% a soliton pulse from alpha, k, and x0
f=inline('12*k^2*alpha./cosh(k*(x-x0)).^2','x','alpha','k','x0');

% load two solitons by adding two different
% copies of the inline function
v=f(x,alpha,1.5,.75*L)+f(x,alpha,2.,.25*L);

plot(x,v)

% choose a time step
tau=input(' Enter the time step - ')

% run until t=10, just before the peak comes back in on the left
tfinal=input(' Enter tfinal - ')
Nsteps=ceil(tfinal/tau);

iskip=input(' Enter the plot skip factor - ')

% Initialize the parts of the A and B matrices that
% do not depend on vstar and load them into At and Bt.
% Make them be sparse so the code will run fast.
```

```
At=sparse(N,N);
Bt=At;

% Build integer arrays for handling the wrapped points at the ends
% (periodic system)

jm=1:N;
jp=mod(jm,N)+1;
jpp=mod(jm+1,N)+1;
jmm=mod(jm-2,N)+1;

% load the matrices with the terms that don't depend on vstar
for j=1:N
    At(j,jmm(j))=-0.5*alpha/h^3;
    At(j,jm(j))=0.5/tau+3/2*alpha/h^3;
    At(j,jp(j))=0.5/tau-3/2*alpha/h^3;
    At(j,jpp(j))=0.5*alpha/h^3;
    Bt(j,jmm(j))=0.5*alpha/h^3;;
    Bt(j,j)=0.5/tau-3/2*alpha/h^3;
    Bt(j,jp(j))=0.5/tau+3/2*alpha/h^3;
    Bt(j,jpp(j))=-0.5*alpha/h^3;
end

figure
for n=1:Nsteps

    % do the predictor step
    A=At;B=Bt;
    % load vstar, then add its terms to A and B
    vstar=v;
    for j=1:N
        tmp=0.25*(vstar(jp(j))+vstar(jm(j)))/h;
        A(j,jm(j))=A(j,jm(j))-tmp;
        A(j,jp(j))=A(j,jp(j))+tmp;
        B(j,jm(j))=B(j,jm(j))+tmp;
        B(j,jp(j))=B(j,jp(j))-tmp;
    end

    % do the predictor solve
    r=B*v;
    vp=A\r;

    % corrector step
     A=At;B=Bt;
     % average current and predicted v's to correct vstar
     vstar=.5*(v+vp);
     for j=1:N
        tmp=0.25*(vstar(jp(j))+vstar(jm(j)))/h;
        A(j,jm(j))=A(j,jm(j))-tmp;
        A(j,jp(j))=A(j,jp(j))+tmp;
        B(j,jm(j))=B(j,jm(j))+tmp;
        B(j,jp(j))=B(j,jp(j))-tmp;
    end
```

```
    % do the final corrected solve
    r=B*v;
    v=A\r;

    [peak,j]=max(v);
    xpeak(n)=(j-1)*h;
    t(n)=n*tau;

    % track the position of the peak so that we can measure the velocity

    if rem(n-1,iskip)==0
      plot(x,v)
      pause(.1)
    end

end
```

Chapter 15

Implicit Methods in 2-Dimensions: Operator Splitting

15.1 How to teach this lab

15.2 Solutions

Problem 15.1

Solution: see the following Maple pages sol16_1.mws

Lab Problem 16.1, Physics 430

```
> restart;
```

Write down the two equations, with Lx(Thalf) written so as to not explicitly depend on Thalf

```
> eq1:=(Thalf-Tn)/(tau/2)=D*(LxofThalf+Ly(Tn));
```

$$eq1 := 2\,\frac{Thalf - Tn}{\tau} = \mathrm{D}\,(LxofThalf + \mathrm{Ly}(Tn))$$

```
> eq2:=(Tnew-Thalf)/(tau/2)=D*(LxofThalf+Ly(Tnew));
```

$$eq2 := 2\,\frac{Tnew - Thalf}{\tau} = \mathrm{D}\,(LxofThalf + \mathrm{Ly}(Tnew))$$

Subtract the two equations to find a formal expression for Thalf

