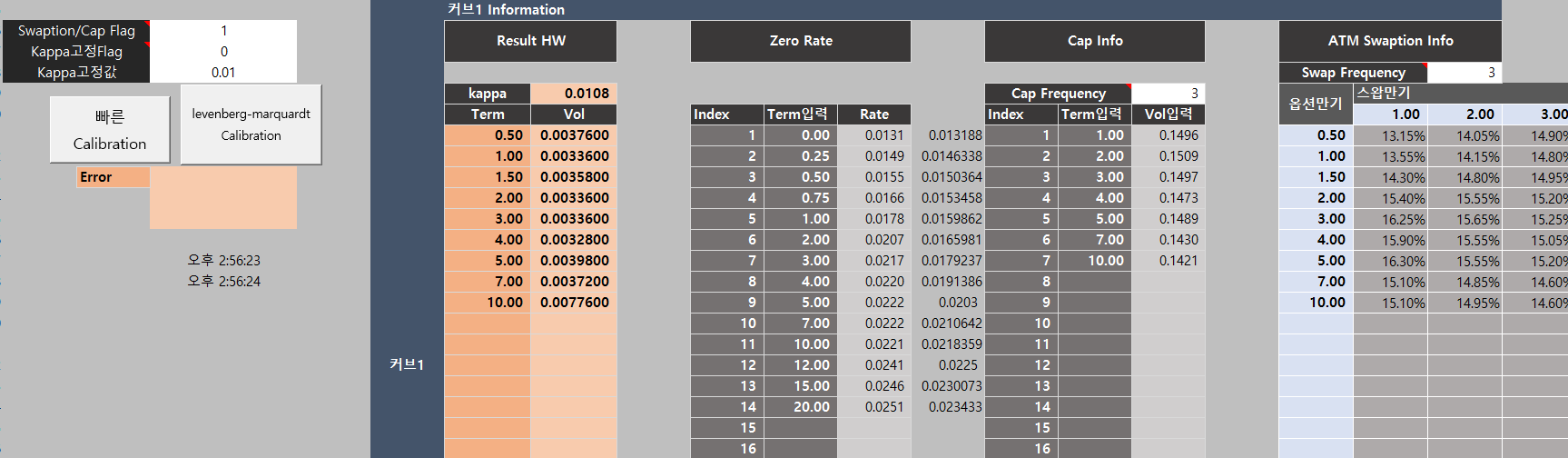
**HW\_Calibration 로직 설명서**

빠른 Calibration, Levenberg-marquardt 알고리즘 두 가지 사용 가능



1. **Hull White 1Factor Swaption – Code 1, 2, 3 참고**
2. **Hull White 2Factor Swaption – Code 4~10 참고**

***이고, 는 주어진 x에 대하여***

1. **Levenberg-marquardt**

(간혹 )

여기서 ,

1. **빠른 Calibration 1F**
   1. 빠른 Calibration 방법론

찾아낸 근방에서 위의 로직을 한 번 더 실행함

* 1. Calibration 예시

Example) 다음과 같이 Swaption Vol이 주어진다고 가정하자.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Swapmat= 1 | Swapmat= 2 | Swapmat= 3 |
| Optmat= 0.5 | 10% | 12% | 14% |
| Optmat= 1 | 11% | 13% | 15% |
| Optmat= 1.5 | 12% | 14% | 16% |

**Calibration은 다음과 같이 실행된다.**

찾아낸 근방에서 위의 로직을 한 번 더 실행함

double B(double s, double t, double kappa)

{

return (1.0 - exp(-kappa \* (t - s))) / kappa;

}

// 적분 계산 공통함수

// I(t) = Int\_0^t sigma(s)^2 A exp(ks) ds

double Integ(

double t,

double A,

double kappa,

double\* tVol,

double\* Vol,

long nVol

)

{

long i;

long NodeNum = 10;

long Point = 0;

double ds = t / (double)NodeNum;

double s;

double value = 0.0;

double sigma;

if (nVol == 1) return A \* Vol[0] \* Vol[0] / (kappa) \* (exp(kappa \* t) - 1.0);

for (i = 0; i < NodeNum; i++)

