Help

```
#include <stdio.h>
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2010+2) //The "#els
#else

#include "
href../../common/math/wienerhopf_rs_h_src.pdfwienerhopf_rs.h"

// model = 1 TSL
// model = 4  KOU

/*
 * A wrapper around pnl_real_ifft_inplace used by the other functions of this
 * file. The order of the storage is changed
 * if i is even then  a(2*i)+ I *  a(2i+1) is the i-th
 * coefficients of the Fourier transform
 * if i is odd then  a(N-2*i)+ I * a(N-2i+1) is the i-th
 * coefficients of the Fourier transform
 */
static void fft_real(PnlVect *a, int fft_size, int sign)
{
  int    i, opposite;
  double last, second;

  switch (sign)
    {
    case 1 : /* backward */
      second = pnl_vect_get(a, 1);
      opposite = 1;
      for (i = 1 ; i < fft_size - 1 ; i++)
        {
          pnl_vect_set(a, i, opposite * pnl_vect_get(a, i + 1));
          opposite = - opposite;
        }
      pnl_vect_set(a , fft_size - 1, second);

      pnl_real_ifft_inplace(a->array, fft_size);
      break;
    case -1 : /* forward */
      pnl_real_fft_inplace(a->array, fft_size);
```

```
      last = pnl_vect_get(a, fft_size - 1);
      opposite = -1;
      for (i = fft_size - 1 ; i > 1 ; i--)
        {
          pnl_vect_set(a, i, opposite * pnl_vect_get(a, i - 1));
          opposite = - opposite;
        }
      pnl_vect_set(a , 1, last);
      break;
    }
}


int readparamskou_rs(int *nstates, PnlVect **rr, PnlVect **divi, PnlVect **sigma
{
  int i, j, flagpos;
  PnlMat *transprob;
  FILE *mypars;
  double psum, el;
  int Nr;

  if ((mypars = fopen(filename, "r")) == NULL)
    {
      printf("Cannot open file %s\ n", filename);
      return 0;
    }
  fscanf(mypars, "%d ", &Nr);

  *rr = pnl_vect_create(Nr + 1);
  *divi = pnl_vect_create(Nr + 1);
  *sigmas = pnl_vect_create(Nr + 1);
  *lambdap = pnl_vect_create(Nr + 1);
  *lambdam = pnl_vect_create(Nr + 1);
  *lambda = pnl_vect_create(Nr + 1);
  *pp = pnl_vect_create(Nr + 1);

  *lam = pnl_mat_create(Nr, Nr);
  transprob = pnl_mat_create(Nr, Nr);

  for (i = 0; i < Nr; i++)
    {
```

```
    fscanf(mypars, "%le %le %le %le %le %le %le ", pnl_vect_lget(*rr, i), pnl_
    LET((*lambdam), i) = - GET((*lambdam), i);
  }

for (i = 0; i < Nr; i++)
  {
    for (j = 0; j < Nr; j++)
      {
        fscanf(mypars, "%le ", pnl_mat_lget(transprob, i, j));
      }
    MLET(transprob, i, i) = 0.0;
  }

fclose(mypars);
flagpos = 1;
for (i = 0; i < Nr; i++)
  {
    psum = 0.0;
    for (j = 0; j < Nr; j++)
      {
        el = MGET(transprob, i, j);
        flagpos = (flagpos && (el >= 0.0));
        psum += el;
      }
    flagpos = (flagpos && (psum < 1.0));
    MLET(transprob, i, i) = 1.0 - psum;
  }

if (flagpos)
  {
    pnl_mat_log(*lam, transprob);
  }
else
  {
    printf("Incorrect transition probabilities!\ n");
    pnl_vect_free(rr);
    pnl_vect_free(divi);
    pnl_vect_free(sigmas);
    pnl_vect_free(lambdap);
    pnl_vect_free(lambdam);
    pnl_vect_free(lambda);
```

```
      pnl_vect_free(pp);
      pnl_mat_free(lam);
    }

  *nstates = Nr;

  pnl_mat_free(&transprob);

  return flagpos;
}

int readparamstsl_rs(int *nstates, PnlVect **rr, PnlVect **divi, PnlVect **nums,
{
  int i, j, flagpos;
  PnlMat *transprob;
  FILE *mypars;
  double psum, el;

  int Nr;

  if ((mypars = fopen(filename, "r")) == NULL)
    {
      printf("Cannot open file %s\ n", filename);
      return 0;
    }
  fscanf(mypars, "%d ", &Nr);

  *rr = pnl_vect_create(Nr + 1);
  *divi = pnl_vect_create(Nr + 1);
  *nums = pnl_vect_create(Nr + 1);
  *nups = pnl_vect_create(Nr + 1);
  *lambdap = pnl_vect_create(Nr + 1);
  *lambdam = pnl_vect_create(Nr + 1);
  *cp = pnl_vect_create(Nr + 1);
  *cm = pnl_vect_create(Nr + 1);
  *lam = pnl_mat_create(Nr + 1, Nr + 1);
  transprob = pnl_mat_create(Nr, Nr);

  for (i = 0; i < Nr; i++)
    {
      fscanf(mypars, "%le %le %le %le %le %le %le %le ", pnl_vect_lget(*rr, i),
```

```
    LET((*lambdam), i) = - GET((*lambdam), i);
  }

for (i = 0; i < Nr; i++)
  {
    for (j = 0; j < Nr; j++)
      {
        fscanf(mypars, "%le ", pnl_mat_lget(transprob, i, j));
      }
    MLET(transprob, i, i) = 0.0;
  }

fclose(mypars);
flagpos = 1;
for (i = 0; i < Nr; i++)
  {
    psum = 0.0;
    for (j = 0; j < Nr; j++)
      {
        el = MGET(transprob, i, j);
        flagpos = (flagpos && (el >= 0.0));
        psum += el;
      }
    flagpos = (flagpos && (psum < 1.0));
    MLET(transprob, i, i) = 1.0 - psum;
  }

if (flagpos)
  {
    pnl_mat_log(*lam, transprob);
  }
else
  {
    printf("Incorrect transition probabilities!\ n");
    pnl_vect_free(rr);
    pnl_vect_free(divi);
    pnl_vect_free(nums);
    pnl_vect_free(nups);
    pnl_vect_free(lambdap);
    pnl_vect_free(lambdam);
    pnl_vect_free(cp);
```

```
      pnl_vect_free(cm);
      pnl_mat_free(lam);
    }

  *nstates = Nr;

  pnl_mat_free(&transprob);

  return flagpos;
}


//---------------------------------------------------------------------
static int findcoefnew(int model, double mu, double sigma, double lm1, double lp
                       double num, double nup,
                       double cnum, double cnup, double q, double r1,
                       double T, double h, long int kmax,
                       double er, long int Nt,
                       PnlVect *al1)

{
  long int Nx;
  PnlVect *bl1;
  PnlVect *alin1;
  double sg2;
  double cpl;
  double cml;

  long int k;
  long int i;
  double mod;

  double xi, xip, xim, anp, anm, sxi, nup1, num1;
  long int Nmax = 2 * kmax;
  double lpm1;


  Nx = Nmax; /*number of space points*/

  /*Memory allocation for space grid*/
```

```
bl1 = pnl_vect_create(Nx + 1);
alin1 = pnl_vect_create(Nx);

////////////////////////// this part depends on the model
if (model == 1/*TSL*/)
  {

    lpm1 = q + pow(lp1, nup) * cnup + pow(-lm1, num) * cnum + fabs(mu) / h;

    nup1 = nup / 2.0;
    num1 = num / 2.0;
    LET(bl1, 0) = q;
    xi = -M_PI / h;
    xip = pow(xi * xi + lp1 * lp1, nup1);
    xim = pow(xi * xi + lm1 * lm1, num1);
    anp = nup * atan(xi / lp1);
    anm = num * atan(xi / lm1);
    LET(bl1, 1) = lpm1 - cnup * xip * cos(anp) - cnum * xim * cos(anm) + fabs(
    LET(bl1, 2 * kmax) = -cnup * xip * sin(anp) - cnum * xim * sin(anm);
    sxi = xi / kmax;
    xi = sxi;

    i = 1;
    do
      {
        xip = pow(xi * xi + lp1 * lp1, nup1);
        xim = pow(xi * xi + lm1 * lm1, num1);
        anp = nup * atan(xi / lp1);
        anm = num * atan(xi / lm1);
        LET(bl1, 2 * i) = lpm1 - cnup * xip * cos(anp) - cnum * xim * cos(anm)
        LET(bl1, 2 * i + 1) = -cnup * xip * sin(anp) - cnum * xim * sin(anm) -
        i++;
        xi = xi + sxi;
      }
    while (i < kmax);
  }// END TSL
if (model == 2/*NIG*/)
  {
    lpm1 = q - sqrt(-lp1 * lm1) * cnup;

    LET(bl1, 0) = q;
```

```
    xi = -M_PI / h;
    xip = xi * xi - lp1 * lm1;
    xim = -(lp1 + lm1) * xi;
    anp = 0.5 * atan(xim / xip);
    LET(bl1, 1) = lpm1 + cnum * pow(xip * xip + xim * xim, 0.25) * cos(anp);
    LET(bl1, 2 * kmax) = cnup * pow(xip * xip + xim * xim, 0.25) * sin(anp);
    sxi = xi / kmax;
    xi = sxi;

    i = 1;
    do
      {
        xip = xi * xi - lp1 * lm1;
        xim = -(lp1 + lm1) * xi;
        anp = 0.5 * atan(xim / xip);
        LET(bl1, 2 * i) = lpm1 + cnum * pow(xip * xip + xim * xim, 0.25) * cos
        LET(bl1, 2 * i + 1) = cnup * pow(xip * xip + xim * xim, 0.25) * sin(an
        i++;
        xi = xi + sxi;
      }
    while (i < kmax);
  }// END NIG
if (model == 3/*VGP*/)
  {
    lpm1 = q - log(-lp1 * lm1) * cnup;

    LET(bl1, 0) = q;
    xi = -M_PI / h;
    xip = xi * xi - lp1 * lm1;
    xim = -(lp1 + lm1) * xi;
    anp = atan(xim / xip);
    LET(bl1, 1) = lpm1 + cnup * log(xip * xip + xim * xim) / 2;
    LET(bl1, 2 * kmax) = cnup * anp - mu * xi;
    sxi = xi / kmax;
    xi = sxi;

    i = 1;
    do
      {
        xip = xi * xi - lp1 * lm1;
        xim = -(lp1 + lm1) * xi;
```

```
        anp = atan(xim / xip);
        LET(bl1, 2 * i) = lpm1 + cnup * log(xip * xip + xim * xim) / 2;
        LET(bl1, 2 * i + 1) = cnup * anp - mu * xi;
        i++;
        xi = xi + sxi;
      }
    while (i < kmax);
  }// END VGP
if (model == 4/*KOU*/)
  {
    sg2 = sigma * sigma / 2;
    cpl = cnup * lp1;
    cml = cnum * lm1;

    LET(bl1, 0) = q;
    xi = -M_PI / h;
    xip = xi * xi + lp1 * lp1;
    xim = xi * xi + lm1 * lm1;
    LET(bl1, 1) = q + (sg2 + cnup / xip + cnum / xim) * xi * xi;
    LET(bl1, 2 * kmax) = xi * (cpl / xip + cml / xim) - mu * xi;
    sxi = xi / kmax;
    xi = sxi;

    i = 1;
    do
      {
        xip = xi * xi + lp1 * lp1;
        xim = xi * xi + lm1 * lm1;
        LET(bl1, 2 * i) = q + (sg2 + cnup / xip + cnum / xim) * xi * xi;
        LET(bl1, 2 * i + 1) = xi * (cpl / xip + cml / xim) - mu * xi;
        i++;
        xi = xi + sxi;
      }
    while (i < kmax);

  }//   END KOU
if (model == 6/*BS or Heston*/)
  {
    // num==var
    double sg2 = sigma / 2;
    if (sg2 > 0.00001)
```

```
  {
    LET(bl1, 0) = q;
    xi = -M_PI / h;
    xip = xi * xi;
    LET(bl1, 1) = q + sg2 * xip;
    LET(bl1, 2 * kmax) = -mu * xi;
    sxi = xi / kmax;
    xi = sxi;

    i = 1;
    do
      {
        xip = xi * xi;
        LET(bl1, 2 * i) = q + sg2 * xip;
        LET(bl1, 2 * i + 1) = -mu * xi;
        i++;
        xi = xi + sxi;
      }
    while (i < kmax);
  }
else
  {
    lpm1 = q + fabs(mu) / h;
    LET(bl1, 0) = q;
    xi = -M_PI / h;
    xip = xi * xi;
    LET(bl1, 1) = lpm1 + sg2 * xip + fabs(mu) / h;
    LET(bl1, 2 * kmax) = 0;
    sxi = xi / kmax;
    xi = sxi;

    i = 1;
    do
      {
        xip = xi * xi;
        LET(bl1, 2 * i) = lpm1 + sg2 * xip - fabs(mu) * cos(xi * h) / h;
        LET(bl1, 2 * i + 1) = -mu * sin(xi * h) / h;
        i++;
        xi = xi + sxi;
      }
    while (i < kmax);
```

```
      }
    } //END Heston
  ////////////////////////////////////////////////////

  LET(alin1, 0) = q / GET(bl1, 0);
  mod = GET(bl1, 2 * kmax) * GET(bl1, 2 * kmax) + GET(bl1, 1) * GET(bl1, 1);
  LET(alin1, 1) = q * GET(bl1, 1) / mod;

  i = 1;
  do
    {
      mod = GET(bl1, 2 * i + 1) * GET(bl1, 2 * i + 1) + GET(bl1, 2 * i) * GET(bl
      LET(alin1, 2 * i + 1) = -q * GET(bl1, 2 * i + 1) / mod;
      LET(alin1, 2 * i) = q * GET(bl1, 2 * i) / mod;
      i++;
    }
  while (i < kmax);


  k = 0;
  while (k < 2 * kmax)
    {
      k++;
      LET(al1, k) = GET(alin1, k - 1);
    }
  pnl_vect_free(&alin1);
  pnl_vect_free(&bl1);
  return 1;
}

//-----------------------------------------findfactor---------------
static int findfactor(double np, double nm, double h, long int kmax,  double lp1
{
  PnlVect *tb1, *tbp1, *tbm1, *ltp1, *ltm1;

  long int i;

  double xi, xip, xim, anp, anm, sxi, abp, abm, nup1, num1;
  double t1, t2;
  double angle;
  double mod;
```

```
long int Nmax = 2 * kmax;

tbp1 = pnl_vect_create(Nmax + 2);
ltp1 = pnl_vect_create(Nmax + 2);
ltm1 = pnl_vect_create(Nmax + 2);
tb1 = pnl_vect_create(Nmax + 2);
tbm1 = pnl_vect_create(Nmax + 2);

nup1 = 0.;
num1 = 0.;

if (np == nm)
  {
    nup1 = np / 2.0;
    num1 = nm / 2.0;
  }
if (np > nm)
  {
    nup1 = np;
    num1 = 0.0;
  }
if (np < nm)
  {
    nup1 = 0.0;
    num1 = nm;
  }

abp = pow(lp1, nup1);
abm = pow(-lm1, num1);

LET(ltp1, 0) = 1;
LET(ltm1, 0) = 1;
xi = -M_PI / h;

xip = pow(xi * xi + lp1 * lp1, nup1 / 2.0) / abp;
xim = pow(xi * xi + lm1 * lm1, num1 / 2.0) / abm;
anp = nup1 * atan(xi / lp1);
anm = num1 * atan(xi / lm1);
LET(ltp1, 1) = xip * cos(anp);
LET(ltm1, 1) = xim * cos(anm);
```

```
LET(ltp1, 2 * kmax) = xip * sin(anp);
LET(ltm1, 2 * kmax) = xim * sin(anm);
sxi = xi / kmax;
xi = sxi;

i = 1;
do
  {
    xip = pow(xi * xi + lp1 * lp1, nup1 / 2.0) / abp;
    xim = pow(xi * xi + lm1 * lm1, num1 / 2.0) / abm;
    anp = nup1 * atan(xi / lp1);
    anm = num1 * atan(xi / lm1);
    LET(ltp1, 2 * i) = xip * cos(anp);
    LET(ltp1, 2 * i + 1) = xip * sin(anp);
    LET(ltm1, 2 * i) = xim * cos(anm);
    LET(ltm1, 2 * i + 1) = xim * sin(anm);
    i++;
    xi = xi + sxi;
  }
while (i < kmax);

LET(alin1, 1) = GET(alin1, 1) * (GET(ltp1, 1) * GET(ltm1, 1) - GET(ltp1, 2 * k
for (i = 1; i <= kmax - 1; i++)
  {
    t1 = GET(alin1, 2 * i);
    t2 = GET(alin1, 2 * i + 1);
    LET(alin1, 2 * i) = t1 * GET(ltm1, 2 * i) - t2 * GET(ltm1, 2 * i + 1);
    LET(alin1, 2 * i + 1) = t2 * GET(ltm1, 2 * i) + t1 * GET(ltm1, 2 * i + 1);
  }
for (i = 1; i <= kmax - 1; i++)
  {
    t1 = GET(alin1, 2 * i);
    t2 = GET(alin1, 2 * i + 1);
    LET(alin1, 2 * i) = t1 * GET(ltp1, 2 * i) - t2 * GET(ltp1, 2 * i + 1);
    LET(alin1, 2 * i + 1) = t2 * GET(ltp1, 2 * i) + t1 * GET(ltp1, 2 * i + 1);
  }

LET(ltp1, 0) = 1 / GET(ltp1, 0);
LET(ltm1, 0) = 1 / GET(ltm1, 0);
LET(ltp1, 1) = GET(ltp1, 1) / (GET(ltp1, 1) * GET(ltp1, 1) + GET(ltp1, 2 * kma
LET(ltm1, 1) = GET(ltm1, 1) / (GET(ltm1, 1) * GET(ltm1, 1) + GET(ltm1, 2 * kma
```

```
i = 1;
do
  {
    mod = GET(ltp1, 2 * i + 1) * GET(ltp1, 2 * i + 1) + GET(ltp1, 2 * i) * GET
    LET(ltp1, 2 * i + 1) = -GET(ltp1, 2 * i + 1) / mod;
    LET(ltp1, 2 * i) = GET(ltp1, 2 * i) / mod;

    mod = GET(ltm1, 2 * i + 1) * GET(ltm1, 2 * i + 1) + GET(ltm1, 2 * i) * GET
    LET(ltm1, 2 * i + 1) = -GET(ltm1, 2 * i + 1) / mod;
    LET(ltm1, 2 * i) = GET(ltm1, 2 * i) / mod;
    i++;
  }
while (i < kmax);

LET(tb1, 0) = log(GET(alin1, 0));
LET(tb1, 1) = log(GET(alin1, 1));

i = 1;
do
  {
    mod = GET(alin1, 2 * i + 1) * GET(alin1, 2 * i + 1) + GET(alin1, 2 * i) *
    LET(tb1, 2 * i) = 0.5 * log(mod);
    if (GET(alin1, 2 * i) == 0)
      {
        if (GET(alin1, 2 * i + 1) > 0)
          {
            LET(tb1, 2 * i + 1) = M_PI / 2.0;
          }
        else
          {
            LET(tb1, 2 * i + 1) = -M_PI / 2.0;
          }
      }
    else
      {
        angle = atan(GET(alin1, 2 * i + 1) / GET(alin1, 2 * i));
        if (GET(alin1, 2 * i) > 0)
          {
            LET(tb1, 2 * i + 1) = angle;
          }
```

```
        if (GET(alin1, 2 * i) < 0)
          {
            if (GET(alin1, 2 * i + 1) < 0)
              {
                LET(tb1, 2 * i + 1) = angle - M_PI;
              }
            else
              {
                LET(tb1, 2 * i + 1) = angle + M_PI;
              }
          }
      }
    i++;
  }
while (i < kmax);


fft_real(tb1, 2 * kmax, 1);

i = 1;
LET(tbp1, 0) = 0;
LET(tbm1, 0) = 0;
do
  {
    LET(tbp1, 0) = GET(tbp1, 0) - GET(tb1, i);
    LET(tbp1, i) = GET(tb1, i);
    LET(tbm1, i) = 0;
    i++;
  }
while (i < kmax);
do
  {
    LET(tbm1, i) = GET(tb1, i);
    LET(tbp1, i) = 0;
    LET(tbm1, 0) = GET(tbm1, 0) - GET(tb1, i);
    i++;
  }
while (i < 2 * kmax);

fft_real(tbp1, 2 * kmax, -1);
fft_real(tbm1, 2 * kmax, -1);
```

```
LET(tp1, 0) = exp(GET(tbp1, 0));
LET(tp1, 1) = exp(GET(tbp1, 1));
LET(tm1, 0) = exp(GET(tbm1, 0));
LET(tm1, 1) = exp(GET(tbm1, 1));

i = 1;
do
  {
    mod = exp(GET(tbp1, 2 * i));
    LET(tp1, 2 * i) = mod * cos(GET(tbp1, 2 * i + 1));
    LET(tp1, 2 * i + 1) = mod * sin(GET(tbp1, 2 * i + 1));
    mod = exp(GET(tbm1, 2 * i));
    LET(tm1, 2 * i) = mod * cos(GET(tbm1, 2 * i + 1));
    LET(tm1, 2 * i + 1) = mod * sin(GET(tbm1, 2 * i + 1));
    i++;
  }
while (i < kmax);

LET(tp1, 0) = GET(tp1, 0) * GET(ltp1, 0);
LET(tp1, 1) = GET(tp1, 1) * GET(ltp1, 1);

LET(tm1, 0) = GET(tm1, 0) * GET(ltm1, 0);
LET(tm1, 1) = GET(tm1, 1) * GET(ltm1, 1);

for (i = 1; i <= kmax - 1; i++)
  {
    t1 = GET(tm1, 2 * i);
    t2 = GET(tm1, 2 * i + 1);
    LET(tm1, 2 * i) = t1 * GET(ltm1, 2 * i) - t2 * GET(ltm1, 2 * i + 1);
    LET(tm1, 2 * i + 1) = t2 * GET(ltm1, 2 * i) + t1 * GET(ltm1, 2 * i + 1);
  }
for (i = 1; i <= kmax - 1; i++)
  {
    t1 = GET(tp1, 2 * i);
    t2 = GET(tp1, 2 * i + 1);
    LET(tp1, 2 * i) = t1 * GET(ltp1, 2 * i) - t2 * GET(ltp1, 2 * i + 1);
    LET(tp1, 2 * i + 1) = t2 * GET(ltp1, 2 * i) + t1 * GET(ltp1, 2 * i + 1);
  }

pnl_vect_free(&tbp1);
```

```
  pnl_vect_free(&ltp1);
  pnl_vect_free(&ltm1);
  pnl_vect_free(&tb1);
  pnl_vect_free(&tbm1);

  return 1;
}//end findfactor

//------------------------------------------------------ REGIME SWITCHING ----

int fastwienerhopf_rs(int model, long int Nr, PnlVect *mu, PnlVect *qu, double o
                      int ifCall, double Spot, PnlVect *lm1, PnlVect *lp1,
                      PnlVect *num, PnlVect *nup, PnlVect *cnum, PnlVect *cnup,
                      PnlVect *r, PnlVect *divid, PnlMat *lam,
                      double T, double h, PnlVect *Strike1,
                      double bar, PnlVect *rebate,
                      double er, long int step, double eps,
                      PnlVect *ptprice, PnlVect *ptdelta)
{

  PnlVect *y, *expy, *vv1;
  PnlVect *alin1, *eom, *al1, *tp1, *tm1;
  PnlVect *coef;

  long int N1 = 0, kmax, i, j, L;

  PnlVect *Lambda1, *gam, *g1, *lamka, *iter, *a2_1;
  PnlMat  *prices_old, *prices_new, *tp, *tm;
  PnlMat  *prices_all, *switcharrow, *G, *lam1;

  double pp, itererr;

  double t1, t2;
  double dif;

  double qq;
  long int k;
  long int Nmax, NO1;
  int nn, n;
  double dt;
```

```
double mu1, kx;
double Sl, Sm, Sr, pricel, pricem, pricer;
double A, B, C;
double np, nm;

if (upordown == 0)
  {
    if (Spot <= bar)
      {
        for (k = 0; k < Nr; k++)
          {
            LET(ptprice, k) = GET(rebate, k);
            LET(ptdelta, k) = 0.;
          }
        return OK;
      }
    if (Spot >= 0.8 * bar * exp(er * log(2.)))
