Implementing BGM

Raphael Yan

April 19, 2005

Abstract

In this study, the implementation of the BGM model is investigated. Theoretical background and numerical techniques are presented. Derivatives on Libor are priced in this model, and numerical results are compared to existing literature.

Contents

1 Introduction

Historically, interest rate modelling begins with a specification of the instantaneous short rate, such as the well-known Vasicek or CIR models. While these short rate models give closed-form solutions for prices of bonds and bond options, they became awkward to use when pricing derivatives on discretely compounded forward rates. For example, consider a cap, which is a collection of call options on individual forward Libor rates (called caplets). The most obviously natural thing to do would be to use the Black-Scholes formula to price a caplet [1], by treating the discretely compounded forward Libor rate as geometric Brownian motion. This has been and still is standard market practice today. However, this is inconsistent with all short rate model, in the sense that no short rate model will result in a lognormal forward rate (see [2] § 6.2).

In an effort to justify the use of Black's formula for Libor-based derivates, researchers began to directly model discretely compounded rates as lognormal processes. In the case of forward rates, Brace, Gatarek, and Musiela [3] derived the no-arbitrage conditions under the celebrated HJM framework. This became known as the BGM model, otherwise known as Libor Market Model, or lognormal forward Libor model, and is the model being investigated in this study.

The BGM model is described in Section 2. Simulation of the forward rates is explained in Section 3. Section 4 verifies that Monte-Carlo simulated caplet prices are close to the exact Black's caplet formula. In Section 5 European swaptions are examined. For Bermudan swaptions, the technique by Longstaff and Schwartz is employed, which is desribed in Section 6. Section 7 presents numerical results of the swaption prices, and compares them with those reported in [4]. Section 8 summarizes what is accomplished in this study.

[1]The version of the formula when the underlying is a forward price is known as Black's formula [1]

2 The BGM model

We will start with a finite set of dates, called the tenor structure

$$0 = T_0 < T_1 < \cdots < T_N < T_{N+1}$$

Let $L(t, T_n, T_{n+1})$ be the forward Libor rate, as seen at time t, for the period $[T_n, T_{n+1}]$. We will abbreviate this to $L_n(t)$. We assume the accrual period $T_{i+1} - T_i$, for all i, is a constant equal to δ. Let $B_n(t)$ be the time t price of a zero coupon bond (or simply "bond") that matures at T_n. We know

$$L_i(t) = \frac{1}{\delta}\left(\frac{B_i(t)}{B_{i+1}(t)} - 1\right), \quad i = 1, \ldots, N$$

On a tenor date T_i, a bond $B_n(T_i)$ that has not matured is computed by

$$B_n(T_i) = \prod_{j=i}^{n-1} \frac{1}{1 + \delta L_j(T_i)} \tag{1}$$

In the BGM model, the forward rates are lognormal, and the SDE is

$$\frac{dL_n(t)}{L_n(t)} = \mu_n(t)dt + \lambda_n(t)dW_t \quad n = 1, \ldots, N \tag{2}$$

where W_t is a d-dimensional Brownian motion, $\lambda_n(t)$ is the volatility, and $\mu_n(t)$ is the drift. In what follows, $d = 1$ for simplicity. The drift is determined by no-arbitrage conditions; but first we have to specify a numeraire.

2.1 Numeraire

From standard theory, we know the time t price, $X(t)$, of a time T-measurable contingent claim is given by

$$X(t) = N(t)E\left[\frac{X(T)}{N(T)}|\mathcal{F}_t\right] \tag{3}$$

where $N(t)$ is the numeraire, and the expectation is taken with respect the measure $\mathcal{Q}$ that makes every tradable assets $\mathcal{Q}$-martingales. This $\mathcal{Q}$ is what is meant by *measure associated with the numeraire*. The choice of numeraire is arbitrary, and is chosen for convenience. However, it determines the measure, and hence the drifts $\mu_n(t)$ of the forward rates under the associated measure.

2.2 Forward measure

To motivate the notion of forward measure, consider the time t price of a caplet that pays $\delta(L_n(T_n)-K)^+$ at time T_{n+1}. T_n is called the fixing, or reset date, T_{n+1} is the payment date, and this caplet price is denoted by $C_n(t)$. The most convenient numeraire for pricing this caplet will be $B_{n+1}(t)$, since then by Eq (3) the price is

$$C_n(t) = B_{n+1}(t)E^{n+1}[\delta(L_n(T_n) - K)^+] \tag{4}$$

where $E^{n+1}[\cdot]$ is the expectation with respect to the so-called T_{n+1}*-forward measure* $\mathcal{Q}^{n+1}$. In other words, the T_{n+1} forward measure is the measure associated with the numeraire being the bond that matures at T_{n+1}.

Now consider

$$L_n(t)B_{n+1}(t) = \frac{1}{\delta}(B_n(t) - B_{n+1}(t))$$

This is a tradable asset, so it follows that $L_n(t)$ has to be a $\mathcal{Q}^{n+1}$-martingale, implying that the dynamics of $L_n(t)$ is

$$dL_n(t) = L_n(t)\lambda_n(t)dW_t$$

In words, $L_n(t)$ is a driftless geometric Brownian motion under $\mathcal{Q}^{n+1}$ for all n. This makes Eq (4) easily computed; it will be given in Section 4.

3 Simulation

From now on, we will choose the bond that matures on the final date of the tenor T_{N+1}, as the numeraire. The measure associated with this numeraire is called the *terminal measure*. We know $L_N(t)$ is driftless under this measure, and we want $L_n(t)$ for $1 \leq n \leq N-1$ to be driftless under their respective forward measures.

However, to simulate all forward rates simultaneously, it is necessary to simulate all of them under one measure. This means we need the dynamics (the drift term) for all $L_n(t)$ under the terminal measure. The main contribution of [3] is the characterization of the arbitrage-free dynamics; we state the SDE of the forward rates under the terminal measure:

Proposition 1 *Under the terminal measure, the drifts of $L_n(t)$ for $1 \leq n \leq N-1$ is*

$$\mu_n(t) = -\sum_{k=n+1}^{N} \frac{\delta L_k(t)\lambda_n(t)^2}{1+\delta L_k(t)} \tag{5}$$

This is *Proposition 6.3.1* in [2], where a proof is also given.

In logarithmic form, the SDE for $L_n(t)$ Eq (2) is

$$d\log L_n(t) = (\mu_n(t) - \frac{1}{2}\lambda_n(t)^2)dt + \lambda_n(t)dW_t$$

For a time step size h, the discretized evolution of L_n is [5]

$$L_n((i+1)h) = L_n(ih)\exp[(\mu_n(ih) - \frac{1}{2}\lambda_n(ih)^2)h + \lambda_n(ih)\sqrt{h}Z_{i+1}]$$

where

$$\mu_n(ih) = -\sum_{k=n+1}^{N} \frac{\delta L_k(ih)\lambda_n(ih)^2}{1+\delta L_k(ih)}$$

and Z_i are independent standard normal random variables. In all subsequent simulations, we choose the step size h to be the same as δ, and we assume that the volatility $\lambda_n(t)$ is constant for all n and t. On Listing 10.1, the C code for simulation is given.

4 Exact Caplet Prices

We now compare simulated caplet prices to their exact values. From Eq (4), the Black's formula for this caplet is [5]

$$C_n(0) \equiv C_{Black}(\bar{\lambda}_n, K, L_n(0), B_{n+1}(0), T_n)$$

where

$$C_{Black}(\sigma, K, r, b, T) = \delta b\left[r\Phi(d_1) - K\Phi(d_2)\right],$$
$$d_1 = \frac{\log(r/K) + \sigma^2 T/2}{\sigma\sqrt{T}}$$
$$d_2 = \frac{\log(r/K) - \sigma^2 T/2}{\sigma\sqrt{T}}$$

$$\bar{\lambda}_n = \sqrt{\frac{1}{T_n}\int_0^{T_n} \lambda_n(t)^2\, dt}$$

If we wish to compute the caplet prices from simulation, we can use Eq (3):

$$C_n(0) = B_{N+1}(0)\,\delta\, E\left[\frac{(L_n(T_n) - K)^+}{B_{N+1}(T_{n+1})}\right]$$

where the expectation E means taking average.

The C code for computing caplet prices, using both exact formula and simulation, are given in Listing 10.2. For initial conditions of $L_n(0)$, we assume the initial zero coupon bond prices $B_n(0)$ are such that

$$B_n(0) = \exp(-0.05 \times T_n)$$

We take $\delta = 0.25$ with $N = 19$, so the caplets have maturities from 3 months to 5 years. All caplets are at-the-money, meaning the strike K for C_n is $L_n(0)$.

The result is given on Table 1. The prices are given in basis points. It can be seen that with 10000 simulation paths, the worst error is no more than 1 bp.

5 European Swaptions

We will now describe the calculation of European swaptions. Details are given in [2]. A swap is a contract that swaps payments between two parties on a given set of future dates. Party A is said to enter a *pay-fixed* swap, if she pays a predetermined amount K on $\{T_{\alpha+1}, \ldots, T_\beta\}$, in exchange for a floating amount $\delta L(\,\cdot\,, T_\alpha, T_{\alpha+1}), \ldots, \delta L(\,\cdot\,, T_{\beta-1}, T_\beta)$, where the Libor rates reset on $\{T_\alpha, \ldots, T_{\beta-1}\}$. The length of the swap, $T_\beta - T_\alpha$, is known as the tenor of the swap.

To make this constract worth zero at time 0, it can be shown that

$$K \equiv S_{\alpha,\beta}(0) = \frac{B_\alpha(0) - B_\beta(0)}{\sum_{i=\alpha+1}^{\beta} \delta B_i(0)} \tag{6}$$

The amount K is called the *swap rate* of the swap. This rate is denoted by $S_{\alpha,\beta}(0)$ for future use.

A European swaption gives the holder the right to enter a swap at a particular swap rate K_s, at the maturity of the swaption, usually at T_α. It can be shown that the payoff of a European swaption at T_α is

$$euro \equiv \left(\delta \sum_{i=\alpha+1}^{\beta} B_i(T_\alpha)(L_{i-1}(T_\alpha) - K_s)\right)^+ \tag{7}$$

This amount will be paid at $T_{\alpha+1}$. Therefore, the time 0 price of this swaption under the terminal measure is computed by

$$B_{N+1}(0)E\left[\frac{euro}{B_{N+1}(T_{\alpha+1}))}\right]$$

The code is given in Listing 10.3. We will always assume $T_\beta = T_{N+1}$. The numerical results will be given in Section 6, with results from Bermuadan swaptions shown together.

6 Bermudan Swaptions

We are now ready to look at Bermudan swaptions. As opposed to the European case where exercise can only take place on a fixed given date, Bermudan swaptions give the holder the right to enter into a swap on any one of a set of dates, ususally on the reset dates $\{T_\alpha, \dots, T_{\beta-1}\}$ of the underlying swap.

We will focus on the version of Bermudan swaption known as *co-terminal* [8], which means the tenor of the underlying swap shrinks as each exercise date is passed. In other words, the maturity date of the underlying swap constract has a fixed T_β. This is in contrast to the *fixed-maturity* version, where the swaps have the same tenor on each exercise date.

The tenor structure for Bermudan swaptions is denoted by $xNCy$, where x is the total length of the swaption, and y is the no-call period. In terms of $\{T_\alpha, \dots, T_\beta\}$, y is T_α and x is T_β. For example, if the contract is $5NC1$, and $\delta = 0.5$, then $\alpha = 2$ and $\beta = 10$.

For all financial derivates with an early exercise feature, the main question to answer is to determine whether exercising today will generate more cash flow than the expected cash flow from continuing [6]. Call the cash flow from exercising e_i if the swaption is exercised on T_i, and call the expected cashflow from holding h_i. e_i is readily computed as in Eq (7)

$$e_i = \left(\delta \sum_{n=i+1}^{\beta} B_n(T_i)(L_{n-1}(T_i) - K) \right)^+$$

where K is the strike of the Bermudan swaption.

h_i is zero if $i = N$, on the last date to exercise the swaption. For $i < N$, $h_i = B_{N+1}(T_i)E[B_{N+1}(T_{i+1})^{-1} \max\{h_{i+1}, e_{i+1}\}]$. Now h_i depends on the swap rate at T_i, which depends on all the forward rates still alive at T_i. Given the high dimensionality of the collection of forward rates, it is obviously undesirable to construct trees for $L_n(t)$ and compute h_i by the well-known backward induction. This is where the *Longstaff and Schwartz* method [7] comes in.

6.1 Longstaff and Schwartz

Essentially, the Longstaff and Schwartz method (LSM, but it is "least-square Monte Carlo" according to [7]) is to approximate h_i by least squares, using cross-sectional information from the simulation. The h_i is the expectation, conditioning on the forward rates at time T_i, of the cash flow from holding on and not exercising.

In the context of pricing Bermudan swaptions in BGM, this method goes as follows (for details, see [2], Ch. 10):

1. simulate `nit` paths of $L_n(t)$;

2. for the `it` path, store swap rate at time T_i as `s[it][i]`, payoff from exercise as `e[it][i]`, numeraire as `numer[it][i]`;

3. Now we compute the cash flow for each time and each path. At time N the cash flow is the amount from exercising, so `cf[it][N]=e[it][N]` for all `it`.

4. Set `n=N-1`.

5. Identify which paths are in-the-money; that is, test whether `e[it][n]>0`, and set the flag `itmflag[it][n]` to be 1 if `e[it][n]>0` and 0 otherwise

6. For the in-the-money paths, store the swap rates in the array `x`. Store the *discounted* cash flow, `cf[it][n+1]*numer[it][n]/numer[it][n+1]`, in the array `y`.

7. Now we regress `y` on `x` by least square. We are fitting the model $y = c_0 + c_1x + \cdots + c_rx^r$ by minimizing the sum of squared errors.

   To find the coefficients, we use matrix notation as follows. y is a column vector obtained from `y` in the previous step. X is a matrix with the first column being all 1, and other columns depend on the order of regression. Usually the order is 1 or 2; from experiments the order has minimal effect, so we take the order to be 2. This means the second column of X is x, and the third column of X is x^2 (component-wise exponentiation). c is the vector of coefficients; for second order regression it has components c_0, c_1 and c_2. From standard linear algebra, we know $c = (X^TX)^{-1}X^Ty$.

8. Once we have the coefficients c_0, c_1 and c_2, we can compare the cashflow from exercising and continuing. The expected cashflow for each in-the-money path from continuing is $h_n = c_0 + c_1x + c_2x^2$. By comparing it with the cashflow from immediate exercise, we can identify whether to exercise or not. Set the exercise flag `eflag[it][n]` to be 1 if optimal to exercise, 0 otherwise.

9. Since a Bermudan option can be exercised once, if `eflag[it][n]`=1, set `eflag[it][k]`=0 for all $n \leq k \leq N$.

10. Now that we know whether to exercise or not for path `it` at time `n`, set the cashflow `cf[it][n]` to be `e[it][n]` if exercising, and set it to be `cf[it][n+1]*numer[it][n]/numer[it][n+1]` if not.

11. Decrement `n` and start over from step 5 until `n=alpha`.

12. Now that we have the expected cash flow for all paths at time T_α, that is, `cf[it][alpha]` for all `it`, we can discount to time 0 and take average.

The code is given in Listing 10.4. The functions `S` is to compute swaprates, `condexp` is to compute conditional expectation, and `regress` is to compute the coefficients of the regression.

7 Result

We now present the simulated European and Bermudan swaption prices in Table 2. They are compared to the numbers shown in [4], Table 3.

Note In [4], the European swaption prices are computed using Monte-Carlo, and Bermudan swaption prices are computed using LSM, so they are computed exactly the same way as done here. As their title suggests, their contribution is to speed up the simulation of the forward rates.

The parameters are $K = 5.06978\%$, $\delta = 0.5$, $\lambda = 0.15$. Again, the initial forward rates $L_n(0)$ are such that

$$B_n(0) = \exp(-0.05 \times T_n)$$

for all n. The row PPR is from [4] (two decimal places are given); the second row are my results. The notation $xNCy$ for Bermudan swaptions is explained before; in case of European swaptions, it means the swaption expires after y years.

8 Summary

The BGM model for modeling Libor forward rates is studied. Theoretical background is given, and three types of derivatives (caplets, European swaptions, Bermudan swaptions) are priced. Caplet prices are computed using Monte-Carlo simulations, and they are compared to the exact solution (Black's formula). European and Bermudan swaptions are also simulated, where the early exercise feature of Bermudan swaptions are handled using the Longstaff-Schwartz method (LSM). These numerical results are compared to that found in the literature. All C codes are given.

9 Tables

n	Exact	Simulated
1	4.89	4.90
2	6.83	6.84
3	8.26	8.46
4	9.41	9.50
5	10.39	10.30
6	11.23	11.15
7	11.98	12.00
8	12.64	12.60
9	13.24	13.35
10	13.77	14.01
11	14.26	14.36
12	14.70	14.77
13	15.11	15.17
14	15.48	15.58
15	15.81	16.05
16	16.12	16.29
17	16.41	16.60
18	16.66	16.99
19	16.90	17.08

Table 1: Caplet prices: exact vs simulated

	Bermudan	European
PPR 2NC1	28.85 28.932156	26.88 26.952090
PPR 3NC1	62.78 62.257950	52.92 51.823018
PPR 4NC1	101.51 99.686210	78.77 76.159858
PPR 4NC3	43.59 42.290254	42.55 41.245924
PPR 5NC1	137.95 141.231511	99.31 100.803660
PPR 5NC3	86.75 86.972307	80.83 81.312193
PPR 6NC1	179.48 182.725274	123.36 121.351069
PPR 6NC3	136.43 132.778107	123.06 119.065604
PPR 6NC5	50.79 50.101503	50.09 49.375011
PPR 7NC1	221.38 224.840564	140.66 142.095048
PPR 7NC3	177.11 177.229566	153.71 153.952364
PPR 7NC5	100.59 100.013132	96.57 96.076194
PPR 8NC1	266.35 273.089711	161.00 161.524309
PPR 8NC3	226.94 225.368319	190.98 187.705001
PPR 8NC5	151.13 149.612847	140.95 140.051147
PPR 8NC7	53.70 52.111712	53.12 51.575026

Table 2: Comparison between PPR and my swaption prices

10 Listings

10.1 Listing 1: Simulation

```
void SimulateTerminal(double** L,double constlam,
                      int N,double del,double h,long idum){
  int printpath=0;
  int j,k,i;
  double z,mu;
  for(j=0;j<=N-1;j++){
    z=gasdev(&idum); //printf("%f\n",z);
    for(i=0;i<=j;i++)
      L[i][j+1]=L[i][j];
    for(i=j+1;i<=N;i++){
      mu=0.0;
      for(k=i+1;k<=N;k++)
        mu-=(del*DSQR(constlam)*L[k][j])/(1.0+del*L[k][j]);
      L[i][j+1]=L[i][j]*exp((mu-0.5*DSQR(constlam))*h
                            +constlam*sqrt(h)*z);
    }
  }
}
```

10.2 Listing 2: Caplet

```
double cblack(double sig,double k,double r,double b,
              double T,double del){
  double d1=(log(r/k)+0.5*DSQR(sig)*T)/(sig*sqrt(T));
  double d2=(log(r/k)-0.5*DSQR(sig)*T)/(sig*sqrt(T));
  return del*b*(r*cumnor(d1)-k*cumnor(d2));
}
void ExactCaplet(double* caplet,double** L,double constlam,
                 int N,double* T,double* K,double del){
  int n,j;
  double* P=dvector(1,N+1);
  for(n=1;n<=N+1;n++){
    P[n]=1.0;
    for(j=0;j<=n-1;j++)
      P[n]*=1.0/(1.0+del*L[j][0]);
  }
  for(n=1;n<=N;n++)
    caplet[n]=cblack(constlam,K[n],L[n][0],P[n+1],T[n],del);
}
void SimCaplet(double* caplet,double** L,double constlam,int N,
               double* T,double* K,double del,double h,
               unsigned long niter){
  int n,j;
  unsigned long iter;
  double num=1.0,den=1.0;
  for(j=0;j<=N;j++)
    num*=1.0/(1.0+del*L[j][0]);
  for(n=1;n<=N;n++)
    caplet[n]=0.0;
  for(iter=1;iter<=niter;iter++){
    SimulateTerminal(L,constlam,N,del,h,-iter);
    for(n=1;n<=N-1;n++){
      den=1.0;
      for(j=n+1;j<=N;j++)
        den*=1.0/(1.0+del*L[j][n+1]);
      caplet[n]+=(num*del*pos(L[n][n]-K[n])/den)/niter;
    }
    caplet[N]+=num*del*pos(L[n][n]-K[n])/niter;
  }
}
```

10.3 Listing 3: European swaption

```
double EuropeanSwaption(int alpha,double** L,double constlam,int N,
                        double* T,double K,double del,double h,
                        unsigned long niter){
  int i,j;
  unsigned long iter;
  double num=1.0,den=1.0,ans,avg,std,swaption;
  for(j=0;j<=N;j++)
    num*=1.0/(1.0+del*L[j][0]);
  double Bi;
  avg=0.0; std=0.0;
  for(iter=1;iter<=niter;iter++){
    SimulateTerminal(L,constlam,N,del,h,-iter);
    den=1.0;
    for(j=alpha+1;j<=N;j++)
      den*=1.0/(1.0+del*L[j][alpha+1]);
    swaption=0.0;
    for(i=alpha+1;i<=N+1;i++){
      Bi=1.0;
      for(j=alpha;j<=i-1;j++)
        Bi*=1.0/(1.0+del*L[j][alpha]);
      swaption+=Bi*del*(L[i-1][alpha]-K);
    }
    ans=num*pos(swaption)/den;
    avg+=ans/niter;
  }
  return avg;
}
```

10.4 Listing 4: Bermudan swaption

```
double S(int alpha,int beta,double* P,int N,double del){
  double den=0.0;
  int i;
  for(i=alpha+1;i<=beta;i++)
    den+=del*P[i];
  return (P[alpha]-P[beta])/den;
}
double condexp(double* coef,int order,double x){
  double ret=coef[0];
  int io;
  for(io=1;io<=order;io++)
    ret+=coef[io]*pow(x,io);
  return ret;
}
double BermudanSwaption(int alpha,double** L,double constlam,int N,
                        double* T,double K,double del,double h,
                        unsigned long nit,int payrec){
  int n,i,j;
  unsigned long it;
  double* B;
  double **s=dmatrix(1,nit,alpha,N);
  double **e=dmatrix(1,nit,alpha,N);
  double **numer=dmatrix(1,nit,alpha,N);
  for(it=1;it<=nit;it++){
    SimulateTerminal(L,constlam,N,del,h,-it);
    for(i=alpha;i<=N;i++){
      B=dvector(i,N+1);
      B[i]=1.0;
      for(n=i+1;n<=N+1;n++){
        B[n]=1.0;
        for(j=i;j<=n-1;j++)
          B[n]*=1.0/(1.0+del*L[j][i]);
      }
      s[it][i]=S(i,N+1,B,N,del);
      e[it][i]=0.0;
      for(n=i+1;n<=N+1;n++){
        if(payrec==PAYFIXED)
          e[it][i]+=B[n]*del*(L[n-1][i]-K);
        else
          e[it][i]+=B[n]*del*(K-L[n-1][i]);
      }
      e[it][i]=pos(e[it][i]);
      numer[it][i]=1.0;
      for(n=i;n<=N;n++)
        numer[it][i]*=1.0/(1.0+del*L[n][i]);
      free_dvector(B,i,N+1);
    }
  }
  double *x,*y;
  double** cf=dmatrix(1,nit,alpha,N);
  int** itmflag=imatrix(1,nit,alpha,N);
  int** eflag=imatrix(1,nit,alpha,N);
  int iitm,nitm,nn;
  int order=2; double* coef=dvector(0,order);
  for(it=1;it<=nit;it++)
    cf[it][N]=e[it][N];
  for(n=N-1;n>=alpha;n--){
    nitm=0;
    for(it=1;it<=nit;it++){
      if(e[it][n]>0.0){
        itmflag[it][n]=1; nitm++;
      }else{
        itmflag[it][n]=0;
      }
    }
    x=dvector(1,nitm); y=dvector(1,nitm);
    iitm=1;
    for(it=1;it<=nit;it++){
      if(itmflag[it][n]){
        x[iitm]=s[it][n];
        y[iitm]=cf[it][n+1]*numer[it][n]/numer[it][n+1];
        iitm++;
      }
    }
    regress(x,y,nitm,order,coef);
    iitm=1;
    for(it=1;it<=nit;it++){
      if(itmflag[it][n]){
        if(e[it][n]>condexp(coef,order,x[iitm])){
          eflag[it][n]=1;
          for(nn=n+1;nn<=N;nn++)
            eflag[it][nn]=0;
        }else{
            eflag[it][n]=0;
        }
        iitm++;
      }else{
        eflag[it][n]=0;
      }
    }
    free_dvector(x,1,nitm); free_dvector(y,1,nitm);
    for(it=1;it<=nit;it++){
      if(eflag[it][n])
        cf[it][n]=e[it][n];
      else
        cf[it][n]=cf[it][n+1]*numer[it][n]/numer[it][n+1];
    }
  }
  double disc,ans,avg=0.0,num=1.0;
  for(j=0;j<=N;j++)
    num*=1.0/(1.0+del*L[j][0]);
  for(it=1;it<=nit;it++){
    disc=num/numer[it][alpha+1];
    ans=cf[it][alpha]*disc;
    avg+=ans/nit;
  }
  free_dmatrix(s,1,nit,alpha,N);
  free_dmatrix(e,1,nit,alpha,N);
  free_dmatrix(numer,1,nit,alpha,N);
  free_dmatrix(cf,1,nit,alpha,N);
  free_imatrix(itmflag,1,nit,alpha,N);
  free_imatrix(eflag,1,nit,alpha,N);
```

```
  return avg;
}
void regress(double* x,double* y,int n,int order,double* coef){
  int i;
  switch(order){
  case 1:{
    double sxy=0.0,sx=0.0,sy=0.0,sx2=0.0;
    for(i=1;i<=n;i++){
      sxy+=x[i]*y[i];
      sx+=x[i];
      sy+=y[i];
      sx2+=DSQR(x[i]);
    }
    coef[1]=(sxy-sx*sy/n)/(sx2-DSQR(sx)/n);
    coef[0]=(sy-coef[1]*sx)/n;
    break;
  }
  case 2:{
    double sx=0.0,sx2=0.0,sx3=0.0,sx4=0.0,sy=0.0,sxy=0.0,sx2y=0.0;
    for(i=1;i<=n;i++){
      sx+=x[i];
      sx2+=DSQR(x[i]);
      sx3+=pow(x[i],3);
      sx4+=pow(x[i],4);
      sy+=y[i];
      sxy+=x[i]*y[i];
      sx2y+=DSQR(x[i])*y[i];
    }
    double **a=dmatrix(1,3,1,3),*b=dvector(1,3),*p=dvector(1,3),*u=dvector(1,3);
    a[1][1]=n; a[1][2]=sx; a[1][3]=sx2;
    a[2][1]=sx; a[2][2]=sx2; a[2][3]=sx3;
    a[3][1]=sx2; a[3][2]=sx3; a[3][3]=sx4;
    b[1]=sy; b[2]=sxy; b[3]=sx2y;
    choldc(a,3,p); cholsl(a,3,p,b,u);
    coef[0]=u[1]; coef[1]=u[2]; coef[2]=u[3];
    free_dvector(u,1,3); free_dvector(b,1,3); free_dvector(p,1,3);
    free_dmatrix(a,1,3,1,3);
    break;
  }
  default:
    nrerror("order not 1");
  }
}
```

References

[1] F. Black *The pricing of commodity contracts*, Journal of Financial Economics, 3:167-179, 1976

[2] Damiano Brigo and Fabio Mercurio *Interest Rate Models: Theory and Practice*, Springer, 2001

[3] Brace, Gatarek, and Musiela *The market model of interest rate dynamics*, Mathematical Finance 7:127-155

[4] R Pietersz, A Pelsser, and M van Regenmortel *Fast drift-approximated prcing in the BGM model*, Journal of Computational Finance, 8(1):93-124

[5] P. Glasserman and X. Zhao *Fast Greeks by Simulation in Forward Libor Models*, Journal of Compuational Finance, 3(1):5-39, 1999

[6] V. V. Piterbarg *A practitioner's guide to pricing and hedging callable libor exotics in forward libor models*, SSRN, 2003

[7] Longstaff and Schwartz *Valuing American Options by Simulation: a simple least-squares approach*, The Review of Financial Studies, 14(1):113-47

[8] R Pietersz and A Pelsser *A comparison of single-factor Markov-functional and multi-factor market models*, Working Paper, 2005