

[Help](#)

```
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2008+2) //The "#els
#else
////////////////////////////////////
// Codé par Céline Labart 31 décembre 2007
////////////////////////////////////

#include <stdlib.h>
#include <stdio.h>
#include <
href../../common/math/cdo/cdo_math_h_src.pdfmath.h>

#include "pnl/pnl_matrix.h"
#include "
href../../common/math/bsnd_math/noyau_bsde_h_src.pdfnoyau_bsde.h"

static double inv_sqrt_2M_PI = 0.3989422804;
static double alpha;

void init_alpha(int d)
{
    alpha = pow(2, d + 2);
}
//donne la fonction noyau en espace
double Kx(double x)
{
    if (fabs(x) <= 4.0) return inv_sqrt_2M_PI * exp(-x * x / 2.0);
    else return 0.0;
}

//donne la dérivée de la fonction noyau en espace
double Kx_prime(double x)
{
    if (fabs(x) <= 4.0)
        return -x * inv_sqrt_2M_PI * exp(-x * x / 2.0);
    else return 0.0;
}
```



```

    if (y < EPS_BSDE) return 0.0;
    if (y > 1.0 - EPS_BSDE) return -1.0 / (y * y);
    else return -3 * y * y + 2.0;
}

//dérivée seconde de la fonction de régularisation

double g0_seconde(double y)
{
    if (y < EPS_BSDE) return 0.0;
    if (y > 1.0 - EPS_BSDE) return 2.0 / (pow(y, 3));
    else return -6 * y;
}

int test(const PnlVect *Xi, double Ti, double s, PnlVect *y, double hx, double h)
{
    int j = 0;
    int d = y->size;
    int p = 0;
    for (j = 0; j < d; j++)
    {
        if (fabs(pnl_vect_get(Xi, j) - pnl_vect_get(y, j)) < 4 * hx)
            p++;
    }
    if ((fabs(s - Ti) < 4 * ht) && (p == d)) return 1;
    else return 0;
}

// fonction d'opérateurs à noyaux qui calcule l'approchée
// d'une fonction au point (s,y) par l'opérateur à noyaux on
// est en dimension d donc on stocke les points aléatoires X
// dans une matrice (Np,d)

double operateur_noyau(PnlMat *X, PnlVect *T_alea, double s,
                       PnlVect *y, double hx, double ht, PnlVect *V,
                       double Np, double T, double support_espace)
{
    int i, j;
    double T_alea_i;

```

```

PnlVect *Xi;
double ms0 = 0.0; //rn
double mu0 = 0.0; //fn
double p = 0.0;
int d = y->size;
double constante = alpha * T * support_espace / (Np * ht * pow(hx, d));
Xi = pnl_vect_create(d);
for (i = 0; i < Np; i++)
{
    T_alea_i = pnl_vect_get(T_alea, i);
    pnl_mat_get_row(Xi, X, i);
    if (test(Xi, T_alea_i, s, y, hx, ht) == 1)
    {
        p = Kt((s - T_alea_i) / ht);
        for (j = 0; j < d; j++)
        {
            p = p * Kx((pnl_vect_get(y, j) - pnl_vect_get(Xi, j)) / hx);
        }
        ms0 = ms0 + p * (pnl_vect_get(V, i)); //calcul de rn(modulo constante
        //multiplicative)
        mu0 = mu0 + p; //calcul de fn(modulo constante
        //multiplicative)
    }
}
pnl_vect_free(&Xi);
return ms0 * constante * g0(mu0 * constante);
}

void derive_x_operateur_noyau(PnlVect *res, PnlMat *X, PnlVect *T_alea, double
                                PnlVect *y, double hx, double ht, PnlVect *V,
                                double Np, double T, double support_espace)
{
    double Kti, Vi, ms0, ms1, mu0, mu1, g, g_prime, prod, p, p_prime, mu0_cst;
    int i, j, k;
    double T_alea_i;
    PnlVect *Xi;
    int d = y->size;
    double constante = (double)alpha * T * support_espace / (Np * ht * pow(hx, d))
    pnl_vect_resize(res, d);
    Xi = pnl_vect_create(d);
    for (k = 0; k < d; k++)

```

```

{
    ms0 = 0.0;
    ms1 = 0.0;
    mu0 = 0.0;
    mu1 = 0.0;
    for (i = 0; i < Np; i++)
    {
        T_alea_i = pnl_vect_get(T_alea, i);
        pnl_mat_get_row(Xi, X, i);
        if (test(Xi, T_alea_i, s, y, hx, ht) == 1)
        {
            Kti = Kt((s - T_alea_i) / ht);
            prod = Kti;
            for (j = 0; j < k; j++)
            {
                prod *= Kx((pnl_vect_get(y, j) - pnl_vect_get(Xi, j)) / hx);
            }
            for (j = k + 1; j < d; j++)
            {
                prod *= Kx((pnl_vect_get(y, j) - pnl_vect_get(Xi, j)) / hx);
            }
            // prod represente le produit des Kxj pour
            // j diff de k et Kt
            p = prod * Kx((pnl_vect_get(y, k) - pnl_vect_get(Xi, k)) / hx);
            // produit des Kxj et Kt
            p_prime = prod * Kx_prime((pnl_vect_get(y, k) - pnl_vect_get(Xi, k) /
            //produit des Kxj sauf Kxk fois K_prime_xk
            Vi = pnl_vect_get(V, i);
            ms0 += p * Vi; //calcul de rn
            ms1 += p_prime * Vi; //calcul de rn' en x
            mu0 += p; //calcul de fn
            mu1 += p_prime; //calcul de fn' en x
        }
    }
    mu0_cst = mu0 * constante;
    g = g0(mu0_cst);
    g_prime = g0_prime(mu0_cst);
    pnl_vect_set(res, k, constante / (hx) * (ms1 * g + constante * ms0 * mu1 *
}
pnl_vect_free(&Xi);
}

```

```

double derive_t_operateur_noyau(PnlMat *X, PnlVect *T_alea, double s,
                                PnlVect *y, double hx, double ht, PnlVect *V,
                                double Np, double T, double support_espace)
{
    double Kti, Kti_prime, Vi;
    double ms0, ms3;
    double mu0, mu3;
    double g, g_prime;
    double prod, p, p_prime;
    double mu0_cst;
    double Ti;
    int i, j;
    double T_alea_i;
    PnlVect *Xi;
    int d = y->size;
    double constante = (double)alpha * T * support_espace / (Np * ht * pow(hx, d))
    Xi = pnl_vect_create(d);
    ms0 = 0.0;
    ms3 = 0.0;
    mu0 = 0.0;
    mu3 = 0.0;
    for (i = 0; i < Np; i++)
    {
        T_alea_i = pnl_vect_get(T_alea, i);
        pnl_mat_get_row(Xi, X, i);
        if (test(Xi, T_alea_i, s, y, hx, ht) == 1)
        {
            Ti = (s - T_alea_i) / ht;
            Kti = Kt(Ti);
            Kti_prime = Kt_prime(Ti);
            p = Kti;
            p_prime = Kti_prime;
            prod = 1.0;
            for (j = 0; j < d; j++)
            {
                prod *= Kx((pnl_vect_get(y, j) - pnl_vect_get(Xi, j)) / hx);
            }
            p_prime *= prod;
            p *= prod;
            Vi = pnl_vect_get(V, i);
            ms0 += p * Vi; //calcul de rn

```

```

        ms3 += p_prime * Vi; //calcul de rn' en t
        mu0 += p; //calcul de fn
        mu3 += p_prime; //calcul de fn' en t
    }
}
mu0_cst = mu0 * constante;
g = g0(mu0_cst);
g_prime = g0_prime(mu0_cst);
pnl_vect_free(&Xi);
return constante / (ht) * (ms3 * g + constante * ms0 * mu3 * g_prime);
}

void derive_xx_operateur_noyau(PnlMat *res, PnlMat *X, PnlVect *T_alea, double s
                                PnlVect *y, double hx, double ht, PnlVect *V,
                                double Np, double T, double support_espace)
{
    double Kti, Vi;
    double ms0, ms1k, ms1l, ms2;
    double mu0, mu1k, mu1l, mu2;
    double g, g_prime, g_seconde;
    double prod, p, p_prime_k, p_prime_l, p_seconde;
    double mu0_cst;
    int i, j, k, l;
    double T_alea_i;
    PnlVect *Xi;
    int d = y->size;
    double constante = (double)alpha * T * support_espace / (Np * ht * pow(hx, d))
    double constante_hx = constante / hx;
    pnl_mat_resize(res, d, d);
    Xi = pnl_vect_create(d);
    for (k = 0; k < d; k++)
    {
        for (l = 0; l < k; l++)
        {
            ms0 = 0.0; //rn
            ms1k = 0.0; //rn' en xk
            ms1l = 0.0; //rn' en xl
            mu0 = 0.0; //fn
            mu1k = 0.0; //fn' en xk
            mu1l = 0.0; //fn' en xl

