Help

```
#include <stdlib.h>
#include  "bsdisdiv1d_std.h"
#include "error_msg.h"
#include "pnl/pnl_matrix.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2009+2) //The "#else" part of the code will be freely av
    ailable after the (year of creation of this file + 2)
static int CHK_OPT(TR_SingularPoints_Down)(void *Opt, void
    *Mod)
{
  return NONACTIVE;
}
int CALC(TR_SingularPoints_Down)(void*Opt,void *Mod,Pricing
    Method *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

static double *vm, *vpm;

static void Sort2Vect(unsigned long n, double *arr,double *
    arr2)
{
  PnlVect a, b;
  PnlVectInt *i;
  a = pnl_vect_wrap_array (arr, n);
  b = pnl_vect_wrap_array (arr2, n);
  i = pnl_vect_int_create(0);

  pnl_vect_qsort_index(&a, i, 'i');
  pnl_vect_permute_inplace(&b, (PnlPermutation *)i);
  pnl_vect_int_free (&i);
}

static double compute_pricesp(double val,int nb_critical_v)
{
  int k;
```

```
  double res;

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

static int SingularPoints_Down(int am,double s,NumFunc_1  *
    p,double t,double r,double sigma,PnlVect *divid_dates,PnlV
    ect *divid_amounts,int N,double h_low,double *ptprice,
    double *ptdelta)
{

  int n;
  double u,d,h,pu,pd,a1;
  double K=p->Par[0].Val.V_DOUBLE;
  double *v_min,*v_max,*new_vm,*new_vm1,*new_vm2,*new_vpm,*
    new_vpm1,*new_vpm2,*coeff;
  double *vect_t;
  int * nb_critical;
  int *divid_steps;
  double value1,value2,value_price1,value_price2;
  double dist1,dist2;
  double TOL1=1.e-8;
  double TOL2=1.e-10;
```

```
double TOL3=1.e-10;
int i,j,k,l,jj,kk,Nb_div,new_nb_critical,n_k;
int max_critical=0;
int i1,nb_critical_put;
int  old_nb_critical;
double x1,x2,y2,y11,a,b;
double m11,m2,m3,x,yy,x11;
int indice1;
int n_max;

/*Number of Dividends Dates*/
Nb_div=divid_dates->size;

//Number maximum of singular points
n_max=50000;

/*Compute steps of the tree*/
n=N*Nb_div;


/*Memory allocations*/
divid_steps= malloc((n+1)*sizeof(int));
nb_critical=(int*)malloc(sizeof(int)*(n+1));
v_min=(double*)malloc(sizeof(double)*(n+1));
v_max=(double*)malloc(sizeof(double)*(n+1));
vm=(double*)malloc(sizeof(double)*(n_max));
vpm=(double*)malloc(sizeof(double)*(n_max));
new_vm=(double*)malloc(sizeof(double)*(n_max));
new_vm1=(double*)malloc(sizeof(double)*(n_max));
new_vm2=(double*)malloc(sizeof(double)*(n_max));
new_vpm=(double*)malloc(sizeof(double)*(n_max));
new_vpm1=(double*)malloc(sizeof(double)*(n_max));
new_vpm2=(double*)malloc(sizeof(double)*(n_max));
coeff=(double*)malloc(sizeof(double)*(n_max));
vect_t= malloc((n+1)*sizeof(double));

/*Down and Down factors*/
h=t/(double)n;
a1= exp(h*r);
u = exp(sigma*sqrt(h));
d= 1./u;
```

```
/*Risk-Neutral Probability*/
pu=(a1-d)/(u-d);
pd=1.-pu;

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

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

for(i=0;i<=n;i++)
  vect_t[i]=h*(double)i;

//Compute steps related to the dividend dates
for(k=0;k<Nb_div;k++)
  {
    i=0;
    while(vect_t[i]<pnl_vect_get(divid_dates,k)) i++;
    if(fabs(pnl_vect_get(divid_dates,k)-vect_t[i])<1.e-10
  )
      divid_steps[k]=i;
    else
      {
        dist1=vect_t[i]-pnl_vect_get(divid_dates,k);
        dist2=pnl_vect_get(divid_dates,k)-vect_t[i-1];
        if (dist1<dist2)
          divid_steps[k]=i;
        else
          divid_steps[k]=i-1;
      }
  }

/*Compute Minimum and Maximum of the stock at each step
 taking in to account of the dividend payements*/
v_min[0]=s;
v_max[0]=s;
j=0;
for(i=1;i<=n;i++)
  {
    v_min[i]=v_min[i-1]*d;
```

```
    v_max[i]=v_max[i-1]*u;

    for(k=0;k<Nb_div;k++)
      if(i==divid_steps[k])
        {
          v_min[i]=v_min[i]-pnl_vect_get(divid_amounts,Nb
  _div-k-1);
          v_max[i]=v_max[i]-pnl_vect_get(divid_amounts,Nb
  _div-k-1);

        }
  }

/***Singular points at Maturity****/
if((v_min[n]<K)&&(v_max[n]>K))
  {
    nb_critical[n]=2;
    //Abscissa
    vm[0]=v_min[n];
    vm[1]=K;
    vm[2]=v_max[n];

    //Ordinate
    vpm[0]=(p->Compute)(p->Par,vm[0]);
    vpm[1]=(p->Compute)(p->Par,vm[1]);
    vpm[2]=(p->Compute)(p->Par,vm[2]);
  }
else
  {
    nb_critical[n]=1;
    //Abscissa
    vm[0]=v_min[n];
    vm[1]=v_max[n];

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

/***Backward algorithm****/
for(i=1;i<=n;i++)
```

```
{
  /*Min point*/
  new_vm[0]=v_min[n-i];//Abscissa
  value1=new_vm[0]*u;
  value_price1=compute_pricesp(value1,nb_critical[n-i+1
]);
  value2=new_vm[0]*d;
  value_price2=compute_pricesp(value2,nb_critical[n-i+1
]);
  new_vpm[0]=pu*value_price1+pd*value_price2;//Ordinate

  /*Middle points*/
  n_k=1;
  for(j=1;j<nb_critical[n-i+1];j++)
    {
      for(jj=-1;jj<=1;jj=jj+2)
        {
          new_vm[n_k]=vm[j]*pow(u,(double)jj);//Abscissa
          if((new_vm[n_k]>v_min[n-i])&&(new_vm[n_k]<v_
max[n-i]))
            {
              value1=new_vm[n_k]*u;
              value_price1=compute_pricesp(value1,nb_cr
itical[n-i+1]);
              value2=new_vm[n_k]*d;
              value_price2=compute_pricesp(value2,nb_cr
itical[n-i+1]);
              new_vpm[n_k]=pu*value_price1+pd*value_
price2;//Ordinate
              n_k++;
            }
        }
    }

  /*Max point*/
  new_vm[n_k]=v_max[n-i];//Abscissa
  value1=new_vm[n_k]*u;
  value_price1=compute_pricesp(value1,nb_critical[n-i+1
]);
  value2=new_vm[n_k]*d;
  value_price2=compute_pricesp(value2,nb_critical[n-i+1
```

```
]);
  new_vpm[n_k]=pu*value_price1+pd*value_price2;//Ordin
ate


  //Shift at the dividends dates
  for(k=0;k<Nb_div;k++)
    if(i==divid_steps[k])
      {
        for(j=0;j<=n_k;j++)
          new_vm[j]+=pnl_vect_get(divid_amounts,Nb_div-
k-1);
      }

  /*Sorting*/
  nb_critical[n-i]=n_k;
  Sort2Vect(nb_critical[n-i],new_vm,new_vpm);

  //Remove singular points very close TOL1=e-10,TOL2=e-
10
  new_vm1[0]=new_vm[0];
  new_vpm1[0]=new_vpm[0];
  kk=0;
  l=0;
  do {
    do {
      l++;
    }while((new_vm[l]<=new_vm1[kk]+TOL1)&&(l<nb_criti
cal[n-i]));
    kk++;
    new_vm1[kk]=new_vm[l];
    new_vpm1[kk]=new_vpm[l];
  }while((l<nb_critical[n-i]));

  new_nb_critical=kk;
  nb_critical[n-i]=kk;

  if(fabs(new_vm1[nb_critical[n-i]]-new_vm1[nb_criti
cal[n-i]-1])<TOL2)
    nb_critical[n-i]=new_nb_critical-1;
```

```
  //LOWER BOUND
  for(i1=1;i1<=nb_critical[n-i];i1++)
    coeff[i1]=(new_vpm1[i1]-new_vpm1[i1-1])/(new_vm1[i1
]-new_vm1[i1-1]);

  new_vm2[0]=new_vm1[0];
  new_vpm2[0]=new_vpm1[0];
  indice1=1;
  if(nb_critical[n-i]==2)
    {
      new_vm2[1]=new_vm1[1];
      new_vpm2[1]=new_vpm1[1];
      indice1=2;
    }

  if (nb_critical[n-1]>2)
    {
      i1=3;
      do{
        m11=coeff[i1-2];
        m2=coeff[i1-1];
        m3=coeff[i1];
        if (fabs(m11-m3)<TOL3)
          i1=i1+3;
         else
          {
            x=(m11*new_vm1[i1-2]-m3*new_vm1[i1]-new_vp
m1[i1-2]+new_vpm1[i1])/
            (m11-m3);
            yy=m11*(x-new_vm1[i1-2])+new_vpm1[i1-2];
            x11=m2*(x-new_vm1[i1-2])+new_vpm1[i1-2]-yy;

            if (fabs(x11)<h_low)
              {
                new_vm2[indice1]=x;
                new_vpm2[indice1]=yy;
                indice1++;
                i1=i1+2;
              }
            else
              {
```

```
                new_vm2[indice1]=new_vm1[i1-2];
                new_vpm2[indice1]=new_vpm1[i1-2];
                indice1++;
                i1++;
              }
          }
        }
        while((i1<=nb_critical[n-i]));

        if(i1==(nb_critical[n-i])+1)
          {
            new_vm2[indice1]=new_vm1[nb_critical[n-i]-1];
            new_vpm2[indice1]=new_vpm1[nb_critical[n-i]-1
];
            indice1++;

      }
      }

    new_vm2[indice1]=new_vm1[nb_critical[n-i]];
    new_vpm2[indice1]=new_vpm1[nb_critical[n-i]];
    nb_critical[n-i]=indice1;


    //American Call Case
    if((am==1)&&((p->Compute)==&Call))//CALL CASE
      {
        old_nb_critical=nb_critical[n-i];
        if(MAX(0.,new_vm2[0]-K)<=new_vpm2[0])
          {
            l=1;
            while((new_vpm2[l]>=MAX(0.,new_vm2[l]-K))&&(
l<=nb_critical[n-i]))
              {
                l++;
                if(l>nb_critical[n-i])  break;
              }
            if(l<=nb_critical[n-i])
              {
                nb_critical[n-i]=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);
              }
          }
      }


    /*
     //Dynamic Programming:American Put Case
     if((am==1)&&((p->Compute)==&Put))
     for(j=0;j<=nb_critical[n-i];j++)
     {

     new_vpm2[j]=MAX(new_vpm2[j],K-new_vm2[j]);
     }*/

    //American Put Case
    else
      if((am==1)&&((p->Compute)==&Put))
        {
          if(MAX(0.,K-new_vm2[0])>=new_vpm2[0])
            {
              new_vm1[0]=new_vm2[0];
              new_vpm1[0]=MAX(0.,K-new_vm2[0]);
              l=1;
              while((new_vpm2[l]<MAX(0.,K-new_vm2[l]))&&(
l<=nb_critical[n-i]))
                {
                  l++;
                  if(l>nb_critical[n-i])  break;
```

```
                }

              if(l>nb_critical[n-i])
                {
                  new_vm1[1]=new_vm2[nb_critical[n-i]];
                  new_vpm1[1]=MAX(0.,K-new_vm1[1]);
                  nb_critical[n-i]=1;

                }
              else
                if(l<=nb_critical[n-i])
                  {
                    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_vm1[1]=(K-b)/(1.+a);
                    new_vpm1[1]=MAX(0.,K-new_vm1[1]);

                    j=l;
                    nb_critical_put=1;
                    while((j<=nb_critical[n-i]))
                      {
                        nb_critical_put++;
                        new_vm1[nb_critical_put]=new_vm2[
j];
                        new_vpm1[nb_critical_put]=new_vp
m2[j];
                        j++;
                      };
                    nb_critical[n-i]=nb_critical_put;
                  }

              for(j=0;j<=nb_critical[n-i];j++)
                {
                  new_vm2[j]=new_vm1[j];
                  new_vpm2[j]=new_vpm1[j];
```

```
                }

            }
        }

    max_critical=MAX(nb_critical[n-i],max_critical);

    //Copy
    for(j=0;j<=nb_critical[n-i];j++)
      {
        vm[j]=new_vm2[j];
        vpm[j]=new_vpm2[j];
      }
    /*Delta*/
    if(i==(n-1))
      *ptdelta=(vpm[nb_critical[n-i]]-vpm[0])/(vm[nb_cr
  itical[n-i]]-vm[0]);

  }

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

//Memory desallocation
free(nb_critical);
free(divid_steps);
free(v_min);
free(v_max);
free(vpm);
free(vm);
free(new_vm);
free(new_vm1);
free(new_vm2);
free(new_vpm);
free(new_vpm1);
free(new_vpm2);
free(coeff);
free(vect_t);


return OK;
```

```
}

int CALC(TR_SingularPoints_Down)(void *Opt,void *Mod,Prici
    ngMethod *Met)
{
  TYPEOPT* ptOpt=(TYPEOPT*)Opt;
  TYPEMOD* ptMod=(TYPEMOD*)Mod;
  double r;

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

  return SingularPoints_Down(ptOpt->EuOrAm.Val.V_BOOL,pt
    Mod->S0.Val.V_PDOUBLE,ptOpt->PayOff.Val.V_NUMFUNC_1,ptOpt->
    Maturity.Val.V_DATE-ptMod->T.Val.V_DATE,r,ptMod->Sigma.Val.V_
    PDOUBLE,ptMod->Dates.Val.V_PNLVECT,ptMod->Amounts.Val.V_PNLV
    ECT,Met->Par[0].Val.V_INT,Met->Par[1].Val.V_PDOUBLE,&(Met->
    Res[0].Val.V_DOUBLE),&(Met->Res[1].Val.V_DOUBLE));
}
static int CHK_OPT(TR_SingularPoints_Down)(void *Opt, void
    *Mod)
{
  if ( (strcmp( ((Option*)Opt)->Name,"PutAmer")==0) || (
    strcmp( ((Option*)Opt)->Name,"CallAmer")==0) || (strcmp( ((
    Option*)Opt)->Name,"PutEuro")==0) || (strcmp( ((Option*)Opt)->Na
    me,"CallEuro")==0))
    return OK;
  return WRONG;
}

#endif //PremiaCurrentVersion
static int MET(Init)(PricingMethod *Met,Option *Opt)
{
  if ( Met->init == 0)
    {
      Met->init=1;

      Met->Par[0].Val.V_INT2=100;
      Met->Par[1].Val.V_PDOUBLE=0.0001;
    }

  return OK;
```

```
}

PricingMethod MET(TR_SingularPoints_Down)=
{
  "TR_SingularPointsInf",
  {{"StepNumbers between dividends dates",INT2,{100},ALLOW}
    ,{"Tolerance Error",PDOUBLE,{100},ALLOW},{" ",PREMIA_NULLT
    YPE,{0},FORBID}},
  CALC(TR_SingularPoints_Down),
  {{"Price",DOUBLE,{100},FORBID},{"Delta",DOUBLE,{100},FORB
    ID} ,{" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(TR_SingularPoints_Down),
  CHK_tree,
  MET(Init)
};
```

References