Solutions to Homework 3

Question 1: 10 points. Figure 2 shows an elastic column fixed at its base and pinned at the top. The critical buckling load, P_{cr}, corresponds to the first positive solution to

$$\lambda = \tan[\lambda] \tag{1}$$

where $P_{cr} = \lambda^2 \left[\frac{EI}{L^2}\right]$.

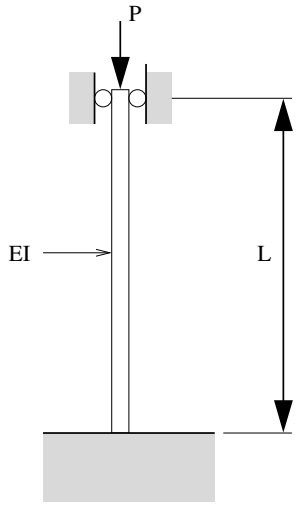


Figure 1: Elastic column carrying an axial load.

Use Python to create a single plot of y_1 = x and y_2 = tan(x). (The intersection of contours on this plot should give you an approximate range within which an accurate solution exists.) Compute the numerical solution to equation 1 using the method of bisection. Print the buckling load corresponding to your solution?

Python Source Code:

```
# ================================================================
# TestElasticColumnBuckling01.py: Use bisection algorithm to compute
# buckling load of an elastic column.
#
# Written by: Mark Austin                                March 2024
# ================================================================
```

```
import math
import matplotlib.pyplot as plt
import numpy as np

import Solutions;

# Compute y = tan(x) - x ...

def f1(x):
    return math.tan(x) - x;

# Main function ...

def main():
    print("--- Part 1: Create plots for y = x, y = tan(x) ... ");

    print("---   Create (x,y) coords for y1 = tan(x) ... ");

    x  = np.arange( -2.0, 6.0, 0.1)
    y1 = np.tan(x)

    print("---   Create (x,y) coords for y2 = x ... ");

    y2 = x

    plt.plot(x,y1, 'b', label='y1 = tan(x)')
    plt.plot(x,y2, 'r', label='y2 = x')
    plt.title('Plot: y = x and y = tan(x)')
    plt.ylabel('y')
    plt.xlabel('x')
    plt.ylim( -10, 10)
    plt.xlim(  -2,  7)
    plt.grid(True)
    plt.legend()
    plt.show()

    print("--- Part 2: Use Bisection algorithm to compute positive root tan(x) - x = 0 ... ");

    # Initialize problem setup ...

    a =  4.0;
    b =  4.7
    tolerance     = 0.01
    maxiterations = 100

    print("--- Inputs:")
    print("---   a = {:5.2f} ...".format(a) )
    print("---   b = {:5.2f} ...".format(b) )
    print("---   tolerance      = {:8.5f} ...".format(tolerance) )
    print("---   max iterations = {:8.2f} ...".format(maxiterations) )

    # Compute roots to equation ...

    print("--- Execution:")
```

```
    root, i, converged = Solutions.bisection(f1, a, b, tolerance, maxiterations )

    # Summary of computations ...

    print("--- Output:")
    print("---   root = {:10.5f} ...".format(root) )
    print("---   f(root) --> {:12.5e} ...".format( f1(root)) )
    print("---   no iterations = {:d} ...".format(i) )
    print("---   converged: {:s} ...".format( str(converged) ) )

    print("--- Part 3: Compute column buckling load ... ");

    print("")
    print("---   Buckling load = {:5.2f} * EL/L^2 ... ".format(root*root));

# call the main method ...

main()
```

Program Output:

```
--- Part 1: Create plots for y = x, y = tan(x) ...

---   Create (x,y) coords for y1 = tan(x) ...
---   Create (x,y) coords for y2 = x ...

--- Part 2: Use Bisection algorithm to compute positive root tan(x) - x = 0 ...

--- Inputs:
---   a =  4.00 ...
---   b =  4.70 ...
---   tolerance      =  0.01000 ...
---   max iterations =   100.00 ...
--- Execution:
---   Initial Conditions:
---   f(a) --> -2.84218e+00 ...
---   f(b) -->  7.60128e+01 ...
---   Main Loop for Root Computation:
---   Iteration 000: dx = 3.50000e-01, x =  4.3500000e+00, f(x) --> -1.7124015e+00 ...
---   Iteration 001: dx = 1.75000e-01, x =  4.5250000e+00, f(x) -->  7.4888339e-01 ...
---   Iteration 002: dx = 8.75000e-02, x =  4.4375000e+00, f(x) --> -8.9176234e-01 ...
---   Iteration 003: dx = 4.37500e-02, x =  4.4812500e+00, f(x) --> -2.3217078e-01 ...
---   Iteration 004: dx = 2.18750e-02, x =  4.5031250e+00, f(x) -->  2.0556909e-01 ...
---   Iteration 005: dx = 1.09375e-02, x =  4.4921875e+00, f(x) --> -2.4530831e-02 ...
---   Iteration 006: dx = 5.46875e-03, x =  4.4976563e+00, f(x) -->  8.7497087e-02 ...
---   Iteration 007: dx = 2.73438e-03, x =  4.4949219e+00, f(x) -->  3.0756122e-02 ...
---   Iteration 008: dx = 1.36719e-03, x =  4.4935547e+00, f(x) -->  2.9343009e-03 ...
--- Output:
---   root =    4.49355 ...
---   f(root) -->  2.93430e-03 ...
---   no iterations = 8 ...
---   converged: True ...
```

```
--- Part 3: Compute column buckling load ...

---   Buckling load = 20.19 * EL/L^2 ...
```

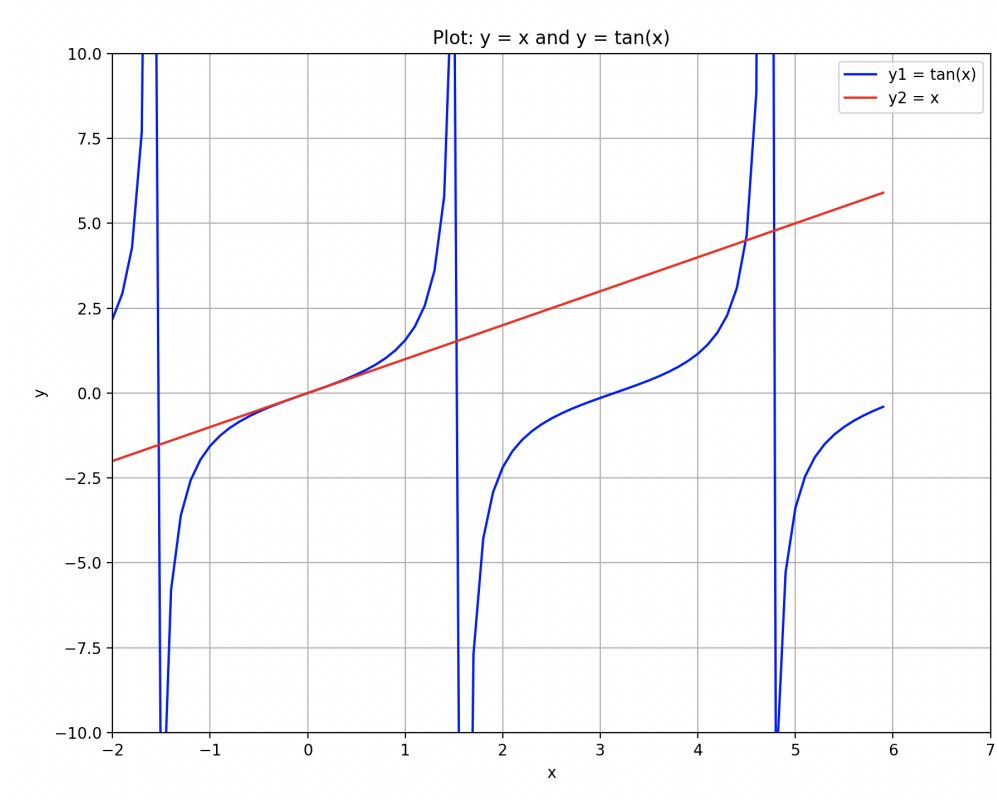


Figure 2: Plots: y_1 = x and y_2 = tan(x).

Question 2: 10 points. An efficient way of computing the cube root of a number N is to compute the root of

$$f(x) = x^3 - N = 0 \tag{2}$$

with the method of Newton Raphson, namely:

$$x_{n+1} = x_n - \left[\frac{f(x_n)}{f'(x_n)}\right] \tag{3}$$

Substituting equation 2 into equation 3 and rearranging terms gives the recursive relationship

$$x_{n+1} = \frac{1}{3} \cdot \left[\frac{2x_n^3 + N}{x_n^2}\right] \tag{4}$$

Write a test Python program that will compute the cube root of a number N via equation 4. The details of the cube root calculation should be contained within a method having the declaration:

```
def cuberoot(  N, tolerance, maxiterations ):
```

Your cube root method should use:

$$\left|\frac{x_{n+1} - x_n}{x_n}\right| < \varepsilon \tag{5}$$

for a test on convergence, where ε is a very small number (i.e., the tolerance). Print the number N, its cube root, whether or not the numerical procedure converged, and finally, the number of iterations. Verify that your test program works for a variety of positive and negative values of N (e.g., N = 8, 1000, -8, -27, etc).

