Computational Methods

Francisco Castillo
Homework 2

March 4, 2018

Problem 1

Considering using cubic splines to interpolate the function

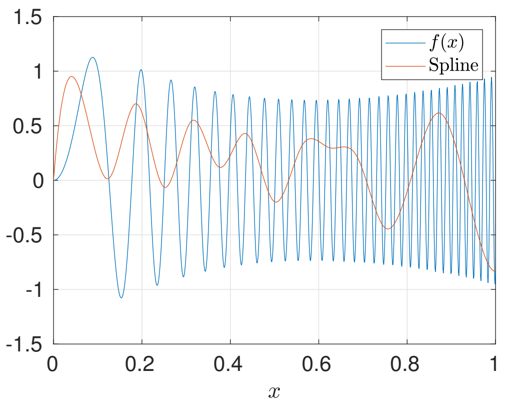
$$f(x) = \frac{e^{3x}\sin(200x^2)}{1+20x^2}, \quad 0 \le x \le 1.$$

Write a short MATLAB script using spline, interpolating this function at equidistant points $x_i = \frac{i}{n}$, $i = 0, 1, ..., n$. Repeat this for $n = 2^j$, $j = 4, 5, ..., 14$. For each calculation record the maximum error at the point $X = 0 : .001 : 1$. Plot these errors against n, using loglog. Make observations in comparison to Figure 10.8.

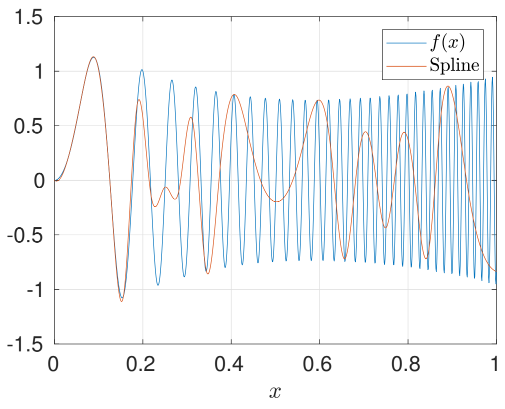
In this problem we explore spline approximation. Given the function above we are going to compute spline approximations for different number of interpolation points. In figure 1 we can see how for $n = 16$ interpolation points the approximation is not good enough. Using $n = 32$ interpolation points the approximation is good at the beginning of the domain, up to $x = 0.2$, but soon it starts to be far from the actual value of the function. Incrementing the number of interpolation points once more to $n = 64$ we see how the approximation matches the function further in the domain, up to $x = 0.3$. This behaviour goes on until we use $n = 256$ nodes, that the approximation seems to match the function very accurately. Incrementing number of interpolation points to $n = 512$ doesn't seem to offer an appreciable improvement in accuracy. However, we will see in figure 2 that this is not true. The figures for higher n are not shown since they are indistinguishable from (f).

In figure 2 we show the maximum error of the approximation against the number of interpolation points used. We can see how incresing n from 256 to 512 reduces the maximum error in an order of magnitude, although we are not able to see it in the plots of figure 1. Notice the high frequency of the function to approximate, this requires high number of points for the spline interpolant to be accurate.

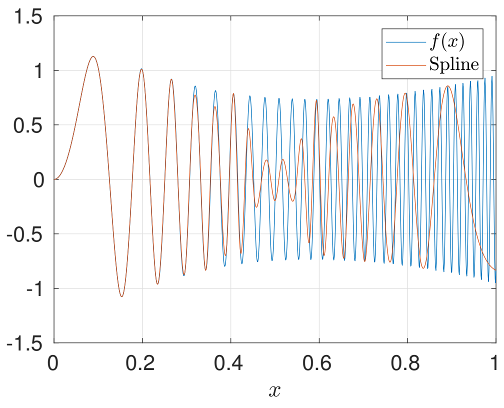
In figure 3 is a *semi-log* version of figure 2, we show it to compare to the figure in the book mentioned by the problem. We can see how with only 201 interpolation points the Chebyshev's interpolation outcomes the spline we have done. The maximum error for the Chebyshev's interpolation using 201 interpolation points is of the order of 10^{-14} and we haven't reached that level of accuracy using up to 16384 points.



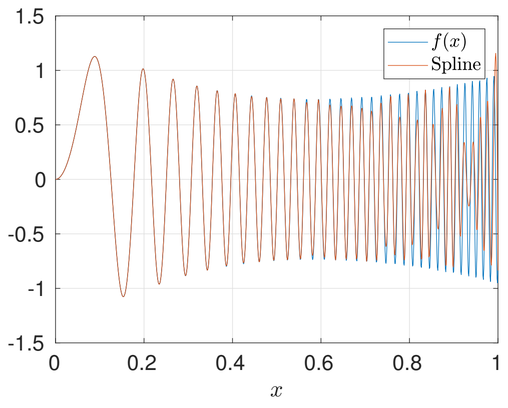
(a) $n = 16$.



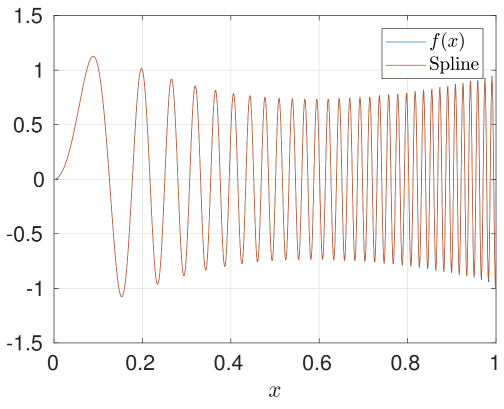
(b) $n = 32$.



