UNDERGRADUATE RESEARCH COURSE
Mathematics Department

☒ Fall ☐ Spring
☐ Summer I ☐ Summer II 20 21

Fox Brandon
Last Name First Name

N 0 3 7 7 2 1 4 2

foxb3@newpaltz.edu
Email address

Course No. **MAT490** Section # 02 CRN# 2528 Credits 3 (LA)

A. Describe the proposed research project. Type/print clearly. Use additional pages if necessary. What is the proposed method of study? Where appropriate include a list of readings, practical experience, and/or a description of your research design.

(See attached document)

Brandon Fox 12/6/21
Signature of Student Date

A. To be completed by instructor: On what basis will the project be evaluated? What assignments will be required?

Please see the attached

Signature of Instructor

Please PRINT instructor's name: Jenngeun Park

12/14/21
Signature of Department Chair Date

"Research in Mathematics": Research Project Description

Name: Brandon Fox

Date: December 3rd, 2021

My research project was on image processing by utilizing the Gaussian diffusion/heat PDE with both the Dirichlet and Neumann boundary conditions. An essential skill to this was knowing how to do MATLAB coding.

Since I was new to MATLAB, I had done weekly MATLAB coding exercises. These exercises were in conjunction with working with images, in general, and image processing. Exercises had included, but were not limited to: image importing and making and formatting plots and subplots with MATLAB code, extraction of color channels, image transforming with power, exponential, logarithmic, linear, and piecewise linear functions, probability histogram analysis of light and dark pixels of particular images, basic linear and matrix transformations performed on images, denoising of an image by Gaussian filtering, one-dimensional and two-dimensional convolution, and variations of edge detecting.

Also, in order to arrive at the actual diffusion equation itself, I had utilized the idea of difference quotients. First, Euler's Method was rethought as a recursive definition, defined by our initial value and what's known as a "forwards difference quotient", since we approximate the next value of a function f for every Δx step. Next, other difference quotients had to be defined. They are obviously called the "backwards difference quotient" and the "centered difference quotient". Since the "forwards difference quotient" and the "backwards difference quotient" are defined by Taylor series, they were used in terms of their respective Taylor series to give the actual diffusion equation itself. Also, other concepts and principles (i.e. the main "Stability

Criterion", $s = t/x$^2 $< 1/2$, the "Maximum Principle") were learned and applied to enhance the idea of how the diffusion equation can be better applied, overall.

Afterwards, to gain an explicit idea of how the diffusion equation works as an application of denoising, I had done one-dimensional denoising of lines over t time. Of course, they were done under the restraints of the Dirichlet and Neumann boundary conditions. These types of boundary conditions are established under our initial data that also consists of general t time, such that these boundary conditions act as noise vanishing lines when a certain t time gets met. By doing this, we would be able to see the evolution of the denoising of the one-dimensional lines as we reach certain points in time, such that we would end up reaching an appropriate solution to the diffusion PDE when t is some time value greater than 0 and $0 < x < L$, where L is a Dirichlet and Neumann boundary limit, retained under our initial data; $u(x, 0) = f(x)$, where f is a general solution to the diffusion/heat problem, depending on what the Dirichlet and Neumann boundary conditions actually are.

Currently, I'm at the point where I hope to apply a two dimensional coefficient of diffusivity in a gradient (∇, or "del") direction towards sharpening the edges of some image. Hopefully, in terms of this and the diffusion equation where denoising occurs, I will arrive at the goal of creating purely denoised images without any blurring. This would well reflect the idea of "image processing", which is to improve the quality of images as effectively as possible.

Readings:

1. *An Image Processing Tour of Mathematics*, by Y. V. Galperin, CRC Press, first edition, pages 1-336, 2021.
2. *Introduction to MATLAB for Engineering Students*, by D. Houcque, *Northwestern University*, version 1.2, pages 1-64, 2005.
3. *Partial Differential Equations: An Introduction*, by W.A. Strauss, John Wiley & Sons Inc., second edition, pages 1-454. 2008.
4. *Perona-Malik equation and its numerical properties*, by M. Weilgus, *Uniwersytet Warszawski*, pages 1-26, 2014.

- Assessment Criteria

Objectives	Outstanding (5)	Good (4)	Average (3)	Deficient (2)	Inadequate (1)
Planning and organization	Student is an active participant in the various stages of developing and implementing project and comes to meetings prepared for collaboration with mentor	Given initial encouragement, completes tasks efficiently and appropriately	Collaborates with faculty support & encouragement, completes most tasks adequately	Passive involvement, requires frequent support & reminders, tasks incomplete	Minimal involvement despite encouragement & reminders, tasks incomplete
Deadlines	Student consistently meets mutually negotiated deadlines and completes assigned tasks in an efficient and timely manner	Given initial encouragement, completes tasks efficiently and appropriately	Collaborates with faculty support & encouragement, completes most tasks adequately	Passive involvement, requires frequent support & reminders, tasks incomplete	Minimal involvement despite encouragement & reminders, tasks incomplete
Initiative	Student consistently takes the initiative to review current literature, develop plan, and propose solutions to problems as they arise	Given initial encouragement, completes tasks efficiently and appropriately	Collaborates with faculty support & encouragement, completes most tasks adequately	Passive involvement, requires frequent support & reminders, tasks incomplete	Minimal involvement despite encouragement & reminders, tasks incomplete
Quality of Work	Work consistently demonstrates effort, critical thought, and a developmentally appropriate	Given initial guidance and feedback, completes tasks efficiently and appropriately	Collaborates with faculty support & encouragement, completes most tasks adequately	Passive involvement, requires frequent support & reminders tasks incomplete	Minimal involvement despite encouragement & reminders, tasks incomplete

	synthesis of information				
Personal Interaction	Student exhibits a positive attitude and works well with faculty mentor. Takes leadership role when appropriate, but also demonstrates a willingness to take direction from others.	Given initial encouragement, completes tasks efficiently and appropriately	Collaborates with faculty support & encouragement, completes most tasks adequately	Passive involvement, requires frequent support &reminders, tasks incomplete	Minimal involvement despite encouragement & reminders, tasks incomplete

Scale: A 22-25, B 19-22 , C 15-18, D <15

- **Required assignments**

1. Regular attendance.
2. Read reading material before meetings and discuss in meetings
   a. Reference 1: Houcque, David. "Introduction to Matlab for engineering students." *Northwestern University* 1 (2005).
   b. Reference 2: Galperin, Yevgeniy V. *An Image Processing Tour of College Mathematics*. CRC Press, 2021.
   c. Reference 3: Strauss, Walter A. *Partial differential equations: An introduction*. John Wiley & Sons, 2007.Reference 4: Wielgus.
   d. Reference 4: Maciek. "Perona-Malik equation and its numerical properties." *arXiv preprint arXiv:1412.6291* (2014).
3. Learn MATLAB as an introduction to programming through Reference 1 by following the guidance.
4. Learn week, learn a practice introduction to image processing with MATLAB through Reference 2. Try practice problems related to MATLAB coding.
5. Learn some basic a type of linear partial differential equations, especially heat equations, through Reference 3.
6. Learn the basic numerical methods to solve differential equations: Euler's method, finite difference method. Solve differential equations numerically by using the methods.
7. Understand how to apply a heat equation to denosing of an image and recognize an arising issue through Reference 4.

Research in Mathematics, MAT490

Name: Brandon Fox

Date: December 14, 2021

Problem 1. . Consider the diffusion equation

$$u_t = u_{xx}, \qquad 0 \leq x \leq 1, \quad t \geq 0, \tag{1}$$

with the Dirichlet boundary condition (2) and initial data (3)

$$u(0,t) = 0 \qquad \text{and} \qquad u(1,t) = 0, \qquad t \geq 0, \tag{2}$$

$$u(x,0) = f(x), \qquad 0 \leq x \leq 1. \tag{3}$$

A. The initial condition is given as

$$f(x) = \begin{cases} 0, & x < 0.4, \quad x > 0.6 \\ 5, & \text{otherwise.} \end{cases} \tag{4}$$

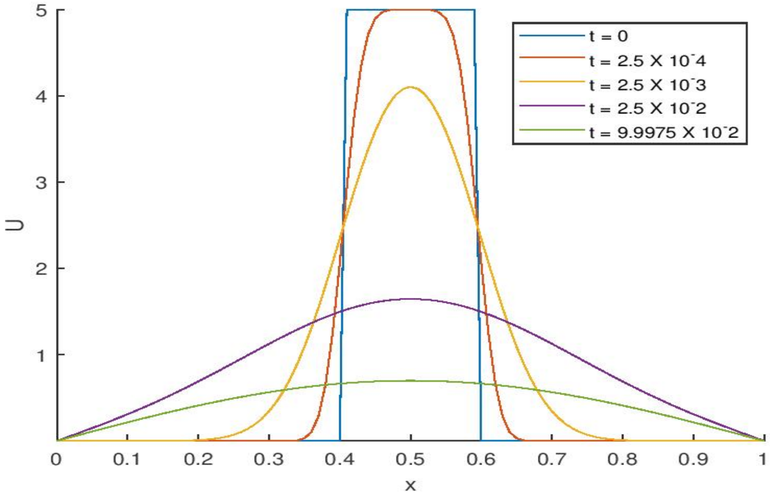


Figure 1: One Dimensional Diffusion with the Dirichlet Boundary Condition and Rectangular Initial Data Over Snapshots of t Time.

```
L = 1;
T = 0.1;
dx = 0.01;
dt = 0.25 * dx * dx;
s = (dt/(dx * dx));
J = L/dx;
K = T/dt;
x = 0:dx:L;
t = 0:dt:T;
U = ones(J + 1, K + 1);
for j = 1:J + 1
 U(j,1) = initial(x(j));
end
for k = 1:K + 1
 U(1,k) = 0;
 U(J + 1,k) = 0;
end
for k = 1:K
for j = 2:J
 U(j,k + 1) = s * (U(j + 1, k) + U(j - 1, k)) + (1 - 2*s) * U(j,k);
 end
end
size(x)
%1 row and 101 columns
size(t)
%1 row and 400001 columns
size(U)
%101 rows and 400001 columns
%1D Dirichlet Diffusion In 2D Space
hold on
plot(x,U(:,1));
plot(x,U(:,11));
plot(x,U(:,101));
plot(x,U(:,1001));
plot(x,U(:,K));
hold off
xticks([0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1]);
xlabel('x');
yticks([1 2 3 4 5]);
ylabel('U');
legend('t = 0','t = 2.5 X 10^-4', 't = 2.5 X 10^-3', 't = 2.5 X 10^-2','t = 9.9975 X ...
    10^-2');
```

```
function f = initial(x)
f = zeros(size(x));
idx = (0.4 < x) && (x < 0.6);
f(idx) = 5;
end
```