Python Source Code:

```
# ============================================================================
# TestCubeRoot01.py: Use newton raphson strategy to compute cube root of N.
#
# Written By: Mark Austin                                          April 2024
# ============================================================================

import math;

# Cube root iteration ...
```

```
def cuberoot( N, tolerance, maxiterations ):

    # Main loop for cube root calculation ....

    x = 1.0
    converged = False
    for i in range(0, maxiterations + 1):
        xnew  = (1/3)*(2*x*x*x + N)/(x*x)

        if math.fabs((xnew - x)/xnew) <= tolerance:
           converged = True
           break

        x = xnew;

    root = x
    return root, i, converged;

# main method ...

def main():
    print("--- Enter TestCubeRoot01.main()               ... ");
    print("--- ======================================== ... ");

    tolerance     = 0.0001
    maxiterations = 100

    # Exercise cuberoot function for N = 8, 1000, -8, -27.

    N = 8.0;
    root, i, converged = cuberoot(N, tolerance, maxiterations );
    if converged == True:
       print("--- Cube root ({:7.1f}) ---> {:16.6f} ...".format( N, root))
       print("---           converged ---> {:s} ...".format( str( converged ) ))
       print("---           tolerance ---> {:f} ...".format( tolerance ))
       print("---       no iterations ---> {:d} ...".format( i ))
    else:
       print("--- Cube root ({:7.1f}) ---> covergence failed ...")

    N = 1000.0;
    root, i, converged = cuberoot(N, tolerance, maxiterations );
    if converged == True:
       print("--- Cube root ({:7.1f}) ---> {:16.6f} ...".format( N, root))
       print("---           converged ---> {:s} ...".format( str( converged ) ))
       print("---           tolerance ---> {:f} ...".format( tolerance ))
       print("---       no iterations ---> {:d} ...".format( i ))
    else:
       print("--- Cube root ({:7.1f}) ---> covergence failed ...")

    N = -8.0;
    root, i, converged = cuberoot(N, tolerance, maxiterations );
    if converged == True:
       print("--- Cube root ({:7.1f}) ---> {:16.6f} ...".format( N, root))
       print("---           converged ---> {:s} ...".format( str( converged ) ))
       print("---           tolerance ---> {:f} ...".format( tolerance ))
```

```
       print("---       no iterations ---> {:d} ...".format( i ))
    else:
       print("--- Cube root ({:7.1f}) ---> covergence failed ...")

    N = -27.0;
    root, i, converged = cuberoot(N, tolerance, maxiterations );
    if converged == True:
       print("--- Cube root ({:7.1f}) ---> {:16.6f} ...".format( N, root))
       print("---           converged ---> {:s} ...".format( str( converged ) ))
       print("---           tolerance ---> {:f} ...".format( tolerance ))
       print("---       no iterations ---> {:d} ...".format( i ))
    else:
       print("--- Cube root ({:7.1f}) ---> covergence failed ...")

    print("--- ======================================== ... ");
    print("--- Leave TestCubeRoot01.main()              ... ");

# call the main method ...

main()
```

Program Output: The textual output is:

```
--- Enter TestCubeRoot01.main()               ...
--- ======================================== ...

--- Cube root (    8.0) --->         2.000005 ...
---           converged ---> True ...
---           tolerance ---> 0.000100 ...
---       no iterations ---> 5 ...
--- Cube root ( 1000.0) --->        10.000000 ...
---           converged ---> True ...
---           tolerance ---> 0.000100 ...
---       no iterations ---> 13 ...
--- Cube root (   -8.0) --->        -2.000000 ...
---           converged ---> True ...
---           tolerance ---> 0.000100 ...
---       no iterations ---> 1 ...
--- Cube root (  -27.0) --->        -3.000127 ...
---           converged ---> True ...
---           tolerance ---> 0.000100 ...
---       no iterations ---> 6 ...

--- ======================================== ...
--- Leave TestCubeRoot01.main()               ...
```

Question 3: 10 points. The quartic function

$$f(x) = (x-3)^2(x-5)^2 \tag{6}$$

has double roots at x = 3 and x = 5. Write a Python program to plot f(x) over the range [2,6]. Demonstrate that while the method of **Newton Raphson** struggles to find value(s) of x for which f(x) = 0, the method of **Modified Newton Raphson** computes the same double roots with ease.

Python Source Code:

```
# ================================================================================
# TestNewtonRaphson02.py: Use newton raphson/modified newton raphson algorithms
# to compute roots of equations:
#
#            f(x) = (x-3)^2 (x-5)^2 = 0.0
#
# Written By: Mark Austin                                                April 2024
# ================================================================================

import math;
import Solutions;

import numpy as np
import matplotlib.pyplot as plt

# Mathematical functions: f(x) = (x-3)*(x-3)*(x-5)*(x-5) = 0.

def f1(x):
    return (x-3)*(x-3)*(x-5)*(x-5);

def df1(x):
    return 4*(x-3)*(x-5)*(x-4)

def ddf1(x):
    return 4*(3*x**2 - 24*x + 47);

# main method ...

def main():
    print("--- Enter TestNewtonRaphson02.main()          ... ");
    print("--- ========================================= ... ");

    print("--- ");
    print("--- Plot f(x) over range [2,6] ... ");
    print("--- ========================================= ... ");

    # Generate x and y coordinate arrays ....

    xcoords = np.linspace(2,6,51,endpoint=True);
    ycoords = f1(xcoords);
```

```
# Plot f(x) ...

plt.plot( xcoords, ycoords )
plt.title('f(x) = (x-3)^2 (x-5)^2');
plt.xlabel('x');
plt.ylabel('f(x)');
plt.grid(True)
plt.show()

print("---");

print("--- ");
print("--- Case Study 1: Solve (x-3)^2 (x-5)^2 = 0, Initial guess: x0 = 10 ... ");
print("--- ================================================================ ... ");

# Initialize problem setup ...

x0 = 10.0;
tolerance     = 0.00000001
maxiterations = 100

print("--- Inputs:")
print("---   x0 = {:5.2f} ...".format(x0) )
print("---   tolerance      = {:16.8f} ...".format(tolerance) )
print("---   max iterations = {:16.8f} ...".format(maxiterations) )

# Compute roots to equation ...

print("--- Execution:")
root, i, converged = Solutions.newtonraphson(f1, df1, x0, tolerance, maxiterations )

# Summary of computations ...

print("--- Output:")
print("---   root = {:10.5f} ...".format(root) )
print("---   f(root)   --> {:16.8e} ...".format(   f1(root)) )
print("---   df(root)  --> {:16.8e} ...".format(  df1(root)) )
print("---   ddf(root) --> {:16.8e} ...".format( ddf1(root)) )
print("---   no iterations = {:d} ...".format(i) )
print("---   converged: {:s} ...".format( str(converged) ) )

print("--- ");
print("--- Case Study 2: Use Solve (x-3)^2 (x-5)^2 = 0, Initial guess: x0 = 10 ... ");
print("--- ================================================================== ... ");

print("--- Inputs:")
print("---   x0 = {:5.2f} ...".format(x0) )
print("---   tolerance      = {:8.5f} ...".format(tolerance) )
print("---   max iterations = {:8.2f} ...".format(maxiterations) )

# Compute roots to equation ...

print("--- Execution:")
root, i, converged = Solutions.modifiednewtonraphson(f1, df1, ddf1, x0, tolerance, maxiterations
```

```
# Summary of computations ...

print("--- Output:")
print("---   root = {:10.5f} ...".format(root) )
print("---   f(root)   --> {:16.8e} ...".format(   f1(root)) )
print("---   df(root)  --> {:16.8e} ...".format(  df1(root)) )
print("---   ddf(root) --> {:16.8e} ...".format( ddf1(root)) )
print("---   no iterations = {:d} ...".format(i) )
print("---   converged: {:s} ...".format( str(converged) ) )

print("--- ");
print("--- Case Study 3: Solve (x-3)^2 (x-5)^2 = 0, Initial guess: x0 = 0.0 ... ");
print("--- ================================================================ ... ");

# Initialize problem setup ...

x0 = 0.0;
tolerance     = 0.00000001
maxiterations = 100

print("--- Inputs:")
print("---   x0 = {:5.2f} ...".format(x0) )
print("---   tolerance      = {:16.8f} ...".format(tolerance) )
print("---   max iterations = {:16.8f} ...".format(maxiterations) )

# Compute roots to equation ...

print("--- Execution:")
root, i, converged = Solutions.newtonraphson(f1, df1, x0, tolerance, maxiterations )

# Summary of computations ...

print("--- Output:")
print("---   root = {:10.5f} ...".format(root) )
print("---   f(root)   --> {:16.8e} ...".format(   f1(root)) )
print("---   df(root)  --> {:16.8e} ...".format(  df1(root)) )
print("---   ddf(root) --> {:16.8e} ...".format( ddf1(root)) )
print("---   no iterations = {:d} ...".format(i) )
print("---   converged: {:s} ...".format( str(converged) ) )

print("--- ");
print("--- Case Study 4: Use Solve (x-3)^2 (x-5)^2 = 0, Initial guess: x0 = 0.0 ... ");
print("--- ================================================================ ... ");

print("--- Inputs:")
print("---   x0 = {:5.2f} ...".format(x0) )
print("---   tolerance      = {:8.5f} ...".format(tolerance) )
print("---   max iterations = {:8.2f} ...".format(maxiterations) )

# Compute roots to equation ...

print("--- Execution:")
root, i, converged = Solutions.modifiednewtonraphson(f1, df1, ddf1, x0, tolerance, maxiterations

# Summary of computations ...
```

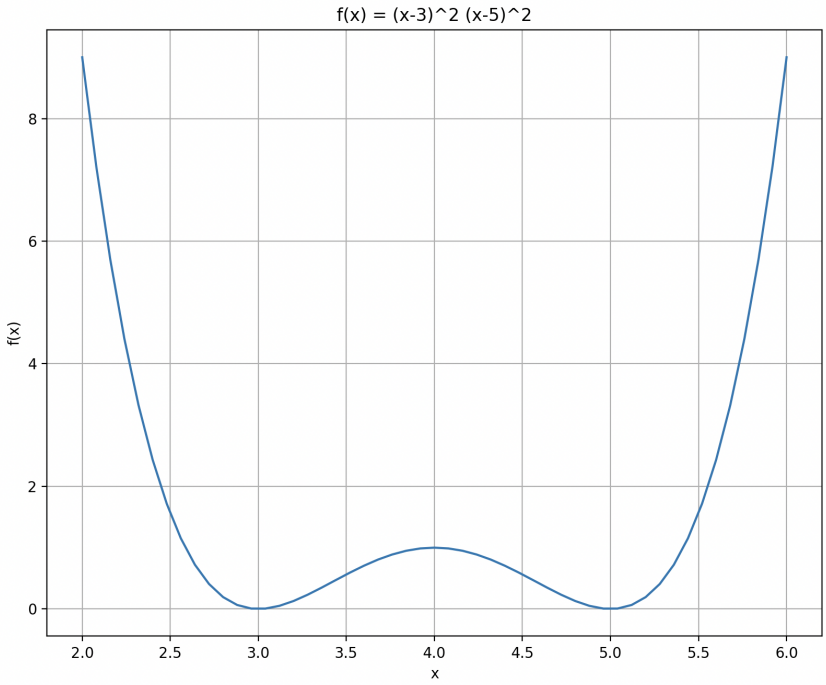
```
    print("--- Output:")
    print("---   root = {:10.5f} ...".format(root) )
    print("---   f(root)   --> {:16.8e} ...".format(   f1(root)) )
    print("---   df(root)  --> {:16.8e} ...".format(  df1(root)) )
    print("---   ddf(root) --> {:16.8e} ...".format( ddf1(root)) )
    print("---   no iterations = {:d} ...".format(i) )
    print("---   converged: {:s} ...".format( str(converged) ) )

    print("--- ======================================== ... ");
    print("--- Leave TestNewtonRaphson02.main()          ... ");

# call the main method ...

main()
```

Program Output: The graphical output is:



The abbreviated textual output is:

```
--- Enter TestNewtonRaphson02.main()          ...
--- ======================================== ...
---
--- Plot f(x) over range [2,6] ...
--- ======================================== ...
---
```

```
--- Case Study 1: Solve (x-3)^2 (x-5)^2 = 0, Initial guess: x0 = 10 ...
--- ============================================================ ...
--- Inputs:
---   x0 = 10.00 ...
---   tolerance      =       0.00000001 ...
---   max iterations =     100.00000000 ...
--- Execution:
---   Initial Conditions:
---     x0     -->  1.00000e+01 ...
---     f(x0)  -->  1.22500e+03 ...
---     df(x0) -->  8.40000e+02 ...
---   Main Loop for Newton Raphson Iteration:
---   Iteration 001: dx = -1.45833e+00, x =  8.54167e+00, f(x) -->  3.85209e+02 ...
---   Iteration 002: dx = -1.08037e+00, x =  7.46130e+00, f(x) -->  1.20573e+02 ...

... lines of output removed ...

---   Iteration 031: dx = -1.76165e-08, x =  5.00000e+00, f(x) -->  1.24137e-15 ...
---   Iteration 032: dx = -8.80827e-09, x =  5.00000e+00, f(x) -->  3.10343e-16 ...
--- Output:
---   root =    5.00000 ...
---   f(root)   -->   3.10342711e-16 ...
---   df(root)  -->   7.04661867e-08 ...
---   ddf(root) -->   8.00000021e+00 ...
---   no iterations = 32 ...
---   converged: True ...
---
--- Case Study 2: Use Solve (x-3)^2 (x-5)^2 = 0, Initial guess: x0 = 10 ...
--- ================================================================== ...
--- Inputs:
---   x0 = 10.00 ...
---   tolerance      =  0.00000 ...
---   max iterations =   100.00 ...
--- Execution:
---   Initial Conditions:
---     x0     -->  1.00000e+01 ...
---     f(x0)  -->  1.22500e+03 ...
---   Main Loop for Modified Newton Raphson Iteration:
---   Iteration 001: dx = -5.67568e+00, x =  4.32432e+00, f(x) -->  8.00692e-01 ...
---   Iteration 002: dx =  2.62589e-01, x =  4.58691e+00, f(x) -->  4.29723e-01 ...
---   Iteration 003: dx =  2.86166e-01, x =  4.87308e+00, f(x) -->  5.65166e-02 ...
---   Iteration 004: dx =  1.17780e-01, x =  4.99086e+00, f(x) -->  3.31181e-04 ...
---   Iteration 005: dx =  9.09880e-03, x =  4.99996e+00, f(x) -->  7.11032e-09 ...
---   Iteration 006: dx =  4.21614e-05, x =  5.00000e+00, f(x) -->  3.16032e-18 ...
---   Iteration 007: dx =  8.88865e-10, x =  5.00000e+00, f(x) -->  0.00000e+00 ...
--- Output:
---   root =    5.00000 ...
---   f(root)   -->   0.00000000e+00 ...
---   df(root)  -->   0.00000000e+00 ...
---   ddf(root) -->   8.00000000e+00 ...
---   no iterations = 7 ...
---   converged: True ...
---
--- Case Study 3: Solve (x-3)^2 (x-5)^2 = 0, Initial guess: x0 = 0.0 ...
--- ============================================================ ...
```

```
--- Inputs:
---   x0 =  0.00 ...
---   tolerance      =       0.00000001 ...
---   max iterations =     100.00000000 ...
--- Execution:
---   Initial Conditions:
---     x0     -->  0.00000e+00 ...
---     f(x0)  -->  2.25000e+02 ...
---     df(x0) --> -2.40000e+02 ...
---   Main Loop for Newton Raphson Iteration:
---   Iteration 001: dx =  9.37500e-01, x =  9.37500e-01, f(x) -->  7.02061e+01 ...
---   Iteration 002: dx =  6.83992e-01, x =  1.62149e+00, f(x) -->  2.16904e+01 ...

... lines of output removed ...

---   Iteration 030: dx =  1.27756e-08, x =  3.00000e+00, f(x) -->  6.52862e-16 ...
---   Iteration 031: dx =  6.38779e-09, x =  3.00000e+00, f(x) -->  1.63215e-16 ...
--- Output:
---   root =    3.00000 ...
---   f(root)   -->   1.63215438e-16 ...
---   df(root)  -->  -5.11023193e-08 ...
---   ddf(root) -->   8.00000015e+00 ...
---   no iterations = 31 ...
---   converged: True ...
---
--- Case Study 4: Use Solve (x-3)^2 (x-5)^2 = 0, Initial guess: x0 = 0.0 ...
--- ================================================================== ...
--- Inputs:
---   x0 =  0.00 ...
---   tolerance      =  0.00000 ...
---   max iterations =   100.00 ...
--- Execution:
---   Initial Conditions:
---     x0     -->  0.00000e+00 ...
---     f(x0)  -->  2.25000e+02 ...
---   Main Loop for Modified Newton Raphson Iteration:
---   Iteration 001: dx =  3.52941e+00, x =  3.52941e+00, f(x) -->  6.06135e-01 ...
---   Iteration 002: dx = -2.99950e-01, x =  3.22946e+00, f(x) -->  1.65056e-01 ...
---   Iteration 003: dx = -1.96424e-01, x =  3.03304e+00, f(x) -->  4.22284e-03 ...
---   Iteration 004: dx = -3.24734e-02, x =  3.00056e+00, f(x) -->  1.27195e-06 ...
---   Iteration 005: dx = -5.63904e-04, x =  3.00000e+00, f(x) -->  1.01344e-13 ...
---   Iteration 006: dx = -1.59173e-07, x =  3.00000e+00, f(x) -->  6.63432e-28 ...
---   Iteration 007: dx = -1.28786e-14, x =  3.00000e+00, f(x) -->  0.00000e+00 ...
--- Output:
---   root =    3.00000 ...
---   f(root)   -->   0.00000000e+00 ...
---   df(root)  -->   0.00000000e+00 ...
---   ddf(root) -->   8.00000000e+00 ...
---   no iterations = 7 ...
---   converged: True ...

--- ======================================== ...
--- Leave TestNewtonRaphson02.main()          ...
```

Question 4: 10 points. Consider the family of matrix equations AX = B defined by

$$\begin{bmatrix} 1 & 0 & 1 \\ a & 8 & 4 \\ 0 & 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ b \end{bmatrix}. \tag{7}$$

Determine the values of 'a' for which matrix A will be singular. Then,

1. Develop a program that uses NumPy to store matrices A and B, and then systematically evaluates the determinant and rank of A, and the rank of augmented matrix [A | B] for values (a,b) corresponding to: (a) zero solutions, (b) infinite solutions.

2. Develop a second program that uses SymPy to store matrices A and B symbolically, and then computes symbolic solution to the matrix equations **??**. For the values of (a,b) that make A singular, evaluate the determinant and rank of A, and the rank of augmented matrix [A | B].

You should find that the Python module SymPy is considerably more powerful than its numerical counterpart NumPy.

Hand Calculation: We begin with:

$$det(A) = \begin{bmatrix} 1 & 0 & 1 \\ a & 8 & 4 \\ 0 & 1 & 2 \end{bmatrix} = det \begin{bmatrix} 8 & 4 \\ 1 & 2 \end{bmatrix} + det \begin{bmatrix} a & 8 \\ 0 & 1 \end{bmatrix} = 12 + a. \tag{8}$$

Setting det(A) = $12 + a$ = 0 gives a = -12. Putting equations **??** in augmented form, and applying row operations gives:

$$\begin{bmatrix} 1 & 0 & 1 & 1 \\ -12 & 8 & 4 & 2 \\ 0 & 1 & 2 & b \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 2 & 14/8 \\ 0 & 0 & 0 & b - 14/8 \end{bmatrix}. \tag{9}$$

Infinite solutions occur when b = $14/8$ and rank(A) = rank(A—b) = 2. Otherwise, we have zero solutions.

Part 1: Python Source Code (with Numpy):

```
# ============================================================
# TestMatrixOperationsNumPy02.py: Compute symbolic solutions to
# linear matrix equations ...
#
# ============================================================
```

```
import numpy as np
from numpy.linalg import matrix_rank

# ================================================================
# Function to print two-dimensional matrices ...
# ================================================================

def PrintMatrix(name, a):
    print("Matrix: {:s} ".format(name) );
    for row in a:
        for col in row:
            print("{:8.2f}".format(col), end=" ")
        print("")

# ================================================================
# main method ...
# ================================================================

def main():
    print("--- Enter TestMatrixOperationsNumPy02.main() ... ");
    print("--- ======================================== ... ");

    print("");
    print("--- Set a = -12, b = 14/8 in matrices A and B ...");
    print("--- ---------------------------------------- ...\n");

    a =  -12
    b = 14/8
    A = np.array( [ [ 1,  0,  1],
                    [ a,  8,  4],
                    [ 0,  1,  2 ] ])
    B = np.array( [ [1], [2], [b] ])

    PrintMatrix("A", A)
    PrintMatrix("B", B)

    print("");
    print("--- determinant(A)  --> {:f} ...".format( np.linalg.det(A) ))
    print("--- rank [A]        --> {:f} ...".format( matrix_rank(A) ))

    print("");
    print("--- Create augmented matrix [ A | B ] ...\n");

    C = np.concatenate((A, B), axis=1)
    PrintMatrix("Matrix [A|B]", C)

    print("");
    print("--- rank [A|B]      --> {:f} ...".format( matrix_rank(C) ))

    print("--- ======================================== ... ");
    print("--- Leave TestMatrixOperationsNumPy01.main() ... ");

# call the main method ...
```

```
main()
```

Program Output The textual output is:

```
--- Enter TestMatrixOperationsNumPy02.main() ...
--- ======================================== ...

--- Set a = -12, b = 14/8 in matrices A and B ...
--- ---------------------------------------- ...

Matrix: A
    1.00     0.00     1.00
  -12.00     8.00     4.00
    0.00     1.00     2.00
Matrix: B
    1.00
    2.00
    1.75

--- determinant(A)  --> 0.000000 ...
--- rank [A]        --> 2.000000 ...

--- Create augmented matrix [ A | B ] ...

Matrix: Matrix [A|B]
    1.00     0.00     1.00     1.00
  -12.00     8.00     4.00     2.00
    0.00     1.00     2.00     1.75

--- rank [A|B]      --> 2.000000 ...

--- ======================================== ...
--- Leave TestMatrixOperationsNumPy01.main() ...
```

Part 2: Python Source Code (with SymPy):

```
# ============================================================
# TestMatrixOperationsSympy02.py: Compute symbolic solutions to
# linear matrix equations ...
#
# ============================================================

import sympy as sp
from sympy import Integral, Matrix, pi, pprint

# main method ...

def main():
    print("--- Enter TestMatrixOperationsSympy02.main() ... ");
    print("--- ======================================== ... ");
```

```
# Define symbolic representation of matrices ...

print("--- Create matrices A and B ...");

a = sp.symbols('a')
b = sp.symbols('b')
A = sp.Matrix(( [ 1,  0,  1],
                [ a,  8,  4],
                [ 0,  1,  2] ))
B = sp.Matrix(( [1], [2], [b] ))

pprint(A)
pprint(B)

print("--- Compute and print matrix determinant ...\n");

print("--- A.det() --> {:s} ...".format( str(A.det() ) ))

print("\n--- General symbolic solution to matrix system ...\n");

solution = A.solve(B)
print(solution)

print("\n--- Expressions for individual solution elements ...\n");
print("--- x0 = {:s} ...".format( str( solution[0] ) ))
print("--- x1 = {:s} ...".format( str( solution[1] ) ))
print("--- x2 = {:s} ...".format( str( solution[2] ) ))

print("");
print("--- Create augmented matrix [ A | B ] ...\n");

C = A.row_join(B)
pprint(C)

print("");
print("--- Set a = -12 in matrix [A|B] ...");
print("--- ---------------------------------- ...\n");

A1 = A.subs( {a:-12} )
C1 = C.subs( {a:-12} )
pprint(C1)

print("");
print("--- A1.det()  --> {:s} ...".format( str( A1.det() ) ))
print("--- A1.rank() --> {:s} ...".format( str( A1.rank() ) ))
print("--- C1.rank() --> {:s} ...".format( str( C1.rank() ) ))

print("");
print("--- Set a = -12, b = 14/8 in matrix [A|B] ...");
print("--- ------------------------------------- ...\n");

A2 = A.subs( {a:-12} )
C2 = C.subs( {a:-12, b:14/8} )
pprint(C2)
```

```
    print("");
    print("--- A2.det()  --> {:s} ...".format( str( A2.det() ) ))
    print("--- A2.rank() --> {:s} ...".format( str( A2.rank() ) ))
    print("--- C2.rank() --> {:s} ...".format( str( C2.rank() ) ))

    print("--- ======================================== ... ");
    print("--- Leave TestMatrixOperationsSympy02.main() ... ");

# call the main method ...

main()
```

Program Output The textual output is:

```
--- Enter TestMatrixOperationsSympy02.main() ...
--- ======================================== ...

--- Create matrices A and B ...

[ 1  0  1 ]
[         ]
[ a  8  4 ]
[         ]
[ 0  1  2 ]
[ 1 ]
[   ]
[ 2 ]
[   ]
[ b ]
--- Compute and print matrix determinant ...

--- A.det() --> a + 12 ...

--- General symbolic solution to matrix system ...

Matrix([[(14 - 8*b)/(a + 12)], [(a*b - 2*a - 4*b + 4)/(a + 12)], [(a + 8*b - 2)/(a + 12)]])

--- Expressions for individual solution elements ...

--- x0 = (14 - 8*b)/(a + 12) ...
--- x1 = (a*b - 2*a - 4*b + 4)/(a + 12) ...
--- x2 = (a + 8*b - 2)/(a + 12) ...

--- Create augmented matrix [ A | B ] ...

[ 1  0  1  1 ]
[            ]
[ a  8  4  2 ]
[            ]
[ 0  1  2  b ]

--- Set a = -12 in matrix [A|B] ...
```

```
--- -------------------------------- ...

[   1  0  1  1 ]
[              ]
[ -12  8  4  2 ]
[              ]
[   0  1  2  b ]

--- A1.det()  --> 0 ...
--- A1.rank() --> 2 ...
--- C1.rank() --> 3 ...

--- Set a = -12, b = 14/8 in matrix [A|B] ...
--- ------------------------------------ ...

[   1  0  1   1   ]
[                 ]
[ -12  8  4   2   ]
[                 ]
[   0  1  2  1.75 ]

--- A2.det()  --> 0 ...
--- A2.rank() --> 2 ...
--- C2.rank() --> 2 ...

--- ======================================== ...
--- Leave TestMatrixOperationsSympy02.main() ...
```

Question 5: 10 points. In the design of crane structures, engineers are often required to compute the maximum and minimum member forces and support reactions due to a variety of loading conditions.

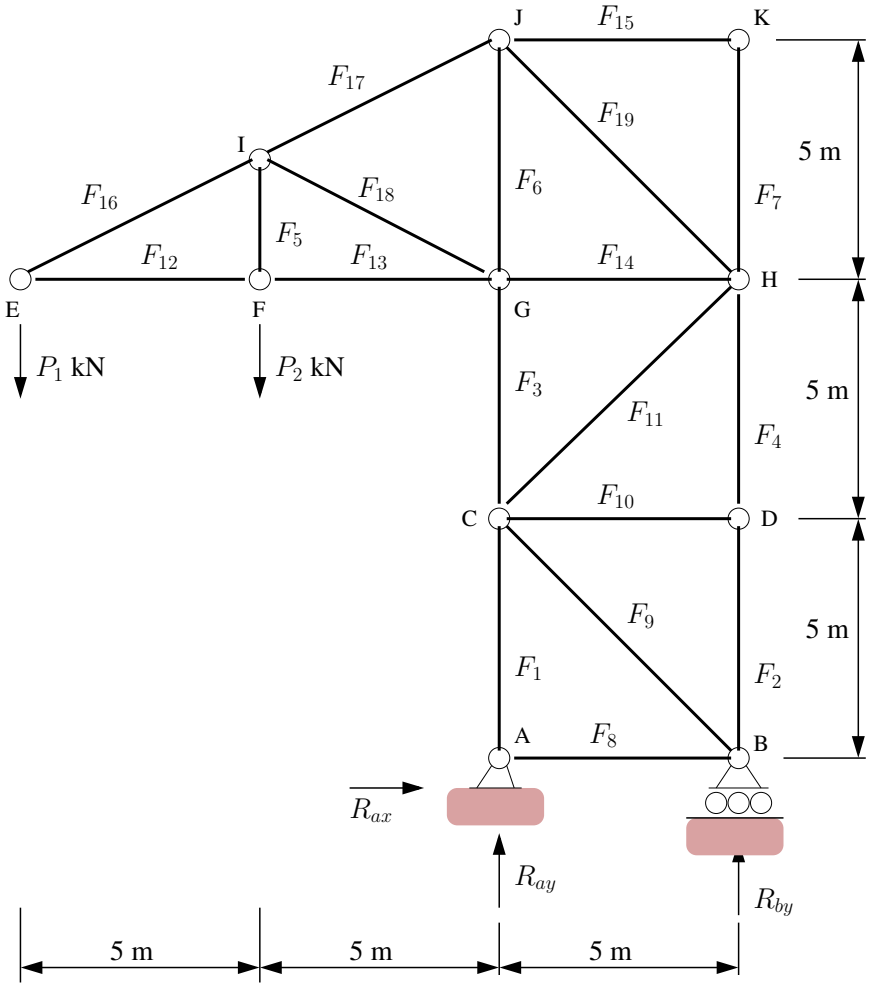


Figure 3: Front elevation of crane tower.

Figure 3 shows a nineteen bar pin-jointed truss carrying vertical loads P_1 kN and P_2 kN at joints `E` and `F`. The symbols F_1, F_2, $\cdots$ F_{19} represent the axial forces in truss members 1 through 19, and R_{ax},R_{ay}, and R_{dy} are the horizontal and vertical support reactions at joints `A` and `B`.

Write down the equations of equilibrium for joints `A` through `K` and put the equations in matrix form. Now suppose that the crane supports a variety of payloads that, for engineering purposes, it can be represented by the sequence of external load vectors

$$\begin{bmatrix} P_1 \\ P_2 \end{bmatrix} = \begin{bmatrix} 15 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} P_1 \\ P_2 \end{bmatrix} = \begin{bmatrix} 10 \\ 10 \end{bmatrix}, \quad \begin{bmatrix} P_1 \\ P_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 25 \end{bmatrix} kN. \tag{10}$$

Develop a Python program that will solve the matrix equations for each of the external load conditions, and compute and print the minimum and maximum support reactions at nodes A and B, and axial forces in each of the truss members.

Equations of Equilibrium:

At Joint A:

$$\sum F_x = 0, \quad R_{ax} + F_8 = 0. \tag{11}$$

$$\sum F_y = 0, \quad R_{ay} + F_1 = 0. \tag{12}$$

At Joint B:

$$\sum F_x = 0, \quad F_8 + \frac{1}{\sqrt{2}} F_9 = 0. \tag{13}$$

$$\sum F_y = 0, \quad F_2 + \frac{1}{\sqrt{2}} F_9 + R_{by} = 0. \tag{14}$$

At Joint C:

$$\sum F_x = 0, \quad \frac{1}{\sqrt{2}} F_9 + F_{10} + \frac{1}{\sqrt{2}} F_{11} = 0. \tag{15}$$

$$\sum F_y = 0, \quad F_1 - F_3 + \frac{1}{\sqrt{2}} F_9 - \frac{1}{\sqrt{2}} F_{11} = 0. \tag{16}$$

At Joint D:

$$\sum F_x = 0, \quad F_{10} = 0. \tag{17}$$

$$\sum F_y = 0, \quad F_2 - F_4 = 0. \tag{18}$$

At Joint E:

$$\sum F_x = 0, \quad F_{12} + \frac{2}{\sqrt{5}} F_{16} = 0. \tag{19}$$

$$\sum F_y = 0, \quad \frac{1}{\sqrt{5}}F_{16} + P_1 = 0. \tag{20}$$

At Joint F:

$$\sum F_x = 0, \quad F_{12} - F_{13} = 0. \tag{21}$$

$$\sum F_y = 0, \quad F_5 - P_2 = 0. \tag{22}$$

At Joint G:

$$\sum F_x = 0, \quad F_{13} - F_{14} + \frac{2}{\sqrt{5}}F_{18} = 0. \tag{23}$$

$$\sum F_y = 0, \quad -F_3 + F_6 + \frac{1}{\sqrt{5}}F_{18} = 0. \tag{24}$$

At Joint H:

$$\sum F_x = 0, \quad \frac{1}{\sqrt{2}}F_{11} + F_{14} + \frac{1}{\sqrt{2}}F_{19} = 0. \tag{25}$$

$$\sum F_y = 0, \quad F_4 - F_7 + \frac{1}{\sqrt{2}}F_{11} - \frac{1}{\sqrt{2}}F_{19} = 0. \tag{26}$$

At Joint I:

$$\sum F_x = 0, \quad \frac{2}{\sqrt{5}}F_{16} - \frac{2}{\sqrt{5}}F_{17} - \frac{2}{\sqrt{5}}F_{18} = 0. \tag{27}$$

$$\sum F_y = 0, \quad F_5 + \frac{1}{\sqrt{5}}F_{16} - \frac{1}{\sqrt{5}}F_{17} + \frac{1}{\sqrt{5}}F_{18} = 0. \tag{28}$$

At Joint J:

$$\sum F_x = 0, \quad F_{15} - \frac{2}{\sqrt{5}}F_{17} + \frac{1}{\sqrt{2}}F_{19} = 0. \tag{29}$$

$$\sum F_y = 0, \quad F_6 + \frac{1}{\sqrt{5}} F_{17} + \frac{1}{\sqrt{2}} F_{19} = 0 \tag{30}$$

At Joint K:

$$\sum F_x = 0, \quad F_{15} = 0. \tag{31}$$

$$\sum F_y = 0, \quad F_7 = 0. \tag{32}$$

System of Matrix Equations: Collecting equations 11 through 32 and writing in matrix form: ...

