계산과학 이론 및 실습2

Project 1

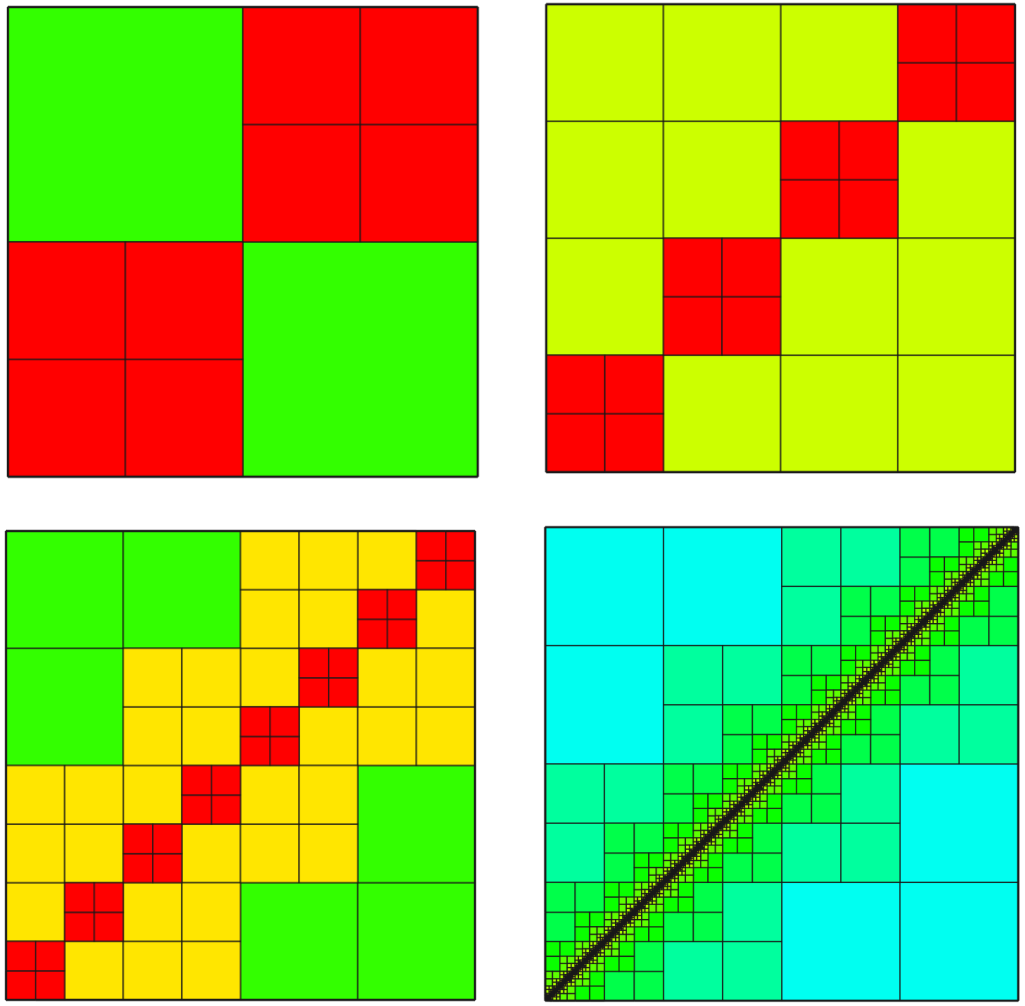
1. Mandelbrot set
2. mean value formulas for Laplace's eq.

made by Jeongwoo Kim

1.

Mandelbrot Set (fractal)

simple fractal



```
void first_grid()
{
  Triangulation<2> triangulation;

  GridGenerator::hyper_cube(triangulation);
  triangulation.refine_global(1);

  for (unsigned int step = 0; step<10; ++step){
    for (auto &cell : triangulation.active_cell_iterators())
    {
      unsigned int score = 0;
      for (const auto v : cell->vertex_indices()){

        if (cell->vertex(v)[0] == cell->vertex(v)[1]){
          score += 1;
          }

        if (score == 2){
          cell->set_refine_flag();
        }
      }
    }
    triangulation.execute_coarsening_and_refinement();
  }

  std::ofstream out("fractal.svg");
  GridOut       grid_out;
  grid_out.write_svg(triangulation, out);
  std::cout << "Grid written to fractal.svg" << std::endl;
}
```

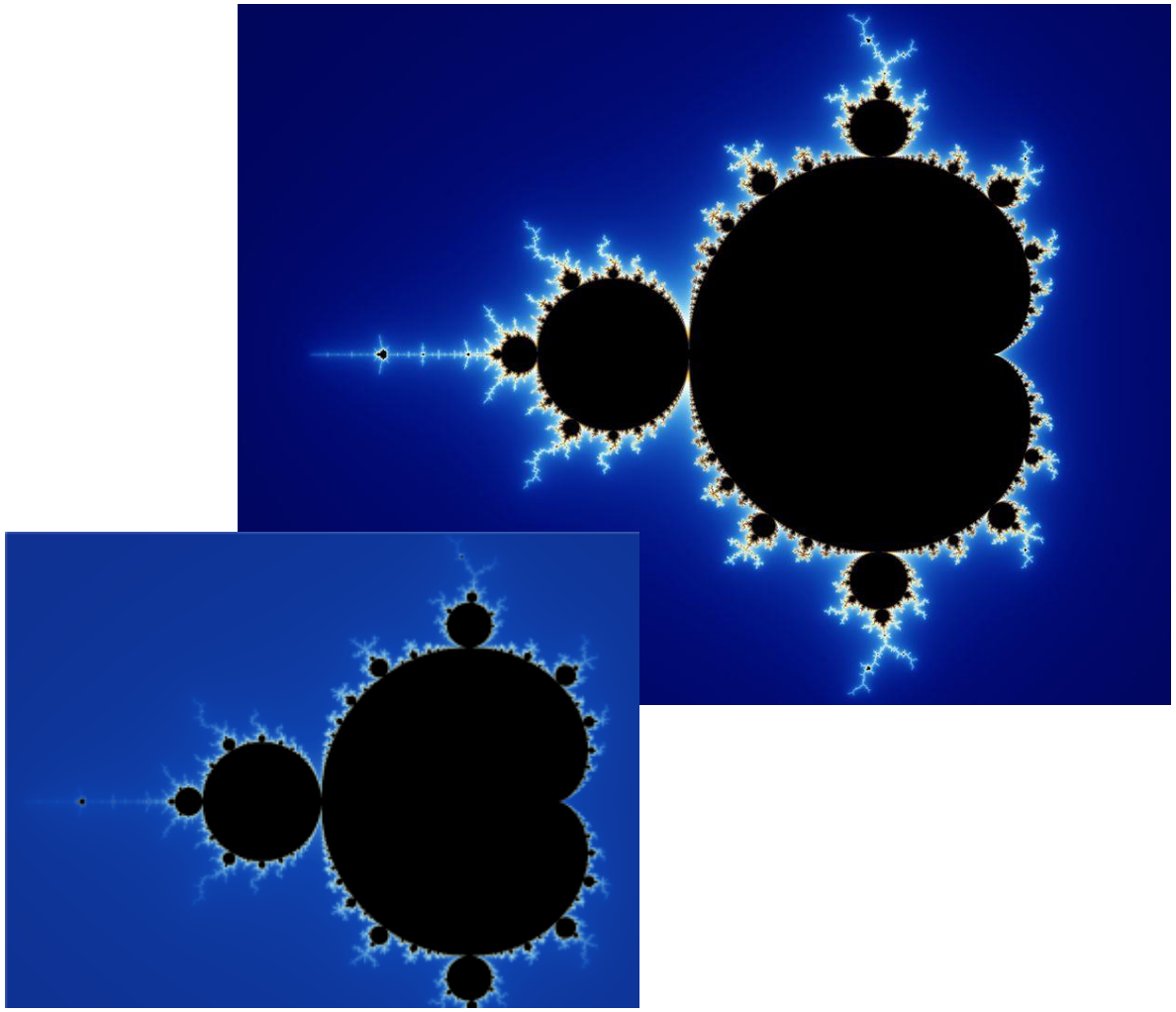
Part 1 : Mandelbrot set

