Computer-based Engineering Mathematics

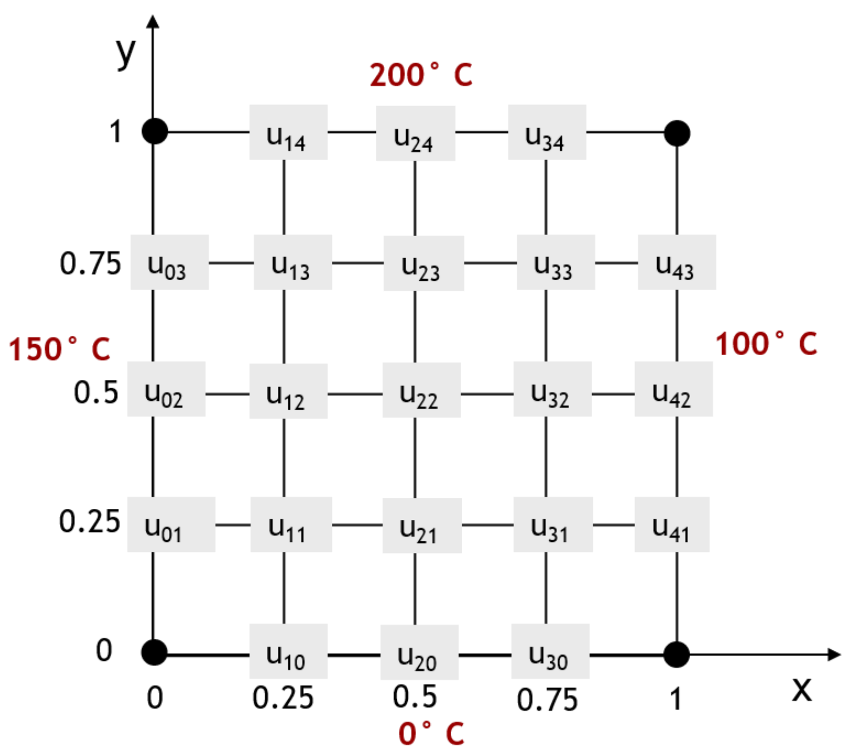
Poisson’s equation – Application to example of hot plate

Dr. Claudia Weis
Faculty of Engineering

Wir machen Ingenieure

Stationary temperature of a quadratic hot plate

We consider the stationary temperature, i.e. the temperature that adjusts after a long time, of a quadratic homogeneous plate whose boundaries are kept at the temperatures



$$u_{i4} = g(x_i, 1) = g_{i,4} = 200°C, i = 1,2,3$$
$$u_{4j} = g(1, y_j) = g_{4,j} = 100°C, j = 1,2,3$$
$$u_{0j} = g(0, y_j) = g_{0,j} = 150°C, j = 1,2,3$$
$$u_{i0} = g(x_i, 0) = g_{i,0} = 0°C, i = 1,2,3$$

The system is described by the equation

$$\Delta u(x,y) = \frac{\partial^2 u}{\partial x^2}(x,y) + \frac{\partial^2 u}{\partial y^2}(x,y) = 0 \tag{1.20}$$

We use the discretization with $N = 3$.

summary of general solution for $N = 3$,

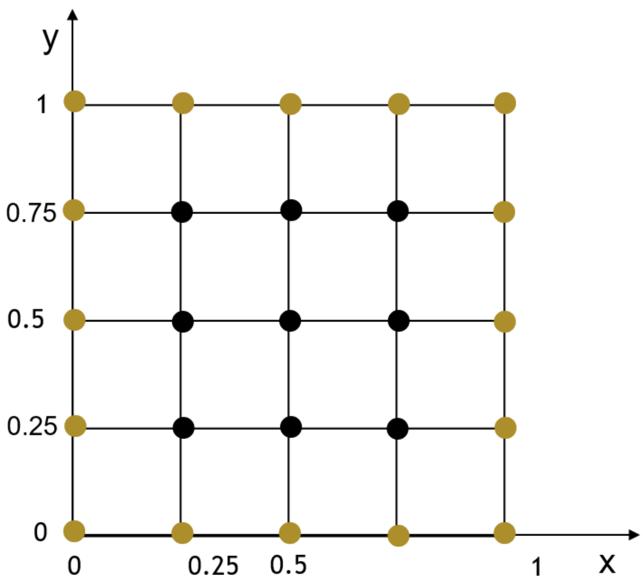
for $N = 3$, the grid width is $h = \frac{1}{3+1} = 0.25$

components of the solution vector: $u = (u_{11}, \dots, u_{13}, u_{21}, \dots, u_{23}, u_{31}, \dots, u_{33}) \in \mathbb{R}^9$

system of equations: $\boldsymbol{Au} = \boldsymbol{b}$

right hand side:

$$\begin{aligned} \boldsymbol{b} &= (b_{11}, \dots, b_{13}, b_{21}, \dots, b_{23}, b_{31}, \dots, b_{33}) \\ &= (h^2 f_{1,1} - g_{0,1} - g_{1,0}, h^2 f_{1,2} - g_{0,2}, h^2 f_{1,3} - g_{0,3} - g_{1,4}, h^2 f_{2,1} - g_{2,0}, h^2 f_{2,2}, h^2 f_{2,3} - g_{2,4}, \\ &\quad h^2 f_{3,1} - g_{4,1} - g_{3,0}, h^2 f_{3,2} - g_{4,2}, h^2 f_{3,3} - g_{4,3} - g_{3,4}) \end{aligned} \tag{1.19}$$



coefficients matrix:

$$\boldsymbol{A} = \begin{pmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{pmatrix}$$

Specific solution for N = 3 and f= 0 with boundary conditions - right hand side

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

boundary conditions

$$u_{i4} = g(x_i, 1) = g_{i,4} = 200°C, i = 1,2,3$$
$$u_{4j} = g(1, y_j) = g_{4,j} = 100°C, j = 1,2,3$$
$$u_{0j} = g(0, y_j) = g_{0,j} = 150°C, j = 1,2,3$$
$$u_{i0} = g(x_i, 0) = g_{i,0} = 0°C, i = 1,2,3$$

right hand side with $f = 0$

$$\boldsymbol{b} = \begin{pmatrix} b_{11} \\ b_{12} \\ b_{13} \\ b_{21} \\ b_{22} \\ b_{23} \\ b_{31} \\ b_{32} \\ b_{33} \end{pmatrix} = \begin{pmatrix} h^2 f_{1,1} - g_{0,1} - g_{1,0} \\ h^2 f_{1,2} - g_{0,2} \\ h^2 f_{1,3} - g_{0,3} - g_{1,4} \\ h^2 f_{2,1} - g_{2,0} \\ h^2 f_{2,2} \\ h^2 f_{2,3} - g_{2,4} \\ h^2 f_{3,1} - g_{4,1} - g_{3,0} \\ h^2 f_{3,2} - g_{4,2} \\ h^2 f_{3,3} - g_{4,3} - g_{3,4} \end{pmatrix} = \begin{pmatrix} -150 \\ -150 \\ -350 \\ 0 \\ 0 \\ -200 \\ -100 \\ -100 \\ -300 \end{pmatrix}$$

IW Wir machen Ingenieure

„staight-forward" implementation (without optimization)

We have to solve this linear system:

$$\begin{pmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{pmatrix} \begin{pmatrix} u_{11} \\ u_{12} \\ u_{13} \\ u_{21} \\ u_{22} \\ u_{23} \\ u_{31} \\ u_{32} \\ u_{33} \end{pmatrix} = \begin{pmatrix} -150 \\ -150 \\ -350 \\ 0 \\ 0 \\ -200 \\ -100 \\ -100 \\ -300 \end{pmatrix} \tag{1.21}$$

- In our first implementation we solve this system of equation using MATLAB, without using the special structure of the coefficients matrix $\boldsymbol{A}$.
- The coefficient matrix contains a lot of zeros. Such a matrix is a sparse matrix.
- MATLAB offers special solution procedures for such type of matrices, which optimize memory usage (they do not write all the zeros to the computer memory).

➢ *We optimize later…*

Implentation 1 (non-optimized)

MATLAB script

```
A = [-4 1 0 1 0 0 0 0 0;
      1 -4 1 0 1 0 0 0 0;
      0 1 -4 0 0 1 0 0 0;
      1 0 0 -4 1 0 1 0 0;
      0 1 0 1 -4 1 0 1 0;
      0 0 1 0 1 -4 0 0 1;
      0 0 0 1 0 0 -4 1 0;
      0 0 0 0 1 0 1 -4 1;
      0 0 0 0 0 1 0 1 -4]

b=[-150;-150;-350;0;0;-200;-100;-100;-300]

u=A\b
```

Result

```
u =


   85.7143
  126.3393
  157.1429
   66.5179
  112.5000
  152.2321
   67.8571
  104.9107
  139.2857
```

Solve the system of equations $Au = b$ with the Backslash-Operator.

improvement: a clever strategy to create the coefficient matrix A

- disadvantages of typing in the matrix $\boldsymbol{A}$ as before
  - takes long, if matrix $\boldsymbol{A}$ is larger (or even impossible if A is really large)
  - error-prone strategy (there are many possibilities to make mistakes)

- improvement:
  - generating A as a function of grid width (as a function of N) automatically
  - ➢ the computer can do it for really huge N

- goals in this lecture:
  - recognize structures that allow for optimization
  - work with huge data sets
  - discuss when optimization is meaningful

improvement: a clever strategy to create the coefficient matrix A

```
%Generate the coefficients matrix A as a function of N
%---------------------------------------------------------------------------
N = 3;

B = -diag(ones(1,N)*4) + diag(ones(1,N-1),1) + diag(ones(1,N-1),-1);

C = zeros(N^2);
for k = 0:N-1
  for i = 1:N
    for j = 1:N
      C(i+k*N, j+k*N) = B(i, j);
    end
  end
end

A = C + diag(ones(1,N^2-N),N) + diag(ones(1,N^2-N),-N)
```

Copy this piece of MATLAB source code to a script file and run the script. Try with different (and large) N.

detailed explanation of improvement (1)

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

```
>> ones(1,N) % Create a row vector with N elements filled with 1

>> ones(1,N)*4 % Multiply the vector with the scalar 4, the result is a row vector with N elements filled with 4

>> diag(ones(1,N)*4) % Create a square matrix from the row vector. The elements on the main diagonal of the matrix are filled with the elements of the row vector.

>> ones(1, N-1) % Create a row vector with N-1 elements filled with 1.

>> diag(ones(1,N-1),1) % Create a matrix from the row vector. The elements on the first diagonal above the main diagonal are filled with the elements of the row vector.

>> diag(ones(1,N-1),-1) % Create a matrix from the row vector. The elements on the first diagonal below the main diagonal are filled with the elements of the row vector.

>> zeros(N^2) % Create a square matrix full of zeros with N^2 by N^2 elements.
```

You can try all the highlighted commands separately and observe the output.

detailed explanation of improvement (2)


```
for k = 0:N-1
  for i = 1:N
    for j = 1:N
      C(i+k*N, j+k*N) = B(i, j); % Fill the elements of C with values from B
    end
  end
end
```

Three `for` loops inside each other to iterate over all
`i = 1:N`
`j = 1:N`
and `k = 0:N-1`
In total, these three loops create all possible combinations of values for i, j, and k.

detailed explanation of improvement (3)

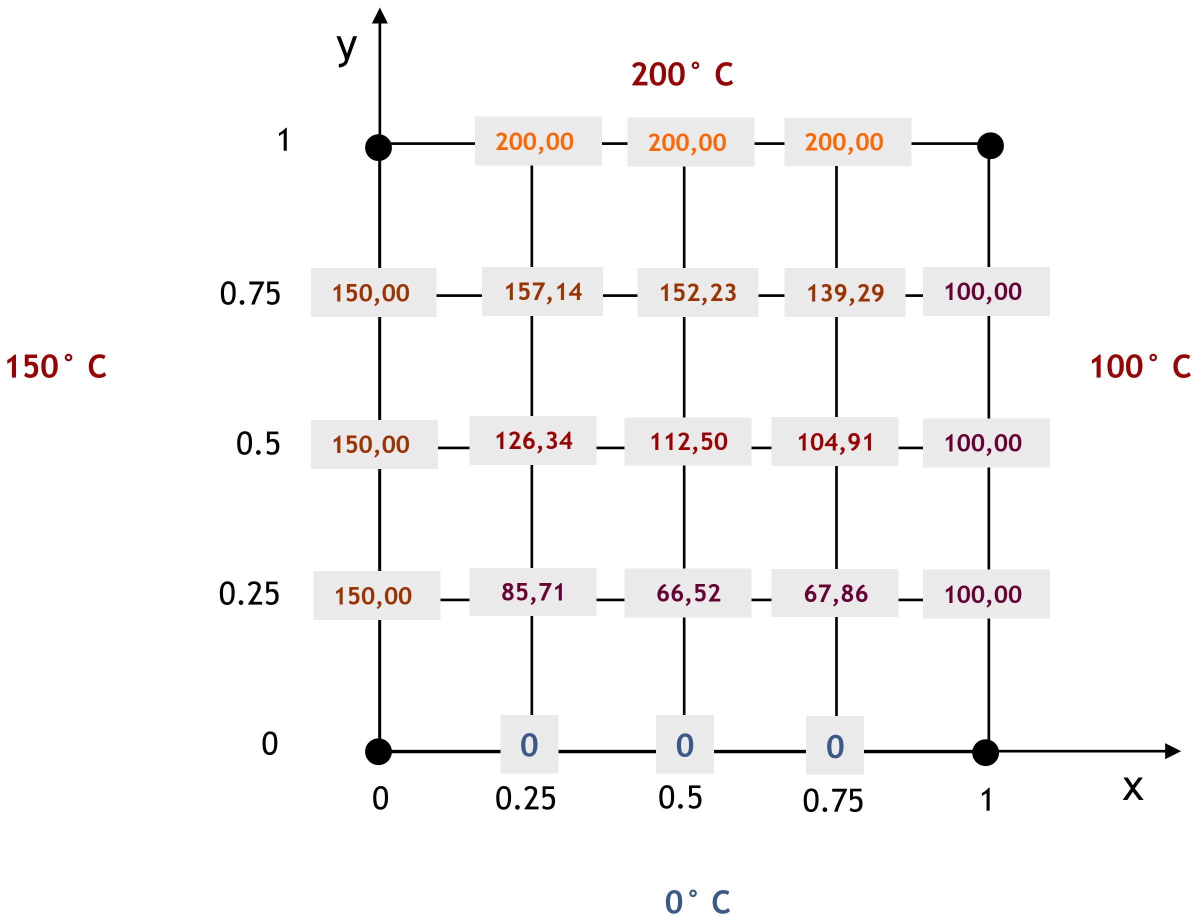
```
B = -diag(ones(1,N)*4) + diag(ones(1,N-1),1) + diag(ones(1,N-1),-1)

A = C + diag(ones(1,N^2-N),N) + diag(ones(1,N^2-N),-N)
```

In both lines of code square matrices of the same size are added (which means element-by-element operations).
The resulting matrices are then assigned to `A` or `B`.

The size of $\boldsymbol{A}$ can easily be adjusted by setting N, e.g. $N = 10$.

Temperature distribution in the grid points for $N = 3$, so $h = 0.25$



The result does not depend on the method we used to create the matrix A, of course.

Interpolation of values between grid points (1)

- The discrete solution vector $\boldsymbol{u}$ gives the temperature at the grid points.
- To approximate the temperature at those points of the plane Ω which do not lie in the discrete set Ω_h, we must interpolate these values.

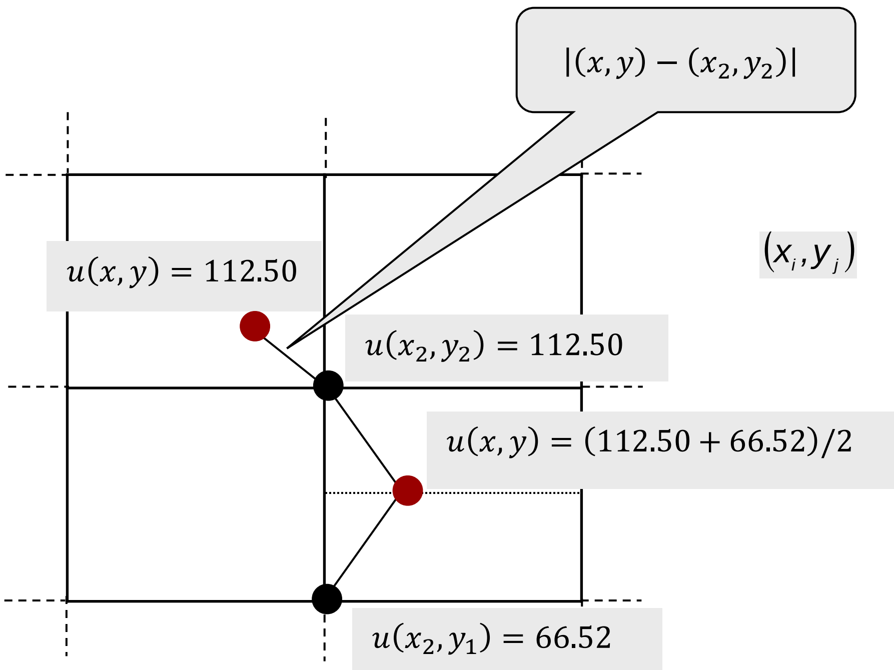


Fig. 1.1 Interpolation 1 of the temperature

Possibility 1:

- The point (x, y) is assigned to the temperature $u(x, y)$ of that grid point which has the smallest distance to the point (x, y).
- If n grid points have the same distance, then the average of temperatures is taken.

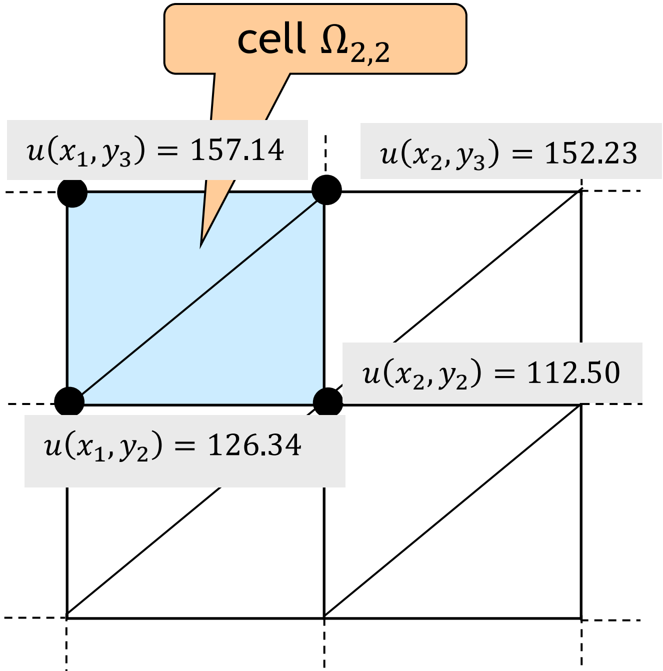


Fig. 1.2 Interpolation 2 of the temperature

Possibility 2:

- Every square cell $\Omega i, j$ of the grid is divided into two triangles.
- To each of these triangles, we assign a plane in $\mathbb{R}^3$ which is determined by three points

$$\left\{\left(x_i, y_j, u\left(x_i, y_j\right)\right), \left(x_i, y_{j+1}, u\left(x_i, y_{j+1}\right)\right), \left(x_{i-1}, y_j, u\left(x_{i-1}, y_j\right)\right)\right\}$$

(upper left triangle)

or

$$\left\{\left(x_{i-1}, y_{j+1}, u\left(x_{i-1}, y_{j+1}\right)\right), \left(x_i, y_{j+1}, u\left(x_i, y_{j+1}\right)\right), \left(x_{i-1}, y_j, u\left(x_{i-1}, y_j\right)\right)\right\}$$

(lower right triangle)

- points in example in Fig 1.2:

$$\{(0.5, 0.5, 112.50), (0.5, 0.75, 152.23), (0.25, 0.5, 126.34)\}$$

or

$$\{(0.25, 0.75, 157.14), (0.5, 0.75, 152.23), (0.25, 0.5, 126.34)\}$$

linear interpolation function

- linear interpolation function for the cell $\Omega_{i,j}, i,j = 1, \dots, N+1$ of the grid:

$$\phi_{ij}(x,y) = \begin{cases} f_{ij}(x,y), x_{i-1} \le x \le x_i \wedge y_j \le y \le x + (j-i+1)h \\ g_{ij}(x,y), x_{i-1} \le x \le x_i \wedge x + (j-i+1)h \le y \le y_{j+1} \end{cases}$$

- where

$$f_{ij}(x,y) = \frac{1}{h}(u_{i-1,j} - u_{ij})x_i + \frac{1}{h}(u_{i,j+1} - u_{i,j})y_j + u_{ij} - \frac{1}{h}(u_{i-1,j} - u_{ij})x - \frac{1}{h}(u_{i,j+1} - u_{i,j})y$$

- and

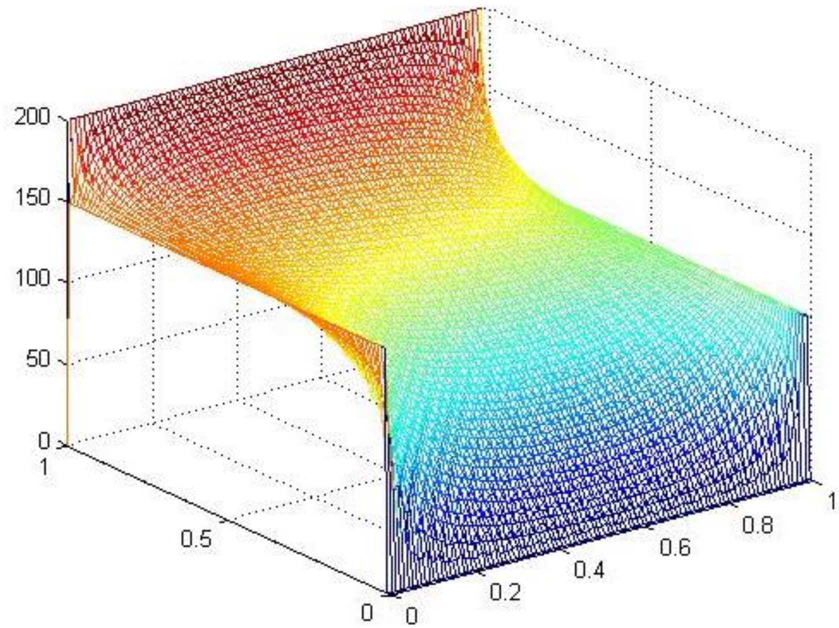
$$g_{ij}(x,y) \\ = -\frac{1}{h}(u_{i,j+1} - u_{i-1,j+1})x_{i-1} + \frac{1}{h}(u_{i-1,j} - u_{i-1,j+1})y_{j+1} + u_{i-1,j+1} + \frac{1}{h}(u_{i,j+1} - u_{i-1,j+1})x \\ - \frac{1}{h}(u_{i-1,j} - u_{i-1,j+1})y$$

- For the cell $\Omega_{2,2}$ in Fig. 1.2, we get:

$$\varphi_{22}(x,y) = \begin{cases} 60.72 - 55.36x + 158.92y, 0.25 \le x \le 0.5 \wedge 0.5 \le y \le x + 0.25 \\ 69.65 - 19.64x - 123.20y, 0.25 \le x \le 0.5 \wedge x + 0.25 \le y \le 0.75 \end{cases}$$

Interpolation converges to the exact solution

- If the exact solution u is sufficiently smooth then the discrete solution u_h, that we called for comfort also u, converges to the exact solution.
- The discrete solution u_h or the on completely $\overline{\Omega}$ defined interpolating function $\tilde{u}_h$ approximate the exact solution u, of course, much better if the grid width $h = 1/(N+1)$ is smaller, i.e. when N is large.
- Reducing the grid with h results in the increase in computation time.



Linear interpolation of the solution including boundary values for
$N = 60$,
Grid width $h = 0.0164$,
x = linspace(0,1,100),
y = linspace(0,1,100)

extra exercise: simple case $f = 0$ and $b = const.$

- With the same algorithm as for the hot plate you can solve the simple case where $f = 0$ and $b = const.$
- Think yourself first…
- … and find the solution on the next slide.

solution for the simple case $f = 0$, $b = const.$ and $N = 3$

```
% Generate coefficient matrix A
N = 3;
B = -diag(ones(1,N)*4) + diag(ones(1,N-1),1) + diag(ones(1,N-1),-1);
C = zeros(N^2);
for k = 0:N-1
  for i = 1:N
    for j = 1:N
      C(i+k*N, j+k*N) = B(i, j);
    end
  end
end
A = C + diag(ones(1,N^2-N),N) + diag(ones(1,N^2-N),-N)

% Set b = const. where k defines the value of that constant.
k = 25; % The value of 25 is just an example. Use any other constant value as a test.
b = (ones(1,N^2)*k)'

% Solve system of linear equations with the backslash operator
u=A\b
```

Questions?

- discussion forum @ Moodle
  (preferred, everyone can reply)

- e-mail to claudia.weis@uni-due.de
  (use your @uni-due.de email address!)