```
   Help
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <assert.h>

#include "pnl/pnl_integration.h"
#include "pnl/pnl_fft.h"
#include "pnl/pnl_finance.h"
#include "pnl/pnl_complex.h"
#include "levy_process.h"
#include "levy_diffusion.h"
#include "carr.h"

dcomplex ln_phi_BS(dcomplex u,double t,double sigma)
 {
   dcomplex psi=RCmul(-sigma*sigma*t*0.5,C_op_apib(Cmul(u,
    u),u));
   //printf( " **> %7.4f +i %7.4f {n",psi.r,psi.i);
   return psi;
}


int CarrMethod(double S0,
               double T,
               double K,
               double CallPut,
               double r,
               double divid,
               double sigma,
               void * Model,
               dcomplex (*ln_phi)(dcomplex u,double t,void
    * model),
               double *ptprice,
               double *ptdelta)
{
  int n;
  dcomplex dzeta,dzetaBS;
  double alpha=0.75;
  //taking account of dividends
```

```
int Nlimit =  2048;//2048;
//number of integral discretization steps
double logstrikestep = 0.01;
double k0 = log(K/S0)-(r-divid)*T;
double h = M_PI/Nlimit/logstrikestep; //integral discret
  ization step
double z,y;
double vn = 0;
dcomplex vn_minus_alpha_plus_uno = Complex(0,-(alpha+1));
dcomplex i_vn_plus_alpha = Complex(alpha,0);
double weight = 1./3; //Simpson's rule weights
dcomplex uno_plus_alpha_plus_ivn=Complex(1+alpha,vn);
//delta
z=0;y=0;
for(n=0; n<Nlimit; n++)
  {
    dzeta=Cadd(ln_phi(vn_minus_alpha_plus_uno,T,Model),
  Complex(0,-vn*k0));
    // printf("%7.4f + i %7.4f {n",dzeta.r,dzeta.i);
    dzetaBS= Cadd(ln_phi_BS(vn_minus_alpha_plus_uno,T,si
  gma),Complex(0,-vn*k0));
    dzeta  = Csub(Cexp(dzeta),Cexp(dzetaBS));
    dzeta  = Cdiv(dzeta,i_vn_plus_alpha);
    dzeta  = RCmul(weight,dzeta);
    //printf(">>%7.4f + i %7.4f {n",dzeta.r,dzeta.i);
    z+=dzeta.r;
    dzeta=Cdiv(dzeta,uno_plus_alpha_plus_ivn);
    y+=dzeta.r;
    //>> Update value
    vn += h;
    vn_minus_alpha_plus_uno.r+=h;
    i_vn_plus_alpha.i+=h;
    uno_plus_alpha_plus_ivn.i+=h;
    weight = (weight<1) ? 4./3 : 2./3; //Simpson's rule
  weights
    weight = (n==(Nlimit-2)) ?2./3. :weight;
  }
//Black-Scholes formula
pnl_cf_call_bs(S0,K,T,r,divid,sigma,ptprice,ptdelta);
S0 *= exp(-divid*T);
/*Call Case*/
```

```
  *ptprice += S0/(Nlimit*logstrikestep)*exp(-alpha*k0)*y;
  //*ptprice = y;
  *ptdelta += exp(-divid*T)/(Nlimit*logstrikestep)*exp(-alp
    ha*k0)*z;
  //Put Case via parity*/
  if (CallPut==2)
    {
      *ptprice =*ptprice-S0+K*exp(-r*T);
      *ptdelta =*ptdelta-exp(-divid*T);
    }
  //memory desallocation
  return OK;
}


int CarrMethod_VectStrike(PnlVect *K,
                          PnlVect * Price,
                          double S0,
                          double T,
                          double B,
                          double CallPut,
                          double r,
                          double divid,
                          double sigma,
                          void * Model,
                          dcomplex (*ln_phi)(dcomplex u,
    double t,void * model))

{
  int n;
  dcomplex dzeta,dzetaBS;
  double alpha=0.75;
  int Nlimit = 4*2048;//2048;
  //>> Should be even => use of real_fft
  //number of integral discretization steps
  double mone;//0.010;
  double Kstep=B*2/(Nlimit); // strike domain is (-B,B)
  double h  = M_2PI/(Nlimit*Kstep);
  //double B  = 0.5*(Nlimit)*Kstep; // strike domain is (-
    B,B)
  double vn = 0;
```

```
dcomplex vn_minus_alpha_plus_uno = Complex(0,-(alpha+1));
dcomplex i_vn_plus_alpha         = Complex(alpha,0);
dcomplex uno_plus_alpha_plus_ivn =Complex(1+alpha,vn);
PnlVectComplex * y = pnl_vect_complex_create(Nlimit);

// Should become output
pnl_vect_resize(K,Nlimit);
pnl_vect_resize(Price,Nlimit);

//delta
mone=1;
//printf("limit integration %7.4f {n",A);
for(n=0; n<Nlimit; n++)
  {
    dzeta   = Cadd(ln_phi(vn_minus_alpha_plus_uno,T,
  Model),Complex(0,vn*B));
    dzetaBS = Cadd(ln_phi_BS(vn_minus_alpha_plus_uno,T,si
  gma),Complex(0,vn*B));
    dzeta   = Csub(Cexp(dzeta),Cexp(dzetaBS));
    dzeta   = Cdiv(dzeta,i_vn_plus_alpha);
    dzeta   = Cdiv(dzeta,uno_plus_alpha_plus_ivn);
    //>> With Simson rules
    pnl_vect_complex_set(y,n,RCmul(3+mone-((n==0)?1:0),
  Conj(dzeta)));
    //>> Update value
    vn += h;
    vn_minus_alpha_plus_uno.r+=h;
    i_vn_plus_alpha.i+=h;
    uno_plus_alpha_plus_ivn.i+=h;
    mone*=-1;
  }
pnl_ifft_inplace(y);
for(n=0;n<Nlimit;n++)
  {
    LET(K,n)=exp(-B+n*Kstep+(r-divid)*T)*(S0);
    pnl_cf_call_bs(S0,GET(K,n),T,r,divid,sigma,&LET(
  Price,n),&vn);
    LET(Price,n)+=2./3* S0/(Kstep)*exp(alpha*(B-n*Kstep)-
  divid*T)*GET_REAL(y,n);
  }
if (CallPut==2)
```

```
    for(n=0;n<Nlimit;n++)
      LET(Price,n)-=S0*exp(-divid*T)+GET(K,n)*exp(-r*T);
  /*
  printf("Price K=  %7.4f  P= %7.4f {n",GET(K,Nlimit/2-5),
    GET(Price,Nlimit/2-5));
  printf("Price K=  %7.4f  P= %7.4f {n",GET(K,Nlimit/2-4),
    GET(Price,Nlimit/2-4));
  printf("Price K=  %7.4f  P= %7.4f {n",GET(K,Nlimit/2-3),
    GET(Price,Nlimit/2-3));
  printf("Price K=  %7.4f  P= %7.4f {n",GET(K,Nlimit/2-2),
    GET(Price,Nlimit/2-2));
  printf("Price K=  %7.4f  P= %7.4f {n",GET(K,Nlimit/2-1),
    GET(Price,Nlimit/2-1));
  printf("Price K=  %7.4f  P= %7.4f {n",GET(K,Nlimit/2+0),
    GET(Price,Nlimit/2+0));
  printf("Price K=  %7.4f  P= %7.4f {n",GET(K,Nlimit/2+1),
    GET(Price,Nlimit/2+1));
  printf("Price K=  %7.4f  P= %7.4f {n",GET(K,Nlimit/2+2),
    GET(Price,Nlimit/2+2));
  printf("Price K=  %7.4f  P= %7.4f {n",GET(K,Nlimit/2+3),
    GET(Price,Nlimit/2+3));
  printf("Price K=  %7.4f  P= %7.4f {n",GET(K,Nlimit/2+4),
    GET(Price,Nlimit/2+4));
  printf("Price K=  %7.4f  P= %7.4f {n",GET(K,Nlimit/2+5),
    GET(Price,Nlimit/2+5));
  printf("Price K=  %7.4f  P= %7.4f {n",GET(K,Nlimit/2+6),
    GET(Price,Nlimit/2+6));
  printf("Price K=  %7.4f  P= %7.4f {n",GET(K,Nlimit/2+7),
    GET(Price,Nlimit/2+7));
  printf("Price K=  %7.4f  P= %7.4f {n",GET(K,Nlimit/2+8),
    GET(Price,Nlimit/2+8));
  pnl_vect_free(&K);
  pnl_vect_free(&Price);
  */
  return OK;
}



int CarrMethod_Vanilla_option(Option_Eqd * opt,
                              double sigma,
                              Levy_process * Model)
{
  if(opt->product_type!=1)
    PNL_ERROR(" Carr method works only for european option
    !","carr.c ");
  return CarrMethod(opt->S0,opt->T,opt->K,opt->product,opt-
    >rate,opt->divid,sigma,Model,
                    &Levy_process_ln_characteristic_
    function_with_cast,
                    &(opt->price),&(opt->delta));
}


int CarrMethod_Vanilla_option_LD(Option_Eqd * opt,
                                 double sigma,
                                 Levy_diffusion * Model)
{
  if(opt->product_type!=1)
    PNL_ERROR(" Carr method works only for european option
    !","carr.c ");
  return CarrMethod(opt->S0,opt->T,opt->K,opt->product,opt-
    >rate,opt->divid,sigma,Model,
                    &Levy_diffusion_ln_characteristic_
    function_with_cast,&(opt->price),&(opt->delta));
}




int CarrMethod_onStrikeList(PnlVect *K,
                            PnlVect * Price,
                            double S0,
                            double T,
                            double CallPut,
                            double r,
                            double divid,
                            double sigma,
                            Levy_diffusion * Model)
```

```
{
  PnlVect * StrikeFFT,*PriceFFT;
  int n,error,ancestor,current,next;
  double delta;
  double strike_min = GET(K, 0);
  double strike_max = GET(K,K->size-1);
  //double nbr_strike = K->size;
  double strike_bnd = 2*MAX(log(strike_max/S0),fabs(log(
    strike_min/S0)));//0.25*log(strike_max/strike_min)/nbr_strike
    ;
  // 2 adjust heuristic parameter, to find four points
  // in fft in which all real strike value from K

  // Stored data for homogen grid in strike
  StrikeFFT=pnl_vect_create(0);
  PriceFFT=pnl_vect_create(0);

  error=CarrMethod_VectStrike(StrikeFFT,PriceFFT,
                              S0,T,strike_bnd,CallPut,r,div
    id,sigma,
                              Model,
                              &Levy_diffusion_ln_characteri
    stic_function_with_cast);
  ancestor=0;
  current=0;
  next=1;
  n=0;
  while(n<K->size)
    {
      if((GET(StrikeFFT,current)<=GET(K,n))&&(GET(Strike
    FFT,next)>GET(K,n)))
        {
          quadratic_interpolation(GET(PriceFFT,ancestor),
                                  GET(PriceFFT,current),
                                  GET(PriceFFT,next),
                                  GET(StrikeFFT,ancestor),
                                  GET(StrikeFFT,current),
                                  GET(StrikeFFT,next),
                                  GET(K,n),
                                  &LET(Price,n),
                                  &delta);
          n++;
        }
      else
        {
          ancestor=current;//not ++ for the first step
          current++;
          next++;
          if(next>StrikeFFT->size)
            PNL_ERROR(" Carr method domain size is too sm
    all for interpolation after FFT !","carr.c ");

        }
    }
  LET(Price,n)=GET(PriceFFT,PriceFFT->size);
  return error;
}
```

```
int CarrMethod_old_verison(double S0,
                           double T,
                           double K,
                           double CallPut,
                           double r,
                           double divid,
                           double sigma,
                           void * Model,
                           dcomplex (*ln_phi)(dcomplex u,
    double t,void * model),
                           double *ptprice,
                           double *ptdelta)
{
  int n;
  dcomplex dzeta,dzetaBS;
  double alpha=0.0;
  //taking account of dividends
  int Nlimit =  2048;
  //number of integral discretization steps
  double logstrikestep = 0.01;
  double k0 = log(K/(S0*exp(-divid*T)));
  double h = M_2PI/Nlimit/logstrikestep; //integral discret
    ization step
  double A = (Nlimit-1)*h; // integration domain is (-A/2,
    A/2)
  PnlVectComplex * z =pnl_vect_complex_create(Nlimit);
  PnlVectComplex * y =pnl_vect_complex_create(Nlimit);
  double vn = -A/2;
  dcomplex vn_minus_alpha_plus_uno = Complex(-A/2,-(alpha+1
    ));
```

```
dcomplex i_vn_plus_alpha = Complex(alpha,-A/2);
double weight = 1./3; //Simpson's rule weights
dcomplex uno_plus_alpha_plus_ivn=Complex(1+alpha,vn);
//delta
for(n=0; n<Nlimit; n++)
  {
    dzeta=   Cadd(ln_phi(vn_minus_alpha_plus_uno,T,Model)
  ,Complex(0,vn*(r*T-k0)));
    dzetaBS= Cadd(ln_phi_BS(vn_minus_alpha_plus_uno,T,si
  gma),Complex(0,vn*(r*T-k0)));
    dzeta  = Csub(Cexp(dzeta),Cexp(dzetaBS));
    dzeta  = Cdiv(dzeta,i_vn_plus_alpha);
    dzeta  = RCmul(weight,dzeta);
    pnl_vect_complex_set(z,n,dzeta);
    dzeta=Cdiv(dzeta,uno_plus_alpha_plus_ivn);
    pnl_vect_complex_set(y,n,dzeta);
    //>> Update value
    vn += h;
    vn_minus_alpha_plus_uno.r+=h;
    i_vn_plus_alpha.i+=h;
    uno_plus_alpha_plus_ivn.i+=h;
    weight = (weight<1) ? 4./3 : 2./3; //Simpson's rule
  weights
    weight = (n==(Nlimit-2)) ?2./3. :weight;
  }
//pnl_vect_complex_print(z);
pnl_fft_inplace(z);
pnl_fft_inplace(y);
//pnl_vect_complex_print(z);

//Black-Scholes formula
pnl_cf_call_bs(S0,K,T,r,divid,sigma,ptprice,ptdelta);
S0 *= exp(-divid*T);
/*Call Case*/
*ptprice += S0*A/M_2PI/(Nlimit-1)*exp(-alpha*k0)*GET_REA
  L(y,0);
*ptdelta += exp(-divid*T)*(A/M_2PI/(Nlimit-1)*exp(-alpha*
  k0)*GET_REAL(z,0));

//Put Case via parity*/
if (CallPut==2)
```

```
    {
      *ptprice =*ptprice-S0+K*exp(-r*T);
      *ptdelta =*ptdelta-exp(-divid*T);
    }
  //memory desallocation
  pnl_vect_complex_free(&z);
  pnl_vect_complex_free(&y);
  return OK;
}
```

References