Finite Element Method

Project 1 & 2

機械系 陳建鳴 E14026046

一、 公式整理

1. Bar Problem

Shape Function (for n node element)

$$N_i(x) = \frac{\prod_{k=1,k\neq i}^{n}(x - x_k^e)}{\prod_{k=1,k\neq i}^{n}(x_i - x_k^e)} \tag{1}$$

Stiffness Matrix

$$\boldsymbol{K}^e = \int_{x_1^e}^{x_2^e} (\boldsymbol{B}^e)^T (A^e E^e) \boldsymbol{B}^e dx \tag{2}$$

Force Vector

$$\boldsymbol{f}^e = (\boldsymbol{N}^e)^T \boldsymbol{A}^e \bar{t}|_{x=0} + \int_{x_1^e}^{x_2^e} (\boldsymbol{N}^e)^T b dx \tag{3}$$

2. Beam Problem

Shape Function (for 2 node element)

$$\boldsymbol{N}^e = \begin{bmatrix} 1 - \frac{3x^2}{L^2} + \frac{2x^3}{L^3} & x - \frac{2x^2}{L} + \frac{x^3}{L^2} & \frac{3x^2}{L^2} - \frac{2x^3}{L^3} & -\frac{x^2}{L} + \frac{x^3}{L^2} \end{bmatrix} \tag{1}$$

Stiffness Matrix

$$\boldsymbol{K} = \frac{EI}{(L^e)^3} \begin{bmatrix} 12 & 6L^e & -12 & 6L^e \\ 6L^e & 4(L^e)^2 & -6L^e & 2(L^e)^2 \\ -12 & -6L^e & 12 & -6L^e \\ 6L^e & 2(L^e)^2 & -6L^e & 4(L^e)^2 \end{bmatrix} \tag{2}$$

Force Vector

$$\boldsymbol{f}^e = (\boldsymbol{N}^e)^T \boldsymbol{A}^e \bar{t}|_{x=0} + \int_{x_1^e}^{x_2^e} (\boldsymbol{N}^e)^T b dx \tag{3}$$

二、 程式說明

1. Programming Language – Golang

Go，又稱golang，是Google 開發的一種靜態強型別、編譯型，並發型，並具有垃圾回收功能的程式語言。—維基百科

A. Data Structure

```
// Linear-element FEM solver for bar
// k: local stifness matrix
// K: global stifness matrix
// Ne: amount of element
// No: order of shape function
// Ng: amount of gaussian points
// Le: lenth of each element
// E: Young's modulus of each element
// A: area of each element
// u: displacement vector
// f: force vector
// uNod, uVal: displacement boundary condition
// fNod, fVal: force boundary condition
type FEMsolver1dBar struct {
  k, K *mat64.SymDense
  Ne, No, Ng int
  Le, E, A, u, f *mat64.Vector
  uNod, fNod []int
  uVal, fVal []float64
}
// Linear-element FEM solver for beam
// k: local stifness matrix
// K: global stifness matrix
// Ne: amount of element
// No: order of shape function
// Ng: amount of gaussian points
// Le: lenth of each element
// E: Young's modulus of each element
// A: area of each element
// d: displacement vector
// f: force vector
// dNod, dVal: displacement boundary condition
// fNod, fVal: force boundary condition
type FEMsolver1dBeam struct {
  k, K *mat64.SymDense
  Ne, No, Ng int
  Le, E, I, d, f *mat64.Vector
  dNod, fNod []int
  dVal, fVal []float64
}
```

B. Functions

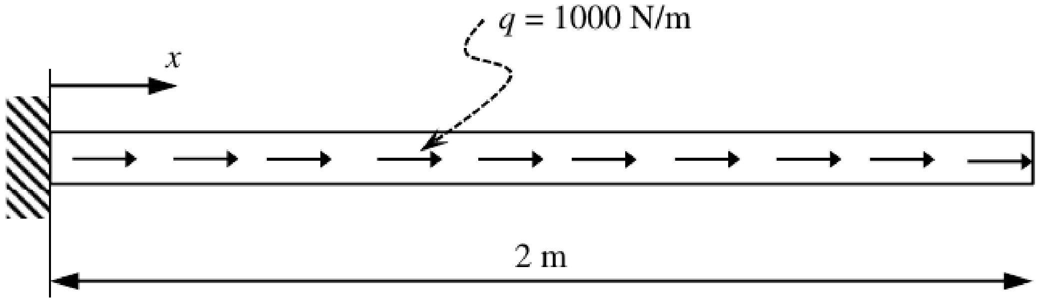
```
// Build a vector from nod and val
// nod: an array of node number which has value
// val: value of the node
func BuildVec(nod []int, val []float64, vec *mat64.Vector)

// Calculate Gaussian Quadracture of f from a to b with num
Gaussian point
func GausQuad(f func(float64) float64, a, b float64, num int)
float64

// Get the value of Gaussian points and the weight. Currently
maximun num is 4.
func getGausQuadPoint(num int) (pt, w []float64)

// Change the interval from a, b to -1, 1
func changeInterval(f func (float64) float64, a, b float64) func
(float64) float64

// Get differential of a function by Symmetric derivative
func d(f func (float64) float64) func (float64) float64

// Create a FEMsolver1dBar
func NewFEMsolver1dBar(No, Ne int, Le, E, A, u, f *mat64.Vector,
uNod, fNod []int, uVal, fVal []float64) *FEMsolver1dBar

// Create a FEMsolver1dBar with const Le, E and A
func NewFEMsolver1dBarConstLeEA(No, Ne int, Le, E, A float64, u, f
*mat64.Vector, uNod, fNod []int, uVal, fVal []float64)
*FEMsolver1dBar

// Create a FEMsolver1dBeam
func NewFEMsolver1dBeam(No, Ne int, Le, E, I, d, f *mat64.Vector,
dNod, fNod []int, dVal, fVal []float64) *FEMsolver1dBeam

// Create a FEMsolver1dBeam with const Le, E and I
func NewFEMsolver1dBeamConstLeEI(No, Ne int, Le, E, I float64, d, f
*mat64.Vector, dNod, fNod []int, dVal, fVal []float64)
*FEMsolver1dBeam
```

