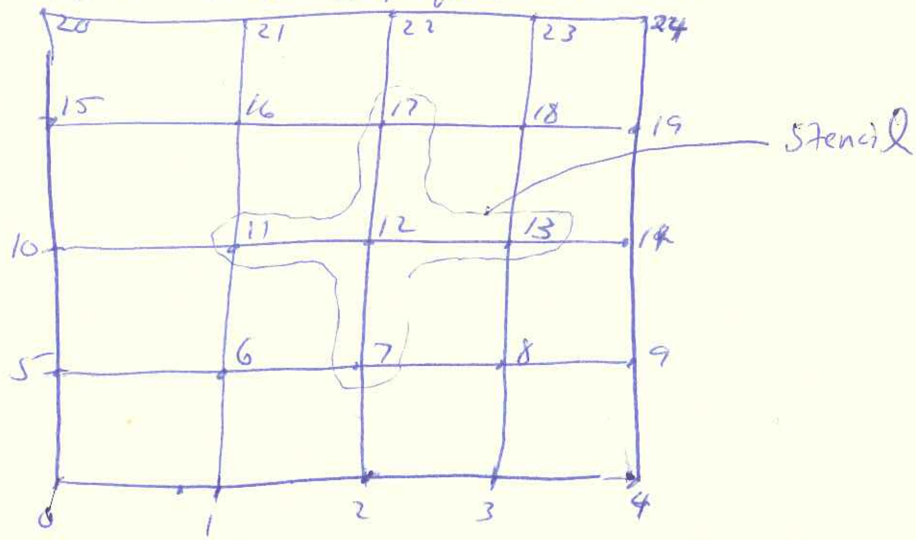
Ok, let's extend our equation to 2D:




$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = -\frac{\dot{q}_g}{k}$$

descritization: $\dfrac{\partial^2 T}{\partial x^2} = \dfrac{T_{i+1,j} - 2T_{ij} + T_{i-1,j}}{\Delta x^2}$

$$\frac{\partial^2 T}{\partial y^2} = \frac{T_{ij+1} - 2T_{ij} + T_{ij-1}}{\Delta y^2}$$

$i$ - x index
$j$ - y index

→ Note, when doing this on computer, I still suggest using a single dimensional array for T

So: $$\frac{T_{i+1,j} - 2T_{ij} + T_{i-1,j}}{\Delta x^2} + \frac{T_{ij+1} - 2T_{ij} + T_{ij-1}}{\Delta y^2} = \frac{-\dot{q}_g}{k}$$

Rearrange:

$$\underbrace{\frac{1}{\Delta y^2}}_{a_S} T_{i,j-1} + \underbrace{\frac{1}{\Delta x^2}}_{a_W} T_{i-1,j} + \underbrace{\left(-\frac{2}{\Delta x^2} - \frac{2}{\Delta y^2}\right)}_{a_P} T_{ij} + \underbrace{\frac{1}{\Delta x^2}}_{a_E} T_{i+1,j} + \underbrace{\frac{1}{\Delta y^2}}_{a_N} T_{i,j+1} = -\frac{\dot{q}_g}{k}$$

Let's setup the coefficient matrix and source vector. for internal pts.

Assume all ~~boundarys~~ boundaries are temperature boundaries: (Dirichlet)

East: $T_{EBC}$ Points 5, 10, 15

West: $T_{WBC}$ Pts 9, 14, 19

South: $T_{SBC}$ Pts 1, 2, 3

North: $T_{NBC}$ Pts 21, 22, 23

~~Corners don't m~~ corners can be either boundary or average

$$
\begin{bmatrix}
a_P & a_E & 0 & a_N & 0 & 0 & 0 & 0 & 0 \\
a_W & a_P & a_E & 0 & a_N & 0 & 0 & 0 & 0 \\
0 & a_W & a_P & 0 & 0 & a_N & 0 & 0 & 0 \\
a_S & 0 & 0 & a_P & a_E & 0 & a_N & 0 & 0 \\
 & & & & & & & & \\
\end{bmatrix}
\begin{bmatrix}
T_6 \\ T_7 \\ T_8 \\ T_{11} \\ T_{12} \\ T_{13} \\ T_{16} \\ T_{17} \\ T_{18}
\end{bmatrix}
=
\begin{bmatrix}
-\frac{\dot{q}_g}{k} - a_W T_{WBC} - a_S T_{SBC} \\
-\dot{q}_g/k - a_S T_{SBC} \\
-\dot{q}_g/k - a_S T_{SBC} - a_E T_{EBC} \\
-\dot{q}_g/k - a_W T_{WBC} \\
 \\
\end{bmatrix}
$$

keep filling in the matrix / b vector

Can solve via LU-Decomposition or Gauss-Seidel or SOR, etc.

If ~~[illegible]~~ uniform mesh $\rightarrow$ matrix is symmetric!

$\Downarrow$

Can be useful for b/c methods ~~used~~ can take advantage of this $\rightarrow$ Cholesky Decomp $A = UL = LL^T$

$\rightarrow$ conjugate gradient methods

What if ~~east~~ west side is Neumann ~~condition~~: condition: ~~$\frac{dT}{dx} = 0$~~

$$\frac{dT}{dx} = 0$$

If we use forward difference:

$$\frac{T_{i+1,j} - T_{i,j}}{\Delta x} = 0$$

$$T_{i+1,j} = T_{i,j}$$

Let's look at Pt. 6!

$$\underbrace{\frac{1}{\Delta y^2}}_{a_S} T_1 + \underbrace{\frac{1}{\Delta x^2}}_{a_W} T_5 + \underbrace{\left(-\frac{2}{\Delta x^2} - \frac{2}{\Delta y^2}\right)}_{a_P} T_6 + \underbrace{\frac{1}{\Delta x^2}}_{a_E} T_7 + \underbrace{\frac{1}{\Delta y^2}}_{a_N} T_{11} = -\frac{\dot{q}_g}{k}$$

$$T_5 = T_6 \qquad T_1 = T_{WBC}$$

So:

$$\begin{bmatrix} (\overbrace{\tfrac{1}{\Delta x^2}}^{a_W} + a_P) & a_E & 0 & a_N & 0 & 0 & 0 & 0 & 0 \\ & & & & & & & & \end{bmatrix} \begin{bmatrix} T_6 \\ T_7 \\ T_8 \\ T_{11} \\ T_{12} \\ \vdots \end{bmatrix} = \begin{bmatrix} -\dot{q}_g/k - a_S T_{SBC} \\ \\ \end{bmatrix}$$

Recall that the forward difference is 1st order and hence we might not have the ~~greatest~~ best accuracy. However, if $\Delta x$ is relatively small, effect shouldn't be too great.

~~Solving for~~

Once the ~~matrix~~ coefficient matrix and the source vector, $\vec{b}$, are setup, using the LU-decomposition algorithm ~~to~~ to solve is easy → it's just a matter of calling those functions.

Note, the $O(N^3)$ includes the factorization

However, LU-Decomp is expensive! It is $O(N^3)$, this means that ~~it~~ the number of calculations performed cubes as ~~the~~ a ~~is~~ function of the # of unknowns! Also notice that there are a lot of zeros! It's a ~~sparse~~ sparse matrix. LU-Decomposition doesn't care if ~~it's sparse~~!

The coefficient matrix is sparse.

Another possible ~~A better~~ ~~better~~ choice ~~for~~ a solution method is Gauss-Seidel or successive over-relaxation (SOR). Of ~~course~~ course, there are better methods than this as well.

For Gauss-Seidel / SOR:

Rearrange the ~~equation~~ difference equation:

$$a_p T_{ij} = -a_E T_{i+1,j} - a_S T_{i,j-1} - a_W T_{i-1,j} - a_N T_{i,j+1} - \frac{\dot{g}_g}{k}$$

$$T_{ij} = \frac{1}{a_p}\left(-a_E T_{i+1,j} - a_S T_{i,j-1} - a_W T_{i-1,j} - a_N T_{i,j+1} - \frac{\dot{g}_g}{k}\right) \text{ — Eq. (1)}$$

Algorithm for Gauss-Seidel:

loop until ~~set Ti~~ converged or until looped over a set # of times (like 100)
loop over all pts.

~~if~~ if East Boundary
set $T_{ij}$ to proper BC Eq
else if West Boundary
set $T_{ij}$ to proper BC
else if North
...
else if South
...

→ continue

continue Gauss-seidel algorithm

else

$T_{ij}$ = equation above (Eq (1))

end if

if $(|T_{ij} - T_{ij\_previous}| > \text{criterion})$

not converged

end if

$T_{ij\_previous} = T_{ij}$

end loop of pts

end outer convergence loop

→ Go to computer to run my program → show input file, show Tecplot file, open Tecplot, show ~~Show inputs to the~~ my function parameters.

General algorithm for HW #2:

Read in input file (heatgen_k, boundary conditions type and value, type of solver, etc)

Write out input data to screen to ensure correct.

Set $a_p, a_w, a_e, a_s, a_n$

Initialize source and Temp arrays (plus any other arrays used)

If LU_Decomp solver

call setCoef function → This sets the coefficient matrix and source vector

~~Call LU-decomp and LU-solve~~

Call LU-decomp functions to solve

Write out Tecplot output file → (NOTE, I'm okay w/ you writting out only the internal pts)

else if Gauss-Seidel

Call Gauss-Seidel function → no need to have full coef. ~~matrix~~ matrix

Write Tecplot file → you'll probably end up including boundary pts here

* Show input file and Tecplot file *