Help

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

class Func
{
public:
    int d;
    int t;
    double dt;
    Func(int t, int d, double dt) : d(d), t(t), dt(dt) { }
    virtual ~Func() { }
    virtual double operator()(PnlMat *DeltaB) = 0;
};

class Func_BT : public Func
{
public:
    Func_BT(int t, int d, double dt) : Func(t, d, dt) { }
    virtual ~Func_BT() { }
    virtual double operator()(PnlMat *DeltaB)
    {
        double b = 0.;
        for (int i = 0 ; i < t ; i++)
        {
            b += MGET(DeltaB, i, d);
        }
        return sqrt(dt) * b;
    }
};

class Func_BTSquare : public Func
{
public:
    Func_BTSquare(int t, int d, double dt) : Func(t, d, dt) { }
    virtual ~Func_BTSquare() { }
    virtual double operator()(PnlMat *DeltaB)
```

```
    {
        double b = 0.;
        for (int i = 0 ; i < t ; i++)
        {
            double dB_i = MGET(DeltaB, i, d);
            b += dB_i * dB_i;
        }
        return dt * b;
    }
};

double test_chaos_decomposition(mcam::RegressionChaos &regC, mcam::Chaos &C, Pnl
{
    PnlVect *Y = pnl_vect_create(nbSamples);
    PnlVect *alpha = pnl_vect_new();

    for (int l = 0 ; l < nbSamples ; l++)
    {
        LET(Y, l) = (*F)(InDeltaB[l]);
    }
    regC.computeCoefficients(alpha, Y);
#ifndef NDEBUG
    pnl_vect_print_asrow(alpha);
#endif
    pnl_vect_free(&Y);

    double sq_err = 0.;
    for (int l = 0 ; l < nbSamples ; l++)
    {
        double y = (*F)(OutDeltaB[l]);
        C.computePath(OutDeltaB[l], alpha);
        double tmp = (y - C.at(F->t));
        sq_err += tmp * tmp;
    }
    return sq_err / nbSamples;
}

int main()
{
    int nbDates = 10;
    int size = 1;
```

```
int nbSamples = 500000;
int order = 2;
PnlRng *rng = pnl_rng_create(PNL_RNG_MERSENNE);
PnlMat **InDeltaB = new PnlMat*[nbSamples];
PnlMat **OutDeltaB = new PnlMat*[nbSamples];

pnl_rng_sseed(rng, 0);

for (int l = 0; l < nbSamples; l++)
{
    InDeltaB[l] = pnl_mat_create(nbDates, size);
    pnl_mat_rng_normal(InDeltaB[l], nbDates, size, rng);
    OutDeltaB[l] = pnl_mat_create(nbDates, size);
    pnl_mat_rng_normal(OutDeltaB[l], nbDates, size, rng);
}

mcam::RegressionChaos regC(size, order, InDeltaB, nbSamples);
regC.setCurrentDate(nbDates);
mcam::Chaos C(nbDates, size, order);
{
    Func_BT *BT = new Func_BT(nbDates, 0., 1.);
    double err = test_chaos_decomposition(regC, C, InDeltaB, OutDeltaB, nbSa
    std::cout << "squared error B_T: " << err << std::endl;
    delete BT;
}

{
    Func_BT *BT_2 = new Func_BT(nbDates / 2, 0., 1.);
    double err = test_chaos_decomposition(regC, C, InDeltaB, OutDeltaB, nbSa
    std::cout << "squared error B_{T/2}: " << err << std::endl;
    delete BT_2;
}

{
    Func_BTSquare *BTSquare = new Func_BTSquare(nbDates, 0., 1.);
    double err = test_chaos_decomposition(regC, C, InDeltaB, OutDeltaB, nbSa
    std::cout << "squared error B_T^2: " << err << std::endl;
    delete BTSquare;
}

{
```

```
        Func_BTSquare *BTSquare = new Func_BTSquare(1, 0., 1.);
        double err = test_chaos_decomposition(regC, C, InDeltaB, OutDeltaB, nbSa
        std::cout << "squared error B_{t_1}^2: " << err << std::endl;
        delete BTSquare;
    }
    return 0;
}
```