1 More on Classes

In [1]: `from scipy import *
from pylab import *
from matplotlib.pyplot import *
%matplotlib inline

1.0.1 Multiple inheritance

Let us take a look at a couple inheretance examples:

In [6]: `class Base(object):
    
    '''With Polygon and Parallelogram inheriting from Base, you avoid a TypeError'''
    def __init__(self,*args):
        pass

class Polygon(Base):
    def __init__(self, polname, *args):
        print(polname, " is a polygon.")
        #print("Polygon class")
        super(Polygon,self).__init__(polname)
        pass

class Parallelogram(Base):
    def __init__(self, paralname, *args):
        print(paralname, " is also a parallelogram")
        #print("Parallelogram class")
        super(Parallelogram,self).__init__(paralname)
        pass

class Square(Polygon):
    def __init__(self, squarename):
        print(squarename, " is also a square.")
        super(Square,self).__init__(squarename)

class Romb(Parallelogram, Square):
    def __init__(self, rombname):
        print(rombname, " is a romb")
super(Romb, self).__init__(rombname)

class Triangle(Polygon):
    pass

p = Polygon('pol3')
print(" ")
p1 = Parallelogram('pp43')
print(" ")
s = Romb('sq3')

pol3 is a polygon.

pp43 is also a parallelogram

sq3 is a romb
sq3 is also a parallelogram
sq3 is also a square.

sq3 is a polygon.

One more example below...

In [5]: class Base(object):
    def __init__(self, x):
        print('0. Base called by {0}'.format(x))
        super().__init__()

class P1(Base):
    def __init__(self, x):
        print('1. This one is called by {0}'.format(x))
        super().__init__('1')

class P2(Base):
    def __init__(self, x):
        print('2. While this is called by {0}'.format(x))
        super().__init__('2')

class C(P2, P1):
    def __init__(self, x):
        print('3. Finally this is called by {0}'.format(x))
        super().__init__('3')

C('Example')
print(C.mro())  # Prints the order of the calling sequence from children to parents.

3. Finally this is called by Example
2. While this is called by 3
1. This one is called by 2
1.0.2 A full example of classes and inheritance

We now present a break-down for the root finding methods Bisection and Newton. Although we have solved these problems previously in class and in the training exercises we will now look at their solution by way of classes. This should help in terms of more concretely understanding how classes can be used to break apart, into sub methods, the structure of a given problem.

In [7]: class RootFinding:
    
    def __init__(self):
        pass
    
    def solve(self):
        pass
    
    def __plot__(self):
        pass

class Bisection(RootFinding):
    '''Uses $x_{(n+1)} = (x_n + x_{(n-1)})/2$ to find root in the interval $(x_n, x_{(n-1)})''' 
    pass

class Newton(RootFinding):
    pass

In [8]: class RootFinding:

    def __init__(self,f,interval,N):
        self.f = f
        self.t0, self.tf = interval[0], interval[1]
        self.N = N
        self.res = 0

    def generate(self):
        '''Iterations generated here'''
        yield self.t0
        yield self.tf

        for t in range(self.N):
            ti = self.step()
            yield ti

    def testdiv(self):
        '''Tests whether the method converges or diverges'''
        seq = self.solution
if abs(seq[-1]-seq[-2]) < abs(seq[-2]-seq[-3]):
    self.res = 1  # Converges
else:
    self.res = 0  # Diverges

def solve(self):
    '''Assembles the iterations from the generator and places that solution to self.solution'''
    self.solution = array(list(self.generate()))

def step(self):
    '''To be overridden by each individual method used later on'''
    raise NotImplementedError()

def plotsol(self):
    '''Plotting the original functions and iterations together'''
    figure(1)
    xint = self.solution
    f = self.f
    xx = linspace(min(xint), max(xint), 100)
    plot(xx, f(xx), 'b-', label='Function')
    plot(xx, 0*xx, 'k')
    plot(xint, f(xint), 'ro', label='Successive roots')
    legend()

def result(self):
    '''Some information about converges or divergence'''
    text = "This root finding method "
    if self.res == 1:
        print(text+" succeeded. Converged!")
    else:
        print(text+" failed. Diverged!")
    return ""

class Bisection(RootFinding):
    '''Uses x_{n+1} = \frac{x_n + x_{n-1}}{2} to find root in the interval \(x_n, x_{n-1}\)'''
    def step(self): 
        # overriding RootFinding step method
        f = self.f
        l = self.t0
        r = self.tf
        m = (l+r)/2

        if f(m)*f(l)<0:
            self.tf = m  # solution after one step
        else:
self.t0 = m # solution after one step
return m

def __repr__(self): # nice printing of the results
    self.testdiv()
    self.result()
    if self.res == 1:
        print("The final root found is %2.3f" %self.solution[-1])
    else:
        print("Try something else!")
    return ""

class Newton(RootFinding):
    def step(self): # overriding RootFinding step method
        x0 = self.t0
        x1 = self.tf
        f = self.f
        der = (f(x1)-f(x0))/(x1-x0) # approximating the derivative
        x = x0-der
        self.t0 = self.tf
        self.tf = x # solution after one step
        return x

def __repr__(self): # nice printing of the results
    self.testdiv()
    self.result()
    if self.res == 1:
        print("The final root found is %2.3f" %self.solution[-1])
    else:
        print("Try something else!")
    return ""

In [9]: b = Bisection(lambda x: -x**2+1, [0, 5], 10)
   r1 = b.step()

In [10]: b.solve()
   b.plotsol()
   b

This root finding method succeeded. Converged!
The final root found is 0.999

Out[10]:
In [11]: # Newton's method is notorious for not being able to produce the root of a given function for a number of reasons such as dividing by 0, or diverging etc...

# Below we encounter exactly this problem: Division by 0
n = Newton(lambda x: -x**2+1, [0,4], 5)
n.solve()

---------------------------------------------------------------------------
ZeroDivisionError Traceback (most recent call last)
<ipython-input-11-8b5cc8a38524> in <module>()
 1 n = Newton(lambda x: -x**2+1, [0,4], 5)
     
----> 2 n.solve()

<ipython-input-8-4769302c6100> in solve(self)
 27     '''Assembles the iterations from the generator and places that solution to self.solution'''
 28     self.solution = array(list(self.generate()))
 ---> 29     self.generate()
13
14    for t in range(self.N):
15        ti = self.step()
16        yield ti
17
<ipython-input-8-4769302c6100> in step(self)
 85      x1 = self.tf
 86      f = self.f
87    ---> 87    der = (f(x1)-f(x0))/(x1-x0) # approximating the derivative
 88      x = x0-der
 89      self.t0 = self.tf

ZeroDivisionError: float division by zero

In [12]: # Thus to make sure that we control the code we should at the very least use the try-except construct.
   # Naturally instead of a simple print statement we should try to catch the error...

n = Newton(lambda x: -x**2+1,[0.4,1.2],5)

try:
    n.solve()
    n.testdiv()
    n.result()
    n.plotsol()

except ZeroDivisionError:
    print("Unfortunately the Newton method did not work! Division by 0!")

This root finding method failed. Diverged!
In [ ]: