QL MC / Early Exercise

Peter Caspers

IKB

February 26, 2014

Terminology I

- Exercise Value = NPV in case of exercise
- Continuation Value = NPV in case of no exercise
- Control Value = NPV of control variate (optional)
- State = State variable(s) of model
- Basis System = Functions of model's state variables

Terminology II

For an exercise time $i \geq 1$, Cumulated Cashflows := NPV of cashflows between i and $i+1$ where $i+1$ may stand for ∞ and $i=0$ is reserved for the period [evaldate, first exercise].
Here "between" is not referring to the payment time but to the the cashflow belonging to the exercise into part for exercise i.

NPV are deflated NPVs always !

We will have to compare NPVs accross different times. This is possible for deflated NPVs, i.e. NPVs divded by the numeraire.
Deflated NPVs are "timeless" (because the transformation to an absolute NPV as of time t is done by multiplication with the numeraire N(t)).

Exercise Strategy

- Exercise if and only if Exercise Value $\geq$ Continuation Value
- Continuation Value is not known
- Estimate Continuation Value as a function of the Basis System
- Exercise is never optimal, therefore option value is underestimated

Side note: Control Variate

Choose product C

- with high correlation to product V to price
- known analytical value

and use

$$E(V) = E(V + \gamma(C - E(C))) \tag{1}$$

Then there is some optimal γ which minimizes the variance of the rhs argument. C is called Control Variate. By renormalization we can assume $\gamma = -1$.

Basis Systems

Typical examples include

- polynomials (of different type) in model's state variables or
- in financial quantities like forward rates, swap rates

Strategies

- Longstaff Schwartz: Continuation Value = Linear function in Basis System
- Parametric Exercise: Exercise = Arbitrary Indicator function in Basis System

Strategy building

Generate m training paths and collect data on the training paths:

- cumulated cash flows
- exercise values ($-\infty$ if “additional” grid point only)
- basis system values
- control values

Longstaff Schwartz

Backward iteration through exercise times, in each step i:

- Least square linear regression of cumulated cash flows on basis system values
- Estimate continuation value by regression and decide whether to exercise (based on information available up to exercise time)
- Update $i - 1$ cumulated cash flows by adding exercise value (if exercised) or cumulated cashflows $\geq i$ (if not exercised, this is not the estimated continuation value, but the actual path value!)

Note that averaging at $i = 0$ already gives the products NPV, because then the paths already look like as if generated with the LS strategy.

Parametric exercise

For a fixed parameter set

- Apply a generic exercise trigger depending on the parameters
- Add exercise value or cumulated cashflows depending on the decision

Optimize the parameters to maximaize the mean of the realized NPV over the training paths.

NodeData

Represents a "node" (exercise time, path). We will have number of paths times exercise times nodes in the end.

```
    struct NodeData {
        Real exerciseValue;
        Real cumulatedCashFlows;
        std::vector<Real> values;
        Real controlValue;
        bool isValid;
    };
```

`values` are the basis system values, the remaining member variables are self explaining.

genericLongstaffSchwartzRegression

Computes least square linear regression coefficients for the basis system.

```
    Real genericLongstaffSchwartzRegression(
                std::vector<std::vector<NodeData> >& simulationData,
                std::vector<std::vector<Real> >& basisCoefficients)
```

`simulationData` should contain exercise times $+ 1$ vectors of size $m =$ number of paths. The first entry in simulation data corresponds to the case $i = 0$ above. The return value is the (deflated !) product NPV (though evaluated not using variance reduction by control variate, this is only used for the regression itself). The coefficients are written to the second input parameter.

ParametricExercise

Base class for arbitrary parametric exercise strategies.

```cpp
class ParametricExercise {
  public:
  virtual ~ParametricExercise() {}
  // possibly different for each exercise
  virtual std::vector<Size> numberOfVariables() const = 0;
  virtual std::vector<Size> numberOfParameters() const = 0;
  virtual bool exercise(Size exerciseNumber,
     const std::vector<Real>& parameters,
     const std::vector<Real>& variables) const = 0;
  virtual void guess(Size exerciseNumber,
     std::vector<Real>& parameters) const = 0;
};
```

genericEarlyExerciseOptimization

Computes an optimal parameteric exercise strategy

```
    Real genericEarlyExerciseOptimization(
        std::vector<std::vector<NodeData> >& simulationData,
        const ParametricExercise& exercise,
        std::vector<std::vector<Real> >& parameters,
        const EndCriteria& endCriteria,
        OptimizationMethod& method);
```

Adding non exercise nodes

One can add nodes where no exercise is possible in order to be able to approximate the continuation NPV by regression for PFE, CVA simulations.
For this we can just set the `exerciseValue` to $-\infty$ at the corresponding nodes.

Gaussian1dMcSwaptionEngine

Can be copied from Gaussian1dSwaptionEngine and then transformed to a MC engine, providing an additional method to approximate NPVs by regression for usage in PFE, CVA simulations.
Working on the generic model process, it can then be used for Hull White 1F and Markov Functional without any further effort.

Thank you, Questions ?