Assignment 1 ME5102

Aakash
Email: me16b001@iittp.ac.in
Find the code: Python Code

January 28, 2019

Method of weighted residuals

Given the differential equation

$$\frac{d}{dt}\left[x\frac{du}{dx}\right] = \frac{2}{x^2}$$

Boundary conditions

$$u(1) = 2$$

and

$$-x\frac{du}{dx}\bigg|_{x=2} = \frac{1}{2}$$

Assuming the trial function as

$$u_h(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

Considering the first four terms only

$$u_h^{\sim}(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$$

Using boundary condition 1, we have

$$a_0 + a_1 + a_2 + a_3 = 2$$

Using boundary condition 2, we have

$$a_1 + 4a_2 + 12a_3 = -\frac{1}{4}$$

Substituting the above to results in the trial function

$$u_h^{\sim}(x) = 2 - \frac{1}{4}(x-1) + a_2(x-1)(x-3) + a_3(x-1)(x^2+x-11)$$

In order to get the residual, R we substitute our trial function in the given differential equation

$$R = \frac{d}{dt}\left[x\frac{du}{dx}\right] - \frac{2}{x^2}$$

$$\frac{du}{dx} = \frac{12a_3x^2 + 8a_2x - 48a_3 - 16a_2 - 1}{4}$$
$$R = \frac{36a_3x^2 + 16a_2x - 48a_3 - 16a_2 - 1}{4} - \frac{2}{x^2}$$

Exact Solution

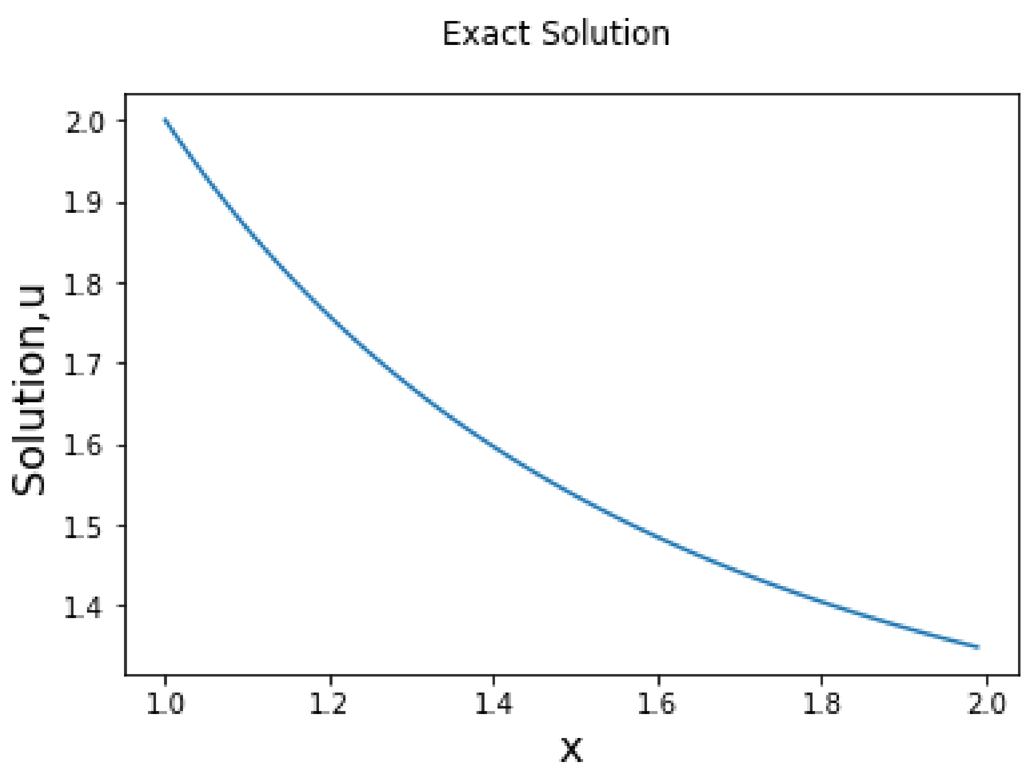
We obtain the following by integrating twice the given differential equation

