In [7]:

```python
import numpy as np
import matplotlib.pyplot as plt
import math


expr = input("Enter the function(in terms of x):")

def xaxis(inp):
    return 0

def func(inp):
    x=inp
    return eval(expr)

def g(gexpr,inp):
    x=inp
    return eval(gexpr)

def derivFunc(dexpr,inp):
    x=inp
    return eval(dexpr)

def curveplot(arr,i):
    x = np.arange(1, i)
    y = np.linspace(-50, 50, 10000)
    f2 = np.vectorize(func)
    f3 = np.vectorize(xaxis)

    plt.figure(1)
    plt.xlabel("x")
    plt.ylabel("f(x)")
    plt.plot(y, f2(y), color="blue")
    plt.plot(y, f3(y), color="green")
    plt.figure(2)
    plt.xlabel("Iteration Number")
    plt.ylabel("Percentage Relative Error")
    plt.bar(x, arr[x - 1], color='red')
    plt.show()


def bisection():
    arr=np.array([])
    a = float(input("Enter the value of x1:\n"))
    b = float(input("Enter the value of x2:"))
    n = int(input("number of iterations"))
    er = float(input("maximum relative error as %"))
    i=1
    if (func(a) * func(b) >= 0):
        print("You have not assumed right initial values\n")
        return

    c = 0.0001
    oldr = 0
    condition = True
    #((b - a) >= 0.0001) &
    while ( i==1 or condition):

        # Find middle point
        c = (a + b) / 2
        i=i+1

        arr = np.append(arr, (math.fabs(oldr - c) / math.fabs(c)) * 100)
```

```python
        condition = (((( math.fabs(oldr-c) / math.fabs(c)) * 100) >= er) and (i <=
        oldr=c


        # Check if middle point is root
        if (func(c) == 0.0):
            break

        # Decide the side to repeat the steps
        if (func(c) * func(a) < 0):
            b = c
        else:
            a = c


    print("The value of root is : ", "%.4f" % c)
    curveplot(arr,i)

def regulaFalsi():
        arr = np.array([])
        a = float(input("Enter the value of x1:\n"))
        b = float(input("Enter the value of x2:"))
        n = int(input("number of iterations"))
        er = float(input("maximum relative error as %"))
        if func(a) * func(b) >= 0:
            print("You have not assumed right a and b")
            return

        c = 0.001  # Initialize result
        oldr=0
        i=1
        condition = True
        while (i==1 or condition):
            i=i+1
            # Find the point that touches x axis
            c = (a * func(b) - b * func(a)) / (func(b) - func(a))
            # Check if the above found point is root
            if func(c) == 0:
                break
            # Decide the side to repeat the steps
            if func(c) * func(a) < 0:
                b = c
            else:
                a = c
            arr = np.append(arr, (math.fabs(oldr-c) / math.fabs(c)) * 100)
            condition= (((( math.fabs(oldr-c) / math.fabs(c)) * 100) >= er) and (i
            oldr=c

        print("The value of root is : ", '%.4f' % c)
        curveplot(arr,i)



def fixedPointIteration():
    # Input Section
    arr = np.array([])
    gexpr = input("Enter the function g(x) such that g(x)=x (in terms of x):")
    x0 = float(input('Enter Guess: '))
    e = float(input('Tolerable Error %: '))
    N = int(input('Maximum Step: '))

    print('\n\n*** FIXED POINT ITERATION ***')
    i = 1
    flag = 1
```

```python
    condition = True
    while condition:
        x1 = g(gexpr,x0)
        #print('Iteration-%d, x1 = %0.6f and f(x1) = %0.6f' % (step, x1, func(x1)),
        arr = np.append(arr, (math.fabs(x1 - x0) / math.fabs(x1)) * 100)
        condition =  (math.fabs(x1 - x0) / math.fabs(x1)) * 100 > e
        x0 = x1

        i = i + 1

        if i > N:
            flag = 0
            break
    if flag == 1:
        print('\nRequired root is: %0.4f' % x1)
    else:
        print('\nNot Convergent.')

    curveplot(arr,i)


def newtonRaphson():

    arr = np.array([])
    dexpr = input("Enter the derivative(in terms of x):")
    x = float(input('Enter Guess: '))
    er = float(input('Enter relative error percentage: '))
    n = int(input("number of iterations"))

    i=1
    while(i<=n):

        i=i+1
        h = func(x) / derivFunc(dexpr,x)
        # x(i+1) = x(i) - f(x) / f'(x)
        arr = np.append(arr, (math.fabs(h) / math.fabs(x-h)) * 100)

        if (((math.fabs(h) / math.fabs(x-h)) * 100 )<er) :
            x=x-h
            break

        x = x - h
    print("The value of the root is : ",
          "%.4f" % x)
    curveplot(arr,i)


def secant():
    x1 = float(input("Enter the value of x1:\n"))
    x2 = float(input("Enter the value of x2:\n"))
    E= float(input("Enter the relative error percentage:"))
    n = int(input("number of iterations"))
    arr=np.array([])
    i=1
    if (func(x1) * func(x2) < 0):
        while True:
            # update number of iteration
            i += 1

            # calculate the intermediate value
            x0 = x2-func(x2)*(x1-x2)/(func(x1)-func(x2))

            condition = (((abs(x2 - x0) / (abs(x0)) * 100) < E) or (i > n))
            arr = np.append(arr, (abs(x2 - x0) / (abs(x0)) * 100))
```

