Exercise 2 – Pg 143

$n_{int} = 3$ Derive gauss quadrature rule.

3 integration points $\Rightarrow$ 5th order polynomial

$$g(\xi) = \alpha_0 + \alpha_1 \xi + \alpha_2 \xi^2 + \alpha_3 \xi^3 + \alpha_4 \xi^4 + \alpha_5 \xi^5$$

The exact integral is:

$$\int_{-1}^{1} g(\xi)\, d\xi = \left[\alpha_0 \xi + \cancel{\frac{\alpha_1 \xi^2}{2}} + \frac{\alpha_2 \xi^3}{3} + \cancel{\frac{\alpha_3 \xi^4}{4}} + \frac{\alpha_4 \xi^5}{5} + \cancel{\frac{\alpha_5 \xi^6}{6}} \right]_{-1}^{1}$$

$$= 2\alpha_0 + \frac{2}{3}\alpha_2 + \frac{2}{5}\alpha_1$$

This is to be equal to: (Note: $\tilde{\xi}_1 = -\tilde{\xi}_3$, $\tilde{\xi}_2 = 0$, $W_1 = W_3$)

$$\sum_{l=1}^{3} g(\tilde{\xi}_l) W_l =$$

$$= W_1 g(\tilde{\xi}_1) + W_2 g(\tilde{\xi}_2) + W_3 g(\tilde{\xi}_3) =$$

$$= W_1 \left[g(\tilde{\xi}_1) + g(-\tilde{\xi}_1) \right] + W_2 g(0)$$

$$= W_1 \left\{ \begin{array}{l} \alpha_0 + \cancel{\alpha_1 \tilde{\xi}_1} + \alpha_2 \tilde{\xi}_1^2 - / + \alpha_4 \tilde{\xi}_1^2 + / \\ \alpha_0 - \cancel{\alpha_1 \xi_1} + \alpha_2 \xi_1^2 - / + \alpha_4 \xi_1^2 \; / \end{array} \right\} + W_2 \alpha_0$$

$$= \alpha_0 (2W_1 + W_2) + \alpha_2 (2W_1 \tilde{\xi}_1^2) + \alpha_4 (W_1 \alpha_4 2) =$$

Hence:

$$\alpha_0 (2W_1 + W_2) + \alpha_2 (2W_1 \tilde{\xi}_1^2) + \alpha_4 (2 W_1 \tilde{\xi}_1^4)$$

$$= 2\alpha_0 + \frac{2}{3}\alpha_2 + \frac{2}{5}\alpha_4$$

$$\begin{cases} 2W_1 + W_2 = 2 \\ 2W_1 \tilde{\xi}_1^2 = 2/3 \rightarrow \tilde{\xi}_1^2 = 1/3W_1 \quad W_1 \neq 0 \\ 2W_1 \xi_1^4 = 2/5 \end{cases}$$

$$\Rightarrow \cancel{2}\cancel{W_1} \frac{1}{9W_1^{\cancel{2}}} = \cancel{2}/5 \rightarrow \boxed{W_1 = 5/9}$$

$$\cancel{2} \times \frac{5}{9} + \frac{W_2}{2} = \cancel{2} \rightarrow \boxed{W_2 = \frac{8}{9}}$$

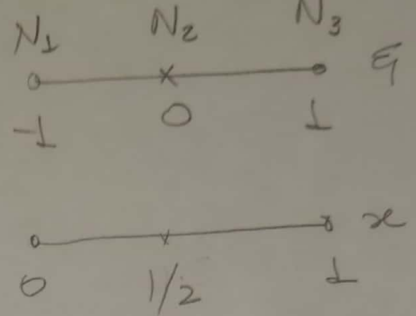
$$\tilde{\xi}_1 = \pm\sqrt{\frac{1}{3} \cdot \frac{9}{5}} = \pm\sqrt{\frac{3}{5}}$$

$$\rightarrow \boxed{\tilde{\xi}_1 = \sqrt{\frac{3}{5}}} \qquad \boxed{\tilde{\xi}_2 = -\xi_1 = -\sqrt{\frac{3}{5}}}$$

Q2 $\tilde{\xi}_i = \begin{Bmatrix} -1/\sqrt{3} \\ 1/\sqrt{3} \end{Bmatrix} \qquad W_i = \begin{Bmatrix} 1 \\ 1 \end{Bmatrix}$

$$N_a(\xi) = \begin{Bmatrix} \xi/2\,(\xi - 1) \\ 1 - \xi^2 \\ \xi/2\,(\xi + 1) \end{Bmatrix} \qquad N_{a,\xi} = \begin{Bmatrix} \xi - 1/2 \\ -2\xi \\ \xi + 1/2 \end{Bmatrix}$$

$$x(\xi) = \sum N_a(\xi)\, x^e_a$$



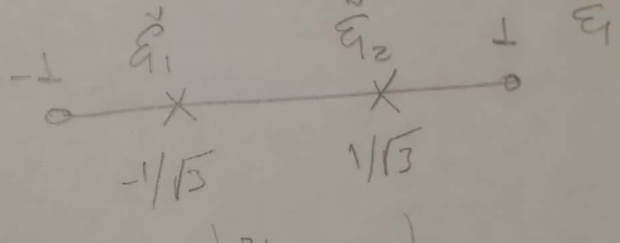
$$= \frac{1}{2} N_2 + N_3$$

$$= \frac{1}{2}(\xi + 1)$$

$$x_{,\xi} = 1/2 \qquad \xi_{,x} = 2$$

$$N_{a,x}^{(\xi)} = N_{a,\xi}\, \xi_{,x} = \begin{Bmatrix} 2\xi - 1 \\ -4\xi \\ 2\xi + 1 \end{Bmatrix}$$

— x —



__Gauss points__

$$N_{a,x}(\tilde{\xi}_1) = \begin{Bmatrix} -\frac{2}{\sqrt{3}} - 1 \\ 4/\sqrt{3} \\ -2/\sqrt{3} + 1 \end{Bmatrix} \qquad N_{a,x}(\tilde{\xi}_2) = \begin{Bmatrix} 2/\sqrt{3} - 1 \\ -4/\sqrt{3} \\ 2/\sqrt{3} + 1 \end{Bmatrix}$$

$$N_a(\tilde{\xi}_1) = \begin{Bmatrix} \frac{1}{6}(1 + \sqrt{3}) \\ 2/3 \\ \frac{1}{6}(1 - \sqrt{3}) \end{Bmatrix} \qquad N_a(\tilde{\xi}_2) = \begin{Bmatrix} \frac{1}{6}(1 - \sqrt{3}) \\ 2/3 \\ \frac{1}{6}(1 + \sqrt{3}) \end{Bmatrix}$$

$$\underline{n}^e(\underline{d}^e) = \begin{Bmatrix} n_1^e(\underline{d}^e) \\ n_2^e(\underline{d}^e) \\ n_3^e(\underline{d}^e \end{Bmatrix}$$

$$n_a^e(\underline{d}^e) = \int N_{a,x}\, k\, u_{,x}\, d\Omega$$

$$u_{,x} = \sum_{1} N_B\, d_B$$

$$n_a^e = \int_{-1}^{1} N_{a,x}\, k(\xi)\, N_B\, d_B\, x_{,\xi}\, d\xi$$

$$= \sum_{l=1}^{2} W_l \left[N_{a,x}\, k\, N_B\, d_B\, x_{,\xi} \right]_{\tilde{\xi}_l}$$

$$n_a^e = \begin{Bmatrix} -2/\sqrt{3} - 1 \\ 4/\sqrt{3} \\ -2\sqrt{3} + 1 \end{Bmatrix} k(-1/\sqrt{3}) \left[\frac{1}{6}(1+\sqrt{3})d_1 + \frac{2}{3}d_2 + \frac{1}{6}(1-\sqrt{3})d_3 \right] \frac{1}{2} +$$

$$+ \begin{Bmatrix} 2/\sqrt{3} - 1 \\ -4\sqrt{3} \\ 2/\sqrt{3} + 1 \end{Bmatrix} k(1/\sqrt{3}) \left[\frac{1}{6}(1-\sqrt{3})d_1 + \frac{2}{3}d_2 + \frac{1}{6}(1+\sqrt{3})d_3 \right] \frac{1}{2}$$

$$\underline{f}^e = \{ f_a^e \}$$

$$f_a^e = \int_\Omega N_a f \, d\Omega + h \, \delta_{a1} \delta_{e1} \quad , \quad f = \sum_{b=1}^{3} N_b f_b^e$$

$$f_a^e = \int_{-1}^{1} N_a \sum_{b=1}^{3} N_b f_b^e \, x_{,\xi} \, d\xi + h \, \delta_{a1} \delta_{e1}$$

$$= \sum_{l=1}^{2} W_l \left[N_a \sum_{b=1}^{3} N_b f_b^e \, x_{,\xi} \right]_{\xi = \tilde{\xi}_l}$$

$$f_a^e = \begin{Bmatrix} \frac{1}{6}(1+\sqrt{3}) \\ 2/3 \\ 1/6\,(1-\sqrt{3}) \end{Bmatrix} \left[\frac{1}{6}(1+\sqrt{3}) f_1^e + \frac{2}{3} f_2^e + \frac{1}{6}(1-\sqrt{3}) f_3^e \right] \frac{1}{2} +$$

$$+ \begin{Bmatrix} \frac{1}{6}(1-\sqrt{3}) \\ 2/3 \\ \frac{1}{6}(1+\sqrt{3}) \end{Bmatrix} \left[\frac{1}{6}(1-\sqrt{3}) f_1^e + \frac{2}{3} f_2^e + \frac{1}{6}(1+\sqrt{3}) f_3^e \right] \frac{1}{2} +$$

$$+ h \, \delta_{a1} \delta_{e1}$$

$$D\underline{n}^e(\underline{d}^e) = \left[\frac{d n_a^e}{d d_b^e}\right] = \int_{\Omega^e} N_{a,x}\, k_{,u}\, N_b \underbrace{\left(\sum N_{c,x}\, d_c^e\right)}_{(\circ)} dx + \int_{\Omega^e} N_{a,x}\, k\, N_{b,x}\, dx$$

$$(\circ)_{\tilde{\xi}_1} = \left(-\frac{2}{\sqrt{3}} - 1\right) d_1^e + \frac{4}{\sqrt{3}} d_2^e + \left(-\frac{2}{\sqrt{3}} + 1\right) d_3^e$$

$$(\circ)_{\tilde{\xi}_2} = \left(2/\sqrt{3} - 1\right) d_1^e - \frac{4}{\sqrt{3}} d_2^e + \left(\frac{2}{\sqrt{3}} + 1\right) d_3^e$$

$$D\underline{n}^e(\underline{d}^e) = \int_{-1}^{1} N_{a,x}\, k_{,u}\, N_b\, (\circ)\, x_{,\xi}\, d\xi + \int_{-1}^{1} N_{a,x}\, k\, N_{b,x}\, x_{,\xi}\, d\xi$$

$$= \sum_{l=1}^{2} W_l \left\{ \; - - - \; \right\}_{\xi = \tilde{\xi}_l}$$

Now subs the values of $N_{a,x}$, N_b, $(\circ)$ etc at $\tilde{\xi}_1 = -1/\sqrt{3}$ and $\tilde{\xi}_2 = 1/\sqrt{3}$ to obtain the full tensor.

I obtained the final expression using python "sympy".

See next page.

In [1]:
```
import sympy

def rat(expr):
    for i in expr.atoms(sympy.Float):
        r = sympy.Rational(str(i)).limit_denominator(1000)
        expr = expr.subs(i, r)
    return expr
```

In [2]:
```
from IPython.display import display, Markdown
import sympy
from sympy import *
import numpy as np
x, xi, u = symbols(r'x \xi u', real=True)
d1, d2, d3 = symbols(r'd_1 d_2 d_3', real=True)

Xi = [ -1/sqrt(3), 1/sqrt(3) ]
W = [ 1, 1 ]

kappa = Function(r"K")(xi)
dkappa_du = Function(r"K_{,u}")(xi)

Na = [
1/2*xi*(xi-1),
1-xi*xi,
1/2*xi*(xi+1)
]

display( Markdown( f"$N_1 = {sympy.latex(Na[0])}$" ))
display( Markdown( f"$N_2 = {sympy.latex(Na[1])}$" ))
display( Markdown( f"$N_3 = {sympy.latex(Na[2])}$" ))
```

$N_1 = 0.5\xi\left(\xi - 1\right)$

$N_2 = 1 - \xi^2$

$N_3 = 0.5\xi\left(\xi + 1\right)$

In [3]:
```
dN1_xi = diff(Na[0], xi)
dN2_xi = diff(Na[1], xi)
dN3_xi = diff(Na[2], xi)

display( Markdown( r"$dN_1/d\xi = "+ f"{sympy.latex(dN1_xi)}$" ))
display( Markdown( r"$dN_2/d\xi = "+ f"{sympy.latex(dN2_xi)}$" ))
display( Markdown( r"$dN_3/d\xi = "+ f"{sympy.latex(dN3_xi)}$" ))
```

$dN_1/d\xi = 1.0\xi - 0.5$

$dN_2/d\xi = -2\xi$

$dN_3/d\xi = 1.0\xi + 0.5$

In [4]:
```
X = Na[1] * 0.5 + Na[2]
X = simplify(X)
dx_dxi = diff(X,xi)
dxi_dx = 1/dx_dxi

display( Markdown( r"$x(\xi) = "+ f"{sympy.latex(X)}$"))
```

```
display( Markdown( r"$dx/d\xi = "+ f"{sympy.latex(dx_dxi)}$" ))
display( Markdown( r"$d\xi/dx = "+ f"{sympy.latex(dxi_dx)}$" ))
```

$x(\xi) = 0.5\xi + 0.5$

$dx/d\xi = 0.5$

$d\xi/dx = 2.0$

In [5]:

```
dN1_x = dN1_xi * dxi_dx
dN2_x = dN2_xi * dxi_dx
dN3_x = dN3_xi * dxi_dx

display( Markdown( r"$dN_1/dx = "+ f"{sympy.latex(dN1_x)}$" ))
display( Markdown( r"$dN_2/dx = "+ f"{sympy.latex(dN2_x)}$" ))
display( Markdown( r"$dN_3/dx = "+ f"{sympy.latex(dN3_x)}$" ))
```

