Numerical Algorithms for Visual Computing II 2010/11
Example Solutions for Assignment 3

Problem 1 (Analysis in 1-D)

1. Let us first have a look at the function $u''(x) = 0$. A proper primitive (Stammfunktion) for this would be $u'(x) = a$ for some constant c and a primitive for this is $u(x) = ax + b$ for another constant b. Now, we need to consider the given boundaries.
   As $u(0) = k_1$, we can calculate $u(0) = a \cdot 0 + b \stackrel{!}{=} k_1$. Therefore, we can set $b = k_1$. Furthermore we have to consider also $u(1) = k_2$. Plugging this into our primitive, we have $u(1) = a \cdot 1 + k_1 = a + k_1 \stackrel{!}{=} k_2$. By setting $a = k_2 - k_1$ we have found a function that satisfies the given constraints, i.e. the exact solution given by the BVP is

   $$u(x) = (k_2 - k_1)x + k_1.$$

   (a) Is a reasonable problem, as there exist a proper solution for this problem with $u(x) = x + 2$.

   (b) Is a reasonable problem, as there exist a proper solution for this problem with $u(x) = 3x + 1$

   (c) Problem is not reasonable. Considering the primitives for $u''(x) = 0$, i.e. $u'(x) = a$ and $u(x) = ax + b$, the sole boundary condition $u(1) = 4$ results in $a + b = 4$. This however does not give any information on the values a and b, i.e. the conditions on the boundaries have not been given properly.

   (d) Is a reasonable problem, as there exist a proper solution for this problem with $u(x) = x + 3$.

   (e) Problem is not reasonable. The additional constraint $u''(1) = 1$ is in direct violation of the initial condition $u''(x) = 0$.

Problem 2 (Explicit Coding)

1. For the explicit time marching algorithm, let us consider the already discretized version of the 1-D Laplace equation $\Delta u = 0$, i.e.

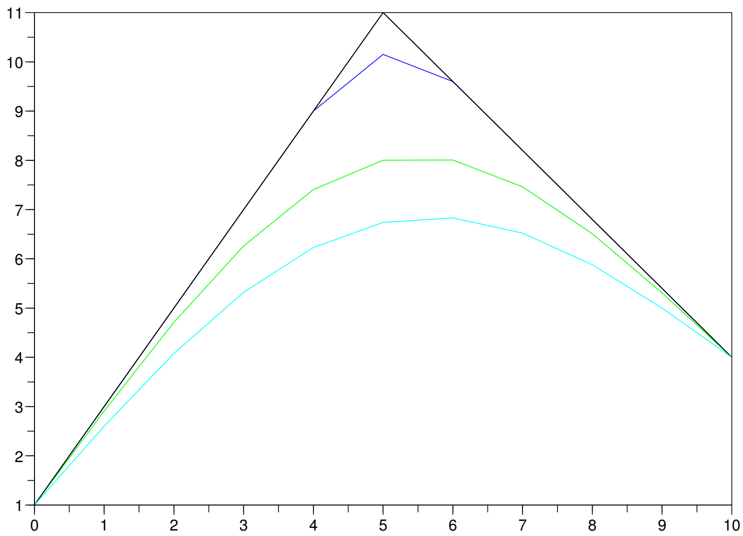
$$\begin{aligned} \frac{u_j^{n+1} - u_j^n}{\Delta t} &= \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{\Delta x^2} \\ \Leftrightarrow \quad u_j^{n+1} &= \left(1 - 2\frac{\Delta t}{\Delta x^2}\right) u_j^n + \frac{\Delta t}{\Delta x^2} u_{j-1}^n + \frac{\Delta t}{\Delta x^2} u_{j+1}^n \end{aligned}$$

By considering our initial vector

$$u^0 = (1, 3, 5, 7, 9, 11, 9.6, 8.2, 6.8, 5.4, 4)^\top,$$

we have the following results for $\tau = 0.25$ after $t = 0.25$ (blue), $t = 2.5$ (green) and $t = 5$ (turquoise) iterations respectively:

$$\begin{aligned} u^1 &= (1, 3, 5, 7, 9, 10.15, 9.6, 8.2, 6.8, 5.4, 4)^\top, \\ u^{10} &= (1, 2.9, 4.7, 6.3, 7.4, 8, 8, 7.5, 6.5, 5.3, 4)^\top \\ u^{20} &= (1, 2.6, 4.1, 5.3, 6.2, 6.7, 6.8, 6.5, 5.9, 5, 4)^\top \end{aligned}$$



One can see that the second order derivative scheme that we used is only smoothing along the edges and it preserves structures that are already linear.

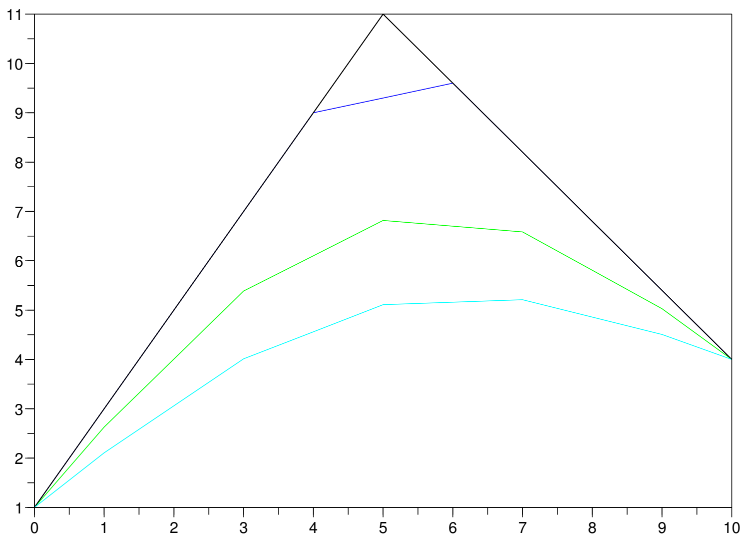
2. The Jacobi iteration scheme can be derived quite easily from

$$\begin{aligned} 0 &= \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{\Delta x^2} \\ \Leftrightarrow \quad u_j^n &= \frac{u_{j-1}^n + u_{j+1}^n}{\Delta x^2} \end{aligned}$$

In general, this boils down to the averaging of both neighbouring pixels. From this we can get the results

$$\begin{aligned} u^1 &= (1, 3, 5, 7, 9, 9.3, 9.6, 8.2, 6.8, 5.4, 4)^\top, \\ u^{10} &= (1, 2.6, 4, 5.4, 6.1, 6.8, 6.7, 6.6, 5.8, 5, 4)^\top \\ u^{20} &= (1, 2.1, 3.1, 4, 4.6, 5.1, 5.2, 5.2, 4.9, 4.5)^\top. \end{aligned}$$

For the entire code implementation, see also the scilab file.



The graphs are the original function (black), after 1 (blue), 10 (green) and 20 (turquoise) iterations. It should be noted that the explicit time marching algorithm converges for $\Delta t = 0.5$ (and for $\Delta x = 1$) to the Jacobi method in this example.

Problem 3 (Direct Coding) The crucial part of this exercise is the derivation of the sought linear system of equations. For our given vector, we know that the boundary values u_0 and u_{10} should remain constant, and from our observation of the first exercise, we would expect a linear function as a result. From this we can conclude, that out of 11 possible equations, 2 are already fixed. We only have to concentrate on the unknowns $u_1, \ldots, u_9$. By use of the equation

$$0 = u_{j+1}^k - 2u_j^k + u_{j-1}^k \qquad j = 1, \ldots, 9,$$

we have the sought system of equations already. We only need to take care at the boundaries, i.e.

$$\begin{aligned} -2u_1^n + u_2^n &= -u_0^n \\ -2u_9^n + u_8^n &= -u_{10}^n, \end{aligned}$$

which gives us the system

$$\begin{pmatrix} -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 \end{pmatrix} u^\infty = \begin{pmatrix} -u_0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -u_{10} \end{pmatrix}$$

This can be easily solved via the Thomas algorithm with the result

$$u^\infty = (1, 1.3, 1.6, 1.9, 2.2, 2.5, 2.8, 3.1, 3.4, 3.7, 4)^\top,$$

which is the linear function we have been looking for. (see the scilab-file for more details on the programming side).

Problem 4 (Implicit Coding) For this problem, we consider the implicit Laplace discretisation

$$\frac{u_j^{n+1}-u_j^n}{\Delta t} = \frac{u_{j+1}^{n+1}-2u_j^{n+1}+u_{j-1}^{n+1}}{\Delta x^2}.$$

This gives us after some computation

$$u_j^n = \left(1+2\frac{\Delta t}{\Delta x^2}\right)u_j^{n+1} - \frac{\Delta t}{\Delta x^2}u_{j-1}^{n+1} - \frac{\Delta t}{\Delta x^2}u_{j+1}^{n+1}.$$

Again, we have to take extra precaution at the boundaries, so our implicit scheme for our problem is as follows:

$$\left(I - \frac{\Delta t}{\Delta x^2}\begin{pmatrix} -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 \end{pmatrix}\right) u^{n+1} = \begin{pmatrix} u_1 + \frac{\Delta t}{\Delta x^2}u_0 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \\ u_9 + \frac{\Delta t}{\Delta x^2}u_{10} \end{pmatrix}$$

As for the experiments, one can see that one can use time step sizes of arbitrary length for the Δt in the implicit scheme, whereas for the explicit scheme, $\Delta t \leq 0.5$ in order to be stable, or else the scheme diverges.

Problem 5 (Jacobi Strikes Back)

1. Similar to the central difference approximation of the second derivative u_{xx} at point $j\Delta x$, we do the same thing for the backward difference approximation

   $$u_{xx}(j\Delta x) \approx \frac{u_x(j\Delta x) - u_x((j-1)\Delta x)}{\Delta x}$$

   The values $u_x(j\Delta x)$ and $u_x((j-1)\Delta x)$ can be approximated by

   $$\frac{u_j - u_{j-1}}{\Delta x} \quad \text{and} \quad \frac{u_{j-1}-u_{j-2}}{\Delta x}$$

Then

$$u_{xx}(j\Delta x) \approx \frac{u_x(j\Delta x) - u_x((j-1)\Delta x)}{\Delta x}$$
$$\approx \frac{\frac{u_j - u_{j-1}}{\Delta x} - \frac{u_{j-1} - u_{j-2}}{\Delta x}}{\Delta x^2}$$
$$\approx \frac{u_j - 2u_{j-1} + u_{j-2}}{\Delta x^2}$$

2. For the one-dimensional Laplace-equation

$$u_{xx} = 0$$

one can take the finite difference approximation obtained from exercise 1a and solve for u_j

$$\frac{u_j^n - 2u_{j-1}^n + u_{j-2}^n}{\Delta x^2} = 0$$

From that we can obtain the iterative scheme

$$u_j^{(n+1)} = 2u_{j-1}^{(n)} - u_{j-2}^{(n)}.$$

3. This iterative scheme is not really useful. The problem with this scheme is that despite the fact that it describes a BVP, it only needs proper boundary conditions on only one side (in our case it is on the right), and on this side, one has also to consider a second boundary condition as well, which may have not been given in advance. This is due to the fact, that the evaluation for point u_j depends solely on points from the left hand side of the pixel. One can easily see that the mass transport that is going on, is only shifting towards one direction. If one would code such a scheme, one immediately has the problem that on the left side there is no boundary condition necessary, and thus may result in disfavourable behaviour, namely the process may become unstable. Also, the derivative scheme approximates better a second derivative approximation for u_{j-1} instead of u_j

---