{

s = (double)(i + 1) \* ds;

sigma = Interpolate\_Linear\_Point(tVol, Vol, nVol, s, Point);

value += sigma \* sigma \* A \* exp(kappa \* s) \* ds;

}

return value;

}

1. **Swaption Pricing**

// 1-factor 모형의 Fixed Payer Swaption 가격 빠른 계산

double HW\_Swaption(

double NA, // 액면금액

double kappa, // 회귀속도

double\* tVol, // 변동성 구간 종점

double\* Vol, // 구간 변동성

long nVol, // 변동성 구간 개수

double\* t, // 할인채 만기

double\* r, // 할인채 가격

long nr, // 할인채 개수

double StrikeRate, // 고정금리(지급부분)

long MaturityDate, // 옵션 만기일까지 일수

long\* Dates, // 지급일: 계산일로부터 각 쿠폰지급일까지의 일수

long nDates, // 지급 회수(계산일 이후 남은 회수

double\* PT // P[0] = P(0, TOpt), P[N] = P(0, TSwp), Shape=nDates + 1

)

{

long i;

double T0, PrevT, T, deltaT;

double VT0, G, H;

double d1, d2;

double value;

if (kappa < -0.002) kappa = -0.002;

for (i = 0; i < nVol; i++)

{

if (Vol[i] < 0.0) Vol[i] = -Vol[i];

if (Vol[i] < Tiny\_Value) Vol[i] = Tiny\_Value;

}

T0 = (double)MaturityDate / 365.0;

VT0 = exp(-2.0 \* kappa \* T0) \* Integ(T0, 1.0, 2.0 \* kappa, tVol, Vol, nVol);

G = PT[nDates];

PrevT = T0;

for (i = 0; i < nDates; i++)

{

T = (double)Dates[i] / 365.0;

deltaT = T - PrevT;

G += StrikeRate \* deltaT \* PT[i + 1];

PrevT = T;

}

G /= PT[0];

H = PT[nDates] \* B(T0, (double)Dates[nDates - 1] / 365.0, kappa);

PrevT = T0;

for (i = 0; i < nDates; i++) {

T = (double)Dates[i] / 365.0;

deltaT = T - PrevT;

H += StrikeRate \* deltaT \* PT[i + 1] \* B(T0, T, kappa);

PrevT = T;

}

H /= G \* PT[0];

d1 = -log(G) / (H \* sqrt(VT0)) + 0.5 \* H \* sqrt(VT0);

d2 = -log(G) / (H \* sqrt(VT0)) - 0.5 \* H \* sqrt(VT0);

value = PT[0] \* (CDF\_N(d1) - G \* CDF\_N(d2));

return NA \* value;

}

double mu\_x(

double kappa1, double kappa2, long nHW1,

double \*HWTerm1, double\* HWVol1, long nHW2,

double\* HWTerm2, double\* HWVol2, double rho,

double T, double t1, double t2

)

{

long i;

long NodeNum = 10;

long Point = 0;

long Point2 = 0;

double du = (t2-t1) / (double)NodeNum;

double u = t1;

double value = 0.0;

double sigma;

double v1, v2;

for (i = 0; i < NodeNum; i++)

{

v1 = Interpolate\_Linear\_Point(HWTerm1, HWVol1, nHW1, u, Point);

v2 = Interpolate\_Linear\_Point(HWTerm2, HWVol2, nHW2, u, Point2);

value += exp(-kappa1 \* (t2 - u)) \* (v1 \* v1 \* B(u, T, kappa1) - rho \* v1 \* v2 \* B(u, T, kappa2)) \* du;

u += du;

}

return value;

}

double V(

double kappa, double kappa2, double t,

double T, double vol, double vol2

)

{

return vol \* vol2 / (kappa \* kappa2) \* (T - t + (exp(-kappa \* (T - t)) - 1.0) / kappa + (exp(-kappa2 \* (T - t)) - 1.0) / kappa2 - (exp(-(kappa + kappa2) \* (T - t)) - 1.0) / (kappa + kappa2));

}

double QV(

double t, double T, double kappa,

long NHWVol, double\* HWVolTerm, double\* HWVol

)

{

long i;

double vol;

double RHS = 0.0;

if (NHWVol == 1 || kappa > 0.1)

{

vol = Interpolate\_Linear(HWVolTerm, HWVol, NHWVol, t);

RHS = 0.5 \*(V(kappa,kappa,t,T,vol,vol)-V(kappa,kappa,0,T,vol,vol)+V(kappa,kappa,0, t,vol,vol));

}

else

{

RHS = 0.0;

long NInteg = 10;

double u = t;

double du = (T - t) / ((double)NInteg);

double Bst, BsT;

for (i = 0; i < NInteg; i++)

{

vol = Interpolate\_Linear(HWVolTerm, HWVol, NHWVol, u);

Bst = B(u, t, kappa);

BsT = B(u, T, kappa);

RHS += 0.5 \* vol \* vol \* (Bst \* Bst - BsT \* BsT) \* du;

u = u + du;

}

}

return RHS;

}

double CQV(

double t, double T, double kappa,

long NHWVol, double\* HWVolTerm, double\* HWVol,

double kappa2, double\* HWVolTerm2, double\* HWVol2,

double rho

)

{

long i;

double Bst, BsT, vol, vol2;

double RHS = 0.0;

double s, ds;

long NInteg = 10.0;

double u = t;

double du = (T - t) / ((double)NInteg);

RHS = 0.0;

if (NHWVol > 1)

{

vol = 0.5\*Interpolate\_Linear(HWVolTerm,HWVol,NHWVol,t)+0.5\* Interpolate\_Linear(HWVolTerm, HWVol, NHWVol, T);

vol2 = 0.5\*Interpolate\_Linear(HWVolTerm,HWVol2,NHWVol,t)+0.5\* Interpolate\_Linear(HWVolTerm, HWVol2, NHWVol, T);

}

else

{

vol = Interpolate\_Linear(HWVolTerm, HWVol, NHWVol, t);

vol2 = Interpolate\_Linear(HWVolTerm, HWVol2, NHWVol, t);

}

RHS = 2.0 \* rho \* 0.5 \* (V(kappa, kappa2, t, T, vol, vol2) - V(kappa, kappa2, 0, T, vol, vol2) + V(kappa, kappa2, 0, t, vol, vol2));

return RHS;

}

// Swaption2F 함수에서 사용

double A(

double t, double T, double kappa1,

double kappa2, double\* tVol,double\* Vol1,

double\* Vol2, double\* Vol12,long nVol,

double rho, double DF\_t, double DF\_T

)

{

return exp(QV(t,T,kappa1,nVol,tVol,Vol1)+QV(t,T,kappa2,nVol,tVol,Vol2)+CQV(t,T,kappa1,nVol, tVol,Vol1,kappa2,tVol,Vol2,rho));

}

// Gauss Hermite Normal 적분 계산에 사용될 x지점과 weight를 계산

// integral\_-inf to inf 1.0/(sigma\*sqrt(2.0PI)) \* exp(-((x-mu)/sigma)^2) f(x) dx = sum(w \* f(x))

void gauss\_hermite\_normal(double\* x, double\* w, double mu, double sigma, long n)

{

long i;

double sqrt2 = 1.4142135623730951;//sqrt(2.0);

double sqrtPI = 1.772453809055159;//sqrt(PI);

gauss\_hermite(x, w, n);

for (i = 0; i < n; i++)

{

x[i] = (x[i] \* sqrt2) \* sigma + mu;

w[i] = (w[i] / sqrtPI);

}

}

// 2-factor 모형의 Fixed Payer Swaption 가격 계산

double \_stdcall Swaption2F(

double NA, double kappa1, double kappa2,

double\* tVol, double\* Vol1, double\* Vol2,

double\* Vol12, long nVol, double rho,

double StrikeRate, long MaturityDate, long\* Dates,

double\* termdates, double\* termC, long nDates,

double\* PT, double P0\_at\_OptMaturity,

long nQuad, //Gauss Normal Quadrature 개수

double\* x, //Gauss Normal Quadrature의 x값

double\* w //Gauss Normal Quadrature의 y값 비율

)

{

long i, j;

double sum, value = 0.0;

double T;

double h1, h2, kappa\_i, y;

double m\_x, sigma\_x, m\_y, sigma\_y, rho\_xy;

if (kappa1 < Tiny\_Value) kappa1 = Tiny\_Value;

if (kappa2 < Tiny\_Value) kappa2 = Tiny\_Value;