$dN_1/dx = 2.0\xi - 1.0$

$dN_2/dx = -4.0\xi$

$dN_3/dx = 2.0\xi + 1.0$

In [11]:

```
def build_N_x( xi_ ) :
    # Derivatives in X space
    N1_x = dN1_xi.subs(xi, xi_) * dxi_dx
    N2_x = dN2_xi.subs(xi, xi_) * dxi_dx
    N3_x = dN3_xi.subs(xi, xi_) * dxi_dx
    return N1_x, N2_x, N3_x

db = [ d1, d2, d3 ]
ne = [ 0, 0, 0 ]

f = symbols(r'f_1 f_2 f_3', real=True)
h = symbols(r'h', real=True)
fe = [ 0, 0, 0 ]
dna_ddb = zeros( 3, 3)
fcol = []
for xi_, W_ in zip( Xi, W ) :
    Na_x_ = build_N_x(xi_)
    x_ = X.subs(xi, xi_)
    kappa_ = kappa.subs( xi, xi_ )
    dkappa_du_ = dkappa_du.subs( xi, xi_ )
    fcol.append(kappa_)
    fcol.append(dkappa_du_)
    Na_ = [0, 0, 0]
    for a in range(3) : Na_[a] = simplify(Na[a].subs(xi, xi_))

    q = 0
    for b in range(3) : q += db[b] * Na_x_[b]
    for a in range(3) :
        ne[a] += W_ * dx_dxi * Na_x_[a] * q * kappa_
        fe[a] += W_ * dx_dxi * Na_[a] * f[a]
        fe[a] = simplify(fe[a])

    for a in range(3) :
        for b in range(3) :
            dna_ddb[a,b] += W_ * dx_dxi * Na_x_[a] * Na_[b] * q * dkappa_du_
            dna_ddb[a,b] += W_ * dx_dxi * kappa_ * Na_x_[a] * Na_x_[b]
```

```
    fe[0] += W_ * h * Na_[0]
    fe[0] = simplify(fe[0])

for a in range(3) :
    ne[a] = cancel(ne[a])
    for c in fcol : ne[a] = collect(ne[a],c)
    for d in db : ne[a] = collect( ne[a], d )
    ne[a] = simplify(ne[a],full=True)

for a in range(3) :
    fe[a] = nsimplify(fe[a])

for a in range(3) :
    for b in range(3) :
        dna_ddb[a,b] = cancel(dna_ddb[a,b])
        for c in fcol : dna_ddb[a,b] = collect(dna_ddb[a,b],c)
        for d in db : dna_ddb[a,b] = collect( dna_ddb[a,b], d )
        dna_ddb[a,b] = simplify(dna_ddb[a,b])
```

In [12]:
```
for a in range(3) :
    display( Markdown( f"$n_{a+1}^e(d^e) = {sympy.latex(rat(ne[a]))}$" ))
```

$$n_1^e(d^e) = \left(d_1\left(\frac{7}{6}-\frac{2\sqrt{3}}{3}\right)-d_2\left(\frac{4}{3}-\frac{2\sqrt{3}}{3}\right)+\frac{d_3}{6}\right)K\left(\frac{\sqrt{3}}{3}\right)+\left(d_1\left(\frac{2\sqrt{3}}{3}+\frac{7}{6}\right)-d_2\left(\frac{2\sqrt{3}}{3}+\frac{4}{3}\right)+\frac{d_3}{6}\right)K\left(-\frac{\sqrt{3}}{3}\right)$$

$$n_2^e(d^e) = -\left(d_1\left(\frac{4}{3}-\frac{2\sqrt{3}}{3}\right)-\frac{8d_2}{3}+d_3\left(\frac{2\sqrt{3}}{3}+\frac{4}{3}\right)\right)K\left(\frac{\sqrt{3}}{3}\right)-\left(d_1\left(\frac{2\sqrt{3}}{3}+\frac{4}{3}\right)-\frac{8d_2}{3}+d_3\left(\frac{4}{3}-\frac{2\sqrt{3}}{3}\right)\right)K\left(-\frac{\sqrt{3}}{3}\right)$$

$$n_3^e(d^e) = \left(\frac{d_1}{6}-d_2\left(\frac{4}{3}-\frac{2\sqrt{3}}{3}\right)+d_3\left(\frac{7}{6}-\frac{2\sqrt{3}}{3}\right)\right)K\left(-\frac{\sqrt{3}}{3}\right)+\left(\frac{d_1}{6}-d_2\left(\frac{2\sqrt{3}}{3}+\frac{4}{3}\right)+d_3\left(\frac{2\sqrt{3}}{3}+\frac{7}{6}\right)\right)K\left(\frac{\sqrt{3}}{3}\right)$$

In [13]:
```
for a in range(3) :
    display( Markdown( f"$f_{a+1}^e = {sympy.latex(rat(fe[a]))}$" ))
```

$$f_1^e = \frac{f_1}{6}+\frac{h}{3}$$

$$f_2^e = \frac{2f_2}{3}$$

$$f_3^e = \frac{f_3}{6}$$

In [15]:
```
for a in range(3) :
    for b in range(3) :
        display( Markdown( r"$\frac{\partial n_"+f"{a+1}"+r"^e}{\partial d_"+f"{b+1}"+r"^e} $ ="+ f"${sympy.latex(rat(dna_ddb[a,b]))}$" ))
    print("\n\n")
```

$$\frac{\partial n_1^e}{\partial d_1^e} = \left(d_1\left(\frac{19}{36}-\frac{11\sqrt{3}}{36}\right)-d_2\left(\frac{5}{9}-\frac{\sqrt{3}}{3}\right)+\frac{d_3\left(1-\sqrt{3}\right)}{36}\right)K_{,u}\left(\frac{\sqrt{3}}{3}\right)+\left(d_1\left(\frac{19}{36}+\frac{11\sqrt{3}}{36}\right)-d_2\left(\frac{5}{9}+\frac{\sqrt{3}}{3}\right)+\frac{d_3\left(1+\sqrt{3}\right)}{36}\right)K_{,u}\left(-\frac{\sqrt{3}}{3}\right)+\left(\frac{2\sqrt{3}}{3}+\frac{7}{6}\right)K\left(-\frac{\sqrt{3}}{3}\right)+\left(\frac{7}{6}-\frac{2\sqrt{3}}{3}\right)K\left(\frac{\sqrt{3}}{3}\right)$$

$$\frac{\partial n_1^e}{\partial d_2^e} = \left(d_1\left(\frac{7}{9}-\frac{4\sqrt{3}}{9}\right)-d_2\left(\frac{8}{9}-\frac{4\sqrt{3}}{9}\right)+\frac{d_3}{9}\right)K_{,u}\left(\frac{\sqrt{3}}{3}\right)+\left(d_1\left(\frac{4\sqrt{3}}{9}+\frac{7}{9}\right)-d_2\left(\frac{4\sqrt{3}}{9}+\frac{8}{9}\right)+\frac{d_3}{9}\right)K_{,u}\left(-\frac{\sqrt{3}}{3}\right)-\left(\frac{2\sqrt{3}}{3}+\frac{4}{3}\right)K\left(-\frac{\sqrt{3}}{3}\right)-\left(\frac{4}{3}-\frac{2\sqrt{3}}{3}\right)K\left(\frac{\sqrt{3}}{3}\right)$$

$$\frac{\partial n_1^e}{\partial d_3^e} = \left(-d_1\left(\frac{5}{36}-\frac{\sqrt{3}}{12}\right)+d_2\left(\frac{1}{9}-\frac{\sqrt{3}}{9}\right)+\frac{d_3\left(1+\sqrt{3}\right)}{36}\right)K_{,u}\left(\frac{\sqrt{3}}{3}\right)+\left(-d_1\left(\frac{5}{36}+\frac{\sqrt{3}}{12}\right)+d_2\left(\frac{1}{9}+\frac{\sqrt{3}}{9}\right)+\frac{d_3\left(1-\sqrt{3}\right)}{36}\right)K_{,u}\left(-\frac{\sqrt{3}}{3}\right)+\frac{K\left(-\frac{\sqrt{3}}{3}\right)}{6}+\frac{K\left(\frac{\sqrt{3}}{3}\right)}{6}$$

$$\frac{\partial n_2^e}{\partial d_1^e} = \left(-d_1\left(\frac{5}{9}-\frac{\sqrt{3}}{3}\right)+\frac{4d_2\left(1-\sqrt{3}\right)}{9}+d_3\left(\frac{1}{9}+\frac{\sqrt{3}}{9}\right)\right)K_{,u}\left(\frac{\sqrt{3}}{3}\right)+\left(-d_1\left(\frac{5}{9}+\frac{\sqrt{3}}{3}\right)+\frac{4d_2\left(1+\sqrt{3}\right)}{9}+d_3\left(\frac{1}{9}-\frac{\sqrt{3}}{9}\right)\right)K_{,u}\left(-\frac{\sqrt{3}}{3}\right)-\left(\frac{2\sqrt{3}}{3}+\frac{4}{3}\right)K\left(-\frac{\sqrt{3}}{3}\right)-\left(\frac{4}{3}-\frac{2\sqrt{3}}{3}\right)K\left(\frac{\sqrt{3}}{3}\right)$$