$$u(x) = \frac{2}{x} + c_1 lnx + c_2$$

Finding values of the integration constants and putting them back

$$u = \frac{2}{x} + \frac{lnx}{2}$$

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

x = np.arange(1, 2, 0.01)
#  expression for exact solution
u_exact = (2/x)+0.5*np.log(x)

plt.plot(x, u_exact)
plt.xlabel('x', fontsize=16)
plt.ylabel('Solution,u', fontsize=16)
plt.suptitle('Exact Solution')
plt.show()
```

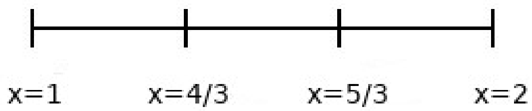


Collocation Method

In the collocation method, we force the residual to be zero at n points $x_1, x_2, \ldots, x_n$ within the domain.That is

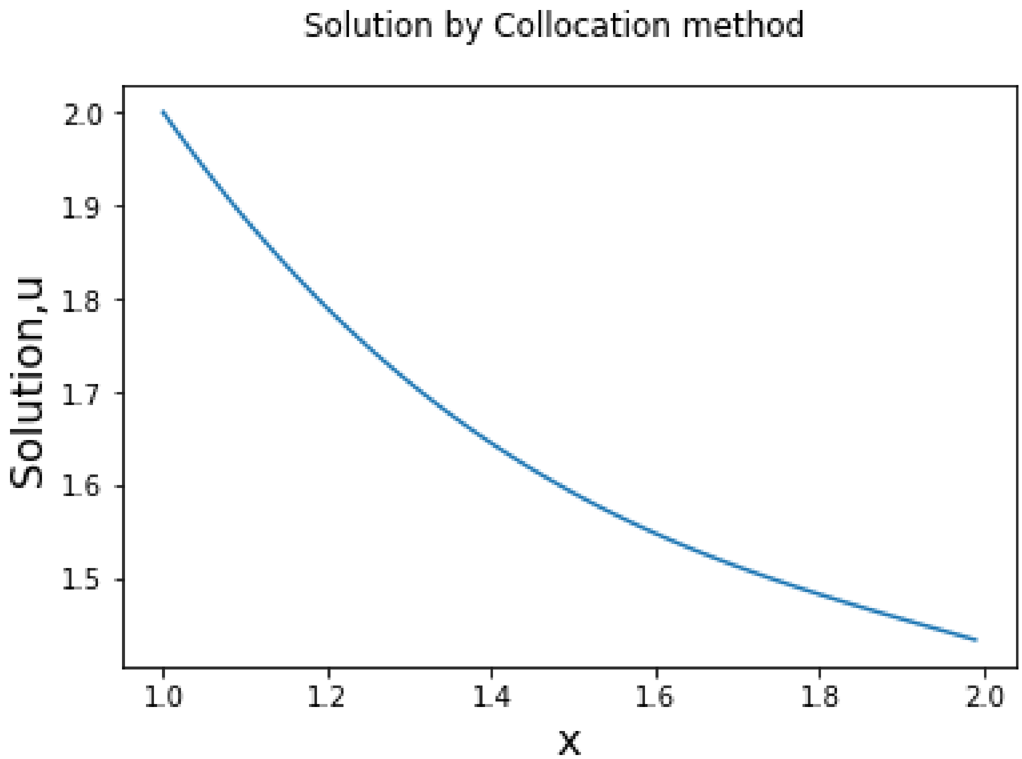
$$R(x_1, a) = 0$$
$$R(x_2, a) = 0$$
$$a = [a_1, a_2]$$

From the above equation we will get the value of a, and solution can be obtained by substituting it back in our trial function. For our problem, the domain of interest is $1 \leq x \leq 2$. Let us pick two points in this domain x_1 and x_2 such that $1 \leq x_1 < x_2 \leq 2$. In this example we choose $x_1 = 4/3$ and $x_2 = 5/3$.




```
In [33]: import matplotlib.pyplot as plt
         import numpy as np
         from sympy import *

         a_col = Symbol('a')
         b_col = Symbol('b')
         val_col = [4/3 ,5/3]
         eqn_col =[]

         def function(x):
                 for x in val_col:
                         # calculating residuals
                         eqn_col.append(-0.25+4*(x-1)*a_col+3*(3*(x**2)-4)*b_col-(2/(x**2)))
                 return(eqn_col)

         solve(function(val_col), [a_col,b_col])
         print("eqn",eqn_col)

         z_col = solve(function(val_col), [a_col,b_col])
         print("a ",z.get(a_col))
         print("b ",z.get(b_col))

         x = np.arange(1, 2, 0.01)
         u_col = 2. - (0.25)*(x-1)+(x-1)*(x-3)*z_col.get  (a_col)+(x-1)*(x**2+x-11)*z_col.get(b_
         plt.plot(x, u_col)
         plt.xlabel('x', fontsize=16)
         plt.ylabel('Solution,u', fontsize=16)
         plt.suptitle('Solution by Collocation method')
         plt.show()
```

```
eqn [1.33333333333333*a + 4.0*b - 1.375, 2.66666666666667*a + 13.0*b - 0.97]
a  2.09925000000000
b  -0.356000000000000
```



Subdomain Method

The subdomain method is another way of forcing the residuals to zero. In this method, we let the "average" of the residual vanish over each domain.

$$\frac{1}{\Delta x_i}\int_{\Delta x_i} R(x)dx = 0$$

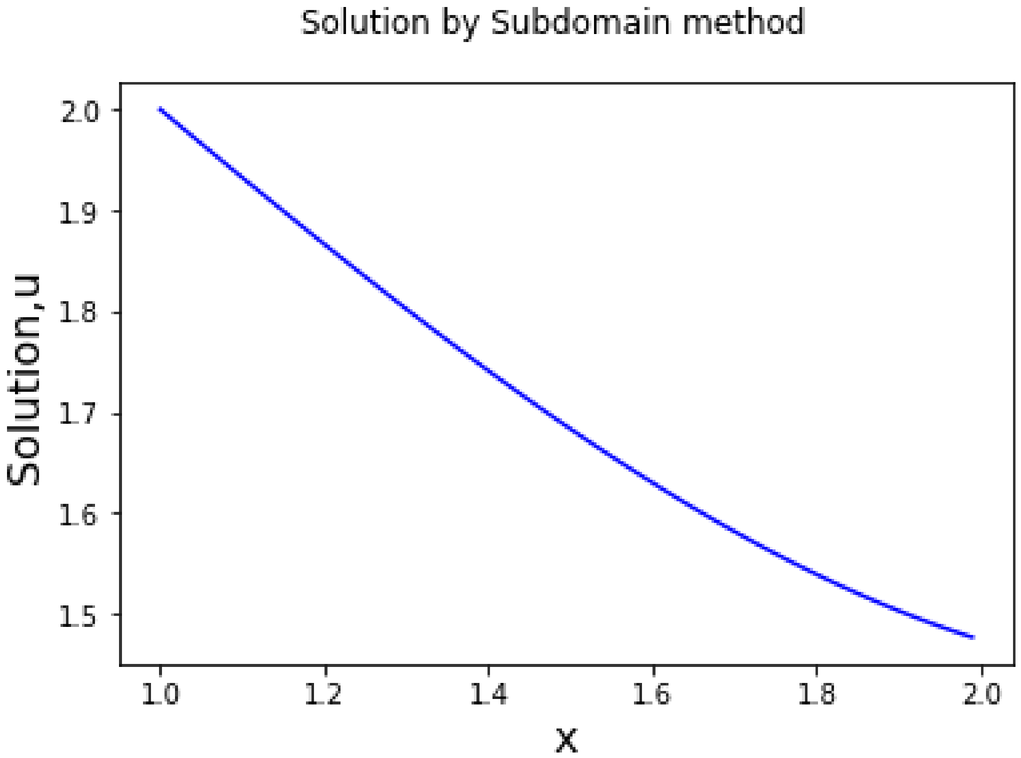
$$\frac{1}{\Delta x_1}\int_1^{\frac{3}{2}} R(x)dx = -\frac{19}{24} + 2a_2 + \frac{9}{8}a_3 = 0 \quad \text{and} \quad \frac{1}{\Delta x_2}\int_{\frac{3}{2}}^{2} R(x)dx = -\frac{11}{24} + \frac{3}{2}a_2 + \frac{63}{8}a_3 = 0$$

solving the above for a and substituiting it back in the trial function.