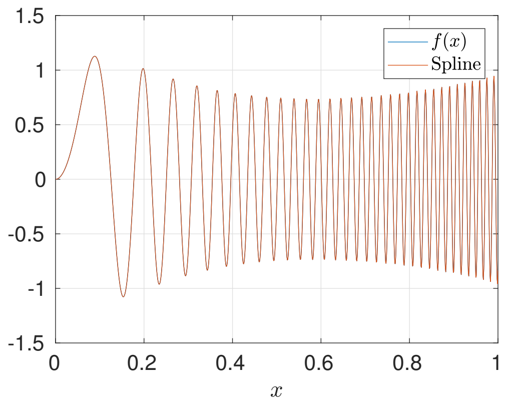
(c) $n = 64$.



(d) $n = 128$.



(e) $n = 256$.



(f) $n = 512$.

Figure 1: Spline approximation of the function $f(x)$ defined above.

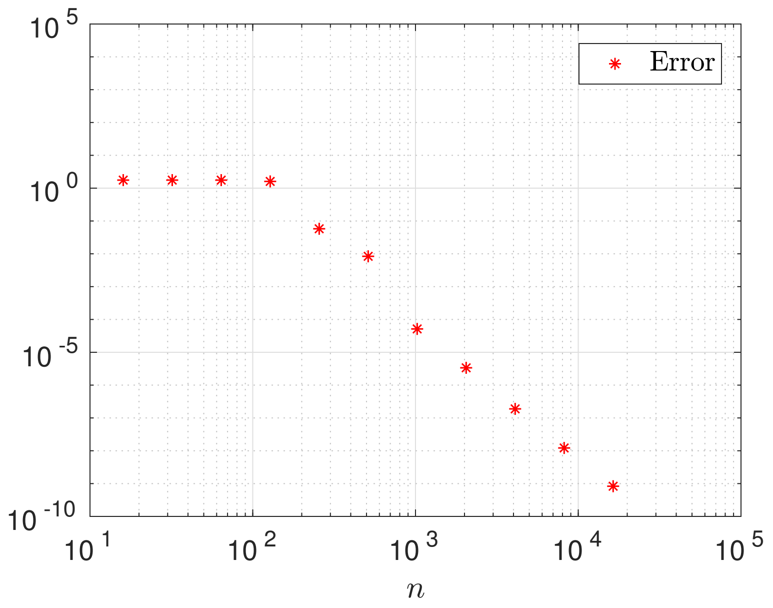


Figure 2: Error of the spline approximation.

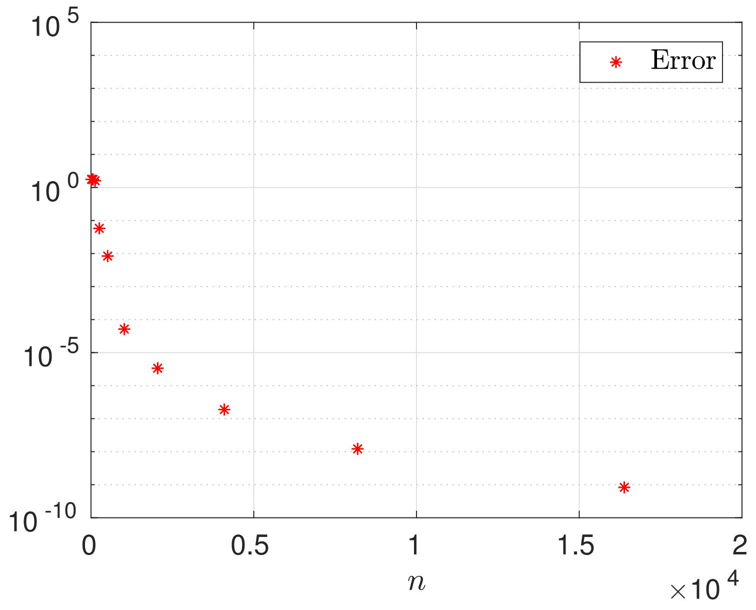


Figure 3: Error of the spline approximation.

Matlab code for this problem

```
%% Problem 1
clear all; close all; format long; clc
legendfontsize=14;
axisfontsize=16;
f = @(x) (exp(3*x).*(sin(200*x.^2)))./(1 + 20*x.^2);
```

```
xx=0:0.001:1;
for j=4:14
    n=2^j;
    x=linspace(0,1,n+1);
    fx=f(x);
    fxx=spline(x,fx,xx);
    error=max(abs(f(xx)-fxx));

    figure(1)
    loglog(n,error,'r*')
    hold on
    grid on
    if j==14
        xlabel('$n$','Interpreter','latex')
        set(gca,'fontsize',14)
        legend({'Error'},...
         'Interpreter','latex','fontsize',legendfontsize)
        txt='Latex/FIGURES/P1_Error';
        saveas(gcf,txt,'epsc')
    end
    figure(2)
    semilogy(n,error,'r*')
    hold on
    grid on
    if j==14
        xlabel('$n$','Interpreter','latex')
        set(gca,'fontsize',14)
        legend({'Error'},...
         'Interpreter','latex','fontsize',legendfontsize)
        txt='Latex/FIGURES/P1_Error_semilog';
        saveas(gcf,txt,'epsc')
    end

    figure
    plot(xx,f(xx))
    hold on
    plot(xx,fxx)
    grid on
    xlabel('$x$','Interpreter','latex')
    legend({'$f(x)$','Spline'},...
        'Interpreter','latex','fontsize',legendfontsize)
    set(gca,'fontsize',axisfontsize)
    txt=['Latex/FIGURES/P1_n=',num2str(n)];
    saveas(gcf,txt,'epsc')
    end
```

Problem 2

Modify the code to `diffusion_eq1_FD.m` with sparse matrices

In this problem we want to show the need to take advantage of some properties of matrices. In particular, the need to use the fact that in some problems the matrices involved have a majority of its entries being zero. This type of matrices are called sparse matrices. If we code using this fact, the computer will store the value and position of the nonzero entries instead of the whole matrix, resulting in a much short computation time, as we will show. The code generates the following solution to the diffusion differential equation (for the case of $N = 100$ nodes).

