Georg-August-Universität Göttingen
Institut für Theoretische Physik
Prof. Dr. F. Heidrich-Meisner
Dr. E. Bothmann
S. L. Villani
SoSe 2021

Übungen zur Vorlesung
Computergestütztes Wissenschaftliches Rechnen
Blatt 10

Lernziele dieses Übungsblattes

- Lösung von partiellen Differenzialgleichungen mit Hilfe von finiten Differenzen.
- Lösung der Diffusionsgleichung in einer Dimension mit Hilfe des FTCS-Verfahrens.
- Lösung der Laplace-Gleichung in zwei Dimensionen mit Hilfe einer modifizierten Gauß–Seidel-Methode.

Aufgabenmodus

Die Aufgabe 24 dieses Arbeitsblattes verfügt über ein Tutorial, das Sie am Ende des Dokumentes finden. Abhängig von Ihren Vorkenntnissen könne Sie die Aufgaben entweder eigenständig bearbeiten, oder dem dazugehörigen Tutorial folgen. Ziel der Tutorials ist es, Sie durch die grundlegenden Implementationsschritte zu führen. Nach Abschluss eines Tutorials haben Sie ein funktionierendes Programm vorliegen und die entsprechende (Teil-)Aufgabe hinreichend bearbeitet.

Die Aufgabe 26 ist auf englisch.

Übungsaufgaben

Aufgabe 24 *Wärmeleitungsgleichung (mit Tutorial)*

Die Wärmeleitungsgleichung ist der Prototyp parabolischer Differentialgleichungen. Das Temperaturfeld $T(t, x)$ genügt der Bewegungsgleichung

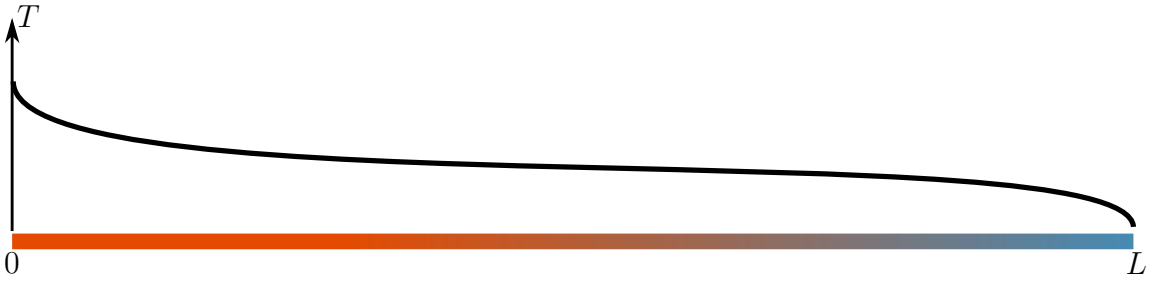
$$\frac{\partial T}{\partial t} = D\frac{\partial^2 T}{\partial x^2},$$

die die zeitliche Ableitung der Temperatur auf der linken Seite mit der zweiten räumlichen Ableitung auf der rechten Seite in Beziehung setzt. Der Parameter D ist die Wärmediffusivität. Man nennt die Gleichung parabolisch, weil sie in den "Potenzen" ähnlich einer Parabelgleichung aussieht:

$$y = x^2\,.$$

Sie sollen die Wärmeleitungsgleichung in einem Stab der Länge L lösen, d.h. bestimmen, wie sich die Temperaturverteilung entlang des Stab mit der Zeit ändert. Dabei werden

verschiedene Randbedingungen durchgespielt. In der folgenden Skizze ist eine solche Temperaturverteilung entlang des Stabs visualisiert, bei der am linken Rand eine hohe, am rechten Rand eine niedrige Temperatur vorliegt.

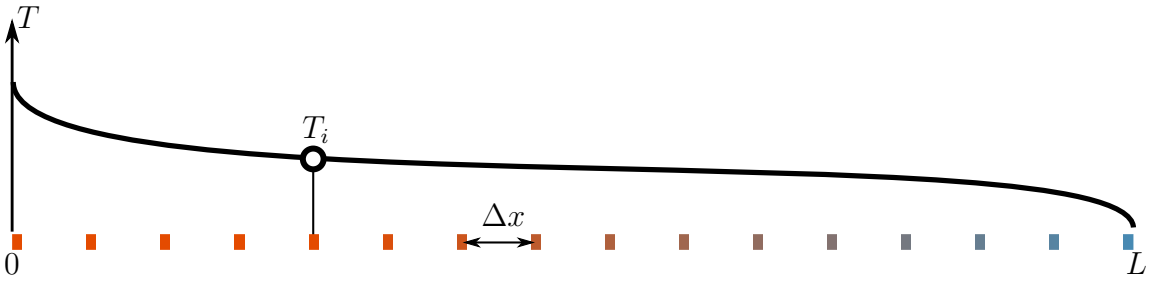


Das Temperaturfeld kann in Ihrem Computerprogramm nicht beliebig genau gespeichert werden. Stattdessen müssen Sie sich für eine definierte Anzahl an *Stützstellen* entscheiden, an denen Sie den Verlauf der Temperatur erfassen.

Es gibt also N Stützstellen mit Abstand Δx und jeweils einer Temperatur T_i. Die PDE an jeder dieser Stützstellen ist

$$\frac{\partial T_i}{\partial t} = D\frac{\partial^2 T_i}{\partial x^2},$$

siehe folgende Skizze zur Illustration der Diskretisierung entlang der x-Richtung:



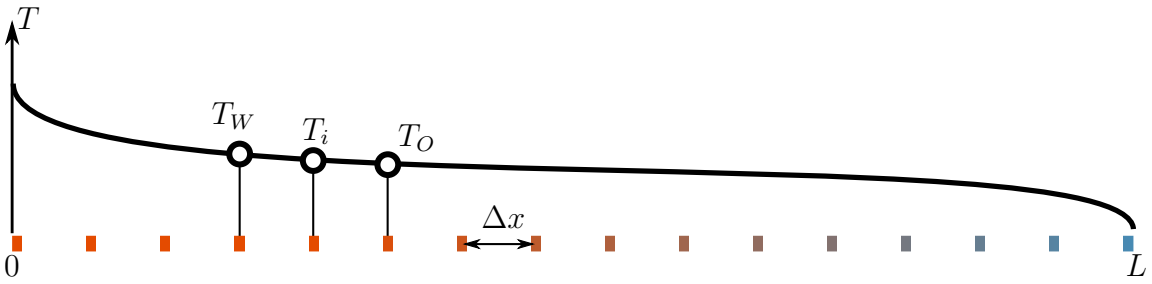
Die Raumableitungen $\frac{\partial^2 T_i}{\partial x^2}$ sind nicht bekannt und müssen numerisch berechnet werden. Auf Blatt 2 am Anfang dieses Semesters haben Sie bereits gesehen, wie dies mit einer zentralen Differenz gelingen kann. Unglücklicherweise ist das hier nur mit starken Einschränkungen möglich: Die Schrittweite der Differenz kann nicht beliebig gewählt werden, weil die Funktion T nur an definierten Stützstellen bekannt ist. Es muss also zwangsläufig als Schrittweite der Stützstellenabstand Δx gewählt werden. Nähert man die Raumableitung durch diese zentrale Differenz, so ergibt sich folgende Bewegungsgleichung:

$$\frac{\partial T_i}{\partial t} = D\frac{T_{i-1} - 2T_i + T_{i+1}}{\Delta x^2}.$$

Dieses Vorgehen zur Bestimmung der Raumableitungen bei einer PDE wird *Finite-Differenzen-Verfahren* genannt.

Häufig werden die benachbarten Gitterpunkte mit “Ost” oder “West” (in 2D Feldern auch “Nord” und “Süd”) abgekürzt. Die drei Punkte T_i, T_W und T_O werden zusammen als 3-Punkt-Stempel (3 point stencil) bezeichnet, die PDE liest sich dann entsprechend (siehe Skizze unten):

$$\frac{\partial T_i}{\partial t} = D\frac{T_W - 2T_i + T_O}{\Delta x^2}.$$



Die Zeitableitung auf der linken Seite der Gleichungen kann durch eine Vorwärtsdifferenz genähert werden. Das geschieht wie beim expliziten Euler-Verfahren für ODEs, indem die rechte Seite der Gleichung zum gegenwärtigen Zeitpunkt ausgewertet wird:

$$\frac{T_i(t+\Delta t) - T_i(t)}{\Delta t} = D\frac{T_W(t) - 2T_i(t) + T_O(t)}{\Delta x^2}.$$

Löst man nun noch nach dem Zustand der Temperatur zum nächsten Zeitpunkt auf, ergibt sich eine Iterationsvorschrift, die zur Integration dieser zeitabhängigen PDEs genutzt werden kann:

$$\boxed{T_i(t+\Delta t) = T_i(t) + \Delta t \cdot D\frac{T_W(t) - 2T_i(t) + T_O(t)}{\Delta x^2}.}$$

Diese Art, die Wärmeleitungsgleichung zu diskretisieren, wird häufig FTCS-Verfahren (Forward Time, Centred Space) genannt.

Eine genauere Betrachtung des FTCS-Schemas für die Wärmeleitungsgleichung ergibt, dass die Wahl des Zeitschritts Δt bei einer gewählten räumlichen Schrittweite Δx eingeschränkt ist, sonst erhält man unsinnige Lösungen und das Verfahren wird instabil. Man findet, dass das Verfahren genau dann stabil ist, wenn folgende Ungleichung erfüllt ist:

$$\Delta t < \frac{\Delta x^2}{2D}\,.$$

Das bedeutet, dass der Zeitschritt immer hinreichend klein gewählt werden muss, anderenfalls werden durch die Diskretisierungsfehler kleine Störungen exponentiell verstärkt.

Ein letztes Detail ergibt sich durch die Ränder. Für den FTCS-Schritt einer Stützstelle sind jeweils der Ost- oder West-Nachbar notwendig. Jedoch ist z.B. am linken Rand kein West-Nachbar verfügbar. Stattdessen werden "virtuelle" Nachbarn (manchmal "Ghosts" genannt) aus Randbedingungen generiert.

Die zwei gängigsten Randbedingungen sind vom Dirichlet- und vom Neumann-Typ. Bei Dirichlet-Randbedingungen wird außerhalb der Simulationsdomäne ein fester Wert für die dynamische Größe festgelegt. Der virtuelle Nachbar wird also einfach auf einen Wert gesetzt; allerdings nimmt er am FTCS-Verfahren teil und der gesetzte Wert kann daher auch explizit zeitabhängig sein, d.h., man kann eine zeitabhängige Funktion für diesen Wert vorgeben:

$$\boxed{\text{Dirichlet Linker Rand:} \qquad T_W = T_{-1} = T_{\text{Rand}}(t)\,.}$$

Neumann-Randbedingungen erfordern, dass die räumliche *Ableitung* der dynamischen Größe einen gewissen Wert annimmt. Damit kann z.B. ein konstanter Wärmefluss am Rand des Systems erzwungen werden.

Beispiel: die Ableitung der Temperatur am rechten Rand soll den Wert w haben,

$$\left.\frac{\partial T_N}{\partial x}\right|_{\text{rechterRand}} = w\,.$$

Mit Hilfe der finiten Differenz wird daraus

$$\frac{T_O - T_W}{2 \cdot \Delta x} = \frac{T_{N+1} - T_{N-1}}{2 \cdot \Delta x} = w,$$

wobei T_O nun ein virtueller Nachbar sein muss, da wir am rechten Rand sind. Aufgelöst nach diesem Ghost T_O ergibt sich

$$T_O = T_W + 2w\Delta x\,.$$

Wenn der Wärmefluss am rechten Rand also vollständig unterdrückt werden soll (Isolation mit $w = 0$), so gilt

Neumann Rechter Rand (kein Fluss durch den Rand): $T_O = T_{N+1} = T_W$.

Hinweis: Die Aufgabe verfügt über ein Tutorial.

1. Schreiben Sie die Funktion

   ```
   void heat_ftcs(double t, double y[], double dt);
   ```

   die für die 1D-Wärmeleitungsgleichung einen einzelnen FTCS-Schritt durchführt. Die Domäne hat eine Länge von $L = 1$ und wird mit $\Delta x = 0.01$ diskretisiert. Es gelte $D = 0.1$.

   Die Funktion nimmt einen Zustandsvektor `y[]` entgegen, der die `N` Temperaturen der Domäne enthält. Berechnen Sie die neuen Werte des Feldes und schreiben Sie sie zurück in `y[]`. Achten Sie darauf, dass Sie die Ergebnisse zunächst in ein Zwischenarray schreiben müssen, damit die rechte Seite der PDE immer zum Zeitpunkt t ausgewertet wird!

2. Initialisieren Sie das Feld auf die konstante Temperatur $T = 0$. Prägen Sie dem System am linken Rand die konstante Temperatur $T_{links} = 1$ und am rechten Rand die konstante Temperatur $T_{rechts} = -1$ auf.

3. Simulieren Sie das System für einige Zeit und stellen Sie die zeitliche Entwicklung des Temperaturfeldes geeignet dar. Verwenden Sie einmal einen Zeitschritt unterhalb des Stabilitätskriteriums und einmal oberhalb des Stabilitätskriteriums. Verändern Sie auch einige Parameter des Systems und überzeugen Sie sich davon, dass das Kriterium tatsächlich funktioniert.

4. Setzen Sie den rechten Rand auf einen Neumann-Rand mit $w = 0$. Erzeugen Sie am linken Rand des Systems mit einem zeitabhängigen Dirichlet-Rand einen Wellenberg:

   $$T_{links}(t) = \exp\left(-(t-5)^2\right)$$

   Stellen Sie ebenfalls den zeitlichen Verlauf des Wellenbergs dar.

5. **Optional 1**: Bestimmen Sie numerisch die Zerfallskonstante der Amplitude und die Ausbreitungsgeschwindigkeit des Wellenbergs.
6. **Optional 2**: Setzen Sie beide Enden der Domäne auf Neumann-Ränder. Pumpen Sie am linken Rand für eine Zeitdauer von $1\,\mathrm{s}$ Wärme in das System und isolieren Sie danach das System. Wie gut wird nach dem Ende der Wärmezufuhr die Gesamtenergie im System erhalten?

Aufgabe 25 *Stabilitätsanalyse*

In der VL wurde die Stabilitätsanalyse für die Wellengleichung vorgestellt. Für eine skalare Funktion $f(x,t)$ lautet diese ($u > 0$):

$$\frac{\partial^2 f(x,t)}{\partial t^2} = u^2 \frac{\partial^2 f(x,t)}{\partial x^2}\,. \tag{1}$$

Dabei wird eine Gleichung für G bestimmt, wobei G wie folgt definiert ist:

$$f(x,t+\Delta t) = e^{i(kx-\omega(t+\Delta t))} = f(x,t)G\,. \tag{2}$$

Schreiben Sie Gl. (1) mittels finiter Differenzen um und leiten Sie daraus die Gleichung

$$G^2 - 2\left[1 - 2\beta^2 \sin^2\left(\frac{k\Delta x}{2}\right)\right] G + 1 = 0\,. \tag{3}$$

her, wobei $\beta \equiv u\Delta t/\Delta x$. Bestimmen Sie daraus das Stabilitätskriterium

$$|G|^2 = 1 \quad \text{für} \quad |\beta| \leq 1\,. \tag{4}$$

Aufgabe 26 *Poisson equation of the electric field in a box*

In this exercise you are asked to solve Maxwell's equations for a time independent electric field $\vec{E}(\vec{x})$ in a grounded two-dimensional box generated by a square electrode inside the box. This is electrostatics, so there is no magnetic field in play. This gives

$$\vec{\nabla} \cdot \vec{E}(\vec{x}) = \frac{\rho(\vec{x})}{\varepsilon_0}, \qquad \vec{\nabla} \times \vec{E}(\vec{x}) = \vec{0}, \tag{5}$$

where $\rho(\vec{x})$ is the charge density and ε_0 is the vacuum permittivity. The best way to solve them is by introducing the electric potential first, i.e. $\vec{E} = -\vec{\nabla}\phi$, and then to solve the resulting equation for ϕ. After that, one can take its derivative to get back the electric field. This approach is preferable to solving the equations directly for $\vec{E}$, because in this way you don't have to deal with a system of partial differential equations for the different components of $\vec{E}$, but instead there is only one for the scalar field ϕ. Using the potential, eqs. (5) becomes the Laplace equation:

$$-\nabla^2\phi(\vec{x}) = -\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)\phi(x,y) = \frac{\rho(\vec{x})}{\varepsilon_0}. \tag{6}$$

In the case considered here the problem can be simplified even further because the charge distribution ρ within the empty volume of the box vanishes. Hence there we have to solve the Poisson equation

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)\phi(x, y) = 0, \tag{7}$$

while the boundary condition are given by the potential applied to the electrode and the grounding of the box walls.

Let's see how this equation can be solved numerically. Using the finite difference method to discretise eq. (7), we get

$$\frac{1}{\Delta x^2}\left(\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}\right) + \frac{1}{\Delta y^2}\left(\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}\right) = 0, \tag{8}$$

where $\phi_{i,j} = \phi(i\Delta x, j\Delta y)$. The error of the thus discretised second order derivative is $\mathcal{O}(\Delta x^2)$ and $\mathcal{O}(\Delta y^2)$.

At this point, we can solve eq. (8) like if it was a system of linear equations. $\phi_{i,j}$ is written as the solution vector $\vec{\phi}$ of dimension $n = n_x \cdot n_y$ of the system $\mathbf{A}\vec{\phi} = \vec{0}$, where $\mathbf{A}$ is a $n \times n$ matrix:

$$\mathbf{A} = \begin{bmatrix} \mathbf{D} & -\mathbf{I} & 0 & \cdots & \cdots & \cdots & 0 \\ -\mathbf{I} & \mathbf{D} & -\mathbf{I} & 0 & \cdots & \cdots & 0 \\ 0 & -\mathbf{I} & \mathbf{D} & -\mathbf{I} & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -\mathbf{I} & \mathbf{D} & -\mathbf{I} & 0 \\ 0 & \cdots & \cdots & 0 & -\mathbf{I} & \mathbf{D} & -\mathbf{I} \\ 0 & \cdots & \cdots & \cdots & 0 & -\mathbf{I} & \mathbf{D} \end{bmatrix}.$$

Here, $\mathbf{I}$ is the identity matrix and

$$\mathbf{D} = \begin{bmatrix} 4 & -1 & 0 & \cdots & \cdots & \cdots & 0 \\ -1 & 4 & -1 & 0 & \cdots & \cdots & 0 \\ 0 & -1 & 4 & -1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -1 & 4 & -1 & 0 \\ 0 & \cdots & \cdots & 0 & -1 & 4 & -1 \\ 0 & \cdots & \cdots & \cdots & 0 & -1 & 4 \end{bmatrix}.$$

To solve the Poisson equation you could now directly solve this system of equations using the methods introduced in the last exercise sheet (sheet 9). But using either Jacobi or Gauss–Seidel as they are is not the most suitable approach for this problem, since the majority of the matrix components are 0, so we'd waste a lot of memory and we'd increase the complexity of the problem making it slower. The best way is to optimise those algorithms for this particular case. Equation (8) can be reorganised as

$$\phi_{i,j} = \frac{1}{2}\frac{1}{\Delta x^2 + \Delta y^2}\left[\Delta y^2\left(\phi_{i+1,j} + \phi_{i-1,j}\right) + \Delta x^2\left(\phi_{i,j+1} + \phi_{i,j-1}\right)\right]. \tag{9}$$

