```
#Q 8A:
# f(x) = x ** 2 + 2 * x + 2
# f(x1) * f(x2) < 0
#ruth avivi 208981555
#eden dahan 318641222
from math import exp, sin



def f_tag(f, x):
    """
    x = c
    f'(x) = lim h->0 (f(c + h) - f(c))/h
    """
    h = 0.0000000001

    return (f(x + h) - f(x))/h



def biscetion(f, start, end, epsilon, max_iteration = 200):
    c = start
    i = 1
    while abs(f(c)) > epsilon and max_iteration > 0:
        c = (end + start)/2
        if f(c) * f(start) > 0:
            start = c
        elif f(c) * f(end) > 0:
            end = c
        else:
            return c
        i += 1
        max_iteration -= 1
```

```
    if max_iteration == 0:
        print(f'guess {c} is not an approximated root')

    return c, i


def newthon_raphson(f, start, end, epsilon, max_iteration = 200):
    guess = (end + start) / 2
    if max_iteration < 0:
        max_iteration *= -1
    i = 1
    while abs(f(guess)) > epsilon and max_iteration >= 0:
        guess = guess - f(guess)/f_tag(f, guess)
        i += 1
        max_iteration -= 1
    if max_iteration == 0:
        print(f'guess {guess} is not an approximated root')
    return guess, i


def secant(f, start, end, epsilon, max_iteration = 200):
    '''

    :param f:
    :param start:
    :param end:
    :param epsilon:
    :param max_iteration:
    :return:
    '''
```

```
    guess1 = (end + start) / 2
    guess2 = (end + start) / 3
    s = 0.00000001
    i = 0
    while abs(f(guess2)) > epsilon and max_iteration > 0:
        tmp = guess2
        if f(guess2) - f(guess1) == 0:
            guess1 += s
        guess2 = guess2 - f(guess2) * ((guess2 - guess1)/ (f(guess2) - f(guess1)))
        guess1 = tmp
        max_iteration -= 1
        i += 1

    if max_iteration == 0:
        print(f'guess {guess2} is not an approximated root')

    return guess2, i


def main():
    # Q8 - Bisection, Newthon Raphson.
    func = lambda x: sin(2*exp(-2*x)) / (x**2 + 5*x + 6)

    #polinom = ((x ** 2) * e ** ((-x ** 2) + 5 * x - 3)) * (3 * x - 1)
    # Q8 -1, 2; Q17 0, 1.5; Q16 0, 3
    start = -1
    end = 2
    jump = 0.1
    epsilon = 0.0000001
    x_epsilon = 0.001
    method_choices = {"1": biscetion, "2": newthon_raphson, "3": secant}
```

```
    choice = input("Please enter method:\n1) Bisection\n2) Newthon raphson\n3) Secant")
    while choice not in method_choices.keys():
        choice = input("Please enter method:\n1) Bisection\n2) Newthon raphson\n3) Secant")
    method = method_choices[choice]
    print(f'{choice} has been selected')
    roots = []
    s = start
    e = round(start + jump, 3)
    root_exits = False
    while e <= end:
        print(f'interval ({s}, {e}):')
        root, i = method(func, s, e, epsilon)
        if abs(func(root)) <= epsilon and start <= root <= end:
            for r in roots:
                if abs(root-r) <= x_epsilon:
                    root_exits = True
                    break
            if not root_exits:
                roots.append(root)
                print("------------------------------------")
                print(f'root : {root}, iterations: {i}')
                print("------------------------------------")
            root_exits = False
        s = e
        e = round(e + jump, 3)
        print("")
    print(roots)


if __name__ == "__main__":
    main()
```

#Q 8b:

```
import datetime
from math import *


def simpson_Method(func, a, b, epsilon):
    print("Simpson method is going by the formula - (h/3)*(f(a)+2*sigma(from j=1 to last even)*f(X2j)+4*sigma(from j=1 to last odd)*f(X2j-1)+f(b))")
    sol1 = 2 * epsilon
    sol = 0

    def run(func, a, b, slices):
        slices += slices
        h = (b - a) / slices
        print("h = ", h)
        print("a, b = {}, {}".format(a, b))
        arr = []
        for i in range(1, slices):
            arr.append(a + (((b - a) * i) / slices))
        arr.insert(0, a)
        arr.append(b)
        sol = func(a) + func(b)
        print(f'Adding f(start) + f(end): {func(a)} + {func(b)}')
        for i in range(1, len(arr) - 1):
            if i % 2 != 0:
                print(f'Iteration {i}, Last sum += 4 * (odd index value):\n{sol} += {4 * func(arr[i])}')
                sol += 4 * func(arr[i])
            else:
                print(f'Iteration {i}, Last sum += 2 * (even index value):\n{sol} += {2 * func(arr[i])}')
```

```
            sol += 2 * func(arr[i])
        print()
    print(f'Result = h/3 * {sol} = {sol * (h / 3)}')
    sol = sol * (h / 3)
    return sol

slices = 5
while abs(sol1 - sol) > epsilon:
    sol = run(func, a, b, slices)
    slices *= 2
    sol1 = run(func, a, b, slices)
return sol1



def find_intervals(a, b, N):
    jump = (b - a)/N
    intervals = []
    j = a
    for i in range(N + 1):
        intervals.append(j)
        j = j + jump
    return intervals



# trapezoidal rule
def trapezoid(f,a,b,N):
    yi=[]
    h = (b-a)/N
    xi = find_intervals(a,b,N)
    for i in xi:
        yi.append(f(i))
```

```
    s = 0.0
    for i in range(1,N):
        if yi[i] != inf and yi[i] != -inf:
            s = s + yi[i]
            print("number iteration of trapezoid method:",i)
            print("Approximation:",s)
        else:
            print("The Approximation of the number iteration",i,"of trapezoid method is -inf or
inf")
    s = (h/2)*(yi[0] + yi[N]) + h*s
    return s



def romberg(f,a,b,eps,nmax):
# f     ... function to be integrated
# [a,b] ... integration interval
# eps   ... desired accuracy
# nmax  ... maximal order of Romberg method
    Q = [[0 for _ in range(nmax)] for __ in range(nmax)]
    converged = 0
    k=0
    count=1
    print("Romberg method is using trapezoid method -  ")
    print("The formula of Trapezoidal is - sigma(from i=1 to N)*(h/2)*(f(Xi-h)+f(Xi))")
    print("The formula of Romberg Method is - R(n,m)=1/(4^m-1)*(4^m*R(n,m-1)-R*(n-1,m-
1))")
    for i in range(0,nmax):
        N = 2**i
        Q[i][0] = trapezoid(f,a,b,N)
        for k in range(0,i):
            n = k + 2
            Q[i][k+1] = 1.0/(4**(n-1)-1)*(4**(n-1)*Q[i][k] - Q[i-1][k])
```

```
            print("number iteration of romberg method:",count)
            print("Approximation:",Q[i][k])
            count+=1
        if (i > 0):
            if (abs(Q[i][k+1] - Q[i][k]) < eps):
                converged = 1
                break
    if nmax == 1:
        return Q[i][k]
    print("The result is -")
    return Q[i][k + 1]


def formated_solution(sol):
    time = datetime.datetime.now()
    return f'{sol}00000{time.day}{time.hour}{time.minute}'



# q8
f = lambda x: sin(2*e**(-2*x)) / (x**2 + 5*x + 6)

# Q8 -0.4, 0.4; Q17 0.5, 1; Q16 0.5, 1
a = -0.4
b = 0.4
# romberg_result = formated_solution(romberg(f, a, b, 0.0001, 10))
# print(romberg_result)
symphson_result = formated_solution(simpson_Method(f, a, b, 0.0001))
print(symphson_result)
```

```
#Q 16a:

# f(x) = x ** 2 + 2 * x + 2

# f(x1) * f(x2) < 0

#ruth avivi 208981555

#eden dahan 318641222

from math import exp, sin



def f_tag(f, x):
    """
    x = c
    f'(x) = lim h->0 (f(c + h) - f(c))/h
    """
    h = 0.0000000001

    return (f(x + h) - f(x))/h



def biscetion(f, start, end, epsilon, max_iteration = 200):
    c = start
    i = 1
    while abs(f(c)) > epsilon and max_iteration > 0:
        c = (end + start)/2
        if f(c) * f(start) > 0:
            start = c
        elif f(c) * f(end) > 0:
            end = c
        else:
            return c
        i += 1
        max_iteration -= 1
```

```
    if max_iteration == 0:
        print(f'guess {c} is not an approximated root')

    return c, i


def newthon_raphson(f, start, end, epsilon, max_iteration = 200):
    guess = (end + start) / 2
    if max_iteration < 0:
        max_iteration *= -1
    i = 1
    while abs(f(guess)) > epsilon and max_iteration >= 0:
        guess = guess - f(guess)/f_tag(f, guess)
        i += 1
        max_iteration -= 1
    if max_iteration == 0:
        print(f'guess {guess} is not an approximated root')
    return guess, i


def secant(f, start, end, epsilon, max_iteration = 200):
    '''

    :param f:
    :param start:
    :param end:
    :param epsilon:
    :param max_iteration:
    :return:
    '''
```

```
    guess1 = (end + start) / 2
    guess2 = (end + start) / 3
    s = 0.00000001
    i = 0
    while abs(f(guess2)) > epsilon and max_iteration > 0:
        tmp = guess2
        if f(guess2) - f(guess1) == 0:
            guess1 += s
        guess2 = guess2 - f(guess2) * ((guess2 - guess1)/ (f(guess2) - f(guess1)))
        guess1 = tmp
        max_iteration -= 1
        i += 1

    if max_iteration == 0:
        print(f'guess {guess2} is not an approximated root')

    return guess2, i


def main():

    # Q16 - Bisection, secant.
    func = lambda x: ((x ** 2) * exp(-(x ** 2) + 5 * x - 3)) * (3 * x - 5)

    #polinom = ((x ** 2) * e ** ((-x ** 2) + 5 * x - 3)) * (3 * x - 1)
    # Q8 -1, 2; Q17 0, 1.5; Q16 0, 3
    start = 0
    end = 3
    jump = 0.1
    epsilon = 0.0000001
    x_epsilon = 0.001
```

```python
    method_choices = {"1": biscetion, "2": newthon_raphson, "3": secant}
    choice = input("Please enter method:\n1) Bisection\n2) Newthon raphson\n3) Secant")
    while choice not in method_choices.keys():
        choice = input("Please enter method:\n1) Bisection\n2) Newthon raphson\n3) Secant")
    method = method_choices[choice]
    print(f'{choice} has been selected')
    roots = []
    s = start
    e = round(start + jump, 3)
    root_exits = False
    while e <= end:
        print(f'interval ({s}, {e}):')
        root, i = method(func, s, e, epsilon)
        if abs(func(root)) <= epsilon and start <= root <= end:
            for r in roots:
                if abs(root-r) <= x_epsilon:
                    root_exits = True
                    break
            if not root_exits:
                roots.append(root)
                print("------------------------------------")
                print(f'root : {root}, iterations: {i}')
                print("------------------------------------")
            root_exits = False
        s = e
        e = round(e + jump, 3)
        print("")
    print(roots)


if __name__ == "__main__":
    main()
```

#Q 16b:

```
import datetime
from math import *


def simpson_Method(func, a, b, epsilon):
    print("Simpson method is going by the formula - (h/3)*(f(a)+2*sigma(from j=1 to last even)*f(X2j)+4*sigma(from j=1 to last odd)*f(X2j-1)+f(b))")
    sol1 = 2 * epsilon
    sol = 0

    def run(func, a, b, slices):
        slices += slices
        h = (b - a) / slices
        print("h = ", h)
        print("a, b = {}, {}".format(a, b))
        arr = []
        for i in range(1, slices):
            arr.append(a + (((b - a) * i) / slices))
        arr.insert(0, a)
        arr.append(b)
        sol = func(a) + func(b)
        print(f'Adding f(start) + f(end): {func(a)} + {func(b)}')
        for i in range(1, len(arr) - 1):
            if i % 2 != 0:
                print(f'Iteration {i}, Last sum += 4 * (odd index value):\n{sol} += {4 * func(arr[i])}')
                sol += 4 * func(arr[i])
            else:
                print(f'Iteration {i}, Last sum += 2 * (even index value):\n{sol} += {2 * func(arr[i])}')
```

```
            sol += 2 * func(arr[i])
        print()
    print(f'Result = h/3 * {sol} = {sol * (h / 3)}')
    sol = sol * (h / 3)
    return sol

slices = 5
while abs(sol1 - sol) > epsilon:
    sol = run(func, a, b, slices)
    slices *= 2
    sol1 = run(func, a, b, slices)
return sol1


def find_intervals(a, b, N):
    jump = (b - a)/N
    intervals = []
    j = a
    for i in range(N + 1):
        intervals.append(j)
        j = j + jump
    return intervals


# trapezoidal rule
def trapezoid(f,a,b,N):
    yi=[]
    h = (b-a)/N
    xi = find_intervals(a,b,N)
    for i in xi:
        yi.append(f(i))
```

```
    s = 0.0
    for i in range(1,N):
        if yi[i] != inf and yi[i] != -inf:
            s = s + yi[i]
            print("number iteration of trapezoid method:",i)
            print("Approximation:",s)
        else:
            print("The Approximation of the number iteration",i,"of trapezoid method is -inf or inf")
    s = (h/2)*(yi[0] + yi[N]) + h*s
    return s


def romberg(f,a,b,eps,nmax):
# f     ... function to be integrated
# [a,b] ... integration interval
# eps   ... desired accuracy
# nmax  ... maximal order of Romberg method
    Q = [[0 for _ in range(nmax)] for __ in range(nmax)]
    converged = 0
    k=0
    count=1
    print("Romberg method is using trapezoid method -  ")
    print("The formula of Trapezoidal is - sigma(from i=1 to N)*(h/2)*(f(Xi-h)+f(Xi))")
    print("The formula of Romberg Method is - R(n,m)=1/(4^m-1)*(4^m*R(n,m-1)-R*(n-1,m-1))")
    for i in range(0,nmax):
        N = 2**i
        Q[i][0] = trapezoid(f,a,b,N)
        for k in range(0,i):
            n = k + 2
            Q[i][k+1] = 1.0/(4**(n-1)-1)*(4**(n-1)*Q[i][k] - Q[i-1][k])
```

```
            print("number iteration of romberg method:",count)
            print("Approximation:",Q[i][k])
            count+=1
        if (i > 0):
            if (abs(Q[i][k+1] - Q[i][k]) < eps):
                converged = 1
                break
    if nmax == 1:
        return Q[i][k]
    print("The result is -")
    return Q[i][k + 1]


def formated_solution(sol):
    time = datetime.datetime.now()
    return f'{sol}00000{time.day}{time.hour}{time.minute}'




# q16
f = lambda x: ((x**2)*e**((-x**2) - 5*x -3))*(3*x - 5)
# Q8 -0.4, 0.4; Q17 0.5, 1; Q16 0.5, 1
a = 0.5
b = 1
romberg_result = formated_solution(romberg(f, a, b, 0.0001, 10))
print(romberg_result)
symphson_result = formated_solution(simpson_Method(f, a, b, 0.0001))
print(symphson_result)
```

```
#q17a:

# f(x) = x ** 2 + 2 * x + 2

# f(x1) * f(x2) < 0

#ruth avivi 208981555

#eden dahan 318641222

from math import exp, sin



def f_tag(f, x):
    """
    x = c
    f'(x) = lim h->0 (f(c + h) - f(c))/h
    """
    h = 0.0000000001

    return (f(x + h) - f(x))/h



def biscetion(f, start, end, epsilon, max_iteration = 200):
    c = start
    i = 1
    while abs(f(c)) > epsilon and max_iteration > 0:
        c = (end + start)/2
        if f(c) * f(start) > 0:
            start = c
        elif f(c) * f(end) > 0:
            end = c
        else:
            return c
        i += 1
        max_iteration -= 1
```

```
    if max_iteration == 0:
        print(f'guess {c} is not an approximated root')

    return c, i


def newthon_raphson(f, start, end, epsilon, max_iteration = 200):
    guess = (end + start) / 2
    if max_iteration < 0:
        max_iteration *= -1
    i = 1
    while abs(f(guess)) > epsilon and max_iteration >= 0:
        guess = guess - f(guess)/f_tag(f, guess)
        i += 1
        max_iteration -= 1
    if max_iteration == 0:
        print(f'guess {guess} is not an approximated root')
    return guess, i


def secant(f, start, end, epsilon, max_iteration = 200):
    '''

    :param f:
    :param start:
    :param end:
    :param epsilon:
    :param max_iteration:
    :return:
    '''
```

```
    guess1 = (end + start) / 2

    guess2 = (end + start) / 3

    s = 0.00000001

    i = 0

    while abs(f(guess2)) > epsilon and max_iteration > 0:

        tmp = guess2

        if f(guess2) - f(guess1) == 0:

            guess1 += s

        guess2 = guess2 - f(guess2) * ((guess2 - guess1)/ (f(guess2) - f(guess1)))

        guess1 = tmp

        max_iteration -= 1

        i += 1


    if max_iteration == 0:

        print(f'guess {guess2} is not an approximated root')


    return guess2, i



def main():


