QF627 Programming and Computational Finance

S0608: Scientific Tools in Python and MATLAB

(part 2)

24. ☑ True / ☐ False (Python) **scipy.misc.derivative(func,x0,dx,n)** computes the **n**th derivative of **func** at **x0** with spacing **dx**. When **n=1**, it computes
(func(x0+dx)-func(x0-dx))/(2*dx)

25. ☑ True / ☐ False (MATLAB) diff can be used to approximate derivatives with the syntax **diff(f)/h**.

26. (Python) Use **scipy.misc.derivative** and **BS_EuroCallV** to approximate Delta $\Delta = \frac{\partial V}{\partial S}$ and Vega $\mathcal{V} = \frac{\partial V}{\partial \sigma}$ for every row in **data**.

```
import pandas as pd
from scipy.stats import norm
from scipy.misc import derivative
from math import log, sqrt, exp
def BS_EuroCallV(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
data=pd.read_csv('dataset01.csv',header=0)
```

	S	K	r	q	sigma	T	c
0	1403	1350	0.0534	0.0118	0.260	0.102778	80.828
1	1403	1375	0.0534	0.0118	0.267	0.102778	66.084
2	1403	1400	0.0534	0.0118	0.231	0.102778	45.894
3	1403	1425	0.0534	0.0118	0.213	0.102778	30.955
4	1403	1450	0.0534	0.0118	0.198	0.102778	19.224

Use one command to apply **scipy.misc.derivative** and **BS_EuroCallV** to each row of **data** to compute Delta $\Delta = \frac{\partial V}{\partial S}$ and save the result to a new column in **data** and name this column **Delta**.

```
data['Delta']=data[['S','K','r','q','sigma','T']].apply(
    lambda x: derivative(lambda s: BS_EuroCallV(s, *(x[1:])),x[0],dx=0.01), axis=1)
```

Use one command to apply **scipy.misc.derivative** and **BS_EuroCallV** to each row of **data** to compute Vega $\mathcal{V} = \frac{\partial V}{\partial \sigma}$ and save the result to a new column in **data** and name this column **Vega**.

```
data['Vega']=data[['S','K','r','q','sigma','T']].apply(
    lambda x: derivative(lambda s: BS_EuroCallV(*(x[0:4]), s, x[5]),x[4],dx=0.01), axis=1)
```

27. (Python) **scipy.integrate.quad(fun, a, b)** computes the definite integral $\int_a^b f(x)dx$. For example, to compute $\int_0^4 x^2 dx$, we can use

```
from scipy import integrate
integrate.quad(lambda x: x**2, 0, 4)
```

28. (MATLAB) **integral(fun, xmin, xmax)** numerically integrates function **fun** from **xmin** to **xmax** using global adaptive quadrature and default error tolerances. For example, to compute $\int_0^4 x^2 dx$, we can use

```
integral(@(x): x.^2, 0, 4)
```

29. Compute $\int_0^\infty e^{-x} dx$.

Python

```
from scipy import integrate
import numpy as np
func=lambda x: np.exp(-x)
integrate.quad(func, 0, np.inf)
```

MATLAB

```
fun=@(x): exp(-x);
integral(fun, 0, inf)
```

30. In Python, **scipy.integrate.dblquad(fun, a, b, gfun, hfun)** computes the definite integral $\int_a^b \int_{g(x)}^{h(x)} f(x,y) dy\, dx$. In MATLAB, **integral2(fun, xmin, xmax, ymin, ymax)** approximates the integral of the function **z=fun(x,y)** over the planar region **xmin≤x≤xmax** and **ymin(x)≤y≤ymax(x)**. For example, to compute $\int_0^2 \int_0^1 xy^2 dydx$, we can use

Python

```
from scipy import integrate
func=lambda x, y: x*y**2
integrate.dblquad(func, 0, 1, lambda y : 0, lambda y : 2)
```

MATLAB

```
fun=@(x,y): x.*(y.^2);
integral(fun, 0, 2, 0, 1)
```

31. ☑ True / ☐ False (Python) **scipy.interpolate.interp1d** is a class which produces callable instances. The **__init__** method has compulsory arguments **x** and **y** and the default interpolation method is the **linear** interpolation. To use cubic spline interpolation, use **kind='cubic'**. **x** and **y** are two of the instance attributes.

32. (Python) Define **`class myinterp1d`**.

```
import matplotlib.pyplot as plt
import numpy as np
class myinterp1d(object):
   def __init__(self, x, y):
      self.x=x
      self.y=y
   def __call__(self, xnew):
      ynew=[]
      for x0 in xnew:
         if x0<=self.x[0]:
            ynew.append(self.y[0])
         elif x0>=self.x[-1]:
            ynew.append(y[-1])
         else:
            hi=next(filter(lambda x: x0<x[1], enumerate(self.x)))[0]
            lo=hi-1
            m=(self.y[hi]-self.y[lo])/(self.x[hi]-sel)
            ynew.append(self.y[lo]+m*(x0-self.x[lo]))
      ynew=np.array(ynew)
      return ynew
x = np.arange(0, 10)
y = np.exp(-x/3.0)
f = myinterp1d(x, y)
xnew = np.arange(0, 9, 0.1)
ynew = f(xnew)
plt.plot(x, y, 'o', xnew, ynew, '-')
plt.show()
```

33. Fix the bug.

```
import numpy as np
import matplotlib.pyplot as plt
class myinterp1d(object):
    def __init__(self, x, y):
        self.x=x
        self.y=y
        self.nInt=len(x)-1  #number of intervals
        self.f=self._linear()

    def _linear(self):
        f=[]
        for i in range(self.nInt):
            m=(y[i+1]-y[i])/(x[i+1]-x[i])
            b=y[i]-m*x[i]
            #f.append(lambda x: m*x+b)
            f.append(lambda x, m=m, b=b: m*x+b)
        return f

    def _linear_interp(self, x0):
        if x0<=self.x[0]:
            return y[0]
        elif x0>=self.x[-1]:
            return y[-1]
        else:
            i=next(filter(lambda s: x0<s[1], enumerate(self.x)))[0]
            return self.f[i-1](x0)

    def __call__(self, xnew):
        return np.array([self._linear_interp(x0) for x0 in xnew])

x = np.arange(0, 10)
y = np.exp(-x/3.0)
f = myinterp1d(x, y)
xnew = np.arange(0, 9, 0.1)
ynew = f(xnew)   # use interpolation function returned by `interp1d`
plt.plot(x, y, 'o', xnew, ynew, '-')
plt.show()
```

34. ☑ True / ☐ False (Python) **`numpy.polyfit(x,y,deg)`** fits a polynomial of degree **`deg`** to points (**`x`**, **`y`**). It returns a vector of coefficients **`p`**. **`numpy.polyval(p,x)`** evaluates a polynomial at **`x`**. When **`deg`** is greater than **`len(x)-1`**, the polynomial can be used as an interpolation function.
35. ☑ True / ☐ False (MATLAB) **`interp1(x,v,xq,method)`** returns interpolated values of a 1-D function at specific query points **`xq`** using **`method`**. Vector **`x`** contains the sample points, and **`v`** contains the corresponding values. When **`method`** is **`spline`**, cubic spline interpolation with not-a-knot end conditions is used.
36. ☑ True / ☐ False (MATLAB) **`polyfit(x,y,n)`** and **`polyval(p,x)`** are similar to the Python functions **`numpy.polyfit`** and **`numpy.polyval`**.