Help

```
/*-------------------------------------*/
/*   BERMUDAN SWAPTION PRICER          */
/* see LMM_bermudan_swptns.dvi  for    */
/* the DOC.                            */
/*-------------------------------------*/
/*  Nicola Moreni, Premia 2005         */
/*-------------------------------------*/

#include  "lmm1d_stdi.h"
#include "enums.h"

#include "math/lmm/lmm_header.h"
#include "math/lmm/lmm_libor.h"
#include "math/lmm/lmm_products.h"
#include "math/lmm/lmm_volatility.h"
#include "math/lmm/lmm_basis.h"
#include "math/lmm/lmm_numerical.h"
#include "math/lmm/lmm_zero_bond.h"

#include <stdlib.h>
#include <string.h>

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2007+2) //The "#else" part of the code will be freely av
    ailable after the (year of creation of this file + 2)
static int CHK_OPT(MC_PED)(void *Opt, void *Mod)
{
  return NONACTIVE;
}
int CALC(MC_PED)(void *Opt,void *Mod,PricingMethod *Met)
{
return AVAILABLE_IN_FULL_PREMIA;
}
#else

static double (*Basis)(double *x, int i);
static int (*computeEvolution)(const PnlVect* ptRand,Libor*
     ptLibOld,Libor* ptLibNew,Volatility* ptVol,double dt,
    double t, double sigma_cost);
```

```
static int check_parameters(float tenor , int numberTime
    Step, int *numFac, double swaptionMat , double swapMat ,
    double payoff_as_Regressor , long numberMCPaths , int *Regr_Basi
    s_Dimension , char* basis_name , char* measure_name , char
    Explanatory , double strike);
// added function for interface

/* Declaration of allocation/liberation/initialization rout
    ines.
 */
/*************************************************************
    *****/
/*                   Auxiliary routines
        */
/*************************************************************
    *****/

static void
mallocBermudaVar(double **RegrVar,double **Res,double** Br
    ownian,double** SwapPrices,
                 double** Numeraire,double **FP, long numb
    erMCPaths,
                 int RegrVarDimension,int RegrBasisDimensio
    n,int numberOfExerciseDates,
                 int Brown_factors)
{
  if ((*RegrVar)==NULL){
    (*RegrVar)=(double*)malloc(numberMCPaths*RegrVarDimens
    ion*sizeof(double));
    if ((*RegrVar)==NULL) abort();
  }
  if ((*Res)==NULL){
    (*Res)=(double*)malloc(RegrBasisDimension*sizeof(
    double));
    if ((*Res)==NULL) abort();
  }
  if ((*Brownian)==NULL){
    (*Brownian)=(double*)malloc(numberMCPaths*numberOfExerc
    iseDates*Brown_factors*sizeof(double));
    if ((*Brownian)==NULL) abort();
  }
```

```
  if ((*SwapPrices)==NULL){
    (*SwapPrices)=(double*)malloc(numberMCPaths*numberOfEx
    erciseDates*sizeof(double));
    if ((*SwapPrices)==NULL) abort();
  }
  if ((*Numeraire)==NULL){
    (*Numeraire)=(double*)malloc(numberMCPaths*numberOfEx
    erciseDates*sizeof(double));
    if ((*Numeraire)==NULL) abort();
  }
  if ((*FP)==NULL){
    (*FP)=(double*)malloc(numberMCPaths*sizeof(double));
    if ((*FP)==NULL) abort();
  }
}
static void freeBermudaVar(double **RegrVar,double **Res,
    double** Brownian,
                           double** SwapPrices,double**
    Numeraire, double **FP)
{

  free(*RegrVar);
  free(*Res);
  free(*Brownian);
  free(*SwapPrices);
  free(*Numeraire);
  free(*FP);
  (*RegrVar)=NULL;
  (*Res)=NULL;
  (*Brownian)=NULL;
  (*SwapPrices)=NULL;
  (*Numeraire)=NULL;
  *FP=NULL;
}

/*initialization of the Explicatory variable: Paying Value,
     Numeraire or Brownian Motion*/
static void
initStateVector(double *X,double *Brownian,double* SwapPric
    es,double *Numeraire,
                int j,long numberMCpaths,int numberOfExerc
```

```
    iseDates,int numberOfFactors,char Rflag)
{
  int k,l;

  for(k=0;k<numberMCpaths;k++){
    if (Rflag=='B'){for(l=0;l<numberOfFactors;l++) X[k*numb
    erOfFactors+l]=Brownian[k*(numberOfFactors*numberOfExerciseD
    ates)+j*numberOfFactors+l];
    }
    else if(Rflag=='S')  {X[k]=ppos(SwapPrices[k*numberOfEx
    erciseDates+j]);}
    else  X[k]=Numeraire[k*numberOfExerciseDates+j];

  }
  return;
}
//*********************************************************
    *********************/
//Regression FOR Bermudan Swaptions.
