UNIVERSITÉ
DE GENÈVE

“Finance”:A Python Module For Becoming Rich!

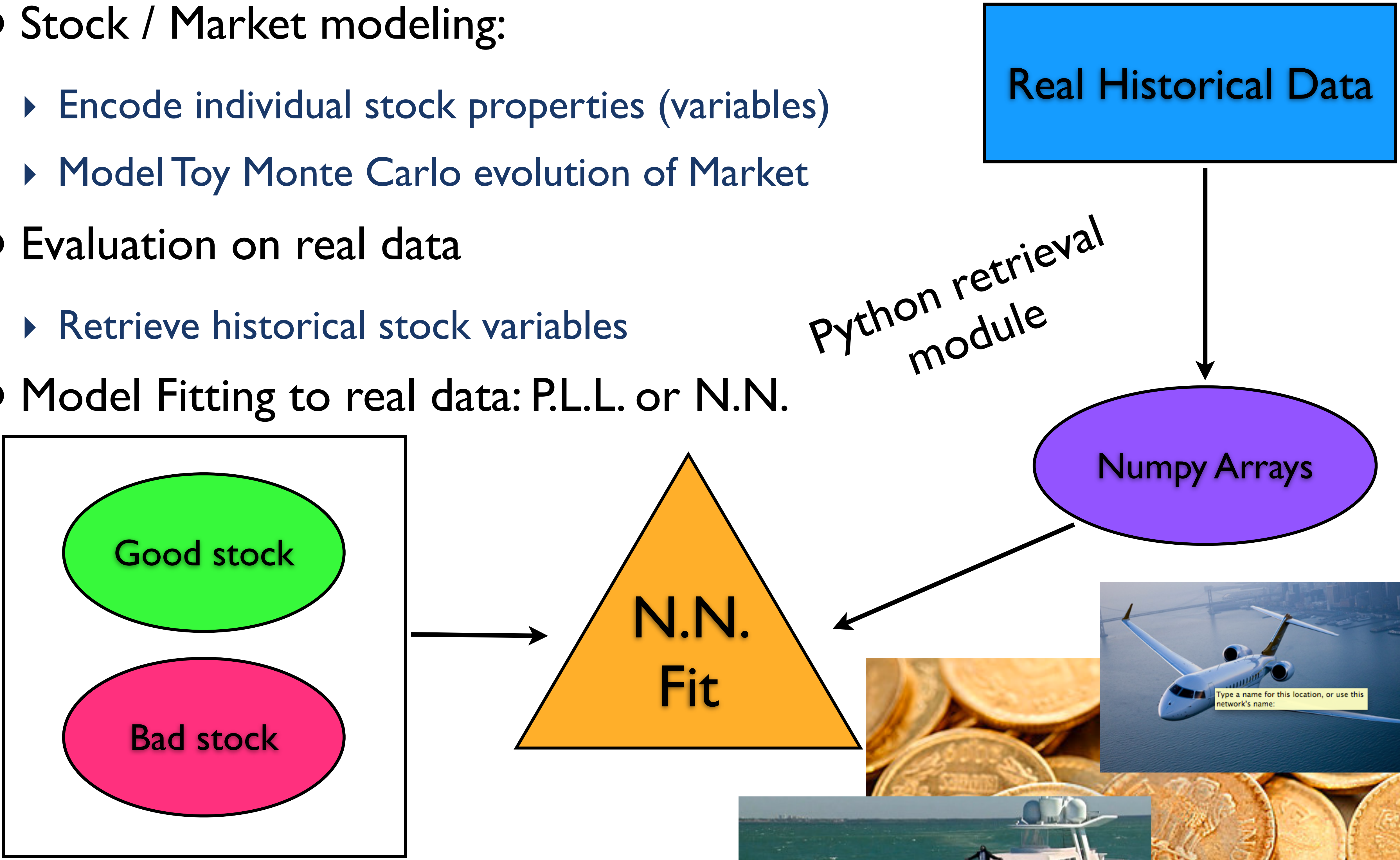
G.Barone, J. Bilbao De Mendizabal, A. Katre

Outline & Introduction

- Aim of the project:
  - Analyze and extrapolate stock behavior under market fluctuations
- Structure of the code
  - Python
  - C++ code
- Prospects
- Documentation:
  - http://gabarone.web.cern.ch/gabarone/finance/
- Repository:
  - github.com/gaebarone/fcl13/finance

Theory behind ....