$$\frac{\partial n_2^e}{\partial d_2^e} = - \left(d_1 \left(\frac{8}{9} - \frac{4\sqrt{3}}{9}\right) - \frac{16 d_2}{9} + d_3 \left(\frac{4\sqrt{3}}{9} + \frac{8}{9}\right)\right) K_{,u}\left(\frac{\sqrt{3}}{3}\right) - \left(d_1 \left(\frac{4\sqrt{3}}{9} + \frac{8}{9}\right) - \frac{16 d_2}{9} + d_3 \left(\frac{8}{9} - \frac{4\sqrt{3}}{9}\right)\right) K_{,u}\left(-\frac{\sqrt{3}}{3}\right) + \frac{8 K\left(-\frac{\sqrt{3}}{3}\right)}{3} + \frac{8 K\left(\frac{\sqrt{3}}{3}\right)}{3}$$

$$\frac{\partial n_2^e}{\partial d_3^e} = \left(d_1 \left(\frac{1}{9} - \frac{\sqrt{3}}{9}\right) + \frac{4 d_2 \left(1 + \sqrt{3}\right)}{9} - d_3 \left(\frac{5}{9} + \frac{\sqrt{3}}{3}\right)\right) K_{,u}\left(\frac{\sqrt{3}}{3}\right) + \left(d_1 \left(\frac{1}{9} + \frac{\sqrt{3}}{9}\right) + \frac{4 d_2 \left(1 - \sqrt{3}\right)}{9} - d_3 \left(\frac{5}{9} - \frac{\sqrt{3}}{3}\right)\right) K_{,u}\left(-\frac{\sqrt{3}}{3}\right) - \left(\frac{4}{3} - \frac{2\sqrt{3}}{3}\right) K\left(-\frac{\sqrt{3}}{3}\right) - \left(\frac{2\sqrt{3}}{3} + \frac{4}{3}\right) K\left(\frac{\sqrt{3}}{3}\right)$$

$$\frac{\partial n_3^e}{\partial d_1^e} = \left(\frac{d_1 \left(1 - \sqrt{3}\right)}{36} + d_2 \left(\frac{1}{9} + \frac{\sqrt{3}}{9}\right) - d_3 \left(\frac{5}{36} + \frac{\sqrt{3}}{12}\right)\right) K_{,u}\left(\frac{\sqrt{3}}{3}\right) + \left(\frac{d_1 \left(1 + \sqrt{3}\right)}{36} + d_2 \left(\frac{1}{9} - \frac{\sqrt{3}}{9}\right) - d_3 \left(\frac{5}{36} - \frac{\sqrt{3}}{12}\right)\right) K_{,u}\left(-\frac{\sqrt{3}}{3}\right) + \frac{K\left(-\frac{\sqrt{3}}{3}\right)}{6} + \frac{K\left(\frac{\sqrt{3}}{3}\right)}{6}$$

$$\frac{\partial n_3^e}{\partial d_2^e} = \left(\frac{d_1}{9} - d_2 \left(\frac{8}{9} - \frac{4\sqrt{3}}{9}\right) + d_3 \left(\frac{7}{9} - \frac{4\sqrt{3}}{9}\right)\right) K_{,u}\left(-\frac{\sqrt{3}}{3}\right) + \left(\frac{d_1}{9} - d_2 \left(\frac{4\sqrt{3}}{9} + \frac{8}{9}\right) + d_3 \left(\frac{4\sqrt{3}}{9} + \frac{7}{9}\right)\right) K_{,u}\left(\frac{\sqrt{3}}{3}\right) - \left(\frac{4}{3} - \frac{2\sqrt{3}}{3}\right) K\left(-\frac{\sqrt{3}}{3}\right) - \left(\frac{2\sqrt{3}}{3} + \frac{4}{3}\right) K\left(\frac{\sqrt{3}}{3}\right)$$

$$\frac{\partial n_3^e}{\partial d_3^e} = \left(\frac{d_1 \left(1 - \sqrt{3}\right)}{36} - d_2 \left(\frac{5}{9} - \frac{\sqrt{3}}{3}\right) + d_3 \left(\frac{19}{36} - \frac{11\sqrt{3}}{36}\right)\right) K_{,u}\left(-\frac{\sqrt{3}}{3}\right) + \left(\frac{d_1 \left(1 + \sqrt{3}\right)}{36} - d_2 \left(\frac{5}{9} + \frac{\sqrt{3}}{3}\right) + d_3 \left(\frac{19}{36} + \frac{11\sqrt{3}}{36}\right)\right) K_{,u}\left(\frac{\sqrt{3}}{3}\right) + \left(\frac{7}{6} - \frac{2\sqrt{3}}{3}\right) K\left(-\frac{\sqrt{3}}{3}\right) + \left(\frac{2\sqrt{3}}{3} + \frac{7}{6}\right) K\left(\frac{\sqrt{3}}{3}\right)$$

```
In [ ]:
```

QUESTION 3

In [1]:
```python
import numpy as np
from scipy.optimize import fsolve
from IPython.display import display, Markdown
from sympy import *
from sympy import latex as ltx

# Derivatives
x = symbols(r'x', real=True)
N1, N2 = symbols(r'N_1 N_2', real=True)
d1, d2, d3 = symbols(r'd_1 d_2 d_3', real=True)
N1 = x * d1 / ( 10 - d1) - 0.5 * d2**2
N2 = d2 - d1

display ( Markdown( f"$\\frac{{\\partial N_1}}{{\\partial d_1}} = {ltx(diff(N1,d1))}$" ) )
display ( Markdown( f"$\\frac{{\\partial N_1}}{{\\partial d_2}} = {ltx(diff(N1,d2))}$" ) )
display ( Markdown( f"$\\frac{{\\partial N_2}}{{\\partial d_1}} = {ltx(diff(N2,d1))}$" ) )
display ( Markdown( f"$\\frac{{\\partial N_2}}{{\\partial d_2}} = {ltx(diff(N2,d2))}$" ) )
```

$\frac{\partial N_1}{\partial d_1} = \frac{d_1 x}{(10-d_1)^2} + \frac{x}{10-d_1}$

$\frac{\partial N_1}{\partial d_2} = -1.0 d_2$

$\frac{\partial N_2}{\partial d_1} = -1$

$\frac{\partial N_2}{\partial d_2} = 1$

In [2]:
```python
import numpy as np
from scipy.optimize import fsolve


# HELPER
def cN1(x,d_) :
    if not len(d_) : return []
    return x * d_[0] / (10. - d_[0]) - 0.5 * d_[1]**2
def cN2(x,d_) :
    if not len(d_) : return []
    return d_[1] - d_[0]
def cN(x,d_) :
    if not len(d_) : return []
    return [ cN1(x,d_), cN2(x,d_) ]
def cG( dd_, R_) : return dd_ @ R_

#
# MAIN ENGINE
#
def run_sim( x, line_search=False, modif_nr=False,  line_search_maxit=1000, line_search_fast_s=False, BFGS=False ) :
    global G0,d, delta_d, K_inv, v_, alpha_, w_, delta_R_, R_, R
    # Load steps
    dF_ = np.array( [ 0.25, 0 ] )
    F_ = np.zeros(2)

    s=1 # line search
    RET_F, RET_D, RET_I = [], [], []

    # Load steps
    d_ = np.array( [ 0. , 0. ] )
    N_ = cN(x,d_)

    for n in range(40) :
        F_ += dF_

        #print(f"Running LOAD STEP {n} (F={F_})")
        CONVERGED = False

        # INITIALIZE THE RESIDUAL
        N_ = cN(x,d_)
        R_ = F_ - N_
        r0 = np.linalg.norm(R_)
        r = r0
        # NEWTON ITERATIONS
        for i in range(15) :
            #print(f"   Netwon step {i} (r:{r:.4f})")
            # UPDATE THE TANGENT MATRIX
            if not modif_nr or not i :
                K = np.array( [ [ d_[0] * x / (10 - d_[0])**2 + x / (10 - d_[0]),           -d_[1] ] ,
                                [-1,                                                            1] ] )

            # UPDATE BFGS PREDICTOR AFTER FIRST TIMESTEP
            if i and BFGS :
                Rbfgs_ = R_ + ( v_ @ R_ ) * w_
                dd0_ = np.linalg.inv( K ) @ Rbfgs_
                dd_ = dd0_ + ( w_ @ v_ ) * dd0_
            else :
                # UPDATE THE SEARCH DIRECTION
```

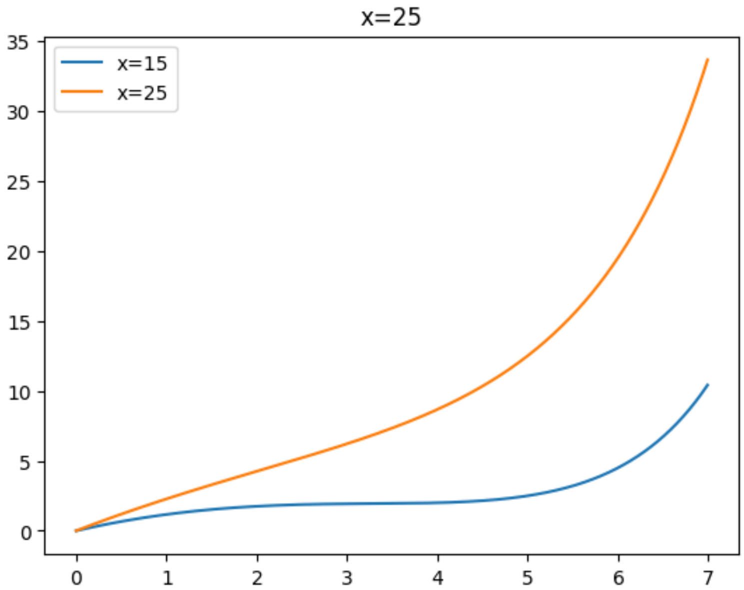
```python
                dd_ = np.linalg.inv(K) @ R_

            # UPDATE LINE SEARCH G0 (before d_ update)
            g0 = cG( dd_, R_ )

            # SAVE & UPDATE d_
            prev_d_ = d_.copy()
            d_ += dd_
            # TRIAL RESIDUAL
            tR_ = F_ - cN(x,d_)

            # UPDATE LINE SEARCH G1 (after d_ update)
            g1 = cG( d_, tR_)

            # UPDATE LINE SEARCH
            if line_search :
                #print("        Line search started.")

                def foo(_s):
                    gs = cG( dd_ ,      F_ - cN(x, prev_d_ + _s*dd_) )
                    [N1,N2] = cN(x, prev_d_ + _s*dd_)
                    if line_search_fast_s :
                        if abs(gs) < abs(g0/2) :
                            #print("        G(s) < G(0)/2. Enough.")
                            return 0
                    return gs

                s = fsolve(foo,0, maxfev=line_search_maxit)[0]

                test_f = abs(foo(s))
                if abs(test_f)>abs(g0/2)+1e-4 :
                    print(f"[FAILED] Cannot find acceptable s in line search - test_f[{test_f:.4e} > abs(g0/2)[{abs(g0/2):.4e}]]")
                    return RET_D, RET_F, RET_I

            # UPDATE THE DOFs BASED ON THE LINE SEARCH
            d_ = prev_d_ + dd_ * s

            # SAVE & UPDATE R_
            prev_R_ = R_.copy()
            R_ = F_ - cN(x,d_)

            # UPDATE GS
            gs = cG( dd_, R_)

            r = np.linalg.norm(R_)
            if r > 1e50 : break  # Crash!
            if r/r0 < 1e-4 :
                #print(f"        Converged (r/r0={r/r0:.4e})")
                CONVERGED = True
                break

            # UPDATE BFGS VECTORS
            if BFGS :
                v_ = dd_ / (gs - g0)
                alpha = np.sqrt( np.max( [ 0, - s * (gs - g0)/ g0 ] ) )
                w_ = (alpha - 1) * R_ - prev_R_
                #print("        Update BFGS vectors.")

        if not CONVERGED :
            print(f"[FAILED] The solver did not converge after {i} iterations. Giving up.")
            return RET_D, RET_F, RET_I

        RET_D.append(d_.copy())
        RET_I.append(i+1)
        RET_F.append(F_[0])

    return RET_D, RET_F, RET_I


#full_run( line_search=0, modif_nr=1, line_search_maxit=10000, line_search_fast_s=False, BFGS=1 )
```

1) Exact $N_1(d_1, d_2 = d_1)$ vs. d_1 for x=15 and x=25

In [3]:
```python
import matplotlib.pyplot as plt
import numpy as np

d0_ = np.arange(0,7,.01)
d0 = np.array( [ i for i in d0_ if i != 10] )

d0 = [ d0, d0 ]
N15exact = cN1(15,d0)
plt.plot( d0[0], N15exact, label="x=15" )
plt.title("x=15")
N25exact = cN1(25,d0)
plt.plot( d0[0], N25exact, label="x=25" )
```

```
plt.title("x=25")
plt.legend()
```

Out[3]: <matplotlib.legend.Legend at 0x1f62d45ee10>

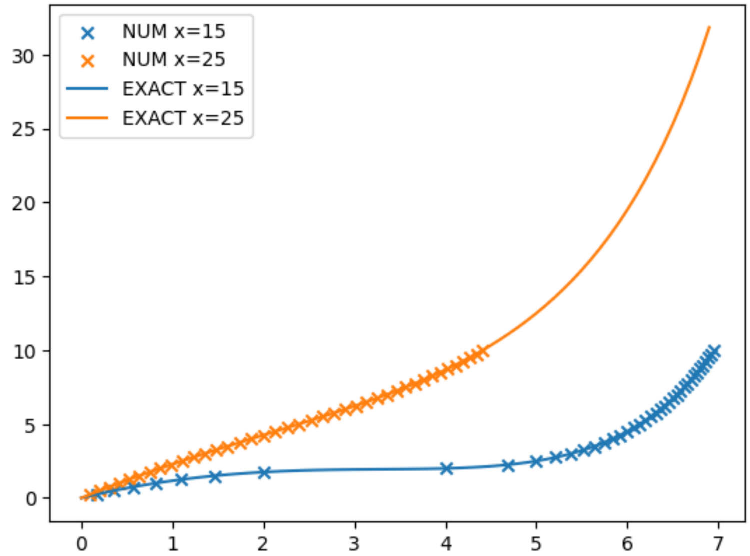


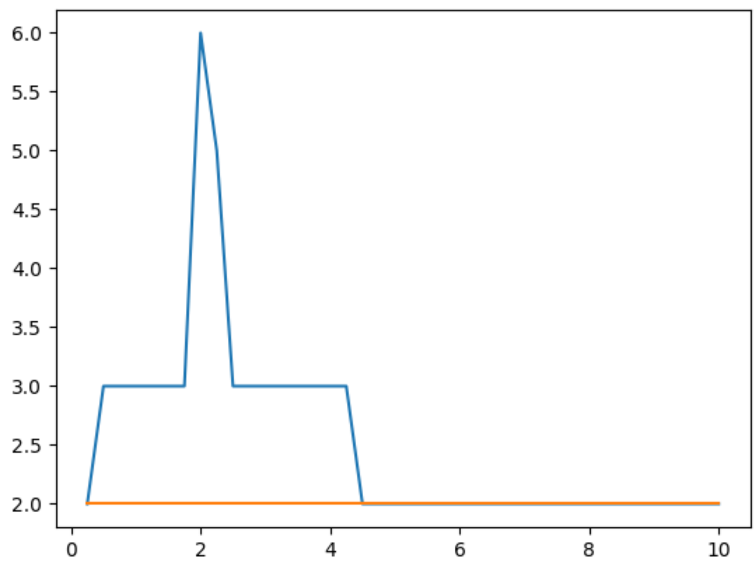
In [4]:
```
import matplotlib.pyplot as plt
def full_run(line_search, modif_nr=False,  line_search_maxit=1000, line_search_fast_s=False, BFGS=False) :
    x=15
    D15,F15,I15 = run_sim( x=x, line_search=line_search, modif_nr=modif_nr, line_search_maxit=line_search_maxit, line_search_fast_s=line_searc
    d15 = np.array( [ [i,i] for i,j in D15 ] ).T
    N15 = cN1(x,d15)

    x=25
    D25,F25,I25 = run_sim( x=x, line_search=line_search, modif_nr=modif_nr, line_search_maxit=line_search_maxit, line_search_fast_s=line_searc
    d25 = np.array( [ [i,i] for i,j in D25 ] ).T
    N25 = cN1(x,d25)

    # PLOT NUMERICAL
    if len(d15) : plt.scatter( d15[0], N15, label="NUM x=15", marker='x' )
    if len(d25) : plt.scatter( d25[0], N25, label="NUM x=25", marker='x' )
    # PLOT EXACT
    d0_ = np.arange(0,7,.1)
    d0 = np.array( [ i for i in d0_ if i != 10] )
    d0 = [ d0, d0 ]
    N15exact = cN1(15,d0)
    plt.plot( d0[0], N15exact, label="EXACT x=15" )
    N25exact = cN1(25,d0)
    plt.plot( d0[0], N25exact, label="EXACT x=25" )
    plt.legend()

    plt.figure()
    plt.plot(F15,I15,label="Iterations x=15")
    plt.plot(F25,I25,label="Iterations x=25")
```

Newton rhapson - consistent tangent

The solver converged in both cases (x=15 and x=25)

In [5]:
```
full_run( line_search=0, modif_nr=False, line_search_maxit=10000, line_search_fast_s=False )
```

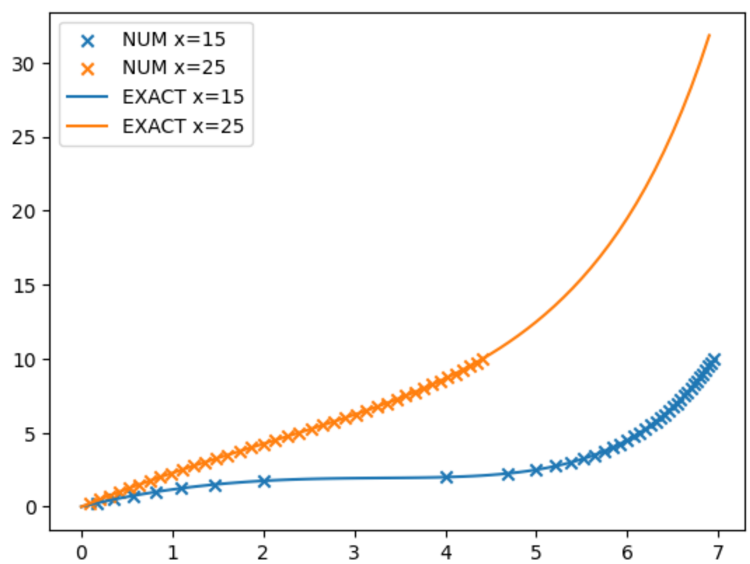


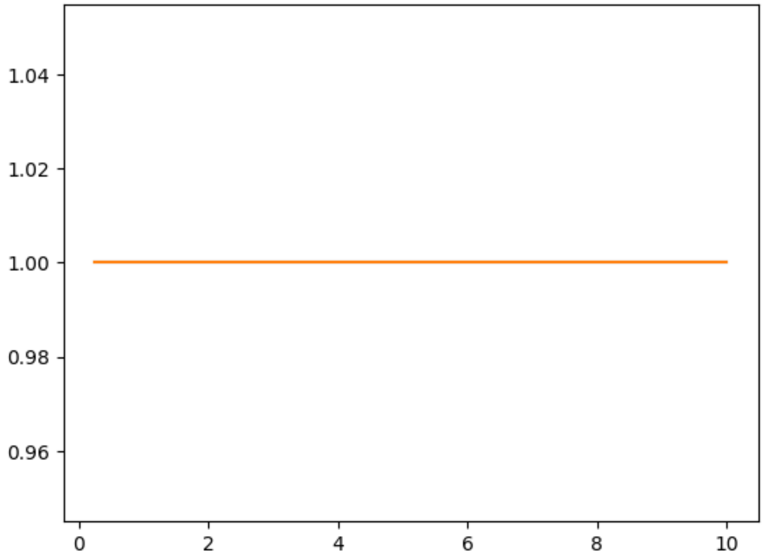


Newton rhapson - consistent tangent + Line Search

The solver converged in both cases (x=15 and x=25), in one iteration

In [6]:
```
full_run( line_search=True, modif_nr=False, line_search_maxit=10000, line_search_fast_s=False )
```



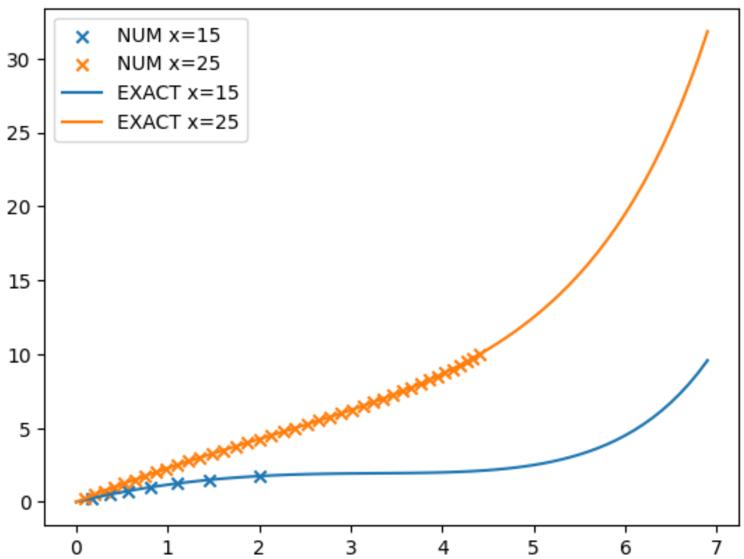


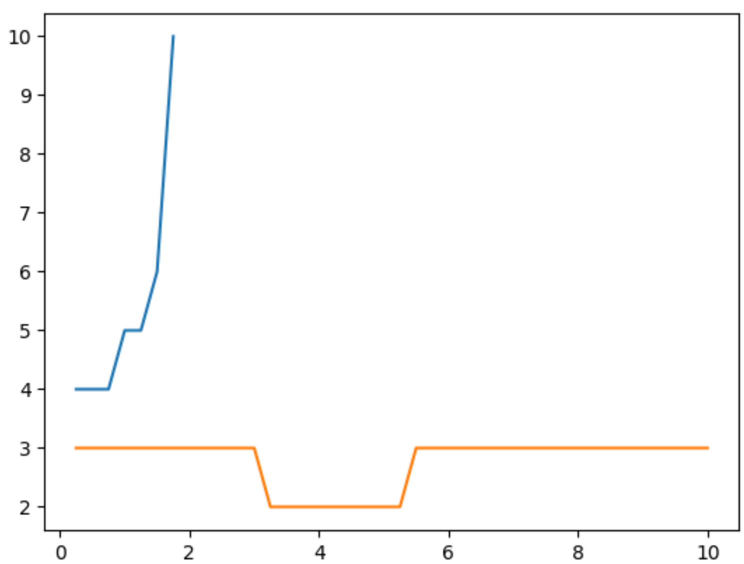
Modified Newton Raphson

The solver converged for x=25 but diverged after some loads for x=25.

```
In [7]: full_run( line_search=False, modif_nr=True, line_search_maxit=10000, line_search_fast_s=False )
```

```
[FAILED] The solver did not converge after 14 iterations. Giving up.
```

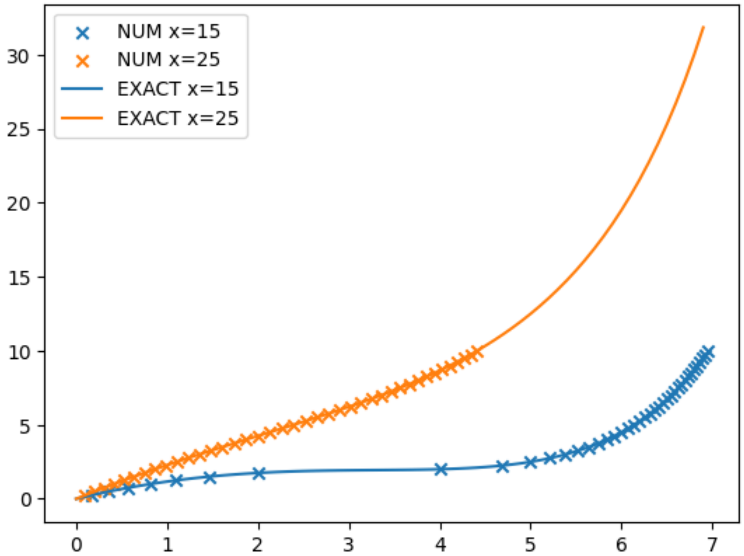


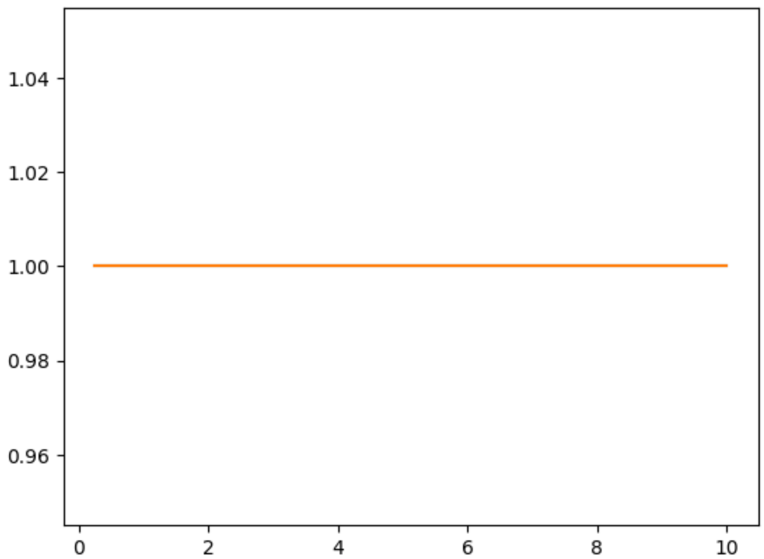


Modified Newton Raphson + Line Search, full resolution

The solver converged in both cases (x=15 and x=25) in one iteration.

In [8]:
```
full_run( line_search=True, modif_nr=True, line_search_maxit=1000, line_search_fast_s=False )
```

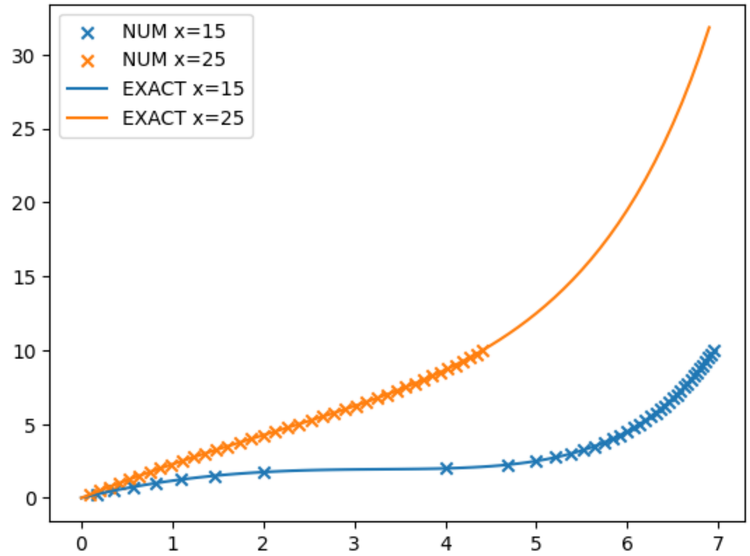


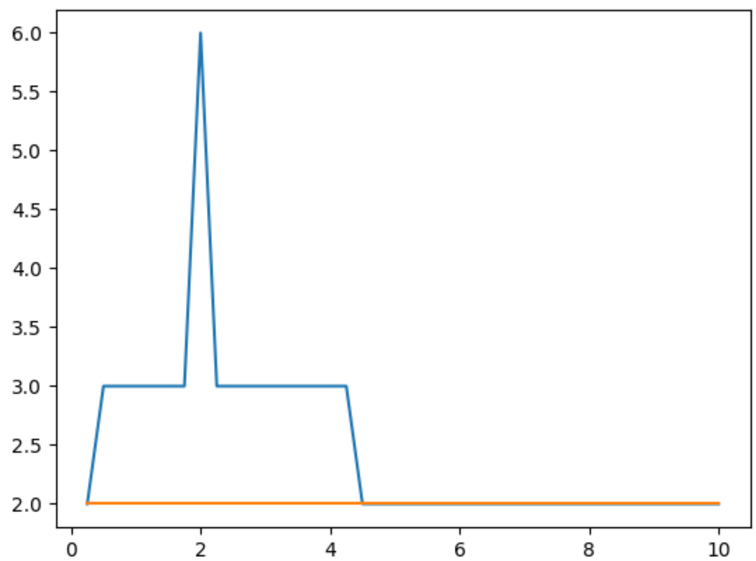


Modified Newton Raphson + Line Search, stop when G(s) < 1/2 G(0)

The solver converged in both cases (x=15 and x=25) in more than one iteration.

In [9]:
```
full_run( line_search=True, modif_nr=True, line_search_maxit=1000, line_search_fast_s=True )
```



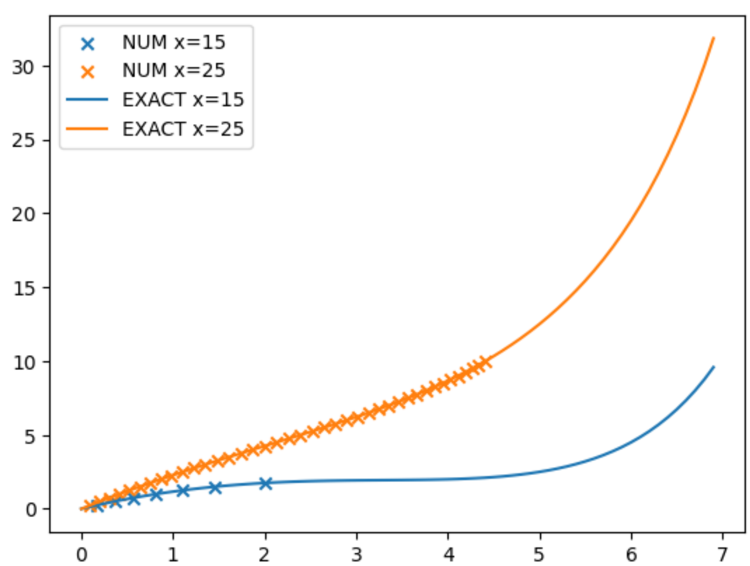


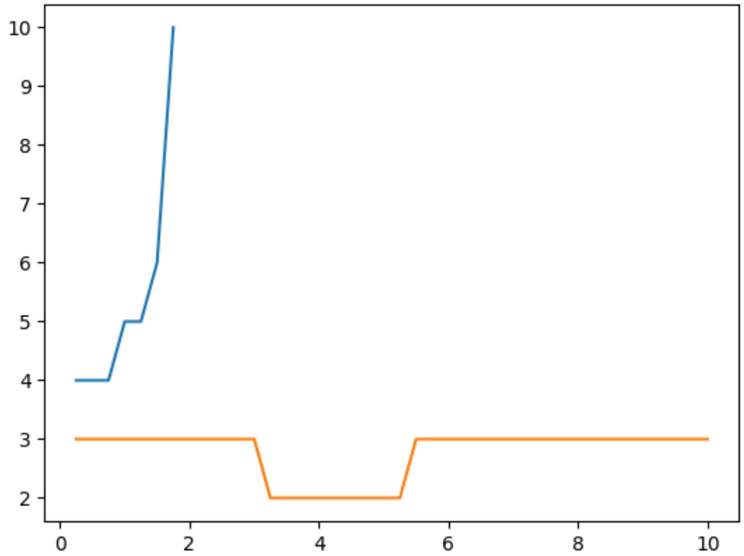
Modified Newton Raphson + Line Search, maximum of 5 iterations

The solver did not converge for x=15.

In [10]:
```
full_run( line_search=False, modif_nr=True, line_search_maxit=5, line_search_fast_s=False, BFGS=False )
```

```
[FAILED] The solver did not converge after 14 iterations. Giving up.
```



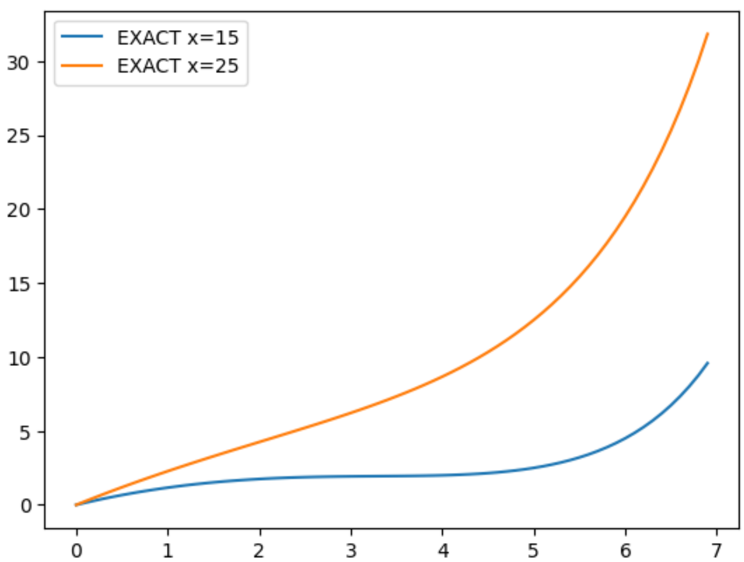


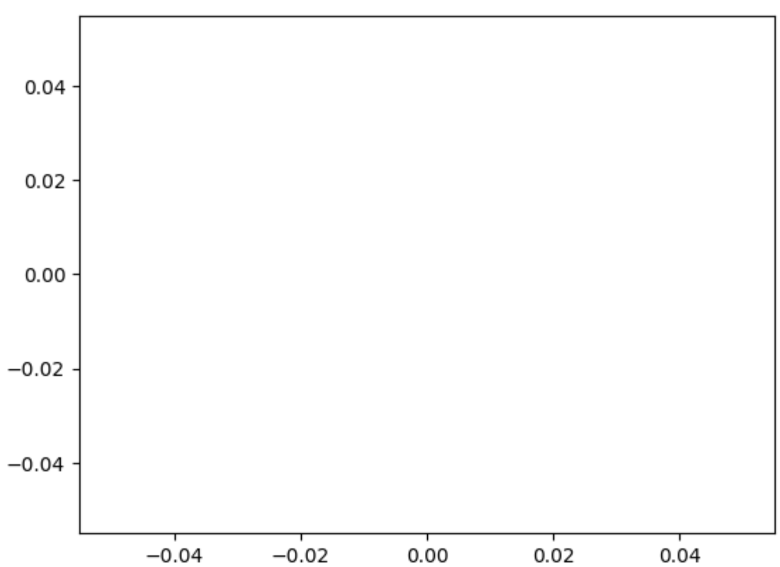
Modified Newton Raphson + BFGS

The solver did not converge.

In [11]:
```
full_run( line_search=False, modif_nr=True, line_search_maxit=10000, line_search_fast_s=False, BFGS=True )
```

```
[FAILED] The solver did not converge after 14 iterations. Giving up.
[FAILED] The solver did not converge after 13 iterations. Giving up.
```

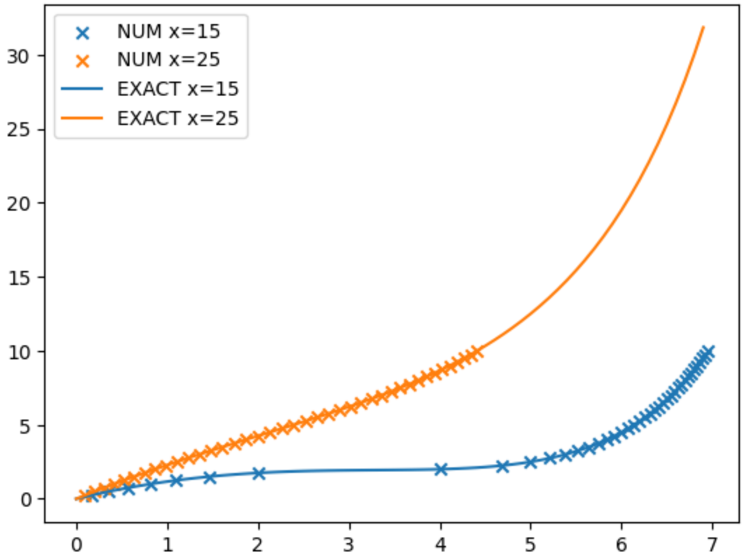


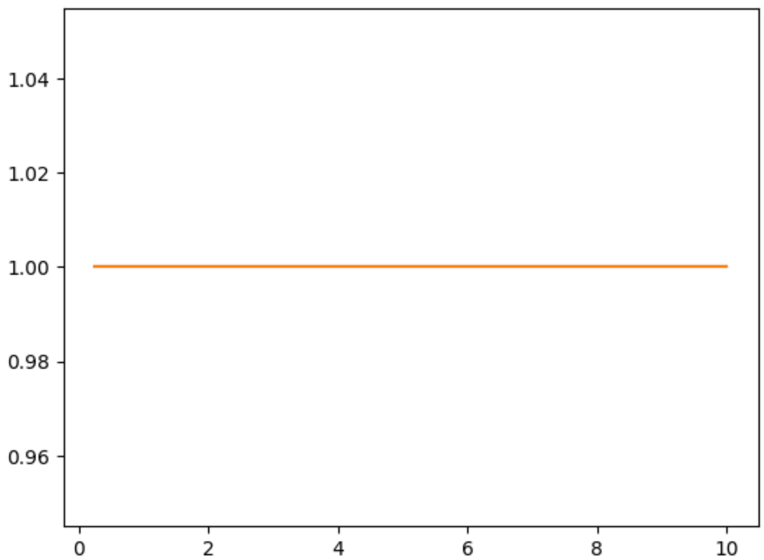


Modified Newton Raphson + BFGS + line search

The solver converged in one iteration (before using the BFGS vectors).

In [12]:
```
full_run( line_search=True, modif_nr=True, line_search_maxit=10000, line_search_fast_s=False, BFGS=True )
```





Comments

1. The consistent tangent strategy works in both cases.
2. The modified Newton fails when the tangent changes significantly.
3. The monified Newton works in both cases when associated with line search.
4. The line search works in both cases, and saves time (less Newton cycles).
5. Even limiting the resolution of the search parameter (max 5 iterations in the s solver), the line search is effective
6. BFGS worked only with line search enabled. This suggest this problem is not suitable to assess BFGS capabilities.

In [ ]: