Help

```
#include "pnl/pnl_mathtools.h"
#include "
href../../../../common/math/mcam/src/wiener_h_src.pdfwiener.hpp"

/**
 * Define the trigonometric basis as in Belomestney (2013)
 *
 * Note that this basis is defined by both sin and cos functions. The sin
 * functions correspond to even values of k, whereas the co functions
 * correpsond to odd values of k.
 *
 *    For |x|<=0.5, trigo(x, 2 * k) = sin(2 * k * x)
 *    For |x|<=0.5, trigo(x, 2 * (k + 1)) = cos(2 * (k + 1) * x)
 *
 * @param x point
 * @param k order
 *
 * @return f_k(x)
 */
static double trigo(double x, int k)
{
    if (x < -0.5) return 0.;
    if (x > 0.5) return 1.;
    if (PNL_IS_EVEN(k))
        return sin((k / 2) * x);
    else
        return cos((k / 2) * x);
}

mcam::Wiener::Wiener()
    : Martingale(), order(0), Basis(NULL), mod(0)
{
    label = "Wiener";
}

mcam::Wiener::~Wiener()
{
    if (Basis != NULL) pnl_basis_free(&Basis);
}
```

```
mcam::Wiener::Wiener(const Param &P, Model *_mod)
    : Martingale()
{
    if (_mod->size != _mod->brownianSize)
    {
        std::cout << "Model size and brownian size are different. Aborting...\ n
        abort();
    }
    label = "wiener";
    subdates = _mod->subdates;
    size = _mod->size;
    mod = _mod;
    Basis = NULL;
}

void mcam::Wiener::print() const
{
    mcam::Martingale::print();
    std::cout << label << " basis " << std::endl;
    std::cout << " order " << order << std::endl;
    std::cout << "*******************************" << std::endl;

}

void mcam::Wiener::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 i = 0; i < size; i++)
    {
        const double S0_i = GET(mod->spot, i);
        for (int k = 0; k < Basis->nb_func; k++)
        {
            int j = i * Basis->nb_func + k;
            for (int t = 1; t < (subdates + 1); t++)
            {
                double xi = MGET(mod->getPath(), t - 1, i) / S0_i;
```

```
                double lxi = log(xi); // / ((subdates - (t - 1)) * mod->dt);
                MLET(Mgrad(), t , j) = MGET(Mgrad(), t - 1, j) + pnl_basis_i(Bas
            }
        }
    }
    pnl_mat_mult_vect_inplace(Mval(), Mgrad(), alpha);
}

mcam::Trigo::Trigo()
    : Wiener()
{
    label = "trigo";
}

mcam::Trigo::Trigo(const Param &P, Model *_mod)
    : Wiener(P, _mod)
{
    int BasisType;
    label = "trigo";

    P.extract("trigo order", order);
    BasisType = pnl_basis_type_register("Trigo", trigo, NULL, NULL);
    Basis = pnl_basis_create_from_degree(BasisType, 2 * order, 1);
    pnl_basis_del_elt_i(Basis, 0);  // Remove the first term because sin(0) = 0
    nbFreedom = Basis->nb_func * size;
}

mcam::Trigo::~Trigo() {}

mcam::Pol::Pol()
    : Wiener()
{
    label = "pol";
}

mcam::Pol::Pol(const Param &P, Model *_mod)
    : Wiener(P, _mod)
{
    label = "pol";

    P.extract("pol order", order);
```

```
    Basis = pnl_basis_create_from_degree(PNL_BASIS_CANONICAL, order, 1);
    nbFreedom = Basis->nb_func * size;
}

mcam::Pol::~Pol() {}

void mcam::Pol::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 i = 0; i < size; i++)
    {
        const double S0_i = GET(mod->spot, i);
        for (int k = 0; k < Basis->nb_func; k++)
        {
            int j = i * Basis->nb_func + k;
            for (int t = 1; t < (subdates + 1); t++)
            {
                double xi = MGET(mod->getPath(), t - 1, i) / S0_i;
                double lxi = log(xi);
                MLET(Mgrad(), t , j) = MGET(Mgrad(), t - 1, j) + pnl_basis_i(Bas
            }
        }
    }
    pnl_mat_mult_vect_inplace(Mval(), Mgrad(), alpha);
}
```