Help

```
extern "C" {
#include "
href../../../mod/rstemperedstable1d/rstemperedstable1d_std/rstemperedstable1d_st
#include "pnl/pnl_vector.h"
#include "pnl/pnl_matrix.h"
#include "pnl/pnl_specfun.h"
}

extern "C" {
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2017+2) //The "#els
  static int CHK_OPT(AP_ITKIN)(void *Opt, void *Mod)
  {
    return NONACTIVE;
  }
  int CALC(AP_ITKIN)(void *Opt, void *Mod, PricingMethod *Met)
  {
    return AVAILABLE_IN_FULL_PREMIA;
  }
#else
typedef struct params
{
double Cm, Cp;
double G;
double M;
double Ym, Yp;
double r;
double mu;
double T;
int N;
double K;
} params;

static void Set_params(params *p, double r, double divid, double ap, double am,
{

if (ap == 1)
{
p->Cp = cp*t;
}
```

```
else
{
p->Cp = cp*t*pnl_sf_gamma(-ap);
}

if (am == 1)
{
p->Cm = cm*t;
}
else
{
p->Cm = cm*t*pnl_sf_gamma(-am);
}
p->G = lm;
p->M = lp;
p->Yp = ap;
p->Ym = am;
p->T = t;
p->N = n;
p->K = k;


if (ap == 1)
{
p->mu = r - divid + cp * (lp * log(lp) - (lp-1) * log(lp - 1));
}
else{ p->mu = r - divid + pnl_sf_gamma(-ap) * cp * (exp(ap * log(lp)) - exp(ap *
if (am == 1)
{
p->mu = p->mu + cm * (lm * log(lm) - (lm+1) * log(lm + 1));
}
else{ p->mu = p->mu + pnl_sf_gamma(-am) * cm * (exp(am * log(lm)) - exp(am * log
p->r = r*t*0.5;


}

static void Dtsl(params *p, double h, PnlMat *res)//computation for L_D
{
double al = 0.5*p->mu*p->T / h;
if (al > 0)
```

```
{
for (int i = 0; i < p->N - 1; i++)
{
MLET(res, i, i) = 1 + al + p->r;
MLET(res, i, i + 1) = -al;
}
MLET(res, p->N - 1, p->N - 1) = 1 + al + p->r;
}
else
{
MLET(res, 0, 0) = 1 - al + p->r;
for (int i = 1; i < p->N; i++)
{
MLET(res, i, i) = 1 - al + p->r;
MLET(res, i, i - 1) = al;
}
}
}


static void righttsl(params *p, double h, PnlMat *res)//computation for L_R
{
PnlMat *A, *B, *C;
PnlVect *v;
A = pnl_mat_create_from_zero(p->N, p->N);
B = pnl_mat_create_from_zero(p->N, p->N);
C = pnl_mat_create_from_zero(p->N, p->N);
v = pnl_vect_create(p->N);
double cap = exp(p->Yp * log(p->M));
double al = -1. / (1 + h*p->M);
//double al0 = log(p->M + 1.5 / h);
//double al1 = 4. / (3 + 2 * h*p->M);
//double al2 = -1. / (3 + 2 * h*p->M);


if (p->Yp == 1)
{
res = A;
}
else if (p->Yp > 1)
{
```

```
res = A;
/* for (int i = 0; i < p->N; i++)
{
MLET(C, i, i) = al0;
}
for (int i = 0; i < p->N - 2; i++)
{
MLET(A, i, i + 1) = al1;
MLET(A, i, i + 2) = al2;
}
MLET(A, p->N - 2, p->N - 1) = al1;

pnl_mat_log(B, A);


B = A;
pnl_mat_axpy(-1., B, C);
for (int i = 2; i < p->N; i++)
{
B = pnl_mat_mult_mat(B, A);
pnl_mat_axpy(-1. / double(i), B, C);
}


for (int i = 0; i < p->N; i++)
{
for (int j = 0; j < p->N; j++)
{
MLET(C, i, j) = (p->Yp - 2)*MGET(C, i, j);
}
}
pnl_mat_exp(C, C);

al0 = p->M*p->M - 2 / (h*h);
al1 = p->M / h + 1 / (h*h);
al2 = -p->M / h + 1 / (h*h);
pnl_mat_set_zero(A);
MLET(A, 0, 0) = al0;
MLET(A, 0, 1) = al2;
for (int i = 1; i < p->N - 1; i++)
{
```

```
MLET(A, i, i) = al0;
MLET(A, i, i - 1) = al1;
MLET(A, i, i + 1) = al2;
}
MLET(A, p->N - 1, p->N - 1) = al0;
MLET(A, p->N - 1, p->N - 2) = al1;

C = pnl_mat_mult_mat(A, C);

for (int i = 0; i < p->N; i++)
{
MLET(C, i, i) = MGET(C, i, i) - cap;
for (int j = 0; j < p->N; j++)
{
MLET(res, i, j) = p->Cp*MGET(C, i, j);
}
}*/

}
else
{
MLET(A, 0, 0) = log(p->M + 1. / h);
for (int i = 1; i < p->N; i++)
{
MLET(A, i, i) = log(p->M + 1. / h);
for (int j = 0; j < p->N - i; j++)
{
MLET(A, j, j + i) = PNL_ALTERNATE(i + 1)*pow(al, i) / i;
}
}

for (int i = 0; i < p->N; i++)
{
for (int j = 0; j < p->N; j++)
{
MLET(A, i, j) = p->Yp*MGET(A, i, j);
}
}
pnl_mat_exp(A, A);
for (int i = 0; i < p->N; i++)
{
```

```
MLET(A, i, i) = MGET(A, i, i) - cap;
for (int j = 0; j < p->N; j++)
{
MLET(res, i, j) = p->Cp*MGET(A, i, j);
}
}
}
pnl_mat_free(&A);
pnl_mat_free(&B);
pnl_mat_free(&C);
pnl_vect_free(&v);
}
static void lefttsl(params *p, double h, PnlMat *res)//computation for L_R
{
PnlMat *A;
A = pnl_mat_create_from_zero(p->N, p->N);
double cam = exp(p->Ym * log(p->G));
double al = -1. / (1 + h*p->G);

if (p->Ym == 1)
{
res=A;
}
else if (p->Ym > 1)
{
res = A;
}
else
{
MLET(A, 0, 0) = log(p->G + 1. / h);
for (int i = 1; i < p->N; i++)
{
MLET(A, i, i) = log(p->G + 1. / h);
for (int j = i; j < p->N; j++)
{
MLET(A, j, j - i) = PNL_ALTERNATE(i + 1)*pow(al, i) / i;
}
}
for (int i = 0; i < p->N; i++)
{
for (int j = 0; j < p->N; j++)
```

```
{
MLET(A, i, j) = p->Ym*MGET(A, i, j);
}
}
pnl_mat_exp(A, A);
for (int i = 0; i < p->N; i++)
{
MLET(A, i, i) = MGET(A, i, i) - cam;
for (int j = 0; j < p->N; j++)
{
MLET(res, i, j) = p->Cm*MGET(A, i, j);
}
}

}
pnl_mat_free(&A);
}


static int ITKIN(double S0, NumFunc_1  *payoff, double T, double r, double divid
  {
    double Strike;
    Strike = payoff->Par[0].Val.V_DOUBLE;

params p;
double xmin, dx, dt;
PnlVect *x, *v, *dummy;
PnlMat *JL, *JR, *DD, *A;

dt = T / n; // time step

Set_params(&p, r, divid, alphap, alpham, lambdap, lambdam, cp, cm, dt, Nx, Strik

x = pnl_vect_create(Nx); ///// space points
v = pnl_vect_create(Nx); ///// prices
dummy = pnl_vect_create(Nx); ///// prices

JL = pnl_mat_create_from_zero(Nx, Nx); /////// L_left
JR = pnl_mat_create_from_zero(Nx, Nx); ////// L_right
DD = pnl_mat_create_from_zero(Nx, Nx); ////// L_D
A = pnl_mat_create_from_zero(Nx, Nx); ////// exp(dt L_J)
```

```
//////////////////////////X grid setup and matrix operators computation /////////

if (Nx % 2 == 1){ Nx = Nx + 1; }

xmin = -L;
dx = -2. * xmin / Nx;

for (long int i = 0; i <Nx; i++)
{

LET(x, i) = xmin + i*dx;
}

righttsl(&p, dx, JR); // L_R
lefttsl(&p, dx, JL); // L_L
Dtsl(&p, dx, DD); // L_D
pnl_mat_plus_mat(JR, JL);
pnl_mat_exp(A, JR);   //exp(J)

/////////////////end of the block////////////////////////
/////////////////the 0-time step/////////////////

for (int i = 0; i < Nx; i++)
{
LET(v, i) = MAX(0, p.K - exp(GET(x, i))*S0);
}


//////////////////////end of the 0-time step//////////////////

/////////////// time-stepping cycle//////////////////////////
if (p.mu > 0)
{
for (int j = 0; j < n; j++)
{
LET(v, Nx - 1) = GET(v, Nx - 1) / MGET(DD, Nx - 1, Nx - 1);
for (int i = Nx - 2; i >= 0; i--)
{
LET(v, i) = (GET(v, i) - MGET(DD, i, i + 1)*GET(v, i + 1)) / MGET(DD, i, i);
}
```

```
pnl_mat_mult_vect_inplace(dummy, A, v);
pnl_vect_clone(v, dummy);
LET(v, Nx - 1) = GET(v, Nx - 1) / MGET(DD, Nx - 1, Nx - 1);
for (int i = Nx - 2; i >= 0; i--)
{
LET(v, i) = (GET(v, i) - MGET(DD, i, i + 1)*GET(v, i + 1)) / MGET(DD, i, i);
}
}
}
else
{
for (int j = 0; j < n; j++)
{
LET(v, 0) = GET(v, 0) / MGET(DD, 0, 0);
for (int i = 1; i<Nx; i++)
{
LET(v, i) = (GET(v, i) - MGET(DD, i, i - 1)*GET(v, i - 1)) / MGET(DD, i, i);
}
pnl_mat_mult_vect_inplace(dummy, A, v);
pnl_vect_clone(v, dummy);
LET(v, 0) = GET(v, 0) / MGET(DD, 0, 0);
for (int i = 1; i<Nx; i++)
{
LET(v, i) = (GET(v, i) - MGET(DD, i, i - 1)*GET(v, i - 1)) / MGET(DD, i, i);
}

}
}
//////////////end of the time stepping cycle////////////////////////////
    /*Put Case via parity*/
    if ((payoff->Compute) == &Call)
   {
*ptprice = GET(v, Nx / 2) + (S0*exp(-divid * T) - exp(-r*T)*p.K);
}
else   // Put
{
*ptprice = GET(v, Nx / 2);
}

pnl_mat_free(&A);
```

```
pnl_mat_free(&JR);
pnl_mat_free(&JL);
pnl_mat_free(&DD);
pnl_vect_free(&x);
pnl_vect_free(&v);
pnl_vect_free(&dummy);

    return OK;
  }

  int CALC(AP_ITKIN)(void *Opt, void *Mod, PricingMethod *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  ITKIN(ptMod->S0.Val.V_PDOUBLE, ptOpt->PayOff.Val.V_NUMFUNC_1, ptOpt-
  }

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

    return WRONG;
  }


#endif //PremiaCurrentVersion
  static int MET(Init)(PricingMethod *Met, Option *Opt)
  {
    static int first = 1;

    if (first)
      {
        Met->Par[0].Val.V_PINT = 80 ;
        Met->Par[1].Val.V_PINT= 800;
Met->Par[2].Val.V_PINT= 2;
```

```
        first = 0;
      }

 return OK;
  }

  PricingMethod MET(AP_ITKIN) =
  {
    "AP_ITKIN",
    {{"Number of Time Steps", PINT, {100}, ALLOW    },
  {"Number of Space Points", PINT, {100}, ALLOW    },
      {"Space Scale parameter", PINT, {100}, ALLOW    },
 {" ", PREMIA_NULLTYPE, {0}, FORBID}},
    CALC(AP_ITKIN),
    {{"Price", DOUBLE, {100}, FORBID}, {" ", PREMIA_NULLTYPE, {0}, FORBID}},
    CHK_OPT(AP_ITKIN),
    CHK_ok,
    MET(Init)
  } ;
}
```