Derivatives Analytics with `Python` & `Numpy`

Dr. Yves J. Hilpisch

24 June 2011

EuroPython 2011

CV Yves Hilpisch

1. **1993–1996** Dipl.-Kfm. ("MBA") at Saarland University (Banks and Financial Markets)
2. **1996–2000** Dr.rer.pol. ("Ph.D.") at Saarland University (Mathematical Finance)
3. **1997–2004** Management Consultant Financial Services & Insurance Industry
4. **2005–present** Founder and MD of Visixion GmbH
   - management and technical consulting work
   - DEXISION—Derivatives Analytics on Demand (`www.dexision.com`)
5. **2010–present** Lecturer Saarland University
   - Course: "Numerical Methods for the Market-Based Valuation of Options"
   - Book Project: "Market-Based Valuation of Equity Derivatives—From Theory to Implementation in `Python`"

What is Derivatives Anlytics about?

- Derivatives Analytics is concerned with the valuation, hedging and risk management of derivative financial instruments
- In contrast to ordinary financial instruments which may have an intrinsic value (like the stock of a company), derivative instruments derive their values from other instruments
- Tyical tasks in this context are
  - simulation
  - data analysis (historical, current, simulated data)
  - discounting
  - arithmetic operations (summing, averaging, etc.)
  - linear algebra (vector and matrix operations, regression)
  - solving optimization problems
  - visualization
  - ...
- `Python` can do all this quite well—but `C`, `C++`, `C#`, `Matlab`, `VBA`, `JAVA` and other languages still dominate the financial services industry

Why Python for Derivatives Analytics?

1. **Open Source**: Python and the majority of available libraries are completely open source
2. **Syntax**: Python programming is easy to learn, the code is quite compact and in general highly readable (= fast development + easy maintenance)
3. **Multi-Paradigm**: Python is as good at functional programming as well as at object oriented programming
4. **Interpreted**: Python is an interpreted language which makes rapid prototyping and development in general a bit more convenient
5. **Libraries**: nowadays, there is a wealth of powerful libraries available and the supply grows steadily; there is hardly a problem which cannot be easily attacked with an existing library
6. **Speed**: a common prejudice with regard to interpreted languages—compared to compiled ones like C++ or C—is the slow speed of code execution; however, financial applications are more or less all about matrix/array manipulations and other operations which can be done at the speed of C code with the essential library Numpy

What does the financial market say about Python?

- in the London area (mainly financial services) the number of Python developer contract offerings evolved as follows (respectively for the three months period ending on 22 April)
  - **142** in year 2009
  - **245** in year 2010
  - **644** in year 2011
- these figures imply a more than fourfold demand for the Python skill in 2011 as compared to 2009
- over the same period, the average daily rate for contract work increased from 400 GBP to 475 GBP[1]
- obviously, Python is catching up at a rapid pace in the financial services industry ...

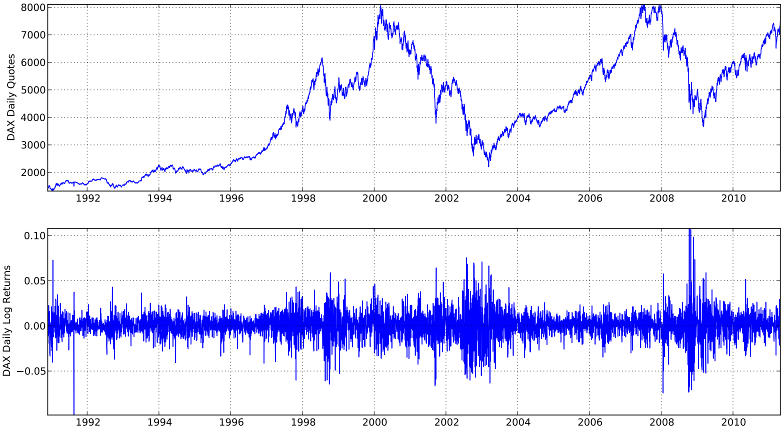
[1] Source: all figures from www.itjobswatch.co.uk/contracts/london/python.do on 24 April 2011.

In Derivatives Analytics you have to analyze different types of data

- Fundamental types of data to be analyzed
  - time series
  - cross sections
- Python libraries suited to analyze and visualize such data
  - `xlrd` (`www.python-excel.org`): reading data from Excel files
  - `Numpy` (`numpy.scipy.org`): array manipulations of any kind
  - `Pandas` (`code.google.com/p/pandas`): time series analysis, cross-sectional data analysis[2]
  - `matplotlib` (`matplotlib.sourceforge.net`): 2d and 3d plotting

[2] Notably, this library was developed by a hedge fund.

DAX time series—index level and daily log returns[3]



[3] Source: `http://finance.yahoo.com`, 29 Apr 2011

Reading data from Excel file ...

```
from xlrd import open_workbook
from pandas import *
from datetime import *
from matplotlib.pyplot import *
from numpy import *

# DAX Open Workbook, Read
xls = open_workbook('DAX_Daily_1990_2011.xls')
for s in xls.sheets():
    datesDAX = []; quoteDAX = []
    for row in range(s.nrows-1,0,-1):
        year  = int(s.cell(row,0).value)
        month = int(s.cell(row,1).value)
        day   = int(s.cell(row,2).value)
        datesDAX.append(date(year,month,day))
        quoteDAX.append(float(s.cell(row,8).value))
    print
```

... and plotting it with Pandas and matplotlib

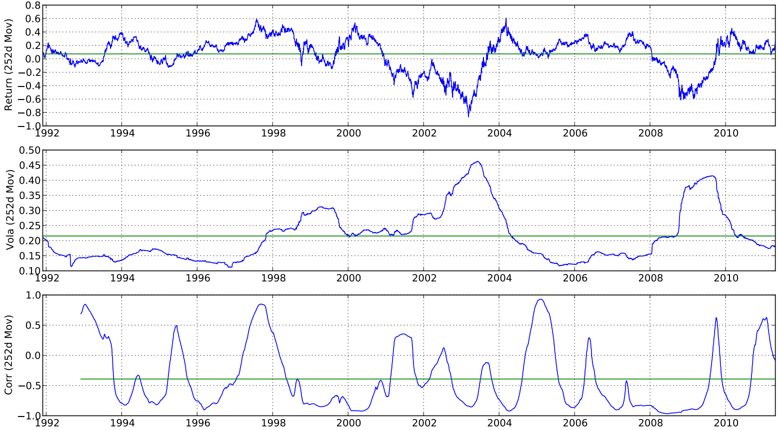
```
DAXq = Series(quoteDAX,index=datesDAX)
DAXr = Series(log(DAXq/DAXq.shift(1)),index=datesDAX)
DAXr = where(isnull(DAXr),0.0,DAXr)

# Data Frames for Quotes and Returns
data = {'QUO':DAXq,'RET':DAXr,'RVO':rv}
DAX = DataFrame(data,index=DAXq.index)

# Graphical Output

figure()
subplot(211)
plot(DAX.index,DAX['QUO'])
ylabel('DAX Daily Quotes')
grid(True);axis('tight')
subplot(212)
plot(DAX.index,DAX['RET'])
ylabel('DAX Daily Log Returns')
grid(True);axis('tight')
```

DAX time series—252 moving mean return, volatility and correlation between both[4]



[4] Source: `http://finance.yahoo.com`, 29 Apr 2011

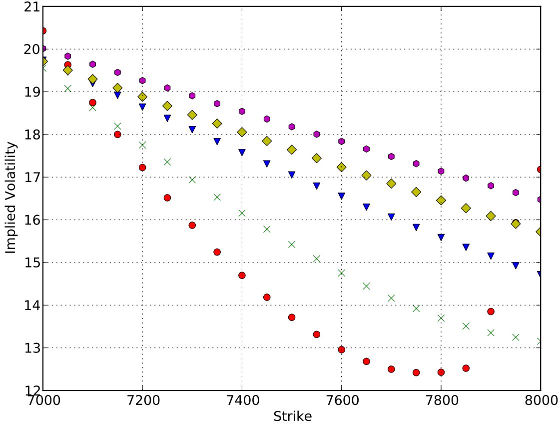
Pandas provides a number of really convenient functions

```
# mean return, volatility and correlation (252 days moving = 1 year)
figure()
subplot(311)
mr252 = Series(rolling_mean(DAX['RET'],252)*252,index=DAX.index)
mr252.plot();grid(True);ylabel('Return (252d Mov)')
x,y=REG(mr252,0);plot(x,y)

subplot(312)
vo252 = Series(rolling_std(DAX['RET'],252)*sqrt(252),index=DAX.index)
vo252.plot();grid(True);ylabel('Vola (252d Mov)')
x,y=REG(vo252,0);plot(x,y);vx=axis()

subplot(313)
co252 = Series(rolling_corr(mr252,vo252,252),index=DAX.index)
co252.plot();grid(True);ylabel('Corr (252d Mov)')
x,y=REG(co252,0);plot(x,y);cx=axis()
axis([vx[0],vx[1],cx[2],cx[3]])
```

