Help

```
#include "
href../../../../common/math/mcam/src/regression_h_src.pdfregression.hpp"
#include "pnl/pnl_specfun.h"

mcam::Regression::Regression()
{
    label = "";
    size = 0;
    samples = 0;
}


mcam::Regression::~Regression() { }


mcam::RegressionPol::RegressionPol(int p_size, int p_degree, PnlMat **p_regresso
{
    label = "RegressionPol";
    size = p_size;
    samples = p_samples;
    degree = p_degree;
    regressors = p_regressors;
    basis = pnl_basis_create_from_degree(PNL_BASIS_HERMITE, degree, size);
}


void mcam::RegressionPol::setReduced(mcam::Model *p_mod)
{
    PnlVect *min = p_mod->getMin();
    PnlVect *max = p_mod->getMax();
    pnl_basis_set_domain(basis, min, max);
    pnl_vect_free(&min);
    pnl_vect_free(&max);
}

void mcam::RegressionPol::setCurrentDate(int p_currentDate)
{
    currentDate = p_currentDate;
}
```

```
mcam::RegressionPol::~RegressionPol()
{
    if (basis != NULL) pnl_basis_free(&basis);
}


double mcam::RegressionPol::computeRegressorValue(int pathNumber, const PnlVect
{
    PnlVect Xt = pnl_vect_wrap_mat_row(regressors[pathNumber], currentDate);
    return pnl_basis_eval_vect(basis, p_alpha, &Xt);
}


void mcam::RegressionPol::computeCoefficients(PnlVect *p_alpha, PnlVect *p_Y, Pn
{
    pnl_vect_resize(p_alpha, basis->nb_func);
    pnl_vect_set_zero(p_alpha);
    PnlVect *phi_k = pnl_vect_create_from_scalar(basis->nb_func, 0.);
    PnlMat *A = pnl_mat_create_from_scalar(basis->nb_func, basis->nb_func, 0.);

    /* Construct A and b, to solve A * p_alpha = b*/
    for (int i = 0 ; i < samples ; i++)
    {
        if (p_Z && GET(p_Z, i) == 0.) continue;
        for (int k = 0 ; k < basis->nb_func ; k++)
        {
            PnlVect xi = pnl_vect_wrap_mat_row(regressors[i], currentDate);
            const double tmp = pnl_basis_i_vect(basis, &xi, k);
            PNL_LET(p_alpha, k) += tmp * PNL_GET(p_Y, i);
            PNL_LET(phi_k, k) = tmp;
        }
        /* A += phi_k' * phi_k */
        pnl_mat_dger(1., phi_k, phi_k, A);
    }

    pnl_mat_ls(A, p_alpha);
    pnl_vect_free(&phi_k);
    pnl_mat_free(&A);
}
```

```
mcam::RegressionChaos::RegressionChaos(int p_size, int p_degree, PnlMat **p_regr
{
    label = "RegressionChaos";
    size = p_size;
    samples = p_samples;
    degree = p_degree;
    regressors = p_regressors;
    basis = NULL;
}


mcam::RegressionChaos::~RegressionChaos()
{
    if (basis != NULL) pnl_basis_free(&basis);
}


void  mcam::RegressionChaos::setCurrentDate(int p_currentDate)
{
    currentDate = p_currentDate;
    if (basis != NULL) pnl_basis_free(&basis);
    basis = pnl_basis_create_from_degree(PNL_BASIS_HERMITE, degree, currentDate
}


double mcam::RegressionChaos::computeRegressorValue(int pathNumber, const PnlVec
{
    return pnl_basis_eval(basis, p_alpha, regressors[pathNumber]->array);
}


void mcam::RegressionChaos::computeCoefficients(PnlVect *p_alpha, PnlVect *p_Y,
{
    pnl_vect_resize(p_alpha, basis->nb_func);
    pnl_vect_set_zero(p_alpha);

    for (int l = 0; l < samples; l++)
    {
        if (p_Z && GET(p_Z, l) == 0.) continue; // Only consider in the money pa
```

```
        const double Yl = GET(p_Y, l);
        if (Yl == 0.) continue;
        for (int i = 0; i < basis->nb_func; i++)
        {
            LET(p_alpha, i) += pnl_basis_i(basis, regressors[l]->array, i) * Yl;
        }
    }
    for (int i = 0; i < basis->nb_func; i++)
    {
        int fact = 1;
        for (int j = basis->SpT->I[i]; j < basis->SpT->I[i + 1]; j++)
        {
            fact *= int(pnl_sf_fact(basis->SpT->array[j]));
        }
        LET(p_alpha, i) /= (fact * samples);
    }
}

mcam::RegressionChaosReduced::RegressionChaosReduced(int p_size, int p_degree, P
{
    label = "RegressionChaosReduced";
    size = p_size;
    samples = p_samples;
    degree = p_degree;
    regressors = new PnlMat*[samples];
    for (int n = 0; n < samples; ++n)
    {
        regressors[n] =  pnl_mat_copy(p_regressors[n]);
        pnl_mat_cumsum(regressors[n], 'r');
        for (int j = 1; j < regressors[n]->m; j++)
        {
            PnlVect row = pnl_vect_wrap_mat_row(regressors[n], j);
            pnl_vect_div_scalar(&row, sqrt(double(j + 1)));
        }
    }
    basis = pnl_basis_create_from_degree(PNL_BASIS_HERMITE, degree, size);
}


mcam::RegressionChaosReduced::~RegressionChaosReduced()
{
```

```
    if (basis != NULL) pnl_basis_free(&basis);
    for (int i = 0; i < samples; ++i)
        pnl_mat_free(&(regressors[i]));
    delete [] regressors;
}

void mcam::RegressionChaosReduced::setCurrentDate(int p_currentDate)
{
    currentDate = p_currentDate;
}


double mcam::RegressionChaosReduced::computeRegressorValue(int pathNumber, const
{
    PnlVect Bt = pnl_vect_wrap_mat_row(regressors[pathNumber], currentDate - 1);
    return pnl_basis_eval_vect(basis, p_alpha, &Bt);
}

void mcam::RegressionChaosReduced::computeCoefficients(PnlVect *p_alpha, PnlVect
{
    pnl_vect_resize(p_alpha, basis->nb_func);

    for (int i = 0; i < basis->nb_func; i++)
    {
        double scalar = 0;
        int fact = 1;
        for (int j = 0; j < basis->nb_variates; j++)
        {
            fact *= int(pnl_sf_fact(MGET_INT(basis->T, i, j)));
        }
        for (int l = 0; l < samples; l++)
        {
            if (p_Z && GET(p_Z, l) == 0.) continue;
            PnlVect Bt = pnl_vect_wrap_mat_row(regressors[l], currentDate - 1);
            scalar += pnl_basis_i_vect(basis, &Bt, i) * GET(p_Y, l);
        }
        LET(p_alpha, i) = (scalar / fact) / samples;
    }
}
```