Help

```
#include <stdlib.h>
#include  "
href../../../mod/bs2d/bs2d_std2d/bs2d_std2d_h_src.pdfbs2d_std2d.h"
#include "
href../../../common/error_msg_h_src.pdferror_msg.h"
#include "
href../../../common/enums_h_src.pdfenums.h"

static double *FP = NULL, *Traj = NULL;
static PnlMat *M = NULL;
static PnlVect *AuxR = NULL, *VBase = NULL, *Res = NULL;

static double *Pont = NULL;
static double (*basis)(double *stock, int l, NumFunc_2 *p);

static int LongRet_Allocation(long MC_Iterations, int DimApprox, int DimBS)
{
  if (FP == NULL)
    FP = malloc(MC_Iterations * sizeof(double));

  if (FP == NULL) return MEMORY_ALLOCATION_FAILURE;

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

  if (Traj == NULL) return MEMORY_ALLOCATION_FAILURE;

  if (M == NULL) M = pnl_mat_create(DimApprox, DimApprox);
  if (M == NULL) return MEMORY_ALLOCATION_FAILURE;

  if (Res == NULL) Res = pnl_vect_create(DimApprox);
  if (Res == NULL) return MEMORY_ALLOCATION_FAILURE;

  if (AuxR == NULL) AuxR = pnl_vect_create(DimApprox);
  if (AuxR == NULL) return MEMORY_ALLOCATION_FAILURE;

  if (VBase == NULL) VBase = pnl_vect_create(DimApprox);
  if (VBase == NULL) return MEMORY_ALLOCATION_FAILURE;
```

```
  if (Pont == NULL)
    Pont = malloc(MC_Iterations * DimBS * sizeof(double));
  if (Pont == NULL) return MEMORY_ALLOCATION_FAILURE;
  return OK;
}

static void LongRet_Liberation()
{
  if (FP != NULL)
    {
      free(FP);
      FP = NULL;
    }
  if (Traj != NULL)
    {
      free(Traj);
      Traj = NULL;
    }
  if (M != NULL)
    {
      pnl_mat_free(&M);
    }
  if (Res != NULL)
    {
      pnl_vect_free(&Res);
    }
  if (AuxR != NULL)
    {
      pnl_vect_free(&AuxR);
    }
  if (VBase != NULL)
    {
      pnl_vect_free(&VBase);
    }

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

```
  return;
}

/*Canonical Basis for Regression*/
double CanonicalD2(double *x, int ind, NumFunc_2 *p)
{
  switch (ind)
    {
    case 0 :
      return 1;

    case 1 :
      return x[0];
    case 2 :
      return x[1];

    case 3 :
      return x[0] * x[0];
    case 4 :
      return x[1] * x[1];

    case 5 :
      return x[0] * x[1];

    case 6 :
      return x[0] * x[0] * x[0];
    case 7 :
      return x[1] * x[1] * x[1];

    case 8 :
      return x[0] * x[1] * x[1];
    case 9 :
      return x[1] * x[0] * x[0];

    default :
      return 1;
    }
}

/*Basis Regression=Payoff + Canonnical*/
double CanonicalOpD2(double *x, int ind, NumFunc_2 *p)
```

```
{
  if (ind == 0) return (p->Compute)(p->Par, *x, *(x + 1));
  else return CanonicalD2(x, ind - 1, p);
}


static void name_to_basis(int name_basis)
{

  switch (name_basis)
    {
    case 1 :
      basis = CanonicalD2;
    case 2 :
      basis = CanonicalOpD2;

    default :
      basis = CanonicalD2;
    }
}

static void InitBridge(long MC_Iterations, 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 * pnl_rand_normal(generator);
      }


}

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

```
  aux1 = (double)k / (double)(k + 1);
  aux2 = sqrt(aux1 * step);
  ad = Pont;
  admax = Pont + 2 * MC_Iterations;

  for (ad = Pont; ad < admax; ad++)
    {
      *ad = aux1 * (*ad) + aux2 * pnl_rand_normal(generator);
    }
  return;
}

static void BackwardPaths(double t, long MC_Iterations, double s1, double s2, do
{

  long n;
  double forward_stock1, forward_stock2;

  forward_stock1 = s1 * exp(((r - divid1) - 0.5 * SQR(sigma11)) * t);
  forward_stock2 = s2 * exp(((r - divid2) - 0.5 * (SQR(sigma21) + SQR(sigma22)))
  for (n = 0; n < MC_Iterations; n++)
    {
      Traj[2 * n] = forward_stock1 * exp(sigma11 * Pont[2 * n]);
      Traj[2 * n + 1] = forward_stock2 * exp(sigma21 * Pont[2 * n] + sigma22 * P
    }

}

static void Regression(long MC_Iterations, NumFunc_2 *p, int DimApp)
{
  int i, j, k;

  pnl_vect_set_double(AuxR, 0.0);
  pnl_mat_set_double(M, 0.0);

  for (k = 0; k < MC_Iterations; k++)
    {
      if ((p->Compute)(p->Par, *(Traj + 2 * k), *(Traj + 2 * k + 1)) > 0)
        {
          for (i = 0; i < DimApp; i++)
```

```
            {
              pnl_vect_set(VBase, i, basis(Traj + 2 * k, i, p));
            }

          for (i = 0; i < DimApp; i++)
            for (j = 0; j < DimApp; j++)
              {
                double tmp = pnl_mat_get(M, i, j);
                pnl_mat_set(M, i, j , tmp + pnl_vect_get(VBase, i) *
                            pnl_vect_get(VBase, j));
              }

          for (i = 0; i < DimApp; i++)
            {
              double tmp = pnl_vect_get(AuxR, i);
              pnl_vect_set(AuxR, i, FP[k] * pnl_vect_get(VBase, i) + tmp);
            }
        }
    }
  pnl_vect_clone(Res, AuxR);
  /* solve in the least square sense, using a QR decomposition */
  pnl_mat_ls(M, Res);

  return;
}


static void LoScRet(double *PrixDir, long MC_Iterations, NumFunc_2 *p, int name_
{
  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, 2);

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

```
/*Initialization of brownian bridge at maturity*/
InitBridge(MC_Iterations, generator, 2, t);

/*Initialization of Black-Sholes Paths at maturity*/
BackwardPaths(t, MC_Iterations, s1, s2, sigma11, sigma21, sigma22, r, divid1,

/*Payoff at maturity*/
discount1 = exp(-r * step);
for (i = 0; i < MC_Iterations; i++)
  {
    FP[i] = (p->Compute)(p->Par, *(Traj + 2 * i), *(Traj + 2 * i + 1));
    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, generator);

    /*Backward simulation of Black-sholes Paths from time k+1 to k*/
    BackwardPaths(k * step, MC_Iterations, s1, s2, sigma11, sigma21, sigma22,

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

    for (i = 0; i < MC_Iterations; i++)
      {
        AuxOption = (p->Compute)(p->Par, *(Traj + 2 * i), *(Traj + 2 * i + 1))

        /*The regression take into account only at the money paths*/
        if (AuxOption > 0)
          {
            AuxScal = 0.;
            for (l = 0; l < DimApprox; l++)
              AuxScal += basis(Traj + 2 * i, l, p) * pnl_vect_get(Res, l);

            if (AuxOption > AuxScal)
              FP[i] = AuxOption;
```

```
          }
        FP[i] *= discount1;
      }
  }

  /*At time 0, regression=mean*/
  AuxOption = (p->Compute)(p->Par, s1, s2);
  if (AuxOption > 0)
    {
      double tmp = 0.;
      for (i = 0; i < MC_Iterations; i++) tmp += FP[i];
      tmp /= MC_Iterations;
      if (!gj_flag)
        {
          if (AuxOption > tmp)
            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 LongstaffSchwartz2DMC(double s1, double s2, NumFunc_2  *p, double t,
{
```

```
double s1_plus, s2_plus, p1, p2, p3, sigma11, sigma21, sigma22;
int simulation_dim = 1, fermeture = 1, init_mc;

/*Initialisation*/
s1_plus = s1 * (1. + inc);
s2_plus = s2 * (1. + inc);

/* Covariance Matrix */
/* Coefficients of the matrix A such that A(tA)=Gamma */
sigma11 = sigma1;
//sigma12= 0.0;
sigma21 = rho * sigma2;
sigma22 = sigma2 * sqrt(1.0 - SQR(rho));

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

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

    /*Geske-Johnson Formulae*/
    if (exercise_date_number == 0)
      {
        LoScRet(&p1, N, p, basis, dimapprox, fermeture, generator, 2, s1, s2,
        LoScRet(&p2, N, p, basis, dimapprox, fermeture, generator, 3, s1, s2,
        LoScRet(&p3, N, p, basis, dimapprox, fermeture, generator, 4, s1, s2,
        *ptprice = p3 + 7. / 2.*(p3 - p2) - (p2 - p1) / 2;
      }
    else
      {
        LoScRet(ptprice, N, p, basis, dimapprox, fermeture, generator, exercis
      }

    /*Delta*/
    if (exercise_date_number == 0)
      {
        LoScRet(&p1, N, p, basis, dimapprox, fermeture, generator, 2, s1_plus,
        LoScRet(&p2, N, p, basis, dimapprox, fermeture, generator, 3, s1_plus,
```

```
          LoScRet(&p3, N, p, basis, dimapprox, fermeture, generator, 4, s1_plus,
          *ptdelta1 = ((p3 + 7. / 2.*(p3 - p2) - (p2 - p1) / 2) - *ptprice) / (s
          LoScRet(&p1, N, p, basis, dimapprox, fermeture, generator, 2, s1, s2_p
          LoScRet(&p2, N, p, basis, dimapprox, fermeture, generator, 3, s1, s2_p
          LoScRet(&p3, N, p, basis, dimapprox, fermeture, generator, 4, s1, s2_p
          *ptdelta2 = ((p3 + 7. / 2.*(p3 - p2) - (p2 - p1) / 2) - *ptprice) / (s
        }
      else
        {
          LoScRet(&p1, N, p, basis, dimapprox, fermeture, generator, 2, s1_plus,
          *ptdelta1 = (p1 - *ptprice) / (s1 * inc);
          LoScRet(&p2, N, p, basis, dimapprox, fermeture, generator, 3, s1, s2_p
          *ptdelta2 = (p2 - *ptprice) / (s2 * inc);
        }
    }

  return init_mc;
}



int CALC(MC_LongstaffSchwartz2D)(void *Opt, void *Mod, PricingMethod *Met)
{
  TYPEOPT *ptOpt = (TYPEOPT *)Opt;
  TYPEMOD *ptMod = (TYPEMOD *)Mod;
  double r, divid1, divid2;

  r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);
  divid1 = log(1. + ptMod->Divid1.Val.V_DOUBLE / 100.);
  divid2 = log(1. + ptMod->Divid2.Val.V_DOUBLE / 100.);

  return LongstaffSchwartz2DMC(ptMod->S01.Val.V_PDOUBLE,
                               ptMod->S02.Val.V_PDOUBLE,
                               ptOpt->PayOff.Val.V_NUMFUNC_2,
                               ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE,
                               r,
                               divid1,
                               divid2,
                               ptMod->Sigma1.Val.V_PDOUBLE,
                               ptMod->Sigma2.Val.V_PDOUBLE,
                               ptMod->Rho.Val.V_RGDOUBLE,
```

```
                                    Met->Par[0].Val.V_LONG,
                                    Met->Par[1].Val.V_ENUM.value,
                                    Met->Par[2].Val.V_DOUBLE,
                                    Met->Par[3].Val.V_ENUM.value,
                                    Met->Par[4].Val.V_INT,
                                    Met->Par[5].Val.V_INT,
                                    &(Met->Res[0].Val.V_DOUBLE),
                                    &(Met->Res[1].Val.V_DOUBLE),
                                    &(Met->Res[2].Val.V_DOUBLE));
}

static int CHK_OPT(MC_LongstaffSchwartz2D)(void *Opt, void *Mod)
{
  Option *ptOpt = (Option *)Opt;
  TYPEOPT *opt = (TYPEOPT *)(ptOpt->TypeOpt);

  if ((opt->EuOrAm).Val.V_BOOL == AMER)
    return OK;

  return  WRONG;

}

static PremiaEnumMember Basis2dMembers[] =
{
  { "Canonical", 1 },
  { "CanonicalOpD2", 2 },
  { NULL, NULLINT }
};

static DEFINE_ENUM(Basis2d, Basis2dMembers);

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 = 0.1;
      Met->Par[3].Val.V_ENUM.value = 2;
      Met->Par[3].Val.V_ENUM.members = &Basis2d;
      Met->Par[4].Val.V_INT = 9;
      Met->Par[5].Val.V_INT = 20;

    }
  return OK;
}


PricingMethod MET(MC_LongstaffSchwartz2D) =
{
  "MC_LongstaffSchwartz2d",
  { {"N iterations", LONG, {100}, ALLOW},
    {"RandomGenerator", ENUM, {100}, ALLOW},
    {"Delta Increment Rel", PDOUBLE, {100}, ALLOW},
    {"Basis", ENUM, {100}, ALLOW},
    {"Dimension Approximation", INT, {100}, ALLOW},
    {"Number of Exercise Dates (0->Geske Johnson Formulae)", INT, {100}, ALLOW},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CALC(MC_LongstaffSchwartz2D),
  { {"Price", DOUBLE, {100}, FORBID},
    {"Delta1", DOUBLE, {100}, FORBID} ,
    {"Delta2", DOUBLE, {100}, FORBID},
    {" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CHK_OPT(MC_LongstaffSchwartz2D),
  CHK_mc,
  MET(Init)
};
```