QF627 Programming and Computational Finance

S0608: Scientific Tools in Python and MATLAB

(part 1)

Learning Outcomes:

1. (Python) **`scipy.optimize.fsolve(func,x0)`** returns ☐ all the roots / ☑ one root of the (non-linear) equations defined by **`func(x)=0`** given a starting estimate **`x0`**.
2. (MATLAB) **`fzero(fun,x0)`** returns ☐ all the roots / ☑ one root of a nonlinear function. **`x0`** can be specified as a real scalar or a 2-element real vector.
3. (MATLAB) **`fsolve(fun, x0)`** returns ☐ all the roots / ☑ one root of a system of nonlinear equations $F(x) = 0$, where $F(x)$ is a function that returns a vector value and x is a vector or a matrix. **`x0`** is the initial point, specified as a real vector or real array.
4. ☑ True / ☐ False To solve $x^2 = 2$,
   1) Write the equation into $f(x) = x^2 - 2 = 0$ format.
   2) Define function $f(x)$.

Python	MATLAB
Define **`f01`** with a **`lambda`** expression:	Define **`f01`** with an anonymous function:
`f01=lambda x: x**2-2`	`f01=@(x) x^2-2;`
Define **`f02`** with a user-define function:	Define **`f02`** with a user-define function:

`def f02(x):`
`    return x**2-2`
	`function y=f02(x)`
	`    y=x^2-2;`
	`end`

   3) Apply the library function.

Python	MATLAB
`fsolve with f01`	`fsolve with f01`
`x=fsolve(f01, 1)`	`x=fsolve(f01, 1)`
`fsolve with f02`	`fsolve with f02`
`x=fsolve(f02, 1)`	`x=fsolve(@f02, 1)`

5. ☑ True / ☐ False (Python) For a one-variable equation, **`fsolve`** returns the solution in a list.
6. (Python) Define **`myfzero`** (using **`fsolve`**) to solve a nonlinear equation and return the solution as a scalar.

```
from scipy.optimize import fsolve
```

```
def myfzero(func,x0):
    return fsolve(func,x0)[0]
```

7. How to obtain the negative root of the equation $x^2 = 2$?

<table>
<tr><td colspan="2">Method 1: Choose a proper initial value</td></tr>
<tr><td>Python (<code>fsolve</code> + <code>f02</code>)</td><td>MATLAB (<code>fzero/fsolve</code> + <code>f02</code>)</td></tr>
<tr><td><code>x=fsolve(f02,-1)[0]</code></td><td><code>x=fsolve(@f02,-1)</code> or
<code>x=fzero(@f02,-1)</code></td></tr>
<tr><td>Python (<code>bisect</code> + <code>f02</code>)</td><td>MATLAB (<code>fzero</code> + <code>f02</code>)</td></tr>
<tr><td><code>from scipy.optimize import bisect</code>
<code>x=bisect(f02,-10,0)</code></td><td><code>x=fzero(@f02,[-10,0])</code></td></tr>
</table>

8. ☑ True / ☐ False Both in Python and MATLAB, we can define functions that return a function/functions.

9. Python versus MATLAB

<table>
<tr><td>Python</td><td>MATLAB</td></tr>
<tr><td><code>i=3</code>
<code>f=lambda x: i*x</code>
<code>i=5</code>
<code>print(f(5))</code>
<code>i=7</code>
<code>print(f(5))</code></td><td><code>i=3;</code>
<code>f=@(x) i*x;</code>
<code>i=5</code>
<code>print(f(5))</code>
<code>i=7</code>
<code>print(f(5))</code></td></tr>
<tr><td>Output</td><td>Output</td></tr>
<tr><td><code>25</code>
<code>35</code></td><td><code>15</code>
<code>15</code></td></tr>
</table>

10. Example 2: Given $f(x,a,b) = ax^3 + b$, solve $f(x,2,3) = 0$ for x.

<table>
<tr><td>Python</td><td>MATLAB</td></tr>
<tr><td><code>def fxab(x, a, b):</code>
<code>   return a*x**3+b</code></td><td><code>function y=fxab(x, a, b)</code>
<code>   y=a*x^3+b;</code>
<code>end</code></td></tr>
<tr><td>Use <code>fsolve</code> and <code>fxab</code> with <code>args</code></td><td rowspan="2">N.A.</td></tr>
<tr><td><code>x=fsolve(fxab,1,args=(2,3))[0]</code></td></tr>
<tr><td>Python (Define a new function)</td><td>Python (Define a new function)</td></tr>
<tr><td><code>a=2</code>
<code>b=3</code>
<code>x=fsolve(lambda x: fxab(x,a,b),1)[0]</code></td><td><code>a=2;</code>
<code>b=3;</code>
<code>x=fzero(@(x) fxab(x,a,b),1)</code></td></tr>
</table>

11. How to compute $\left(-\frac{3}{2}\right)^{\frac{1}{3}}$?

Python (using `**`)	MATLAB (using `^`)
`-(3/2)**(1/3)`	`-(3/2)^(1/3)`
Python (using `mynthroot`)	**MATLAB (using `nthroot`)**

`def mynthroot(x, n):`
`    if n%2==0:`
`        return (x)**(1/n)`
`    else:`
`        if x>=0:`
`            return (x)**(1/n)`
`        else:`
`            return -(-x)**(1/n)`
`mynthroot(-3/2, 3)`
	`nthroot(-3/2,3)`

12. (Python) Implied volatility

```
from scipy.stats import norm
from math import log, sqrt, exp
def BS_EuroCall(S,K,r,q,sigma,T):
    d1=(log(S/K)+(r-q+sigma**2/2.)*T)/(sigma*sqrt(T))
    d2=d1-sigma*sqrt(T)
    c=S*exp(-q*T)*norm.cdf(d1)-K*exp(-r*T)*norm.cdf(d2)
    return c
```

Example 2: Given $S = 490$, $K = 470$, $r = 0.033$, $q = 0$, $T = 0.08$ and $c = 24.5941$, use function **`scipy.optimize.fsolve`** / **`myfzero`** and **`BS_EuroCall`** to solve the Black-Scholes equation for the implied volatility (σ).

```
S=490
K=470
r=0.033
q=0
T=0.08
c=24.5941
sigma_imp=fsolve(lambda x: BS_EuroCall(S,K,r,q,x,T)-c,1)
print(sigma_imp)
```

```
sigma_imp=myfzero(lambda x: BS_EuroCall(S,K,r,q,x,T)-c,1)
print(sigma_imp)
```

13. (MATLAB) Implied volatility

BS_EuroCall.m

```
function c=BS_EuroCall(S,K,r,q,sigma,T)
d1=(log(S/K)+(r-q+sigma^2/2)*T)/(sigma*sqrt(T));
d2=d1-sigma*sqrt(T);
c=S*exp(-q*T)*normcdf(d1)-K*exp(-r*T)*normcdf(d2);
end
```

Example 2: Given $S = 490$, $K = 470$, $r = 0.033$, $q = 0$, $T = 0.08$ and $c = 24.5941$, use function **fzero** to solve the Black-Scholes equation for the implied volatility (σ).

```
S=490;
K=470;
r=0.033;
q=0;
T=0.08;
c=24.5941;
sigma_imp=fzero(@(x) BS_EuroCall(S,K,r,q,x,T)-c,1)
```

14. (Python) **mypartial**

```
def mypartial(fun,*args,**kwargs):
   def f(x):
      return fun(*args, x, **kwargs)
   return f
S=490
K=470
r=0.033
q=0
T=0.08
BS3=mypartial(BS_EuroCall, S,K,r,q,x,T=T)
c=BS3(0.2)
sigma_imp=fsolve(lambda x: BS3(x)-c, 0.5)
```

15. (MATLAB) Example 4: Compute implied volatility for every row in **data**.

1. Use **readtable** to get the data from file **dataset01.csv**. Name the data imported as **data**.

```
data=readtable('dataset01.csv');
disp(data)
```

2. Compute implied volatility for each row, and put the result in a new column with the column name **ImpVol**.

```
data(:,'ImpVol')=rowfun(@(S,K,r,q,T,c) fzero(@(x) ...
BS_EuroCall(S,K,r,q,x,T)-c,0.5),...
data(:,{'S','K','r','q','T','c'})...
'OutputVariableNames','ImpVol');
```

3. Write **data** to a CSV file **output.csv**.

```
writetable(data,'output.csv');
```

16. (Python) Compute implied volatility for every row in **`data`**.

```
import pandas as pd
from scipy.optimize import fsolve
```

1. Load data from **`dataset01.csv`** and name the data imported as **`data`**.
`data=pd.read_csv('dataset01.csv');`
2. Compute implied volatility for each row, and put the result in a new column with the column name **`ImpVol`**.
`data['ImpVol'] = data[['S', 'K', 'r', 'q', 'T', 'c']].apply( lambda x: fsolve(lambda s: BS_EuroCall(*(x[0:4]),s,x[4])-x[5], 0.5)[0], axis=1)`
3. Write **`data`** to a CSV file **`output.csv`**.
`data.to_csv('output.csv');`

17. ☑ True / ☐ False To solve the following system of nonlinear equations:

$$\begin{cases} x_0\cos(x_1) = 4 \\ x_0 x_1 - x_1 = 5 \end{cases}$$

1) Write the equation into a new format.

$$\begin{cases} x_0\cos(x_1) - 4 = 0 \\ x_0 x_1 - x_1 - 5 = 0 \end{cases}$$

2) Define function $f(x)$.

Python	MATLAB
Define **`f03`** with a user-define function:	Define **`f03`** with a user-define function:

`from math import cos`
`def f03(x):`
`    y=[0, 0]`
`    y[0] = x[0]*cos(x[1]) - 4`
`    y[1] = x[0]*x[1] - x[1] - 5`
`    return y`
	`function y=f03(x)`
	`    y(1)=x(1)*cos(x(2))-4;`
	`    y(2)=x(1)*x(2)-x(2)-5;`
	
	`end`

3) Apply the library function.

Python	MATLAB
`fsolve with f03`	**`fsolve with f03`**
`x=fsolve(f03, [1,1])`	`x=fsolve(@f03, [1,1])`

18. (Python) **`scipy.optimize.minimize`** is for the minimization of scalar function of one or more variables. In general, the optimization problems are of the form:

```
minimize f(x) subject to
g_i(x) >= 0, i = 1,…,m
h_j(x) = 0, j = 1,…,p
```

19. (MATLAB) **`fmincon`** finds minimum of constrained nonlinear multivariable function. The problem is specified by

```
min f(x) subject to
c(x) <= 0
ceq(x) = 0
A*x <= b
Aeq*x = beq
lb<=x<=ub
```

20. Example 6: Optimization with constraints

$$
\begin{aligned}
& \min_x \; q(x) = (x_1 - 1)^2 + (x_2 - 2.5)^2 \\
\text{subject to} \quad & x_1 - 2x_2 + 2 \geq 0, \\
& -x_1 - 2x_2 + 6 \geq 0, \\
& -x_1 + 2x_2 + 2 \geq 0, \\
& x_1 \geq 0, \\
& x_2 \geq 0.
\end{aligned}
$$

Python

```python
from scipy.optimize import minimize
fun = lambda x: (x[0] - 1)**2 + (x[1] - 2.5)**2
cons = ({'type': 'ineq', 'fun': lambda x: x[0] - 2 * x[1] + 2},
        {'type': 'ineq', 'fun': lambda x: -x[0] - 2 * x[1] + 6},
        {'type': 'ineq', 'fun': lambda x: -x[0] + 2 * x[1] + 2})
bnds = ((0, None), (0, None))
res = minimize(fun, (2, 0), bounds=bnds, constraints=cons)
print(res.x)
```

MATLAB

```matlab
fun=@(x) (x(1)-1)^2+(x(2)-2.5)^2;
A=[-1 2;1 2;1 -2];
b=[2;6;2];
Aeq=[];
beq=[];
lb=[0 0];
ub=[];
x0=[0 0];
x= fmincon(fun,x0,A,b,Aeq,beq,lb,ub)
```

21. (MATLAB) **quadprog** solves quadratic objective functions with linear constraints. It finds a minimum for a problem specified by

```
min (1/2)*(x')*H*x + (f')*x subject to
A*x <= b
Aeq*x = beq
lb<=x<=ub
```

22. Example 7: Quadratic Programming with linear constraints

$$f(x) = \frac{1}{2}x_1^2 + x_2^2 - x_1x_2 - 2x_1 - 6x_2$$

subject to the constraints

$$x_1 + x_2 \le 2$$
$$-x_1 + 2x_2 \le 2$$
$$2x_1 + x_2 \le 3.$$

Python

```
from scipy.optimize import minimize
fun = lambda x: 0.5*x[0]**2+x[1]**2-x[0]*x[1]-2*x[0]-6*x
cons = ({'type': ''ineq'', 'fun': lambda x: 2-x[0]-x[1]},
        {'type': ''ineq'', 'fun': lambda x: 2+x[0]-2*x[1]},
        {'type': ''ineq'', 'fun': lambda x: 3-2*x[0]-x[1]})
bnds = ((None, None), (None, None))
res = minimize(fun, (0, 0), bounds=bnds, constraints=cons)
print(res.x)
```