      {
        for (k = 0; k < Nr; k++)
          {
            LET(ptprice, k) = 0.;
            LET(ptdelta, k) = 0.;
          }
        printf("Spot is out of range. Increase scale parameter \ n");
        return OK;
      }
  }
else
  {
    if (Spot >= bar)
      {
        for (k = 0; k < Nr; k++)
          {
            LET(ptprice, k) = GET(rebate, k);
            LET(ptdelta, k) = 0.;
          }
        return OK;
      }
    if (Spot <= bar * exp(-er * log(2.)) * 1.2)
      {
        for (k = 0; k < Nr; k++)
```

```
          {
            LET(ptprice, k) = 0.;
            LET(ptdelta, k) = 0.;
          }
        printf("Spot is out of range. Increase scale parameter \ n");
        return OK;
      }
  }

 k = 64;

 kx = ceil(er * log(2.0) / h);
 while (k < kx)
   {
     k = k * 2;
   }
 kmax = k;

 N1 = step;
 Nmax = 2 * kmax;
 NO1 = kmax - 1;
 nn = 3;
//                              Memory allocation
 Lambda1 = pnl_vect_create(Nr + 1);
 gam = pnl_vect_create(Nr + 1);
//
 y = pnl_vect_create(Nmax + 1); //log space grid points
 expy = pnl_vect_create(Nmax + 1); //space grid points
 vv1 = pnl_vect_create(Nmax + 2);
 alin1 = pnl_vect_create(Nmax + 2);
 tp1 = pnl_vect_create(Nmax + 2);
 tm1 = pnl_vect_create(Nmax + 2);
 al1 = pnl_vect_create(Nmax + 2);
 eom = pnl_vect_create(Nmax + 1);
 prices_old = pnl_mat_create(Nr + 1, Nmax + 2);
 prices_new = pnl_mat_create(Nr + 1, Nmax + 2);
 prices_all = pnl_mat_create(Nr + 1, Nmax + 2);
 g1 = pnl_vect_create(Nmax + 2);
 G = pnl_mat_create(Nr + 1, Nmax + 2);
 lam1 = pnl_mat_create(Nr + 1, Nr + 1);
```

```
  tm = pnl_mat_create(Nr + 1, Nmax + 2);
  tp = pnl_mat_create(Nr + 1, Nmax + 2);
  a2_1 = pnl_vect_create(Nr + 1);
  iter = pnl_vect_create(Nr + 1);
  lamka = pnl_vect_create(Nr + 1);

  coef = pnl_vect_create(nn + 1);

  for (i = 0; i < Nr; i++)
    {
      LET(Lambda1, i) = 0;
      MLET(lam, i, i) = 0.0;
      for (k = 0; k < Nr; k++)
        {
          LET(Lambda1, i) = GET(Lambda1, i) + MGET(lam, i, k);
        }
      LET(lp1, i) = GET(lp1, i) + om;
      LET(lm1, i) = GET(lm1, i) + om;
      LET(ptprice, i) = 0.;
      LET(ptdelta, i) = 0.;
    }

  np = 0.0;
  nm = 0.0;

//============================ compute coefficients
  if (model == 1) /*TSL*/
    {
      mu1 = 0.;
      for (i = 0; i < Nr; i++)
        {
          LET(qu, i) = GET(qu, i) + GET(Lambda1, i) - GET(mu, i) * om;
        }
    }

  if (model == 4) /*KOU*/
    {
      for (i = 0; i < Nr; i++)
        {
          LET(mu, i) = GET(mu, i) + GET(nup, i) * GET(nup, i) * om; //sigma*sigm
          LET(cnum, i) = GET(cnum, i) * (GET(lm1, i) - om) / GET(lm1, i);
```

```
          LET(cnup, i) = GET(cnup, i) * (GET(lp1, i) - om) / GET(lp1, i);
          LET(qu, i) = GET(qu, i) + GET(Lambda1, i);
        }
      mu1 = 0.0;
      np = 2.0;
      nm = 2.0;
    }

//-----------------------------------------------weghts-------------------------
  LET(coef, 1) = 0.5;
  LET(coef, 2) = -4;
  LET(coef, 3) = 4.5;

//-----------------------------------------------------------------loop in weighted
  for (n = 1; n <= nn; n++)
    {
      dt = T / (n * N1 + 1);
      for (k = 0; k < Nr; k++)
        {
          qq = GET(qu, k) + 1.0 / dt;
          findcoefnew(model, GET(mu, k) - mu1, GET(nup, k), GET(lm1, k), GET(lp1

          LET(a2_1, k) = qq * dt;

          for (i = 0; i < 2 * kmax; i++) LET(alin1, i) = GET(al1, i + 1);
          findfactor(np, nm, h, kmax, GET(lp1, k), GET(lm1, k), alin1, tp1, tm1)
          i = 0;
          do
            {
              MLET(tm, k, i) = GET(tm1, i);
              MLET(tp, k, i) = GET(tp1, i);
              i++;
            }
          while (i < 2 * kmax);
        }

      for (k = 0; k < Nr; k++)
        {
          LET(lamka, k) = 0.0;
          for (j = 0; j < Nr; j++)
            {
```

```
        LET(lamka, k) = GET(lamka, k) + MGET(lam, k, j) * GET(rebate, j) *
        MLET(lam1, k, j) = MGET(lam, k, j) * dt;
      }
    LET(lamka, k) = GET(lamka, k) - dt * (GET(r, k) + GET(Lambda1, k)) * G
    LET(lamka, k) = GET(lamka, k) / GET(a2_1, k);
  }


/*Put Pay-off functions*/
if (upordown == 0) //DOWN
  for (j = 0; j < kmax; j++)
    {
      LET(y, j) = (j - N01 - 0.5) * h;
      LET(eom, j) = exp(-om * GET(y, j));
      LET(expy, j) = bar * exp(GET(y, j));
      for (k = 0; k < Nr; k++)
        {
          MLET(prices_all, k, j) = 0;
        }
    }
else // UP
  for (j = kmax - 1; j < Nmax; j++)
    {
      LET(y, j) = (j - N01 + 0.5) * h;
      LET(eom, j) = exp(-om * GET(y, j));
      LET(expy, j) = bar * exp(GET(y, j));
      for (k = 0; k < Nr; k++)
        {
          MLET(prices_all, k, j) = 0;
        }
    }

if (upordown == 0) // IF DOWN
  {
    j = kmax;

    if (ifCall == 2) /* DIGITAL */
      {
        do
          {
            LET(y, j) = (j - N01 - 0.5) * h;
```

```
        LET(eom, j) = exp(-om * GET(y, j));
        LET(expy, j) = bar * exp(GET(y, j));
        for (k = 0; k < Nr; k++)
          {
            MLET(prices_all, k, j) = -GET(rebate, k) * GET(eom, j);
          }
        j++;
      }
    while (j < Nmax);
  }
else if (ifCall == 1) /*  CALL */
  {
    do
      {
        LET(y, j) = (j - N01 - 0.5) * h;
        LET(eom, j) = exp(-om * GET(y, j));
        LET(expy, j) = bar * exp(GET(y, j));
        for (k = 0; k < Nr; k++)
          {
            if (GET(expy, j) >= GET(Strike1, k))
              {
                MLET(prices_all, k, j) = (-GET(Strike1, k) + GET(expy,
              }
            else
              {
                MLET(prices_all, k, j) = -GET(rebate, k) * GET(eom, j)
              }
          }
        j++;
      }
    while (j < Nmax);
  }
else              /* PUT */
  {
    do
      {
        LET(y, j) = (j - N01 - 0.5) * h;
        LET(eom, j) = exp(-om * GET(y, j));
        LET(expy, j) = bar * exp(GET(y, j));
        for (k = 0; k < Nr; k++)
          {
```

```
                if (GET(expy, j) <= GET(Strike1, k))
                  {
                    MLET(prices_all, k, j) = (GET(Strike1, k) - GET(expy,
                  }
                else
                  {
                    MLET(prices_all, k, j) = -GET(rebate, k) * GET(eom, j)
                  }
              }
            j++;
          }
        while (j < Nmax);
      }
  }// END IF DOWN-OUT
else // IF UP-OUT
  {
    j = kmax - 2;

    if (ifCall == 2)   /* DIGITAL */
      {
        do
          {
            LET(y, j) = (j - NO1 + 0.5) * h;
            LET(eom, j) = exp(-om * GET(y, j));
            LET(expy, j) = bar * exp(GET(y, j));
            for (k = 0; k < Nr; k++)
              {
                MLET(prices_all, k, j) = -GET(rebate, k) * GET(eom, j);
              }
            j--;
          }
        while (j >= 0);
      }
    else if (ifCall == 1) /* CALL */
      {
        do
          {
            LET(y, j) = (j - NO1 + 0.5) * h;
            LET(eom, j) = exp(-om * GET(y, j));
            LET(expy, j) = bar * exp(GET(y, j));
            for (k = 0; k < Nr; k++)
```

```
            {
              if (GET(expy, j) >= GET(Strike1, k))
                {
                  MLET(prices_all, k, j) = (-GET(Strike1, k) + GET(expy,
                }
              else
                {
                  MLET(prices_all, k, j) = -GET(rebate, k) * GET(eom, j)
                }
            }
          j--;
        }
      while (j >= 0);
    }
  else                  /* PUT */
    {
      do
        {
          LET(y, j) = (j - N01 + 0.5) * h;
          LET(eom, j) = exp(-om * GET(y, j));
          LET(expy, j) = bar * exp(GET(y, j));
          for (k = 0; k < Nr; k++)
            {
              if (GET(expy, j) <= GET(Strike1, k))
                {
                  MLET(prices_all, k, j) = (GET(Strike1, k) - GET(expy,
                }
              else
                {
                  MLET(prices_all, k, j) = -GET(rebate, k) * GET(eom, j)
                }
            }
          j--;
        }
      while (j >= 0);
    }
 }// END IF UP


//======= EXCHANGE tp AND tm ARRAYS FOR UP-OUT
if (upordown == 1)
```

```
    {
      switcharrow = tp;
      tp = tm;
      tm = switcharrow;
    }
  //===========================================
  for (k = 0; k < Nr; k++)
    {
      for (i = 0; i < 2 * kmax; i++) MLET(tm, k, i) = MGET(tm, k, i) / GET(a
    }

  for (k = 0; k < Nr; k++)
    {
      for (i = 0; i < Nmax; i++)
        {
          MLET(prices_old, k, i) = MGET(prices_all, k, i);
        }
      for (i = 0; i < Nmax; i++)
        {
          MLET(G, k, i) = GET(lamka, k) * GET(eom, i);
        }
    }

//=======================*L-Loop*=======================/

  for (L = 1; L <= N1 * n + 1; L++)
    {

      itererr = 1;
      while (itererr > eps)
        {
          for (k = 0; k < Nr; k++)
            {
              for (i = 0; i < Nmax; i++)
                {
                  LET(vv1, i) = MGET(prices_all, k, i);

                  for (j = 0; j < Nr; j++)
                    {
                      LET(vv1, i) = GET(vv1, i) + MGET(lam1, k, j) * MGET(pr
                    }
```

```
  }

fft_real(vv1, 2 * kmax, -1);

LET(vv1, 0) = GET(vv1, 0) * MGET(tm, k, 0);
LET(vv1, 1) = GET(vv1, 1) * MGET(tm, k, 1);

for (i = 1; i <= kmax - 1; i++)
  {
    t1 = GET(vv1, 2 * i);
    t2 = GET(vv1, 2 * i + 1);
    LET(vv1, 2 * i) = t1 * MGET(tm, k, 2 * i) - t2 * MGET(tm,
    LET(vv1, 2 * i + 1) = t2 * MGET(tm, k, 2 * i) + t1 * MGET(
  }

fft_real(vv1, 2 * kmax, 1);

if (upordown == 0) // IF DOWN-OUT!!
