```
   Help
/* Control Variables Kemna & Vorst Monte Carlo si
    mulation for a Call or Put Fixed Asian option.
   In the case of Monte Carlo simulation, the pr
    ogram provides estimations for price and delta w
    ith a confidence interval.
   In the case of Quasi-Monte Carlo simulation,
    the program just provides estimations for price
    and delta. */
#include  "bs1d_pad.h"

#define JMAX     40
static double mu[50000];
static double t,sig, ri, dvd, S0, strike, step_nb
    ;
/*-----------------------------------------------
    ---------------------*/
  /* Methode de dichotomies permet de trouver u
    n zero d'une fonction*/
  /* sachant que ce zero se trouve entre x1 et x
    2. Precision = xacc*/
  /*-----------------------------------------------
    ------------------------*/
double rtbis(double (*func)(double),double x1,
    double x2,double xacc)
{
  int j;
  double dx,f,fmid,xmid,rtb;

  f=(*func)(x1);
  fmid=(*func)(x2);

  if(f*fmid>=0.0){
  printf("La racine ne se trouve pas dans [x1,x2
    ]");
  exit(-1);
  }

  rtb=f<0.0?(dx=x2-x1,x1):(dx=x1-x2,x2); /* orie
    nte la recherche*/
```

```
  for(j=1;j<=JMAX;j++){
  fmid=(*func)(xmid=rtb+(dx*=0.5));
  if(fmid<=0.0)rtb=xmid;
  if(fabs(dx)<xacc||fmid==0.0)return rtb;
  }

 return 0.0;
}

/*-----------------------------------------------
    --------------------------*/
/*Premiere partie : recherche du mu optimal*/
/*La fonction ci-dessous est celle qu'il faut app
    eller pour trouver le mu */
/*optimal. On cherche d'abord son unique racine q
    u'on reinjecte ensuite*/
/*dans les z[1..PAS] et s[1..PAS]; le dernier z[]
     est alors le mu optimal.*/
double ghs_func(double g){
  int i;
  double z=0.0;
  double s;
  double dt,ans,s_dt,trend;

  dt=t/step_nb;

  s=S0;s_dt=sig*sqrt(dt);
  trend=ri-dvd-0.5*SQR(sig);

  if(g>0.0){ /*"g>0" correspond au cas du Call */
  ans=0;   /*"alors que "g<0" definit le cas du
    Put.*/
  z=s_dt*(g+strike)/g;
  for(i=1;i<step_nb;i++){
    s=s*exp(trend*dt+s_dt*z);
    z=z-s_dt*s/(step_nb*g);
    ans+=s;
  }
  ans+=(s=s*exp(trend*dt+s_dt*z));
  ans/=step_nb;
  }
```

```
  return (ans=(ans-strike-g));
}
/* ------------------------------------------------
    -------------------------- */
/* Calculus of the average A'(T0,T) and C'(T0,T)
    of the asian option with one of the 3 different
    schemes
   One iteration of the Monte Carlo method called
     from the "FixedAsian_KemanVorst" function */
/* ------------------------------------------------
    -------------------------- */

static void Simul_StockAndAverage_Glassermann(
    int generator, int mc_or_qmc, int step_number,
    double T,  double x, double r, double divid, double si
    gma, NumFunc_2 *p, double K)
{
  double integral,  S_t, g1;
  double h = T / step_number;
  double sqrt_h = sqrt(h);
  double trend= (r -divid)- 0.5 * SQR(sigma);
  double ss_dt=sigma*sqrt(h);

  int i,ii;
  double dot1,payoff,payoffcarre,val_test,temp,
    expo,val;
  double dot2,g;
  double* NormalValue;
  double *m_Theta;

  t=T;ri=r;
  S0=x;strike=K;
  sig=sigma;
  dvd=divid;
  step_nb=step_number;

  for(i=0;i<step_number;i++)
    mu[i]=0.;

  g=rtbis(ghs_func,0.1,100.0,1e-8); /*calcul du
    mu optimal*/
```

```
  /*printf("g  = %f{n",g);*/

  if(g>0)
  mu[0]=ss_dt*(g+K)/g;

  S_t=x;
  for(i=1;i<step_number;i++){
  S_t=S_t*exp(trend*h+ss_dt*mu[i-1]);
  mu[i]=mu[i-1]-ss_dt*S_t/(step_number*g);
  }

  return;
}

/* ----------------------------------------------
    --------------------*/
/* Pricing of a asian option by the Monte Carlo K
    emna & Vorst method
   Estimator of the price and the delta.
   s et K are pseudo-spot and pseudo-strike. */
/* ----------------------------------------------
    ------------------- */
static int  FixedAsian_Glassermann(double s,
    double K, double time_spent, NumFunc_2 *p, double t,
    double r, double divid, double sigma, long nb, int M,
    int generator, double confidence, double *ptprice,
    double *ptdelta, double *pterror_price, double *pt
    error_delta, double *inf_price, double *sup_price,
    double *inf_delta, double *sup_delta)
{
  long i,ipath;

  double price_sample  , delta_sample, mean_pric
    e, mean_delta, var_price, var_delta;
  int init_mc, mc_or_qmc;
  int simulation_dim;
  double alpha, z_alpha,dot1,dot2,inc=0.001;
  double integral, S_t, g1;
  double h = t /(double)M;
  double sqrt_h = sqrt(h);
  double trend= (r -divid)- 0.5 * SQR(sigma);
```

```
int step_number=M;


/* Value to construct the confidence interval *
  /
alpha= (1.- confidence)/2.;
z_alpha= Inverse_erf(1.- alpha);

/*Initialisation*/
mean_price= 0.0;
mean_delta= 0.0;
var_price= 0.0;
var_delta= 0.0;

/* Size of the random vector we need in the si
  mulation */
simulation_dim= M;

/* MC sampling */
init_mc= InitGenerator(generator, simulation_
  dim,nb);
/* Test after initialization for the generator
  */
if(init_mc == OK)
  {
    mc_or_qmc= Rand_Or_Quasi(generator);

    /* Price  */
    (void)Simul_StockAndAverage_Glassermann(gen
  erator, mc_or_qmc, M, t, s,r, divid, sigma, p, K);

    dot2=0;
    for(i=0;i<step_number;i++)
dot2+=mu[i]*mu[i];

    for(ipath= 1;ipath<= nb;ipath++)
{
  /* Begin of the N iterations */

  g1= Gaussians[mc_or_qmc](step_number, CREA
  TE, 0, generator);
```

```
  integral=0.0;
  S_t=s;dot1=0.;
  for(i=0 ; i< step_number ; i++) {
  g1= Gaussians[mc_or_qmc](step_number, RETRI
  EVE, i, generator);
    S_t *=exp(trend *h +sigma*sqrt_h*(g1+mu[i]
  ));
    integral+=S_t;
  dot1+=mu[i]*g1;
  }

        price_sample=(p->Compute)(p->Par, s,
  integral/(double)step_number)*exp(-dot1-0.5*dot2);


  /* Delta */
  if(price_sample >0.0)
    delta_sample=(1-time_spent)*(integral/(s*(
  double)step_number))*exp(-dot1-0.5*dot2);
  else  delta_sample=0.;

  /* Sum */
  mean_price+= price_sample;
  mean_delta+= delta_sample;

  /* Sum of squares */
  var_price+= SQR(price_sample);
  var_delta+= SQR(delta_sample);
}
    /* End of the N iterations */

    /* Price estimator */
    *ptprice=(mean_price/(double)nb);
    *pterror_price= exp(-r*t)*sqrt(var_price/(
  double)nb-SQR(*ptprice))/sqrt((double)nb-1);
    *ptprice= exp(-r*t)*(*ptprice);

    /* Price Confidence Interval */
    *inf_price= *ptprice - z_alpha*(*pterror_p
  rice);
    *sup_price= *ptprice + z_alpha*(*pterror_p
```

```
    rice);


      /* Delta estimator */
      *ptdelta=exp(-r*t)*(mean_delta/(double)nb);
      if((p->Compute) == &Put_OverSpot2)
  *ptdelta *= (-1);
      *pterror_delta= sqrt(exp(-2.0*r*t)*(var_de
    lta/(double)nb-SQR(*ptdelta)))/sqrt((double)nb-1)
    ;

      /* Delta Confidence Interval */
      *inf_delta= *ptdelta - z_alpha*(*pterror_d
    elta);
      *sup_delta= *ptdelta + z_alpha*(*pterror_d
    elta);
    }
  return init_mc;
}



int CALC(MC_FixedAsian_Glassermann)(void *Opt,vo
    id *Mod,PricingMethod *Met)
{
  TYPEOPT* ptOpt=(TYPEOPT*)Opt;
  TYPEMOD* ptMod=(TYPEMOD*)Mod;

  double T, t_0, T_0;
  double r, divid, time_spent, pseudo_strike, tru
    e_strike, pseudo_spot;
  int return_value;

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

  T= ptOpt->Maturity.Val.V_DATE;
  T_0 = ptMod->T.Val.V_DATE;
  t_0= (ptOpt->PathDep.Val.V_NUMFUNC_2)->Par[0].
    Val.V_PDOUBLE;
  time_spent= (T_0-t_0)/(T-t_0);
```

```
if(T_0 < t_0)
  {
    Fprintf(TOSCREEN,"T_0 < t_0, untreated cas
  e{n{n{n");
    return_value = WRONG;
  }

/* Case t_0 <= T_0 */
else
  {
    pseudo_spot= (1.-time_spent)*ptMod->S0.Val.
  V_PDOUBLE;
    pseudo_strike= (ptOpt->PayOff.Val.V_
  NUMFUNC_2)->Par[0].Val.V_PDOUBLE-time_spent*(ptOpt->Pat
  hDep.Val.V_NUMFUNC_2)->Par[4].Val.V_PDOUBLE;


    true_strike= (ptOpt->PayOff.Val.V_NUMFUNC_2
  )->Par[0].Val.V_PDOUBLE;

    (ptOpt->PayOff.Val.V_NUMFUNC_2)->Par[0].Val
  .V_PDOUBLE= pseudo_strike;

    if (pseudo_strike<=0.)
{
  Fprintf(TOSCREEN,"FORMULE ANALYTIQUE{n{n{n")
  ;
  return_value= Analytic_KemnaVorst(pseudo_sp
  ot,
            pseudo_strike,
            time_spent,
            ptOpt->PayOff.Val.V_NUMFUNC_2,
            T-T_0,
            r,
            divid,
            &(Met->Res[0].Val.V_DOUBLE),
            &(Met->Res[1].Val.V_DOUBLE));

}
    else
```

```
  return_value= FixedAsian_Glassermann(pseudo_sp
    ot,
                pseudo_strike,
                time_spent,
                ptOpt->PayOff.Val.V_NUMFUNC_
    2,
                T-T_0,
                r,
                divid,
                ptMod->Sigma.Val.V_PDOUBLE,
                Met->Par[2].Val.V_LONG,
                Met->Par[0].Val.V_INT2,
                Met->Par[1].Val.V_INT,
                Met->Par[4].Val.V_DOUBLE,
                &(Met->Res[0].Val.V_DOUBLE),
                &(Met->Res[1].Val.V_DOUBLE),
                &(Met->Res[2].Val.V_DOUBLE),
                &(Met->Res[3].Val.V_DOUBLE),
                &(Met->Res[4].Val.V_DOUBLE),
                &(Met->Res[5].Val.V_DOUBLE),
                &(Met->Res[6].Val.V_DOUBLE),
                &(Met->Res[7].Val.V_DOUBLE))
    ;

      (ptOpt->PayOff.Val.V_NUMFUNC_2)->Par[0].Val
    .V_PDOUBLE=true_strike;
    }
  return return_value;
}



int CHK_OPT(MC_FixedAsian_Glassermann)(void *Opt,
     void *Mod)
{
  if ( (strcmp( ((Option*)Opt)->Name,"
    AsianCallFixedEuro")==0) || (strcmp( ((Option*)Opt)->Name,"
    AsianPutFixedEuro")==0) )
    return OK;

  return WRONG;
```

```
}



static int MET(Init)(PricingMethod *Met)
{
  static int first=1;
  int type_generator;

  type_generator= Met->Par[1].Val.V_INT;


  if (first)
    {
      Met->Par[0].Val.V_INT2= 50;
      Met->Par[1].Val.V_INT= 0;
      Met->Par[2].Val.V_LONG= 20000;
      Met->Par[4].Val.V_DOUBLE= 0.95;

      first=0;
    }
  if(Rand_Or_Quasi(type_generator)==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;
}



PricingMethod MET(MC_FixedAsian_Glassermann)=
{
  "MC_FixedAsian_Glassermann",
  {{"TimeStepNumber",INT2,100,ALLOW},
   {"RandomGenerator",GENER,100,ALLOW},
   {"N iterations",LONG,100,ALLOW},
   {"Confidence Value",DOUBLE,100,ALLOW},
   {" ",END,0,FORBID}},
  CALC(MC_FixedAsian_Glassermann),
  {{"Price",DOUBLE,100,FORBID},
   {"Delta",DOUBLE,100,FORBID} ,
   {"Error Price",DOUBLE,100,FORBID},
   {"Error Delta",DOUBLE,100,FORBID} ,
   {"Inf Price",DOUBLE,100,FORBID},
   {"Sup Price",DOUBLE,100,FORBID} ,
   {"Inf Delta",DOUBLE,100,FORBID},
   {"Sup Delta",DOUBLE,100,FORBID} ,
   {" ",END,0,FORBID}},
  CHK_OPT(MC_FixedAsian_Glassermann),
  CHK_ok,
  MET(Init)
};
```

References