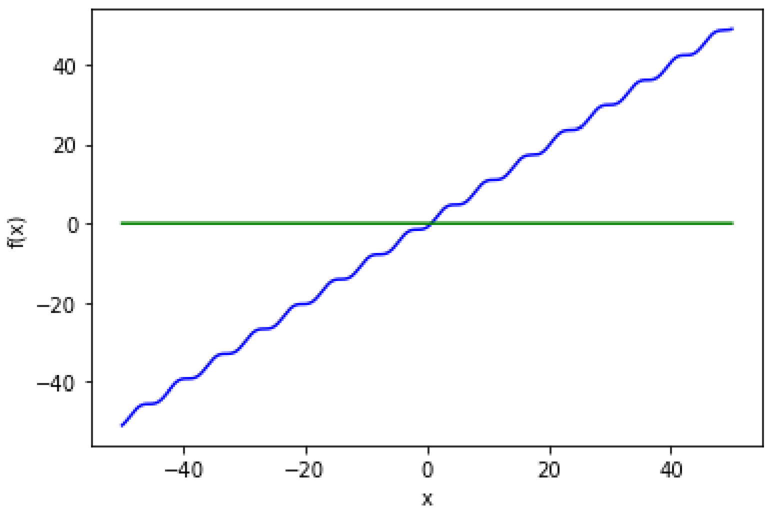
```
            if condition:
                break
            # update the value of interval
            x1 = x2
            x2 = x0


        print("Root of the given equation =",
              round(x0, 4));
    else:
        print("Can not find a root in ",
              "the given interval");
    curveplot(arr,i)
```

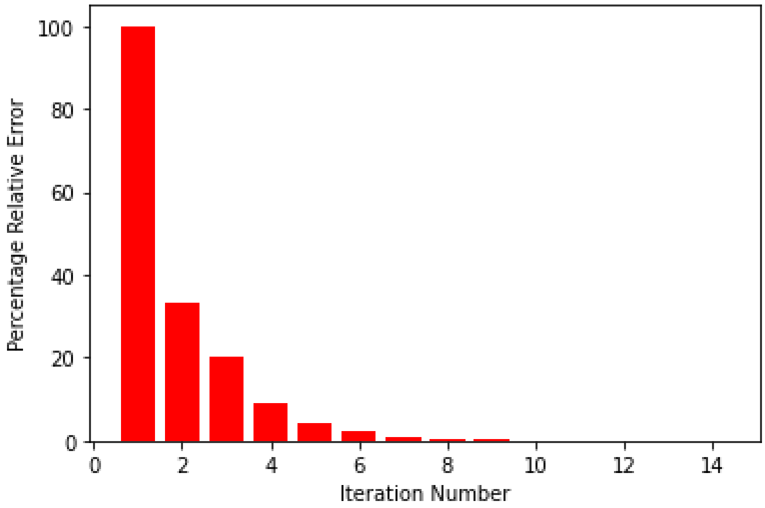
```
Enter the function(in terms of x):x-math.cos(x)
```

In [2]:

```
choice = int(input("choose type of method \n1.Bisection\n2.Regula Falsi\n3.Fixed Po

if (choice==1):
    bisection()
elif (choice==2):
    regulaFalsi()
elif(choice==3):
    fixedPointIteration()
elif(choice==4):
    newtonRaphson()
elif(choice==5):
    secant()
else:
    print("Incorrect Input")
```

