Pricing Cancellation Product

Lee, David

FinPricing

8 August 2022

Pricing Cancellation Product

David Lee

Abstract

This article describes a valuation framework to build most common kinds of cancellation schedules and cancellation evens. The model can price generic cancellation derivatives accurately. It is very useful for derivatives trading and risk management.

Key words: derivatives products, cancellable structured note, derivatives valuation

1 Introduction

During a lifetime of a structured product events will happen that affect its value. Generally, these can be grouped as follows:

On payment events, payments are made to one of the parties in the deal. Market observables that the payment depends on are fixed on a

fixing date, and the amount to be paid is accrued over the *accrual period*, specified by start and end dates. In practice the dates are rolled according to rolling conventions and holiday calendars used, so one refers to adjusted and unadjusted dates. The significance is that, intuitively, a payment is made for the period between *unadjusted start* date and *unadjusted end* date.

On cancellation events, the remaining part of the trade is potentially cancelled as a result of a trigger condition or an exercise option. Commonly there is a penalty or redemption payment associated. The decision to exercise is made, or the trigger condition is checked on *notice date*, and the cancellation takes effect on the *effective cancellation* date. What this means in practice will depend on the product definition. In most cases, once a cancellation occurred, no payments are made after the effective cancellation date. Similarly, to a fixing date, notice date is always adjusted, and similarly to start and end date, one can talk about adjusted or unadjusted effective dates.

Practitioners commonly think (and describe products) in terms of accrual periods, so that each payment is made for a period and a cancellation event cancels all periods starting from the next one. In contrast, generic products normally only refer to one date per event, (indeed, syntactically an event in a generic product definition is associated with a single date). In practice, the date chosen is usually the first date at which all the information is available to calculate the payment value or make a decision to cancel. Thus, fixing date is normally the event date for payment events and notice date is normally the event date for cancellation events.

This means that in practice, in generic product description it is not unusual to end up with a product definition where the effect of

cancellation is to cancel a payment that was already calculated (when fixing date for the cancellable leg happens to be before the notice date), or a payment after, but not immediately after the notice date event (this situation is quite common with products where a cancellable leg is fixed in arrears).

2 Product description language support

Within product description language all algorithms and methodology described below are available through the expression Cancellation, taking a variable number of parameters, depending on the number of cancellable legs in the product.

The first parameter is the cancellation condition. For trigger-type products this would be a Boolean expression. If the expression evaluates to a non-zero value, cancellation occurs at the event, if it evaluates to zero, cancellation does not occur.

As a special case, if the condition is just expression. the value of optimal cancellation strategy would be computed. This feature makes pricing callable products possible.

The second parameter is a penalty payment expression. This represents value of a penalty or a rebate paid if cancellation occurs.

The rest of the parameters are pairs of cancellable leg names and corresponding “first cancelled” dates.

3 Cancellation legs

Legs in a generic product are represented by columns of related events, and generally correspond to product legs as described in a term-sheet. The values of paying legs are total values of all payments on those legs (regardless of cancellation) and the value of a cancellation leg is the negated total value of cancelled flows on the paying legs.

This is a rather broad definition, covering both trigger-type products and callable products. In practice even for callable products the decision to exercise will depend on the current state of the market, and so these are often modeled by introducing some kind of *exercise boundary*, i.e. a function of market observables describing a multidimensional boundary beyond which it is optimal to exercise. This has the advantage of separating two problems - making a decision to exercise and calculating the value of the cancellation leg.

For the most part, the subject of this document is the later of the two, that is, the correct calculation of the value of a cancellation leg. There are special cases where the two can be merged to great advantage that cannot be ignored and these are considered below.

In general, there may be any number of cancellation legs in a product, and a cancellation leg will cancel a fixed number of other legs. Legs that can be cancelled as an effect of valuation of a cancellation leg will be referred to as *cancellable legs*. It is possible for a cancellation leg to be cancellable by another cancellation leg.

We will however assume that in cases where there are more than one cancellation legs cancelling same legs the term-sheet defines clearly an order of precedence between them (i.e. which decision is made first), and that if two cancellation legs cancel a single leg in common, then their

actions are mutually exclusive (i.e. cancellation can only occur on one of them).

These assumptions are not restrictive in practice as they will hold for any "reasonable" product, and even if they do not, it is always possible to split a single cancellation leg into a number of legs, each checking the same condition on same dates but affecting different cancellable legs. We will also assume that all legs cancellable by a particular cancellation leg appear to the left of it in the product definition. This makes computation more efficient and helps prevent errors in coding, and is again not restrictive as all valuation engines consider product legs in isolation.

Further, we will be concerned only with Bermudan trades, that is, we will assume that the decision to cancel the deal must be made on one of the set of pre-agreed dates. This type of products fit naturally within the structure of generic products definition language, and we are still to come across a realistic product that cannot be described in terms of a finite set of dates.

Finally, depending on whether the valuation environment is backwards looking (Monte Carlo) or forward looking (backwards induction engines), value of a cancellation leg is calculated differently and the two cases will be considered separately.

3.1 Cancellation and continuation values

When implementing optional cancellation using numerical methods it is common to compare the values of the product when cancellation occurs and when it does not. The two quantities are also useful in computing the value of a cancellation leg, it being the difference of the two. So we

define:

- *Cancellation value* of a product for a particular cancellation event date on a particular cancellation leg, is the value of cancellable legs if cancellation does occur on that event date and on that leg.

- *Continuation value* of a product for a particular cancellation leg is the total value of all legs cancellable by that cancellation leg if cancellation does not occur on that cancellation leg.

It can now be seen that the value of a cancellation leg is the negated expectation of a difference between continuation value and cancellation value for that cancellation leg and for the event date when cancellation occurred.

3.2 Cancellation schedules

A schedule in generic products is a collection of dates pertaining to a product leg. There will typically be several dates associated with each event, only one of which is actually used as an event date in the product definition. For a typical paying leg these would be:

- Unadjusted start and end date - the unadjusted start and end of the accrual period.

- Adjusted start and end date - the adjusted start and end of the accrual period.

- Fixing date - the event fixing date, always adjusted.

- Payment date - the date when the payment is made, always adjusted. These are built from a number of parameters.

The structure of the schedule for payment legs reflects the fact that each payment made on the leg pertains to a period. In contrast, a cancellation event is global, it pertains to the whole of the product duration and effectively divides the product schedule into two parts - time before cancellation, that is, a collection of events that will not be cancelled, and time after cancellation, that is, the collection of events that will be cancelled if cancellation does occur on the cancellation event. Further, depending of the nature of the product, there may be additional entries in the schedule for the penalty payment dates and accrual end date for product that pay accrued coupon on cancellation.

A general cancellation schedule would contain, for each event:

- Unadjusted effective date

- Notice date

- Penalty payment date

- Accrual end date

- First cancelled event dates

- Accrued event dates

When the product pays the last paid coupon pro-rated, this is the event date of the pro-rated payment. Again, there will be one of these for each cancellable leg. A simple way to calculate is as the event date before the first cancelled event.

4 Computing the value of a cancellation leg

We will now consider the process of computing the value of an arbitrary cancellation leg, and introduce notation that will be used in the rest of the document.

d_i	i-th event date
d_x	cancellation notice date
d_v	valuation date
d_e	last event date in the product
l_i	i-th cancellable leg
l_x	cancellation leg
$d_f(l_i)$	first cancelled date on the i-th cancellable leg
P	a penalty payment
$cont(l_1, l_2, ...)$	continuation value for legs 1, 2, ...
$canc(d_x, l_1, l_2, ...)$	cancellation value for date d_x and legs 1, 2, ...
$C(d_x)$	value of the cancellation leg if cancellation occurs on date
d_x $EC(d_x)$	value of the cancellation event
$VoP(l_i, d_j, d_k)$	value of payments on leg l_i, between dates d_j and d_k, including any payments on date d_j but not those on date d_k, deflated or inflated as appropriate
$Pm(l_i)$	value of a payment on leg l_i on event date when the function is evaluated

$V(l_i, d_j+)$	expected value of payments on leg i starting from but not including date d_j
$V(l_i, d_j)$	expected value of payments on leg i starting from and including date d_j [10]
$Lv(l_i, d_j)$	total value of payments on leg l_i up to but not including date d_j, inflated or deflated as appropriate
$D_f(d_i, d_j)$	discount factor between dates d_i and d_j
C	the set of indices of cancellable legs ($\{1, 2, ...\}$)
L	the set of names of cancellable legs ($\{l_i : i \in C\}$)

All the functions above are assumed to be evaluated at a particular event in the product definition, thus to be strict, they should all have an additional parameter denoting the evaluation event date. It will however always be clear from the context when the functions are being evaluated, so for clarity we omit the parameter here.

Now, if we are to evaluate continuation and cancellation values on the cancellation event, we have:

$$cont(L) = \sum_{i \in C}(Lv(li, df\ (li))\ +\ V\ (li, df\ (li)))\ +\ V\ (lx, dx+) \qquad (4.1)$$

$$canc(d_x, L) = P + \sum_{i \in C} Lv(li, df\ (li)) \qquad (4.2)$$

$$C(d_x) = P\ - \sum_{i \in C} V\ (li, df\ (li)) - V\ (l_x, d_x+) \qquad (4.3)$$

$$= canc(d_x, L) - cont(L) \qquad (4.4)$$

where the last term is added to continuation value to cater for any events that may have value on the cancellation leg after d_x.

4.1 Valuation Engines

A generic product is valued in the context of a *valuation engine*, an abstraction encapsulated a valuation methodology, evaluation of expressions, processing and collecting of valuation results. Within the library there are implementations of several valuation engines that can be broadly grouped, by the nature of the implemented methodology, into *forward looking*, implementing any backwards induction methodology, and *backwards looking*, implementing Monte Carlo simulations methodology.

The most significant difference in terms of valuation of generic products is in the order of valuation of ex- pressions. Forward looking valuation engines evaluate expressions left-to-right and bottom-to-top, thus making available information about events in the future; backwards looking valuation engines evaluate expressions left-to-right and top-to-bottom, thus making available information about events in the past (relative to the event currently being evaluated).

In both cases each product leg is considered in isolation, that is, a valuation engine will evaluate, maintain and store values of separate legs without regard for other legs in the product definition. Thus, the valuation will consist of evaluating expressions in the prescribed order, maintaining model state variables between "moves" from one event date to the next, and collecting the values of legs into a report (the value of a leg being a total value of all payments on that leg).

The interface between a valuation engine and a generic product instance is a collection of data with the implied agreement on ownership of data and maintenance of values. Regardless of the nature of the valuation engine, the values maintained are well defined at all times, and available during valuation of expressions. For the purposes of this document, only the following groups of values are relevant:

- Accumulated leg values

For each leg in the product description there is a single number used to store the value accumulated so far. The values stored here have the property that if they are read at any point (at any event) and the value returned as a payment, the discounted value is the same as the total value of the events on the corresponding leg evaluated so far.

Note that this means that the meaning of this value is different in forward looking and backwards looking engines. Using notation described above, if k-th entry from this collection is evaluated at event d_i, in forward looking engines this corresponds to $V(l_k, d_i+)$, that is, conditional expected value of all payment on the leg starting from but not including the current event.

In backwards looking engines, the value corresponds to $Lv(l_k, d_i)$, that is, accumulated values of all payments on the leg on the current path, from the start to the current event, not including the current event.

- Current leg values

For each leg in the product description there is again a single number here, representing the value of any payments on the leg at the current

event level. In the notation above, $Pm(l_i)$. Note that evaluation of expressions left-to-right means that only values of legs to the left of the current one is stored.

• Register values

This is storage for values used internally by the language interpreter implementation. They are written, updated and read by the implementation of expressions, and have the property that their present value is maintained by valuation engines. This means that forward looking engines maintain the conditional expectation of values stored here, and backwards looking ones inflate them by the ratio of numeraire as the valuation moves along.

These values are in fact an implementation artifact and discussed in more detail elsewhere in the text. For the time being it is sufficient to know that they provide means of implementing function $VoP\ (l_i, d_j, d_k)$ described above. In forward looking valuation engines both date parameters must be on or after the current event date, and the value is in fact conditional expectation of the payments between them.

In backwards looking engines, both date parameters must be on or before the current event date, and the value is the total value of payments made on the current path between the two dates. Note that the "register values" are more general than this, and only described by the property above - they are used for different purposes elsewhere.

• Discount factors

Regardless of the methodology used, it is always possible to inflate or deflate a value during valuation.

To summarize, on each event date d_m, the following values are available:

- in forward looking valuation environment:
  - $V(l_i, d_j+)$
  - $VoP(l_i, d_j, d_k)$
  - $Pm(l_i)$
  - $D_f(d_m, d_n)$

  with $d_i, d_j, d_k >= d_m$ and l_i to the left of the current leg.

- in the backwards looking environment:
  - $Lv(l_i, d_j)$
  - $V oP(l_i, d_j, d_k)$
  - $Pm(l_i)$
  - $D_f(d_m, d_n)$

  with $d_j, d_k <= d_m$ and l_i to the left of the current leg.

Differences in methodologies mean that implementation of the computations described above will be significantly different in different valuation environments. However, with the analysis below it is possible to automate the process (almost) completely.

4.2 Relative position of dates

The restrictions on date parameters to functions described above imply

that calculations of cancellation leg values have to be considered carefully and that the algorithm used will depend on the position of "first cancelled" dates in relation to cancellation notice date.

We will refer to the situation when the first cancelled date is before the notice date ($d_f(i) < d_x$) as *past cancellation*, and when the first cancelled date is after or on the notice date ($d_f(i) >= d_x$)as *future cancellation*. It is important to remember that these terms do not refer to payment dates, but to event dates (usually fixing dates), the actual cash flows being cancelled are **always** after the notice date (even if they are fixed before it).

4.3 Cancellation leg value in forward looking environment

In forward looking environment, computing cancellation leg value in the future cancellation case is straightforward, all the functions required in (4.2) are readily available.

For past cancellation, we use the fact that the value of a leg is propagated as a conditional expectation of discounted accumulated values of payments made in the expressions evaluated so far, and split the computation into two parts.

Consider the value of a cancellation event in a product description. On points in the underlying grid in the model state space where the cancellation condition is satisfied it is given by (4.3), and where no cancellation occurs it is 0. In effect, we are calculating the expectation of the value with respect to the probability that cancellation occurs.

To extend this to the case of past cancellation, let $EC_1(d_x)$ denote the

expected value of events on the cancellation leg before the notice date, and $EC_2(d_x)$ of those on or after the notice date. We can easily evaluate the value of EC_2 on the cancellation event

Note that we only take into account the events on cancellable legs from and including the cancellation event date, thus staying within the limitations described in section 4.1. Once we reach the valuation date through backwards induction, the value of the cancellation event will then indeed be the conditional expectation (w.r.t. martingale measure) of the discounted expectation (w.r.t. probability of cancellation) of the event value.

Now, given the conditional probability of cancellation at the first cancelled event($Pr(cancellation)$), we could for the first cancellable leg (at the event $d_f(1)$) compute

$$EC_1(d_x) = Pr(cancellation) * (-VoP(l_1, d_f(1), d_x))$$

But,

$$Pr(cancellation) = \mathrm{E}(\varphi_{CCO}|F_{d_f})$$

where φ_{CCO} is the characteristic function of *CancellationCondition*().

Remembering that it is a property of registers that in forward looking valuation engines they contain conditional expectations of discounted values stored in them as valuation propagates backwards in time, we can compute the value of φ_{CCO} at the cancellation date, store it in a register and retrieve it at the first cancelled date to get:

$$\mathrm{E}(D_f(d_f(1), d_x) * \varphi_{CCO}|F_{d_f})$$

Dividing this by $D_f(d_f(1), d_x)$ finally gives us $Pr(cancellation)$.

This process should of course be applied for each cancellable leg whose first cancelled date is before the notice date.

In summary, to compute the value of a cancellation event in a forward looking environment when some of the first cancelled dates are before the cancellation notice date:

- At the cancellation event compute the value of CancellationCondition() predicate. If it is true, store 1 in a register, compute

  $P - \sum_{i \in C} V(l_i, max(d_x, d_f(l_i))) - V(l_x, d_x+)$ and return it as the value of the event. If it is false store 0 in the register, and return 0 as the value of the event.

- For each cancellable leg i for which $d_f(i) \prec d_x$, on each event date in the range $[d_f(i), d_x)$ retrieve the value of the register, divide it by $D_f(d_f(1), d_x)$ and multiply it by $-V\,oP(l_i, d_f(1), d_x)$ and return it as the value of the event.

4.3.1 Optional cancellation in forward looking environment

Calculating the value of optional cancellation events in forward looking environments is particularly simple. The basic idea is to compute cancellation and continuation values and if cancellation value is the largest of the two, cancel.

This can be rearranged to yield:

$$EC(d_x) = Max(canc(d_x, L) - cont(L), 0)$$
$$= Max(C(d_x), 0)$$

In the case of future valuation, this can be computed on the cancellation event date. In the case of past cancellation, the only term in the equation that must be computed on the cancellation date is the penalty payment (since it may depend on the current state of the market), everything else may be computed on the earliest first cancellation date. So, we can compute the value of the penalty payment on the cancellation date, store it in a register, and compute the value of *EC* on the earliest first cancellation date, using the value in the register instead of the penalty payment.

4.4 Cancellation leg value in backwards looking environment

In backwards looking environment we make extensive use of (4.4), calculating cancellation and continuation value for cancellation events.

As with valuation in forward looking environment, we make use of the properties of registers and leg values and compute values at those events at which all the information needed is available.

To calculate continuation value we make use of the fact that at the last event date in the product (4.1) becomes:

$$cont(L) = \sum_{i \in C}(Lv(li, de) + Pm(li))$$

All of the terms here are readily available in a backwards looking environment. Cancellation value presents more of a challenge. Because of restrictions described in section 4.1, we have to split the computation of

(4.2) into parts according to the events where the data needed becomes available. This can be done as follows:

- For legs that are past cancelled the term $Lv(l_i, d_f(l_i))$ can be computed on the cancellation date as $Lv(l_i, d_x) - VoP(l_i, d_f(l_i), d_x)$

- For legs where first cancelled date is the cancellation date we have, at can- cellation event: $Lv(l_i, d_f(l_i)) = Lv(l_i, d_x)$

- For legs where first cancelled date is after the cancellation date we can compute:

  - At the cancellation event: $Lv(l_i, d_x) + Pm(l_i)$

  - At every event after the cancellation event and before or on first cancelled date we compute a "correction" to the cancellation value: $Pm(l_i)$

The penalty payment term can be computed at the cancellation event. The sum of all the values thus computed gives the cancellation value. In implementation, we can return these values from expressions at appropriate events if *cancellation condition* was true on the cancellation event, and the negated continuation value as the value of the last event on the cancellation leg. This way, for a single path, the total value of the cancellation leg will be 0 if no cancellation occurred and equal to (4.4) if it did, as required.

4.4.1 Optional cancellation in backwards looking environment

At present no algorithms are implemented for valuing generic product

with optional cancellation features in backwards looking environment. However, any future implementation is likely to be using a sample of cancellation and continuation values, and functionality is in place to collect and store this data.

5 Notes on implementation

5.1 Expressions

The implementation follows the method described in the previous section closely, providing internal expressions (i.e. not creatable by users) implementing every step of the computation.

Expressions Payment Value, Value Of Payments and Value are provided to users corresponding to functions Pm, VoP and V described above.

Further, a number of *internal* expressions are implemented as private to the class implementing the Cancellation expression. These are used by other ex- pressions, but not directly available to users. So, expressions Continuation Value and Cancellation Value Correction compute continuation value and corrections to cancellation value (section 4.4).

Where communication between expressions on different event dates using registers is required, the expressions are often implemented in pairs. For com- putting and retrieving the value of penalty payment as described in section 4.3.1, expressions Past Penalty and Past Cancellation Penalty Value are implemented, the first one calculating the value and storing it into a register, and the second one retrieving it as necessary. Similarly

expressions Past Cancellation Condition and Past Cancellation implement calculation and storing of the characteristic function of cancellation condition and calculating exercise probability and using it to calculate the cancellation value respectively (section 4.3).

To support the methodology above, extensive use is made of the ability to transform the product definition during compilation. All of the above expressions are inserted into the appropriate positions according to the procedures described above. Expression Cancellation is under certain circumstances (in forward looking environment, section 4.3) itself replaced with corresponding If or Max expression.

5.2 Registers

The interface between valuation engines and a generic product during valuation is encapsulated into a class Workspace. Product leg values and register values are stored in a vector of known Values. As valuation engines maintain the values in registers, it is crucial to be able to minimize the number of them.

This is achieved using a variant of register allocation algorithm (used in com- piler design). The algorithm is demand driven, that is, the objects that use a register need to request it, specifying the required lifetime - the period during which they will be using it. The allocation algorithm is run after all expressions requests are complete, a minimal number of registers is computed and registers are allocated to requesting expressions.

The algorithm guarantees exclusive use of a register during the lifetime requested, including the start date but not including the end date. The implementation of the algorithm is very well commented and

the reader is referred to the source code for further details.

5.3 Cancellation information

A massive amount of data about cancellation is potentially collected during valuation. This is useful for implementations of optional cancellation pricing in Monte Carlo environment and for providing detailed valuation reports. The level of detail is specified by valuation engines at runtime, but at least information about cancellation on the current path is always collected - and used internally. Historic valuation is treated as a single path during its run, and the data collected is used as a base set of values for all Monte Carlo paths.

The logic and data storage are implemented in the class Cancellation Information and instance of which is a member of the Workspace object. The data structure is fairly complicated, reflecting the fact that there is a lot of data collected and saving space is a priority.

References

[1] Ammann, M, Kind, A., and Wilde, C. (2008) Simulation-based pricing of convertible bonds. *Journal of empirical finance*, 15: 310-331.

[2] Brace, A., D. Gatarek, and M. Musiela. "The market model of interest rate dynamics." Mathematical Finance, Vol. 7, No. 4 (1997), pp. 127-155.

[3] Carr, P. and Linetsky, V. (2006) A jump to default extended CEV model: an application of Bessel processes. *Finance and Stochastics* 10: 303-330.

[4] FinPricing valuation, https://finpricing.com/lib/EqConvertible.html 2021.

[5] Gandhi, S. and P. Hunt. “Numerical option pricing using conditioned diffusions,” Mathematics of Derivative Securities, Cambridge University Press, Cambridge, 1997.

[6] Martzoukos, H., and L. Trigeorgis. “Real (investment) options with multiple sources of rare events.” European Journal of Operational Research, 136 (2002), 696-706.

[7] Piterbarg, V. “A Practitioner’s guide to pricing and hedging callable LIBOR exotics in LIBOR Market Models.” SSRN Working paper, 2003.