```
   Help
#include  "garch1d_std.h"
#include "enums.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2011+2) //The "#else" part of the code will be freely av
    ailable after the (year of creation of this file + 2)
static int CHK_OPT(MC_Duan)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
static int MET(Init)(PricingMethod *Met,Option *Opt)
{
  return OK;
}

int CALC(MC_Duan)(void*Opt,void *Mod,PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else


/**
 * compute the sum of all element in the row of matrix
 *
 * @param[in] mat matrix
 * @param[in] nb_row the number of row
 * @return sum of all element in the row of matrix
 *
 */
static double sum_row_matrix(const PnlMat* mat,int nb_row)
{
  double result;

  PnlVect* V=pnl_vect_create(mat->m);
  pnl_mat_get_row(V,mat,nb_row);
  result= pnl_vect_sum(V);
  pnl_vect_free(&V);

  return result;
```

```
}

/**
 * ems carries out the EMS correction.
 *
 * @param[in] mat matrix which contains the paths that need
     to be corrected
 * @param[in] interest the annulized interest
 * @param[in] frequency the rate at which the quotes
    contained in matrix are given
 for example :1 for daily 5 for weekly
 * @param[out] ems_path matrix after EMS correction
 * @return OK if ok otherwise return FAIL
 *
 */

static int ems(const PnlMat* path,double interest,int frequ
    ency,PnlMat* ems_path)
{


  int row,colum;
  int i=0,j;
  double sum_row,s1=0;
  double r=interest*frequency/252;

  row=path->m;
  colum=path->n;
  ems_path->m=row;
  ems_path->n=colum;
  memcpy(ems_path->array,path->array,row*colum*sizeof(
    double));

  if(row>=1)
    {
      s1=pnl_mat_get(ems_path,0,0);
    }

  while(i<row)
    {
      if(i==0)
```

```
        {
          for(j=0;j<colum;j++)
            {
              pnl_mat_set(ems_path,i,j,s1);
            }
        }
      else
        {
          sum_row=sum_row_matrix(path,i)/(colum*pow(M_E,i*
    r));
          for(j=0;j<colum;j++)
            {
              pnl_mat_set(ems_path,i,j,s1*pnl_mat_get(ems_
    path,i,j)/sum_row);
            }

        }
      i++;
    }

  return OK;
}



/**
 * compute the paths from vector which contains historical
    price and stock the result in a matrix with dimensions (T+1)*N
    for GARCH model
 * @param[in] h vector which contain all values history of price
 * @param[out] path matrix stock all path after compute
 * @param[in] T Exercise time
 * @param[in] alpha_zero  garch parameter
 * @param[in] alpha_one  garch parameter
 * @param[in] interest interest rate
 * @param[in] lambda the constant unit risk premium
 * @param[in] beta_one parameter garch
 * @param[in] type_generator the type of generator for rand
    om number
 * @return OK if ok otherwise return FAIL
 */
```

```
static int calculate_path(const PnlVect* h,PnlMat* path,
    double interest,int frequency,int N,int T,double alpha_zero,
    double alpha_one,double lambda,double beta_one,int type_generator)
{

  double sigma,s,sigma_t;
  int size;//size of vector
  double s_t;//price at time t
  double s_0;//price init
  double presigma=alpha_zero/(1-beta_one-alpha_one*(1+pow(
    lambda,2)));
  double preepsilon=0;
  PnlMat* eps;//matrix contains all variables epsilon with
    value random
  PnlMat* array_sigma;
  int dimension=N;//colum of matrix
  int samples;//row of matrix
  int i,j;


  interest=(frequency*interest)/252.;
  size=h->size;
  s_0=pnl_vect_get(h,0);
  eps=pnl_mat_create(T-size+1,N);//matrix contains all
    variables epsilon with value random
  array_sigma=pnl_mat_create(T-size+1,N);
  samples=T-size+1;
  pnl_rand_init(type_generator,dimension,samples);
  pnl_mat_rand_normal(eps,samples,dimension,type_    generator);


  if(size==1)
    {
      s_t=s_0;
    }
  else
    {
      s_t=pnl_vect_get(h,size-1);
    }
  if(size>1)
    {
```

```
    for(i=0;i<size-1;i++)
      {
        presigma=sqrt(alpha_zero+alpha_one*pow((presigma*
  preepsilon-lambda*presigma),2)+beta_one*pow(presigma,2));
        preepsilon=(1/presigma)*(log(pnl_vect_get(h,i+1)/
  pnl_vect_get(h,i))-interest+0.5*pow(presigma,2));
      }
  }

for(i=0;i<dimension;i++)
  {
    for(j=0;j<samples;j++)
      {
        if(j==0)
          {
            sigma=presigma;
            pnl_mat_set(array_sigma,0,i,sigma);
            if(size>1)
              {
                s=s_t*pow(M_E,(interest-0.5*pow(pnl_mat_
  get(array_sigma,0,i),2)+pnl_mat_get(array_sigma,0,i)*preeps
  ilon));
              }
            else
              {
                s=s_0*pow(M_E,(interest-0.5*pow(pnl_mat_
  get(array_sigma,0,i),2)+pnl_mat_get(array_sigma,0,i)*pnl_
  mat_get(eps,0,i)));
              }
            pnl_mat_set(path,0,i,s);
          }
        else
          {
            sigma_t=pnl_mat_get(array_sigma,j-1,i);

            sigma=sqrt(alpha_zero+alpha_one*pow((sigma_t*
  pnl_mat_get(eps,j-1,i)-lambda*sigma_t),2)+beta_one*pow(si
  gma_t,2));

            pnl_mat_set(array_sigma,j,i,sigma);
            s=pnl_mat_get(path,j-1,i)*pow(M_E,(interest-0
```

```
    .5*pow(sigma,2)+sigma*pnl_mat_get(eps,j,i)));
              pnl_mat_set(path,j,i,s);
            }
        }
    }

  pnl_mat_free(&eps);
  pnl_mat_free(&array_sigma);
  return OK;
}

/**
 * Computes garch price for GARCH model
 * @param[in] today_price taday price
 * @param[in] alpha_zero garch parameter
 * @param[in] alpha_one garch parameter
 * @param[in] lambda the constant unit risk premium
 * @param[in] beta_one garch parameter
 * @param[in] interest the annulized interest
 * @param[in] K exercise price
 * @param[in] frequency frequency
 * @param[in] T time to mutrity
 * @param[in] choice emscorrection(ems_on or ems_off)
 * @param[in] type_generator the type of generator for rand
    om number
 * @param[out] garch option price
 *             garch->call obtains call option price
 *             garch->put  obtains put option price
 * @return OK if ok otherwise return FAIL
 */
static int garch_price(NumFunc_1  *p,double today_price,
    double alpha_zero,double alpha_one,double beta_one,double lambd
    a,double interest,int frequency,double K,int T,int N,int
    choice,int type_generator,double *price,double *delta)
{
  double   sum_callorput;
  double   sum_delta;
  double   s_T;
  double   garch_delta;
  int      i;
  PnlMat  *path_ems, *path, *path1D;
```

```
PnlVect *h;
path=pnl_mat_create(T,N);
h=pnl_vect_create (1);

sum_callorput=0;
sum_delta=0;

pnl_vect_set(h,0,today_price);

if (calculate_path(h,path,interest,frequency,N,T,alpha_ze
  ro,alpha_one,lambda,beta_one,type_generator)==FAIL)
  {
    pnl_vect_free(&h);
    pnl_mat_free(&path);
    return FAIL;
  }
//if we choose ems option
switch (choice)
  {
  case 1:
    pnl_vect_free(&h);
    pnl_rand_init(type_generator,N,T);
    path_ems=pnl_mat_create(T,N);
    if(ems(path,interest,frequency,path_ems)==FAIL)
      {
        pnl_mat_free(&path);
        pnl_mat_free(&path_ems);
        return FAIL;
      }
    pnl_mat_clone(path, path_ems);
    for(i=0;i<N;i++)
      {
        s_T=pnl_mat_get(path,T-1,i);
        sum_callorput=sum_callorput+(p->Compute)(p->Par,
  pnl_mat_get(path,T-1,i));
        if(s_T>K) garch_delta=1.;

        sum_delta=sum_delta+(s_T/today_price)*garch_delt
  a;
      }
    pnl_mat_free(&path_ems);
```

```
      break;
    case 2:
      path1D=pnl_mat_create(T,1);
      pnl_rand_init(type_generator,1,T);
      for(i=0;i<N;i++)
        {
          calculate_path(h,path1D,interest,frequency,1,T,
    alpha_zero,alpha_one,lambda,beta_one,type_generator);
          s_T=pnl_mat_get(path1D,T-1,0);
          sum_callorput=sum_callorput+(p->Compute)(p->Par,
    pnl_mat_get(path,T-1,i));
          if(s_T>K) garch_delta=1.;
          sum_delta=sum_delta+(s_T/today_price)*garch_delt
    a;
        }
      pnl_vect_free(&h);
      pnl_mat_free(&path1D);
      break;
    default:
      printf ("Wrong value for parameter EMS{n");
      return FAIL;
    }

  interest=(interest*frequency)/252.;

  //Price
  *price=sum_callorput/(N*pow(M_E,(interest*T)));
  *delta=sum_delta/(N*pow(M_E,(T*interest)));
  if ((p->Compute)==&Put) *delta=*delta-1;

  pnl_mat_free(&path);
  return OK ;
}

static int MCDuan(double s, NumFunc_1  *p, double T,
    double r,double alpha0 ,double alpha1, double beta1,double lambd
    a,int FREQUENCY,long N, int ems,int type_generator,double *
    ptprice, double *ptdelta)
{
  int dummy,nb_days;
  double K,price=0,delta=0;
```

```
  nb_days=(int)(T*252);
  K= p->Par[0].Val.V_PDOUBLE;
  dummy=garch_price(p,s,alpha0,alpha1,beta1,lambda,r,FREQU
    ENCY,K,nb_days,N,ems,type_generator,&price,&delta);

  //Price
  *ptprice=price;

  //Delta
  *ptdelta=delta;

  return dummy;
}

int CALC(MC_Duan)(void *Opt, void *Mod, PricingMethod *Met)
{
  TYPEOPT* ptOpt=(TYPEOPT*)Opt;
  TYPEMOD* ptMod=(TYPEMOD*)Mod;
  double r;

  r=log(1.+ptMod->R.Val.V_DOUBLE/100.);

  return MCDuan(ptMod->S0.Val.V_PDOUBLE,
                ptOpt->PayOff.Val.V_NUMFUNC_1,
                ptOpt->Maturity.Val.V_DATE-ptMod->T.Val.V_
    DATE,
                r,
                ptMod->alpha0.Val.V_PDOUBLE,
                ptMod->alpha1.Val.V_PDOUBLE,
                ptMod->beta1.Val.V_PDOUBLE,
                ptMod->lambda.Val.V_PDOUBLE,
                Met->Par[0].Val.V_PINT,
                Met->Par[1].Val.V_LONG,
                Met->Par[2].Val.V_ENUM.value,
                Met->Par[3].Val.V_ENUM.value,
                &(Met->Res[0].Val.V_DOUBLE),
                &(Met->Res[1].Val.V_DOUBLE));
}
```

```
static PremiaEnumMember EMSMethodMembers[] =
{
  { "WITH EMS CORRECTION", 1 },
  { "WITHOUT EMS CORRECTION", 2 },
  { NULL, NULLINT }
};

static DEFINE_ENUM(EMSMethod, EMSMethodMembers);

static int CHK_OPT(MC_Duan)(void *Opt, void *Mod)
{
  if ((strcmp( ((Option*)Opt)->Name,"CallEuro")==0)
      ||(strcmp( ((Option*)Opt)->Name,"PutEuro")==0))

    return OK;
  return WRONG;
}

static int MET(Init)(PricingMethod *Met,Option *Opt)
{
  int type_generator;
  if ( Met->init == 0)
    {
      Met->init=1;

      Met->Par[0].Val.V_PINT=1;
      Met->Par[1].Val.V_LONG=1000;
      Met->Par[2].Val.V_ENUM.value=1;
      Met->Par[2].Val.V_ENUM.members=&EMSMethod;
      Met->Par[3].Val.V_ENUM.value=0;
      Met->Par[3].Val.V_ENUM.members=&PremiaEnumRNGs;


    }

  type_generator= Met->Par[2].Val.V_ENUM.value;


  if(pnl_rand_or_quasi(type_generator)==PNL_QMC)
    {
      Met->Res[2].Viter=IRRELEVANT;
```

```
      Met->Res[3].Viter=IRRELEVANT;
      Met->Res[4].Viter=IRRELEVANT;
      Met->Res[5].Viter=IRRELEVANT;
      Met->Res[6].Viter=IRRELEVANT;
      Met->Res[7].Viter=IRRELEVANT;

    }
  else
    {
      Met->Res[2].Viter=ALLOW;
      Met->Res[3].Viter=ALLOW;
      Met->Res[4].Viter=ALLOW;
      Met->Res[5].Viter=ALLOW;
      Met->Res[6].Viter=ALLOW;
      Met->Res[7].Viter=ALLOW;
    }
  return OK;
}
#endif //PremiaCurrentVersion


PricingMethod MET(MC_Duan)=
{
  "MC_Duan",
  { {"Frequency",PINT,{100},ALLOW},
    {"N iterations",LONG,{100},ALLOW},
    {"EMS",ENUM,{100},ALLOW},
    {"RandomGenerator",ENUM,{100},ALLOW},
    {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(MC_Duan),
  {{"Price",DOUBLE,{100},FORBID},
   {"Delta",DOUBLE,{100},FORBID} ,
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(MC_Duan),
  CHK_mc,
  MET(Init)
};
```

References