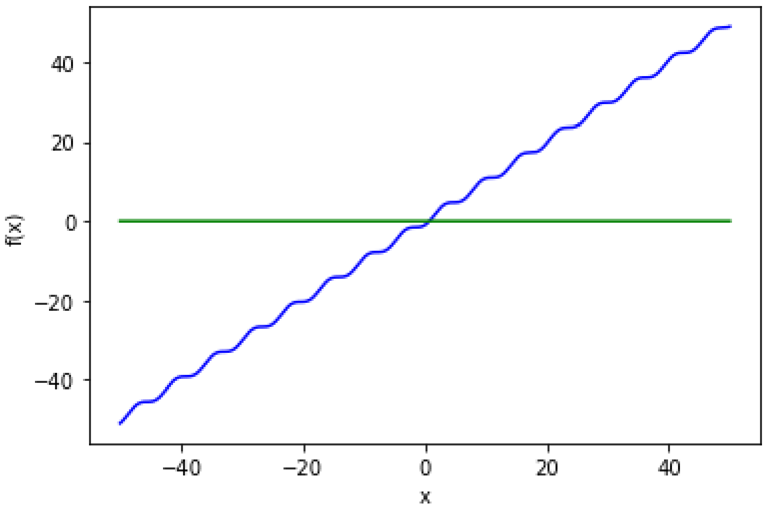
```
choose type of method
1.Bisection
2.Regula Falsi
3.Fixed Point
4.Newton Raphson
5.Secant1
Enter the value of x1:
0
Enter the value of x2:1
number of iterations50
maximum relative error as %0.01
The value of root is :  0.7391
```

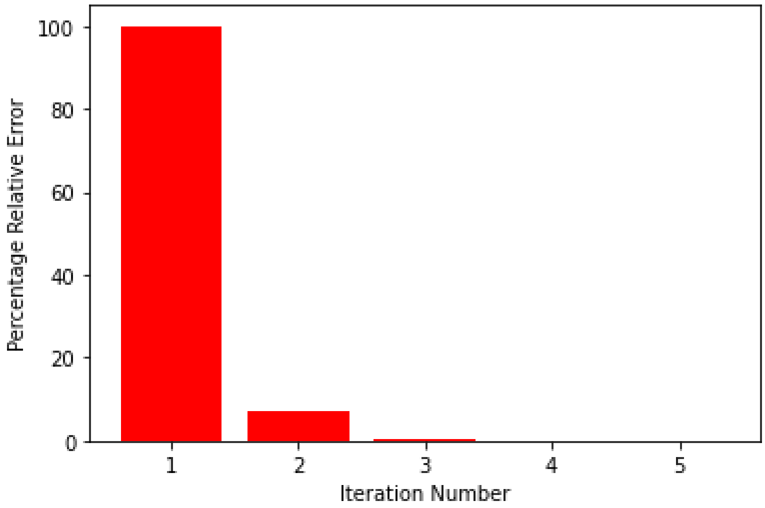




In [3]:
```
choice = int(input("choose type of method \n1.Bisection\n2.Regula Falsi\n3.Fixed Po

if (choice==1):
    bisection()
elif (choice==2):
    regulaFalsi()
elif(choice==3):
    fixedPointIteration()
elif(choice==4):
    newtonRaphson()
elif(choice==5):
    secant()
else:
    print("Incorrect Input")
```

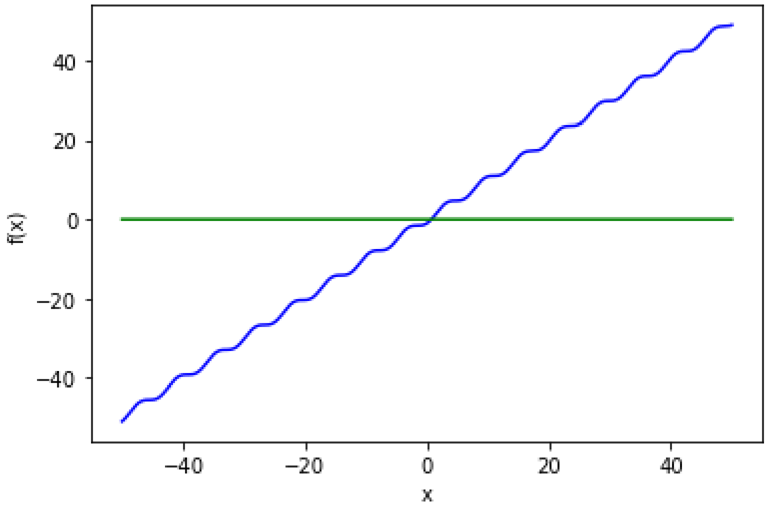
```
choose type of method
1.Bisection
2.Regula Falsi
3.Fixed Point
4.Newton Raphson
5.Secant2
Enter the value of x1:
0
Enter the value of x2:1
number of iterations50
maximum relative error as %0.01
The value of root is :  0.7391
```

