In [4]:

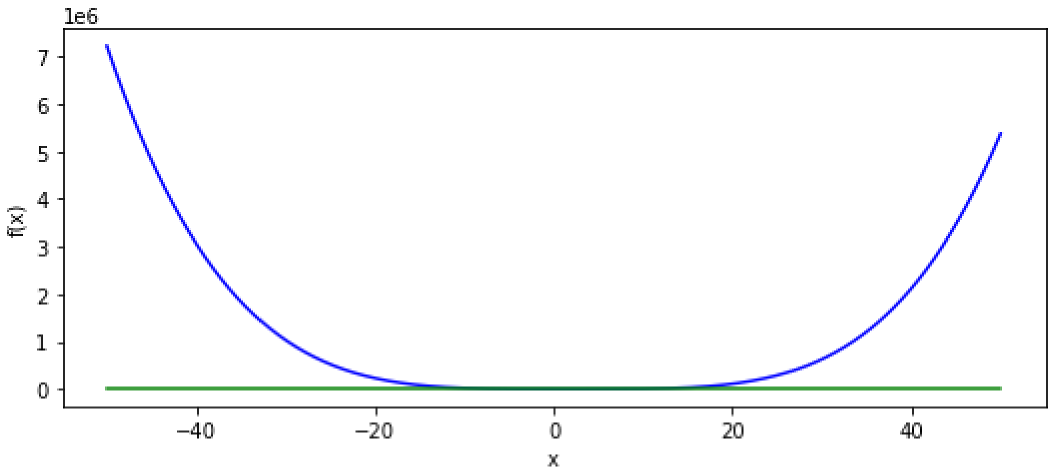
```
import numpy as np
import matplotlib.pyplot as plt
import math
from typing import *
from cmath import sqrt


expr = ""

#x=2
#print(eval(expr))




def xaxis(inp):
    return 0

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


def curveplot():
    plt.rcParams["figure.figsize"] = [7.50, 3.50]
    plt.rcParams["figure.autolayout"] = True
    plt.xlabel("x")
    plt.ylabel("f(x)")
    y = np.linspace(-50, 50, 1000)
    f2 = np.vectorize(func)
    f3 = np.vectorize(xaxis)
    z=f2(y)
    plt.plot(y,z, color="blue")
    plt.plot(y, f3(y), color="green")


    plt.show()

def quadroot(r,s):
    #print("I am here")
    dis = r**2 + 4*s
    if dis>0:
        r1=(r+ math.sqrt(dis))/2
        r2=(r- math.sqrt(dis))/2
        print("Roots :","%.4f"%r1," and ","%.4f"%r2)
    else:
        r1=r/2
        r2=r/2
        i1=math.sqrt(abs(dis))/2
        i2=-i1
        print("Roots: ", "%.4f" %r1, " + i","%.4f" %i1," and ","%.4f" % r2," + i",



def bairstow():
    coeff=[]
    b = []
    c = []
    deg=int(input("enter the degree of the polynomial"))
    i=0
    while(i<=deg):
        print("enter the coefficient a[",i,"]: ")
        coeff.append((float(input(" "))))
```

```
        b.append(0.0)
        c.append(0.0)

        i=i+1
    a=coeff
    error=float(input("enter relative percentage error"))
    N=int(input("maximum number of iterations"))
    n=deg

    i=1
    global expr
    while(i<=deg):
        expr = expr +"+"+str(coeff[i])+"*x**"+str(i)
        i+=1
    expr=str(coeff[0])+expr
    #print(expr)


    while (n>=3) :
        i=0
        rr = float(input("Enter initial guess for r"))
        ss = float(input("Enter initial guess for s"))
        r=rr
        s=ss
        condition = True
        while condition  :
                i+=1
                b[n]=a[n]
                b[n-1]=a[n-1] + r*b[n]
                c[n]=b[n]
                c[n-1]=b[n-1]+r*c[n]
                j=n-2
                while(j>=0):
                    b[j]=a[j]+r*b[j+1]+s*b[j+2]
                    c[j] = b[j] + r*c[j + 1] + s*c[j + 2]
                    j-=1
                det=c[2]*c[2]-c[3]*c[1]

                if det!=0 :
                    dr = ((-1)*b[1]*c[2] + b[0]*c[3])/det
                    ds = ((-1)*b[0]*c[2] + b[1]*c[1])/det
                    r=r+dr
                    s=s+ds
                else :
                    i=0
                    r=r+1
                    s=s+1

                if (((abs(dr/r)*100 < error) & (abs(ds/s)*100 < error)) | (i>N)) :
                    condition = False
                    b[n] = a[n]
                    b[n - 1] = a[n - 1] + r * b[n]
                    j = n - 2
                    while (j >= 0):
                        b[j] = a[j] + r * b[j + 1] + s * b[j + 2]
                        j -= 1

        n-=2
        #print(b)
        quadroot(r, s)
        j=0
        while(j<=n):
            a[j]=b[j+2]
            j+=1
```

```
        #print(a)

    if(n==2):
        r=-1*a[1]/a[2]
        s=-1*a[0]/a[2]
        quadroot(r,s)
    else:
        ro = (-1.0)*a[0]/a[1]
        print("Root: ","%.4f" %ro)

    curveplot()


Num = Union[float, complex]
Func = Callable[[Num], Num]

def div_diff(f: Func, xs: List[Num]):
    """Calculate the divided difference f[x0, x1, ...]."""
    if len(xs) == 2:
        a, b = xs
        return (f(a) - f(b)) / (a - b)
    else:
        return (div_diff(f, xs[1:]) - div_diff(f, xs[0:-1])) / (xs[-1] - xs[0])

def mullers_method(f: Func, xs: (Num, Num, Num), iterations: int,error) -> float:
    """Return the root calculated using Muller's method."""
    x0, x1, x2 = xs
    for _ in range(iterations):
        w = div_diff(f, (x2, x1)) + div_diff(f, (x2, x0)) - div_diff(f, (x2, x1))
        s_delta = sqrt(w ** 2 - 4 * f(x2) * div_diff(f, (x2, x1, x0)))
        denoms = [w + s_delta, w - s_delta]
        # Take the higher-magnitude denominator
        x3 = x2 - 2 * f(x2) / max(denoms, key=abs)
        # Advance
        if(abs((x3-x2)/x3*100)<error):
            break
        x0, x1, x2 = x1, x2, x3
    return x3

def f_example(x: Num) -> Num:
    """The example function. With a more expensive function, memoization of the las
    return func(x)



def muller():


    coeff = []
    deg = int(input("enter the degree of the polynomial"))
    i = 0
    while (i <= deg):
        print("enter the coefficient a[", i, "]: ")
        coeff.append((float(input(" "))))
        i = i + 1

    x0 = float(input("enter the first value"))
    x1= float(input("enter the second value"))
    x2= float(input("enter the third value"))

    error = float(input("enter relative percentage error"))
    N = int(input("maximum number of iterations"))

    i = 1
```

```
    global expr
    expr=""
    while (i <= deg):
        expr = expr + "+" + str(coeff[i]) + "*x**" + str(i)
        i += 1
    expr = str(coeff[0]) + expr
    print(expr)

    root = mullers_method(f_example, (x0, x1, x2), N,error)
    print("Root: {}".format(root))

    curveplot()

ch = int(input("Enter Choice\n1.Muller\n2.Bairstow"))
if ch==1:
    muller()
elif ch==2:
     bairstow()
else:
    print("Wrong Choice!")
```

