02_non_linear_equations

October 16, 2019

1 Nonlinear Equations

We want to find a root of the nonlinear function f using different methods.

1. Bisection method
2. Newton method
3. Chord method
4. Secant method
5. Fixed point iterations

[ ]:
```
%matplotlib inline
from numpy import *
from matplotlib.pyplot import *
import sympy as sym
```

[ ]:
```
t = sym.symbols('t')

f_sym = t/8. * (63.*t**4 - 70.*t**2. +15.) # Legendre polynomial of order 5

f_prime_sym = sym.diff(f_sym,t)

f = sym.lambdify(t, f_sym, 'numpy')
f_prime = sym.lambdify(t,f_prime_sym, 'numpy')

phi = lambda x : 63./70.*x**3 + 15./(70.*x)
#phi = lambda x : 70.0/15.0*x**3 - 63.0/15.0*x**5
#phi = lambda x : sqrt((63.*x**4 + 15.0)/70.)

# Let's plot
n = 1025

x = linspace(-1,1,n)
c = zeros_like(x)

_ = plot(x,f(x))
_ = plot(x,c)
_ = grid()
```

```python
[ ]: # Initial data for the variuos algorithms

# interval in which we seek the solution
a = 0.7
b = 1.

# initial points
x0 = (a+b)/2.0
x00 = b
```

```python
[ ]: # stopping criteria
eps = 1e-10
n_max = 1000
```

1.1 Bisection method

$$x^k = \frac{a^k + b^k}{2}$$

```
if (f(a_k) * f(x_k)) < 0:
   b_k1 = x_k
   a_k1 = a_k
else:
   a_k1 = x_k
   b_k1 = b_k
```

```python
[ ]: def bisect(f,a,b,eps,n_max):
    a_new = a
    b_new = b
    x = mean([a,b])
    err = eps + 1.
    errors = [err]
    it = 0
    while (err > eps and it < n_max):
        if ( f(a_new) * f(x) < 0 ):
            # root in (a_new,x)
            b_new = x
        else:
            # root in (x,b_new)
            a_new = x

        x_new = mean([a_new,b_new])

        #err = 0.5 *(b_new -a_new)
        err = abs(f(x_new))
        #err = abs(x-x_new)
```

```
        errors.append(err)
        x = x_new
        it += 1

    semilogy(errors)
    print it
    print x
    print err
    return errors

errors_bisect = bisect(f,a,b,eps,n_max)
```

[ ]:
```
# is the number of iterations coherent with the theoretical estimation?
```

In order to find out other methods for solving non-linear equations, let's compute the Taylor's series of $f(x^k)$ up to the first order

$$f(x^k) \simeq f(x^k) + (x - x^k)f'(x^k)$$

which suggests the following iterative scheme

$$x^{k+1} = x^k - \frac{f(x^k)}{f'(x^k)}$$

The following methods are obtained applying the above scheme where

$$f'(x^k) \approx q^k$$

1.2 Newton's method

$$q^k = f'(x^k)$$

$$x^{k+1} = x^k - \frac{f(x^k)}{q^k}$$

[ ]:
```
def newton(f,f_prime,x0,eps,n_max):
    pass # TODO

%time errors_newton = newton(f,f_prime,1.0,eps,n_max)
```

1.3 Chord method

$$q^k \equiv q = \frac{f(b) - f(a)}{b - a}$$

$$x^{k+1} = x^k - \frac{f(x^k)}{q}$$

[ ]:
```
def chord(f,a,b,x0,eps,n_max):
    pass # TODO

errors_chord = chord (f,a,b,x0,eps,n_max)
```

1.4 Secant method

$$q^k = \frac{f(x^k) - f(x^{k-1})}{x^k - x^{k-1}}$$

$$x^{k+1} = x^k - \frac{f(x^k)}{q^k}$$

Note that this algorithm requirs **two** initial points

[ ]:
```
def secant(f,x0,x00,eps,n_max):
    pass # TODO

errors_secant = secant(f,x0,x00,eps,n_max)
```

1.5 Fixed point iterations

$$f(x) = 0 \to x - \phi(x) = 0$$

$$x^{k+1} = \phi(x^k)$$

[ ]:
```
def fixed_point(phi,x0,eps,n_max):
    pass # TODO

errors_fixed = fixed_point(phi,0.3,eps,n_max)
```

1.6 Comparison

[ ]:
```
# plot the error convergence for the methods
loglog(errors_bisect, label='bisect')
loglog(errors_chord, label='chord')
loglog(errors_secant, label='secant')
loglog(errors_newton, label ='newton')
loglog(errors_fixed, label ='fixed')
_ = legend()
```

[ ]:
```
# Let's compare the scipy implmentation of Newton's method with our..
```

[ ]:
```
import scipy.optimize as opt
%time opt.newton(f, 1.0, f_prime, tol = eps)
```

[ ]:
```

```