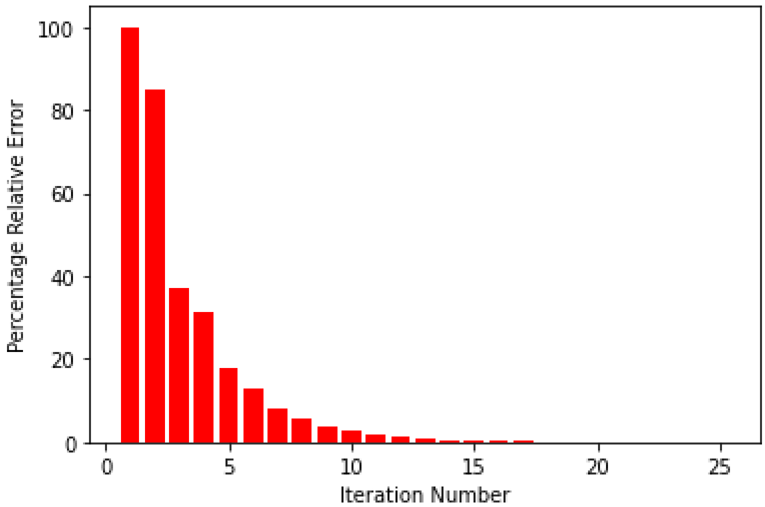




In [4]:
```
choice = int(input("choose type of method \n1.Bisection\n2.Regula Falsi\n3.Fixed Po

if (choice==1):
    bisection()
elif (choice==2):
    regulaFalsi()
elif(choice==3):
    fixedPointIteration()
elif(choice==4):
    newtonRaphson()
elif(choice==5):
    secant()
else:
    print("Incorrect Input")
```

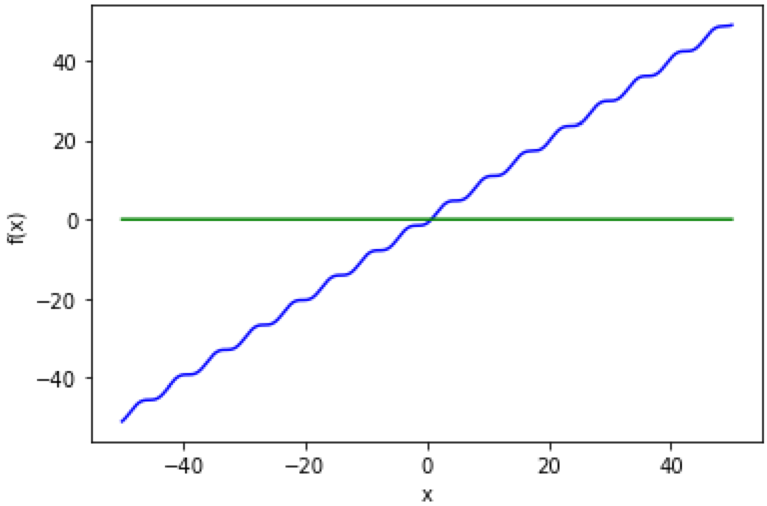
```
choose type of method
1.Bisection
2.Regula Falsi
3.Fixed Point
4.Newton Raphson
5.Secant3
Enter the function g(x) such that g(x)=x (in terms of x):math.cos(x)
Enter Guess: 0
Tolerable Error %: 0.01
Maximum Step: 50


*** FIXED POINT ITERATION ***

Required root is: 0.7391
```

