KU LEUVEN kulak

Bachelor Informatica
Numerieke Wiskunde
Koen Van Den Abeele / Nithin Govindarajan

Practicum: De Warmtevergelijking

Academiejaar 2021 – 2022

Dieter Demuynck

Inleiding

De warmtevergelijking is een redelijk gekende vergelijking in de fysica. Hij kan worden geschreven als

$$\frac{\partial u(\vec{x},t)}{\partial t} = \alpha \Delta u(\vec{x},t) \tag{1}$$

waarbij $u(\vec{x},t)$ de temperatuurverdeling op een positie $\vec{x} \in \Omega$ met Ω een domein, een tijdsinterval $t \in [0,T]$, de diffusiviteit en Δ de Laplaciaan is. Deze vergelijking wordt gebruikt om de diffusie van warmte in de ruimte en tijd te berekenen aan de hand van begin- en randvoorwaarden.

1 Warmtevergelijking in 1D

1.1 De vergelijking opstellen

De één-dimensionale warmtevergelijking $u(x,t)$ kan gebruikt worden om de warmte te berekenen in simpele objecten, zoals een staaf. Om deze warmtevergelijking op te stellen, wordt de wet van Fourier gebruikt. De wet van Fourier beweert dat de snelheid van de stroming van warmte per oppervlakte-eenheid $\vec{q}$, een vectorveld, door een oppervlakte proportioneel is met min de gradiënt van de warmteverdeling $u(\vec{x},t)$. Dit geeft de vergelijking

$$\vec{q} = -k\nabla u \tag{2}$$

waarbij k de warmtegeleidingscoëfficiënt is. In een één-dimensionaal stelsel wordt de positie voorgesteld door 1 coördinaat x, waardoor $q(x,t)$ een scalair veld wordt en de gradiënt $\nabla u(x,t)$ simpelweg een afgeleide naar x wordt. Dus wordt de vergelijking

$$q = -k\frac{\partial u}{\partial x}$$

We definiëren nu ook de functie $Q(x,t)$ dat de interne warmte energie per eenheid volume van de staaf beschrijft op elk punt x op tijdstip t. Wanneer er geen warmte wordt toegevoegd aan het systeem, is de snelheid van de verandering van de interne warmte energie per volume-eenheid Q proportioneel met de snelheid van de verandering van de temperatuur u. Wanneer van een constante dichtheid en warmte capaciteit wordt uitgegaan, geldt dat

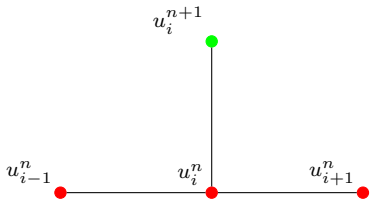
$$\frac{\partial Q}{\partial t} = c\rho\frac{\partial u}{\partial t} \tag{3}$$

waarbij c de specifieke warmte capaciteit en ρ de dichtheid van het materiaal. Wanneer hierop de wet van behoud van energie wordt toegepast op een klein gebied rond elk punt x, concludeert men dat de afgeleide van Q naar t gelijk is aan min de afgeleide van q naar x. In symbolen:

$$\frac{\partial Q}{\partial t} = -\frac{\partial q}{\partial x}$$

waaruit volgt dat

$$\begin{aligned}\frac{\partial u}{\partial t} &= -\frac{1}{c\rho}\frac{\partial q}{\partial x} \\ &= -\frac{1}{c\rho}\frac{\partial}{\partial x}\left(-k\frac{\partial u}{\partial x}\right) \\ &= \frac{k}{c\rho}\frac{\partial^2 u}{\partial x^2}\end{aligned}$$



Figuur 1: Stencil voor het berekenen van een waarde u_i^{n+1} op een volgend tijdstip met index n+1 met behulp van de expliciete methode

Stellen we $\alpha = \frac{k}{c\rho}$ geeft dit de warmtevergelijking in 1 dimensie

$$\frac{\partial u(x,t)}{\partial t} = \alpha \frac{\partial^2 u(x,t)}{\partial x^2} \tag{4}$$

Hierbij noemt α de warmtediffusiviteit.