```
In [34]: from sympy import *
         import matplotlib.pyplot as plt
         import numpy as np
         import matplotlib.patches as mpatches

         a_sub = Symbol('a')
```

```
b_sub = Symbol('b')
x_sub = Symbol('x_sub')
eqn_sub =[]
eqn_sub.append(integrate(-0.25+4*(-1)*a_sub+3*(3*(x_sub**2)-4)*b_sub-(2/(x_sub**2)), (x
eqn_sub.append(integrate(-0.25+4*(x_sub-1)*a_sub+3*(3*(x_sub**2)-4)*b_sub-(2/(x_sub**2)
z_sub = solve(eqn_sub, [a_sub,b_sub])
print("a",z_sub.get(a_sub))
print("b",z_sub.get(b_sub))
x_sub = np.arange(1, 2., 0.01)
u_sub = 2 - 0.25*(x_sub-1)+(x_sub-1)*(x_sub-3)*z_sub.get(a_sub)+(x_sub-1)*(x_sub**2+x_s

plt.plot(x_sub, u_sub , color='b', label="Subdomain")
plt.xlabel('x', fontsize=16)
plt.ylabel('Solution,u', fontsize=16)
plt.suptitle('Solution by Subdomain method')
plt.show()
```

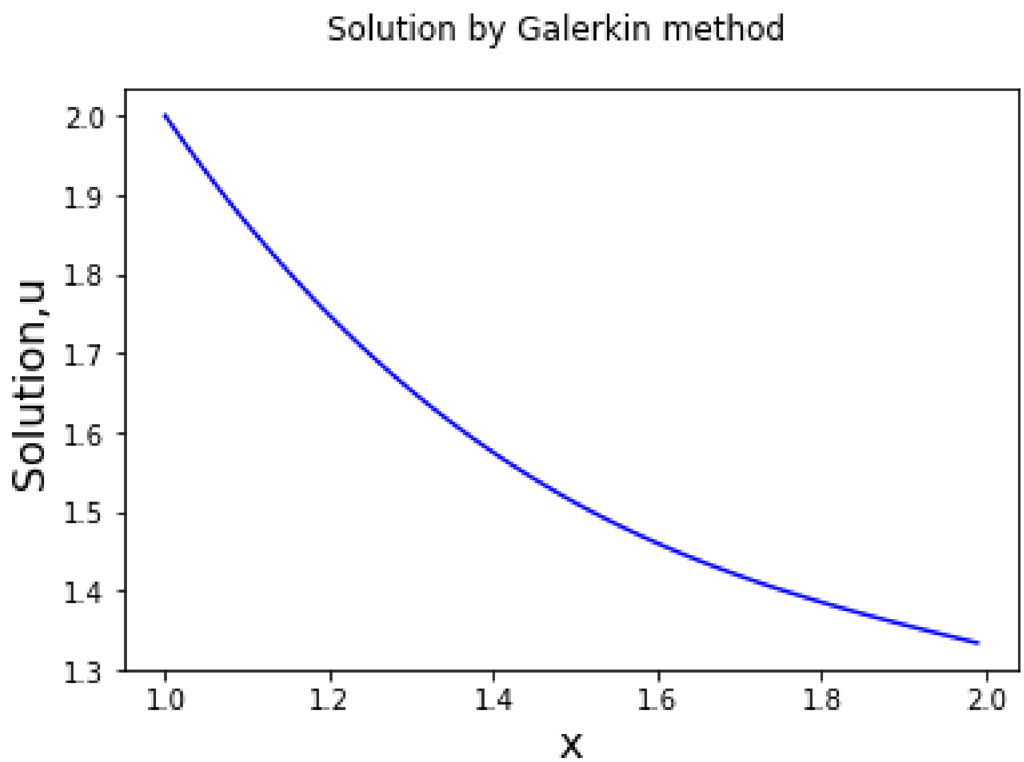
```
a -0.327956989247312
b 0.120669056152927
```



Galerkin Method