//It is of type
//     FP~scalprod(Res,VBase(X))
//with FP's,X's as inputs, Res as output.
//Least squares approach:
//Res=argmin_a[sum_k (FP^k-scalprod(a,VBase(X^k)))^2]
//Solution obtained by differentiating wrt to Res, rewrit
    ing as M*Res=AuxR and then
//finding Cholesky square root of M.
//We regress only on at the money path (excercing value ar
    e needed by algorithm)
/*********************************************************
    *********************/
static void Regression(long NumberMCPaths,int numberOfExerc
    iseDates,
                       int RegrBasis_Dimension,
                       int X_Dimension,
                       int Swap_Entry_Time,
                       int PayOff_As_Regressor,
                       double* X,double* FP,double* Swap_
    Prices,
                       double* Res)
{
```

```
int i,j;
double *XPaths_ptr=X,AuxOption;
PnlVect *AuxR=NULL,*VBase=NULL;
PnlMat *M = NULL;
long k,InTheMoney=0;


//Memory Allocation for auxiliary  pointers
M=pnl_mat_create (RegrBasis_Dimension, RegrBasis_Dimensio
  n);
AuxR = pnl_vect_create (RegrBasis_Dimension);
VBase = pnl_vect_create (RegrBasis_Dimension);

//Initialisation of auxiliary pointers
pnl_vect_set_double (AuxR, 0.0);
pnl_mat_set_double (M, 0.0);

for(k=0;k<NumberMCPaths;k++){
  //kth regressor value
  AuxOption=ppos(Swap_Prices[k*numberOfExerciseDates+Swa
  p_Entry_Time]);
  //only the at-the-monney path are taken into account
  if (AuxOption>0){
    InTheMoney++;
    //value of the regressor basis on the kth path
    if (PayOff_As_Regressor<=Swap_Entry_Time){
      //here, the payoff function is introduced in the
  regression basis
      pnl_vect_set (VBase, 0, AuxOption);
      for (i=1;i<RegrBasis_Dimension;i++){
        pnl_vect_set (VBase, i, Basis(XPaths_ptr,i-1));
      }
    } else {
      for (i=0;i<RegrBasis_Dimension;i++){
        pnl_vect_set (VBase, i, Basis(XPaths_ptr,i));
      }
    }
    //empirical regressor dispersion matrix
    for (i=0;i<RegrBasis_Dimension;i++)
      for (j=0;j<RegrBasis_Dimension;j++){
        double tmp = pnl_mat_get (M, i, j);
```

```
          pnl_mat_set (M, i, j , tmp + pnl_vect_get (VBase,
     i) *
                       pnl_vect_get (VBase,j));
        }

      for (i=0;i<RegrBasis_Dimension;i++){
        double tmp = pnl_vect_get(AuxR, i);
        pnl_vect_set (AuxR, i, FP[k] * pnl_vect_get (VBase,
    i) + tmp);
      }
    }
    XPaths_ptr+=X_Dimension;
  }
  if (InTheMoney==0){
    for (i=0;i<RegrBasis_Dimension;i++) {Res[i]=0.;}
  }
  else {
    /* solve in the least square sense, using a QR decompo
    sition */
    pnl_mat_ls (M, AuxR);
    memcpy (Res, AuxR->array, AuxR->size*sizeof(double));
  }

  pnl_mat_free(&M);
  pnl_vect_free(&AuxR);
  pnl_vect_free(&VBase);
}



static int computeBermudeanSwaption(long numberMCPaths,int
    numberTimeStep,
                                    Libor *ptLib, Swaption*
     ptSwpt, Volatility* ptVol,
                                    int RegrBasisDimension,
                                    double payoff_as_regres
    sor,
                                    char* Basis_Choice,cha
    r* Measure_Choice,
                                    char Explanatory,int     generator,double lo
{
```

```
int i,j,l;
long k;
double time,dt,AuxNumSpot,AuxOption,AuxScal;
double *X=NULL,*Res=NULL,*FP=NULL; //X is the state vec
  tor for regression
double *Brownian=NULL,*SwapPrices=NULL,*Numeraire=NULL;
double *W=NULL;
int s,numberOfExerciseDates,PayOff_As_Regressor;
int RegrVarDimension;
char auxstring[10];
char ErrorMessage[1000];
Libor *ptLibTemp=NULL;
Libor *ptLibOld;
PnlVect *ptRand;
ErrorMessage[0]='{0';

/*Initialization of Auxiliary Constants:time step for SD
  E discretization
  (Euler scheme), index of first exercing date, payoffasr
  egressor.....*/
dt=ptLib->tenor/(double)numberTimeStep;
s=(int)(ptSwpt->swaptionMaturity/ptLib->tenor);
numberOfExerciseDates=ptLib->numberOfMaturities-s;
PayOff_As_Regressor=(int)(payoff_as_regressor/ptLib->ten
  or)-s;
ptRand = pnl_vect_create (ptVol->numberOfFactors);

/*RegrVarDim,nametobasis,nametomeasure*/
if(Explanatory=='B') RegrVarDimension=ptVol->numberOfFact
  ors;
else RegrVarDimension=1;
sprintf(auxstring,"%d",RegrVarDimension);
strcat(Basis_Choice,auxstring);/*Basis_Choice Must be "
  CanD$(Regr_Var_Dimension)" or "HerD$(Regr_Var_Dimension)"*/
Name_To_Basis(ErrorMessage,Basis_Choice,&Basis,RegrVarDim
  ension);
Name_To_Measure(ErrorMessage, Measure_Choice, &computeE    volution);

/*Bermuda Variables Memory Allocation*/
mallocLibor(&ptLibOld,ptLib->numberOfMaturities,ptLib->
```

```
  tenor,10);
/*ptLibOld keeps record of the initial values*/
mallocLibor(&ptLibTemp,ptLib->numberOfMaturities,ptLib->
  tenor,10);
pnl_rand_init(generator,ptVol->numberOfFactors,numberMCPa
  ths);
mallocBermudaVar(&X,&Res,&Brownian,&SwapPrices,&Numerair
  e,&FP,numberMCPaths,RegrVarDimension,RegrBasisDimension,
  numberOfExerciseDates,ptVol->numberOfFactors);
W=(double*)malloc(ptVol->numberOfFactors*sizeof(double));

/*Libor Time evolution + record of Brownian Paths, Numera
  ire Paths and Swap Prices */
for(k=0;k<numberMCPaths;k++){
  time=0.0;
  AuxNumSpot=(1.0+ptLib->tenor*GET(ptLib->libor,0));/*AuX
  NumSpot=NumerSpot(T_1)*/
  copyLibor(ptLibOld,ptLib);
  Set_to_Zero(W,ptVol->numberOfFactors);

  for (j=0;j<=(ptLib->numberOfMaturities-2);j++){
    for(i=0;i<numberTimeStep;i++){
      /*time evolution from T_j+i*dt to T_j+(i+1)*dt*/
      pnl_vect_rand_normal (ptRand, ptVol->numberOfFactor
  s, generator);
      computeEvolution(ptRand,ptLib,ptLibTemp,ptVol,dt,
  time, sigma_cost);
      copyLibor(ptLibTemp,ptLib);
      time+=dt;
      for(l=0;l<ptVol->numberOfFactors;l++) W[l]+=(sqrt(
  dt)*GET(ptRand, l));
    }
    AuxNumSpot*=(1.0+ptLib->tenor*GET(ptLib->libor,j+1));
  /*AuxNumSpot=NumeraireSpot(T_{j+2})*/
    computeNumeraire(Measure_Choice,ptLib,ptSwpt,Numerair
  e,j,k,AuxNumSpot);
    if ((s-1)<=j) {
      SwapPrices[k*numberOfExerciseDates+(j+1-s)]=compu
  teSwapPrice(ptLib,ptSwpt,j+1,j+1,ptLib->numberOfMaturities);
      for(l=0;l<ptVol->numberOfFactors;l++) Brownian[k*(
  numberOfExerciseDates*ptVol->numberOfFactors)+(j+1-s)*ptVol->
```

```
    numberOfFactors+l]=W[l];

      }
      putLiborToZero(ptLib,j+1);

    }

  }

  /*Backward programming

    Price at last Excercise date is just excercise price */
  for(k=0;k<numberMCPaths;k++)
    {
      FP[k]=ppos(SwapPrices[k*numberOfExerciseDates+(numb
    erOfExerciseDates-1)]);
      //If s=e-1 the option is indeed European, actualiza
    tion changes....