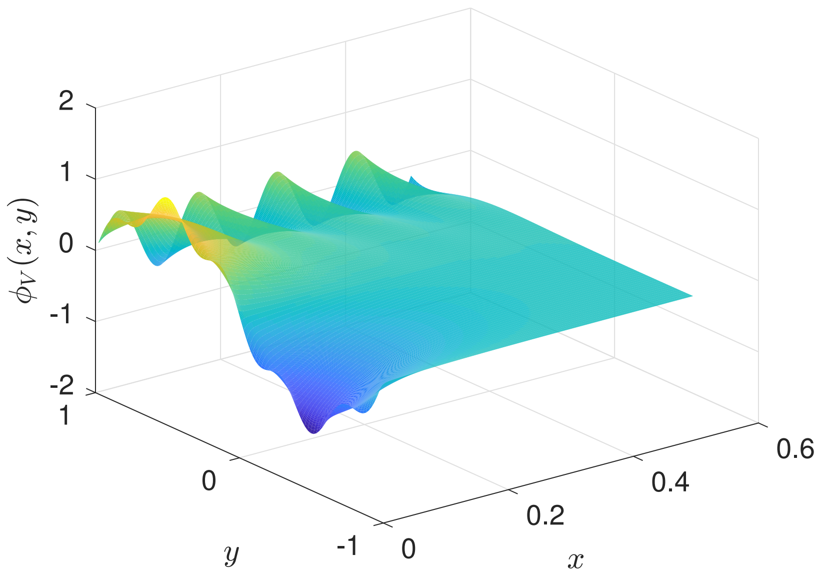


Figure 4: Solution of the diffusion equation, $N = 100$.

We have measured the time used to calculate the solution for different number of nodes. In figure 2 we can see how using sparse matrices makes our code run faster and the bigger N the bigger the difference between the two speeds.

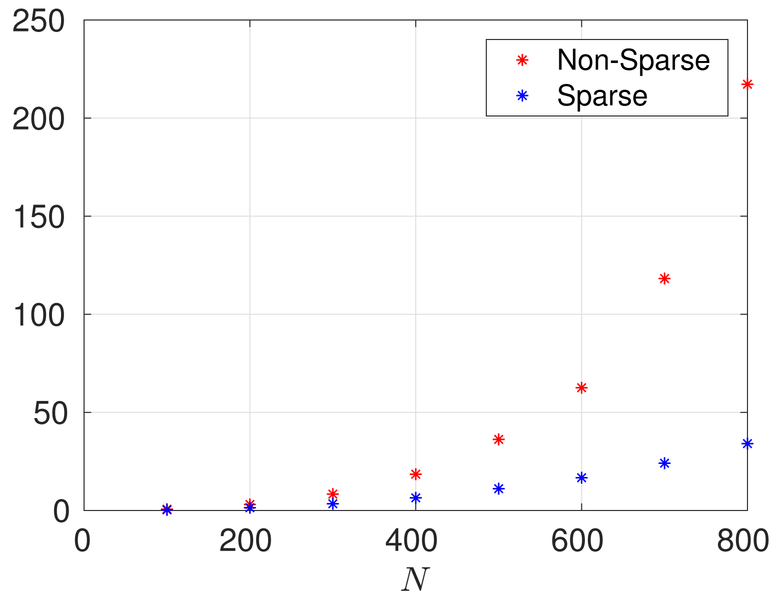


Figure 5: Elapsed time for regular and sparse matrices.

In the next table we can see the measured times for both cases for different values for N as well as the ratio between the two. As we could observe in figure 2, the ratio gets bigger as we increase the dimesion of the matrices, which proves the importance of taking advantage of sparse matrices specially for bigger systems.

N	Regular Matrices	Sparse Matrices	Ratio
100	0.6716	0.3347	2.0064
200	3.0701	1.4052	2.1848
300	8.3650	3.4260	2.4416
400	18.4907	6.4504	2.8666
500	36.2399	11.1142	3.2607
600	62.5694	16.6995	3.7468
700	118.2346	24.1097	4.9040
800	217.2509	34.0877	6.3733

Table 1: L_2 error norm of different methods.