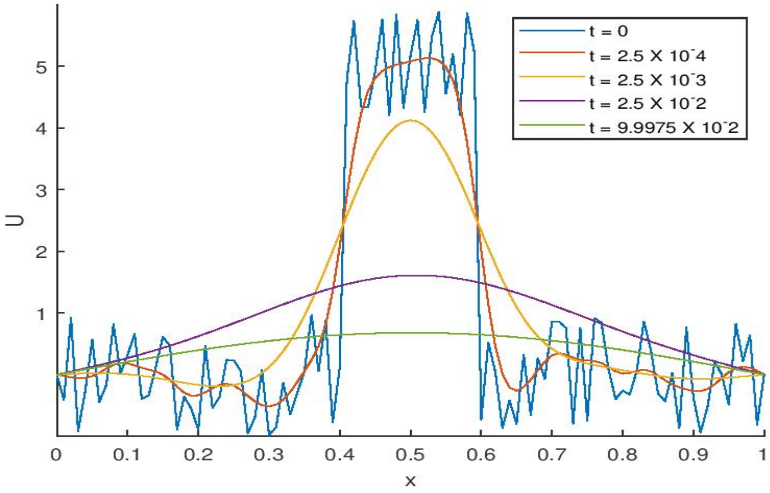


Figure 2: One Dimensional Diffusion With the Dirichlet Boundary Condition and Rectangular Initial Data Over Snapshots of t Time as an Application for Denoising.

```
L = 1;
T = 0.1;
dx = 0.01;
dt = 0.25 * dx * dx;
s = (dt/(dx * dx));
J = L/dx;
K = T/dt;
x = 0:dx:L;
t = 0:dt:T;
U = ones(J + 1, K + 1);
for j = 1:J + 1
 U(j,1) = initial_noise(x(j));
end
for k = 1:K + 1
 U(1,k) = 0;
 U(J + 1,k) = 0;
end
for k = 1:K
for j = 2:J
 U(j,k + 1) = s * (U(j + 1, k) + U(j - 1, k)) + (1 - 2*s) * U(j,k);
 end
end
size(x)
%1 row and 101 columns
size(t)
%1 row and 400001 columns
size(U)
%101 rows and 400001 columns
```

```
%1D Dirichlet Diffusion In 2D Space
hold on
plot(x,U(:,1));
plot(x,U(:,11));
plot(x,U(:,101));
plot(x,U(:,1001));
plot(x,U(:,K));
hold off
xticks([0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1]);
xlabel('x');
yticks([1 2 3 4 5]);
ylabel('U');
legend('t = 0','t = 2.5 X 10^-4', 't = 2.5 X 10^-3', 't = 2.5 X 10^-2','t =
 9.9975 X 10^-2');
```

```
function f = initial_noise(x)
f = zeros(size(x)) + (1 - 2 * (rand(1)));
idx = (0.4 < x) && (x < 0.6);
f(idx) = 5 + (1 - 2 * (rand(1)));
end
```

B. The initial condition is given as

$$f(x) = \begin{cases} x, & x \leq 0.5 \\ 1 - x, & \text{otherwise.} \end{cases} \tag{5}$$

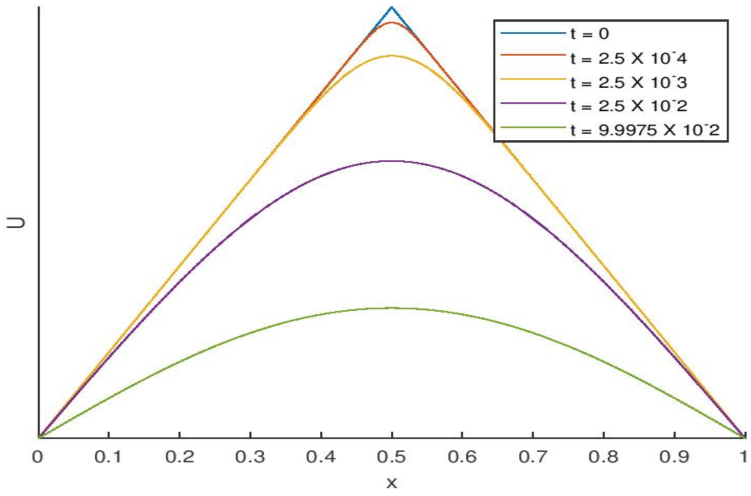


Figure 3: One Dimensional Diffusion With the Dirichlet Boundary Condition and Triangular Initial Data Over Snapshots of t Time.

```
L = 1;
T = 0.1;
dx = 0.01;
dt = 0.25 * dx * dx;
s = (dt/(dx * dx));
J = L/dx;
K = T/dt;
x = 0:dx:L;
t = 0:dt:T;
U = ones(J + 1, K + 1);
for j = 1:J + 1
 U(j,1) = initial_triangle(x(j));
end
for k = 1:K + 1
 U(1,k) = 0;
 U(J + 1,k) = 0;
end
for k = 1:K
for j = 2:J
 U(j,k + 1) = s * (U(j + 1, k) + U(j - 1, k)) + (1 - 2*s) * U(j,k);
 end
end
size(x)
%1 row and 101 columns
size(t)
%1 row and 400001 columns
size(U)
%101 rows and 400001 columns
%1D Dirichlet Diffusion In 2D Space
hold on
plot(x,U(:,1));
plot(x,U(:,11));
plot(x,U(:,101));
plot(x,U(:,1001));
plot(x,U(:,K));
hold off
xticks([0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1]);
xlabel('x');
yticks([1 2 3 4 5]);
ylabel('U');
legend('t = 0','t = 2.5 X 10^-4', 't = 2.5 X 10^-3', 't = 2.5 X 10^-2','t =
 9.9975 X 10^-2');
```

```
function f = initial_triangle(x)
f = zeros(size(x));
idx = (0.5 < x);
f(idx)=1-x;
idx = (x<0.5);
f(idx)=x;
idx = (x==0.5);
f(idx)=0.5;
```

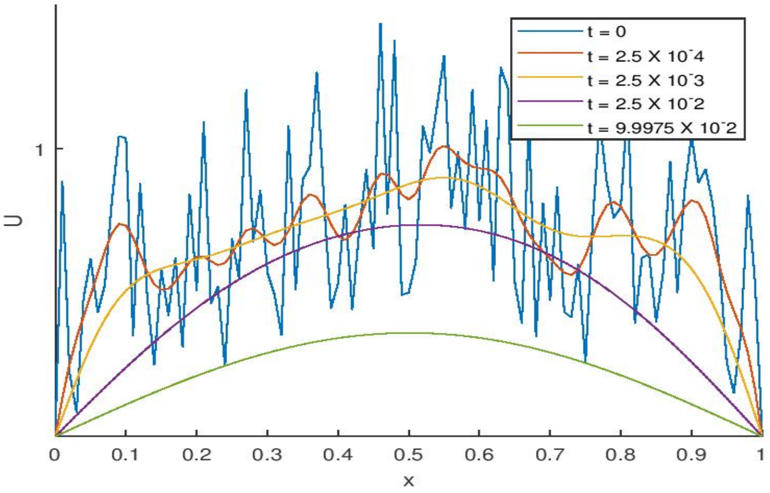


Figure 4: One Dimensional Diffusion With the Dirichlet Boundary Condition and Triangular Initial Data Over Snapshots of t Time As An Application for Denoising.

```
L = 1;
T = 0.1;
dx = 0.01;
dt = 0.25 * dx * dx;
s = (dt/(dx * dx));
J = L/dx;
K = T/dt;
x = 0:dx:L;
t = 0:dt:T;
U = ones(J + 1, K + 1);
for j = 1:J + 1
 U(j,1) = initial_triangle_noise(x(j));
end
for k = 1:K + 1
 U(1,k) = 0;
 U(J + 1,k) = 0;
end
for k = 1:K
 for j = 2:J
 U(j,k + 1) = s * (U(j + 1, k) + U(j - 1, k)) + (1 - 2*s) * U(j,k);
 end
end
size(x)
%1 row and 101 columns
size(t)
%1 row and 400001 columns
size(U)
%101 rows and 400001 columns
```

```
%1D Dirichlet Diffusion In 2D Space
hold on
plot(x,U(:,1));
plot(x,U(:,11));
plot(x,U(:,101));
plot(x,U(:,1001));
plot(x,U(:,K));
hold off
xticks([0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1]);
xlabel('x');
yticks([1 2 3 4 5]);
ylabel('U');
legend('t = 0','t = 2.5 X 10^-4', 't = 2.5 X 10^-3', 't = 2.5 X 10^-2','t = 9.9975 X 10^-2');
```

```
function f = initial_triangle_noise(x)
f = zeros(size(x));
idx = (0.5 < x);
f(idx) = 1 - x + rand(1);
idx = (x < 0.5);
f(idx) = x + rand(1);
idx = (x == 0.5);
f(idx) = 0.5;
```

Problem 2. . Consider the diffusion equation

$$u_t = u_{xx}, \qquad 0 \le x \le 1, \quad t \ge 0, \tag{6}$$

with the Neumann boundary condition (7) and initial data (8)

$$u_x(0,t) = 0 \qquad \text{and} \qquad u_x(1,t) = 0, \qquad t \ge 0, \tag{7}$$

$$u(x,0) = f(x), \qquad 0 \le x \le 1. \tag{8}$$

A. The initial condition is given as

$$f(x) = \begin{cases} 0, & x < 0.4, \quad x > 0.6 \\ 5, & \text{otherwise.} \end{cases} \tag{9}$$