    # Q17 - Bisection, secant.

    func = lambda x: ((x**2) * exp(-(x**2) + 5*x -3))*(3*x - 1)


    #polinom = ((x ** 2) * e ** ((-x ** 2) + 5 * x - 3)) * (3 * x - 1)

    # Q8 -1, 2; Q17 0, 1.5; Q16 0, 3

    start = 0

    end = 1.5

    jump = 0.1

    epsilon = 0.0000001

    x_epsilon = 0.001
```

```
    method_choices = {"1": biscetion, "2": newthon_raphson, "3": secant}
    choice = input("Please enter method:\n1) Bisection\n2) Newthon raphson\n3) Secant")
    while choice not in method_choices.keys():
        choice = input("Please enter method:\n1) Bisection\n2) Newthon raphson\n3) Secant")
    method = method_choices[choice]
    print(f'{choice} has been selected')
    roots = []
    s = start
    e = round(start + jump, 3)
    root_exits = False
    while e <= end:
        print(f'interval ({s}, {e}):')
        root, i = method(func, s, e, epsilon)
        if abs(func(root)) <= epsilon and start <= root <= end:
            for r in roots:
                if abs(root-r) <= x_epsilon:
                    root_exits = True
                    break
            if not root_exits:
                roots.append(root)
                print("------------------------------------")
                print(f'root : {root}, iterations: {i}')
                print("------------------------------------")
            root_exits = False
        s = e
        e = round(e + jump, 3)
        print("")
    print(roots)


if __name__ == "__main__":
    main()
```

#Q17b:

```
import datetime
from math import *


def simpson_Method(func, a, b, epsilon):
    print("Simpson method is going by the formula - (h/3)*(f(a)+2*sigma(from j=1 to last even)*f(X2j)+4*sigma(from j=1 to last odd)*f(X2j-1)+f(b))")
    sol1 = 2 * epsilon
    sol = 0

    def run(func, a, b, slices):
        slices += slices
        h = (b - a) / slices
        print("h = ", h)
        print("a, b = {}, {}".format(a, b))
        arr = []
        for i in range(1, slices):
            arr.append(a + (((b - a) * i) / slices))
        arr.insert(0, a)
        arr.append(b)
        sol = func(a) + func(b)
        print(f'Adding f(start) + f(end): {func(a)} + {func(b)}')
        for i in range(1, len(arr) - 1):
            if i % 2 != 0:
                print(f'Iteration {i}, Last sum += 4 * (odd index value):\n{sol} += {4 * func(arr[i])}')
                sol += 4 * func(arr[i])
            else:
                print(f'Iteration {i}, Last sum += 2 * (even index value):\n{sol} += {2 * func(arr[i])}')
```

```
            sol += 2 * func(arr[i])
          print()
        print(f'Result = h/3 * {sol} = {sol * (h / 3)}')
        sol = sol * (h / 3)
        return sol

    slices = 5
    while abs(sol1 - sol) > epsilon:
        sol = run(func, a, b, slices)
        slices *= 2
        sol1 = run(func, a, b, slices)
    return sol1


def find_intervals(a, b, N):
    jump = (b - a)/N
    intervals = []
    j = a
    for i in range(N + 1):
        intervals.append(j)
        j = j + jump
    return intervals


# trapezoidal rule
def trapezoid(f,a,b,N):
    yi=[]
    h = (b-a)/N
    xi = find_intervals(a,b,N)
    for i in xi:
        yi.append(f(i))
```

```
    s = 0.0
    for i in range(1,N):
        if yi[i] != inf and yi[i] != -inf:
            s = s + yi[i]
            print("number iteration of trapezoid method:",i)
            print("Approximation:",s)
        else:
            print("The Approximation of the number iteration",i,"of trapezoid method is -inf or
inf")
    s = (h/2)*(yi[0] + yi[N]) + h*s
    return s


def romberg(f,a,b,eps,nmax):
# f     ... function to be integrated
# [a,b] ... integration interval
# eps   ... desired accuracy
# nmax  ... maximal order of Romberg method
    Q = [[0 for _ in range(nmax)] for __ in range(nmax)]
    converged = 0
    k=0
    count=1
    print("Romberg method is using trapezoid method -  ")
    print("The formula of Trapezoidal is - sigma(from i=1 to N)*(h/2)*(f(Xi-h)+f(Xi))")
    print("The formula of Romberg Method is - R(n,m)=1/(4^m-1)*(4^m*R(n,m-1)-R*(n-1,m-
1))")
    for i in range(0,nmax):
        N = 2**i
        Q[i][0] = trapezoid(f,a,b,N)
        for k in range(0,i):
            n = k + 2
            Q[i][k+1] = 1.0/(4**(n-1)-1)*(4**(n-1)*Q[i][k] - Q[i-1][k])
```

```
            print("number iteration of romberg method:",count)
            print("Approximation:",Q[i][k])
            count+=1
        if (i > 0):
            if (abs(Q[i][k+1] - Q[i][k]) < eps):
                converged = 1
                break
    if nmax == 1:
        return Q[i][k]
    print("The result is -")
    return Q[i][k + 1]


def formated_solution(sol):
    time = datetime.datetime.now()
    return f'{sol}00000{time.day}{time.hour}{time.minute}'




# q17
f = lambda x: ((x**2)*e**((-x**2) - 5*x -3))*(3*x - 1)

# Q8 -0.4, 0.4; Q17 0.5, 1; Q16 0.5, 1
a = 0.5
b = 1
romberg_result = formated_solution(romberg(f, a, b, 0.0001, 10))
print(romberg_result)
symphson_result = formated_solution(simpson_Method(f, a, b, 0.0001))
print(symphson_result)
```

```
#Q19:
'''
Eden Dahan 318641222
Ruth Avivi 208981555
Ron Mansharof  208839787
Benny Shalom 203500780

Q19 solved by gauss elimination and gauss-seidel

A=[[1,0.5,1/3],[0.5,1/3,1/4],[1/3,1/4,1/5]]
b=[[1],[0],[0]]
'''



def print_Matrix(M):
    '''


    :param M:Matrix
    :return: print matrix
    '''
    for i in M:
        for j in i:
            print(j, end=" ")
        print()



def mul_matrix2(A,B):
    result = [[0 for i in range(len(A))] for j in range(len(B))]