Matlab code for this problem

```
%% Problem 2
clear all; close all
legendfontsize=14;
axisfontsize=14;
labelfontsize=16;
%% Original code, non-sparse matrices.
time=zeros(8,1);
for j=1:8
```

```
    tic
    N = 100*j;
    x = linspace(-1,1,N)';
    dx = x(2)-x(1);

    unos = ones(N-2,1);
    D2 = ( diag(unos(1:end-1),-1)+diag(unos(1:end-1),1)-2*diag(unos,0) )/dx^2;
    B = zeros(N-2,2);
    B(1,1) = 1/dx^2; B(end,end) =1/dx^2;
    % initial condition
    u0 = sin(pi*x)+0.2*sin(5*pi*x);
    % boundary conditions
    g1 = @(t) .5*sin(50*t);
    g2 = @(t) 0*t;

    t = 0:0.001:.5;

    [T,U] = ode45(@(t,u) D2*u+B*[g2(t);g1(t)], t, u0(2:end-1));
%     for k = 2:length(t)
%         plot(x,[0;U(k,:)';g1(t(k))],'*-')
%         ylim([-1.5 1.5])
%         shg
%         drawnow
%     end
    time(j)=toc;
    if N==100
        figure
        [T,X]= meshgrid(t,x(2:end-1));
        surf(T,X,U','edgecolor','none')
        xlabel('$x$','fontsize',labelfontsize,...
            'interpreter','latex')
        ylabel('$y$','fontsize',labelfontsize,...
            'interpreter','latex')
        zlabel('$\phi_V(x,y)$','fontsize',labelfontsize,...
            'interpreter','latex')
        set(gca,'fontsize',axisfontsize)
        txt='Latex/FIGURES/P2_surf1';
        saveas(gcf,txt,'epsc')
    end
end

%% Modified code, sparse matrices.
sparsetime=zeros(8,1);
for j=1:8
    tic
    N = 100*j;
    x = linspace(-1,1,N)';
    dx = x(2)-x(1);

    unos = ones(N-2,1);
    D2 = sparse(( diag(unos(1:end-1),-1)+diag(unos(1:end-1),1)-2*diag(unos,0) )/dx^2);
    B = sparse(zeros(N-2,2));
    B(1,1) = 1/dx^2; B(end,end) =1/dx^2;
    % initial condition
    u0 = sin(pi*x)+0.2*sin(5*pi*x);
    % boundary conditions
    g1 = @(t) .5*sin(50*t);
    g2 = @(t) 0*t;

    t = 0:0.001:.5;

    [T,U] = ode45(@(t,u) D2*u+B*[g2(t);g1(t)], t, u0(2:end-1));
%     for k = 2:length(t)
%         plot(x,[0;U(k,:)';g1(t(k))],'*-')
%         ylim([-1.5 1.5])
%         shg
%         drawnow
%     end
    sparsetime(j)=toc;
    if N==100
        figure
        [T,X]= meshgrid(t,x(2:end-1));
        surf(T,X,U','edgecolor','none')
        txt='Latex/FIGURES/P2_surf2';
        saveas(gcf,txt,'epsc')
    end
end
%%
figure
Nj=100:100:800;
plot(Nj,time,'r*',Nj,sparsetime,'b*')
legend('Non-Sparse','Sparse')
grid on
txt='Latex/FIGURES/P2_times';
saveas(gcf,txt,'epsc')
```

Problem 3

Modify the code diffusion `eq1_Neumann.m` to solve the diffusion equation with Neumann boundary conditions using finite differences.

In this problem we use second order centered finite differences to calculate the second derivative in the interior

$$u_j'' = \frac{u_{j-1} - 2u_j + u_{j+1}}{h^2}.$$

This discretization, if expressed in matrix form, gives us a tridiagonal matrix with $1, -2, 1$ as lower diagonal, diagonal and upper diagonal, respectively. We only need to solve the following matrix equation

$$\frac{d}{dt}\mathbf{u} = D_2\mathbf{u},$$

for the interior since we have the boundary conditions. They have been taking into account using a first order sided finite differences

$$u_1' = \frac{u_2 - u_1}{h} = g_1(t) \;\Rightarrow\; u_1 = u_2 - hg_1(t),$$

$$u_{N+1}' = \frac{u_{N+1} - u_N}{h} = g_2(t) \;\Rightarrow\; u_{N+1} = hg_2(t) + u_N,$$

We will get rid of the first and last row and column the matrix D_2, obtaining $\tilde{D}_2$ and add another term to account for the boundary conditions of the Neumann kind in this case. The system obtained is the following

$$\frac{d}{dt}\tilde{\mathbf{u}} = \tilde{D}_2\tilde{\mathbf{u}} + \frac{1}{h}\begin{pmatrix} -1 & 1 \end{pmatrix}\begin{pmatrix} g_1(t) \\ g_2(t) \end{pmatrix},$$

where $g_1(t)$ and $g_2(t)$ are the Neumann boundary condition functions at $x = -1$ and $x = 1$, respectively. We defined them generally althought in this problem $g_1(t) = g_2(t) = 1$. When solving for the interior, we will need the values at the boundaries that will be substituted by the two equations above. As a result, including the boundary conditions hasn't just altered the system adding that extra term, but it also changes the first and last entries of the matrix $\tilde{D}_2$. Computing the first and last nodes of the interior, and including the boundary conditions, we have

$$\begin{aligned} u_2'' = \frac{u_1 - 2u_2 + u_3}{h^2} &= \frac{u_2 - hg_1(t) - 2u_2 + u_3}{h^2} \\ &= \frac{-u_2 + u_3}{h^2} - \frac{1}{h}g_1(t), \end{aligned}$$

and

$$\begin{aligned} u_N'' = \frac{u_{N-1} - 2u_N + u_{N+1}}{h^2} &= \frac{u_{N-1} - 2u_N + hg_2(t) + u_N}{h^2} \\ &= \frac{u_{N-1} - u_N}{h^2} + \frac{1}{h}g_2(t). \end{aligned}$$

This implies that the system is modified as showed above and the first and second entries of the matrix $\tilde{D}_2$ are

$$\tilde{D}_2(1,1) = \tilde{D}_2(N-1,N-1) = -\frac{1}{h^2}.$$

Note that the $1/h^2$ factor is already included in $\tilde{D}_2$. The solution for the PDE is shown in the next figure.