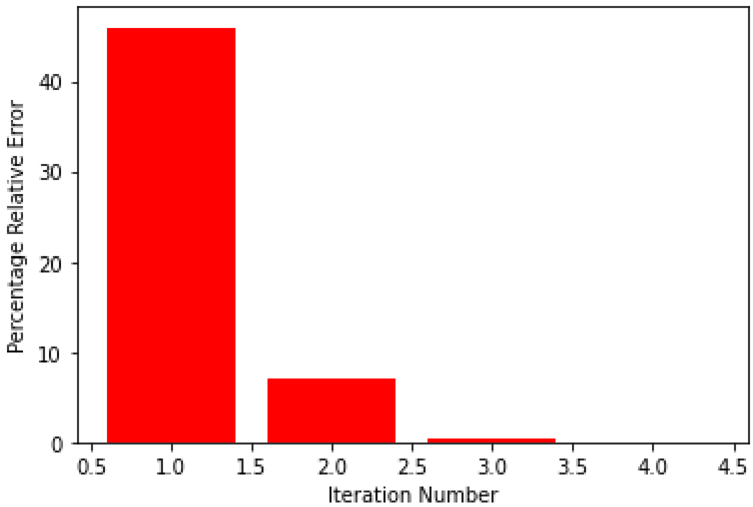




In [6]:
```
choice = int(input("choose type of method \n1.Bisection\n2.Regula Falsi\n3.Fixed Po

if (choice==1):
    bisection()
elif (choice==2):
    regulaFalsi()
elif(choice==3):
    fixedPointIteration()
elif(choice==4):
    newtonRaphson()
elif(choice==5):
    secant()
else:
    print("Incorrect Input")
```

```
choose type of method
1.Bisection
2.Regula Falsi
3.Fixed Point
4.Newton Raphson
5.Secant5
Enter the value of x1:
0
Enter the value of x2:
1
Enter the relative error percentage:0.01
number of iterations50
45.96976941318602   0.01
6.957179142390695   0.01
0.3815844156322459   0.01
0.004634078484797597   0.01
Root of the given equation = 0.7391
```

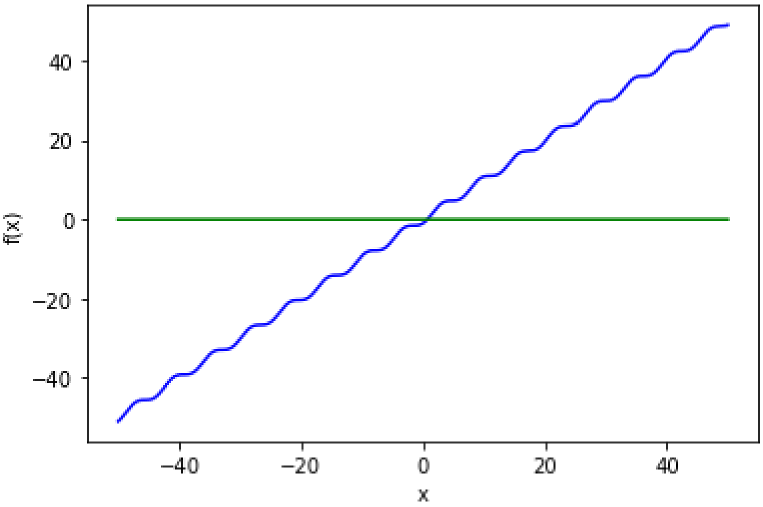


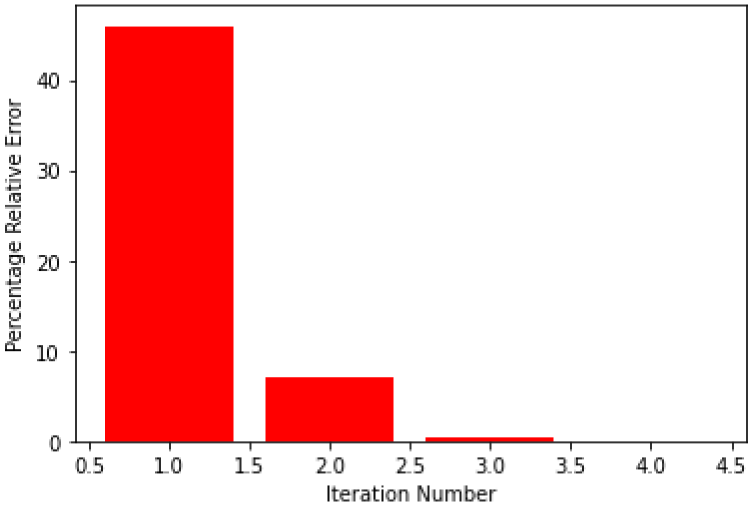


In [8]:
```
choice = int(input("choose type of method \n1.Bisection\n2.Regula Falsi\n3.Fixed Po

if (choice==1):
    bisection()
elif (choice==2):
    regulaFalsi()
elif(choice==3):
    fixedPointIteration()
elif(choice==4):
    newtonRaphson()
elif(choice==5):
    secant()
else:
```

```python
    print("Incorrect Input")
```

```
choose type of method
1.Bisection
2.Regula Falsi
3.Fixed Point
4.Newton Raphson
5.Secant5
Enter the value of x1:
0
Enter the value of x2:
1
Enter the relative error percentage:0.01
number of iterations50
Root of the given equation = 0.7391
```





In [ ]: