Help

```
#include "pnl/pnl_vector.h"
#include "pnl/pnl_finance.h"
#include "math/equity_pricer/finance_tool_box.h"
#include "math/equity_pricer/implied_bs.h"
#include  "varswap3d_std.h"

////////////////////////////////////////////////////////
// Stochastic Variance Swap Model
////////////////////////////////////////////////////////

/**
 * free stochastic variance swap models
 *
 * @param pointer on VARSWAP3D_MOD
 */
void svs_model_free(VARSWAP3D_MOD **M)
{
  if (*M != NULL)
    {
      pnl_vect_free(&((*M)->Beta));
      pnl_vect_free(&((*M)->MeanReversion));
      pnl_vect_free(&((*M)->SqrtMeanReversion));
      free(*M);
      *M = NULL;
    }
}

/**
 * initilisation of some coefficients in svs models
 *
 * @param pointer on VARSWAP3D_MOD
 */
void svs_model_initialise(VARSWAP3D_MOD *M)
{
  int i;
  M->SqrtMeanReversion = pnl_vect_create(M->MeanReversion->size);
  for (i = 0; i < M->MeanReversion->size; i++)
    LET(M->SqrtMeanReversion, i) = M_SQRT2 * sqrt(GET(M->MeanReversion, i));
  M->Sum_Beta = pnl_vect_sum(M->Beta);
```

```
  M->V0_sqr = M->V0 * M->V0;
}

/**
 * Compute volatility for time dependant function
 * @param pointer on VARSWAP3D_MOD
 * @param T time
 */
void svs_sigma_time(VARSWAP3D_MOD *M, double T)
{
  M->V0_time = M->V0;
  M->V0_sqr = M->V0_time * M->V0_time;
}


VARSWAP3D_MOD *svs_model_create_from_Model(VARSWAP3D *Model)
{
  VARSWAP3D_MOD *M = malloc(sizeof(VARSWAP3D_MOD));
  M->S0 = Model->S0.Val.V_PDOUBLE;
  M->V0_time = M->V0;
  M->Beta = pnl_vect_copy(Model->Beta.Val.V_PNLVECT);
  M->MeanReversion = pnl_vect_copy(Model->MeanReversion.Val.V_PNLVECT);
  M->V0 = Model->V0.Val.V_PDOUBLE;
  M->V0_time = M->V0;
  M->Rho = Model->Rho.Val.V_DOUBLE;
  M->Nb_factor = M->Beta->size;
  M->Divid = log(1. + Model->Divid.Val.V_DOUBLE / 100.);
  M->R = log(1. + Model->R.Val.V_DOUBLE / 100.);
  M->T = -Model->T.Val.V_DATE;
  svs_model_initialise(M);
  return M;
}

void svs_model_initialise_from_Option(VARSWAP3D_MOD *M, TYPEOPT *ptOpt)
{
  M->is_call = ((ptOpt->PayOff.Val.V_NUMFUNC_1)->Compute == &Call) ? 1 : 0;
  M->Strike = (ptOpt->PayOff.Val.V_NUMFUNC_1)->Par[0].Val.V_DOUBLE;
  M->T += ptOpt->Maturity.Val.V_DATE;
  M->F0 = pnl_forward_price(M->S0, M->R, M->Divid, M->T);
  M->Bond = exp(-M->R * M->T);
}
```

```
int MOD_OPT(ChkMix)(Option *Opt, Model *Mod)
{
  TYPEOPT *ptOpt = (TYPEOPT *)(Opt->TypeOpt);
  TYPEMOD *ptMod = (TYPEMOD *)(Mod->TypeModel);
  int status = OK;

  if ((ptOpt->Maturity.Val.V_DATE) <= (ptMod->T.Val.V_DATE))
    {
      Fprintf(TOSCREENANDFILE, "Current date greater than maturity!\ n");
      status += 1;
    };

  return status;
}
extern PricingMethod MET(FD_AchdouPommier);
extern PricingMethod MET(MC_VARSWAP3D);
PricingMethod *MOD_OPT(methods)[] =
{
  &MET(FD_AchdouPommier),
  &MET(MC_VARSWAP3D),
  NULL
};

DynamicTest *MOD_OPT(tests)[] =
{
  NULL
};

Pricing MOD_OPT(pricing) =
{
  ID_MOD_OPT,
  MOD_OPT(methods),
  MOD_OPT(tests),
  MOD_OPT(ChkMix)
};
```