

## [Help](#)

```
/*
 * Written by David Pommier <david.pommier@gmail.com>
 * INRIA 2009
 */

#include "
href../../../../common/math/equity_pricer/gd_list_h_src.pdfgd_list.h"
#include "pnl/pnl_vector.h"
#include "
href../../../../common/math/equity_pricer/gridsparse_constructor_h_src.pdfgridspars

// To compute sparse grid size with
// recurrence relation \ $f a_{n,d} \ $f
// the numbers of point in level n and dimension d.
// a_{n,d} = a_{n,d-1} + 2 a_{n-1,d}
int Size_GridSparse(int dim, int lev)
{
    if (lev == 0)
        return 1;
    if (dim == 0)
        return 1;
    if (lev == 1)
        return 1;
    if (dim == 1)
        return (1 << lev) - 1;
    return Size_GridSparse(dim - 1, lev) + 2 * Size_GridSparse(dim, lev - 1);
}

// To compute sparse grid size with
// recurrence relation \ $f a_{n,d} \ $f
// the numbers of point in level n and dimension d.
// a_{n,d} = a_{n,d-1} + 2 a_{n-1,d}
int Size_GridSparse_With_Bnd(int dim, int lev)
{
    int pow2 = 2;
    int cdk = dim;
    int sum = Size_GridSparse(dim, lev);
    int k = 1;
```

```

while (k < dim)
{
    //cout<<cdk<<" "<<pow2<<" "<<k<<endl;
    sum += cdk * pow2 * Size_GridSparse(dim - k, lev);
    pow2 *= 2;
    cdk = (1.0 * (dim - (k + 1))) / k;
    k++;
}
sum += pow2 * Size_GridSparse(0, lev);

return sum;
}

extern void complexity_sparse(int d)
{
    int dim = 2, lev = 2;
    int a;
    int b;
    while (lev > 1)
    {
        printf(" level \ n");
        //cin>>lev;
        printf(" dim \ n");
        //cin>>dim;
        a = Size_GridSparse(dim, lev);
        b = Size_GridSparse_With_Bnd(dim, lev);
        printf(" a= %d  b= %d rapport %7.4f \ n", a, b, a * 100.0 / b);
    }
}

static void Search(PremiaSortListSparsePoint *set,
                  PnlVectInt *P,
                  int *current_index,
                  const int Dir,
                  const int Father_Left,
                  const int Father_Right,
                  GridSparse *G)
// Search Node and add this if is not in the Set;

```

```

// If Node in Set, update father in Dir - not computed before this step
{
    int inserted;
    PremiaNodeSparsePoint **current = malloc(sizeof(PremiaNodeSparsePoint *));
    int PP[3] = {Dir, 0, 0};
    inserted = premia_sort_list_sparse_point_find_dicho(set, current, P, *current);
    PP[1] = (*current)->obj->value;
    if (inserted)
        (*current_index)++;
    pnl_hmat_int_set(G->Ind_Father, PP, Father_Left);
    PP[2]++;
    pnl_hmat_int_set(G->Ind_Father, PP, Father_Right);
    free(current);
};

void Print_Set(PremiaSortListSparsePoint *current_set)
// Use to debug
{
    printf(">>>>> Print set \ n");
    premia_sort_list_sparse_point_print(current_set);
    printf(">>>>> End current set ---- \ n");
};

void construct_SP_Grid_In_Level(PremiaSortListSparsePoint *current_set,
                                PnlVectInt *Father_Node,
                                GridSparse *G,
                                int *current_index)
{
    int i, dir, PF[3];
    PnlVectInt *Current_Point = pnl_vect_int_create(G->dim);
    // cout<<" Level increase "<<current_index<<endl;
    for (dir = 0; dir < G->dim; dir++)
    {
        PF[0] = dir;
        for (i = 0; i < Father_Node->size; i++)
        {
            PF[1] = Father_Node->array[i];
            // Index of the root nodes
            pnl_mat_int_get_row(Current_Point, G->Points, PF[1]);
            Current_Point->array[dir] <= 1;
        }
    }
}

```

```

        // extract root node from Grid
        // Compute son of root node in each directions
        // and add it to current_set if needed,
        // else update relation (father relation)
        PF[2] = 0;
        Search(current_set,
                Current_Point, /// Left Son
                current_index, dir,
                pnl_hmat_int_get(G->Ind_Father, PF), PF[1], G);

        Current_Point->array[dir] += 1;
        PF[2] = 1;
        Search(current_set,
                Current_Point, /// Right Son
                current_index, dir,
                PF[1], pnl_hmat_int_get(G->Ind_Father, PF),
                G);
    }
}
pnl_vect_int_free(&Current_Point);
};

void create_grid_sparse_cpp(int dim,
                           int lev,
                           GridSparse *G)
{
    int current_index;
    PnlVectInt *Father_Node;
    PremiaNodeSparsePoint **current;
    int i, j, Tab_Size[3] = {dim, 0, 2};
    current = malloc(sizeof(PremiaNodeSparsePoint *));
    G->dim = dim;
    G->lev = lev;
    G->size = Size_GridSparse(G->dim, G->lev) + 1;
    G->size_in_level = pnl_vect_int_create_from_int(G->lev, 0);
    pnl_vect_int_set(G->size_in_level, lev - 1, G->size);
    Tab_Size[1] = G->size;
    //complexity_sparse(dim);
    // +1 is for root node (0,...,0) for easayer
    // computing hiertonode and inverse
    // the root node is added and V[0]=0,

```

[illegible]