The Mandelbrot set is a set of complex numbers such that the sequence defined by the following recurrence formula has a non-diverging property.

$$z_0 = 0$$
$$z_{n+1} = z_n^2 + c$$

If the formula is expressed only by real numbers rather than complex numbers,
the equation changes as follows.

$$x_{n+1} = x_n^2 - y_n^2 + x_0$$
$$y_{n+1} = 2x_n y_n + y_0$$
$$x_n^2 + y_n^2 < 2^2$$



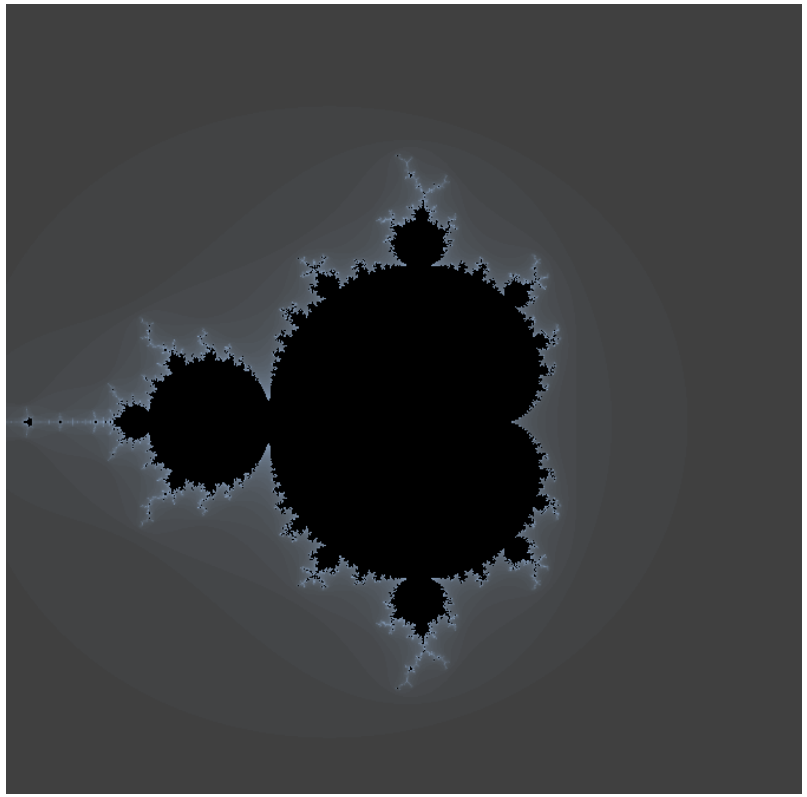
Code

```
void second_grid(unsigned int n)
{
  Triangulation<2> triangulation;

  GridGenerator::hyper_cube(triangulation, -2, 2);
  triangulation.refine_global(2);

  for (unsigned int step = 0; step < n ; ++step){
    for (auto &cell : triangulation.active_cell_iterators())
      {
        unsigned int score = 0;
        for (const auto v : cell->vertex_indices()){

          double x = cell->vertex(v)[0];
          double y = cell->vertex(v)[1];

          double x1 = pow(x,2) - pow(y,2) +x;
          double y1 = 2*x*y + y;
```

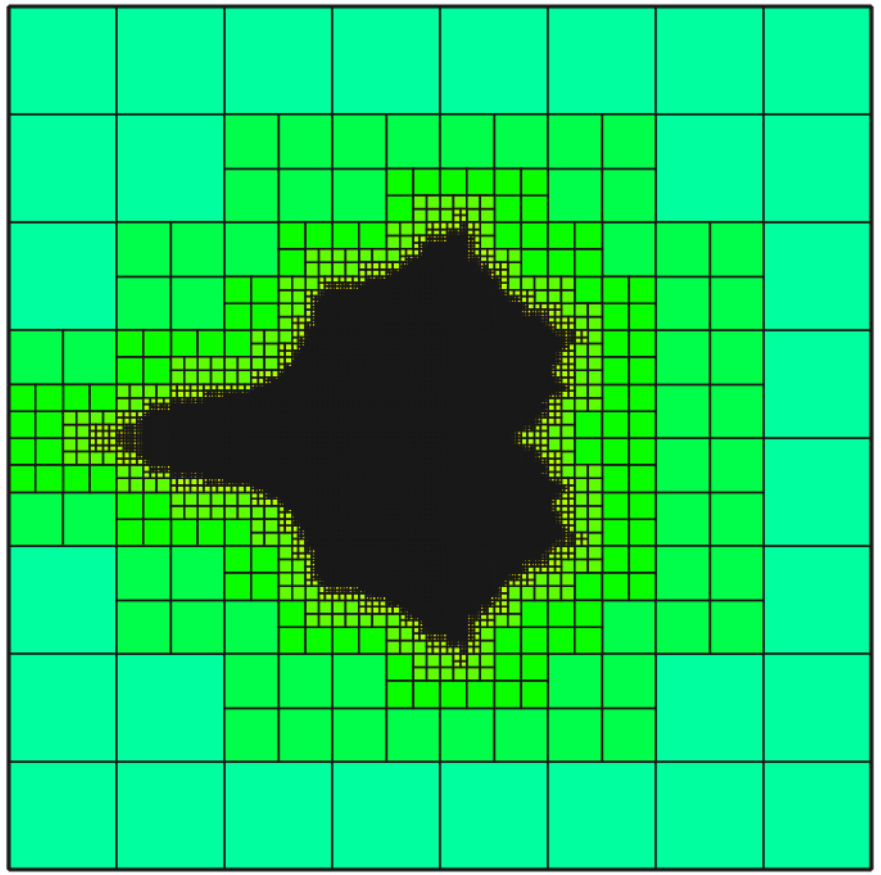
```
          for (unsigned int num=0; num <n ; ++num){
            x1 = pow(x1,2) - pow(y1,2) +x;
            y1 = 2*x1*y1 +y;

            if ((pow(x1,2) + pow(y1,2)) >= 4){
              break;
            }

            if (num == n-1) score +=1;
          }
        }
        if (score >= 3){
          cell->set_refine_flag();
        }
      }
    triangulation.execute_coarsening_and_refinement();
  }

  std::ofstream out("Mandelbrot_set1.svg");
  GridOut       grid_out;
  grid_out.write_svg(triangulation, out);
  std::cout << "Grid written to Mandelbrot_set.svg" << std::endl;
}
```

Result

Real Result(n=32)



My Result(n=6)



2.

Mean value Formulas for Laplace's eq.

Part 2 : Mean value Formulas for Laplace's eq.

Poisson's eq. $\nabla^2 V = -\frac{\rho}{\varepsilon_0}$

Laplace's eq. $\nabla^2 V = 0$

V : Electric Potential
ρ : Charge Density
ε_0 : Vacuum permittivity

When there is no charge(density) in the area, the potential is determined by the average of the surrounding values around that point.
= There is no maximum or minimum point inside.

Part 2 : Mean value Formulas for Laplace's eq.

Consider a square domain without any sources. The Poisson equation reduces to the Laplace equation $(\rho = 0)$ inside the domain. The boundary conditions are indicated in figure 1 below. We choose $L = 1$, $a = 0$ and $b = 1$ (with the necessary dimensions) without loss of generality.

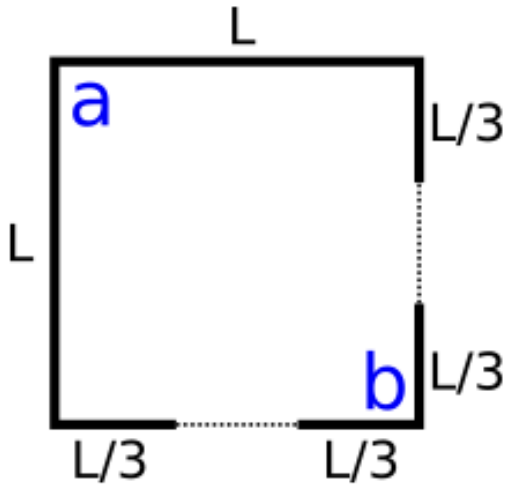


Figure 1: *Setup used in the example. The value of ψ along the solid lines are a and b as indicated. ψ vary evenly between a and b along the dotted lines.*

This setup can for example model a conductor (the upper left part, a) and a conductor with a uniformly distributed charge (the lower right part, b). The electric field is in turn given by $\mathbf{E} = \nabla\psi$.

From *"electricity and magnetism" class(2022-2, 김도현)* in SNU

Code

```
void Step3::make_grid()
{
  GridGenerator::hyper_cube(triangulation, -1.5, 1.5);

  for (auto &face : triangulation.active_face_iterators()){
    if (std::fabs(face->center()(0) - (1.5)) < 1e-12)
      face->set_boundary_id(1);
    if (std::fabs(face->center()(1) - (-1.5)) < 1e-12)
      face->set_boundary_id(2);
  }
  triangulation.refine_global(6);

  std::cout << "Number of active cells: " << triangulation.n_active_cells()
            << std::endl;
}
```

Code

```
template <int dim, int axis>
class MyBoundary : public Function<dim>
{
public:
  virtual double value(const Point<dim> & p,
                       const unsigned int component = 0) const override;
};

template <int dim, int axis>
double MyBoundary<dim, axis>::value(const Point<dim> &p,
                                    const unsigned int /*component*/) const
{
  if (p.operator()(axis) < -0.5) return 1*axis;
  else if (p.operator()(axis) > 0.5) return 1-axis;
  else return p.operator()(axis)*pow(-1,axis)+0.5;
}
```

Code

```
void Step3::assemble_system()
{
  QGauss<2> quadrature_formula(fe.degree + 1);
  FEValues<2> fe_values(fe,
                        quadrature_formula,
                        update_values | update_gradients | update_JxW_values);

  const unsigned int dofs_per_cell = fe.n_dofs_per_cell();

  FullMatrix<double> cell_matrix(dofs_per_cell, dofs_per_cell);
  Vector<double>     cell_rhs(dofs_per_cell);

  std::vector<types::global_dof_index> local_dof_indices(dofs_per_cell);

  for (const auto &cell : dof_handler.active_cell_iterators())
    {
      fe_values.reinit(cell);

      cell_matrix = 0;
      cell_rhs    = 0;
```

Code

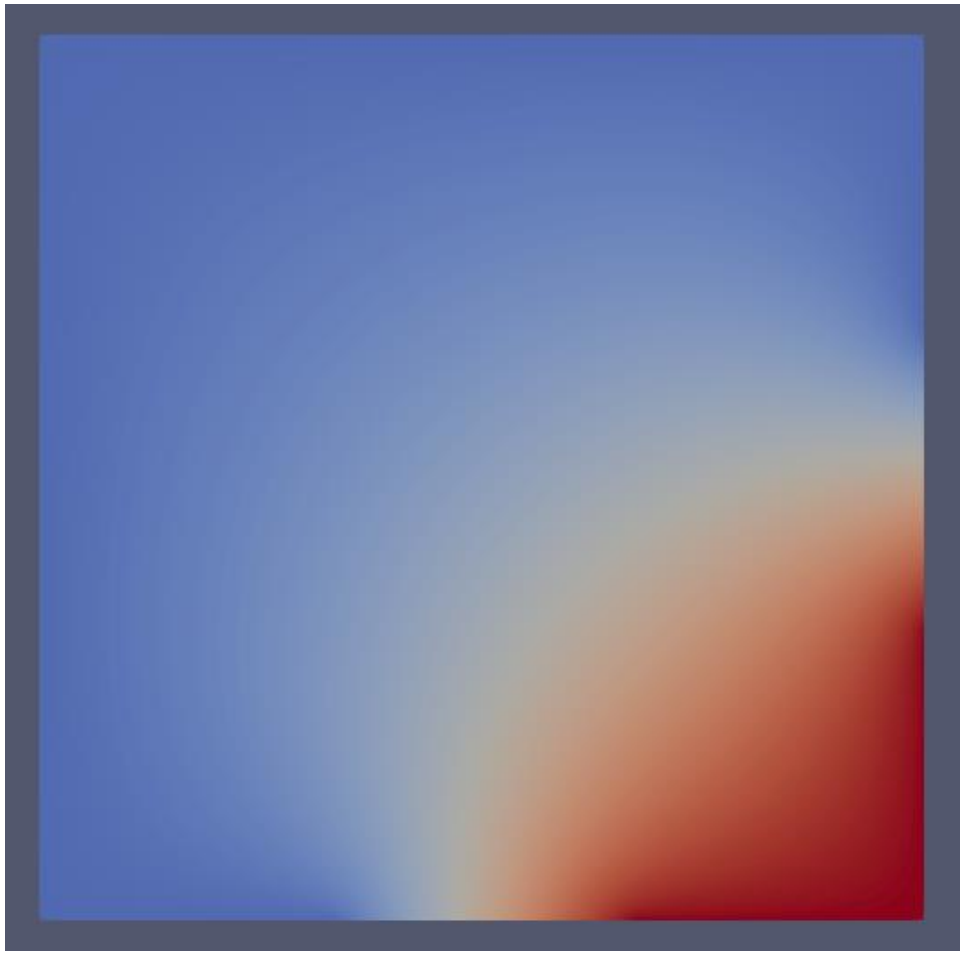
```
for (const unsigned int q_index : fe_values.quadrature_point_indices())
  {
    for (const unsigned int i : fe_values.dof_indices())
      for (const unsigned int j : fe_values.dof_indices())
        cell_matrix(i, j) +=
          (fe_values.shape_grad(i, q_index) * // grad phi_i(x_q)
           fe_values.shape_grad(j, q_index) * // grad phi_j(x_q)
           fe_values.JxW(q_index));           // dx

    for (const unsigned int i : fe_values.dof_indices())
      cell_rhs(i) += (fe_values.shape_value(i, q_index) * // phi_i(x_q)
                      (0) *                               // f(x_q)
                      fe_values.JxW(q_index));            // dx
  }
cell->get_dof_indices(local_dof_indices);

for (const unsigned int i : fe_values.dof_indices())
  for (const unsigned int j : fe_values.dof_indices())
    system_matrix.add(local_dof_indices[i],
                      local_dof_indices[j],
                      cell_matrix(i, j));

for (const unsigned int i : fe_values.dof_indices())
  system_rhs(local_dof_indices[i]) += cell_rhs(i);
}
```

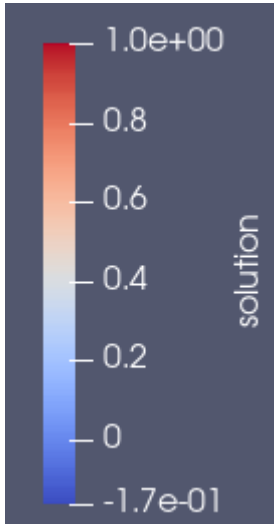
Code

```
  std::map<types::global_dof_index, double> boundary_values;
  VectorTools::interpolate_boundary_values(dof_handler,
                                           0,
                                           Functions::ZeroFunction<2>(),
                                           boundary_values);
  VectorTools::interpolate_boundary_values(dof_handler,
                                           1,
                                           MyBoundary<2,1>(),
                                           boundary_values);
  VectorTools::interpolate_boundary_values(dof_handler,
                                           2,
                                           MyBoundary<2,0>(),
                                           boundary_values);
  MatrixTools::apply_boundary_values(boundary_values,
                                     system_matrix,
                                     solution,
                                     system_rhs);
}
```

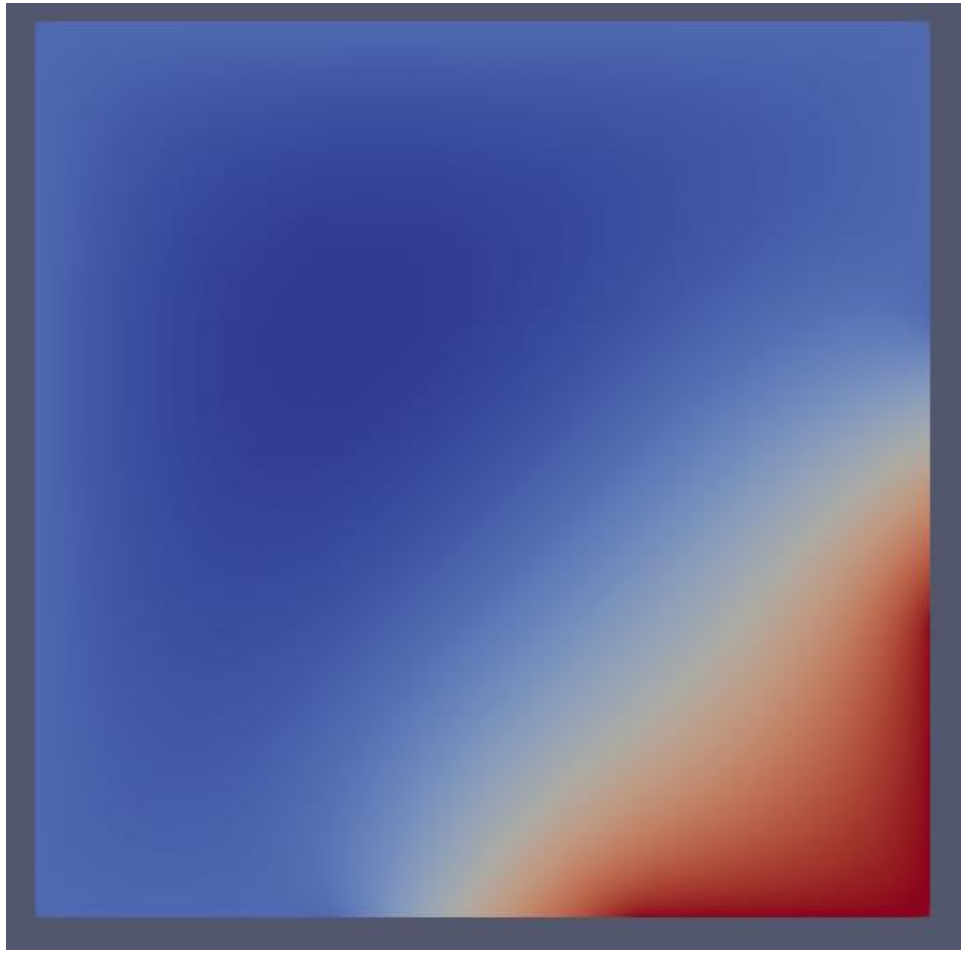
Result

$f = 0$ (Laplace's eq.)





$f = -0.5$ (non-Laplace's eq.)



계산과학 이론 및 실습2

THANK YOU

made by Jeongwoo Kim