MATLAB (using quadprog)

```
H=[1 -1;-1 2];
f=[-2; -6];
A=[1 1; -1 2; 2 1];
b=[2;2;3];
x=quadprog(H,f,A,b)
```

23. Example 8: Minimum-Variance Portfolio

$$\min_{w} \; w\Sigma w'$$
$$wR' = R_0$$
$$w_i \ge 0$$
$$\sum w_i = 1$$

$w = [w_1, w_2, w_3, w_4]$

$R = [0.07, 0.08, 0.09, 0.10], R_0 = 0.077$

$$\Sigma = \begin{pmatrix} 0.0225 & 0.009 & 0.013125 & 0.01125 \\ 0.009 & 0.04 & 0.019 & 0.006 \\ 0.013125 & 0.019 & 0.0625 & 0.01125 \\ 0.01125 & 0.006 & 0.01125 & 0.09 \end{pmatrix}$$

Python (using `np.matrix`)

```
from scipy.optimize import minimize
import numpy as np
R=np.matrix([0.07,0.08,0.09,0.1])
x0=[0.1,0.1,0.1,0.1]
#x0=np.array([0.1,0.1,0.1,0.1])
#x0=np.array([[0.1,0.1,0.1,0.1]])
R0=0.077
sigma=np.matrix([[0.0225, 0.009, 0.013125, 0.01125],
                 [0.009, 0.04, 0.019, 0.006],
                 [0.013125, 0.019, 0.0625, 0.01125],
                 [0.01125, 0.006, 0.01125, 0.09]])
def fun(w, sigma):
    Mw=np.matrix(w)
    return (Mw*sigma*(Mw.T))[0,0]
cons = ({'type': 'eq', 'fun': lambda x: (np.matrix(x)*(R.T))[0,0]-R0},
        {'type': 'eq', 'fun': lambda x: np.sum(x)-1.0})
bnds = ((0, 1), )*4
res = minimize(fun, x0, args=sigma, bounds=bnds, constraints=cons)
res.x
```

Python (using `np.array` and a 2D array for `R`)

```
from scipy.optimize import minimize
import numpy as np
R=np.array([[0.07,0.08,0.09,0.1]])
x0=[0.1,0.1,0.1,0.1]
#x0=np.array([0.1,0.1,0.1,0.1])
#x0=np.array([[0.1,0.1,0.1,0.1]])
R0=0.077
sigma=np.array([[0.0225, 0.009, 0.013125, 0.01125],
                 [0.009, 0.04, 0.019, 0.006],
                 [0.013125, 0.019, 0.0625, 0.01125],
                 [0.01125, 0.006, 0.01125, 0.09]])
def fun(w, sigma):
    return (w@sigma@(w.T))
cons = ({'type': 'eq', 'fun': lambda x: (x@(R.T))[0]-R0},
        {'type': 'eq', 'fun': lambda x: np.sum(x)-1.0})
bnds = ((0, 1), )*4
res = minimize(fun, x0, args=sigma, bounds=bnds, constraints=cons)
res.x
```

Python (using `np.array` and a 1D array for `R`)

```
from scipy.optimize import minimize
import numpy as np
R=np.array([0.07,0.08,0.09,0.1])
x0=[0.1,0.1,0.1,0.1]
#x0=np.array([0.1,0.1,0.1,0.1])
#x0=np.array([[0.1,0.1,0.1,0.1]])
R0=0.077
sigma=np.array([[0.0225, 0.009, 0.013125, 0.01125],
                 [0.009, 0.04, 0.019, 0.006],
                 [0.013125, 0.019, 0.0625, 0.01125],
                 [0.01125, 0.006, 0.01125, 0.09]])
def fun(w, sigma):
    return (w@sigma@(w.T))

cons = ({'type': 'eq', 'fun': lambda x: (x@(R.T))-R0},
        {'type': 'eq', 'fun': lambda x: np.sum(x)-1.0})
bnds = ((0, 1), )*4

res = minimize(fun, x0, args=sigma, bounds=bnds, constraints=cons)
res.x
```

24. `(to be continued)`