Help

```
#include "
href../../../common/math/kirkby/ap_bermudan_proj_h_src.pdfap_bermudan_proj.h"
#include "
href../../../common/math/kirkby/model_proj_h_src.pdfmodel_proj.h"
#include "
href../../../common/math/kirkby/proj_integrand_h_src.pdfproj_integrand.h"
#include <pnl/pnl_fft.h>
#include <cmath>
using namespace std;

void get_grid_params(double alph, int N, double W, double S_0,
int* nbar_pt, double* dx_pt, double* a_pt, double* zmin_pt, double* xmin_pt, dou
{
int K = N / 2;
double dx = 2 * alph / (N - 1.);    // Initial desired grid spacing, will be adj
double a = 1. / dx;

// Center value grid around the relative strike, [lws - alph, lws + alph]
double lws = log(W / S_0);  // Will adjust grid to ensure that log(S_0 / S_0) =

// Now Adjust Grid Spacing to Align K and S_0 on grid
double dxtil = dx;  // Initialize grid spacing to the intial desired spacing of
int nbar = (int)floor(lws * a + K / 2);   // index of strike,  NOTE: this index

if (ABS(lws) < dxtil) {
dx = dxtil;
}
else if (lws < 0) {
dx = lws / (1. + nbar - K / 2.);
nbar = nbar + 1;
}
else if (lws > 0) {
dx = lws / (nbar - K / 2.);
}

// Set Parameters to Return
*nbar_pt = nbar;
*dx_pt = dx;
*a_pt = 1 / dx;  // Redefined after perturbing dx to align strike and S_0 on gri
```

```
*zmin_pt = (1. - K) * dx;   // Leftmost point for the density projection
*xmin_pt = (1. - K / 2.) * dx;   // leftmost gridpoint on backward induction val
*dw_pt = 2 * M_PI * a / N;
}

int price_bermudan_put_for_model(Model_proj* model, double S_0, double W, int M0
double* delta, int logN) {
double dx, a, zmin, xmin, dw, dt, alph;
int nbar, N;

dt = T / M0;  // Timestep for backward induction, ie uniform time increment betw
N = (int)pow(2., logN);  // Number of density basis elements,  HARDCODED, this c

PnlVectComplex* grand = pnl_vect_complex_create_from_zero(N);
alph = model->get_truncation_alpha(L1, T);
get_grid_params(alph, N, W, S_0, &nbar, &dx, &a, &zmin, &xmin, &dw);

linear_fourier_integrand(model, grand, dt, N, dw, a, zmin);

return price_bermudan_put(S_0, W, M0, L1, r, q, T, N, dx, xmin, nbar, grand, pri
}

int price_bermudan_put(double S_0, double W, int M0, int L1, double r, double q,
double T, int N, double dx, double xmin, int nbar,  PnlVectComplex* grand, doubl
{
/*
This version prices a Bermudan put independently of the model. The model depende
which incorporates the ChF for any given model.
*/

double dt = T / M0;  // Timestep for backward induction, ie uniform time increme
int K = N / 2;   // Number of points on grid for backward induction (half the nu
double a = 1 / dx;
//double lws = log(W / S_0);  // Will adjust grid to ensure that log(S_0 / S_0)

////////////////////////////
// Initialize Beta (Projection Coefficients)

pnl_fft_inplace(grand);
PnlVect* beta = pnl_vect_create_from_zero(N);
PnlVectComplex* toepM = pnl_vect_complex_create_from_zero(N);
```

```
double Cons2 = 24 * pow(a, 2) * exp(-r * dt) / N;

for (int i = 0; i < K; i++){
LET(beta, i) = Cons2 * pnl_vect_complex_get_real(grand, K - 1 - i);
}

LET(beta, K) = 0;

for (int i = K + 1; i < 2 * K; i++){
LET(beta, i) = Cons2 * pnl_vect_complex_get_real(grand, 3 * K - i - 1);
}

pnl_real_fft(beta, toepM);

////////////////////////////
/// Initialize coefficients(recursion proceeds backwards in time)
double Cons3 = 1. / 48;
double Cons4 = 1. / 12;

// Gaussian Quadudrature Constants
double q_plus = (1 + sqrt(3. /5.)) / 2.;
double q_minus = (1 - sqrt(3. / 5.)) / 2.;

double b3 = sqrt(15.);
double b4 = b3 / 10.;

// Payoff Constants
double varthet_01 = exp(.5 * dx) * (5 * cosh(b4 * dx) - b3 * sinh(b4 * dx) + 4)
double varthet_m10 = exp(-.5 * dx) * (5 * cosh(b4 * dx) + b3 * sinh(b4 * dx) + 4
double varthet_star = varthet_01 + varthet_m10;

// Value Coefficients  (NOTE: these are all initialized to twice the size, due t
PnlVect* Gs = pnl_vect_create_from_zero(2*K);   // Terminal Payoff Values
PnlVect* ThetM = pnl_vect_create_from_zero(2*K);  // Terminal Value Coefficients
PnlVect* Thet = pnl_vect_create_from_zero(2*K);  // Backward Induction Value Coe
PnlVect* Cont = pnl_vect_create_from_zero(2*K);  // Continuation Values

for (int i = 0; i < nbar; i++) {
double g = exp(xmin + dx * i) * S_0;
```

```
LET(ThetM, i) = W - varthet_star * g;
LET(Gs, i) = W - g;
}
LET(ThetM, nbar - 1) = W * (0.5 - varthet_m10);

//////////////
pnl_real_fft(ThetM, grand);
for (int i = 0; i < 2 * K; i++) {
LET_COMPLEX(grand, i) = Cmul(GET_COMPLEX(toepM, i), GET_COMPLEX(grand, i));
}
pnl_ifft_inplace(grand);

for (int i = 0; i < K; i++){
LET(Cont, i) = pnl_vect_complex_get_real(grand, i);
}

//////////////
// Main Backward Induction Loop
int kstr = nbar + 1;

double Ck1, Ck2, Ck3, Ck4, Gk2, Gk3, tmp1, tmp2, xstrs, xkstr;
double rho, zeta, zeta2, zeta3, zeta4, zeta_plus, zeta_minus, rho_plus, rho_minu
double dbar_1, dbar_0, d_0, d_1;

for (int m = M0 - 2; m >= 0; m--){

while (kstr > 0 && (GET(Cont, kstr) > GET(Gs, kstr))) {
kstr--;
}
if (kstr >= 1) {
xkstr = xmin + kstr * dx;

Ck1 = GET(Cont, kstr - 1);
Ck2 = GET(Cont, kstr);
Ck3 = GET(Cont, kstr + 1);

// Linear interp of payoff
Gk2 = GET(Gs, kstr);
Gk3 = GET(Gs, kstr + 1);

// Get Exercise point, xstr, by interpolation
```

```
tmp1 = Ck2 - Gk2; tmp2 = Ck3 - Gk3;
xstrs = ((xkstr + dx) * tmp1 - xkstr * tmp2) / (tmp1 - tmp2);
}
else { // Interpolation from the left boundary
xkstr = xmin;
kstr = 0; xstrs = xmin;
Ck2 = GET(Cont, kstr); Ck1 = Ck2;  Ck3 = GET(Cont, kstr + 1);
}

rho = xstrs - xkstr;
zeta = a * rho;
zeta2 = pow(zeta, 2); zeta3 = zeta * zeta2; zeta4 = zeta * zeta3;
zeta_plus = zeta * q_plus; zeta_minus = zeta * q_minus;
rho_plus = rho * q_plus; rho_minus = rho * q_minus;
ed1 = exp(rho_minus); ed2 = exp(rho / 2); ed3 = exp(rho_plus);

dbar_1 = zeta2 / 2.;
dbar_0 = zeta - dbar_1; // dbar_1 = zeta + .5 * ((zeta - 1) ^ 2 - 1);
d_0 = zeta * (5 * ((1 - zeta_minus) * ed1 + (1 - zeta_plus) * ed3) + 4 * (2 - ze
d_1 = exp(-dx) * zeta * (5 * (zeta_minus * ed1 + zeta_plus * ed3) + 4 * zeta * e

for (int i = 0; i < kstr; i++) {
LET(Thet, i) = GET(ThetM, i);
}

Ck4 = GET(Cont, kstr + 2);

LET(Thet, kstr) = W * (.5 + dbar_0) - S_0 * exp(xkstr) * (varthet_m10 + d_0)
+ zeta4 / 8 * (Ck1 - 2 * Ck2 + Ck3) + zeta3 / 3 * (Ck2 - Ck1)
+ zeta2 / 4 * (Ck1 + 2 * Ck2 - Ck3) - zeta * Ck2
- Ck1 / 24 + 5 / 12. * Ck2 + Ck3 / 8.;

LET(Thet, kstr+1) = W * dbar_1 - S_0 * exp(xkstr + dx) * d_1 + zeta4 / 8. * (-Ck
+ zeta3 / 6. * (3 * Ck2 - 4 * Ck3 + Ck4) - .5 * zeta2 * Ck2
+ Cons4 * (Ck2 + 10 * Ck3 + Ck4);

for (int i = kstr + 2; i < K - 1; i++) {
LET(Thet, i) = Cons4 * (GET(Cont, i-1) + 10 * GET(Cont, i) + GET(Cont, i+1));
}

LET(Thet, K-1) = Cons3 * (13 * GET(Cont, K-1) + 15 * GET(Cont, K - 2) - 5 * GET(
```

```
pnl_real_fft(Thet, grand);
for (int i = 0; i < 2 * K; i++) LET_COMPLEX(grand, i) = Cmul(GET_COMPLEX(toepM,
pnl_ifft_inplace(grand);

for (int i = 0; i < K; i++){
LET(Cont, i) = pnl_vect_complex_get_real(grand, i);
}
}

int nnot = K / 2; // location of S_0 on grid, shifted to right by one from MATLA
*price = GET(Cont, nnot - 1);
*delta = (GET(Cont, nnot - 1) - GET(Cont, nnot - 2)) / (S_0 * exp(xmin + (nnot -

pnl_vect_free(&ThetM);
pnl_vect_free(&Thet);
pnl_vect_free(&beta);
pnl_vect_free(&Cont);
pnl_vect_free(&Gs);

pnl_vect_complex_free(&grand);
pnl_vect_complex_free(&toepM);

return OK;
}

int Kirkby_PROJ_kou_amerput(double Spot, double sigma, double lambda, double lam
double r, double divid,
double T, double Strike,
int logN, int step, int L1,
double *ptprice, double *ptdelta)
{
Kou_Model_proj* model = new Kou_Model_proj(r, divid, sigma, lambda, P, lambdap,
int status = price_bermudan_put_for_model(model, Spot, Strike, step, L1, r, divi
delete model;
return status;
}

int Kirkby_PROJ_CGMY_amerput(double Spot, double C, double G, double M, double Y
double T, double Strike,
int logN, int step, int L1,
```

```
double *ptprice, double *ptdelta)
{
CGMY_Model_proj* model = new CGMY_Model_proj(r, divid, C, G, M, Y);
int status = price_bermudan_put_for_model(model, Spot, Strike, step, L1, r, divi
delete model;
return status;
}

int Kirkby_PROJ_NIG_amerput(double Spot, double Sigma, double Theta, double Kapp
double T, double Strike,
int logN, int step, int L1,
double *ptprice, double *ptdelta)
{
double delta = Sigma / sqrt(Kappa);
double beta = Theta / SQR(Sigma);
double alpha = sqrt(1 / (Kappa*SQR(Sigma)) + beta*beta);

NIG_Model_proj* model = new NIG_Model_proj(r, divid, alpha, beta, delta);
int status = price_bermudan_put_for_model(model, Spot, Strike, step, L1, r, divi
delete model;
return status;
}



//double price_bermudan_put_cgmy(double S_0, double W, int M0, int L1, double r,
// double T, double* price, double* delta, int logN)
//{
// double dx, a, zmin, xmin, dw, dt, alph;
// int nbar, N;
//
// dt = T / M0;  // Timestep for backward induction, ie uniform time increment b
// N = (int)pow(2, logN);  // Number of density basis elements,  HARDCODED, this
//
// PnlVectComplex* grand = pnl_vect_complex_create_from_zero(N);
//
// ////////////////////////////////////////
// // Code Specific to the CGMY Model
// params_cgmy p;
// Set_CGMY_params(&p, r, q, Y, M, G, C, dt);
// alph = getTruncationAlpha(T, L1, p.c2, p.c4);
```

```
// get_grid_params(alph, N, W, S_0, &nbar, &dx, &a, &zmin, &xmin, &dw);
//
// // Initialize Fourier Integrand
// linear_fourier_integrand_cgmy(p, grand, N, dw, a, zmin);
// ////////////////////////////////////////
//
// return price_bermudan_put( S_0,  W,  M0,  L1,  r,  q, T,  N,  dx,  xmin,  nba
//}
```