Name: Altaf Ahmad

Roll : 18MA20005

Solving Non Linear differential equations using the block tridiagonal System of Equations with the help of Newton - Raphson Technique

1. Solve the diffferential equaiton
   $f''' + ff'' + 1 - (f')^2 = 0,\ f(0) = 0, f'(0) = 0, f'(10) = 1$

Let's assume, $F = f' \Rightarrow$ the equation becomes :

$$F'' + fF' + 1 - F^2 = 0,\ f(0) = 0, F(0) = 0, F(10) = 1$$

Now, discretizing these equations, we get

$$g_a = f_i - f_{i-1} - \frac{h \times (F_i + F_{i-1})}{2} \ \rightarrow (a)$$

$$g_b = \frac{F_{i+1} - 2 \times F_i + F_{i-1}}{h^2} + f_i \frac{F_{i+1} - F_{i-1}}{2h} + 1 - F_i^2 \ \rightarrow (b)$$

Now, we have to solve $a \ \& \ b$ set of equations.
At any iteration $(k+1)$, we have,

$$f_i^{(k+1)} = f_i^{(k)} + \Delta f_i \ \& \ F_i^{(k+1)} = F_i^{(k)} + \Delta F_i$$

So, for any iteration $(k+1)$, we have,

$$g_a^{(k+1)} = g_a^{(k)} + \frac{\partial g_a}{\partial f_{i-1}}\Big|^{(k)} \Delta f_{i-1} + \frac{\partial g_a}{\partial f_i}\Big|^{(k)} \Delta f_i + \frac{\partial g_a}{\partial f_{i+1}}\Big|^{(k)} \Delta f_{i+1} + \frac{\partial g_a}{\partial F_{i-1}}\Big|^{(k)} \Delta F_{i-1} + \frac{\partial g_a}{\partial F_i}\Big|^{(k)} \Delta F_i + \frac{\partial g_a}{\partial F_{i+1}}\Big|^{(k)} \Delta F_{i+1} = 0$$

And ,

$$g_b^{(k+1)} = g_b^{(k)} + \frac{\partial g_b}{\partial f_{i-1}}\Big|^{(k)} \Delta f_{i-1} + \frac{\partial g_b}{\partial f_i}\Big|^{(k)} \Delta f_i + \frac{\partial g_b}{\partial f_{i+1}}\Big|^{(k)} \Delta f_{i+1} + \frac{\partial g_b}{\partial F_{i-1}}\Big|^{(k)} \Delta F_{i-1} + \frac{\partial g_b}{\partial F_i}\Big|^{(k)} \Delta F_i + \frac{\partial g_b}{\partial F_{i+1}}\Big|^{(k)} \Delta F_{i+1} = 0$$

Now, consider, $X_i = \begin{bmatrix} \Delta f_i \\ \Delta F_i \end{bmatrix}, A_i = \begin{bmatrix} \left.\frac{\partial g_a}{\partial f_{i-1}}\right|^{(k)} & \left.\frac{\partial g_a}{\partial F_{i-1}}\right|^{(k)} \\ \left.\frac{\partial g_b}{\partial f_{i-1}}\right|^{(k)} & \left.\frac{\partial g_b}{\partial F_{i-1}}\right|^{(k)} \end{bmatrix}, B_i = \begin{bmatrix} \left.\frac{\partial g_a}{\partial f_i}\right|^{(k)} & \left.\frac{\partial g_a}{\partial F_i}\right|^{(k)} \\ \left.\frac{\partial g_b}{\partial f_i}\right|^{(k)} & \left.\frac{\partial g_b}{\partial F_i}\right|^{(k)} \end{bmatrix},$

$$C_i = \begin{bmatrix} \left.\frac{\partial g_a}{\partial f_{i+1}}\right|^{(k)} & \left.\frac{\partial g_a}{\partial F_{i+1}}\right|^{(k)} \\ \left.\frac{\partial g_b}{\partial f_{i+1}}\right|^{(k)} & \left.\frac{\partial g_b}{\partial F_{i+1}}\right|^{(k)} \end{bmatrix}, D_i = \begin{bmatrix} -g_a^{(k)} \\ -g_b^{(k)} \end{bmatrix}$$

So, the above equations can be written as

$$A_i X_{i-1} + B_i X_i + C_i X_{i+1} = D_i$$

Here, we have obtained a Block Tri Diagonal system which can be solved using the modification of the thomas algorithm

In [31]:
```
import numpy as np
from matplotlib import pyplot as plt
```

First, we have to define $g_a,\ g_b$ and it's partial derivatives with $f_{i-1}, f_i, f_{i+1}, F_{i-1}, F_i \ \&\ F_{i+1}$