      if(numberOfExerciseDates==1)
        {
          FP[k]/=Numeraire[k*numberOfExerciseDates+0];
        }
      else
        {
          FP[k]*=(Numeraire[k*numberOfExerciseDates+(numb
    erOfExerciseDates-2)]/Numeraire[k*numberOfExerciseDates+(
    numberOfExerciseDates-1)]);
        }
    }
  //Price at time T_j: regression of FP over X(T_j))
  for(j=numberOfExerciseDates-2;j>=0;j--)
    {
      initStateVector(X,Brownian,SwapPrices,Numeraire,j,
    numberMCPaths,
                      numberOfExerciseDates,ptVol->numberO
    fFactors,Explanatory);
      Regression(numberMCPaths,numberOfExerciseDates,RegrB
    asisDimension,ptVol->numberOfFactors,j,
                 PayOff_As_Regressor,X,FP,SwapPrices,Res);

      for(k=0;k<numberMCPaths;k++){//exercise value
```

```
    AuxOption=ppos(SwapPrices[k*numberOfExerciseDates+
j]);
    //approximated continuation value, only the at-the-
monney paths are taken into account
    if (AuxOption>0){
      // if PayOff_As_Regressor<=j, excercise value is
introduced into  regression basis
      if (PayOff_As_Regressor<=j){
        AuxScal=AuxOption*Res[0];
        for (l=1;l<RegrBasisDimension;l++){
          AuxScal+=Basis(X+k*RegrVarDimension,l-1)*Res[
l];
        }
      } else {
        AuxScal=0.;
        for (l=0;l<RegrBasisDimension;l++){
          AuxScal+=Basis(X+k*RegrVarDimension,l)*Res[l]
;
        }
      }
      // AuxScal contains the approximated continuatio
n value
      // if AuxScal< exercise value, the optimal stopp
ing time is modified
      if (AuxOption>AuxScal){
        FP[k]=AuxOption;
      }
    }
    //Discount Factor from time T_{s+j} to T_{s+j-1}}
    if(j>0) FP[k]*=(Numeraire[k*numberOfExerciseDates+
j-1]/Numeraire[k*numberOfExerciseDates+j]);
    //Discount from T_s downto  time=0.0
    else FP[k]*=(1.0/Numeraire[k*numberOfExerciseDates+
0]);
  }

}


//price at time t=0.0 is only the mean discounted value
ptSwpt->price=0.0;
```

```
  for(k=0;k<numberMCPaths;k++){
    ptSwpt->price+=FP[k];
  }
  ptSwpt->price/=(double)numberMCPaths;

  if (ErrorMessage[0] == '{0')
    printf("Warning : {n%s{n", ErrorMessage);
  //freeing memory
  freeBermudaVar(&X,&Res,&Brownian,&SwapPrices,&Numeraire,&
    FP);
  freeLibor(&ptLibTemp);
  freeLibor(&ptLibOld);
  pnl_vect_free (&ptRand);
  free(W);
  return(1);
}


static double
lmm_swaption_payer_bermudan_LS_pricer(float tenor , int
    numberTimeStep,
                                      int numFac, double
    swaptionMat, double swapMat,
                                      double payoff_as_Reg
    ressor, long numberMCPaths,
                                      int Regr_Basis_Dimens
    ion, char* basis_name ,
                                      char* measure_name,
    char Explanatory, double strike,
                                      int generator,
    double l0, double sigma_cost)
{

  // int numberTimeStep=10;
  // int numFac=1;
  // double  swaptionMat in (years)
  // double  swapMat in  (years)
  // payoff_as_Regressor in (years) Maturity after which
    payoff is included in regression
  // long numberMCPaths number of monte carlo paths
  // int Regr_Basis_Dimension=4; //finite-dimensional appro
```

```
  x. of Lš
// char Explanatory = 'N' or 'B' or 'S' ---> Explanatory
  variable for regression B=Brownian, S=Nominal Swap Paying Value
  , N=Numeraire;
//  int numMat ;       //WARNING: including T_0=0, matu
  rities number is 'numMat+1'

Volatility *ptVol;
Libor *ptLib;
Swaption *ptSwpt;


double  priceVal=0.20;
double K=strike;//strike
int numMat;
char Basis_Choice[10];  //See in the followings
char Measure_Choice[10];// "    "    "    "
double p;
/*double DT=(tenor/(double)numberTimeStep);*/


check_parameters( tenor , numberTimeStep,  &numFac,  swa
  ptionMat ,  swapMat ,  payoff_as_Regressor , numberMCPaths
  , &Regr_Basis_Dimension , basis_name , measure_name ,  Exp
  lanatory ,  strike);

strcpy(Basis_Choice,basis_name); //"CanD" for canonical
  basis or "HerD" for hermite
strcpy(Measure_Choice,measure_name);//"Spot" or "Fwd" cfr
   numerical.c for evolution

//Memory Allocation and Libor/Swap Structures Initializ
  ation
numMat=(int)(swapMat/tenor);

mallocLibor(&ptLib,numMat,tenor,10);
mallocVolatility(&ptVol,numFac,sigma_cost);
mallocSwaption(&ptSwpt,swaptionMat,swapMat,priceVal,K,ten
  or);

//Swaption Computation
```

```
  computeBermudeanSwaption(numberMCPaths,numberTimeStep,pt
    Lib,ptSwpt,ptVol,Regr_Basis_Dimension, payoff_as_Regressor,
    Basis_Choice,Measure_Choice,Explanatory,generator,l0,sigma_
    cost);

  p=ptSwpt->price;
  //Printing Results to STDOutput
  //printf("s=%d, e=%d, exdates=%d, payoffasregr=%d{n",s,pt
    Lib->numberOfMaturities,numberOfExerciseDates,PayOff_As_Reg
    ressor);

  //initLibor(ptLib);
  //I=computeSwaptionPriceSpot(numberMCPaths,numberTimeStep
    ,DT,ptLib,ptSwpt,ptVol);

  //freeing Memory
  freeSwaption(&ptSwpt);
  freeLibor(&ptLib);
  freeVolatility(&ptVol);

  return(p);
}


static int
check_parameters(float tenor , int numberTimeStep, int *
    numFac, double swaptionMat,
                double swapMat , double payoff_as_Regres
    sor , long numberMCPaths,
                int *Regr_Basis_Dimension , char* basis_na
    me , char* measure_name ,
                char Explanatory , double strike)
{
  double s;
  double M;

  if(swaptionMat>=swapMat)
    {
      printf(" swaption maturity must be lower than swap
    maturity !{n");
      exit(-1);
```

```
  }
s=(int)(swaptionMat/tenor);
if (fabs(swaptionMat-s*tenor)> 0.00001)
  {
    printf(" swaption maturity must be a multiple of pe
  riod{n");
    exit(-1);
  }
M=(int)(swapMat/tenor);
if (fabs(swapMat-M*tenor)> 0.0)
  {
    printf(" swap maturity must be a multiple of period{
  n");
    exit(-1);
  }
if((Explanatory!='N') && (Explanatory!='B')  && (Explan
  atory!='S') )
  {
    printf("Explanatory variable must be either B, N or
  S {n");
    exit(-1);
  }
if( *numFac>2 )
  {
    printf("Number of factors too high switching to 2 {n"
  );
    *numFac=2;
  }
if( (*numFac==1) && (*Regr_Basis_Dimension>6))
  {
    printf("Regression basis dimension is too high swicth
  ing to 6 {n");
    *Regr_Basis_Dimension=6;
  }
if( (*numFac==2) && (*Regr_Basis_Dimension>21))
  {
    printf("Regression basis dimension is too high swicth
  ing to 6 {n");
    *Regr_Basis_Dimension=6;
  }
if( (payoff_as_Regressor<swaptionMat) || (payoff_as_Reg
```

```
    ressor>(swapMat-tenor)) )
    {
      printf("payoff_as_Regressor value is not valid, must
    be within [ swaptionMat  swapMat-tenor ] {n");
      printf("chosing payoff_as_Regressor=swapMat-tenor wil
    l not include the payoff in the basis! {n");
      exit(-1);
    }

  return(1);
}


static int
mc_pederesen_bermswaptionlmm1d(NumFunc_1 *p,double l0,
    double t0, double sigma,
                                int numFac,double swapMat,
    double swaptionMat,
                                int am,double Nominal,
    double strike,double tenor,
                                int generator,int numberTime
    Step,long numberMCPaths,double *price)
{
  int numMat;
  double* maturities;
  int Regr_Basis_Dimension=4 ;    //finite-dimensional app
    rox. of Lš
  char Explanatory='N';           //Explanatory variable fo
    r regression B=Brownian,
  //                 S=Nominal Swap Paying Value, N=Numera
    ire;
  char* basis_name="HerD";        //Hermite basis
  char* measure_name="Spot" ;     // spot "    "    "    "
  double payoff_as_Regressor; //(years) Maturity after whic
    h payoff is included in regression

  swapMat=swapMat-t0;
  swaptionMat=swaptionMat-t0;

  payoff_as_Regressor=swaptionMat+tenor;
  numMat=(int)(swapMat/tenor);
```

```
  maturities=(double*)malloc(numMat*sizeof(double));

  /* if ((p->Compute)==&Put) */
  /*   payer_or_receiver=1; */
  /* else */
  /*   if ((p->Compute)==&Call) */
  /*     payer_or_receiver=0; */

  *price = Nominal * lmm_swaption_payer_bermudan_LS_pricer(
    tenor ,numberTimeStep, numFac, swaptionMat , swapMat,payoff_
    as_Regressor,numberMCPaths,Regr_Basis_Dimension ,basis_name,
    measure_name , Explanatory,strike,generator,l0,sigma);

  free(maturities);

  return OK;
}

int CALC(MC_PED)(void *Opt,void *Mod,PricingMethod *Met)
{
  TYPEOPT* ptOpt=(TYPEOPT*)Opt;
  TYPEMOD* ptMod=(TYPEMOD*)Mod;

  return mc_pederesen_bermswaptionlmm1d(ptOpt->PayOff.Val.
    V_NUMFUNC_1,ptMod->l0.Val.V_PDOUBLE,
                                          ptMod->T.Val.V_DA
    TE,
                                          ptMod->Sigma.Val.V_
    PDOUBLE,
                                          ptMod->NbFactors.
    Val.V_ENUM.value,
                                          ptOpt->BMaturity.
    Val.V_DATE,
                                          ptOpt->OMaturity.
    Val.V_DATE,
                                          ptOpt->EuOrAm.Val.
    V_BOOL,
                                          ptOpt->Nominal.Val.
    V_PDOUBLE,
                                          ptOpt->FixedRate.
    Val.V_PDOUBLE,
```

```
                                        ptOpt->ResetPeriod.
    Val.V_DATE,
                                        Met->Par[0].Val.V_
    ENUM.value,
                                        Met->Par[1].Val.V_
    PINT,
                                        Met->Par[2].Val.V_
    LONG,
                                        &(Met->Res[0].Val.
    V_DOUBLE));

}

static int CHK_OPT(MC_PED)(void *Opt, void *Mod)
{

  if ((strcmp(((Option*)Opt)->Name,"PayerBermudanSwaption")
    ==0))
    return OK;
  else
    return WRONG;
}


#endif //PremiaCurrentVersion
static int MET(Init)(PricingMethod *Met,Option *Opt)
{
  if ( Met->init == 0)
    {
      Met->init=1;

      Met->Par[0].Val.V_ENUM.value=0;
      Met->Par[0].Val.V_ENUM.members=&PremiaEnumRNGs;
      Met->Par[1].Val.V_INT=10;
      Met->Par[2].Val.V_LONG=10000;
    }
  return OK;
}

PricingMethod MET(MC_PED)=
{
```

```
  "MC_Pedersen_BermSwaption",
  {{"RandomGenerator",ENUM,{100},ALLOW},
   {"Time Steps",INT,{100},ALLOW},
   {"N Simulation",LONG,{100},ALLOW},
   {" ",PREMIA_NULLTYPE,{0},FORBID}},
  CALC(MC_PED),
  {{"Price",DOUBLE,{100},FORBID}/*,{"Delta",DOUBLE,{100},FO
    RBID}*/ ,{" ",PREMIA_NULLTYPE,{0},FORBID}},
  CHK_OPT(MC_PED),
  CHK_ok,
  MET(Init)
} ;
```

References