Help

```
#include <stdlib.h>
#include "
href../../common/optype_h_src.pdfoptype.h"
#include "
href../../common/math/linsys_h_src.pdflinsys.h"
#include "
href../../common/math/jump_h_src.pdfjump.h"
#include "pnl/pnl_mathtools.h"
#include <
href../../common/math/cdo/cdo_math_h_src.pdfmath.h>
#include <stdio.h>
#include "
href../../common/error_msg_h_src.pdferror_msg.h"

#define PRECISION 1.0e-7
int Gaussian_data(double mu, double gamma2, DENSITY *g)
{


  g->par1 = mu;
  g->par2 = gamma2;
  g->Eu = exp(mu + 0.5 * gamma2) - 1.;
  g->zmin = mu - sqrt(2.0 * gamma2 * log(1.0 / (PRECISION * sqrt(2.0 * M_PI * ga
  g->zmax = mu + sqrt(2.0 * gamma2 * log(1.0 / (PRECISION * sqrt(2.0 * M_PI * ga
  return RETURNOK;
}

int Gaussian_vect(int l, int u, double h, DENSITY *g)
{
  int j;

  g->d = malloc((u - l) * sizeof(double));
  if (g->d == NULL) return MEMORY_ALLOCATION_FAILURE;
  memset(g->d, 0, (u - l)*sizeof(double));

  for (j = l; j < u; j++)(g->d)[j] = 1.0 / (sqrt(2.0 * M_PI * g->par2)) * exp(-S

  return RETURNOK;
}
```

```
/* ---------------------------- */

void freeDensity(DENSITY *g)
{
  if (g->d != NULL)
    {
      free(g->d);
      g->d = NULL;
    }
}

int set_parameter(double s, double K, double t, double r, double sigma, double d
{

  p->s = s;
  p->K = K;
  p->T = t;
  p->r = r;
  p->sigma = sigma;
  p->divid = divid;
  p->lambda = lambda;
  p->Eu = Eu;

  return RETURNOK;
}

int equation(PARAM p, EQ *eq)
{

  eq->dif = 0.5 * SQR(p.sigma);
  eq->adv = -(p.r - p.divid - p.lambda * p.Eu - eq->dif);
  eq->lin = p.r + p.lambda;

  return RETURNOK;
}

int set_mesh(EQ eq, int N, MESH *m)
{

  m->h = (m->xmax - m->xmin) / (double)(N - 1);
```

```
  if ((m->h * fabs(eq.adv)) <= eq.dif)
    {
      m->upwind_alphacoef = 0.5;
    }
  else
    {
      if (eq.adv > 0.) m->upwind_alphacoef = 0.0;
      else if (eq.adv <= 0.) m->upwind_alphacoef = 1.0;
    }

  return RETURNOK;
}

int set_weights_espl(double T, EQ eq, MESH *m, WEIGHT *w)
{
  m->k = SQR(m->h) / (eq.lin * SQR(m->h) + 2.*eq.dif + eq.adv * m->h * (1.0 - 2.
  m->M = (int)(T / m->k);

  w->p1 = eq.dif / (SQR(m->h)) + eq.adv / m->h * (1.0 - m->upwind_alphacoef);
  w->p2 = 2.*eq.dif / SQR(m->h) + eq.adv / m->h * (1.0 - 2.0 * m->upwind_alphaco
  w->p3 = eq.dif / (SQR(m->h)) - eq.adv / m->h * m->upwind_alphacoef;

  return RETURNOK;
}

int set_weights_impl(int M, double T, EQ eq, MESH *m, WEIGHT *w)
{

  m->k = T / ((double) M);
  m->M = M;

  w->p1 = eq.dif / (SQR(m->h)) + eq.adv / m->h * (1.0 - m->upwind_alphacoef);
  w->p2 = 2.*eq.dif / SQR(m->h) + eq.adv / m->h * (1.0 - 2.0 * m->upwind_alphaco
  w->p3 = eq.dif / (SQR(m->h)) - eq.adv / m->h * m->upwind_alphacoef;

  return RETURNOK;
}

int initgrid_1Dbis(PARAM p, DENSITY g, EQ eq, int N, MESH *m, IMESH *Im)
{
```

```
  double l;

  /* Space localization */
  l = 2.*p.sigma * sqrt(p.T) * sqrt(log(1.0 / PRECISION)) + fabs(eq.adv * p.T);
  m->xmin = log(p.s) - l + MIN(0.0, g.zmin);
  m->xmax = log(p.s) + l + MAX(0.0, g.zmax);

  set_mesh(eq, N, m);

  Im->min = (int) floor((double)(fabs(MIN(0.0, g.zmin)) / m->h));
  Im->max = N - (int) floor((double)(MAX(0.0, g.zmax) / m->h));
  Im->N = Im->min + (int) floor((double)(MAX(0.0, g.zmax) / m->h));
  m->N = N;

  m->Index = 0;
  while ((m->xmin + m->Index * m->h) < log(p.s)) m->Index++;
  m->Index--;

  return RETURNOK;
}

int set_boundaryAA(int bound, MESH m, PARAM p, IMESH Im, double *in, double *out
{
  int j, i, N, M;
  double ex_l, ex_u, gl, gu, hl, hu, *p1, *p2, *p3, *tnoto, *b2, *b1;

  PARAM pbs;
  EQ eqbs;
  MESH mbs;
  WEIGHT wbs;

  N = m.N;
  if (bound == 0)
    {
      for (j = 0; j < N; j++) out[j] = in[j];
    }
  else
    {

      set_parameter(p.s, p.K, p.T, p.r, p.sigma, p.divid, 0.0, 0.0, &pbs);
      equation(pbs, &eqbs);
```

```
M = m.N;
mbs.xmin = m.xmin;
mbs.xmax = m.xmax;
set_mesh(eqbs, N, &mbs);

set_weights_impl(M, p.T, eqbs, &mbs, &wbs);

b1 = malloc(N * sizeof(double));
if (b1 == NULL) return MEMORY_ALLOCATION_FAILURE;
memset(b1, 0, N * sizeof(double));
b2 = malloc(N * sizeof(double));
if (b2 == NULL) return MEMORY_ALLOCATION_FAILURE;
memset(b2, 0, N * sizeof(double));
tnoto = malloc(N * sizeof(double));
if (tnoto == NULL) return MEMORY_ALLOCATION_FAILURE;
memset(tnoto, 0, N * sizeof(double));
p1 = malloc(N * sizeof(double));
if (p1 == NULL) return MEMORY_ALLOCATION_FAILURE;
memset(p1, 0, N * sizeof(double));
p2 = malloc(N * sizeof(double));
if (p2 == NULL) return MEMORY_ALLOCATION_FAILURE;
memset(p2, 0, N * sizeof(double));
p3 = malloc(N * sizeof(double));
if (p3 == NULL) return MEMORY_ALLOCATION_FAILURE;
memset(p3, 0, N * sizeof(double));

for (j = 0; j < N; j++) b1[j] = in[j];

p1[0] = 0.;
p2[0] = 1.0;
p3[0] = 0.;
for (j = 1; j < N - 1; j++)
  {
    p1[j] = -mbs.k / 2.*wbs.p1;
    p2[j] = 1.0 + mbs.k / 2.*wbs.p2;
    p3[j] = -mbs.k / 2.*wbs.p3;
  }
p1[N - 1] = 0.;
p2[N - 1] = 1.0;
p3[N - 1] = 0.;
```

```
      for (i = 1; i <= M; i++)
        {
          tnoto[0] = in[0];
          for (j = 1; j < N - 1; j++) tnoto[j] = mbs.k / 2.*wbs.p1 * b1[j - 1] +
          tnoto[N - 1] = in[N - 1];
          /* tridiagonal system */
          tridiagsystem(p1, p2, p3, tnoto, b2, N);
          for (j = 0; j < N; j++) b1[j] = b2[j];
        }

      ex_l = exp(m.xmin + Im.min * m.h) - exp(m.xmin);
      ex_u = exp(m.xmax) - exp(m.xmin + Im.max * m.h);
      gl = (b2[Im.min] - b2[0]) / ex_l;
      gu = (b2[m.N - 1] - b2[Im.max]) / ex_u;
      hl = (exp(m.xmin + Im.min * m.h) * b2[0] - exp(m.xmin) * b2[Im.min]) / ex_
      hu = (exp(m.xmax) * b2[Im.max] - exp(m.xmin + Im.max * m.h) * b2[m.N - 1])

      for (j = 0; j < Im.min; j++) out[j] = gl * exp(m.xmin + j * m.h) + hl;
      for (j = Im.min; j < Im.max; j++) out[j] = 0.0;
      for (j = Im.max; j < m.N; j++) out[j] = gu * exp(m.xmin + j * m.h) + hu;

      free(b1);
      free(b2);
      free(tnoto);
      free(p1);
      free(p2);
      free(p3);
    }
  return RETURNOK;
}

/* ------------------------------------------------------------------------ */
int tridiagsystem(double *a, double *b, double *c, double *r, double *u, int n)
{
  int j;
  double bet, *gam;

  gam = malloc(n * sizeof(double));
  if (gam == NULL) return MEMORY_ALLOCATION_FAILURE;
  memset(gam, 0, n * sizeof(double));
```

```
  /* if (b[0] == 0.0) nerror("Error 1 in tridiag\ n",1); */
  u[0] = r[0] / (bet = b[0]);
  for (j = 1; j < n; j++)
    {
      gam[j] = c[j - 1] / bet;
      bet = b[j] - a[j] * gam[j];
      /* if (bet==0.0) nerror("Error 2 in tridiag\ n",1); */
      u[j] = (r[j] - a[j] * u[j - 1]) / bet;
    }
  for (j = (n - 2); j >= 0; j--) u[j] -= gam[j + 1] * u[j + 1];

  if (gam != NULL)
    {
      free(gam);
      gam = NULL;
    }
  return RETURNOK;

}
int tridiag_bis(double *a, double *b, double *c, double *r, double *u, unsigned
{
  int j;
  double bet, *gam;
  gam = malloc((n + 1) * sizeof(double));
  if (gam == NULL) return MEMORY_ALLOCATION_FAILURE;
  memset(gam, 0, (n + 1)*sizeof(double));

  u[1] = r[1] / (bet = b[1]);
  for (j = 2; j <= (int)n; j++)
    {
      gam[j] = c[j - 1] / bet;
      bet = b[j] - a[j] * gam[j];
      u[j] = (r[j] - a[j] * u[j - 1]) / bet;
    }
  for (j = (n - 1); j >= 1; j--) u[j] -= gam[j + 1] * u[j + 1];
  free(gam);
  return OK;
}
```

```
/* ------------------------------------------- */
double calc_int(int nodes, double *weight, double *pu)
{
  int i;
  double somma;

  somma = 0.0;
  for (i = 0; i < nodes; i++, pu++)
    somma += weight[i] * *pu;

  return somma;
}

int d1_intcomp(int nodes, double h, double *weight, double *jdensity, int int_me
{
  int i;

  switch (int_method)
    {
    case TR:
      for (i = 0; i < nodes - 1; i++)
        {
          int_trapez(h, &jdensity[i], &weight[i]);
        }
      break;
    case SIMP:
      for (i = 0; i < nodes - 2; i++)
        {
          int_simp(h, &jdensity[i], &weight[i]);
        }
      break;
    case NC4:
      for (i = 0; i < nodes - 3; i++)
        {
          int_nc4(h, &jdensity[i], &weight[i]);
        }
      break;
    case NC6:
      for (i = 0; i < nodes - 5; i++)
        {
          int_nc6(h, &jdensity[i], &weight[i]);
```

```
        }
      break;
    default:
      /*nerror("Error in d1_intcomp, varible int_method not define\ n",1);*/
      break;
    }

  return RETURNOK;
}

/* ------------------------------------------------- */
int int_trapez(REAL h, REAL *d, REAL *p)
{
  *p += h / 2.0 * *d;
  *(p + 1) += h / 2.0 * *(d + 1);

  return RETURNOK;
}

int int_simp(REAL h, REAL *d, REAL *p)
{
  *p += h / 6.0 * *d;
  *(p + 1) += 4.0 * h / 6.0 * *(d + 1);
  *(p + 2) += h / 6.0 * *(d + 2);

  return RETURNOK;
}

int int_nc4(REAL h, REAL *d, REAL *p)
{
  *p += 3.0 * h / 8.0 * *d;
  *(p + 1) += 9.0 * h / 8.0 * *(d + 1);
  *(p + 2) += 9.0 * h / 8.0 * *(d + 2);
  *(p + 3) += 3.0 * h / 8.0 * *(d + 3);

  return RETURNOK;
}

int int_nc6(REAL h, REAL *d, REAL *p)
{
  *p += 5.0 * 19.0 * h / 288.0 * *d;
```

```
  *(p + 1) += 5.0 * 75.0 * h / 288.0 * *(d + 1);
  *(p + 2) += 5.0 * 50.0 * h / 288.0 * *(d + 2);
  *(p + 3) += 5.0 * 50.0 * h / 288.0 * *(d + 3);
  *(p + 4) += 5.0 * 75.0 * h / 288.0 * *(d + 4);
  *(p + 5) += 5.0 * 19.0 * h / 288.0 * *(d + 5);

  return RETURNOK;
}

/* ------------------------------------------- */
/* Numerical Recipes (p. 507) FFT */
void dfour1(double *data, unsigned long nn, int isign)
/* Replace data[1..2*nn] by its Fourier transform, if isign is input as 1; or re
{
  unsigned long n, mmax, m, j, istep, i;
  double wtemp, wr, wpr, wpi, wi, theta;
  double tempr, tempi;

  n = nn << 1;
  j = 1;
  for (i = 1; i < n; i += 2)
    {
      if (j > i)
        {
          SWAP(data[j], data[i]);
          SWAP(data[j + 1], data[i + 1]);
        }
      m = n >> 1;
      while (m >= 2 && j > m)
        {
          j -= m;
          m >>= 1;
        }
      j += m;
    }
  mmax = 2;
  while (n > mmax)
    {
      istep = mmax << 1;
      theta = isign * (6.28318530717959 / mmax);
      wtemp = sin(0.5 * theta);
```

```
      wpr = -2.0 * wtemp * wtemp;
      wpi = sin(theta);
      wr = 1.0;
      wi = 0.0;
      for (m = 1; m < mmax; m += 2)
        {
          for (i = m; i <= n; i += istep)
            {
              j = i + mmax;
              tempr = wr * data[j] - wi * data[j + 1];
              tempi = wr * data[j + 1] + wi * data[j];
              data[j] = data[i] - tempr;
              data[j + 1] = data[i + 1] - tempi;
              data[i] += tempr;
              data[i + 1] += tempi;
            }
          wr = (wtemp = wr) * wpr - wi * wpi + wr;
          wi = wi * wpr + wtemp * wpi + wi;
        }
      mmax = istep;
    }

}


/* ------------------------------------------- */
