Help

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <ctype.h>
#include <stdarg.h>
#include "finance_tool_box.h"
#include "pnl/pnl_finance.h"
#include "pnl/pnl_mathtools.h"
#include "pnl/pnl_root.h"
#include "implied_bs.h"


Option_Eqd * option_eqd_create(int am_,int product_,int
    product_type_,
                               double S0_, double K_,
    double T_,double rebate_,double barrier_)
 {
  Option_Eqd * op = malloc(sizeof(Option_Eqd));
  op->am=am_;
  op->product=product_;
  op->product_type=product_type_;
  op->S0=S0_;
  op->K=K_;
  op->T=T_;
  op->rebate=rebate_;
  op->barrier=barrier_;
  op->t_start=0.0;
  op->price=0;
  op->delta=0;
  op->implied_vol=0;
  return op;

 }

Option_Eqd * option_eqd_create_forwardstart(int am_,int
    product_,int product_type_,
                                            double S0_,
    double K_, double T_,
                                            double t_start_
```

```
    ,double rebate_,double barrier_)
 {
  Option_Eqd * op = malloc(sizeof(Option_Eqd));
  op->am=am_;
  op->product=product_;
  op->product_type=product_type_;
  op->S0=S0_;
  op->K=K_;
  op->T=T_;
  op->rebate=rebate_;
  op->barrier=barrier_;
  op->t_start=t_start_;
  op->price=0;
  op->delta=0;
  op->implied_vol=0;
  return op;

 }

void option_eqd_set_rate(Option_Eqd * opt,double rate_,
    double divid_)
{
  opt->rate=rate_;
  opt->divid=divid_;
}


double option_eqd_forward_price(Option_Eqd * op)
{
  return pnl_forward_price(op->S0,op->rate,op->divid,op->T)
    ;
}

int option_eqd_compute_implied_vol(Option_Eqd * op)
{
  double  bond=exp(-op->rate*(op->T));
  if(op->product_type==1)
      return op->implied_vol=pnl_bs_impli_implicit_vol (fab
    s(op->product-2),op->price,bond,option_eqd_forward_price(
    op),op->K,op->T);
  if(op->product_type==7)
```

```
    {
      double actu =exp(op->rate*op->t_start);
      double forward =pnl_forward_price(1.0,op->rate,op->
    divid,op->T-op->t_start);
      bond/=actu;
      return op->implied_vol=pnl_bs_impli_implicit_vol (fab
    s(op->product-2),op->price*actu,bond,forward,
                                                    op->K,op-
    >T-op->t_start);
    }
  return 100;
}

double init_cond(const double x,
                 const double S0,
                 const double K0,
                 const int product)
{
  double S=S0,K=K0;
  S*=exp(x);
  switch(product){
  case 1: return (S-K > 0) ? (S-K) : 0.0; // Call
  case 2: return (K-S > 0) ? (K-S) : 0.0; // Put
  case 3: return  S-K; // forward
  default: PNL_ERROR("Invalid product number","
    finance_tool_box/init_cond");
  }
  /* just to avoid a warning */
  return 0;
}

double init_cond_with_dupire(const double x,
                             const double S0,
                             const double K0,
                             const int dupire,
                             const int product)
{
  double S=S0,K=K0;
  if(dupire==0)
    S*=exp(x);
  else
```

```
    K*=exp(x);
  switch(product){
  case 1: return (S-K > 0) ? (S-K) : 0.0; // Call
  case 2: return (K-S > 0) ? (K-S) : 0.0; // Put
  case 3: return  S-K; // forward
  default: PNL_ERROR("Invalid product number","
    finance_tool_box/init_cond");
  }
  /* just to avoid a warning */
  return 0;
}

