

# Program construction in C++ for Scientific Computing

## Teacher: Michael Hanke

Ilian Häggmark  
mail ilianh@kth.se

Andreas Karlsson  
mail andreas.a.karlsson@ki.se

December 29, 2016

## Project 4

### Task 1 - Redesign Domain class

The Domain class is given a few new short methods to enable access to variables from the outside. It is moreover split up in a source file and header file.

### Task 2 - GFkt class

The GFkt class holds two important objects. A Domain object that describes the grid and a matrix object that contain function values of a function  $fp$  on the grid. Apart from the basic structure with constructor/copy-constructor etc. the GFkt class contains methods to perform basic (point-wise) arithmetic operation on the matrix containing the function values. Finally the class have methods for calculating approximative discrete partial derivatives.

The first partial derivatives in  $x$  and  $y$  are calculated as.

$$\frac{\partial}{\partial x} u(x_i, y_j) = \frac{u(x_{i+1}, y_j) - u(x_{i-1}, y_j)}{x_{i+1} - x_{i-1}} \quad \frac{\partial}{\partial y} u(x_i, y_j) = \frac{u(x_i, y_{j+1}) - u(x_i, y_{j-1})}{y_{j+1} - y_{j-1}}$$

These are central derivatives that yield good accuracy, but they cannot be applied to all borders (first and last column for  $x$  and firsts and last row for  $y$ ). We therefore need one-sided derivatives to estimate the border derivatives.

$$\frac{\partial}{\partial x} u(x_i, y_j) = \frac{u(x_{i+1}, y_j) - u(x_i, y_j)}{x_{i+1} - x_i} \quad \frac{\partial}{\partial y} u(x_i, y_j) = \frac{u(x_i, y_{j+1}) - u(x_i, y_j)}{y_{j+1} - y_j}$$

This would be for the first column and first row respectively. The one-sided derivatives have lower accuracy compared to the central derivatives. We therefore derive two a simple expression for a one-sides approximate derivative on a nonequidistant grid with Taylor expansion.

$$u(x + h_1) = u(x) + u'(x) \cdot h_1 + \frac{1}{2} u''(x) \cdot h_1^2 + \mathcal{O}(h^3)$$

$$u(x + h_2) = u(x) + u'(x) \cdot h_2 + \frac{1}{2} u''(x) \cdot h_2^2 + \mathcal{O}(h^3)$$

Multiply first equation with  $h_2^2$  and the second with  $-h_1^2$ . Then sum is them:

$$h_2^2 u(x + h_1) - h_1^2 u(x + h_2) = u(x)(h_2^2 - h_1^2) + u'(x)(h_1 h_2^2 - h_2 h_1^2) + \mathcal{O}(h^3)$$

We neglect the higher order terms and solve for  $u'(x)$

$$u'(x) = \frac{-u(x)(h_2^2 - h_1^2) + h_2^2 u(x + h_1) - h_1^2 u(x + h_2)}{h_1 h_2^2 - h_2 h_1^2}$$

This 3-stencil approximation is marginally more complicated but gives much high accuracy.

For the Laplacian,  $\Delta$ , we can use a similar approach. The laplacian of a function  $u(x, y)$  is simply  $\frac{\partial^2}{\partial x^2} u(x, y) + \frac{\partial^2}{\partial y^2} u(x, y)$ . The two second order derivatives can be calculated separately and then added together. Each second order derivative is calculated with a central difference,

$$\frac{\partial^2}{\partial x^2} u(x_i, y_j) = \frac{u_x(x_{i+1}, y_j) - u_x(x_{i-1}, y_j)}{x_{i+1} - x_{i-1}} \quad \frac{\partial^2}{\partial y^2} u(x_i, y_j) = \frac{u_y(x_i, y_{j+1}) - u_y(x_i, y_{j-1})}{y_{j+1} - y_{j-1}}$$

where the only difference from the first order derivatives is that here the first order derivative is the input instead of the function. Also here we get the problem with border values and one-sided derivatives derived with Taylor expansion are used again. This time however we use a 4-stencil to achieve enough accuracy.

$$\begin{aligned} u(x + h_1) &= u(x) + u'(x) \cdot h_1 + \frac{1}{2} u''(x) \cdot h_1^2 + \frac{1}{6} u'''(x) \cdot h_1^3 + \mathcal{O}(h^4) \\ u(x + h_2) &= u(x) + u'(x) \cdot h_2 + \frac{1}{2} u''(x) \cdot h_2^2 + \frac{1}{6} u'''(x) \cdot h_2^3 + \mathcal{O}(h^4) \\ u(x + h_3) &= u(x) + u'(x) \cdot h_3 + \frac{1}{2} u''(x) \cdot h_3^2 + \frac{1}{6} u'''(x) \cdot h_3^3 + \mathcal{O}(h^4) \end{aligned}$$

We eliminate the first order and third order derivatives, neglect the high order terms and solve for the second order derivative. After some simple, but somewhat lengthy algebraic manipulations we get

$$\begin{aligned} u''(x) &= 2 \cdot [-u(x)((h_3^3 - h_1^3)(h_2 h_3^3 - h_3 h_2^3) - (h_3^3 - h_2^3)(h_1 h_3^3 - h_3 h_1^3)) \\ &\quad + (h_2 h_3^3 - h_3 h_2^3)(h_3^3 u(x + h_1) - h_1^3 u(x + h_3)) - (h_1 h_3^3 - h_3 h_1^3)(h_3^3 u(x + h_2) - h_2^3 u(x + h_3))] \\ &\quad / [(h_1^2 h_3^3 - h_3^2 h_1^3)(h_2 h_3^3 - h_3 h_2^3) - (h_2^2 h_3^3 - h_3^2 h_2^3)(h_1 h_3^3 - h_3 h_1^3)] \end{aligned}$$

This expression gives a reasonable accuracy (depends on application of course).

### Task 3 - Discrete differential operators

The function  $u(x, y)$  and its derivatives are shown below

$$\begin{aligned} u(x, y) &= \sin\left(\left(\frac{x}{10}\right)^2\right) \cos(x/10) + y \\ \frac{\partial}{\partial x} u(x, y) &= \frac{2}{100} x \cos\left(\left(\frac{x}{10}\right)^2\right) \cos\left(\frac{x}{10}\right) - \frac{1}{10} \sin\left(\left(\frac{x}{10}\right)^2\right) \sin\left(\frac{x}{10}\right) \\ \frac{\partial}{\partial y} u(x, y) &= \frac{\partial}{\partial y} y = 1 \\ \frac{\partial^2}{\partial x^2} u(x, y) &= \frac{2}{100} \cos\left(\left(\frac{x}{10}\right)^2\right) \cos\left(\frac{x}{10}\right) - \left(\frac{2}{100} x\right)^2 \sin\left(\left(\frac{x}{10}\right)^2\right) \cos\left(\frac{x}{10}\right) \\ &\quad - \frac{1}{250} x \cos\left(\left(\frac{x}{10}\right)^2\right) \sin\left(\frac{x}{10}\right) - \frac{1}{100} \sin\left(\left(\frac{x}{10}\right)^2\right) \cos\left(\frac{x}{10}\right) \\ \frac{\partial^2}{\partial y^2} u(x, y) &= 0 \\ \Delta u(x, y) &= \frac{\partial^2}{\partial x^2} u(x, y) + \frac{\partial^2}{\partial y^2} u(x, y) = \frac{\partial^2}{\partial x^2} u(x, y) \end{aligned}$$

The code below plot the algebraic expressions in MATLAB. Figure 1 shows the output.

```

1  % Algebraic expressions of first and second partial derivatives in x
2
3  x = linspace(-10,5,50);           % length set to 50 to agree with data from GFkt
4
5  figID = figure(101);               % open figure
6
7  ux = sin((x./10).^2).*cos(x./10); % x-component of u(x,y)
8
9  dxu = 2/100.*x.*cos((x./10).^2).*cos(x./10) ... % first partial derivative
10      - 1/10*sin((x./10).^2).*sin(x./10);
11
12  dxxu = 2/100*cos((x./10).^2).*cos(x./10) ... % second partial derivative
13      - (2/100.*x).^2.*sin((x./10).^2).*cos(x./10) ...
14      - 1/250*cos((x./10).^2).*x.*sin(x./10) ...
15      - 1/100*sin((x./10).^2).*cos(x./10);
16
17  subplot(1,3,1); plot(x,ux);
18  subplot(1,3,2); plot(x,dxu);           % plot result in subplot
19  subplot(1,3,3); plot(x,dxxu);

```

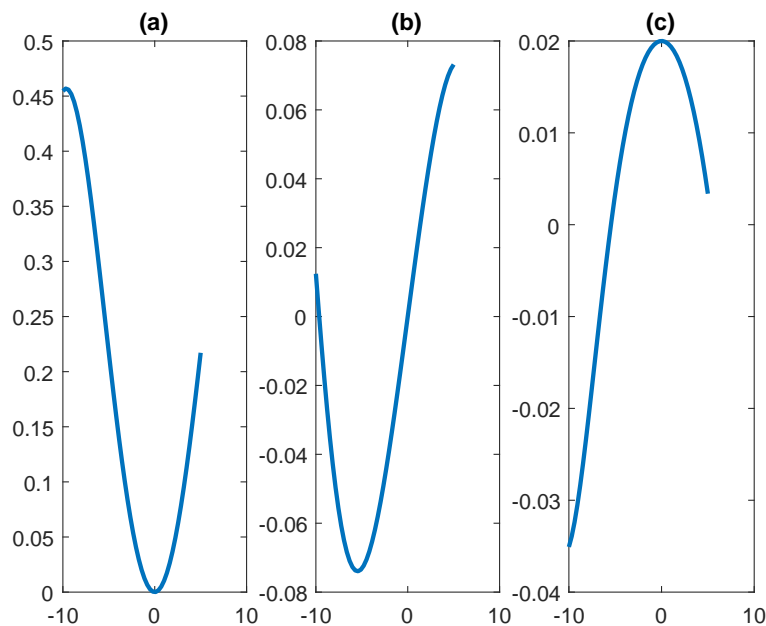


Figure 1: Plot of algebraic expressions in MATLAB: (a)  $x$ -component of  $u(x,y)$ . (b) First partial derivative of  $u(x,y)$  in  $x$ . (c) Second partial derivative of  $u(x,y)$  in  $x$ .

