اولین همایش ملی
معماری، عمران ومحیط زیست شهری
تاریخ : ۹۳/۰۳/۰۱

Second-order elastic frame analysis with various solution algorithm methods

Mostafa Fathi Sepahvand[1], AhmadrezaJafari[2]

1- Ph.D. Student at Malayer University, Malayer, Iran mostafa_fathi_s@yahoo.com

2- Ph.D. Student at Malayer University, Malayer, Iran **arjafari2004@gmail.com**

Abstract

In this paper, various types of pseudo-codes for the solution of second order elastic analysis of frame structures are presented. For this purpose five different types solution consisting of Simple Load Control method, Newton-Raphson Load Control method, Displacement Control method, Arc Length Control method and Work Control method are presented. To demonstrate the correctness of presented pseudo-codes we implement them in MATLAB software afterward were solved a numerical example and verified by MASTAN software results.

Keywords: Second-order Elastic, Nonlinear, Pseudo-code, Load Control, Displacement Control

1. Introduction

Unlike a first-order analysis in which the solutions can be obtained in a rather simple and direct manner, a second-order analysis often entails an iterative type procedure to obtain solutions. This is due to the fact that the deformed geometry of the structure is not known during the deformation of the equilibrium and kinematic relationships. Thus, the analysis usually proceeds in a step-by-step incremental manner. The deformed geometry of the structure obtained from a preceding cycle of calculations is used as the basis for formulating the equilibrium and kinematic relationships for the current cycle of calculations [1]. There are various iterative schemes available for the solution of second-order problems. In this paper pseudo-code for implementing this solutions have been presented.

Nomenclature

$\boldsymbol{K}$	Stiffness matrix of frame structure in global coordinate
$\boldsymbol{R}$	Nodal force vector of frame structure in global coordinate
$\boldsymbol{D}$	Nodal displacement vector of frame structure in global coordinate
λ	Load increment factor
$\Delta\boldsymbol{R}$	Load increment vector
$\Delta\boldsymbol{D}$	Displacement increment vector
$\Delta\boldsymbol{D}'$	Displacement increment vector associated with $\boldsymbol{R_e}$
$\Delta\boldsymbol{D}''$	Displacement increment vector associated with $\boldsymbol{Q_i^j}$
P	Axial force of element
$\boldsymbol{Q}$	Unbalance force vector
$\boldsymbol{r}$	Nodal force vector of element in local coordinate
$\boldsymbol{d}$	Nodal displacement vector of element in local coordinate
E	Elastic modulus of element
A	Cross section of element
I	Moment inertia of element
α	Chord angle of element
β	Updated chord angle of element
L_0	Initial length of element
L_f	Updated length of element
u	Axial displacement
$\boldsymbol{L_k}$	Transform matrix of stiffness matrix from local to global coordinate
$\boldsymbol{L_r}$	Transform matrix of force vector from local to global coordinate
θ_A and θ_B	End rotations of element
M_A and M_B	End moments of element
M_A^F and M_B^F	Fixed end moments of element
φ_1 to φ_4	Stability stiffness functions

s_{ii} *and* s_{ij}	Stability functions
ds	Arc length
SLC	Simple load control
NR	Newton-Raphson
DC	Displacement control
ALC	Arc length control
WC	Work control
Subscript:	
e	Elastic
i	Step number
Superscript:	
j	Iteration number
T	Transpose operator

2. Solution algorithms for 2nd order rigid frame analysis

In this section the five methods for second-order analysis of a 2D simple frame are presented.

2.1. Simple Load Control Method

The pseudo-code of implementation of simple load control method with constant load increment factor is given as follows. Except of Simple Load Control method, other solution methods perform iterations to eliminating drift off error.

1. Start
2. Initialization
2.1. Describe the characteristics of frame structure as $\boldsymbol{Model}$
2.2. Take a value for *Number of Steps* $(\lambda = \frac{1}{Number\ of\ Steps})$
3. $[\boldsymbol{K_e}, \boldsymbol{R_e}, \boldsymbol{D_e}] = LinearAnalysis(\boldsymbol{Model})$
4. $\Delta\boldsymbol{R} = \lambda\boldsymbol{R_e}$
5. $\boldsymbol{K_0} = \boldsymbol{K_e}$
6. $\boldsymbol{R_0} = \boldsymbol{0}$
7. $\boldsymbol{D_0} = \boldsymbol{0}$
8. For $i = 1$ to *Number of Steps*:
8.1. $\Delta\boldsymbol{D_i} = (\boldsymbol{K_{i-1}})^{-1}\Delta\boldsymbol{R}$
8.2. $\boldsymbol{R_i} = \boldsymbol{R_{i-1}} + \Delta\boldsymbol{R}$

8.3.		$\boldsymbol{D_i} = \boldsymbol{D_{i-1}} + \Delta \boldsymbol{D_i}$
8.4.		$\boldsymbol{K_i} = TangentStiffness(\boldsymbol{Model}\,,\ \boldsymbol{D_i})$
9.	End-For	
10.	Show Output Results	
11.	End.	

In above algorithm, the function $LinearAnalysis$ performs a linear analysis on frame structure where the structural characteristics such as *Nodal Coordinates, Elements Connectivity, Area section properties, Material properties, Nodal Forces, Prescribed DOFs* and etc. as $\boldsymbol{Model}$ object are inputted. This function returns the nodal displacements, nodal forces and elastic stiffness of structure as output. The algorithm of $LinearAnalysis$ function to observe brevity it is not expressed here. This algorithm can be found in Reference [2] or [3].
Moreover the function $TangentStiffness$ computes the updated stiffness of structure by inputted structure nodal displacements. This function is given as follows:

1.	Function $[\boldsymbol{K}\,,\ \boldsymbol{R}] = TangentStiffness(\boldsymbol{Model}\,,\ \boldsymbol{D})$		
1.1.		$\boldsymbol{K} = \boldsymbol{0}$	
1.2.		$\boldsymbol{R} = \boldsymbol{0}$	
1.3.		For e = 1 To $Model.Elements.Numbers$	
1.3.1.			$EA = Model.Material.E(e) * Model.Section.A(e)$
1.3.2.			$EI = Model.Material.E(e) * Model.Section.I(e)$
1.3.3.			$\alpha = Model.Elements.AngleDirection(e)$
1.3.4.			$L_0 = Model.Elements.Length(e)$
1.3.5.			$elementDof = \boldsymbol{Model.Elements.DegreeOfFreedoms}(e)$
1.3.6.			$\boldsymbol{d} = \boldsymbol{D}_{(\boldsymbol{elementDof})}$
1.3.7.			$d_h = d_4 + L_0 \cos\alpha - d_1$
1.3.8.			$d_v = d_5 + L_0 \sin\alpha - d_2$
1.3.9.			$L_f = \sqrt{d_h^2 + d_v^2}$
1.3.10.			$u = L_f - L_0$
1.3.11.			$P = \frac{EA}{L_f} * u$
1.3.12.			$\beta = \sin^{-1} d_v/L_f$
1.3.13.			$\theta_A = \alpha + d_3 - \beta$
1.3.14.			$\theta_B = \alpha + d_6 - \beta$
1.3.15.			$\boldsymbol{L_k} = \begin{bmatrix} \cos\beta & \sin\beta & 0 & 0 & 0 & 0 \\ -\sin\beta & \cos\beta & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & \cos\beta & \sin\beta & 0 & 0 \\ 0 & 0 & -\sin\beta & \cos\beta & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$

1.3.16. $(kL)^2 = PL_f^2 / EI$

1.3.17. $N = 10$

1.3.18. $\varphi = \frac{1}{12} + \sum_{n=1}^{N} \frac{2(n+1)}{(2n+4)!} [(kL)^2]^n$

1.3.19. $\varphi_1 = \frac{1}{12\varphi} \left\{ 1 + \sum_{n=1}^{N} \frac{1}{(2n+1)!} [(kL)^2]^n \right\}$

1.3.20. $\varphi_2 = \frac{1}{6\varphi} \left\{ \frac{1}{2} + \sum_{n=1}^{N} \frac{1}{(2n+2)!} [(kL)^2]^n \right\}$

1.3.21. $\varphi_3 = \frac{1}{4\varphi} \left\{ \frac{1}{3} + \sum_{n=1}^{N} \frac{2(n+1)}{(2n+3)!} [(kL)^2]^n \right\}$

1.3.22. $\varphi_4 = \frac{1}{2\varphi} \left\{ \frac{1}{6} + \sum_{n=1}^{N} \frac{1}{(2n+3)!} [(kL)^2]^n \right\}$

1.3.23. $M_A^F = Model.Elements.FixedMomentA(e)$

1.3.24. $M_B^F = Model.Elements.FixedMomentB(e)$

1.3.25. $s_{ii} = 4\,\varphi_3$

1.3.26. $s_{ij} = 2\,\varphi_4$

1.3.27. $M_A = \frac{EI}{L_f}(s_{ii}\theta_A + s_{ij}\theta_B) + M_A^F$

1.3.28. $M_B = \frac{EI}{L_f}(s_{ii}\theta_B + s_{ij}\theta_A) + M_B^F$

1.3.29. $\boldsymbol{r} = \begin{bmatrix} M_A \\ M_B \\ P \end{bmatrix}$

1.3.30. $\boldsymbol{L_r} = \begin{bmatrix} -\sin\beta/L_f & -\sin\beta/L_f & -\cos\beta \\ \cos\beta/L_f & \cos\beta/L_f & -\sin\beta \\ 1 & 0 & 0 \\ \sin\beta/L_f & \sin\beta/L_f & \cos\beta \\ -\cos\beta/L_f & -\cos\beta/L_f & \sin\beta \\ 0 & 1 & 0 \end{bmatrix}$

1.3.31. $\boldsymbol{k} =$

$$\begin{bmatrix} EA/L_f & 0 & 0 & -EA/L_f & 0 \\ 0 & 12EI\varphi_1/L_f^3 & 6EI\varphi_2/L_f^2 & 0 & -12EI\varphi_3/L_f^3 \\ 0 & 6EI\varphi_2/L_f^2 & 4EI\varphi_3/L_f & 0 & -6EI\varphi_2/L_f^2 \\ -EA/L_f & 0 & 0 & EA/L_f & 0 \\ 0 & -12EI\varphi_1/L_f^3 & -6EI\varphi_2/L_f^2 & 0 & 12EI\varphi_1/L_f^3 \\ 0 & 6EI\varphi_2/L_f^2 & 2EI\varphi_4/L_f^3 & 0 & -6EI\varphi_2/L_f^2 \end{bmatrix}$$

1.3.32. $\boldsymbol{K}_{(\boldsymbol{elementDof},\ \boldsymbol{elementDof})} = \boldsymbol{K}_{(\boldsymbol{elementDof},\ \boldsymbol{elementDof})} + \boldsymbol{L_k^T k L_k}$

1.3.33. $\boldsymbol{R}_{(\boldsymbol{elementDof})} = \boldsymbol{R}_{(\boldsymbol{elementDof})} + \boldsymbol{L_r\, r}$

1.4. End-For

2. End-Function

It is worth mentioning that in the above algorithm for compute stability factors ($\varphi_1$to φ_4) just ten terms of series ($N = 10$) will converge to a high degree of accuracy [1]. This function is as well as used for the other analysis methods. In the simple load control since the amount of $\boldsymbol{R}$ is determined therefore the $\boldsymbol{R}$calculated by the $TangentStiffness$ does not require.

2.2. Newton-Raphson Load Control Method

The pseudo-codes algorithm of Newton-Raphson load control method with constant load increment factor is presented in this section:

1. Start
2. Initialization:
2.1. Describe the characteristics of frame structure as $\boldsymbol{Model}$
2.2. Take a value for *Number of Steps* ($\lambda = \frac{1}{Number\ of\ Steps}$)
2.3. Take a value for *Maximum Error of unbalance load*
2.4. Take a value for *Maximum number of Iterations*
3. $[\boldsymbol{K_e}, \boldsymbol{R_e}, \boldsymbol{D_e}] = LinearAnalysis(\boldsymbol{Model})$
4. $\boldsymbol{K_0} = \boldsymbol{K_e}$
5. $\boldsymbol{R_0} = \boldsymbol{0}$
6. $\boldsymbol{D_0} = \boldsymbol{0}$
7. $\Delta\boldsymbol{R} = \lambda\boldsymbol{R_e}$
8. For $i = 1$ to *Number of Steps*:
8.1. $\boldsymbol{R_i} = \boldsymbol{R_{i-1}} + \Delta\boldsymbol{R}$
8.2. $\boldsymbol{D_i^1} = \boldsymbol{D_{i-1}}$
8.3. $\boldsymbol{R_i^1} = \boldsymbol{R_{i-1}}$
8.4. $\boldsymbol{K_i^1} = \boldsymbol{K_{i-1}}$
8.5. $\boldsymbol{Q_i^1} = \boldsymbol{0}$
8.6. $\Delta\boldsymbol{D_i^1} = (\boldsymbol{K_i^1})^{-1}\Delta\boldsymbol{R}$
8.7. *Converged* = False
8.8. While Not *Converged*:
8.8.1. $j = j + 1$
8.8.2. $\boldsymbol{D_i^j} = \boldsymbol{D_i^{j-1}} + \Delta\boldsymbol{D_i^{j-1}}$
8.8.3. $[\boldsymbol{K_i^j}, \boldsymbol{R_i^j}] = TangentStiffness(\boldsymbol{Model}, \boldsymbol{D_i^j})$

8.8.4. $\boldsymbol{Q_i^j} = \boldsymbol{R_i} - \boldsymbol{R_i^j}$

8.8.5. If $\left(\boldsymbol{Q_i^j}\right)^{\boldsymbol{T}} . \boldsymbol{Q_i^j} < Maximum\ Error\ of\ unbalance\ load$ Then *Converged* = True

8.8.6. If $j \geq Maximum\ number\ of\ Iterations$ Then Exit-While

8.9. End-While

8.10. $\boldsymbol{K_i} = \boldsymbol{K_i^j}$

8.11. $\boldsymbol{D_i} = \boldsymbol{D_i^j}$

9. End-For

10. Show Output Results

11. End.

2.3. Displacement Control Method

The pseudo-codes algorithm of Displacement Control method is presented in this section:

1. Start

2. Initialization:

2.1. Describe the characteristics of frame structure as $\boldsymbol{Model}$

2.2. Take a value for the *prescribed displacement increment* (λ)

2.3. Take a value for *Maximum Error of unbalance load*

2.4. Take a value for *Maximum number of Iterations*

2.5. Select one of the active DOF as *m*

3. $[\boldsymbol{K_e}, \boldsymbol{R_e}, \boldsymbol{D_e}] = LinearAnalysis(\boldsymbol{Model})$

4. $\Delta\boldsymbol{D} = \lambda\boldsymbol{D_e}$

5. $\boldsymbol{K_0} = \boldsymbol{K_e}$

6. $\boldsymbol{R_0} = \boldsymbol{0}$

7. $\boldsymbol{D_0} = \boldsymbol{0}$

8. $i = 1$

9. *EndOfLoad* = False

10. While Not *End of Loading*

10.1. $\Delta\boldsymbol{D_i^1} = \Delta\boldsymbol{D}$

10.2. $\Delta\boldsymbol{D'}_{\boldsymbol{i}}^{\boldsymbol{1}} = (\boldsymbol{K_{i-1}})^{-\boldsymbol{1}}\boldsymbol{R_e}$

10.3. $\lambda_i^1 = \Delta D_i^1(m)/\Delta D'^1_i(m)$

10.4. $\Delta\boldsymbol{R_i} = \lambda_i^1\boldsymbol{R_e}$

10.5. $\boldsymbol{R_i} = \boldsymbol{R_{i-1}} + \Delta\boldsymbol{R_i}$

10.6. $\boldsymbol{D_i} = \boldsymbol{D_{i-1}} + \Delta\boldsymbol{D}$

10.7. $j = 1$

10.8. $\boldsymbol{R_i^1} = \boldsymbol{R_i}$

10.9. $\boldsymbol{K_i^1} = \boldsymbol{K_{i-1}}$

10.10. $\boldsymbol{D_i^1} = \boldsymbol{D_i}$

10.11. $\boldsymbol{Q_i^1} = \boldsymbol{0}$

10.12. *Converged* = False

10.13. While Not *Converged*:

10.13.1. $j = j + 1$

10.13.2. $[\boldsymbol{K_i^j}, \boldsymbol{R_i^j}] = TangentStiffness(\boldsymbol{Model}, \boldsymbol{D_i^{j-1}})$

10.13.3. $\boldsymbol{Q_i^j} = \boldsymbol{R_i^{j-1}} - \boldsymbol{R_i^j}$

10.13.4. $\Delta\boldsymbol{D'}_{\boldsymbol{i}}^{\boldsymbol{j}} = (\boldsymbol{K_i^j})^{-\boldsymbol{1}}\boldsymbol{R_e}$

10.13.5. $\Delta\boldsymbol{D''}_{\boldsymbol{i}}^{\boldsymbol{j}} = (\boldsymbol{K_i^j})^{-\boldsymbol{1}}\boldsymbol{Q_i^j}$

10.13.6. $\lambda_i^j = -\Delta D''^1_{im} / \Delta D'^1_{im}$

10.13.7. $\Delta\boldsymbol{D_i^j} = \lambda_i^j \ \Delta\boldsymbol{D'}_{\boldsymbol{i}}^{\boldsymbol{j}} + \Delta\boldsymbol{D''}_{\boldsymbol{i}}^{\boldsymbol{j}}$

10.13.8. $\boldsymbol{D_i^j} = \boldsymbol{D_i^{j-1}} + \Delta\boldsymbol{D_i^j}$

10.13.9. $\Delta\boldsymbol{R_i^j} = \lambda_i^j\boldsymbol{R_e}$

10.13.10. $\boldsymbol{R_i^j} = \boldsymbol{R_i^{j-1}} + \Delta\boldsymbol{R_i^j}$

10.13.11. If $(\boldsymbol{Q_i^j})^{\boldsymbol{T}}.\boldsymbol{Q_i^j} < Maximum\ Error\ of\ unbalance\ load$ Then *Converged* = True

10.13.12. If $j \geq Maximum\ number\ of\ Iterations$ Then Exit While

10.14. End-While

10.15. $\boldsymbol{K_i} = \boldsymbol{K_i^j}$

10.16. $\boldsymbol{R_i} = \boldsymbol{R_i^j}$

10.17. $\boldsymbol{D_i} = \boldsymbol{D_i^j}$

10.18. $i = i + 1$

10.19. If Any Component Of $\boldsymbol{R_i}$>$\boldsymbol{R_e}$Then *EndOfLoad* = True
11. End-While
12. Show Output Results
13. End.

2.4. Arc Length Control Method

The pseudo-codes algorithm of Arc Length Control method is presented in this section. This method indeed is a displacement control method can solve the snapback and snap-through behavior.

1. Start
2. Initialization:
2.1. Describe the characteristics of frame structure as $\boldsymbol{Model}$
2.2. Take a value for the *prescribed displacement increment* (λ)
2.3. Take a value for *Maximum Error of unbalance load*
2.4. Take a value for *Maximum number of Iterations*
3. $[\boldsymbol{K_e}, \boldsymbol{R_e}, \boldsymbol{D_e}] = LinearAnalysis(\boldsymbol{Model})$
4. $\Delta\boldsymbol{D} = \lambda\boldsymbol{D_e}$
5. $ds = \lambda * \sqrt{\boldsymbol{D_e^T} * \boldsymbol{D_e} + \boldsymbol{R_e^T} * \boldsymbol{R_e}}$
6. $\boldsymbol{K_0} = \boldsymbol{K_e}$
7. $\boldsymbol{R_0} = \boldsymbol{0}$
8. $\boldsymbol{D_0} = \boldsymbol{0}$
9. $i = 1$
10. *EndOfLoad* = False
11. While Not *End of Loading*
11.1. $\Delta\boldsymbol{D_i^1} = \Delta\boldsymbol{D}$
11.2. $\Delta\boldsymbol{D'}_i^{\boldsymbol{1}} = (\boldsymbol{K_{i-1}})^{-\boldsymbol{1}}\boldsymbol{R_e}$
11.3. $\lambda_i^1 = \sqrt{\dfrac{ds^2}{1+\left(\Delta\boldsymbol{D'}_i^{\boldsymbol{1}}\right)^T * \Delta\boldsymbol{D'}_i^{\boldsymbol{1}}}}$
11.4. $\Delta\boldsymbol{R_i} = \lambda_i^1\boldsymbol{R_e}$
11.5. $\boldsymbol{R_i} = \boldsymbol{R_{i-1}} + \Delta\boldsymbol{R_i}$
11.6. $\boldsymbol{D_i} = \boldsymbol{D_{i-1}} + \Delta\boldsymbol{D}$
11.7. $j = 1$
11.8. $\boldsymbol{R_i^1} = \boldsymbol{R_i}$
11.9. $\boldsymbol{K_i^1} = \boldsymbol{K_{i-1}}$
11.10. $\boldsymbol{D_i^1} = \boldsymbol{D_i}$
11.11. $\boldsymbol{Q_i^1} = \boldsymbol{0}$
11.12. *Converged* = False

11.13. While Not *Converged*:
11.13.1. $j = j + 1$
11.13.2. $[\boldsymbol{K}_i^j ,\ \boldsymbol{R}_i^j] = TangentStiffness(\boldsymbol{Model} ,\ \boldsymbol{D}_i^{j-1})$
11.13.3. $\boldsymbol{Q}_i^j = \boldsymbol{R}_i^{j-1} - \boldsymbol{R}_i^j$
11.13.4. $\Delta \boldsymbol{D}'^j_i = \left(\boldsymbol{K}_i^j\right)^{-1} \boldsymbol{R}_e$
11.13.5. $\Delta \boldsymbol{D}''^j_i = \left(\boldsymbol{K}_i^j\right)^{-1} \boldsymbol{Q}_i^j$
11.13.6. $\lambda_i^j = -\frac{\left(\Delta \boldsymbol{D}'^1_i\right)^T \Delta \boldsymbol{D}''^j_i}{\left(\Delta \boldsymbol{D}'^1_i\right)^T \Delta \boldsymbol{D}'^j_i}$
11.13.7. $\Delta \boldsymbol{D}_i^j = \lambda_i^j\ \Delta \boldsymbol{D}'^j_i + \Delta \boldsymbol{D}''^j_i$
11.13.8. $\boldsymbol{D}_i^j = \boldsymbol{D}_i^{j-1} + \Delta \boldsymbol{D}_i^j$
11.13.9. $\Delta \boldsymbol{R}_i^j = \lambda_i^j \boldsymbol{R}_e$
11.13.10. $\boldsymbol{R}_i^j = \boldsymbol{R}_i^{j-1} + \Delta \boldsymbol{R}_i^j$
11.13.11. If $\left(\boldsymbol{Q}_i^j\right)^{\boldsymbol{T}} . \boldsymbol{Q}_i^j < Maximum\ Error\ of\ unbalance\ load$ Then *Converged* = True
11.13.12. If $j \geq Maximum\ number\ of\ Iterations$ Then Exit While
11.14. End-While
11.15. $\boldsymbol{K}_i = \boldsymbol{K}_i^j$
11.16. $\boldsymbol{R}_i = \boldsymbol{R}_i^j$
11.17. $\boldsymbol{D}_i = \boldsymbol{D}_i^j$
11.18. $i = i + 1$
11.19. If Any Component Of $\boldsymbol{R}_i$>$\boldsymbol{R}_e$Then *EndOfLoad* = True
12. End-While
13. Show Output Results
14. End.

2.5. Work Control Method

The pseudo-codes algorithm of Work Control method is presented in this section.This method like ALC is a displacement control method and also can solve the snapback and snap-through behavior.

1. Start
2. Initialization:
2.1. Describe the characteristics of frame structure as $\boldsymbol{Model}$
2.2. Take a value for the *prescribed displacement increment* (λ)
2.3. Take a value for *Maximum Error of unbalance load*
2.4. Take a value for *Maximum number of Iterations*

3. $[\boldsymbol{K_e}, \boldsymbol{R_e}, \boldsymbol{D_e}] = LinearAnalysis(\boldsymbol{Model})$
4. $\Delta \boldsymbol{D} = \lambda \boldsymbol{D_e}$
5. $dW = \lambda * \Delta \boldsymbol{D}^{\boldsymbol{T}} * \boldsymbol{R_e}$
6. $\boldsymbol{K_0} = \boldsymbol{K_e}$
7. $\boldsymbol{R_0} = \boldsymbol{0}$
8. $\boldsymbol{D_0} = \boldsymbol{0}$
9. $i = 1$
10. *EndOfLoads* = False
11. While Not *End of Loading*

11.1. $\Delta \boldsymbol{D}_{\boldsymbol{i}}^{\boldsymbol{1}} = \Delta \boldsymbol{D}$

11.2. $\Delta \boldsymbol{D'}_{\boldsymbol{i}}^{\boldsymbol{1}} = (\boldsymbol{K_{i-1}})^{-\boldsymbol{1}} \boldsymbol{R_e}$

11.3. $\lambda_i^1 = \sqrt{\frac{dW}{\left(\Delta \boldsymbol{D'}_i^1\right)^T * \boldsymbol{R_e}}}$

11.4. $\Delta \boldsymbol{R_i} = \lambda_i^1 \boldsymbol{R_e}$

11.5. $\boldsymbol{R_i} = \boldsymbol{R_{i-1}} + \Delta \boldsymbol{R_i}$

11.6. $\boldsymbol{D_i} = \boldsymbol{D_{i-1}} + \Delta \boldsymbol{D}$

11.7. $j = 1$

11.8. $\boldsymbol{R}_{\boldsymbol{i}}^{\boldsymbol{1}} = \boldsymbol{R_i}$

11.9. $\boldsymbol{K}_{\boldsymbol{i}}^{\boldsymbol{1}} = \boldsymbol{K_{i-1}}$

11.10. $\boldsymbol{D}_{\boldsymbol{i}}^{\boldsymbol{1}} = \boldsymbol{D_i}$

11.11. $\boldsymbol{Q}_{\boldsymbol{i}}^{\boldsymbol{1}} = \boldsymbol{0}$

11.12. *Converged* = False
11.13. While Not *Converged*:
11.13.1. $j = j + 1$

11.13.2. $\left[\boldsymbol{K}_{\boldsymbol{i}}^{\boldsymbol{j}}, \boldsymbol{R}_{\boldsymbol{i}}^{\boldsymbol{j}}\right] = TangentStiffness\left(\boldsymbol{Model}, \boldsymbol{D}_{\boldsymbol{i}}^{\boldsymbol{j-1}}\right)$

11.13.3. $\boldsymbol{Q}_{\boldsymbol{i}}^{\boldsymbol{j}} = \boldsymbol{R}_{\boldsymbol{i}}^{\boldsymbol{j-1}} - \boldsymbol{R}_{\boldsymbol{i}}^{\boldsymbol{j}}$

11.13.4. $\Delta \boldsymbol{D'}_{\boldsymbol{i}}^{\boldsymbol{j}} = \left(\boldsymbol{K}_{\boldsymbol{i}}^{\boldsymbol{j}}\right)^{-\boldsymbol{1}} \boldsymbol{R_e}$

11.13.5. $\Delta \boldsymbol{D''}_{\boldsymbol{i}}^{\boldsymbol{j}} = \left(\boldsymbol{K}_{\boldsymbol{i}}^{\boldsymbol{j}}\right)^{-\boldsymbol{1}} \boldsymbol{Q}_{\boldsymbol{i}}^{\boldsymbol{j}}$

11.13.6. $\lambda_i^j = -\frac{\left(\Delta \boldsymbol{D}''^{j}_{i}\right)^{T} * \boldsymbol{R_e}}{\left(\Delta \boldsymbol{D}'^{j}_{i}\right)^{T} * \boldsymbol{R_e}}$

11.13.7. $\Delta \boldsymbol{D}_i^j = \lambda_i^j \ \Delta \boldsymbol{D}'^{j}_{i} + \Delta \boldsymbol{D}''^{j}_{i}$

11.13.8. $\boldsymbol{D}_i^j = \boldsymbol{D}_i^{j-1} + \Delta \boldsymbol{D}_i^j$

11.13.9. $\Delta \boldsymbol{R}_i^j = \lambda_i^j \boldsymbol{R_e}$

11.13.10. $\boldsymbol{R}_i^j = \boldsymbol{R}_i^{j-1} + \Delta \boldsymbol{R}_i^j$

11.13.11. If $\left(\boldsymbol{Q}_i^j\right)^{\boldsymbol{T}} . \boldsymbol{Q}_i^j < Maximum\ Error\ of\ unbalance\ load$ Then *Converged* = True

11.13.12. If $j \geq Maximum\ number\ of\ Iterations$ Then Exit While

11.14. End-While

11.15. $\boldsymbol{K_i} = \boldsymbol{K}_i^j$

11.16. $\boldsymbol{R_i} = \boldsymbol{R}_i^j$

11.17. $\boldsymbol{D_i} = \boldsymbol{D}_i^j$

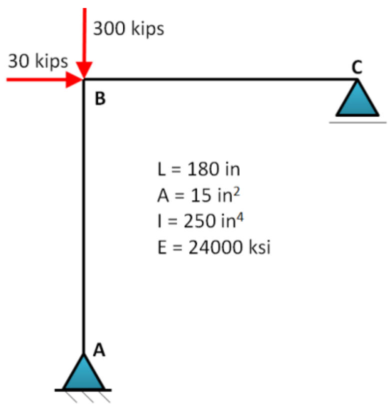
11.18. $i = i + 1$

11.19. If Any Component Of $\boldsymbol{R_i} > \boldsymbol{R_e}$ Then *EndOfLoads* = True

12. End-While
13. Show Output Results
14. End.

3. Numerical Study

Figure 1. Frame structure for numerical study

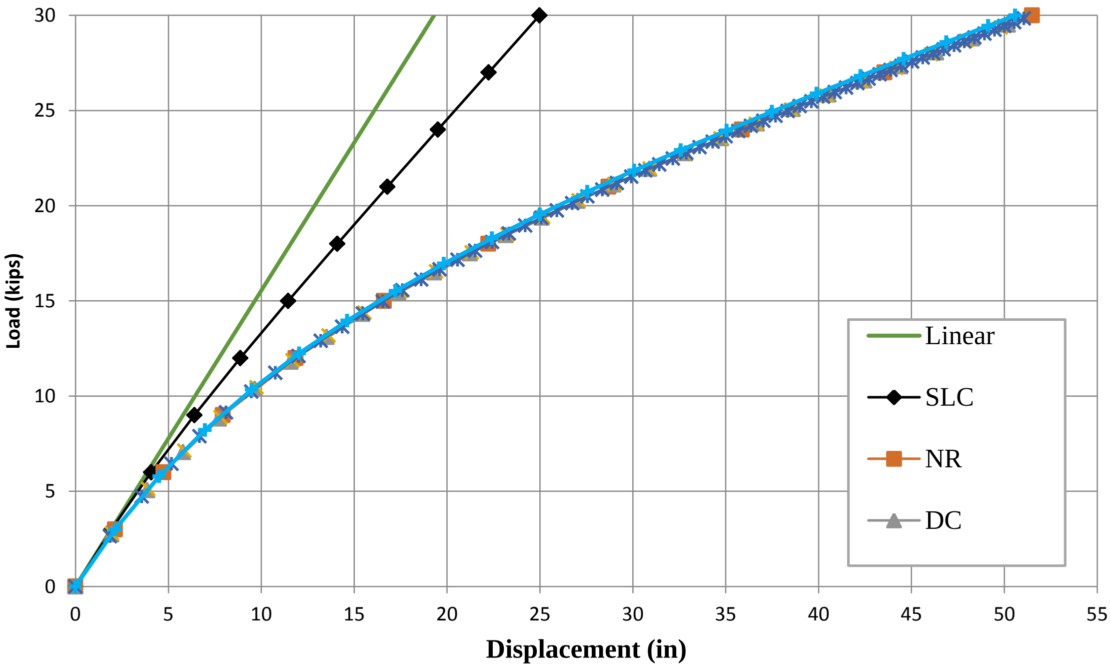


All of solution algorithms that mentioned in this study have been implemented in MATLAB software [4].

For numerical study an L shape structure has been considered. The material and cross-sectional properties of two elements of this structure are same (Figure 1).

The analysis with different solution algorithms has been performed. The load-displacement curve of horizontal component of B node has been depicted in Figure 2. Also to verification, the structure was analyzed in MASTAN software [5] . According to the Figure 2 can be seen that second-order elastic analysis with different solution algorithms have very good match on the results of MASTAN software. Also it is noteworthy that because of accumulated drift of error, SLC method in compared of other solution algorithms that using the iterations has considerable difference.

Figure 2. Load-displacement curve of horizontal component of B node



4. Conclusions

In this paper the various solution algorithm methods (as pseudo-codes) for second-order elastic frame analysis is presented. By implementing this solution algorithms performing a numerical example and compared it with the MASTAN software, the validation of this pseudo-codes have been demonstrated.

5. References

[1]. **Ferreira, A.J.M.***MATLAB Codes for Finite Element Analysis, Solids and Structures.* Universidade do Porto, Portugal : Springer, 2009. ISBN 978-1-4020-9199-5.

[2]. **Kwon, Young W. and Bang, Hychoong.***The Finite Element Method using MATLAB.* Boca Raton, Florida : CRC Press, 1997. ISBN 0-8493-9653-0.

[3]. **Chen, Wai-Fah and Lui, E. M.***Stability Design of Steel Frames.* Florida : CRC Press, 2000. ISBN 0-8493-8606-3.
[4]. *MATLAB, The Language of Technical Computing.* s.l. : MathWorks, Inc, 2010.
[5]. **Ziemian, R. D. and McGuire, W.***MASTAN2 version 3.3.* s.l. : WILEY, 2010.