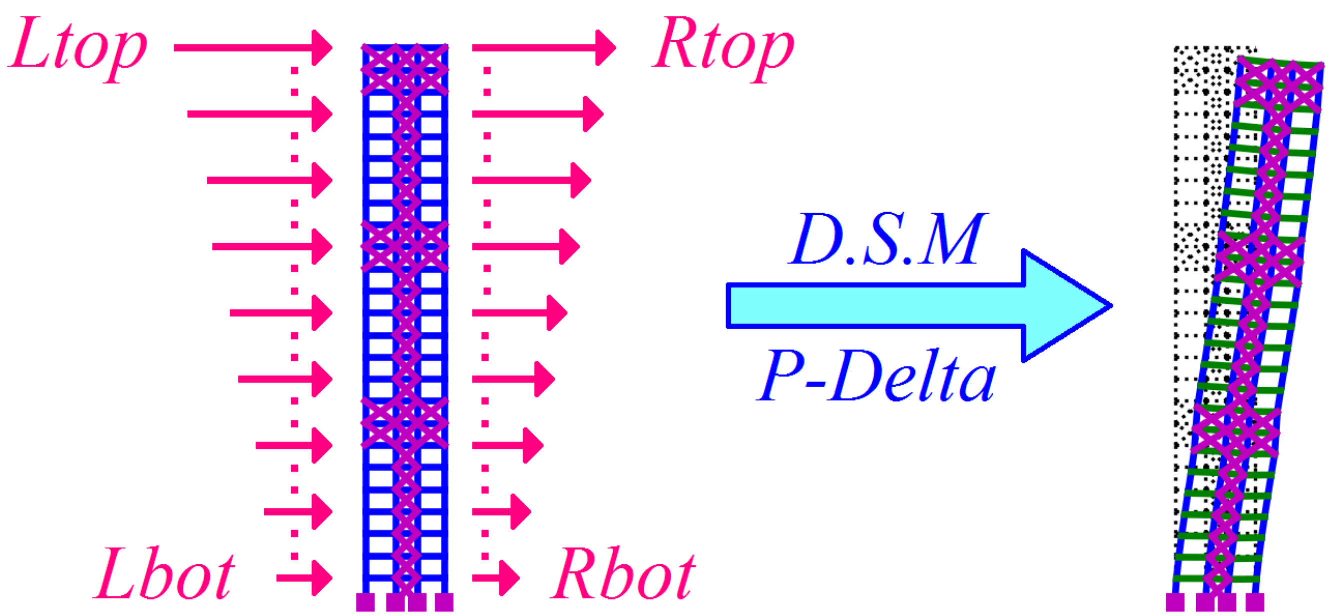
"P_Delta_Analysis"

of a 25 story frame with core & outriggers

Under Triangular Nodal Forces on both sides

and rectangular distributed beam load



In [1]:

```
# (auto) importing modules needed

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import copy
```

Structure Data (input)

In [2]:

```
# (input)

nStory      = 25                             # number of stories

hStory_typ  = 3600                           # height of typical stories
hStory_Base = 4000                           # height of base_floor

wBay = [ 5000, 3500, 4500 ]                  # width of bays ( any amount )

Braced_Bays    = [ 2 ]                       # Doubly (X) Braced  Bays
Braced_Stories = [ 8,9,16,17,24,25 ]         # Doubly (X) Braced Storys

# sections:
# All Columns:    IPBv 220:  A=14900 (mm2)    I=146.0*10**6 (mm4)
# All Beams:      IPE  300:  A= 5380 (mm2)    I= 83.6*10**6 (mm4)
# All Diagonals:  UPE  120:  A= 1700 (mm2)

E_Cols, E_Beams, E_Diags = 200,200,200       # Modulus of Elasticity (KN/mm2)
A_Cols, A_Beams, A_Diags = 14900,5380,1700   # Area (mm2)
I_Cols, I_Beams          = 146e6,83.6e6      # Moment of Inertia (mm4)

W_Beams = 0.02                               # rectangular Distributed Load on Beams (K
                                             # towards Ground is +ve

# Nodal Forces on left and right side of structure (KN)
# Just imput values for roof and story 1
# mid-points Loads will be auto-generated by interpolation
# +ve direction is from left to right

Ltop = 5.0          # Point Load applied to "Roof" from Left-side of the frame (KN)
Lbot = 2.0          # Point Load applied to "story 1" from Left-side of the frame (KN)
                    # Point Load of stories in between are interpolated.

Rtop = 3.0          # Point Load applied to "Roof"  from right-side of the frame (KN)
Rbot = 1.0          # Point Load applied to "story 1" from right-side of the frame (KN)
                    # Point Load of stories in between are interpolated.
```

Structure Data (auto)

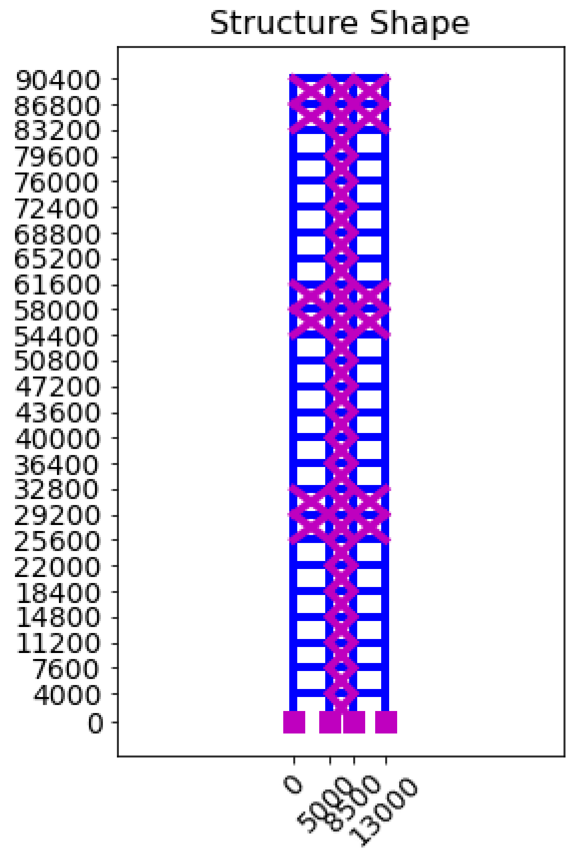
In [3]:

```
# (auto) Checking & completing

hStory     = np.ones( (nStory,1) ) * hStory_typ
hStory[0] = hStory_Base

nBay = len(wBay)     # number of bays

Braced_Bays = sorted( Braced_Bays )
keep = []
for i in Braced_Bays:
    if i <= nBay: keep.append(i)
    else: print('\n Warning!',
                '\n Bay', i, "does not exist thus can't be braced.",
                i, 'is removed from Braced_Bays' )
Braced_Bays = sorted(keep)

Braced_Stories = sorted( Braced_Stories )
keep = []
for i in Braced_Stories:
    if i <= nStory: keep.append(i)
    else: print('\n Warning!',
                '\n Story', i, "does not exist thus can't be braced.",
                i, 'is removed from Braced_Stories' )
Braced_Stories = sorted(keep)
```

Points & Connectivity Matrices

In [4]:

```
# (auto) Making Frame & Checking Shape

# defining a function called PC
# to calculate Points & Connectivity matrices of Columns, beams and diagonals

def PC( wBay, hStory, Braced_Bays, Braced_Stories ):
    ...
    return [ Points, CnC, CnB, CnD ]

[ Points, CnC, CnB, CnD ] = PC( wBay, hStory, Braced_Bays, Braced_Stories )


# define a function called Shape
# to check shape of frame

def  Shape( Points, CnC, CnB, CnD, wBay, hStory ):
   ...
   return None

Shape( Points, CnC, CnB, CnD, wBay, hStory )
```



Finding Static Coefficients

In [5]:

```
# (auto) Coefficients
# define a function called Static_Coeff
# to find Static Analysis Coefficients:

def Static_Coeff( Points, CnC, CnB, CnD, wBay ):
    ...
    return [NOP, NOD, RD, FD, NFD, NC, NB, ND, IndxC, IndxB, IndxD]

[NOP, NOD, RD, FD, NFD, NC, NB, ND, IndxC, IndxB, IndxD] = Static_Coeff(Points,CnC,CnB,

```

Assigning

In [6]:

```
# (auto) define a function called Elements
# to asign material & shape Properties & distributed Load
# to columns, beams and trusses

def Elements(
       NB, IndxB, E_Beams, A_Beams, I_Beams, W_Beams
    , NC, IndxC, E_Cols,  A_Cols,  I_Cols
    , ND, IndxD, E_Diags, A_Diags ):
    ...
    return [Beams, Columns, Diagonals]

[ Beams, Columns, Diagonals ] = Elements( NB, IndxB, E_Beams, A_Beams, I_Beams, W_Beams
                                        , NC, IndxC, E_Cols,  A_Cols,  I_Cols
                                        , ND, IndxD, E_Diags, A_Diags )
```

Nodal Forces

In [7]:

```
# Nodal Forces

def Nodal_Forces( Lbot, Ltop, Rbot, Rtop, NOD, nStory, nBay ):
    ...
    return NF

NF = Nodal_Forces( Lbot, Ltop, Rbot, Rtop, NOD, nStory, nBay )
```

Elastic Stiffness Matrices

In [8]:

```
# (auto) defining Stiffness Matrices

def ke_frame( A, E, I, L ):
    ...
    return df

def ke_truss( A, E, L ):
    ...
    return df

def kg_frame( P, L ):
    ...
    return df

def kg_truss( P, L ):
    ...
    return df

def T_frame( c, s ):
    ...
    return df.T

def T_truss( c, s ):
    ...
    return df.T
```

Ke & Qf

In [9]:

```
# (auto) defining a function called KeQf
# to form:
#    Elastic Stiffness Matrices, Ke, for all elements
#    External Distributed Loads Matrix, Qf

def KeQf(  Points,   NOD
         , CnB, NB, IndxB, Beams
         , CnC, NC, IndxC, Columns
         , CnD, ND, IndxD, Diagonals ):
    ...
    return [ KE,QF, LC,TC,keC,KeC, LB,TB,keB,KeB,qfB, LD,TD,keD,KeD ]

[ KE,QF, LC,TC,keC,KeC, LB,TB,keB,KeB,qfB, LD,TD,keD,KeD ] = KeQf(
        Points,   NOD
      , CnB, NB, IndxB, Beams
      , CnC, NC, IndxC, Columns
      , CnD, ND, IndxD, Diagonals )
```

P-Delta Analysis

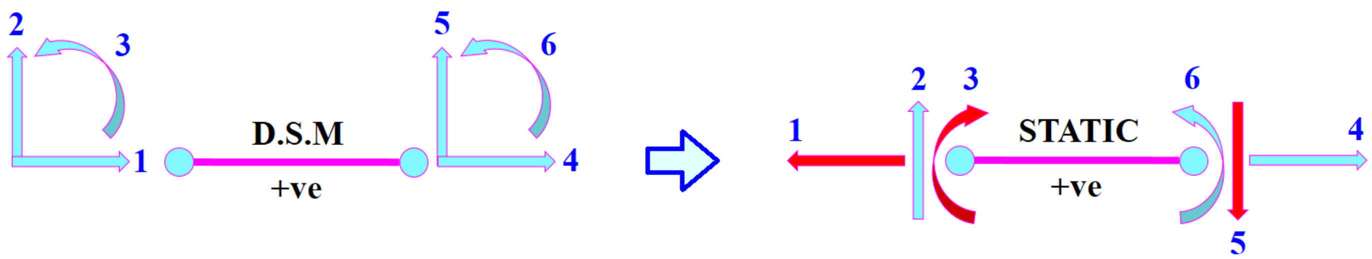
In [10]:

```
# (auto) define a function called P-Delta
# to perform the analysis

def P_Delta(
        NF,  NOD,   FD, KE, QF
      , NC, IndxC, TC, keC, KeC, Columns
      , NB, IndxB, TB, keB, KeB, Beams, qfB
      , ND, IndxD, TD, keD, KeD, Diagonals  ):
      ...
      ...
      ...
      return [ qC, qB, qD, R, U ]

# P-Delta Analysis
[ qC, qB, qD, R, U ] = P_Delta(
        NF,  NOD,   FD, KE, QF
      , NC, IndxC, TC, keC, KeC, Columns
      , NB, IndxB, TB, keB, KeB, Beams, qfB
      , ND, IndxD, TD, keD, KeD, Diagonals  )
```

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
P-Delta Analysis Converged after 5 Iterations
```

Converting +ve directions

Direct Stiffness Method has a +ve direction convension for axial-load, shear and bending-moment which is different from Standard Static +ve direction convensions; and will be corrected.



In [11]:

```python
# (auto) define a function called D2S
# o convert Direct Stiffness Method +ve direction
# to Standard Static +ve direction

def D2S( NC,NB,ND, qC,qB,qD ):
    ...
    return [ qCs,qBs,qDs ]

# Converting Directions (D.S.M. to Standard.Static)
# qX  --->  qXs
[ qCs,qBs,qDs ] = D2S( NC,NB,ND, qC,qB,qD )
```

Global Displacements, Support Reactions & Internal Member Forces

In [12]:

```python
# (auto) define a function called xyzNVM
# to generate Global Displacements, Structural Reaction, & Internal Member Forces

def xyzNVM( NC,NB,ND, U, R, qCs, qBs, qDs, nBay ):
    ...
    return [ Uxyz, Rxyz, AxC,VC,MC, AxB,VB,MB, AxD ]

[ Uxyz, Rxyz, AxC,VC,MC, AxB,VB,MB, AxD ] = xyzNVM(
    NC, NB, ND, U, R, qCs, qBs, qDs, nBay )

Rxyz
```

Out[12]:

	Rx	Ry	Rz
0.0	0.74	1117.56	9033.78
1.0	1.19	1174.38	22145.39
2.0	-127.83	2348.93	19541.23
3.0	-10.61	1859.13	26731.12

Adding Member-Forces to Elements DataFrame

In [14]:

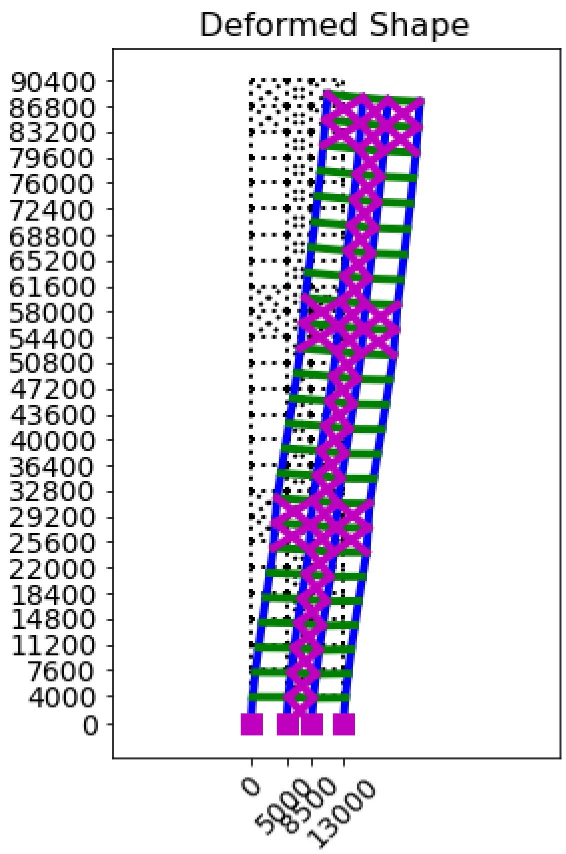
```python
def Add_member_forces( AxC,VC,MC, AxB,VB,MB, AxD, Columns, Beams, Diagonals ):
    ...
    return [ Columns, Beams, Diagonals ]

[ Columns, Beams, Diagonals ] = Add_member_forces( AxC,VC,MC, AxB,VB,MB, AxD, Columns,
```

Plotting Deformed Shape

(set the Scale manually)

In [15]:

```
def Plot_Deformed( Scale, Undeformed_YN, Points, Uxyz, CnC,CnB,CnD, NC,NB,ND, wBay,hSto
    ...
    return None

Scale = 100
Undeformed_YN = 'Y'

Plot_Deformed( Scale, Undeformed_YN, Points, Uxyz, CnC,CnB,CnD, NC,NB,ND, wBay,hStory )
```

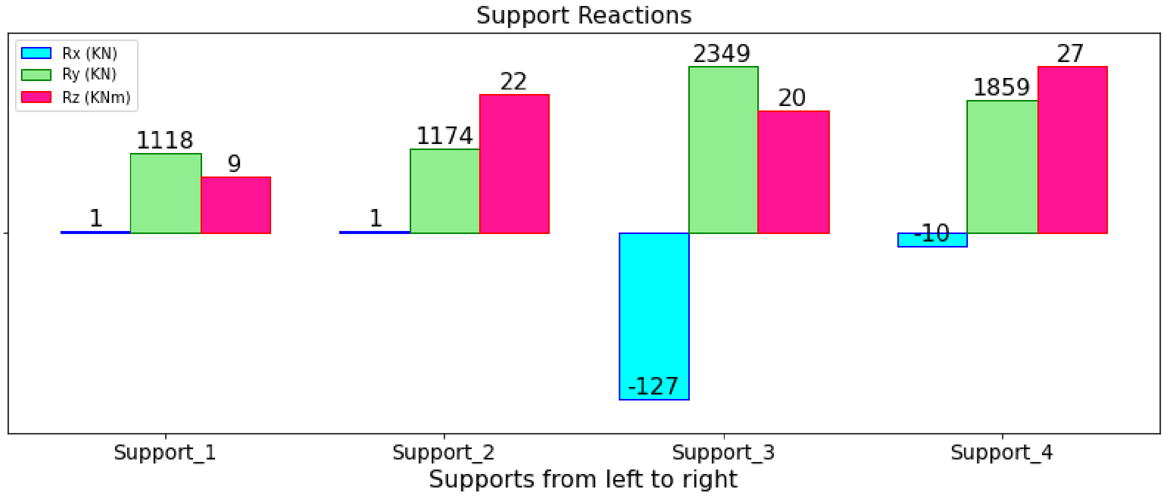


Visualizing Support-Reactions

- Rx: horizontal reaction (KN), +ve: left to right
- Ry: vertical reaction (KN), +ve: upward
- Rz: resistant-moment (KNm), +ve: counter-clockwise
- left outer support is Support_1

In [16]:

```python
def Plot_Support_Reactions( Rxyz, nBay ):
    ...
    return None

# Visualizing Support Reactions
Plot_Support_Reactions( Rxyz, nBay )
```



Important DataFrames

In [17]:

```python
#Uxyz          # Global Displacements
#Rxyz          # Support Reactions
#Columns       # Internal Member Forces
#Beams         # Internal Member Forces
#Diagonals     # Internal Member Forces
```

Important Visualizers

In [18]:

```python
#Shape( Points, CnC, CnB, CnD, wBay, hStory )    # Undeformed Shape
#Plot_Deformed(Scale,Undeformed_YN,Points,Uxyz,CnC,CnB,CnD,NC,NB,ND,wBay,hStory) # Def
#Plot_Support_Reactions( Rxyz, nBay )            # Visualizing Support Reactions
```