In [2]:
```
def g_a(f_i_minus, f_i , f_i_plus,F_i_minus, F_i , F_i_plus, h, x_i):
    k = f_i - f_i_minus - h* ((F_i + F_i_minus)/2.0)
    return k
def g_a_dash_f_minus(f_i_minus, f_i , f_i_plus,F_i_minus, F_i , F_i_plus, h, x_i):
    k = -1.0
    return k
def g_a_dash_f_i(f_i_minus, f_i , f_i_plus,F_i_minus, F_i , F_i_plus, h, x_i):
    k = 1.0
    return k
def g_a_dash_f_plus(f_i_minus, f_i , f_i_plus,F_i_minus, F_i , F_i_plus, h, x_i):
    k = 0.0
    return k
def g_a_dash_F_minus(f_i_minus, f_i , f_i_plus,F_i_minus, F_i , F_i_plus, h, x_i):
    k = -h/2
    return k
def g_a_dash_F_i(f_i_minus, f_i , f_i_plus,F_i_minus, F_i , F_i_plus, h, x_i):
    k = -h/2
    return k
def g_a_dash_F_plus(f_i_minus, f_i , f_i_plus,F_i_minus, F_i , F_i_plus, h, x_i):
    k = 0.0
    return k

def g_b(f_i_minus, f_i , f_i_plus,F_i_minus, F_i , F_i_plus, h, x_i):
    k = ((F_i_plus - 2 * F_i + F_i_minus)/ (h*h)) + f_i *((F_i_plus - F_i_minus)/ (2*h)) + 1 - F_i
*F_i
    return k
def g_b_dash_f_minus(f_i_minus, f_i , f_i_plus,F_i_minus, F_i , F_i_plus, h, x_i):
    k = 0.0
    return k
def g_b_dash_f_i(f_i_minus, f_i , f_i_plus,F_i_minus, F_i , F_i_plus, h, x_i):
    k = (F_i_plus - F_i_minus)/ (2*h)
    return k
def g_b_dash_f_plus(f_i_minus, f_i , f_i_plus,F_i_minus, F_i , F_i_plus, h, x_i):
    k = 0.0
    return k
def g_b_dash_F_minus(f_i_minus, f_i , f_i_plus,F_i_minus, F_i , F_i_plus, h, x_i):
    k = (1/(h*h)) - (f_i/(2*h))
    return k
def g_b_dash_F_i(f_i_minus, f_i , f_i_plus,F_i_minus, F_i , F_i_plus, h, x_i):
    k = (-2/(h*h)) - 2 * F_i
    return k
def g_b_dash_F_plus(f_i_minus, f_i , f_i_plus,F_i_minus, F_i , F_i_plus, h, x_i):
```

```
    k = (1/(h*h)) + (f_i/(2*h))
    return k
```

Now, we have to check for the value of $h \;\&\; x_0, x_n, n$

In [45]:
```
h = 1
x_0 = 0
x_n = 10
n = int((x_n- x_0)/h  + 0.5 )
x = np.array([])
for i in range(n+1):
    x = np.append(x, i*h)
```

Now, we have to place the initial guess for $f_i \;\&\; F_i \; i = 1, 2, 3..., n-1$

In [37]:
```
f_0 = 0
F_0 = 0
F_n = 1.0
del_F = (F_n - F_0)/n
f_elements = np.array([])
F_elements = np.array([])
k = F_0
for i in range(n+1):
    f_elements = np.append(f_elements, 0)
    F_elements = np.append(F_elements, k)
    k = k + del_F
```

In [7]:
```
error = 0.00001
diff = 1
```

Now, we define our matrices $A_i, B_i, C_i \;\&\; D_i$

In [12]:
```
A = np.array([ [[g_a_dash_f_minus(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1]),g_a_dash_F_minus(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1])],[g_b_dash_f_minus(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1]), g_b_dash_F_minus(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1])]] ])
B = np.array([ [[g_a_dash_f_i(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1]),g_a_dash_F_i(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1])],[g_b_dash_f_i(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1]), g_b_dash_F_i(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1])]] ])
C = np.array([ [[g_a_dash_f_plus(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1]),g_a_dash_F_plus(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1])],[g_b_dash_f_plus(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1]), g_b_dash_F_plus(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1])]] ])
D = np.array([ [[-g_a(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1])],[-g_b(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1])]] ])
print("A_1 =" )
print(A)
print("B_1 =" )
print(B)
print("C_1 =" )
print(C)
print("D_1 =" )
print(D)
```

```
A_1 =
[[[-1.  -0.5]
  [ 0.   1. ]]]
B_1 =
[[[ 1.  -0.5]
  [ 0.1 -2.2]]]
C_1 =
[[[0. 0.]
  [0. 1.]]]
D_1 =
[[[ 0.05]
  [-0.99]]]
```

In [13]:
```
for i in range(1,n):
    A = np.append(A,[ [[g_a_dash_f_minus(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i]),g_a_dash_F_minus(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i])],[g_b_dash_f_minus(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i]), g_b_dash_F_minus(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i])]] ], axis=0)
    B = np.append(B,[ [[g_a_dash_f_i(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i]),g_a_dash_F_i(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i])],[g_b_dash_f_i(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i]), g_b_dash_F_i(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i])]] ], axis=0)
    C = np.append(C,[ [[g_a_dash_f_plus(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i]),g_a_dash_F_plus(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i])],[g_b_dash_f_plus(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i]), g_b_dash_F_plus(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i])]] ], axis=0)
    D = np.append(D,[ [[-g_a(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i])],[-g_b(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i])]] ], axis=0)
```

Now, we will define the X_i matrices that will finally store the answers.