T = (double)MaturityDate / 365.0;

double exp\_minus\_kappa1\_T = exp(-kappa1 \* T);

double exp\_minus2\_kappa1\_T = exp(-2.0 \* kappa1 \* T);

double exp\_minus\_kappa2\_T = exp(-kappa2 \* T);

double exp\_minus2\_kappa2\_T = exp(-2.0 \* kappa2 \* T);

m\_x = -mu\_x(kappa1, kappa2, nVol, tVol, Vol1, nVol, tVol, Vol2, rho, T, 0, T);

m\_y = -mu\_x(kappa2, kappa1, nVol, tVol, Vol2, nVol, tVol, Vol1, rho, T, 0, T);

double v1, v2;

v1 = Interpolate\_Linear(tVol, Vol1, nVol, T);

v2 = Interpolate\_Linear(tVol, Vol2, nVol, T);

sigma\_x = sqrt(v1 \* v1 \* (1.0 - exp(-2.0 \* kappa1 \* T)) / (2.0 \* kappa1));

sigma\_y = sqrt(v2 \* v2 \* (1.0 - exp(-2.0 \* kappa2 \* T)) / (2.0 \* kappa2));

rho\_xy = rho \* exp(-(kappa1 + kappa2) \* T) \* Integ(T, 1.0, kappa1 + kappa2, tVol, Vol12, nVol) / (sigma\_x \* sigma\_y);

if (sigma\_x < 0.0) sigma\_x = -sigma\_x;

if (sigma\_x < Tiny\_Value) sigma\_x = Tiny\_Value;

if (sigma\_y < 0.0) sigma\_y = -sigma\_y;

if (sigma\_y < Tiny\_Value) sigma\_y = Tiny\_Value;

rho\_xy = max(-0.9999,min(0.9999,rho\_xy));

gauss\_hermite\_normal(x, w, m\_x, sigma\_x, nQuad);

for (i = 0; i < nQuad; i++) {

for (j = 0; j < nDates; j++)

{

if (j == 0)

{

termC[j] = StrikeRate \* (termdates[j] - T) \* A(T, termdates[j], kappa1, kappa2, tVol, Vol1, Vol2, Vol12, nVol, rho, P0\_at\_OptMaturity, PT[j]) \* exp(-B(T, termdates[j], kappa1) \* x[i]);

}

else if (j == nDates - 1)

{

termC[j] = (1.0 + StrikeRate \* (termdates[j] - termdates[j - 1])) \* A(T, termdates[j], kappa1, kappa2, tVol, Vol1, Vol2, Vol12, nVol, rho, P0\_at\_OptMaturity, PT[j]) \* exp(-B(T, termdates[j], kappa1) \* x[i]);

}

else

{

termC[j] = StrikeRate \* (termdates[j] - termdates[j - 1]) \* A(T, termdates[j], kappa1, kappa2, tVol, Vol1, Vol2, Vol12, nVol, rho, P0\_at\_OptMaturity, PT[j]) \* exp(-B(T, termdates[j], kappa1) \* x[i]);

}

}

y = Find\_Sol2(kappa2, termC, T, termdates, nDates);

h1 = ((y - m\_y) / sigma\_y - rho\_xy \* (x[i] - m\_x) / sigma\_x) / sqrt(1.0 - rho\_xy \* rho\_xy);

sum = CDF\_N(-h1);

for (j = 0; j < nDates; j++)

{

h2 = h1 + B(T, termdates[j], kappa2) \* sigma\_y \* sqrt(1.0 - rho\_xy \* rho\_xy);

kappa\_i = -B(T, termdates[j], kappa2) \* (m\_y - 0.5 \* (1.0 - rho\_xy \* rho\_xy) \* sigma\_y \* sigma\_y \* B(T, termdates[j], kappa2) + rho\_xy \* sigma\_y \* (x[i] - m\_x) / sigma\_x);

sum -= termC[j] \* exp(kappa\_i) \* CDF\_N(-h2);

}

value += w[i] \* sum;

}

return NA \* value \* P0\_at\_OptMaturity;

}