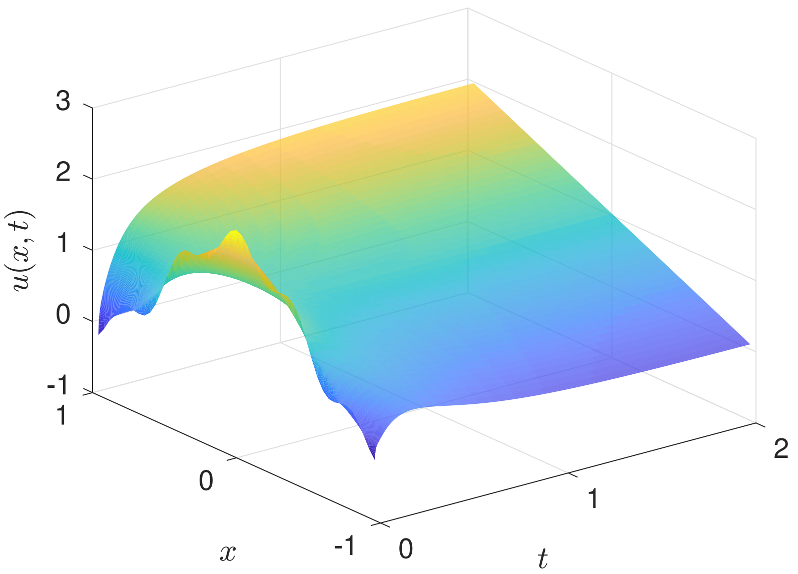


Figure 6: Solution of the diffusion equation, $N = 50$.

We can see how at time zero we have the initial condition and in two seconds we reach the steady state where the whole domain has a slope 1, as imposed by the boundary conditions in the first and last nodes.

Matlab code for this problem

```
%% Problem 3
% Simple code to solve the diffusion equation
% u_t = u_xx -1<x<1, with Neumann boundary conditions
clear all; close all; clc
legendfontsize=14;
axisfontsize=14;
labelfontsize=16;
N = 50;
L=2;
h=L/N;
x=-1:h:1;
D2 = gallery('tridiag',N+1,1,-2,1); % In sparse form.
%%
```

```
% initial condition
u0 = cos(pi*x)+0.2*cos(5*pi*x)+1;
% boundary conditions
g1 = @(t) 0*t+1;
g2 = @(t) 0*t+1;

A = D2(2:N,2:N)/h^2;
A(1,1)=-1/h^2;
A(end,end)=-1/h^2;
B = zeros(N-1,2);
B(1,1) = -1/h;
B(end,end) = 1/h;

t = 0:0.001:2;
%%
[T,U] = ode45(@(t,u) A*u+B*[g1(t);g2(t)], t, u0(2:end-1));
for k = 2:length(t)
    plot(x(2:end-1),U(k,:)','*-')
    ylim([-1 2])
    shg
    drawnow
end
[T,X]= meshgrid(t,x(2:end-1));
surf(T,X,U','edgecolor','none')
xlabel('$t$','fontsize',labelfontsize,...
            'interpreter','latex')
ylabel('$x$','fontsize',labelfontsize,...
            'interpreter','latex')
zlabel('$u(x,t)$','fontsize',labelfontsize,...
            'interpreter','latex')
set(gca,'fontsize',axisfontsize)
txt='Latex/FIGURES/P3_surf';
saveas(gcf,txt,'epsc')
```

Problem 4

Consider the numerical solution of the wave equation with Dirichlet boundary condition:

$$u_{tt} = u_{xx}, \quad x \in (1,1), \quad u(t,1) = u(t,1) = 0, \quad u(0,x) = e^{-32x^2}, \quad u_t(0,x) = 0.$$

Assuming sufficient smoothness in u and appropriate initial conditions, show that solving (1) is equivalent to solving the acoustic equations

$$\begin{aligned} u_t &= p_x, \\ p_t &= u_x, \end{aligned}$$

with boundary conditions $u(t,1) = u(t,1) = 0$. Find the appropriate initial condition for p.

Solve this problem using second order centered finite differences in space and *ode*45 in time. Use a staggered equispaced grid. That is, discretize u at the gridpoints $x_j = 1 + jh$ and p at the nodes $x_{\frac{j+1}{2}} = 1 + (j + \frac{1}{2})h$. Notice that boundary conditions for p are not required in this case. Plot your solution (u and p on the same figure) for $t = 0.5, 1, 1.5, 2$.

Estimate the accuracy of your answer at $t = 4$, notice that $u(4,x) = u(0,x)$.

For the first part note that

$$u_{tt} = \frac{\partial}{\partial t} p_x = p_{xt},$$

and

$$u_{xx} = \frac{\partial}{\partial x} p_t = p_{tx}.$$

Since u is sufficiently smooth, both p_{xt} and p_{tx} are continuous. Therefore they must be equal and

$$u_{tt} = p_{xt} = p_{tx} = u_{xx}.$$

To find the initial condition for p we simply use the initial conditions for u,

$$p_t(0,x) = u_x(0,x) = -64xe^{-32x^2}.$$

and

$$p_x(0,x) = u_t(0,x) = 0.$$

For the second part of the problem, let us express the acoustic equations in matrix form

$$\frac{d}{dt}\begin{bmatrix} u \\ p \end{bmatrix} = \begin{bmatrix} 0 & \frac{\partial}{\partial x} \\ \frac{\partial}{\partial x} & 0 \end{bmatrix} \begin{bmatrix} u \\ p \end{bmatrix}.$$

We are using staggered meshes so the space derivative operator for u and p are going to be different, letting the previous equation in matrix form be like

$$\frac{d}{dt}\begin{bmatrix} u \\ p \end{bmatrix} = \begin{bmatrix} 0 & D_p \\ D_u & 0 \end{bmatrix}\begin{bmatrix} u \\ p \end{bmatrix},$$

or

$$\frac{d}{dt}\mathbf{V} = M\mathbf{V},$$

where $\mathbf{V} = \begin{bmatrix} u \\ p \end{bmatrix}$ and M is the matrix composed by D_u, D_p and zeros. Since we have $N+1$ nodes of u and we only solve for the interior, our matrix D_u will only have $N-1$ columns. However, we want to evaluate u_x in the N nodes for p, therefore D_u will have N rows. Thus, D_u is $N \times N-1$. We are going to use central finite differences to evaluate the first derivative of u, but since we are going to evaluate the result in the staggered p nodes we have

$$\frac{\partial}{\partial t}p_j = \frac{\partial}{\partial x}u_{j+\frac{1}{2}} = \frac{u_{j+1} - u_j}{h},$$

where $j = 1, ...N$, since we are using *Matlab* indices. Since we are only solving the interior for u there is going to be a shift in the matrix and the real system we are going to solve is the following

$$\frac{\partial}{\partial t}\begin{bmatrix} u \\ p \end{bmatrix} = \begin{bmatrix} 0 & D_p \\ \tilde{D}_u & 0 \end{bmatrix}\begin{bmatrix} u \\ p \end{bmatrix},$$

or

$$\frac{\partial}{\partial t}\mathbf{V} = \tilde{M}\mathbf{V}.$$

Therefore, the discretization yields

$$\frac{\partial}{\partial t}p_j = \frac{\partial}{\partial x}\tilde{u}_{j-\frac{1}{2}} = \frac{\tilde{u}_j - \tilde{u}_{j-1}}{h}.$$

Note that $\tilde{u}_j = u_{j+1}$. For the first and last points we include the boundary conditions,

$$\frac{\partial}{\partial t}p_1 = \frac{\partial}{\partial x}\tilde{u}_{-\frac{1}{2}} = \frac{\tilde{u}_1 - \tilde{u}_0}{h} = \frac{u_2 - u_1}{h} = \frac{u_2}{h} = \frac{\tilde{u}_1}{h},$$

and

$$\frac{\partial}{\partial t}p_N = \frac{\partial}{\partial x}\tilde{u}_{N-\frac{1}{2}} = \frac{\tilde{u}_N - \tilde{u}_{N-1}}{h} = \frac{u_{N+1} - u_N}{h} = \frac{-u_N}{h} = -\frac{\tilde{u}_{N-1}}{h}.$$

From the previous equations we obtain

$$\frac{\partial}{\partial t}\begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ \vdots \\ p_{N-1} \\ p_N \end{bmatrix} = \frac{1}{h}\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ -1 & 1 & 0 & \cdots & 0 \\ 0 & -1 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & & 0 & -1 & 1 \\ 0 & \cdots & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} \tilde{u}_1 \\ \tilde{u}_2 \\ \tilde{u}_3 \\ \vdots \\ \tilde{u}_{N-2} \\ \tilde{u}_{N-1} \end{bmatrix}.$$

Therefore,

$$D_u = \frac{1}{h}\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ -1 & 1 & 0 & \cdots & 0 \\ 0 & -1 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -1 & 1 \\ 0 & \cdots & 0 & 0 & 1 \end{bmatrix}.$$

We repeat the process now for D_p,

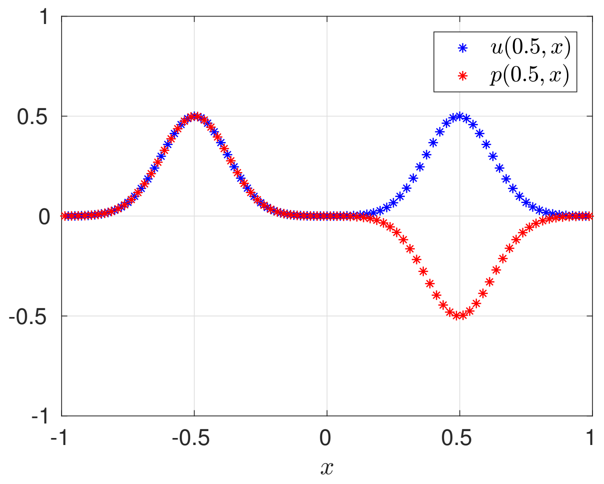
$$\frac{\partial}{\partial t}\tilde{u}_j = \frac{\partial}{\partial x}p_{j+\frac{1}{2}} = \frac{p_{j+1} - p_j}{h},$$

where we are only solving for the interior of u which involves all the nodes of p and we don't have to impose any boundary conditions. Thus, according to the previous discretization, the matrix

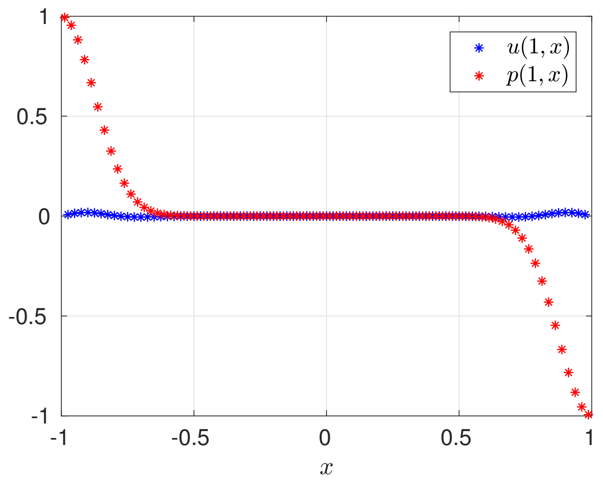
$$D_p = \frac{1}{h}\begin{bmatrix} -1 & 1 & 0 & \cdots & 0 \\ 0 & -1 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -1 & 1 \end{bmatrix},$$

which has dimension $N - 1 \times N$. With this two matices and we can complete M and solve the acoustic equations.

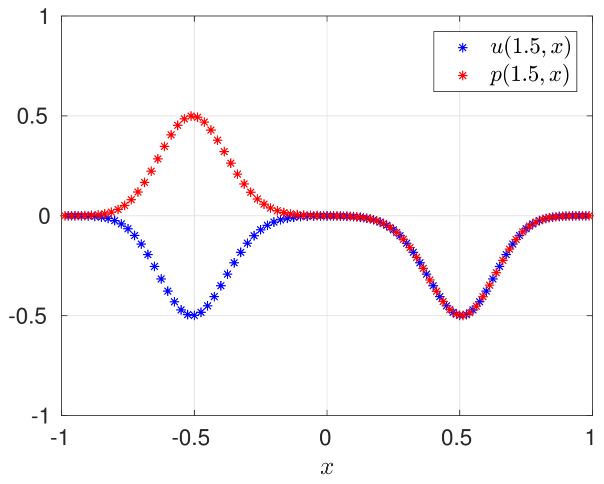
The solution is shown in the next figure, where we can see the velocity and pressure waves at different values of time.



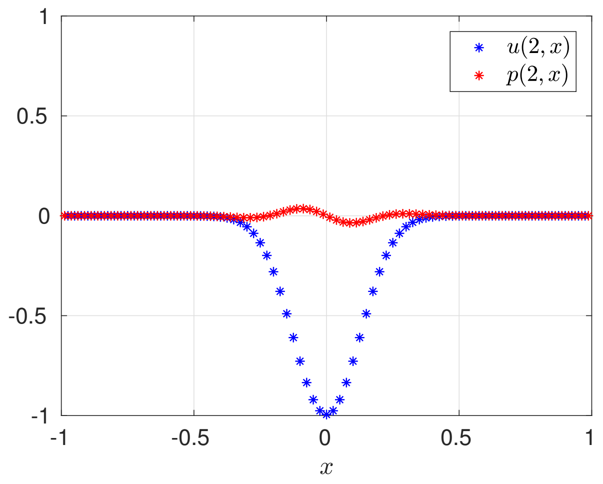
(a) $t = 0.5$ s.



(b) $t = 1.0$ s.



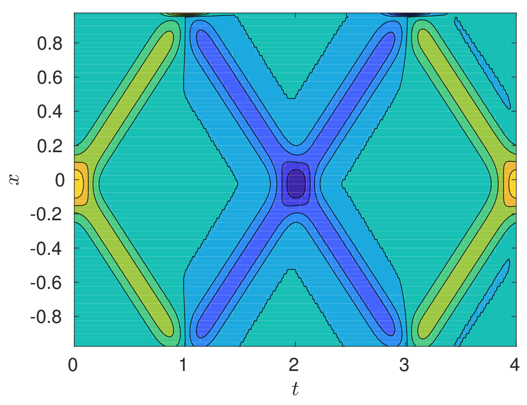
(c) $t = 1.5$ s.



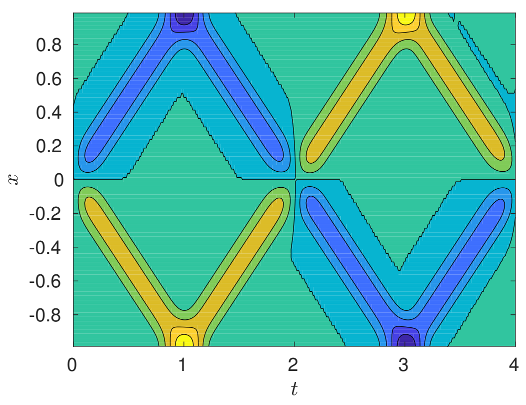
(d) $t = 2.0$ s.

Figure 7: Solution of acoustic equations at different times.

In the next figure we can see the contour plot of the solutions. Note that in fact after four seconds the velocity profile is the same, allowing us to assume a period of four seconds. In the movies generated in *Matlab* but not shown in this document we could see how the velocity and pressure waves would travel together sometimes in phase and sometimes with a phase shift of π radians. We can also observe this in the contours by ”superposing” the images.



(a) Velocity contours.



(b) Pressure contours.

Figure 8: Solution of the acoustic equations.

For the last part of the problem we estimate the accuracy of the solution. In the following figure we can see the initial condition and the solution after one period. Using $N = 80$ the accuracy seems to be good enough. The L_1-norm of the difference between the two is

$$e_{L1} = 1.839686 \cdot 10^{-2}.$$

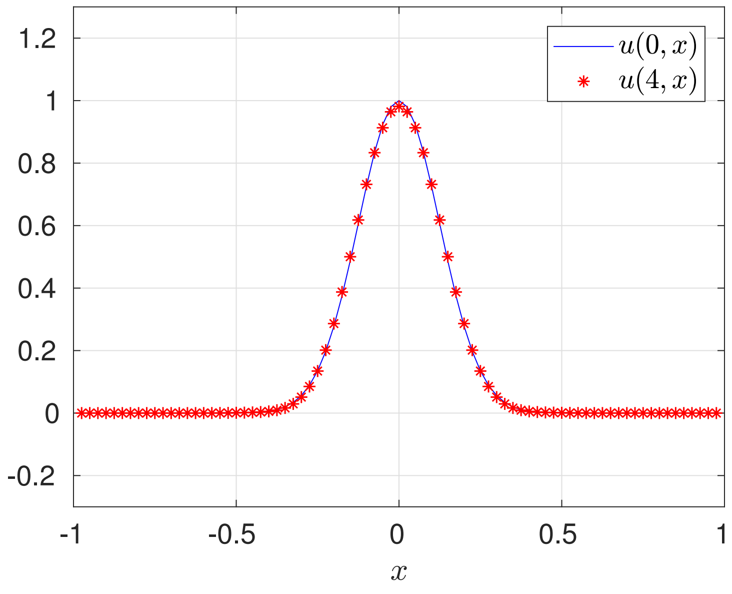


Figure 9: Comparison of the solution after 1 period, $N = 80$.