```
In [36]: from sympy import *
         import matplotlib.pyplot as plt
         import numpy as np
         import matplotlib.patches as mpatches

         a_gal = Symbol('a')
         b_gal = Symbol('b')
         x_gal = Symbol('x')
         eqn_gal =[]
         r_gal = -0.25+4*(x_gal-1)*a_gal+3*(3*(x_gal**2)-4)*b_gal-(2/(x_gal**2))
         print(diff(r_gal,a_gal))
         eqn_gal.append(integrate(r_gal*x_gal, (x_gal, 1, 2)))
         eqn_gal.append(integrate(r_gal*x_gal*x_gal, (x_gal, 1, 2)))
         print(eqn_gal)
         z_gal = solve(eqn_gal, [a_gal,b_gal])

         x = np.arange(1, 2., 0.01)
         u_gal = 2 - 0.25*(x-1)+(x-1)*(x-3)*z_gal.get(a_gal)+(x-1)*((x**2)+x-11)*z_gal.get(b_gal
         plt.plot(x, u_gal , color='b')
         plt.xlabel('x', fontsize=16)
         plt.ylabel('Solution,u', fontsize=16)
         plt.suptitle('Solution by Galerkin method')

         plt.show()
```

```
4*x - 4
[3.33333333333333*a + 15.75*b - 2.0*log(2) - 0.375, 5.66666666666667*a + 27.8*b - 2.583333333333
```



Least Squares Method

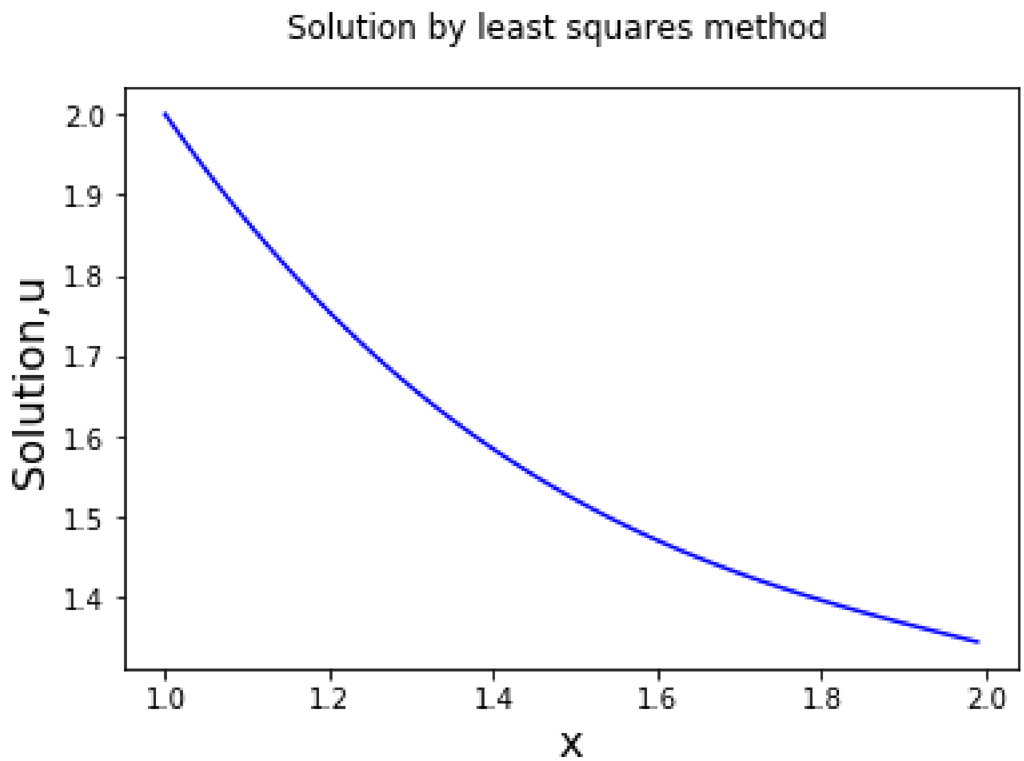
In this method we force the residuals to be zero as

$$\int_0^1 R(x)\,\frac{\partial R(x)}{\partial a_i}\,dx = 0$$

In [37]:
```
from sympy import *
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.patches as mpatches