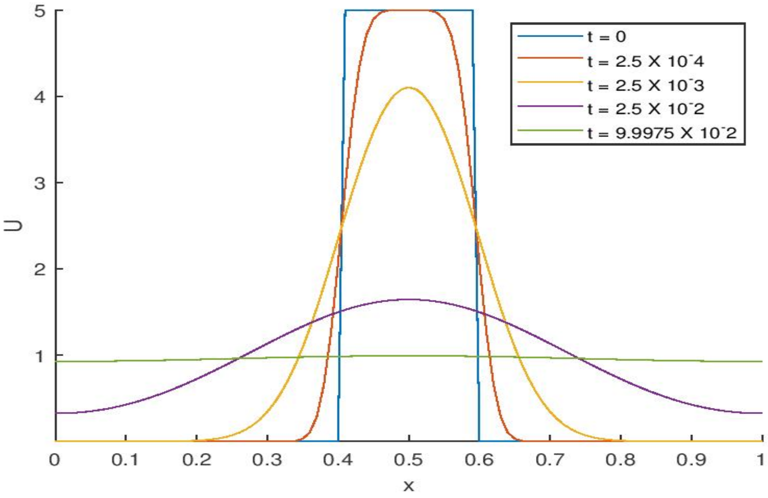


Figure 5: One Dimensional Diffusion With the Neumann Boundary Condition and Rectangular Initial Data Over Snapshots of t Time.

```
L = 1;
T = 0.1;
dx = 0.01;
dt = 0.25 * dx * dx;
s = (dt/(dx * dx));
J = L/dx;
K = T/dt;
x = 0:dx:L;
t = 0:dt:T;
U = ones(J + 1, K + 1);
for j = 1:J + 1
 U(j,1) = initial(x(j));
end
for k = 1:K
for j = 2:J
 U(j,k + 1) = s * (U(j + 1, k) + U(j - 1, k)) + (1 - 2*s) * U(j,k);
end
 U(1,k + 1) = U(2, k + 1);
 U(J + 1,k + 1) = U(J, k + 1);
end
size(x)
%1 row and 101 columns
size(t)
%1 row and 400001 columns
size(U)
%101 rows and 400001 columns
%1D Neumann Diffusion In 2D Space
hold on
```

```
plot(x,U(:,1));
plot(x,U(:,11));
plot(x,U(:,101));
plot(x,U(:,1001));
plot(x,U(:,K));
hold off
xticks([0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1]);
xlabel('x');
yticks([1 2 3 4 5]);
ylabel('U');
legend('t = 0','t = 2.5 X 10^-4', 't = 2.5 X 10^-3', 't = 2.5 X 10^-2','t =
 9.9975 X 10^-2');
```

```
function f = initial(x)
f = zeros(size(x));
idx = (0.4 < x) && (x < 0.6);
f(idx) = 5;
end
```

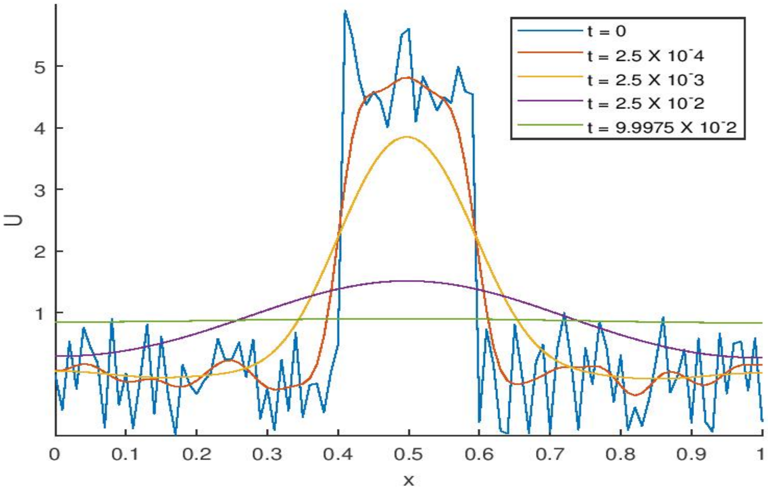


Figure 6: One Dimensional Diffusion With the Neumann Boundary Condition and Rectangular Initial Data Over Snapshots of t Time as an Application for Denoising.

```
L = 1;
T = 0.1;
dx = 0.01;
dt = 0.25 * dx * dx;
s = (dt/(dx * dx));
J = L/dx;
K = T/dt;
x = 0:dx:L;
t = 0:dt:T;
U = ones(J + 1, K + 1);
for j = 1:J + 1
 U(j,1) = initial_noise(x(j));
end
for k = 1:K
for j = 2:J
 U(j,k + 1) = s * (U(j + 1, k) + U(j - 1, k)) + (1 - 2*s) * U(j,k);
end
 U(1,k + 1) = U(2, k + 1);
 U(J + 1,k + 1) = U(J, k + 1);
end
size(x)
%1 row and 101 columns
size(t)
%1 row and 400001 columns
size(U)
%101 rows and 400001 columns
%1D Neumann Diffusion In 2D Space
hold on
plot(x,U(:,1));
plot(x,U(:,11));
plot(x,U(:,101));
plot(x,U(:,1001));
plot(x,U(:,K));
hold off
xticks([0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1]);
xlabel('x');
yticks([1 2 3 4 5]);
ylabel('U');
legend('t = 0','t = 2.5 X 10^-4', 't = 2.5 X 10^-3', 't = 2.5 X 10^-2','t =
 9.9975 X 10^-2');
```

```
function f = initial_noise(x)
f = zeros(size(x)) + (1 - 2 * (rand(1)));
idx = (0.4 < x) && (x < 0.6);
f(idx) = 5 + (1 - 2 * (rand(1)));
end
```

B. The initial condition is given as

$$f(x) = \begin{cases} x, & x \leq 0.5 \\ 1 - x, & \text{otherwise.} \end{cases} \tag{10}$$

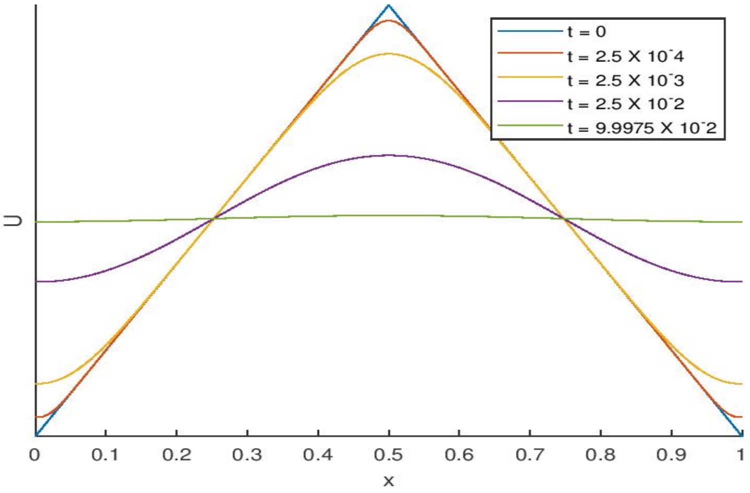


Figure 7: One Dimensional Diffusion With the Neumann Boundary Condition and Triangular Initial Data Over Snapshots of t Time.

```
L = 1;
T = 0.1;
dx = 0.01;
dt = 0.25 * dx * dx;
s = (dt/(dx * dx));
J = L/dx;
K = T/dt;
x = 0:dx:L;
t = 0:dt:T;
U = ones(J + 1, K + 1);
for j = 1:J + 1
 U(j,1) = initial_triangle(x(j));
end
for k = 1:K
for j = 2:J
 U(j,k + 1) = s * (U(j + 1, k) + U(j - 1, k)) + (1 - 2*s) * U(j,k);
end
 U(1,k + 1) = U(2, k + 1);
 U(J + 1,k + 1) = U(J, k + 1);
end
size(x)
%1 row and 101 columns
size(t)
%1 row and 400001 columns
size(U)
%101 rows and 400001 columns
%1D Neumann Diffusion In 2D Space
hold on
```

```
plot(x,U(:,1));
plot(x,U(:,11));
plot(x,U(:,101));
plot(x,U(:,1001));
plot(x,U(:,K));
hold off
xticks([0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1]);
xlabel('x');
yticks([1 2 3 4 5]);
ylabel('U');
legend('t = 0','t = 2.5 X 10^-4', 't = 2.5 X 10^-3', 't = 2.5 X 10^-2','t =
 9.9975 X 10^-2');
```

```
function f = initial_triangle(x)
f = zeros(size(x));
idx = (0.5 < x);
f(idx)=1-x;
idx = (x<0.5);
f(idx)=x;
idx = (x==0.5);
f(idx)=0.5;
```

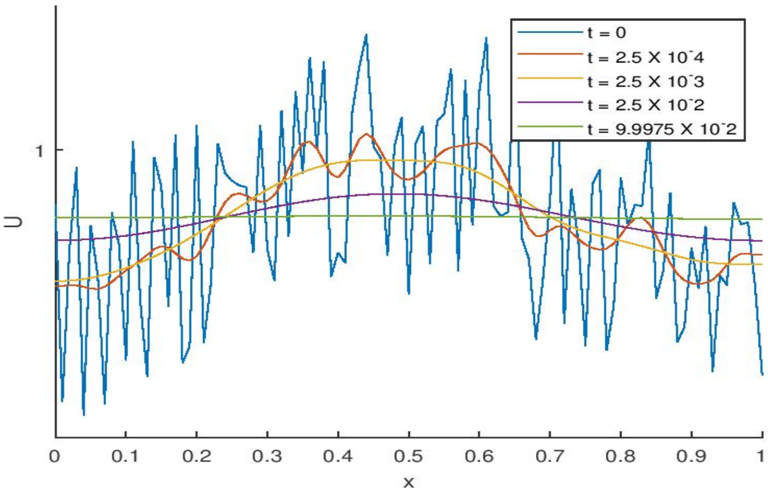


Figure 8: One Dimensional Diffusion With the Neumann Boundary Condition and Triangular Initial Data Over Snapshots of t Time as an Application for Denoising.

```
L = 1;
T = 0.1;
dx = 0.01;
dt = 0.25 * dx * dx;
```

```
s = (dt/(dx * dx));
J = L/dx;
K = T/dt;
x = 0:dx:L;
t = 0:dt:T;
U = ones(J + 1, K + 1);
for j = 1:J + 1
 U(j,1) = initial_triangle_noise(x(j));
end
for k = 1:K
for j = 2:J
 U(j,k + 1) = s * (U(j + 1, k) + U(j - 1, k)) + (1 - 2*s) * U(j,k);
end
 U(1,k + 1) = U(2, k + 1);
 U(J + 1,k + 1) = U(J, k + 1);
end
size(x)
%1 row and 101 columns
size(t)
%1 row and 400001 columns
size(U)
%101 rows and 400001 columns
%1D Neumann Diffusion In 2D Space
hold on
plot(x,U(:,1));
plot(x,U(:,11));
plot(x,U(:,101));
plot(x,U(:,1001));
plot(x,U(:,K));
hold off
xticks([0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1]);
xlabel('x');
yticks([1 2 3 4 5]);
ylabel('U');
legend('t = 0','t = 2.5 X 10^-4', 't = 2.5 X 10^-3', 't = 2.5 X 10^-2','t =
 9.9975 X 10^-2');
```

```
L = 1;
T = 0.1;
dx = 0.01;
dt = 0.25 * dx * dx;
s = (dt/(dx * dx));
J = L/dx;
K = T/dt;
x = 0:dx:L;
t = 0:dt:T;
U = ones(J + 1, K + 1);
for j = 1:J + 1
U(j,1) = initial_triangle_noise(x(j));
L = 1;
T = 0.1;
dx = 0.01;
dt = 0.25 * dx * dx;
s = (dt/(dx * dx));
J = L/dx;
K = T/dt;
x = 0:dx:L;
t = 0:dt:T;
U = ones(J + 1, K + 1);
for j = 1:J + 1
 U(j,1) = initial_triangle_noise(x(j));
end
for k = 1:K
for j = 2:J
 U(j,k + 1) = s * (U(j + 1, k) + U(j - 1, k)) + (1 - 2*s) * U(j,k);
end
 U(1,k + 1) = U(2, k + 1);
 U(J + 1,k + 1) = U(J, k + 1);
end
size(x)
%1 row and 101 columns
size(t)
%1 row and 400001 columns
size(U)
%101 rows and 400001 columns
%1D Neumann Diffusion In 2D Space
hold on
plot(x,U(:,1));
plot(x,U(:,11));
plot(x,U(:,101));
plot(x,U(:,1001));
plot(x,U(:,K));
hold off
xticks([0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1]);
xlabel('x');
yticks([1 2 3 4 5]);
ylabel('U');
legend('t = 0','t = 2.5 X 10^-4', 't = 2.5 X 10^-3', 't = 2.5 X 10^-2','t = 9.9975 X 10^-2');
```

```
function f = initial_triangle_noise(x)
f = zeros(size(x));
idx = (0.5 < x);
f(idx) = 1 - x + rand(1);
idx = (x < 0.5);
f(idx) = x + rand(1);
idx = (x == 0.5);
f(idx) = 0.5;
```

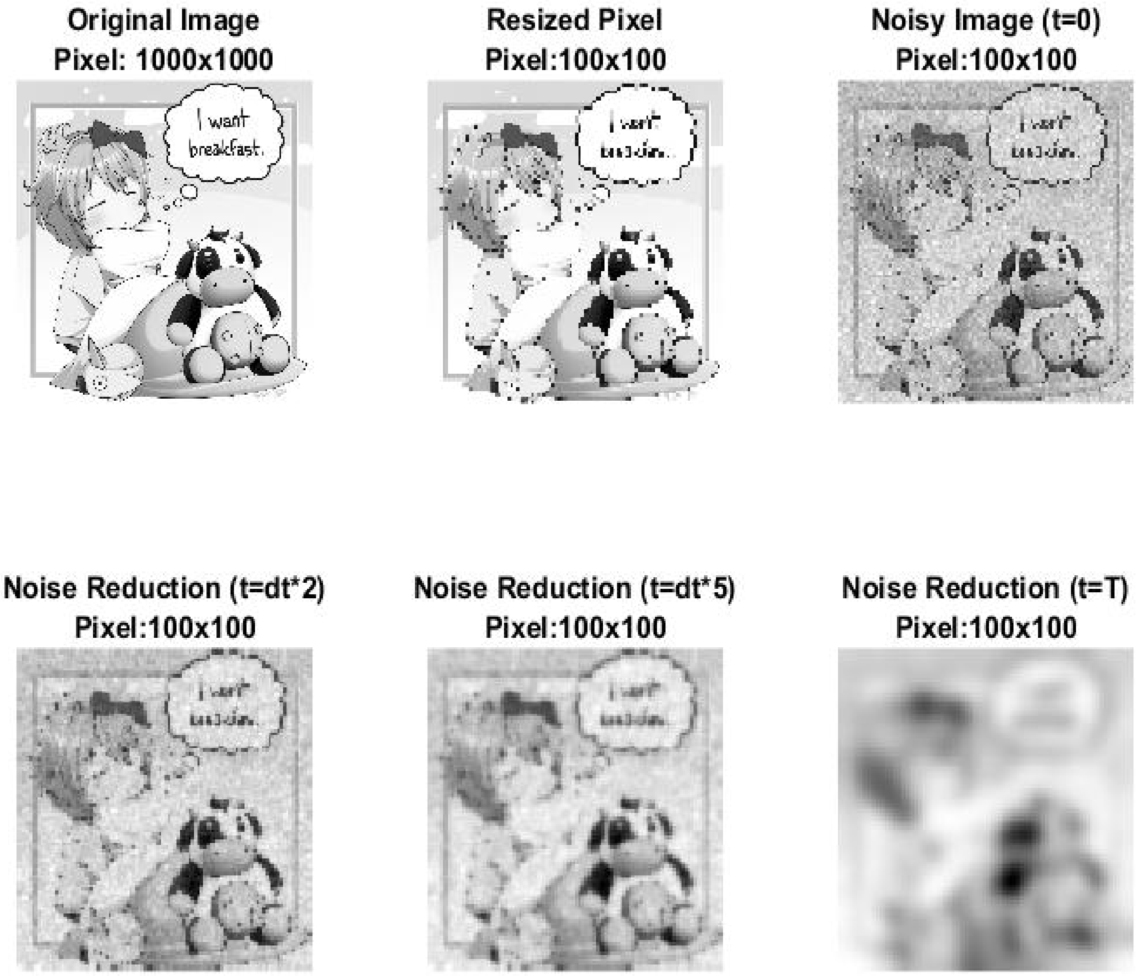


Figure 9: Two Dimensional Diffusion of A Selected Image Over t Snapshots of Time as an Application of Image Denoising. Image had to be dramatically resized from pixel size 1000 X 1000 to 100 X 100, due to memory issues appearing while executing the code. The complete noise reduced image at time $t = T$ still requires edge detection work along the varying pixels, explaining the extreme blurriness and smoothness of the image.

```
Image_25 = imread('C:\Users\Brand\OneDrive\Desktop\ResearchInMathematics\Images\Image_25.jpg');
Image_25 = rgb2gray(Image_25);
Image_25=double(Image_25);
L = 1;
T = 0.001;
dx = 0.01;
dy = 0.01; %JP
dt = 0.25 * dx * dx *0.5; %JP
s = (dt/(dx * dx));
J1 = int32(L/dx)-1;
J2 = int32(L/dy)-1; %JP
K = int32(T/dt);
x = 0:dx:L;
y = 0:dy:L; %JP
t = 0:dt:T;
U = ones(J1 + 1, J2 +1, K + 1); %JP

% initial data for 1d
%for j = 1:J + 1
```

```
% U(j,1) = %initial_triangle_noise(x(j));
%end

%initial data for 2d JP

%%edited
% Image_25 = imread('Image_25.jpg');
% Image_25 = rgb2gray(Image_25);
% %Image_25=double(Image_25);
% size(Image_25);
% sigma = 90;
% Noise = normrnd(0,sigma,[1000,1000]);
% Image_25_Noise=Image_25+Noise;

[m,n] = size(Image_25);
for i = 1:10:m
 for j = 1:10:n
    resized_Image_25(int32(floor(i/10 +1)),int32(floor(j/10 +1)))=Image_25(i,j);
  end
end

sigma = 40;
Noise = normrnd(0,sigma,[100,100]);
resized_Image_25_Noise1=resized_Image_25+Noise;
resized_Image_25_Noise2=(resized_Image_25)+(resized_Image_25_Noise1);
resized_Image_25_Noise2=double(resized_Image_25_Noise2);
%resized_Image_25_Noise2=padarray(resized_Image_25_Noise2,[0 0]);

initial_data = resized_Image_25_Noise2;

% v=Image_25_Noise;
% [m,n] = size(Image_25_Noise);
% for i = 1:10:m
%  for j = 1:10:n
%     initial_data(int32(floor(i/10 +1)),int32(floor(j/10 +1)))=Image_25_Noise(i,j);
%   end
% end
%
%initial_data =  %reshape(new_image,100,100);

[M, N] = size(initial_data);
U(:,:, 1) = initial_data;

for k = 1:K

for j1 = 2:J1
    for j2 = 2:J2
    U(j1,j2,k + 1) = s * (U(j1 + 1, j2, k) + U(j1 - 1, j2,k)) + (1 - 2*s) * U(j1, j2,k) + s * ...
        (U(j1, j2+1, k) + U(j1, j2-1,k)) + (1 - 2*s) * U(j1, j2,k);
    end
end

 U(1,:, k + 1) = U(2, :, k + 1);
 U(:, 1, k+1) = U(:, 2, k+1);
 U(J1 + 1,:, k + 1) = U(J1, :, k + 1);
 U(:, J2+1, k+1)=U(:, J2, k+1);
end
size(x)
%1 row and 101 columns
size(y)
%1 row and 101 columns
size(t)
%1 row and 400001 columns
size(U)
%101 rows and 400001 columns
```

```
%1D Neumann Diffusion In 2D Space

subplot(2,3,1)
imshow(Image_25,[]);
title({'Original Image','Pixel: 1000x1000'},'FontSize', 7)

subplot(2,3,2)
imshow(initial_data,[]);
title({'Resized Pixel', 'Pixel:100x100'},'FontSize', 7)

subplot(2,3,3)
u1 = U(:,:,1);
U1 = reshape(u1, M,N);
imshow(U1,[]);
title({'Noisy Image (t=0)','Pixel:100x100'},'FontSize', 7)

subplot(2,3,4)
u2 = U(:,:,3);
U2 = reshape(u2, M,N);
imshow(U2,[]);
title({'Noise Reduction (t=dt*2)','Pixel:100x100'},'FontSize', 7)

subplot(2,3,5)
u3 = U(:,:,6);
U3 = reshape(u3, M,N);
imshow(U3,[]);
title({'Noise Reduction (t=dt*5)','Pixel:100x100'},'FontSize', 7)

subplot(2,3,6)
u4 = U(:,:,K);
U4 = reshape(u4, M,N);
imshow(U4,[]);
title({'Noise Reduction (t=T)','Pixel:100x100'},'FontSize', 7)
```

Project Diary

1. September 16 meeting
    a. Regular meeting : Mondays 5:00pm
    b. Reading materials:
        i. R1: An image processing tour of college mathematics by Y. V. Galperin (Chap1 ~ Chap 7 (hopefully))
        ii. R2: PDEs by W. A. Strauss (Chap1 and Chap2 (partly))
        iii. Intro. MATLAB pdf file (only W1 and W2)
    c. MATLAB
2. W1: September 20
    a. Euler's method
    b. MATLAB
    c. To do:
        i. R1 - Chapter 1. Read carefully including examples. Do MATLAB exercises. Put some exercise results into a folder 'W1'
        ***UNABLE TO DO CH.1.3 and CH.1.4: PROBLEM 2b); IN SUMMARY, TOO MANY THINGS WENT WRONG WITH THE INSTALLATION OF THE TOOLBOXES AND IMPORTING THE IMAGES ONTO THE MATLAB ON REGULAR BROWSER IN THE VDI INTERFACE!**
        [*Okay! --Jeungeun*]
        ii. Intro. MATLAB pdf file - p.4 ~ p.34. Just try examples.
        iii. R2 - Read 1.1
3. W2: September 27
    a. Review W1
    b. To do:
        i. R1 - 1.2 - Try Q5 again. Let me know if you have any further question.
        ii. R1 - Chapter 2: MATLAB exercises. Please out some exercise results into W2 folder
        iii. Intro MATLAB pdf file - p.35 ~ end. Try examples (not exercises), and read carefully since some sections do not have examples.
        iv. R2 - Read 1.2. Please follow the examples very carefully.
4. W3: October 4
    a. Review W2 - (J. Park: Check Section 2.6 - Q2)
    b. To do:
        i. R1 - Chapter 3
        ii. R2 - Read 1.3

5. W4: October 15
    a.
    b. To do:
        i. R1 - Read Chap 3 - Exercise (Jeungeun)
        ii. R2 - Read 1.4 and 1.5

6. W5: October 18 (This upcoming Monday? Also, the 22nd is my 22nd birthday (22 on 22!)! - Brandon Fox)

   a. Chap3, Pseudo-code
   b. To do:
      i. R1 - Chap 4
      ii. R2 - Read 2.3 and 2.4

7. W6: October 25
   a. Review: Brandon’s email question. Pseudo code for Euler method
   b. To do:
      i. R1 - Sections 5.1-5.2
      ii. R2 - Section 8.1. Please go over Example 1 and try to digest it so we can use it for our matlab code.

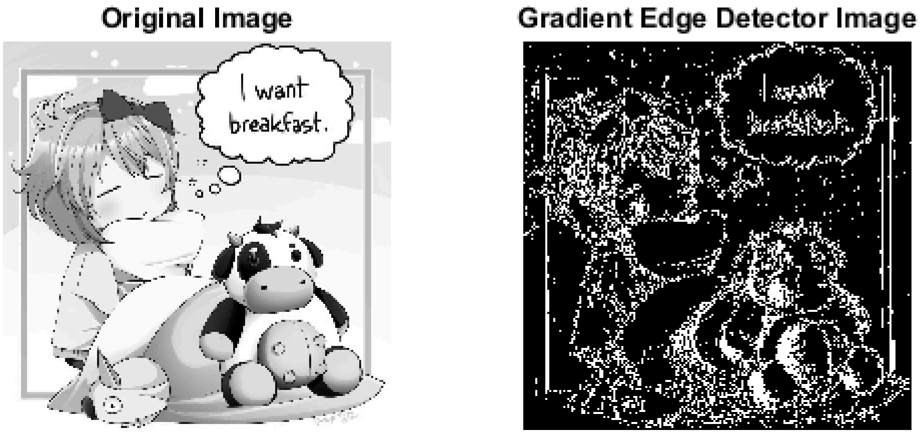
8. W7: Nov 1
   a. Review
   b. To do:
      i. R1 - Sections 5.3
      ii. R2 - Chap 8
      iii. Tomorrow: (Computational methods)MATLAB code: Euler method for y’=y, y(0)=1 .. Pseudo code.

9. W8: Nov 8
   a. Review
   b. To do:
      i. R1 - take a break for this week :)
      ii. R2 - Chap 8.2 - Boundary value problems

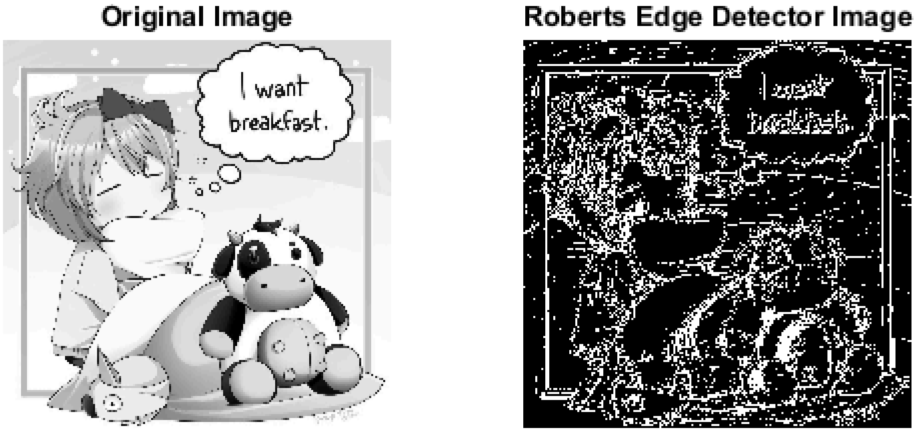
10. W9 - W16:
    a. Review
    b. To do:
       i. R2 - Chap 8.2 - Boundary value problems - finite difference methods
       ii. Read reading material on Perona-Malik equation
       iii. Solve diffusion equations with MATLAB:
          Euler method to solve y’=f(t,y)
          1D Diffusion equation with Dirichlet or Neumann boundary condition
          2D Diffusion equation with Neumann boundary condition
          Add noise in an initial datum
          Reduce noise in an initial datum as time changes
       iv. Write a result in LaTex

Example of weekly work by Brandon Fox: Image processing textbook Practice problem in Section 5.3

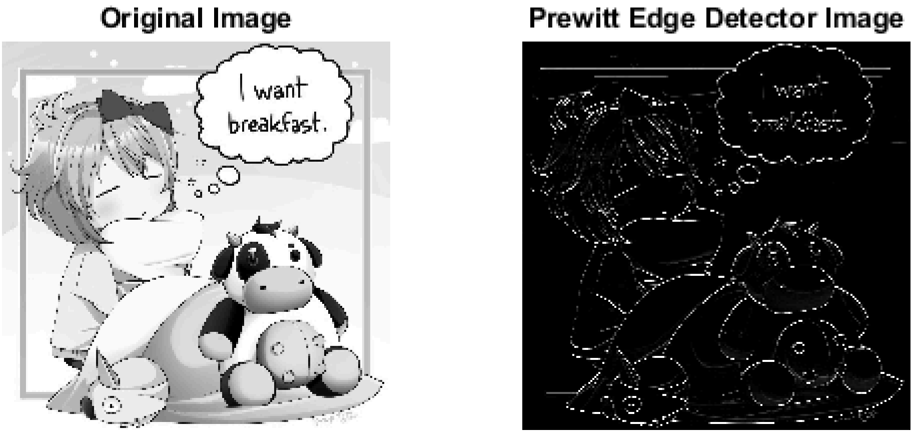
```
%1
Image_25 = imread('C:\Users\Brand\OneDrive\Desktop\ResearchInMathematics
\Images\Image_25.jpg');
Image_25 = rgb2gray(Image_25);
imshow(Image_25);

%2 and 3

%Gradient edge dector
[M, N]=size(Image_25);
G=zeros(M,N);
Gx=G; Gy=G;
for m=1:M-1
for n=1:N-1
Gx(m,n)=Image_25(m,n+1)-Image_25(m,n);
Gy(m,n)=Image_25(m+1,n)-Image_25(m,n);
G(m,n)=(sqrt(Gx(m,n))) & (2+Gy(m,n)) & (2);
end
end
subplot(1,2,1); imshow(Image_25); title('Original Image');
subplot(1,2,2); imshow(abs(G),[]); title('Gradient Edge Detector Image');
```



```
%Roberts edge detector
[M, N]=size(Image_25);
G=zeros(M,N);
G_1=G; G_2=G;
for m=1:M-1
for n=1:N-1
G_1(m,n)=Image_25(m+1,n+1)-Image_25(m,n);
G_2(m,n)=Image_25(m+1,n)-Image_25(m,n+1);
G(m,n)=(sqrt(G_1(m,n)) & (2+G_2(m,n)) & (2));
end
end
subplot(1,2,1); imshow(Image_25); title('Original Image');
subplot(1,2,2); imshow(abs(G),[]); title('Roberts Edge Detector Image');
```



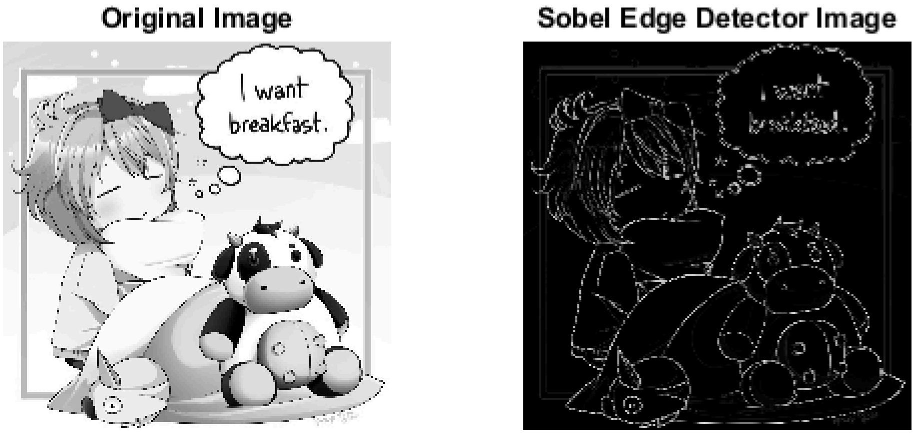
```
%Prewitt edge detector
f1 = (double(Image_25));
f2 = (double(Image_25));

h = fspecial('Prewitt');
P=uint8(round(filter2(h,Image_25)));

subplot(1,2,1); imshow(Image_25); title('Original Image');
subplot(1,2,2); imshow(P); title('Prewitt Edge Detector Image');
```



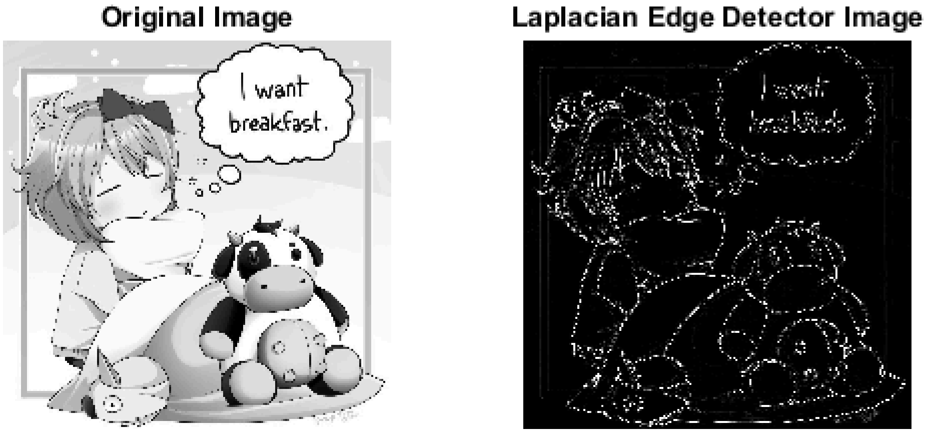
Original Image

Prewitt Edge Detector Image

```
%Sobel edge detector
I = double(Image_25);
for i=1:size(I,1)-2
    for j=1:size(I,2)-2
        G_1=((2*I(i+2,j+1)+I(i+2,j)+I(i+2,j+2))-(2*I(i,j+1)+I(i,j)+I(i,j+2)));
        G_2=((2*I(i+1,j+2)+I(i,j+2)+I(i+2,j+2))-(2*I(i+1,j)+I(i,j)+I(i+2,j)));
        G(i,j)=sqrt(G_1.^2+G_2.^2);
    end
end
subplot(1,2,1); imshow(Image_25); title('Original Image');
subplot(1,2,2); imshow(abs(G),[]); title('Sobel Edge Detector Image');
```



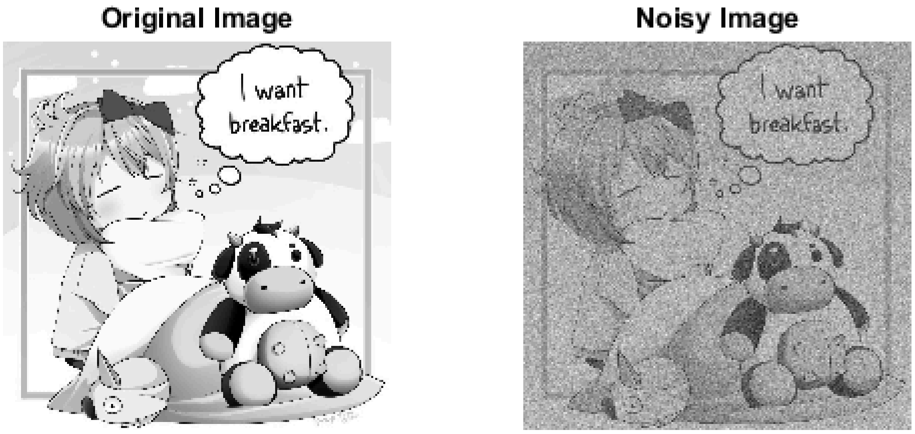
```
%Laplacian edge detector
L=[-1 -1 -1;-1 8 -1; -1 -1 -1];
G=uint8(filter2(L,Image_25,'same'));
subplot(1,2,1); imshow(Image_25); title('Original Image');
subplot(1,2,2); imshow(abs(G),[]); title('Laplacian Edge Detector Image');
```



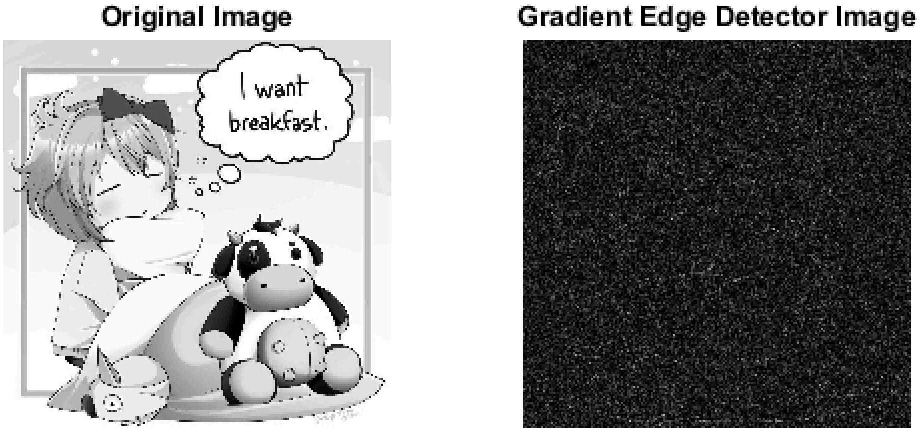
Original Image

Laplacian Edge Detector Image

```
%5
size(Image_25);
Image_25=double(Image_25);
sigma = 40;
Noise = normrnd(0,sigma,[1000,1000]);
Image_25_Noise=Image_25+Noise;

imshow(Image_25_Noise,[]); title('Noisy Image');
```



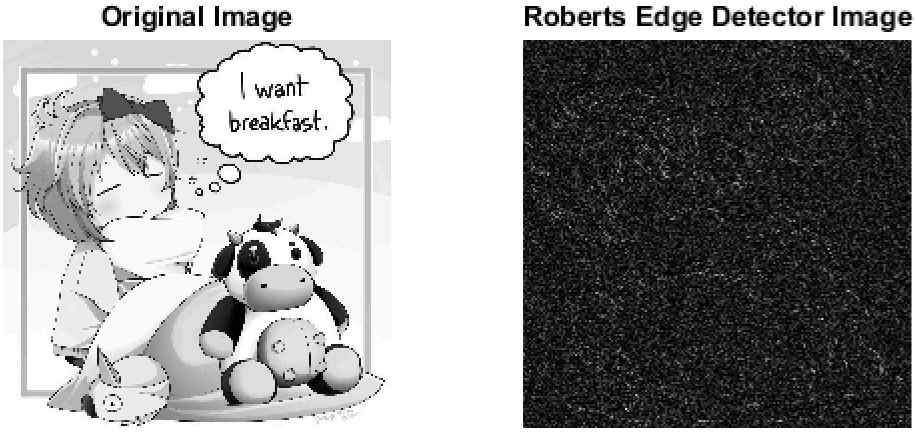
```
%Gradient edge dector
size(Image_25);
Image_25=double(Image_25);
sigma = 40;
Noise = normrnd(0,sigma,[1000,1000]);
Image_25_Noise=Image_25+Noise;
[M, N]=size(Image_25_Noise);
G=zeros(M,N);
Gx=G; Gy=G;
for m=1:M-1
for n=1:N-1
Gx(m,n)=Image_25_Noise(m,n+1)-Image_25_Noise(m,n);
Gy(m,n)=Image_25_Noise(m+1,n)-Image_25_Noise(m,n);
G(m,n)=(sqrt(Gx(m,n))) + (2+Gy(m,n)) + (2);
end
end
imshow(abs(G),[]); title('Gradient Edge Detector Image');
```



Original Image

Gradient Edge Detector Image

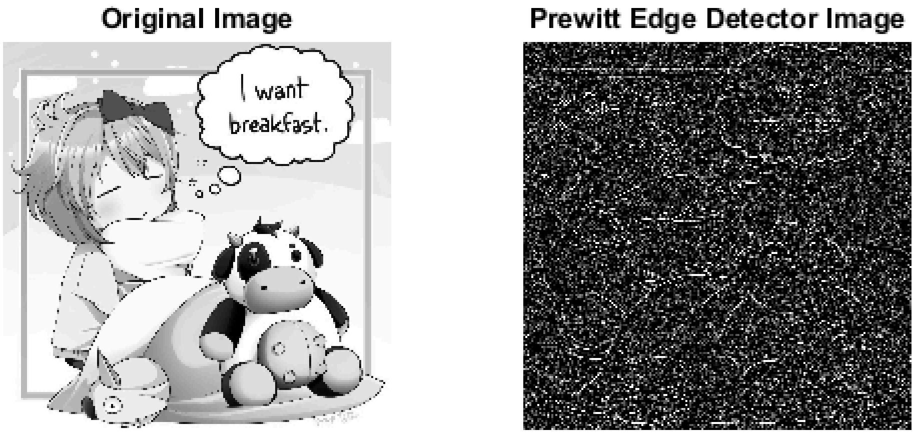
```
%Roberts edge detector
size(Image_25);
Image_25=double(Image_25);
sigma = 40;
Noise = normrnd(0,sigma,[1000,1000]);
Image_25_Noise=Image_25+Noise;
[M, N]=size(Image_25_Noise);
G=zeros(M,N);
G_1=G; G_2=G;
for m=1:M-1
for n=1:N-1
G_1(m,n)=Image_25_Noise(m+1,n+1)-Image_25_Noise(m,n);
G_2(m,n)=Image_25_Noise(m+1,n)-Image_25_Noise(m,n+1);
G(m,n)=(sqrt(G_1(m,n)) + (2+G_2(m,n)) + (2));
end
end
imshow(abs(G),[]); title('Roberts Edge Detector Image');
```



Original Image

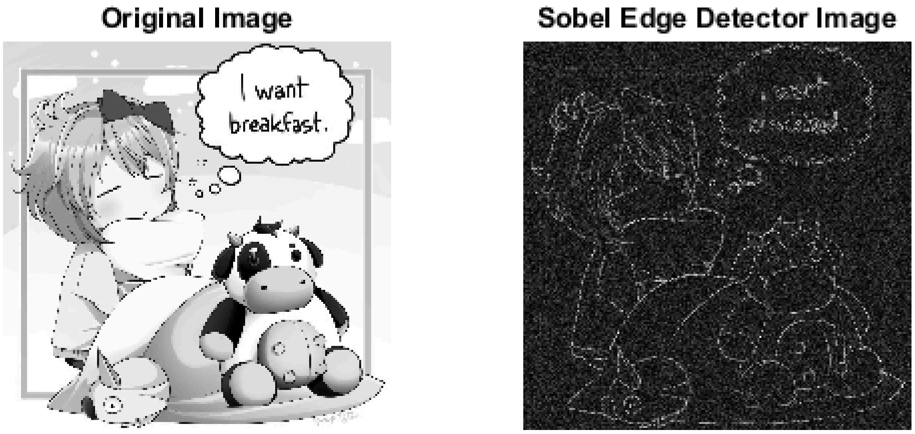
Roberts Edge Detector Image

```
%Prewitt edge detector
size(Image_25);
Image_25=double(Image_25);
sigma = 40;
Noise = normrnd(0,sigma,[1000,1000]);
Image_25_Noise=Image_25+Noise;

f1 = (double(Image_25_Noise));
f2 = (double(Image_25_Noise));

h = fspecial('Prewitt');
P=uint8(round(filter2(h,Image_25_Noise)));

imshow(P); title('Prewitt Edge Detector Image');
```


Original Image

Prewitt Edge Detector Image

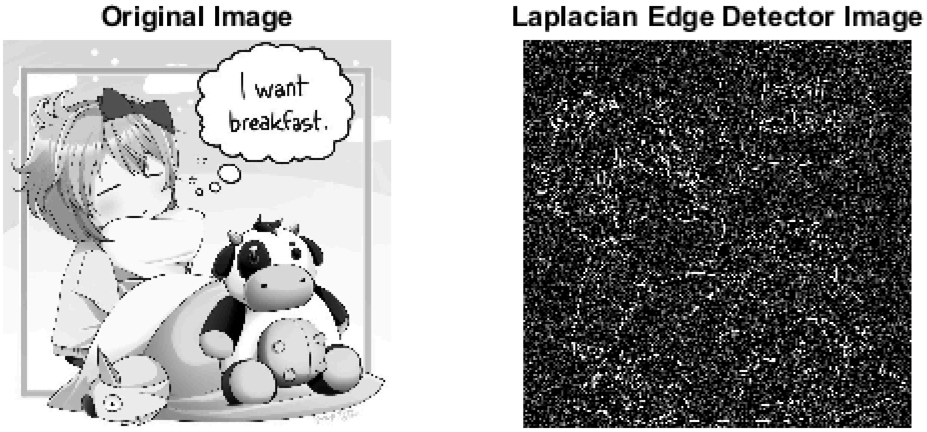
```
%Sobel edge detector
size(Image_25);
Image_25=double(Image_25);
sigma = 40;
Noise = normrnd(0,sigma,[1000,1000]);
Image_25_Noise=Image_25+Noise;
for i=1:size(Image_25_Noise,1)-2
    for j=1:size(Image_25_Noise,2)-2
        G_1=((2*Image_25_Noise(i+2,j+1)+Image_25_Noise(i+2,j)+Image_25_Noise(i+2,j+2))-(2*Image_25_Noise(i,j+1)+Image_25_Noise(i,j)+Image_25_Noise(i,j+2)));
        G_2=((2*Image_25_Noise(i+1,j+2)+Image_25_Noise(i,j+2)+Image_25_Noise(i+2,j+2))-(2*Image_25_Noise(i+1,j)+Image_25_Noise(i,j)+Image_25_Noise(i+2,j)));
        G(i,j)=sqrt(G_1.^2+G_2.^2);
    end
end
imshow(abs(G),[]); title('Sobel Edge Detector Image');
```



Original Image

Sobel Edge Detector Image

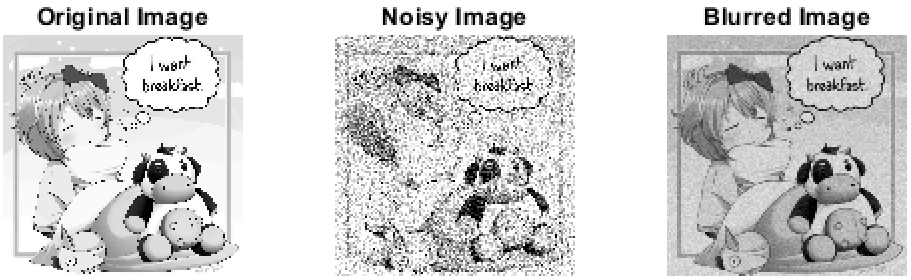
```
%Laplacian edge detector
size(Image_25);
Image_25=double(Image_25);
sigma = 10;
Noise = normrnd(0,sigma,[1000,1000]);
Image_25_Noise=Image_25+Noise;
L=[-1 -1 -1;-1 8 -1; -1 -1 -1];
G=uint8(filter2(L,Image_25_Noise,'same'));
imshow(abs(G),[]); title('Laplacian Edge Detector Image');
```



```
%7
Image_25=double(Image_25);
Image_25 = Image_25 - min(Image_25(:));
Image_25 = Image_25 / max(Image_25(:));
noise_per=2;
delta = (noise_per*std(Image_25(:)))^2
Image_25_Noise1=normrnd(0,delta,[1000 1000]);
Image_25_Noise2=(Image_25)+(Image_25_Noise1);
Image_25_Noise2=double(Image_25_Noise2);

sigma = 2.06;
Box_Size=3;
[x,y] = meshgrid(-Box_Size:Box_Size,-Box_Size:Box_Size);
M = size(x,1)-1;
N = size(y,1)-1;

Exp = -(x.^2+y.^2)/(2*sigma.^2);
Filter = ((1)/(2*pi*sigma.^2))*exp(Exp);
```

```
Blurred_Image=size(Image_25_Noise2);
Image_25_Noise2 = padarray(Image_25_Noise2,[0 0]);

for i=2:size(Image_25_Noise2,1)-M
    for j=2:size(Image_25_Noise2,2)-N
        T= Image_25_Noise2(i:i+M,j:j+M).*Filter;
        Blurred_Image(i,j)=sum(T(:));
    end
end
subplot(1,3,1); imshow(Image_25); title('Original Image');
subplot(1,3,2); imshow(Image_25_Noise2); title('Noisy Image');
subplot(1,3,3); imshow(Blurred_Image); title('Blurred Image');
```

```
delta =

    0.2273
```



```
%Gradient edge dector
```

```
Image_25=double(Image_25);
Image_25 = Image_25 - min(Image_25(:));
Image_25 = Image_25 / max(Image_25(:));
noise_per=2;
delta = (noise_per*std(Image_25(:)))^2
Image_25_Noise1=normrnd(0,delta,[1000 1000]);
Image_25_Noise2=(Image_25)+(Image_25_Noise1);
Image_25_Noise2=double(Image_25_Noise2);

Grad_Grad = imgradient(Image_25_Noise2);

sigma = 2;
S = imgaussfilt(Image_25_Noise2, sigma);
G = edge(S, 'Gradient');

subplot(1,3,1); imshow(Image_25); title('Original Image');
subplot(1,3,2); imshow(Image_25_Noise2); title('Noisy Image');
subplot(1,3,3); imshow(abs(G),[]); title('Gradient Edge Detector Image');
```

```
delta =

    0.2273
```

```
Error using edge>parse_inputs (line 518)
Invalid input string or character vector: 'Gradient'.

Error in edge (line 213)
[a,method,thresh,sigma,thinning,H,kx,ky] = parse_inputs(args{:});

Error in Ch_5_3 (line 198)
G = edge(S, 'Gradient');

Error in evalmxdom>instrumentAndRun (line 114)
text = evalc(evalstr);

Error in evalmxdom (line 21)
[data,text,laste] =
 instrumentAndRun(file,cellBoundaries,imageDir,imagePrefix,options);

Error in publish

Error in connector.internal.fevalMatlab

Error in connector.internal.fevalJSON

Error in internal.matlab.publish.captureFigures (line 14)
drawnow;

Error in internal.matlab.publish.PublishFigures/leavingCell (line 78)
            newFigures = internal.matlab.publish.captureFigures;

Error in snapnow>leavingCell (line 208)
```

```
            newFiles = data.plugins(iPlugins).instance.leavingCell(iCell);

Error in snapnow (line 144)
                        data = leavingCell(iCell(k), data, doCapture(k));

Error in Ch_5_3 (line 152)
Image_25=double(Image_25);

%Roberts edge detector
Image_25=double(Image_25);
Image_25 = Image_25 - min(Image_25(:));
Image_25 = Image_25 / max(Image_25(:));
noise_per=2;
delta = (noise_per*std(Image_25(:)))^2
Image_25_Noise1=normrnd(0,delta,[1000 1000]);
Image_25_Noise2=(Image_25)+(Image_25_Noise1);
Image_25_Noise2=double(Image_25_Noise2);

Roberts_Grad = imgradient(Image_25_Noise2);

sigma = 2;
S = imgaussfilt(Image_25_Noise2, sigma);
G = edge(S, 'Roberts');

subplot(1,3,1); imshow(Image_25); title('Original Image');
subplot(1,3,2); imshow(Image_25_Noise2); title('Noisy Image');
subplot(1,3,3); imshow(abs(G),[]); title('Roberts Edge Detector Image');

%Prewitt edge detector
Image_25=double(Image_25);
Image_25 = Image_25 - min(Image_25(:));
Image_25 = Image_25 / max(Image_25(:));
noise_per=2;
delta = (noise_per*std(Image_25(:)))^2
Image_25_Noise1=normrnd(0,delta,[1000 1000]);
Image_25_Noise2=(Image_25)+(Image_25_Noise1);
Image_25_Noise2=double(Image_25_Noise2);

Prewitt_Grad = imgradient(Image_25_Noise2);

sigma = 2;
S = imgaussfilt(Image_25_Noise2, sigma);
G = edge(S, 'Prewitt');

subplot(1,3,1); imshow(Image_25); title('Original Image');
subplot(1,3,2); imshow(Image_25_Noise2); title('Noisy Image');
subplot(1,3,3); imshow(abs(G),[]); title('Prewitt Edge Detector Image');

%Sobel edge detector
Image_25=double(Image_25);
Image_25 = Image_25 - min(Image_25(:));
Image_25 = Image_25 / max(Image_25(:));
noise_per=2;
delta = (noise_per*std(Image_25(:)))^2
```

```
Image_25_Noise1=normrnd(0,delta,[1000 1000]);
Image_25_Noise2=(Image_25)+(Image_25_Noise1);
Image_25_Noise2=double(Image_25_Noise2);

Sobel_Grad = imgradient(Image_25_Noise2);
hy = -fspecial('sobel')
hx = hy';

sigma = 2;
S = imgaussfilt(Image_25_Noise2, sigma);
G = imgradient(S,'CentralDifference');

subplot(1,3,1); imshow(Image_25); title('Original Image');
subplot(1,3,2); imshow(Image_25_Noise2); title('Noisy Image');
subplot(1,3,3); imshow(abs(G),[]); title('Sobel Edge Detector Image');

%Laplacian edge detector
Image_25=double(Image_25);
Image_25 = Image_25 - min(Image_25(:));
Image_25 = Image_25 / max(Image_25(:));
noise_per=2;
delta = (noise_per*std(Image_25(:)))^2
Image_25_Noise1=normrnd(0,delta,[1000 1000]);
Image_25_Noise2=(Image_25)+(Image_25_Noise1);
Image_25_Noise2=double(Image_25_Noise2);

Laplacian_Grad = imgradient(Image_25_Noise2);

sigma = 2;
S = imgaussfilt(Image_25_Noise2, sigma);
G = edge(S,'log');

subplot(1,3,1); imshow(Image_25); title('Original Image');
subplot(1,3,2); imshow(Image_25_Noise2); title('Noisy Image');
subplot(1,3,3); imshow(abs(G),[]); title('Laplacian Edge Detector Image');

%8
Binary_Image_25 = Image_25 > 0.5;

subplot(1,2,1); imshow(Image_25); title("Original Image");
subplot(1,2,2); imshow(Binary_Image_25); title("Binary Image");

%10

%a
Binary_Image_25 = Image_25 > 0.5;
S_E =[0 0 0; 0 1 0; 0 0 0];
L1 = imdilate(Binary_Image_25,S_E);

subplot(1,2,1); imshow(Binary_Image_25); title('Binary Image');
subplot(1,2,2); imshow(L1); title('Linear Edge Dilated Image #1');

Binary_Image_25 = Image_25 > 0.5;
S_E =[0 0 0; 1 1 1; 0 0 0];
m=length(Binary_Image_25);
```

```
n=length(S_E);
N=max(m,n);
Binary_Image_25=[Binary_Image_25,zeros(3,N-m)];
S_E=[S_E,zeros(3,N-n)];
for n=1:N
    Y(n)=0;
    for i=1:N
        j=n-i+1;
        if(j<=0)
            j=N+j;
        end
        Y(n)=[Y(n)+Binary_Image_25(i)*S_E(j)];
    end
end
subplot(1,2,1); imshow(Binary_Image_25); title('Binary Image');
subplot(1,2,2); imshow(Y); title('Circular Edge Dilated Image #1');

%b
Binary_Image_25 = Image_25 > 0.5;
S_E =[0 0 0; 1 1 1; 0 0 0];
L2 = imdilate(Binary_Image_25,S_E);

subplot(1,2,1); imshow(Binary_Image_25); title('Binary Image');
subplot(1,2,2); imshow(L2); title('Linear Edge Dilated Image #2');

Binary_Image_25 = Image_25 > 0.5;
S_E =[0 0 0; 1 1 1; 0 0 0];
m=length(Binary_Image_25);
n=length(S_E);
N=max(m,n);
Binary_Image_25=[Binary_Image_25,zeros(3,N-m)];
S_E=[S_E,zeros(3,N-n)];
for n=1:N
    Y(n)=0;
    for i=1:N
        j=n-i+1;
        if(j<=0)
            j=N+j;
        end
        Y(n)=[Y(n)+Binary_Image_25(i)*S_E(j)];
    end
end
subplot(1,2,1); imshow(Binary_Image_25); title('Binary Image');
subplot(1,2,2); imshow(Y); title('Circular Edge Dilated Image #2');

%c
Binary_Image_25 = Image_25 > 0.5;
S_E =[0 1 0; 0 1 0; 0 1 0];
L3 = imdilate(Binary_Image_25,S_E);

subplot(1,2,1); imshow(Binary_Image_25); title('Binary Image');
subplot(1,2,2); imshow(L3); title('Linear Edge Dilated Image #3');

Binary_Image_25 = Image_25 > 0.5;
S_E =[0 1 0; 0 1 0; 0 1 0];
```

```
m=length(Binary_Image_25);
n=length(S_E);
N=max(m,n);
Binary_Image_25=[Binary_Image_25,zeros(3,N-m)];
S_E=[S_E,zeros(3,N-n)];
for n=1:N
    Y(n)=0;
    for i=1:N
        j=n-i+1;
        if(j<=0)
            j=N+j;
        end
        Y(n)=[Y(n)+Binary_Image_25(i)*S_E(j)];
    end
end
subplot(1,2,1); imshow(Binary_Image_25); title('Binary Image');
subplot(1,2,2); imshow(Y); title('Circular Edge Dilated Image #3');

%d
Binary_Image_25 = Image_25 > 0.5;
S_E =[0 1 0; 1 1 1; 0 1 0];
L4 = imdilate(Binary_Image_25,S_E);

subplot(1,2,1); imshow(Binary_Image_25); title('Binary Image');
subplot(1,2,2); imshow(L4); title('Linear Edge Dilated Image #4');

Binary_Image_25 = Image_25 > 0.5;
S_E =[0 1 0; 1 1 1; 0 1 0];
m=length(Binary_Image_25);
n=length(S_E);
N=max(m,n);
Binary_Image_25=[Binary_Image_25,zeros(3,N-m)];
S_E=[S_E,zeros(3,N-n)];
for n=1:N
    Y(n)=0;
    for i=1:N
        j=n-i+1;
        if(j<=0)
            j=N+j;
        end
        Y(n)=[Y(n)+Binary_Image_25(i)*S_E(j)];
    end
end
subplot(1,2,1); imshow(Binary_Image_25); title('Binary Image');
subplot(1,2,2); imshow(Y); title('Circular Edge Dilated Image #4');

%e
Binary_Image_25 = Image_25 > 0.5;
S_E = [1 0 1; 0 1 0; 1 0 1];
L5 = imdilate(Binary_Image_25,S_E);

subplot(1,2,1); imshow(Binary_Image_25); title('Binary Image');
subplot(1,2,2); imshow(L5); title('Linear Edge Dilated Image #5');

Binary_Image_25 = Image_25 > 0.5;
```

```
S_E = [1 0 1; 0 1 0; 1 0 1];
m=length(Binary_Image_25);
n=length(S_E);
N=max(m,n);
Binary_Image_25=[Binary_Image_25,zeros(3,N-m)];
S_E=[S_E,zeros(3,N-n)];
for n=1:N
    Y(n)=0;
    for i=1:N
        j=n-i+1;
        if(j<=0)
            j=N+j;
        end
        Y(n)=[Y(n)+Binary_Image_25(i)*S_E(j)];
    end
end
subplot(1,2,1); imshow(Binary_Image_25); title('Binary Image');
subplot(1,2,2); imshow(Y); title('Circular Edge Dilated Image #5');

%f
Binary_Image_25 = Image_25 > 0.5;
S_E = ones(3,3);
L6 = imdilate(Binary_Image_25,S_E);

subplot(1,2,1); imshow(Binary_Image_25); title('Binary Image');
subplot(1,2,2); imshow(L6); title('Linear Edge Dilated Image #6');

Binary_Image_25 = Image_25 > 0.5;
S_E = ones(3,3);
m=length(Binary_Image_25);
n=length(S_E);
N=max(m,n);
Binary_Image_25=[Binary_Image_25,zeros(3,N-m)];
S_E=[S_E,zeros(3,N-n)];
for n=1:N
    Y(n)=0;
    for i=1:N
        j=n-i+1;
        if(j<=0)
            j=N+j;
        end
        Y(n)=[Y(n)+Binary_Image_25(i)*S_E(j)];
    end
end
subplot(1,2,1); imshow(Binary_Image_25); title('Binary Image');
subplot(1,2,2); imshow(Y); title('Circular Edge Dilated Image #6');
```

Published with MATLAB® R2021b