```
> eq3:=simplify(eq1-eq2);
```

$$eq3 := -2\,\frac{-2\,Thalf + Tn + Tnew}{\tau} = \mathrm{D}\,\mathrm{Ly}(Tn) - \mathrm{D}\,\mathrm{Ly}(Tnew)$$

```
> Thalf:=solve(eq3,Thalf);
```

$$Thalf := \frac{1}{2}\,Tn + \frac{1}{2}\,Tnew + \frac{1}{4}\,\mathrm{D}\,\mathrm{Ly}(Tn)\,\tau - \frac{1}{4}\,\mathrm{D}\,\mathrm{Ly}(Tnew)\,\tau$$

Equation 2 now looks like this

```
> eq2;
```

$$2\,\frac{\frac{1}{2}\,Tnew - \frac{1}{2}\,Tn - \frac{1}{4}\,\mathrm{D}\,\mathrm{Ly}(Tn)\,\tau + \frac{1}{4}\,\mathrm{D}\,\mathrm{Ly}(Tnew)\,\tau}{\tau} = \mathrm{D}\,(LxofThalf + \mathrm{Ly}(Tnew))$$

Now use the expression for Thalf to convert Lx(Thalf) into an expanded expression

```
> LxofThalf:=1/2*(Lx(Tn)+Lx(Tnew))+1/4*D*tau*Lx(Ly(Tn))-1/4*D*tau*Lx
  (Ly(Tnew));
```

$$LxofThalf := \frac{1}{2}\,\mathrm{Lx}(Tn) + \frac{1}{2}\,\mathrm{Lx}(Tnew) + \frac{1}{4}\,\mathrm{D}\,\tau\,\mathrm{Lx}(\mathrm{Ly}(Tn)) - \frac{1}{4}\,\mathrm{D}\,\tau\,\mathrm{Lx}(\mathrm{Ly}(Tnew))$$

which now puts eq2 into this form

```
> eq2;
```

$$2\,\frac{\frac{1}{2}\,Tnew - \frac{1}{2}\,Tn - \frac{1}{4}\,\mathrm{D}\,\mathrm{Ly}(Tn)\,\tau + \frac{1}{4}\,\mathrm{D}\,\mathrm{Ly}(Tnew)\,\tau}{\tau} =$$

$$\mathrm{D}\left(\frac{1}{2}\,\mathrm{Lx}(Tn) + \frac{1}{2}\,\mathrm{Lx}(Tnew) + \frac{1}{4}\,\mathrm{D}\,\tau\,\mathrm{Lx}(\mathrm{Ly}(Tn)) - \frac{1}{4}\,\mathrm{D}\,\tau\,\mathrm{Lx}(\mathrm{Ly}(Tnew)) + \mathrm{Ly}(Tnew)\right)$$

Now rearrange terms so that it looks as much like Crank-Nicholson as possible

```
> lhsnew:=simplify(lhs(eq2)+1/2*D*Ly(Tn)-1/2*D*Ly(Tnew));
```

$$lhsnew := -\frac{-Tnew + Tn}{\tau}$$

```
> rhsnew:=simplify(rhs(eq2)+1/2*D*Ly(Tn)-1/2*D*Ly(Tnew));
```

$$rhsnew := \frac{1}{2}\,\mathrm{D}\,\mathrm{Lx}(Tn) + \frac{1}{2}\,\mathrm{D}\,\mathrm{Lx}(Tnew) + \frac{1}{4}\,\mathrm{D}^2\,\tau\,\mathrm{Lx}(\mathrm{Ly}(Tn)) - \frac{1}{4}\,\mathrm{D}^2\,\tau\,\mathrm{Lx}(\mathrm{Ly}(Tnew)) + \frac{1}{2}\,\mathrm{D}\,\mathrm{Ly}(Tnew) + \frac{1}{2}\,\mathrm{D}\,\mathrm{Ly}(Tn)$$

On the left we recognize the forward time difference of Crank-Nicholson, and on the right we also see the time-averaged spatial operators. The leftover term on the right is

```
> leftover:=-D^2*tau/4*Lx(Ly(Tnew-Tn));
```

$$leftover := -\frac{1}{4}\,\mathrm{D}^2\,\tau\,\mathrm{Lx}(\mathrm{Ly}(Tnew - Tn))$$

which becomes, if we interpret Tnew-Tn as a time derivative by putting τ underneath it

```
> leftover:=-D^2*tau^2/4*Diff(Diff(Diff(T(x,y,t),t),y$2),x$2);
```

$$leftover := -\frac{1}{4}\,\mathrm{D}^2\,\tau^2\left(\frac{\partial^5}{\partial x^2\,\partial y^2\,\partial t}\,\mathrm{T}(x, y, t)\right)$$

```
>
```

Problem 15.2

Listing 15.1 (lab15p2.m)

```
% Lab Problem 15.2, Physics 430

% this code solves the diffusion equation using tridag.m
% to implement Crank-Nicholson

% simple introductory problem, zero boundary conditions all around

clear; close all;

N=input(' Enter N - ')
L=5;
h=2*L/(N-1);
x=-L:h:L;
y=x;

for i=1:N
    for j=1:N
        T(i,j)=cos(.5*pi*x(i)/L)*cos(.5*y(j)*pi/L);
        T0(i,j)=T(i,j);
    end
end


% apply zero boundary conditions
T(1,:)=0;T(:,1)=0;T(N,:)=0;T(:,N)=0;

% get the maximum of T for setting the plot frame
Tmax=2.1;Tmin=0;

% load the diffusion coefficient
D=2.3;

% Suggest a timestep by giving the time step for
% explicit stability, then ask for the time step

fprintf(' Maximum explicit time step: %g \n',h^2/max(D))
tau = input(' Enter the time step - ')

tfinal=input(' Enter the total run time - ')

% set the number of time steps to take
nsteps=tfinal/tau;

% define a useful constant
coeff=tau/h^2 ;

% load the tridiagonal matrix A with the left side of Crank-Nicholson
% operator and B-, B0, and B+, assuming that the x and y grids are
% identical in size
A=zeros(N);
Bm=D*.5*coeff;
```

```
B0=1-D*coeff;
Bp=Bm;

% put the boundary conditions in: zero values at each end
A(1,1)=1;A(N,N)=1;


for j=2:N-1

   A(j,j-1) = -D*.5*coeff;
   A(j,j) = (1+D*coeff);
   A(j,j+1) = -D*.5*coeff;
end

% this is the time advance loop

% define the right-hand side vector.
r=zeros(N,1);

for m=1:nsteps

% Crank-Nicholson advance
% first do the fully implicit step in x by
% doing a solve in i(x) at each j(y)

for j=2:N-1 % start of j-loop

% load the right-hand side vector
for i=2:N-1
    r(i)=Bm*T(i,j-1)+B0*T(i,j)+Bp*T(i,j+1);
end
% set the boundary conditions in i
r(1)=0;
r(N)=0;

% do the solve to find Thalf
tmp=A\r;

Thalf(:,j)=tmp;
end % end of j-loop
% set the boundary conditions in j
Thalf(:,1)=0;Thalf(:,N)=0;

% now do the second implicit step in y by
% doing a solve in j(y) at each i(x)

for i=2:N-1 % start i-loop

    % load r
    for j=2:N-1
        r(j)=Bm*Thalf(i-1,j)+B0*Thalf(i,j)+Bp*Thalf(i+1,j);
    end
    r(1)=0;r(N)=0; % set the boundary conditions in j

    % do the final solve
```

```
    tmp=A\r;

    T(i,:)=tmp';

end % end of i-loop
% set the boundary conditions in i
T(1,:)=0;T(N,:)=0;


  t=m*tau; % set the time

  if rem(m,5)==1
   % check the error by comparing with the exact solution
   Texact=T0*exp(-2*pi^2*.5^2/L^2*t*D);
   chk=max(max(abs(T-Texact)))

   surf(x,y,T)
   axis([ -L L -L L 0 1.2])


   pause(.1)
   end

end
```

Problem 15.3

Listing 15.2 (lab15p3.m)

```
% Lab Problem 15.3, Physics 430
clear; close all;

% this code solves the diffusion equation using tridag.m
% to implement Crank-Nicholson

% boundary conditions modified to zero derivative at the edges


N=input(' Enter N - ')
L=5;
h=2*L/(N-1);
x=-L:h:L;
y=x;

% the initial state doesn't satisfy the boundary conditions, but diffusion will
% quickly take care of that.
for i=1:N
    for j=1:N
        T(i,j)=cos(.5*pi*x(i)/L)*cos(.5*y(j)*pi/L);
    end
end
```

```
% get the maximum of T for setting the plot frame
Tmax=2.1;Tmin=0;

% load the diffusion coefficient
D=2.3;

% Suggest a timestep by giving the time step for
% explicit stability, then ask for the time step

fprintf(' Maximum explicit time step: %g \n',h^2/max(D))
tau = input(' Enter the time step - ')

tfinal=input(' Enter the total run time - ')

% set the number of time steps to take
nsteps=tfinal/tau;

% define a useful constant
coeff=tau/h^2 ;

% load the tridiagonal matrix A with the left side of Crank-Nicholson
% operator and B-, B0, and B+, assuming that the x and y grids are
% identical in size
A=zeros(N);
Bm=D*.5*coeff;
B0=1-D*coeff;
Bp=Bm;



% put the boundary conditions in, zero derivative.
% We are on a cell edge grid, so fit a parabola to the
% first three points, then evaluate its derivative at
% x=0 (or x=L) and set it to zero.  The result is
% T'(0)=-1.5/h*T(0)+2/h*T(h)-0.5/h*T(2*h)

A(1,1)=-1.5/h;A(1,2)=2/h;A(1,3)=-.5/h;
A(N,N)=-1.5/h;A(N,N-1)=2/h;A(N,N-2)=-.5/h;


for j=2:N-1
   A(j,j-1) = -D*.5*coeff;
   A(j,j) = (1+D*coeff);
   A(j,j+1) = -D*.5*coeff;
end

% this is the time advance loop

% define the right-hand side vector.
r=zeros(N,1);

for m=1:nsteps

% Crank-Nicholson advance
% first do the fully implicit step in x by
```

```
% doing a solve in i(x) at each j(y)

for j=2:N-1 % start of j-loop

% load the right-hand side vector
for i=2:N-1
    r(i)=Bm*T(i,j-1)+B0*T(i,j)+Bp*T(i,j+1);
end
% set the boundary conditions in i
r(1)=0;
r(N)=0;

% do the solve to find Thalf
tmp=A\r;

Thalf(:,j)=tmp;
end % end of j-loop
% set the boundary conditions in j
Thalf(:,1)=4/3*Thalf(:,2)-1/3*Thalf(:,3);
Thalf(:,N)=4/3*Thalf(:,N-1)-1/3*Thalf(:,N-2);

% now do the second implicit step in y by
% doing a solve in j(y) at each i(x)

for i=2:N-1 % start i-loop

    % load r
    for j=2:N-1
        r(j)=Bm*Thalf(i-1,j)+B0*Thalf(i,j)+Bp*Thalf(i+1,j);
    end
    r(1)=0;r(N)=0; % set the boundary conditions in j

    % do the final solve
    tmp=A\r;

    T(i,:)=tmp';

end % end of i-loop
% set the boundary conditions in i
T(1,:)=4/3*T(2,:)-1/3*T(3,:);
T(N,:)=4/3*T(N-1,:)-1/3*T(N-2,:);


  t=m*tau; % set the time

   surf(x,y,T)
   axis([ -L L -L L 0 1.2])


   pause(.1)

end
```

Problem 15.4

Listing 15.3 (lab15p4.m)

