UNIVERSITÁ DEGLI STUDI DI MILANO-BICOCCA

Facoltá di Economia

Corso di Laurea Magistrale in Economia e Finanza

Abcd +

Relatore: Prof. Ferdinando M. AMETRANO

Tesi di Laurea di
GABRIELE GIUDICI
Nr. Matricola: 783898

Anno Accademico 2017-2018

abcdabcdabcd

Abstract

Abstract.

Contents

Introduction **iv**

1 Interest rate **1**

2 Multicurve bootstrapping **2**

3 The ABCD of interest Rate Basis Spread **3**

4 Abcd Framework **4**
- 4.1 Tenor Basis 4
- 4.2 Abcd Tenor Basis 11
- 4.3 Discount Corrected Term Structure 13

5 Abcd Reparameterization **15**
- 5.1 Parameters conversion 15
- 5.2 Idea adjustment 17

6 Abcd Framework+ **18**
- 6.1 Nested Calibration 18
- 6.2 Calibrator 18

7 Fixing of c empirical analysis **20**
- 7.1 Globally shared c 20
  - 7.1.1 Not incremental approach 20
  - 7.1.2 Incremental approach 24
- 7.2 Basis shifting 27
  - 7.2.1 Not incremental approach 28
  - 7.2.2 Incremental approach 32

8 Fixing of the time of maximum: an empirical analysis **36**

Introduction

Given the ABCD model [1], the aim of this research is to reparameterize it in accordance with the following explanation and to empirically verify the robustness of our methodology.

Common c

Looking at the abcd continuous form of the basis and considering the following statements, it is possible to understand the reasons for the introduction of a new inputs form: "If the exponential term is the same, then a difference between abcd/ABCD basis will still be an abcd/ABCD function". This is clear from a mathematical point of view indicating:

- x as the first generic tenor;
- y as the second generic tenor, where: $x > y$;
- x, y as the difference of the above mentioned tenor: $x - y$;

It follows that, because of the tenor basis dominance explained in [1], given two absolute basis:

$$s_x(t) > s_y(t), \forall t > 0$$

and

$$s_x(t) - s_y(t), \forall t > 0$$

such that their respective abcd forms are:

$$s_x(t) = (a_x + b_x t)e^{-c_x t} + d_x$$

and

$$s_y(t) = (a_y + b_y t)e^{-c_y t} + d_y$$

if $c_x = c_y = c_{x,y}$ then the relative basis is still an abcd function:

$$
\begin{aligned}
s_{x,y}(t) &= s_x(t) - s_y(t) \\
&= (a_x + b_x t)e^{-c_x t} + d_x - ((a_y + b_y t)e^{-c_y t} + d_y) \\
&= (a_x - a_y + (b_x - b_y)t)e^{-c_{x,y} t} + d_x - d_y \\
&= (a_{x,y} + b_{x,y} t)e^{-c_{x,y} t} + d_{x,y}
\end{aligned} \tag{1}
$$

Where the above reported equation is obtained using the below reported notation:

$$
\begin{aligned}
a_{x,y} &= a_x - a_y; \\
b_{x,y} &= b_x - b_y; \\
d_{x,y} &= d_x - d_y;
\end{aligned} \tag{2}
$$

This result has been shown for continuous basis, but obviously it is equally valid for simple ones.

Considering f_x for a generic tenors x, the above reported relation can be exploited. Given that:

$$
f_x(t) = s_x(t) + f_{ON}(t) \tag{3}
$$

Then:

$$
\begin{aligned}
s_{x,y}(t) &= f_x(t) - f_y(t) \\
&= s_x(t) + f_{ON}(t) - s_y(t) - f_{ON}(t) \\
&= s_x(t) - s_y(t)
\end{aligned} \tag{4}
$$

is still an abcd basis, if the two basis are sharing the common exponential term.

For this reason, it is possible to use the abcd framework from [1], where the calibration of the model has been made with respect to a tenor which differs from the f_{ON} .

However, is it true that, given two generic tenors x and y, if:

$$
c_x \neq c_y \tag{5}
$$

then the relative basis $s_{x,y}$ is not an abcd function? Empirical results [1] show how it seems that $s_{x,y}$ is still an abcd.

[1] *Abcd_Double_Hump_Research.xlsx* from the sheet *impact_time_dependent_d_lab*

Before going through algebraic calculus to proof this guess, it's preferred to proceed with a "reductio per absurdum". If the statement: "an abcd function shows only one t_{max}" is contradicted, then $s_{x,y}$ won't be an abcd function.

Given:

$$\begin{aligned} s_{x,y}(t) &= s_x(t) - s_y(t) \\ &= (a_x + b_x t)e^{-c_x t} + d_x - ((a_y + b_y t)e^{-c_y t} + d_y) \end{aligned} \tag{6}$$

considering $t_{max}(x)$ equation for s_x :

$$[-c_x(a_x + b_x t_{max}(x)) + b_x]e^{-c_x t_{max}(x)} = 0 \tag{7}$$

then $t_{max}(x,y)$ equation for $s_{x,y}$ is:

$$\begin{aligned} &[-c_x(a_x + b_x t_{max}(x,y)) + b_x]e^{-c_x t_{max}(x,y)} \\ -&[-c_y(a_y + b_y t_{max}(x,y)) + b_y]e^{-c_y t_{max}(x,y)} = 0 \end{aligned} \tag{8}$$

Unfortunately, it's impossible to retrieve $t_{max}(x,y)$ because of the problematic form:

$$y = b(e^b) \tag{9}$$

Because of this result it's impossible to study the problem with a "reductio per absurdum" and it's necessary to make an attempt to retrieve an abcd from $s_{x,y}$:

$$\begin{aligned} s_{x,y}(t) &= s_x(t) - s_y(t) \\ &= (a_x + b_x t)e^{-c_x t} + d_x - ((a_y + b_y t)e^{-c_y t} + d_y) \\ &= (a_x + b_x t)e^{-c_x t} + d_{x,y} \end{aligned} \tag{10}$$

Where: $d_{x,y} = d_x - ((a_y + b_y t)e^{-c_y t} + d_y)$. This means that $s_{x,y}$ is still an abcd basis, but d is time dependent, this is the reason why looking at the empirical results it seemed an abcd one. Therefore, it makes no sense to talk about relative abcd retrieved from absolute abcd with different c.

Given this findings, what did it happen when $s_{x,y}$ was treated as an abcd in [1]? They obtained by construction $s_{x,6M}$ as abcd, for doing this they calibrated the basis on s_{6M} according to (4). Given that s_{6M} and $s_{x,6M}$ were abcd by construction, but with different c, it follows that s_x was not abcd. The only drawback in the scheme is that the theoretical approach is not completely respected and therefore is not possible to swap from relative to absolute basis. Acting in this way [1] created two different models, one for the absolute basis and one for the relative ones, that can be reconciled guaranteeing that c is the same for both the absolute and relative basis. Obviously, according to the calibration that will be performed (incremental or not), different parameters will be estimated, but with a common c will be possible to swap from a basis to the other indepentendly from the calibration type.

Maximum time

Independently from the considered tenor, the market uncertainty is expected to be on the same point of time t_{max}. In fact,there are no financial reasons why the market should forecast different uncertainty horizons for different tenors. Theoretically, this means that:

$$t_{max}(x) = \frac{1}{c_x} - \frac{a_x}{b_x} \tag{11}$$

is not a function of tenor, but an implied bound instead:

$$t_{max} = \frac{1}{c_x} - \frac{a_x}{b_x} \tag{12}$$

The above equation shows how the t_{max} needs to be the same for each tenor, thanks to the relation (tenor dependent) amongst a,b and c.

Recalling the importance of c, its value is strictly related with the maximum time, because given the maximum time functional (continuous time) form (11) and considering that empirically the magnitude of:

$$\frac{1}{c_x}$$

dominates the ratio between a_x and b_x, the value of c explains the value of t_{max}. Therefore, in order to fix t_{max} it makes sense to fix c. If c is fixed then (12) becomes:

$$t_{max} = \frac{1}{c} - \frac{a_x}{b_x} \tag{13}$$

This means that, in order to have the same t_{max} for different tenors, there should be an implied relation between a and b that holds for each tenor, so that, given two generic tenor x and y:

$$\frac{a_x}{b_x} = \frac{a_y}{b_y} \tag{14}$$

that need to be empirically evaluated.

