Help

```
#include <stdlib.h>
#include "bs1d_pad.h"
#include "error_msg.h"

static double **vpm, * *vm;
static int *nb_critical;
static int current_index1, current_index2;

static void SortVect(unsigned long n, double *arr)
{
  PnlVect a;
  PnlVectInt *i;

  a = pnl_vect_wrap_array(arr, n);
  i = pnl_vect_int_create(0);
  pnl_vect_qsort_index(&a, i, 'i');
  pnl_vect_int_free(&i);
}

static double linear_interpolation1(double val, int j)
{
  int k;
  double res;
  int nb_critical_v;

  nb_critical_v = nb_critical[j];
  if (val < vm[j][0])
    return vpm[j][0];
  else if (val > vm[j][nb_critical_v])
    return vpm[j][nb_critical_v];
  else if (fabs(val - vm[j][nb_critical_v]) < 1.e-8)
    return vpm[j][nb_critical_v];
  else
    {
      k = current_index1;
      while ((vm[j][k] < val) && (k <= nb_critical_v)) k++;
      current_index1 = k;
      if (k == 0)
        res = vpm[j][0];
```

```
      else
        res = ((val - vm[j][k - 1]) * vpm[j][k] + (vm[j][k] - val) * vpm[j][k -
      return res;
    }
}

static double linear_interpolation2(double val, int j)
{
  int k;
  double res;
  int nb_critical_v;

  nb_critical_v = nb_critical[j];
  if (val < vm[j][0])
    return vpm[j][0];
  else if (val > vm[j][nb_critical_v])
    return vpm[j][nb_critical_v];
  else if (fabs(val - vm[j][nb_critical_v]) < 1.e-8)
    return vpm[j][nb_critical_v];
  else
    {
      k = current_index2;
      while ((vm[j][k] < val) && (k <= nb_critical_v)) k++;
      current_index2 = k;
      if (k == 0)
        res = vpm[j][0];
      else
        res = ((val - vm[j][k - 1]) * vpm[j][k] + (vm[j][k] - val) * vpm[j][k -

      return res;
    }
}

static int Asian_SingularPoints_Sup(int am, double s, double pseudo_spot, double
{
  double u, d, h, pu, pd, spot_value, val_int;
  double K = p->Par[0].Val.V_DOUBLE;
  double *v_min, *v_max;
  double *new_vm, *new_vm1, *new_vm2, *new_vpm, *new_vpm1, *new_vpm2;
  double *VectS;
  double stock, upperstock;
```

```
double average1, average2, price1, price2;
double TOL1 = 0.00000000001;
double TOL2 = 0.00000000001;
int i, j, k = 0, l, jj, kk, new_nb_critical;
int i1, index;
double m1, error, errp;
int  old_nb_critical;
double x1, x2, y2, y11, a, b;
int n_max;

//Number maximum of singular points
n_max = 50000;

/*Memory allocations*/
nb_critical = (int *)malloc(sizeof(int) * (n + 2));

vm = (double **)malloc(sizeof(double *) * (n + 2));
vpm = (double **)malloc(sizeof(double *) * (n + 2));

new_vm = (double *)malloc(sizeof(double) * (n_max));
new_vpm = (double *)malloc(sizeof(double) * (n_max));
new_vm1 = (double *)malloc(sizeof(double) * (n_max));
new_vpm1 = (double *)malloc(sizeof(double) * (n_max));
new_vm2 = (double *)malloc(sizeof(double) * (n_max));
new_vpm2 = (double *)malloc(sizeof(double) * (n_max));
VectS = (double *)malloc(sizeof(double) * (2 * n + 2));
v_min = (double *)malloc(sizeof(double) * (n + 2));
v_max = (double *)malloc(sizeof(double) * (n + 2));

for (i = 0; i <= n + 1; i++)
  {
    vm[i] = (double *)malloc(n_max * sizeof(double));
    vpm[i] = (double *)malloc(n_max * sizeof(double));
  }

/*Up and Down factors*/
h = t / (double)n;
u = exp(sigma * sqrt(h));
d = 1. / u;

/*Risk-Neutral Probability*/
```

```
pu = (exp(h * (r - divid)) - d) / (u - d);
pd = 1. - pu;

if ((pd >= 1.) || (pd <= 0.))
  return NEGATIVE_PROBABILITY;

pu *= exp(-r * h);
pd *= exp(-r * h);

//Asset values
upperstock = s;
for (i = 0; i < n; i++)
  upperstock *= u;
stock = upperstock;

for (i = 0; i < 2 * n + 1; i++)
  {
    stock *= d;
    VectS[i] = stock;
  }
/***Singular points at Maturity****/
for (j = 0; j <= n; j++)
  {
    v_min[j] = pseudo_spot / (n + 1) * ((1. - pow(d, (double)(j + 1))) / (1. -
                                        + pow(d, (double)(j)) * ((1 - pow(u, (
    v_max[j] = pseudo_spot / (n + 1) * ((1. - pow(u, (double)(n - j + 1))) / (
                                        + pow(u, (double)(n - j)) * ((1 - pow(
    if ((v_min[j] < K) && (v_max[j] > K))
      {
        nb_critical[j] = 2;

        //Abscissa
        vm[j][0] = v_min[j] + asian_spot;
        vm[j][1] = K + asian_spot;
        vm[j][2] = v_max[j] + asian_spot;

        //Ordinate
        vpm[j][0] = (p->Compute)(p->Par, pseudo_spot, vm[j][0]);
        vpm[j][1] = (p->Compute)(p->Par, pseudo_spot, vm[j][1]);
        vpm[j][2] = (p->Compute)(p->Par, pseudo_spot, vm[j][2]);
      }
```

```
      else
        {
          nb_critical[j] = 1;

          /*Abscissa*/
          vm[j][0] = v_min[j] + asian_spot;
          vm[j][1] = v_max[j] + asian_spot;

          /*Ordinate*/
          vpm[j][0] = (p->Compute)(p->Par, pseudo_spot, vm[j][0]);
          vpm[j][1] = (p->Compute)(p->Par, pseudo_spot, vm[j][1]);

        }
    }

  /***Backward algorithm****/
  for (i = n - 1; i >= 0; i--)
    {
      //Compute Singular Pints on Node j at time n-i
      for (j = 0; j <= i; j++)
        {
          spot_value = VectS[n - 1 - i + 2 * j];

          //Compute Singular Abscissa

          //Average Min and Max
          v_min[j] = ((i + 2) * v_min[j] - spot_value * u) / (double)(i + 1);
          v_max[j] = ((i + 2) * v_max[j + 1] - spot_value * d) / (double)(i + 1)
          //Average Min
          new_vm[0] = v_min[j];

          //Interior Average
          if (i > 0)
            {
              k = 1;
              //up
              for (l = 1; l < nb_critical[j]; l++)
                {
                  val_int = ((double)(i + 2) * vm[j][l] - spot_value * u) / (dou
                  if ((val_int <= v_max[j]) && (val_int >= v_min[j]))
                    {
```

```
              new_vm[k] = val_int;
              k++;
            }
        }
      //down
      for (l = 1; l < nb_critical[j + 1]; l++)
        {
          val_int = ((double)(i + 2) * vm[j + 1][l] - spot_value * d) /
          if ((val_int <= v_max[j]) && (val_int >= v_min[j]))
            {
              new_vm[k] = val_int;
              k++;
            }
        }
    }
//Average Max
new_vm[k] = v_max[j];
new_nb_critical = k;

/*Sorting*/
SortVect(new_nb_critical, new_vm);

for (k = 0; k <= new_nb_critical; k++)
  new_vm1[k] = new_vm[k];

//Remove singular points very close TOL1=e-10,TOL2=e-10
new_vm1[0] = new_vm[0];
kk = 0;
l = 0;
do
  {
    do
      {
        l++;
      }
    while ((new_vm[l] <= new_vm1[kk] + TOL1) && (l < new_nb_critical))
    kk++;

    new_vm1[kk] = new_vm[l];
  }
while ((l < new_nb_critical));
```

```
new_nb_critical = kk;

if (fabs(new_vm1[new_nb_critical] - new_vm1[new_nb_critical - 1]) < TO
  new_nb_critical--;

current_index1 = 0;
current_index2 = 0;

//Compute Singular Ordinate
for (k = 0; k <= new_nb_critical; k++)
  {
    average1 = ((double)(i + 1) * new_vm1[k] + spot_value * d) / (doub
    price1 = linear_interpolation1(average1, j + 1);
    average2 = ((double)(i + 1) * new_vm1[k] + spot_value * u) / (doub
    price2 = linear_interpolation2(average2, j);
    new_vpm1[k] = pd * price1 + pu * price2;
  }

for (k = 0; k <= new_nb_critical; k++)
  {
    new_vm2[k] = new_vm1[k];
    new_vpm2[k] = new_vpm1[k];
  }

//Upper bound
i1 = 0;
index = 0;
new_vm2[0] = new_vm1[0];
new_vpm2[0] = new_vpm1[0];
while (i1 < new_nb_critical - 1)
  {
    l = 1;
    do
      {
        l++;
        m1 = (new_vpm1[i1 + l] - new_vpm1[i1]) / (new_vm1[i1 + l] - ne
        error = 0.;
        for (jj = 1; jj <= l - 1; jj++)
          {
            errp = m1 * (new_vm1[i1 + jj] - new_vm1[i1]) + new_vpm1[i1
```

```
            if (errp < 0) errp = -errp;
            if (errp > error) error = errp;
          }
      }
    while (!((error > h_up) || ((i1 + l) == new_nb_critical)));
    index++;
    new_vm2[index] = new_vm1[i1 + l - 1];
    new_vpm2[index] = new_vpm1[i1 + l - 1];
    i1 = i1 + l - 1;
  }
while (i1 < new_nb_critical)
  {
    index++;
    new_vm2[index] = new_vm1[i1 + 1];
    new_vpm2[index] = new_vpm1[i1 + 1];
    i1 = i1 + 1;
  }
new_nb_critical = index;

//American Call case
if (am == 1)
  {
    old_nb_critical = new_nb_critical;
    if (MAX(0., new_vm2[0] - K) <= new_vpm2[0])
      {
        l = 1;
        while ((new_vpm2[l] >= MAX(0., new_vm2[l] - K)) && (l <= new_n
          {
            l++;
            if (l > new_nb_critical)  break;
          }
        if (l <= new_nb_critical)
          {
            new_nb_critical = l + 1;
            x1 = new_vm2[l - 1];
            x2 = new_vm2[l];
            y11 = new_vpm2[l - 1];
            y2 = new_vpm2[l];

            a = (y2 - y11) / (x2 - x1);
            b = (y11 * x2 - x1 * y2) / (x2 - x1);
```

```
                    new_vm2[l] = (K + b) / (1. - a);

                    new_vpm2[l] = MAX(0., new_vm2[l] - K);

                    new_vm2[l + 1] = new_vm2[old_nb_critical];
                    new_vpm2[l + 1] = MAX(0., new_vm2[l + 1] - K);
                  }
              }
          }
        nb_critical[j] = new_nb_critical;
        //Copy
        for (l = 0; l <= nb_critical[j]; l++)
          {
            vm[j][l] = new_vm2[l];
            vpm[j][l] = new_vpm2[l];
          }
      }
    //Delta
    if (i == 1)
      *ptdelta = (vpm[1][0] - vpm[0][0]) / (2.*(vm[1][0] - vm[0][0]));
  }

/*Price*/
*ptprice = vpm[0][0];

//Memory desallocation
for (i = 0; i <= n + 1; i++)
  free(vm[i]);
free(vm);

for (i = 0; i <= n + 1; i++)
  free(vpm[i]);
free(vpm);

free(nb_critical);
free(new_vm);
free(new_vm1);
free(new_vm2);
free(new_vpm);
free(new_vpm1);
```

```
  free(new_vpm2);
  free(VectS);
  free(v_min);
  free(v_max);

  return OK;
}

int CALC(TR_Asian_SingularPointsSup)(void *Opt, void *Mod, PricingMethod *Met)
{
  TYPEOPT *ptOpt = (TYPEOPT *)Opt;
  TYPEMOD *ptMod = (TYPEMOD *)Mod;
  double r, divid, time_spent, asian_spot, pseudo_spot, T_0, t_0, T;
  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 = (ptOpt->PathDep.Val.V_NUMFUNC_2)->Par[0].Val.V_PDOUBLE;
  T_0 = ptMod->T.Val.V_DATE;

  time_spent = (T_0 - t_0) / (T - t_0);
  asian_spot = (ptOpt->PathDep.Val.V_NUMFUNC_2)->Par[4].Val.V_PDOUBLE * time_spe
  pseudo_spot = (1. - time_spent) * ptMod->S0.Val.V_PDOUBLE;

  if (T_0 < t_0)
    {
      return_value = 0;
    }
  else
    {
      /* if  (((ptOpt->PayOff.Val.V_NUMFUNC_2)->Compute==Call_StrikeSpot2)||
      *  ((ptOpt->PayOff.Val.V_NUMFUNC_2)->Compute==Put_StrikeSpot2))
       *   Floating Case
       *   type_asian=1;
       * else type_asian=0; */


      return_value = Asian_SingularPoints_Sup(ptOpt->EuOrAm.Val.V_BOOL, ptMod->S
    }
```

```
  return return_value;
}

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

  return WRONG;
}

static int MET(Init)(PricingMethod *Met, Option *Opt)
{
  if (Met->init == 0)
    {
      Met->init = 1;
      Met->Par[0].Val.V_INT2 = 200;
      Met->Par[1].Val.V_PDOUBLE = 0.0001;
    }
  return OK;
}


PricingMethod MET(TR_Asian_SingularPointsSup) =
{
  "TR_Asian_SingularPointsSup",
  {{"StepNumber", INT2, {100}, ALLOW}, {"Tollerance Error", PDOUBLE, {100}, ALLO
  CALC(TR_Asian_SingularPointsSup),
  {{"Price", DOUBLE, {100}, FORBID}, {"Delta", DOUBLE, {100}, FORBID} , {" ", PR
  CHK_OPT(TR_Asian_SingularPointsSup),
  CHK_tree,
  MET(Init)
};
```