|  |  |  |
| --- | --- | --- |
|  |  | iteLogo_small  I.T.&e Limited  ACN 088 299 512  Level 11/25 Bligh Street  Sydney NSW 2000  Telephone 61 2 9221 7304  Facsimile 61 2 9235 3055  [www.ite.com.au](http://www.ite.com.au) |
|  |
| Razor Software Development Lifecycle  Pricing – Scripted Monte Carlo | |

# Razor Monte Carlo Pricing Adaptor with Scripted Payoff

Contents

[Razor Monte Carlo Pricing Adaptor with Scripted Payoff 2](#_Toc195351072)

[Contents 2](#_Toc195351073)

[1.1 Introduction 4](#_Toc195351074)

[1.1.1 Advantages of a Generalized Monte Carlo Engine 4](#_Toc195351075)

[1.1.2 Disadvantages 4](#_Toc195351076)

[1.2 Requirements 4](#_Toc195351077)

[1.2.1 Business Requirements 4](#_Toc195351078)

[1.2.1.1 User Defined Payoffs 4](#_Toc195351079)

[1.2.1.2 Define SDE and variance reduction rules 4](#_Toc195351080)

[1.2.1.3 Templates 4](#_Toc195351081)

[1.2.1.4 User Interface 5](#_Toc195351082)

[1.2.1.5 Static Configuration 5](#_Toc195351083)

[1.2.2 Technical Requirements 5](#_Toc195351084)

[1.2.2.1 Scripting Language 5](#_Toc195351085)

[1.2.2.2 FPML Integration 5](#_Toc195351086)

[1.2.2.3 Library Support 5](#_Toc195351087)

[1.2.2.4 Path Generation Engine 6](#_Toc195351088)

[1.2.2.5 Pricing Adaptor 6](#_Toc195351089)

[1.2.2.6 Razor Client Integration 6](#_Toc195351090)

[1.2.2.7 Performance Considerations 6](#_Toc195351091)

[1.3 Project Timeline and Work Estimation 6](#_Toc195351092)

[1.4 Design 7](#_Toc195351093)

[1.4.1 Phase 1 – Single Asset 7](#_Toc195351094)

[1.4.1.1 Product Scope 7](#_Toc195351095)

[1.4.1.2 Script Interpreter 7](#_Toc195351096)

[1.4.1.3 Support Library 9](#_Toc195351097)

[1.4.1.4 Path Generation Engine 9](#_Toc195351098)

[1.4.1.5 Razor C++ Pricing Adaptor 9](#_Toc195351099)

[1.4.1.6 Razor Client Front-End 9](#_Toc195351100)

[1.4.2 Phase 2 – Multi Asset 10](#_Toc195351101)

[1.4.2.1 Product Scope 10](#_Toc195351102)

[1.4.2.2 Performance Considerations 11](#_Toc195351103)

**Document Amendment History**

|  |  |  |  |
| --- | --- | --- | --- |
| Version Number | Description of Change | Person making change | Date |
| 0.1 | Initial draft created. | Richard Lewis | 1 April 2008 |
| 0.2 | Added screen mockup | Richard Lewis | 2 April 2008 |

**References**

| No | Document Title | Location |
| --- | --- | --- |
| 1 | Razor Financial Principles | svn::razor\trunk\Analytics |
| 2 | Benchmarking and Implementation of Probability-Based Simulations on Programmable Graphics Cards | svn::razor\trunk\Analytics\Documentation\research\GPU MonteCarlo.pdf |

## Introduction

The intention of this project is provide a generalized Monte Carlo framework for pricing financial derivatives. The framework will consist of a path generation engine, Razor pricing adapter, user-defined payout function, template maintenance and a GUI front-end. Generalization is achieved by allowing asset payouts and stochastic processes to be modeled in a high level embedded scripting language available either through the Razor Client or trade configuration.

### Advantages of a Generalized Monte Carlo Engine

* Easy to implement for complex derivatives
* Scripted payoff functions in an easy to understand high level language
* Allow engineers and clients to generate new product pricing models
* Fast turnaround of new products
* Toolbox for model validation

### Disadvantages

* Heavy computational costs
* Slow rate of convergence
* May not be suitable for portfolio pricing and scenario analysis due to performance constraints

## Requirements

### Business Requirements

#### User Defined Payoffs

* Ability to define payoff functions and stochastic processes outside of C++ Razor
* Expressive, easy to learn scripting language with variables, functions and basic flow control

#### Define SDE and variance reduction rules

* Library of stochastic models exposed as script functions
* Library of random sequence generators including pseudo and quasi random exposed as script functions
* Ability to select difference variance reduction techniques

#### Templates

* Save payoff scripts as a template
* Template maintenance utility
* Associate templates with existing option trade types as extensions
* Sharing and Security
* Versioning

#### User Interface

The user interface will consists of two types of input screens. The first is a generic form that will allow editing of the payoff function. The second is an existing option screen that allows association of a payoff script via trade extensions.

##### Generic Option Form

* Form screen for standard fields (strike, volatility etc)
* Editor for scripts with syntax checking
* Ability to create, store, load and share templates
* Specify seeds for random generator
* Select number of paths and repeats
* Select which variance reduction techniques should be used
* Graphical display of path distribution
* Display error estimate in analysis

##### Trade Extensions to existing Option Forms

* Allow the user to specify a template script name as a trade extension. The underlying pricing adapter is switched according to the trade mapping rules.

#### Static Configuration

* Allow trade mapping rules to rebind trades with extensions to the monte carlo adapter
* Monte Carlo Parameters – number of paths, number of repeats, time step size, define variance reduction rules
* Random seed storage
* Associate script templates within product configurations

### Technical Requirements

#### Scripting Language

* Easy to learn syntax
* Well known
* Platform agnostic
* Can be called from C++ , can call C++
* Free, without license restrictions
* Can be compiled into C++ (or directly into a shared object)

#### FPML Integration

* There will be a generic derivative type defined as FpmlGenericOption

#### Library Support

A common library of routines to support operations required for Monte Carlo simulation

* Vectorized versions of basic math functions provided by Intel Math Kernel library
* Low discrepancy number generation
* Stochastic processes returned a stream of generated paths
* Call wrapper for Python generated scripts

#### Path Generation Engine

A separate engine is to be constructed that generates and stores paths based on vectorized random sequence generation. Vector streams are to be stored and associated trade objects. Paths are generated from a vectorized random sequence

#### Pricing Adaptor

The pricing adaptor will support both FpmlGenericOption and any option type that has a payout script associated with the trade as a trade extension.

##### Trade Extensions

Payoff scripts can be associated with a trade a trade extension.

##### Trade Mapping Rules

Trade mapping rules allow overriding static pricing adaptor binding to existing option types by switching to the Monte Carlo adapter if a trade extension exists for that trade.

#### Razor Client Integration

Integration with the Razor client will include a new form to support FpmlGenericOptions allowing editing of scripts, basic management of templates and pricing. Price, standard error and generated paths will need to be returned by the pricing adaptor for use in graphical analysis of distribution.

#### Performance Considerations

* Can be run either on a single node or a cluster of nodes. A node can be defined as a single light weight thread or as a distributed remote LWP.
* Path generation coded such that it can easily be vectorized
* Support for variance reduction

## Project Timeline and Work Estimation



## Design

### Phase 1 – Single Asset

#### Product Scope

##### FX and Equity Single Asset Models

Since spot derivatives contain a single asset they are the easiest to simulate. Therefore, support for these products will be delivered in the initial phase. Multi-asset and multi-factor models will be provided in the next phase. For both FX and Equities options on spot the underlying asset price follows a simple geometric Brownian motion *dS* = *μSdt* + *σSdz*. Volatility, risk-free rate (domestic rate for FX) and dividend yield (foreign rate for FX) remain constant. Payoff support will be for Vanilla European, Bermudan, Asian and Barrier options. Baskets and other multi-asset types require correlations and will included in the next phase.

##### Single Factor Interest Rate (Short-rate) Models

It is assumed that volatility is constant and that a single underlying interest rate term structure is simulated.

* Vasicek Model can be used to define a short interest rate risk neutral process as: *dr* = *a*(*b − r*)*dt* + *σdz* where a, b and are constant.

#### Script Interpreter

Model scripts are to be written in the embedded scripting language Python and include the following structure:

* Model Identifier through class extension
* Variables – support for stochastic and constant variables
* Random Sequences
* Stochastic Process models
* Analytical models
* Payoff Function

##### Variable Definitions

All variables are vectors. Scalars are a vector of type variant and length 1. Vectors can be multidimensional. Standard vector operators such as “:” used for slicing are available.

##### Random Sequences

To include support for types: pseudo, quasi and a distribution (uniform, normal, lognormal) etcetera. An example would be:

R = random(“pseudo”,”normal”, seed)

The function is empty and will be replaced at run-time by an appropriate generator based on the Intel Math Kernel Library. Sequences are generated as vectors and stored for each path.

* Sequence storage and retrieval
* Look-ahead and leap-frog partitioning

##### Stochastic Asset Models

These models define the type of path generation that is to be performed based on a geometric Brownian motion.

* Black Scholes
* Vasicek

These assets underlying these models are assumed to be lognormally distribut accept a random stream vector, a expected rate of return (drift rate)  and a volatility 

* Lognormal process
* Mean-reverting (normal and lognormal)

##### Analytical Models

Basic analytic models are provided as a convenience to support put/call and barrier parities and for possible use as a control variate used for variance reduction.

* Black-Scholes
* Garman-Kohlhagen (for FX)

These models will need to return price, delta and gamma

##### Payoff Function

The payoff function is to take type-less vector arguments and returns a scalar value as a one dimensional vector. The standard set of math functions is to be included. Additional functions e.g.: ABS, MIN, MAX, AVG (arithmetic, geometric, harmonic) will also be provided. These functions will be inherently vector based. Flow control is provided via for iteration and if statements

###### Examples Payoff Functions

The following are examples in Python of how the payoff function can be written.

Vanilla

class Vanilla(Option):

def payoff(S, K): # S is a vector, K is scalar

return max(S-K,0)

Asian

Asian option payoffs are calculated based on the average of the underlying asset price. Two possible solutions to the payoff function can be provided as:

class Asian(Option):

def payoff(S, K):

return max(avga(S)-K,0) #using arithmetic average

class Asian(Option):

def payoff(S, K):

ret = 0

for s in S: # using a for loop

ret = ret + s

return max(ret/(S.length) – K, 0)

Barrier

# Knock-out

class Barrier(Option):

B = 0

class KnockOut(Barrier):

def payoff(S, K, B):

ret = 0

hit = False

for s in S: # using a for loop

if (s < B):

hit = true

break

if (hit):

ret = max(S-K,0)

return ret

Bermudan

class Bermudan(Option):

def payoff(S, K, t): # t is vector of observation points

St = S[t] # create a partition of S over t

return max(St-K,0)

#### Support Library

The support library will contain convenience methods for obtaining averages and standard deviation (or error) across each path, wrapping of the Python payoff function and wrapping support for the Kernel Library.

#### Path Generation Engine

A grid (multi-node) based engine which generates and stores random number sequences and generates paths for a give stochastic process. The engine will initially be thread based and subsequent versions will be both thread and grid based.

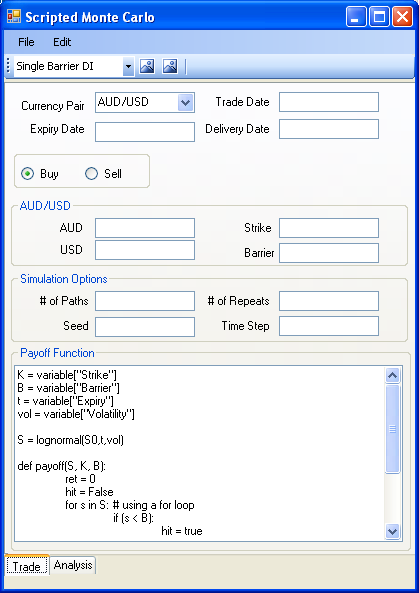
#### Razor C++ Pricing Adaptor

The pricing adapter will adopt the same protocol as existing adapters. There is to be no path simulation or calculation of price within the adapter as this will be supported by path generation engine.

#### Razor Client Front-End

This will consist of a new form to support creation and analysis of

* Extension of FPML to support payoff structures as arbitrary strings?
* GUI to support variable initialization and script generation through an editor



Template selection determines form behavior.

These fields are dynamically constructed from the template class

### Phase 2 – Multi Asset

#### Product Scope

##### American Options

Boyle, Broadie, Glasserman (1997)

##### General Multi-Asset

* Spread Options

##### Equities

* Baskets
* Compound Options

##### Interest Rates

* Bermudian Swaptions
* Cross-Currency Basis Swaptions
* Stochastic Volatility

##### Commodities

* Baskets
* Future Options

##### Hybrids

* Convertible Bonds
* PDRC

#### Performance Considerations

##### Script Optimization

* Compiled payoff function
* scripts are compiled into shared objects (DLL) at runtime
* cache maintained for compiled templates

##### Variance Reduction

* Antithetic variates – use symmetries inherent in distributions to constrain the integration domain
* Browian bridge
* Control variates
* Latin Hypercube sampling

##### Low Discrepancy Sequences

* SOBOL
* Faure
* Halton
* Mercene Twistors

##### Vectorization

* Intel Math Kernel Library 10r2
* Vectorized math functions compiled explicitly against SSE2 instruction set

##### GPU Libraries

Vectorization of path generation can be achieved through the [NVIDIA Cg Compiler](http://developer.nvidia.com/object/cg_toolkit.html). This is a general purpose high-level language

[General-Purpose Computation Using Graphics Hardware](http://www.gpgpu.org/%20)

|  |  |
| --- | --- |
| **1.** | Assign each processing element a random sequence. Each processing element must use a different random number sequence, which should be uncorrelated with the sequences used by all other processors. |
| **2.** | Propagate the simulation parameters (for example, S0) to all processing elements, and tell them how many simulation runs to execute. |
| **3.** | Generate random number streams for use by each processing element. |
| **4.** | Execute the simulation kernel on the processing elements in parallel. |
| **5.** | Gather the simulation outputs from each processing element and combine them to produce the approximate results. |

##### Grid Architecture

[DataSynapse Grid Server](http://www.datasynapse.com/en/products/gridserver.php)