In [18]:
```
X_ans = np.array([[[0],[0]]],dtype = float)
for i in range(n):
    X_ans = np.append(X_ans, [[[0],[0]]] ,axis = 0)
```

Here, we need to add the initial conditions, i.ie

$$X_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \;\&\; X_n = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Now, we enter the loop

Now, we need to modify the value of D_1 because we have $A_1X_0 + B_1X_1 + C_1X_2 = D_1$ and we already know the value of X_0, so it can be shifted to the RHS
$\Rightarrow B_1X_1 + C_1X_2 = D_1 - A_1X_0 \;\; \Rightarrow D_1 = D_1 - A_1X_0$

In [20]:
```
D[1] = D[1] - np.dot(A[1], X_ans[0])
```

Now, we will transform the given equations into the form :

$$X_i + C_i'X_{i+1} = D_i' \; i = 1, 2, .., n-1$$

where, $C_1' = (B_1)^{-1}C_1$ & $D_1' = (B_1)^{-1}D_1$
and $B_i' = (B_i - A_iC_{i-1}')$
$C_i' = (B_i')^{-1}C_i$
$D_i' = (B_i')^{-1}(D_i - A_iD_{i-1}')$
for $i = 2, 3, \ldots, n-1$

In [21]:
```
cc_dash = np.dot(np.linalg.inv(B[1]),C[1] )
dd_dash = np.dot(np.linalg.inv(B[1]),D[1] )
C_dash = np.array([cc_dash])
D_dash = np.array([dd_dash])
C_dash = np.append(C_dash, [cc_dash], axis = 0)
D_dash = np.append(D_dash, [dd_dash], axis = 0)
for i in range(2,n):
    b_dash = (B[i] - np.dot(A[i],C_dash[i-1]) )
    b_dashinv = np.linalg.inv(b_dash)
    cc_dash = np.dot(b_dashinv,C[i])
    dd_dash = np.dot(b_dashinv , D[i] - np.dot(A[i],D_dash[i-1]))
    C_dash = np.append(C_dash, [cc_dash], axis = 0)
    D_dash = np.append(D_dash, [dd_dash], axis = 0)
```

Now, we have $X_{n-1} = (B_{n-1} - A_{n-1}C_{n-2}')^{-1}(D_{n-1} - C_{n-1}X_n - A_{n-1}D_{n-2}')$

In [23]:
```
b_dash = (B[n-1] - np.dot(A[n-1],C_dash[n-2]) )
b_dashinv = np.linalg.inv(b_dash)
final_ans = np.dot(b_dashinv, D[n-1] - np.dot(C[n-1],X_ans[n]) - np.dot(A[n-1], D_dash[n-2]))
X_ans[n-1] =  np.array(final_ans)
```

After that we have, $X_i = D'_i - C'_i X_{i+1} \; \forall \, i = (n-2), (n-3), \ldots, 1$

In [25]:
```
for i in range(n-2, 0,-1):
    #D_dash[i] - np.dot(C_dash[i], X_ans[i+1])
    X_ans[i] = np.array(D_dash[i] - np.dot(C_dash[i], X_ans[i+1]))
    print("X["+str(i) +"] = ")
    print(X_ans[i])
```

```
X[8] =
[[15.1648056 ]
 [ 1.09906354]]
X[7] =
[[13.21793271]
 [ 1.29468225]]
X[6] =
[[11.18506017]
 [ 1.47106283]]
X[5] =
[[9.07242237]
 [1.65421278]]
X[4] =
[[6.87814934]
 [1.83433328]]
X[3] =
[[4.63392996]
 [1.95410547]]
X[2] =
[[2.47040325]
 [1.87294795]]
X[1] =
[[0.71696464]
 [1.33392928]]
```

Now, the f_elements and F_elements need to be updated

In [26]:
```
for i in range(1,n):
        f_elements[i] = f_elements[i] + X_ans[i][0][0]
        F_elements[i] = F_elements[i] + X_ans[i][1][0]
```

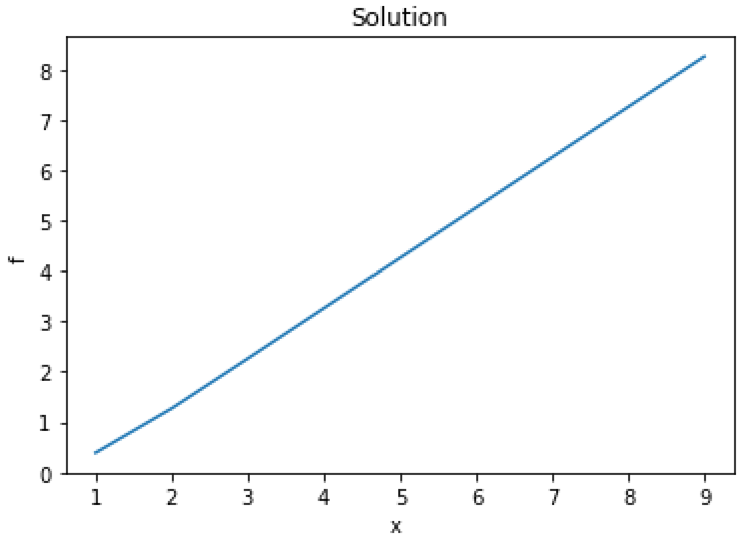
This has to be done again and again. We will make use of a loop to find this

In [46]:
```
f_0 = 0
F_0 = 0
F_n = 1.0
del_F = (F_n - F_0)/n
f_elements = np.array([])
F_elements = np.array([])
k = F_0
error = 0.00001
diff = 1
for i in range(n+1):
    f_elements = np.append(f_elements, 0)
    F_elements = np.append(F_elements, k)
    k = k + del_F
iteration = 0
while(diff > error):
    iteration = iteration +1
    if(iteration > 20):
        break
    print('\n Iteration : ' + str(iteration) )
    A = np.array([ [[g_a_dash_f_minus(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1]),g_a_dash_F_minus(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1])],[g_b_dash_f_minus(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1]), g_b_dash_F_minus(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1])]] ])
    B = np.array([ [[g_a_dash_f_i(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1]),g_a_dash_F_i(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1])],[g_b_dash_f_i(f_elements[0], f_elements[1] , f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1]), g_b_dash_F_i(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1])]] ])
    C = np.array([ [[g_a_dash_f_plus(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1]),g_a_dash_F_plus(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1])],[g_b_dash_f_plus(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1]), g_b_dash_F_plus(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1])]] ])
    D = np.array([ [[-g_a(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1])],[-g_b(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1])]] ])
    for i in range(1,n):
        A = np.append(A,[ [[g_a_dash_f_minus(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i]),g_a_dash_F_minus(f_elements[i-1], f_elemen
```

```
ts[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i])],[g_b_dash_f_m
inus(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[
i+1] , h, x[i]), g_b_dash_F_minus(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1
], F_elements[i], F_elements[i+1] , h, x[i])]] ], axis=0)
        B = np.append(B,[ [[g_a_dash_f_i(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_eleme
nts[i-1], F_elements[i], F_elements[i+1] , h, x[i]),g_a_dash_F_i(f_elements[i-1], f_elements[i] ,f
_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i])],[g_b_dash_f_i(f_eleme
nts[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[
i]), g_b_dash_F_i(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i
], F_elements[i+1] , h, x[i])]] ], axis=0)
        C = np.append(C,[ [[g_a_dash_f_plus(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_el
ements[i-1], F_elements[i], F_elements[i+1] , h, x[i]),g_a_dash_F_plus(f_elements[i-1], f_elements
[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i])],[g_b_dash_f_plu
s(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1
] , h, x[i]), g_b_dash_F_plus(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F
_elements[i], F_elements[i+1] , h, x[i])]] ], axis=0)
        D = np.append(D,[ [[-g_a(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1
], F_elements[i], F_elements[i+1] , h, x[i])],[-g_b(f_elements[i-1], f_elements[i] ,f_elements[i+1
] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i])]] ], axis=0)

    X_ans = np.array([[[0],[0]]],dtype = float)
    for i in range(n):
        X_ans = np.append(X_ans, [[[0],[0]]] ,axis = 0)
    #print('Here, we have : ')
    #for i in range(1,n):
    #    print(str(A[i]) + str(X_ans[i-1]) + ' + ' + str(B[i])+ str(X_ans[i]) + ' + ' + str(C[i])
 + str(X_ans[i+1]) + ' = '+ str(D[i]) )
    D[1] = D[1] - np.dot(A[1], X_ans[0])
    cc_dash = np.dot(np.linalg.inv(B[1]),C[1] )
    dd_dash = np.dot(np.linalg.inv(B[1]),D[1] )
    C_dash = np.array([cc_dash])
    D_dash = np.array([dd_dash])
    C_dash = np.append(C_dash, [cc_dash], axis = 0)
    D_dash = np.append(D_dash, [dd_dash], axis = 0)
    for i in range(2,n):
        b_dash = (B[i] - np.dot(A[i],C_dash[i-1]) )
        b_dashinv = np.linalg.inv(b_dash)
        cc_dash = np.dot(b_dashinv,C[i])
        dd_dash = np.dot(b_dashinv , D[i] - np.dot(A[i],D_dash[i-1]))
        C_dash = np.append(C_dash, [cc_dash], axis = 0)
        D_dash = np.append(D_dash, [dd_dash], axis = 0)
    b_dash = (B[n-1] - np.dot(A[n-1],C_dash[n-2]) )
    b_dashinv = np.linalg.inv(b_dash)
```

```
    final_ans = np.dot(b_dashinv, D[n-1] - np.dot(C[n-1],X_ans[n]) - np.dot(A[n-1], D_dash[n-2]))
    X_ans[n-1] =  np.array(final_ans)
    #print("X["+str(n-1) +"] = ")
    #print(X_ans[n-1])
    for i in range(n-2, 0,-1):
        #D_dash[i] - np.dot(C_dash[i], X_ans[i+1])
        X_ans[i] = np.array(D_dash[i] - np.dot(C_dash[i], X_ans[i+1]))
        #print("X["+str(i) +"] = ")
        #print(X_ans[i])

    diff = np.max(np.abs(X_ans))
    for i in range(1,n):
        f_elements[i] = f_elements[i] + X_ans[i][0][0]
        F_elements[i] = F_elements[i] + X_ans[i][1][0]

if(iteration < 20):
    print('The solution converged to a final answer \n And the values of f are ')
    for i in range(n):
        print('f(' +str(x[i])+') = ' +str(f_elements[i]) )
        print('F(' +str(x[i])+') = ' +str(F_elements[i]) + str('\n'))
    y_axis = np.array([],dtype = float)
    x_axis = np.array([],dtype = float)
    for i in range(1,n):
        x_axis = np.append(x_axis, x[i])
        y_axis = np.append(y_axis, f_elements[i])
    plt.title("Solution")
    plt.xlabel("x ")
    plt.ylabel("f ")
    plt.plot(x_axis,y_axis)
    plt.show()
```

```
 Iteration : 1

 Iteration : 2

 Iteration : 3

 Iteration : 4

 Iteration : 5

 Iteration : 6
The solution converged to a final answer
 And the values of f are
f(0.0) = 0.0
F(0.0) = 0.0

f(1.0) = 0.39043962253931846
F(1.0) = 0.7808792450786369

f(2.0) = 1.2709693835224427
F(2.0) = 0.9801802768876116

f(3.0) = 2.2613612057732966
F(3.0) = 1.000603367614096

f(4.0) = 3.2616215368650763
F(4.0) = 0.9999172945694632

f(5.0) = 4.261589648134574
F(5.0) = 1.0000189279695324

f(6.0) = 5.261596267691999
F(6.0) = 0.9999943111453187

f(7.0) = 6.261594540417186
F(7.0) = 1.0000022343050556

f(8.0) = 7.261595272102825
F(8.0) = 0.9999992290662226

f(9.0) = 8.261595188340909
F(9.0) = 1.000000603409944
```



Now, for a different value of h

In [48]:
```
h = 0.1
x_0 = 0
x_n = 10
n = int((x_n- x_0)/h  + 0.5 )
x = np.array([])
for i in range(n+1):
    x = np.append(x, i*h)
```

In [49]:
```
f_0 = 0
F_0 = 0
F_n = 1.0
del_F = (F_n - F_0)/n
f_elements = np.array([])
F_elements = np.array([])
k = F_0
error = 0.00001
diff = 1
for i in range(n+1):
    f_elements = np.append(f_elements, 0)
    F_elements = np.append(F_elements, k)
    k = k + del_F
iteration = 0
while(diff > error):
    iteration = iteration +1
    if(iteration > 20):
        break
    print('\n Iteration : ' + str(iteration) )
    A = np.array([ [[g_a_dash_f_minus(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1]),g_a_dash_F_minus(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1])],[g_b_dash_f_minus(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1]), g_b_dash_F_minus(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1])]] ])
    B = np.array([ [[g_a_dash_f_i(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1]),g_a_dash_F_i(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1])],[g_b_dash_f_i(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1]), g_b_dash_F_i(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1])]] ])
    C = np.array([ [[g_a_dash_f_plus(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1]),g_a_dash_F_plus(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1])],[g_b_dash_f_plus(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1]), g_b_dash_F_plus(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1])]] ])
    D = np.array([ [[-g_a(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1])],[-g_b(f_elements[0], f_elements[1] ,f_elements[2] , F_elements[0], F_elements[1], F_elements[2] , h, x[1])]] ])
    for i in range(1,n):
        A = np.append(A,[ [[g_a_dash_f_minus(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i]),g_a_dash_F_minus(f_elements[i-1], f_elemen
```

```
ts[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i])],[g_b_dash_f_minus(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i]), g_b_dash_F_minus(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i])]] ], axis=0)
        B = np.append(B,[ [[g_a_dash_f_i(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i]),g_a_dash_F_i(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i])],[g_b_dash_f_i(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i]), g_b_dash_F_i(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i])]] ], axis=0)
        C = np.append(C,[ [[g_a_dash_f_plus(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i]),g_a_dash_F_plus(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i])],[g_b_dash_f_plus(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i]), g_b_dash_F_plus(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i])]] ], axis=0)
        D = np.append(D,[ [[-g_a(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i])],[-g_b(f_elements[i-1], f_elements[i] ,f_elements[i+1] , F_elements[i-1], F_elements[i], F_elements[i+1] , h, x[i])]] ], axis=0)

    X_ans = np.array([[[0],[0]]],dtype = float)
    for i in range(n):
        X_ans = np.append(X_ans, [[[0],[0]]] ,axis = 0)
    #print('Here, we have : ')
    #for i in range(1,n):
    #    print(str(A[i]) + str(X_ans[i-1]) + ' + ' + str(B[i])+ str(X_ans[i]) + ' + ' + str(C[i]) + str(X_ans[i+1]) + ' = '+ str(D[i]) )
    D[1] = D[1] - np.dot(A[1], X_ans[0])
    cc_dash = np.dot(np.linalg.inv(B[1]),C[1] )
    dd_dash = np.dot(np.linalg.inv(B[1]),D[1] )
    C_dash = np.array([cc_dash])
    D_dash = np.array([dd_dash])
    C_dash = np.append(C_dash, [cc_dash], axis = 0)
    D_dash = np.append(D_dash, [dd_dash], axis = 0)
    for i in range(2,n):
        b_dash = (B[i] - np.dot(A[i],C_dash[i-1]) )
        b_dashinv = np.linalg.inv(b_dash)
        cc_dash = np.dot(b_dashinv,C[i])
        dd_dash = np.dot(b_dashinv , D[i] - np.dot(A[i],D_dash[i-1]))
        C_dash = np.append(C_dash, [cc_dash], axis = 0)
        D_dash = np.append(D_dash, [dd_dash], axis = 0)
    b_dash = (B[n-1] - np.dot(A[n-1],C_dash[n-2]) )
    b_dashinv = np.linalg.inv(b_dash)
```

```
    final_ans = np.dot(b_dashinv, D[n-1] - np.dot(C[n-1],X_ans[n]) - np.dot(A[n-1], D_dash[n-2]))
    X_ans[n-1] =  np.array(final_ans)
    #print("X["+str(n-1) +"] = ")
    #print(X_ans[n-1])
    for i in range(n-2, 0,-1):
        #D_dash[i] - np.dot(C_dash[i], X_ans[i+1])
        X_ans[i] = np.array(D_dash[i] - np.dot(C_dash[i], X_ans[i+1]))
        #print("X["+str(i) +"] = ")
        #print(X_ans[i])

    diff = np.max(np.abs(X_ans))
    for i in range(1,n):
        f_elements[i] = f_elements[i] + X_ans[i][0][0]
        F_elements[i] = F_elements[i] + X_ans[i][1][0]

if(iteration < 20):
    print('The solution converged to a final answer \n And the values of f are ')
    for i in range(n):
        print('f(' +str(x[i])+') = ' +str(f_elements[i]) )
        print('F(' +str(x[i])+') = ' +str(F_elements[i]) + str('\n'))
    y_axis = np.array([],dtype = float)
    x_axis = np.array([],dtype = float)
    for i in range(1,n):
        x_axis = np.append(x_axis, x[i])
        y_axis = np.append(y_axis, f_elements[i])
    plt.title("Solution")
    plt.xlabel("x ")
    plt.ylabel("f ")
    plt.plot(x_axis,y_axis)
    plt.show()
```

```
 Iteration : 1

 Iteration : 2

 Iteration : 3

 Iteration : 4

 Iteration : 5

 Iteration : 6
The solution converged to a final answer
 And the values of f are
f(0.0) = 0.0
F(0.0) = 0.0

f(0.1) = 0.005914678830995267
F(0.1) = 0.11829357661990533

f(0.2) = 0.023162360455530103
F(0.2) = 0.2266600558707914

f(0.30000000000000004) = 0.05076039047060412
F(0.30000000000000004) = 0.325300544430689

f(0.4) = 0.08775153964396078
F(0.4) = 0.41452243903644426

f(0.5) = 0.13321362581593255
F(0.5) = 0.49471928440299123

f(0.6000000000000001) = 0.18626720559033225
F(0.6000000000000001) = 0.5663523110850024

f(0.7000000000000001) = 0.2460815005270152
F(0.7000000000000001) = 0.6299335876486561

f(0.8) = 0.3118787169674288
F(0.8) = 0.6860107411596155

f(0.9) = 0.3829369147223141
F(0.9) = 0.7351532139380914
```

```
f(1.0) = 0.45859157708034093
F(1.0) = 0.7779400332224449

f(1.1) = 0.538236032441179
F(1.1) = 0.8149490739943165

f(1.2000000000000002) = 0.6213208758355451
F(1.2000000000000002) = 0.8467477938930056

f(1.3) = 0.7073525361883347
F(1.3) = 0.8738854131627867

f(1.4000000000000001) = 0.7958911319691192
F(1.4000000000000001) = 0.8968865024529019

f(1.5) = 0.8865477534831381
F(1.5) = 0.9162459278274755

f(1.6) = 0.978981304204884
F(1.6) = 0.932425086607444

f(1.7000000000000002) = 1.072895026081321
F(1.7000000000000002) = 0.9458493509212956

f(1.8) = 1.168032824599442
F(1.8) = 0.9569066194411242

f(1.9000000000000001) = 1.2641754987240597
F(1.9000000000000001) = 0.9659468630512276

f(2.0) = 1.3611369687922883
F(2.0) = 0.9732825383133479

f(2.1) = 1.4587605824317493
F(2.1) = 0.9791897344758722

f(2.2) = 1.5569155649562993
F(2.2) = 0.9839099160151269

f(2.3000000000000003) = 1.6554936669331362
F(2.3000000000000003) = 0.9876521235216089

f(2.4000000000000004) = 1.7544060481600916
```

```
F(2.4000000000000004) = 0.9905955010174993

f(2.5) = 1.8535804245638183
F(2.5) = 0.9928920270570338

f(2.6) = 1.952958492892004
F(2.6) = 0.9946693395066827

f(2.7) = 2.05249363780966
F(2.7) = 0.9960335588464369

f(2.8000000000000003) = 2.152148917312842
F(2.8000000000000003) = 0.9970720312172003

f(2.9000000000000004) = 2.251895315339261
F(2.9000000000000004) = 0.9978559293111858

f(3.0) = 2.351710245089517
F(3.0) = 0.9984426656939257

f(3.1) = 2.451576282800628
F(3.1) = 0.9988780885282945

f(3.2) = 2.5514801093963797
F(3.2) = 0.9991984433867419

f(3.3000000000000003) = 2.651411636391685
F(3.3000000000000003) = 0.9994320965193684

f(3.4000000000000004) = 2.7513632924380227
F(3.4000000000000004) = 0.9996010244073869

f(3.5) = 2.851329447741032
F(3.5) = 0.9997220816527925

f(3.6) = 2.951305955041144
F(3.6) = 0.9998080643494534

f(3.7) = 3.0512897877217395
F(3.7) = 0.9998685892624504

f(3.8000000000000003) = 3.1512787577193793
F(3.8000000000000003) = 0.9999108106903457
```

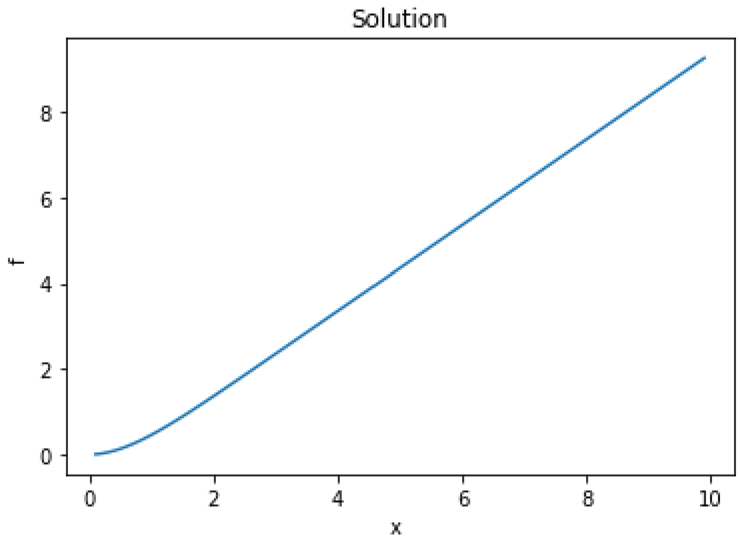
```
f(3.9000000000000004) = 3.2512712981093936
F(3.9000000000000004) = 0.9999399971099439

f(4.0) = 3.3512662974108656
F(4.0) = 0.9999599889194934

f(4.1000000000000005) = 3.451262974711851
F(4.1000000000000005) = 0.999973557100219

f(4.2) = 3.5512607866003627
F(4.2) = 0.9999826806700152

f(4.3) = 3.6512593585652096
F(4.3) = 0.9999887586269215

f(4.4) = 3.751258434988928
F(4.4) = 0.9999927698474476

f(4.5) = 3.851257843093536
F(4.5) = 0.9999953922447052

f(4.6000000000000005) = 3.9512574672303398
F(4.6000000000000005) = 0.9999970904913723

f(4.7) = 4.051257230746182
F(4.7) = 0.9999981798254771

f(4.800000000000001) = 4.151257083332825
F(4.800000000000001) = 0.9999988719073909

f(4.9) = 4.251256992297569
F(4.9) = 0.9999993073874718

f(5.0) = 4.351256936605136
F(5.0) = 0.9999995787638714

f(5.1000000000000005) = 4.451256902855229
F(5.1000000000000005) = 0.9999997462379769

f(5.2) = 4.5512568825963635
F(5.2) = 0.9999998485847357
```

```
f(5.300000000000001) = 4.651256870551547
F(5.300000000000001) = 0.9999999105189386

f(5.4) = 4.7512568634589645
F(5.4) = 0.9999999476294029

f(5.5) = 4.851256859322737
F(5.5) = 0.9999999696460442

f(5.6000000000000005) = 4.951256856933954
F(5.6000000000000005) = 0.9999999825782959

f(5.7) = 5.051256855567807
F(5.7) = 0.9999999900987708

f(5.800000000000001) = 5.151256854794163
F(5.800000000000001) = 0.9999999944283343

f(5.9) = 5.251256854360369
F(5.9) = 0.9999999968957961

f(6.0) = 5.3512568541195495
F(6.0) = 0.9999999982878114

f(6.1000000000000005) = 5.451256853987197
F(6.1000000000000005) = 0.9999999990651403

f(6.2) = 5.551256853915193
F(6.2) = 0.9999999994947883

f(6.300000000000001) = 5.651256853876425
F(6.300000000000001) = 0.9999999997298321

f(6.4) = 5.751256853855771
F(6.4) = 0.9999999998570926

f(6.5) = 5.85125685384489
F(6.5) = 0.9999999999252843

f(6.6000000000000005) = 5.951256853839226
F(6.6000000000000005) = 0.9999999999614462

f(6.7) = 6.051256853836319
```

```
F(6.7) = 0.9999999999804241

f(6.800000000000001) = 6.151256853834855
F(6.800000000000001) = 0.999999999990281

f(6.9) = 6.251256853834136
F(6.9) = 0.9999999999953484

f(7.0) = 6.3512568538338
F(7.0) = 0.999999999997928

f(7.1000000000000005) = 6.451256853833658
F(7.1000000000000005) = 0.9999999999992292

f(7.2) = 6.551256853833614
F(7.2) = 0.9999999999998808

f(7.300000000000001) = 6.651256853833618
F(7.300000000000001) = 1.0000000000002058

f(7.4) = 6.751256853833647
F(7.4) = 1.0000000000003686

f(7.5) = 6.8512568538336875
F(7.5) = 1.0000000000004514

f(7.6000000000000005) = 6.951256853833735
F(7.6000000000000005) = 1.0000000000004958

f(7.7) = 7.051256853833786
F(7.7) = 1.0000000000005214

f(7.800000000000001) = 7.151256853833839
F(7.800000000000001) = 1.0000000000005378

f(7.9) = 7.251256853833893
F(7.9) = 1.00000000000055

f(8.0) = 7.351256853833949
F(8.0) = 1.00000000000056

f(8.1) = 7.451256853834005
F(8.1) = 1.0000000000005687
```

```
f(8.200000000000001) = 7.5512568538340625
F(8.200000000000001) = 1.000000000000576

f(8.3) = 7.651256853834121
F(8.3) = 1.0000000000005824

f(8.4) = 7.751256853834179
F(8.4) = 1.0000000000005875

f(8.5) = 7.851256853834238
F(8.5) = 1.000000000000591

f(8.6) = 7.951256853834297
F(8.6) = 1.0000000000005929

f(8.700000000000001) = 8.051256853834357
F(8.700000000000001) = 1.0000000000005924

f(8.8) = 8.151256853834415
F(8.8) = 1.0000000000005893

f(8.9) = 8.251256853834475
F(8.9) = 1.0000000000005829

f(9.0) = 8.351256853834531
F(9.0) = 1.0000000000005724

f(9.1) = 8.451256853834588
F(9.1) = 1.0000000000005576

f(9.200000000000001) = 8.551256853834643
F(9.200000000000001) = 1.0000000000005371

f(9.3) = 8.651256853834695
F(9.3) = 1.00000000000051

f(9.4) = 8.751256853834745
F(9.4) = 1.0000000000004752

f(9.5) = 8.85125685383479
F(9.5) = 1.000000000000431
```

```
f(9.600000000000001) = 8.951256853834831
F(9.600000000000001) = 1.000000000000376

f(9.700000000000001) = 9.051256853834865
F(9.700000000000001) = 1.000000000000308

f(9.8) = 9.15125685383489
F(9.8) = 1.0000000000002247

f(9.9) = 9.251256853834908
F(9.9) = 1.0000000000001235
```



In [ ]: