Algorithm:

1. Enter the size of the matrix [square matrix]

**matrix\_size = n**

1. Enter the reliability value as **reliability = r**
2. If matrix\_size > 1 :

Call the input matrix function as input\_matrix()

If matrix\_size == 1 or matrix\_size < 1:

**input\_matrix\_value** = [1,1,1]

1. exit() **[only in the case of when the matrix size is 1 or less than 1]**
2. else:

for i in range(0,**n**):

enter the value of matrix

**Eg:**

1 1 1 : r1c1 row1-column1

2 2 2 : r1c2

3 3 3 : r1c3

**In the program we enter the matrix in the form of list*(3-D list )*:**

***[ [ [1.0, 1.0, 1.0] , [1.5, 2.0, 2.5] , [1.5, 2.0, 2.5] ],***

***[ [0.4, 0.5, 0.67] , [1.0, 1.0, 1.0], [0.67, 1.0, 2.0] ],***

***[ [0.4, 0.5, 0.67] , [0.5, 1.0, 1.5] , [1.0, 1.0, 1.0] ]***

***]***

1. Sum of Rows – Column wise:

Let b = [0,0,0] is an empty matrix

For i in range(0,**n**):

For j in range(0,**n**):

For k in range(0,**3**):

b[i][k] = b[i][k] + **input\_matrix\_value[i][j][k]**

1. Then b is the matrix with SA(i), where i = 1,2,3…
2. After calculating the sum of rows column-wise add them all **eg. Let Sall = (SA1 + SA2 + SA3)**
3. Now, calculate the reciprocal of **Sall** => 1/ **Sall**
4. After reciprocal, swap the first with last element

**Eg.**

1/ **Sall**  = [ 1/7.97 , 1/10 ,1/12.84]

=>> [ 1/12.84, 1/10, 1/7.97]

1. Now , calculate SF(i) ,where i = 1,2,3….

SF(i)  = SA(i)  X (1 / **Sall** )

1. Now, calculate the value of p :
2. Compare SF(i) with SF(i+1) and vice-versa:

SF(i) = [li , mi , ui]

If mi >= mi + 1

Then p is 1

If li + 1 >= ui

Then p is 0

If Both the above condition not applicable:

Then

Swap, li with li + 1

mi with mi+1

ui with ui+1,

and then calculate,

***P = (li – ui+1) / (mi­+1  - ui + 1) - (mi  - li)***

1. Store every p value

**Eg. For SF1 p = [1,1]**

**For SF2 p = [1.73078,1]**

1. Then find the **min** of every **p** for **SF(i)**
2. **Eg. For SF1 p1 = min[1,1] = 1**
3. **For SF2 p2 = min[1.73078,1] = 1**
4. Calculate weights = ∑ pi where i = 1,2,3…
5. After that calculate normalized weights as **N**

**N = ( pi / weights )**

**eg**

(1/3, 1/3, 1/3 )

1. Now, Enter the Total reliability of a system (**Sys\_rel**)
2. Reliability allocation is calculated as (**Sys\_rel**)­­**N(i)**

**Eg :** For F1 = (0.95)0.3333  = 0.98304

1. Call Module function()
2. Now , Enter Number of Modules .- **m**
3. Fi is connected to how many modules - **o**
4. If **o < m**:
5. REPEAT from Step 2 to Step 21.
6. After these steps, the final result is reliability allocation.

**Eg :** Reliability Allocation for F 1 is :( 0.88 ) \*\* 0.0 : 1.0

F 1 is: 1.0

Reliability Allocation for F 2 is :( 0.88 ) \*\* 1.0 : 0.88

F 2 is: 0.88

**Program**

## Main 2

def input\_matrix(matrix\_size):

    if matrix\_size == 1 or matrix\_size < 1:

        input\_matrix\_values = [[[1,1,1]]]

    else:

        print("Enter Values in the matrix with number1[space]number2[space]number3 \n Example: 1 1 1 [enter]" )

        print("Enter the values of Matrix : \n")

        input\_matrix\_values = []

        for i in range(0,matrix\_size):

            a = []   #temporaray list

            for i in range(0,matrix\_size):

                a.append([float(x) for x in input().split()])

            input\_matrix\_values.insert(i,a)

            del a

        print("\n Main Inputted Matrix is :\n",input\_matrix\_values)

#     Main 3

# Sum of Rows Column Wise

    b = []

    for j in range (0,matrix\_size):

        b.append([0,0,0])

    for i in range(0,matrix\_size):

        for j in range(0,matrix\_size):

            for k in range(0,3):

                b[i][k] = b[i][k] + input\_matrix\_values[i][j][k]

    length = len(b)//matrix\_size

    sum1 = 0

    for i in range(0,matrix\_size):

        print("SA",i+1,"is",b[sum1:length])

        sum1 = length

        length = length + 1

# Main 4

# Sum of All above matrix --> SA1 + SA2 + SA3

    c = [0,0,0]

    for i in range(0,matrix\_size):

        for j in range(0,3):

            c[j] = c[j] + b[i][j]

            c[j] = round(c[j],4)

    print("\n Sum of SA1 + SA2 + SA3 is:S.all. -->",c)

#     Main 5

# Reciprocal 1/S.all.

    for i in range(0,3):

        c[i] = 1/c[i]

        c[i] = round(c[i],4)  # round upto 4 decimal places

    print("\n Reciprocal [1/S.all.] is :\t ",c)

# Main 6

# swap the first with last element

    length = len(c)

    for i in range(length//2):

        c[i],c[length-i-1] = c[length-i-1],c[i]

    print("\n c is S.all.[after swapping]:\t",c)

#     Main 7

# multiply

    d = []

    for i in range(0,matrix\_size):

        for j in range(0,3):

            var = c[j] \* b[i][j]

            var = round(var,4) # round upto 4 decimal places

            d.append(var)

    # print(d)

    length1 = len(d)//matrix\_size  # floor division means [5/2 --> 2.5 --> 2(floor value)]

    sum = 0

    f = []

    for i in range(0,matrix\_size):

        print("SF",i+1,"is",d[sum:length1])

        a = d[sum:length1]

        f.append(a)

        sum = length1

        length1 = length1 + 3

#     Main 8

# Find the value of p`

    P = []

    min\_list = []

    for i in range(0,matrix\_size):

        P = f[i]

        j = 0

        l1 = f[i][j]         #SF1

        m1 = f[i][j+1]

        u1 = f[i][j+2]

        for j in range(0,matrix\_size):

            p = 99999

            if(f[i] == f[j]):

                min\_list1 = min\_list.append(p)

            else:

                k = 0

                l2 = f[j][k]

                m2 = f[j][k+1]

                u2 = f[j][k+2]

                if(m1 >= m2):

                    p = 1

                    min\_list1 = min\_list.append(p)

                elif(l2 >= u1):

                    p = 0

                    min\_list1 = min\_list.append(p)

                else:

                    l1 , l2 = l2 ,l1

                    u1 , u2 = u2 , u1

                    m1 , m2 = m2 , m1

                    var1 = p = ((l1 - u2)/((m2 - u2) - (m1 - l1)))

                    p = round(var1,4)

                    min\_list1 = min\_list.append(p)

    print("Final Min List is ",min\_list)

    g = []

    sum = 0

    length2 = len(min\_list) // matrix\_size

    for i in range(0,matrix\_size):

        a = min\_list[sum:length2]

        g.append(a)

        sum = length2

        length2 = length2 + matrix\_size

    store\_p = []

    for j in range(0,matrix\_size):

        p = min(g[j])

        store\_p.append(p)

        print("p",j+1,"` is:",p)

    print("store\_p is :",store\_p)

    p\_sum = 0

    for i in range(0,matrix\_size):

        p\_sum += store\_p[i]

    global norm\_w , n\_re , length3

    norm\_w = []

    n\_re = []

    for i in range(0,matrix\_size):

        n\_re = store\_p[i] / p\_sum

        n\_re = round(n\_re,4)

        norm\_w.append(n\_re)

    print("Normalized Weight is :",norm\_w)

    length3 = len(norm\_w)

# Find Reliability

def for\_reliability(reliability,length3,norm\_w):

    R = 0.0000

    allreli = []

    global alloc

    for j in range(0,length3):

        R = reliability \*\* norm\_w[j]

        R = round(R,4)

        alloc.append(R)

        allreli.append(R)

        print("Reliability Allocation for F",j+1,"is :" "(",reliability,") \*\* ",norm\_w[j]," :",R)

        print("F",j+1,"is :",R,"\n")

    print(allreli)

    return alloc

# Main 9

def module\_case(matrix\_size,alloc):

    for i in range(0,len(alloc)):

        print("F",i+1,"->",alloc[i],"is connected to how many modules")

        matrix\_size1 = int(input())

        input\_matrix(matrix\_size1)

# Main 1

alloc = []

matrix\_size = int(input("ENTER THE NUMBER OF ROWS AND COLUMN[Rows must be equal to column] :"))

reliability = float(input("Total Required reliability of System is :"))

if matrix\_size > 1:

    input\_matrix(matrix\_size)

    for\_reliability(reliability,length3,norm\_w)

    a = int(input("Enter Number of Modules:"))

for i in range(len(alloc)):

        print("F",i+1,"is :",alloc[i])

        print("F",i+1,"is connected to how many modules:")

        matrix\_size1 = int(input(""))

        input\_matrix(matrix\_size1)

        reliability = alloc[i]

        for\_reliability(reliability,length3,norm\_w)

**OUTPUT SCREEN RESULT**

ENTER THE NUMBER OF ROWS AND COLUMN[Rows must be equal to column] :3

Total Required reliability of System is :0.88

Enter Values in the matrix with number1[space]number2[space]number3

Example: 1 1 1 [enter]

Enter the values of Matrix :

1 1 1

.2857 .3333 .4

.3333 .4 .5

2.5 3 3.5

1 1 1

.4 .5 .6667

2 2.5 3

1.5 2 2.5

1 1 1

Main Inputted Matrix is :

[[[1.0, 1.0, 1.0], [0.2857, 0.3333, 0.4], [0.3333, 0.4, 0.5]],

[[2.5, 3.0, 3.5], [1.0, 1.0, 1.0], [0.4, 0.5, 0.6667]],

[[2.0, 2.5, 3.0], [1.5, 2.0, 2.5], [1.0, 1.0, 1.0]]]

SA 1 is [[1.619, 1.7332999999999998, 1.9]]

SA 2 is [[3.9, 4.5, 5.1667]]

SA 3 is [[4.5, 5.5, 6.5]]

Sum of SA1 + SA2 + SA3 is:S.all. --> [10.019, 11.7333, 13.5667]

Reciprocal [1/S.all.] is : [0.0998, 0.0852, 0.0737]

c is S.all.[after swapping]: [0.0737, 0.0852, 0.0998]

SF 1 is [0.1193, 0.1477, 0.1896]

SF 2 is [0.2874, 0.3834, 0.5156]

SF 3 is [0.3317, 0.4686, 0.6487]

// 99999 is the max p used in the program to find min(p) : Comment

Final Min List is [99999, 0, 0, 1, 99999, 0.6834, 1, 1, 99999]

p 1 ` is: 0

p 2 ` is: 0.6834

p 3 ` is: 1

store\_p is : [0, 0.6834, 1]

Normalized Weight is : [0.0, 0.406, 0.594]

Reliability Allocation for F 1 is :( 0.88 ) \*\* 0.0 : 1.0

F 1 is : 1.0

Reliability Allocation for F 2 is :( 0.88 ) \*\* 0.406 : 0.9494

F 2 is : 0.9494

Reliability Allocation for F 3 is :( 0.88 ) \*\* 0.594 : 0.9269

F 3 is : 0.9269

[1.0, 0.9494, 0.9269]

Enter Number of Modules:2

F 1 is : 1.0

F 1 is connected to how many modules:

2

Enter Values in the matrix with number1[space]number2[space]number3

Example: 1 1 1 [enter]

Enter the values of Matrix :

1 1 1

1.5 2 2.5

.4 .5 .6667

1 1 1

Main Inputted Matrix is :

[[[1.0, 1.0, 1.0], [1.5, 2.0, 2.5]], [[0.4, 0.5, 0.6667], [1.0, 1.0, 1.0]]]

SA 1 is [[2.5, 3.0, 3.5]]

SA 2 is [[1.4, 1.5, 1.6667]]

Sum of SA1 + SA2 + SA3 is:S.all. --> [3.9, 4.5, 5.1667]

Reciprocal [1/S.all.] is : [0.2564, 0.2222, 0.1935]

c is S.all.[after swapping]: [0.1935, 0.2222, 0.2564]

SF 1 is [0.4838, 0.6666, 0.8974]

SF 2 is [0.2709, 0.3333, 0.4273]

Final Min List is [99999, 1, 0, 99999]

p 1 ` is: 1

p 2 ` is: 0

store\_p is : [1, 0]

Normalized Weight is : [1.0, 0.0]

Reliability Allocation for F 1 is :( 1.0 ) \*\* 1.0 : 1.0

F 1 is : 1.0

Reliability Allocation for F 2 is :( 1.0 ) \*\* 0.0 : 1.0

F 2 is : 1.0

[1.0, 1.0]

F 2 is : 0.9494

F 2 is connected to how many modules:

2

Enter Values in the matrix with number1[space]number2[space]number3

Example: 1 1 1 [enter]

Enter the values of Matrix :

1 1 1

1.5 2 2.5

.4 .5 .6667

1 1 1

Main Inputted Matrix is :

[[[1.0, 1.0, 1.0], [1.5, 2.0, 2.5]], [[0.4, 0.5, 0.6667], [1.0, 1.0, 1.0]]]

SA 1 is [[2.5, 3.0, 3.5]]

SA 2 is [[1.4, 1.5, 1.6667]]

Sum of SA1 + SA2 + SA3 is:S.all. --> [3.9, 4.5, 5.1667]

Reciprocal [1/S.all.] is : [0.2564, 0.2222, 0.1935]

c is S.all.[after swapping]: [0.1935, 0.2222, 0.2564]

SF 1 is [0.4838, 0.6666, 0.8974]

SF 2 is [0.2709, 0.3333, 0.4273]

Final Min List is [99999, 1, 0, 99999]

p 1 ` is: 1

p 2 ` is: 0

store\_p is : [1, 0]

Normalized Weight is : [1.0, 0.0]

Reliability Allocation for F 1 is :( 0.9494 ) \*\* 1.0 : 0.9494

F 1 is : 0.9494

Reliability Allocation for F 2 is :( 0.9494 ) \*\* 0.0 : 1.0

F 2 is : 1.0

[0.9494, 1.0]

F 3 is : 0.9269

F 3 is connected to how many modules:

2

Enter Values in the matrix with number1[space]number2[space]number3

Example: 1 1 1 [enter]

Enter the values of Matrix :

1 1 1

1.5 2 2.5

.4 .5 .6667

1 1 1

Main Inputted Matrix is :

[[[1.0, 1.0, 1.0], [1.5, 2.0, 2.5]], [[0.4, 0.5, 0.6667], [1.0, 1.0, 1.0]]]

SA 1 is [[2.5, 3.0, 3.5]]

SA 2 is [[1.4, 1.5, 1.6667]]

Sum of SA1 + SA2 + SA3 is:S.all. --> [3.9, 4.5, 5.1667]

Reciprocal [1/S.all.] is : [0.2564, 0.2222, 0.1935]

c is S.all.[after swapping]: [0.1935, 0.2222, 0.2564]

SF 1 is [0.4838, 0.6666, 0.8974]

SF 2 is [0.2709, 0.3333, 0.4273]

Final Min List is [99999, 1, 0, 99999]

p 1 ` is: 1

p 2 ` is: 0

store\_p is : [1, 0]

Normalized Weight is : [1.0, 0.0]

Reliability Allocation for F 1 is :( 0.9269 ) \*\* 1.0 : 0.9269

F 1 is : 0.9269

Reliability Allocation for F 2 is :( 0.9269 ) \*\* 0.0 : 1.0

F 2 is : 1.0

[0.9269, 1.0]

**//FINAL RESULT**

You can also find the program in my Github: <https://github.com/Aadish-jain/Research-work-Project>