DAX cross-sectional data—implied volatility surface[5]



maturities: 21 (red dots), 49 (green crosses), 140 (blue triangles), 231 (yellow stones) and 322 days (purple hectagons)

[5] Source: `http://www.eurexchange.com`, 29 Apr 2011

Model economy—Black-Scholes-Merton continuous time

- economy with final date $T, 0 < T < \infty$
- uncertainty is represented by a filtered probability space $\{\Omega, \mathcal{F}, \mathbb{F}, P\}$
- for $0 \leq t \leq T$ the risk-neutral index dynamics are given by the SDE

$$\frac{dS_t}{S_t} = rdt + \sigma dZ_t \tag{1}$$

- S_t index level at date t, r constant risk-less short rate, σ constant volatility of the index and Z_t standard Brownian motion
- the process S generates the filtration $\mathbb{F}$, i.e. $\mathcal{F}_t \equiv \mathcal{F}(S_{0 \leq s \leq t})$
- a risk-less zero-coupon bond satisfies the DE

$$\frac{dB_t}{B_t} = rdt \tag{2}$$

- the time t value of a zero-coupon bond paying one unit of currency at T with $0 \leq t < T$ is $B_t(T) = e^{-r(T-t)}$

Model economy—Black-Scholes-Merton discrete time

- to simulate the financial model, i.e. to generate numerical values for S_t, the SDE (1) has to be discretized
- to this end, divide the given time interval $[0, T]$ in equidistant sub-intervals Δt such that now $t \in \{0, \Delta t, 2\Delta t, ..., T\}$, i.e. there are $M + 1$ points in time with $M \equiv T/\Delta t$
- a discrete version of the continuous time market model (1)–(2) is

$$\frac{S_t}{S_{t-\Delta t}} = e^{\left(r - \frac{\sigma^2}{2}\right)\Delta t + \sigma\sqrt{\Delta t} z_t} \quad (3)$$

$$\frac{B_t}{B_{t-\Delta t}} = e^{r\Delta t} \quad (4)$$

  for $t \in \{\Delta t, ..., T\}$ and standard normally distributed z_t
- this scheme is an Euler discretization which is known to be exact for the geometric Brownian motion (1)

Option valuation by simulation—European options

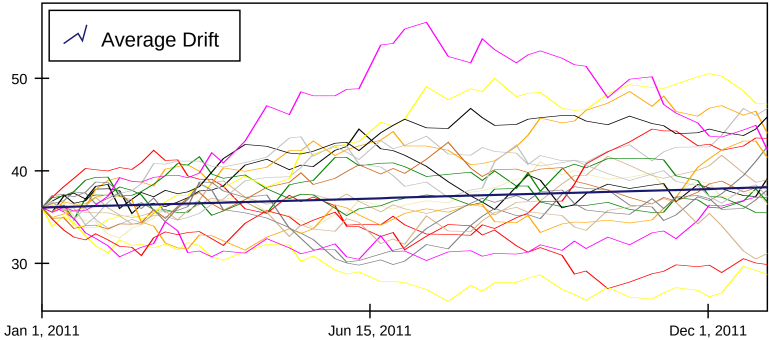
- a European put option on the index S pays at maturity T

$$h(S_T) \equiv \max[K - S_T, 0]$$

  for a fixed strike price K
- to value such an option, simulate I paths of S_t such that you get I values $S_{T,i}, i \in \{1, ..., I\}$
- the Monte Carlo estimator for the put option value then is

$$V_0 = e^{-rT} \frac{1}{I} \sum_{i=1}^{I} h(S_{T,i})$$

Simulating the index level for European option valuation[6]



20 simulated index level paths; thick blue line = average drift

[6] Source: `analytics.dexision.com`

Numpy offers all you need for an efficient implementation (I)

```
#
# Valuation of European Put Option
# by Monte Carlo Simulation
#
from numpy import *
from numpy.random import standard_normal,seed
from time import time
t0=time()

## Parameters -- American Put Option
S0 = 36.          # initial stock level
K  = 40.          # strike price
T  = 1.0          # time-to-maturity
vol= 0.2          # volatility
r  = 0.06         # short rate
## Simulation Parameters
seed(150000)      # seed for Python RNG
M  = 50           # time steps
I  = 50000        # simulation paths
dt = T/M          # length of time interval
df = exp(-r*dt)   # discount factor per time interval
```

Numpy offers all you need for an efficient implementation (II)

```
## Index Level Path Generation
S=zeros((M+1,I),'d')            # index value matrix
S[0,:]=S0                       # initial values
for t in range(1,M+1,1):        # stock price paths
    ran=standard_normal(I)      # pseudo-random numbers
    S[t,:]=S[t-1,:]*exp((r-vol**2/2)*dt+vol*ran*sqrt(dt))

## Valuation
h=maximum(K-S[-1],0)            # inner values at maturity
V0=exp(-r*T)*sum(h)/I           # MCS estimator

## Output
print"Time elapsed in Seconds   %8.3f" %(time()-t0)
print"----------------------------------------"
print"European Put Option Value %8.3f" %V0
print"----------------------------------------"
```

American options—solving optimal stopping problems (I)

- to value American options by Monte Carlo simulation, a discrete optimal stopping problem has to be solved:

$$V_0 = \sup_{\tau \in \{0, \Delta t, 2\Delta t, \ldots, T\}} \mathbf{E}_0^Q (B_0(\tau) h_\tau(S_\tau)) \tag{5}$$

- it is well-known that the value of the American option at date t is then given by

$$V_t(s) = \max[h_t(s), C_t(s)] \tag{6}$$

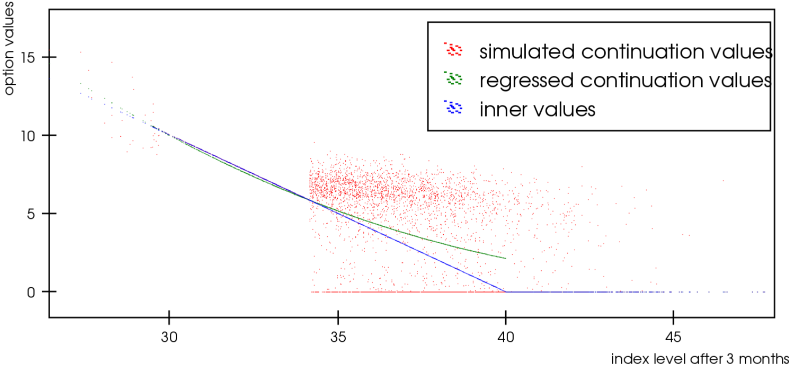
i.e. the maximum of the payoff $h_t(s)$ of immediate exercise and the expected payoff $C_t(s)$ of not exercising; this quantity is given as

$$C_t(s) = \mathbf{E}_t^Q (e^{-r\Delta t} V_{t+\Delta t}(S_{t+\Delta t}) | S_t = s) \tag{7}$$

American options—solving optimal stopping problems (II)

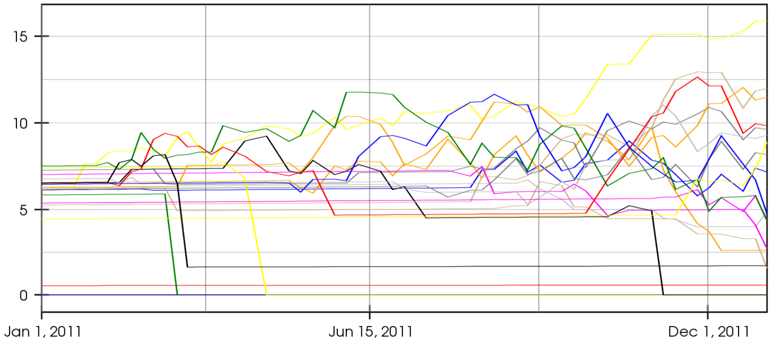
- **problem**: given a date t and a path i, you do not know the expected value in (7)—you only know the single simulated continuation value $Y_{t,i}$
- **solution of Longstaff and Schwartz (2001)**: estimate the continuation values $C_{t,i}$ by ordinary least-squares regression—given the I simulated index levels $S_{t,i}$ and continuation values $Y_{t,i}$ (use cross section of simulated data at date t)
- their algorithm is called **Least Squares Monte Carlo (LSM)**

The LSM algorithm—regression for American put option[7]



[7] Source: `analytics.dexision.com`

The LSM algorithm—backwards exercise/valuation[8]



[8] Source: `analytics.dexision.com`

Again, the Python/Numpy implementation is straightforward (I)

```
#
# Valuation of American Put Option
# with Least-Squares Monte Carlo
#
from numpy import *
from numpy.random import standard_normal,seed
from matplotlib.pyplot import *
from time import time
t0=time()

## Simulation Parameters
seed(150000)      # seed for Python RNG
M  = 50           # time steps
I  = 4*4096       # paths for valuation
reg= 9            # no of basis functions
AP = True         # antithetic paths
MM = True         # moment matching of RN

## Parameters -- American Put Option
r  = 0.06         # short rate
vol= 0.2          # volatility
S0 = 36.          # initial stock level
T  = 1.0          # time-to-maturity
V0_right=4.48637 # American Put Option (500 steps bin. model)
dt = T/M          # length of time interval
df = exp(-r*dt)   # discount factor per time interval
```

Again, the Python/Numpy implementation is straightforward (II)

```
## Function Definitions
def RNG(I):
    if AP == True:
            ran=standard_normal(I/2)
            ran=concatenate((ran,-ran))
    else:
            ran=standard_normal(I)
    if MM == True:
        ran=ran-mean(ran)
        ran=ran/std(ran)
    return ran
def GenS(I):
    S=zeros((M+1,I),'d')           # index level matrix
    S[0,:]=S0                      # initial values
    for t in range(1,M+1,1):       # index level paths
        ran=RNG(I)
        S[t,:]=S[t-1,:]*exp((r-vol**2/2)*dt+vol*ran*sqrt(dt))
    return S
def IV(S):
    return maximum(40.-S,0)
```

Again, the Python/Numpy implementation is straightforward (III)

```
## Valuation by LSM
S=GenS(I)                   # generate stock price paths
h=IV(S)                     # inner value matrix
V=IV(S)                     # value matrix
for t in range(M-1,-1,-1):
    rg=polyfit(S[t,:],V[t+1,:]*df,reg)          # regression at time t
    C=polyval(rg,S[t,:])                        # continuation values
    V[t,:]=where(h[t,:]>C,h[t,:],V[t+1,:]*df)   # exercise decision
V0=sum(V[0,:])/I # LSM estimator

## Output
print"Time elapsed in Seconds   %8.3f" %(time()-t0)
print"----------------------------------------"
print"Right Value               %8.3f" %V0_right
print"----------------------------------------"
print"LSM  Value for Am. Option %8.3f" %V0
print"Absolute Error            %8.3f" %(V0-V0_right)
print"Relative Error in Percent %8.3f" %((V0-V0_right)/V0_right*100)
print"----------------------------------------"
```

The Challenge—"dozens of minutes" in `Matlab`

- realistic market models generally include **multiple sources of randomness** which are possibly correlated
- the simulation of such complex models in combination with Least Squares Monte Carlo is **computationally demanding and time consuming**
- in their research paper, Medvedev and Scaillet (2009) analyze the valuation of American put options in the presence of stochastic volatility and stochastic short rates
- Medvedev and Scaillet (2009) write on page 16:

  "To give an idea of the computational advantage of our method, a Matlab code implementing the algorithm of Longstaff and Schwartz (2001) takes dozens of minutes to compute a single option price while our approximation takes roughly a tenth of a second."

The Results—"only seconds" in Python

- Python is well-suited to implement efficent, i.e. fast and accurate, numerical valuation algorithms
  - MCS/LSM with 25 steps/35,000 paths:

    180 megabytes of data crunched in 1.5 seconds
  - MCS/LSM with 50 steps/100,000 paths:

    980 megabytes of data crunched in 8.5 seconds
- reported times are from my 3 years old notebook ...
- the speed-up compared to the times reported in Medvedev and Scaillet (2009) is 480+ times (1.5 seconds vs. 720+ seconds)
- to reach this speed-up, our algorithm mainly uses variance reductions techniques (like moment matching and control variates) which allows to reduce the number of time steps and paths significantly

Results from 3 simulation runs for the 36 American put options of Medvedev and Scaillet (2009)

```
----------------------------------------------------
Start Calculations       2011-06-22 13:43:02.163000
----------------------------------------------------
Name of Simulation         Base_3_25_35_TTF_2.5_1.5
Seed Value for RNG                           150000
Number of Runs                                    3
Time Steps                                       25
Paths                                         35000
Control Variates                               True
Moment Matching                                True
Antithetic Paths                              False
Option Prices                                   108
Absolute Tolerance                           0.0250
Relative Tolerance                           0.0150
Errors                                            0
Error Ratio                                  0.0000
Aver Val Error                              -0.0059
Aver Abs Val Error                           0.0154
Time in Seconds                            135.7890
Time in Minutes                              2.2631
Time per Option                              1.2573
----------------------------------------------------
End   Calculations       2011-06-22 13:45:17.952000
----------------------------------------------------
```

DEXISION can handle a number of financial derivatives ranging from plain vanilla to complex and exotic

- Example products:
  - plain vanilla options
  - American options
  - Asian options
  - options on baskets
  - certificates (bonus, express, etc.)
  - swaps, swaptions
  - real options
  - portfolios of options
  - life insurance contracts
- Example underlyings:
  - indices
  - stocks
  - bonds
  - interest rates
  - currencies
  - commodities

DEXISION can be beneficially applied in a number of areas

- **financial research**: researchers, lecturers and students in (mathematical) finance find in DEXISION an easy-to-learn tool to model, value and analyze financial derivatives
- **financial engineering**: financial engineers and risk managers in investment banks, hedge funds, etc. can use DEXISION to quickly model and value diverse financial products, to cross-check valuations and to assess risks of complex derivatives portfolios
- **actuarial calculations**: those responsible for the design, valuation and risk management of market-oriented insurance products can engineer, value and test new and existing products easily
- **financial reporting**: IFRS and other reporting standards require the use of formal (option) pricing models when there are no market prices; DEXISION considerably simplifies the modelling, valuation and risk assessment for illiquid, complex, non-traded financial instruments and embedded options
- **real options valuation**: DEXISION offers unique capabilities to account for the specifics of real options (as compared to financial options)

DEXISION is based on a Python-LAMP environment and makes heavy use of Numpy

- Suse Linux 11.1 as 64 bit operating system
- Apache 2 as Web server
- MySQL 5.0.67 as relational database
- Python 2.6 as core language (integrated via mod_python in Apache)
- Numpy 1.3.0 as fast linear algebra library
- Dojo 1.0 as JavaScript framework for the GUI
- SVG for all custom graphics
- MoinMoin Wiki (Python powered) for Web documentation

Our aim is to make DEXISION the Google of Derivatives Analytics

- recently, Visixion added Web services to DEXISION's functionalities which allow to integrate it into any environment
- once a structure is modeled in DEXISION, updates of valuations can be received in real-time via these Web services (with data delivered e.g. in XML format)
- during the Web service call, data/variables can also be provided
- a call to value an American put option on the DAX index could look like:

  https://company.dexision.com/DEXISIONeval.py?company=X&user=Y&pwd=Z&paths=50000&steps=150&portfolio=DAX/DAX_Am_Put_Dec_2011&DAX_current=7200&DAX_vola=0.175&rate=0.03&strike=6800

Contact

Dr. Yves J. Hilpisch
Visixion GmbH
Rathausstrasse 75-79
66333 Voelklingen
Germany

www.visixion.com — Derivatives Analytics and Python Programming
www.dexision.com — Derivatives Analytics On Demand

E contact@visixion.com
T/F +49 3212 1129194