Python Source Code:

```
# ============================================================
# TestTrussAnalysis04.py: Compute distribution of element forces
# and support reactions in a 19-bar crane truss.
#
# Written by: Mark Austin                                April 2024
# ============================================================

import math
import numpy as np
from numpy.linalg import matrix_rank

# ============================================================
# Function to print one- and two-dimensional matrices ...
# ============================================================

def PrintMatrix(name, matrix):
    NoColumns = 6;

    # Compute no of blocks of rows to be printed .....

    if matrix.ndim == 1:
       noMatrixRows = matrix.shape[0]
       noMatrixCols = 1

    if matrix.ndim == 2:
       noMatrixRows = matrix.shape[0]
       noMatrixCols = matrix.shape[1]

    #  Compute number of blocks to be printed ...

    if noMatrixCols % NoColumns == 0:
       iNoBlocks = noMatrixCols/NoColumns;
    else:
       iNoBlocks = noMatrixCols/NoColumns + 1;
```

```
    #  Loop over the number of blocks ...

    for ib in range( int(iNoBlocks) ):
        iFirstColumn = ib*NoColumns + 1
        iLastColumn  = min ( (ib+1)*NoColumns, noMatrixCols )

        # Print title of matrix at top of each block ....

        print("Matrix: {:s} ".format(name) );

        # Label row and column nos */

        print("row/col     ", end="")
        colList = range(iFirstColumn, iLastColumn + 1)
        for col in [ *colList ]:
            print("       {:3d}    ".format(col),end="")
        print("")

        # Loop over rows and print matrix elements ....

        ii = 1
        for row in matrix:
            print(" {:3d}     ".format(ii),end="")
            colList = range( iFirstColumn, iLastColumn + 1)
            for col in [ *colList ]:
                if matrix.ndim == 1:
                   print(" {:12.5e} ".format( matrix[ii-1] ), end="")
                else:
                   print(" {:12.5e} ".format(matrix[ii-1][col-1]), end="")
            print("")
            ii = ii + 1
        print("")

# ================================================================
# Compute maximum and minumun of three numbers ...
# ================================================================

def maximum(a, b, c):
    list = [a, b, c]
    return max(list)

def minimum(a, b, c):
    list = [a, b, c]
    return min(list)

# ================================================================
# Print element forces ...
# ================================================================

def printElementForces(name, minF, maxF):
    if( minF < 0):
        print("---     Minimum {:s} = {:7.2f} (C) ... ".format( name, minF ) )
    else:
        print("---     Minimum {:s} = {:7.2f} (T) ... ".format( name, minF ) )
```

```
    if( maxF < 0):
        print("---     Maximum {:s} = {:7.2f} (C) ... ".format( name, maxF ) )
    else:
        print("---     Maximum {:s} = {:7.2f} (T) ... ".format( name, maxF ) )

# ============================================================
# main method ...
# ============================================================

def main():
    print("--- Enter TestTrussAnalysis04.main()         ... ");
    print("--- ======================================== ... ");

    print("--- ");
    print("--- Part 1: Initialize coefficients for matrix equations ... ");

    nonodes  = 11;  # <--- no of nodes in crane structure ...
    maxterms =  4;  # <--- max no terms in an equation of equilibrium ...

    # Equilibrium and destination arrays ...

    equilibrium = np.zeros(( 2*nonodes, maxterms ))
    destination = np.zeros(( 2*nonodes, maxterms ))

    PrintMatrix("Equilibrium", equilibrium );
    PrintMatrix("Destination", destination );

    # Node A ...

    equilibrium[0][0] = 1   # < --- equilibrium in x direction ...
    equilibrium[0][1] = 1

    destination[0][0] =  8;
    destination[0][1] = 20;

    equilibrium[1][0] = 1   # < --- equilibrium in y direction ...
    equilibrium[1][1] = 1

    destination[1][0] =  1;
    destination[1][1] = 21;

    # Node B ...

    equilibrium[2][0] = 1    # < --- equilibrium in x direction ...
    equilibrium[2][1] = 1/math.sqrt(2)

    destination[2][0] = 8;
    destination[2][1] = 9;

    equilibrium[3][0] = 1    # < --- equilibrium in y direction ...
    equilibrium[3][1] = 1/math.sqrt(2)
    equilibrium[3][2] = 1

    destination[3][0] =  2;
```

```
destination[3][1] =  9;
destination[3][2] = 22;

# Node C ...

equilibrium[4][0] = 1/math.sqrt(2) # <--- equilibrium in x direction ...
equilibrium[4][1] = 1
equilibrium[4][2] = 1/math.sqrt(2)

destination[4][0] =  9;
destination[4][1] = 10;
destination[4][2] = 11;

equilibrium[5][0] =  1              # <--- equilibrium in y direction ...
equilibrium[5][1] = -1
equilibrium[5][2] =  1/math.sqrt(2)
equilibrium[5][3] = -1/math.sqrt(2)

destination[5][0] =  1;
destination[5][1] =  3;
destination[5][2] =  9;
destination[5][3] = 11;

# Node D ...

equilibrium[6][0] =  1; # <--- equilibrium in x direction ...
destination[6][0] = 10;

equilibrium[7][0] =  1; # <--- equilibrium in y direction ...
equilibrium[7][1] = -1;

destination[7][0] = 2;
destination[7][1] = 4;

# Node E ...

equilibrium[8][0] = 1                 # <--- equilibrium in x direction ...
equilibrium[8][1] = 2/math.sqrt(5)

destination[8][0] = 12;
destination[8][1] = 16;

equilibrium[9][0] = -1/math.sqrt(5)   # <--- equilibrium in y direction ...
destination[9][0] = 16;

# Node F ...

equilibrium[10][0] =  1     # <--- equilibrium in x direction ...
equilibrium[10][1] = -1

destination[10][0] = 12;
destination[10][1] = 13;

equilibrium[11][0] = -1     # <--- equilibrium in y direction ...
destination[11][0] =  5;
```

```
# Node G ...

equilibrium[12][0] =  1     # <--- equilibrium in x direction ...
equilibrium[12][1] = -1
equilibrium[12][2] =  2/math.sqrt(5)

destination[12][0] = 13;
destination[12][1] = 14;
destination[12][2] = 18;

equilibrium[13][0] = -1;     # <--- equilibrium in y direction ...
equilibrium[13][1] =  1;
equilibrium[13][2] =  1/math.sqrt(5);

destination[13][0] =  3;
destination[13][1] =  6;
destination[13][2] = 18;

# Node H ...

equilibrium[14][0] =  1/math.sqrt(2);  # <--- equilibrium in x direction ...
equilibrium[14][1] =  1;
equilibrium[14][2] =  1/math.sqrt(2);

destination[14][0] = 11;
destination[14][1] = 14;
destination[14][2] = 19;

equilibrium[15][0] =   1               # <--- equilibrium in y direction ...
equilibrium[15][1] =   1/math.sqrt(2);
equilibrium[15][2] =  -1
equilibrium[15][3] =  -1/math.sqrt(2);

destination[15][0] =   4;
destination[15][1] =  11;
destination[15][2] =   7;
destination[15][3] =  19;

# Node I ...

equilibrium[16][0] =  2/math.sqrt(5);  # <--- equilibrium in x direction ...
equilibrium[16][1] = -2/math.sqrt(5);
equilibrium[16][2] = -2/math.sqrt(5);

destination[16][0] = 16;
destination[16][1] = 17;
destination[16][2] = 18;

equilibrium[17][0] =  1;               # <--- equilibrium in y direction ...
equilibrium[17][1] =  1/math.sqrt(5);
equilibrium[17][2] = -1/math.sqrt(5);
equilibrium[17][3] =  1/math.sqrt(5);

destination[17][0] =  5;
```

```
destination[17][1] = 16;
destination[17][2] = 17;
destination[17][3] = 18;

# Node J ...

equilibrium[18][0] =  1;               # <--- equilibrium in x direction ...
equilibrium[18][1] = -2/math.sqrt(5);
equilibrium[18][2] =  1/math.sqrt(2);

destination[18][0] = 15;
destination[18][1] = 17;
destination[18][2] = 19;

equilibrium[19][0] =  1;               # <--- equilibrium in y direction ...
equilibrium[19][1] =  1/math.sqrt(5);
equilibrium[19][2] =  1/math.sqrt(2);

destination[19][0] =  6;
destination[19][1] = 17;
destination[19][2] = 19;

# Node K ...

equilibrium[20][0] =  1;               # <--- equilibrium in x direction ...
destination[20][0] = 15;

equilibrium[21][0] =  1;               # <--- equilibrium in y direction ...
destination[21][0] =  7;

PrintMatrix("Equilibrium", equilibrium );
PrintMatrix("Destination", destination );

print("--- ");
print("--- Part 2: Systematically Assemble Matrix A ... ");
print("--- ");

# Allocate memory for A matrix ...

A = np.zeros(( 2*nonodes, 2*nonodes ))

# Systematically assemble A matrix from equilibrium and destination arrays ...

for row in range(2*nonodes):
    print("--- ");
    for col in range(maxterms):
        eitem = equilibrium[row][col];
        ditem = int( destination[row][col] );
        if eitem != 0 or ditem != 0:
            print("--- (row,col) --> {:2d}, {:d}: eitem --> {:9.5f}, ditem --> {:3d} ...".format
            A[row][ditem-1] = eitem;

PrintMatrix("A", A);

print("--- ");
```

```
print("--- Part 2: Initialize 22x1 load vectors ... ");
print("--- ");

print("--- Load Case 1: Node e_y = -15 ...");

B1 = np.zeros(( 2*nonodes, 1 ))
B1[ 9][0] = -15.0;
B1[11][0] =   0.0;

PrintMatrix("Load Case 1:", B1);

print("--- Load Case 2: Node e_y = -10, node f_y = -10 ...");

B2 = np.zeros(( 2*nonodes, 1 ))
B2[ 9][0] = -10.0;
B2[11][0] = -10.0;

PrintMatrix("Load Case 2:", B2);

print("--- Load Case 5: Node f_y = -25 ...");

B3 = np.zeros(( 2*nonodes, 1 ))
B3[ 9][0] =   0.0;
B3[11][0] = -25.0;

PrintMatrix("Load Case 3:", B3);

print("--- ");
print("--- Part 4: Check properties of matrix A ... ");
print("--- ");

rank = matrix_rank(A)
det  = np.linalg.det(A)

print("--- Matrix A: rank = {:f}, det = {:f}  ...".format(rank, det) );

print("--- ");
print("--- Part 5: Solve A.F = B for three load cases ... ");
print("--- ");

F1 = np.linalg.solve(A, B1)
PrintMatrix("Load Case 1: Forces ...", F1);

F2 = np.linalg.solve(A, B2)
PrintMatrix("Load Case 2: Forces ...", F2);

F3 = np.linalg.solve(A, B3)
PrintMatrix("Load Case 3: Forces ...", F3);

print("--- ");
print("--- Part 6: Print support reactions and element-level forces ... ");

print("--- ");
print("--- Support Reactions and Element-Level Forces: Load Case 1:");
print("--- ");
```

```
print("---   Reaction A: R_ax = {:7.2f} ... ".format( F1[19][0] ) );
print("---              : R_ay = {:7.2f} ... ".format( F1[20][0] ) );
print("---   Reaction D: R_dy = {:7.2f} ... ".format( F1[21][0] ) );
print("--- ");
print("--- Element Level Forces:");
print("--- ");
print("---   Element A-C: F1  = {:7.2f} ... ".format( F1[0][0] ) );
print("---   Element B-D: F2  = {:7.2f} ... ".format( F1[1][0] ) );
print("---   Element C-G: F3  = {:7.2f} ... ".format( F1[2][0] ) );
print("---   Element D-H: F4  = {:7.2f} ... ".format( F1[3][0] ) );
print("---   Element F-I: F5  = {:7.2f} ... ".format( F1[4][0] ) );
print("---   Element G-J: F6  = {:7.2f} ... ".format( F1[5][0] ) );
print("---   Element H-K: F7  = {:7.2f} ... ".format( F1[6][0] ) );
print("---   Element A-B: F8  = {:7.2f} ... ".format( F1[7][0] ) );
print("---   Element B-C: F09 = {:7.2f} ... ".format( F1[8][0] ) );
print("---   Element C-D: F10 = {:7.2f} ... ".format( F1[9][0] ) );
print("---   Element C-H: F11 = {:7.2f} ... ".format( F1[10][0] ) );
print("---   Element E-F: F12 = {:7.2f} ... ".format( F1[11][0] ) );
print("---   Element F-G: F13 = {:7.2f} ... ".format( F1[12][0] ) );
print("---   Element G-H: F14 = {:7.2f} ... ".format( F1[13][0] ) );
print("---   Element J-K: F15 = {:7.2f} ... ".format( F1[14][0] ) );
print("---   Element E-I: F16 = {:7.2f} ... ".format( F1[15][0] ) );
print("---   Element I-J: F17 = {:7.2f} ... ".format( F1[16][0] ) );
print("---   Element G-I: F18 = {:7.2f} ... ".format( F1[17][0] ) );
print("---   Element H-J: F19 = {:7.2f} ... ".format( F1[18][0] ) );

print("--- ");
print("--- Support Reactions and Element-Level Forces: Load Case 2:");
print("--- ");
print("---   Reaction A: R_ax = {:7.2f} ... ".format( F2[19][0] ) );
print("---              : R_ay = {:7.2f} ... ".format( F2[20][0] ) );
print("---   Reaction D: R_dy = {:7.2f} ... ".format( F2[21][0] ) );
print("--- ");
print("--- Element Level Forces:");
print("--- ");
print("---   Element A-C: F1  = {:7.2f} ... ".format( F2[0][0] ) );
print("---   Element B-D: F2  = {:7.2f} ... ".format( F2[1][0] ) );
print("---   Element C-G: F3  = {:7.2f} ... ".format( F2[2][0] ) );
print("---   Element D-H: F4  = {:7.2f} ... ".format( F2[3][0] ) );
print("---   Element F-I: F5  = {:7.2f} ... ".format( F2[4][0] ) );
print("---   Element G-J: F6  = {:7.2f} ... ".format( F2[5][0] ) );
print("---   Element H-K: F7  = {:7.2f} ... ".format( F2[6][0] ) );
print("---   Element A-B: F8  = {:7.2f} ... ".format( F2[7][0] ) );
print("---   Element B-C: F09 = {:7.2f} ... ".format( F2[8][0] ) );
print("---   Element C-D: F10 = {:7.2f} ... ".format( F2[9][0] ) );
print("---   Element C-H: F11 = {:7.2f} ... ".format( F2[10][0] ) );
print("---   Element E-F: F12 = {:7.2f} ... ".format( F2[11][0] ) );
print("---   Element F-G: F13 = {:7.2f} ... ".format( F2[12][0] ) );
print("---   Element G-H: F14 = {:7.2f} ... ".format( F2[13][0] ) );
print("---   Element J-K: F15 = {:7.2f} ... ".format( F2[14][0] ) );
print("---   Element E-I: F16 = {:7.2f} ... ".format( F2[15][0] ) );
print("---   Element I-J: F17 = {:7.2f} ... ".format( F2[16][0] ) );
print("---   Element G-I: F18 = {:7.2f} ... ".format( F2[17][0] ) );
print("---   Element H-J: F19 = {:7.2f} ... ".format( F2[18][0] ) );
```

```
print("--- ");
print("--- Support Reactions and Element-Level Forces: Load Case 3:");
print("--- ");
print("---   Reaction A: R_ax = {:7.2f} ... ".format( F3[19][0] ) );
print("---             : R_ay = {:7.2f} ... ".format( F3[20][0] ) );
print("---   Reaction D: R_dy = {:7.2f} ... ".format( F3[21][0] ) );
print("--- ");
print("--- Element Level Forces:");
print("--- ");
print("---   Element A-C: F1  = {:7.2f} ... ".format( F3[0][0] ) );
print("---   Element B-D: F2  = {:7.2f} ... ".format( F3[1][0] ) );
print("---   Element C-G: F3  = {:7.2f} ... ".format( F3[2][0] ) );
print("---   Element D-H: F4  = {:7.2f} ... ".format( F3[3][0] ) );
print("---   Element F-I: F5  = {:7.2f} ... ".format( F3[4][0] ) );
print("---   Element G-J: F6  = {:7.2f} ... ".format( F3[5][0] ) );
print("---   Element H-K: F7  = {:7.2f} ... ".format( F3[6][0] ) );
print("---   Element A-B: F8  = {:7.2f} ... ".format( F3[7][0] ) );
print("---   Element B-C: F09 = {:7.2f} ... ".format( F3[8][0] ) );
print("---   Element C-D: F10 = {:7.2f} ... ".format( F3[9][0] ) );
print("---   Element C-H: F11 = {:7.2f} ... ".format( F3[10][0] ) );
print("---   Element E-F: F12 = {:7.2f} ... ".format( F3[11][0] ) );
print("---   Element F-G: F13 = {:7.2f} ... ".format( F3[12][0] ) );
print("---   Element G-H: F14 = {:7.2f} ... ".format( F3[13][0] ) );
print("---   Element J-K: F15 = {:7.2f} ... ".format( F3[14][0] ) );
print("---   Element E-I: F16 = {:7.2f} ... ".format( F3[15][0] ) );
print("---   Element I-J: F17 = {:7.2f} ... ".format( F3[16][0] ) );
print("---   Element G-I: F18 = {:7.2f} ... ".format( F3[17][0] ) );
print("---   Element H-J: F19 = {:7.2f} ... ".format( F3[18][0] ) );

print("--- ");
print("--- Summary of Max/Min Reactions and Element-Level Forces ...");
print("--- ------------------------------------------------- ...");

MinRax = minimum( F1[19][0], F2[19][0], F3[19][0] )
MaxRax = maximum( F1[19][0], F2[19][0], F3[19][0] )

MinRay = minimum( F1[20][0], F2[20][0], F3[20][0] )
MaxRay = maximum( F1[20][0], F2[20][0], F3[20][0] )

MinRby = minimum( F1[21][0], F2[21][0], F3[21][0] )
MaxRby = maximum( F1[21][0], F2[21][0], F3[21][0] )

print("--- ");
print("---   Reaction A: Minimum R_ax = {:7.2f}, Maximum R_ax = {:7.2f} ... ".format( MinRax, Max
print("---             : Minimum R_ay = {:7.2f}, Maximum R_ay = {:7.2f} ... ".format( MinRay, Max
print("--- ");
print("---   Reaction B: Minimum R_by = {:7.2f}, Maximum R_by = {:7.2f} ... ".format( MinRby, Max

MinF1 = minimum( F1[0][0], F2[0][0], F3[0][0] )
MaxF1 = maximum( F1[0][0], F2[0][0], F3[0][0] )

MinF2 = minimum( F1[1][0], F2[1][0], F3[1][0] )
MaxF2 = maximum( F1[1][0], F2[1][0], F3[1][0] )

MinF3 = minimum( F1[2][0], F2[2][0], F3[2][0] )
```

```
MaxF3 = maximum( F1[2][0], F2[2][0], F3[2][0] )

MinF4 = minimum( F1[3][0], F2[3][0], F3[3][0] )
MaxF4 = maximum( F1[3][0], F2[3][0], F3[3][0] )

MinF5 = minimum( F1[4][0], F2[4][0], F3[4][0] )
MaxF5 = maximum( F1[4][0], F2[4][0], F3[4][0] )

MinF6 = minimum( F1[5][0], F2[5][0], F3[5][0] )
MaxF6 = maximum( F1[5][0], F2[5][0], F3[5][0] )

MinF7 = minimum( F1[6][0], F2[6][0], F3[6][0] )
MaxF7 = maximum( F1[6][0], F2[6][0], F3[6][0] )

MinF8 = minimum( F1[7][0], F2[7][0], F3[7][0] )
MaxF8 = maximum( F1[7][0], F2[7][0], F3[7][0] )

MinF9 = minimum( F1[8][0], F2[8][0], F3[8][0] )
MaxF9 = maximum( F1[8][0], F2[8][0], F3[8][0] )

MinF10 = minimum( F1[9][0], F2[9][0], F3[9][0] )
MaxF10 = maximum( F1[9][0], F2[9][0], F3[9][0] )

MinF11 = minimum( F1[10][0], F2[10][0], F3[10][0] )
MaxF11 = maximum( F1[10][0], F2[10][0], F3[10][0] )

MinF12 = minimum( F1[11][0], F2[11][0], F3[11][0] )
MaxF12 = maximum( F1[11][0], F2[11][0], F3[11][0] )

MinF13 = minimum( F1[12][0], F2[12][0], F3[12][0] )
MaxF13 = maximum( F1[12][0], F2[12][0], F3[12][0] )

MinF14 = minimum( F1[13][0], F2[13][0], F3[13][0] )
MaxF14 = maximum( F1[13][0], F2[13][0], F3[13][0] )

MinF15 = minimum( F1[14][0], F2[14][0], F3[14][0] )
MaxF15 = maximum( F1[14][0], F2[14][0], F3[14][0] )

MinF16 = minimum( F1[15][0], F2[15][0], F3[15][0] )
MaxF16 = maximum( F1[15][0], F2[15][0], F3[15][0] )

MinF17 = minimum( F1[16][0], F2[16][0], F3[16][0] )
MaxF17 = maximum( F1[16][0], F2[16][0], F3[16][0] )

MinF18 = minimum( F1[17][0], F2[17][0], F3[17][0] )
MaxF18 = maximum( F1[17][0], F2[17][0], F3[17][0] )

MinF19 = minimum( F1[18][0], F2[18][0], F3[18][0] )
MaxF19 = maximum( F1[18][0], F2[18][0], F3[18][0] )

print("--- ");
print("---   Element A-C: ")
printElementForces( "F1", MinF1, MaxF1)

print("---   Element B-D: ")
```

```
printElementForces( "F2", MinF2, MaxF2)

print("---   Element C-G: ")
printElementForces( "F3", MinF3, MaxF3)

print("---   Element D-H: ")
printElementForces( "F4", MinF4, MaxF4)

print("---   Element F-I: ")
printElementForces( "F5", MinF5, MaxF5)

print("---   Element G-J: ")
printElementForces( "F6", MinF6, MaxF6)

print("---   Element H-K: ")
printElementForces( "F7", MinF7, MaxF7)

print("---   Element A-B: ")
printElementForces( "F8", MinF8, MaxF8)

print("---   Element B-C: ")
printElementForces( "F9", MinF9, MaxF9)

print("---   Element C-D: ")
printElementForces( "F10", MinF10, MaxF10)

print("---   Element C-H: ")
printElementForces( "F11", MinF11, MaxF11)

print("---   Element E-F: ")
printElementForces( "F12", MinF12, MaxF12)

print("---   Element F-G: ")
printElementForces( "F13", MinF13, MaxF13)

print("---   Element G-H: ")
printElementForces( "F14", MinF14, MaxF14)

print("---   Element J-K: ")
printElementForces( "F15", MinF15, MaxF15)

print("---   Element E-I: ")
printElementForces( "F16", MinF16, MaxF16)

print("---   Element I-J: ")
printElementForces( "F17", MinF17, MaxF17)

print("---   Element G-I: ")
printElementForces( "F18", MinF18, MaxF18)

print("---   Element H-J: ")
printElementForces( "F19", MinF19, MaxF19)

print("--- ======================================== ... ");
print("--- Leave TestTrussAnalysis02.main()         ... ");
```

```
# call the main method ...

main()
```

Abbreviated Program Output:

```
--- Enter TestTrussAnalysis04.main()         ...
--- ======================================== ...
---
--- Part 1: Initialize coefficients for matrix equations ...

Matrix: Equilibrium
row/col              1             2             3             4
   1       1.00000e+00   1.00000e+00   0.00000e+00   0.00000e+00
   2       1.00000e+00   1.00000e+00   0.00000e+00   0.00000e+00
   3       1.00000e+00   7.07107e-01   0.00000e+00   0.00000e+00
   4       1.00000e+00   7.07107e-01   1.00000e+00   0.00000e+00
   5       7.07107e-01   1.00000e+00   7.07107e-01   0.00000e+00
   6       1.00000e+00  -1.00000e+00   7.07107e-01  -7.07107e-01
   7       1.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00
   8       1.00000e+00  -1.00000e+00   0.00000e+00   0.00000e+00
   9       1.00000e+00   8.94427e-01   0.00000e+00   0.00000e+00
  10      -4.47214e-01   0.00000e+00   0.00000e+00   0.00000e+00
  11       1.00000e+00  -1.00000e+00   0.00000e+00   0.00000e+00
  12      -1.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00
  13       1.00000e+00  -1.00000e+00   8.94427e-01   0.00000e+00
  14      -1.00000e+00   1.00000e+00   4.47214e-01   0.00000e+00
  15       7.07107e-01   1.00000e+00   7.07107e-01   0.00000e+00
  16       1.00000e+00   7.07107e-01  -1.00000e+00  -7.07107e-01
  17       8.94427e-01  -8.94427e-01  -8.94427e-01   0.00000e+00
  18       1.00000e+00   4.47214e-01  -4.47214e-01   4.47214e-01
  19       1.00000e+00  -8.94427e-01   7.07107e-01   0.00000e+00
  20       1.00000e+00   4.47214e-01   7.07107e-01   0.00000e+00
  21       1.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00
  22       1.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00

Matrix: Destination
row/col              1             2             3             4
   1       8.00000e+00   2.00000e+01   0.00000e+00   0.00000e+00
   2       1.00000e+00   2.10000e+01   0.00000e+00   0.00000e+00
   3       8.00000e+00   9.00000e+00   0.00000e+00   0.00000e+00
   4       2.00000e+00   9.00000e+00   2.20000e+01   0.00000e+00
   5       9.00000e+00   1.00000e+01   1.10000e+01   0.00000e+00
   6       1.00000e+00   3.00000e+00   9.00000e+00   1.10000e+01
   7       1.00000e+01   0.00000e+00   0.00000e+00   0.00000e+00
   8       2.00000e+00   4.00000e+00   0.00000e+00   0.00000e+00
   9       1.20000e+01   1.60000e+01   0.00000e+00   0.00000e+00
  10       1.60000e+01   0.00000e+00   0.00000e+00   0.00000e+00
  11       1.20000e+01   1.30000e+01   0.00000e+00   0.00000e+00
  12       5.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00
  13       1.30000e+01   1.40000e+01   1.80000e+01   0.00000e+00
  14       3.00000e+00   6.00000e+00   1.80000e+01   0.00000e+00
```

```
  15       1.10000e+01   1.40000e+01   1.90000e+01   0.00000e+00
  16       4.00000e+00   1.10000e+01   7.00000e+00   1.90000e+01
  17       1.60000e+01   1.70000e+01   1.80000e+01   0.00000e+00
  18       5.00000e+00   1.60000e+01   1.70000e+01   1.80000e+01
  19       1.50000e+01   1.70000e+01   1.90000e+01   0.00000e+00
  20       6.00000e+00   1.70000e+01   1.90000e+01   0.00000e+00
  21       1.50000e+01   0.00000e+00   0.00000e+00   0.00000e+00
  22       7.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00

---
--- Part 2: Systematically Assemble Matrix A ...
---
--- (row,col) -->  0, 0: eitem -->   1.00000, ditem -->   8 ...
--- (row,col) -->  0, 1: eitem -->   1.00000, ditem -->  20 ...
---

... lines of output removed ...

---
--- (row,col) --> 20, 0: eitem -->   1.00000, ditem -->  15 ...
---
--- (row,col) --> 21, 0: eitem -->   1.00000, ditem -->   7 ...

Matrix: A
row/col              1             2             3             4             5             6
   1       0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00
   2       1.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00
   3       0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00
   4       0.00000e+00   1.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00
   5       0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00
   6       1.00000e+00   0.00000e+00  -1.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00
   7       0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00
   8       0.00000e+00   1.00000e+00   0.00000e+00  -1.00000e+00   0.00000e+00   0.00000e+00
   9       0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00
  10       0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00
  11       0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00
  12       0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00  -1.00000e+00   0.00000e+00
  13       0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00
  14       0.00000e+00   0.00000e+00  -1.00000e+00   0.00000e+00   0.00000e+00   1.00000e+00
  15       0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00
  16       0.00000e+00   0.00000e+00   0.00000e+00   1.00000e+00   0.00000e+00   0.00000e+00
  17       0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00
  18       0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   1.00000e+00   0.00000e+00
  19       0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00
  20       0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   1.00000e+00
  21       0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00
  22       0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00

Matrix: A
row/col              7             8             9            10            11            12
   1       0.00000e+00   1.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00
   2       0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00
   3       0.00000e+00   1.00000e+00   7.07107e-01   0.00000e+00   0.00000e+00   0.00000e+00
   4       0.00000e+00   0.00000e+00   7.07107e-01   0.00000e+00   0.00000e+00   0.00000e+00
   5       0.00000e+00   0.00000e+00   7.07107e-01   1.00000e+00   7.07107e-01   0.00000e+00
```

