Institut für Informatik — TUM
Scientific **C**omputing in **C**omputer **S**cience
K. Rahnema, M. Schreiber

WT 2012

Tutorial
(Advanced Programming)
Worksheet 6:

Assignment 1: Heat equation

Maybe some of you are already familiar with the heat equation from the SciComp Lab. In that case, we encourage you to use your Matlab code as a template for this worksheet as well as the upcoming assignments related to this topic! For those of you which are not familiar with the heat equation, we give a short introduction to the heat equation right here:

Heat equation:

We consider the heat-equation

$$\frac{dT(x,y,t)}{dt} = \alpha \Delta T(x,y,t) + E(x,y), \quad (x,y) \in \Omega$$

for isotropic material with the temperature distribution T, an external energy input E on our domain $\Omega = [0;1]^2$ our material is exposed to, and α being the thermal diffusivity. Since we are interested in the stationary solution only, we set $\frac{dT(x,y)}{dt} = 0$. We also consider the thermal diffusivity to be equal to 1. Thus our equation simplifies to

$$\Delta T(x,y) = -E(x,y)$$

.

Getting a determined system:

This PDE is still not very useful to run any computations on it since this would lead to a not well-posed problem: We miss some boundary conditions which we set to 0 (homogenous Dirichlet boundary condition):

$$T(x,y) = 0, \quad (x,y) \in d\Omega$$

To check and validate our solution, we use the temperature distribution $T(x,y) = sin(\pi x)sin(\pi y)$. Computing the partial derivative $\Delta T(x,y)$ leads to the external energy input $E(x,y) = 2\pi^2 sin(\pi x)sin(\pi y)$. This energy input has to be applied to get the artificial temperature distribution in order to test our solver.

Discretization:

When programming C++, the computer does not understand analytical operators, such as the expression $\Delta T(x, y)$. Our numerical way to solve this is the discretization of the PDE with N grid-points in each dimension and to compute a solution using an iterative solver.

We store the temperature values at $N \cdot N$ grid-points for each discrete grid-point (i, j) into a 2D array. Each array entry (i, j) is associated to the spatial coordinate $\left(i \cdot \frac{1}{N+1}, j \cdot \frac{1}{N+1}\right)$.

Next we have to discretize the Laplace operator. This can be done by a so called stencil which numerically approximates the 2nd partial derivatives:

$$\Delta \approx \frac{1}{h^2} \begin{pmatrix} . & 1 & . \\ 1 & -4 & 1 \\ . & 1 & . \end{pmatrix}$$

with the grid-cell size h. This stencil is applied by computing the discrete convolution using this stencil as the convolution kernel.

This allows us to express the discretized heat-equation using linear algebra:

$$\mathbf{A}x = b$$

with the Laplace operator represented by $\mathbf{A}$ as well as b for more sophisticated boundary conditions. The external energy is handled by modification of b. The solution T in which we are interested in is then given by x.

Assignment:

- Make a sketch of the matrix A and the vector b.
- Write a program which initializes b with the external energy and x to zero.

Iterative solver:

Since solving large linear algebra systems by using direct solvers – e. g. Gauss-elimination – would destroy the sparsity pattern of the matrix, we employ an iterative solver to compute an approximate solution:

The Jacobi-Solver[1] is one of such iterative solvers which we use right now to solve our system of equations: The matrix A is formally decomposed into $A = L + D + R$ with L the lower diagonal components, D the diagonal components and R the upper diagonal components. One solver-iteration is executed by $x^{(i+1)} = D^{-1}(b - (L + U)x^{(i)})$ with $x^{(k)}$ storing the approximated solution of x after the k-th iteration.

- Write a function `jacobi_iteration` which executes a single iterative solver step without setting up the matrix A explicitly. The Dirichlet boundary conditions values may not be explicitly stored in $x^{(k)}$! Parameters for this functions should be the two arrays storing the values for x and b as well as the problem size N.

[1] `http://en.wikipedia.org/wiki/Jacobi_method`

Stopping criteria:

We also need a stopping criterion to decide whether our approximation is close enough to the analytical solution. Ideally, we could use some measure of the error and stop once it is below a limit, but since we don't know the error, we will use the residual in its place.

For this assignment, we use the Euclidean norm on the residual $r^{(i)} = A \cdot x^{(i)} - b$:

$$||r^{(i)}||_2 = \frac{1}{N}\sqrt{\sum_k (r_k^{(i)})^2}$$

- Write a function *compute_residual_norm* which computes the residual norm. Parameters for this functions should be the two arrays storing the values for x and b as well as the problem size N.

Putting all together:

- Finally write a program which combines all previously developed functions to approximate the stationary temperature distribution on a metal plate.
- Validate your solution with the analytical solution $T(x, y) = sin(\pi x)sin(\pi y)$.

Assignment 2: VTK-Output

Write a VTK ASCII Output (See [2]) using fstream[3] to store the temperature distribution and visualize it with Paraview.

Homework assignment 1: Gauss-Seidel

Implement the Gauss-Seidel iterative solver[4]

Questions:

Answer the following questions:

- Assignment 1: This question is related to the subsection 'Stopping criteria'. In case that we know the error $e = t^* - t$ with t^* being an approximated solution and t the analytical correct solution, would we still have to use the iterative solver?
- Assignment 1: When implementing the implicit Laplace-stencil without setting up the matrix A to compute one iterative solver step and the residual, what kind of problems regarding processor performance exist? Do you have a solution for this?

[2] `http://www.vtk.org/VTK/img/file-formats.pdf`
[3] `http://www.cplusplus.com/reference/iostream/fstream/`
[4] `http://en.wikipedia.org/wiki/Gauss%E2%80%93Seidel_method`