a_lea = Symbol('a')
b_lea = Symbol('b')
x_lea = Symbol('x')
eqn_lea =[]
r_lea = -0.25+4*(x_lea-1)*a_lea+3*(3*(x_lea**2)-4)*b_lea-(2/(x_lea**2))
print(diff(r_lea,a_lea))
eqn_lea.append(integrate(r_lea*diff(r_lea,a_lea), (x_lea, 1, 2)))
eqn_lea.append(integrate(r_lea*diff(r_lea,b_lea), (x_lea, 1, 2)))
print(eqn_lea)
z_lea = solve(eqn_lea, [a_lea,b_lea])
print("a",z.get(a))
```

```
print("b",z.get(b))
x = np.arange(1, 2., 0.01)
u_lea = 2 - 0.25*(x-1)+(x-1)*(x-3)*z_lea.get(a_lea)+(x-1)*((x**2)+x-11)*z_lea.get(b_lea
plt.plot(x, u_lea , color='b')
plt.xlabel('x', fontsize=16)
plt.ylabel('Solution,u', fontsize=16)
plt.suptitle('Solution by least squares method')
plt.show()
```

```
4*x - 4
[5.33333333333333*a + 27.0*b - 8.0*log(2) + 3.5, 27.0*a + 142.2*b - 8.25]
a 2.09925000000000
b -0.356000000000000
```

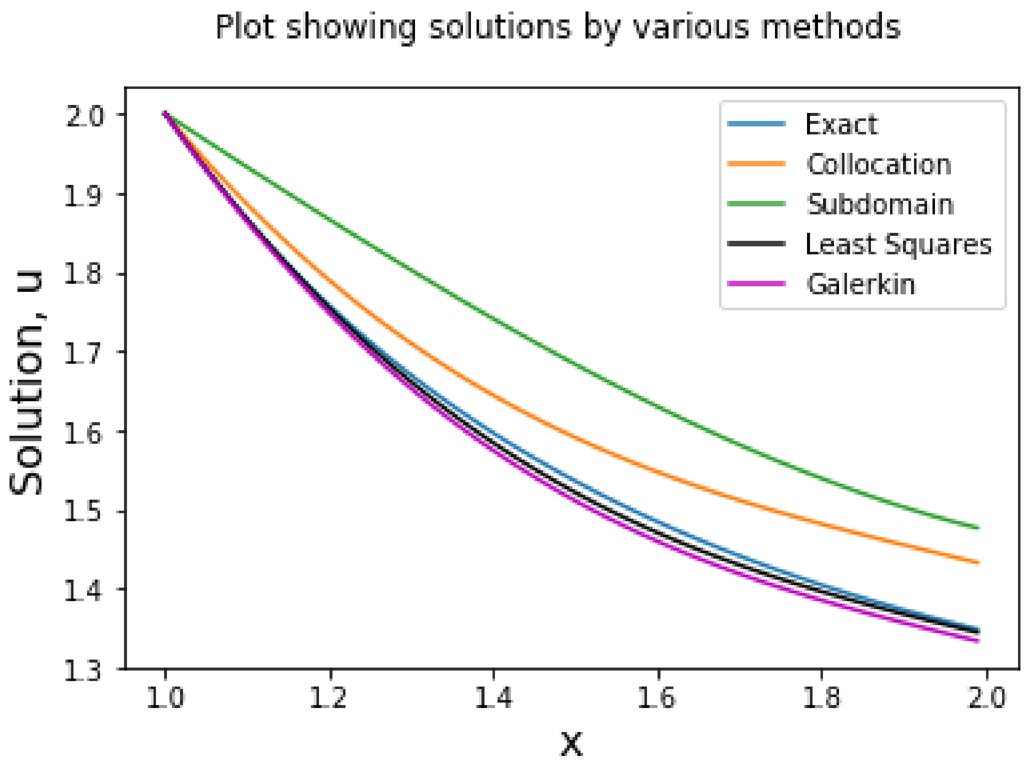


Comparision

```
In [38]: import matplotlib.pyplot as plt
         import numpy as np
         from sympy import *
         ############################## col
         a_col = Symbol('a')
         b_col = Symbol('b')
```

```
val_col = [4/3 ,5/3]
eqn_col =[]

def function(x):
        for x in val_col:
                # calculating residuals
                eqn_col.append(-0.25+4*(x-1)*a_col+3*(3*(x**2)-4)*b_col-(2/(x**2)))
        return(eqn_col)

z_col = solve(function(val_col), [a_col,b_col])
############################# sub
a_sub = Symbol('a')
b_sub = Symbol('b')
x_sub = Symbol('x_sub')
eqn_sub =[]
eqn_sub.append(integrate(-0.25+4*(-1)*a_sub+3*(3*(x_sub**2)-4)*b_sub-(2/(x_sub**2)), (x
eqn_sub.append(integrate(-0.25+4*(x_sub-1)*a_sub+3*(3*(x_sub**2)-4)*b_sub-(2/(x_sub**2)
z_sub = solve(eqn_sub, [a_sub,b_sub])

############################# least square
a_lea = Symbol('a')
b_lea = Symbol('b')
x_lea = Symbol('x')
eqn_lea =[]
r_lea = -0.25+4*(x_lea-1)*a_lea+3*(3*(x_lea**2)-4)*b_lea-(2/(x_lea**2))
eqn_lea.append(integrate(r_lea*diff(r_lea,a_lea), (x_lea, 1, 2)))
eqn_lea.append(integrate(r_lea*diff(r_lea,b_lea), (x_lea, 1, 2)))
z_lea = solve(eqn_lea, [a_lea,b_lea])
############################# galerian method
a_gal = Symbol('a')
b_gal = Symbol('b')
x_gal = Symbol('x')
eqn_gal =[]
r_gal = -0.25+4*(x_gal-1)*a_gal+3*(3*(x_gal**2)-4)*b_gal-(2/(x_gal**2))
eqn_gal.append(integrate(r_gal*x_gal, (x_gal, 1, 2)))
eqn_gal.append(integrate(r_gal*x_gal*x_gal, (x_gal, 1, 2)))
z_gal = solve(eqn_gal, [a_gal,b_gal])
############################# method end

x = np.arange(1, 2, 0.01)

u_col = 2. - (0.25)*(x-1)+(x-1)*(x-3)*z_col.get(a_col)+(x-1)*(x**2+x-11)*z_col.get(b_co
u_exact = (2/x)+0.5*np.log(x)
u_sub = 2 - 0.25*(x-1)+(x-1)*(x-3)*z_sub.get(a_sub)+(x-1)*(x**2+x-11)*z_sub.get(b_sub)
u_lea = 2 - 0.25*(x-1)+(x-1)*(x-3)*z_lea.get(a_lea)+(x-1)*((x**2)+x-11)*z_lea.get(b_lea
u_gal = 2 - 0.25*(x-1)+(x-1)*(x-3)*z_gal.get(a_gal)+(x-1)*((x**2)+x-11)*z_gal.get(b_gal

plt.plot(x, u_exact, label="Exact")
```

```
plt.plot(x, u_col, label="Collocation")
plt.plot(x, u_sub, label="Subdomain")
plt.plot(x, u_lea, label="Least Squares", color="black")
plt.plot(x, u_gal, label="Galerkin", color ="m")

plt.legend()
plt.xlabel('x', fontsize=16)
plt.ylabel('Solution, u', fontsize=16)
plt.suptitle('Plot showing solutions by various methods')
plt.show()
```




```
In [ ]:
```