

## Help

```
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2008+2) //The "#els
#else
/*****
*   CPS - A simple C PDE solver                                     *
*                                                                 *
*   Copyright (c) 2007,                                           *
*   Maya Briani          <m.briani@iac.rm.cnr.it>,               *
*   Francesco Ferreri    <francesco.ferreri@gmail.com>,          *
*   Roberto Natalini     <r.natalini@iac.rm.cnr.it>,              *
*   Marco Papi           <m.papi@iac.rm.cnr.it>                   *
*                                                                 *
*****/
#include "
href../../common/math/highdim_solver/cps_pde_integral_term_h_src.pdfcps_pde_i
#include "
href../../common/math/highdim_solver/cps_function_h_src.pdfcps_function.h"
#include "
href../../common/math/highdim_solver/cps_grid_h_src.pdfcps_grid.h"
#include "
href../../common/math/highdim_solver/cps_grid_node_h_src.pdfcps_grid_node.h"
#include "
href../../common/math/highdim_solver/cps_utils_h_src.pdfcps_utils.h"
#include "
href../../common/math/highdim_solver/cps_assertions_h_src.pdfcps_assertions.h

/* static functions */

#define IS_ODD(n) ((n % 2))
#define IS_EVEN(n) (!(n % 2))

/*
static double func_beta(const pde_integral_term *term, int xt){
  REQUIRE("term_not_null", term != NULL);
  REQUIRE("grid_not_null", term->source_grid != NULL);

  double result;
  double c;
  const grid *grid = term->source_grid;
```

```

if(xt == grid_iterator_first(grid,X_DIM) || xt == grid_iterator_last(grid,X_DIM))
c = 1;
}
else{
if(IS_ODD(xt)){
c = 4;
}
else{
c = 2;
}
}
result = term->source_grid->delta[X_DIM]/3.0 * c;
return result;
}
*/

```

```

static double func_beta2(const pde_integral_term *term, int xt)
{
double result;
const grid *grid;
REQUIRE("term_not_null", term != NULL);
REQUIRE("grid_not_null", term->source_grid != NULL);
REQUIRE("valid_x_tick", xt >= grid_iterator_first(term->source_grid, X_DIM) &&
xt <= grid_iterator_last(term->source_grid, X_DIM));

grid = term->source_grid;

if (xt == grid_iterator_first(grid, X_DIM) || xt == grid_iterator_last(grid, X_DIM))
{
result = 0.5 * term->source_grid->delta[X_DIM];
}
else
{
result = term->source_grid->delta[X_DIM];
}

return result;
}

```

```

static double func_integral(const pde_integral_term *term, double x, double z)
{

```

```

double result;
double a = term->alpha;
double m = term->m;
const double M_PI2 = 6.2831853071795864769252866;
REQUIRE("term_not_null", term != NULL);
REQUIRE("x_not_zero", x != 0.0);

if (z == 0.0 || z == 1.0)
{
    result = 0.0;
}
else
{
    result = 1.0 / (a * sqrt(M_PI2)) * (1.0 - x) / (z * pow((1.0 - z), 2.0))
        * exp(-0.5 / pow(a, 2.0) * pow((log(z / (1.0 - z)) + log((1.0 -
/*
    result = 1.0/(a * sqrt(M_PI2)) * 1.0/(z * (1.0 - z))
        * exp(-0.5/pow(a,2.0)* pow((log(z/(1.0 - z)) + log((1.0 - x)/x) - m), 2.0
    */
}
return result;
}

/* public interface */

int pde_integral_term_create(pde_integral_term **term)
{
    STANDARD_CREATE(term, pde_integral_term);
    return OK;
}

int pde_integral_term_destroy(pde_integral_term **term)
{
    STANDARD_DESTROY(term);

    return OK;
}

```

```

int pde_integral_term_set_lambda(pde_integral_term *term, double l)
{
    REQUIRE("term_not_null", term != NULL);

    term->lambda = l;
    return OK;
}

int pde_integral_term_set_alpha(pde_integral_term *term, double a)
{
    REQUIRE("term_not_null", term != NULL);

    term->alpha = a;
    return OK;
}

int pde_integral_term_set_m(pde_integral_term *term, double m)
{
    REQUIRE("term_not_null", term != NULL);

    term->m = m;
    return OK;
}

int pde_integral_term_set_grid(pde_integral_term *term, const grid *grid)
{
    REQUIRE("term_not_null", term != NULL);
    REQUIRE("grid_not_null", grid != NULL);
    REQUIRE("grid_is_rescaled", grid->is_rescaled);

    term->source_grid = grid;
    return OK;
}

double pde_integral_term_evaluate(const pde_integral_term *term, const grid_node
{
    int lt;
    double z;
    grid_node *unode;
    const grid *grid ;
    double x ;

```

```

int xt;
int yt;
double result ;
/* evaluate integral term for i */
REQUIRE("term_not_null", term != NULL);
REQUIRE("node_not_null", node != NULL);
REQUIRE("grid_is_set", term->source_grid != NULL);

grid = term->source_grid;
x = node->value[X_DIM];
xt = node->tick[X_DIM];
yt = node->tick[Y_DIM];
result = 0.0;

if (x == 0.0)
{
    grid_loose_item(grid, xt, yt, &unode);
    result = term->lambda * V_GetCmp(U, unode->order);
    grid_node_destroy(&unode);
}
else if (x == 1.0)
{
    grid_loose_item(grid, xt, yt, &unode);
    result = term->lambda * exp(0.5 * pow(term->alpha, 2) + term->m)
        * V_GetCmp(U, unode->order);
    grid_node_destroy(&unode);
}
else
{
    for (lt = grid_iterator_first(grid, X_DIM);
        lt <= grid_iterator_last(grid, X_DIM); lt++)
    {
        z = grid->min_value[X_DIM] + ((double)lt) * grid->delta[X_DIM];
        grid_loose_item(grid, lt, yt, &unode);
        result += term->lambda * func_beta2(term, lt)
            * func_integral(term, x, z) * V_GetCmp(U, unode->order);
        grid_node_destroy(&unode);
    }
}
return result;

```

```
}  
/* end -- pde_integral_term.c */  
  
#endif //PremiaCurrentVersion
```