

Partial Differential Equations Matlab Exercises

Aarre Urtamo

October 2024

Contents

1	Week 1	3
	1.1	3
	1.2	4
2	Week 2	6
	2.1	6
	2.2	10
3	Week 3	13
	3.1	13
	3.2	14
4	Week 4	17
	4.1	17
	4.2	19
5	Week 5	22
	5.1	22
	5.2	25
6	Week 6	27
	6.1	27
	6.2	28
7	Week 7	32
	7.1	32
	7.2	33

1 Week 1

1.1

Thus

$$2u_x + 6u_y = 0 \Leftrightarrow \begin{bmatrix} 2 \\ 6 \end{bmatrix} \cdot \begin{bmatrix} u_x \\ u_y \end{bmatrix} = 0 \Leftrightarrow \begin{bmatrix} 2 \\ 6 \end{bmatrix} \cdot \nabla u = 0, \quad (1)$$

we get the common solution $u(x, y) = f(6x - 2y)$.

To determine the specific form of $f(6x - 2y)$, let's use the initial condition $u(0, y) = \cos(y)$. Let's denote $z = 6x - 2y$.

$$\begin{aligned} f(6 \cdot 0 - 2y) &= f(-2y) = \cos(y) \\ z = -2y &\Leftrightarrow y = -\frac{z}{2} \\ f(z) &= \cos\left(-\frac{z}{2}\right) \end{aligned} \quad (2)$$

We get the form $u(x, y) = f(6x - 2y) = \cos\left(-\frac{6x-2y}{2}\right) = \cos(y - 3x)$.

```
%% 1.1
clearvars
close all
clc
% b
u = @(x,y) cos(2*y-6*x);
amount = 40;
border = 1;
area = linspace(-border,border,amount);
[x,y] = meshgrid(area,area);
z = u(x,y);
mesh(x,y,z)
% c
hold on
% Let's draw e.g 10 lines
lineAmount = 10;
% Let's create different constant terms for the lines
d = linspace(-1,1,lineAmount)';
x_l = area;
y_l = 3*x_l+d;
z_l = u(x_l,y_l);
for i = 1:lineAmount
    plot3(x_l,y_l(i,:),z_l(i,:))
end
```

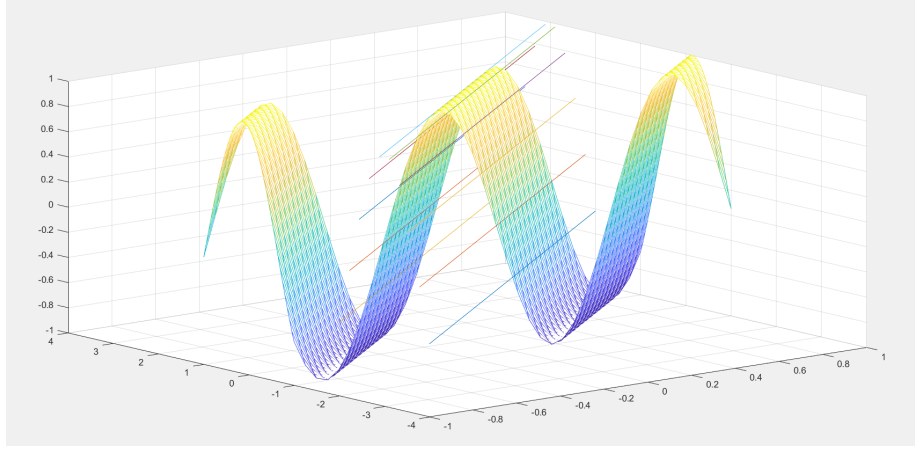


Figure 1: The surface and the lines

1.2

The equation $yu_x + u_y = \frac{1}{3}$, the initial condition $u(x, 0) = \sin(\frac{x}{5})$.

$$\begin{aligned}
 yu_x + u_y &= \frac{1}{3} \\
 \Leftrightarrow yu_x + u_y - \frac{1}{3} &= 0 \\
 \Leftrightarrow \begin{bmatrix} u_x \\ u_y \\ -1 \end{bmatrix} \cdot \begin{bmatrix} y \\ 1 \\ \frac{1}{3} \end{bmatrix} &= 0
 \end{aligned} \tag{3}$$

The system of characteristic equations:

$$\begin{cases} x'(t) = y \\ y'(t) = 1 \\ u'(t) = \frac{1}{3} \end{cases} \Leftrightarrow \begin{cases} y(t) = \int 1 dt = t + C_2 \\ u(t) = \int \frac{1}{3} dt = \frac{1}{3}t + C_3 \\ x(t) = \int (t + C_2) dt = \frac{1}{2}t^2 + C_2t + C_1 \end{cases}, t \in \mathbf{R} \tag{4}$$

Considering the initial condition:

$$u(x, 0) = \sin\left(\frac{x}{5}\right) \Rightarrow \begin{cases} \tilde{x}(s) = s \\ \tilde{y}(s) = 0 \\ \tilde{u}(s) = \sin\left(\frac{s}{5}\right) \end{cases}, s \in \mathbf{R} \tag{5}$$

Let's pick t so that the initial condition is satisfied when $t = 0$. Thus

$$\begin{cases} x(0) = \tilde{x}(s) \\ y(0) = \tilde{y}(s) \\ u(0) = \tilde{u}(s) \end{cases} \Leftrightarrow \begin{cases} \frac{1}{2} \cdot 0^2 + C_2 \cdot 0 + C_1 = s \\ 0 + C_2 = 0 \\ \frac{1}{3} \cdot 0 + C_3 = 0 + \sin(\frac{s}{5}) \end{cases} \Leftrightarrow \begin{cases} C_1 = s \\ C_2 = 0 \\ C_3 = \sin(\frac{s}{5}) \end{cases} \quad (6)$$

The solution is thus in the form:

$$\begin{cases} x(s, t) = \frac{1}{2}t + s \\ y(s, t) = t \\ u(s, t) = \frac{1}{3} + \sin(\frac{s}{5}) \end{cases} \quad (7)$$

Let's express the solution in terms of variables x and y .

$$\begin{aligned} y = t \Rightarrow x &= \frac{1}{2}y^2 + s \Rightarrow s = x - \frac{1}{2}y^2 \\ \Rightarrow u(x, y) &= \frac{1}{3} + \sin\left(\frac{x - \frac{1}{2}y^2}{5}\right) \end{aligned} \quad (8)$$

```
%% 1.2
clearvars
close all
clc

% b
u = @(x,y) 1/3*y+sin((x-1/2*y.^2)/5);

border = 10;
density = 0.5;
[x,y] = meshgrid(-border:density:border,-border:density:border);
z = u(x,y);
figure
mesh(x,y,z)

% c
t = -5:density:5;
x_curve = 1/2*t.^2;
y_curve = t;
u_curve = 1/3*t;

hold on
plot3(x_curve,y_curve,u_curve,"k");

% Let's check that the wanted point is on the curve
plot3(0,0,0,"k*")
```

```

% d
% Let's draw the vectors.
density_v = 1.5;
[x_v,y_v] = meshgrid(-border:density_v:border,-border:density_v:border);
z_v = u(x_v,y_v);
v_x = y_v;
v_y = ones(size(x_v));
v_u = 1/3*ones(size(x_v));

% Let's do normalization and draw.
v_xn = v_x./sqrt(v_x.^2+v_y.^2+v_u.^2);
v_yn = v_y./sqrt(v_x.^2+v_y.^2+v_u.^2);
v_un = v_u./sqrt(v_x.^2+v_y.^2+v_u.^2);
quiver3(x_v,y_v,z_v,v_xn,v_yn,v_un,0.5,"r")

```

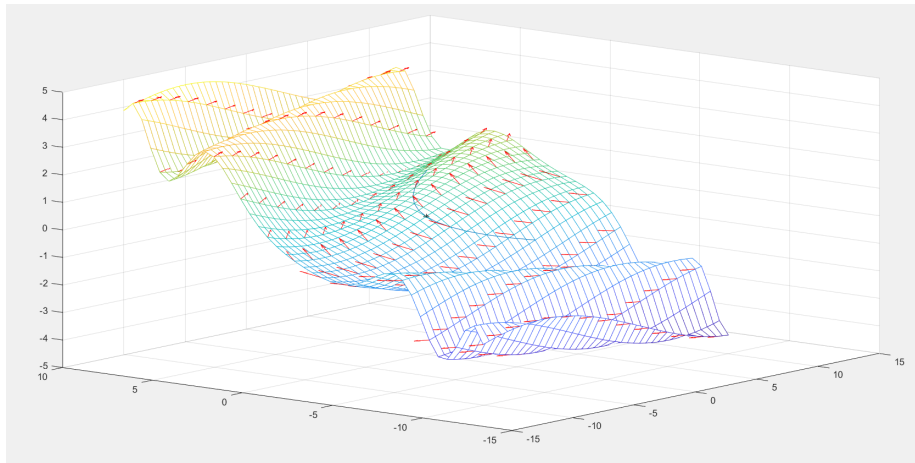


Figure 2: The final figure

2 Week 2

2.1

Let's solve the PDE $2u_x + u_t = \frac{1}{x^2+1}$ with boundary conditions $u(x, 0) = 0$ and $u(0, t) = \frac{1}{2}\arctan(2t)$ using finite difference method.

Let's start by writing the differential equation using difference approximation.

$$\begin{aligned}
2u_x + u_t &= \frac{1}{x^2 + 1} \\
\Rightarrow 2 \cdot \frac{u_{i,n} - u_{i-1,n}}{\Delta x} + \frac{u_{i,n+1} - u_{i,n}}{\Delta t} &= \frac{1}{x_i^2 + 1} \\
\Rightarrow \frac{2\Delta t}{\Delta x} (u_{i,n} - u_{i-1,n}) + u_{i,n+1} - u_{i,n} &= \frac{\Delta t}{x_i^2 + 1}
\end{aligned} \tag{9}$$

Let's denote $s = \frac{2\Delta t}{\Delta x}$ and solve $u_{i,n+1}$ from the equation.

$$u_{i,n+1} = \frac{\Delta t}{x_i^2 + 1} - su_{i,n} + su_{i-1,n} + u_{i,n} = su_{i-1,n} + (1-s)u_{i,n} + \frac{\Delta t}{x_i^2 + 1} \tag{10}$$

Let's denote the number of points we use in x -direction with I . Therefore, we get the following system of equations. The first equation we get from the boundary condition $u(0, t) = \frac{1}{2} \arctan(2t)$.

$$\begin{cases}
u_{1,n+1} = \frac{1}{2} \arctan(2t_{n+1}) \\
u_{2,n+1} = su_{1,n} + (1-s)u_{2,n} + \frac{\Delta t}{x_2^2 + 1} \\
u_{2,n+1} = su_{1,n} + (1-s)u_{2,n} + \frac{\Delta t}{x_i^2 + 1} \\
\vdots \\
u_{I,n+1} = su_{I-1,n} + (1-s)u_{I,n} + \frac{\Delta t}{x_i^2 + 1}
\end{cases} \tag{11}$$

We can also write the system of equations in a matrix form as

$$\begin{bmatrix} u_{1,n+1} \\ u_{2,n+1} \\ \vdots \\ u_{I,n+1} \end{bmatrix} = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 \\ s & 1-s & \cdots & 0 & 0 \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & s & 1-s \end{bmatrix} \begin{bmatrix} u_{1,n} \\ u_{2,n} \\ \vdots \\ u_{I,n} \end{bmatrix} + \begin{bmatrix} \frac{1}{2} \arctan(2t_{n+1}) \\ \frac{\Delta t}{x_2^2 + 1} \\ \vdots \\ \frac{\Delta t}{x_I^2 + 1} \end{bmatrix} \tag{12}$$

Let's write the Matlab code to solve the equation.

```

%% 2.1
clearvars
close all
clc

% Let's make the needed variables.
I = 7;
N = 21;

```

```

x = linspace(0,1,I)';
t = linspace(0,1,N)';

dt = t(2)-t(1);
dx = x(2)-x(1);
u1 = zeros(I,1);
s = 2*dt/dx;

% Let's make the multiplier matrix A
A = zeros(I,I);
for n = 2:I
    A(n,n-1) = s;
    A(n,n) = 1-s;
end

% Let's make the b vector
b = dt./(x.^2+1);
b(1) = 1/2*atan(2*t(2));

% U is matrix where the solution is saved.
U = zeros(I,N);
U(:,1) = u1;

% Let's make the first iteration outside the loop
u2 = A*u1+b;
U(:,2) = u2;

% Let's make a loop for the rest of the iterations.
for n = 3:N
    b(1) = 1/2*atan(2*t(n));
    un = A*U(:,n-1)+b;
    U(:,n) = un;
end

% Let's draw the solution.
figure
[X,T] = meshgrid(x,t);
Z = 1/2*atan(X)-1/2*atan(X-2*T);
mesh(X,T,Z,'FaceAlpha','0.6')
xlabel('x', 'interpreter', 'latex')
ylabel('t', 'interpreter', 'latex')
zlabel('u', 'interpreter', 'latex')

hold on
for i = 1:I

```



```

plot3(x(i)*ones(1,N),t,U(i,:), 'k*')
end

```

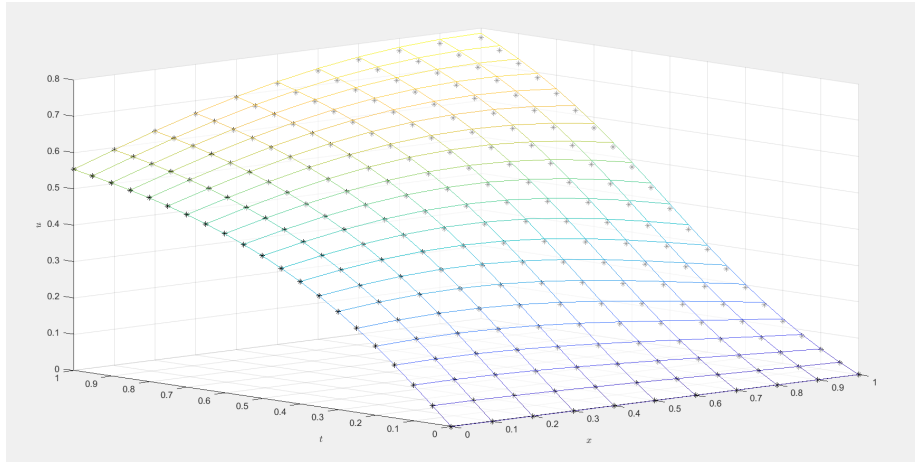


Figure 3: The graphic solution

We got the current stable solution when we used $I = 11$ points in the x -direction and $N = 21$ points in the t -direction. Let's change the amount of points in the x -direction to $I = 15$ and run the code again. Now the step size in x -direction decreases.

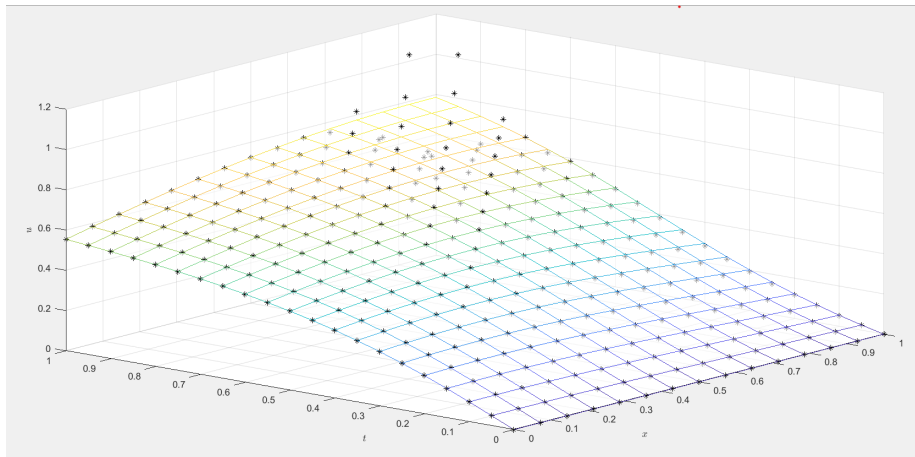


Figure 4: The graphic solution with $I = 15$ in the x -direction

As we can see from the graph, now the solution is unstable. Some of the

solution points are not on the surface of the exact solution function's graph. Before we increased the amount of points, the value of our variable $s = \frac{2\Delta t}{\Delta x}$ was $s = 1$. Now the value is $s = 2$.

Let's then decrease the amount of the points. Let's use $I = 7$ points. Then the step size gets bigger.

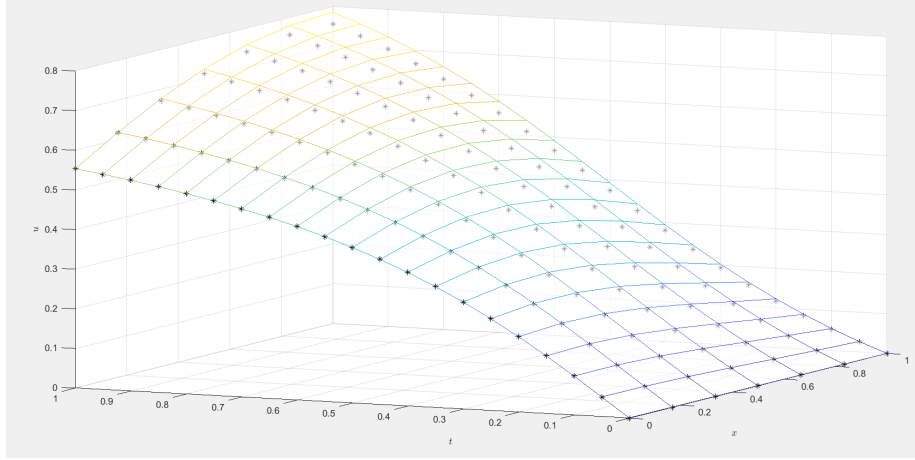


Figure 5: The graphic solution with $I = 7$ in the x -direction

As we can see from the graph, the solution is now stable again. Let's notice that now $s = 0.6$.

These changes in the stability of the solution appear because for any advection equation the condition $s = \frac{\Delta t}{\Delta x}$ must hold with $0 \leq s \leq 1$ for the finite difference method to be stable.

2.2

Let's solve the PDE $2u_x - u_t = 0$ with boundary conditions $u(x, 0) = \frac{1}{x^2+1}$, $u(1, t) = \frac{1}{(1+2t)^2+1}$ using finite difference method.

Because the boundary condition $u(1, t) = \frac{1}{(1+2t)^2+1}$ is given in the end point of x 's interval of consideration, we need to use forward difference approximation.

Let's start by writing the differential equation using difference approximations.

$$\begin{aligned} 2u_x - u_t = 0 &\Rightarrow 2 \cdot \frac{u_{i+1,n} - u_{i,n}}{\Delta x} - \frac{u_{i,n+1} - u_{i,n}}{\Delta t} = 0 \\ &\Rightarrow \frac{2\Delta t}{\Delta x} (u_{i+1,n} - u_{i,n}) - u_{i,n+1} + u_{i,n} = 0 \end{aligned} \quad (13)$$

Let's denote $s = \frac{2\Delta t}{\Delta x}$ and solve $u_{i,n+1}$ from the equation.

$$\begin{aligned}
u_{i,n} &= su_{i+1,n} - su_{i,n} + u_{i,n} \\
&= (1-s)u_{i,n} + su_{i+1,n}
\end{aligned} \tag{14}$$

Let's denote the number of points we use in x -direction with I . Therefore, we get the following system of equations. Let's directly write it in the matrix form. The last equation we get from the boundary condition $u(1, t) = \frac{1}{(1+2t)^2+1}$.

$$\begin{bmatrix} u_{1,n+1} \\ u_{2,n+1} \\ \vdots \\ u_{I,n+1} \end{bmatrix} = \begin{bmatrix} 1-s & s & 0 & \cdots & 0 \\ 0 & 1-s & s & \cdots & 0 \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} u_{1,n} \\ u_{2,n} \\ \vdots \\ u_{I,n} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ \frac{1}{(1+2t_{n+1})^2+1} \end{bmatrix} \tag{15}$$

Let's denote the number of points in the t 's interval of consideration with N and write the Matlab code to solve the equation.

```

%% 2.2
clearvars
close all
clc

% Let's define the needed variables
I = 11;
N = 31;
x = linspace(0,1,I)';
t = linspace(0,1,N)';

dt = t(2)-t(1);
dx = x(2)-x(1);

uI = 1./(x.^2+1);

s = 2*dt/dx;

% Let's make the multiplier matrix A
A = zeros(I,I);
for n = 1:I-1
    A(n,n) = 1-s;
    A(n,n+1) = s;
end

% Let's make the b vector
b = zeros(I,1);

% Let's make a matrix U where we save the solution

```

```

U = zeros(I,N);
U(:,1) = uI;

% Let's make a loop to calculate the solutions.
for n = 1:N-1
    b(end) = 1./((1+2*t(n+1)).^2+1);
    un = A*U(:,n)+b;
    U(:,n+1) = un;
end

% Let's draw the solution
figure
[X,T] = meshgrid(x,t);
Z = 1./((X+2*T).^2+1);
mesh(X,T,Z,'FaceAlpha','0.6')
xlabel('x','interpreter','latex')
ylabel('t','interpreter','latex')
zlabel('u','interpreter','latex')
hold on

for i = 1:I
    plot3(x(i)*ones(1,N),t,U(i,:), 'k*')
end

```

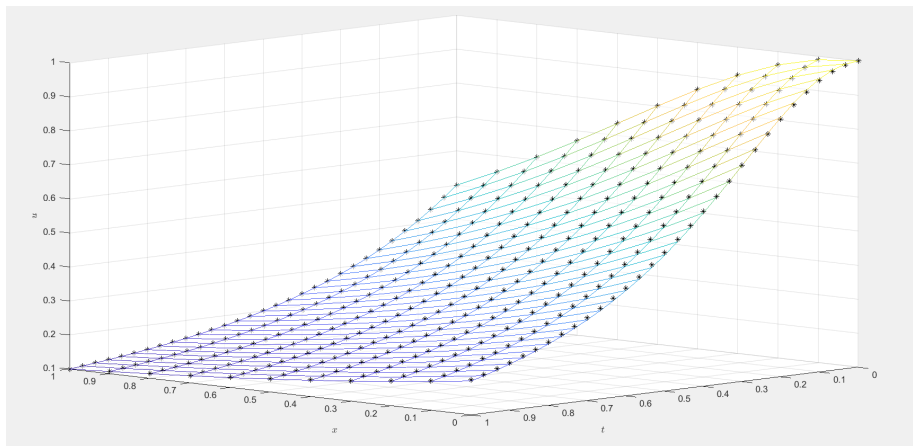


Figure 6: The graphic solution

3 Week 3

3.1

Let's implement a MATLAB function that plots a given function $f \in \mathbb{R}^2$ within close ball domain $B(a, r)$. Let's then plot the function $f = xy$ in $B(a, r)$ with $a = (1, 1)$ and $r = 5$.

```
%% 3.1
clearvars
close all
clc

% The function call
f = @(x,y) x.*y;
a = [1,1];
r = 5;
w3e1(f,r,a);

function w3e1(f,r0,a)

    % Let's define needed values
    m = 50;
    t = linspace(0,2*pi,m)';
    r = linspace(0,r0,m);

    % For practice, let's make the meshgrid without the meshgrid command
    T = zeros(m,m);
    for i = 1:m
        T(:,i) = ones(m,1)*t(i);
    end

    R = zeros(m,m);
    for i = 1:m
        R(i,:) = ones(1,m)*r(i);
    end

    % Let's define x and y using polar coordinates
    x = R.*cos(T)+a(1);
    y = R.*sin(T)+a(2);
    z = f(x,y);

    mesh(x,y,z,'FaceAlpha','0.6')
    xlabel("x")
    ylabel("y")
    zlabel("z")
```

```

    title("Function f in a closed ball domain B(a,r)")
end

```

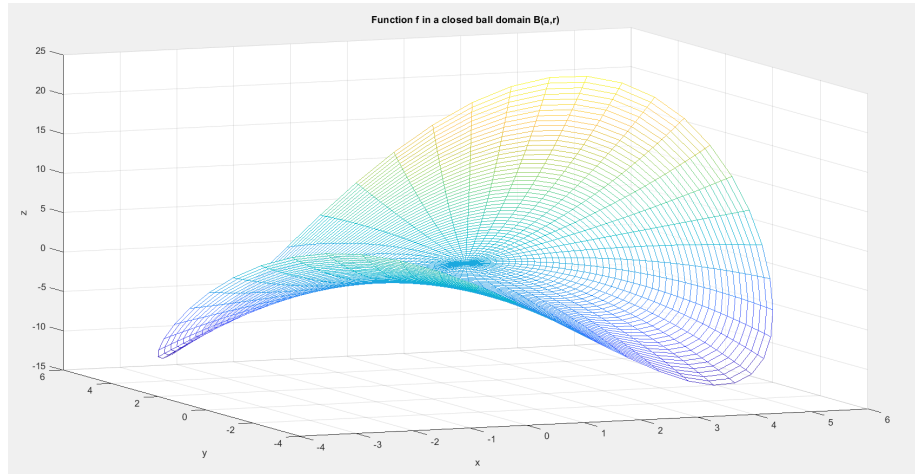


Figure 7: The function $f = xy$ in $B(a, r)$

3.2

Let's first solve x_2 in terms of x_1 from the definition of the p -norm.

$$\begin{aligned}
 \|\mathbf{x}\|_p &= (|x_1|^p + |x_2|^p)^{\frac{1}{p}} \\
 \Rightarrow 1 &= (|x_1|^p + |x_2|^p)^{\frac{1}{p}} \\
 \Rightarrow 1 &= |x_1|^p + |x_2|^p \\
 \Rightarrow |x_2|^p &= (1 - |x_1|^p) \\
 \Rightarrow |x_2| &= (1 - |x_1|^p)^{\frac{1}{p}}
 \end{aligned} \tag{16}$$

Let's plot an unit circle in \mathbb{R}^2 and demonstrate how it changes when $p \rightarrow \infty$. Because $\|\mathbf{x}\|_p = 1$, we get $|x_1| \leq 1$ and $|x_2| \leq 1$. Therefore, we can define x_1 as a vector that has values from 0 to 1.

```

%% 3.2
clearvars
close all
clc

% Let's define needed variables
m = 50;

```

```

x1 = linspace(0,1,m);

% Let's make a numerical function out of the p-norm formula
x2 = @(p) (1-x1.^p).^(1/p);

% Let's initialize the figure
figure
hold on
axis([-1.5,1.5,-1.5,1.5])
xlabel("x1")
ylabel("x2")
title("p-norm when the value of p increases ")

% To draw the whole circle we need 4 different plot commands
P1 = plot(x1,x2(2),'b');
P2 = plot(-x1,x2(2),'b');
P3 = plot(x1,-x2(2),'b');
P4 = plot(-x1,-x2(2),'b');

% Let's implement the animation using a loop
for p = 3:500
    pause(1)
    % We only need to change the data of x2
    % x1 remains the same
    P1.YData = x2(p);
    P2.YData = x2(p);
    P3.YData = -x2(p);
    P4.YData = -x2(p);
end

```

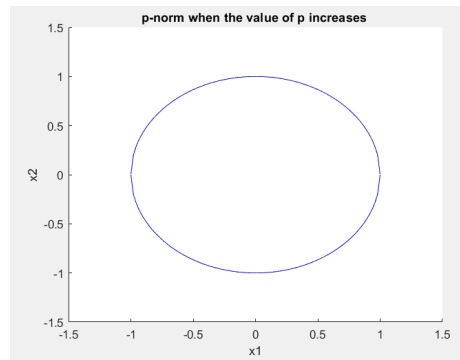


Figure 8: $p = 3$

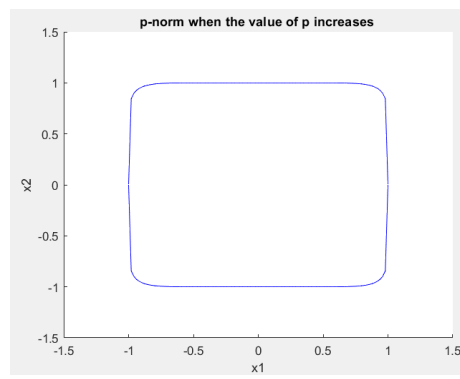


Figure 9: $p = 20$

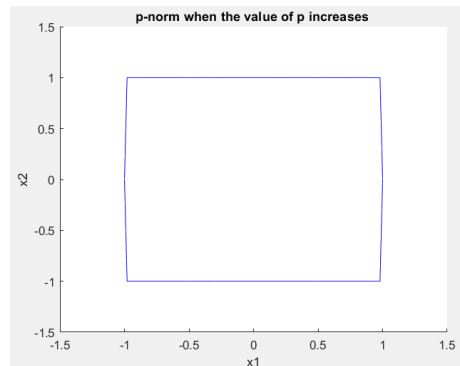


Figure 10: $p = 500$

The graph seems to turn into a square.

4 Week 4

4.1

Let's implement the General Leibniz Rule for two functions $f, g \in C^k(\mathbb{R}^2)$.

```
%% 4.1
% Leibniz rule task
clc
close all
clear
% Define two functions  $C^k(\mathbb{R}^2)$ 
syms x y
f(x,y) = x^2+y^2;
g(x,y) = exp(x+y);

% Define the order of differentiation for each variable
alpha = [4,3];

%Evaluation point
x_diff = [1,1];

% Compute the derivative using General Leibniz Rule
D_alpha_fg = general_leibniz_rule(f, g, alpha, x_diff, [x,y]);

% Check
% h(x,y) = f(x,y)*g(x,y);
% dhx(x,y) = diff(h,x,alpha(1));
```

```

% dhxy(x,y) = diff(dhx,y,alpha(2));
% dhxy(x_diff(1),x_diff(2));

function D_alpha_fg = general_leibniz_rule(f, g, alpha, x_diff, var)
    % Initialize the result
    D_alpha_fg = 0;

    x = var(1);
    y = var(2);
    % Generate all multi-indices beta such that beta <= alpha
    betas = generate_multi_indices(alpha);

    % Loop over all beta multi-indices
    for i = 1:size(betas, 1)
        beta = betas(i,:);
        alpha_minus_beta = alpha-beta;

        % Compute the multinomial coefficient
        coeff = multinomial_coeff(alpha, beta);

        % Implement symbolic derivation logic:
        Df_abx = diff(f,x,alpha_minus_beta(1));
        Df_abxy = diff(Df_abx,y,alpha_minus_beta(2));

        Dg_bx = diff(g,x,beta(1));
        Dg_bxy = diff(Dg_bx,y,beta(2));

        A = subs(Df_abxy,[x,y],x_diff);
        B = subs(Dg_bxy,[x,y],x_diff);

        % Update the solution value,alpha_minus_beta(1)
        D_alpha_fg = D_alpha_fg + coeff*A*B;
    end
end

function coeff = multinomial_coeff(alpha, beta)
    coeff_f = @(j) factorial(alpha(j))...
        / (factorial(beta(j))*factorial(alpha(j)-beta(j)));
    coeff = coeff_f(1)*coeff_f(2);
end

function indices = generate_multi_indices(alpha)
    % Generate all multi-indices beta such that beta <= alpha

```

```

k = max(alpha);
n = factorial(k+1)/factorial(k+1-2);
indices = zeros(n,2);

lineStart = 1;
lineEnd = 0;

for i = 1:alpha(1)+1
    lineEnd = lineEnd + alpha(2)+1;
    indices(lineStart:lineEnd,1) = i-1;
    indices(lineStart:lineEnd,2) = 0:alpha(2);
    lineStart = lineStart + alpha(2)+1;
end

% Let's take empty lines out
if lineEnd < n
    indices = indices(1:lineEnd,:);
end
end

```

4.2

Let's implement a function that generates a Taylor polynomial of degree k of function $f \in C^k(\mathbb{R}^2)$.

```

%% 2.2
clc
close all
clearvars

% Define the function
syms x y
f = 0.2*exp(0.5*x+3*y);

expansion_point = [1,1];
expansion_order = 5;

% % Plot the original function

r = 0.5;
m = 50;
p_line = linspace(1-r,1+r,m);
[x_p,y_p] = meshgrid(p_line,p_line);

```

```

z_p = 0.2*exp(0.5*x_p+3*y_p);
m_o = mesh(x_p,y_p,z_p,'FaceAlpha','0.6',FaceColor='r');
hold on
xlabel("x")
ylabel("y")
zlabel("z")

% Go through each of the different expansion orders and plot each surface
for k = 1:expansion_order
    ts = taylor_series(f, [x,y], expansion_point, k);
    ts_p = matlabFunction(ts);
    z_p = ts_p(x_p,y_p);
    mesh(x_p,y_p,z_p,'FaceAlpha','0.6',FaceColor='none');
end

function ts = taylor_series(func, vars, point, k)
    % Initialize output
    taylor_expansion = 0;

    % Generate all values of alphas
    alphas = generate_combs(k);

    % Generate the series
    % Go through each tuple alpha (combination of derivatives)
    for i = 1:length(alphas)
        cur_alpha = alphas(i,:);

        % Implement derivation logic:
        dfx = diff(func,vars(1),cur_alpha(1));
        df = diff(dfx,vars(2),cur_alpha(2));

        % Evaluate the value of the derivative at expansion point
        dval = subs(df,vars,point);

        % Generate one term of the Taylor expansion
        alpha_fac = factorial(cur_alpha(1))*factorial(cur_alpha(2));
        term = dval/alpha_fac*(vars(1)-point(1))^cur_alpha(1)...
            *(vars(2)-point(2))^cur_alpha(2);

        % Add the new term to the expansion
        taylor_expansion = taylor_expansion + term;
    end
    ts = simplify(taylor_expansion);
end

```

```

function derivatives = generate_combs(k)
    n = factorial(k+1)/factorial(k+1-2);
    derivatives = zeros(n,2);
    line = 1;
    for i = 1:k+1
        for j = 1:k+1
            if i == 1 && j == 1
                line = line+1;
            elseif ((i-1)+(j-1) <= k )
                derivatives(line,:) = [i-1,j-1];
                line = line+1;
            end
        end
    end
    if line < n
        if derivatives(line,1) == 0 && derivatives(line,2) == 0
            derivatives = derivatives(1:line-1,:);
        end
    end
end

```

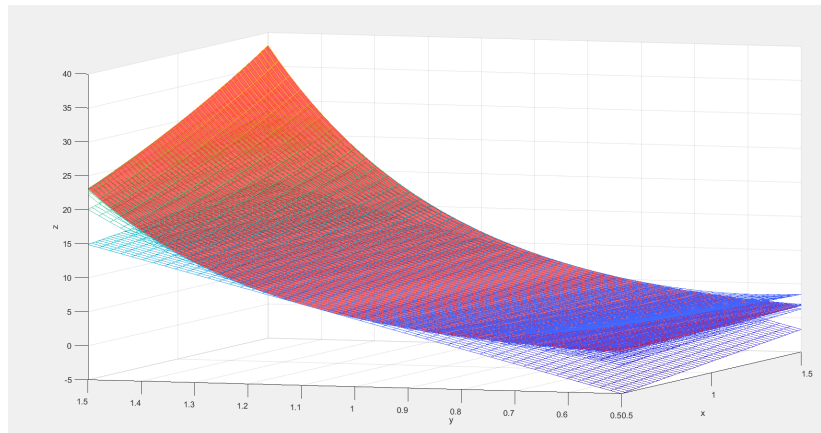


Figure 11: The red graph is the graph of the original function f and the blue graphs are Taylor approximations when $k = 1, 2, 3, 4, 5$

5 Week 5

5.1

Let's use MATLAB to verify the Mean Value Theorem for the following functions $f(x, y) = x^2 + y^2$ and $g(x, y) = \sin(x) + \cos(x)$.

```
%% 5.1
clearvars
close all
clc

% Let's initialize needed values.
f = @(x) x(1).^2+2*x(2).^2;
g = @(x) sin(x(1))+cos(x(2));

a = [1,2]';
b = [3,4]';

f_minus = f(b)-f(a);
g_minus = g(b)-g(a);

% Let's define the gradients of f and g
f_grad = @(x) numGrad(f,x);
g_grad = @(x) numGrad(g,x);

% Let's make a tolerance to check that our values are close enough zero
tol = 1e-4;

% Let's define the direction vector of the line through points a and b.
v = b-a;

% Let's make the points c using the parametric presentation of a line
m = 1e+6;
t = linspace(0,1,m);
c = a+v*t;

% Let's make a loop where we check if the desired point exists.
for i = 1:m
    value_f = f_grad(c(:,i))'*v;
    if abs(value_f-f_minus) < tol
        ind_f = i;
        c_f = c(:,ind_f);
        t_f = t(ind_f);
        break
    end
end
```

```

        end
    end

    for i = 1:m
        value_g = g_grad(c(:,i))'*v;
        if abs(value_g-g_minus) < tol
            ind_g = i;
            c_g = c(:,ind_g);
            t_g = t(ind_g);
            break
        end
    end

    disp('Desired point in case of function f:')
    disp(c_f)
    disp('Desired point in case of function g:')
    disp(c_g)

    % Let's implement the graphical solution.
    m_plot = 1e+2;
    t_plot = linspace(0,1,m_plot);
    c_plot = a+v*t_plot;

    f_values = zeros(m_plot,1);
    g_values = zeros(m_plot,1);

    for i = 1:m_plot
        f_values(i) = f_grad(c_plot(:,i))'*(b-a);
        g_values(i) = g_grad(c_plot(:,i))'*(b-a);
    end

    figure
    plot(t_plot,f_values)
    hold on
    plot(t_plot,f_minus*ones(m_plot,1))
    plot(t_f,value_f,'ko')
    title("f")
    xlabel("t")

    figure
    plot(t_plot,g_values)
    hold on
    plot(t_plot,g_minus*ones(m_plot,1))
    plot(t_g,value_g,'ko')
    title("g")
    xlabel("t")

```

```

% Let's make a numerical function for calculation gradient
function g = numGrad(f,x)
    h = 1e-8;
    n = length(x);
    g = zeros(n,1);
    for i = 1:n
        xi1 = x;
        xi2 = x;
        xi1(i) = x(i)+h;
        xi2(i) = x(i)-h;
        g(i) = (f(xi1)-f(xi2))/(2*h);
    end
end

```

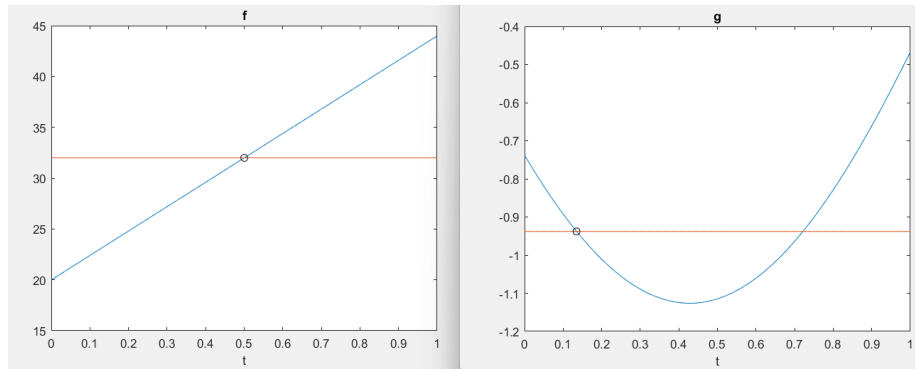


Figure 12: The values of the variable $t \in [0, 1]$ are in the horizontal axis. The red lines are the differences $f(\mathbf{b}) - f(\mathbf{a})$ and $g(\mathbf{b}) - g(\mathbf{a})$. The blue graphs are the functions $\nabla f(\mathbf{c})(\mathbf{b} - \mathbf{a})$ and $\nabla g(\mathbf{c})(\mathbf{b} - \mathbf{a})$, where $\mathbf{c} = \mathbf{a} + t \cdot (\mathbf{b} - \mathbf{a})$


```

Command Window

Desired point in case of function f:
    1.999992999993000
    2.999992999993000

Desired point in case of function g:
    1.269358269358269
    2.269358269358269

fx >>

```

Figure 13: The desired points \mathbf{c} are printed when running the code.

5.2

Let's examine numerically whether $u(r, \theta) = \ln(r) + \cos(\theta)$ is a harmonic function.

```

%% 5.2
clearvars
close all
clc

% Let's define the function u as a numerical function
u = @(r,t) log(r)+cos(t);

% Let's make approximations of the derivatives
dr = 1e-4;
dt = pi*1e-5;
u_r = @(r,t) (u(r+dr,t)-u(r-dr,t))./(2*dr);
u_rr = @(r,t) (u(r+dr,t)-2*u(r,t)+u(r-dr,t))./dr^2;
u_tt = @(r,t) (u(r,t+dt)-2*u(r,t)+u(r,t-dt))./dt^2;

% Let's make the delta u function using polar Laplacian operator
delta_u = @(r,t) u_rr(r,t)+1./r.*u_r(r,t)+1./r.^2.*u_tt(r,t);

% Let's make points where we calculate the value of delta u
m = 1e+3;
r = 100*rand(m,1)+0.1;
t = linspace(-pi,pi,m)';
[R,T] = meshgrid(r,t);

% Let's calculate the sum

```

```

sum_of_values = sum(sum(delta_u(R,T)));
str = string(sum_of_values);
disp('The sum is '+str)

```

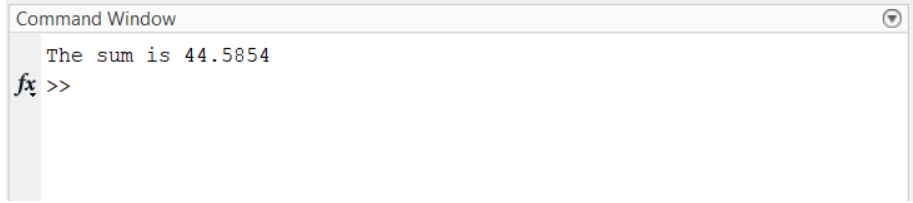


Figure 14: The MATLAB program prints the sum of the calculated values of Δu .

We don't get a value near zero every time we run the code. Therefore, the function u is not harmonic function according to the numerical examination. Let's proof this analytically.

The function u is a harmonic function if it satisfies the Laplace equation

$$-\Delta u = 0 \Leftrightarrow \frac{\partial^2 u}{\partial r^2} + \frac{1}{r} \frac{\partial u}{\partial r} + \frac{1}{r^2} \frac{\partial^2 u}{\partial \theta^2} = 0, \quad \forall (r, \theta) \in (0, \infty) \times (-\pi, \pi] \quad (17)$$

Let's calculate the partial derivatives that are in the equation.

$$\begin{aligned} \frac{\partial u}{\partial r} &= \frac{1}{r} \\ \frac{\partial^2 u}{\partial r^2} &= -\frac{1}{r^2} \end{aligned} \quad (18)$$

$$\frac{\partial u}{\partial \theta} = -\sin(\theta)$$

$$\frac{\partial^2 u}{\partial \theta^2} = -\cos(\theta)$$

We get

$$\frac{\partial^2 u}{\partial r^2} + \frac{1}{r} \frac{\partial u}{\partial r} + \frac{1}{r^2} \frac{\partial^2 u}{\partial \theta^2} = -\frac{1}{r^2} + \frac{1}{r} \cdot \frac{1}{r} + \frac{1}{r^2} \cdot (-\cos(\theta)) = -\frac{\cos(\theta)}{r^2} \quad (19)$$

If $\theta \neq \frac{\pi}{2} + n \cdot \pi, n \in \mathbb{Z}$, then $-\Delta u \neq 0$. Thus u is not harmonic.

If we run the same MATLAB code with function $u(r, \theta) = \ln(r) + \theta$, the value of the sum is near zero every time. Therefore, in this case the function u is a harmonic function according to the numerical calculations.

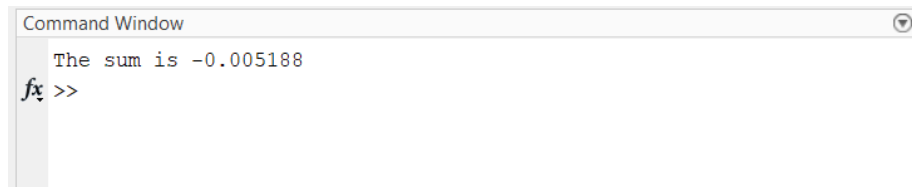


Figure 15: The printed output of the MATLAB program when the function is $u(r, \theta) = \ln(r) + \theta$.

6 Week 6

6.1

Let's numerically verify that the Mean Value Theorem is satisfied for the function $f(x, y) = x^2 - y^2$.

```
%% 6.1
clearvars
close all
clc

f = @(x,y) x.^2-y.^2;

n = 2;
r = 1;
alpha2 = pi;

surface_area = n*alpha2*r.^(n-1);

f_tilde = @(theta) f(r*cos(theta),r*sin(theta));

f_zero_mvt = 1/surface_area*integral(f_tilde,0,2*pi);

disp('f(0,0) = '+string(f(0,0)))
disp('Using MVT we get f(0,0) = '+string(f_zero_mvt))
```

```
Command Window
f(0,0) = 0
Using MVT we get f(0,0) = -9.7184e-17
fx >>
```

Figure 16: The printed output when the program is run

With the expression of the function f we get $f(0,0) = 0$ and numerically using Mean Value Theorem we get $f(0,0) \approx 0$. Therefore, the Mean Value Theorem is numerically verified for the function f .

6.2

Let's perform mollification of the function $f(x) = |x|$ using a mollifier function η_ϵ .

```
%% 6.2
clearvars
close all
clc

% Let's define needed variables.
f = @(x) abs(x);
m = 1000;
y_p = linspace(-1,1,m)';
x_p = linspace(-1,1,m);

% Let's define the mollifier function
mol = @(x) exp(1./(abs(x).^2-1));
C = 1./integral(mol,-1,1);
mol_eps = @(epsilon,x) epsilon.^(-1).*C.*mol(x./epsilon);

% Let's plot the original function.
plot(x_p,f(x_p),'k')
hold on

% Let's make a variable for the differences of x and y
differences = x_p-y_p;
```

```

% Let's calculate values for the mollified function for different values of
% epsilon

plot_iterations = 5;
epsilon_values = linspace(0,1,plot_iterations);
for epsilon = epsilon_values
    f_epsilon = convolution(f,y_p,differences,epsilon,m,mol_eps);
    plot(x_p,f_epsilon)
end

% Let's adjust epsilon
epsilon = 0;

f_epsilon = convolution(f,y_p,differences,epsilon,m,mol_eps);

% Let's use least squares method to adjust the epsilon parameter
min_sq_sum = sum((f(y_p)-f_epsilon).^2);
best_epsilon = epsilon;

iterations = 500;
best_f_epsilon = f_epsilon;

for epsilon = linspace(0,0.1,iterations)

    f_epsilon = convolution(f,y_p,differences,epsilon,m,mol_eps);
    sq_sum = sum((f(y_p)-f_epsilon).^2);

    % Let's update the value of the square sum
    if sq_sum < min_sq_sum
        min_sq_sum = sq_sum;
        best_epsilon = epsilon;
        best_f_epsilon = f_epsilon;
    end
end

plot(y_p,best_f_epsilon,'--')
disp("Adjusted epsilon: "+string(best_epsilon))

function f_epsilon = convolution(f,y_p,differences,epsilon,m,mol_eps)

    % Let's define the step size in the integral approximations
    dy = abs(y_p(2)-y_p(1));

```

```

% Let's implement the piecewise functions using logical operations
ind1 = abs(y_p) < 1;
ind2 = abs(differences/epsilon) < 1;
ind = ind1.*ind2;

% Let's initialize a vector for the values of the mollified function.
f_epsilon = zeros(m,1);

% Let's make a loop where we calculate the values of the mollified
% function
for i = 1:length(y_p)
    % Let's find the indexes where the mollifier function and f have
    % non-zero values
    non_zero_index = find(ind(:,i) > 0);
    y = y_p(non_zero_index);
    difference = differences(non_zero_index,i);
    % Let's calculate the integral
    f_epsilon(i) = dy*sum(f(y).*mol_eps(epsilon,difference));
end
end

```

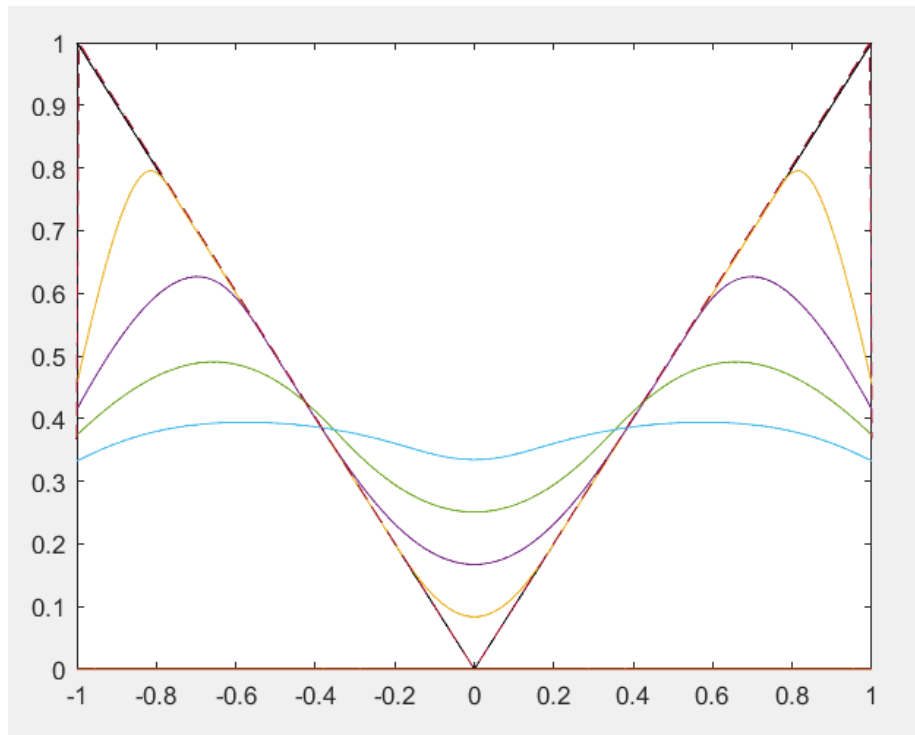


Figure 17: The graph of the original function f and the graphs of the mollified functions for different values of ϵ

It seems that the mollified function approximates the function f well when the parameter ϵ is chosen near zero. However, when the value gets below a certain point, the approximation gets unstable.

```

Command Window
Adjusted epsilon: 0.0062124
fx >>

```

Figure 18: The output of the program

7 Week 7

7.1

Let's verify the maximum-minimum principle for the function $u(x, y) = x^2 - y^2$ on the domain $\Omega = [-1, 1] \times [-1, 1]$.

```
%% 7.1
clearvars
close all
clc

% Let's define the function
u = @(x,y) x.^2-y.^2;

% Let's define polar coordinates
x = @(r,t) r.*cos(t);
y = @(r,t) r.*sin(t);

% Let's define variables for the polar coordinates
t = linspace(0,2*pi);
r = linspace(0,1-1e-3);

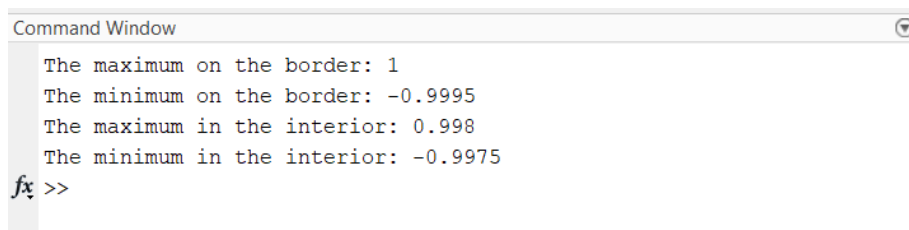
% Let's calculate the maximum value of the border
border = u(x(1,t),y(1,t));
max_border = max(max(border));
min_border = min(min(border));

% Let's make a grid for the interior
[T,R] = meshgrid(t,r);

% Let's calculate the maximum value of the interior
interior = u(x(R,T),y(R,T));
max_interior = max(max(interior));
min_interior = min(min(interior));

disp("The maximum on the border: "+string(max_border))
disp("The minimum on the border: "+string(min_border))

disp("The maximum in the interior: "+string(max_interior))
disp("The minimum in the interior: "+string(min_interior))
```


A screenshot of a MATLAB Command Window. The window has a title bar that says "Command Window" and a small icon on the right. The text inside the window is as follows:

```
The maximum on the border: 1
The minimum on the border: -0.9995
The maximum in the interior: 0.998
The minimum in the interior: -0.9975
fx >>
```

Figure 19: The output of the program

As we can see from the output of the program, the function u has its maximum and minimum values on the boundary of the domain. Therefore, the maximum-minimum principle for the function $u(x, y) = x^2 - y^2$ is numerically verified.

7.2

Let's numerically verify that the derivative estimates theorem holds for a function $u(x, y) = \sin(x)\sin(y)$ on the domain $\Omega = [-\pi, \pi] \times [-\pi, \pi]$.

```
%% 7.2
clearvars
close all
clc

% Let's initialize needed variables
u = @(x,y) sin(x).*sin(y);
m = 10;
x = linspace(-pi,pi,m);
y = x;

% Let's calculate the values of u in the domain
[X,Y] = meshgrid(x,y);
U = u(X,Y);
```

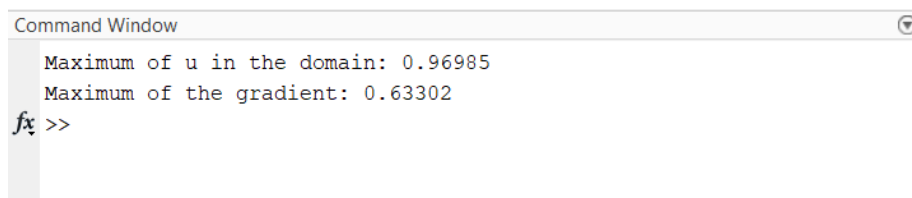
```

% Let's define the gradient
[u_x,u_y] = gradient(U);
grad_mag = sqrt(u_x.^2+u_y.^2);

% Let's calculate the maximum of the gradient
u_max = max(max(U));
grad_max = max(max(grad_mag));

disp("Maximum of u in the domain: "+string(u_max))
disp("Maximum of the gradient: "+string(grad_max))

```



The image shows a MATLAB Command Window with the following text:

```

Command Window
Maximum of u in the domain: 0.96985
Maximum of the gradient: 0.63302
fx >>

```

Figure 20: The output of the program

As we can see from the output of the program, the maximum value of the magnitude of the gradient vector is smaller than the maximum value of the function u in the domain. Therefore the magnitude of the gradient is bounded by the maximum value of u on the domain and the derivative estimates theorem holds.