  {
    i = 0;
    do
      {
        LET(vv1, i) = 0;
        i++;
      }
    while (GET(y, i) <= 0.);
    do
      {
        LET(vv1, i) = GET(vv1, i) + MGET(G, k, i);
        i++;
      }
    while (i < Nmax);
  }
else  //IF UP-OUT
  {
    i = Nmax - 1; //-1;
    do
      {
        LET(vv1, i) = 0;
        i--;
      }
```

```
        while (GET(y, i) >= 0.);
        do
          {
            LET(vv1, i) = GET(vv1, i) + MGET(G, k, i);
            i--;
          }
        while (i >= 0);
      }

    fft_real(vv1, 2 * kmax, -1);

    LET(vv1, 0) = GET(vv1, 0) * MGET(tp, k, 0);
    LET(vv1, 1) = GET(vv1, 1) * MGET(tp, k, 1);

    for (i = 1; i <= kmax - 1; i++)
      {
        t1 = GET(vv1, 2 * i);
        t2 = GET(vv1, 2 * i + 1);
        LET(vv1, 2 * i) = t1 * MGET(tp, k, 2 * i) - t2 * MGET(tp,
        LET(vv1, 2 * i + 1) = t2 * MGET(tp, k, 2 * i) + t1 * MGET(
      }

    fft_real(vv1, 2 * kmax, 1);

    LET(iter, k) = 0;
    for (i = 0; i < Nmax; i++)
      {
        dif = fabs(GET(vv1, i) - MGET(prices_old, k, i));
        LET(iter, k) = GET(iter, k) > dif ? GET(iter, k) : dif;
      }

    LET(iter, k) = GET(iter, k) / GET(Strike1, k);

    for (i = 0; i < Nmax; i++)
      {
        MLET(prices_new, k, i) = GET(vv1, i);
      }
  }// end loop in k

itererr = 0;
for (k = 0; k < Nr; k++)
```

```
          {
            itererr = itererr > GET(iter, k) ? itererr : GET(iter, k);
          }

        for (k = 0; k < Nr; k++)
          {
            for (i = 0; i < Nmax; i++)
              {
                MLET(prices_old, k, i) = MGET(prices_new, k, i);
              }
          }

      }//end while

    for (k = 0; k < Nr; k++)
      {
        for (i = 0; i < Nmax; i++)
          {
            MLET(prices_all, k, i) = MGET(prices_new, k, i);
          }
      }


  }// ======================================== PREMIA_NULLTYPE L-LOOP ====

for (k = 0; k < Nr; k++)
  {
    for (i = 0; i < Nmax; i++)
      {
        MLET(prices_all, k, i) = MGET(prices_new, k, i) / GET(eom, i) + GE
      }
  }

pp = log(Spot / bar);

i = 2;

while ((GET(y, i) <= pp) && (i < Nmax))
  {
    i++;
  }
```

```
    i--;
    for (k = 0; k < Nr; k++)
      {
        //Price, quadratic interpolation
        Sl = GET(expy, i - 1);
        Sm = GET(expy, i);
        Sr = GET(expy, i + 1);
        // S0 is between Sm and Sr
        pricel = MGET(prices_all, k, i - 1);
        pricem = MGET(prices_all, k, i);
        pricer = MGET(prices_all, k, i + 1);

        //quadratic interpolation
        A = pricel;
        B = (pricem - pricel) / (Sm - Sl);
        C = (pricer - A - B * (Sr - Sl)) / (Sr - Sl) / (Sr - Sm);

        //Price
        pricem = A + B * (Spot - Sl) + C * (Spot - Sl) * (Spot - Sm);

        //Delta
        pricel = B + C * (2 * Spot - Sl - Sm);
        LET(ptprice, k) = GET(ptprice, k) + LET(coef, n) * pricem;
        LET(ptdelta, k) = GET(ptdelta, k) + LET(coef, n) * pricel;
      }// end for(k)
  }// end for(n)

for (k = 0; k < Nr; k++)
  {
    if (GET(ptprice, k) < 0)
      {
        LET(ptprice, k) = 0.0;
        LET(ptdelta, k) = 0.0;
      }
  }

/*Memory desallocation*/
pnl_vect_free(&Lambda1);
pnl_vect_free(&gam);
pnl_vect_free(&y);
```

```
  pnl_vect_free(&expy);
  pnl_vect_free(&vv1);
  pnl_vect_free(&alin1);
  pnl_vect_free(&tp1);
  pnl_vect_free(&tm1);
  pnl_vect_free(&al1);
  pnl_vect_free(&eom);

  pnl_mat_free(&prices_old);

  pnl_mat_free(&prices_new);

  pnl_mat_free(&prices_all);
  pnl_vect_free(&g1);

  pnl_mat_free(&G);
  pnl_mat_free(&lam1);
  pnl_mat_free(&tm);

  pnl_mat_free(&tp);
  pnl_vect_free(&a2_1);
  pnl_vect_free(&iter);
  pnl_vect_free(&lamka);
  pnl_vect_free(&coef);

  return OK;
}
//-------------------------------------------------------------------------------
void cuteom(PnlVect *eomcut, PnlVect *eom, long int Nmax, long int NO1, double f
{
  long int i;
  if (flagCall > 0)
    {
      for (i = NO1 + 1; i < Nmax; i++)
        {
          LET(eomcut, i) = 0.;
        }
      for (i = 0; i < NO1 + 1; i++)
        {
          LET(eomcut, i) = GET(eom, i);
        }
```

```
    }
  else
    {
      for (i = 0; i < NO1; i++)
        {
          LET(eomcut, i) = 0.;
        }
      for (i = NO1; i < Nmax; i++)
        {
          LET(eomcut, i) = GET(eom, i);
        }
    }
}

// ------------------------------------------ AMERICAN RS --------------------

int fastwienerhopfamerican_rs(int model, long int Nr, PnlVect *mu, PnlVect *qu,
                              int ifCall, double Spot, PnlVect *lm1, PnlVect *lp
                              PnlVect *num, PnlVect *nup, PnlVect *cnum, PnlVect
                              PnlVect *r, PnlVect *divid, PnlMat *lam,
                              double T, double h, PnlVect *Strike1,
                              double er, long int step, double eps,
                              PnlVect *ptprice, PnlVect *ptdelta)
{
  PnlVect *y, *expy, *vv1, *tp1, *tm1, *alin1, *eom, *eomcut, *al1;

  long int N1 = 0, kmax, i, j, L;

  PnlVect *Lambda1, *gam,  *g1, *lamka, *iter, *a2_1;
  PnlMat *tp, *tm, *prices_old, *prices_new, *prices_all, *switcharrow, *G;

  double pp, itererr;
  double t1, t2;
  double dif;

  double Kmax, Kmin;
  long int k;
  long int Nmax, NO1;
  double dt;

  double mu1, kx;
```

```
double Sl, Sm, Sr, pricel, pricem, pricer;
double A, B, C;
double np, nm;
double flagCall;//=1 for call and =-1 for put

double K = 1.0;

for (k = 0; k < Nr; k++) K = K * GET(Strike1, k);

K = pow(K, 1.0 / Nr);

if (ifCall == 0) //IF PUT
  {
    if (Spot >= 0.8 * K * exp(er * log(2.)))
      {
        for (k = 0; k < Nr; k++)
          {
            LET(ptprice, k) = 0.;
            LET(ptdelta, k) = 0.;
          }
        printf("Spot is out of range. Increase scale parameter \ n");
        return OK;
      }
    if (Spot <= 1.2 * K * exp(-er * log(2.)))
      {
        for (k = 0; k < Nr; k++)
          {
            LET(ptprice, k) = GET(Strike1, k) - Spot;
            LET(ptdelta, k) = -1.0;
          }
        printf("Spot is out of range. Increase scale parameter \ n");
        return OK;
      }
  }
else
  {
    if (Spot >= 0.8 * K * exp(er * log(2.)))
      {
        for (k = 0; k < Nr; k++)
          {
            LET(ptprice, k) = - GET(Strike1, k) + Spot;
```

```
            LET(ptdelta, k) = 1.;
          }
        printf("Spot is out of range. Increase scale parameter \ n");
        return OK;
      }
    if (Spot <= K * exp(-er * log(2.)) * 1.2)
      {
        for (k = 0; k < Nr; k++)
          {
            LET(ptprice, k) = 0.0;
            LET(ptdelta, k) = 0.0;
          }
        printf("Spot is out of range. Increase scale parameter \ n");
        return OK;
      }
  }

Kmax = K;
Kmin = K;

for (k = 0; k < Nr; k++)
  {
    if (Kmax < GET(Strike1, k))
      {
        Kmax = GET(Strike1, k);
      }
    if (Kmin > GET(Strike1, k))
      {
        Kmin = GET(Strike1, k);
      }
  }

k = 64;

kx = ceil(er * log((2.5 * Kmax - 0.4 * Kmin) / K) / h);
while (k < kx)
  {
    k = k * 2;
  }
kmax = k;
```

```
  N1 = step;
  Nmax = 2 * kmax;
  N01 = kmax - 1;
//                              Memory allocation
  Lambda1 = pnl_vect_create(Nr + 1);
  gam = pnl_vect_create(Nr + 1);

  y = pnl_vect_create(Nmax + 1);
  expy = pnl_vect_create(Nmax + 1);
  vv1 = pnl_vect_create(Nmax + 2);
  tp1 = pnl_vect_create(Nmax + 2);
  alin1 = pnl_vect_create(Nmax + 2);
  tm1 = pnl_vect_create(Nmax + 2);
  al1 = pnl_vect_create(Nmax + 2);
  eom = pnl_vect_create(Nmax + 1);
  eomcut = pnl_vect_create(Nmax + 1);

  prices_old = pnl_mat_create(Nr + 1, Nmax + 2);
  prices_new = pnl_mat_create(Nr + 1, Nmax + 2);
  prices_all = pnl_mat_create(Nr + 1, Nmax + 2);
  g1 = pnl_vect_create(Nmax + 2);
  G = pnl_mat_create(Nr + 1, Nmax + 2);

  tm = pnl_mat_create(Nr + 1, Nmax + 2);
  tp = pnl_mat_create(Nr + 1, Nmax + 2);
  a2_1 = pnl_vect_create(Nr + 1);
  iter = pnl_vect_create(Nr + 1);
  lamka = pnl_vect_create(Nr + 1);

  /*Time step*/
  dt = T / N1;

  for (i = 0; i < Nr; i++)
    {
      LET(Lambda1, i) = 0;
      MLET(lam, i, i) = 0.0;
      for (k = 0; k < Nr; k++)
        {
          LET(Lambda1, i) = GET(Lambda1, i) + MGET(lam, i, k);
        }
```

```
      LET(lp1, i) = GET(lp1, i) + om;
      LET(lm1, i) = GET(lm1, i) + om;
    }
//============================ compute coefficients
  np = 0.0;
  nm = 0.0;
  if (model == 1) /*TSL*/
    {
      mu1 = 0;
      for (i = 0; i < Nr; i++)
        {
          LET(qu, i) = GET(qu, i) + 1.0 / dt + GET(Lambda1, i) - GET(mu, i) * om
        }
    }
  if (model == 4) /*KOU*/
    {
      for (i = 0; i < Nr; i++)
        {
          mu1 = 0.0;
          LET(mu, i) = GET(mu, i) + GET(nup, i) * GET(nup, i) * om; //sigma*sigm
          LET(cnum, i) = GET(cnum, i) * (GET(lm1, i) - om) / GET(lm1, i);
          LET(cnup, i) = GET(cnup, i) * (GET(lp1, i) - om) / GET(lp1, i);
          LET(qu, i) = GET(qu, i) + 1.0 / dt + GET(Lambda1, i);
        }
//np=2.0;
//nm=2.0;
    }

  for (k = 0; k < Nr; k++)
    {
      findcoefnew(model, GET(mu, k) - mu1, GET(nup, k), GET(lm1, k), GET(lp1, k)

      LET(a2_1, k) = GET(qu, k) * dt;

      for (i = 0; i < 2 * kmax; i++) LET(alin1, i) = GET(al1, i + 1);
      findfactor(np, nm, h, kmax, GET(lp1, k), GET(lm1, k), alin1, tp1, tm1);

      i = 0;
      do
        {
          MLET(tm, k, i) = GET(tm1, i);
```

```
        MLET(tp, k, i) = GET(tp1, i);
        i++;
      }
    while (i < 2 * kmax);
  }// end for k

for (k = 0; k < Nr; k++)
  {
    LET(lamka, k) = 0.0;
    for (j = 0; j < Nr; j++)
      {
        MLET(lam, k, j) = MGET(lam, k, j) * dt;
        LET(lamka, k) = GET(lamka, k) + MGET(lam, k, j) * GET(Strike1, j);
      }
    LET(Lambda1, k) = GET(Lambda1, k) * dt;
    LET(lamka, k) = GET(lamka, k) - (dt * GET(r, k) + GET(Lambda1, k)) * GET(S
    LET(lamka, k) = GET(lamka, k) / GET(a2_1, k);
  }

flagCall = ifCall ? 1.0 : -1.0; //  1.0 for Call, -1.0 for Put

for (j = 0; j < Nmax; j++)
  {
    LET(y, j) = (j - N01) * h;
    LET(expy, j) = K * exp(GET(y, j));
    LET(eom, j) = exp(-om * GET(y, j));
    for (k = 0; k < Nr; k++)
      {
        MLET(prices_all, k, j) = 0;
      }
  }


if (ifCall == 0)
  {
    for (k = 0; k < Nr; k++)
      {
        j = Nmax - 1;
        do
          {
            MLET(prices_all, k, j) = (-GET(Strike1, k) + GET(expy, j)) * GET(e
```

```
            j--;
          }
        while ((GET(Strike1, k) < GET(expy, j)) && (j >= 0));
      }
  }
else
  {
    for (k = 0; k < Nr; k++)
      {
        j = 0;
        do
          {
            MLET(prices_all, k, j) = (GET(Strike1, k) - GET(expy, j)) * GET(eo
            j++;
          }
        while ((GET(Strike1, k) > GET(expy, j)) && (j < Nmax));
      }
  }

//======= EXCHANGE tp AND tm ARRAYS FOR CALL
if (ifCall == 1)
  {
    switcharrow = tp;
    tp = tm;
    tm = switcharrow;
  }
//============================================
for (k = 0; k < Nr; k++)
  {

    cuteom(eomcut, eom, Nmax, N01, flagCall);

    fft_real(eomcut, 2 * kmax, -1);

    LET(eomcut, 0) = GET(eomcut, 0) * MGET(tm, k, 0);
    LET(eomcut, 1) = GET(eomcut, 1) * MGET(tm, k, 1);

    for (i = 1; i <= kmax - 1; i++)
      {
        t1 = GET(eomcut, 2 * i);
        t2 = GET(eomcut, 2 * i + 1);
```

```
        LET(eomcut, 2 * i) = t1 * MGET(tm, k, 2 * i) - t2 * MGET(tm, k, 2 * i
        LET(eomcut, 2 * i + 1) = t2 * MGET(tm, k, 2 * i) + t1 * MGET(tm, k, 2
      }
    fft_real(eomcut, 2 * kmax, 1);

    LET(lamka, k) = GET(lamka, k) * GET(eomcut, N01);
  }


for (k = 0; k < Nr; k++)
  {
    for (i = 0; i < 2 * kmax; i++) MLET(tm, k, i) = MGET(tm, k, i) / GET(a2_1,
  }


for (k = 0; k < Nr; k++)
  {
    for (i = 0; i < Nmax; i++)
      {
        LET(g1, i) = GET(divid, k) * dt * GET(eom, i) * GET(expy, i);
        MLET(prices_old, k, i) = MGET(prices_all, k, i);
      }

    fft_real(g1, 2 * kmax, -1);

    LET(g1, 0) = GET(g1, 0) * MGET(tm, k, 0);
    LET(g1, 1) = GET(g1, 1) * MGET(tm, k, 1);

    for (i = 1; i <= kmax - 1; i++)
      {
        t1 = GET(g1, 2 * i);
        t2 = GET(g1, 2 * i + 1);
        LET(g1, 2 * i) = t1 * MGET(tm, k, 2 * i) - t2 * MGET(tm, k, 2 * i + 1)
        LET(g1, 2 * i + 1) = t2 * MGET(tm, k, 2 * i) + t1 * MGET(tm, k, 2 * i
      }

    fft_real(g1, 2 * kmax, 1);

    for (i = 0; i < Nmax; i++)
      {
        MLET(G, k, i) = - (GET(g1, i) + GET(lamka, k) * GET(eom, i)) * flagCal
```

```
        }
    }// END FOR k

//=======================* L- Loop on time grid*=======================/

  for (L = 2; L <= N1 + 1; L++)
    {

      itererr = 1;
      while (itererr > eps)
        {
          for (k = 0; k < Nr; k++)
            {
              for (i = 0; i < Nmax; i++)
                {
                  LET(vv1, i) = MGET(prices_all, k, i);
                  for (j = 0; j < Nr; j++)
                    {
                      LET(vv1, i) = GET(vv1, i) + MGET(lam, k, j) * MGET(prices_
                    }
                }

              fft_real(vv1, 2 * kmax, -1);

              LET(vv1, 0) = GET(vv1, 0) * MGET(tm, k, 0);
              LET(vv1, 1) = GET(vv1, 1) * MGET(tm, k, 1);

              for (i = 1; i <= kmax - 1; i++)
                {
                  t1 = GET(vv1, 2 * i);
                  t2 = GET(vv1, 2 * i + 1);
                  LET(vv1, 2 * i) = t1 * MGET(tm, k, 2 * i) - t2 * MGET(tm, k, 2
                  LET(vv1, 2 * i + 1) = t2 * MGET(tm, k, 2 * i) + t1 * MGET(tm,
                }

              fft_real(vv1, 2 * kmax, 1);

              for (i = 0; i < Nmax; i++)
                {
                  LET(vv1, i) = GET(vv1, i) + MGET(G, k, i);
                  if (GET(vv1, i) < 0)
```

```
        {
          LET(vv1, i) = 0.0;
        }
    }

  fft_real(vv1, 2 * kmax, -1);

  LET(vv1, 0) = GET(vv1, 0) * MGET(tp, k, 0);
  LET(vv1, 1) = GET(vv1, 1) * MGET(tp, k, 1);

  for (i = 1; i <= kmax - 1; i++)
    {
      t1 = GET(vv1, 2 * i);
      t2 = GET(vv1, 2 * i + 1);
      LET(vv1, 2 * i) = t1 * MGET(tp, k, 2 * i) - t2 * MGET(tp, k, 2
      LET(vv1, 2 * i + 1) = t2 * MGET(tp, k, 2 * i) + t1 * MGET(tp,
    }

  fft_real(vv1, 2 * kmax, 1);

  LET(iter, k) = 0;
  for (i = 0; i < Nmax; i++)
    {
      dif = fabs(GET(vv1, i) - MGET(prices_old, k, i));
      LET(iter, k) = GET(iter, k) > dif ? GET(iter, k) : dif;
    }

  LET(iter, k) = GET(iter, k) / GET(Strike1, k);

  for (i = 0; i < Nmax; i++)
    {
      MLET(prices_new, k, i) = GET(vv1, i);
    }
}// end loop in k

itererr = 0;
for (k = 0; k < Nr; k++)
  {
    itererr = itererr > GET(iter, k) ? itererr : GET(iter, k);
  }
```

```
        for (k = 0; k < Nr; k++)
          {
            for (i = 0; i < Nmax; i++)
              {
                MLET(prices_old, k, i) = MGET(prices_new, k, i);
              }
          }
      }//end while


    for (k = 0; k < Nr; k++)
      {
        for (i = 0; i < Nmax; i++)
          {
            MLET(prices_all, k, i) = MGET(prices_new, k, i);
          }
      }

  }// ======================================== PREMIA_NULLTYPE L-LOOP ========

for (k = 0; k < Nr; k++)
  {
    for (i = 0; i < Nmax; i++)
      {
        MLET(prices_all, k, i) = MGET(prices_new, k, i) / GET(eom, i) + flagCa
      }
  }

pp = log(Spot / K);

i = 2;

while ((GET(y, i) <= pp) && (i < Nmax))
  {
    i++;
  }

i--;
for (k = 0; k < Nr; k++)
  {
    //Price, quadratic interpolation
```

```
      Sl = GET(expy, i - 1);
      Sm = GET(expy, i);
      Sr = GET(expy, i + 1);
// S0 is between Sm and Sr
      pricel = MGET(prices_all, k, i - 1);
      pricem = MGET(prices_all, k, i);
      pricer = MGET(prices_all, k, i + 1);

      //quadratic interpolation
      A = pricel;
      B = (pricem - pricel) / (Sm - Sl);
      C = (pricer - A - B * (Sr - Sl)) / (Sr - Sl) / (Sr - Sm);

      //Price
      LET(ptprice, k) = A + B * (Spot - Sl) + C * (Spot - Sl) * (Spot - Sm);
      if (GET(ptprice, k) < 0)
        {
          LET(ptprice, k) = 0.0;
        }
      //Delta
      LET(ptdelta, k) = B + C * (2 * Spot - Sl - Sm);
    }


  /*Memory desallocation*/
  pnl_vect_free(&Lambda1);
  pnl_vect_free(&gam);
  pnl_vect_free(&y);
  pnl_vect_free(&expy);
  pnl_vect_free(&vv1);
  pnl_vect_free(&tp1);
  pnl_vect_free(&alin1);
  pnl_vect_free(&tm1);
  pnl_vect_free(&al1);
  pnl_vect_free(&eom);
  pnl_vect_free(&eomcut);

  pnl_mat_free(&prices_old);

  pnl_mat_free(&prices_new);
```

```
  pnl_mat_free(&prices_all);
  pnl_vect_free(&g1);

  pnl_mat_free(&G);

  pnl_mat_free(&tm);

  pnl_mat_free(&tp);
  pnl_vect_free(&a2_1);

  pnl_vect_free(&iter);
  pnl_vect_free(&lamka);

  return OK;
}

////////////////////////////////////////Heston barrier and digital///////////////
int fastwienerhopf_hs(int model, long int Nr, PnlVect *mu, PnlVect *qu, double o
                      int ifCall, double Spot, double lm1, double lp1,
                      PnlVect *sg, PnlVect *num, PnlVect *nup, double cnum, doub
                      double r, double divid, PnlMat *lam,
                      double T, double h, double Strike1,
                      double bar, double rebate,
                      double er, long int step, PnlVect *ptprice, PnlVect *ptdel
{

  PnlVect *y, *expy, *vv1;
  PnlVect *alin1, *eom, *al1, *tp1, *tm1;
  PnlVect *coef;

  long int N1 = 0, kmax, i, j, L;

  PnlVect *Lambda1, *lamka, *a2_1;
  PnlMat  *prices_old, *prices_new, *tp, *tm;
  PnlMat  *prices_all, *switcharrow, *G, *lam1;

  double pp;

  double t1, t2;
//double dif;
```

```
double qq;
long int k, i0;
long int Nmax, N01;
int nn, n;
double dt;

double kx;
double S0, Sl, Sm, Sr, pricel, pricem, pricer;
double A, B, C;
double np, nm;

if (upordown == 0)
  {
    if (Spot <= bar)
      {
        for (k = 0; k < Nr; k++)
          {
            LET(ptprice, k) = rebate;
            LET(ptdelta, k) = 0.;
          }
        return OK;
      }
    if (Spot >= 0.8 * bar * exp(er * log(2.)))
      {
        for (k = 0; k < Nr; k++)
          {
            LET(ptprice, k) = 0.;
            LET(ptdelta, k) = 0.;
          }
        printf("Spot is out of range. Increase scale parameter \ n");
        return OK;
      }
  }
else
  {
    if (Spot >= bar)
      {
        for (k = 0; k < Nr; k++)
          {
            LET(ptprice, k) = rebate;
            LET(ptdelta, k) = 0.;
```

```
            }
          return OK;
        }
      if (Spot <= bar * exp(-er * log(2.)) * 1.2)
        {
          for (k = 0; k < Nr; k++)
            {
              LET(ptprice, k) = 0.;
              LET(ptdelta, k) = 0.;
            }
          printf("Spot is out of range. Increase scale parameter \ n");
          return OK;
        }
    }

  k = 64;

  kx = ceil(er * log(2.0) / h);
  while (k < kx)
    {
      k = k * 2;
    }
  kmax = k;

  N1 = step;
  Nmax = 2 * kmax;
  NO1 = kmax - 1;
  nn = 3;
//                              Memory allocation
  Lambda1 = pnl_vect_create(Nr + 1);
//gam= pnl_vect_create(Nr+1);
//
  y = pnl_vect_create(Nmax + 1); //log space grid points
  expy = pnl_vect_create(Nmax + 1); //space grid points
  vv1 = pnl_vect_create(Nmax + 2);
  alin1 = pnl_vect_create(Nmax + 2);
  tp1 = pnl_vect_create(Nmax + 2);
  tm1 = pnl_vect_create(Nmax + 2);
  al1 = pnl_vect_create(Nmax + 2);
  eom = pnl_vect_create(Nmax + 1);
  prices_old = pnl_mat_create(Nr + 1, Nmax + 2);
```

```
  prices_new = pnl_mat_create(Nr + 1, Nmax + 2);
  prices_all = pnl_mat_create(Nr + 1, Nmax + 2);
//g1= pnl_vect_create(Nmax+2);
  G = pnl_mat_create(Nr + 1, Nmax + 2);
  lam1 = pnl_mat_create(Nr + 1, Nr + 1);

  tm = pnl_mat_create(Nr + 1, Nmax + 2);
  tp = pnl_mat_create(Nr + 1, Nmax + 2);
  a2_1 = pnl_vect_create(Nr + 1);
//iter= pnl_vect_create(Nr+1);
  lamka = pnl_vect_create(Nr + 1);

  coef = pnl_vect_create(nn + 1);

  lp1 = lp1 + om;
  lm1 = lm1 + om;
  for (i = 0; i < Nr; i++)
    {
      LET(Lambda1, i) = 0;
      MLET(lam, i, i) = 0.0;
      for (k = 0; k < Nr; k++)
        {
          LET(Lambda1, i) = GET(Lambda1, i) + MGET(lam, i, k);
        }

      LET(ptprice, i) = 0.;
      LET(ptdelta, i) = 0.;
    }



//============================ compute coefficients
  if (model == 6) //Heston//
    {
      for (i = 0; i < Nr; i++)
        {
          LET(qu, i) = GET(qu, i) + GET(Lambda1, i);
        }
      np = 0.0;
      nm = 0.0;
    }
```

```
//-----------------------------------------------weghts-------------------------
  LET(coef, 1) = 0.5;
  LET(coef, 2) = -4;
  LET(coef, 3) = 4.5;
//-------------------------------------------------------------------loop in weighted
  for (n = 1; n <= nn; n++)
    {
      dt = T / (n * N1 + 1);
      for (k = 0; k < Nr; k++)
        {
          qq = GET(qu, k) + 1.0 / dt;
          findcoefnew(model, GET(mu, k), GET(sg, k), lm1, lp1, GET(num, k), GET(

          LET(a2_1, k) = qq * dt;
          if (GET(sg, k) > 0.004)
            {
              np = 2.0;
              nm = 2.0;
            }
          else
            {
              np = 0.0;
              nm = 0.0;
            }
          for (i = 0; i < 2 * kmax; i++) LET(alin1, i) = GET(al1, i + 1);
          findfactor(np, nm, h, kmax, lp1, lm1, alin1, tp1, tm1);
          i = 0;
          do
            {
              MLET(tm, k, i) = GET(tm1, i);
              MLET(tp, k, i) = GET(tp1, i);
              i++;
            }
          while (i < 2 * kmax);
        }


////////////////////non zero correlation ////////////////////////
      for (j = 1; j < Nr - 1; j++)
```

```
        {
          MLET(lam1, j, j - 1) = MGET(lam, j, j - 1) * dt * exp(-GET(num, j) * o
          MLET(lam1, j, j + 1) = MGET(lam, j, j + 1) * dt * exp(GET(nup, j) * om
          LET(lamka, j) = (MGET(lam1, j, j - 1) + MGET(lam1, j, j + 1)) * rebate
        }
      MLET(lam1, Nr - 1, Nr - 2) = MGET(lam, Nr - 1, Nr - 2) * dt * exp(-GET(num
      MLET(lam1, 0, 1) = MGET(lam, 0, 1) * dt * exp(GET(nup, 0) * om);
      LET(lamka, 0) = MGET(lam1, 0, 1) * rebate;
      LET(lamka, Nr - 1) = MGET(lam1, Nr - 1, Nr - 2) * rebate;

      for (k = 0; k < Nr; k++)
        {
          //LET(lamka,k) = GET(lamka,k) -dt*(r+GET(Lambda1,k))*rebate;
          LET(lamka, k) = -dt * r * rebate;
          LET(lamka, k) = GET(lamka, k) / GET(a2_1, k);
        }
////////////////////////////////////////

//Put Pay-off functions//
      if (upordown == 0) //DOWN
        for (j = 0; j < kmax; j++)
          {
            LET(y, j) = (j - NO1 - 0.5) * h;
            LET(eom, j) = exp(-om * GET(y, j));
            LET(expy, j) = bar * exp(GET(y, j));
            for (k = 0; k < Nr; k++)
              {
                MLET(prices_all, k, j) = 0;
              }
          }
      else // UP
        for (j = kmax - 1; j < Nmax; j++)
          {
            LET(y, j) = (j - NO1 + 0.5) * h;
            LET(eom, j) = exp(-om * GET(y, j));
            LET(expy, j) = bar * exp(GET(y, j));
            for (k = 0; k < Nr; k++)
              {
                MLET(prices_all, k, j) = 0;
              }
          }
```

```
if (upordown == 0) // IF DOWN
  {
    j = kmax;

    if (ifCall == 2) // DIGITAL //
      {
        do
          {
            LET(y, j) = (j - NO1 - 0.5) * h;
            LET(eom, j) = exp(-om * GET(y, j));
            LET(expy, j) = bar * exp(GET(y, j));
            for (k = 0; k < Nr; k++)
              {
                MLET(prices_all, k, j) = -rebate * GET(eom, j);
              }
            j++;
          }
        while (j < Nmax);
      }
    else if (ifCall == 1) //  CALL //
      {
        do
          {
            LET(y, j) = (j - NO1 - 0.5) * h;
            LET(eom, j) = exp(-om * GET(y, j));
            LET(expy, j) = bar * exp(GET(y, j));
            for (k = 0; k < Nr; k++)
              {
                if (GET(expy, j) >= Strike1)
                  {
                    MLET(prices_all, k, j) = (-Strike1 + GET(expy, j) - re
                  }
                else
                  {
                    MLET(prices_all, k, j) = -rebate * GET(eom, j);
                  }
              }
            j++;
          }
        while (j < Nmax);
```

```
      }
    else              // PUT //
      {
        do
          {
            LET(y, j) = (j - NO1 - 0.5) * h;
            LET(eom, j) = exp(-om * GET(y, j));
            LET(expy, j) = bar * exp(GET(y, j));
            for (k = 0; k < Nr; k++)
              {
                if (GET(expy, j) <= Strike1)
                  {
                    MLET(prices_all, k, j) = (Strike1 - GET(expy, j) - reb
                  }
                else
                  {
                    MLET(prices_all, k, j) = -rebate * GET(eom, j);
                  }
              }
            j++;
          }
        while (j < Nmax);
      }
  }// END IF DOWN-OUT
else // IF UP-OUT
  {
    j = kmax - 2;

    if (ifCall == 2)   // DIGITAL //
      {
        do
          {
            LET(y, j) = (j - NO1 + 0.5) * h;
            LET(eom, j) = exp(-om * GET(y, j));
            LET(expy, j) = bar * exp(GET(y, j));
            for (k = 0; k < Nr; k++)
              {
                MLET(prices_all, k, j) = -rebate * GET(eom, j);
              }
            j--;
          }
```

```
    while (j >= 0);
  }
else if (ifCall == 1) // CALL //
  {
    do
      {
        LET(y, j) = (j - NO1 + 0.5) * h;
        LET(eom, j) = exp(-om * GET(y, j));
        LET(expy, j) = bar * exp(GET(y, j));
        for (k = 0; k < Nr; k++)
          {
            if (GET(expy, j) >= Strike1)
              {
                MLET(prices_all, k, j) = (-Strike1 + GET(expy, j) - re
              }
            else
              {
                MLET(prices_all, k, j) = -rebate * GET(eom, j);
              }
          }
        j--;
      }
    while (j >= 0);
  }
else                  // PUT //
  {
    do
      {
        LET(y, j) = (j - NO1 + 0.5) * h;
        LET(eom, j) = exp(-om * GET(y, j));
        LET(expy, j) = bar * exp(GET(y, j));
        for (k = 0; k < Nr; k++)
          {
            if (GET(expy, j) <= Strike1)
              {
                MLET(prices_all, k, j) = (Strike1 - GET(expy, j) - reb
              }
            else
              {
                MLET(prices_all, k, j) = -rebate * GET(eom, j);
              }
```

```
                }
              j--;
            }
          while (j >= 0);
        }
    }// END IF UP


  //======= EXCHANGE tp AND tm ARRAYS FOR UP-OUT
  if (upordown == 1)
    {
      switcharrow = tp;
      tp = tm;
      tm = switcharrow;
    }
  //=============================================
  for (k = 0; k < Nr; k++)
    {
      for (i = 0; i < 2 * kmax; i++) MLET(tm, k, i) = MGET(tm, k, i) / GET(a
    }

  for (k = 0; k < Nr; k++)
    {
      for (i = 0; i < Nmax; i++)
        {
          MLET(prices_old, k, i) = MGET(prices_all, k, i);
        }
      for (i = 0; i < Nmax; i++)
        {
          MLET(G, k, i) = GET(lamka, k) * GET(eom, i);
        }
    }

//======================*L-Loop*======================/

  for (L = 1; L <= N1 * n + 1; L++)
    {

      for (k = 0; k < Nr; k++)
        {
          if (upordown == 0) // IF DOWN
```

```
{
  for (i = 0; i < Nmax; i++)
    {
      LET(vv1, i) = MGET(prices_all, k, i);

      if (k > 0)
        {
          i0 = (int)ceil(GET(num, k) / h);
          if (GET(y, i) > GET(num, k)) //down-and-out
            {
              if (i - i0 < Nmax - 1)
                {
                  S0 = bar * exp(GET(y, i) - GET(num, k));
                  Sl = GET(expy, i - i0 - 1);
                  Sm = GET(expy, i - i0);
                  Sr = GET(expy, i - i0 + 1);
                  // S0 is between Sm and Sr
                  pricel = MGET(prices_old, k - 1, i - i0 - 1);
                  pricem = MGET(prices_old, k - 1, i - i0);
                  pricer = MGET(prices_old, k - 1, i - i0 + 1);
                  //quadratic interpolation
                  A = pricel;
                  B = (pricem - pricel) / (Sm - Sl);
                  C = (pricer - A - B * (Sr - Sl)) / (Sr - Sl) /
                  //Price
                  LET(vv1, i) = GET(vv1, i) + MGET(lam1, k, k -
                }
            }
        }
      if (k < Nr - 1)
        {
          i0 = (int)ceil(GET(nup, k) / h);
          if (GET(y, i) > -GET(nup, k)) //down-and-out
            {
              if (i + i0 < Nmax)
                {
                  S0 = bar * exp(GET(y, i) + GET(nup, k));
                  Sl = GET(expy, i + i0 - 2);
                  Sm = GET(expy, i + i0 - 1);
                  Sr = GET(expy, i + i0);
                  // S0 is between Sm and Sr
```

```
                    pricel = MGET(prices_old, k + 1, i + i0 - 2);
                    pricem = MGET(prices_old, k + 1, i + i0 - 1);
                    pricer = MGET(prices_old, k + 1, i + i0);
                    //quadratic interpolation
                    A = pricel;
                    B = (pricem - pricel) / (Sm - Sl);
                    C = (pricer - A - B * (Sr - Sl)) / (Sr - Sl) /
                    //Price
                    LET(vv1, i) = GET(vv1, i) + MGET(lam1, k, k +
                  }
                }
              }

      }//end i
  }//end if upordown=0
if (upordown == 1) // IF UP
  {
    for (i = 0; i < Nmax; i++)
      {
        LET(vv1, i) = MGET(prices_all, k, i);

        if (k > 0)
          {
            i0 = (int)ceil(GET(num, k) / h);
            if (GET(y, i) < GET(num, k)) //up-and-out
              {
                if (i - i0 > 0)
                  {
                    S0 = bar * exp(GET(y, i) - GET(num, k));
                    Sl = GET(expy, i - i0 - 1);
                    Sm = GET(expy, i - i0);
                    Sr = GET(expy, i - i0 + 1);
                    // S0 is between Sm and Sr
                    pricel = MGET(prices_old, k - 1, i - i0 - 1);
                    pricem = MGET(prices_old, k - 1, i - i0);
                    pricer = MGET(prices_old, k - 1, i - i0 + 1);
                    //quadratic interpolation
                    A = pricel;
                    B = (pricem - pricel) / (Sm - Sl);
                    C = (pricer - A - B * (Sr - Sl)) / (Sr - Sl) /
                    //Price
```

```
                    LET(vv1, i) = GET(vv1, i) + MGET(lam1, k, k -
                  }
                }
              }
            if (k < Nr - 1)
              {
                i0 = (int)ceil(GET(nup, k) / h);
                if (GET(y, i) < -GET(nup, k)) //up-and-out
                  {
                    if (i + i0 > 1)
                      {
                        S0 = bar * exp(GET(y, i) + GET(nup, k));
                        Sl = GET(expy, i + i0 - 2);
                        Sm = GET(expy, i + i0 - 1);
                        Sr = GET(expy, i + i0);
                        // S0 is between Sm and Sr
                        pricel = MGET(prices_old, k + 1, i + i0 - 2);
                        pricem = MGET(prices_old, k + 1, i + i0 - 1);
                        pricer = MGET(prices_old, k + 1, i + i0);
                        //quadratic interpolation
                        A = pricel;
                        B = (pricem - pricel) / (Sm - Sl);
                        C = (pricer - A - B * (Sr - Sl)) / (Sr - Sl) /
                        //Price
                        LET(vv1, i) = GET(vv1, i) + MGET(lam1, k, k +
                      }
                  }
              }

          }//end i
      }//end if upordown=1

    fft_real(vv1, 2 * kmax, -1);

    LET(vv1, 0) = GET(vv1, 0) * MGET(tm, k, 0);
    LET(vv1, 1) = GET(vv1, 1) * MGET(tm, k, 1);

    for (i = 1; i <= kmax - 1; i++)
      {
        t1 = GET(vv1, 2 * i);
        t2 = GET(vv1, 2 * i + 1);
```

```
    LET(vv1, 2 * i) = t1 * MGET(tm, k, 2 * i) - t2 * MGET(tm, k, 2
    LET(vv1, 2 * i + 1) = t2 * MGET(tm, k, 2 * i) + t1 * MGET(tm,
  }

fft_real(vv1, 2 * kmax, 1);

if (upordown == 0) // IF DOWN-OUT!!
  {
    i = 0;
    do
      {
        LET(vv1, i) = 0;
        i++;
      }
    while (GET(y, i) <= 0.);
    do
      {
        LET(vv1, i) = GET(vv1, i) + MGET(G, k, i);
        i++;
      }
    while (i < Nmax);
  }
else  //IF UP-OUT
  {
    i = Nmax - 1; //-1;
    do
      {
        LET(vv1, i) = 0;
        i--;
      }
    while (GET(y, i) >= 0.);
    do
      {
        LET(vv1, i) = GET(vv1, i) + MGET(G, k, i);
        i--;
      }
    while (i >= 0);
  }

fft_real(vv1, 2 * kmax, -1);
```

```
    LET(vv1, 0) = GET(vv1, 0) * MGET(tp, k, 0);
    LET(vv1, 1) = GET(vv1, 1) * MGET(tp, k, 1);

    for (i = 1; i <= kmax - 1; i++)
      {
        t1 = GET(vv1, 2 * i);
        t2 = GET(vv1, 2 * i + 1);
        LET(vv1, 2 * i) = t1 * MGET(tp, k, 2 * i) - t2 * MGET(tp, k, 2
        LET(vv1, 2 * i + 1) = t2 * MGET(tp, k, 2 * i) + t1 * MGET(tp,
      }

    fft_real(vv1, 2 * kmax, 1);



    for (i = 0; i < Nmax; i++)
      {
        MLET(prices_new, k, i) = GET(vv1, i);
      }
  }// end loop in k



for (k = 0; k < Nr; k++)
  {
    for (i = 0; i < Nmax; i++)
      {
        MLET(prices_old, k, i) = MGET(prices_new, k, i);
      }
  }



for (k = 0; k < Nr; k++)
  {
    for (i = 0; i < Nmax; i++)
      {
        MLET(prices_all, k, i) = MGET(prices_new, k, i);
      }
  }
```

```
  }// ======================================== PREMIA_NULLTYPE L-LOOP ====

for (k = 0; k < Nr; k++)
  {
    for (i = 0; i < Nmax; i++)
      {
        MLET(prices_all, k, i) = MGET(prices_new, k, i) / GET(eom, i) + re
      }
  }

pp = log(Spot / bar);

i = 2;

while ((GET(y, i) <= pp) && (i < Nmax))
  {
    i++;
  }

i--;
for (k = 0; k < Nr; k++)
  {
    //Price, quadratic interpolation
    Sl = GET(expy, i - 1);
    Sm = GET(expy, i);
    Sr = GET(expy, i + 1);
    // S0 is between Sm and Sr
    pricel = MGET(prices_all, k, i - 1);
    pricem = MGET(prices_all, k, i);
    pricer = MGET(prices_all, k, i + 1);

    //quadratic interpolation
    A = pricel;
    B = (pricem - pricel) / (Sm - Sl);
    C = (pricer - A - B * (Sr - Sl)) / (Sr - Sl) / (Sr - Sm);

    //Price
    pricem = A + B * (Spot - Sl) + C * (Spot - Sl) * (Spot - Sm);

    //Delta
```

```
          pricel = B + C * (2 * Spot - Sl - Sm);
          LET(ptprice, k) = GET(ptprice, k) + LET(coef, n) * pricem;
          LET(ptdelta, k) = GET(ptdelta, k) + LET(coef, n) * pricel;
        }// end for(k)
    }// end for(n)

  for (k = 0; k < Nr; k++)
    {
      if (GET(ptprice, k) < 0)
        {
          LET(ptprice, k) = 0.0;
          LET(ptdelta, k) = 0.0;
        }
    }

  /*Memory desallocation*/
  pnl_vect_free(&Lambda1);
//pnl_vect_free(&gam);
  pnl_vect_free(&y);
  pnl_vect_free(&expy);
  pnl_vect_free(&vv1);
  pnl_vect_free(&alin1);
  pnl_vect_free(&tp1);
  pnl_vect_free(&tm1);
  pnl_vect_free(&al1);
  pnl_vect_free(&eom);

  pnl_mat_free(&prices_old);

  pnl_mat_free(&prices_new);

  pnl_mat_free(&prices_all);
//pnl_vect_free(&g1);

  pnl_mat_free(&G);
  pnl_mat_free(&lam1);
  pnl_mat_free(&tm);

  pnl_mat_free(&tp);
  pnl_vect_free(&a2_1);
//pnl_vect_free(&iter);
```

```
  pnl_vect_free(&lamka);
  pnl_vect_free(&coef);

  return OK;
}



//////////////////////American Heston////////////////////////////////////
//////////////////////////////////////Heston barrier and digital///////////////
int fastwienerhopfamer_hs(int model, long int Nr, PnlVect *mu, PnlVect *qu, doub
                          int ifCall, double Spot, double lm1, double lp1,
                          PnlVect *sg, PnlVect *num, PnlVect *nup, double cnum,
                          double r, double divi, PnlMat *lam,
                          double T, double h, double Strike1,
                          double er, long int step, PnlVect *ptprice, PnlVect *p
{

  PnlVect *y, *expy, *vv1;
  PnlVect *alin1, *eom, *eomcut, *al1, *tp1, *tm1;
  PnlVect *coef;

  long int N1 = 0, kmax, i, j, L;

  PnlVect *Lambda1, *lamka, *a2_1, *divid, *g1;
  PnlMat  *prices_old, *prices_new, *tp, *tm;
  PnlMat  *prices_all, *switcharrow, *G, *lam1;

  double pp;

  double t1, t2;
//double dif;

  double qq;
  long int k, i0;
  long int Nmax, N01;
  int nn, n;
  double dt;

  double kx;
  double S0, Sl, Sm, Sr, pricel, pricem, pricer;
```

```
double A, B, C;
double np, nm;
double flagCall;//=1 for call and =-1 for put

if (ifCall == 0) //IF PUT
  {
    if (Spot >= 0.8 * Strike1 * exp(er * log(2.)))
      {
        for (k = 0; k < Nr; k++)
          {
            LET(ptprice, k) = 0.;
            LET(ptdelta, k) = 0.;
          }
        printf("Spot is out of range. Increase scale parameter \ n");
        return OK;
      }
    if (Spot <= 1.2 * Strike1 * exp(-er * log(2.)))
      {
        for (k = 0; k < Nr; k++)
          {
            LET(ptprice, k) = Strike1 - Spot;
            LET(ptdelta, k) = -1.0;
          }
        printf("Spot is out of range. Increase scale parameter \ n");
        return OK;
      }
  }
else
  {
    if (Spot >= 0.8 * Strike1 * exp(er * log(2.)))
      {
        for (k = 0; k < Nr; k++)
          {
            LET(ptprice, k) = - Strike1 + Spot;
            LET(ptdelta, k) = 1.;
          }
        printf("Spot is out of range. Increase scale parameter \ n");
        return OK;
      }
    if (Spot <= Strike1 * exp(-er * log(2.)) * 1.2)
      {
```

```
          for (k = 0; k < Nr; k++)
            {
              LET(ptprice, k) = 0.0;
              LET(ptdelta, k) = 0.0;
            }
          printf("Spot is out of range. Increase scale parameter \ n");
          return OK;
        }
    }

  k = 64;

  kx = ceil(er * log(3.0) / h);
  while (k < kx)
    {
      k = k * 2;
    }
  kmax = k;

  N1 = step;
  Nmax = 2 * kmax;
  NO1 = kmax - 1;
  nn = 3;
//                              Memory allocation
  Lambda1 = pnl_vect_create(Nr + 1);
//gam= pnl_vect_create(Nr+1);
//
  y = pnl_vect_create(Nmax + 1); //log space grid points
  expy = pnl_vect_create(Nmax + 1); //space grid points
  vv1 = pnl_vect_create(Nmax + 2);
  alin1 = pnl_vect_create(Nmax + 2);
  tp1 = pnl_vect_create(Nmax + 2);
  tm1 = pnl_vect_create(Nmax + 2);
  al1 = pnl_vect_create(Nmax + 2);
  eom = pnl_vect_create(Nmax + 1);
  eomcut = pnl_vect_create(Nmax + 1);
  prices_old = pnl_mat_create(Nr + 1, Nmax + 2);
  prices_new = pnl_mat_create(Nr + 1, Nmax + 2);
  prices_all = pnl_mat_create(Nr + 1, Nmax + 2);
  g1 = pnl_vect_create(Nmax + 2);
  divid = pnl_vect_create(Nmax + 2);
```

```
  G = pnl_mat_create(Nr + 1, Nmax + 2);
  lam1 = pnl_mat_create(Nr + 1, Nr + 1);

  tm = pnl_mat_create(Nr + 1, Nmax + 2);
  tp = pnl_mat_create(Nr + 1, Nmax + 2);
  a2_1 = pnl_vect_create(Nr + 1);
//iter= pnl_vect_create(Nr+1);
  lamka = pnl_vect_create(Nr + 1);

  coef = pnl_vect_create(nn + 1);

  lp1 = lp1 + om;
  lm1 = lm1 + om;
  for (i = 0; i < Nr; i++)
    {
      LET(Lambda1, i) = 0;
      MLET(lam, i, i) = 0.0;
      for (k = 0; k < Nr; k++)
        {
          LET(Lambda1, i) = GET(Lambda1, i) + MGET(lam, i, k);
        }

      LET(ptprice, i) = 0.;
      LET(ptdelta, i) = 0.;
    }



//============================ compute coefficients
  if (model == 6) //Heston//
    {
      for (i = 0; i < Nr; i++)
        {
          LET(qu, i) = GET(qu, i) + GET(Lambda1, i);
        }
      np = 0.0;
      nm = 0.0;
    }


//-----------------------------------------------weghts-------------------------
```

```
  LET(coef, 1) = 0.5;
  LET(coef, 2) = -4;
  LET(coef, 3) = 4.5;
  //LET(coef,1) = 1.;
  //nn=1;
//-----------------------------------------------------------------loop in weighted
  for (n = 1; n <= nn; n++)
    {
      dt = T / (n * N1 + 1);
      for (k = 0; k < Nr; k++)
        {
          qq = GET(qu, k) + 1.0 / dt;
          findcoefnew(model, GET(mu, k), GET(sg, k), lm1, lp1, GET(num, k), GET(

          LET(a2_1, k) = qq * dt;
          //if(GET(sg,k)>0.004){np=2.0;nm=2.0;}
          //else{np=0.0;nm=0.0;}
          for (i = 0; i < 2 * kmax; i++) LET(alin1, i) = GET(al1, i + 1);
          findfactor(np, nm, h, kmax, lp1, lm1, alin1, tp1, tm1);
          i = 0;
          do
            {
              MLET(tm, k, i) = GET(tm1, i);
              MLET(tp, k, i) = GET(tp1, i);
              i++;
            }
          while (i < 2 * kmax);
        }


////////////////non zero correlation ///////////////////////
      for (j = 1; j < Nr - 1; j++)
        {
          MLET(lam1, j, j - 1) = MGET(lam, j, j - 1) * dt * exp(-GET(num, j) * o
          MLET(lam1, j, j + 1) = MGET(lam, j, j + 1) * dt * exp(GET(nup, j) * om
        }
      MLET(lam1, Nr - 1, Nr - 2) = MGET(lam, Nr - 1, Nr - 2) * dt * exp(-GET(num
      MLET(lam1, 0, 1) = MGET(lam, 0, 1) * dt * exp(GET(nup, 0) * om);

      for (k = 0; k < Nr; k++)
        {
```

```
          LET(lamka, k) = -dt * r * Strike1;
          LET(lamka, k) = GET(lamka, k) / GET(a2_1, k);
        }
////////////////////////////////////////

      flagCall = ifCall ? 1.0 : -1.0; //  1.0 for Call, -1.0 for Put

      for (j = 0; j < Nmax; j++)
        {
          LET(y, j) = (j - N01) * h;
          LET(expy, j) = Strike1 * exp(GET(y, j));
          LET(eom, j) = exp(-om * GET(y, j));
          for (k = 0; k < Nr; k++)
            {
              MLET(prices_all, k, j) = 0;
            }
        }


      if (ifCall == 0)
        {
          for (k = 0; k < Nr; k++)
            {
              j = Nmax - 1;
              do
                {
                  MLET(prices_all, k, j) = (-Strike1 + GET(expy, j)) * GET(eom,
                  j--;
                }
              while ((Strike1 < GET(expy, j)) && (j >= 0));
            }
        }
      else
        {
          for (k = 0; k < Nr; k++)
            {
              j = 0;
              do
                {
                  MLET(prices_all, k, j) = (Strike1 - GET(expy, j)) * GET(eom, j
                  j++;
```

```
          }
        while ((Strike1 > GET(expy, j)) && (j < Nmax));
      }
  }

//======= EXCHANGE tp AND tm ARRAYS FOR CALL
if (ifCall == 1)
  {
    switcharrow = tp;
    tp = tm;
    tm = switcharrow;
  }
//=============================================
for (k = 0; k < Nr; k++)
  {

    cuteom(eomcut, eom, Nmax, N01, flagCall);
    //    printf ("   %.3f\ n", GET(eomcut, N01));
    fft_real(eomcut, 2 * kmax, -1);

    LET(eomcut, 0) = GET(eomcut, 0) * MGET(tm, k, 0);
    LET(eomcut, 1) = GET(eomcut, 1) * MGET(tm, k, 1);

    for (i = 1; i <= kmax - 1; i++)
      {
        t1 = GET(eomcut, 2 * i);
        t2 = GET(eomcut, 2 * i + 1);
        LET(eomcut, 2 * i) = t1 * MGET(tm, k, 2 * i) - t2 * MGET(tm, k, 2
        LET(eomcut, 2 * i + 1) = t2 * MGET(tm, k, 2 * i) + t1 * MGET(tm, k
      }
    fft_real(eomcut, 2 * kmax, 1);
    //    printf ("%.3f %.3f %.5f\ n", GET(eomcut, N01), MGET(tm,k,0), GET
    LET(lamka, k) = GET(lamka, k) * GET(eomcut, N01);
  }


for (k = 0; k < Nr; k++)
  {
    for (i = 0; i < 2 * kmax; i++) MLET(tm, k, i) = MGET(tm, k, i) / GET(a
  }
```

```
      for (k = 0; k < Nr; k++)
        {
          LET(divid, k) = r + GET(Lambda1, k) - GET(mu, k) + (om - 0.5) * GET(sg
          if (k > 0)
            {
              LET(divid, k) = GET(divid, k) - MGET(lam, k, k - 1) * exp(-GET(num
            }
          if (k < Nr - 1)
            {
              LET(divid, k) = GET(divid, k) - MGET(lam, k, k + 1) * exp(GET(nup,
            }
          for (i = 0; i < Nmax; i++)
            {
              LET(g1, i) = GET(divid, k) * dt * GET(eom, i) * GET(expy, i);
              MLET(prices_old, k, i) = MGET(prices_all, k, i);
            }

          fft_real(g1, 2 * kmax, -1);

          LET(g1, 0) = GET(g1, 0) * MGET(tm, k, 0);
          LET(g1, 1) = GET(g1, 1) * MGET(tm, k, 1);

          for (i = 1; i <= kmax - 1; i++)
            {
              t1 = GET(g1, 2 * i);
              t2 = GET(g1, 2 * i + 1);
              LET(g1, 2 * i) = t1 * MGET(tm, k, 2 * i) - t2 * MGET(tm, k, 2 * i
              LET(g1, 2 * i + 1) = t2 * MGET(tm, k, 2 * i) + t1 * MGET(tm, k, 2
            }

          fft_real(g1, 2 * kmax, 1);

          for (i = 0; i < Nmax; i++)
            {
              MLET(G, k, i) = - (GET(g1, i) + GET(lamka, k) * GET(eom, i)) * fla
            }
        }// END FOR k

//======================*L-Loop*=======================/
```

```
for (L = 1; L <= N1 * n; L++)
  {

    for (k = 0; k < Nr; k++)
      {
        for (i = 0; i < Nmax; i++)
          {
            LET(vv1, i) = MGET(prices_all, k, i);

            if (k > 0)
              {
                i0 = (int)ceil(GET(num, k) / h);
                if ((i - i0 > 0) && (i - i0 < Nmax - 2))
                  {
                    S0 = Strike1 * exp(GET(y, i) - GET(num, k));
                    Sl = GET(expy, i - i0 - 1);
                    Sm = GET(expy, i - i0);
                    Sr = GET(expy, i - i0 + 1);
                    // S0 is between Sm and Sr
                    pricel = MGET(prices_old, k - 1, i - i0 - 1);
                    pricem = MGET(prices_old, k - 1, i - i0);
                    pricer = MGET(prices_old, k - 1, i - i0 + 1);
                    //quadratic interpolation
                    A = pricel;
                    B = (pricem - pricel) / (Sm - Sl);
                    C = (pricer - A - B * (Sr - Sl)) / (Sr - Sl) / (Sr - S
                    //Price
                    LET(vv1, i) = GET(vv1, i) + MGET(lam1, k, k - 1) * (A
                  }
              }
            if (k < Nr - 1)
              {
                i0 = (int)ceil(GET(nup, k) / h);
                if ((i + i0 > 1) && (i + i0 < Nmax - 1))
                  {
                    S0 = Strike1 * exp(GET(y, i) + GET(nup, k));
                    Sl = GET(expy, i + i0 - 2);
                    Sm = GET(expy, i + i0 - 1);
                    Sr = GET(expy, i + i0);
                    // S0 is between Sm and Sr
                    pricel = MGET(prices_old, k + 1, i + i0 - 2);
```

```
            pricem = MGET(prices_old, k + 1, i + i0 - 1);
            pricer = MGET(prices_old, k + 1, i + i0);
            //quadratic interpolation
            A = pricel;
            B = (pricem - pricel) / (Sm - Sl);
            C = (pricer - A - B * (Sr - Sl)) / (Sr - Sl) / (Sr - S
            //Price
            LET(vv1, i) = GET(vv1, i) + MGET(lam1, k, k + 1) * (A
          }
      }

  }//end i


fft_real(vv1, 2 * kmax, -1);

LET(vv1, 0) = GET(vv1, 0) * MGET(tm, k, 0);
LET(vv1, 1) = GET(vv1, 1) * MGET(tm, k, 1);

for (i = 1; i <= kmax - 1; i++)
  {
    t1 = GET(vv1, 2 * i);
    t2 = GET(vv1, 2 * i + 1);
    LET(vv1, 2 * i) = t1 * MGET(tm, k, 2 * i) - t2 * MGET(tm, k, 2
    LET(vv1, 2 * i + 1) = t2 * MGET(tm, k, 2 * i) + t1 * MGET(tm,
  }

fft_real(vv1, 2 * kmax, 1);

for (i = 0; i < Nmax; i++)
  {
    LET(vv1, i) = GET(vv1, i) + MGET(G, k, i);
    if (GET(vv1, i) < 0)
      {
        LET(vv1, i) = 0.0;
      }
  }

fft_real(vv1, 2 * kmax, -1);

LET(vv1, 0) = GET(vv1, 0) * MGET(tp, k, 0);
```

```
    LET(vv1, 1) = GET(vv1, 1) * MGET(tp, k, 1);

    for (i = 1; i <= kmax - 1; i++)
      {
        t1 = GET(vv1, 2 * i);
        t2 = GET(vv1, 2 * i + 1);
        LET(vv1, 2 * i) = t1 * MGET(tp, k, 2 * i) - t2 * MGET(tp, k, 2
        LET(vv1, 2 * i + 1) = t2 * MGET(tp, k, 2 * i) + t1 * MGET(tp,
      }

    fft_real(vv1, 2 * kmax, 1);



    for (i = 0; i < Nmax; i++)
      {
        MLET(prices_new, k, i) = GET(vv1, i);
      }
  }// end loop in k



for (k = 0; k < Nr; k++)
  {
    for (i = 0; i < Nmax; i++)
      {
        MLET(prices_old, k, i) = MGET(prices_new, k, i);
      }
  }



for (k = 0; k < Nr; k++)
  {
    for (i = 0; i < Nmax; i++)
      {
        MLET(prices_all, k, i) = MGET(prices_new, k, i);
      }
  }
```

```
  }// ======================================== PREMIA_NULLTYPE L-LOOP ====

for (k = 0; k < Nr; k++)
  {
    for (i = 0; i < Nmax; i++)
      {
        MLET(prices_all, k, i) = MGET(prices_new, k, i) / GET(eom, i) + fl
      }
  }


pp = log(Spot / Strike1);

i = 2;

while ((GET(y, i) <= pp) && (i < Nmax))
  {
    i++;
  }

i--;
for (k = 0; k < Nr; k++)
  {
    //Price, quadratic interpolation
    Sl = GET(expy, i - 1);
    Sm = GET(expy, i);
    Sr = GET(expy, i + 1);
    // S0 is between Sm and Sr
    pricel = MGET(prices_all, k, i - 1);
    pricem = MGET(prices_all, k, i);
    pricer = MGET(prices_all, k, i + 1);

    //quadratic interpolation
    A = pricel;
    B = (pricem - pricel) / (Sm - Sl);
    C = (pricer - A - B * (Sr - Sl)) / (Sr - Sl) / (Sr - Sm);

    //Price
    pricem = A + B * (Spot - Sl) + C * (Spot - Sl) * (Spot - Sm);

    //Delta
```

```
          pricel = B + C * (2 * Spot - Sl - Sm);
          LET(ptprice, k) = GET(ptprice, k) + LET(coef, n) * pricem;
          LET(ptdelta, k) = GET(ptdelta, k) + LET(coef, n) * pricel;
        }// end for(k)
    }// end for(n)

  for (k = 0; k < Nr; k++)
    {
      if (GET(ptprice, k) < 0)
        {
          LET(ptprice, k) = 0.0;
          LET(ptdelta, k) = 0.0;
        }
    }

  /*Memory desallocation*/
  pnl_vect_free(&Lambda1);
//pnl_vect_free(&gam);
  pnl_vect_free(&y);
  pnl_vect_free(&expy);
  pnl_vect_free(&vv1);
  pnl_vect_free(&alin1);
  pnl_vect_free(&tp1);
  pnl_vect_free(&tm1);
  pnl_vect_free(&al1);
  pnl_vect_free(&eom);
  pnl_vect_free(&eomcut);
  pnl_vect_free(&g1);
  pnl_vect_free(&divid);
  pnl_mat_free(&prices_old);

  pnl_mat_free(&prices_new);

  pnl_mat_free(&prices_all);
//pnl_vect_free(&g1);

  pnl_mat_free(&G);
  pnl_mat_free(&lam1);
  pnl_mat_free(&tm);

  pnl_mat_free(&tp);
```

```
  pnl_vect_free(&a2_1);
//pnl_vect_free(&iter);
  pnl_vect_free(&lamka);
  pnl_vect_free(&coef);

  return OK;
}



#endif //PremiaCurrentVersion
```