University of Waterloo

Faculty of Engineering

Department of Electrical and Computer Engineering

NE 217

Laboratory 6

Prepared by

Chockalingam Shanmugam, Anish

UW Student ID Number: 20814528

UW User ID: a2chocka @uwaterloo.ca

Prepared by

Sharda, Vid

UW Student ID Number: 20833192

UW User ID: v2sharda @uwaterloo.ca

2B Nanotechnolgy Engineering

5 August 2021

**6.1** **Stating the Problem**

Summarize in a sentence or two the problem of approximating the solution to the heat-conduction/diffusion and wave equations



in two and three dimension together with initial and either Dirichlet or insulated boundary conditions.

Solving the diffusion/wave equation in 2-dimensions with the help of finite divided difference approximation for a set of points in space over a time interval given the initial and boundary conditions.

**6.2** **Determining the Formal Parameters**

We are approximating the solutions of the diffusion and wave equations in two and three dimensions. Describe the properties of the parameters in Table 6.2*a*.

Table 6.2*a* Input parameters to the functions diffusion2d, diffusion3d, wave2d and wave3d.

|  |  |
| --- | --- |
| **Given Values** | **Description** |
| ** | rate of transfer of heat |
| *C* | propagation speed of the wave |
| **U**init | 2D (3D) nxxny (nxxnyxnz) array of initial values |
| **U**init | 2D (3D) nxxny (nxxnyxnz) array of initial rate of change values |
| [*t*inital, *t*final] | A 2x1 row vector of initial and final value of time |

We will define a function Ubndry(*t*), where Ubndry:R→**R***nx* × *ny* or Ubndry:R→**R***nx* × *ny* × *nz*. Provide your own description of this function in Table 6.2*b* for the two and three dimension cases.

Table 6.2*b* Description of the Ubndry(*t*) function for the two and three dimension cases.

|  |  |  |
| --- | --- | --- |
| **Given Functions** | **Arguments** | **Description** |
| Ubndry | *t* | A nxxny (nxxnyxnz) array with Finite real values for Dirichlett, NaN for any insulated boundary values and -Inf for any unknown values |

The numerical method described breaks the region in the spatial dimensions into a grid of points separated by a distance *h*. The interval [*t*initial, *t*final] is broken into *nt* points. Describe each of these parameters in Table 6.2*c* in your own words.

Table 6.2*c* Parameters that control the numerical solver.

|  |  |
| --- | --- |
| **Method Parameters** | **Description** |
| *h* | Step size in the space interval |
| *nt* | Number of points to divide t-interval into |

**6.3** **Determining the Return Values**

The function has to return a vector and a matrix:

1. A vector of *t* values of the *nt* points from *t*initial to *t*final, and
2. An *nx* × *ny* × *nt* or an *nx* × *ny* × *nz* × *nt* array of *u* values approximating the solution.

Describe each of these return values in Table 6.3 in your own words.

Table 6.3 Return values.

|  |  |
| --- | --- |
| **Symbol** | **Description** |
| **t** | A vector of all t values |
| **U**out | A nxxnyxnt (nxxnyxnzxnt) matrix of all x, y, z, and t values |

**6.4** **The Signature and Description of the Matlab Function**

Given the responses in 6.1, 6.2 and 6.3, fill in the description of the function wave2d here.

% REPLACE THIS TEMPLATE WITH YOUR COMMENTS AND SIGNATURE

% wave2d

% Solving the diffusion/wave equation in 2-dimensions with the help of

% finite divided difference approximation for a set of points in space over a time interval given the initial and boundary conditions.

%

% Parameters

% ==========

% c = propagation speed of the wave

% t\_int = A 2x1 row vector of initial and final value of time

%

% U\_init = 2D (3D) nxxny (nxxnyxnz) array of initial values

% dU\_init = 2D (3D) nxxny (nxxnyxnz) array of initial rate of change values

% U\_bndry = A nxxny (nxxnyxnz) array with Finite real values for

% Dirichlett, NaN for any insulated boundary values and -Inf for any unknown values

%

% h = Step size in the space interval

% n\_t = Number of points to divide t-interval into

%

% Return Values

% =============

% t = A vector of all t values

% U\_out = A nxxnyxnt (nxxnyxnzxnt) matrix of all x, y, z, and t values

function [t, U\_out] = wave2d( c, h, U\_init, dU\_init, U\_bndry, t\_int, n\_t )

**6.5** **Argument Checking**

The next step is to check the argument. Provide the appropriate error checking for each of the arguments of the function wave2d here. Note that you must check that *h* > 0.

function [t, U\_out] = wave2d( c, h, U\_init, dU\_init, U\_bndry, t\_int, n\_t )

% Argument Checking

% ARGUMENT CHECKING

if ~isscalar( c )

throw( MException( 'MATLAB:invalid\_argument', ...

'the argument c is not a scalar' ) );

end

if ~isscalar( h )

throw( MException( 'MATLAB:invalid\_argument', ...

'the argument h is not a scalar' ) );

end

if ~all( size( U\_init ) == [2, 2] )

throw( MException( 'MATLAB:invalid\_argument', ...

'the argument U\_init is not a 2-dimensional matrix' ) );

end

if ~all( size( dU\_init ) == [2, 2] )

throw( MException( 'MATLAB:invalid\_argument', ...

'the argument dU\_init is not a 2-dimensional matrix' ) );

end

if ~isa( U\_bndry, 'function\_handle' )

throw( MException( 'MATLAB:invalid\_argument', ...

'the argument U\_bndry is not a function handle' ) );

end

if ~isscalar( n\_t ) || ( nx ~= round( n\_t ) )

throw( MException( 'MATLAB:invalid\_argument', ...

'the argument n\_t is not an integer' ) );

end

if ~all( size( t\_int ) == [1, 2] )

throw( MException( 'MATLAB:invalid\_argument', ...

'the argument t\_int is not a 2-dimensional row vector' ) );

end

**6.6** **Describing the Solution**

Provide an explanation in English for each of the following steps of the solution.

1. Errors and Warnings

Calculate the error ratio (c \* del\_t / h)^2. Throw an exception it is >= 0.25. Provide a suitable value of n\_t to the user

2. Initialization

Find the size of the initial matrix.

Initialize the solution variables and set solution matrix at initial time to initial matrix.

3. Solving

Go through all values in solution matrix and if it is -Inf and not NaN, using finite difference formula find the corresponding value.

Remember to describe your steps in English. This should not be Matlab-specific instructions, but rather a description of what needs to be done.

**6.7** **Implementing the Solution**

The next step is to translate each of the steps in question 6.6 into one or more Matlab commands. For each step, indicate the appropriate Matlab commands, and if you define a local variable in any step, document the purpose of the variable.

% Error and Warning Checking

% ==========================

%

% Your description here.

Your commands here...

% Initialization

% ==============

%

% Your description here.

Your commands here...

% Solving

% =======

%

% Your description here.

Your commands here...

You will be graded on coding style as well as functionality. Lack of proper indentation and comments will result in a loss of marks.

Copy the entire function here.

% wave2d

% Solving the diffusion/wave equation in 2-dimensions with the help of

% finite divided difference approximation for a set of points in space over a time interval given the initial and boundary conditions.

%

% Parameters

% ==========

% c = propagation speed of the wave

% t\_int = A 2x1 row vector of initial and final value of time

%

% U\_init = 2D (3D) nxxny (nxxnyxnz) array of initial values

% dU\_init = 2D (3D) nxxny (nxxnyxnz) array of initial rate of change values

% U\_bndry = A function that returns a 2D nxxny (nxxnyxnz) array with Finite real values for

% Dirichlett, NaN for any insulated boundary values and -Inf for any unknown values

%

% h = Step size in the space interval

% n\_t = Number of points to divide t-interval into

%

% Return Values

% =============

% t = A vector of all t values

% U\_out = A nxxnyxnt (nxxnyxnzxnt) matrix of all x, y, z, and t values

function [t, U\_out] = wave2d( c, h, U\_init, dU\_init, U\_bndry, t\_int, n\_t )

% ARGUMENT CHECKING

if ~isscalar( c )

throw( MException( 'MATLAB:invalid\_argument', ...

'the argument c is not a scalar' ) );

end

if ~isscalar( h )

throw( MException( 'MATLAB:invalid\_argument', ...

'the argument h is not a scalar' ) );

end

if ~all( size( U\_init ) == [2, 2] )

throw( MException( 'MATLAB:invalid\_argument', ...

'the argument U\_init is not a 2-dimensional matrix' ) );

end

if ~all( size( dU\_init ) == [2, 2] )

throw( MException( 'MATLAB:invalid\_argument', ...

'the argument dU\_init is not a 2-dimensional matrix' ) );

end

if ~isa( U\_bndry, 'function\_handle' )

throw( MException( 'MATLAB:invalid\_argument', ...

'the argument U\_bndry is not a function handle' ) );

end

if ~isscalar( n\_t ) || ( nx ~= round( n\_t ) )

throw( MException( 'MATLAB:invalid\_argument', ...

'the argument n\_t is not an integer' ) );

end

if ~all( size( t\_int ) == [1, 2] )

throw( MException( 'MATLAB:invalid\_argument', ...

'the argument t\_int is not a 2-dimensional row vector' ) );

end

% INITIALIZATION

ti = t\_int(1);

tf = t\_int(2);

dt = (tf - ti)/(n\_t - 1);

t = linspace( ti, tf, n\_t );

[nx, ny] = size( U\_init );

U\_out = zeros( nx, ny, n\_t );

% Set solution matrix at initial time

U\_out(:, :, 1) = U\_init;

r = (c\*dt/h)^2;

% ERROR AND WARNINGS

% Calculate the error ratio (c \* del\_t / h)^2. Throw an exception

% if it is >= 0.25

% Provide a suitable value of n\_t

err\_rat = (c\*dt/h)^2;

if err\_rat>= 0.25

% calculate minimum integer nt, so that err\_rat < 1

noft = ceil((2\*c\*(tf-t0)/h) + 1);

throw( MException( 'MATLAB:invalid\_argument', ...

'The ratio (c \* del\_t / h)^2 = %f >= 0.25, consider using n\_t = %d', ...

err\_rat, noft ) );

end

% SOLVING

% Go through all values in solution matrix and if it is -Inf and

% not NaN, using finite difference formula find the corresponding value.

U\_out(:, :, 2) = U\_out(:, :, 1) + dt\*dU\_init;

for it = 3:n\_t

U\_out(:, :, it) = U\_bndry( t(it), nx, ny );

for ix = 1:nx

for iy = 1:ny

if U\_out(ix, iy, it) == -Inf

Utmp = U\_out(ix, iy, it - 1);

U\_out(ix, iy, it) = 2\*Utmp - U\_out(ix, iy, it - 2);

% Use finite difference as a sum of differences

for dxy = [-1 1 0 0; 0 0 -1 1]

dix = ix + dxy(1);

diy = iy + dxy(2);

% Add the difference if it is not NaN

if ~isnan( U\_out(dix, diy, it - 1) )

U\_out(ix, iy, it) = U\_out(ix, iy, it) + ...

r\*( U\_out(dix, diy, it - 1) - Utmp );

end

end

end

end

end

end

end

**6.8** **Testing your Implementation**

**6.8*a*** Suppose we have a cable of diameter 1 with an inner current-carrying cable of diameter 0.2. Assume half of the outside of the cable is exposed to cooling that keeps the boundary at 0 °C while the other side is insulated. Suppose also that initially, the interior cable is at 70 °C. This setting is shown in Figure 1.

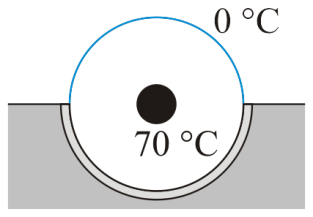


Figure 1. A cooled coaxial cable embedded in a substrate.

The following code defines the boundary conditions and the temperature in the interior of the cable. The plot is shown in Figure 2 and the last few entries indicate that the warmest temperature near the insulated surface is 51.0283 °C.

n = 201;

U = -Inf\*ones( n, n );

for j = 1:n

for k = 1:n

x = (j - 1)/(n - 1);

y = (k - 1)/(n - 1);

r = sqrt( (x - 0.5)^2 + (y - 0.5)^2 );

if r < 0.1

U(j, k) = 70;

elseif r >= (n - 1)/(2\*n);

if y <= 0.5

U(j, k) = 0;

else

U(j, k) = NaN;

end

end

end

end

U\_steady = laplace2d( U );

mesh( U\_steady );

U\_steady( round( (n + 1)/2 ), (end - 2):end )

ans =

51.0303 51.0283 NaN

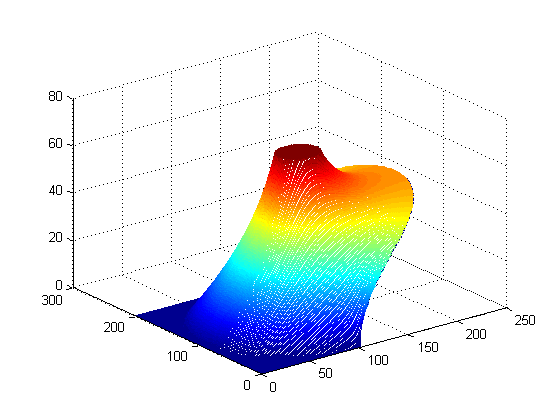


Figure 2. The steady-state temperature when the interior cable is at 70 °C.

Suppose that we must protect the exterior insulation from a surge that causes the interior cable to heat to 200°C for duration of 0.1s. If the exterior insulation fails once its temperature exceeds 120°C, estimate the largest possible thermal diffusivity coefficient ** which prevents any insulation failure in a 1 s period in which the interior cable heats to 200°C in the first 0.1s, then cool down to 70°C in the next 0.9s. Use the output **U**steady (with *n* significantly smaller than 201) as an initial state to diffusion2d. Use the given MATLAB code to construct the function **U**bndry(*t*), and note that, the code should be modified to account for the variation of the temperature of the internal cable between 200°C to 70°C as described above. When calling the function diffusion2d, recall that *h* is inversely proportional to *n­­x*.

Document what you did to approximate ** and conclude with a statement of a reasonable value for **. Full marks are awarded for having reasonable documentation and a relative error of less than 0.05 (5 %) that does not exceed the maximum value. More marks will be deduced for having too large a value of ** than too small. Your documentation must support your recommended value of **.

function [U] = U6\_bnd\_a(t, nx, ny, nz )

n = 101;

U = -Inf\*ones( n, n );

for j = 1:n

for k = 1:n

x = (j - 1)/(n - 1);

y = (k - 1)/(n - 1);

r = sqrt( (x - 0.5)^2 + (y - 0.5)^2 );

if r < 0.1 && t == 0.1

U(j, k) = 200;

elseif r < 0.1 && t == 1

U(j, k) = 70;

elseif r >= (n - 1)/(2\*n)

if y <= 0.5

U(j, k) = 0;

else

U(j, k) = NaN;

end

end

end

end

end

**And then a script was run to determine k value**

greater = [];

for k = linspace(0.01, 1000, 10)

[t, U] = diffusion2d( k, 0.1, U\_steady, @U6\_bnd\_a, [0, 1], 51 );

z = [];

n = 101;

for j = 1:n

for f = 1:n

x = (j - 1)/(n - 1);

y = (f - 1)/(n - 1);

r = sqrt( (x - 0.5)^2 + (y - 0.5)^2 );

if r > 0.1 && r < (n - 1)/(2\*n)

if U(j, f) > 120

z(end+1) = 1;

end

end

end

end

greater(end+1) = sum(z);

end

**However, no value of K in the selected range had a temperature of 120.**

**6.8*b*** Suppose you have a cylinder of material that you want to dope with another element. To perform this doping, the cylinder is placed into a bath providing a source of the doping element atoms at a concentration of *c*bath = 1. The cylinder will remain in contact with the bath for 10 s. After these 10s, the cylinder is removed from the bath, its boundaries are insulated, and the doping material is allowed to continue diffusing internally for another 10s. After these 20s of diffusion (inside and outside of the bath), the cylinder is cooled to the room temperature and the diffusion stops. The base of the cylinder is a circle of radius 0.5 centered at (.5, .5, 0) and the diffusion coefficient.

The following code constructs the matrix U\_init:

n = 21;

c\_bath = 1;

U\_init = zeros( n, n, n );

U\_init(:, :, [1, end]) = c\_bath;

for i = 1:n

for j = 1:n

for k = 1:n

x = (i - 1)/(n - 1);

y = (j - 1)/(n - 1);

r = sqrt( (x - 0.5)^2 + (y - 0.5)^2 );

if r >= 0.5

U\_init(i, j, k) = 1;

end

end

end

end

The boundary value function is given here:

function [U] = u6b\_bndry( t, n1, n2, n3 )

c\_bath = 1;

T\_bath = 10;

if t <= T\_bath

val = c\_bath;

else

val = NaN;

end

U = -Inf\*ones( n1, n2, n3 );

U(:, :, [1, end]) = val;

for i = 1:n1

for j = 1:n2

for k = 1:n3

x = (i - 1)/(n1 - 1);

y = (j - 1)/(n2 - 1);

r = sqrt( (x - 0.5)^2 + (y - 0.5)^2 );

if r >= 0.5

U(i, j, k) = val;

end

end

end

end

end

% Solve for the diffusion

T\_final = 20;

n\_t = 500;

[t, U\_out] = diffusion3d( 4, 1, U\_init, @u6b\_bndry, [0, T\_final], n\_t );

% Find the minimum and maximum concentrations in the cylinder for each point in time  
c\_max = zeros( 1, n\_t );

c\_min = zeros( 1, n\_t );

for k = 1:n\_t

c\_max(k) = max( max( max( U\_out(:, :, :, k) ) ) );

c\_min(k) = min( min( min( U\_out(:, :, :, k) ) ) );

end

% Plot the minimum and maximum temperatures

plot( t, c\_max );

hold on;

plot( t, c\_min );

The last three lines plot the minimum and maximum concentrations within the cylinder and the plot is shown in Figure 3.

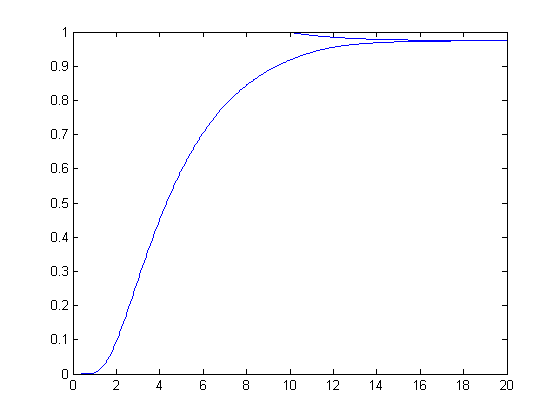


Figure 3. The minimum and maximum concentrations.

After 20 s, the minimum and maximum concentrations are 0.9744 and 0.9749. If the target is to have a concentration of 1, the maximum absolute error would be

|1 – 0.9744| = 0.0256.