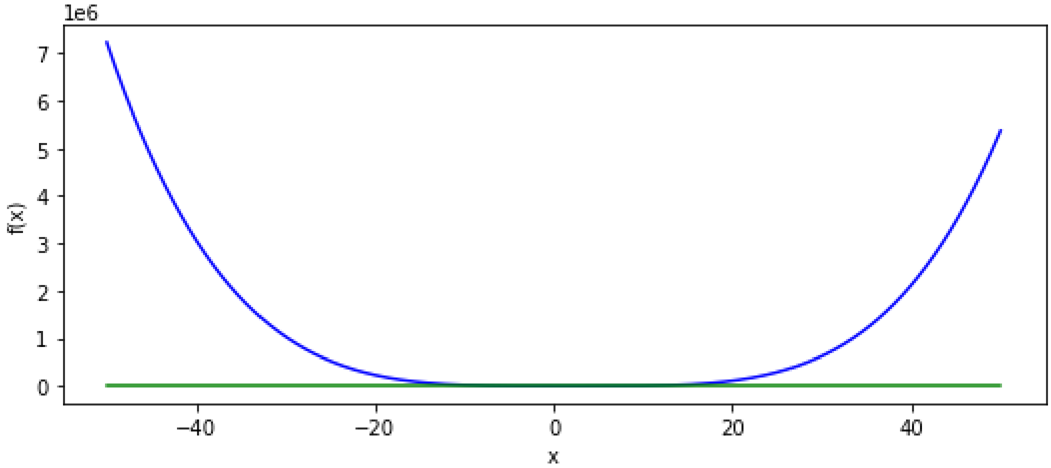
```
enter the coefficient a[ 0 ]:

enter the coefficient a[ 1 ]:

enter the coefficient a[ 2 ]:

enter the coefficient a[ 3 ]:

enter the coefficient a[ 4 ]:

9.6448+-24.184*x**1+20.44*x**2+-7.4*x**3+1.0*x**4
Root: (2.2000007199176186+0j)
```



In [5]:
```
ch = int(input("Enter Choice\n1.Muller\n2.Bairstow"))
if ch==1:
    muller()
elif ch==2:
     bairstow()
else:
    print("Wrong Choice!")
```

```
enter the coefficient a[ 0 ]:

enter the coefficient a[ 1 ]:

enter the coefficient a[ 2 ]:

enter the coefficient a[ 3 ]:
```

```
enter the coefficient a[ 4 ]:

9.6448+-24.184*x**1+20.44*x**2+-7.4*x**3+1.0*x**4
Root: (2.199999989412073+0j)
```




```
In [ ]:
```