Markov Functional Model

Peter Caspers

IKB

November 13, 2013

Disclaimer

Table of contents

Term Sheet

Consider the following interest rate swap, with 10y maturity.

- We receive yearly coupons of type EUR CMS 10y
- We pay Euribor 6m $+$ 26.7294bp
- We are short a bermudan yearly call right

What is a suitable way to price this deal ?

Model requirements

A good start[1] would be to use a model that

- prices the underlying CMS coupons consistently with given swaption volatility smiles
- can in addition be calibrated to swaptions representing the call right (say for the moment to atm coterminals)
- gives us control over intertemporal correlations

A plain Hull White model fulfills $\#2$ and $\#3$ but clearly fails to meet $\#1$.

[1] one important requirement - the decorrelation of Euribor6m and CMS10y rates - is missing here, and actually not satisfied by the Markov model

The markov model approach

This is where the Markov Functional Model jumps in. Before going into details on how it works, we give an example in terms of QuantLib Code. Let's suppose you have already constructed a

```
Handle<YieldTermStructure> yts
```

representing a 6m swap curve and a

```
Handle<SwaptionVolatilityStructure> swaptionVol
```

representing a swaption volatility cube (suitable for CMS coupons pricing).

Model construction

We create a markov model instance as follows

```
boost::shared_ptr<MarkovFunctional> markov(
         new MarkovFunctional(yts,reversion,sigmaSteps,sigma,
                              swaptionVol,cmsFixingDates,
                              cmsTenors,swapIndexBase));
```

with a mean reversion

```
Real reversion
```

controlling intertemporal correlations (see below), a piecewise volatility (for the coterminal calibraiton) given by

```
std::vector<Date> sigmaSteps
std::vector<Real> sigma
```

Model construction (ctd)

our structured coupon fixing dates and tenors

```
std::vector<Date> cmsFixingDates
std::vector<Period> cmsTenors
```

and a swap index

```
boost::shared_ptr<SwapIndex> swapIndexBase
```

codifying the conventions of our cms coupons.

Model calibration

The calibration to the constant maturity swaption smiles is done automagically (see below). The calibration to the coterminal swaptions is done as usual by defining a calibration basket

```
std::vector<boost::shared_ptr<CalibrationHelper> >
                                                coterminalHelpers
```

and then calibrating the model with

```
markov->calibrate(coterminalHelpers,optimizer,endCriteria)
```

where the sigmaSteps are the coterminals' expiry dates. Note that $n+1$ volatilities are needed for n options with the first volatility kept fixed during calibration (which should become clearer later on).

Instrument

Our callable swap is represented as two instruments, the swap without the call right and the call right itself. The former can be constructed as

```
boost::shared_ptr<FloatFloatSwap> swap(new FloatFloatSwap( ... ));
```

and the latter as a swaption

```
boost::shared_ptr<FloatFloatSwap>
                            underlying(new FloatFloatSwap( ... ));

boost::shared_ptr<Exercise>
                    exercise(new BermudanExercise(exerciseDates));

boost::shared_ptr<FloatFloatSwaption>
           swaption(new FloatFloatSwaption(underlying, exercise));
```

Pricing Engine

To do the actual pricing we need a suitable pricing engine which can be constructed by

```
boost::shared_ptr<PricingEngine>
          engine(new Gaussian1dFloatFloatSwaptionEngine(markov));
```

and assigned to our instrument by means of

```
swaption->setPricingEngine(engine);
```

which then allows to extract the dirty npv

```
Real npv = swaption->NPV();
```

The npv of the swap on the other hand can be retrieved with standard cms coupon pricers implementing a replication in some swap rate model [2].

[2] this is slightly different from pricing the full structured swap in the markov model, since it does not give exactly the same price as some semianalytical model for theoretical as well as numerical differences

A full example: Market Data

```
#include <ql/quantlib.hpp>
using namespace QuantLib;

int main(int, char * []) {

    try {

        Date refDate(13, November, 2013);
        Date settlDate = TARGET().advance(refDate, 2 * Days);
        Settings::instance().evaluationDate() = refDate;

        Handle<Quote> rateLevel(new SimpleQuote(0.03));
        Handle<YieldTermStructure> yts(
            new FlatForward(refDate, rateLevel, Actual365Fixed()));

        boost::shared_ptr<IborIndex> iborIndex(new Euribor(6 * Months, yts));
        boost::shared_ptr<SwapIndex> swapIndex(
            new EuriborSwapIsdaFixA(10 * Years, yts));

        iborIndex->addFixing(refDate, 0.0200);
        swapIndex->addFixing(refDate, 0.0315);

        Handle<Quote> volatilityLevel(new SimpleQuote(0.30));
        Handle<SwaptionVolatilityStructure> swaptionVol(
            new ConstantSwaptionVolatility(refDate, TARGET(), Following,
                                           volatilityLevel, Actual365Fixed()));
```

A full example: Cms Swap and Exercise Schedule

```
Date termDate = TARGET().advance(settlDate, 10 * Years);

Schedule sched1(settlDate, termDate, 1 * Years, TARGET(),
                ModifiedFollowing, ModifiedFollowing,
                DateGeneration::Forward, false);
Schedule sched2(settlDate, termDate, 6 * Months, TARGET(),
                ModifiedFollowing, ModifiedFollowing,
                DateGeneration::Forward, false);

Real nominal = 100000.0;
boost::shared_ptr<FloatFloatSwap> cmsswap(new FloatFloatSwap(
    VanillaSwap::Payer, nominal, nominal, sched1, swapIndex,
    Thirty360(), sched2, iborIndex, Actual360(),
    false,false,1.0,0.0,Null<Real>(),Null<Real>(),1.0,0.00267294));

std::vector<Date> exerciseDates;
std::vector<Date> sigmaSteps;
std::vector<Real> sigma;

sigma.push_back(0.01);
for (Size i = 1; i < sched1.size() - 1; i++) {
    exerciseDates.push_back(swapIndex->fixingDate(sched1[i]));
    sigmaSteps.push_back(exerciseDates.back());
    sigma.push_back(0.01);
}
```

A full example: Call Right

```
boost::shared_ptr<Exercise> exercise(
    new BermudanExercise(exerciseDates));
boost::shared_ptr<FloatFloatSwaption> callRight(
    new FloatFloatSwaption(cmsswap, exercise));

std::vector<Date> cmsFixingDates(exerciseDates);
std::vector<Period> cmsTenors(exerciseDates.size(), 10 * Years);
```

