Queens College, CUNY, Department of Computer Science
Computational Finance
CSCI 365 / 765
Fall 2017
Instructor: Dr. Sateesh Mane

October 4, 2017

3 Project Part 3

3.1 Sample classes

- Here are some derived classes to help you get started.

  (Note that you need to write the abstract base class first.)

- **Note also that the design is not final, so these implementations could change.**

- A simple `Stock` class for a stock.

  The stock pays a continuous dividend yield q (indicative data).

- A simple `Futures` class for a stock futures.

  1. The futures expires at time T (indicative data).
  2. The interest rate is r (market data).
  3. The valuation date is t_0 (market data).
  4. The `Futures` class contains an instance of a `Stock` class.
  5. The `Futures` class calls public methods of the `Stock` class.

- **These examples contain some lessons we need to learn.**
  They will be discussed later.

  1. For example, the `Futures` class does not derive from the `Stock` class.
  2. *Why not?*

3.2 Stock class

```
class Stock : public ABC
{
public:
  Stock();

  // virtual functions
  virtual int getIndicativeData(const IndicativeData & indData);
  virtual int getMarketData(const MarketData & mktData);
  virtual int validate() const;

  // data
  double _S;   // stock price
  double _q;   // dividend yield

  double _FairValue;
  double _Delta;

protected:
  virtual void setOutputs(OutputData & outData) const;
  virtual int calc();
};

Stock::Stock()
{
  _S = 0.0;
  _q = 0.0;

  _FairValue = 0.0;
  _Delta = 0.0;
}
```

```
int Stock::getIndicativeData(const IndicativeData & indData)
{
  _q = indData._divYield;
  if (_q < 0.0) return 1;  // fail
  return 0;
}

int Stock::getMarketData(const MarketData & mktData)
{
  _S = mktData._marketPrice;
  if (_S < 0.0) return 1;  // fail
  return 0;
}

int Stock::validate() const
{
  if (_S < 0.0) return 1;  // fail
  if (_q < 0.0) return 1;  // fail
  return 0;
}

int Stock::calc()
{
  _FairValue = _S;
  _Delta = 1.0;
  return 0;
}

void Stock::setOutputs(OutputData & outData) const
{
  outData.clear();
  outData._FairValue = _FairValue;
  outData._Delta = _Delta;
}
```

3.3 Futures class

```
class Futures : public ABC
{
public:
  Futures();

  // virtual functions
  virtual int getIndicativeData(const IndicativeData & indData);
  virtual int getMarketData(const MarketData & mktData);
  virtual int validate() const;

  // data
  double _r;
  double _T;
  double _t0;
  Stock _stock;

  double _FairValue;
  double _Delta;
  double _Rho;

protected:
  virtual void setOutputs(OutputData & outData) const;
  virtual int calc();
};

Futures::Futures()
{
  _r = 0.0;
  _T = 0.0;
  _t0 = 0.0;

  _FairValue = 0.0;
  _Delta = 0.0;
  _Rho = 0.0;
}
```

```
int Futures::getIndicativeData(const IndicativeData & indData)
{
  int rc = _stock.getIndicativeData(indData);
  if (rc) return rc;
  _T = indData._expirationDate;
  return 0;
}

int Futures::getMarketData(const MarketData & mktData)
{
  int rc = _stock.getMarketData(mktData);
  if (rc) return rc;
  _r = mktData._interestRate;
  _t0 = mktData._t0;
  return 0;
}

int Futures::validate() const
{
  int rc = _stock.validate();
  if (rc) return rc;
  if (_T - _t0 < 0.0) return 1;  // fail, expiration T must be after t0
  return 0;
}

int Futures::calc()
{
  double & S = _stock._S;
  double b = 0.01*(_r - _stock._q);
  double dt_exp = _T - _t0;
  double tmp = exp(b*dt_exp);
  _FairValue = S * tmp;
  _Delta = tmp;
  _Rho = _FairValue * dt_exp;
  return 0;
}

void Futures::setOutputs(OutputData & outData) const
{
  outData.clear();
  outData._FairValue = _FairValue;
  outData._Delta = _Delta;
  outData._Rho = _Rho;
}
```