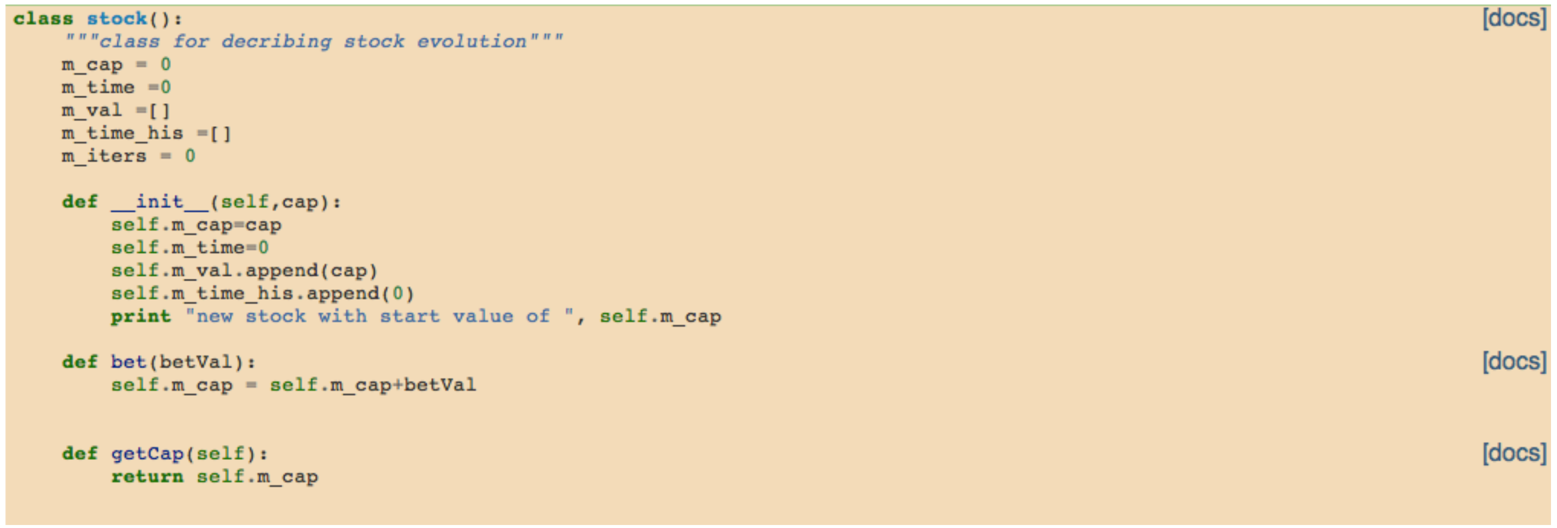
- Stock / Market modeling:
  - Encode individual stock properties (variables)
  - Model Toy Monte Carlo evolution of Market
- Evaluation on real data
  - Retrieve historical stock variables
- Model Fitting to real data: P.L.L. or N.N.



Market Virtualization

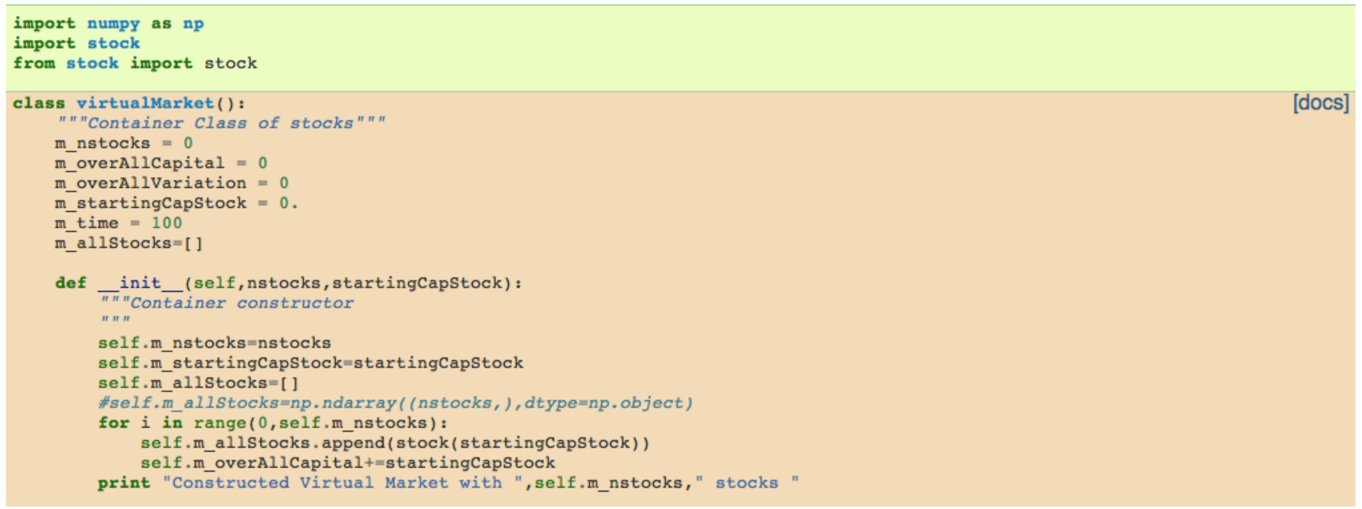
Stock Modelization

- Basic stock properties: price, time, ...
  - Base class: stock



```
class stock():                                                    [docs]
    """class for decribing stock evolution"""
    m_cap = 0
    m_time =0
    m_val =[]
    m_time_his =[]
    m_iters = 0

    def __init__(self,cap):
        self.m_cap=cap
        self.m_time=0
        self.m_val.append(cap)
        self.m_time_his.append(0)
        print "new stock with start value of ", self.m_cap

    def bet(betVal):                                              [docs]
        self.m_cap = self.m_cap+betVal


    def getCap(self):                                             [docs]
        return self.m_cap
```

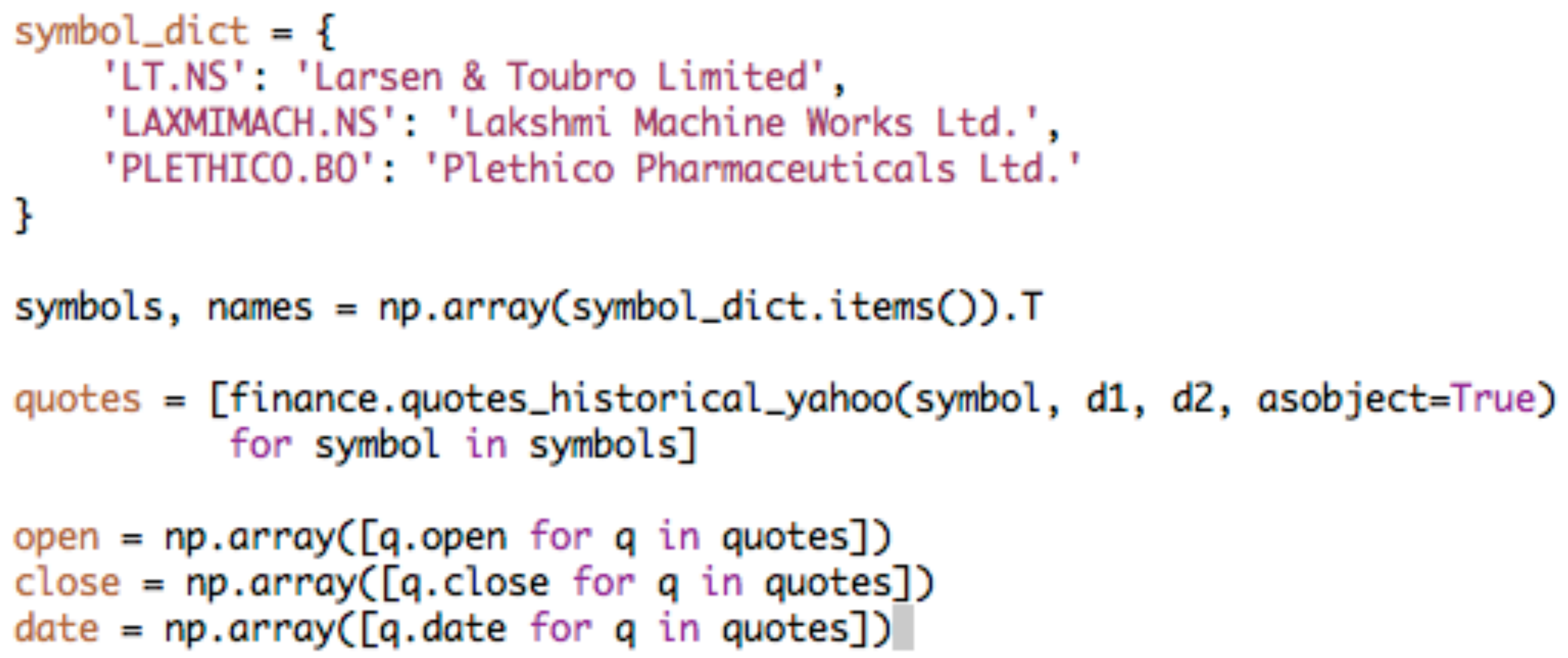
- Template behavior modeling:
  - Inherited classes from base: goodStock, badStock
  - Model given behavior under certain hypothesis
- Pile everything in container class
  - ~essentially list / array of classes



```
import numpy as np
import stock
from stock import stock

class virtualMarket():                                            [docs]
    """Container Class of stocks"""
    m_nstocks = 0
    m_overAllCapital = 0
    m_overAllVariation = 0
    m_startingCapStock = 0.
    m_time = 100
    m_allStocks=[]

    def __init__(self,nstocks,startingCapStock):
        """Container constructor
        """
        self.m_nstocks=nstocks
        self.m_startingCapStock=startingCapStock
        self.m_allStocks=[]
        #self.m_allStocks=np.ndarray((nstocks,),dtype=np.object)
        for i in range(0,self.m_nstocks):
            self.m_allStocks.append(stock(startingCapStock))
            self.m_overAllCapital+=startingCapStock
        print "Constructed Virtual Market with ",self.m_nstocks," stocks "
```

Historical Data Retrieval

- Use Finance module from scipy
  - Data from yahoo finance
- Modifications:
  - Use Numpy arrays to store open/close stock variables

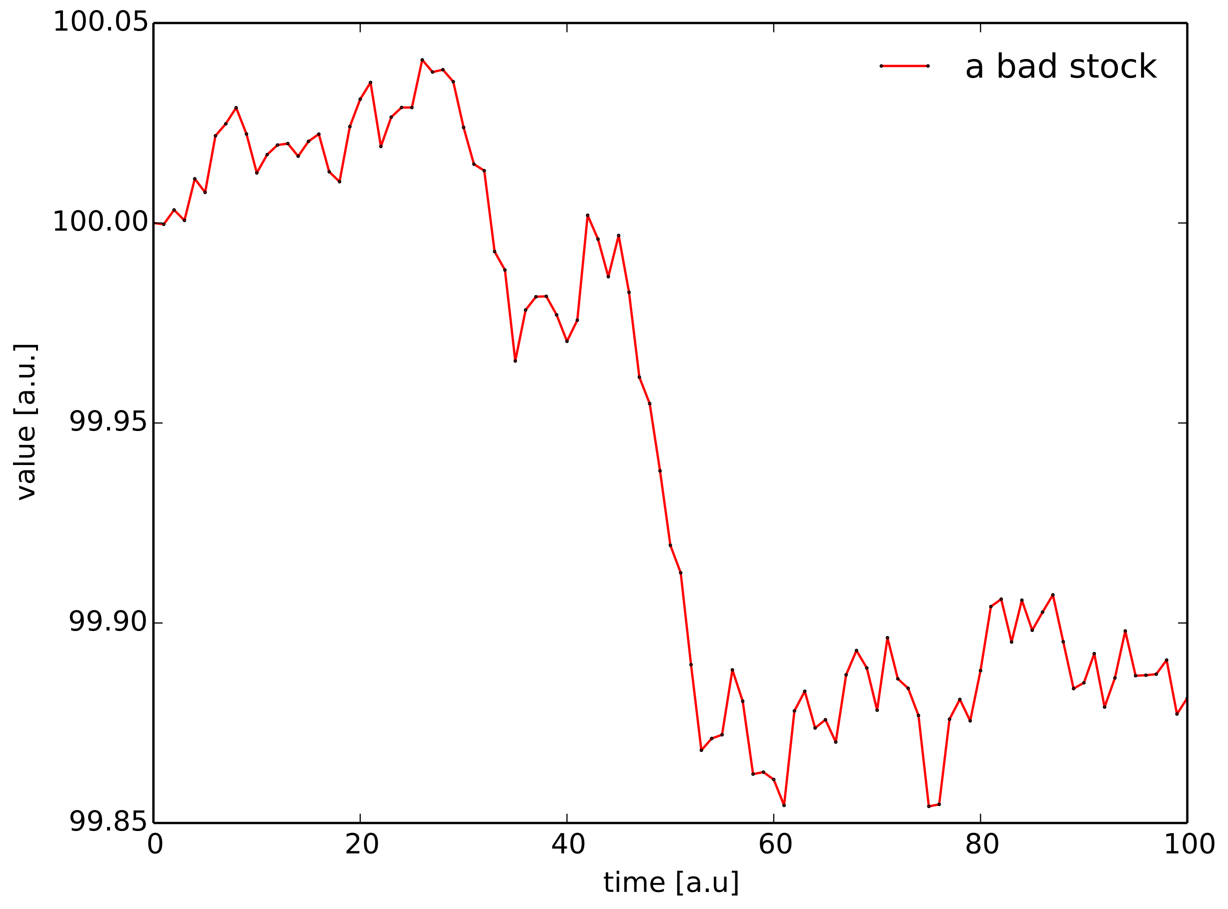


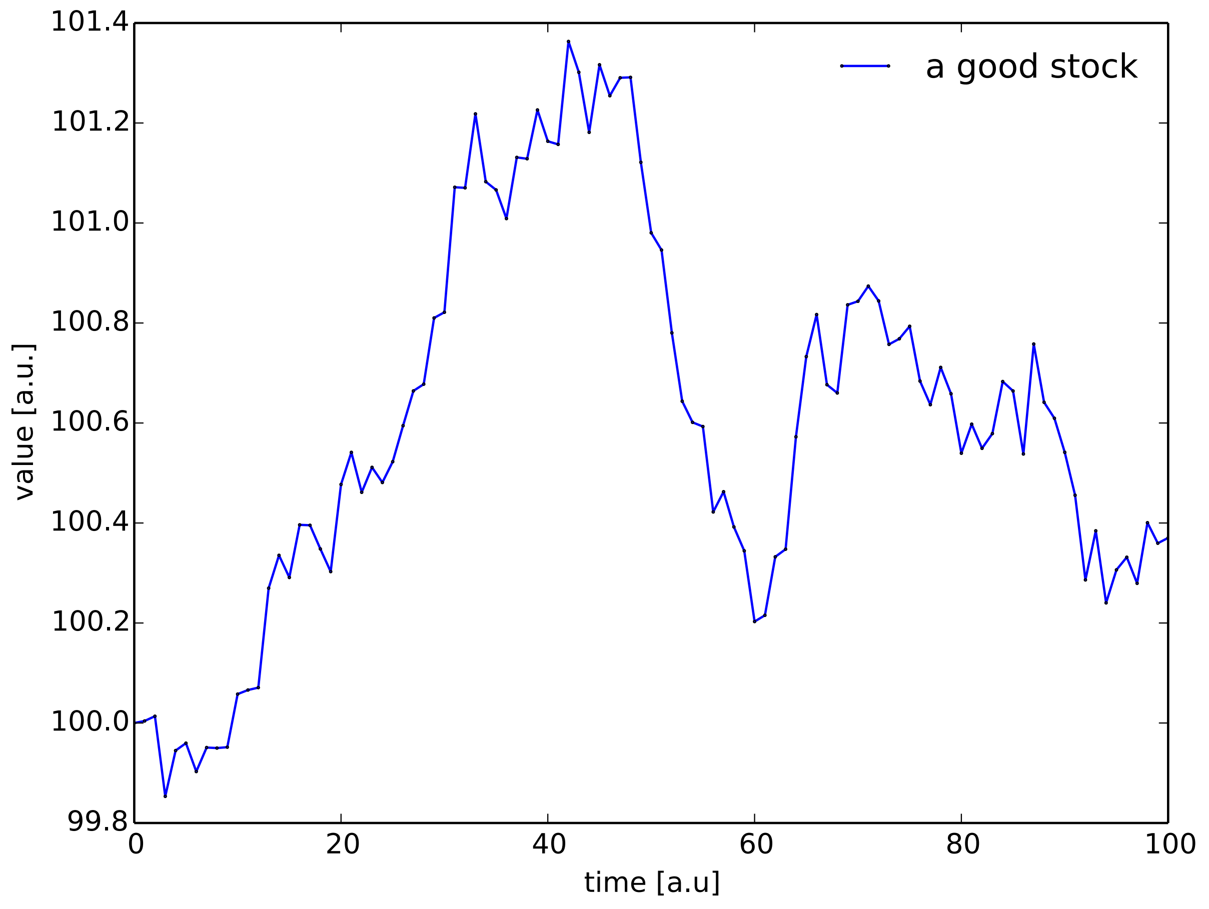
```
symbol_dict = {
    'LT.NS': 'Larsen & Toubro Limited',
    'LAXMIMACH.NS': 'Lakshmi Machine Works Ltd.',
    'PLETHICO.BO': 'Plethico Pharmaceuticals Ltd.'
}

symbols, names = np.array(symbol_dict.items()).T

quotes = [finance.quotes_historical_yahoo(symbol, d1, d2, asobject=True)
          for symbol in symbols]

open = np.array([q.open for q in quotes])
close = np.array([q.close for q in quotes])
date = np.array([q.date for q in quotes])
```

- Feed Information in data container of stock variables

Templates / Container

- Template Class Behavior





  - Constant up/down variation + random fluctuation
  - Derive Templates
- Container Class:
  - holds information about N stocks
    - real / arbitrary
  - Evolution overall of them in time range

Fit Model / Extrapolation

- Estimation
  - Profile likelihood ratio:

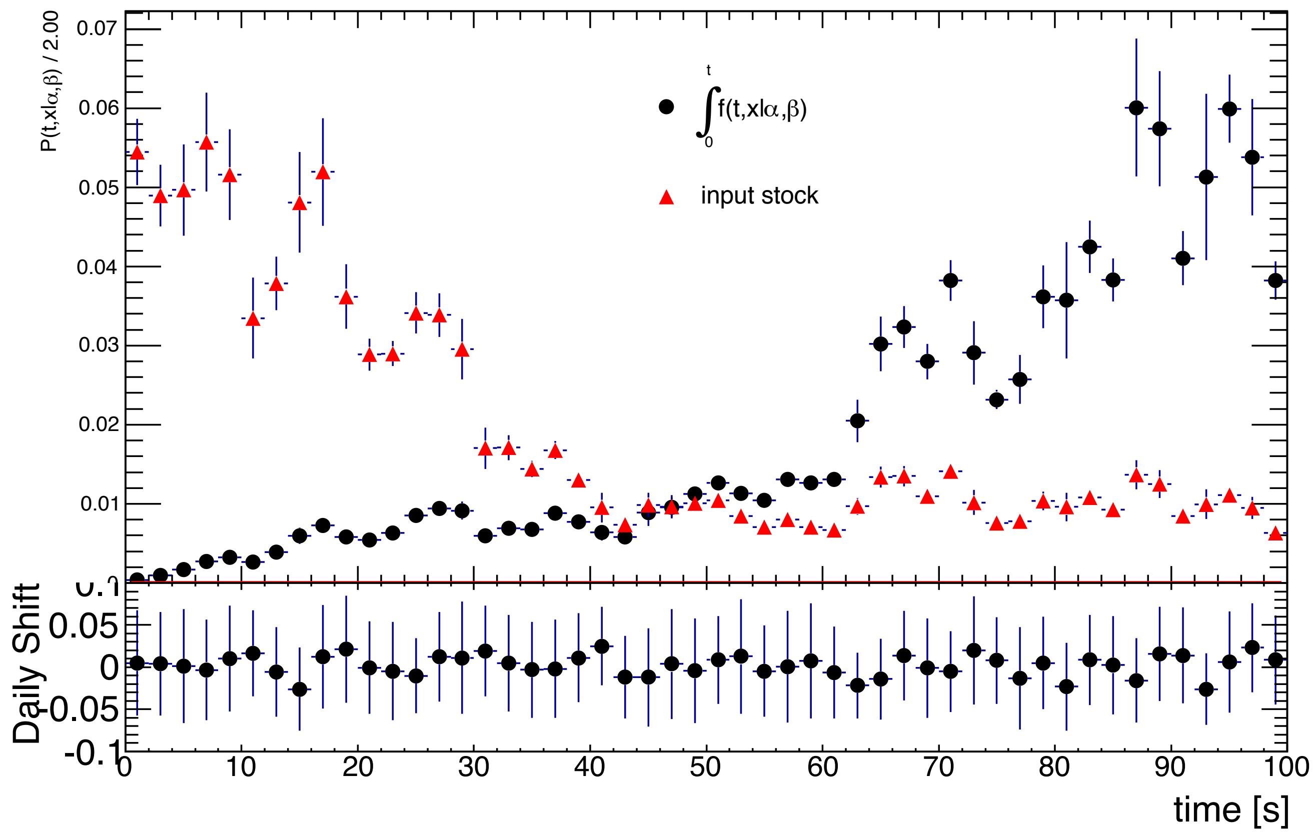
$$L = \left[N_{\text{good}} \cdot T(t, x_{i..n} | N_{\text{good}}) + N_{\text{bad}} \cdot T(t, x_{i..n} | N_{\text{bad}})\right] \Pi_{\alpha_0}^{\alpha_N} \frac{1}{\sqrt{2\pi\sigma_{\theta_i}^2}} \exp\left[-\frac{(\theta_i - \hat{\theta}_i)^2}{2\sigma_{\theta_i}^2}\right]$$

template upwards fluctuations

template downwards fluctuations

Constraints modeling extra knowledge

  - C++/ ROOT implementation
    - ✦ to be ported in python
    - ✦ pyRoot



Prospects

- Basic stock description
  - Include mode variables
    - currently time, opening price
- Templates:
  - Incorporate more systematic variations in derivation
  - Slicing in time bins
- Estimation
  - To be ported in python (pyRoot)
  - or interface C/C++ to python (time consuming)