But this already is in the Gauss–Seidel method form if we consider $\phi_{i-1,j}$ and $\phi_{i,j-1}$ as the values that have already been updated, and $\phi_{i+1,j}$ and $\phi_{i,j+1}$ as the ones that have not yet been updated. To fully recover the Gauss-Seidel method, we only need to introduce

an iteration to eq. 9. Luckily for us, the matrix **A** satisfies a convergence requirement for the method. So we know that the succession

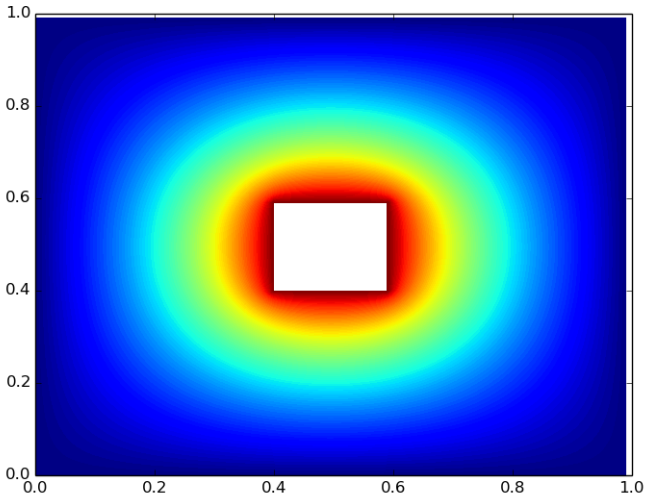
$$\phi_{i,j}^{(k+1)} = \frac{1}{2}\frac{1}{\Delta x^2 + \Delta y^2}\left[\Delta y^2\left(\phi_{i+1,j}^{(k)} + \phi_{i-1,j}^{(k+1)}\right) + \Delta x^2\left(\phi_{i,j+1}^{(k)} + \phi_{i,j-1}^{(k+1)}\right)\right] \tag{10}$$

will converge to the solution.

As you can see from equation 10 we have no double sum over i and j on the r.h.s. as we would have in the naive Gauss–Seidel method. This reduces the complexity of the problem by $\mathcal{O}(n^2)$.

1. Solve the Poisson equation for the electric potential generated by a square electrode (of dimension 0.2×0.2 cm and $\phi = 1\,\mathrm{V}$) located at the centre of a closed two-dimensional box (with dimensions 1×1 cm), which is connected to the ground, i.e. the potential at the walls of the box is $\phi = 0\,\mathrm{V}$. For this, implement this adapted version of Gauss–Seidel. Modify the algorithm in order to take the boundary conditions into account (i.e. by using the respective values as input on the right hand side of eq. (10), but making sure that the iteration over i and j never modifies values at the boundaries, similar to what is done in the tutorial for the previous problem). Set $\Delta x = \Delta y = 0.01$ and iterate over k either 5000 times, or until the residual $\varepsilon_k = ||\phi^k - \phi^{k-1}||$ becomes smaller than 10^{-5}.

2. Plot your result for the electric potential ϕ, you should get a field qualitatively similar to the one in the figure below. Also plot ε_k over k to check the convergence of your implementation.



3. Now that you have found the electric potential you can derive the electric field by taking the derivate of the potential field. Using the finite difference method, the two components of the two-dimensional electric field are

$$E_x(x,y) = \frac{\partial}{\partial x}\phi(x,y) \approx \frac{\phi_{i+1,j} - \phi_{i-1,j}}{2\Delta x} = E_x(i,j),$$
$$E_y(x,y) = \frac{\partial}{\partial y}\phi(x,y) \approx \frac{\phi_{i,j+1} - \phi_{i,j-1}}{2\Delta y} = E_y(i,j).$$

The numerical derivative used here is the average of the backward and forward method. Remember that his approximation of the first order derivative has an accuracy of $\mathcal{O}(\Delta x^2)$.

4. Plot the electric field.

Selbsttest

- Wie kann man sich Dirichlet- und Neumann-Randbedingungen veranschaulichen?
- Was versteht man unter einem Finite-Differenzen-Verfahren? Kann man diese in beliebigen Dimensionen formulieren?
- Was ist das FTCS-Verfahren?
- Wie kann man das Lösen einer stationären (also zeitunabhängigen) PDE auf das Lösen eines linearen Gleichungssystems abbilden?
- Rufen Sie sich in Erinnerung, welche Konvergenzkriterien es im Gauß–Seidel-Verfahren an die Matrix $\mathbf{A}$ gibt.

Tutorials

Tutorial 24: Wärmegleichung

1. Legen Sie in `heat_ftcs` zunächst ein neues Array für den nächsten Zeitschritt an. Für große Felder sollten Sie in Erwägung ziehen, auf dem Heap zu arbeiten.

```
double y_new[N];
```

2. Beginnen Sie mit einer `for`-Schleife über sämtliche Gitterpunkte:

```
for(size_t i = 0; i < N; ++i)
{
    ...
}
```

3. Erstellen Sie in der Schleife drei Hilfsvariablen mit den Werten für T_W, T_O und T_i:

```
double T = y[i % N];
double TW = y[(i - 1) % N];
double TO = y[(i + 1) % N];
```

Die Modulo-Operation stellt sicher, dass die Zugriffe auf das Array innerhalb der Speichergrenzen geschieht. Dadurch werden automatisch periodische Randbedingungen erzeugt.

Vorsicht: Es kann wie im obigen Fall sicherer sein, beim Rechnen mit Indizes vorzeichenlose Datentypen zu verwenden, wie z.B. `unsigned` oder das speziell darauf ausgelegte `size_t`! Wäre `i = 0` ein `int`, so wäre das Ergebnis von

```
(i - 1) % N
```

eine *negative* Zahl, was sicherlich nicht Ihr gesuchter Index ist. Solche Fehler können zu subtilen und schwer findbaren Bugs führen. In diesem Falle würden

die periodischen Ränder nicht mehr funktionieren, sondern *garbage* vom Stack in die Simulation eindringen. Sie können statische Tools wie `cppcheck` verwenden, um solche Fehler zu entdecken.

Vorzeichenlose ganzzahlige Typen haben ein kontrolliertes Überlauf- und Unterlaufverhalten und sind zuverlässig (und vom C-Standard garantiert) kompatibel mit Modulo-Rechnungen. Das bedeutet, dass Sie mit vorzeichenlosen Daten immer das richtige Ergebnis erhalten, wenn am Ende der Rechnung eine Modulo-Operation steht. Das gilt insbesondere auch wenn in den Zwischenrechnungen die Kapazität des Datentyps zeitweise überschritten (oder unterschritten) wurde.

4. Implementieren Sie die Ränder, die nicht periodisch sein sollen. Eine Neumann-Bedingung am rechten Rand kann z.B. wie folgt erstellt werden:

```
if(i == N - 1)
    TO = TW;
```

5. Berechnen Sie nun den neuen Wert des Gitterpunktes gemäß der folgenden Iterationsvorschrift.

```
y_new[i] = y[i] + dt * ...
```

6. Nachdem Sie alle Gitterpunkte berechnet haben, übertragen Sie den Inhalt von `y_new` nach `y`.
7. Denken Sie daran, ggf. allokierten Heap-Speicher wieder freizugeben.
8. Erstellen Sie den Rest der Simulation in der `main`-Funktion. Geben Sie den Zustand des Feldes zeilenweise pro Zeitschritt in eine Datei aus.
9. Sie können die entstehende "2D-Grafik" in Python z.B. mit `imshow` visualisieren:

```
data = np.loadtxt("heat.dat", delimiter=",")
plt.imshow(data, aspect='auto')
plt.show()
```

Wenn Sie die einzelnen Temperaturkurven lieber als Graphen sehen wollen, können Sie mit `.T` ein Array transponieren. Mit folgendem Befehl plotten Sie halbtransparent jede zehnte Zeile Ihrer Ausgabe:

```
plt.plot(data[::10,:].T, alpha = 0.5)
```