Quant DSL Language Guide

Appropriate Software Foundation

March 13, 2012

Contents

Abstract

This article describes Quant DSL, a domain specific language for quantitative analytics. Language elements are combined according to the language syntax to form a statement of value which is evaluated according to the language semantics. The syntax of the language is defined with Backus–Naur Form. The semantics are defined with mathematical expressions commonly used within quantitative analytics. The validity of Monte Carlo simulation for all possible expressions in the language is proven by induction. Quant DSL has been implemented in Python as a part of the Quant software application.

1 Synthesis

Quant DSL is a domain specific language for quantitative analytics. The value of any domain specific language consists in obtaining a declarative syntax by which domain functionality can be invoked with potentially infinite variation, so that a complex domain can be supported with relatively simple software. Once underlying functionality has been abstracted to the level of a domain specific language, support for a new case can be established in a relatively short time. Because new code does not need to be written for a new case, proliferation of code that is hard to test (and therefore expensive to maintain) can be avoided.

Quant DSL is used to record and evaluate statements of value. It is hoped that the elements of the Quant DSL (for example "Market", "Fixing", "Settlement") are recognised as common terms within quantitative analytics; whilst the elements are defined fully in this article, each element may be familiar to people who are familiar with the domain. Consequently, it is hoped that statements of value that are written in Quant DSL will be readable (as well as writable).

Given the infinite variation of expression in the language, it is necessary to obtain an inductive proof of the integrity of the language for any possible expression. Although alternative proofs may be obtained, an inductive proof has been devised.

Quant DSL was invented during the development of the Quant Python package. Quant is an open source application of both the SciPy Python package (a library for scientific computation) and the Domain Model Python package (a toolkit for enterprise applications). Recent advances in software engineering practice (for example Martin Fowler's patterns of enterprise application architecture, the agile approach, or open source software) have suggested new ways to obtain appropriate functionality. In particular, the recent maturation of dynamic languages such as Python means the focus of development can remain on the supported domain. Quant has benefited from these tendencies.

Quant DSL is based on professional experience in financial institutions in London, academic training in mathematics and mathematical engineering, and professional experience in the architecture and development of enterprise applications. It is hoped that Quant DSL constitutes a fresh approach to a common concern.

2 Syntax

In Quant DSL, a statement of value is an expression in the following form.

2.1 Expressions

An expression is either a constant value, or an expected market price, or fixes an expression to a date, or settles an expression on a date, or waits until a date for an expression, or chooses between the two expressions, or is the maximum (or addition or subtraction or multiplication or division) of two expressions, or negates an expression.

```
<Expression> ::=  <Constant>
               |  "Market(" <MarketId> ")"
               |  "Fixing(" <Date> "," <Expression> ")"
               |  "Settlement(" <Date> "," <Expression> ")"
               |  "Wait(" <Date> "," <Expression> ")"
               |  "Choice(" <Expression> "," <Expression> ")"
               |  "Max(" <Expression> "," <Expression> ")"
               |  <Expression> "+" <Expression>
               |  <Expression> "-" <Expression>
               |  <Expression> "*" <Expression>
               |  <Expression> "/" <Expression>
               |  "-" <Expression>
```

2.2 Markets and Dates

Market identifiers are constrained in practice by the object model of the market exchange. Dates start with a four digit year and have dashes.

```
<MarketId> ::=  "#"<Integer>
    <Date> ::=  <Year>"-"<Month>"-"<Day>
    <Year> ::=  <Digit><Digit><Digit><Digit>
   <Month> ::=  <Digit><Digit>
     <Day> ::=  <Digit><Digit>
```

2.3 Constants

A constant is either an integer or a floating point number. A digit is a digit in base 10.

```
<Constant> ::=  <Float> | <Integer>
   <Float> ::=  <Integer>"."<Integer>
 <Integer> ::=  <Integer><Digit> | <Digit>
   <Digit> ::=  "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

3 Semantics

A statement of value is an expression v evaluated at present time t_0.

$$E[[\![v]\!](t_0)]$$

3.1 Functions of Time

Expression v defines a function $[\![v]\!](t)$ from present time t to a random variable in a probability space.

$$
\begin{aligned}
[\![Market(i)]\!](t) &= S_i e^{\sigma_i z(t-t_0) - \frac{1}{2}\sigma_i^2 (t-t_0)} \\
[\![Fixing(d,x)]\!](t) &= [\![x]\!](d) \\
[\![Settlement(d,x)]\!](t) &= e^{r(t-d)}[\![x]\!](t) \\
[\![Wait(d,x)]\!](t) &= [\![Settlement(d, Fixing(d,x))]\!](t) \\
[\![Choice(x,y)]\!](t) &= max(E[[\![x]\!](t)|F(t)], E[[\![y]\!](t)|F(t)]) \\
[\![Max(x,y)]\!](t) &= max([\![x]\!](t), [\![y]\!](t)) \\
[\![x+y]\!](t) &= [\![x]\!](t) + [\![y]\!](t) \\
[\![x-y]\!](t) &= [\![x]\!](t) - [\![y]\!](t) \\
[\![x*y]\!](t) &= [\![x]\!](t)[\![y]\!](t) \\
[\![x/y]\!](t) &= [\![x]\!](t)/[\![y]\!](t) \\
[\![-x]\!](t) &= -[\![x]\!](t)
\end{aligned}
$$

For market i, the last price S_i and volatility σ_i are determined using only market price data generated before t_0.

Brownian motion z is used in diffusion. Constant interest rate r is used in discounting. Expectation E is conditioned on filtration F.

4 Simulation

A random variable is simulated with a number N of samples from its distribution. The language semantics are simulated such that for any valid expression v, the simulated value $[\![v]\!]_m(t)$ converges in mean square and in probability to the true value of $[\![v]\!](t)$ as the number N of paths in the simulation goes to infinity.

4.1 Proof by Induction

The inductive hypothesis $I(v)$ is defined to be true if and only if, for any time t, the simulated value $[\![v]\!]_m(t)$ converges in mean square and in probability to the true value $[\![v]\!](t)$ as the number N of paths in the simulation goes to infinity.

Below, $I(v)$ is shown to be true for all v.

4.1.1 Market

Let i reference a market price history. Let z_m be a simulatation of Brownian motion z. Suppose $I(z)$.

The true value $[\![Market(i)]\!](t)$ is simulated as $S_i e^{\sigma_i z_m(t-t_0)-\frac{1}{2}\sigma_i^2(t-t_0)}$.

Since Brownian motion, exponentiation and multiplication by a constant are continuous, the simulated value $S_i e^{\sigma_i z_m(t-t_0)-\frac{1}{2}\sigma_i^2(t-t_0)}$ converges in mean square and in probability to the true value $S_i e^{\sigma_i z(t-t_0)-\frac{1}{2}\sigma_i^2(t-t_0)}$ as the number N of paths in the simulation goes to infinity.

Thus, $[\![Market(i)]\!]_m(t)$ converges in mean square and in probability to $[\![Market(i)]\!](t)$ as the number N of paths in the simulation goes to infinity.

Hence $I(Market(i))$.

4.1.2 Fixing

Let d be a date. Suppose $I(x)$.

The true value $[\![Fixing(x,d)]\!](t)$ is simulated as $[\![x]\!]_m(d)$.

Since d is also a time, the simulated value $[\![x]\!]_m(d)$ converges in mean square and in probability to the true value $[\![x]\!](d)$ as the number N of paths in the simulation goes to infinity.

Thus, $[\![Fixing(x,d)]\!]_m(t)$ converges in mean square and in probability to $[\![Fixing(x,d)]\!](t)$ as the number N of paths in the simulation goes to infinity.

Hence $I(Fixing(x,d))$.

4.1.3 Settlement

Let x be an expression, and d a date. Suppose $I(x)$.

The true value $[\![Settlement(x,d)]\!](t)$ is simulated as $e^{r(t-d)}[\![x]\!]_m(t)$.

Since $e^{r(t-d)}$ is constant, and multiplication by a constant is continuous, the simulated value $e^{r(t-d)}[\![x]\!]_m(d)$ converges in mean square and in probability to the true value $e^{r(t-d)}[\![x]\!](d)$ as the number N of paths in the simulation goes to infinity.

Thus, $[\![Settlement(x,d)]\!]_m(t)$ converges in mean square and in probability to $[\![Settlement(x,d)]\!](t)$ as the number N of paths in the simulation goes to infinity.

Hence $I(Settlement(x,d))$.

4.1.4 Wait

Let x be an expression, and d a date. Suppose $I(Fixing(d,x))$ and $I(Settlement(d,x))$.

The true value $[\![Wait(x,d)]\!](t)$ is simulated as $[\![Settlement(d, Fixing(d,x))]\!]_m(t)$.

Since $Fixing(d,x)$ is an expression, the simulated value converges in mean square and in probability to $[\![Settlement(d, Fixing(d,x))]\!](t)$ as the number N of paths in the simulation goes to infinity.

Thus, $[\![Wait(x,d)]\!]_m(t)$ converges in mean square and in probability to $[\![Wait(x,d)]\!](t)$ as the number N of paths in the simulation goes to infinity.

Hence $I(Wait(x,d))$.

4.1.5 Choice

Let x and y be expressions. Let $R = \{x,y\}$ be the alternatives for a decision to be made at time t. Let $P(r) = 0$ be the immediate payoff associated with alternative r. Let $V(r)$ be the true continuation value associated with alternative r, where $V(x) = [\![x]\!]$, and $V(y) = [\![y]\!]$. Suppose $I(x)$ and $I(y)$.

The true value $[\![Choice(x,y)]\!](t)$ is simulated as $LSM(P_m, V_m)$, where LSM is a function which computes a single step of the Longstaff and Schwartz least-squares monte carlo algorithm [1], where $P_m(x) = 0$ and $P_m(y) = 0$, and where $V_m(x) = [\![x]\!]_m(t)$ and $V_m(y) = [\![y]\!]_m(t)$.

According to Longstaff and Schwartz, the simulated value $LSM(P_m, V_m)$ converges in mean square and in probability to the true value of the maxmimum over R of $E[P(r) + V(r)|F(t)]$ as the number N of paths in the simulation goes to infinity.

By substituting for R, P and V, the maxmimum over R of $E[P(r) + V(r)|F(t)]$ becomes $max(E[[\![x]\!](t)|F(t)], E[[\![y]\!](t)|F(t)])$.

Thus, $[\![Choice(x,y)]\!]_m(t)$ converges in mean square and in probability to the true value $[\![Choice(x,y)]\!](t)$ as the number N of paths in the simulation goes to infinity.

Hence $I(Choice(x,y))$.

4.1.6 Max

Let x and y be expressions. Suppose $I(x)$ and $I(y)$.

The true value $[\![Max(x,y)]\!](t)$ is simulated as $max([\![x]\!]_m(t), [\![y]\!]_m(t))$.

Since maximisation is continuous, the simulated value $max([\![x]\!]_m(t), [\![y]\!]_m(t))$ converges in mean square and in probability to the true value $max([\![x]\!](t), [\![y]\!](t))$.

Thus, $[\![Max(x,y)]\!]_m(t)$ converges in mean square and in probability to $[\![Max(x,y)]\!](t)$ as the number N of paths in the simulation goes to infinity.

Hence $I(Max(x,y))$.

4.1.7 Addition

Let x and y be expressions. Suppose $I(x)$ and $I(y)$.

The true value $[\![x+y]\!](t)$ is simulated as $[\![x]\!]_m(t) + [\![y]\!]_m(t)$.

Since addition is continuous, the simulated value $[\![x]\!]_m(t) + [\![y]\!]_m(t)$ converges in mean square and in probability to the true value $[\![x]\!](t) + [\![y]\!](t)$.

Thus, $[\![x+y]\!]_m(t)$ converges in mean square and in probability to $[\![x+y]\!](t)$ as the number N of paths in the simulation goes to infinity.

Hence $I(x+y)$.

4.1.8 Subtraction

Let x and y be expressions. Suppose $I(x)$ and $I(y)$.

The true value $[\![x-y]\!](t)$ is simulated as $[\![x]\!]_m(t) - [\![y]\!]_m(t)$.

Since subtraction is continuous, the simulated value $[\![x]\!]_m(t) - [\![y]\!]_m(t)$ converges in mean square and in probability to the true value $[\![x]\!](t) - [\![y]\!](t)$.

Thus, $[\![x-y]\!]_m(t)$ converges in mean square and in probability to $[\![x-y]\!](t)$ as the number N of paths in the simulation goes to infinity.

Hence $I(x-y)$.

4.1.9 Multiplication

Let x and y be expressions. Suppose $I(x)$ and $I(y)$.

The true value $[\![x*y]\!](t)$ is simulated as $[\![x]\!]_m(t)[\![y]\!]_m(t)$.

Since multiplication is continuous, $[\![x]\!]_m(t)[\![y]\!]_m(t)$ converges in mean square and in probability to the true value $[\![x]\!](t)[\![y]\!](t)$.

Thus, $[\![x*y]\!]_m(t)$ converges in mean square and in probability to $[\![x*y]\!](t)$ as the number N of paths in the simulation goes to infinity.

Hence $I(x*y)$.

4.1.10 Division

Let x and y be expressions. Suppose $I(x)$ and $I(y)$.

Todo: Show that $I(x/y)$.

Hence $I(x/y)$.

5 Software

5.1 Quant Python Package

Quant DSL has been implemented within a module (*quant/dsl.py*) of the Python package called Quant [2].

Quant is an application of SciPy [3] and Domain Model [4]. Quant provides a Web user interface that allows books of trades to be defined and evaluated with real market data. Quant can be installed on Linux platforms.

6 Future Development

Various enhancements are anticipated.

Elements to simplify common expressions, such as *Option* or *European*, will be added soon.

$$\begin{aligned}
[\![Option(d,k,x,y)]\!](t) &= [\![Wait(d, Choice(x-k,y))]\!](t) \\
[\![European(d,k,x)]\!](t) &= [\![Option(d,k,x,0)]\!](t)
\end{aligned}$$

Many users will want to express optionality over a period of time.

$$\begin{aligned}
[\![American(d,k,x)]\!](t) = [\![&Option(t,k,x,Option(t+\delta t,k,x,Option(\\
&t+2\delta t,k,x,...Option(d-\delta t,k,x,Option(d,k,x,0))))))]\!](t)
\end{aligned}$$

Some users may wish to define new elements of the language, and so an extension mechanism would be desirable.

Others may benefit from evaluating statements by using analytic solutions instead of a Monte Carlo solution, where that would be a valid evaluation of the statement.

Others may wish to be warned about common usage errors, for example it should be possible to detect if a Max expression operates on expressions which depend on the underlyings at a different present time.

Others may wish to customise the random processes applied to market prices. Others may wish to evaluate statements with variable interest rates. Others may wish to derive expected cash flows from the statement of value.

References

[1] Francis A. Longstaff and Eduardo S. Schwartz. Valuing american options by simulation: A simple least-squares approach. *Review of Financial Studies*, pages 113–147, 2001.

[2] Appropriate Software Foundation. Quant. *Python Package Index*, (http://pypi.python.org/pypi/quant), 2011.

[3] The SciPy Community. Scipy. *SciPy Website*, (http://www.scipy.org), 2011.

[4] Appropriate Software Foundation. Domain model. *Python Package Index*, (http://pypi.python.org/pypi/domainmodel), 2011.