1.2 Expliciete numerieke simulatie

Om de één-dimensionale warmtevergelijking numeriek op te lossen zullen we gebruik maken van de formules voor voorwaartse differenties, en centrale differenties voor equidistante punten, elk voor de afgeleide in de tijd respectievelijk voor de (tweede) afgeleide in de ruimte. We noemen deze methode expliciet wegens de manier waarop we elke waarde op een volgend tijdstip berekenen. Uiteindelijk bekomen we een formule van de vorm

$$Y(t + \Delta t) = F(Y(t))$$

We gebruiken de notatie $u_i^n = u(x_i, t_n)$ waarbij $x_i = i\Delta x$ voor $i = 0, 1, ..., N_x - 1$ en $t_n = n\Delta t$ waarbij $n = 0, 1, ..., N_t - 1$. Hierbij is N_x en N_t het aantal waarden voor x resp. t. Dan geldt:

$$\partial_t u_i^n = \frac{u_i^{n+1} - u_i^n}{\Delta t} + O(\Delta t) \tag{5}$$

$$\partial_{xx} u_i^n = \frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{(\Delta x)^2} + O((\Delta x)^2) \tag{6}$$

Deze formules zullen benaderend gelijk zijn na het weglaten van de termen binnenin de O-notatie. Die benaderende formules invullen in de warmtevergelijking 4 geeft dan:

$$\begin{aligned} \frac{u_i^{n+1} - u_i^n}{\Delta t} &\approx \alpha \frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{(\Delta x)^2} \\ \Leftrightarrow \qquad u_i^{n+1} &\approx \frac{\alpha \Delta t}{(\Delta x)^2} u_{i-1}^n + \left(1 - \frac{2\alpha \Delta t}{(\Delta x)^2}\right) u_i^n + \frac{\alpha \Delta t}{(\Delta x)^2} u_{i+1}^n \end{aligned}$$

Stel $r = \frac{\alpha \Delta t}{(\Delta x)^2}$, dan is deze vergelijking makkelijker te schrijven als

$$u_i^{n+1} \approx r u_{i-1}^n + (1 - 2r)\, u_i^n + r u_{i+1}^n \tag{7}$$

Dit is een formule om de warmte op een punt x op volgend tijdstip te berekenen aan de hand van de warmte in het punt x, links van x en recht van x op het huidig tijdstip. Figuur 1 geeft een grafische voorstelling van deze bewerking onder de vorm van een stencil. De horizontale as stelt de x-as voor, en de verticale as geeft een verplaatsing in de tijd weer.
Deze recursieve formule kan worden genoteerd als een matrix matrixvermenigvuldiging

$$\begin{bmatrix} & & \\ & T & \\ & & \end{bmatrix} \begin{bmatrix} u_0^n \\ \vdots \\ u_{N_x-1}^n \end{bmatrix} = \begin{bmatrix} u_0^{n+1} \\ \vdots \\ u_{N_x-1}^{n+1} \end{bmatrix} \tag{8}$$

Hierbij is T een matrix waarbij de diagonaalelementen gelijk zijn aan $1-2r$ en alle elementen net boven en onder de diagonaal zijn dan gelijk aan r. Alle andere elementen zijn gelijk aan nul. Dit soort matrices, met elementen verschillend van nul op, boven en onder de diagonaal met alle andere elementen gelijk aan nul, noemen we een *tridiagonaal matrix*.

$$\left[\begin{array}{c} \\ T \\ \\ \end{array}\right] = \begin{bmatrix} 1-2r & r & 0 & \dots & 0 \\ r & 1-2r & r & \dots & 0 \\ 0 & r & 1-2r & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1-2r \end{bmatrix} \tag{9}$$

Merk op dat er een probleem optreed bij de randwaarden, waar $i = 0$ en $i = N_x - 1$. Om bijvoorbeeld u_0^{n+1} te berekenen, zijn er waarden nodig die buiten het toegelaten interval voor x liggen. Echter is dit geen probleem, aangezien deze elementen gekend zijn door de opgelegde randvoorwaarden en hoeven ze dus niet berekend te worden a.d.h.v. de recursieformule.

1.2.1 Implementatie bij constante randvoorwaarden

De implementatie `heat_explicit.m` (zie Appendix A) voor het numeriek berekenen van de warmtevergelijking in 1D werkt als volgt:
De lengte van de staaf, de tijd tot wanneer we de warmteverspreiding willen simuleren en het aantal stappen, samen met de diffusiviteit van de staaf worden allen ingegeven als constanten. De constante randvoorwaarde $u = T_{om}$ wordt gevraagd voor positie $x = L$ voor alle $t \in [0, T]$, en op positie $x = 0$ wordt ervan uitgegaan dat daar de constante randvoorwaarde $u = 0$ geldt voor alle $t \in [0, T]$. De beginvoorwaarden worden niet rechtstreeks meegegeven, maar eerder als een aparte functie `initval(x)` die bij het document zit. Er wordt gevraagd dat deze functie de positie x afbeeldt op de begintemperatuur van de staaf op die positie. De functie `heat_explicit` zal zelf de randvoorwaarden gebruiken om de initiële waarden te overschrijven, zodat de gebruiker niet per se een functie moet opstellen die aan deze randvoorwaarden voldoet. Na de berekeningen voor een nieuw tijdstip worden de waarden op de rand van de staaf overschreven met de opgelegde randvoorwaarden.

1.2.2 Implementatie bij convectie-randvoorwaarde

Stel dat op $x = 0$ de randvoorwaarde een convectie vergelijking is, zoals

$$\frac{\partial u(0,t)}{\partial x} = \frac{H}{K}\left[u(0,t) - T_{omgeving}\right] \tag{10}$$

Bij het toepassen van centrale differenties op de afgeleide in bovenstaande vergelijking geldt:

$$\partial_x u_0^n \approx \frac{u_1^n - u_{-1}^n}{2(\Delta x)^2}$$

waardoor

$$\begin{aligned} \frac{u_1^n - u_{-1}^n}{2(\Delta x)^2} &\approx \frac{H}{K}\left[u(0,t) - T_{omgeving}\right] \\ \Leftrightarrow \qquad u_{-1}^n &\approx \frac{2H(\Delta x)^2}{K} T_{omgeving} - \frac{2H(\Delta x)^2}{K} u_0^n + u_1^n \end{aligned}$$

Stel $q = \frac{2H(\Delta x)^2}{K}$, dan is de vergelijking te schrijven als

$$u_{-1}^n \approx q T_{omgeving} - q u_0^n + u_1^n \tag{11}$$

Als we formule 7 toepassen op u_0^n bekomen we

$$u_0^{n+1} \approx r u_{-1}^n + (1-2r)\, u_0^n + r u_1^n$$

Wanneer we daarin vergelijking 11 toepassen, resulteert dit in

$$\begin{aligned} u_0^{n+1} &\approx r\left(qT_{omgeving} - qu_0^n + u_1^n\right) + (1-2r)\,u_0^n + ru_1^n \\ &\approx 2ru_1^n + (-qr + 1 - 2r)u_0^n + rqT_{omgeving} \\ &\approx 2ru_1^n + (1-(2+q)r)u_0^n + rqT_{omgeving} \end{aligned}$$

Door gebruik te maken van de beginvoorwaarde in $u(0,0)$ kunnen we zo dus recursief de temperatuur op een volgend tijdstip op positie 0 berekenen. Door de eerste twee elementen op de eerste rij van de matrix T aan te passen naar $(1-(2+q)r)$ voor het eerste element en $2r$ voor het tweede, deze matrix te vermenigvuldigen met de temperatuurvector op het huidig tijdstip, en daarna bij het eerste element (wat de temperatuur op positie $x = 0$ op een volgend tijdstip voorstelt) nog $rqT_{omgeving}$ op te tellen, bekomen we onmiddellijk de correcte temperatuur.
De implementatie met de convectieterm zal dan ook verschillen met de implementatie bij constante randvoorwaarden A(zie sectie 1.2.1) enkel en alleen bij de stappen uitgelegd in vorige paragraaf. De code voor deze implementatie is te vinden in appendix B.

1.3 Impliciete numerieke simulatie

Naast de expliciete methode zoals eerder vermeld is er ook een impliciete methode om de waarden op een volgend tijdstip te berekenen. Deze methode geeft ons dan een vergelijking van de vorm

$$G(Y(t+\Delta t), Y(t)) = 0$$

Om deze vergelijk op te stellen gebruiken we centrale differenties voor de (tweede) afgeleide in beide tijd en ruimte in de warmtevergelijking 4. We discretiseren de vergelijking telkens op halve tijdstippen $t = (j + 1/2)\Delta t$, waardoor de centrale differenties voor de tijdsafgeleide steeds hele tijdstippen $t = \Delta t$ gebruikt. Dit zorgt echter voor problemen voor de differenties van de ruimtelijke afgeleide, aangezien we geen waarden hebben voor deze halve tijdstippen. Om deze waarden te benaderen gebruiken we de methode van Lax, die deze benadering berekend door het gemiddelde te nemen van de aangrenzende waarden:

$$u_i^{t+1/2} \approx \frac{u_i^t + u_i^{t+1}}{2} \tag{12}$$

De centrale differenties voor de afgeleiden geeft dan volgende vergelijkingen

$$\partial_t u_i^{n+\frac{1}{2}} = \frac{u_i^{n+1} - u_i^n}{2\Delta t} + O((\Delta t)^2)$$

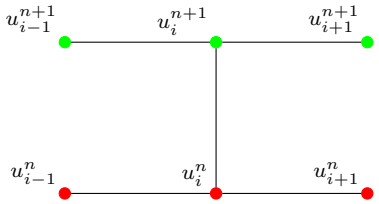
$$\begin{aligned} \partial_{xx} u_i^{n+\frac{1}{2}} &= \frac{u_{i-1}^{n+\frac{1}{2}} - 2u_i^{n+\frac{1}{2}} + u_{i+1}^{n+\frac{1}{2}}}{(\Delta x)^2} + O((\Delta x)^2) \\ &= \frac{u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i+1}^{n+1} + u_{i-1}^n - 2u_i^n + u_{i+1}^n}{2(\Delta x)^2} + O((\Delta x)^2) \end{aligned}$$

Deze formules ingeven in de warmtevergelijking 4 en de hogere orde termen te verwaarlozen resulteert in:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} \approx \alpha \frac{u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i+1}^{n+1} + u_{i-1}^n - 2u_i^n + u_{i+1}^n}{(\Delta x)^2}$$

Stel $r = \frac{\alpha \Delta t}{(\Delta x)^2}$, dan is deze vergelijking gelijk aan

$$-ru_{i-1}^{n+1} + (1+2r)\,u_i^{n+1} - ru_{i+1}^{n+1} \approx ru_{i-1}^n + (1-2r)\,u_i^n + ru_{i+1}^n \tag{13}$$



Figuur 2: Stencil voor het berekenen van een waarde u_i^{n+1} op een volgend tijdstip met index n+1 met behulp van de impliciete methode

Een grafische voorstelling van de waarden die worden gebruikt tijdens elke berekening is te vinden op figuur 2. Opnieuw stelt de horizontale as de x-as voor en de verticale as de tijds-as.
De vergelijking kunnen we ook in matrixvorm schrijven als

$$\begin{bmatrix} & A & \end{bmatrix} \begin{bmatrix} u_0^{n+1} \\ \vdots \\ u_{N_x-1}^{n+1} \end{bmatrix} = \begin{bmatrix} b_0^n \\ \vdots \\ b_{N_x-1}^n \end{bmatrix} \tag{14}$$

waarbij

$$\begin{bmatrix} & A & \end{bmatrix} = \begin{bmatrix} 1+2r & -r & 0 & \dots & 0 \\ -r & 1+2r & -r & \dots & 0 \\ 0 & -r & 1+2r & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1+2r \\ & & & & \end{bmatrix}$$

en

$$b_i^n = (1-2r)u_i^n + r(u_{i-1}^n + u_{i+1}^n)$$

Merk weer op dat dit problemen veroorzaakt op de rand van het ruimtelijk interval, maar opnieuw zal dit niet belangrijk zijn dankzij de opgelegde randvoorwaarden.

1.3.1 Implementatie bij constante randvoorwaarden

Om de waarden u_i^{n+1} uit formule 14 te bereken in de implementatie C moet een stelsel opgelost worden. Dit maakt het moeilijker om de randvoorwaarden toe te passen tijdens de implementatie: de waarden tussen de randen op een volgend tijdstip zijn nu ook afhankelijk van de waarden op de randen op dit volgend tijdstip. In de expliciete methode**??** was dit niet het geval, daar konden we de randwaarden simpelweg overschrijven met de opgelegde randvoorwaarden na de berekeningen.
Het stelsel zal lichtjes moeten aangepast worden om de randvoorwaarden toe te passen tijdens de berekeningen. Aangezien we uitgaan van constante randvoorwaarden $u(0,t) = 0$ en $u(L,t) = T_{omgeving}$, kunnen we dit noteren in het stelsel door de eerste rij en de laatste rij in de matrix A te vervangen door $\begin{pmatrix} 1 & 0 & 0 & \dots & 0 \end{pmatrix}$ respectievelijk $\begin{pmatrix} 0 & \dots & 0 & 0 & 1 \end{pmatrix}$. Daarbij kunnen we dan b_0^n en $b_{N_x-1}^n$ gelijkstellen aan 0 respectievelijk $T_{omgeving}$. Daarna hoeven we enkel een stelsel op te lossen. Dit doen we door de ingebouwde functie `rref(A)` van *MatLab* toe te passen op de uitgebreide matrix, en de laatste kolom te beschouwen.

1.3.2 Implementatie bij convectie-randvoorwaarde

Stel dat we in formule 13 weer $i = 0$ stellen. Dan bekomen we

$$-ru_{-1}^{n+1} + (1+2r)\,u_0^{n+1} - ru_1^{n+1} \approx ru_{-1}^n + (1-2r)\,u_0^n + ru_1^n$$

We kunnen dan formule 11 toepassen om de termen met $i = -1$ om te vormen naar waarden die we wel kennen of kunnen berekenen:

$$-r\left(qT_{omgeving} - qu_0^{n+1} + u_1^{n+1}\right) + (1+2r)\,u_0^{n+1} - ru_1^{n+1} \approx r\left(qT_{omgeving} - qu_0^n + u_1^n\right) + (1-2r)\,u_0^n + ru_1^n$$
$$\Leftrightarrow \qquad (1+r(2+q))\,u_0^{n+1} - 2ru_1^{n+1} \approx (1-r(2+q))\,u_0^n + 2ru_1^n + 2rqT_{omgeving}$$

Waarbij opnieuw $q = \frac{2H(\Delta x)^2}{K}$.
Door de eerste twee termen in de eerste rij in de matrix A in formule 14 te vervangen met de coëfficiënten zoals hierboven beschreven, en de term b_0^n gelijk te stellen aan het rechterlid, kunnen we de convectieterm voor $x = 0$ toepassen in de berekeningen. We gebruiken dezelfde methode beschreven in sectie 1.3.1 voor de randwaarde in $x = L$ en voor het stelsel op te lossen. De implementatie hiervan is te vinden in appendix D

2 Warmtevergelijking in 2D

De warmtevergelijking in 2D ziet er als volgt uit:

$$\frac{\partial u(x,y,t)}{\partial t} = \alpha\left(\frac{\partial^2 u(x,y,t)}{\partial x^2} + \frac{\partial^2 u(x,y,t)}{\partial y^2}\right) \tag{15}$$

We geven geen bewijs van deze vergelijking in dit verslag. We zullen deze vergelijking oplossen voor een eenheidsvierkant $[0,1] \times [0,1]$ met volgende randvoorwaarden:

$$u(x,0,t) = u(x,1,t) = 0 \tag{16}$$
$$u(0,y,t) = sin(\pi y) \tag{17}$$
$$\frac{\partial u(1,y,t)}{\partial x} = \frac{H}{K}\left(u(1,y,t) - T_{omgeving}\right) \tag{18}$$

en volgende beginvoorwaarde:

$$u(x,y,0) = 0 \tag{19}$$

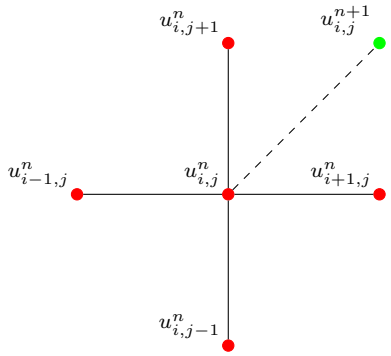
We zullen een recursiemethode opstellen om deze vergelijking met de begin- en randvoorwaarden numeriek te berekenen. We herdefiniëren de notatie voor de waarde $u(x_i, y_j, t_n)$ als $u_{i,j}^n$ waarbij x_i, y_j, t_n op een analoge manier zijn gedefinieerd als bij sectie 1.2. Ook op een analoge manier als bij de formules 5 en 6 vinden we dat

$$\frac{u_{i,j}^{n+1} - u_{i,j}^n}{\Delta t} \approx \alpha\left(\frac{u_{i-1,j}^n - 2u_{i,j}^n + u_{i+1,j}^n}{(\Delta x)^2} + \frac{u_{i,j-1}^n - 2u_{i,j}^n + u_{i,j+1}^n}{(\Delta y)^2}\right)$$
$$\Leftrightarrow \qquad u_{i,j}^{n+1} \approx \frac{\alpha\Delta t}{(\Delta x)^2}\left(u_{i-1,j}^n - 2u_{i,j}^n + u_{i+1,j}^n\right) + \frac{\alpha\Delta t}{(\Delta y)^2}\left(u_{i,j-1}^n - 2u_{i,j}^n + u_{i,j+1}^n\right) + u_{i,j}^n$$

Stel $r_x = \frac{\alpha\Delta t}{(\Delta x)^2}$ en $r_y = \frac{\alpha\Delta t}{(\Delta y)^2}$, dan geldt:

$$u_{i,j}^{n+1} \approx r_x\left(u_{i-1,j}^n + u_{i+1,j}^n\right) + r_y\left(u_{i,j-1}^n + u_{i,j+1}^n\right) + (1 - 2r_x - 2r_y)u_{i,j}^n \tag{20}$$

Figuur 3 geeft een grafische weergave van de waarden die gebruikt worden of berekend worden tijdens de berekening. In dit stencil stelt de horizontale as de x-as voor, de verticale de y-as en de stippellijn stelt een verplaatsing in de tijd voor. Merk wel op dat de stippellijn in de realiteit eigenlijk loodrecht uit het blad zou moeten komen, en dat de groene knoop in werkelijkheid zich recht boven de verbonden rode knoop bevindt.



Figuur 3: Stencil voor het berekenen van een waarde $u^{n+1}_{i,j}$ op een volgend tijdstip met index n+1 voor de twee-dimensionale warmtevergelijking

2.1 Implementatie

Wanneer $n = 0$ of $j = 0$ of $j = N_y$ geldt door de begin- en randvoorwaarden dat $u^n_{i,j} = 0$, en wanneer $i = 0$ geldt dat $u^n_{i,j} = sin(\pi y)$, wat ook voldoet aan de daarnet vermelde voorwaarden. Deze waarden kunnen we expliciet invullen tijdens de berekeningen. Enkel de convectie term 18 brengt nog problemen. Op een analoge manier als voor de berekeningen voor formule 11 kunnen we een formule opstellen voor de termen $u^n_{N_x+1,j}$:

$$u^n_{N_x+1,j} \approx q_x u^n_{N_x,j} + u^n_{N_x-1,j} - q_x T_{omgeving} \tag{21}$$

waarbij opnieuw $q_x = \frac{H(\Delta x)^2}{K}$. Dit resulteert dan uiteindelijk in de expliciete formule

$$u^{n+1}_{N_x,j} \approx r_y u^n_{N_x,j-1} + r_y u^n_{N_x,j+1} + 2r_x u^n_{N_x-1,j} + (1 + r_x(q-2) - 2r_y)\, u^n_{N_x,j} \tag{22}$$

Aan de hand van deze formules kunnen we de temperatuur in de plaat berekenen. De implementatie is te vinden in Appendix E. In plaats van 1 grote matrix T te gebruiken voor de berekeningen door die te vermenigvuldigen met de waarden op een huidig tijdstip, zullen we de waarden 1 voor 1 berekenen via de vermelde formules.

Besluit

Afsluitende tekst.

Appendices

A Bestand: heat_explicit.m

```
function [u, x, t] = heat_explicit(L, Nx, T, Nt, alpha, Tom)
    % OUTPUTS:
    % u = matrix of Nx by Nt with in the n-th column the solution after
    % n-th timestep.
    % x = array of length Nx with coordinates of discrete points in space
    % t = array of length Nt with coordinates of discrete points in time
    %
    % INPUTS:
    % L = length of rod
    % Nx = amount of discrete points in space
    % T = end time of the requested time interval
    % Nt = amount of discrete points in time
```

```
    % alpha = diffusivity constant
    % Tom = a nice guy- I mean, the ambient temperature
    %
    % ASSUMED GLOBAL VARIABLES:
    % initval(x), a function which maps each position x between 0 and L to
    % the respective initial temperature value of the rod at position x.

    % 1. continuous intervals to discrete points for x and t
    delta_x = L/(Nx-1);
    delta_t = T/(Nt-1);
    x = 0:delta_x:L;
    t = 0:delta_t:T;

    % 2. Calculate first column of u
    u = zeros(Nx, Nt);
    u(:, 1) = initval(x);

    % 3. Set edge values to correct values
    u(1, 1) = 0;
    u(Nx, 1) = Tom;

    % 4. Calculate the matrix bigT
    r = alpha .* delta_t / (delta_x .^ 2);
    bigT = (1-2.*r) .* eye(Nx);
    for i = 2:Nx-1
        bigT(i, i-1) = r;
        bigT(i, i+1) = r;
    end
    % element (1,2) and (Nx, Nx-1) have not yet been set to r, however,
    % these do not matter, as we'll overwrite the values calculated by
    % these rows by the border conditions

    % 5. Calculate the rest of matrix u using matrix bigT
    for n = 2:Nt
        u(:, n) = bigT * u(:, n-1);
        % Remember: we set the outer values to 0 and Tom
        u(1, n) = 0;
        u(Nx, n) = Tom;
    end
end
```

B Bestand: convheat_explicit.m

```
function [u, x, t] = convheat_explicit(L, Nx, T, Nt, alpha, Tom, H, K)
    % OUTPUTS:
    % u = matrix of Nx by Nt with in the n-th column the solution after
    % n-th timestep.
    % x = array of length Nx with coordinates of discrete points in space
    % t = array of length Nt with coordinates of discrete points in time
    %
    % INPUTS:
    % L = length of rod
    % Nx = amount of discrete points in space
    % T = end time of the requested time interval
    % Nt = amount of discrete points in time
    % alpha = diffusivity constant
    % Tom = a nice guy- I mean, the ambient temperature
    % H = heat transfer coefficient
    % K = heat conduction coefficient
    %
    % ASSUMED GLOBAL VARIABLES:
    % initval(x), a function which maps each position x between 0 and L to
    % the respective initial temperature value of the rod at position x.

    % 1. continuous intervals to discrete points for x and t
    delta_x = L/(Nx-1); % We can use these to calculate r
    delta_t = T/(Nt-1);
    x = 0:delta_x:L;
    t = 0:delta_t:T;

    % 2. Calculate first column of u
    u = zeros(Nx, Nt);
    u(:, 1) = initval(x);

    % 3. Set edge values to correct values
    u(Nx, 1) = Tom;

    % 4. Calculate the matrix bigT
    r = alpha * delta_t / (delta_x .^ 2);
    bigT = (1-2.*r) .* eye(Nx);
    for i = 2:Nx-1
        bigT(i, i-1) = r;
        bigT(i, i+1) = r;
    end
    % Addapting the matrix for the convective border conditions at x = 0:
    q = 2 .* H .* delta_x.^2 / K;
    bigT(1, 1) = 1 - (2 + q) * r;
    bigT(1, 2) = 2 * r;

    % 5. Calculate the rest of matrix u using matrix bigT
    for n = 2:Nt
        u(:, n) = bigT * u(:, n-1);
        % Remember: the outer values are either given or must be calculated
        % independently!
        u(Nx, n) = Tom;
        % Adding the last requested term to apply for the convection
```

```
        % property at x = 0:
        u(1, n) = u(1, n) + r * q * Tom;
    end
end
```

C Bestand: heat_implicit.m

```
function [u, x, t] = heat_implicit(L, Nx, T, Nt, alpha, Tom)
    % OUTPUTS:
    % u = matrix of Nx by Nt with in the n-th column the solution after
    % n-th timestep.
    % x = array of length Nx with coordinates of discrete points in space
    % t = array of length Nt with coordinates of discrete points in time
    %
    % INPUTS:
    % L = length of rod
    % Nx = amount of discrete points in space
    % T = end time of the requested time interval
    % Nt = amount of discrete points in time
    % alpha = diffusivity constant
    % Tom = a nice guy- I mean, the ambient temperature
    %
    % ASSUMED GLOBAL VARIABLES:
    % initval(x), a function which maps each position x between 0 and L to
    % the respective initial temperature value of the rod at position x.

    % 1. continuous intervals to discrete points for x and t
    delta_x = L/(Nx-1);
    delta_t = T/(Nt-1);
    x = 0:delta_x:L;
    t = 0:delta_t:T;

    % 2. Calculate first column of u
    u = zeros(Nx, Nt);
    u(:, 1) = initval(x);

    % 3. Set edge values to correct values
    u(1, 1) = 0;
    u(Nx, 1) = Tom;

    % 4. Calculate the matrix bigA
    r = alpha .* delta_t / (delta_x .^ 2);
    bigA = (1+2.*r) .* eye(Nx);
    for i = 2:Nx-1
        bigA(i, i-1) = -r;
        bigA(i, i+1) = -r;
    end
    % Set first and last row to those of the unit matrix:
    bigA(1, 1) = 1;
    bigA(Nx, Nx) = 1;

    % 5. Calculate the matrix bigB to easily create the vector b
    bigB = (1-2.*r) .* eye(Nx);
    for i = 2:Nx-1
        bigB(i, i-1) = r;
        bigB(i, i+1) = r;
    end
    % To simplify calculation. The values calculated by these rows will be
    % overwritten.
    bigB(1, 1) = 0;
```

```
    bigB(Nx, Nx) = 0;

    % 6. Solve the equation using the almost-tridiagonal matrix bigA:
    for n = 2:Nt
        % 6.1: Calculate vector b
        b = bigB * u(:, n-1);
        b(1) = 0;  % condition at x = 0
        b(Nx) = Tom;  % condition at x = L

        % 6.2 Create extended matrix and solve
        A = [bigA, b];
        A = rref(A);

        % 6.3 Perform a check to see if there is in fact a proper solution
        for i = 1:Nx
            if A(i, i) ~= 1
                error("singular matrix at n %d position (%d, %d)", n, i, i);
            end
        end

        % 6.4 Add the last column to the resulting matrix u
        u(:, n) = A(:, Nx + 1);
    end
end
```