There are three variables that we can modify:

1. The concentration of the bath (*c*bath) by using a higher (but fixed) concentration,
2. The time period that the cylinder is immersed in the bath, *T*bath, and
3. The time period during which the cylinder is removed from the bath but allowed to continue diffusing, (*T*bath, *T*final].

Your goal is to minimize the total time required to keep the cylinder at a high temperature (i.e. the total diffusion time inside and outside of the bath) and to minimize the largest absolute error from an expected concentration of 1. Complete Table 1 with your values.

Table 1. Values used to dope the cylinder.

|  |  |
| --- | --- |
| **Variable** | **Value** |
| *T*bath | 1.9 |
| *T*final | 2 |
| *c*bath | 10 |
| *c­*min at *T*­final | 0.9930 |
| *c­*max at *T*­final | 9.6464 |

You should calculate your grade on this question as follows:

>> 10 – T\_final - log2( delta )

where T\_final is the total time the cylinder is kept at a high temperature, and delta is the maximum absolute error of the concentration from 1, i.e. delta = max(abs(c\_max(end)-1),abs(c\_min(end)-1)). Calculate your grade and copy-and-paste your solutions here.

>> c\_max(end)

ans =

9.6464

>> c\_min(end)

ans =

0.9930

grade =

4.8879

**6.8*c*-*d*** The following two questions demonstrate the relationship between waves and conic sections.

**6.8*c*** An example (pg. 59) in the presentation demonstrates the response when a disturbance is made in the middle of a cup containing a liquid. If, however, the disturbance is made to the right of centre, waves travel to the walls of the cup, reflect, and concentrate, though less cleanly, at a point symmetric through the centre of the cup. The first step is shown in Figure 4, while after the wave begins to cross the cup, there are numerous transient waves appearing behind the initial wave, as shown in Figure 5.

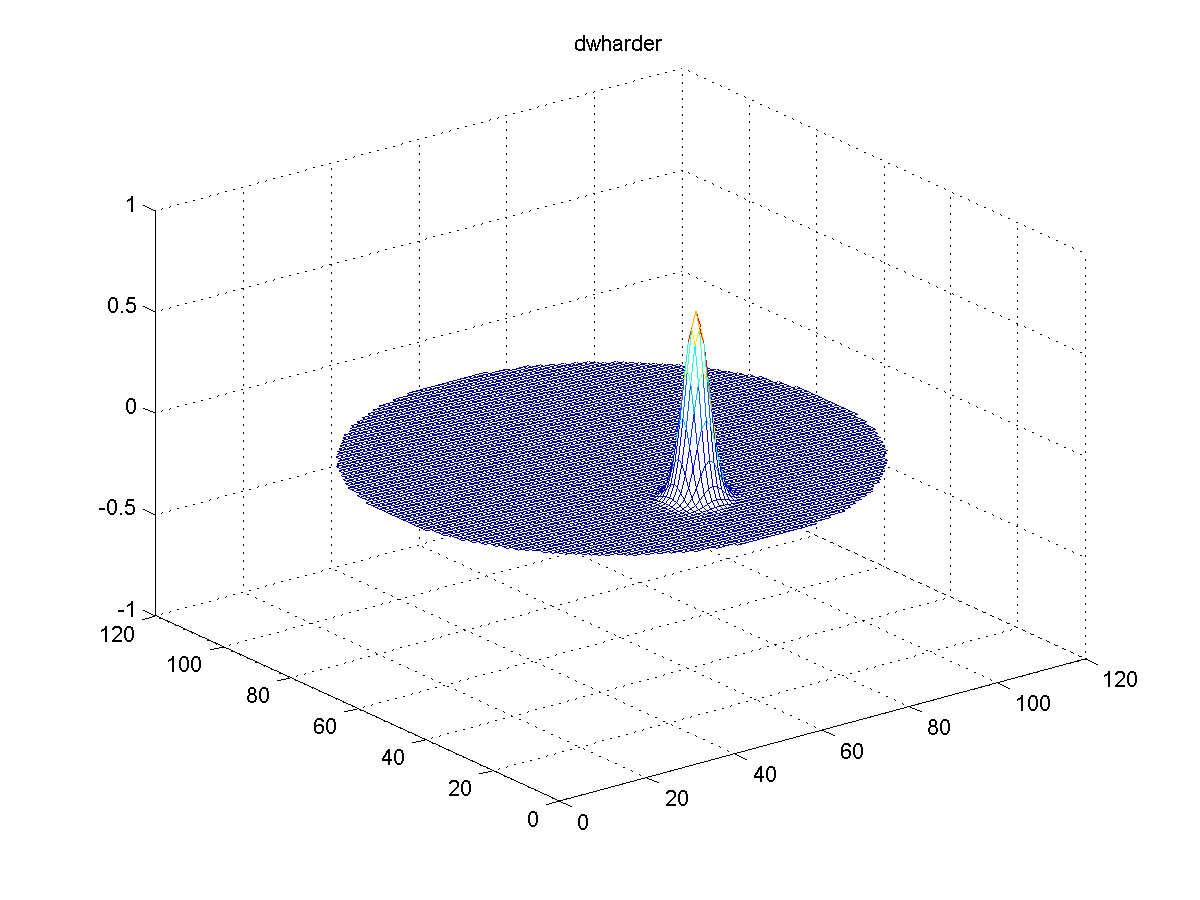


Figure 4. An off-centred disturbance.

