Help

```
#include "
href../../../mod/cev1d/cev1d_std/cev1d_std_h_src.pdfcev1d_std.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2019+2) //The "#els
static int CHK_OPT(TREE_Cev)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(TREE_Cev)(void *Opt, void *Mod, PricingMethod *Met)
{
  return AVAILABLE_IN_FULL_PREMIA;
}
#else

static int TreeCev(int am,double x0, NumFunc_1  *p, double T, double r, double q
{
  int i,k,ku,kd;
  double x0s,h,hs,valnod,pu;
  double d;

  d=r-q; //drift of the diffusion

  h=T/(double)N;    //discretization of time

  // memory allocation for triangular matrices
  int *Upj;
  int *Downj;
  double *X;
  double *V;

  Upj    = (int*)malloc((2*N+1)*sizeof(int));
  Downj  = (int*)malloc((2*N+1)*sizeof(int));
  X      = (double*)malloc((2*N+1)*sizeof(double));
  V      = (double*)malloc((2*N+1)*sizeof(double));

  hs=sigma*(1-beta)*sqrt(h); // some parameters
  x0s=pow(x0,1-beta);
```

```
  for(i=0;i<=2*N;i++)
{  /* tree initialization */
  valnod = x0s +(i-N) * hs;

  if(valnod <=0)     /* truncation at zero if we obtain negative values */
{
  X[i]=0 ;
}
  else{
X[i] = pow(valnod,1/(1-beta));
  }
} // End of tree initialization

  for(i=-N+1;i<=N-1;i++)
{  // research of the "good" jumps

  ku=i+1;

  kd=i-1;

  while( (d * X[N+i]) * h + X[N+i] >= X[N + ku])
ku=ku + 2;

  Upj[N+i]=ku;

  while( (d * X[N+i]) * h + X[N+i] <= X[N + kd])
kd=kd - 2;
  Downj[N+i]=kd;
}

  for (i=0;i<=N;i++)    // backward calculus initialization
  V[2*i]=(p->Compute)(p->Par,X[2*i]);

  for(i=N-1;i>=0;i--)
{    // backward calculus

  for(k=-i;k<=i;k=k+2)
{

  if(Upj[N+k]>i+1)
Upj[N+k]=i+1;
```

```
  if(Downj[N+k]<-i-1)
Downj[N+k]=-i-1;

  pu = ((d * X[N+k]) * h + X[N+k] - X[N+Downj[N+k]])/(X[N+Upj[N+k]] - X[N+Downj[

  if (pu<0)
pu=0;

  if (pu>1)
pu=1;

  V[N+k]=exp(-r*h)*(V[N + Upj[N+k]]*pu + V[N + Downj[N+k]]*(1-pu));
  if(am)
V[N+k]=MAX(V[N+k],(p->Compute)(p->Par,X[N+k]));
}
}

  *ptprice= V[N];
  *ptdelta=(V[N+1] - V[N-1])/(X[N+1] - X[N-1]);

  free(Upj);
  free(Downj);
  free(X);
  free(V);

return OK;
}

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

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

  status = TreeCev(ptOpt->EuOrAm.Val.V_BOOL, ptMod->S0.Val.V_PDOUBLE,
   ptOpt->PayOff.Val.V_NUMFUNC_1,
```

```
   ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_DATE,
   r,
   divid,
   ptMod->v.Val.V_PDOUBLE,
   ptMod->beta.Val.V_PDOUBLE,Met->Par[0].Val.V_PINT,
   &(Met->Res[0].Val.V_DOUBLE), &(Met->Res[1].Val.V_DOUBLE));

  return status;
}

static int CHK_OPT(TREE_Cev)(void *Opt, void *Mod)
{
  if ((strcmp(((Option *)Opt)->Name, "CallEuro") == 0) || (strcmp(((Option *)Opt
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;
    }

  return OK;
}

PricingMethod MET(TREE_Cev) =
{
  "TREE_Cev",
  {{"Step Numbers", INT2, {100}, ALLOW},{" ", PREMIA_NULLTYPE, {0}, FORBID}},
  CALC(TREE_Cev),
  { {"Price", DOUBLE, {100}, FORBID}, {"Delta", DOUBLE, {100}, FORBID},
{" ", PREMIA_NULLTYPE, {0}, FORBID}
  },
  CHK_OPT(TREE_Cev),
  CHK_ok,
  MET(Init)
```

```
};
```