D Bestand: convheat_implicit.m

```
function [u, x, t] = convheat_implicit(L, Nx, T, Nt, alpha, Tom, H, K)
    % OUTPUTS:
    % u = matrix of Nx by Nt with in the n-th column the solution after
    % n-th timestep.
    % x = array of length Nx with coordinates of discrete points in space
    % t = array of length Nt with coordinates of discrete points in time
    %
    % INPUTS:
    % L = length of rod
    % Nx = amount of discrete points in space
    % T = end time of the requested time interval
    % Nt = amount of discrete points in time
    % alpha = diffusivity constant
    % Tom = a nice guy- I mean, the ambient temperature
    % H = heat transfer coefficient
    % K = heat conduction coefficient
    %
    % ASSUMED GLOBAL VARIABLES:
    % initval(x), a function which maps each position x between 0 and L to
    % the respective initial temperature value of the rod at position x.

    % 1. continuous intervals to discrete points for x and t
    delta_x = L/(Nx-1);
    delta_t = T/(Nt-1);
    x = 0:delta_x:L;
    t = 0:delta_t:T;

    % 2. Calculate first column of u
    u = zeros(Nx, Nt);
    u(:, 1) = initval(x);

    % 3. Set edge values to correct values
    u(Nx, 1) = Tom;

    % 4. Calculate the matrix bigA
    r = alpha .* delta_t / (delta_x .^ 2);
    bigA = (1+2.*r) .* eye(Nx);
    for i = 2:Nx-1
        bigA(i, i-1) = -r;
        bigA(i, i+1) = -r;
    end
    % Set the last row to those of the unit matrix:
    bigA(Nx, Nx) = 1;
    % Set the first row to the requested calculation for the convection
    % term
    q = 2 .* H .* delta_x.^2 / K;
    bigA(1, 1:2) = [1 + r .* (2 + q), - 2 .* r];

    % 5. Calculate the matrix bigB to easily create the vector b
    bigB = (1-2.*r) .* eye(Nx);
    for i = 2:Nx-1
        bigB(i, i-1) = r;
        bigB(i, i+1) = r;
```

```
    end
    % To simplify calculation.
    bigB(Nx, Nx) = 0;
    % Including calculation for the convection term
    % Note: no constant term has been added yet.
    bigB(1, 1:2) = [1 - r .* (2 + q), 2 .* r];

    % 6. Solve the equation using the almost-tridiagonal matrix bigA:
    for n = 2:Nt
        % 6.1: Calculate vector b
        b = bigB * u(:, n-1);
        b(1) = b(1) + 2 .* r .* q .* Tom;  % constant term for condition at x = 0
        b(Nx) = Tom;  % condition at x = L

        % 6.2 Create extended matrix and solve
        A = [bigA, b];
        A = rref(A);

        % 6.3 Perform a check to see if there is in fact a proper solution
        for i = 1:Nx
            if A(i, i) ~= 1
                error("singular matrix at n %d position (%d, %d)", n, i, i);
            end
        end

        % 6.4 Add the last column to the resulting matrix u
        u(:, n) = A(:, Nx + 1);
    end
end
```

E Bestand: heat2d.m

```
function [u, x, y] = heat2d(Nx, Ny, T, Nt, alpha, Tom, H, K)
    % OUTPUTS:
    % u = matrix of Nx by Ny with the temperature values at t = T
    % x = array of length Nx with coordinates of discrete spacial points
    % y = array of length Nt with coordinates of discrete spacial points
    %
    % INPUTS:
    % Nx = amount of discrete points in space on the x-axis
    % Ny = amount of discrete points in space on the y-axis
    % T = end time of the requested time interval
    % Nt = amount of discrete points in time
    % alpha = diffusivity constant
    % Tom = a nice guy- I mean, the ambient temperature
    % H = heat transfer coefficient
    % K = heat conduction coefficient
    %
    % ASSUMED GLOBAL VARIABLES:
    % NONE: for the initial values we assume all temperatures to be 0

    % 1. continuous intervals to discrete points for x and t
    delta_x = L/(Nx-1); % We can use these to calculate r
    delta_t = T/(Nt-1);
    x = 0:delta_x:L;
```

```
    t = 0:delta_t:T;

    % 2. Calculate first column of u
    u = zeros(Nx, Nt);
    u(:, 1) = initval(x);

    % 3. Set edge values to correct values
    u(Nx, 1) = Tom;

    % 4. Calculate the matrix bigT
    r = alpha * delta_t / (delta_x .^ 2);
    bigT = (1-2.*r) .* eye(Nx);
    for i = 2:Nx-1
        bigT(i, i-1) = r;
        bigT(i, i+1) = r;
    end
    % Addapting the matrix for the convective border conditions at x = 0:
    q = 2 .* H .* delta_x.^2 / K;
    bigT(1, 1) = 1 - (2 + q) * r;
    bigT(1, 2) = 2 * r;

    % 5. Calculate the rest of matrix u using matrix bigT
    for n = 2:Nt
        u(:, n) = bigT * u(:, n-1);
        % Remember: the outer values are either given or must be calculated
        % independently!
        u(Nx, n) = Tom;
        % Adding the last requested term to apply for the convection
        % property at x = 0:
        u(1, n) = u(1, n) + r * q * Tom;
    end


end
```