Implementing Numerical Methods for Complex Options

Nick Webber

Motivation

Lots of different options to value.

Industry environment:

Expensive to have tailored valuation tools for each option,
Need a generic tool, capable of valuing broad classes of options.

Implication:

Need to be able to describe the options you can value.

This paper: works the other way around.

1) Come up with a formal definition of a generic option,
2) Describes a generic valuation algorithm.

Key Questions

What is a derivative?	How to define an option? Encapsulates the option contract.
What is a model?	How to specify eg the processes in a model?
What is valuation?	How to define and employ valuation methods? Explicit solution? Monte Carlo?

In each case must:

Define the scope of what a valuation algorithm can do.

The option

Simple European call option:

Attributes are maturity time T, strike X.

But can have:

Option payoffs depending on path statistics,

Early exercise (exchange into a range of alternative options),

Exercise can be: Mandatory, conditional, voluntary.

Many underlying assets.

The model:

Many state variables; hard to simulate.

Usually no explicit solutions; usually do not know distributions.

Key ideas:

An option is a (particular sort of) graph.

Valuation methods operate on graphs.

Valuation methods can be generic: widely applicable.

Formal specification of an Option

An option is a set

$\{\tau, \partial, E_1,\ldots,E_N, F_1,\ldots,F_M\}$

where $0 \le \tau \le T_{max}$ is the end time of the option,

Final time of the current exchange bundle.

∂ is the continuous dividend yield received by option holder,

(will be zero. Ignored in the sequel.)

Exchange specifications:

E_i, $i = 1,\ldots,N$, $N \ge 0$, exchanges active over $[0, \tau]$, 'initial'

F_j, $j = 1,\ldots,M$, $M \ge 0$, mandatory exchanges at τ only, 'terminal'.

Exchange specifications?

$E = \{\mathcal{P}, \mathcal{M}, O, R\}$.

$\mathcal{P}$,	Condition: true if can exchange, else false.
$\mathcal{M}$,	Choice type: mandatory or discretionary.
(O, R),	An option-rebate pair that is entered into upon exchange.

Recursive definition:

Assume each branch terminates after finite number of steps.

Implies that the option definition:

Includes static replication trading strategies,

Excludes continuous trading strategies.

Upon an Exchange

If exchanged at $0 \leq t \leq \tau$, immediately receive

i) The option O,

ii) Cash of $R(t, g_1(\omega),\ldots,g_q(\omega))$, (may be negative.)

Condition:

Function of path statistics, eg, if a barrier has been hit.

Choice type:

Marked-point process, values in symbol set {M, D}.

Function of path statistics, but in practice constant.

(Ignore exchanges at option of counterparty.)

Rebate:

Function $R(t, g_1(\omega_t),\ldots,g_q(\omega_t))$ of path statistics, g_i of the path to date ω.

Basic examples

Formulation contains most (all?) plain and exotic options.

Distinguished option: **0**. Zero options, has no exchanges or cash-flows.

Call option, strike X**:**

$c = \{T, F_1\}$,

$F_1 = \{\text{true}, M, \mathbf{0}, R\}$,

$R(x) = (x - X)_+$, $g(\omega_T) = S_T$.

American put option, strike X**:**

$c = \{T, E_1\}$,

$E_1 = \{\text{true}, D, \mathbf{0}, R\}$,

$R(x) = (X - x)_+$, $g(\omega_T) = S_T$.

Certain cash-flow:

To get a certain cash-flow of R at time T,

$c = \{T, F_1\}$,

$F_1 = \{\text{true}, M, \mathbf{0}, R\}$,

where R defines the cash-flow.

Up and out Barrier call option, strike X**:**

$c = \{T, E_1, F_1\}$,

$E_1 = \{S_t > U, M, \mathbf{0}, 0\}$,

$F_1 = \{\text{true}, M, \mathbf{0}, R\}$,

$$R(x) = (x - X)_+, \quad g(\omega_T) = S_T.$$

Up and in Barrier option:

$b = \{T, E_1, F_1\}$,

$E_1 = \{S_t > U, \; M, \; c, \; 0\}$,

$F_1 = \{\text{true}, \; M, \; \mathbf{0}, \; 0\}$,

where c is the underlying call option maturing at T.

Discrete dividends: Represented as a compound option.

Sequence of dividend dates: $0 = t_0 < t_1 < ... < t_N = T_{max}$.

eg, Europe call, c_0, discrete dividend d_i at times t_i.

$c_i = \{t_{i+1}, F_i\}$, $i = 0,\ldots,N-2$,

$F_i = \{\text{true}, \; M, \; c_{i+1}, \; d_{i+1}\}$,

$c_{N-1} = \{t_N, F_{N-1}\}$,

$F_{N-1} = \{\text{true}, \; M, \; \mathbf{0}, R\}$, for a payoff R.

Bermudan options: Represent as a compound option.

Times $0 = t_0 < t_1 < ... < t_N = T_{max}$.

eg, Bermudan option c_0, exercisable at t_i, $i = 1,\dots,N$, payoff function R.

$c_i = \{t_{i+1}, F^1{}_i, F^2{}_i\}$, $i = 0,\dots,N-2$

$F^1{}_i = \{\text{true}, D, \mathbf{0}, R\}$,

$F^2{}_i = \{\text{true}, D, c_{i+1}, 0\}$,

$c_{N-1} = \{t_N, F_{N-1}\}$,

$F_{N-1} = \{\text{true}, M, \mathbf{0}, R\}$.

Underlying has constant time of maturity?

Could define with a condition true only at reset times.

Exchange Specifications

General exchange specification,	$\mathcal{E}$,	$\{\mathcal{P}, \mathcal{M}, O, R\}$, $t_a < t_b$,
Terminal exchanges,	$\mathcal{F}$,	$\{\mathcal{P}, \mathcal{M}, O, R\}$, $t_a = t_b$,
Unconditional mandatory cash,	$\mathcal{E}_R$,	$\{\text{true}, M, \mathbf{0}, R\}$,
Unconditional discretionary cash,	$\mathcal{E}_A$,	$\{\text{true}, D, \mathbf{0}, R\}$,
Conditional mandatory cash,	$\mathcal{E}_P$,	$\{\mathcal{P}, M, \mathbf{0}, R\}$,
Conditional cash,	$\mathcal{E}_V$,	$\{\mathcal{P}, \mathcal{M}, \mathbf{0}, R\}$,
Conditional mandatory into Europeans,	$\mathcal{E}_B$,	$\{\mathcal{P}, M, O, R\}$, $O \in E$,
Conditional into Europeans,	$\mathcal{E}_M$,	$\{\mathcal{P}, \mathcal{M}, O, R\}$, $O \in E$,
Conditional mandatory exchanges,	$\mathcal{E}_G$,	$\{\mathcal{P}, M, O, R\}$, $O \in \mathcal{O}$,

Option types (i)

(N, M)	Option specification	Symbol	Name
(*,*),		$\mathcal{O}$,	General options,
(0,*),	$F_1 \in \mathcal{E}_R$,	$\mathcal{C}$,	General compound options,
(0,1),		$\mathcal{C}_1$,	Mandatory compound options,
(0,1),	$F_1 \in \mathcal{E}_R$,	E,	European options,
(0,2),		$\mathcal{C}_2$,	General chooser options,
(0,2),	$F_1, F_2 \in \mathcal{E}_B \cap \mathcal{F}$,	$\mathcal{C}_R$,	Restricted compound options,
(0,2),	$F_1 \in \mathcal{E}_R$, $F_2 \in \mathcal{E}_B \cap \mathcal{F}$,	$\mathcal{C}_C$,	Ordinary chooser options,

Option types (ii)

(N, M)	Option specification	Symbol	Name
(1,0),	$E_1 \in \mathcal{E}_A$,	$\mathcal{A}_V$,	Simple vanilla Americans,
	$E_1 \in \mathcal{E}_V$,	$\mathcal{V}$,	Vanilla options,
	$E_1 \in \mathcal{E}_M$,	$\mathcal{M}$,	American compound options,
(1,1),	$\mathcal{E}_1 \in \mathcal{E}_P$, $F_1 \in \mathcal{E}_R$,	$\mathcal{R}$,	Rebates,
	$E_1 \in \mathcal{E}_A$, $F_1 \in \mathcal{E}_R$,	$\mathcal{A}$,	Simple American options,
	$E_1 \in \mathcal{E}_B$, $F_1 \in \mathcal{E}_R$,	$\mathcal{B}_1$,	Simple barrier options,
	$E_1 \in \mathcal{E}_2$, $F_1 \in \mathcal{E}_R$,	$\mathcal{G}_R$,	Restricted barrier options,
(2,1),	$F_1 \in \mathcal{E}_R$, $E_1, E_2 \in \mathcal{E}_B$,	$\mathcal{B}_2$,	Duplex barrier options,
(*,*),	$\{E_i\} \cup \{F_j\} \in \mathcal{E}_G$,	$\mathcal{G}$,	General barrier options,

Graph theoretic formulation

Represent an option as a graph.

Vertices are options,

Edges are exchange specifications.

Valuation algorithms operate on graph data structures.

Option graphs are a bit special.

Underlying data structure is a:

Directed acyclic rooted terminated ordered bi-edged graph,

Augmented with edge data.

Acyclic? Can relax this.

Directed:

Exchanges are in one direction only,

Acyclic:

Can't exchange back into a previously held option (non-returning)

Rooted:

Has a single common ancestor vertex – the option you are valuing.

Terminated:

All exchanges end in an exchange into the zero option **0**.

Ordered:

Maturity dates provide an edge-consistent vertex ordering.

Bi-edged:

Edges are one of two colours: Indigo (initial) or Turquoise (terminal).

Augmented:

Edges have data attached: condition/rebate/choice type.

Graphs

Underlying data structure is an:

directed acyclic rooted terminated ordered bi-edged graph, augmented with edge data.

A graph is a set G = (V, E) of vertices and edges.

a) a set of vertices, V,
b) a set of edges, E.

An edge e is $(u, v) \in V \times V$; multiple edges are allowed.

If $e = (u,v) \in E$ then e is a (directed) edge from u to v.

Projection operators $\pi_i : E \rightarrow V$, $i = 1,2$,

$\pi_1(u,v) = u$, projection onto the parent node,

$\pi_2(u,v) = v$, projection onto the child node.

For $e \in E$ also write p_e and c_e (or e_p and e_c) for $\pi_1(e)$ and $\pi_2(e)$.

Acyclic directed graphs

A chain of edges is a sequence $e_1,\ldots,e_N$ such that
for all $i = 2,\ldots,N-1$, $\pi_1(e_i) = \pi_2(e_{i-1})$,

A cycle is a chain $e_1,\ldots,e_N$ such that $\pi_2(e_N) = \pi_1(e_1)$.

A graph is acyclic if it contains no cycles.

An option corresponding to an acyclic graph cannot return to a previous state.

eg, a European call option with an up-barrier turning it into a European put, the European put has a down barrier turning it back into the European call.

Root vertices

For $v \in V$,

a parent vertex is a $u \in V$ such that $(u, v) \in E$,

a child vertex is a $u \in V$ such that $(v, u) \in E$,

A root vertex is one that has no parents, ie

v is a root vertex if there exists no $u \in V$ such that $(u, v) \in E$.

A graph G is rooted if it has exactly one root vertex.

Write $* \in V$ for the unique root vertex in a rooted graph.

If G is rooted acyclic then for all $v \in V$ then $\exists$ a chain $e_1, \ldots, e_N$ such that

$\pi_1(e_1) = *$ and $\pi_2(e_N) = v$.

Leaf vertices

A leaf vertex is one that has no children, ie

v is a leaf vertex if there exists no $u \in V$ such that $(v, u) \in E$.

A graph is terminated if it has a exactly one leaf vertex.

Write $\circ \in V$ for the unique leaf vertex.

If G is acyclic terminated then $\forall v \in V$ then $\exists$ a chain $e_1,\ldots,e_N$ such that

$\pi_1(e_1) = v$ and $\pi_2(e_N) = \circ$.

For an option the leaf node is the zero option,
one that has no cash flows, ever.

Ordered

We suppose $\exists$ an edge-compatible ordering $\leq$ on V, ie $\leq : V \times V \rightarrow \mathbb{R}$ st

i) $\forall u, v \in V$ either $u \leq v$ or $v \leq u$,

ii) $\forall u, v, w \in V$ if $u \leq v$ and $v \leq w$ the $u \leq w$.

iii) If $(u, v) \in E$ then $u \leq v$.

We suppose we have selected a map $t : V \rightarrow \mathbb{R}$ st $u \leq v$ iff $t(u) \leq t(v)$.

In a graph with cycles, if $u \leq v$ and $v \leq u$ then $t(u) = t(v)$

We may write u_t or t_u for $t(u)$.

By convention set $t(*) > 0$, $t(\circ) = \infty$, so

$$0 < t(*) \leq t(u) < t(\circ)$$

for all $\circ \neq u \in V$.

For an option, $t(v)$ is the end time or maturity time of the option v.

Bi-edged graphs

Vertices and edges may be coloured,
ie marked by a property taking values in a (finite) set (of eg colours).

Bi-edged?: Edges take one of two colours, eg indigo and turquoise.
Set $E = E^I \cup E^T$, the union of indigo edges and turquoise edges.

For an option:

- Indigo edges correspond to initial exchanges.
- Turquoise edges correspond to terminal exchanges,

For $v \in V$ write

- $P^I(v)$ for the parent vertices of v connected to v by indigo edges,
- $P^T(v)$ for the parent vertices of v connected to v by turquoise edges.

Start times

Define the (effective) start time at a vertex.

The start time of a vertex $v \in V$ is $t^s(v)$ for $t^s : V \rightarrow \mathbb{R}$ st

i) $t^s(*) = 0$ (by convention)

ii) for $v \in V$, $v \neq *$, $t^s(v) = \min\{ \{t^s(u)\}_{u \in P^I(v)} \cup \{t(u)\}_{u \in P^T(v)} \}$.

The start time of an option is the earliest time at which an exchange into the option may take place.

The final time:

Define $t_{max} = \max_{v \in V\backslash\circ}\{t(v)\}$.

t_{max} is the greatest maturity time occurring in the option specification

Exchange times

Let T_v be the set of times when an exchange into v may take place.

Have

$$T_v = \bigcup_{u \in P^I(v)} [t^s(u), t(u)] \bigcup \bigcup_{u \in P^T(v)} \{t(u)\}$$

$$\subseteq [t^s(v), t(v)] \subseteq [t^s(*), t(v)] = [0, t(v)],$$

so $t^s(v) = \min\{ t \in T_v \}$.

Let T_e be the set of times when an exchange through e may take place.

Then $T_e = [t^s(e_p), t(e_p)]$, if $e \in E^I$,

$T_e = \{t(e_p)\}$, if $e \in E^T$,

A vertex v is exchange-active at time t if $t \in T_v$,

An edge e is exchange-active at time t if $t \in T_e$.

Vertex and edge data

Vertices and edges may contain data,

$f^V : V \rightarrow D^V$,

$f^E : E \rightarrow D^E$,

for data sets D^V and D^E.

In a directed graph:

Edge data: Mediates the channel between parent and child vertices.

Vertex data: State information, eg the yield δ.

For an option:

Edge data: The condition, rebate and exchange type functions.

(Vertex data: Just the maturity time.)

Examples:

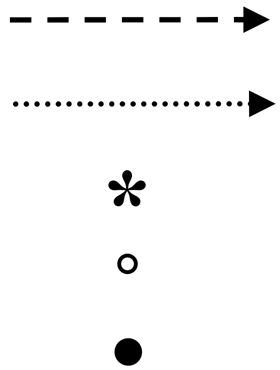
Indigo edge:

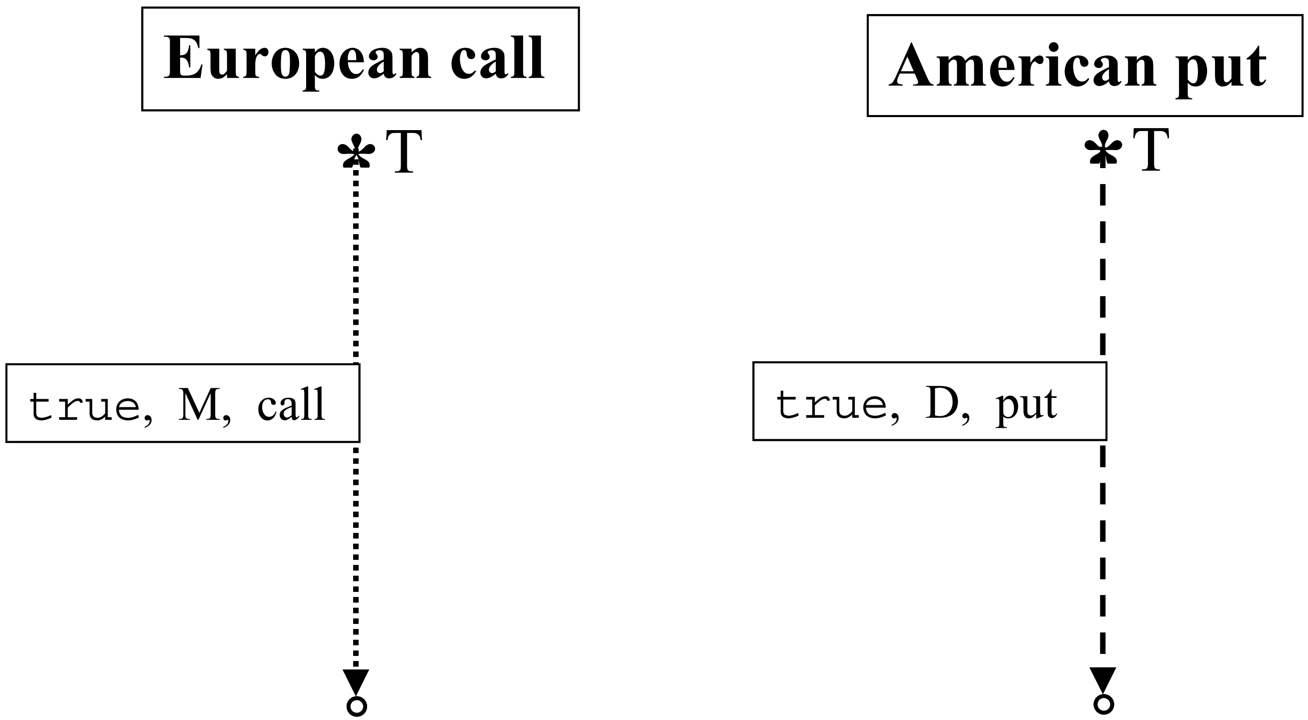
Turquoise edge:

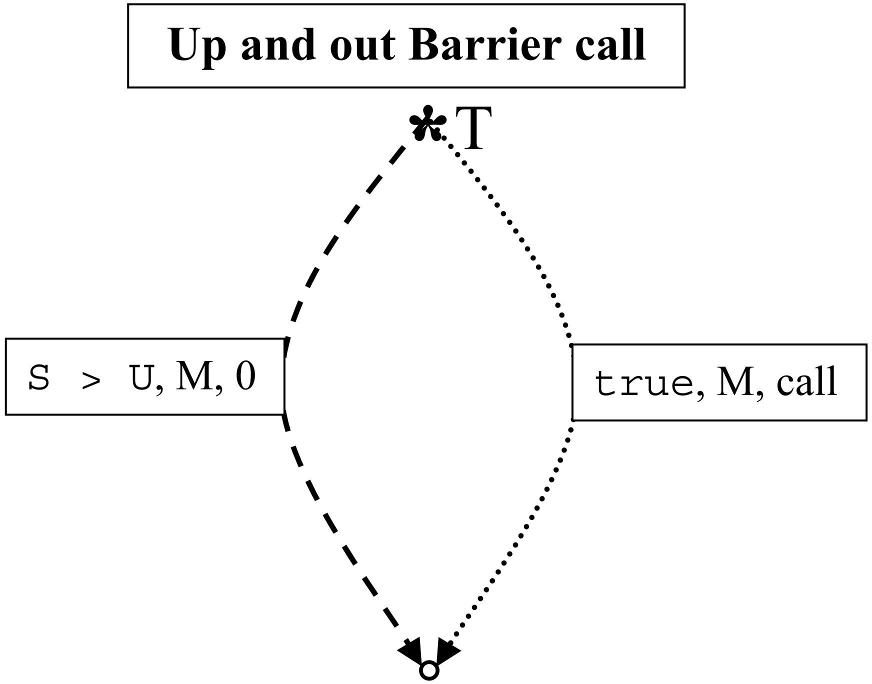
Root vertex:

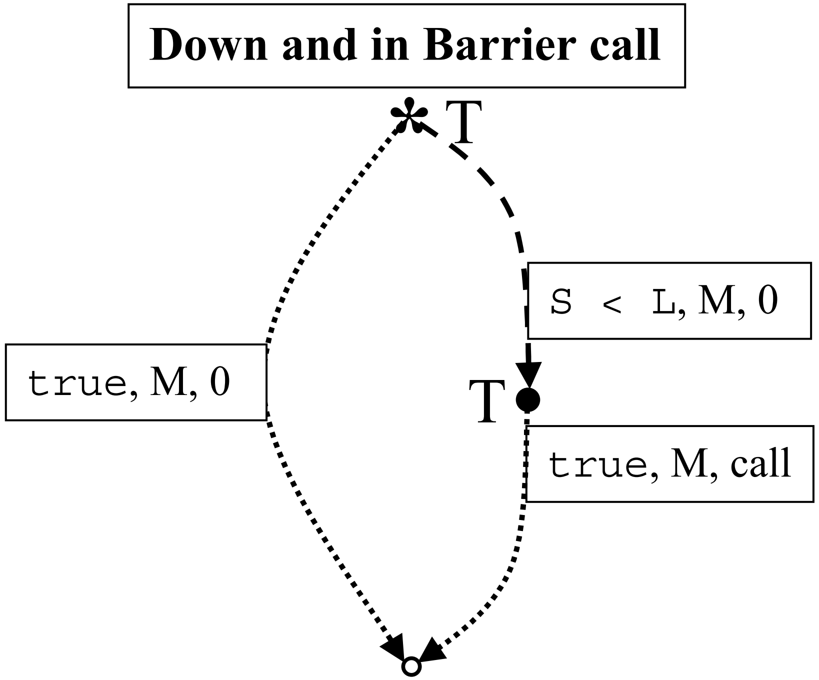
Terminal vertex:

Ordinary vertex:









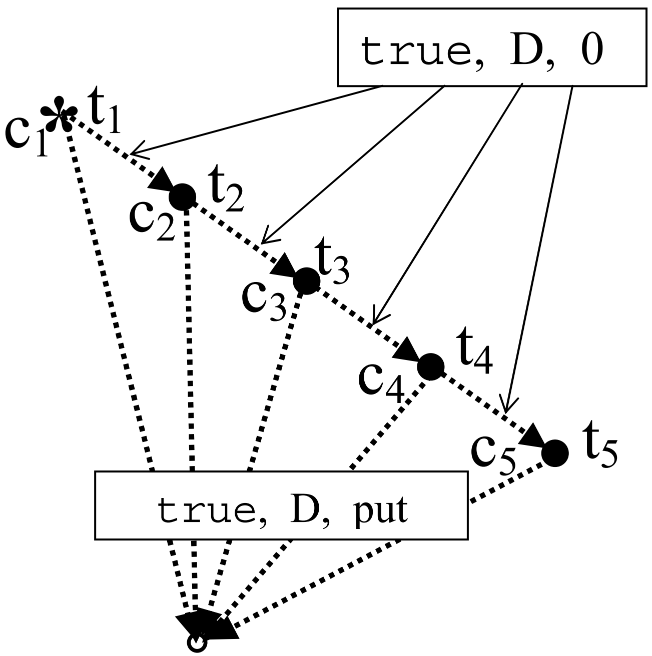
In-barrier options are structurally different to out-barrier options.

Bermudan put

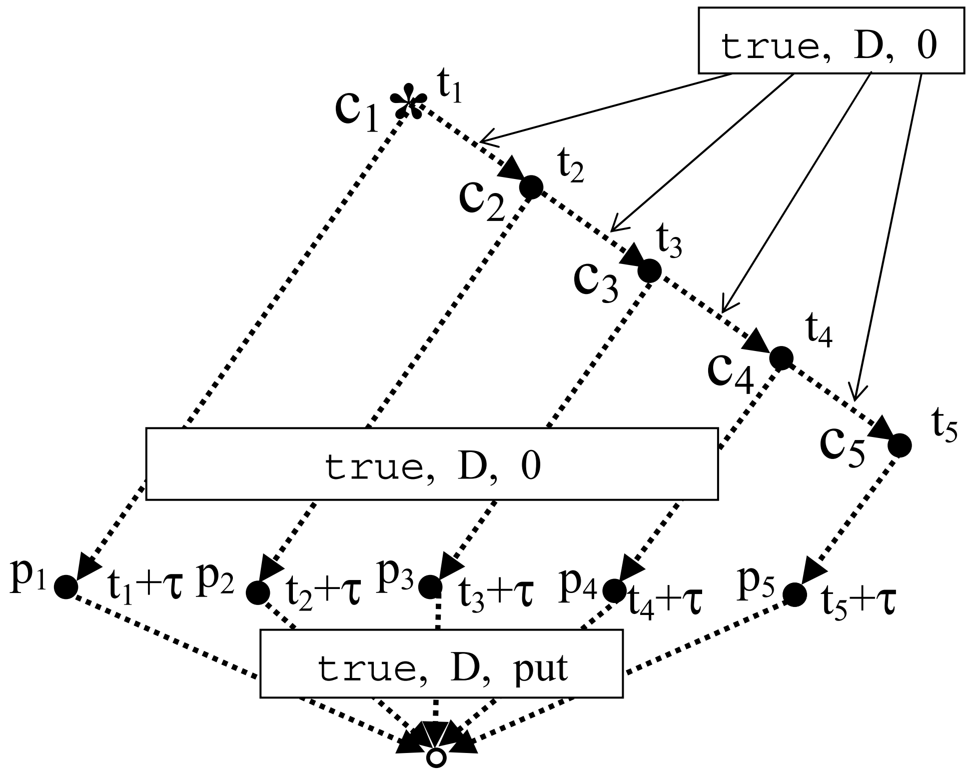
Represented as compound options.

Exercisable at $0 = t_0 < t_1 < \ldots < t_N = T_{max}$.

Ordinary



Reset interval of τ



Valuation algorithms

Underlying data structure: a graph.

Vertex objects representing options,

Edge objects representing exchange specifications.

Attach to each vertex a vertex-method object. Controls:

i) Times for which option values need to be constructed,

ii) Production of continuation values at that vertex,

ii) Comparison of values for each possible exchange.

Attach to each edge an edge-method object. Controls:

Mediation between its parent and child vertices.

Algorithm steps

Numerical methods differ, but two main stages.
May be trivial in a given method

i) Roll-forward: Generate sets of states, (may not proceed strictly forward).

ii) Roll-back: Computes sets of values

Roll-back at a given step has four stages:

i) Get states for that time
ii) Compute continuation values from option values at other slices
iii) Assemble non-continuation (exercise) values
iv) Compare values from continuation and non-continuation values to find option values

Algorithms

1) Traverse the graph, perhaps more than once, with some traversal method.

2) Maintain data. Two types:
    a) On the graph, ie on nodes and/or vertices,
    b) Independent to the graph.

For an option valuation algorithm:

Vertex data:	Construction times and values for each option,
Edge data:	Condition, rebate and exchange type values,
Independent data:	States for each construction time.

Option Valuation Algorithms

Construction times at a vertex, $\hat{T}_v$:

Times at which option values must be computed.

Construction times required at:

i) Discretised exchange times, $\tilde{T}_v$,

ii) Additional times as required by the algorithm.

Mesh times:

All times at which values must be constructed,

Contains all construction times for individual vertices.

Discrete Exchange Times

Given exchange times T_v, for $v \in V$, set

$$\overline{T}_v = \{ t \in T_v \mid t = t(v);\; t = t^s(u),\; u \in P^I(v);\; t = t(u),\; u \in P^T(v) \},$$

the start and end times in T_v.

For $v \in V$, a set of discrete exchange times for v with refinement $\varepsilon > 0$ is a set $\widetilde{T}_v = \{t_i\}_{i \in I_v} \subseteq T_v$, for some index set I_v, st

i) $\overline{T}_v \subseteq \widetilde{T}_v$,

ii) $\forall t \in T_v \; \exists \; t_i \in \widetilde{T}_v$ st $| t - t_i | < \varepsilon/2$.

A set of discrete exchange times is regular if

$\forall \; i \in I_v, \; \exists \; \Delta t > 0$ st $t_i = t_0 + k_i \Delta t$ for some t_0 and $0 \leq k_i \in \mathbb{Z}$.

Algorithm construction times

A method is long-step roll-back if
option values at time t_1 can be computed to within acceptable accuracy
directly from those at time $t_2 > t_1$,
for any t_2.

A method is short-step roll-back if
option values at time t_1 can be computed to within acceptable accuracy
from those at time $t_2 > t_1$,
only if t_2 is close enough to t_1.

ie, results are accurate to within δ only if $|t_2 - t_1| < \varepsilon$.

Algorithm construction times

Have a discrete exchange times $\tilde{T}_v$.
If the algorithm is short-step need to add algorithm construction times.

A set of construction times is $\hat{T}_v = \{t_i\}_{i \in I'_v} \subseteq [t^s(v), t(v)]$ st

i) $\overline{T}_v \subseteq \hat{T}_v$,

ii) $\forall t \in [t^s(v), t(v)]\ \exists\ t_i \in \hat{T}_v$ st $| t - t_i | < \varepsilon/2$.

If the algorithm is long-step roll-back set $\hat{T}_v = \tilde{T}_v$.

A set of construction times is regular if

$\forall\ i \in I'_v,\ \exists\ \Delta t > 0$ st $t_i = t_0 + k_i\Delta t$ for some t_0 and $0 \le k_i \in \mathbb{Z}$.

Mesh times

Mesh times: times on the graph when option values must be computed

Set $T = \bigcup_{v \in V \backslash \circ} \{t^s(v), t(v)\}$.

$\hat{T} = \{t_i\}_{i \in I}$, $I = \{0,\ldots,N\}$, is a set of mesh times with refinement $\varepsilon > 0$ if

i) $t_{i-1} < t_i$, $\forall\, i \in I \backslash \{0\}$,

ii) $| t_i - t_{i-1} | < \varepsilon$, $\forall\, i \in I \backslash \{0\}$,

iii) $\hat{T}_v \subseteq \hat{T}$ $\forall\, v \in V \backslash ^\circ$.

A set of mesh times $\hat{T}$ is regular if

$\forall\, i \in I$, $\exists\, \Delta t > 0$ st $t_i = k_i \Delta t$ for some t_0 and $0 \le k_i \in \mathbb{Z}$

(in practice, a multiple of a whole number of days) and complete if

$t_i = i \Delta t$ $\forall\, i \in I$.

Option Valuation Algorithms

Have found: i) Construction times $\hat{T}_v$ for each vertex,

ii) Mesh times $\hat{T}$ for the graph as whole.

Mesh times:

If algorithm is long-step roll-back assume $\hat{T}$ is complete,

If algorithm is short-step roll-back assume $\hat{T}$ is regular.

Algorithms have two phases:

Roll-forward, generating states at each construction time.

Roll-back, generating option values at each construction time.

Examples of a few algorithms by step-type

<table>
<tr><td colspan="2" rowspan="2">Algorithms
by step type</td><td colspan="2">Roll-forward</td></tr>
<tr><td>short-step</td><td>long-step</td></tr>
<tr><td rowspan="2">Roll-back</td><td>short-step</td><td>LRS lattice,
Interest rate MC</td><td>Backwards induction lattice,
Some interest rate MC,
PDE</td></tr>
<tr><td>long-step</td><td>Forwards induction lattice,
Plain asset MC</td><td>Direct integration methods,
Some asset MC,
Explicit solutions</td></tr>
</table>

MC:	Long-step forward if process amenable
Path-dependent MC:	Usually short-step forward
Interest rate MC:	Long-step back if can compute discount factor
PDE methods:	Short-step back, long-step forwards

Options with I-exchanges are short step back.

Vertex algorithm

At a vertex $v \in V$, receive a request for option values for time $t_i \leq t_v$.

0) If values not found for end time t_v, compute them:
   Ask each T-edge for its t_v values and condition,
   Compare them to get vertex v values for time t_v.
   (Comparison depends on edge exchange types.)
1) If values already found, return them.
2) Let $t_i < t_j$ be nearest future time at which values have been found.
   If long-step back is possible:
   Compute t_i values from t_j values and return them
   If short-step back is necessary:
   Compute values iteratively from t_j values back to t_i values.
   Return t_i values

Computing t_i values from t_{i+1} values

From t_{i+1} values compute continuation values at t_i,

Values at t_i if have not exchanged up to t_i,

but do so optimally thereafter.

If there are no I-edges:

Return continuation values.

If there are I-edges:

Ask each I-edge for its t_i values and condition,

Compare them with continuation values get values for time t_i.

(Comparison depends on edge exchange types.)

Edge algorithm

At an edge $e \in E$, receive a request (from e_p) for time $t_i \leq t_{e_c}$ values and condition.

1) Compute condition for time t_i.
2) Where condition is true,
   - Ask e_c for values at time t_i.
   - Compute rebate values.
   - Return condition, rebate + e_c values.

May be able to cache condition/rebate values it

1) Condition and rebate are time homogenous,
2) States are time homogenous.

Independent data

For mesh times construct sets of states: slices.

Write:

$\mathcal{S}_t$ for the (continuous) state space at time t.

$\mathcal{S} = \bigcup_{t \geq t^s(*)} \mathcal{S}_t$ for the full state space.

May have $\mathcal{S}_{t_1} \equiv \mathcal{S}_{t_2}$ for all t_1 and t_2.

Write $S_t \subseteq \mathcal{S}_t$ for a discrete set of states used by the algorithm at time t:

Construct option values at time t for states $s \in S_t$.

S_t is a slice at time t.

Slices

Slices have:

i) A geometry,

ii) A mechanism to evolve forward (and maybe back) through time.

The geometry:

Has dimension, in practice small,

Spatial arrangement, eg vector, array, icosahedral, etc

Evolutionary mechanism:

Based on continuous time process.

Identical geometries and processes can have different state evolution

(eg order of branching; different MC schemes.)

Slices

At each time, compute option values for a set of states: a slice.

A slice has a geometry, eg vector, array, hexagonal lattice, etc.

State manager object. Responsible for:

1) Constructing slices,
2) Giving access to objects that request them.

Has: roll-forward, roll-back and process objects.

Nodes in a slice carry three types of information:

States:	Controlled by the slice manager,
Method data:	Condition, rebate and option values,
Option data:	Temporary values used by numerical methods.

Components of an algorithm

An algorithm specifies:

1) The geometry and manner of evolution of slices,
2) The computation of continuation values,
3) The way in which values from disparate edges are compared to one another and the continuation values.

Conclusion

Graphs are a natural way to define options.

Valuation methods operate on these graphs.

Valuation methods can be generic:
Widely applicable to large classes of options.

Cost overhead?
Surprisingly small...