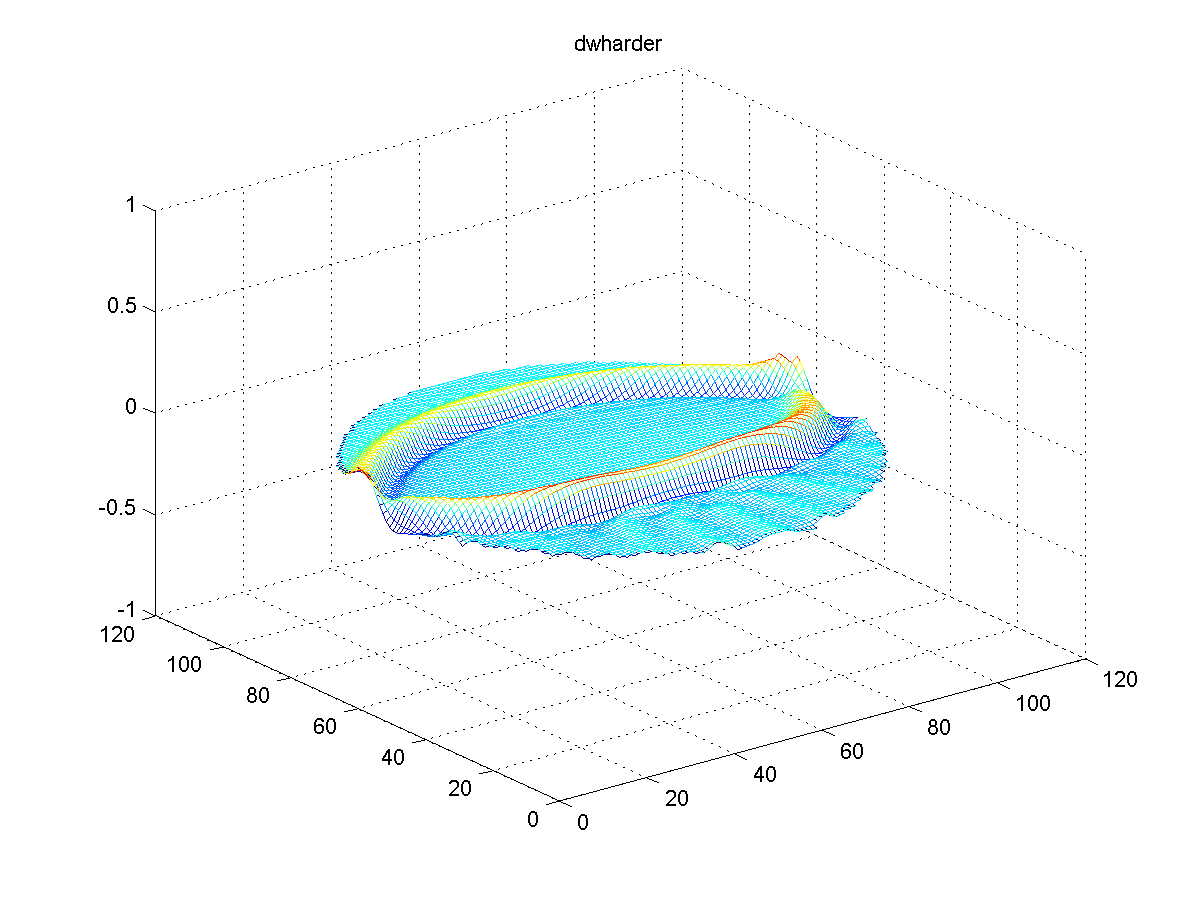


Figure 5. The wave after 60 seconds with transient waves forming after the wave front.

The wave never really reconstitutes itself at the symmetric point, as is shown in Figure 6.

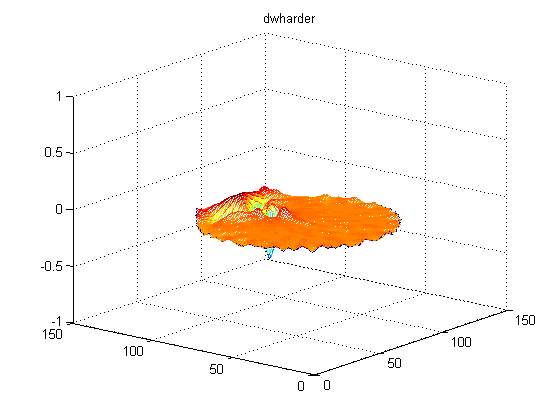


Figure 6. The image with the deepest trough at 113s.

By the time the crests reach the original point, it creates two peaks at 222s and 267s, as shown in Figure 7.

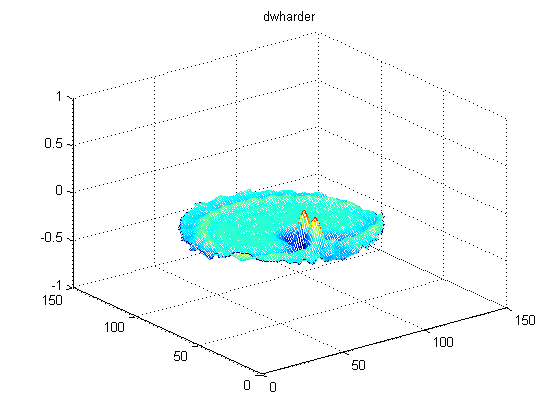
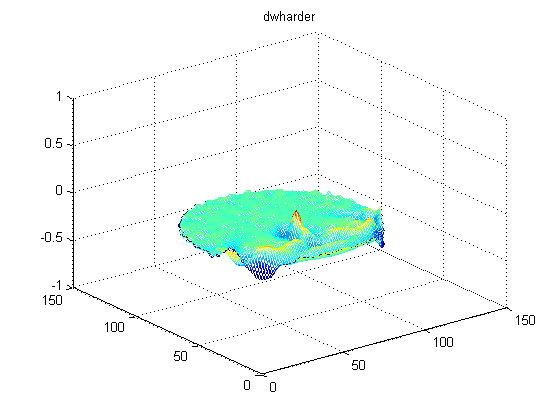


Figure 7. Reflection peaks formed at 222s and 267s.

These images were generated using the script

n = 101;

U6c\_init = zeros( n, n );

dU6c\_init = zeros( n, n );

for j = 1:n

for k = 1:n

% (x, y) is a point in [0, 1] x [0, 1]

x = (j - 1)/(n - 1);

y = (k - 1)/(n - 1);

% The initial disturbance is centred at (0.25, 0.5)

U6c\_init(j, k) = exp( -1000\*((x - 0.25)^2 + (y - 0.5)^2) );

end

end

[t6c, U6c] = wave2d( 1, 1, U6c\_init, dU6c\_init, @U6c\_bndry, [0, 300], 650 );

frames6c = animate( U6c );

The boundary conditions are provided by

function [U] = U6c\_bndry( t, n\_x, n\_y )

U = -Inf\*ones( n\_x, n\_y );

for j = 1:n\_x

for k = 1:n\_y

% (x, y) is a point in [0, 1] x [0, 1]

x = (j - 1)/(n\_x - 1);

y = (k - 1)/(n\_y - 1);

% Determine if a point is outside a circle of radius 0.5

% centred at the point (0.5, 0.5)

if sqrt( (x - 0.5)^2 + (y - 0.5)^2 ) >= 0.5

U(j, k) = NaN;

end

end

end

end

Your task is to find a shape with area greater than 1 that allows a deformation at (0.5, 0.5) to be reasonably well reconstructed at the point (1.5, 0.5). You can modify the script as follows but you must find the appropriate boundary conditions.

n\_x = 201;

n\_y = 101;

U6c\_init = zeros( n\_x, n\_y );

dU6c\_init = zeros( n\_x, n\_y );

for j = 1:n\_x

for k = 1:n\_y

x = 2\*(j - 1)/(n\_x - 1);

y = (k - 1)/(n\_y - 1);

U6c\_init(j, k) = exp( -1000\*((x - 0.5)^2 + (y - 0.5)^2) );

end

end

[t6c, U6c] = wave2d( 1, 1, U6c\_init, dU6c\_init, @U6c\_bndry, [0, 350], 710 );

frames6c = animate( U6c );

You should determine which time slice *t*min has the largest negative value at the point (1.5, 0.5) and enter it here:

>> [z\_min, t\_min] = min( U6c(151, 51, :) )

>> t( t\_min )

Copy and paste your Matlab output here.

and you should copy an image of that frame into Figure 8.

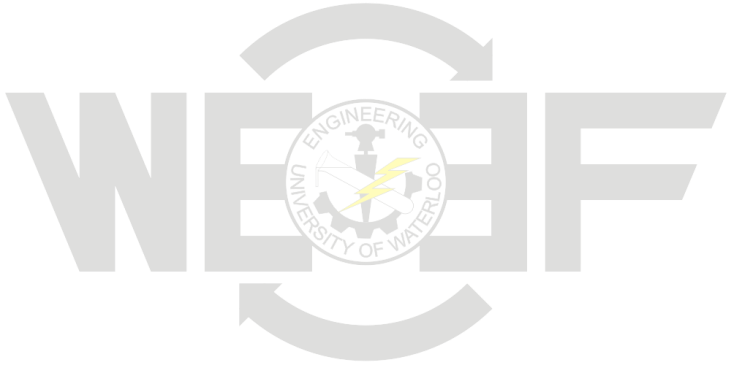


Figure 8. The point at which the minimum is achieved.