With the approximative derivatives implemented in task 2 we can now plot the algebraic derivatives alongside the C++ approximations (Fig 2 and 3). We note with no surprise that the approximative derivatives have a good agreement with the algebraic derivatives in all places except the borders. The agreement is so good that the green line showing the approximation is hard to discern from the red line. A blue error line with a separate axis is therefore shown. As stated above accuracy is not an absolute thing in real applications, but must be set in relation to something else.

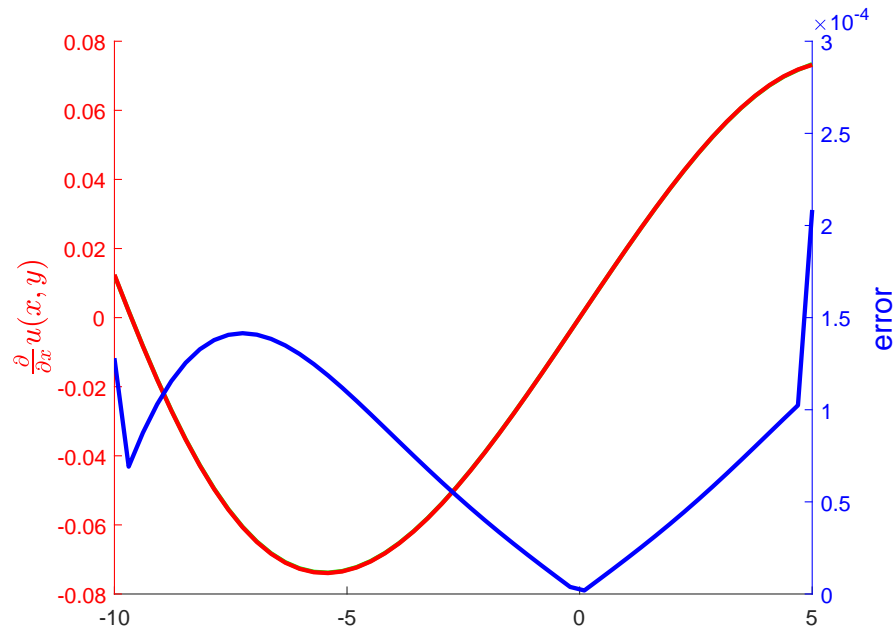


Figure 2: algebraic first order x-derivative (red), approximation (green), and error (blue).

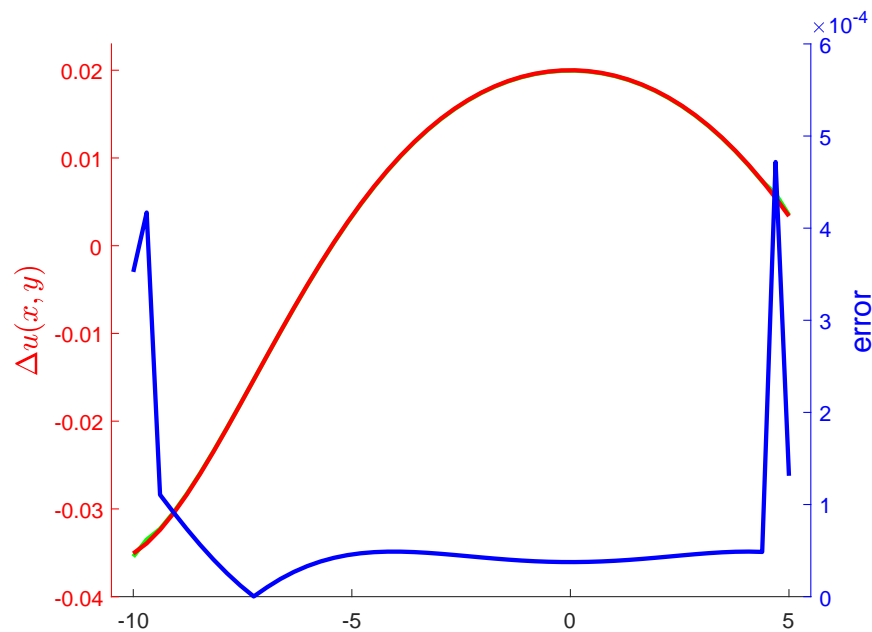


Figure 3: algebraic second order x-derivative (red), approximation of Laplacian (green), and error (blue).