In the end, strictly related with t_{max} is the corresponding value of the absolute abcd:

$$s_x(t_{max}) = \frac{b_x}{c_x} e^{(\frac{a_x c_x}{b_x} - 1)} + d_x \tag{15}$$

Aim

For these above mentioned reasons, the ideas that need to be investigated are:

1. reparameterization of the model, in its continuous form, relying on the following parameters:

$$a_x, d_x, t_{max}, s_x(t_{max}) \tag{16}$$

   This new parametric form makes sense, because:

   - $a_x + d_x$ is the value of the abcd corresponding to $t = 0$;
   - d_x is the long run value of the abcd basis;
   - t_{max} represents the peak of uncertainty in terms of time;
   - $s_x(t_{max})$ is the value of the abcd at the peak of uncertainty;

   Therefore, with this view is possible to give financial meaning to model parameters. Remarkable feature of our theoretical approach should be that, after retrieving the new parametric form, all the analytical findings from [1] could be exploited, just shifting from the new parametric form to the old abcd one. In this way, the users are given an interface, which allows them to fix the maxima of the basis as observed on the markets.

2. modification of the framework and testing the fitting globally sharing c, ensuring the property of moving from absolute to relative basis and vice versa;

3. modification of the framework and testing the fitting globally sharing t_{max}, studying the financial intuition behind it;

4. modification of the framework and testing the fitting globally sharing t_{max} and c, in order to completely validate the model and the idea.

Chapter 1

Interest rate

Chapter 2

Multicurve bootstrapping

Chapter 3

The ABCD of interest Rate Basis Spread

Chapter 4

Abcd Framework

In order to improve the [1] framework it is essential to understand the previous implementation, that is written in C++ language and available in the QuantLib library in the "experimental" folder. The abcd implementation relies on the QuantLib modularity which acts as a base framework from which to extend all the further derivations.

4.1 Tenor Basis

Tenor basis is the super class from which the abcd scheme derives. It is modelled as a *CalibratedModel*:

```
class TenorBasis : public CalibratedModel {
    public:
        TenorBasis(Size nArguments,
                   boost::shared_ptr<IborIndex> iborIndex,
                   boost::shared_ptr<IborIndex> baseIborIndex,
                   Date referenceDate = Date())
                   ... }
```

Therefore it inherits its structure and calibrate methods. The most important methods, for continuous basis calibration, are:

```
Rate accrualFactor(Date d1,Date d2) const;
```

and

```
void calibrate(
   const std::vector<boost::shared_ptr<RateHelper> >&,
   OptimizationMethod& method,
   const EndCriteria& endCriteria
   = EndCriteria(1000, 100, 1.0e-8, 0.3e-4, 0.3e-4),
   const std::vector<Real>& weights = std::vector<Real>(),
   const std::vector<bool>& fixParameters = std::vector<bool>());
```

For better understanding these two functions, it needs to be explained:

```
class TenorBasisYieldTermStructure : public YieldTermStructure {
  public:
    TenorBasisYieldTermStructure
    (const boost::shared_ptr<TenorBasis>& basis);
    const Date& referenceDate() const;
    Calendar calendar() const;
    Natural settlementDays() const;
    Date maxDate() const;
  private:
    DiscountFactor discountImpl(Time) const;
    boost::shared_ptr<TenorBasis> basis_;
    };
```

and its constructor:

```
    TenorBasisYieldTermStructure::TenorBasisYieldTermStructure(
                    const boost::shared_ptr<TenorBasis>& basis)
    : YieldTermStructure(basis->dayCounter()), basis_(basis) {}
```

that simply builds an *YieldTermStructure* object and stores a *TenorBasis* object called basis in *basis_*. Moreover, it designs a *discountImpl* which will be fundamental and therefore later explained. The core of this code is the *calibrate* method which follows:

```
    void TenorBasis::calibrate(
    const std::vector<boost::shared_ptr<RateHelper> >& helpers,
    OptimizationMethod& method,
    const EndCriteria& endCriteria,
    const std::vector<Real>& weights,
    const std::vector<bool>& fixParameters) {
    TenorBasisYieldTermStructure
     yts(boost::shared_ptr<TenorBasis>(this, no_deletion));
    std::vector<boost::shared_ptr<CalibrationHelperBase> >
                                   cHelpers(helpers.size());
    for (Size i = 0; i<helpers.size(); ++i) {
         helpers[i]->setTermStructure(&yts);
         cHelpers[i] = helpers[i];
    }
    CalibratedModel::calibrate(cHelpers, method, endCriteria,
                       constraint(), weights, fixParameters);
    }
```

A *TenorBasisYieldTermStructure*, which stores the *TenorBasis* object itself as *basis_*, is built and called yts. Then a vector of *CalibrationHelperBase* is instantiated. It is fundamental the chain which links *CalibrationHelper* and *RateHelper*: indeed the former it is child of *CalibrationHelperBase*, while the latter inherits from *BootstrapHelper*, which is child of *CalibrationHelperBase* too. Given these chain of properties it is possible writes

$cHelpers[i] = helpers[i]$. In the end the method from the parent class *CalibratedModel* is exploited:

```
   void CalibratedModel::calibrate(
                 const vector<shared_ptr<CalibrationHelper> >& h,
                 OptimizationMethod& method,
                 const EndCriteria& endCriteria,
                 const Constraint& additionalConstraint,
                 const vector<Real>& weights,
                 const vector<bool>& fixParameters) {
    vector<shared_ptr<CalibrationHelperBase> > tmp(h.size());
    for (Size i=0; i<h.size(); ++i)
    tmp[i] = static_pointer_cast<CalibrationHelperBase>(h[i]);
    calibrate(tmp, method, endCriteria, additionalConstraint,
    weights, fixParameters);
    }
```

This piece of code simply casts the helpers in *CalibrationHelperBase* objects and then calls the overloaded polymorphic method *calibrate*, which is the core of the calibration process:

```
void CalibratedModel::calibrate(
        const vector<shared_ptr<CalibrationHelperBase> >& h,
        OptimizationMethod& method,
        const EndCriteria& endCriteria,
        const Constraint& additionalConstraint,
        const vector<Real>& w,
        const vector<bool>& fixParameters) {

    QL_REQUIRE(!h.empty(), "no helpers provided");

    Array prms = params();
    if (fixParameters.empty()) {
        fixedParameters_.resize(prms.size());
        std::fill(fixedParameters_.begin(),
        fixedParameters_.end(), false);
    } else {
        QL_REQUIRE(fixParameters.size() == prms.size(),
               "mismatch between number of parametrs (" <<
               h.size() << ") and fixed parameters booleans
               (" <<fixParameters.size() << ")");
        fixedParameters_ = fixParameters;
    }
    Projection proj(prms, fixedParameters_);

    if (w.empty()) {
        weights_.resize(h.size());
        std::fill(weights_.begin(), weights_.end(), 1.0);
    } else {
       //checks
```

```
}
CalibrationFunction f(this, h, weights_, proj);

Constraint c;
if (additionalConstraint.empty())
 c = *constraint_;
else
c = CompositeConstraint(*constraint_, additionalConstraint);
ProjectedConstraint pc(c, proj);

Problem prob(f, pc, proj.project(prms));
shortRateEndCriteria_ = method.minimize(prob, endCriteria);
Array result(prob.currentValue());
setParams(proj.include(result));
problemValues_ = prob.values(result);

notifyObservers();
}
```

In the following, the above reported code is dissected. Firstly, the fixing of parameters is managed: if *fixParameters* is empty by default the values are not fixed, instead if some value is fixed, it is required that the number of fixing indicator should equals the number of parameters.

Afterwards, a *Projection* object is instantiated. There a bit of the machinery is repeated (about checking the number of fixing indicator) and then the number of free parameters is set. Furthermore, there is a fundamental method *Projection::project* which returns only the not fixed values, i.e. those parameters that should be calibrated during the process, for this reason called *projectedParameters*.

After checking the presence of weights and assigned a default values of 1 for each entry of the *weights_* vector, it builds the *CalibrationFunction*, this is possible because it is an inner class of *CalibratedModel*. This function is essential, because allows to retrieve the errors in order to calibrate the model, but will be later better explained.

Moreover, a *ProjectedConstraint* object is built and it allows to manage the additional constraint along with the fixed parameters.

Therefore, all the ingredients necessary for the problem are provided, the problem set and the solver method can minimize the problem request with a specific *EndCriteria*. Here the trick is performed. Considering that the selected method for abcd framework is *LevenbergMarquardt*:

```
EndCriteria::Type LevenbergMarquardt::minimize(Problem& P,
                        const EndCriteria& endCriteria) {
        EndCriteria::Type ecType = EndCriteria::None;
        P.reset();
        Array x_ = P.currentValue();
```

```
    currentProblem_ = &P;
    initCostValues_ = P.costFunction().values(x_);
    int m = initCostValues_.size();
    int n = x_.size();
    if(useCostFunctionsJacobian_) {
        initJacobian_ = Matrix(m,n);
        P.costFunction().jacobian(initJacobian_, x_);
    }
    boost::scoped_array<Real> xx(new Real[n]);
    std::copy(x_.begin(), x_.end(), xx.get());
    boost::scoped_array<Real> fvec(new Real[m]);
    boost::scoped_array<Real> diag(new Real[n]);
    int mode = 1;
    Real factor = 1;
    int nprint = 0;
    int info = 0;
    int nfev =0;
    boost::scoped_array<Real> fjac(new Real[m*n]);
    int ldfjac = m;
    boost::scoped_array<int> ipvt(new int[n]);
    boost::scoped_array<Real> qtf(new Real[n]);
    boost::scoped_array<Real> wa1(new Real[n]);
    boost::scoped_array<Real> wa2(new Real[n]);
    boost::scoped_array<Real> wa3(new Real[n]);
    boost::scoped_array<Real> wa4(new Real[m]);
    //error messages
    QL_REQUIRE(n > 0, "no variables given");
    QL_REQUIRE(m >= n,
               "less functions (" << m <<
               ") than available variables (" << n << ")");
    QL_REQUIRE(endCriteria.functionEpsilon() >= 0.0,
               "negative f tolerance");
    QL_REQUIRE(xtol_ >= 0.0, "negative x tolerance");
    QL_REQUIRE(gtol_ >= 0.0, "negative g tolerance");
    QL_REQUIRE(endCriteria.maxIterations() > 0,
               "null number of evaluations");

    MINPACK::LmdifCostFunction lmdifCostFunction =
    boost::bind(&LevenbergMarquardt::fcn, this,
    _1, _2, _3, _4, _5);
    MINPACK::LmdifCostFunction lmdifJacFunction =
    useCostFunctionsJacobian_
    ? boost::bind(&LevenbergMarquardt::jacFcn, this,
    _1, _2, _3, _4, _5)
        : MINPACK::LmdifCostFunction(NULL);
    MINPACK::lmdif(m, n, xx.get(), fvec.get(),
                   endCriteria.functionEpsilon(),
                   xtol_,
                   gtol_,
```

```
                        endCriteria.maxIterations(),
                        epsfcn_,
                        diag.get(), mode, factor,
                        nprint, &info, &nfev, fjac.get(),
                        ldfjac, ipvt.get(), qtf.get(),
                        wa1.get(), wa2.get(), wa3.get(), wa4.get(),
                        lmdifCostFunction,
                        lmdifJacFunction);
        info_ = info;
        // check requirements & endCriteria evaluation
        //QLREQUIRE checks
        std::copy(xx.get(), xx.get()+n, x_.begin());
        P.setCurrentValue(x_);
        P.setFunctionValue(P.costFunction().value(x_));

        return ecType;
    }
```

It resets the problem and sets the variables, it performs a series of checks, it sets a particular cost function *LevenbergMarquardt::fcn* and then *lmdif* performs the minimization, after that the current problem value and the cost function one are set. Without being too specific *lmdif* logic acts as follows: the error that should be minimized are retrieved through the values function embedded in :

```
void LevenbergMarquardt::fcn(int, int n, Real* x,
Real* fvec, int*) {
    Array xt(n);
    std::copy(x, x+n, xt.begin());
    if (currentProblem_->constraint().test(xt)) {
        const Array& tmp = currentProblem_->values(xt);
        std::copy(tmp.begin(), tmp.end(), fvec);
    } else {
    std::copy(initCostValues_.begin(),
    initCostValues_.end(), fvec);
    }
    }
        }
```

and written in *fvec*. The implementation of *values*, from the *model* class, follows:

```
    virtual Disposable<Array>values(const Array& params)const {
            model_->setParams(projection_.include(params));
            Array values(helpers_.size());
            for (Size i=0; i<helpers_.size(); ++i) {
                values[i] = helpers_[i]->calibrationError() *
                std::sqrt(weights_[i]);
            }
            return values;
```

```
        }
```

The overall error it is retrieved exploiting *calibrationError()*:

```
Real calibrationError() const {
return quote_->value() - impliedQuote(); }
```

and there it is clear the mimic of the bootstrapping *"Ã la QuantLib"* which creates an error, exploiting the polymorphic feature of the particular instantiated rate helper. This is not all: when the *impliedQuote* method is called, then it asks at the *termStructure_* to return a *discount*. Besides, the *zeroRate* and *forwardRate* functions in QuantLib are all implemented as functions of *discount*, that it is a function of *disocuntImpl*, therefore they are linked. Remembering what anticipated above about the importance of *accrualFactor*, the *TenorBasisYieldTermStructure::discountImpl* is shown:

```
 DiscountFactor TenorBasisYieldTermStructure::
 discountImpl(Time t) const {
        Date ref = referenceDate();
        Date d = basis_->dateFromTime(t);
        Real accrFactor = basis_->accrualFactor(ref, d);
        return 1.0 / accrFactor;
   }
```

this is the key: the error depends on *discountImpl* that it depends on *accrualFactor*, which models the pseudo discount with the abcd framework :

```
   Rate TenorBasis::accrualFactor(Date d1,
                                  Date d2) const {
   QL_REQUIRE(d1 <= d2,
              "d2 (" << d2 << ") cannot be
               before d1 (" << d1 << ")");
   // baseCurve must be a discounting curve...
   // otherwise it could not provide fwd(t1, t2) with t2-t1!=tau
   Handle<YieldTermStructure> baseCurve =
   baseIborIndex_->forwardingTermStructure();
   Real accrFactor =
   baseCurve->discount(d1) / baseCurve->discount(d2);
   Real instContBasisIntegral = sign_ * integrate_(d1, d2);
   accrFactor *= std::exp(instContBasisIntegral);
   return accrFactor;
   }
```

Therefore, the *accrFactor* is initially retrieved from the *baseCurve* through the *discount* method and then multiplied by the compounding factor, where the exponential factor is the integrated instantaneous basis. Note : the sign of the abcd factor depends on whether or not we are calibrating with respect to a *baseCurve* with a greater *tenor* with respect to the benchmark curve whom base is searched.

4.2 Abcd Tenor Basis

AbcdTenorBasis is a child class of the above presented *TenorBasis* class. The main feaure exploited in the excel framework is its constructor:

```
AbcdTenorBasis::AbcdTenorBasis(shared_ptr<IborIndex> iborIndex,
                         boost::shared_ptr<IborIndex> baseIborIndex,
                         Date referenceDate,
                         bool isSimple,
                         const std::vector<Real>& coeff)
    : TenorBasis(4, iborIndex, baseIborIndex, referenceDate) {
        //std::vector<Real> y = inverse(coeff);
        std::vector<Real> y = coeff;
        arguments_[0] = ConstantParameter(y[0], NoConstraint());
        arguments_[1] = ConstantParameter(y[1], NoConstraint());
        arguments_[2] = ConstantParameter(y[2], NoConstraint());
        arguments_[3] = ConstantParameter(y[3], NoConstraint());
        isSimple_ = isSimple;
        generateArguments();
    }
```

It takes a vector of guess *coeff* and it stores them in an vector of object: *arguments_*. Then, in order to choose the correct algorithm, it ask whether or not the calibration is on simple basis and then it generates the problem parameters:

```
void AbcdTenorBasis::generateArguments() {
    std::vector<Real> x(4);
    x[0] = arguments_[0](0.0);
    x[1] = arguments_[1](0.0);
    x[2] = arguments_[2](0.0);
    x[3] = arguments_[3](0.0);
    //std::vector<Real> y = direct(x);
    std::vector<Real> y = x;
    if (isSimple_) {
        basis_ = shared_ptr<AbcdMathFunction>(
            new AbcdMathFunction(y[0], y[1], y[2], y[3]));
        vector<Real> c =
        basis_->definiteDerivativeCoefficients(0.0, tau_);
        c[0] *= tau_;
        c[1] *= tau_;
        // unaltered c[2] (the c in abcd)
        c[3] *= tau_;
        instBasis_ =
        shared_ptr<AbcdMathFunction>(new AbcdMathFunction(c));
    } else {
        instBasis_ = shared_ptr<AbcdMathFunction>(
            new AbcdMathFunction(y[0], y[1], y[2], y[3]));
        vector<Real> c =
```

```
        instBasis_->definiteIntegralCoefficients(0.0, tau_);
        c[0] /= tau_;
        c[1] /= tau_;
        // unaltered c[2] (the c in abcd)
        c[3] /= tau_;
        basis_ =
        shared_ptr<AbcdMathFunction>(new AbcdMathFunction(c));
    }
    }
```

All the *arguments_* entries are written in the vector x, then according to the type of searched basis (*isSimple* or not) it creates, in the continuous basis specific case, an *instBasis_* and *AbcdMathFunction* with the given parameters:

```
AbcdMathFunction::AbcdMathFunction(Real aa, Real bb,
Real cc, Real dd)
    : a_(aa), b_(bb), c_(cc), d_(dd), abcd_(4), dabcd_(4) {
        abcd_[0]=a_;
        abcd_[1]=b_;
        abcd_[2]=c_;
        abcd_[3]=d_;
        initialize_();
    }
```

It creates *abcd_* and *dabcd_* vectors, where *dabcd_* is the vector of derivative coefficients, it sets *abcd_* and calls *initialize*:

```
  void AbcdMathFunction::initialize_() {
        validate(a_, b_, c_, d_);
        da_ = b_ - c_*a_;
        db_ = -c_*b_;
        dabcd_[0]=da_;
        dabcd_[1]=db_;
        dabcd_[2]=c_;
        dabcd_[3]=0.0;

        pa_ = -(a_ + b_/c_)/c_;
        pb_ = -b_/c_;
        K_ = 0.0;

        dibc_ = b_/c_;
        diacplusbcc_ = a_/c_ + dibc_/c_;
    }
```

Before defining a series of variables that will be exploited in the algorithm, it calls *AbcdMathFunction::validate* that simply checks that specific abcd framework features are matched. Going back to *generateArguments*, another abcd, but simple one, is instantiated transforming the *instBasis_* parameters.

Given this framework, the *AbcdTenorBasis* can be calibrated according to the parent class above explained method.

4.3 Discount Corrected Term Structure

Differently from the above shown framework, the *DiscountCorrectedTermStructure* one has been already too well explained in [2] (Thank you for this book Luigi), because it is a particular case of the famous *PiecewiseYieldCurve* scheme. However, it is worth to further explain some details of this specific implementation:

```
class DiscountCorrectedTermStructure :
        public YieldTermStructure,
        protected InterpolatedCurve<Linear>,
        public LazyObject {
  public:
  typedef Discount traits_type;
  typedef Linear interpolator_type;
  DiscountCorrectedTermStructure(
  const Handle<YieldTermStructure>& bestFitCurve,
  const std::vector<boost::shared_ptr<RateHelper> >& instruments,
  Real accuracy = 1.0e-12);
  const Date& referenceDate() const;
  DayCounter dayCounter() const;
  Calendar calendar() const;
  Natural settlementDays() const;
  Date maxDate() const;
  const std::vector<Time>& times() const;
  const std::vector<Date>& dates() const;
  const std::vector<Real>& data() const;
      void update();
  private:
  DiscountFactor discountImpl(Time) const;
  void performCalculations() const;
  // data members
  Handle<YieldTermStructure> bestFitCurve_;
  std::vector<boost::shared_ptr<RateHelper> > instruments_;
  Real accuracy_;
  mutable std::vector<Date> dates_;

  // bootstrapper classes are declared as friend to manipulate
  // the curve data. They might be passed the data instead, but
  // it would increase the complexity---which is high enough
  // already.
  friend class IterativeBootstrap<DiscountCorrectedTermStructure>;
  friend class BootstrapError<DiscountCorrectedTermStructure>;
  IterativeBootstrap<DiscountCorrectedTermStructure> bootstrap_;
```

```
};
```

Firstly, the *traits_type* is a discount and the *interpolator_type* is linear, it means that the bootstrapping is performed on the pseudo discount factor with a linear interpolation. Furthermore, the other method that matter is:

```
DiscountFactor DiscountCorrectedTermStructure:
discountImpl(Time t) const {
    calculate();
    DiscountFactor d = bestFitCurve_->discount(t, true);
    Real k = interpolation_(t, true);
    return k*d;
}
```

The two of this feature together means that: the algorithm starts the bootstrap process from a pillar guess that is close to a possible value of the discount factor (1, also the optimal correction factor value). Then, inside the *IterativeBootstrap* code, through a *BootstrapError:: operator*:

```
    template <class Curve>
Real BootstrapError<Curve>::operator()(Real guess) const {
    Traits::updateGuess(curve_->data_, guess, segment_);
    curve_->interpolation_.update();
    return helper_->quoteError();
}
```

it updates the curve with the new guess, it interpolates with a linear interpolator, updates the observers and returns the error, but the *quoteError()* interface is:

```
Real BootstrapHelper::quoteError()
const { return quote_->value() - impliedQuote(); }
```

Therefore, as in the previous framework an *impliedQuote* is retrieved, that leads to a call to the above shown *discountImpl* method, that interpolates the correction factors that compose the curve that is currently bootstrapped. To better explain the problem that has been solved with this algorithm, it is possible to think in this way: "given a curve that returns a certain fixing value for a certain curve pillar (the base one with respect to the basis is searched)), what is the correction that needs to be applied in order to perfectly repricing the observed quotes?". Or better: when *discountImpl* is invoked, it doesn't know that there is already a discount from $bestFitCurve\_$, therefore it will just try to solve its problem and, given the presence of this particular *discountImpl* implementation that provide a basis value, this will lead it implicitly retrieving a bootstrapped curve of correction factors.

Note: *QuantLib* is **The QuantLib**.

Chapter 5

Abcd Reparameterization

5.1 Parameters conversion

Going through the practical side of the exhibited idea, problems are encountered retrieving b and c functional forms.

Given that the choice of how to specify the parameters is ambiguous, the equations of $s_x(t_{max})$ and t_{max} have been plugged into a system, in order to explicit c and b:

$$\begin{cases} s_x(t_{max}) = \frac{b_x}{c_x} e^{(\frac{a_x c_x}{b_x} - 1)} + d_x & (15) \\ t_{max} = \dfrac{1}{c_x} - \dfrac{a_x}{b_x} & (12) \end{cases}$$

Note: to achieve better readability during this mathematical steps:

- $s_x(t_{max})$ will be written as: s;
- t_{max} will be written as: t;
- all the parameters will be expressed without considering the tenor, in a generic way;

Starting from (12):

$$t = \frac{1}{c} - \frac{a}{b}$$
$$\frac{1}{c} = t + \frac{a}{b}$$

This equation is obtained:

$$c = \frac{b}{a + tb} \tag{5.1}$$

Then, working on (15):

$$s = \frac{b}{c}e^{(\frac{ac}{b}-1)} + d$$

and plugging (5.1), the following is obtained:

$$s = (bt + a)e^{(\frac{a}{bt+a}-1)} + d$$

but b cannot be retrieved because of the presence of a form such as (9). A further solution can be to retrieve b from (12):

$$b = \frac{ac}{1 - tc} \tag{5.2}$$

and then substitute in (15), obtaining:

$$s = \frac{a}{1 - tc}e^{(-tc)} + d$$

However, the above mentioned problem persists (9).

Another solution can be to consider that because of the nested calibration t_{max} is a constant, therefore (15) simply becomes:

$$s = (a + bt)e^{(-ct)} + d \tag{5.3}$$

Rewriting the above equation, b is obtained:

$$b = \frac{s - d}{te^{(-ct)}} - \frac{a}{t}$$

Plugging (5.1):

$$b = \frac{s - d}{te^{(-\frac{bt}{a+tb})}} - \frac{a}{t}$$

Unfortunately, once again, the problem remains.
In the end, even when plugging (5.2) in (5.3) the problem still remains:

$$s = (a + \frac{act}{1 - tc})e^{(-ct)} + d$$

These attempts lead to a mandatory adjustment of the main idea previously exhibited and are exposed in the following paragraphs.

5.2 Idea adjustment

Given the above mentioned considerations, it is necessary to rethink the main idea of the model reparameterization that has to be shifted from the recalibration based on (16) to an input form such:

$$a_x, c_x, t_{max}, d_x \tag{5.4}$$

This allows to work on the parameters that has been spotted that are essential for the new framework, i.e. c and t_{max} along with a and d that have a clear financial meaning, without facing problem as the one in the previous section.

Chapter 6

Abcd Framework+

6.1 Nested Calibration

The general idea at the basis of nested calibration is that the minimization problem can be split in two problems. The first minimization is only with respect to a certain variable k. Once fixed, it can act as an implied bound for the other parameters of the model. The second step is to internally minimize with respect to other model parameters. In the specific case of this research, it is interesting to fix t_{max} or c in order to follow the points defined in the "Relation between c and t_{max}" section. In the following for each test are followed two framework: the first it's about the calibration of absolute basis, while the second exploit the relative basis calibration. Differently from what has been done in [1], the calibration of the simple basis is abandoned, because the results with the continuous one are more sounds.

6.2 Calibrator

Before going through the specific cases it is interesting to study the mechanism of the calibrator. Given the market quote for different fixing dates, the calibrator, starting from the guesses of parameters, changes them in order to minimize the root mean square difference between model values $\hat{q}_i$ and market values q_i adjusted with some arbitrary weights w:

$$\sqrt{\frac{\sum_{i=1}^{n}((\hat{q}_i - q_i) * w)^2}{n}} \tag{6.1}$$

according to the idea that the model has to correctly reprice the market quote and not the legacy curve. Moreover, the guesses in our framework can be singularly externally fixed such that the final calibrated fixed parameters are equal to their guesses. Given the idea to apply a nested calibration this feature is fundamental, because it allows to fix externally the parameters. Therefore only in " fixed mode" it is possible to force the calibrator to maintain the fixed guess, otherwise it is only a particular one that following the rule of the calibrator will be changed during the calibration process.

..... it follows the new code explanation.....

Chapter 7

Fixing of c empirical analysis

7.1 Globally shared c

This section tests effects of c fixing, which grants that all relative basis are abcd, on the model fitting quality, for the continuous calibration with both incremental and not incremental calibration method. Given the spreadsheets used to get the results exhibited in the chapter 3, with a little modification it is possible to design the algorithm that allows to globally fix c.

7.1.1 Not incremental approach

Starting from the not incremental method available in the spreadsheet "BasisCalibration", in the spreadsheet named "*BasisCalibration_fixed_c*" the idea is to create a control panel which allows to work on all the other sheets. Therefore, the cells which contain c_x parameters, that feed each calibrator, take their value from a common cell, which is the guessed global c. Moreover, have been also fixed their boolean values which indicates whether or not the respective parameters should be calibrated during the calibration process. Successively, starting from the guessed c the Excel Solver has been requested to change c in order to minimize the quantity:

$$\sum_{j=1}^{4} \sqrt{\frac{\sum_{i=1}^{n} (\hat{q_{j,i}} - q_{j,i})^2}{n}} \tag{7.1}$$

where:

- q is the market quote;
- $\hat{q}$ is the repriced quote from model;

- j indicates the basis;
- i indicates the $i - basis$;

The implementation choices are:

1. guesses of the parameters have been chosen picking from the results of the previous paper, that exhibits the best fit as possible according with this framework, while for c have been manually tempted different values and in the end the empirical attempts lead to a value of 0.7, which avoid the abortion of the output;
2. if a curve can't be calibrated, the algorithm fails;
3. the algorithm has been tried for each available excel solver method;

The outputs follow:

	Output		
Solving method	GRG Non linear	Simplex LP	Evolutionary
Guessed c	0,7 V 0,498261576047114	0,498261576	0,498261576
Calibrated c	0,498261576	Linear Condition not satisfied	Error
Error function	7,41	-	-

Figure 7.1: c fixing output

As it is possible to see, the Solver works only with the "GRG non linear" algorithm, probably because the problem is highly not linear (yeah, not so smart consideration). However, considering that the distribution of the correction factor is the parameter that represents the goodness of the calibration in the abcd framework, the results are excellent:

6M Continuous Basis	
k max	1,02470
k min	0,99573
Max Error (bps)	4,79
Root Mean Square Error	2,16

3M Continuous Basis	
k max	1,02118
k min	0,99691
Max Error (bps)	3,05
Root Mean Square Error	1,36

1Y Continuous Basis	
k max	1,03197
k min	0,99460
Max Error (bps)	5,76
Root Mean Square Error	3,62

1M Continuous Basis	
k max	1,00168
k min	0,99953
Max Error (bps)	0,68
Root Mean Square Error	0,27

Figure 7.2: Correction factors k from not incremental calibration with global c

Range of k values is closed to 1 that is the best value because indicates that no correction are needed. Moreover, the statistics shows an interesting features for the following of the research:

6M Continuous Basis Parameters	Calibration	
	Continuous	Simple
a	0,0436%	0,0885%
b	0,0023	0,0021
c	0,498	0,498
d	0,1373%	0,1373%
a+d	0,1809%	0,2257%
Max Location	21-dic-17	23-set-17
a/b	0,186939453	0,429116034
1/c	2,006978834	2,006978834
T max th	1,820039381	1,5778628
T max from Location	1,825518833	1,583342252
Max Value	0,3263%	0,3258%

3M Continuous Basis Parameters	Calibration	
	Continuous	Simple
a	0,0097%	0,0221%
b	0,0011	0,0011
c	0,498	0,498
d	0,0980%	0,0980%
a+d	0,1077%	0,1201%
Max Location	26-gen-18	13-dic-17
a/b	0,086462611	0,20886817
1/c	2,006978834	2,006978834
T max th	1,920516223	1,798110664
T max from Location	1,925995675	1,803590116
Max Value	0,1847%	0,1846%

1Y Continuous Basis Parameters	Calibration	
	Continuous	Simple
a	0,0616%	0,2144%
b	0,0045	0,0036
c	0,498	0,498
d	0,1670%	0,1670%
a+d	0,2286%	0,3814%
Max Location	09-gen-18	22-lug-17
a/b	0,135647088	0,602401527
1/c	2,006978834	2,006978834
T max th	1,871331746	1,404577307
T max from Location	1,876811198	1,410056759
Max Value	0,5256%	0,5217%

1M Continuous Basis Parameters	Calibration	
	Continuous	Simple
a	0,0104%	0,0097%
b	-0,0001	-0,0001
c	0,498	0,498
d	0,0184%	0,0184%
a+d	0,0289%	0,0281%
Max Location	24-dic-18	08-dic-18
a/b	-0,82314956	-0,77903318
1/c	2,006978834	2,006978834
T max th	2,830128394	2,786012012
T max from Location	2,835607846	2,791491464
Max Value	0,0122%	0,0122%

Figure 7.3: Parameters from not incremental calibration with global c

for 6M, 3M and 1Y the $t_{max}(x)$ values are closed to the other and this may mean that a global shared value of t_{max} is a sane idea.

The only problem that arises is that with respect to the legacy curve, the new basis seems losing fitting on the legacy one. Anyway it should not be a problem in the extend that is valid the idea that the legacy curve brings with itself more noise than signal with respect to an abcd basis.

Graphically, it appears that the matter, already encountered in [1], about the shape of 1M remains:

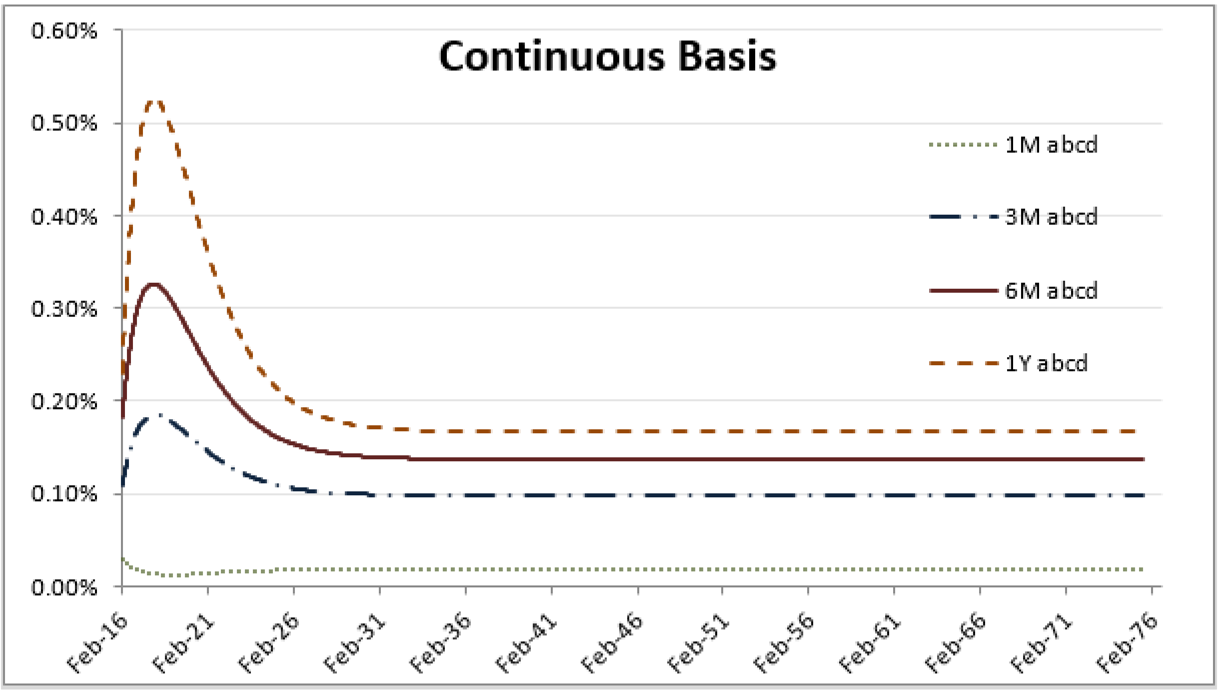

Figure 7.4: Absolute basis s_x from not incremental calibration

7.1.2 Incremental approach

Swapping to a incremental approach, the same modifications above reported are employed in order to get the searched result starting from the spreadsheet "*BasisCalibration_Incremental*".

The outputs follow:

	Working column	Output		
Solving method	GRG Non linear	GRG Non linear	Simplex LP	Evolutionary
Guessed c	0.498261576	0.498261576	0.498261576	0.498261576
Calibrated c	0.305782434	0.305782434	Linear Condition not satisfi	Error
Error function	3.40	3.40	-	-

Figure 7.5: c fixing output

As it is possible to appreciate, results come once again only from the "GRG non linear" excel optimization. However, considering that the distribution of the correction factor is what was looking for, the results are still excellent:

6M Continuous Basis	
k max	1,00941
k min	0,99733
Max Error (bps)	1,86
Root Mean Square Error	0,98

3M Continuous Basis	
k max	1,01074
k min	0,99769
Max Error (bps)	1,51
Root Mean Square Error	0,74

1Y Continuous Basis	
k max	1,00849
k min	0,99818
Max Error (bps)	4,36
Root Mean Square Error	1,35

1M Continuous Basis	
k max	1,00154
k min	0,99946
Max Error (bps)	0,80
Root Mean Square Error	0,33

Figure 7.6: Correction factors k from incremental calibration with global c

Range of k values is closed to 1 that is the best value for the calibration. Unfortunately, the statistics don't are as good as before:

Basis Parameters	Simple	Continuous
A	0,1502%	0,1246%
B	0,0014	0,0015
C	0,306	0,306
D	0,0871%	0,0871%
A+D	0,2373%	0,2117%
Max Location	09-mag-18	08-ago-18
a/b	1,065708441	0,819440891
1/c	3,270299043	3,270299043
T max th	2,204590601	2,450858152
T max from Locatior	2,205479452	2,454794521
Max Value	0,3219%	0,3222%

3M Simple	Calibration	
Basis Parameters	Simple	Continuous
A	0,0852%	0,0793%
B	0,0007	0,0007
C	0,306	0,306
D	0,0146%	0,0146%
A+D	0,0998%	0,0940%
Max Location	26-mar-18	09-mag-18
a/b	1,191271692	1,067864154
1/c	3,270299043	3,270299043
T max th	2,079027351	2,202434889
T max from Locatior	2,084931507	2,205479452
Max Value	0,1385%	0,1385%

12M Simple	Calibration	
Basis Parameters	Simple	Continuous
A	0,1155%	0,0853%
B	0,0009	0,0010
C	0,306	0,306
D	0,0174%	0,0174%
A+D	0,1330%	0,1027%
Max Location	06-feb-18	01-ago-18
a/b	1,321819106	0,838536558
1/c	3,270299043	3,270299043
T max th	1,948479936	2,431762484
T max from Locatior	1,953424658	2,435616438
Max Value	0,1749%	0,1756%

1M Simple	Calibration	
Basis Parameters	Simple	Continuous
A	0,0403%	0,0371%
B	0,0008	0,0008
C	0,306	0,306
D	0,0540%	0,0540%
A+D	0,0943%	0,0911%
Max Location	12-dic-18	27-dic-18
a/b	0,483273503	0,439030394
1/c	3,270299043	3,270299043
T max th	2,78702554	2,831268648
T max from Locatior	2,8	2,84109589
Max Value	0,1702%	0,1702%

Figure 7.7: Parameters from incremental calibration with global c

for 6M, 3M and 1Y the $t_{max}(x)$ values are not so closed to the other, but the $t_{max}(1M)$ is nearer to them , in the next chapter this issue will be addressed. Graphically, the obtained shapes make sense:

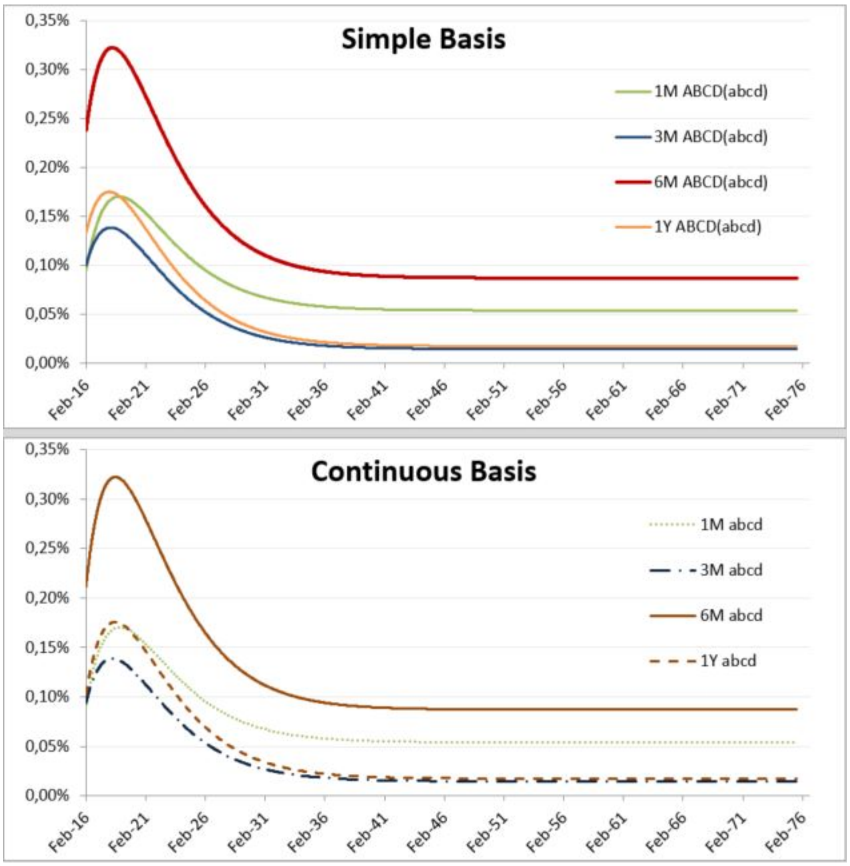

Figure 7.8: Relative basis from incremental calibration

Both the approach bring with them excellent results, because show that a relative abcd is not losing fitting power and is qualifying the entire scheme.

7.2 Basis shifting

Certified that also with a fixed value of c the model works, in the following is tested the capabilities of repricing market quote exploiting basis retrieved thanks to (4).

7.2.1 Not incremental approach

After globally calibrated the model, where relative basis for 3M, 12M and 1M on ON have been built according to the abcd framework as (3), the relative basis can be retrieved thanks to (4). In particular have been retrieved:

$$s_{x,6M} = s_y - s_{6M} \tag{7.2}$$

where: $x \in (3M, 12M, 1M)$. The parameters of $s_{x,6M}$ have been retrieved exploiting (2).

If $x > 6M$:

$$\begin{aligned} a_{x,6M} &= a_x - a_{6M}; \\ b_{x,6M} &= b_x - b_{6M}; \\ d_{x,6M} &= d_x - d_{6M}; \end{aligned}$$

Otherwise:

$$\begin{aligned} a_{x,6M} &= a_{6M} - a_x; \\ b_{x,6M} &= b_{6M} - b_x; \\ d_{x,6M} &= d_{6M} - d_x; \end{aligned}$$

Given the basis, for each one has been made the repricing of the market quotes, modelling the considered instantaneous forward rate as:

$$f_x = f_{ON} + s_{6M} \pm s_{x,6M} \tag{7.3}$$

where the sign of the relative basis depends on whether or not the considered x tenor is greater or smaller than 6M.

Moreover, according to the incremental principle, the 1M it is also repriced as a basis on 3M.

The results for the basis with respect to 6M are:

3M, 6M Continuous Basis	
k max	1.02116
k min	0.99689
Max Error (bps)	3.04
Root Mean Square Error	1.46

1Y, 6M Continuous Basis	
k max	1.02239
k min	0.99374
Max Error (bps)	7.56
Root Mean Square Error	5.27

1M, 6M Continuous Basis	
k max	1.00167
k min	0.99952
Max Error (bps)	0.75
Root Mean Square Error	0.28

Figure 7.9: Correction factors k from not incremental calibration with global c and relative basis

while for 1M from $s_{1M,3M}$ are:

1M, 3M Continuous Basis	
k max	1.00170
k min	0.99954
Max Error (bps)	1.82
Root Mean Square Error	0.68

Figure 7.10: 1M correction factors k from not incremental calibration with global c and relative basis $s_{1M,3M}$

The k are generally closed to 1, but the root mean square error and the maximum error in basis points for 12M are high. Therefore, it seems that the k factor range is not a good metric for understanding how much the repricing is good, then a better metric is needed to summarize and interpret the correction factors.

Looking at the parameters and statistics:

3M,6M Continuous Basis Parameters	Calibration	
	Continuous	Simple
a	0.0215%	0.0349%
b	0.0013	0.0012
c	0.498	0.498
d	0.0392%	0.0392%
a+d	0.0608%	0.0742%
Max Location	27-Dec-17	13-Nov-17
a/b	0.1687559	0.29116146
1/c	2.00697883	2.00697883
T max th	1.83822294	1.71581738
T max from Locatic	1.84370239	1.72129683
Max Value	0.1416%	0.1416%

1Y,6M Continuous Basis Parameters	Calibration	
	Continuous	Simple
a	0.1708%	0.1788%
b	0.0012	0.0010
c	0.498	0.498
d	0.0297%	0.0297%
a+d	0.2005%	0.2085%
Max Location	07-Oct-16	19-Apr-16
a/b	1.39251694	1.85927138
1/c	2.00697883	2.00697883
T max th	0.6144619	0.14770746
T max from Locatic	0.61994135	0.15318691
Max Value	0.2110%	0.2090%

1M,6M Continuous Basis Parameters	Calibration	
	Continuous	Simple
a	0.0339%	0.0438%
b	0.0025	0.0024
c	0.498	0.498
d	0.1189%	0.1189%
a+d	0.1528%	0.1627%
Max Location	08-Jan-18	23-Dec-17
a/b	0.13811392	0.18223031
1/c	2.00697883	2.00697883
T max th	1.86886491	1.82474853
T max from Locatic	1.87434436	1.83022798
Max Value	0.3132%	0.3131%

Figure 7.11: Parameters from not incremental calibration with global c and relative basis

the parameters make sense, the time of maximum are closed, the only problem appears for the 12M that seems to act in a different way w.r.t. the other (it is really likely that there is an implementation error that the author can't currently see).

Moreover for $s_{1M,3M}$:

1M,3M Continuous	Calibration	
Basis Parameters	Continuous	Simple
a	0.0124%	0.0172%
b	0.0012	0.0012
c	0.498	0.498
d	0.0796%	0.0796%
a+d	0.0920%	0.0969%
Max Location	04-Jan-18	20-Jan-18
a/b	0.1050444	0.14916078
1/c	2.00697883	2.00697883
T max th	1.90193443	1.85781805
T max from Locatic	1.8632975	1.90741389
Max Value	0.1715%	0.1715%

Figure 7.12: Parameters from not incremental calibration with global c and relative basis $s_{1M,3M}$

Also looking at graphical comparison (excluding $s_{1M,3M}$):

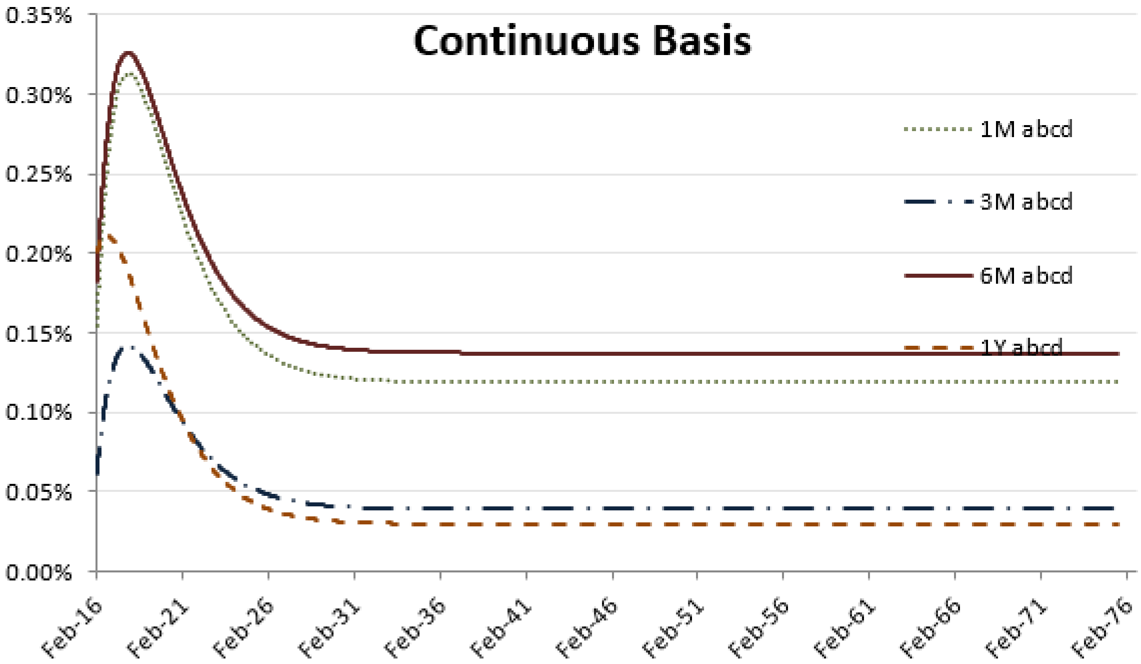

Figure 7.13: Absolute s_{6M} and its respective $s_{x,6M}$ relative basis

the basis dominance is respected and it makes sense that the $s_{1M,6M}$ is next to the s_{6M}.

7.2.2 Incremental approach

After globally calibrated the model, where relative basis for 3M, 12M and 1M on ON have been built according to the abcd framework as (4), the absolute basis can be retrieved. In particular have been retrieved:

$$s_x = s_{x,6M} + s_{6M} \tag{7.4}$$

where: $x \in (3M, 12M)$.

While for 1M because of incremental principle:

$$s_x = s_{x,3M} + s_{3M} \tag{7.5}$$

Where: $s_{3M} = s_{x,3M} + s_{3M}$

The parameters of s_x have been retrieved exploiting (2). If $x > y$:

$$\begin{aligned} a_x &= a_{x,y} + a_y; \\ b_x &= b_{x,y} + b_y; \\ d_x &= d_{x,y} + d_y; \end{aligned}$$

Otherwise:

$$\begin{aligned} a_x &= a_y - a_{x,y}; \\ b_x &= b_y - b_{x,y}; \\ d_x &= d_y - d_{x,y}; \end{aligned}$$

Where: $y \in (6M, 3M)$ in this specific case.

Given the basis, for each one has been made the repricing of the market quotes, modelling the considered instantaneous forward rate as:

$$f_x = f_{ON} + s_y \pm s_{x,y} \tag{7.6}$$

where the sign of the relative basis depends on whether or not the considered x tenor is greater or smaller than y.

The results are:

3M Continuous Basis	
k max	1,01049
k min	0,99748
Max Error (bps)	0,95
Root Mean Square Error	1,11

1Y Continuous Basis	
k max	1,00537
k min	0,99741
Max Error (bps)	4,01
Root Mean Square Error	2,29

1M Continuous Basis	
k max	1,00131
k min	0,99927
Max Error (bps)	1,57
Root Mean Square Error	0,94

Figure 7.14: Correction factors k from incremental calibration with global c and absolute basis

The k are generally closed to 1, but the root mean square error and the maximum error in basis points for 12M are still high. Therefore, it seems that the k factor range is not a good metric for understanding how much the repricing is good, then a better metric is needed to summarize and interpret the correction factors.

Looking at the parameters and statistics:

6M Continuous Basis Parameters	Calibration	
	Continuous	Simple
a	0,1246%	0,1502%
b	0,0015	0,0014
c	0,306	0,306
d	0,0871%	0,0871%
a+d	0,2117%	0,2373%
Max Location	08-ago-18	10-mag-18
a/b	0,819440891	1,065708441
1/c	3,270299043	3,270299043
T max th	2,450858152	2,204590601
T max from Location	2,456337604	2,210070053
Max Value	0,3222%	0,3219%

3M Continuous Basis Parameters	Calibration	
	Continuous	Simple
a	0,0650%	0,0708%
b	0,0007	0,0007
c	0,306	0,306
d	0,0724%	0,0724%
a+d	0,1374%	0,1433%
Max Location	26-giu-18	12-mag-18
a/b	0,936359388	1,059766927
1/c	3,270299043	3,270299043
T max th	2,333939654	2,210532116
T max from Location	2,339419106	2,216011568
Max Value	0,1836%	0,1836%

1Y Continuous Basis Parameters	Calibration	
	Continuous	Simple
a	0,2657%	0,3231%
b	0,0023	0,0020
c	0,306	0,306
d	0,1045%	0,1045%
a+d	0,3702%	0,4276%
Max Location	04-apr-18	10-ott-17
a/b	1,16373285	1,647015399
1/c	3,270299043	3,270299043
T max th	2,106566192	1,623283644
T max from Location	2,112045644	1,628763096
Max Value	0,4966%	0,4950%

1M Continuous Basis Parameters	Calibration	
	Continuous	Simple
a	0,0247%	0,0238%
b	-0,0001	-0,0001
c	0,306	0,306
d	0,0184%	0,0184%
a+d	0,0432%	0,0422%
Max Location	15-mar-21	27-feb-21
a/b	-1,78425548	-1,74001238
1/c	3,270299043	3,270299043
T max th	5,054554527	5,010311419
T max from Location	5,060033979	5,015790871
Max Value	0,0087%	0,0087%

Figure 7.15: Parameters from incremental calibration with global c and absolute basis

the parameters make sense, the time of maximum are closed, the problem with 1M still appears.

Also looking at graphical comparison:

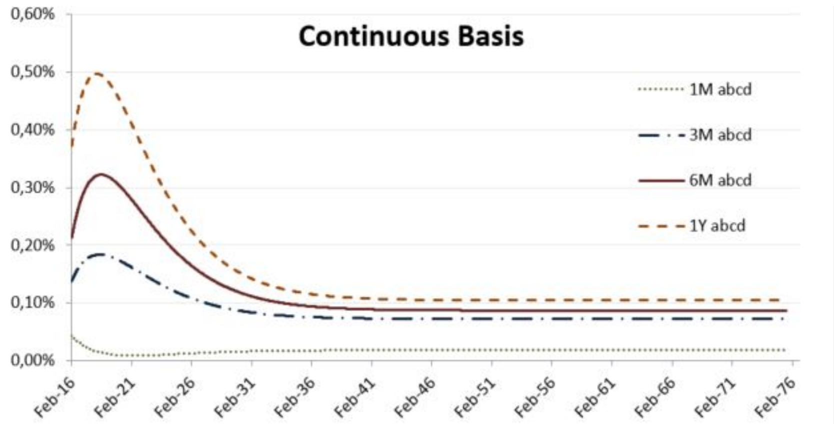

Figure 7.16: Absolute basis from relative ones

the basis dominance is respected and it makes sense that problem with 1M still remains for the absolute basis with both the approach.

What has been discovered in this chapter qualifies the model and allows to close the second point of the research schedule. Moreover,until now the previous framework has been exploited as much as possible, but given the goodness of the outputs it makes sense to implement a specific solution in order to give a tools to perform this calibration without recurring to the Solver help.

Chapter 8

Fixing of the time of maximum: an empirical analysis

Before to implement the approach, the parametric form of b, for a generic tenor x, needs to be retrieved:

$$b_x = \frac{a_x c_x}{1 - t_{max} c_x} \tag{8.1}$$

As seen in the chapter chapter 7, the first idea is to exploit excel capabilities in order to perform our analysis. The first idea is to choose t_{max} through the Solver, while leaving the choice of a, c and d to the Calibrator and posing a condition on the outputs. The algorithm works as follows:

1. the Solver minimize (7.1) with respect to the time;
2. guesses enters the calibrator allowing it to change them (therefore the respective indicator has been set as "FALSE" in the framework);
3. multiple conditions on the Solver's output parameter have been posed such that $\hat{b}_x$, for each $x \in (1M, 3M, 6M, 12M)$ it's equal to:

$$\hat{b}_x = \frac{\hat{a}_x \hat{c}_x}{1 - t_{max} \hat{c}_x} \tag{8.2}$$

The solver is set as follows:

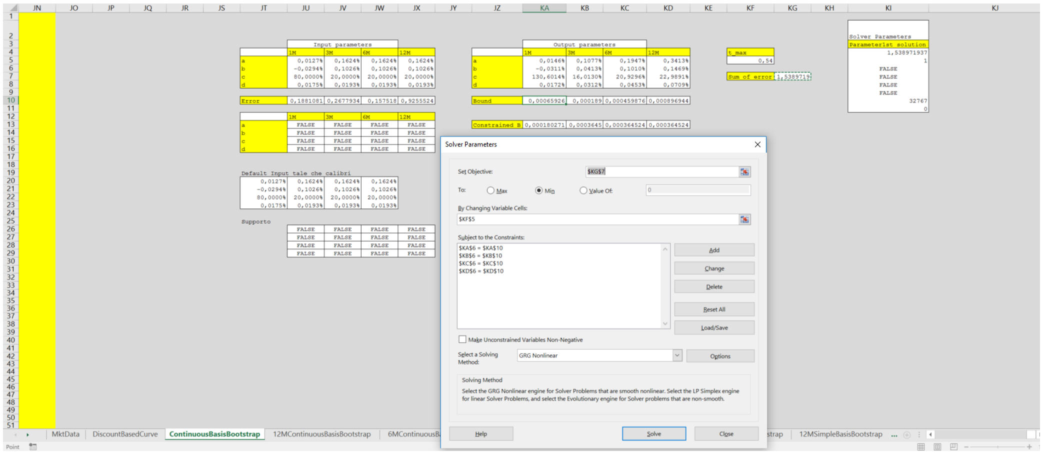

Figure 8.1: Fixed T- First Problem

and the output is:

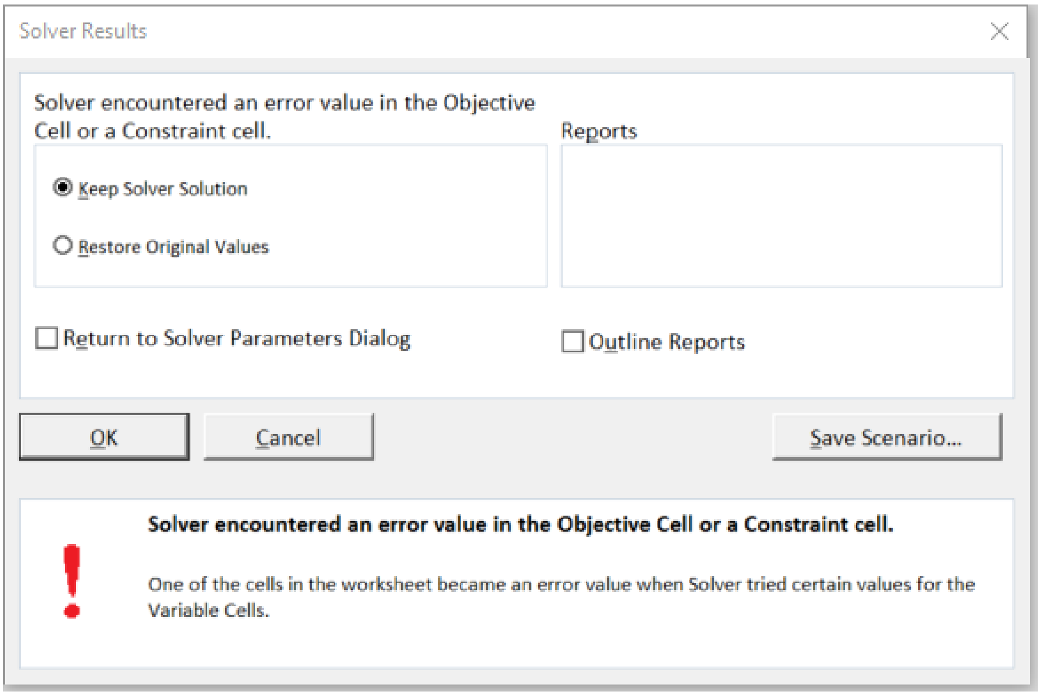

Figure 8.2: Fixed T-First Output

These results come from a not incremental approach, but are valid for both [1], and the problem are commonly shared too:

- given that the calibrator ignores the bounds that the solver has to respect, each time that the Solver refuses the calibrator's solution, the calibrator comes once again to the same solution and they conflict.
- the algorithm doesn't converge to a solution, therefore the input points and the algorithm itself need to be changed, the matter is how to deal with this changes;

Given these results, the nested calibration can't be put on practice through the solver, therefore the decided strategy aims to modify the source code.

Considering the will to built a global calibrator such that is possible also to calibrate with respect to c, in order to guarantee the future extensibility of the framework, the best solution it is to create a framework which allows to fix or not the parameters internally an globally fixing a variable, that , can change also dynamically in the framework for example with t_max that is a function of the other parameters concurring in specifying the model.

[1] Nested incremental Calibration fixed t.xlsx and Nested Calibration fixed t.xlsx

Bibliography

[1] Ferdinando M. Ametrano, Luigi Ballabio, Paolo Mazzocchi *The abcd of Interest Rate Basis Spread (July 2016)*. Located at https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2696743

[2] Luigi Ballabio *Implementing QuantLib (2017-08-21)*. Located at https://leanpub.com/implementingquantlib

[3] Damiano Brigo, Fabio Mercurio *Interest Rate Models - Theory and Practice (2006)*. Located at https://link.springer.com/book/10.1007/978-3-540-34604-3

[4] Martin Fowler *UML Distilled. A Brief Guide to the Standard Object Modeling Language (2003)*. Located at https://www.martinfowler.com/books/uml.html