Matlab code for this problem

```
%% Problem 4
clear all;close all;format long;clc
legendfontsize=14;
axisfontsize=14;
```

```
labelfontsize=16;
N = 80;
L=2;
h=L/N;
xu=linspace(-1,1,N+1);
xp=linspace(-1+h/2,1-h/2,N);

Du = gallery('tridiag',N,-1,1,0); % In sparse form.
Du(:,end)=[];
Du=Du/h;
Dp = gallery('tridiag',N,0,-1,1); % In sparse form.
Dp(end,:)=[];
Dp=Dp/h;
Z1 = zeros(N-1,N-1);
Z2 = zeros(N,N);
M = [Z1 Dp ; Du Z2];

% initial condition
u0 = exp(-32*xu.^2)';
p0 = zeros(N,1);
v0 = [u0(2:end-1) ; p0];

t = 0:0.01:4;

[T,V] = ode45(@(t,v) M*v, t, v0);
for k = 2:length(t)
    plot(xu(2:end-1),V(k,1:N-1)','b*-',xp,V(k,N:2*N-1)','r*-')
    grid on
    ylim([-1 1])
    shg
    drawnow
end
%%

[T,X] = meshgrid(t,xu(2:end-1));
figure
contourf(T,X,round(V(:,2:N)',3))
xlabel('$t$','fontsize',labelfontsize,...
            'interpreter','latex')
ylabel('$x$','fontsize',labelfontsize,...
            'interpreter','latex')
set(gca,'fontsize',axisfontsize)
txt='Latex/FIGURES/P4_contour1';
saveas(gcf,txt,'epsc')
```

```
[T,X] = meshgrid(t,xp);
figure
contourf(T,X,round(V(:,N:end)',3))
xlabel('$t$','fontsize',labelfontsize,...
            'interpreter','latex')
ylabel('$x$','fontsize',labelfontsize,...
            'interpreter','latex')
set(gca,'fontsize',axisfontsize)
txt='Latex/FIGURES/P4_contour2';
saveas(gcf,txt,'epsc')
%%
figure
plot(xu(2:end-1),V(find(t==0.5),1:N-1),'b*')
hold on
plot(xp,V(find(t==0.5),N:2*N-1),'r*')
grid on
axis([-1 1 -1 1])
xlabel('$x$','fontsize',labelfontsize,...
            'interpreter','latex')
legend({'$u(0.5,x)$','$p(0.5,x)$'},...
        'Interpreter','latex','fontsize',legendfontsize)
set(gca,'fontsize',axisfontsize)
txt='Latex/FIGURES/P4_05';
saveas(gcf,txt,'epsc')

figure
plot(xu(2:end-1),V(find(t==1),1:N-1),'b*')
hold on
plot(xp,V(find(t==1),N:2*N-1),'r*')
grid on
axis([-1 1 -1 1])
xlabel('$x$','fontsize',labelfontsize,...
            'interpreter','latex')
legend({'$u(1,x)$','$p(1,x)$'},...
        'Interpreter','latex','fontsize',legendfontsize)
set(gca,'fontsize',axisfontsize)
txt='Latex/FIGURES/P4_1';
saveas(gcf,txt,'epsc')

figure
plot(xu(2:end-1),V(find(t==1.5),1:N-1),'b*')
hold on
plot(xp,V(find(t==1.5),N:2*N-1),'r*')
grid on
axis([-1 1 -1 1])
```

```
xlabel('$x$','fontsize',labelfontsize,...
            'interpreter','latex')
legend({'$u(1.5,x)$','$p(1.5,x)$'},...
        'Interpreter','latex','fontsize',legendfontsize)
set(gca,'fontsize',axisfontsize)
txt='Latex/FIGURES/P4_15';
saveas(gcf,txt,'epsc')

figure
plot(xu(2:end-1),V(find(t==2),1:N-1),'b*')
hold on
plot(xp,V(find(t==2),N:2*N-1),'r*')
grid on
axis([-1 1 -1 1])
xlabel('$x$','fontsize',labelfontsize,...
            'interpreter','latex')
legend({'$u(2,x)$','$p(2,x)$'},...
        'Interpreter','latex','fontsize',legendfontsize)
set(gca,'fontsize',axisfontsize)
txt='Latex/FIGURES/P4_2';
saveas(gcf,txt,'epsc')


u4=V(length(t),1:N-1)';
figure
plot(xu(2:end-1),u0(2:end-1),'b',xu(2:end-1),u4,'r*')
grid on
axis([-1 1 -0.3 1.3])
xlabel('$x$','fontsize',labelfontsize,...
            'interpreter','latex')
legend({'$u(0,x)$','$u(4,x)$'},...
        'Interpreter','latex','fontsize',legendfontsize)
set(gca,'fontsize',axisfontsize)
txt='Latex/FIGURES/P4_periodic';
saveas(gcf,txt,'epsc')
err=norm(u0(2:end-1)-u4,inf)
```