Help

```
#include "
href../../../../common/math/mcam/src/chaos_h_src.pdfchaos.hpp"
#include "pnl/pnl_specfun.h"

mcam::Chaos::Chaos()
    : Martingale(), order(0), Basis(NULL), colMax(NULL)
{
    label = "chaos";
}

mcam::Chaos::~Chaos()
{
    if (Basis != NULL) pnl_basis_free(&Basis);
    if (colMax != NULL) delete [] colMax;
}

mcam::Chaos::Chaos(const Param &P, Model *M, bool prune)
        : Martingale()
{
    int BasisType = PNL_BASIS_HERMITE;
    P.extract("chaos order", order);
    subdates = M->subdates;
    size = M->brownianSize;

    label = "chaos";
    Basis = pnl_basis_create_from_degree(BasisType, order, subdates * size);
    pnl_basis_del_elt_i(Basis, 0);  // Remove the constant term

    if (prune)
    {
        // Remove terms mixing long term increments
        for (int l = Basis->nb_func - 1; l >= 0; l--)
        {
            int min_t = MAX_INT, max_t = 0;;
            for (int j = Basis->SpT->I[l]; j < Basis->SpT->I[l + 1]; j++)
            {
                int t = Basis->SpT->J[j] / size;
                if (t < min_t) min_t = t;
                if (t > max_t) max_t = t;
```

```
            }
            if (max_t - min_t >= 2) pnl_basis_del_elt_i(Basis, l);
        }
    }

    nbFreedom = Basis->nb_func;
    initColMax();
}

mcam::Chaos::Chaos(int _T, int _size, int _order)
{
    int BasisType = PNL_BASIS_HERMITE;
    label = "chaos";
    subdates = _T;
    size = _size;
    order = _order;
    Basis = pnl_basis_create_from_degree(BasisType, order, subdates * size);
    nbFreedom = Basis->nb_func;
    initColMax();
    std::cout << "Number of degrees of freedom: " << nbFreedom << std::endl;
}

/**
 * Compute the index of the largest time step with positive degree in the basis.
 * This is used to determine the largest time up to which all the conditional ex
 */
void mcam::Chaos::initColMax()
{
    colMax = new int[Basis->nb_func];
    for (int j = 0; j < Basis->nb_func; j++)
    {
        if (Basis->SpT->I[j]  == Basis->SpT->I[j + 1])
        {
            colMax[j] = 0;
        }
        else
        {
            colMax[j] = Basis->SpT->J[Basis->SpT->I[j + 1] - 1] / size;
        }
    }
}
```

```
void mcam::Chaos::print() const
{
    mcam::Martingale::print();
    std::cout << " chaos order " << order << std::endl;
    std::cout << "******************************" << std::endl;

}

void mcam::Chaos::computePath(const PnlMat *G, const PnlVect *alpha)
{
    pnl_mat_resize(Mgrad(), subdates + 1, nbFreedom);
    pnl_vect_resize(Mval(), subdates + 1);
    pnl_mat_set_zero(Mgrad());
    pnl_vect_set_zero(Mval());

    for (int j = 0; j < Basis->nb_func; j++)
    {
        /*
         * Since the Basis is orthogonal for the standard normal distribution,
         * if the partial degree w.r.t any increment after time
         * k is strictly positive, the conditional expectation of this part
         * vanishes
         */
        const double alpha_j = GET(alpha, j);
        const double pol_j = pnl_basis_i(Basis, G->array, j);
        double val_j = pol_j * alpha_j;
        for (int k = subdates; k > colMax[j]; k--)
        {
            MLET(Mgrad(), k , j) = pol_j;
            LET(Mval(), k) += val_j;
        }
    }
}
```