/* Numerical Recipes (p. 543) - Convolution and Deconvolution using FFT */
void drealft(double *data, unsigned long n, int isign)
/* Calculates the Fourier trasform of a set of n real-valued data points. Replac
{
  unsigned long i, i1, i2, i3, i4, np3;
  double c1 = 0.5, c2, h1r, h1i, h2r, h2i;
  double wr, wi, wpr, wpi, wtemp, theta;

  theta = 3.141592653589793 / (double)(n >> 1);
  if (isign == 1)
    {
      c2 = -0.5;
      dfour1(data, n >> 1, 1);
    }
  else
```

```
    {
      c2 = 0.5;
      theta = -theta;
    }
  wtemp = sin(0.5 * theta);
  wpr = -2.0 * wtemp * wtemp;
  wpi = sin(theta);
  wr = 1.0 + wpr;
  wi = wpi;
  np3 = n + 3;
  for (i = 2; i <= (n >> 2); i++)
    {
      i4 = 1 + (i3 = np3 - (i2 = 1 + (i1 = i + i - 1)));
      h1r = c1 * (data[i1] + data[i3]);
      h1i = c1 * (data[i2] - data[i4]);
      h2r = -c2 * (data[i2] + data[i4]);
      h2i = c2 * (data[i1] - data[i3]);
      data[i1] = h1r + wr * h2r - wi * h2i;
      data[i2] = h1i + wr * h2i + wi * h2r;
      data[i3] = h1r - wr * h2r + wi * h2i;
      data[i4] = -h1i + wr * h2i + wi * h2r;
      wr = (wtemp = wr) * wpr - wi * wpi + wr;
      wi = wi * wpr + wtemp * wpi + wi;
    }
  if (isign == 1)
    {
      data[1] = (h1r = data[1]) + data[2];
      data[2] = h1r - data[2];
    }
  else
    {
      data[1] = c1 * ((h1r = data[1]) + data[2]);
      data[2] = c1 * (h1r - data[2]);
      dfour1(data, n >> 1, -1);
    }

}

/* ---------------------------------------------- */
/* Numerical Recipes (p. 511) */
/* Given two real input arrays data1[1..n] and data2[1..n], this routin calls fo
```

```
/* void dtwofft(double *data1, double *data2, double *fft1, double *fft2, unsign
 {
 unsigned long nn3,nn2,jj,j;
 double rep,rem,aip,aim;

 nn3=1+(nn2=2+n+n);
 for (j=1,jj=2;j<=n;j++,jj+=2) {
 fft1[jj-1]=data1[j];
 fft1[jj]=data2[j];
 }
 dfour1(fft1,n,1);
 fft2[1]=fft1[2];
 fft1[2]=fft2[2]=0.0;
 for (j=3;j<=n+1;j+=2) {
 rep=0.5*(fft1[j]+fft1[nn2-j]);
 rem=0.5*(fft1[j]-fft1[nn2-j]);
 aip=0.5*(fft1[j+1]+fft1[nn3-j]);
 aim=0.5*(fft1[j+1]-fft1[nn3-j]);
 fft1[j]=rep;
 fft1[j+1]=aim;
 fft1[nn2-j]=rep;
 fft1[nn3-j] = -aim;
 fft2[j]=aip;
 fft2[j+1] = -rem;
 fft2[nn2-j]=aip;
 fft2[nn3-j]=rem;
 }

 }
 */
/* ---------------------------------------------- */
/* Numerical Recipes (p. 546) - Correlation */
/*Computes the correlation of two real data set data1[], data2[] (including any
/* int dcorrel(double *data1, double *data2, unsigned long n, double *ans)
 {


 unsigned long no2,i,nap;
 double dum,*fft;

 nap = n<<1;
```

```
fft= malloc((size_t) ((nap+1)*sizeof(double)));
if (fft==NULL) return MEMORY_ALLOCATION_FAILURE;
memset(fft,0,(nap+1)*sizeof(double));

dtwofft(data1,data2,fft,ans,n);
no2=n>>1;
for (i=2;i<=n+2;i+=2) {
ans[i-1]=(fft[i-1]*(dum=ans[i-1])+fft[i]*ans[i])/no2;
ans[i]=(fft[i]*dum-fft[i-1]*ans[i])/no2;
}
ans[2]=ans[n+1];
drealft(ans,n,-1);
free(fft);

return OK;
}
*/
```