Help

```
#include <stdlib.h>
#include  "bs1d_std.h"
#include "error_msg.h"
#include "enums.h"
#include "pnl/pnl_matrix.h"

static double *FP, *Paths=NULL, *NextPaths=NULL, *Res=NULL;

static double *M=NULL, *AuxR=NULL, *VBasis, *Brownian_Brid
    ge=NULL,*Aux_BS=NULL;
static double (*Basis)(double *x, int i);
static double basis_norm;
static int TsRoB_Allocation(long MC_Iterations, int DimApp
    rox,int BS_Dimension)
{


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

  if (Paths==NULL)
    Paths= malloc(MC_Iterations*BS_Dimension*sizeof(double)
    );
  if (Paths==NULL)  return MEMORY_ALLOCATION_FAILURE;

  if (NextPaths==NULL)
    NextPaths= malloc(MC_Iterations*BS_Dimension*sizeof(
    double));
  if (NextPaths==NULL)  return MEMORY_ALLOCATION_FAILURE;

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

  if (Brownian_Bridge==NULL)
    Brownian_Bridge= malloc(MC_Iterations*BS_Dimension*size
    of(double));
  if (Brownian_Bridge==NULL) return MEMORY_ALLOCATION_FAILU
    RE;
```

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

  if (AuxR==NULL)
    AuxR= malloc(DimApprox*sizeof(double));
  if (AuxR==NULL) return MEMORY_ALLOCATION_FAILURE;

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

  if (Aux_BS==NULL)
    Aux_BS= malloc(BS_Dimension*sizeof(double));
  if (Aux_BS==NULL)
    return MEMORY_ALLOCATION_FAILURE;

  return OK;
}

static void TsRoB_Liberation()
{
  if (FP!=NULL) {
    free(FP);
    FP=NULL;
  }
  if (Paths!=NULL) {
    free(Paths);
    Paths=NULL;
  }
  if (NextPaths!=NULL) {
    free(NextPaths);
    NextPaths=NULL;
  }
  if (M!=NULL) {
    free(M);
    M=NULL;
  }
  if (AuxR!=NULL) {
    free(AuxR);
```

```
    AuxR=NULL;
  }
  if (VBasis!=NULL) {
    free(VBasis);
    VBasis=NULL;
  }
  if (Res!=NULL) {
    free(Res);
    Res=NULL;
  }
  if (Aux_BS!=NULL) {
    free(Aux_BS);
    Aux_BS=NULL;
  }
  if (Brownian_Bridge!=NULL) {
    free(Brownian_Bridge);
    Brownian_Bridge=NULL;
  }
  return;
}

static double product(double *vect,int AL_Basis_Dimension)
{
  int i;
  double aux=0;
  for (i=0;i<AL_Basis_Dimension;i++) aux+=Res[i]*Basis(vec
    t,i);
  return aux;
}

static void Regression(NumFunc_1 *p,long AL_MonteCarlo_
    Iterations, int OP_Exercise_Dates,int AL_Basis_Dimension, int
    BS_Dimension, int Time,int PayOff)
{
  double Aux,AuxOption;
  int i,j;
  long k;
  PnlMat _M;
  PnlVect _Res, _AuxR;
  for (i=0;i<AL_Basis_Dimension;i++){
    AuxR[i]=0;
```

```
    for (j=0;j<AL_Basis_Dimension;j++) M[i*AL_Basis_Dimens
    ion+j]=0;
  }

  for(k=0;k<AL_MonteCarlo_Iterations;k++){
    if (PayOff){
      VBasis[0]=(p->Compute) (p->Par,*(Paths+k*BS_Dimensio
    n));
      for (i=1;i<AL_Basis_Dimension;i++){
        VBasis[i]=Basis(Paths+k*BS_Dimension,i-1);
      }
    } else {
      for (i=0;i<AL_Basis_Dimension;i++){
        VBasis[i]=Basis(Paths+k*BS_Dimension,i);
      }
    }
    for (i=0;i<AL_Basis_Dimension;i++)
      for (j=0;j<AL_Basis_Dimension;j++)
        M[i*AL_Basis_Dimension+j]+=VBasis[i]*VBasis[j];


    AuxOption=(p->Compute) (p->Par,*(NextPaths+k*BS_Dimens
    ion));
    if (Time==OP_Exercise_Dates-2){
      Aux=AuxOption;
      FP[k]=AuxOption;
    } else {
      Aux=MAX(AuxOption,product(NextPaths+k*BS_Dimension,
    AL_Basis_Dimension));
      if (AuxOption==Aux){
        FP[k]=Aux;
      }
    }

    for (i=0;i<AL_Basis_Dimension;i++)
      AuxR[i]+=Aux*VBasis[i];
  }
  for (i=0;i<AL_Basis_Dimension;i++){
    AuxR[i]/=(double)AL_MonteCarlo_Iterations;
    for (j=0;j<AL_Basis_Dimension;j++){
      M[i*AL_Basis_Dimension+j]/=(double)AL_MonteCarlo_
```

```
    Iterations;
    }
  }

  _M =  pnl_mat_wrap_array(M, AL_Basis_Dimension, AL_Basis_
    Dimension);
  _Res =  pnl_vect_wrap_array(Res, AL_Basis_Dimension);
  _AuxR =  pnl_vect_wrap_array(AuxR, AL_Basis_Dimension);
  pnl_vect_clone (&_Res, &_AuxR);
  pnl_mat_ls (&_M, &_Res);
}

static double HermiteD1(double *x, int ind)
{
  double tmp;

  tmp = *x / basis_norm;
  switch (ind){
    case 0 : return 1;
    case 1 : return 1.414213562*(tmp);
    case 2 : return 1.414213562*(tmp)*(tmp)-0.707106781;
    case 3 : return (1.154700538*(tmp)*(tmp)-1.732050808)*(
    tmp);
    case 4 : return (0.816496581*(tmp)*(tmp)-2.449489743)*(
    tmp)*(tmp)+0.612372436;
    case 5 : return ((0.516397779*(tmp)*(tmp)-2.581988897)*
    (tmp)*(tmp)+1.936491673)*(tmp);

    default : return 1;
    }
}


static void Init_BrownianBridge(int generator,long MC_Itera
    tions,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++)
      Brownian_Bridge[j*dim+i]=squareroott*pnl_rand_normal(    generator);

}
static void Compute_Brownian_Bridge(int generator,double *
    Brownian_Bridge, double Time, double Step,
                                    int BS_Dimension,long
    MonteCarlo_Iterations)
{
  double aux1,aux2,*ad,*admax;

  aux1=Time/(Time+Step);
  aux2=sqrt(aux1*Step);
  ad=Brownian_Bridge;
  admax=Brownian_Bridge+BS_Dimension*MonteCarlo_Iterations;


  for (ad=Brownian_Bridge;ad<admax;ad++)
    *ad=aux1*(*ad)+aux2*pnl_rand_normal(generator);
}

static void Backward_Path(double *Paths, double *Brownian_
    Bridge,
                          double *BS_Spot,double Time,
                          long MonteCarlo_Iterations, int
    BS_Dimension,double *Sigma)
{
  int j,k;
  long n,auxad;
  double aux;

  auxad=0;
  for (n=0;n<MonteCarlo_Iterations;n++){
    for (j=0;j<BS_Dimension;j++){
      aux=0.;
      for (k=0;k<=j;k++)
        aux+=Sigma[j*BS_Dimension+k]*Brownian_Bridge[auxad+
    k];
      aux-=Time*Aux_BS[j];
```

```
      Paths[auxad+j]=BS_Spot[j]*exp(aux);
    }
    auxad+=BS_Dimension;
  }
}

static void TsRoB(double *AL_Price,long AL_MonteCarlo_Itera
    tions,NumFunc_1 *p,int AL_Basis_Dimension,int AL_ShuttingDo
    wn,int generator,int OP_Exercise_Dates,double *BS_Spot,
    double BS_Maturity,double BS_Interest_Rate,double *BS_Dividend_
    Rate,double *BS_Volatility)
{
  double DiscountStep,Step,Aux,AuxOption,AL_FPrice,AL_BPric
    e;
  long i;
  int j,k,PayOff,BS_Dimension=1;
  PayOff=0;
  /*Initialization of the regression basis*/
  basis_norm = *BS_Spot;
  Basis = HermiteD1;

  /*Memory Allocation*/
  TsRoB_Allocation(AL_MonteCarlo_Iterations,AL_Basis_Dimens
    ion,BS_Dimension);

  for(j=0;j<AL_Basis_Dimension;j++)
    Res[j]=0;
  AL_BPrice=0.;
  AL_FPrice=0.;

  /*Black-Sholes initalization parameters*/
  Aux_BS[0]=0.5*SQR(BS_Volatility[0])-BS_Interest_Rate+BS_
    Dividend_Rate[0];
  Step=BS_Maturity/(double)(OP_Exercise_Dates-1);
  DiscountStep=exp(-BS_Interest_Rate*Step);

  /*Initialization of brownian bridge at maturity*/
  Init_BrownianBridge(generator,AL_MonteCarlo_Iterations,
    BS_Dimension,BS_Maturity);

  /*Initialization of Black-Sholes Paths at maturity*/
```

```
Backward_Path(NextPaths,Brownian_Bridge,BS_Spot,BS_Matu
  rity,AL_MonteCarlo_Iterations,BS_Dimension,BS_Volatility);

/*Backward dynamical programming(backward and forward ar
  e both computed*/
for (k=OP_Exercise_Dates-2;k>=1;k--)
  {
    Compute_Brownian_Bridge(generator,Brownian_Bridge,k*
  Step,Step,BS_Dimension,AL_MonteCarlo_Iterations);
    Backward_Path(Paths,Brownian_Bridge,BS_Spot,k*Step,
  AL_MonteCarlo_Iterations,BS_Dimension,BS_Volatility);

    Regression(p,AL_MonteCarlo_Iterations,OP_Exercise_Da
  tes,
               AL_Basis_Dimension,BS_Dimension,k,PayOff);
    for (i=0;i<AL_MonteCarlo_Iterations;i++)
      {
        FP[i]*=DiscountStep;
      }

    for (j=0;j<AL_Basis_Dimension;j++)
      {
        Res[j]*=DiscountStep;
      }

    for (i=0;i<AL_MonteCarlo_Iterations;i++)
      {
        for (j=0;j<BS_Dimension;j++)
          {
            NextPaths[i*BS_Dimension+j]=Paths[i*BS_Dimens
  ion+j];
          }
      }
  }

for (i=0;i<AL_MonteCarlo_Iterations;i++)
  {
    AuxOption=(p->Compute) (p->Par,*(NextPaths+i*BS_Dim
  ension));
    Aux=MAX(AuxOption,product(NextPaths+i*BS_Dimension,
  AL_Basis_Dimension));
```

```
      if (fabs (AuxOption - Aux) < 0.0000001)
        {
          FP[i]=AuxOption;
        }
      AL_BPrice+=Aux;

      FP[i]*=DiscountStep;
    }

  AuxOption=(p->Compute)(p->Par,*BS_Spot);

  /*Backward Price*/
  AL_BPrice*=DiscountStep/(double)AL_MonteCarlo_Iterations;
  AL_BPrice=MAX(AuxOption,AL_BPrice);

  Aux=0;
  for (i=0;i<AL_MonteCarlo_Iterations;i++)
    {
      Aux+=FP[i];
    }
  Aux*=DiscountStep/(double)AL_MonteCarlo_Iterations;

  /* Forward Price */
  AL_FPrice=MAX(AuxOption,Aux);

  /*Price = Mean of Forward and Backward Price*/
  *AL_Price=0.5*(AL_FPrice+AL_BPrice);

  /*Memory Disallocation*/
  if (AL_ShuttingDown){
    TsRoB_Liberation();
  }
}

static int MCTsitsiklisVanRoy(double s, NumFunc_1  *p,
    double t, double r, double dividend, double sig, long N, int     generator,
    double *ptprice, double *ptdelta)
{

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

```
double s_vector[1];
double s_vector_plus[1];
double divid[1];
double sigma[1];

/*Initialisation*/
s_vector[0]=s;
s_vector_plus[0]=s*(1.+inc);
divid[0]=dividend;
sigma[0]=sig;

/*MC sampling*/
init_mc= pnl_rand_init(generator,simulation_dim,N);

/* Test after initialization for the generator */
if(init_mc != OK) return init_mc;

/*Geske-Johnson Formulae*/
if (exercise_date_number==0)
  {
    TsRoB(&p1,N,p,dimapprox,fermeture,generator,2,s_vec
  tor,t,r,divid,sigma);
    TsRoB(&p2,N,p,dimapprox,fermeture,generator,3,s_vec
  tor,t,r,divid,sigma);
    TsRoB(&p3,N,p,dimapprox,fermeture,generator,4,s_vec
  tor,t,r,divid,sigma);
    *ptprice=p3+7./2.*(p3-p2)-(p2-p1)/2.;
  }
else
  {
    TsRoB(ptprice,N,p,dimapprox,fermeture,generator,exerc
  ise_date_number,s_vector,t,r,divid,sigma);
  }

/*Delta*/
init_mc= pnl_rand_init(generator,simulation_dim,N);

if (exercise_date_number==0)
  {
    TsRoB(&p1,N,p,dimapprox,fermeture,generator,2,s_vec
  tor_plus,t,r,divid,sigma);
```

```
      TsRoB(&p2,N,p,dimapprox,fermeture,generator,3,s_vec
    tor_plus,t,r,divid,sigma);
      TsRoB(&p3,N,p,dimapprox,fermeture,generator,4,s_vec
    tor_plus,t,r,divid,sigma);
      *ptdelta=((p3+7./2.*(p3-p2)-(p2-p1)/2.)-*ptprice)/(s*
    inc);
    }
  else
    {
      TsRoB(&p1,N,p,dimapprox,fermeture,generator,exercise_
    date_number,s_vector_plus,t,r,divid,sigma);
      *ptdelta=(p1-*ptprice)/(s*inc);
    }
  return OK;
}

int CALC(MC_TsitsiklisVanRoy)(void *Opt, void *Mod, Pricing
    Method *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 MCTsitsiklisVanRoy(ptMod->S0.Val.V_PDOUBLE,
                            ptOpt->PayOff.Val.V_NUMFUNC_1,
                            ptOpt->Maturity.Val.V_DATE-pt
    Mod->T.Val.V_DATE,
                            r,
                            divid,
                            ptMod->Sigma.Val.V_PDOUBLE,
                            Met->Par[0].Val.V_LONG,
                            Met->Par[1].Val.V_ENUM.value,
                            Met->Par[2].Val.V_PDOUBLE,
                            Met->Par[3].Val.V_INT,
                            Met->Par[4].Val.V_INT,
                            &(Met->Res[0].Val.V_DOUBLE),
                            &(Met->Res[1].Val.V_DOUBLE));
```

```
}

static int CHK_OPT(MC_TsitsiklisVanRoy)(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,Option *Opt)
{
  if ( Met->init == 0)
    {
      Met->init=1;

      Met->Par[0].Val.V_LONG=50000;
      Met->Par[1].Val.V_ENUM.value=0;
      Met->Par[1].Val.V_ENUM.members=&PremiaEnumMCRNGs;
      Met->Par[2].Val.V_PDOUBLE=0.01;
      Met->Par[3].Val.V_INT=8;
      Met->Par[4].Val.V_INT=20;

    }


  return OK;
}

PricingMethod MET(MC_TsitsiklisVanRoy)=
{
  "MC_TsitsiklisVanRoy",
  {{"N iterations",LONG,{100},ALLOW},
   {"RandomGenerator",ENUM,{100},ALLOW},
   {"Delta Increment Rel",PDOUBLE,{100},ALLOW},
   {"Dimension Approximation",INT,{100},ALLOW},
   {"Number of Exercise Dates (0->Geske Johnson Formulae",
```

```
    INT,{100},ALLOW},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(MC_TsitsiklisVanRoy),
  {{"Price",DOUBLE,{100},FORBID},
   {"Delta",DOUBLE,{100},FORBID} ,
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(MC_TsitsiklisVanRoy),
  CHK_mc,
  MET(Init)
};
```

References