In [7]:

```
%run CPL_Library.ipynb # Running the entire library in one line because
                       # importing does not work with JupyterLab
```

P346 Comp Phy Lab - Mid-sem exam

Date - 9 Oct 2021

Chandan Kumar Sahu - 1911055

Question 1

In [8]:

```
# Putting a precision 2 more than asked in question for better answer

def f1(x):
    return (x-5)*math.exp(x)+5

eps=10**-6 # giving accuracy 2 more than required for better functioning

print("NEWTON RAPHSON METHOD\n")
# guess # Chandan Kumar Sahu - 1911055
x=5
root=newton_raphson(x,f1)
print("Nearest root of the given function for the given value of x = " + str(x) + " is = "+str(root))

# Given x = hc/LkT (L = lambda)
# b = LT = hc/kx
h=6.626*10**(-34) # Js
k=1.381*10**(-23) # kg/Ks^2
c=3*10**8 # m/s # Chandan Kumar Sahu - 1911055

b = (h*c)/(k*x)
print("\nValue of b upto atleast upto precision 10^-4 is = " + str(ROUND(b,6)))
```

```
NEWTON RAPHSON METHOD

Nearest root of the given function for the given value of x = 5 is = 4.965114231744276

Value of b upto atleast upto precision 10^-4 is = 0.002879
```

Question 2

In [9]:

```
print("The augmented matrix is: ")
A,ro,co=read_matrix('MSEmatrixA.txt')

print_matrix(A,ro,co)

GJ, d=gauss_jordan(A,ro,co)
print("Determinant of the given matrix is = " + str(d))
print("Det is not zero, inverse exists.")
print()
C,ro,co=read_matrix('MSEmatrixA.txt')

if GJ!=None:
    if d!=0:
        # Chandan Kumar Sahu - 1911055
        # Finding the inverse and printing in rounded form
        M=get_inv(A,ro)

        MM,k,l=matrix_multiply(M,ro,ro,C,ro,ro) # using only n x n matrix i.e. unaugmented matrix
        M=round_matrix(M)

        print("The inverse matrix is: ")
        print_matrix(M,ro,ro)

        print("Verification: after multiplying the matrix and its inverse, we get : ")
        MM=round_matrix(MM) # Chandan Kumar Sahu - 1911055
        print_matrix(MM,ro,ro)

    else:
        print("No solution")
else:
    print("No solution")
```

```
The augmented matrix is:
0.0    0.0    0.0    2.0    1.0    0.0    0.0    0.0
0.0    0.0    3.0    0.0    0.0    1.0    0.0    0.0
0.0    4.0    0.0    0.0    0.0    0.0    1.0    0.0
5.0    0.0    0.0    0.0    0.0    0.0    0.0    1.0

Determinant of the given matrix is = 120.0
The inverse matrix is:
0    0    0    0.2
0    0    0.25    0
0    0.33    0    0
0.5    0    0    0

Verification: after multiplying the matrix and its inverse, we get :
1.0    0    0    0
0    1.0    0    0
0    0    1.0    0
0    0    0    1.0
```

Question 3

In [10]:

```
# LU decomposition using Doolittle's condition L[i][i]=1

print("The matrix is: ")
A1,ro,co = read_matrix('MSEmatrixB.txt')
print_matrix(A1,ro,co)

vector=[-9, 5, 7, 11] # Chandan Kumar Sahu - 1911055

# partial pivoting to avoid division by zero at pivot place
A1, vector = partial_pivot_LU(A1, vector, ro)
A1 = LU_doolittle(A1,ro)

x = [0 for i in range(ro)]

x = for_back_subs_doolittle(A1,ro,vector)

print("Solutions are : ")
for i in range(ro): # Chandan Kumar Sahu - 1911055
    print("x["+str(i)+"] = "+str(x[i]))

# Checking if these are correct, putting in first equation, must give -9
print()
print(3*x[0]-7*x[1]-2*x[2]+2*x[3])
```

```
The matrix is:

3.0    -7.0    -2.0    2.0

-3.0    5.0    1.0    0.0

6.0    -4.0    0.0    -5.0

-9.0    5.0    -5.0    12.0


Solutions are :
x[0] = 3.0
x[1] = 4.0
x[2] = -6.0
x[3] = -1.0

-9.0
```

Question 4

In [11]:

```
def f2(x):
    return 4*math.exp(-x)*math.sin(x)-1

eps=10**-6

p=0
q=1 # Chandan Kumar Sahu - 1911055
a,b=bracketing(p,q,f2)

print("\nBISECTION METHOD")
root=bisection(a,b,f2)
if p==a and q==b:
    print("Root of the given function in the interval (" + str(p) + "," + str(q) + ") = "+str(root))
else:
    print("Root does not lie in the given range (" + str(p) + "," + str(q)+")")
    print("We change the interval to (" + str(a) + "," + str(b)+")")
    print("Root of the given function in the interval (" + str(a) + "," + str(b) + ") is "+str(root))

print("\nREGULA FALSI METHOD")
root=regula_falsi(a,b,f2)
if p==a and q==b:
    print("Root of the given function in the interval (" + str(p) + "," + str(q) + ") = "+str(root))
else:
    print("Root does not lie in the given range (" + str(p) + "," + str(q)+")")
    print("We change the interval to (" + str(a) + "," + str(b)+")") # Chandan Kumar Sahu - 1911055
    print("Root of the given function in the interval (" + str(a) + "," + str(b) + ") is "+str(root))
```

```
BISECTION METHOD
Root of the given function in the interval (0,1) = 0.3705587387084961

REGULA FALSI METHOD
Root of the given function in the interval (0,1) = 0.3705584003334566
```

In [12]:

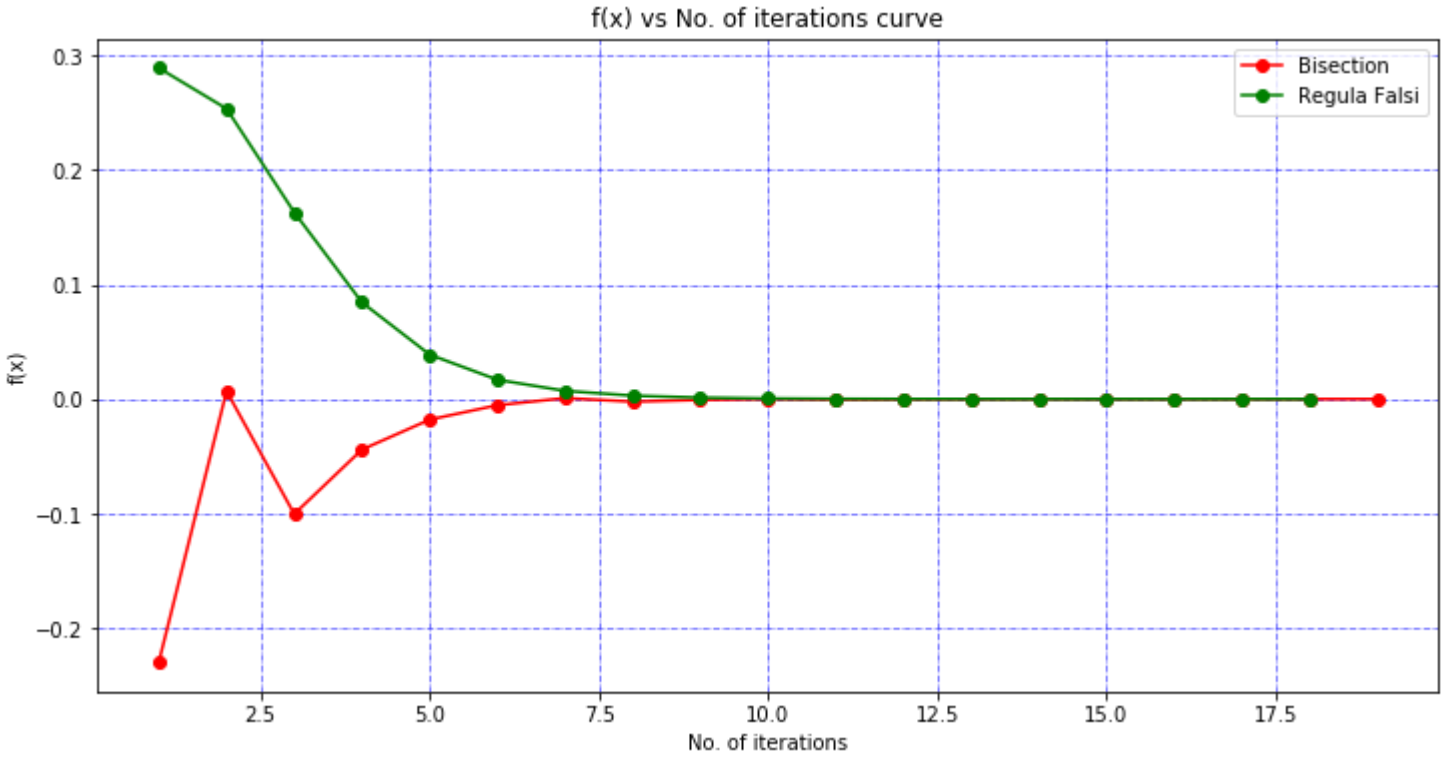
```
plt.figure(figsize=(12,6))

p=0
q=1

a,b=bracketing(p,q,f2) # Chandan Kumar Sahu - 1911055
x_bis, y_bis, z_bis = bisection_for_plotting(a,b,f2)
x_rf, y_rf, z_rf = regula_falsi_for_plotting(a,b,f2)

print("\nBISECTION METHOD")
print ("{:<20} {:<20} {:<20}".format('No. of iterations', 'f(x)', 'Root convergence'))
for i in range(len(x_bis)):
    print ("{:<20} {:<20} {:<20}".format(ROUND(x_bis[i],6), ROUND(y_bis[i],6), ROUND(z_bis[i],6)))

print("\n\nREGULA FALSI METHOD")
print ("{:<20} {:<20} {:<20}".format('No. of iterations', 'f(x)', 'Root convergence'))
for i in range(len(x_rf)):
    print ("{:<20} {:<20} {:<20}".format(ROUND(x_rf[i],6), ROUND(y_rf[i],6), ROUND(z_rf[i],6)))

plt.plot(x_bis, y_bis, 'r-o', label='Bisection')
plt.plot(x_rf, y_rf, 'g-o', label='Regula Falsi')
# Chandan Kumar Sahu - 1911055
plt.grid(color='b', ls = '-.', lw = 0.5)
plt.xlabel('No. of iterations')
plt.ylabel('f(x)')
plt.title('f(x) vs No. of iterations curve')
plt.legend()
plt.show()
```

BISECTION METHOD

No. of iterations	f(x)	Root convergence
1.0	-0.229286	0.25
2.0	0.006941	0.375
3.0	-0.100293	0.3125
4.0	-0.044068	0.34375
5.0	-0.017925	0.359375
6.0	-0.005334	0.367188
7.0	0.000842	0.371094
8.0	-0.002236	0.369141
9.0	-0.000694	0.370117
10.0	7.5e-05	0.370605
11.0	-0.00031	0.370361
12.0	-0.000118	0.370483
13.0	-2.2e-05	0.370544
14.0	2.7e-05	0.370575
15.0	3e-06	0.37056
16.0	-9e-06	0.370552
17.0	-3e-06	0.370556
18.0	-0.0	0.370558
19.0	1e-06	0.370559

REGULA FALSI METHOD

No. of iterations	f(x)	Root convergence
1.0	0.288962	0.807598
2.0	0.253465	0.626549
3.0	0.163004	0.499854
4.0	0.084451	0.429796
5.0	0.038868	0.396325
6.0	0.016911	0.381497
7.0	0.007178	0.375153
8.0	0.003014	0.372479
9.0	0.00126	0.37136
10.0	0.000526	0.370892
11.0	0.000219	0.370697
12.0	9.1e-05	0.370616
13.0	3.8e-05	0.370582
14.0	1.6e-05	0.370568
15.0	7e-06	0.370562
16.0	3e-06	0.37056
17.0	1e-06	0.370559
18.0	0.0	0.370558


f(x) vs No. of iterations curve
Bisection
Regula Falsi
f(x)
No. of iterations
0.3
0.2
0.1
0.0
−0.1
−0.2
2.5
5.0
7.5
10.0
12.5
15.0
17.5

In [ ]:

```
```