University of Waterloo

Faculty of Engineering

Department of Electrical and Computer Engineering

MATH 212/NE 217

Laboratory 6

Prepared by

Surname, Legal Given Name(s)

UW Student ID Number: 2NNNNNNN

UW User ID: uwuserid @uwaterloo.ca

Prepared by

Surname, Legal Given Name(s)

UW Student ID Number: 2NNNNNNN

UW User ID: uwuserid @uwaterloo.ca

2B Electrical/Nanotechnolgy Engineering

**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.

Your summary here.

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

We are approximating a solution to Laplace’s equation in two and three dimensions. There is only one required parameter:

1. The coefficient ,
2. The spatial interval [a, b], and
3. The time interval over which we will approximate the solution [*t*initial, *t*final].

Describe the properties of this array in Table 6.2*a*.

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

|  |  |
| --- | --- |
| **Given Values** | **Description** |
| ** | Your description here ... |
| *c* | Your description here ... |
| **U**init | Your description here ... |
| **U**init | Your description here ... |
| [*t*inital, *t*final] | Your description here ... |

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 these functions in Table 6.2*b*.

Table 6.2*b* Functions given in the mathematical problem.

|  |  |  |
| --- | --- | --- |
| **Given Functions** | **Arguments** | **Description** |
| Ubndry | *t* | Your description here ... |

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 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* | Your description here... |
| *nt* | Your description here... |

**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** | The vector **t** is ... |
| **U**soln | The matrix **U**soln is ... |

**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

% Copy the description from 6.1 here

%

% Parameters

% ==========

% c

% t\_int

%

% U\_init

% dU\_init

% U\_bndry

%

% h

% n\_t

%

% Return Values

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

% t

% U\_soln

function [t, U\_soln] = 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.

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

 % Argument Checking

 Copy your error checking here

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

Three specific steps are described in the course slides. For each of the steps, provide an explanation in English.

1. Errors and Warnings

Your description here.

2. Initialization

Your description here.

3. Mapping the unknown points to a unique number from 1 to *m*

Your description here.

4. Creating and solving a system of linear equations

Your description here.

5. Substituting the values back into the matrix **U**soln

Your description here.

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 convert your code to Matlab code. Each of the steps you indicated in Question 5.6 will be translated into one or more Matlab commands. For each step, indicate the appropriate Matlab commands. 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...

% Mapping the unknown points to a unique number from 1 to m

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

%

% Your description here.

Your commands here...

% Creating and solving a system of linear equations

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

%

% Your description here.

Your commands here...

% Substituting the values back into the matrix U\_soln

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

%

% 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.

Once you have tested your function, copy the entire function here.

**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 is shown in Figure 1.



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

The following code defines these boundary conditions and approximates 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 for a surge that causes the interior cable to heat to 200 °C for duration of 0.1 s (a peicewise constant function) but the exterior insulation breaks down once the temperature exceeds 120 °C. Estimate the largest possible thermal diffusivity coefficient ** such that the exterior does not allow the boundary to exceed
120 °C in a 1 s period where the interior cable heats to 200 °C in the first 0.1 s followed by another 0.9 s where the temperature returns to 70 °C.

The value of *n* = 201 is too large; therefore, use the output **U**steady when *n* = 51 the initial state to diffusion2d and use the code to construct a boundary-value function **U**bndry(*t*) that has an internal cable with temperature 200 °C.

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 **.

**6.8*b*** Suppose you have a cylinder of material that you want to dope with another element. To do this, we could, for a period of time, place the cylinder into a bath providing a source of element atoms at a concentration of *c*bath = 1 and allowing the cylinder to remain in contact with the bath for 10 s followed by another 10 s during which the cylinder is removed from the bath but the boron within the silicon is allowed to continue diffusing internally (insulated boundaries). After these 20 s, the cylinder is cooled to room temperature and the diffusion stops.

The base of cylinder is a circle of radius 0.5 centered at (.5, .5, 0).

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

% Solve for the diffusion

T\_final = 20;

n\_t = 500;

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

% Find the minimum and maximum concentrations 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\_soln(:, :, :, k) ) ) );

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

end

% Plot the minimum and maximum temperatures

plot( t, c\_max );

hold on;

plot( t, c\_min );

The boundary value function is given here:

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

 c\_bath = 3;

 T\_bath = 0.47;

 if t <= T\_bath

 val = 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

The last two lines find 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 will be to minimize the total time required to keep the cylinder at a high temperature 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 | ??? |
| *T*final | ??? |
| *c*bath | ??? |
| *c­*min at *T*­final | ??? |
| *c­*max at *T*­final | ??? |

You should calculate your mark on this question by letting *T* be the total time the cylinder is kept at a high temperature and ** is the maximum absolute error of the concentration from 1.

>> 10 - T - log2( delta )

Calculate your grade and copy-and-paste your solutions here.

Copy and paste your Matlab commands and the output here.

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

**6.8*c*** An example 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. Transient waves are 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 113 s.

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



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

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 and enter it here:

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

Copy and paste your Matlab output here.

>> t( t\_min );

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*** This code has a wave formed at one end with all other boundaries insulated. The reflection of the wave on the other side is essentially perfect; however, the signal strength (the amplitude) is not very large at any one point. Your goal is to change the shape of the wall near *x* = 0 in order to focus the signal at the point (0.25, 0.5, 0.5) and maximize the signal at that point. Currently, the wall at *x* = 0 is straight. Another 3D-boundary shape could be used to focus the reflecting waves to a 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**soln) value in the 60 s.

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

Copy and paste your Matlab output here.

**6.8*e*** The code presented in the slides is a triply nested for loop:

for it = 2:n\_t

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

 for ix = 1:n\_x

 for iy = 1:n\_y

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

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

 U\_soln (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\_soln(dix, diy, it - 1) )

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

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

 end

 end

 end

 end

 end

 end

Consider the following code:

for it = 2:n\_t

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

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

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

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

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

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

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

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

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

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

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

 );

end