```
% Lab Problem 15.4, Physics 430
clear; close all;

% this code solves the diffusion equation using tridag.m
% to implement Crank-Nicholson

% Zero at the edges, and in a square at large x and y,
% leaving a region defined by mask(i,j) in the interior
% set to zero


N=input(' Enter N - ')
N2=ceil(N/2); % for i and j at N2 and beyond T=0
L=5;
h=2*L/(N-1);
x=-L:h:L;
y=x;

% these initial conditions don't match the L-shape, but diffusion will
% take care of it in a few steps
for i=1:N
    for j=1:N
        T(i,j)=cos(.5*pi*x(i)/L)*cos(.5*y(j)*pi/L);

    end
end

% set the mask

for i=1:N
    for j=1:N
        % ellipse in the middle
        chk=x(i)^2/9+y(j)^2/4;
        if chk < 1
           mask(i,j)=1;
       else
            mask(i,j)=0;
        end
    end
end

% get the maximum of T for setting the plot frame
Tmax=2.1;Tmin=0;

% load the diffusion coefficient
D=2.3;

% Suggest a timestep by giving the time step for
% explicit stability, then ask for the time step

fprintf(' Maximum explicit time step: %g \n',h^2/max(D))
```

```
tau = input(' Enter the time step - ')

tfinal=input(' Enter the total run time - ')

% set the number of time steps to take
nsteps=tfinal/tau;

% define a useful constant
coeff=tau/h^2 ;

% load the tridiagonal matrix A with the left side of Crank-Nicholson
% operator and B-, B0, and B+, assuming that the x and y grids are
% identical in size
A=zeros(N);
Bm=D*.5*coeff;
B0=1-D*coeff;
Bp=Bm;

% put the boundary conditions in: zero values at each end
A(1,1)=1;
A(N,N)=1;

for j=2:N-1
   A(j,j-1) = -D*.5*coeff;
   A(j,j) = (1+D*coeff);
   A(j,j+1) = -D*.5*coeff;
end



% this is the time advance loop

% define the right-hand side vector.
r=zeros(N,1);

for m=1:nsteps

% Crank-Nicholson advance
% first do the fully implicit step in x by
% doing a solve in i(x) at each j(y)

for j=2:N-1 % start of j-loop

% load the right-hand side vector and the
% mask-modified operator Ap
Ap=A;
for i=2:N-1

    if mask(i,j) == 1
       r(i)=0;
       Ap(i,i-1)=0;
       Ap(i,i)=1.;
       Ap(i,i+1)=0;

    else
```

```
       r(i)=Bm*T(i,j-1)+B0*T(i,j)+Bp*T(i,j+1);
    end

end

% set the boundary conditions in i
r(1)=0;
r(N)=0;


% do the solve to find Thalf
tmp=Ap\r;

Thalf(:,j)=tmp;

end % end of j-loop

% set the boundary conditions in j
Thalf(:,1)=0;
Thalf(:,N)=0;

% now do the second implicit step in y by
% doing a solve in j(y) at each i(x)


for i=2:N-1 % start i-loop

    % load r and Ap, the mask modified A
    Ap=A;
    for j=2:N-1
        if mask(i,j) == 1
           r(j)=0.;
           Ap(j,j-1)=0.;
           Ap(j,j)=1.;
           Ap(j,j+1)=0.;
        else
           r(j)=Bm*Thalf(i-1,j)+B0*Thalf(i,j)+Bp*Thalf(i+1,j);
         end

    end
    r(1)=0;r(N)=0; % set the boundary conditions in j
    % for the masked region

    % do the final solve
    tmp=A\r;

    T(i,:)=tmp';

end % end of i-loop

% set the boundary conditions in i
T(1,:)=0;
T(N,:)=0;

  t=m*tau; % set the time
```

```
    surf(x,y,T)
    axis([ -L L -L L 0 1.2])
    view(41,41)


    pause(.1)

end
```