6	0.00000e+00	0.00000e+00	7.07107e-01	0.00000e+00	-7.07107e-01	0.00000e+00
7	0.00000e+00	0.00000e+00	0.00000e+00	1.00000e+00	0.00000e+00	0.00000e+00
8	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00
9	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	1.00000e+00
10	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00
11	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	1.00000e+00
12	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00
13	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00
14	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00
15	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	7.07107e-01	0.00000e+00
16	-1.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	7.07107e-01	0.00000e+00
17	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00
18	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00
19	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00
20	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00
21	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00
22	1.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00

Matrix: A

row/col	13	14	15	16	17	18
1	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00
2	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00
3	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00
4	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00
5	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00
6	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00
7	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00
8	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00
9	0.00000e+00	0.00000e+00	0.00000e+00	8.94427e-01	0.00000e+00	0.00000e+00
10	0.00000e+00	0.00000e+00	0.00000e+00	-4.47214e-01	0.00000e+00	0.00000e+00
11	-1.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00
12	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00
13	1.00000e+00	-1.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	8.94427e-01
14	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	4.47214e-01
15	0.00000e+00	1.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00
16	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00
17	0.00000e+00	0.00000e+00	0.00000e+00	8.94427e-01	-8.94427e-01	-8.94427e-01
18	0.00000e+00	0.00000e+00	0.00000e+00	4.47214e-01	-4.47214e-01	4.47214e-01
19	0.00000e+00	0.00000e+00	1.00000e+00	0.00000e+00	-8.94427e-01	0.00000e+00
20	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	4.47214e-01	0.00000e+00
21	0.00000e+00	0.00000e+00	1.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00
22	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00

Matrix: A

row/col	19	20	21	22
1	0.00000e+00	1.00000e+00	0.00000e+00	0.00000e+00
2	0.00000e+00	0.00000e+00	1.00000e+00	0.00000e+00
3	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00
4	0.00000e+00	0.00000e+00	0.00000e+00	1.00000e+00
5	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00
6	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00
7	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00
8	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00
9	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00
10	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00

```
  11       0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00
  12       0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00
  13       0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00
  14       0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00
  15       7.07107e-01   0.00000e+00   0.00000e+00   0.00000e+00
  16      -7.07107e-01   0.00000e+00   0.00000e+00   0.00000e+00
  17       0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00
  18       0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00
  19       7.07107e-01   0.00000e+00   0.00000e+00   0.00000e+00
  20       7.07107e-01   0.00000e+00   0.00000e+00   0.00000e+00
  21       0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00
  22       0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00

---
--- Part 2: Initialize 22x1 load vectors ...
---

--- Load Case 1: Node e_y = -15 ...

Matrix: Load Case 1:
row/col              1
   1       0.00000e+00
   2       0.00000e+00
   3       0.00000e+00

... lines of output removed ...

   9       0.00000e+00
  10      -1.50000e+01
  11       0.00000e+00

... lines of output removed ...

  21       0.00000e+00
  22       0.00000e+00

--- Load Case 2: Node e_y = -10, node f_y = -10 ...

Matrix: Load Case 2:
row/col              1
   1       0.00000e+00
   2       0.00000e+00

... lines of output removed ...

   9       0.00000e+00
  10      -1.00000e+01
  11       0.00000e+00
  12      -1.00000e+01
  13       0.00000e+00

... lines of output removed ...

  20       0.00000e+00
  21       0.00000e+00
```

```
  22       0.00000e+00

--- Load Case 5: Node f_y = -25 ...

Matrix: Load Case 3:
row/col              1
   1       0.00000e+00
   2       0.00000e+00
   3       0.00000e+00

... lines of output removed ...

  11       0.00000e+00
  12      -2.50000e+01
  13       0.00000e+00

... lines of output removed ...

  20       0.00000e+00
  21       0.00000e+00
  22       0.00000e+00

---
--- Part 4: Check properties of matrix A ...
---
--- Matrix A: rank = 22.000000, det = -0.126491  ...

---
--- Part 5: Solve A.F = B for three load cases ...
---

Matrix: Load Case 1: Forces ...
row/col              1
   1      -4.50000e+01
   2       3.00000e+01
   3      -4.50000e+01
   4       3.00000e+01
   5      -0.00000e+00
   6      -4.50000e+01
   7      -2.00187e-15
   8       0.00000e+00
   9       0.00000e+00
  10       3.55271e-15
  11      -5.02430e-15
  12      -3.00000e+01
  13      -3.00000e+01
  14      -3.00000e+01
  15      -2.49675e-15
  16       3.35410e+01
  17       3.35410e+01
  18       0.00000e+00
  19       4.24264e+01
  20       0.00000e+00
  21       4.50000e+01
  22      -3.00000e+01
```

```
Matrix: Load Case 2: Forces ...
row/col              1
   1      -5.00000e+01
   2       3.00000e+01
   3      -5.00000e+01

... lines of output removed ...

  20       0.00000e+00
  21       5.00000e+01
  22      -3.00000e+01

Matrix: Load Case 3: Forces ...
row/col              1
   1      -5.00000e+01
   2       2.50000e+01
   3      -5.00000e+01

... lines of output removed ...

  20       0.00000e+00
  21       5.00000e+01
  22      -2.50000e+01

---
--- Part 6: Print support reactions and element-level forces ...
---
--- Support Reactions and Element-Level Forces: Load Case 1:
---
---   Reaction A: R_ax =    0.00 ...
---             : R_ay =   45.00 ...
---   Reaction D: R_dy =  -30.00 ...
---
--- Element Level Forces:
---
---   Element A-C: F1  =  -45.00 ...
---   Element B-D: F2  =   30.00 ...
---   Element C-G: F3  =  -45.00 ...
---   Element D-H: F4  =   30.00 ...
---   Element F-I: F5  =   -0.00 ...
---   Element G-J: F6  =  -45.00 ...
---   Element H-K: F7  =   -0.00 ...
---   Element A-B: F8  =    0.00 ...
---   Element B-C: F09 =    0.00 ...
---   Element C-D: F10 =    0.00 ...
---   Element C-H: F11 =   -0.00 ...
---   Element E-F: F12 =  -30.00 ...
---   Element F-G: F13 =  -30.00 ...
---   Element G-H: F14 =  -30.00 ...
---   Element J-K: F15 =   -0.00 ...
---   Element E-I: F16 =   33.54 ...
---   Element I-J: F17 =   33.54 ...
---   Element G-I: F18 =    0.00 ...
---   Element H-J: F19 =   42.43 ...
```

```
---
--- Support Reactions and Element-Level Forces: Load Case 2:
---
---   Reaction A: R_ax =    0.00 ...
---             : R_ay =   50.00 ...
---   Reaction D: R_dy =  -30.00 ...
---
--- Element Level Forces:
---
---   Element A-C: F1  =  -50.00 ...
---   Element B-D: F2  =   30.00 ...
---   Element C-G: F3  =  -50.00 ...

... lines of output removed ...

---   Element I-J: F17 =   33.54 ...
---   Element G-I: F18 =  -11.18 ...
---   Element H-J: F19 =   42.43 ...
---
--- Support Reactions and Element-Level Forces: Load Case 3:
---
---   Reaction A: R_ax =    0.00 ...
---             : R_ay =   50.00 ...
---   Reaction D: R_dy =  -25.00 ...
---
--- Element Level Forces:
---
---   Element A-C: F1  =  -50.00 ...
---   Element B-D: F2  =   25.00 ...
---   Element C-G: F3  =  -50.00 ...

... lines of output removed ...

---   Element I-J: F17 =   27.95 ...
---   Element G-I: F18 =  -27.95 ...
---   Element H-J: F19 =   35.36 ...
---
--- Summary of Max/Min Reactions and Element-Level Forces ...
--- ------------------------------------------------- ...
---
---   Reaction A: Minimum R_ax =    0.00, Maximum R_ax =    0.00 ...
---             : Minimum R_ay =   45.00, Maximum R_ay =   50.00 ...
---
---   Reaction B: Minimum R_by =  -30.00, Maximum R_by =  -25.00 ...
---
---   Element A-C:
---     Minimum F1 =  -50.00 (C) ...
---     Maximum F1 =  -45.00 (C) ...
---   Element B-D:
---     Minimum F2 =   25.00 (T) ...
---     Maximum F2 =   30.00 (T) ...
---   Element C-G:
---     Minimum F3 =  -50.00 (C) ...
---     Maximum F3 =  -45.00 (C) ...
---   Element D-H:
```

```
---     Minimum F4 =   25.00 (T) ...
---     Maximum F4 =   30.00 (T) ...
---   Element F-I:
---     Minimum F5 =   -0.00 (T) ...
---     Maximum F5 =   25.00 (T) ...
---   Element G-J:
---     Minimum F6 =  -45.00 (C) ...
---     Maximum F6 =  -37.50 (C) ...
---   Element H-K:
---     Minimum F7 =   -0.00 (C) ...
---     Maximum F7 =    0.00 (T) ...
---   Element A-B:
---     Minimum F8 =    0.00 (T) ...
---     Maximum F8 =    0.00 (T) ...
---   Element B-C:
---     Minimum F9 =    0.00 (T) ...
---     Maximum F9 =    0.00 (T) ...
---   Element C-D:
---     Minimum F10 =   -0.00 (T) ...
---     Maximum F10 =    0.00 (T) ...
---   Element C-H:
---     Minimum F11 =   -0.00 (C) ...
---     Maximum F11 =    0.00 (T) ...
---   Element E-F:
---     Minimum F12 =  -30.00 (C) ...
---     Maximum F12 =    0.00 (T) ...
---   Element F-G:
---     Minimum F13 =  -30.00 (C) ...
---     Maximum F13 =    0.00 (T) ...
---   Element G-H:
---     Minimum F14 =  -30.00 (C) ...
---     Maximum F14 =  -25.00 (C) ...
---   Element J-K:
---     Minimum F15 =   -0.00 (C) ...
---     Maximum F15 =    0.00 (T) ...
---   Element E-I:
---     Minimum F16 =   -0.00 (C) ...
---     Maximum F16 =   33.54 (T) ...
---   Element I-J:
---     Minimum F17 =   27.95 (T) ...
---     Maximum F17 =   33.54 (T) ...
---   Element G-I:
---     Minimum F18 =  -27.95 (C) ...
---     Maximum F18 =    0.00 (T) ...
---   Element H-J:
---     Minimum F19 =   35.36 (T) ...
---     Maximum F19 =   42.43 (T) ...

--- ======================================== ...
--- Leave TestTrussAnalysis04.main()         ...
```