```
   Help
#include  "bs1d_std.h"

static double *FP=NULL,*Traj=NULL,*Res=NULL;
static double *M=NULL,*AuxR=NULL, *VBasis=NULL;

static double *Pont=NULL;
static double (*basis)(double *stock,int l,
    NumFunc_1 *p);

static int LongRet_Allocation(long MC_Iterations,
     int DimApprox,int DimBS)
{
  int dummy;

  if (FP==NULL)
  FP=(double*)malloc(MC_Iterations*sizeof(
    double));

  if (FP==NULL) return MEMORY_ALLOCATION_FAILURE;

  if (Traj==NULL)
  Traj=(double*)malloc(MC_Iterations*DimBS*sizeo
    f(double));

  if (Traj==NULL) return MEMORY_ALLOCATION_FAILU
    RE;

  if (M==NULL)
  M=(double*)malloc(DimApprox*DimApprox*sizeof(
    double));

  if (M==NULL) return MEMORY_ALLOCATION_FAILURE;

  if (Res==NULL)
  Res=(double*)malloc(DimApprox*sizeof(double));

  if (Res==NULL) return MEMORY_ALLOCATION_FAILURE
    ;

  if (AuxR==NULL)
```

```
  AuxR=(double*)malloc(DimApprox*sizeof(double))
    ;

  if (AuxR==NULL) return MEMORY_ALLOCATION_FAILU
    RE;

  if (VBasis==NULL)
  VBasis=(double*)malloc(DimApprox*sizeof(
    double));

  if (VBasis==NULL) return MEMORY_ALLOCATION_FAIL
    URE;

  if (Pont==NULL)
  Pont=(double*)malloc(MC_Iterations*DimBS*sizeo
    f(double));
  if (Pont==NULL) return MEMORY_ALLOCATION_FAILU
    RE;

  dummy=Chol_Allocation(DimApprox);


  return OK;
}

static void LongRet_Liberation()
{
  if (FP!=NULL){
  free(FP);
  FP=NULL;
  }
  if (Traj!=NULL) {
  free(Traj);
  Traj=NULL;
  }
  if (M!=NULL) {
  free(M);
  M=NULL;
  }
  if (Res!=NULL) {
  free(Res);
```

```
  Res=NULL;
  }
  if (AuxR!=NULL) {
  free(AuxR);
  AuxR=NULL;
  }
  if (VBasis!=NULL) {
  free(VBasis);
  VBasis=NULL;
  }
  if (Pont!=NULL) {
  free(Pont);
  Pont=NULL;
  }

  Chol_Liberation();

  return;
}

/*Canonical Basis for Regression*/
static double CanonicalD1(double *x, int ind,
    NumFunc_1 *p)
{
  double aux;
  int i;

  aux=1.;
  for (i=0;i<ind;i++)
  aux*=(*x);
  return aux;
}

/*Basis Regression=Payoff + Canonnical*/
static double CanonicalOpD1(double *x, int ind,
    NumFunc_1 *p)
{
  if (ind==0) return (p->Compute)(p->Par,*x);
  else return CanonicalD1(x,ind-1,p);
}
```

```
/*Normalized Laguerre Basis*/
static double LaguerreD1(double *x, int ind,
    NumFunc_1 *p)
{
  switch (ind){
  case 0 : return 1;
  case 1 : return exp(-(*x)*0.5);
  case 2 : return exp(-(*x)*0.5)*(1-(*x));
  case 3 : return exp(-(*x)*0.5)*(1-2*(*x)+0.5*(*
    x)*(*x));
  case 4 : return exp(-(*x)*0.5)*(-0.5*(*x)*(*x)*
    (*x)+4.5*(*x)*(*x)
                -9*(*x)+3);
  case 5 : return exp(-(*x)*0.5)*(0.5*(*x)*(*x)*(
    *x)*(*x)-8*(*x)*(*x)*(*x)
                +36*(*x)*(*x)-48*(*x)+1
    2);
  case 6 : return exp(-(*x)*0.5)*(-0.5*(*x)*(*x)*
    (*x)*(*x)*(*x)
                +12.5*(*x)*(*x)*(*x)*(*
    x)
                -100*(*x)*(*x)*(*x)+300
    *(*x)*(*x)
                -300*(*x)+60);
  default : return 1;
  }
}

static void name_to_basis(int name_basis)
{

  switch (name_basis){
  case 1 : basis=CanonicalD1;
  case 2 : basis=LaguerreD1;
  case 3 : basis=CanonicalOpD1;

  default : basis=CanonicalD1;
  }
}

static void InitBridge(long MC_Iterations,int
```

```
    mc_or_qmc,int generator,int dim,double t)
{
  int i;
  long j;
  double squareroott;

  squareroott=sqrt(t);

  for (j=0;j<MC_Iterations;j++)
  for (i=0;i<dim;i++)
    Pont[j*dim+i]=squareroott*Gaussians[mc_or_q
    mc](1, CREATE, 0, generator);

}

static void ComputeBridge(int k,double step, long
     MC_Iterations,int mc_or_qmc,int generator)
{
  double aux1,aux2,*ad,*admax;

  aux1=(double)k/(double)(k+1);
  aux2=sqrt(aux1*step);
  ad=Pont;
  admax=Pont+MC_Iterations;

  for (ad=Pont;ad<admax;ad++)
  *ad=aux1*(*ad)+aux2*Gaussians[mc_or_qmc](1, CR
    EATE, 0, generator);

  return;
}

static void BackwardPaths(double t, long MC_Itera
    tions,double s,double sigma,
              double r,double divid)
{

  long n,auxad;
  double forward_stock;

  forward_stock=s*exp(((r-divid)-0.5*SQR(sigma))*
```

```
    t);
  auxad=0;
  for (n=0;n<MC_Iterations;n++)
  Traj[n]=forward_stock*exp(sigma*Pont[n]);
}

static void Regression(long MC_Iterations,
    NumFunc_1 *p,int DimApp)
{
  int i,j,k;
  double *imDimApppj;

  for (i=0;i<DimApp;i++){
  AuxR[i]=0;
    for (j=0;j<DimApp;j++) M[i*DimApp+j]=0;
  }

  for(k=0;k<MC_Iterations;k++) {
  if ((p->Compute)(p->Par,*(Traj+k))>0){
    for (i=0;i<DimApp;i++){
    VBasis[i]=basis(Traj+k,i,p);
    }

    imDimApppj=M;
    for (i=0;i<DimApp;i++)
    for (j=0;j<DimApp;j++){
      *imDimApppj+=VBasis[i]*VBasis[j];
      imDimApppj+=1;
    }

    for (i=0;i<DimApp;i++){
    AuxR[i]+=FP[k]*VBasis[i];
    }
    }
  }

  Cholesky(M,DimApp);
  Resolution(AuxR,Res,M,DimApp);

  return;
```

```
}


static void LoScRet(double *PrixDir,long MC_Itera
    tions,NumFunc_1 *p,int name_basis,int DimApprox,
    int Fermeture,int mc_or_qmc,int generator,int exerc
    ise_date_number,double s, double t, double r,
    double divid, double sigma,int gj_flag)
{
  long i;
  int k,l;
  double AuxOption,discount1,step,AuxScal;

  /*Initialization of the regression basis*/
  name_to_basis(name_basis);

  /*Memory Allocation*/
  LongRet_Allocation(MC_Iterations,DimApprox,1);


  step=t/(exercise_date_number-1.);
  *PrixDir=0;

  /*Initialization of brownian bridge at maturity
    */
  InitBridge(MC_Iterations,mc_or_qmc,generator,1,
    t);

  /*Initialization of Black-Sholes Paths at matu
    rity*/
  BackwardPaths(t,MC_Iterations,s,sigma,r,divid);

  /*Payoff at maturity*/
  discount1=exp(-r*step);
  for (i=0;i<MC_Iterations;i++)
  {
    FP[i]=(p->Compute)(p->Par,*(Traj+i));
    if (FP[i]>0) FP[i]=discount1*FP[i];
  }

  /*Backward dynamical programming*/
```

```
for (k=exercise_date_number-2;k>=1;k--){

/*Backward simulation of the brownian bridge
  from time k+1 to k*/
ComputeBridge(k,step,MC_Iterations,mc_or_qmc,g
  enerator);

/*Backward simulation of Black-sholes Paths fr
  om time k+1 to k*/
BackwardPaths(k*step,MC_Iterations,s,sigma,r,
  divid);

/*Regression of FP with respect to Black-Shol
  es Paths at time k*/
  Regression(MC_Iterations,p,DimApprox);

/*Dynamical programming*/
  for (i=0;i<MC_Iterations;i++){
  AuxOption=(p->Compute)(p->Par,*(Traj+i));

  /*The regression take into account only at
  the money paths*/
  if (AuxOption>0){
  AuxScal=0.;
  for (l=0;l<DimApprox;l++)
    AuxScal+=basis(Traj+i,l,p)*Res[l];

  if (AuxOption> AuxScal)
    FP[i]=AuxOption;
  }
  FP[i]*=discount1;
}
}

/*At time 0, regression=mean*/
AuxOption=(p->Compute)(p->Par,s);
if (AuxOption>0){
Res[0]=0;
for (i=0;i<MC_Iterations;i++)
  Res[0]+=FP[i];
Res[0]/=MC_Iterations;
```

```
  if (!gj_flag){
    if (AuxOption>Res[0])
    for (i=0;i<MC_Iterations;i++)
      FP[i]=AuxOption;
  }
  }

  /*Mean along the optimal stopping time*/
  for (i=0;i<MC_Iterations;i++){
    *PrixDir+=FP[i];
  }

  /* Forward Price*/
  *PrixDir/=(double)MC_Iterations;

  /*Memory Disallocation*/
  if (Fermeture){
  LongRet_Liberation();
  }

  return;

}


static int MCLongstaffSchwartz(double s, NumFunc_
    1  *p, double t, double r, double divid, double
    sigma, long N, int generator, double inc,int basi
    s,int dimapprox, int exercise_date_number,double
    *ptprice, double *ptdelta)
{


  double s_plus,p1,p2,p3;
  int simulation_dim= 1,fermeture=1,init_mc,
    mc_or_qmc;

  /*Initialisation*/
  s_plus= s*(1.+inc);

  /*MC sampling*/
```

```
init_mc= InitGenerator(generator,simulation_dim
  ,N);

/* Test after initialization for the generator
  */
if(init_mc == OK)
  { mc_or_qmc= Rand_Or_Quasi(generator);

/*Geske-Johnson Formulae*/
if (exercise_date_number==0){
  LoScRet(&p1,N,p,basis,dimapprox,fermeture,
  mc_or_qmc,generator,2,s,t,r,divid,sigma,1);
  LoScRet(&p2,N,p,basis,dimapprox,fermeture,
  mc_or_qmc,generator,3,s,t,r,divid,sigma,1);
  LoScRet(&p3,N,p,basis,dimapprox,fermeture,
  mc_or_qmc,generator,4,s,t,r,divid,sigma,1);
  *ptprice=p3+7./2.*(p3-p2)-(p2-p1)/2.;
} else {
  LoScRet(ptprice,N,p,basis,dimapprox,fermetu
  re,mc_or_qmc,generator,exercise_date_number,s,t,
  r,divid,sigma,0);
}

/*Delta*/
if (exercise_date_number==0){
  LoScRet(&p1,N,p,basis,dimapprox,fermeture,
  mc_or_qmc,generator,2,s_plus,t,r,divid,sigma,1);
  LoScRet(&p2,N,p,basis,dimapprox,fermeture,
  mc_or_qmc,generator,3,s_plus,t,r,divid,sigma,1);
  LoScRet(&p3,N,p,basis,dimapprox,fermeture,
  mc_or_qmc,generator,4,s_plus,t,r,divid,sigma,1);
  *ptdelta=((p3+7./2.*(p3-p2)-(p2-p1)/2.)-*pt
  price)/(s*inc);
} else {
  LoScRet(&p1,N,p,basis,dimapprox,fermeture,
  mc_or_qmc,generator,exercise_date_number,s*(1+inc),
  t,r,divid,sigma,0);
  *ptdelta=(p1-*ptprice)/(s*inc);
}
}
```

```
  return init_mc;
}

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

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


  return MCLongstaffSchwartz(ptMod->S0.Val.V_PDOU
    BLE,
                ptOpt->PayOff.Val.V_
    NUMFUNC_1,
                ptOpt->Maturity.Val.V_DATE-
    ptMod->T.Val.V_DATE,
                r,
                divid,
                ptMod->Sigma.Val.V_PDOUBLE,
                Met->Par[0].Val.V_LONG,
                Met->Par[1].Val.V_INT,
                Met->Par[2].Val.V_PDOUBLE,
                Met->Par[3].Val.V_INT,
                Met->Par[4].Val.V_INT,
                Met->Par[5].Val.V_INT,
                &(Met->Res[0].Val.V_DOUBLE)
    ,
                &(Met->Res[1].Val.V_DOUBLE)
    );
}

int CHK_OPT(MC_LongstaffSchwartz)(void *Opt, void
     *Mod)
{
  Option* ptOpt=(Option*)Opt;
  TYPEOPT* opt=(TYPEOPT*)(ptOpt->TypeOpt);
```

```
  if ((opt->EuOrAm).Val.V_BOOL==AMER)
    return OK;
else
    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_LONG=50000;
      Met->Par[1].Val.V_INT=0;
      Met->Par[2].Val.V_PDOUBLE=0.1;
    Met->Par[3].Val.V_INT=1;
    Met->Par[4].Val.V_INT=4;
    Met->Par[5].Val.V_INT=20;
    first=0;
    }
  return OK;
}

PricingMethod MET(MC_LongstaffSchwartz)=
{
  "MC_LongstaffSchwartz",
  {{"N iterations",LONG,100,ALLOW},
   {"RandomGenerator",GENER,100,ALLOW},
   {"Delta Increment Rel",PDOUBLE,100,ALLOW},
   {"Basis:(1)CanonicalD1 (2)LaguerreD1 (3)Canon
    icalOpD1",INT,100,ALLOW},
   {"Dimension Approximation",INT,100,ALLOW},
   {"Number of Exercise Dates (0->Geske Johnson
    Formulae",INT,100,ALLOW},
   {" ",END,0,FORBID}},
  CALC(MC_LongstaffSchwartz),
  {{"Price",DOUBLE,100,FORBID},
   {"Delta",DOUBLE,100,FORBID} ,
```

```
   {" ",END,0,FORBID}},
  CHK_OPT(MC_LongstaffSchwartz),
  CHK_mc,
  MET(Init)
};
```

References