A full example: Models and Engines

```
Handle<Quote> reversionLevel(new SimpleQuote(0.02));

boost::shared_ptr<NumericHaganPricer> haganPricer(
    new NumericHaganPricer(swaptionVol,
                           GFunctionFactory::NonParallelShifts,
                           reversionLevel));
setCouponPricer(cmsswap->leg(0), haganPricer);

boost::shared_ptr<MarkovFunctional> mf(new MarkovFunctional(
    yts, reversionLevel->value(), sigmaSteps, sigma, swaptionVol,
    cmsFixingDates, cmsTenors, swapIndex));

boost::shared_ptr<Gaussian1dFloatFloatSwaptionEngine> floatEngine(
    new Gaussian1dFloatFloatSwaptionEngine(mf));

callRight->setPricingEngine(floatEngine);
```

A full example: Calibration Basket

```
boost::shared_ptr<SwapIndex> swapBase(
    new EuriborSwapIsdaFixA(30 * Years, yts));

std::vector<boost::shared_ptr<CalibrationHelper> > basket =
    callRight->calibrationBasket(swapBase, *swaptionVol,
                                 BasketGeneratingEngine::Naive);

boost::shared_ptr<Gaussian1dSwaptionEngine> stdEngine(
    new Gaussian1dSwaptionEngine(mf));

for (Size i = 0; i < basket.size(); i++)
    basket[i]->setPricingEngine(stdEngine);
```

A full example: Model Calibration and Pricing

```
        LevenbergMarquardt opt;
        EndCriteria ec(2000, 500, 1E-8, 1E-8, 1E-8);
        mf->calibrate(basket, opt, ec);

        std::cout << "model vol & swaption market & swaption model \\\\" << std::endl;
        for (Size i = 0; i < basket.size(); i++) {
            std::cout << mf->volatility()[i] << " & "
                      << basket[i]->marketValue() << " & "
                      << basket[i]->modelValue() << " \\\\" << std::endl;
        }
        std::cout << mf->volatility().back() << std::endl;

        Real analyticSwapNpv = CashFlows::npv(cmsswap->leg(1), **yts, false) -
                               CashFlows::npv(cmsswap->leg(0), **yts, false);
        Real callRightNpv = callRight->NPV();
        Real firstCouponNpv = - cmsswap->leg(0)[0]->amount() * yts->discount(cmsswap->leg(0)[0]->date()) +
                              cmsswap->leg(1)[0]->amount() * yts->discount(cmsswap->leg(1)[0]->date());
        Real underlyingNpv = callRight->result<Real>("underlyingValue") + firstCouponNpv;

        std::cout << "Swap Npv (Hagan)     & " << analyticSwapNpv << "\\\\" << std::endl;
        std::cout << "Call Right Npv (MF)  & " << callRightNpv << "\\\\" << std::endl;
        std::cout << "Underlying Npv (MF)  & " << underlyingNpv << "\\\\" << std::endl;
        std::cout << "Model trace : " << std::endl << mf->modelOutputs() << std::endl;
    }
    catch (std::exception &e) {
        std::cerr << e.what() << std::endl;
        return 1;
    }
}
```

A full example: Model Calibration (Coterminals)

model vol	swaption market	swaption model
0.01	0.0273707	0.0273706
0.0107835	0.0337373	0.0337373
0.0106409	0.0354739	0.0354741
0.0109936	0.0344716	0.0344715
0.0109297	0.0315097	0.0315096
0.0111361	0.0270958	0.0270957
0.0111917	0.0215014	0.0215015
0.0112365	0.0150493	0.0150493
0.0113241	0.00783272	0.00783278
0.00998595		

A full example: Pricing

The expectation is to get a similar price of the underlying cms swap both in the Markov and the replication model, since both are consistent with the input swaption smile. Note that the underlying of the swaption is receiving the cms side while the cms swap is paying.

Swap Npv (Hagan)	-0.00
Call Right Npv (MF)	604.50
Underlying Npv (MF)	1.00

The match is very close (0.1 bp times the nominal). It should be noted that from theory we can not even expect a perfect match since the swap rate dynamics is not the same in the Hagan model and the Markov model respectively.

A full example: Benchmarking against Hull White

We change the code to replace the Markov model with a Hull White model. This is easily done by

```
        std::vector<Date> sigmaSteps2(sigmaSteps.begin(), sigmaSteps.end() - 1);
        std::vector<Real> sigma2(sigma.begin(), sigma.end() - 1);
        boost::shared_ptr<Gsr> gsr(new Gsr(yts,sigmaSteps2,sigma2,reversionLevel->value()));
```

and replacing `mf` by `gsr` (we could have use a generic name of course...). The calibration now reads

```
        gsr->calibrate(basket, opt, ec, Constraint(), std::vector<Real>(), model->FixedReversions());
```

or also

```
        gsr->calibrateVolatilitiesIterative(basket, opt, ec);
```

A full example: Pricing in the Hull White model

The fit to the coterminals is exact, just as above (with different model volatilities of course). The pricing compares ot the Markov model as follows:

Swap Npv (Hagan)	-0.00	-0.00
	Markov	Hull White
Call Right Npv	604.50	1012
Underlying Npv	1.00	657.81

The underlying and the option are drastically overpriced (both) by over $60bp$ in the Hull White model.

A full example: A more realistic smile surface

Swaption smiles are far from being flat. We test the model with a SABR volatility cube (with constant parameters $\alpha = 0.15$, $\beta = 0.80$, $\nu = 0.20$, $\rho = -0.30$) by setting

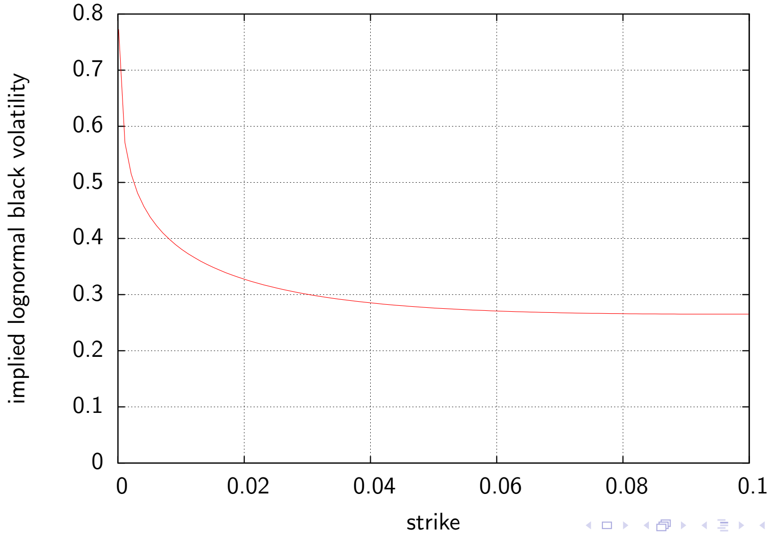
```
    Handle<SwaptionVolatilityStructure> swaptionVol(
        new SingleSabrSwaptionVolatility(refDate, TARGET(), Following, 0.15,
                                         0.80, -0.30, 0.20,
                                         Actual365Fixed(), swapIndex));
```

Swap Npv (Hagan)	246.00	246.00
	Markov	Hull White
Call Right Npv	696.96	1004.79
Underlying Npv	259.96	648.26

Again the underlying and the option is overpriced (by $40bp$ resp. $53bp$) in the Hull White model. In the Markov model the fit is still good ($1.4bp$ underlying price difference).

A full example: A more realistic smile surface ctd

Example smile 10y into 10y, $\alpha = 0.15$, $\beta = 0.80$, $\nu = 0.20$, $\rho = -0.30$



A full example: Model diagnostics

The markov model can generate information on the calibration process with

```
std::cout << "Model trace : " << std::endl << mf->modelOutputs()
          << std::endl;
```

The output contains information on

- numerical model parameters
- settings for smile preconditioning
- yield term structure calibration results
- volatility smile calibration results

and can help identifying calibration problems, resp. confirm a successful calibration.

A full example: Model diagnostics (model parameters / smile settings)

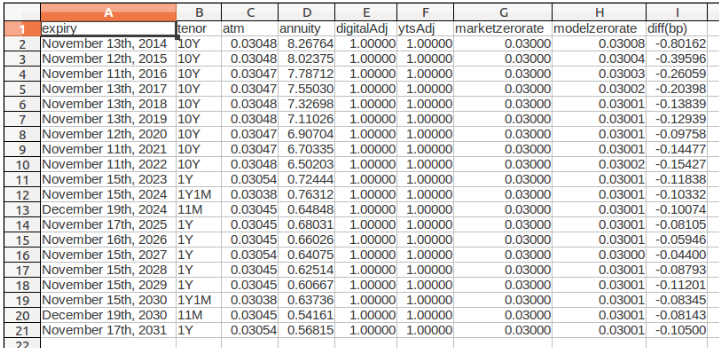
```
Markov functional model trace output
Model settings
Grid points y          : 64
Std devs y             : 7
Lower rate bound       : 0
Upper rate bound       : 2
Gauss Hermite points : 32
Digital gap            : 1e-05
Adjustments            : Kahale SmileExp
Smile moneyness checkpoints:
```

A full example: Model diagnostics (yield term structure fit)

The raw output

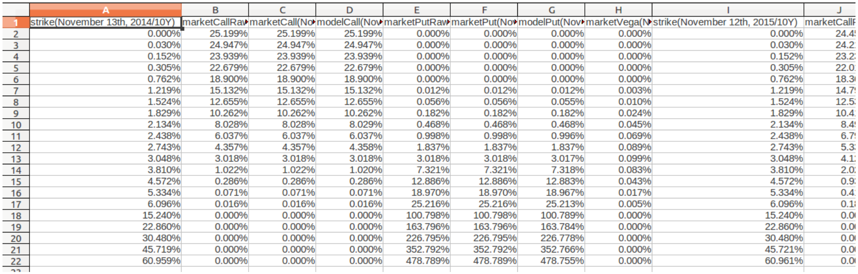
```
Yield termstructure fit:
expiry;tenor;atm;annuity;digitalAdj;ytsAdj;marketzerorate;modelzerorate;diff(bp)
November 13th, 2014;10Y;0.03047961644430512;8.267635590910791;1;1;0.03000000000000008;0.03008016185429107;-0.8
November 12th, 2015;10Y;0.0304803836917478;8.023747530203391;1;1;0.02999999999999999;0.03003959599137232;-0.39
[...]
```

is meant to be analyzed in another application like office



	A	B	C	D	E	F	G	H	I
1	expiry	tenor	atm	annuity	digitalAdj	ytsAdj	marketzerorate	modelzerorate	diff(bp)
2	November 13th, 2014	10Y	0.03048	8.26764	1.00000	1.00000	0.03000	0.03008	-0.80162
3	November 12th, 2015	10Y	0.03048	8.02375	1.00000	1.00000	0.03000	0.03004	-0.39596
4	November 11th, 2016	10Y	0.03047	7.78712	1.00000	1.00000	0.03000	0.03003	-0.26059
5	November 13th, 2017	10Y	0.03047	7.55030	1.00000	1.00000	0.03000	0.03002	-0.20398
6	November 13th, 2018	10Y	0.03048	7.32698	1.00000	1.00000	0.03000	0.03001	-0.13839
7	November 13th, 2019	10Y	0.03048	7.11026	1.00000	1.00000	0.03000	0.03001	-0.12939
8	November 12th, 2020	10Y	0.03047	6.90704	1.00000	1.00000	0.03000	0.03001	-0.09758
9	November 11th, 2021	10Y	0.03047	6.70335	1.00000	1.00000	0.03000	0.03001	-0.14477
10	November 11th, 2022	10Y	0.03048	6.50203	1.00000	1.00000	0.03000	0.03002	-0.15427
11	November 15th, 2023	1Y	0.03054	0.72444	1.00000	1.00000	0.03000	0.03001	-0.11838
12	November 15th, 2024	1Y1M	0.03038	0.76312	1.00000	1.00000	0.03000	0.03001	-0.10332
13	December 19th, 2024	11M	0.03045	0.64848	1.00000	1.00000	0.03000	0.03001	-0.10074
14	November 17th, 2025	1Y	0.03045	0.68031	1.00000	1.00000	0.03000	0.03001	-0.08105
15	November 16th, 2026	1Y	0.03045	0.66026	1.00000	1.00000	0.03000	0.03001	-0.05946
16	November 15th, 2027	1Y	0.03054	0.64075	1.00000	1.00000	0.03000	0.03000	-0.04400
17	November 15th, 2028	1Y	0.03045	0.62514	1.00000	1.00000	0.03000	0.03001	-0.08793
18	November 15th, 2029	1Y	0.03045	0.60667	1.00000	1.00000	0.03000	0.03001	-0.11201
19	November 15th, 2030	1Y1M	0.03038	0.63736	1.00000	1.00000	0.03000	0.03001	-0.08345
20	December 19th, 2030	11M	0.03045	0.54161	1.00000	1.00000	0.03000	0.03001	-0.08143
21	November 17th, 2031	1Y	0.03054	0.56815	1.00000	1.00000	0.03000	0.03001	-0.10500
22									

A full example: Model diagnostics (volatility smile fit)



	A	B	C	D	E	F	G	H	I	J
1	strike(November 13th, 2014/10Y)	marketCallRaw	marketCall(No	modelCall(Nov	marketPutRaw	marketPut(No	modelPut(Nov	marketVega(N	strike(November 12th, 2015/10Y)	marketCallF
2	0.000%	25.199%	25.199%	25.199%	0.000%	0.000%	0.000%	0.000%	0.000%	24.4
3	0.030%	24.947%	24.947%	24.947%	0.000%	0.000%	0.000%	0.000%	0.030%	24.2
4	0.152%	23.939%	23.939%	23.939%	0.000%	0.000%	0.000%	0.000%	0.152%	23.2
5	0.305%	22.679%	22.679%	22.679%	0.000%	0.000%	0.000%	0.000%	0.305%	22.0
6	0.762%	18.900%	18.900%	18.900%	0.000%	0.000%	0.000%	0.000%	0.762%	18.3
7	1.219%	15.132%	15.132%	15.132%	0.012%	0.012%	0.012%	0.003%	1.219%	14.7
8	1.524%	12.655%	12.655%	12.655%	0.056%	0.056%	0.055%	0.010%	1.524%	12.5
9	1.829%	10.262%	10.262%	10.262%	0.182%	0.182%	0.182%	0.024%	1.829%	10.4
10	2.134%	8.028%	8.028%	8.029%	0.468%	0.468%	0.468%	0.045%	2.134%	8.4
11	2.438%	6.037%	6.037%	6.037%	0.998%	0.998%	0.996%	0.069%	2.438%	6.7
12	2.743%	4.357%	4.357%	4.358%	1.837%	1.837%	1.837%	0.089%	2.743%	5.3
13	3.048%	3.018%	3.018%	3.018%	3.018%	3.018%	3.017%	0.099%	3.048%	4.1
14	3.810%	1.022%	1.022%	1.020%	7.321%	7.321%	7.318%	0.083%	3.810%	2.0
15	4.572%	0.286%	0.286%	0.286%	12.886%	12.886%	12.883%	0.043%	4.572%	0.9
16	5.334%	0.071%	0.071%	0.071%	18.970%	18.970%	18.967%	0.017%	5.334%	0.4
17	6.096%	0.016%	0.016%	0.016%	25.216%	25.216%	25.213%	0.005%	6.096%	0.1
18	15.240%	0.000%	0.000%	0.000%	100.798%	100.798%	100.789%	0.000%	15.240%	0.0
19	22.860%	0.000%	0.000%	0.000%	163.796%	163.796%	163.784%	0.000%	22.860%	0.0
20	30.480%	0.000%	0.000%	0.000%	226.795%	226.795%	226.778%	0.000%	30.480%	0.0
21	45.719%	0.000%	0.000%	0.000%	352.792%	352.792%	352.766%	0.000%	45.721%	0.0
22	60.959%	0.000%	0.000%	0.000%	478.789%	478.789%	478.755%	0.000%	60.961%	0.0

Axiomatic Hull White

Any gaussian one factor HJM model which satisfies *separability*, i.e.

$$\sigma_f(t,T) = g(t)h(T) \tag{1}$$

for the instantaneous forward rate volatility with deterministic $g, h > 0$, necessarily fulfills

$$dr(t) = (\theta(t) - a(t)r(t))dt + \sigma(t)dW(t) \tag{2}$$

for the short rate r, which means, it is a Hull White one factor model.

The T-forward numeraire

Set $x(t) := r(t) - f(0,t)$ and fix a horizon T, then in the T-forward measure the numeraire can be written

$$N(t) = P(t,T) = \frac{P(0,T)}{P(0,t)} e^{-x(t)A(t,T)+B(t,T)} \tag{3}$$

with A, B dependent on the model parameters. The Hull White model is called an *affine* model.

Smile in the Hull White Model

- The distribution of $N(t)$ is lognormal. The shape of the distribution can not be controlled by any of the model parameters.
- For fixed t you can calibrate the model to one market quoted interest rate optoin (typically a caplet or swaption).
- You can choose the strike of the option, but the rest of the smile is implied by the model.

Callable vanilla swaps

Pricing of callable fix versus Libor swaps may be done in a Hull White model which is calibrated as follows:

- For each call date find a market quoted swaption which is equivalent to the call right (in some sense, e.g. by matching the npv and its first and second derivative of the underlying at $E(x(t))$).
- Calibrate the volatility function $\sigma(t)$ to match the basket of these swaptions.
- Choose the mean reversion of the model to control serial correlations.

Intertemporal correlations

To understand the role of the reversion parameter assume σ and a constant for a moment. Then it is easy to see

$$\mathrm{corr}(x(T_1), x(T_2)) = \sqrt{\frac{e^{2aT_2} - 1}{e^{2aT_1} - 1}} = e^{-a(T_2 - T_1)} \sqrt{\frac{1 - e^{-2aT_1}}{1 - e^{-2aT_2}}} \tag{4}$$

which shows that for $a = 0$ the correlation is $\sqrt{T_1/T_2}$ and goes to zero if $a \to \infty$ and to one if $a \to -\infty$.

Callable cms swaps

The calll rights in a callable cms swap are options on a swap exchanging cms coupons against fix or Libor rates. Such underlying swaps are drastically mispriced in the Hull White model in general.

- cms coupons are replicated using swaptions covering the whole strike continuum $(0, \infty)$
- The swaption smile in the Hull White model is generally not consistent with the market smile and so are the prices of cms coupons

Obviously we need a more flexible model to price such structures

Model requirements

The wishlist for the model is as follows

- We want to be capable of calibrating to a whole smile of (constant maturity) swaptions, not only to one strike, for all fixing dates of the cms coupons. This is to match the coupons of the underlying.
- In addition we would like to calibrate to (possibly strike / maturity adjusted) coterminal swaptions to match the options representing the call rights.
- Finally we need some control over intertemporal correlations, i.e. something operating like the reversion parameter in the Hull White model

The idea to do so is to relax the functional dependency between the state variable x and the numeraire $N(t, x)$.

The driving process

We start with a markov process driving the dynamics of the model as follows:

$$dx = \sigma(t)e^{at}dW(t) \tag{5}$$

and $x(0) = 0$. The intertemporal correlation of the state variable x is the same as for the Hull White model, see (4), i.e. the parameter a can be used to control the correlation just as the reversion parameter in the Hull White model.

The numeraire surface

The model is operated in the T-forward measure, T chosen big enough to cover all cashflows relevant for the actual pricing under consideration. The link between the state $x(t)$ and the numeraire $P(t,T)$ is given by

$$P(t,T,x) = N(t,x) \tag{6}$$

which we allow to be a non parametric surface to have maximum flexibility in calibration.

Calibrating the numeriare surface to market smiles

The price of a digital swaption paying out an annuity $A(t)$ on expiry t if the swap rate $S(t) \geq K$ in our model is

$$\text{dig}_{\text{model}} = P(0,T) \int_{y^*}^{\infty} \frac{A(t,y)}{P(t,T)} \phi(y) dy \tag{7}$$

where y^* is the strike in the normalized state variable space (the correspondence between y and $S(t)$ is constructed to be monotonic).

Implying the swap rate

Given the market smile of $S(t)$ we can compute the market price $\text{dig}_{\text{mkt}}(\text{K})$ of digitals for strikes K. For given y^* we can solve the equation

$$\text{dig}_{\text{mkt}}(K) = P(0,T)\int_{y^*}^{\infty} \frac{A(t,y)}{P(t,T)}\phi(y)dy \tag{8}$$

for K to find the swap rate corresponding to the state variable value y^*. For this $\text{dig}_{mkt}(\cdot)$ should be a monotonic function whose image is equal to the possible digital prices $(0, A(0)]$. We will revisit this later.

Computing the deflated annuity

To compute the deflated annuity

$$\frac{A(t)}{P(t,T)} = \sum_{k=1}^{n} \tau_k \frac{P(t,t_k)}{P(t,T)} \tag{9}$$

we observe that

$$\left.\frac{P(t,u)}{P(t,T)}\right|_{y(t)} = E\left(\frac{1}{P(u,T)} \middle| y(t) \right) \tag{10}$$

i.e. we have to integrate the reciprocal of the numeraire at future times. Working backward in time we can assume that we know the numeraire at these times (starting with $N(T) \equiv 1$).

Converting swap rate to numeraire

Having computed the swap rate $S(t)$ we have to convert this value to a numeraire value $N(t)$. Since

$$S(t)A(t) + P(t,t^*) = 1 \tag{11}$$

we get (by division by $N(t)$)

$$N(t) = \frac{1}{S(t)\frac{A(t)}{N(t)} + \frac{P(t,t^*)}{N(t)}} \tag{12}$$

all terms on the right hand side computable via deflated zerobonds as shown above. Note that we use a slightly modified swap rate here, namely one without start delay.

Calibration to a second instrument set

Up to now we have not made use of the volatility $\sigma(t)$ in the driving markov process of the model. This parameter can be used to calibrate the model to a second instrument set, however only a single strike can be matched obviously for each expiry. A typical set up would be

- calibrate the numeraire to an underlying rate smile, e.g. constant maturity swaptions for cms coupon pricing
- calibrate $\sigma(t)$ to (standard atm or possibly adjusted) coterminal swaptions for call right calibration

Note that after changing $\sigma(t)$ the numeraire surface needs to be updated, too.

Input smile preconditioning

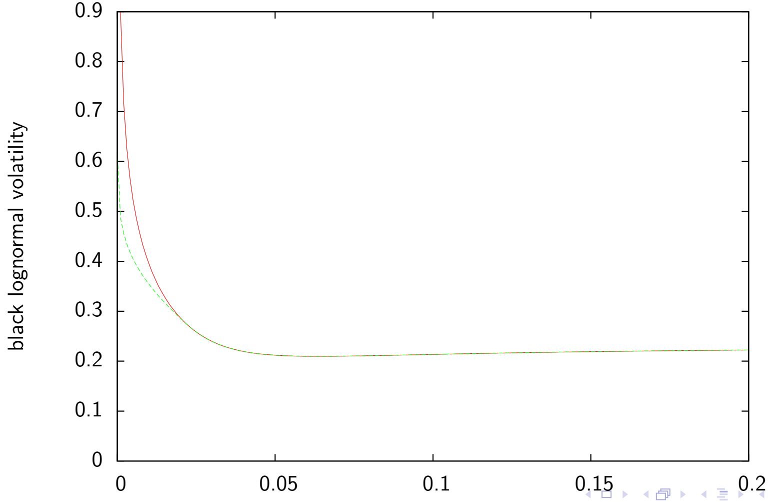
To ensure a bijective mapping

$$\text{dig}_{\text{mkt}} : (0, \infty) \to (0, A(0)) \tag{13}$$

it is sufficient to have an arbitrage free input smile. It is possible to allow for negative rates and generalize the interval $(0, \infty)$ to $(-\kappa, \infty)$ with some suitable $\kappa > 0$, e.g. $\kappa = 1\%$. In general input smiles are not arbitrage free, so some preconditioning is advisable, since arbitrageable smiles will break the numeraire calibration.

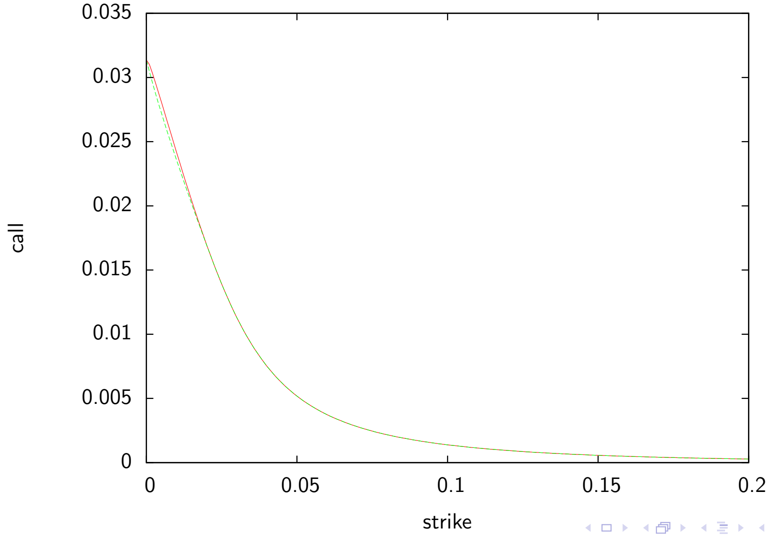
Kahale extrapolation

SABR 14y/1y implied black lognormal volatilities as of 14-11-2012, input (solid) and Kahale (dashed)



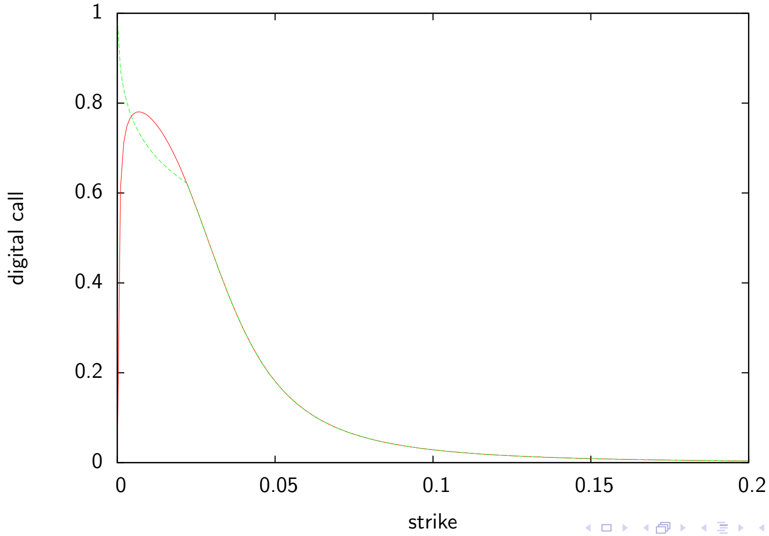
Kahale extrapolation

SABR 14y/1y call prices as of 14-11-2012, input (solid) and Kahale (dashed)



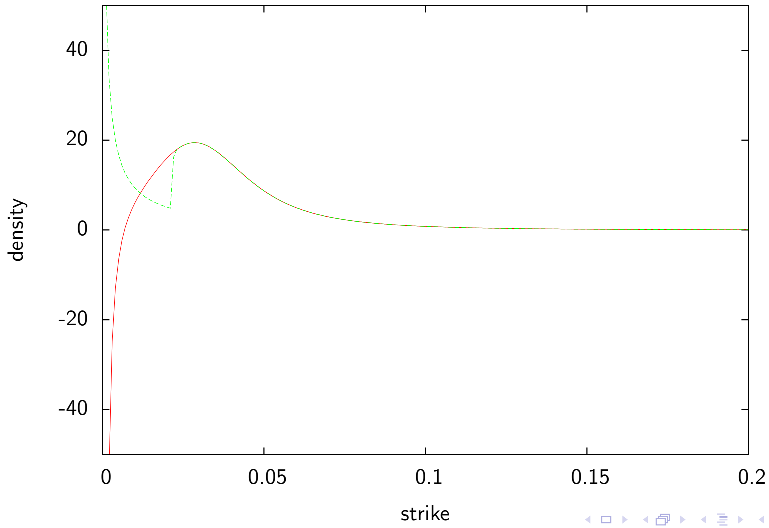
Kahale extrapolation

SABR 14y/1y digital prices as of 14-11-2012, input (solid) and Kahale (dashed)



Kahale extrapolation

SABR 14y/1y density as of 14-11-2012, input (solid) and Kahale (dashed)



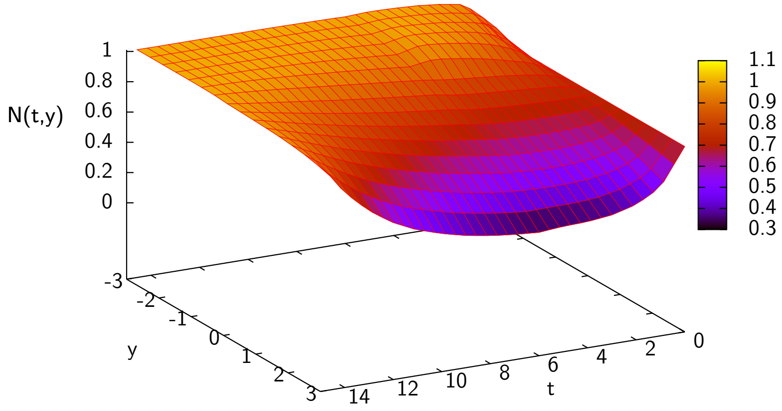
Interpolation of the numeraire

In a numerical implementation of the model we will need to discretize the numeraire surface on a grid (t_i, y_j).

- in y-direction we interpolate $N(t, y)$ (normalized by todays market forward numeraire) with monotonic cubic splines with Lagrange end condition. Outside a range of specified standard deviations (defaulted to 7), we extrapolate flat.
- in t-direction we interpolate the reciprocal of the normalized numeraire linearly. This ensures a perfect match of todays input yield curve even for interpolated times as can easily be seen from (10). After the horizon T we extrapolate flat ($N \equiv 1$ there anyway), only for technical reasons.

Sample numeraire surface

Numeraire surface for market data as of 14-11-2012



Interpolation of payoffs

Payoffs occuring in the numeraire bootstrap (digitals) or later in pricing exotics are also interpolated with Lagrange splines. We leave it as an option to

- restrict to integration of the payoff to a specified number of standard deviations,
- extrapolate the payoff flat
- extrapolate the payoff according to the Lagrange end condition

The results should not depend significantly on this choice, otherwise the numerical parameters should be increased.

Numerical Integration for deflated zero bonds

To compute deflated zerobond prices according to (10) it turns out that it is fast and accurate to

- use the celebrated Gauss Hermite Integration scheme where
- 32 points are more than enough usually to ensure a good accuracy.

This is because the integrand is globally well approximated by polynomials.

Numerical Integration for payoffs

For the numerical integration of

- digitals during numeraire bootstrapping or
- exotic pricing

Gauss Hermite is possible but leading not to satisfactory accuracy. This is due to the non global nature of the integrand in this case. Here we rely on exact integration of the piecewise 3rd order polynomials against the gaussian density, which is possible in closed form only involving the error function erf

How is the yield curve matched after all ?

It is interesting to note that the initial yield curve is matched by calibrating the numeraire to market input smiles. The yield curve is never a direct input though, as it is e.g. for the Hull White model. It is reconstructed by the model via the numeraire density coded in and read off the market smile during numeraire bootstrapping.

Long term constant maturity calibration

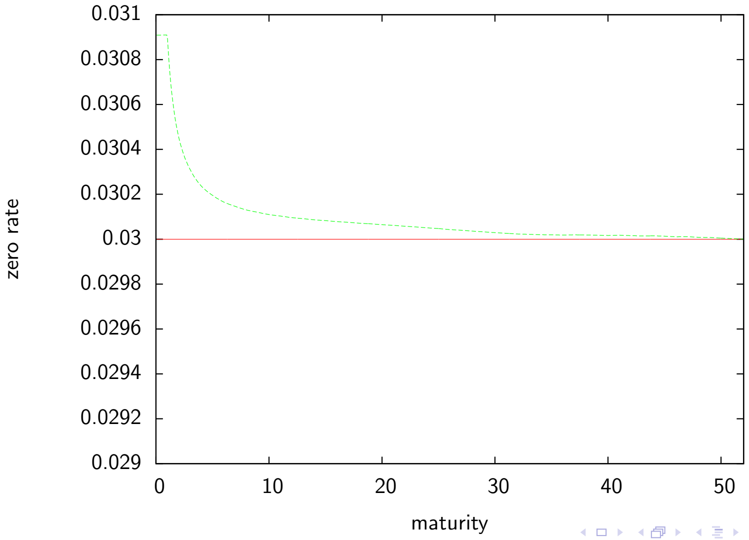
The hardest case of calibration is a long term calibration to constant maturity swaptions (or caplets).

- We start at maturity T and bootstrap the numeraire at some $t_k < T$, introducing some numerical noise in $N(t_k)$.
- We then boostrap $N(t_{k-1})$ for $t_{k-1} < t_k$, relying on the already bootstrapped future numeraire values.
- In case of a coterminal calibration we largely still use $N(T)$ and $N(t_k)$ only to a smaller degree.
- In case of cm calibration however we have to rely on $N(t_k)$ with t_k nearer to our calibration time t.

Therefore in long term cm calibration numerical noise may pile up giving large errors for the shorter term numeraire surface.

Yield curve match with standard numerical parameters

Fit to Yts flat @ 3%, 2y cm swaptions @ 20%, 7 standard deviations, 200 % upper rate bound



Increasing numerical accuracy for long term cm baskets

The first idea in this case is to increase the numerical accuracy by increasing

- the number of covered standard deviations (e.g. from 7 to 12)
- the upper cut off point for rates (e.g. from 200% to 400%)
- (maybe the number of discretization points, though less critical)

However this breaks the calibration totally, because the standard ("double") 53 bit mantissa numerical precision is not sufficient to do the computations numerically stable any more.

NTL high precision computing

The NTL and boost libraries provide support for arbitrary floating point precision. We incorporated NTL support as an option replacing the standard double precision by an arbitrary mantissa length precision in the critical sections of the computation (which turned out to be the integration of payoffs against the gaussian density, where large integrand values are multiplied by small density values). NTL is activated by commenting out

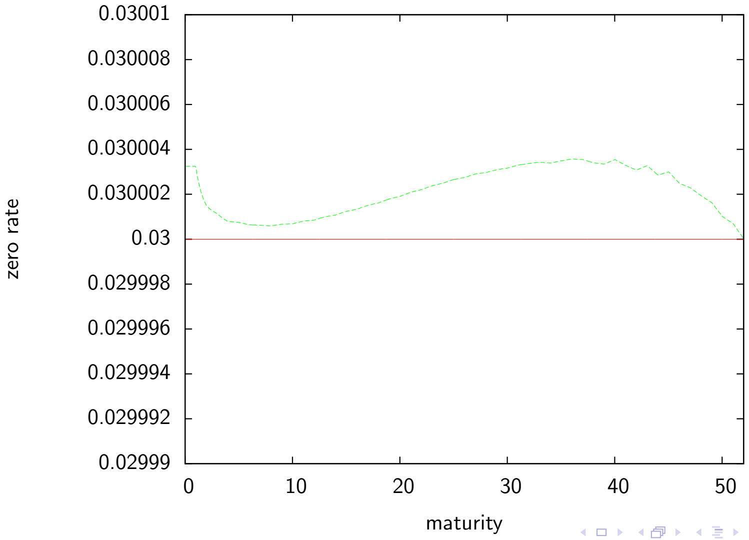
```
// uncomment to enable NTL support
#define GAUSS1D_ENABLE_NTL
```

in `gaussian1dmodel.hpp`. The mantissa length is then set (e.g. to 113bit) with

```
boost::math::ntl::RR:SetPrecision(113)
```

Yield curve match using high precision computing

Fit to Yts flat @ 3%, 2y cm swaptions @ 20%, 150bit mantissa, 12 standard deviations, 400 % upper rate bound



A more pragmatic approach: Adjustment factors

Computations with NTL are slow. A more practical way to stabilize the calibration in difficult circumstances are adjustment factors forcing the numeraire to match the market input yield curve. The adjustment factor is introduced by replacing

$$N(t_i, y_j) \rightarrow N(t_i, y_j) \frac{P^{\text{model}}(0, t_i)}{P^{\text{market}}(0, t_i)} \tag{14}$$

This option should be used with some care because it may lower the accuracy of the volatility smile match. In most situations the adjustment factors are moderate though, in the example we had before:

Adjustment factors in the example above

Date	Adjustment Factor
November 14th, 2013	1.00000029079227
November 14th, 2014	0.999999566861981
November 14th, 2015	0.999999720414697
November 14th, 2016	0.999999838009949
November 14th, 2017	0.999999380770489
...	...
November 14th, 2056	0.999804362556495
November 14th, 2057	0.999904959217797
November 14th, 2058	0.99988000473961
November 14th, 2059	0.999816723715493
November 14th, 2060	0.999830021528368
November 14th, 2061	0.999737006370401
November 14th, 2062	0.999761860227793
November 14th, 2063	0.999887598113548

MarkovFunctional Options - Numerical Parameters

- `yGridPoint` Number of discretization points for normalized state variable (default 64)
- `yStdDevs` Number of standard deviations of normalized state variable covered (default 7)
- `gaussHermitePoints` Number of points used in Gauss hermite integration (default 32)
- `digitalGap` Numerical stepsize to compute market digitals from call spreads (default 10^{-5})
- `marketRateAccuracy` Accuracy (in model swap rate) when matching market digitals (default 10^{-7})
- `lowerRateBound` Lower bound for model's (underlying) rates (default 0%)
- `upperRateBound` Upper bound for model's (underlying) rates (default 200%)

MarkovFunctional Options - Adjustments Overview

- `AdjustDigitals` – force the model's digital at the lower rate bound to be 1. Limited impact, not recommended, does not seem to be useful in practice.
- `AdjustYts` – force the model's yts to match the input yts by applying a constant adjustment factor per expiry. Recommended only for very long term calibrations, smile fit may suffer
- `ExtrapolatePayoffFlat` – Instead of extrapolating using the spline, extrapolate the digital's payoff flat outside the given standard deviations of the state variable. Recommended only for sanity checks, should have no impact, otherwise numerical parameters have to be adjusted
- `NoPayoffExtrapolation` – Do not extrapolate the payoff at all. Same as for ExtrapolatePayoffFlat holds

MarkovFunctional Options - Adjustments Overview ctd.

- `KahaleSmile` – Use Kahale smile preconditioning on arbitrageable wings, recommended (or `SabrSmile`) unless you are sure to provide an arbitrage free smile input
- `KahaleInterpolation` – Use Kahale smile interpolation (implies `KahaleSmile`), for naively interpolated smiles (linear, ...), however generates "Bat Man shaped" Densities, not recommended unless `SabrSmile` fails to work or is too inaccurate w.r.t. fit
- `SmileExponentialExtrapolation` – Use Exponential smile extrapolation on right wing (implies `KahaleSmile`), for slowly decreasing call price functions, recommended

MarkovFunctional Options - Adjustments Overview ctd.

- `SmileDeleteArbitragePoints` – Delete intermediate points from smile that cause arbitrage (implies `KahaleInterpolation`), recommended if you use `KahaleInterpolation`
- `SabrSmile` – Use a Sabr fitted smile (with Kahale wing sanitization), recommended for naively interpolated smiles
- `SmileMoneynessCheckPoints` – Custom strike grid for smile arbitrage check and Kahale interpolation in K/F space (optional), default is working in most cases

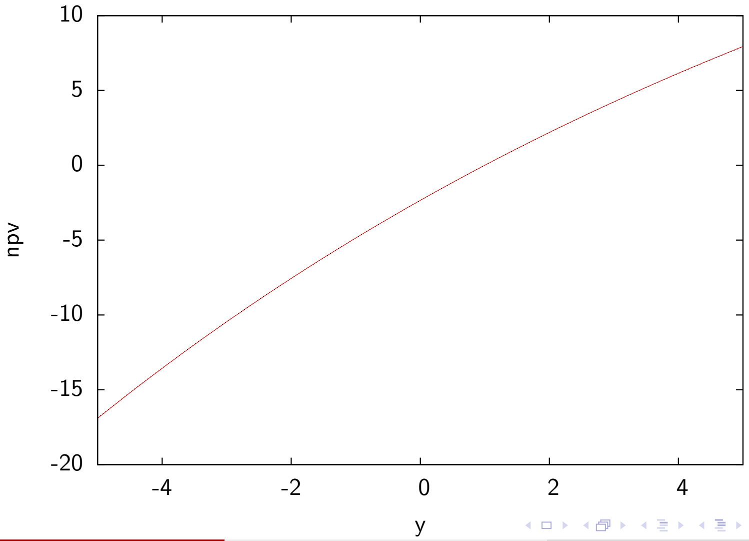
Representative Basket Approach

- Problem: Find standard swaptions that represent an exotic call right appropriately in your Markov Functional (or any one factor) model.
- Strategy: Match NPV, Delta[3], Gamma around some "central" value of the model's state variable. This works well for amortizing, accreting and step up/down swaptions for example. It is worth a try if we can use the method also in our initial example with the CMS swaption.

[3] which means the first derivative w.r.t. the state variable of the model here

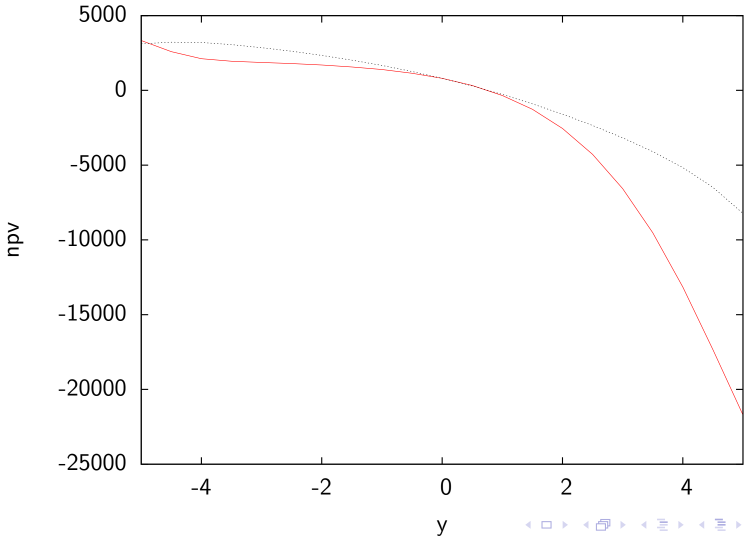
Global amortizing vanilla swap fit

(solid = exotic npv, dotted = standard underlying npv)



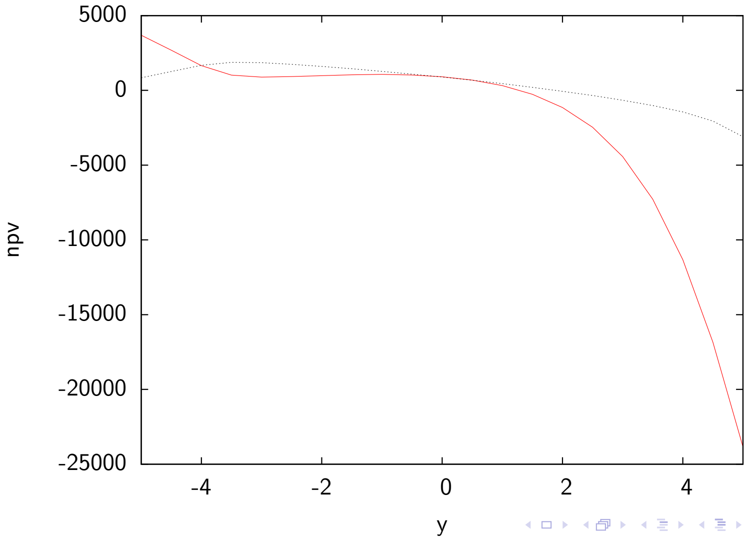
Global CMS swap fit (Flat smile)

(solid = exotic npv, dotted = standard underlying npv)



Global CMS swap fit (SABR smile)

(solid = exotic npv, dotted = standard underlying npv)



Representative Basket Approach - Conclusion

It does not seem to be clear to what secondary instrument set you should calibrate your model in order to represent calls on a cms (or more generally float float) swaps, although in practice atm coterminals seem to be a natural choice.

QuantLib Implementation

A base version of the model is available since release 1.3. Recent developemts (≥ 1.4 ?) include

- support for a (zero volatility) spread between discounting and forwarding curves
- a float float and non standard swaption pricing engine
- representative calibration basket generation
- support for pricing under credit risk (credit linked swaptions, exotic bonds)
- SABR-Kahale smile preconditioning

Outlook / Future Tasks

- Finite Difference pricing engines
- Additional smile preconditioning algorithms

References

- Johnson, Simon: Numerical methods for the markov functional model, Wilmott magazine, http://www.wilmott.com/pdfs/110802_johnson.pdf
- Kahale, Nabil: An arbitrage free interpolation of volatilities, Risk May 2004, http://nkahale.free.fr/papers/Interpolation.pdf
- Shoup, Victor: NTL A Library for doing Number theory, http://www.shoup.net/ntl/
- QuantLib A free/open-source library for quantitative finance, http://www.quantlib.org
- Caspers, Peter: Markov Functional One Factor Interest Rate Model Implementation in QuantLib, http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2183721
- Caspers, Peter: Representative Basket Method Applied, http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2320759

Thank you, Q and A