```

```

ms2 = 0.0; //rn'' en xkxl
mu2 = 0.0; //fn'' en xkxl
for (i = 0; i < Np; i++)
{
    T_alea_i = pnl_vect_get(T_alea, i);
    pnl_mat_get_row(Xi, X, i);
    if (test(Xi, T_alea_i, s, y, hx, ht) == 1)
    {
        Kti = Kt((s - T_alea_i) / ht);
        prod = Kti;
        for (j = 0; j < l; j++)
        {
            prod *= Kx((pnl_vect_get(y, j) - pnl_vect_get(Xi, j)) / hx)
        }
        for (j = l + 1; j < k; j++)
        {
            prod *= Kx((pnl_vect_get(y, j) - pnl_vect_get(Xi, j)) / hx)
        }
        for (j = k + 1; j < d; j++)
        {
            prod *= Kx((pnl_vect_get(y, j) - pnl_vect_get(Xi, j)) / hx)
        } // prod represente le produit des Kxj pour
        // j diff de k et Kt
        p = prod * Kx((pnl_vect_get(y, k) - pnl_vect_get(Xi, k)) / hx)
        // produit des Kxj et Kt
        p_prime_k = prod * Kx_prime((pnl_vect_get(y, k) - pnl_vect_get
        //produit des Kxj sauf Kxk fois K_prime_xk
        p_prime_l = prod * Kx_prime((pnl_vect_get(y, l) - pnl_vect_get
        //produit des Kxj sauf Kxl fois K_prime_xl
        p_seconde = prod * Kx_prime((pnl_vect_get(y, k) - pnl_vect_get
        //produit des Kxj sauf Kxk et Kxl fois
        //K_prime_xk fois K_prime_xl
        Vi = pnl_vect_get(V, i);
        ms0 += p * Vi; //calcul de rn
        ms1k += p_prime_k * Vi; //calcul de rn' en xk
        ms1l += p_prime_l * Vi; //calcul de rn' en xl
        ms2 += p_seconde * Vi; //calcul de rn'' en xkxl
        mu0 += p; //calcul de fn
        mu1k += p_prime_k; //calcul de fn' en xk
        mu1l += p_prime_l; //calcul de fn' en xl
        mu2 += p_seconde; //calcul de fn'' en xkxl
    }
}

```

```

    }
}
mu0_cst = mu0 * constante;
g = g0(mu0_cst);
g_prime = g0_prime(mu0_cst);
g_seconde = g0_seconde(mu0_cst);
pnl_mat_set(res, k, l, constante_hx / hx * (ms2 * g + constante * (g_p
pnl_mat_set(res, l, k, pnl_mat_get(res, k, l));
}
ms0 = 0.0; //rn
ms1k = 0.0; //rn' en xk
mu0 = 0.0; //fn
mulk = 0.0; //fn' en xk
ms2 = 0.0; //rn'' en xk
mu2 = 0.0; //fn'' en xk
for (i = 0; i < Np; i++)
{
    T_alea_i = pnl_vect_get(T_alea, i);
    pnl_mat_get_row(Xi, X, i);
    if (test(Xi, T_alea_i, s, y, hx, ht) == 1)
    {
        Kti = Kt((s - T_alea_i) / ht);
        prod = Kti;
        for (j = 0; j < k; j++)
        {
            prod *= Kx((pnl_vect_get(y, j) - pnl_vect_get(Xi, j)) / hx);
        }
        for (j = k + 1; j < d; j++)
        {
            prod *= Kx((pnl_vect_get(y, j) - pnl_vect_get(Xi, j)) / hx);
        }
        // prod represente le produit des Kxj pour
        // j diff de k et Kt
        p = prod * Kx((pnl_vect_get(y, k) - pnl_vect_get(Xi, k)) / hx);
        // produit des Kxj et Kt
        p_prime_k = prod * Kx_prime((pnl_vect_get(y, k) - pnl_vect_get(Xi,
        //produit des Kxj sauf Kxk fois K_prime_xk
        p_seconde = prod * Kx_seconde((pnl_vect_get(y, k) - pnl_vect_get(X
        //produit des Kxj sauf Kxk et fois K_seconde_xk
        Vi = pnl_vect_get(V, i);
        ms0 += p * Vi; //calcul de rn
        ms1k += p_prime_k * Vi; //calcul de rn' en xk
    }
}

```

```

        ms2 += p_seconde * Vi; //calcul de rn'' en xk
        mu0 += p; //calcul de fn
        mu1k += p_prime_k; //calcul de fn' en xk
        mu2 += p_seconde; //calcul de fn'' en xk
    }
}
mu0_cst = mu0 * constante;
g = g0(mu0_cst);
g_prime = g0_prime(mu0_cst);
g_seconde = g0_seconde(mu0_cst);
pnl_mat_set(res, k, k, constante_hx / hx * (ms2 * g + constante * (g_prime
}
pnl_vect_free(&Xi);
}
#endif //PremiaCurrentVersion

```