For both bar and beam, I have

```
// Return shape function of e-th element
func (fem *FEMsolver1dBar) NElem(j, e int) func(float64) float64
func (fem *FEMsolver1dBeam) NElem(j, e int) func(float64) float64

// Return derivative shape function of e-th element
func (fem *FEMsolver1dBar) BElem(j, e int) func(float64) float64
func (fem *FEMsolver1dBeam) BElem(j, e int) func(float64) float64

// Add body force (or distributed force) b to the solver
func (fem *FEMsolver1dBar) AddBodyForce(b func(float64) float64, Ng
int)
func (fem *FEMsolver1dBeam) AddBodyForce(b func(float64) float64,
Ng int)

// Calculate local k matrix
func (fem *FEMsolver1dBar) CalcLocK()
func (fem *FEMsolver1dBeam) CalcLocK()

// Calculate the global K matrix
func (fem *FEMsolver1dBar) CalcK()
func (fem *FEMsolver1dBeam) CalcK()

// Solve the problem
func (fem *FEMsolver1dBar) Solve()
func (fem *FEMsolver1dBeam) Solve()

// Get stress at x
func (fem *FEMsolver1dBar) Stress(x float64) float64
func (fem *FEMsolver1dBeam) Stress(x float64) float64

// Get displacement at x
func (fem *FEMsolver1dBar) Disp(x float64) float64
func (fem *FEMsolver1dBeam) Disp(x float64) float64
```

三、 討論

1. Bar Problem



Cross section A = 0.01 m x 0.01 m =0.0001 m^2, Young's modulus E = 100 GPa

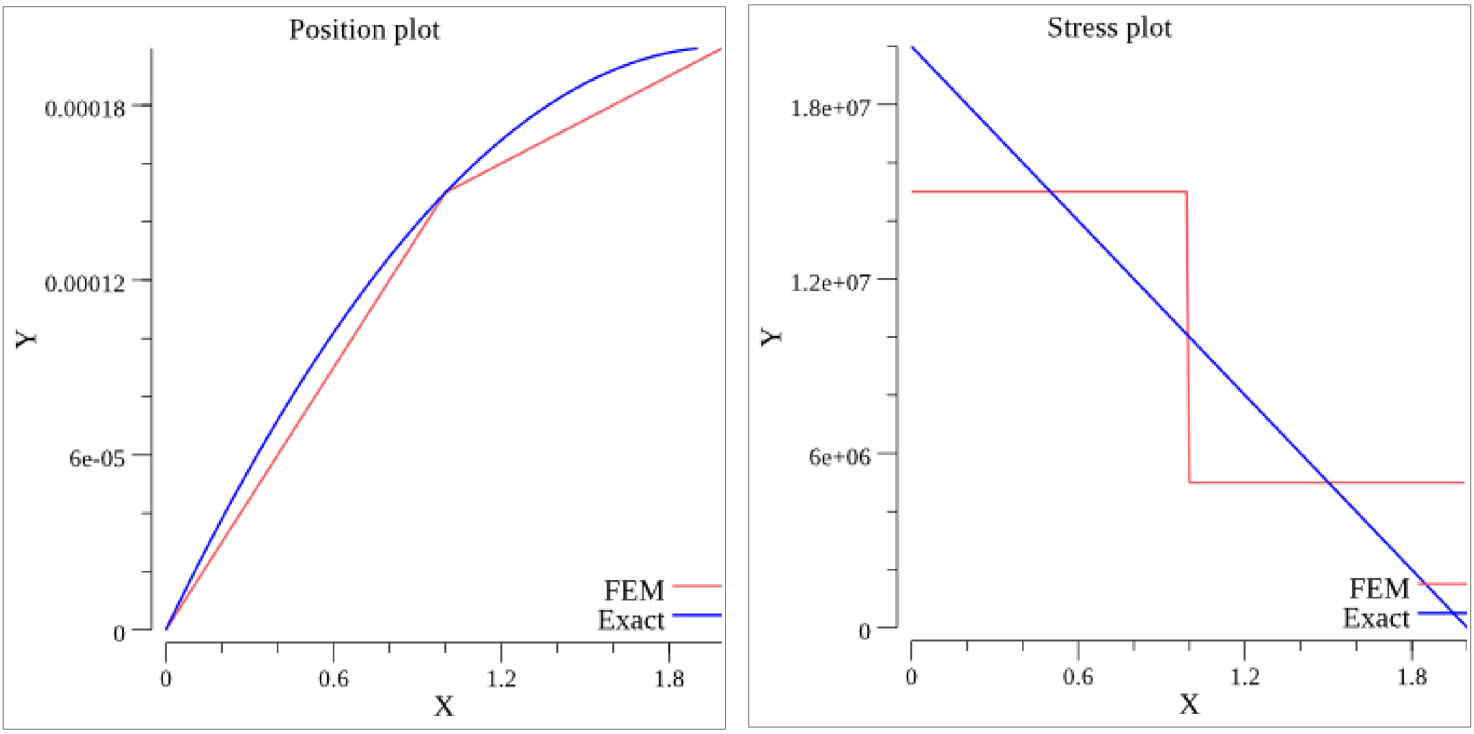
A. Compare with exact solution

		Two 2-node element		Eight 2-node element		Exact Solution
		Value	Error	Value	Error	Value
Disp (m)	d(0.5)	7.5e-5	14%	8.75e-5	0%	8.75e-5
	d(1.5)	1.75e-4	7%	1.875e-4	0%	1.875e-4
Stress (MPa)	s(0.5)	15	0%	13.75	8%	15
	s(1.5)	5	0%	3.75	25%	5

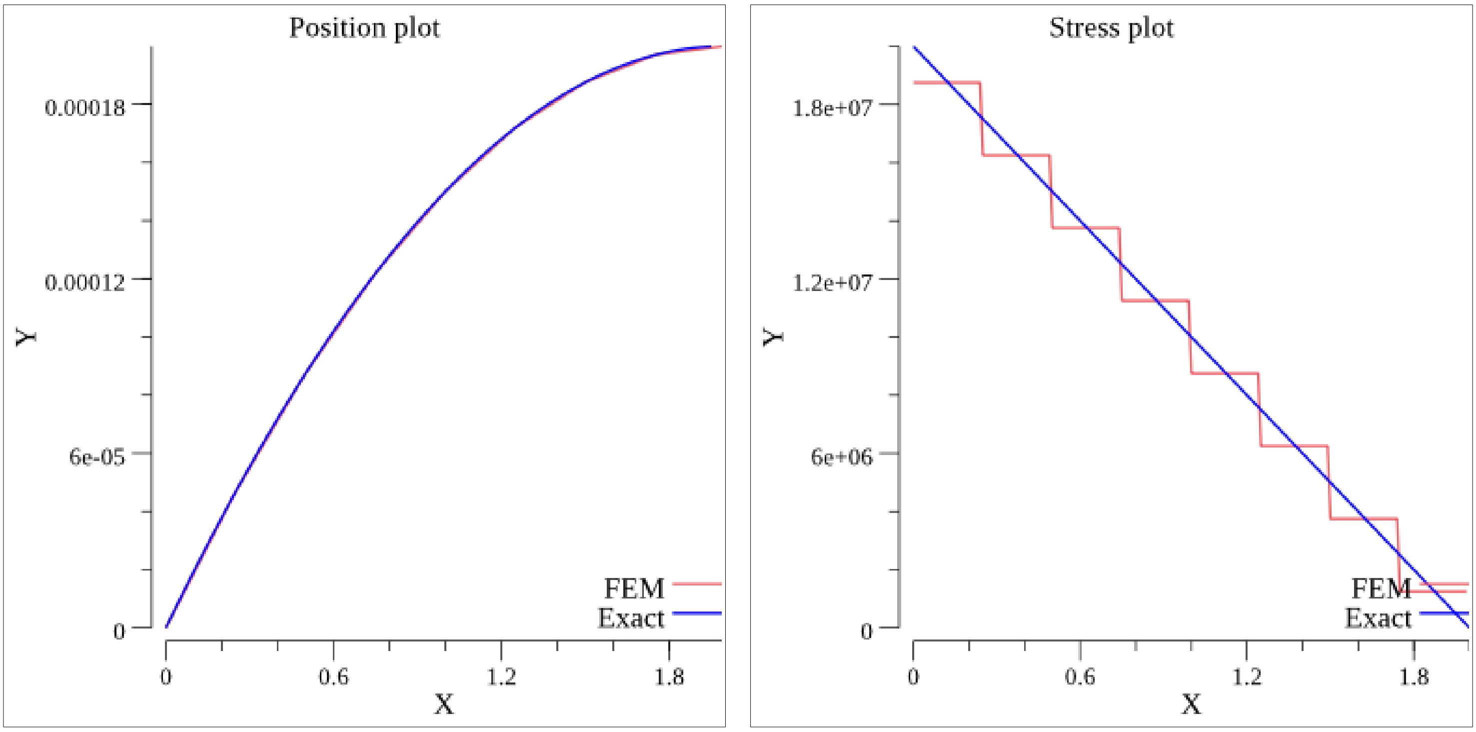
表一 兩節點元素

		Two 3-node element		Two 4-node element		Exact Solution
		Value	Error	Value	Error	Value
Disp (m)	d(0.5)	8.749e-5	0.011%	8.75e-5	0%	8.75e-5
	d(1.5)	1.8749e-4	0.005%	1.875e-4	0%	1.875e-4
Stress (MPa)	s(0.5)	14.9	0.067%	15	0%	15
	s(1.5)	4.9	0.2%	5	0%	5

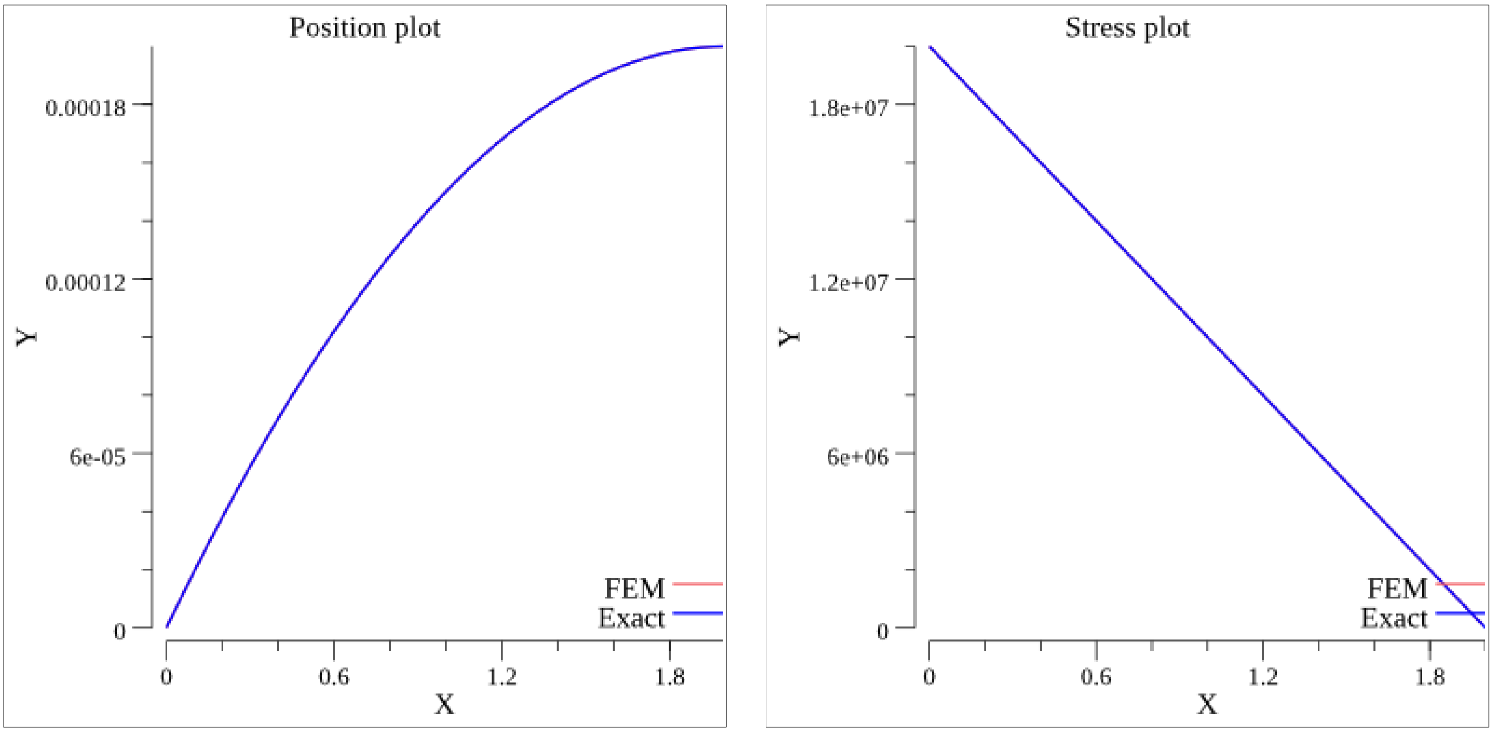
表二 三節點元素與四節點元素



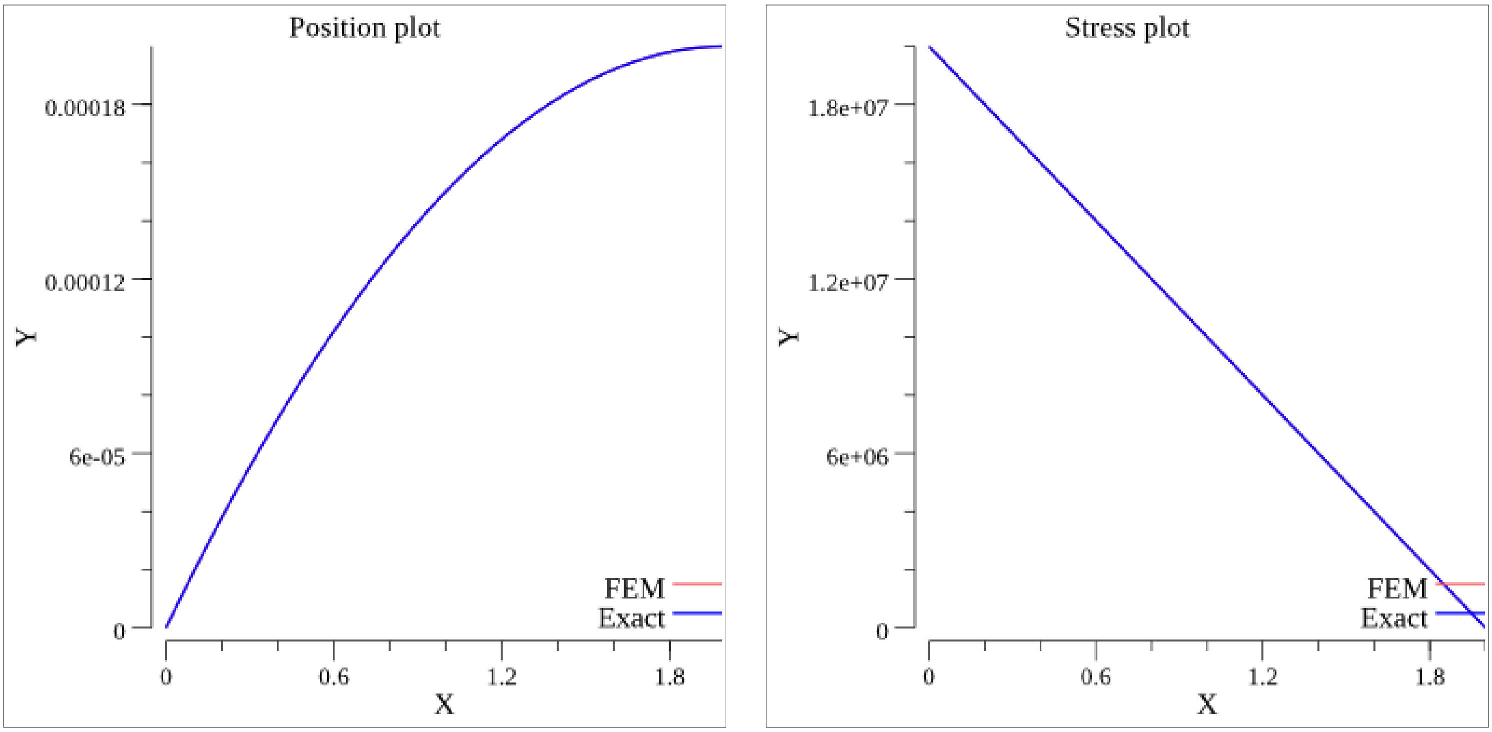
圖一 兩個兩節點元素



圖二 八個兩節點元素



圖三 兩個三節點元素



圖四 兩個四節點元素

B. Concentrated load and distributed load

在計算 Force vector 時，分佈力無法直接使用而必須用 Shape function 去近似，而集中力可直接考慮作用在節點上的力，若 Shape function 的階數小於分佈力則會造成誤差。若力為集中力要注意集中力必須作用在 mesh 的節點上，否則會造成集中力沒被考慮到。

C. How the number of element affect the results

如表一所示，兩個兩節點元素時，因為 x=0.5、1.5 在元素內部，且 Shape function 為線性，位移的誤差較大，而八個元素因為兩個點剛好位於節點上，位移的值是準的。

但考慮應力時，則是相反的結果，元素內部的應力值會比較準，節點上是最不準的。

以上可歸納出節點數量越多可讓節點分佈得更密集，位移的值也會更接近 Exact solution，同時節點跟節點間更接近也會縮小 Stress 的誤差。

D. How the type of element affect the results

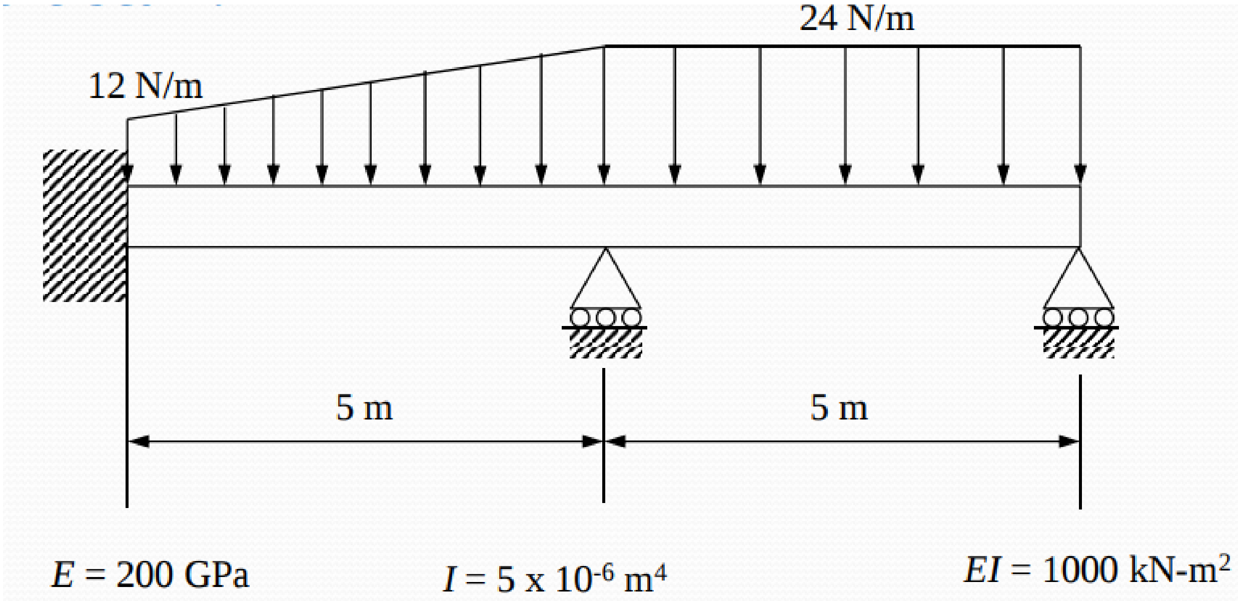
如表二所示，兩個三節點元素時位移和應力都已經十分接近 Exact solution，但仍有些許誤差，到了四節點元素時幾乎無誤差。

元素的節點數增加表示 Shape function 的階數增加，可以更準確的近似位移函數，以此題來說位移函數為二次曲線，理論上三節點元素就應該可以有很好的近似，誤差可能來自於高斯積分。

E. How to reduce the error when Young’s modulus or cross section varies with x

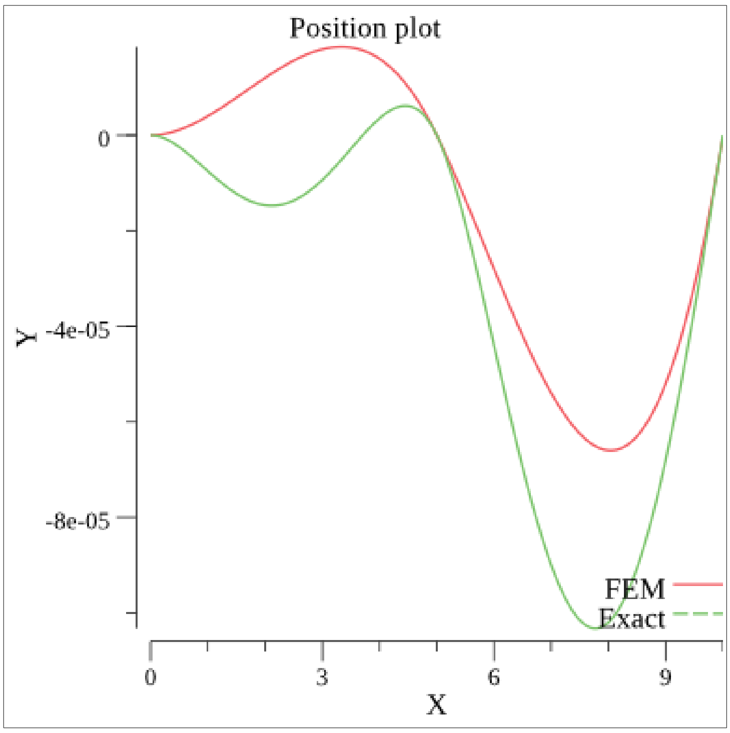
若節面積或楊氏模數會隨著 x 變化，會增加 Exact solution 的階次，可透過增加 Shape function 的階次、增加元素數量、增加高斯機分點數量等方法來降低誤差

2. Beam Problem

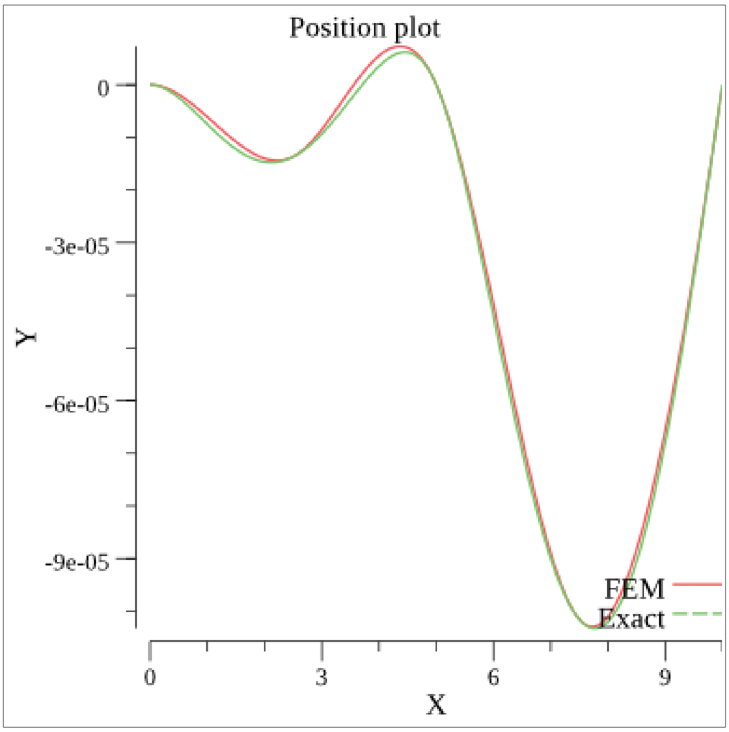


A. Compare with exact solution (Problem in ch7)

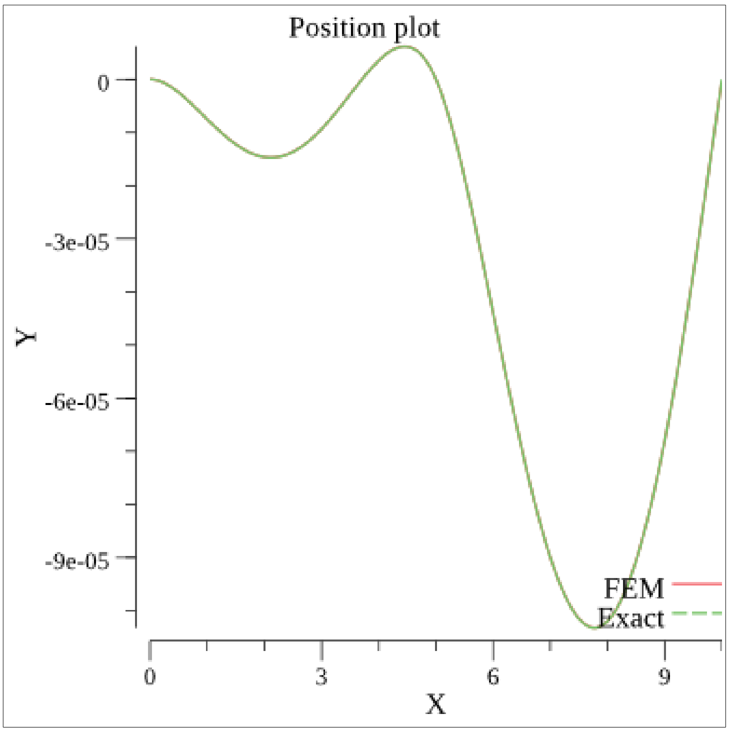
- Displacement at node



圖一 兩個元素

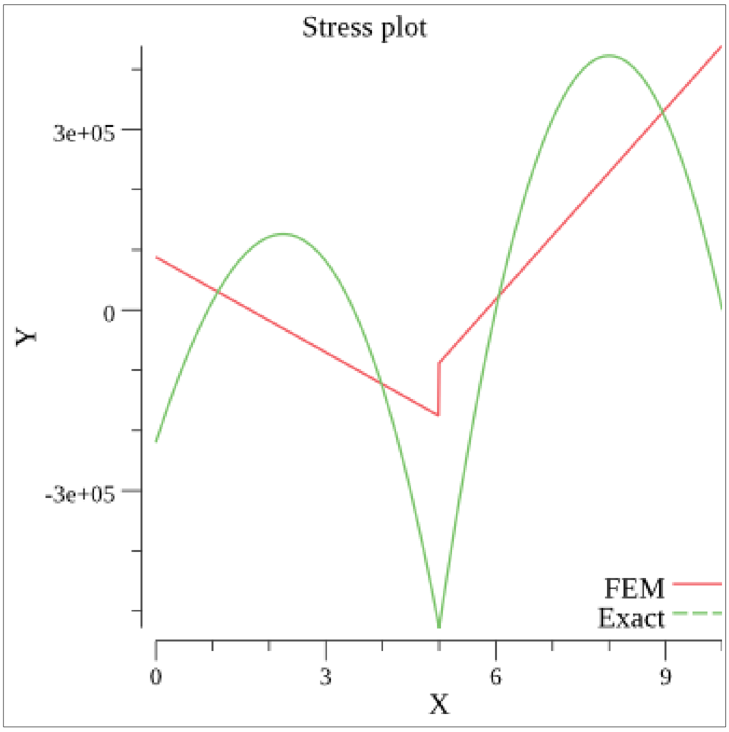


圖二 四個元素

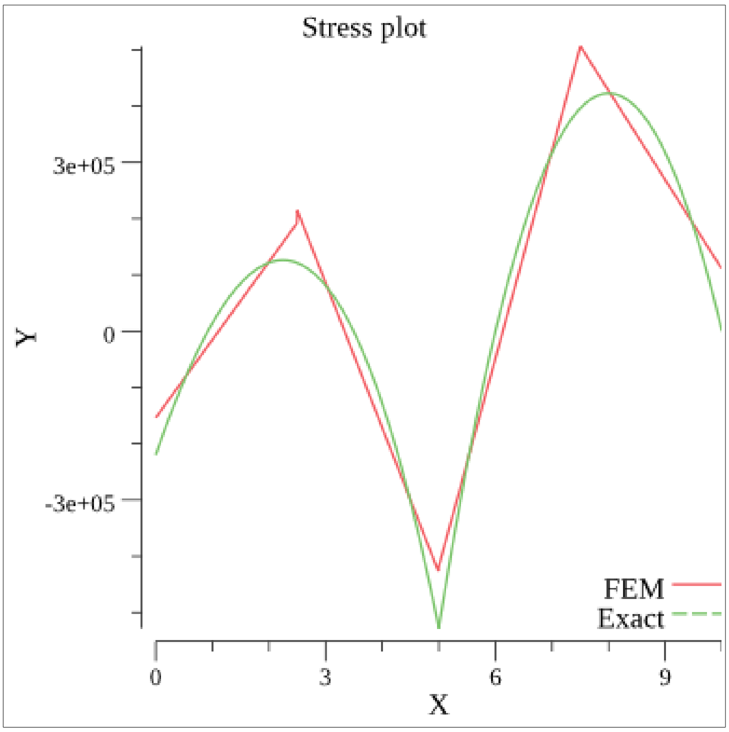


圖三 十個元素

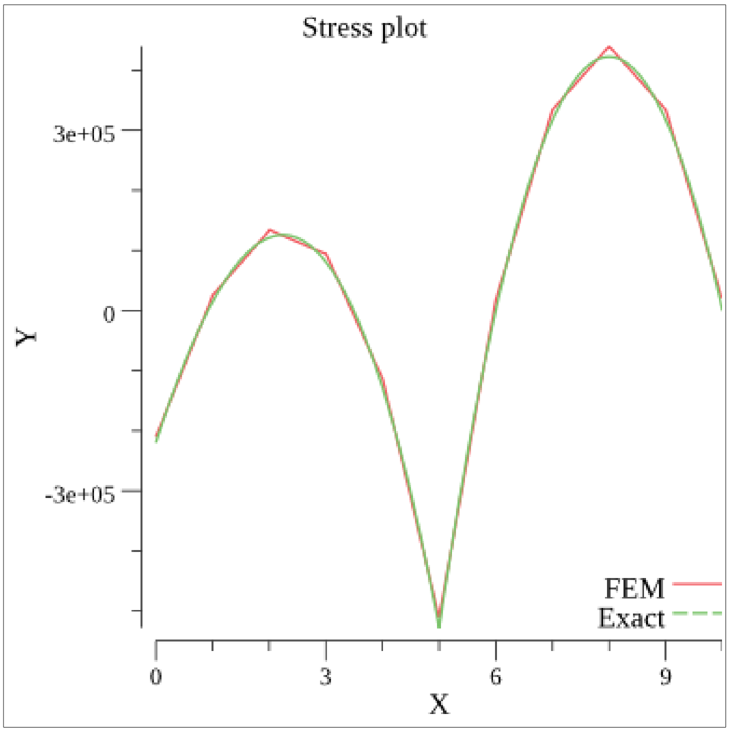
- Stresses at node and Gaussian points



圖四 兩個元素



圖五 四個元素



圖六 十個元素

B. How the number of element affect the results

- Displacement

從圖一～圖三可以發現元素的數量越多位移越接近 Exact solution，且在節點上的值會比較準確、在元素內部的點誤差較大。從圖二可以觀察到，前後半段比較的話後半段的誤差較小，因為前半段的分佈力是一次式，而後半段為常數，次方差一次，積分後得到位移分別為五次及四次，後者與 Shape function 的差異較小，因此誤差也會較小。

- Stress

從圖四～圖五同樣可發現元素的數量越多應力越接近 Eaxct solution，且在高斯積分點上的值會比較準確、在節點上的值誤差最大。從途中可以發現在節點處會有值不連續的狀況，這是因為應力值與位移的兩次微分有關，且不同的元素有自己的 Shape function 微分兩次後的值會有所差異。