double bound_cond(const double x,const double S0,const
    double K,
                  const double rebate,const double barrier,
    const double ttm,const double r,const double div,
                  const int product,const int product_type)
{
  switch(product_type){
  case 1: return exp(-r*ttm)*init_cond(x+(r-div)*ttm,S0,K,
    product); // European
  case 2: return (exp(x)*S0>barrier) ? rebate : exp(-r*ttm)
    *init_cond(x+(r-div)*ttm,S0,K,product); // Up-and-Out
  case 3: return (exp(x)*S0>barrier) ? exp(-r*ttm)*init_
    cond(x+(r-div)*ttm,S0,K,product): rebate; // Down-and-Out
  case 4: return rebate; // Double barrier
  default: PNL_ERROR("Invalid option_eqd type","
    finance_tool_box/bound_cond");
  }
  /* just to avoid a warning */
  return 0;
}

double option_eqd_init_cond(const Option_Eqd * Op,const
    double x)
{return init_cond(x,Op->S0,Op->K,Op->product);}

double option_eqd_bound_cond(const Option_Eqd * Op,const
    double x,double ttm)
{
  return bound_cond(x,Op->S0,Op->K,Op->rebate,Op->barrier,
```

```
    ttm,Op->rate,Op->divid,Op->product,Op->product_type);
}


double Double_Primitive_Call_Put(const double K, const
    double S0, const double x,const int is_call)
{
  return (x<0)?(0.5*x*x-exp(x)+x+1):0;
}
//if(is_call)
// return (x>log(K/S0))?(S0*exp(x)-K*x*x):0;
//return (x<log(K/S0))?(K*x*x-S0*exp(x)):0;


double Compute_Projection_U0(const double K,const double S0
    ,const double x,const double h)
{
  // K not here ...
  double res=Double_Primitive_Call_Put(K,S0,x-h,0);
  res+=Double_Primitive_Call_Put(K,S0,x+h,0);
  res-= 2*Double_Primitive_Call_Put(K,S0,x,0);
  return K/h*res;
}


List_Option_Eqd * list_option_eqd_create(int am_,double S0_
    )
 {
  List_Option_Eqd * op = malloc(sizeof(List_Option_Eqd));
  op->am=am_;
  op->product_type=1;
  op->S0=S0_;
  op->rebate=0;
  op->nb_maturity=0;
  op->nb_options=0;
  op->K=pnl_vect_create(0);
  op->T=pnl_vect_create(0);
  op->t_start=pnl_vect_create(0);

  op->rate=0.0;
  op->divid=0.0;
```

```
  op->product=pnl_vect_int_create_from_int(0,0);
  op->index_maturity=pnl_vect_int_create(0);
  op->price=pnl_vect_create_from_zero(0);
  op->implied_vol=pnl_vect_create_from_zero(0);
  return op;
 }


List_Option_Eqd * list_option_eqd_create_with_data(int am_,
    double S0_,PnlVectInt * product_, PnlVectInt * index_matu, PnlV
    ect * Matu,PnlVect * Strike)
 {
  List_Option_Eqd * op = malloc(sizeof(List_Option_Eqd));
  op->am=am_;
  op->product_type=1;
  op->S0=S0_;
  op->rebate=0;
  op->nb_maturity=Matu->size;
  op->nb_options=Strike->size;
  op->K=pnl_vect_create(0);
  op->T=pnl_vect_create(0);
  op->t_start=pnl_vect_create(0);

  op->rate=0.0;
  op->divid=0.0;

  op->product=pnl_vect_int_create(0);
  op->index_maturity=pnl_vect_int_create(0);


  if((op->nb_options!=Matu->size)||(op->nb_options!=produc
    t_->size))
     PNL_ERROR ("size of list option_eqds are not consisten
    t ", "list_option_eqd_create");
  *(op->K)=pnl_vect_wrap_subvect(Strike,0,op->nb_options);
  *(op->T)=pnl_vect_wrap_subvect(Matu,0,op->nb_maturity);
  *(op->product)=pnl_vect_int_wrap_subvect(product_,0,op->
    nb_options);
  *(op->index_maturity)=pnl_vect_int_wrap_subvect(index_
    matu,0,op->nb_maturity);
```

```
  op->price=pnl_vect_create(op->nb_options);
  op->implied_vol=pnl_vect_create(op->nb_options);
  return op;
 }

List_Option_Eqd * list_option_eqd_create_forwardstart_with_
    data(int am_,double S0_,PnlVectInt * product_, PnlVectInt *
     index_matu, PnlVect * Matu,PnlVect *Start_Date,PnlVect *
    Strike)
 {
  List_Option_Eqd * op = malloc(sizeof(List_Option_Eqd));
  op->am=am_;
  op->product_type=7;
  op->S0=S0_;
  op->rebate=0;
  op->nb_maturity=Matu->size;
  op->nb_options=Strike->size;
  op->K=pnl_vect_create(0);
  op->T=pnl_vect_create(0);
  op->t_start=pnl_vect_create(0);

  op->rate=0.0;
  op->divid=0.0;

  op->product=pnl_vect_int_create(0);
  op->index_maturity=pnl_vect_int_create(0);


  if((op->nb_options!=Matu->size)||(op->nb_options!=produc
    t_->size))
     PNL_ERROR ("size of list option_eqds are not consisten
    t ", "list_option_eqd_create");
  *(op->K)=pnl_vect_wrap_subvect(Strike,0,op->nb_options);
  *(op->T)=pnl_vect_wrap_subvect(Matu,0,op->nb_maturity);
  *(op->t_start)=pnl_vect_wrap_subvect(Start_Date,0,op->nb_
    maturity);
  *(op->product)=pnl_vect_int_wrap_subvect(product_,0,op->
    nb_options);
  *(op->index_maturity)=pnl_vect_int_wrap_subvect(index_
    matu,0,op->nb_maturity);
  op->price=pnl_vect_create(op->nb_options);
```

```
  op->implied_vol=pnl_vect_create(op->nb_options);
  return op;
 }

void list_option_eqd_set_rate(List_Option_Eqd * lopt,
    double rate_,double divid_)
{
  lopt->rate=rate_;
  lopt->divid=divid_;
}



List_Option_Eqd * list_option_eqd_copy(const List_Option_
    Eqd * op_in)
 {
  List_Option_Eqd * op_out = malloc(sizeof(List_Option_Eqd)
    );
  op_out->am=op_in->am;
  op_out->product_type=op_in->product_type;
  op_out->S0=op_in->S0;
  op_out->rebate=op_in->rebate;
  op_out->nb_maturity=op_in->nb_maturity;
  op_out->nb_options=op_in->nb_options;

  op_out->K=pnl_vect_create(0);
  op_out->T=pnl_vect_create(0);
  op_out->t_start=pnl_vect_create(0);
  op_out->product=pnl_vect_int_create(0);
  op_out->index_maturity=pnl_vect_int_create(0);
  op_out->rate=op_in->rate;
  op_out->divid=op_in->divid;

  *(op_out->K)=pnl_vect_wrap_subvect(op_in->K,0,op_in->nb_
    options);
  *(op_out->product)=pnl_vect_int_wrap_subvect(op_in->prod
    uct,0,op_in->nb_options);

  *(op_out->T)=pnl_vect_wrap_subvect(op_in->T,0,op_in->nb_
    maturity);
  if (op_out->product_type==7)
```

```
    *(op_out->t_start)=pnl_vect_wrap_subvect(op_in->t_star
    t,0,op_in->T->size);
  *(op_out->index_maturity)=pnl_vect_int_wrap_subvect(op_
    in->index_maturity,0,op_in->nb_maturity);
  op_out->price=pnl_vect_create(op_in->nb_options);
  op_out->implied_vol=pnl_vect_create(op_in->nb_options);
  return op_out;
 }


void list_option_eqd_free(List_Option_Eqd ** op)
{
  if (*op != NULL)
    {
      pnl_vect_free(&(*op)->K);
      pnl_vect_free(&(*op)->T);
      pnl_vect_free(&(*op)->t_start);
      pnl_vect_free(&(*op)->price);
      pnl_vect_free(&(*op)->implied_vol);
      pnl_vect_int_free(&(*op)->product);
      pnl_vect_int_free(&(*op)->index_maturity);
      free(*op);
      *op=NULL;
    }
}

Option_Eqd list_option_eqd_get_value(List_Option_Eqd * lop
    t,int it,int k)
{
  Option_Eqd op;
  op.am=lopt->am;
  op.product=pnl_vect_int_get(lopt->product,0);
  op.product_type=1;
  op.S0=lopt->S0;
  op.K=GET(lopt->K,k);
  op.T=GET(lopt->T,it);
  op.t_start=(op.product_type==7)?GET(lopt->t_start,it):0;
  op.rebate=0.0;
  op.barrier=0.0;
  op.price=GET(lopt->price,k);
  op.delta=0.0;
```

```
  op.implied_vol=GET(lopt->implied_vol,k);
  op.rate=lopt->rate;
  op.divid=lopt->divid;
  return op;
}

void list_option_eqd_readmarketdata(List_Option_Eqd *op,
    const char * file)
{
  /*File variable of the code*/
  int m,n, etat;
  double old_matu,matu;
  char car,prev = '{0', empty=1;
  FILE *Entrees = fopen( file, "r");
  if( Entrees == NULL )
    {PNL_ERROR("Cannot open file", "list_option_eqd_readmar
    ketdata");}
  /* first pass to determine dimensions */
  m = 0; n = 1;
  etat=0;
  while((car=fgetc(Entrees))!='{n')
    {
      if (isdigit(car) || car == '-' || car == '.')
        {
          empty = 0;
          if (prev == ' ' || prev == '{t' ) ++n;
        }
      prev = car;
    }
  /*if (!empty && car =='{n' && isdigit(prev)) ++n; */
  if (!empty) ++m;
  empty = 1;
  while((car=fgetc(Entrees))!= EOF)
    {
      if( car=='{n' )
        {
          if (!empty) { ++m; empty = 1;}
          else break;
        }
      else if (empty && isdigit(car)) empty=0;
    }
```

```
rewind(Entrees);
if (m==0 || (n<3) || (n>4)) // With or without implied volatility
  {
    PNL_ERROR ("No data found in input file",  "list_
  option_eqd_readmarketdata");
  }
op->nb_options=m;
op->nb_maturity=0;
pnl_vect_resize(op->K,op->nb_options);
pnl_vect_resize(op->T,op->nb_options);
pnl_vect_resize(op->price,op->nb_options);
pnl_vect_resize(op->implied_vol,op->nb_options);
pnl_vect_int_resize(op->index_maturity,op->nb_options);
pnl_vect_int_resize(op->product,op->nb_options);
pnl_vect_int_set_int(op->product,1);
m=0;
old_matu=0.0;
if(n==4)
  while(m<op->nb_options)
    {
      etat+=fscanf(Entrees, "%lf %lf %lf %lf", &matu,&LET
  (op->K,m),&LET(op->price,m),&LET(op->implied_vol,m));
      if(old_matu!=matu)
        {
          pnl_vect_int_set(op->index_maturity,op->nb_matu
  rity,m);
          LET(op->T,op->nb_maturity)=matu;
          old_matu=matu;
          op->nb_maturity++;
        }
      m++;
    }
if(n==3)
  while(m<op->nb_options)
    {
      etat+=fscanf(Entrees, "%lf %lf %lf ", &LET(op->T,m)
  ,&LET(op->K,m),&LET(op->price,m));
      if(old_matu!=matu)
        {
          pnl_vect_int_set(op->index_maturity,op->nb_matu
  rity,m);
```

```
            LET(op->T,op->nb_maturity)=matu;
            old_matu=matu;
            op->nb_maturity++;
          }
        m++;
      }
  pnl_vect_resize(op->T,op->nb_maturity);
  pnl_vect_int_resize(op->index_maturity,op->nb_maturity);
  etat+=fclose(Entrees);
}


typedef struct Pnl_Data_Vol_Impli_BS{
  int is_call;
  double Price, Bond, Forward, Strike, Maturity;
}Pnl_Data_Vol_Impli_BS;

static void pnl_bs_impli_increment_call_put_Type(double x,
    double * fx,double * dfx,const Pnl_Data_Vol_Impli_BS * Data)
{
  *fx = Data->Price - pnl_bs_impli_call_put (Data->is_call,
     x, Data->Bond, Data->Forward,Data->Strike, Data->Maturit
    y);
  *dfx = -1.*pnl_bs_impli_vega(x,Data->Bond, Data->Forward,
    Data->Strike, Data->Maturity);
}

static void pnl_bs_impli_increment_call_put(double x,
    double * fx,double * dfx,void* Data)
{ pnl_bs_impli_increment_call_put_Type(x,fx,dfx,Data);}


int list_option_eqd_compute_implied_vol_todaystart(List_
    Option_Eqd * op)
{
  int j,k,last;
  int error=0;
  Pnl_Data_Vol_Impli_BS *data;
  PnlFuncDFunc func;
  data=malloc(sizeof(Pnl_Data_Vol_Impli_BS));
  func.function = pnl_bs_impli_increment_call_put;
```

```
func.params = data;
for(j=0;j<op->nb_maturity;j++)
  {
    data->Maturity=GET(op->T,j);
    data->Bond=exp(-op->rate*data->Maturity);
    data->Forward=op->S0*exp((op->rate-op->divid)*data->
  Maturity);
    k=pnl_vect_int_get(op->index_maturity,j);
    last=(j<op->nb_maturity-1)?pnl_vect_int_get(op->ind
  ex_maturity,j+1):op->nb_options;
    while(k<last)
      {
        data->Price=GET(op->price,k);
        data->is_call=fabs(pnl_vect_int_get(op->product,
  k)-2);
        data->Strike=GET(op->K,k);
        if(data->is_call)
          {
            if (data->Price <= MAX (data->Bond*data->Forw
  ard - data->Bond*data->Strike, 0.0))
              LET(op->implied_vol,k)=-1.0;

          else
            if (data->Price >= data->Bond*data->Forward)
              LET(op->implied_vol,k)=-1.0;
          }
        else
          {
            if (data->Price <= data->Bond* MAX (data->
  Strike - data->Forward, 0.0))
              LET(op->implied_vol,k)=-1.0;
            else
              if (data->Price >= data->Bond* data->Strike
  )
                LET(op->implied_vol,k)=-1.0;
          }
        error+=pnl_find_root(&func,0.001,10.0,0.0001,20,&
  (LET(op->implied_vol,k)));
        k++;
      }
  }
```

```
  free(data);
  return error;
}

int list_option_eqd_compute_implied_vol_forwardstart(List_
    Option_Eqd * op)
{
  int j,k,last;
  int error=0;
  double actu;
  Pnl_Data_Vol_Impli_BS *data;
  PnlFuncDFunc func;
  data=malloc(sizeof(Pnl_Data_Vol_Impli_BS));
  func.function = pnl_bs_impli_increment_call_put;
  func.params = data;
  for(j=0;j<op->nb_maturity;j++)
    {
      data->Maturity=GET(op->T,j)-GET(op->t_start,j);
      data->Bond=exp(-op->rate*data->Maturity);
      data->Forward=exp((op->rate-op->divid)*data->Maturit
    y);
      k=pnl_vect_int_get(op->index_maturity,j);
      actu=exp(op->rate*GET(op->t_start,j));
      last=(j<op->nb_maturity-1)?pnl_vect_int_get(op->ind
    ex_maturity,j+1):op->nb_options;
      while(k<last)
        {
          data->Price=GET(op->price,k)*actu;
          data->is_call=fabs(pnl_vect_int_get(op->product,
    k)-2);
          data->Strike=GET(op->K,k);
          if(data->is_call)
            {
              if (data->Price <= MAX (data->Bond*data->Forw
    ard - data->Bond*data->Strike, 0.0))
                LET(op->implied_vol,k)=-1.0;

            else
              if (data->Price >= data->Bond*data->Forward)
                LET(op->implied_vol,k)=-1.0;
            }
```

```
          else
            {
              if (data->Price <= data->Bond* MAX (data->
    Strike - data->Forward, 0.0))
                LET(op->implied_vol,k)=-1.0;
              else
                if (data->Price >= data->Bond* data->Strike
    )
                  LET(op->implied_vol,k)=-1.0;
            }
          error+=pnl_find_root(&func,0.001,10.0,0.0001,20,&
    (LET(op->implied_vol,k)));
          k++;
        }
    }
  free(data);
  return error;
}
int list_option_eqd_compute_implied_vol(List_Option_Eqd *
    op)
{
  if((op->product_type==1)&&(!op->am))
    return list_option_eqd_compute_implied_vol_todaystart(
    op);
  if((op->product_type==7)&&(!op->am))
    return list_option_eqd_compute_implied_vol_forwardsta
    rt(op);
  return 100;
}


void list_option_eqd_fprint(FILE *fic,List_Option_Eqd *op)
{
  int j,k,last;
  for(j=0;j<op->nb_maturity;j++)
    {
      k=pnl_vect_int_get(op->index_maturity,j);
      last=(j<op->nb_maturity-1)?pnl_vect_int_get(op->ind
    ex_maturity,j+1):op->nb_options;
      while(k<last)
        {
```

```
          fprintf(fic,"%lf   %lf   %lf   %lf {n",GET(op->T,
    j),log(GET(op->K,k)/op->S0)*100
                ,GET(op->price,k),GET(op->implied_vol,k))
    ;
          k++;
        }
    }
}

void list_option_eqd_print(List_Option_Eqd *op)
{
  list_option_eqd_fprint(stdout,op);
}

void list_option_eqd_print_nsp(List_Option_Eqd *op)
{
  int i,j,k,last;
  PnlVect V;
  for(j=0;j<op->nb_maturity;j++)
    {
      printf(" S = %f {n",op->S0 );
      printf(" maturity = %f {n",GET(op->T,j) );
      k=pnl_vect_int_get(op->index_maturity,j);
      last=(j<op->nb_maturity-1)?pnl_vect_int_get(op->ind
    ex_maturity,j+1):op->nb_options;
      V=pnl_vect_wrap_subvect_with_last(op->K,k,last-1);
      printf(" log strike / S0 = ");
      printf("[ ");
      for(i=0;i<(&V)->size;i++)
        {
          printf("%7.4f",log(GET(&V,i)/op->S0)*100);
          printf("; ");
        }
      printf(" ]; {n");
      V=pnl_vect_wrap_subvect_with_last(op->price,k,last-1)
    ;
      printf(" price = ");
      pnl_vect_print_nsp(&V);
      V=pnl_vect_wrap_subvect_with_last(op->implied_vol,k,
    last-1);
      printf(" vol = ");
```

```
      pnl_vect_print_nsp(&V);
    }
}


void list_option_eqd_savemarketdata(List_Option_Eqd *op,
    const char * file)
{
  /*File variable of the code*/
  FILE *Entrees = fopen( file, "w");
  if( Entrees == NULL )
    {PNL_ERROR("Cannot open file", "list_option_eqd_savemar
    ketdata");}
  list_option_eqd_fprint(Entrees,op);
  fclose(Entrees);
}
```

References