**6.8*d*** Consider a sinusoidal signal formed at one face of a cube which has an edge length equal to 1. The cube has all the other faces insulated, and the reflection of the wave from the face opposite to the one where the sinusoidal signal exists is perfect. However, the signal strength (the amplitude) is not very large at any one point in the cube. Your goal is to change the shape of the wall near *x* = 0 (the face opposite to the sinusoidal signal) in order to focus the signal at the point (0.25, 0.5, 0.5). Currently, the wall at *x* = 0 is straight. Hence, another 3D-boundary shape should be used to focus the reflecting waves to the mentioned point.

n\_x = 33;

n\_y = 33;

n\_z = 33;

U6d\_init = zeros( n\_x, n\_y, n\_z );

dU6d\_init = zeros( n\_x, n\_y, n\_z );

[t6d, U6d] = wave3d( 1, 1, U6d\_init, dU6d\_init, @U6d\_bndry, [0, 60], 150 );

function [U] = U6d\_bndry( t, n\_x, n\_y, n\_z )

U = -Inf\*ones( n\_x, n\_y, n\_z );

U(end,:,:) = sin(t).\*(t <= 2\*pi);

U(:,[1,end],:) = NaN;

U(:,:,[1,end]) = NaN;

for i = 1:n\_x

for j = 1:n\_y

for k = 1:n\_z

x = (i - 1)/(n\_x - 1);

y = (j - 1)/(n\_y - 1);

z = (k - 1)/(n\_z - 1);

% Modify this to determine which points at the end  
 % of the region will be set to NaN to reflect and  
 % focus the signal. Consider the example where   
 % the points were determined by the distance from   
 % the centre of a circle.

if x == 0

U(i, j, k) = NaN;

end

end

end

end

end

Indicate the maximum value of the signal. Your grade will be based on the maximum you achieve close to the point (0.25, 0.5, 0.5).

Indicate in a paragraph what your strategy will be for designing the reflector.

Your paragraph here.

Copy your modified **U**6d, bndry function here:

function [U] = U6d\_bndry( t, n\_x, n\_y, n\_z )

Indicate the maximum signal strength (maximum value in **U**out) value in the 60 s.

>> max( U\_out( 9, 17, 17, : ) );

Copy and paste your Matlab output here.

**6.8*e*** The code presented in the slides for the diffusion2d function has a triply nested for loop:

for it = 2:n\_t

U\_out(:, :, it) = U\_bndry( ts(it), n\_x, n\_y );

for ix = 1:n\_x

for iy = 1:n\_y

if U\_out (ix, iy, it) == -Inf

Utmp = U\_out (ix, iy, it - 1);

U\_out (ix, iy, it) = Utmp;

for dxy = [-1 1 0 0; 0 0 -1 1]

dix = ix + dxy(1);

diy = iy + dxy(2);

if ~isnan( U\_out(dix, diy, it - 1) )

U\_out (ix, iy, it) = U\_out (ix, iy, it) + ...

r\*( U\_out(dix, diy, it - 1) - Utmp );

end

end

end

end

end

end

Just for your knowledge, the following code is an equivalent to the previous code but uses only one for loop:

for it = 2:n\_t

U\_out(:, :, it) = U\_bndry( ts(it), n\_x, n\_y );

idx = find( U\_out == -Inf );

U\_out(idx) = U\_out(idx - n\_x\*n\_y) + r \* ( ...

~isnan( U\_out(idx - 1 - n\_x\*n\_y) ) .\* ...

(U\_out(idx - 1 - n\_x\*n\_y) - U\_out(idx - n\_x\*n\_y)) ...

+ ~isnan( U\_out(idx + 1 - n\_x\*n\_y) ) .\* ...

(U\_out(idx + 1 - n\_x\*n\_y) - U\_out(idx - n\_x\*n\_y)) ...

+ ~isnan( U\_out(idx - n\_x - n\_x\*n\_y) ) .\* ...

(U\_out(idx - n\_x - n\_x\*n\_y) - U\_out(idx - n\_x\*n\_y)) ...

+ ~isnan( U\_out(idx + n\_x - n\_x\*n\_y) ) .\* ...

(U\_out(idx + n\_x - n\_x\*n\_y) - U\_out(idx - n\_x\*n\_y)) ...

);

end

\*NOTE: THIS IS JUST FOR LEARNING PURPOSES. NO ANSWERS NEED BE GIVEN FOR THIS QUESTION