Fundamentals of Scientific Computing

Final Exam

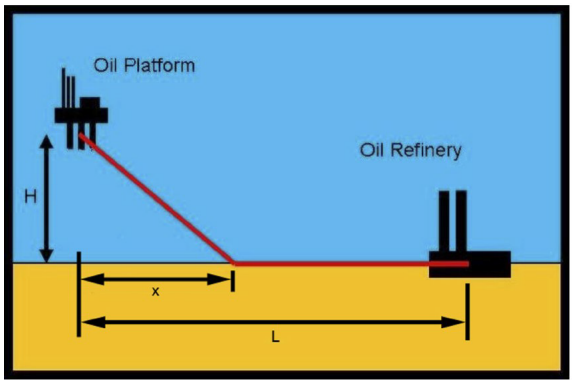
Third Year Students

1. Use

```
import numpy as np
T = -1*np.eye(10,10,k=-1)+np.eye(10,10)*2+-1*np.eye(10,10,k=1)
```

to create a tridiagonal matrix T. Write a program to factorize T into L and U.

2.



Optimization problem:

Note that you have to use bisection method or a bracketing methid for this problem. Bisection or bracketing method algorithm is as follows:

```
input:Function f,
       endpoint values a, b
       tolerance TOL
       maximum iterations NMAX
conditions: a < b
            either f(a) < 0 and f(b) > 0 or f(a) > 0 and f(b) < 0
output: value which differs from a root of f(x) = 0 by less than TOL

N← 1
while N ≤ NMAX do // limit iterations to prevent infinite loop
       c ← (a + b)/2 // new midpoint
       if f(c) = 0 or (b - a)/2 < TOL then // solution found
          Output(c)
          Stop
       end if
       N ← N + 1 // increment step counter
       if sign(f(c)) = sign(f(a)) then a ← c else b ← c // new interval
end while
Output("Method failed.") // max number of steps exceeded
Courtesy: Wikipedia
```

We are tasked with constructing a pipeline that stretches from an offshore oil platform to an onshore refinery station. The pipeline's construction costs vary depending on the environment. The objective is to find the optimal path for the pipeline that minimizes the total construction cost.

Refer to the attached figure for a visual representation of the problem. The oil platform is located a distance H miles from the shoreline, while the refinery is L miles along the coast from the point perpendicular to the platform. The pipeline will run in a straight line from the platform to a point x on the shoreline, and then continue along the shore to the refinery.

Cost Calculation

The cost to lay the pipeline is C_ocean dollars per mile in the ocean and C_land dollars per mile on land.

Objective

Write a function my_pipe_builder(C_ocean, C_land, L, H) that computes the most cost-effective point x on the shoreline to connect the pipeline from the platform to the refinery.

C_ocean: Cost per mile of laying pipeline in the ocean.

C_land: Cost per mile of laying pipeline on land.

L: The total distance along the shoreline from the point perpendicular to the platform to the refinery.

H: The distance from the oil platform to the shoreline.

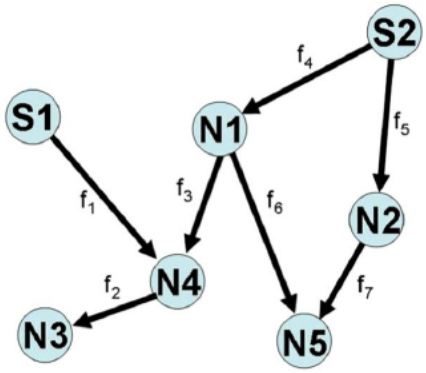
Method

Use the bisection method to find the value of x that minimizes the total cost. The method should converge to the solution within a tolerance of 1×10^-6. Start with an initial bound of a = 0 and b = L.

Test Cases:

In: my_pipe_builder(20, 10, 100, 50)
Out: 28.867512941360474
In: my_pipe_builder(30, 10, 100, 50)
Out: 17.677670717239380
In: my_pipe_builder(30, 10, 100, 20)
Out: 7.071067392826080

3.



We are given a power distribution network consisting of two power supply stations and five nodes. Each node has a specific power demand that must be met exactly. The network is structured as follows:

S1 and S2 are the power supply stations with known capacities. N1 to N5 are the nodes with specified power demands. Arrows connecting the nodes (f1 to f7) represent the power flow between them.The capacity of the power supply stations is represented by the vector S, and the power demand at each node is represented by the vector d. The flow of power along each arrow is denoted by f_j, where a negative flow implies power is running in the opposite direction of the arrow.

Objective

Write a function my_flow_calculator(S, d) that calculates the power flow through the network:

S is a 1x2 vector representing the capacity of each power supply station (S[0] for S1, S[1] for S2). d is a 1x5 vector representing the demand at each node (d[0] for N1 to d[4] for N5).

The function should return a 1x7 vector f representing the flow in the network (f[0] for f1 to f[6] for f7).

Constraints

The total flow into a node must equal the total flow out of the node plus the demand at that node. Power supply stations must run at their capacity.

The sum of the capacities of the power supply stations equals the sum of the demands at the nodes ($\sum S_j = \sum d_i$).

Notes

There may be multiple solutions to the system of equations. The function should ensure that all constraints are satisfied and each node's demand is exactly met.

Test Cases:

```
s = np.array([[10, 10]])
d = np.array([[4, 4, 4, 4, 4]])
# f = [[10.0, 4.0, -2.0, 4.5, 5.5, 2.5, 1.5]]
f = my_flow_calculator(s, d)
s = np.array([[10, 10]])
d = np.array([[3, 4, 5, 4, 4]])
# f = [[10.0, 5.0, -1.0, 4.5, 5.5, 2.5, 1.5]]
f = my_flow_calculator(s, d)
```

4. Write a Python code to find inverse of any n by n matrix without using any inbuilt libraries.

5. Write an iterative algorithm to solve the Tower of Hanoi problem with 'n' number of pegs.

6. Given an N×N chessboard, place N queens on it so that no two queens can attack each other. Represent the solution as a matrix with 'Q' for queens and '.' for empty spaces. Find a valid arrangement.

Input:

Integer N (1 <= N <= 15), the size of the chessboard.

Output:

N×N matrix with valid queen placement.

Example:

Input: N = 4

Output:

```
. Q . .
. . . Q
Q . . .
. . Q .
```

Solve the N-Queens puzzle, adhering to the constraints.

7. You are given an integer n, representing the number of pairs of balanced parentheses. Your task is to generate all possible sequences of balanced parentheses using these n pairs.

Examples:

```
generate_balanced_parentheses(1)
# Output: ['()']
generate_balanced_parentheses(2)
# Output: ['(())', '()()']
generate_balanced_parentheses(3)
# Output: ['((()))', '(()())', '(())()', '()(())', '()()()']
```