

[Help](#)

```
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2007+2) //The "#els
#else

#include"
href../../../../common/math/lmm/lmm_header_h_src.pdf lmm_header.h"

static double period;

static double f1_lmm(double t , double T, double sigma)
{
    if (t >= T)
        return (0.0);
    else
        return (sigma);
}

/* variable sigma unused but present for type coherence with
* f1_lmm */
static double f2_lmm(double t , double T, double sigma)
{
    double val;
    if (t >= T)
        return (0.0);
    else
        val = 1. / sqrt(0.04 + 0.00075 * t) * (0.01 - 0.05 * exp(-0.1 * (T - t)));
    return (val);
}

/* variable sigma unused but present for type coherence with
* fd1_lmm */
static double fd2_lmm(double t, double T, double sigma)
{
    int j;
    int k;
    double val;

    if (t >= T)
```

```

        return (0.0);
    else
    {

        j = (int)(t / period);
        k = (int)(T / period);
        val = 0.01 - 0.05 * exp(-0.1 * (j - k)) / sqrt(0.04 + 0.00075 * j);
        return (val);
    }
}

```

```

static double fd1_lmm(double t , double T, double sigma)
{
    if (t >= T)
        return (0.0);
    else
        return (sigma);
}

```

```

int mallocVolatility(Volatility **ptVol , int numOfFac, double sigma)
{

    Volatility *pt;
    pt = (Volatility *)malloc(sizeof(Volatility));

    pt->flat_sigma = sigma;

    if (numOfFac > 2)
    {
        printf("only two factors allowed !!!");
        numOfFac = 2;
    }
    pt->numberOfFactors = numOfFac;

    pt->vol = (funcVol *)malloc(sizeof(funcVol) * pt->numberOfFactors);
    if (numOfFac == 1)
    {
        pt->vol[0] = f1_lmm;
    }
}

```

```

    }
else
    {
        pt->vol[0] = f1_lmm;
        pt->vol[1] = f2_lmm;
    }

    *ptVol = pt;
    return (1);
}

```

```

int mallocVolatilityInteger(Volatility **ptVol , int numOfFac, float tenor, double sigma)
{
    Volatility *pt;
    pt = (Volatility *)malloc(sizeof(Volatility));

    pt->flat_sigma = sigma;
    period = tenor;

    if (numOfFac > 2)
    {
        printf("only two factors allowed !!!");
        numOfFac = 2;
    }
    pt->numberOfFactors = numOfFac;

    pt->vol = (funcVol *)malloc(sizeof(funcVol) * pt->numberOfFactors);
    if (numOfFac == 1)
    {
        pt->vol[0] = fd1_lmm;
    }
    else
    {
        pt->vol[0] = fd1_lmm;
        pt->vol[1] = fd2_lmm;
    }
    *ptVol = pt;
    return (1);
}

```

```

double evalVolatility(Volatility *ptVol , int factorNumber, double t, double T)
// returns the value at time t of the volatility factor number factorNumber with
{
    double sigma = ptVol->flat_sigma;

    if (factorNumber > ptVol->numberOfFactors)
    {
        printf("not enough factors!\n n");
        exit(1);
    }

    else
    {
        return (ptVol->vol[factorNumber](t, T, sigma));
    }
}

void freeVolatility(Volatility **ptVol)
{
    free((*ptVol)->vol);
    free(*ptVol);
    *ptVol = NULL;
}

int copyVolatility(Volatility *ptSrc, Volatility *ptDest)
{
    int j;

    ptDest->flat_sigma = ptSrc->flat_sigma;

    ptDest->numberOfFactors = ptSrc->numberOfFactors; /*numOfFac;*/

    for (j = 0; j < ptSrc->numberOfFactors; j++)
    {
        ptDest->vol[j] = ptSrc->vol[j];
    }

    return (1);
}

```

```
};
```

```
#endif //PremiaCurrentVersion
```