    for i in range(len(X)):
        # iterate through columns of Y
        for j in range(len(Y[0])):
```

```
        # iterate through rows of Y
        for k in range(len(Y)):
            result[i][j] += X[i][k] * Y[k][j]



def mul_Matrix(A,B):
    '''

    :param A:matrix
    :param B: matrix
    :return: mul A and B (A is in the left, B is in the right)
    '''
    size = len(A)
    size2=len(B[0])
    result = [[0 for i in range(size2)] for j in range(size)]
    for i in range(len(A)):
        for j in range(size2):
            for k in range(len(B)):
                result[i][j] += A[i][k] * B[k][j]
    #print("TEST",result)
    return result




def swapRows(A,row1,row2):
    '''

    :param A: matrix
    :param row1: row
    :param row2: row
    :return: swap row1 and row2 and return the new matrix(A is update)
```

```
    '''
    temp=A[row1]
    A[row1]=A[row2]
    A[row2]=temp
    return A
#///////////////////////////////////////////////
def Pivot(A,row,b):
    '''

    :param A: matrix
    :param row: row index
    :return: make sure all the numbers on the diagonal are the largest in the column(A is
update)
    '''
    maximum=abs(A[row][row])
    help=row
    if row!=len(A)-1:
        for i in range(row+1,len(A)):
            if abs(A[i][row])>=maximum:
                maximum=abs(A[i][row])
                help=i
    if help!=row:
        swapRows(A,row,help)
        swapRows(b,row,help)
    return A,b
#///////////////////////////////////////////////
def identity_Matrix(A):
    '''

    :param A:matrix
    :return: identity matrix in size of A matrix
```

```
    '''
    size = len(A)
    b= [[0 for i in range(size)] for j in range(size)]
    for i in range(0,size):
        for j in range(0,size):
            if i==j:
                b[j][i]=1
    return b
#///////////////////////////////////////////////
def elementary_matrix(A,r):
    '''

    :param A:matrix
    :param r: row index
    :return: elementary matrix
    '''
    maximum=A[r][r]
    k=identity_Matrix(A)
    for i in  range(r+1,len(A)):
        if A[i][r]!=0 and maximum!=0:
            k[i][r]=-1*(A[i][r]/maximum)
    return k
#///////////////////////////////////////////////
#///////////////////////////////////////////////
def copyMatrix(M):
    '''
    :param M: Matrix
    :return: copy of matrix M
    '''
    size=len(M)
    size1=len((M[0]))
```

```
    m=[[0 for i in range(size1)] for j in range(size)]
    for i in range(0,size):
        for j in range(0,size1):
                m[i][j]=M[i][j]
    return m

#///////////////////////////////////////////////
def LU(A,r,b):
    """


    :param A: Matrix
    :param r: row index
    :return: U,L Matrices
    """
    #Pivot(A,0,b)
    size=len(A)
    U=copyMatrix(A)
    L=identity_Matrix(A)
    k = 1
    for i in range(size):
        Pivot(U, i,b)
        help=elementary_matrix(U,i)
        if help != identity_Matrix(A):
            print("J{0}: {1}".format(k,help))
            k=k+1
        L=mul_Matrix(L,help)
        U=mul_Matrix(help,U)
    if U[size - 1][size - 1]<0:
        U[size - 1][size - 1] = U[size - 1][size - 1] * (-1)
    for i in range(size):
        for j in range(size):
```

```
            if i!=j and L[i][j]!=0:
                L[i][j]=-1*L[i][j]
    print("L= ",end=" ")
    for i in range (1,k):
        if i == k-1:
            print("J{0}⁻¹".format(i),end=" ")
        else:
            print("J{0}⁻¹*".format(i),end=" ")
    print(" =",L)

    print("U= ", end=" ")
    for i in range(k-1,0,-1):
        print("J{0}*".format(i),end="")
    print("A =", U)
    return (U,L)
# ////////////////////////////////////////////////////
def zeros_matrix(A):
    '''

    :param A:matrix
    :return: zero matrix in A size
    '''
    b=copyMatrix(A)
    for i in range(len(A)):
        for j in range(len(A[0])):
            b[i][j]=0.0
    return b
# ////////////////////////////////////////////////////
def inverseMatrix(A):
    '''
```

```
    :param A: matrix
    :return: A^-1
    '''
    A2=copyMatrix(A)
    I=identity_Matrix(A)
    for a in range(len(A)):
        div1 = 1.0 / A2[a][a]
        for j in range(len(A)):
            A2[a][j] *= div1
            I[a][j] *= div1
        for i in list(range(len(A)))[0:a] + list(range(len(A)))[a + 1:]:
            div2 = A2[i][a]
            for j in range(len(A)):
                A2[i][j] = A2[i][j] - div2 * A2[a][j]
                I[i][j] = I[i][j] - div2 * I[a][j]
    return I

# ////////////////////////////////////////////////

def cond(A,A1):
    '''


    :param A:matrix
    :param A1: inverse matrix of A
    :return: cond (||A||*||A^-1||)
    '''
    size=len(A)
    #size1=len(A1)
    a=0
    a1=0
    for i in range(size):
```

```
        temp=0
        temp1=0
        for j in range (size):
            temp=temp+abs(A[i][j])
            temp1=temp1+abs(A1[i][j])
        if temp>a:
            a=temp
        if temp1>a1:
            a1=temp1
    print("||A||= {0}  , ||A1||= {1}".format(a,a1))
    print("Cond ||A||*||A^-1||= {0}".format(a*a1))
    return a*a1

# ////////////////////////////////////////////////////
def LU_CALC(A,b):
    cond(A,inverseMatrix(A))
    U,L = LU(A,0,b)
    L1=inverseMatrix(L)
    U1=inverseMatrix(U)
    x=mul_Matrix(L1,b)
    x=mul_Matrix(U1,x)
    print("X : {", end=" ")
    for i in range (len(x)):
        print("%0.6f00000131936," % x[i][0], end=" ")
    print("}",end=" ")
    print(" ")




# ////////////////////////////////////////////////////
def seidel_Calculation(A,b):
```

```
'''

    :param A: matrix in any size not only 3X3
    :param b: matrix of solution
    :return: find x (Ax=b) return x calculate in seidel calculation
    '''
 #if (seidel_Converge(A,b) == False):
  #   print("The matrix does not converge")
# else:
 p=1
 if(p==1):
    x = copyMatrix(b)
    x = zeros_matrix(b)
    x1 = zeros_matrix(x)
    flag = True
    epsilon = 2**(-52)
    counter = 0
    #print("Count      x           y           z")
    print("Count    ",end=" ")
    for i in range(len(x)):
        print("var{0}        ".format(i+1),end=" ")
    print(" ")
    while flag:
        x = copyMatrix(x1)
        p=0
        print(counter,end = " ")
        while p<len(x):
            print("     ", end=" ")
            #print("%0.6f"%x[p][0],end = " ")
            print(x[p][0], end=" ")
```

```
            #print("      ",end = " ")
            p+=1
        for i in range(len(A)):
            temp = b[i][0]
            for j in range(len(A)):
                if i != j:
                    temp = temp - A[i][j] * x1[j][0]
            temp = temp / A[i][i]
            x1[i][0] = temp
        flag = abs(x1[0][0] - x[0][0]) > epsilon
        counter += 1
        print(" ")
    print("Solution={", end=" ")
    for i in range(len(x)):
        print("var{0}=".format(i+1), end=" ")
        print("%0.6f00000131936"%x1[i][0],end = " ")
    print("}")
    print(" ")


A=[[1,0.5,1/3],[0.5,1/3,1/4],[1/3,1/4,1/5]]
b=[[1],[0],[0]]
print("By LU Method: A = LU")
LU_CALC(A,b)
print("\n\n")
print("By seidel Method: ")
seidel_Calculation(A,b)
```

```
#Q21:
'''
Eden Dahan 318641222
Ruth Avivi 208981555
Ron Mansharof  208839787
Benny Shalom 203500780

Q21 solved by gauss elimination and jaacobian

A=[[10, 8, 1],[4, 10, -5],[5, 1, 10]]
b=[[-7],[2],[1.5]]
'''



def print_Matrix(M):
    '''

    :param M:Matrix
    :return: print matrix
    '''
    for i in M:
        for j in i:
            print(j, end=" ")
        print()



def mul_Matrix(A,B):
    '''

    :param A:matrix
    :param B: matrix
```

```
    :return: mul A and B (A is in the left, B is in the right)
    '''
    size = len(A)
    size2=len(B[0])
    result = [[0 for i in range(size2)] for j in range(size)]
    for i in range(len(A)):
        for j in range(size2):
            for k in range(len(B)):
                result[i][j] += A[i][k] * B[k][j]
    #print("TEST",result)
    return result



def swapRows(A,row1,row2):
    '''

    :param A: matrix
    :param row1: row
    :param row2: row
    :return: swap row1 and row2 and return the new matrix(A is update)
    '''
    temp=A[row1]
    A[row1]=A[row2]
    A[row2]=temp
    return A
#////////////////////////////////////////////////
def Pivot(A,row,b):
    '''

    :param A: matrix
```

```
    :param row: row index
    :return: make sure all the numbers on the diagonal are the largest in the column(A is
update)
    '''
    maximum=abs(A[row][row])
    help=row
    if row!=len(A)-1:
        for i in range(row+1,len(A)):
            if abs(A[i][row])>=maximum:
                maximum=abs(A[i][row])
                help=i
    if help!=row:
        swapRows(A,row,help)
        swapRows(b,row,help)
    return A,b
#///////////////////////////////////////////////
def identity_Matrix(A):
    '''

    :param A:matrix
    :return: identity matrix in size of A matrix
    '''
    size = len(A)
    b= [[0 for i in range(size)] for j in range(size)]
    for i in range(0,size):
        for j in range(0,size):
            if i==j:
                b[j][i]=1
    return b
#///////////////////////////////////////////////
def elementary_matrix(A,r):
```

```
    '''

    :param A:matrix
    :param r: row index
    :return: elementary matrix
    '''
    maximum=A[r][r]
    k=identity_Matrix(A)
    for i in  range(r+1,len(A)):
        if A[i][r]!=0 and maximum!=0:
            k[i][r]=-1*(A[i][r]/maximum)
    return k
#////////////////////////////////////////////////
#////////////////////////////////////////////////
def copyMatrix(M):
    '''
    :param M: Matrix
    :return: copy of matrix M
    '''
    size=len(M)
    size1=len((M[0]))
    m=[[0 for i in range(size1)] for j in range(size)]
    for i in range(0,size):
        for j in range(0,size1):
                m[i][j]=M[i][j]
    return m

#////////////////////////////////////////////////
def LU(A,r,b):
    """
```

```
:param A: Matrix
:param r: row index
:return: U,L Matrices
"""
#Pivot(A,0,b)
size=len(A)
U=copyMatrix(A)
L=identity_Matrix(A)
k = 1
for i in range(size):
   Pivot(U, i,b)
   help=elementary_matrix(U,i)
   if help != identity_Matrix(A):
      print("J{0}: {1}".format(k,help))
      k=k+1
   L=mul_Matrix(L,help)
   U=mul_Matrix(help,U)
if U[size - 1][size - 1]<0:
   U[size - 1][size - 1] = U[size - 1][size - 1] * (-1)
for i in range(size):
   for j in range(size):
      if i!=j and L[i][j]!=0:
         L[i][j]=-1*L[i][j]
print("L= ",end=" ")
for i in range (1,k):
   if i == k-1:
      print("J{0}⁻¹".format(i),end=" ")
   else:
      print("J{0}⁻¹*".format(i),end=" ")
print(" =",L)
```

```
    print("U= ", end=" ")
    for i in range(k-1,0,-1):
        print("J{0}*".format(i),end="")
    print("A =", U)
    return (U,L)
# ////////////////////////////////////////////////



def Cond(A,A1):
    '''

    :param A:matrix
    :param A1: inverse matrix of A
    :return: cond (||A||*||A^-1||)
    '''
    size=len(A)
    #size1=len(A1)
    a=0
    a1=0
    for i in range(size):
        temp=0
        temp1=0
        for j in range (size):
            temp=temp+abs(A[i][j])
            temp1=temp1+abs(A1[i][j])
        if temp>a:
            a=temp
        if temp1>a1:
            a1=temp1
    print("||A||= {0}  , ||A1||= {1}".format(a,a1))
```

```
    print("Cond ||A||*||A^-1||= {0}".format(a*a1))
    return a*a1

# ////////////////////////////////////////////////



def inverseMatrix(A):
    '''


    :param A: matrix
    :return: A^-1
    '''
    A2=copyMatrix(A)
    I=identity_Matrix(A)
    for a in range(len(A)):
        div1 = 1.0 / A2[a][a]
        for j in range(len(A)):
            A2[a][j] *= div1
            I[a][j] *= div1
        for i in list(range(len(A)))[0:a] + list(range(len(A)))[a + 1:]:
            div2 = A2[i][a]
            for j in range(len(A)):
                A2[i][j] = A2[i][j] - div2 * A2[a][j]
                I[i][j] = I[i][j] - div2 * I[a][j]
    return I





# ////////////////////////////////////////////////
def LU_CALC(A,b):
```

```
    Cond(A,inverseMatrix(A))
    U,L = LU(A,0,b)
    L1=inverseMatrix(L)
    U1=inverseMatrix(U)
    x=mul_Matrix(L1,b)
    x=mul_Matrix(U1,x)
    print("X : {", end=" ")
    for i in range (len(x)):
        print("%0.6f00000132016," % x[i][0], end=" ")
    print("}",end=" ")
    print(" ")
# ////////////////////////////////////////////////



#////////////////////////////////////////////////
def add_matrix(A,B):
    '''

    :param A:matrix
    :param B: matrix
    :return: add of 2 maricses (A+B)
    '''
    result=copyMatrix(A)
    for i in range(len(A)):
        for j in range(len(A[0])):
            result[i][j] = A[i][j] + B[i][j]
    return result
#////////////////////////////////////////////////
def zeros_matrix(A):
    '''
```

```
    :param A:matrix
    :return: zero matrix in A size
    '''
    b=copyMatrix(A)
    for i in range(len(A)):
        for j in range(len(A[0])):
            b[i][j]=0.0
    return b
#////////////////////////////////////////////////
def mul_Matrix(A,B):
    '''

    :param A:matrix
    :param B: matrix
    :return: mul A and B (A is in the left, B is in the right)
    '''
    rowsA = len(A)
    colsA = len(A[0])
    rowsB = len(B)
    colsB = len(B[0])
    if colsA != rowsB:
        raise ArithmeticError(
            'Number of A columns must equal number of B rows.')
    C = zeros_matrix(B)
    for i in range(rowsA):
        for j in range(colsB):
            total = 0
            for ii in range(colsA):
                total += A[i][ii] * B[ii][j]
            C[i][j] = total
```

```
    return C
#////////////////////////////////////////////////
def mul_Num_Matrix(n,A):
    '''

    :param n:number
    :param A: matrix
    :return: mul n and A
    '''
    size = len(A)
    M=copyMatrix(A)
    for i in range(size):
        for j in range(size):
            if M[i][j]!=0:
                M[i][j]=n*M[i][j]
    return M
#////////////////////////////////////////////////
def swapRows(A,row1,row2):
    '''

    :param A: matrix
    :param row1: row
    :param row2: row
    :return: swap row1 and row2 and return the new matrix(A is update)
    '''
    temp=A[row1]
    A[row1]=A[row2]
    A[row2]=temp
    return A
#////////////////////////////////////////////////
```

```
def Pivot(A,row,b):
   '''


   :param A: matrix
   :param row: row index
   :return: make sure all the numbers on the diagonal are the largest in the column(A is
update)
   '''
   maximum=abs(A[row][row])
   help=row
   if row!=len(A)-1:
      for i in range(row+1,len(A)):
         if abs(A[i][row])>=maximum:
            maximum=abs(A[i][row])
            help=i
   if help!=row:
      swapRows(A,row,help)
      swapRows(b,row,help)
   return A
#///////////////////////////////////////////////
def copyMatrix(M):
   '''
   :param M: Matrix
   :return: copy of matrix M
   '''
   size=len(M)
   size1=len((M[0]))
   m=[[0 for i in range(size1)] for j in range(size)]
   for i in range(0,size):
      for j in range(0,size1):
            m[i][j]=M[i][j]
```

```
    return m


#//////////////////////////////////////////////
def identity_Matrix(A):
    '''
    :param A:matrix
    :return: identity matrix in size of A matrix
    '''
    size = len(A)
    b= [[0 for i in range(size)] for j in range(size)]
    for i in range(0,size):
        for j in range(0,size):
            if i==j:
                b[j][i]=1
    return b
#//////////////////////////////////////////////
def inverseMatrix(A):
    '''
    :param A: matrix
    :return: A^-1
    '''
    A2=copyMatrix(A)
    I=identity_Matrix(A)
    for a in range(len(A)):
        div1 = 1.0 / A2[a][a]
        for j in range(len(A)):
            A2[a][j] *= div1
            I[a][j] *= div1
        for i in list(range(len(A)))[0:a] + list(range(len(A)))[a + 1:]:
            div2 = A2[i][a]
            for j in range(len(A)):
```

```
            A2[i][j] = A2[i][j] - div2 * A2[a][j]
            I[i][j] = I[i][j] - div2 * I[a][j]
    return I

# ///////////////////////////////////////////////
def cond(A):
    '''

    :param A:matrix
    :return: cond of matrix A
    '''
    size=len(A)
    a=0
    for i in range(size):
        temp=0
        for j in range (size):
            temp=temp+abs(A[i][j])
        if temp>a:
            a=temp
    return a
# ///////////////////////////////////////////////
def jaacobian_Converge(A,b):
    '''

    :param A: matrix
    :return:  check if ||G|| < 1 (G = (-D^-1)*(L+U) for jaacobian
    '''
    L,D,U=LDU(A,b)
    G=add_matrix(L,U)
    D=mul_Num_Matrix(-1,inverseMatrix(D))
    G=mul_Matrix(D,G)
```

```
    a=cond(G)
    print("||G||= {0}".format(a))
    if a < 1:
        print("Converge")
        return True
    else:
        print("Does not converge")
        return False
# ////////////////////////////////////////////////
def jaacobian_Calculate(A,b):
    '''

    :param A: matrix in any size not only 3X3
    :param b: matrix of solution
    :return: find x (Ax=b) return x calculate in jaacobian calculation
    '''
    if (jaacobian_Converge(A,b)== False):
        print("The matrix does not converge")
    else:
        x = copyMatrix(b)
        x = zeros_matrix(b)
        x1 = zeros_matrix(x)
        flag = True
        epsilon= 0.00001
        counter = 0
        print("Count    ", end=" ")
        for i in range(len(x)):
            print("var{0}        ".format(i + 1), end=" ")
        print(" ")
        while flag:
            x=copyMatrix(x1)
```

```
            p = 0
            print(counter, end=" ")
            while p < len(x):
                print("     ", end=" ")
                print("%0.5f" % x[p][0], end=" ")
                # print("      ",end = " ")
                p += 1
            for i in range(len(A)):
                temp=b[i][0]
                for j in range(len(A)):
                    if i != j:
                        temp=temp-A[i][j]*x[j][0]
                temp=temp/A[i][i]
                x1[i][0]=temp
            flag=abs(x1[0][0]-x[0][0])>epsilon
            counter+=1
            print(" ")
            #print current x,y,z
        print("Solution : {", end=" ")
        for i in range(len(x)):
            print("var{0}=".format(i+1),end=" ")
            print("%0.4f00000132016," % x[i][0], end=" ")
        print("}", end=" ")
        print(" ")
# ////////////////////////////////////////////////



# ////////////////////////////////////////////////
def LDU(A,b):
    '''
```

```
    :param A:matrix
    :return: L , D , U matrices
    '''
    #Pivot(A,0,b)
    L=copyMatrix(A)
    D=copyMatrix(A)
    U=copyMatrix(A)
    size=len(A)
    for i in range(size):
        for j in range(size):
            if i != j:
                D[i][j]=0
    for i in range(size):
        L[i][i]=0
        U[i][i]=0
    for i in range(1,size):
        for j in range(i-1,-1,-1):
            L[j][i]=0
    for i in range(size):
        for j in range(i,size):
            U[j][i]=0
    print("L=",L)
    print("D=",D)
    print("U=",U)
    return L,D,U
# ////////////////////////////////////////////////////
# ////////////////////////////////////////////////////



A=[[10, 8, 1],[4, 10, -5],[5, 1, 10]]
```

```
b=[[-7],[2],[1.5]]

print("By LU Method:")

LU_CALC(A,b)

print("\n\n")

print("By jaacobian Method:")

jaacobian_Calculate(A,b)
```

```
#Q32:
#Git link:
'''
Eden Dahan 318641222
Ruth Avivi 208981555
Ron Mansharof  208839787
Benny Shalom 203500780

Question num 32 : solved by lagrange interpolation , linear interpolation.
'''
import datetime



def lagrangeInterpolation(table,point):
   sol=0
   l=1
   counter=0
   p="P{0}({1})".format(len(table)-1,point)

   for i in range(len(table)):
      print("l{0}({1})=".format(i,point),end=" ")
      for j in range(len(table)):
         if i != j :
            if (i==len(table)-1) and j == (len(table) - 2):
               print("(({0}-{1})/({2}-{3}))=".format(point, table[j][0], table[counter][0],
table[j][0]), end=" ")
            elif j != len(table)-1:
               print("(({0}-{1})/({2}-
{3}))*".format(point,table[j][0],table[counter][0],table[j][0]),end=" ")
            else:
               print("(({0}-{1})/({2}-{3}))=".format(point, table[j][0], table[counter][0],
table[j][0]), end=" ")
```

```
                l*=(point-table[j][0])/(table[counter][0]-table[j][0])
        print(l,end=" ")
        print(" ")
        l*=table[counter][1]
        counter+=1
        sol+=l
        l=1
    print("P{0}({1}) = ".format(len(table)-1,point),end=" ")
    for i in range (len(table)):
        if i == len(table)-1:
            print("l{0}({1})*y{2}  ".format(i,point,i),end=" ")
        else:
            print("l{0}({1})*y{2} + ".format(i,point,i),end=" ")
    print(" ")
    return (sol)

#//////////////////////////////////////////////////////////////////////////////
def linearInterpolation(table,point):
    for i in range (0,len(table)):
        if point<table[i+1][0] and point > table[i][0]:
            print("f(x)=((y₁-y₂)/(x₁-x₂))*point + (y₂x₁ - y₁x₂)/(x₁-x₂)")
            y1=table[i][1]
            y2=table[i+1][1]
            x1=table[i][0]
            x2=table[i+1][0]
            sol=((y1-y2)/(x1-x2))*point+(y2*x1-y1*x2)/(x1-x2)
            print("f(x)=(({0}-{1})/({2}-{3}))*{10} + ({4} * {5} - {6} * {7})/({8}-{9})
".format(y1,y2,x1,x2,y2,x1,y1,x2,x1,x2,point))
            print("f({0}) = {1}".format(point,sol))
            return sol
    return "Not sol for this table "
```

```
#///////////////////////////////////////////////////////////////////////////////

k=[[0.2, 13.7241], [0.35, 13.9776], [0.45, 14.0625], [0.6, 13.9776], [0.75,13.7241], [0.85, 13.3056], [0.9, 12.7281]]

sol_lagrange=lagrangeInterpolation(k, 0.65)

time = datetime.datetime.now()

print("lagrange formula - sigma(from i=1 to n)*Li(x)*Yi")

print("lagrange sol = ",sol_lagrange,end = "")

print("00000",end = "")

print(time.day,end = "")

print(time.hour,end = "")

print(time.minute,end = "")

print("")

print("\n")

sol_linear=linearInterpolation(k, 0.65)

print("linear sol = ",sol_linear,end = "")

print("00000",end = "")

print(time.day,end = "")

print(time.hour,end = "")

print(time.minute,end = "")
```