Homework
Overview

Author: Anıl Karatay
anilkaratay@iyte.edu.tr

Task-1: Matrix Operations

Consider an $n \times n$ symmetric matrix. A symmetric matrix $A \in \mathbb{R}^{n \times n}$ is said to be positive definite if for any non-zero vector $x \in \mathbb{R}^n$, the quadratic form $x^T Ax$ is strictly positive, i.e., $x^T Ax > 0$ for all $x \neq 0$. The definiteness of the given matrix can be determined with principal minors method. For example,

$$A = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 10 & 0 \\ 0 & 0 & 18 \end{bmatrix} \tag{1}$$

$$D_1 = det \begin{bmatrix} 2 \end{bmatrix} > 0 \tag{2}$$

$$D_2 = det \begin{bmatrix} 2 & 1 \\ 1 & 10 \end{bmatrix} > 0 \tag{3}$$

$$D_3 = det \begin{bmatrix} 2 & 1 & 0 \\ 1 & 10 & 0 \\ 0 & 0 & 18 \end{bmatrix} > 0 \tag{4}$$

If all the leading principal minors have positive determinant, the matrix is positive definite.

- A is positive definite if and only if D_k > 0 for all leading principal minors.
- A is negative definite if and only if $(-1)^k D_k$ > 0 for all leading principal minors.
- A is positive semidefinite if and only if $D_k \geq 0$ for all leading principal minors.
- A is negative semidefinite if and only if $(-1)^k D_k \geq 0$ for all leading principal minors.

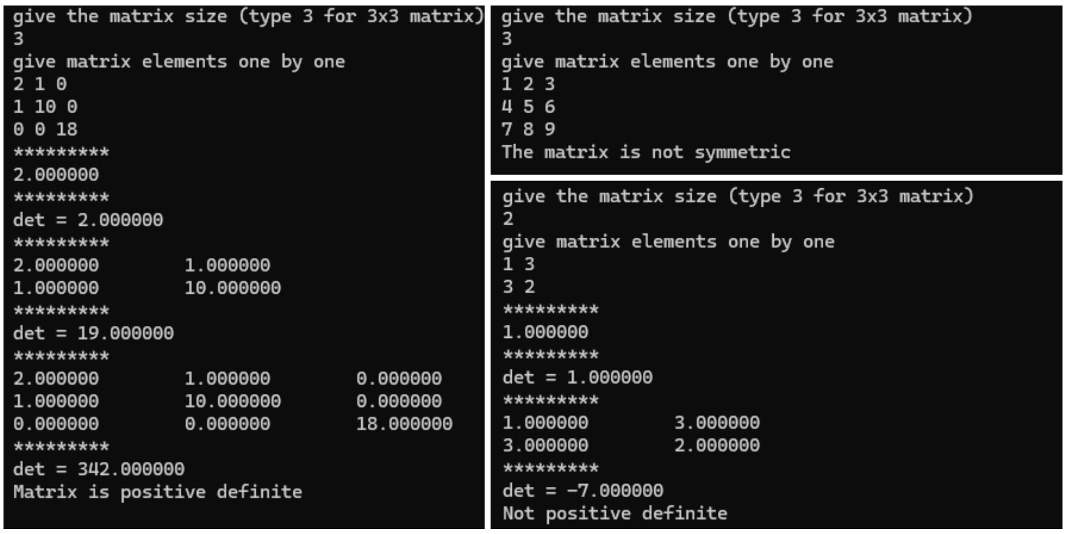
Quadratic forms involving positive definite matrices are always convex. This is useful in optimization. Such quadratic forms often arise as objective functions or regularizers in machine learning. Therefore, within the scope of this task, you are supposed to determine whether a given symmetric matrix is positive definite. The program to be written must meet the following specifications.

- Implement a C program that prints whether a given symmetric matrix is positive definite or not.
- Input matrix size should be set by the user and the matrix must be allocated dynamically.
- The program must work for input square matrix up to 3×3 size.
- The written determinant function must calculate the determinant of any $n \times n$ sized square matrix up to 3×3.
- The matrix must be defined in main() as a pointer double. Sample outputs are given in Figure 1.

- Do not forget to warning message if the matrix is not symmetric, e.g.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \tag{5}$$

- We also recommend that you check negative definiteness, positive semidefiniteness and negative semidefiniteness for study purposes.



```
give the matrix size (type 3 for 3x3 matrix)
3
give matrix elements one by one
2 1 0
1 10 0
0 0 18
*********
2.000000
*********
det = 2.000000
*********
2.000000        1.000000
1.000000        10.000000
*********
det = 19.000000
*********
2.000000        1.000000        0.000000
1.000000        10.000000       0.000000
0.000000        0.000000        18.000000
*********
det = 342.000000
Matrix is positive definite
```

```
give the matrix size (type 3 for 3x3 matrix)
3
give matrix elements one by one
1 2 3
4 5 6
7 8 9
The matrix is not symmetric
```

```
give the matrix size (type 3 for 3x3 matrix)
2
give matrix elements one by one
1 3
3 2
*********
1.000000
*********
det = 1.000000
*********
1.000000        3.000000
3.000000        2.000000
*********
det = -7.000000
Not positive definite
```

Figure 1: Sample output of Task-1

Task-2: Root Finding

The Newton's method is a numerical technique for root finding. It begins the search for a root with some initial guess. Then, it fits a tangent to the function through the guess and finds the root of the tangent line which becomes the next guess. This process continues until the difference between the guess and the root of the tangent line is small. Newton's method uses the derivative of the function to find the slope of the tangent line. It is based on solving the scalar equation f(x) = 0, where f is a continuous function. A tangent line is drawn using the known point (x_n, $f(x_n)$) on the graph of f(x). After the derivative of $f(x_n)$, $f'(x_n)$ from the tangent line is achieved, the equation is written as:

$$f(x_n) + (x_{n+1} - x_n)f'(x_n) = 0 \tag{6}$$

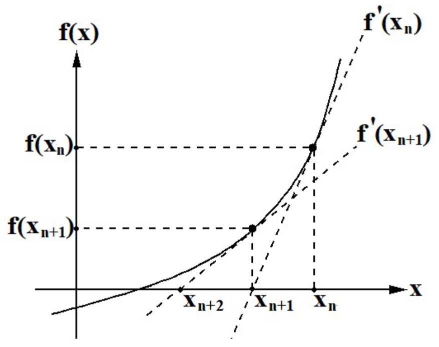


Figure 2: Geometrical representation of the Newton's method

In Figure 2, $f'(x_n)$ can be written as $f'(x_n) = f(x_n)/(x_n - x_{n+1})$ since it means the slope of the tangent line. Thus, x_{n+1} is assigned at the intersection point between the tangent line and the x-axis and can be written as:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{7}$$

The sequence of x_n values is obtained consequently, and this sequence becomes the linear interpolation of $f(x)$. x_n sequence produces an iterative solution and the point where the iteration converges, gives the root of $f(x)$. This method is used in many fields of engineering and science, as well as in electromagnetism. In this task, reconstruction of dielectric constant measurement data in an RF system will be addressed by Newton's method. We will first complete the iteration for the following function with Newton's method.

$$f(x_n) = tan(x_n) - Kx_n = 0 \tag{8}$$

where

$$K = \frac{tan\left(\frac{2\pi(d+L)}{\lambda_g}\right)}{2\pi\left(\frac{d}{\lambda_g}\right)} \tag{9}$$

Derivative of $f(x_n)$:

$$f'(x_n) = sec^2(x_n) - K \tag{10}$$

Our main aim is to find the value of dielectric constant, e_r. The value of e_r is found using the root of the function $f(x_n)$.

$$e_r = \frac{\left(\frac{a}{d}\right)^2\left(\frac{x_n}{\pi}\right)^2 + 1}{\left(\frac{2a}{\lambda_g}\right)^2 + 1} \tag{11}$$

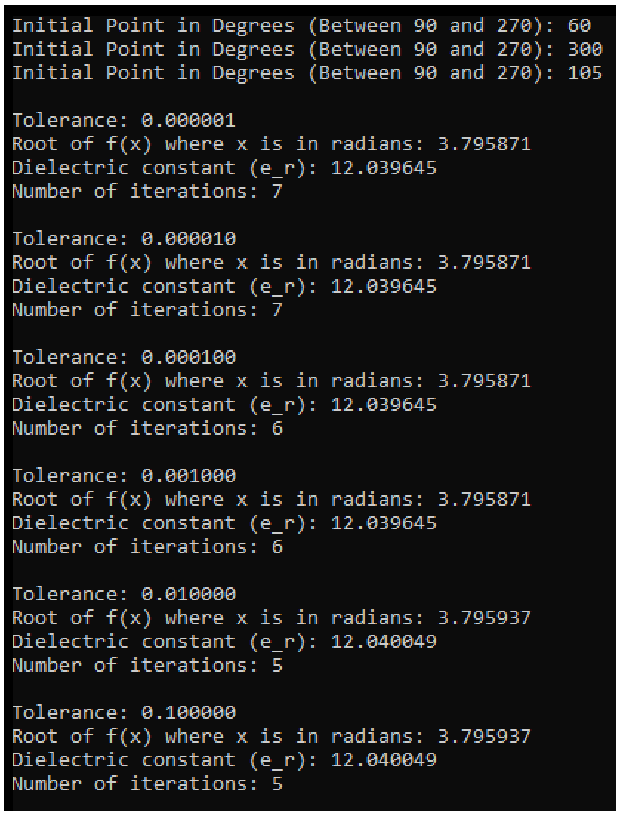
After the iteration is completed, the value of x_n can be replaced and the e_r value can then be calculated. In the above equations, some unknown parameters are determined by measurements. In this lab, we will assume that these values have been measured as follows:

$$d = 0.5,\ L = 1.4,\ \lambda_g = 3.6,\ a = 2.28 \tag{12}$$

- Define the necessary parameters as global variables.
- Declare f , f', K and e_r as separate functions.
- Your program should ask the user for the initial guess of x in main. Force the user to enter the initial guess in degrees, and the initial guess must be greater than 90 and no greater than 270.
- Write a function that converts the degree to radian to properly operate with tan() function. If you fail to write this function, you can get the initial guess from the user directly in radians, roughly between 1.6 and 4.7.
- The tolerance value to end the iteration must be generated in a separate function. This function should generate 6 different logarithmically increasing tolerance values from 10^{-6} to 10^{-1}, and return these values appropriately. You have been asked to write a function that returns a pointer double, considering that it should return more than one value. The arguments of the function must be left blank as in the prototype given below.

```
double *tolerance_value_generator(); //returns 6 different tolerance value
// with a single pointer: 10^-6, 10^-5, ..., 10^-1
```

- Write a function that calculates the root of f by using the Newton's method.
- Print the root in radians, e_r and number of iterations in main() in a single loop for six different tolerance values generated by *tolerance_value_generator().



```
Initial Point in Degrees (Between 90 and 270): 60
Initial Point in Degrees (Between 90 and 270): 300
Initial Point in Degrees (Between 90 and 270): 105

Tolerance: 0.000001
Root of f(x) where x is in radians: 3.795871
Dielectric constant (e_r): 12.039645
Number of iterations: 7

Tolerance: 0.000010
Root of f(x) where x is in radians: 3.795871
Dielectric constant (e_r): 12.039645
Number of iterations: 7

Tolerance: 0.000100
Root of f(x) where x is in radians: 3.795871
Dielectric constant (e_r): 12.039645
Number of iterations: 6

Tolerance: 0.001000
Root of f(x) where x is in radians: 3.795871
Dielectric constant (e_r): 12.039645
Number of iterations: 6

Tolerance: 0.010000
Root of f(x) where x is in radians: 3.795937
Dielectric constant (e_r): 12.040049
Number of iterations: 5

Tolerance: 0.100000
Root of f(x) where x is in radians: 3.795937
Dielectric constant (e_r): 12.040049
Number of iterations: 5
```

Figure 3: Sample output of Task-2

Task-3: Numerical Integration

Numerical integration is the approximate computation of an integral using numerical techniques. There are a wide range of methods available for numerical integration. Two of these are the rectangular and the trapezoidal methods. The rectangular method is one of the simplest integration techniques that approximates the integral of a function with a rectangle. It uses rectangles to approximate the area under the curve, see Figure 4. The width of the rectangle is determined by the integration range. The integration interval can be divided into n smaller intervals of equal length and n rectangles are used to approximate the integral. In a fixed integration range, the more rectangles are used, the more accurate the approach; each rectangle narrows, and the height of the rectangle better captures the values of the function in that range.

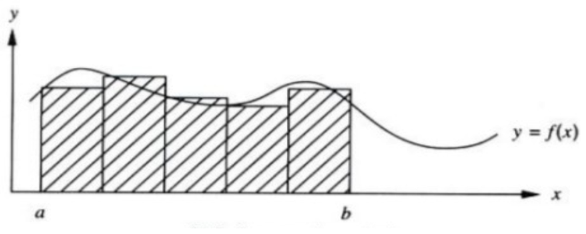


Figure 4: Left rectangular method

Trapezoidal method is another numerical techniques to approximate a definite integral. In this method, region under the graph is calculated by regarding it as a combination of many trapezoid panels. As number of panels increases, approximation error decreases. Region under the function f(x) between a and b (assuming no singularity exists in that range) approximated by the trapezoids and can be found by calculating the area of these trapezoids.

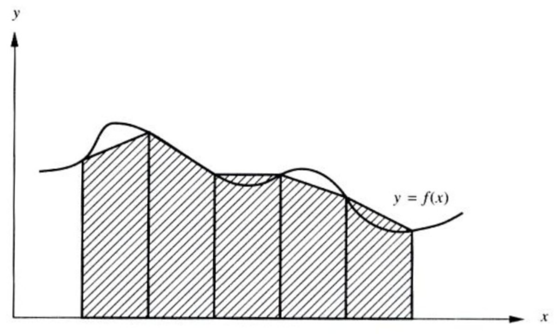


Figure 5: Trapezoidal method

In this task, integration of two different functions and their differences over a certain interval will be addressed. With the help of rectangular and trapezoidal panels, these integrals will be calculated and the performances will be compared. What is expected from the program you will write is to calculate the integrals of the functions given in Figure 6 using rectangular and trapezoidal methods for both functions between the given interval and calculate the corresponding area by finding the differences of these definite integrals. By repeating this process with different number of panels, we will observe which method is more accurate.

- Declare f_1 and f_2 as separate functions. You must follow the prototypes below.

```
double f1(double x);//returns the value of f1
double f2(double x);//returns the value of f2
```

- Write two functions which calculate the definite integral of any function, f, with left rectangular and trapezoidal methods.

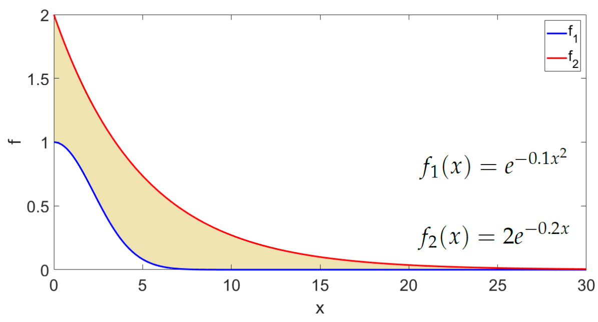
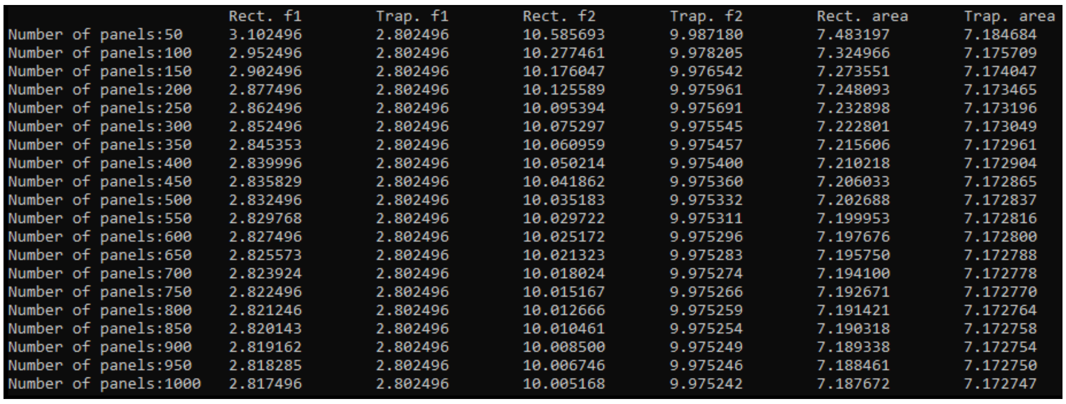


Figure 6: Gaussian curves with various parameters

- Write a function that generates the number of panels in a dynamic array and it returns a pointer integer. The number of panels should be in the range of the sample output; see Figure 7.

```c
int *number_of_panels_generator()
```

- Print the results in main() for each panel numbers. Using printf() anywhere other than main() is strictly forbidden.



```
                         Rect. f1         Trap. f1         Rect. f2         Trap. f2         Rect. area       Trap. area
Number of panels:50      3.102496         2.802496         10.585693        9.987180         7.483197         7.184684
Number of panels:100     2.952496         2.802496         10.277461        9.978205         7.324966         7.175709
Number of panels:150     2.902496         2.802496         10.176047        9.976542         7.273551         7.174047
Number of panels:200     2.877496         2.802496         10.125589        9.975961         7.248093         7.173465
Number of panels:250     2.862496         2.802496         10.095394        9.975691         7.232898         7.173196
Number of panels:300     2.852496         2.802496         10.075297        9.975545         7.222801         7.173049
Number of panels:350     2.845353         2.802496         10.060959        9.975457         7.215606         7.172961
Number of panels:400     2.839996         2.802496         10.050214        9.975400         7.210218         7.172904
Number of panels:450     2.835829         2.802496         10.041862        9.975360         7.206033         7.172865
Number of panels:500     2.832496         2.802496         10.035183        9.975332         7.202688         7.172837
Number of panels:550     2.829768         2.802496         10.029722        9.975311         7.199953         7.172816
Number of panels:600     2.827496         2.802496         10.025172        9.975296         7.197676         7.172800
Number of panels:650     2.825573         2.802496         10.021323        9.975283         7.195750         7.172788
Number of panels:700     2.823924         2.802496         10.018024        9.975274         7.194100         7.172778
Number of panels:750     2.822496         2.802496         10.015167        9.975266         7.192671         7.172770
Number of panels:800     2.821246         2.802496         10.012666        9.975259         7.191421         7.172764
Number of panels:850     2.820143         2.802496         10.010461        9.975254         7.190318         7.172758
Number of panels:900     2.819162         2.802496         10.008500        9.975249         7.189338         7.172754
Number of panels:950     2.818285         2.802496         10.006746        9.975246         7.188461         7.172750
Number of panels:1000    2.817496         2.802496         10.005168        9.975242         7.187672         7.172747
```

Figure 7: Sample output of Task-3