Technische Universiteit Delft
Faculteit Elektrotechniek, Wiskunde en Informatica
Delft Institute of Applied Mathematics

De Discontinue Galerkin(DG)methode Toegepast Op Chemotaxis

(Discontinuous Galerkin Method Applied On Chemotaxis)

Verslag ten behoeve van het
Delft Institute of Applied Mathematics
als onderdeel ter verkrijging

van de graad van

BACHELOR OF SCIENCE
in
TECHNISCHE WISKUNDE

door

LEON VAN HOUWELINGEN

Delft, Nederland
December 2018

Samenvatting

Chemotaxis is een verschijnsel uit de biologie dat zich op verschillende wijze voordoet in het menselijk lichaam en bij bacteriën. Er is onderzocht hoe het ééndimensionaal geval op te lossen is met de discontinue Galerkinmethode. Hierin is het model eerst behoorlijk versimpeld en steeds verder uitgebreid. Zo is gekeken naar de discontinue Galerkinmethode en de toepassing op de transportvergelijking en kleine aanpassingen hierop én op de toepassing op de warmte vergelijking.

Inhoudsopgave

Samenvatting 1
Lijst met gebruikte symbolen 3
Verklarende woordenlijst 3
1. Inleiding 4
2. Wiskundig Model 5
3. Numerieke Aanpak 7
3.1. Discontinue Galerkinmethode 7
3.2. De transportvergelijking 8
3.3. De gemodificeerde transportvergelijking 12
3.4. De warmtevergelijking 15
4. Simulaties 17
4.1. De transportvergelijking 17
4.2. De gemodificeerde transportvergelijking 20
4.3. De warmtevergelijking 25
5. Conclusies 27
Literatuurlijst 28
6. Bijlagen 29
Bijlage 1: Pythoncode behorend bij de transportvergelijking en de gemodificeerde transportvergelijking: 29

Lijst met gebruikte symbolen

- k: De orde van de Legendre-polynomen,
- φ_k: Het k^e-orde Legendre-polynoom,
- Ω_j: Element j uit de discretisatie,
- Δx: De grootte van een element,
- Δt: De tijdstap.

Verklarende woordenlijst

- DG-methode: Discontinue Galerkinmethode,
- Discretisatie: Het continue gebied omzetten in een discreet gebied,
- Elementen: Deelintervallen uit de discretisatie,
- Legendre-polynomen: Polynomen van de vorm
$$\varphi_n(x) = \frac{1}{2^n n!} \frac{\mathrm{d}^n}{\mathrm{d}x^n}[(x^2-1)^n],$$
- PDV: Partiële differentiaalvergelijking,
- Steady-stateoplossing: De stabiele oplossing waar een differentiaalvergelijking naar convergeert.

1. Inleiding

In dit verslag wordt een toepassing van de numerieke discontinue Galerkin-methode, kortweg DG-methode, bekeken. Numerieke methoden worden gebruikt om oplossingen van differentiaalvergelijkingen te benaderen. Voor gewone differentiaalvergelijkingen zijn er de 'bekende' methoden als Euler voorwaarts, Euler achterwaarts, de trapezoidal methode en de Runge-Kutta methode (Vuik, C et al. 2016). Voor partiële differentiaalvergelijkingen is dat een lastiger verhaal en moet gekeken worden naar discretisatiemethoden zoals eindige-elementenmethode of eindige-volumemethode (van Kan, J et al., 2014). In dit laatste rijtje past de DG-methode, de DG-methode is namelijk een discretisatiemethode. De DG-methode lijkt enigszins op een tussenvorm van de eindige-elementenmethode en de eindige-volumemethode, maar er zijn ook verschillen. Dit zal kort worden toegelicht in het hoofdstuk Numerieke Methoden.

In dit onderzoek zal er specifiek gekeken worden naar het toepassen van de DG-methode op chemotaxis. Per definitie is de chemotaxis "de richtingsbeweging van een organisme of een levende beweeglijke cel in reactie op bepaalde diffusieerbare chemicaliën in het milieu," aldus Biology Online (z.d.). Oftewel, cellen verplaatsen zich met een bepaalde snelheid in een bepaalde richting afhankelijk van de gradiënt van concentraties in de omgeving. Voorbeelden hiervan zijn spermacellen die aangetrokken worden door chemische substanties die van de buitenste laag van de eicel komen of de beweging van fibroblasten - dat zijn de belangrijkste cellen van bindweefsel - in beschadigde delen van het lichaam (Roberts et al., 2012) (britannica.com, z.d.). Andere situaties waar chemotaxis voorkomt zijn tumorgroei en genezing van wonden. Wondgenezing gaat in een aantal stappen. Allereerst gaat een wond bloeden, dat bloed stolt na verloop van tijd, zodat een soort beschermlaag ontstaat. Vervolgens worden door witte bloedcellen de afvalstoffen afgevoerd. Het verplaatsen van deze wittebloedcellen kan ook beschreven worden met behulp van chemotaxis.

Allereerst zal in worden gegaan op het wiskundig model behorend bij de chemotaxis. Vervolgens zal dit wat vereenvoudigd worden. Vanuit dit wiskundig model zal de numerieke aanpak worden beschreven. Wat op zijn beurt weer simulaties genereert, waaruit conclusies getrokken kunnen worden over de chemotaxis.

2. Wiskundig Model

"Het klassieke Keller-Segelmodel (KS-model) wordt weergegeven door een tweetal vergelijkingen. Daarbij representeert (1) de 'cell density variation over time' en (2) de 'chemical attractant concentration variation over time'", aldus Roberts et al. (2012).

$$\begin{aligned} \frac{\partial u}{\partial t} &= \nabla \cdot (D_1 \nabla u - \chi u \nabla v) + f, \quad (1) \\ \frac{\partial v}{\partial t} &= D_2 \Delta v + g - h. \quad (2) \end{aligned}$$

Aangezien het eendimensionale geval beschouwd wordt, kan dit herschreven worden als

$$\begin{aligned} \frac{\partial u}{\partial t} &= D_1 \frac{\partial^2 u}{\partial x^2} - \frac{\partial}{\partial x}(\chi u \frac{\partial v}{\partial x}) + f, \\ \frac{\partial v}{\partial t} &= D_2 \frac{\partial^2 v}{\partial x^2} + g - h. \end{aligned}$$

Het model dat hier bekeken gaat worden is kleine modificatie, en daarmee een kleine versimpeling, hiervan. Er zal namelijk gekeken worden naar het model waarbij $f = 0, g = h, D_2 = \epsilon > 0, D_1 = 0$ en $\chi = \zeta$, met Neumann-randvoorwaarden. We beschouwen dus het volgende stelsel vergelijkingen dat de chemotaxis beschrijft:

$$\begin{cases} 0 = \frac{\partial u}{\partial t} + \frac{\partial}{\partial x}(\zeta \frac{\partial c}{\partial x} u) & x \in (0,1), t > 0, \\ 0 = \frac{\partial c}{\partial t} - \epsilon \frac{\partial^2 c}{\partial x^2} & x \in (0,1), t > 0, \\ u(x,t) = u_0(x) & x \in (0,1), \\ 0 = \frac{\partial c}{\partial x}(t,0) = \frac{\partial c}{\partial x}(t,1) & t > 0. \end{cases} \quad (3)$$

Echter, dit stelsel zal niet in één keer opgelost worden. Dit gaat versimpeld worden tot verschillende vergelijkingen. Eerst zal een oplossing gezocht worden voor de vergelijking waarbij $\frac{\partial c}{\partial x} = 1$ over heel het interval $(0,1)$ én met $u_0(x) = \cos(x)$. Dan zal dus het volgende stelsel - dit wordt ook wel de transportvergelijking genoemd - opgelost worden:

$$\begin{cases} 0 = \frac{\partial u}{\partial t} + \zeta \frac{\partial u}{\partial x} & x \in (0,1), t > 0, \\ u(x,t) = \cos(x) & x \in (0,1). \end{cases} \quad (4)$$

Deze vergelijking zullen we eerst bekijken met randvoorwaarde $u(x+k,t) = u(x,t), k \in \mathbb{Z}$ en later met randvoorwaarde $u(0,t) = 1$.

Daarna zal gekeken worden naar onderstaande PDV (deze zal in dit verslag de gemodificeerde transportvergelijking genoemd worden):

$$\begin{cases} 0 = \frac{\partial u}{\partial t} + \zeta \frac{\partial}{\partial x}(c'u) & x \in (0,1), t > 0, \\ u(x,t) = \cos(x) & x \in (0,1), \end{cases} \quad (5)$$

met $c(x) = e^{\frac{-x^2}{\epsilon^2}}$.

Vervolgens zal worden gekeken naar de tweede vergelijking van (3), ook wel de warmtevergelijking genoemd:

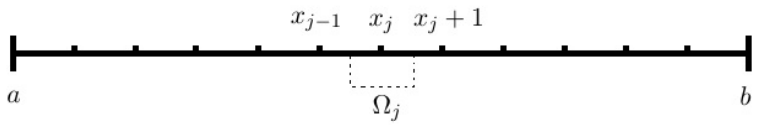
$$\begin{cases} 0 = \frac{\partial c}{\partial t} - \epsilon \frac{\partial^2 c}{\partial x^2} & x \in (0,1), t > 0, \\ 0 = \frac{\partial c}{\partial x}(t,0) = \frac{\partial c}{\partial x}(t,1) & t > 0, \\ c(x,0) = sin(\pi x) & x \in (0,1). \end{cases} \tag{6}$$

Tot slot zal dit alles gecombineerd moeten worden tot een oplossing voor het stelsel partiële differentiaalvergelijkingen (3).

3. Numerieke Aanpak

3.1. **Discontinue Galerkinmethode.** De discontinue Galerkinmethode, kortweg DG-methode is een discretisatiemethode, zoals ook de eindige-elementenmethode dit is. Dit is niet de enige overeenkomst. Zo introduceren ze beide een testfunctie en basisfuncties. Er zijn echter niet alleen overeenkomsten. Zoals de naam al doet vermoeden kan de DG-methode discontinuïteiten in een oplossing verwerken (L.Y.D. Crapts, 2012).

Zoals gezegd wordt het gebied opgedeeld in elementen. Een interval $[a, b]$ wordt opgedeeld in n deelintervallen $\overline{\Omega}_j = [\frac{x_{j-1}+x_j}{2}, \frac{x_j+x_{j+1}}{2}]$, zie figuur 1. Voor het gemak zal $\overline{\Omega}_j$ vanaf hier geschreven worden als Ω_j.



Figuur 1. Het interval $[a, b]$ opgedeeld in kleinere deelintervallen.

Verder zullen als testfuncties de Legendre-polynomen gebruikt worden. Deze Legendre-polynomen hebben de vorm: $\varphi_n(x) = \frac{1}{2^n n!} \frac{\mathrm{d}^n}{\mathrm{d}x^n}[(x^2 - 1)^n]$. Hierbij is de graad van φ_n gelijk aan n.

3.2. **De transportvergelijking.** Allereerst zal geprobeerd worden probleem (4), met randvoorwaarde $u(x+k,t) = u(x,t), k \in \mathbb{Z}$ op te lossen. Zonder verlies van algemeenheid kan dit bekeken worden met $\zeta = 1$. Dit betreft dan de volgende PDV:

$$\text{PDV} : \begin{cases} \frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} = 0 & , x \in (0,1), t > 0, \\ u(x+k,t) = u(x,t) & , t > 0, \\ u(x,0) = \cos(2\pi x) & , x \in (0,1). \end{cases} \tag{7}$$

Deze PDV zal bekeken worden met behulp van de zwakke formulering van de discontinue Galerkinmethode. Er is bekend dat $u(x,t)$ te benaderen is met polynomen, dus ook met zogeheten Legendre-polynomen, $\varphi_n(x)$. Daartoe wordt geschreven $u(x,t) = u^N(x,t) = \sum_{i=1}^{N} c_i(t)\varphi_i(x)$. Dit invullen in (7) geeft:

$$\begin{aligned} & \frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} = 0 \\ \Leftrightarrow \quad & \sum_{i=1}^{N} c_i'(t)\varphi_i(x) + \sum_{i=1}^{N} c_i(t)\varphi_i'(x) = 0. \end{aligned}$$

Vervolgens vermenigvuldigen met de testfunctie $\varphi_m'(x)$ en integreren over het controle-interval Ω_j, geeft het volgende:

$$\begin{aligned} & \sum_{i=1}^{N} c_i'\varphi_i + \sum_{i=1}^{N} c_i\varphi_i' = 0 \\ \Leftrightarrow \quad & \int_{\Omega_j} \left(\sum_{i=1}^{N} c_i'\varphi_i + \sum_{i=1}^{N} c_i\varphi_i' \right) \varphi_m \, \mathrm{d}\Omega = \int_{\Omega_j} 0 \, \mathrm{d}\Omega. \end{aligned}$$

De eerste integraal splitsen én de tweede integraal uitrekenen geeft

$$\int_{\Omega_j} \sum_{i=1}^{N} c_i'\varphi_i\varphi_m \, \mathrm{d}\Omega + \int_{\Omega_j} \sum_{i=1}^{N} c_i\varphi_i'\varphi_m \, \mathrm{d}\Omega = 0. \tag{8}$$

Daarna kan de sommatie en de integratie omgedraaid worden, zo wordt het volgende verkregen:

$$\sum_{i=1}^{N} c_i' \int_{\Omega_j} \varphi_i\varphi_m \, \mathrm{d}x + \sum_{i=1}^{N} c_i \int_{\Omega_j} \varphi_i'\varphi_m \, \mathrm{d}x = 0.$$

Daarna wordt een coördinatentransformatie van Ω_j naar $[-1,1]$ toegepast. De coördinatentransformatie zorgt voor een extra constante - de Jacobiaan is in dit geval een constante - waarmee vermenigvuldigd moet worden, namelijk $J = \frac{x_{j+\frac{1}{2}} - x_{j-\frac{1}{2}}}{2}$. Geschreven wordt $\hat{c}_i(t) = Jc_i(t)$. In het vervolg wordt geschreven $c_i(t) = \hat{c}_i(t)$. Dit geeft dan de volgende vergelijking:

$$\sum_{i=1}^{N} c_i' \int_{-1}^{1} \varphi_i\varphi_m \, \mathrm{d}x + \sum_{i=1}^{N} c_i \int_{-1}^{1} \varphi_i'\varphi_m \, \mathrm{d}x = 0.$$

Door partiëel te integreren, wordt dan verkregen

$$\sum_{i=1}^{N} c_i' \int_{-1}^{1} \varphi_i \varphi_m \,\mathrm{d}x + \sum_{i=1}^{N} c_i \left[\varphi_i \varphi_m\right]_{-1}^{1} - \sum_{i=1}^{N} c_i \int_{-1}^{1} \varphi_i \varphi_m' \,\mathrm{d}x = 0. \qquad (9)$$

3.2.1. *Discretisatiematrices.* Nu wordt de eerste som uit (9) apart genomen:

$$\sum_{i=1}^{N} c_i'(t) \int_{-1}^{1} \varphi_i \varphi_m \,\mathrm{d}x$$

$$= \begin{pmatrix} \int_{-1}^{1} \varphi_1 \varphi_m \,\mathrm{d}x & \int_{-1}^{1} \varphi_2 \varphi_m \,\mathrm{d}x & \cdots & \int_{-1}^{1} \varphi_N \varphi_m \,\mathrm{d}x \end{pmatrix} \begin{pmatrix} c_1' \\ c_2' \\ \vdots \\ c_N' \end{pmatrix}.$$

Wordt dit zo voor iedere m gedaan, dan kunnen die vergelijkingen in de volgende matrixvergelijking gevangen worden:

$$\begin{pmatrix} \int_{-1}^{1} \varphi_1 \varphi_1 \,\mathrm{d}x & \int_{-1}^{1} \varphi_2 \varphi_1 \,\mathrm{d}x & \cdots & \int_{-1}^{1} \varphi_N \varphi_1 \,\mathrm{d}x \\ \int_{-1}^{1} \varphi_1 \varphi_2 \,\mathrm{d}x & \int_{-1}^{1} \varphi_2 \varphi_2 \,\mathrm{d}x & \cdots & \int_{-1}^{1} \varphi_N \varphi_2 \,\mathrm{d}x \\ \vdots & \vdots & \ddots & \vdots \\ \int_{-1}^{1} \varphi_1 \varphi_N \,\mathrm{d}x & \int_{-1}^{1} \varphi_2 \varphi_N \,\mathrm{d}x & \cdots & \int_{-1}^{1} \varphi_N \varphi_N \,\mathrm{d}x \end{pmatrix} \begin{pmatrix} c_1' \\ c_2' \\ \vdots \\ c_N' \end{pmatrix} =: M \mathbf{c}_J'. \qquad (10)$$

Merk op: de φ_j zijn eigenfuncties en derhalve orthogonaal. Dat wil zeggen dat de $\int_{-1}^{1} \varphi_j \varphi_i \mathrm{d}x = 0$, als $i \neq j$. Hierdoor is de matrix M een diagonaalmatrix met op de diagonaal $\int_{-1}^{1} \varphi_i^2 \mathrm{d}x$:

$$M = \begin{pmatrix} \int_{-1}^{1} \varphi_1 \varphi_1 \,\mathrm{d}x & 0 & \cdots & 0 \\ 0 & \int_{-1}^{1} \varphi_2 \varphi_2 \,\mathrm{d}x & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \int_{-1}^{1} \varphi_N \varphi_N \,\mathrm{d}x \end{pmatrix}.$$

Op dezelfde manier valt te beargumenteren dat de derde som uit (9) gerepresenteerd kan worden door

$$S\mathbf{c}_J := \begin{pmatrix} \int_{-1}^{1} \varphi_1 \varphi_1' \,\mathrm{d}x & \int_{-1}^{1} \varphi_2 \varphi_1' \,\mathrm{d}x & \cdots & \int_{-1}^{1} \varphi_N \varphi_1' \,\mathrm{d}x \\ \int_{-1}^{1} \varphi_1 \varphi_2' \,\mathrm{d}x & \int_{-1}^{1} \varphi_2 \varphi_2' \,\mathrm{d}x & \cdots & \int_{-1}^{1} \varphi_N \varphi_2' \,\mathrm{d}x \\ \vdots & \vdots & \ddots & \vdots \\ \int_{-1}^{1} \varphi_1 \varphi_N' \,\mathrm{d}x & \int_{-1}^{1} \varphi_2 \varphi_N' \,\mathrm{d}x & \cdots & \int_{-1}^{1} \varphi_N \varphi_N' \,\mathrm{d}x \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_N \end{pmatrix}.$$

De tweede som voor de stoktermen ligt iets gecompliceerder. Deze produceert immer twee matrixvergelijkingen, met gebruik van een upwind discretisatie Bij de berekening van de matrices A en B zal gebruik worden gemaakt van upwind flux. Ter herinnering, up-wind flux:

$$\overline{u}(x_{j-\frac{1}{2}}) = u(x^{-}_{j-\frac{1}{2}}).$$

Merk op:

$$\sum_{i=1}^{N} \left[c_i(t)\varphi_i(x)\varphi_m(x)\right]_{-1}^{1} = \sum_{i=1}^{N} \left[c_i(t)\varphi_i(x)\varphi_m(x)\right]_{x=1} - \sum_{i=1}^{N} \left[c_i(t)\varphi_i(x)\varphi_m(x)\right]_{x=-1}. \qquad (11)$$

Eigenlijk moet $\varphi_i(x)$ geschreven worden als $\varphi_i^J(x)$, anders is immers niet bekend over welk element Ω_J het hier gaat. Merk op:

$$\varphi_j^J(1) = 1^j, \tag{12}$$
$$\varphi_j^J(-1) = (-1)^j. \tag{13}$$

Dan wordt verkregen:

$$\begin{aligned}
\sum_{i=1}^{N} \left[c_i^J(t)\varphi_i^J(x)\varphi_m^J(x)\right]_{x=1} &= \sum_{i=1}^{N} \left[c_i^J(t)\varphi_i^J(x)\right]_{x=1} \varphi_m^J(1) \text{ en} \\
\sum_{i=1}^{N} \left[c_i^J(t)\varphi_i(x)\varphi_m(x)\right]_{x=-1} &= \sum_{i=1}^{N} \left[c_i^J(t)\varphi_i^J(x)\right]_{x=-1} \varphi_m^J(-1) \\
&= \sum_{i=1}^{N} \left[c_i^{J-1}(t)\varphi_i^{J-1}(x)\right]_{x=1} \varphi_m^J(-1).
\end{aligned}$$

En met upwind flux geeft dit:

$$\begin{aligned}
\sum_{i=1}^{N} c_i^J(t)\varphi_i^J(1)\varphi_m^J(1) &= \sum_{i=1}^{N} c_i^J(t), \text{ vanwege (12) en} \\
\sum_{i=1}^{N} c_i^{J-1}(t)\varphi_i^{J-1}(1)\varphi_m^J(-1) &= \sum_{i=1}^{N} (-1)^m c_i^{J-1}(t).
\end{aligned}$$

Vergelijking (11) kan dus ook weergegeven worden door $Ac_J - Bc_{J-1}$

$$\begin{aligned}
A\mathbf{c}_J &:= \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_N \end{pmatrix}, \\
B\mathbf{c_{J-1}} &:= \begin{pmatrix} 1 & 1 & \cdots & 1 \\ -1 & -1 & \cdots & -1 \\ \vdots & \vdots & \ddots & \vdots \\ -(-1)^N & -(-1)^N & \cdots & -(-1)^N \end{pmatrix} \begin{pmatrix} c_1^{J-1} \\ c_2^{J-1} \\ \vdots \\ c_N^{J-1} \end{pmatrix}.
\end{aligned}$$

Hierbij is $\mathbf{c}_{J-1}$ een representatie van $\mathbf{c}(t)$ van het controle-interval Ω_{j-1} naast het huidige controle-interval Ω_j – of, als J=1, het interval $[x_N, 1]$, dit vanwege de periodiciteit van de oplossing. Vanaf hier zal echter, voor de leesbaarheid, geschreven worden $\mathbf{c}_{J-1}$.

Op bovenstaande wijze is het probleem dus gereduceerd tot een matrixvergelijking, en wel de volgende:

$$M\mathbf{c}'_J - S\mathbf{c}_J + A\mathbf{c}_J - B\mathbf{c}_{J-1} = 0.$$

Het hele bovenstaande verhaal kan ook toegepast worden op de transportvergelijking met randvoorwaarde $u(0,t) = 1$:

$$\text{PDV} : \begin{cases} \frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} = 0 & , x \in (0,1), t > 0, \\ u(0,t) = 1 & , t > 0, \\ u(x,0) = \sin(2\pi x) & , x \in (0,1). \end{cases} \tag{14}$$

Dat resulteert eveneens in de matrixvergelijking $M\mathbf{c}'_J - S\mathbf{c}_J + A\mathbf{c}_J - B\mathbf{c}_{J-1} = 0$. Hierin zien de matrices er alleen iets anders uit. Matrices M,S en A blijven exact hetzelfde als voorheen én in matrix B verandert alleen de eerste kolom in $[1\ 0\ \dots]^T$.

3.2.2. *Tijdsintegratie.* De vergelijking kan opgelost worden met verschillende methoden. Er is voor gekozen om de simpelste methode te gebruiken, namelijk Euler Voorwaarts. Met Euler Voorwaarts wordt de volgende vergelijking verkregen voor de k-de tijdstap

$$M\mathbf{c}_J^{k+1} = (M + \Delta t(S - A))\mathbf{c}_J^k + \Delta t B\mathbf{c}_{J-1}^k.$$

Er had ook gekozen kunnen worden voor een methode met een hogere orde, zoals Runge-Kutta methode. Omdat dit vooral een illustratief voorbeeld is, is gekozen voor de simpele methode Euler Voorwaarts, waardoor er dus een iets mindere nauwkeurigheid is.

3.3. **De gemodificeerde transportvergelijking.** Een kleine uitbreiding van het bovenstaande is het toevoegen van de functie $c(x) = e^{\frac{-x^2}{\epsilon^2}}$, zoals in (5). Ook nu kan zonder verlies van algemeenheid ζ gelijk worden genomen aan 1. Wat resulteert in de volgende PDV, :

$$\text{PDV} : \begin{cases} 0 = \frac{\partial u}{\partial t} + \frac{\partial}{\partial x}(c'u) & x \in (0,1), t > 0, \\ u(0,t) = u(1,t) & t > 0, \\ u(x,0) = \cos(2\pi x) & x \in (0,1). \end{cases} \tag{15}$$

Deze vergelijking wordt op dezelfde manier aangepakt als de transportvergelijking. Eerst wordt vermenigvuldigd met testfunctie φ_m en vervolgens wordt geïntegreerd over het interval Ω_j.

$$0 = \int_{\Omega_j} \frac{\partial u}{\partial t}\varphi_m + \big(\frac{\partial}{\partial x}(c'u)\big)\varphi_m \mathrm{d}\Omega.$$

Vervolgens schrijven we $u(x,t)$ weer als $\sum_{i=1}^N c_i(t)\varphi_i(x)$ en vullen dit in

$$\begin{aligned} 0 &= \int_{\Omega_j} \big(\frac{\partial}{\partial t}\sum_{i=1}^N c_i\varphi_i\big)\varphi_m + \Big(\frac{\partial}{\partial x}\big(c'\sum_{i=1}^N c_i\varphi_i\big)\Big)\varphi_m \mathrm{d}\Omega \\ &= \sum_{i=1}^N c_i' \int_{\Omega_j} \varphi_i\varphi_m \mathrm{d}\Omega + \sum_{i=1}^N c' c_i\varphi_i\varphi_m\Big|_{\Omega_j} + \sum_{i=1}^N \int_{-1}^1 c' c_i\varphi_i \frac{\partial \varphi_m}{\partial x}\mathrm{d}\Omega. \end{aligned}$$

Ook nu weer wordt de coördinatentransformatie van Ω_j naar $[-1,1]$ toegepast, wat zorgt voor een extra constante, ten gevolge van de Jacobiaan, waarmee vermenigvuldigd moet worden, namelijk $J = \frac{x_{j+\frac{1}{2}} - x_{j-\frac{1}{2}}}{2}$. Geschreven wordt $\hat{c}_i(t) = Jc_i(t)$. Daarna wordt geschreven $c_i(t) = \hat{c}_i(t)$. Hiermee wordt de bovenstaande vergelijking geschreven als

$$0 = \sum_{i=1}^N c_i' \int_{-1}^1 \varphi_i\varphi_m \mathrm{d}x + \sum_{i=1}^N c' c_i\varphi_i\varphi_m\Big|_{-1}^1 + \sum_{i=1}^N \int_{-1}^1 c' c_i\varphi_i \frac{\partial \varphi_m}{\partial x}\mathrm{d}x.$$

3.3.1. *Discretisatiematrices.* Nu wordt de eerste som apart genomen:

$$\begin{aligned} &\sum_{i=1}^N c_i' \int_{-1}^1 \varphi_i\varphi_m \,\mathrm{d}x \\ &= \begin{pmatrix} \int_{-1}^1 \varphi_1\varphi_m \,\mathrm{d}x & \int_{-1}^1 \varphi_2\varphi_m \,\mathrm{d}x & \cdots & \int_{-1}^1 \varphi_N\varphi_m \,\mathrm{d}x \end{pmatrix} \begin{pmatrix} c_1' \\ c_2' \\ \vdots \\ c_N' \end{pmatrix}. \end{aligned}$$

Wordt dit zo voor iedere m gedaan, dan kunnen die vergelijkingen in de volgende matrixvergelijking gevangen worden:

$$\begin{pmatrix} \int_{-1}^1 \varphi_1\varphi_1 \,\mathrm{d}x & \int_{-1}^1 \varphi_2\varphi_1 \,\mathrm{d}x & \cdots & \int_{-1}^1 \varphi_N\varphi_1 \,\mathrm{d}x \\ \int_{-1}^1 \varphi_1\varphi_2 \,\mathrm{d}x & \int_{-1}^1 \varphi_2\varphi_2 \,\mathrm{d}x & \cdots & \int_{-1}^1 \varphi_N\varphi_2 \,\mathrm{d}x \\ \vdots & \vdots & \ddots & \vdots \\ \int_{-1}^1 \varphi_1\varphi_N \,\mathrm{d}x & \int_{-1}^1 \varphi_2\varphi_N \,\mathrm{d}x & \cdots & \int_{-1}^1 \varphi_N\varphi_N \,\mathrm{d}x \end{pmatrix} \begin{pmatrix} c_1' \\ c_2' \\ \vdots \\ c_N' \end{pmatrix} =: M\mathbf{c}_J'. \tag{16}$$

Vanwege de eerder genoemde orthogonaliteit, is ook nu weer M te schrijven als de diagonaalmatrix,

$$M = \begin{pmatrix} \int_{-1}^{1} \varphi_1\varphi_1 \,\mathrm{d}x & 0 & \cdots & 0 \\ 0 & \int_{-1}^{1} \varphi_2\varphi_2 \,\mathrm{d}x & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \int_{-1}^{1} \varphi_N\varphi_N \,\mathrm{d}x \end{pmatrix}.$$

Ook weer op dezelfde manier valt te beargumenteren dat de derde som gerepresenteerd kan worden door

$$S\mathbf{c}_J \;:=\; \begin{pmatrix} \int_{-1}^{1} c'\varphi_1\varphi_1' \,\mathrm{d}x & \int_{-1}^{1} c'\varphi_2\varphi_1' \,\mathrm{d}x & \cdots & \int_{-1}^{1} c'\varphi_N\varphi_1' \,\mathrm{d}x \\ \int_{-1}^{1} c'\varphi_1\varphi_2' \,\mathrm{d}x & \int_{-1}^{1} c'\varphi_2\varphi_2' \,\mathrm{d}x & \cdots & \int_{-1}^{1} c'\varphi_N\varphi_2' \,\mathrm{d}x \\ \vdots & \vdots & \ddots & \vdots \\ \int_{-1}^{1} c'\varphi_1\varphi_N' \,\mathrm{d}x & \int_{-1}^{1} c'\varphi_2\varphi_N' \,\mathrm{d}x & \cdots & \int_{-1}^{1} c'\varphi_N\varphi_N' \,\mathrm{d}x \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_N \end{pmatrix}.$$

De tweede som ligt iets gecompliceerder en produceert twee matrixvergelijkingen. De som kan als volgt ontleed worden:

$$\sum_{i=1}^{N} \left[c' c_i\varphi_i\varphi_m\right]_{-1}^{1} = \sum_{i=1}^{N} \left[c' c_i\varphi_i\varphi_m\right]_{x=1} - \sum_{i=1}^{N} \left[c' c_i\varphi_i\varphi_m\right]_{x=-1}. \tag{17}$$

Bij de berekening van de matrices A en B zal gebruik worden gemaakt, net als in de berekening in sectie 3.2.1, van upwind discretisatie.

Merk op: Eigenlijk moet φ_i geschreven worden als φ_i^J, anders is immers niet bekend over welk element Ω_J het hier gaat.

Dan wordt verkregen:

$$\begin{aligned} \sum_{i=1}^{N} \left[c'(x)c_i^J(t)\varphi_i^J(x)\varphi_m^J(x)\right]_{x=1} &= \sum_{i=1}^{N} \left[c_i^J(t)\varphi_i^J(x)\right]_{x=1} c'(1)\varphi_m^J(1) \text{ en} \\ \sum_{i=1}^{N} \left[c'(x)c_i^J(t)\varphi_i(x)\varphi_m(x)\right]_{x=-1} &= \sum_{i=1}^{N} \left[c_i^J(t)\varphi_i^J(x)\right]_{x=-1} c'(-1)\varphi_m^J(-1) \\ &= \sum_{i=1}^{N} \left[c_i^{J-1}(t)\varphi_i^{J-1}(x)\right]_{x=1} c'(-1)\varphi_m^J(-1). \end{aligned}$$

En met upwind flux geeft dit:

$$\begin{aligned} \sum_{i=1}^{N} c_i^J(t)\varphi_i^J(1)c'(1)\varphi_m^J(1) &= c'(1)\sum_{i=1}^{N} c_i^J(t), \text{ vanwege (12) en} \\ \sum_{i=1}^{N} c_i^{J-1}(t)\varphi_i^{J-1}(1)c'(-1)\varphi_m^J(-1) &= c'(-1)\sum_{i=1}^{N}(-1)^m c_i^{J-1}(t). \end{aligned}$$

Vergelijking (11) kan dus ook weergegeven worden door $Ac_J - Bc_{J-1}$

$$A\mathbf{c}_J := \begin{pmatrix} c'(1) & c'(1) & \cdots & c'(1) \\ c'(1) & c'(1) & \cdots & c'(1) \\ \vdots & \vdots & \ddots & \vdots \\ c'(1) & c'(1) & \cdots & c'(1) \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_N \end{pmatrix},$$

$$B\mathbf{c_{J-1}} := \begin{pmatrix} c'(-1) & c'(-1) & \cdots & c'(-1) \\ -c'(-1) & -c'(-1) & \cdots & -c'(-1) \\ \vdots & \vdots & \ddots & \vdots \\ -(-1)^N c'(-1) & -(-1)^N c'(-1) & \cdots & -(-1)^N c'(-1) \end{pmatrix} \begin{pmatrix} c_1^{J-1} \\ c_2^{J-1} \\ \vdots \\ c_N^{J-1} \end{pmatrix}.$$

Hierbij is $\mathbf{c}_{J-1}$ een representatie van $\mathbf{c}(t)$ van het controle-interval Ω_{j-1} naast het huidige controle-interval Ω_j – of, als J=1, het 'virtuele' interval $[-\Delta x_j, 0]$, waardoor de randvoorwaarde zich op de juiste manier manifesteert. Vanaf hier zal echter, voor de leesbaarheid, geschreven worden $\mathbf{c}_{J-1}$.

Op bovenstaande wijze is het probleem dus wederom gereduceerd tot de matrixvergelijking $M\mathbf{c}'_J - S\mathbf{c}_J + A\mathbf{c}_J - B\mathbf{c}_{J-1} = 0$.

3.3.2. *Tijdsintegratie.* In tegenstelling tot eerder, bij de transportvergelijking, wordt nu gebruik gemaakt van een integratiemethode met hogere orde, namelijk de Runge-Kutta methode. De fout wordt hiermee van orde $\mathcal{O}(\Delta x^4)$ in plaats van $\mathcal{O}(\Delta x)$.

De Runge-Kutta methode wordt toegepast op de matrixvergelijking. Het betreft de volgende vergelijking

$$M\mathbf{c}'_J = S\mathbf{c}_J - A\mathbf{c}_J + B\mathbf{c}_{J-1}.$$

Die kan worden omgeschreven tot

$$\mathbf{c}'_J = M^{-1}(S\mathbf{c}_J - A\mathbf{c}_J + B\mathbf{c}_{J-1}).$$

Noem $f(c_J) = M^{-1}(S\mathbf{c}_J - A\mathbf{c}_J + B\mathbf{c}_{J-1})$. Dus $\mathbf{c}'_J = f(\mathbf{c}_J)$. Dan geldt:

$$\begin{aligned} \mathbf{c}_{i+1} &= \mathbf{c}_i + \frac{\Delta t}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4), \\ \mathbf{k}_1 &= f(\mathbf{c}_J), \\ \mathbf{k}_2 &= f(\mathbf{c}_J + \frac{\Delta t}{2}\mathbf{k}_1), \\ \mathbf{k}_3 &= f(\mathbf{c}_J + \frac{\Delta t}{2}\mathbf{k}_2), \\ \mathbf{k}_4 &= f(\mathbf{c}_J + \Delta t\mathbf{k}_1). \end{aligned}$$

3.4. **De warmtevergelijking.** De volgende vergelijking die bekeken wordt is de warmtevergelijking met $c(x,0) = \sin(\pi x)$:

$$\begin{cases} 0 = \frac{\partial c}{\partial t} - \epsilon \frac{\partial^2 c}{\partial x^2} & x \in (0,1), t > 0, \epsilon > 0, \\ 0 = \frac{\partial c}{\partial x}(0,t) = \frac{\partial c}{\partial x}(1,t) & t > 0, \\ c(x,0) = \sin(\pi x) & x \in (0,1). \end{cases}$$

In wezen is de aanpak ook hier weer gelijk aan de aanpak van de vorige problemen. De vergelijking wordt vermenigvuldigd met een testfunctie, vervolgens geïntegreerd én uiteindelijk zal door discretisatie een aantal matrices gevonden worden waarmee een oplossing voor deze vergelijking benaderd wordt. Echter de dubbele afgeleide naar x maakt het een beetje lastig. Daarom wordt de tweede-orde differentiaalvergelijking eerst omgeschreven naar een stelsel van twee eerste-orde differentiaalvergelijkingen. Daartoe wordt een dummyvariabele p geïntroduceerd:

$$\begin{cases} 0 = \frac{\partial c}{\partial t} - \epsilon \frac{\partial^2 c}{\partial x^2} & x \in (0,1), t > 0, \epsilon > 0, \\ 0 = \frac{\partial c}{\partial x}(0,t) = \frac{\partial c}{\partial x}(1,t) & t > 0, \\ c(x,0) = \sin(\pi x) & x \in (0,1). \end{cases}$$

Wat met geïntroduceerde dummyvariabel geschreven kan worden als:

$$\begin{cases} 0 = \frac{\partial c}{\partial t} - \epsilon \frac{\partial p}{\partial x} & x \in (0,1), t > 0, \epsilon > 0, \\ 0 = p - \frac{\partial c}{\partial x} & x \in (0,1), t > 0, \\ 0 = \frac{\partial c}{\partial x}(0,t) = \frac{\partial c}{\partial x}(1,t) & t > 0, \\ c(x,0) = \sin(\pi x) & x \in (0,1), \\ p(x,0) = \pi \cos(\pi x) & x \in (0,1). \end{cases}$$

Nu kan worden verder gegaan met het toepassen van de DG-methode. Op die manier wordt verkregen:

$$\begin{aligned} & \begin{cases} 0 = \frac{\partial c}{\partial t} - \epsilon \frac{\partial p}{\partial x}, \\ 0 = p - \frac{\partial c}{\partial x}, \end{cases} \\ \Leftrightarrow \quad & \begin{cases} 0 = \int_{\Omega_j} (\frac{\partial c}{\partial t} - \epsilon \frac{\partial p}{\partial x}) \varphi_m \mathrm{d}\Omega, \\ 0 = \int_{\Omega_j} (p - \frac{\partial c}{\partial x}) \varphi_m \mathrm{d}\Omega, \end{cases} \\ \Leftrightarrow \quad & \begin{cases} 0 = \int_{\Omega_j} \frac{\partial c}{\partial t} \varphi_m \mathrm{d}\Omega - \int_{\Omega_j} \epsilon \frac{\partial p}{\partial x} \varphi_m \mathrm{d}\Omega, \\ 0 = \int_{\Omega_j} p \varphi_m \mathrm{d}\Omega - \int_{\Omega_j} \frac{\partial c}{\partial x} \varphi_m \mathrm{d}\Omega. \end{cases} \end{aligned}$$

Vanaf nu schrijven we $c(x,t) = \sum_{i=1}^{N} c_i(t)\varphi_i(x)$ en $p(x,t) = \sum_{i=1}^{N} p_i(t)\varphi_i(x)$. Tevens zal wederom de coördinatentransformatie van Ω_j naar [-1,1] worden doorgevoerd, waardoor eigenlijk dus geschreven zou moeten worden $\hat{c}_i$, ook nu weer zal dit, voor de leesbaarheid, geschreven worden als "c_i". De berekening wordt

weer hervat door bovenstaande aanpassingen door te voeren:

$$\begin{cases} 0 = \int_{-1}^{1} (\frac{\partial}{\partial t} \sum_{i=1}^{N} c_i \varphi_i) \varphi_m \mathrm{d}x - \epsilon \int_{-1}^{1} (\frac{\partial}{\partial x} \sum_{i=1}^{N} p_i \varphi_i) \varphi_m \mathrm{d}x, \\ 0 = \int_{-1}^{1} \sum_{i=1}^{N} p_i \varphi_i \varphi_m \mathrm{d}x - \int_{-1}^{1} (\frac{\partial}{\partial x} \sum_{i=1}^{N} c_i \varphi_i) \varphi_m \mathrm{d}x, \end{cases}$$

$$\Leftrightarrow \quad \begin{cases} 0 = \sum_{i=1}^{N} c_i' \int_{-1}^{1} \varphi_i \varphi_m \mathrm{d}x - \epsilon \sum_{i=1}^{N} \int_{-1}^{1} p_i \varphi_i' \varphi_m \mathrm{d}x, \\ 0 = \sum_{i=1}^{N} \int_{-1}^{1} p_i \varphi_i \varphi_m \mathrm{d}x - \sum_{i=1}^{N} \int_{-1}^{1} c_i \varphi_i' \varphi_m \mathrm{d}x, \end{cases}$$

$$\Leftrightarrow \quad \begin{cases} 0 = \sum_{i=1}^{N} c_i' \int_{-1}^{1} \varphi_i \varphi_m \mathrm{d}x + \epsilon \sum_{i=1}^{N} \int_{-1}^{1} p_i' \varphi_i \varphi_m \mathrm{d}x - \epsilon \sum_{i=1}^{N} p_i \Big[\varphi_i \varphi_m \Big]_{-1}^{1}, \\ 0 = \sum_{i=1}^{N} p_i \int_{-1}^{1} \varphi_i \varphi_m \mathrm{d}x + \sum_{i=1}^{N} \int_{-1}^{1} c_i' \varphi_i \varphi_m \mathrm{d}x - \sum_{i=1}^{N} c_i \Big[\varphi_i \varphi_m \Big]_{-1}^{1}. \end{cases}$$

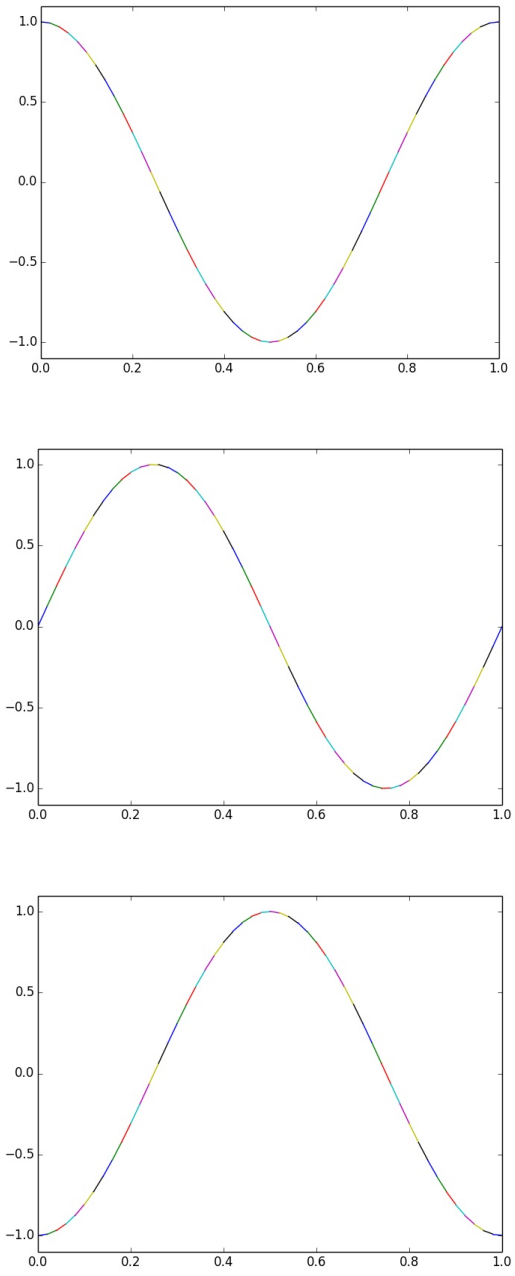
Deze laatste sommen komen overeen met de sommen uit sectie 3.2.1. Zodoende kan dit gezien worden als

$$\begin{cases} 0 = M \mathbf{c_J'} - \epsilon (S - A) \mathbf{p_J} - \epsilon B \mathbf{p_{J-1}}, \\ 0 = M \mathbf{p_J} - (S - A) \mathbf{c_J} - B \mathbf{c_{J-1}}. \end{cases}$$

Waarbij de matrices M, S, A en B gelijk zijn aan de gelijknamige matrices berekend in sectie 3.2.1.

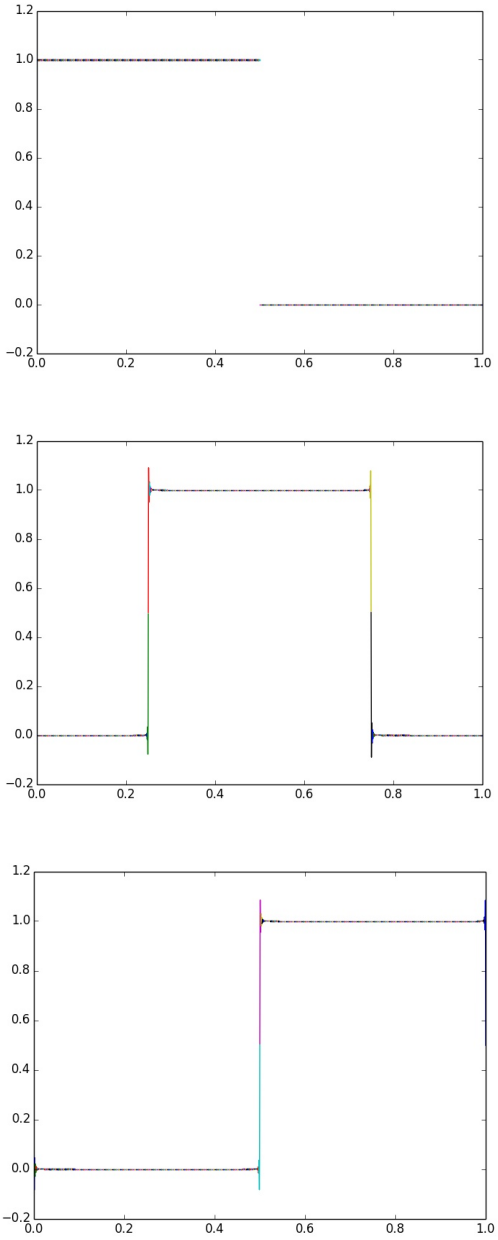
4. Simulaties

4.1. **De transportvergelijking.** De oplossing van het simpele model van differentiaalvergelijking (7), met bijbehorende randvoorwaarde kent een exacte oplossing, namelijk een lopende golf: $u(x,t) = \cos(2\pi(x-t))$. Dit komt overeen met de uitkomsten van het model (zie Figuur 2).



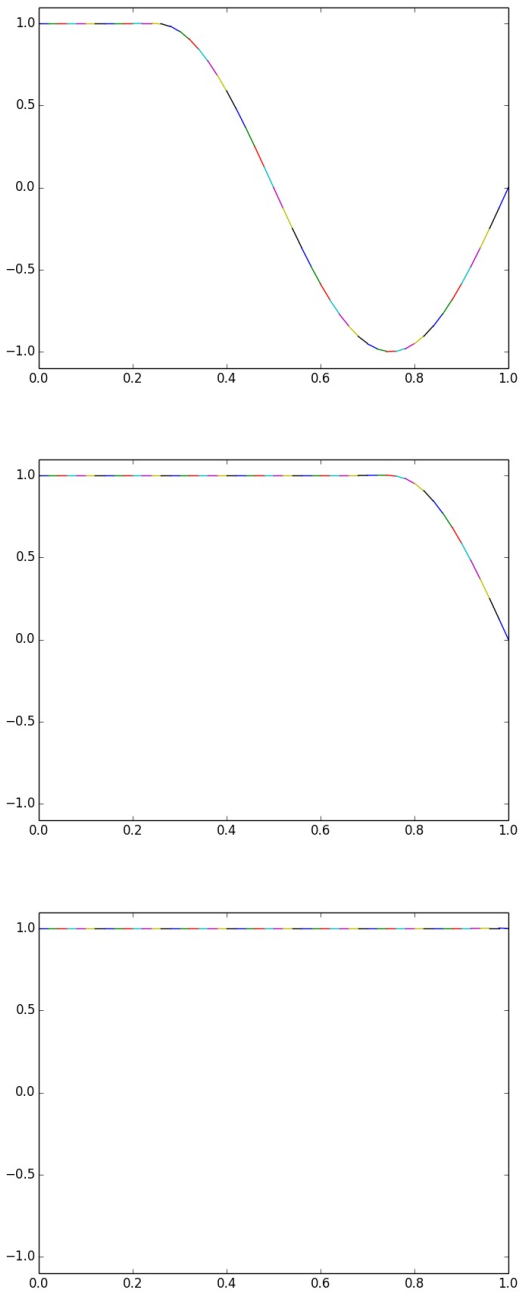
Figuur 2. Een numerieke benadering van de oplossing, met $k = 2$, $\Delta x = \frac{1}{50}$, $\Delta t = \frac{1}{1000}$, t is respectievelijk 0, 0.25 en 0.5.

Het kan interessant zijn bovenstaande simulatie te bekijken met een discontinue beginvoorwaarde $u(x,0) = \begin{cases} 1 & \text{, als } x < 0.5 \\ 0 & \text{, elders} \end{cases} = 1 - H(x - 0.5)$, met H de heavisidefuctie. In dit geval wordt nog steeds een 'lopende golf' verwacht als uitkomst, met periode gelijk aan 1. De DG-methode gaat redelijk om met de discontinuïteiten, zie figuur 3 . Merk op: de wiggles bij de sprongen zijn met filteringstechnieken weg te werken.



FIGUUR 3. Een numerieke benadering van de oplossing, met $k = 8$, $\Delta x = \frac{1}{400}$, $\Delta t = \frac{1}{10000}$, t is respectievelijk 0, 0.25 en 0.5.

De oplossing van het model van differentiaalvergelijking (14), kent een iets andere oplossing. Deze convergeert, als t maar groot genoeg, naar de steady-stateoplossing : $u(x,t) = 1$. Dit komt overeen met de uitkomsten van het model (zie Figuur 4). Er is hier alleen gekozen voor de Runge-Kutta methode voor een kleinere fout.



FIGUUR 4. Een numerieke benadering van de oplossing, met $k = 2$, $\Delta x = \frac{1}{50}$, $\Delta t = \frac{1}{1000}$, t is respectievelijk 0.25, 0.75 en 1.0.

4.2. **De gemodificeerde transportvergelijking.** Bekend is dat $\frac{\partial u}{\partial t} = -\alpha^2 u$ een oplossing heeft die naar 0 convergeert, als t naar oneindig gaat én dat de oplossing van $\frac{\partial u}{\partial t} = -\alpha^2 u$ divergeert, als t naar oneindig gaat. Ter herinnering, de gemodificeerde transportvergelijking luidde:

$$\text{PDV} : \begin{cases} 0 = \frac{\partial u}{\partial t} + \frac{\partial}{\partial x}(c'u) & x \in (0,1), t > 0, \\ u(0,t) = u(1,t) & t > 0, \\ u(x,0) = \cos(2\pi x) & x \in (0,1). \end{cases}$$

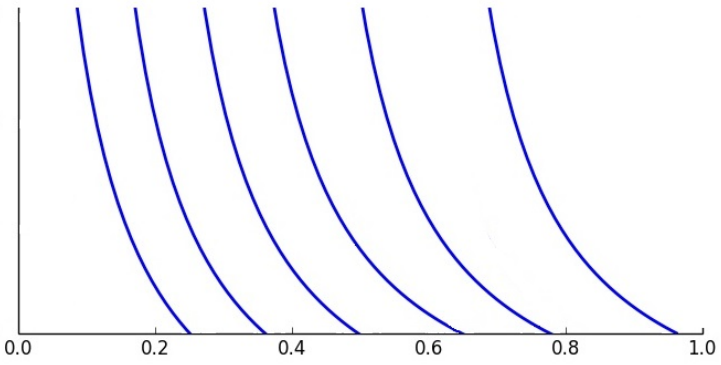
Wordt het bovenstaande bekeken met over karakteristieken dan is het volgende te zien:

$$\begin{cases} \frac{\mathrm{d}}{\mathrm{d}t} u(x(t),t) & = -c''(x(t))u(x(t),t), \\ \frac{\mathrm{d}x}{\mathrm{d}t} & = c'(x(t)), \end{cases}$$

(Voor het gemak en de leesbaarheid wordt geschreven $x = x(t)$)

$$\Leftrightarrow \quad \begin{cases} \frac{\mathrm{d}}{\mathrm{d}t} u(x,t) & = -\frac{2}{\epsilon^2} e^{\frac{-x^2}{2\epsilon^2}} (\frac{2x^2}{\epsilon^2} - 1) u(x,t), \\ \frac{\mathrm{d}x}{\mathrm{d}t} & = -\frac{2x}{\epsilon^2} e^{\frac{-x^2}{2\epsilon^2}}. \end{cases}$$

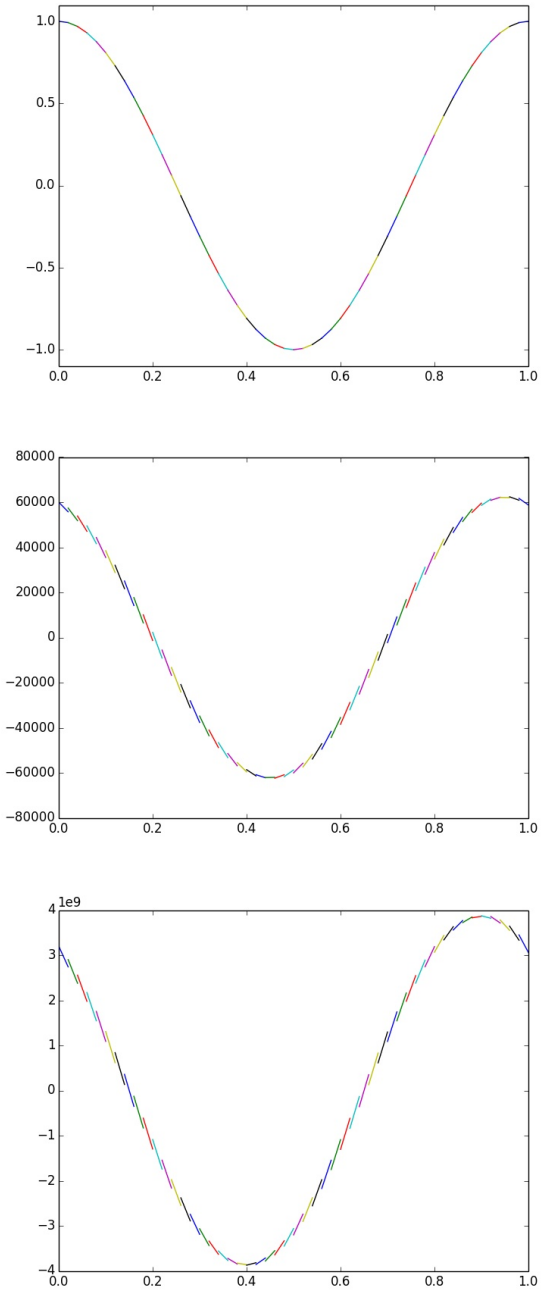
$-\frac{2}{\epsilon^2} e^{\frac{-x^2}{2\epsilon^2}} (\frac{2x^2}{\epsilon^2} - 1) > 0 \Leftrightarrow (\frac{2x^2}{\epsilon^2} - 1) < 0$. Ongeacht welke ϵ gekozen wordt komt is er een waarde waarvoor geldt dat alle x kleiner dan die waarde divergeren. Als we kijken naar de karakteristieken. Dan zien we dat $\frac{\partial x}{\partial t} < 0$. Dus zien de karakteristieken er ongeveer uit als in figuur 5:



FIGUUR 5. Een schets van hoe de karakteristieken lopen.

Kortom, $x(t)$ wordt als maar dichter naar nul getrokken, de vergelijking zal dus na verloop van tijd divergeren, ongeacht de keuze van ϵ.

Ter illustratie daarvan is in figuur 6 de numerieke oplossing weergegeven met $\epsilon = 2$.



FIGUUR 6. Een numerieke benadering van de oplossing, met $k = 2$, $\Delta x = \frac{1}{50}$, $\Delta t = \frac{1}{1000}$, t is respectievelijk 0, 0.5, en 1.0.

Daar deze divergeert is het misschien interessanter om te kijken naar dezelfde partiële differentiaalvergelijking, maar dan met $c(x)$ de primitieve van de $c(x) = e^{\frac{-x^2}{\epsilon^2}}$. In dat geval kan weer gekeken worden naar de karakteristieken. Dan is

het volgende te zien:

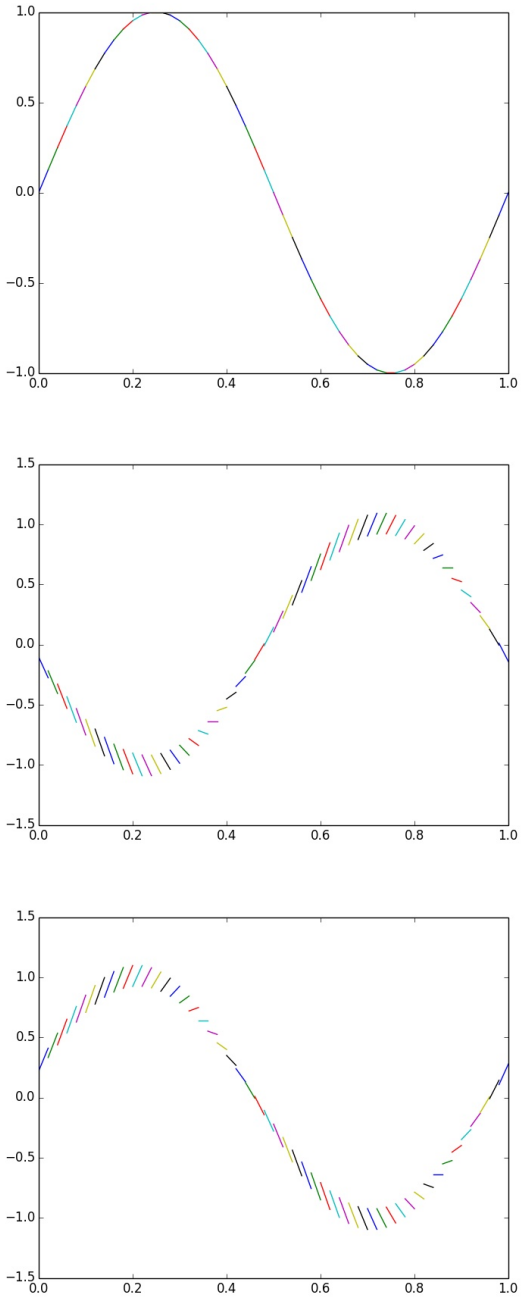
$$\begin{cases} \frac{\mathrm{d}}{\mathrm{d}t}u(x(t),t) &= -c''(x(t))u(x(t),t), \\ \frac{\mathrm{d}x}{\mathrm{d}t} &= c'(x(t)), \end{cases}$$

(Wederom voor gemak en leesbaarheid wordt geschreven $x = x(t)$)

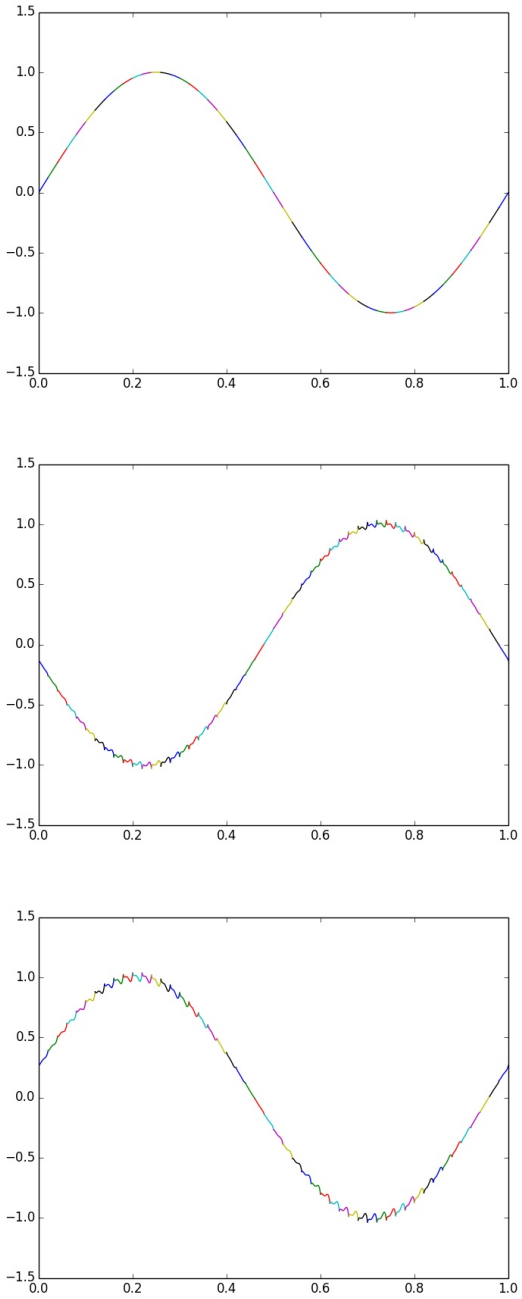
$$\Leftrightarrow \quad \begin{cases} \frac{\mathrm{d}}{\mathrm{d}t}u(x,t) &= \frac{2x}{\epsilon^2}e^{\frac{-x^2}{2\epsilon^2}}u(x,t), \\ \frac{\mathrm{d}x}{\mathrm{d}t} &= e^{\frac{-x^2}{\epsilon^2}}. \end{cases}$$

Hierbij zien we dat $\frac{2x}{\epsilon^2}e^{\frac{-x^2}{2\epsilon^2}} > 0$ én ook $\frac{\partial x}{\partial t} = e^{\frac{-x^2}{\epsilon^2}} > 0$. Kortom, de karakteristieken lopen tegenovergesteld en deze differentiaalvergelijking zal altijd convergeren.

Het programma laten runnen met $k = 2$ laat dit al zien (zie figuur 7), echter geeft de benadering met tweede-orde Legendre-polynomen een grote fout. Hoewel ook de benadering met Legendre-polynomen van orde acht nog wat rimpelingen (Engels: 'wiggles') vertoont, is duidelijk te zien dat dit convergeert (zie figuur 8).



FIGUUR 7. Een numerieke benadering van de oplossing, met $k = 2$, $\Delta x = \frac{1}{50}$, $\Delta t = \frac{1}{1000}$, t is respectievelijk 0, 0.5, en 1.0.



FIGUUR 8. Een numerieke benadering van de oplossing, met $k = 8$, $\Delta x = \frac{1}{50}$, $\Delta t = \frac{1}{1000}$, t is respectievelijk 0, 0.5, en 1.0.

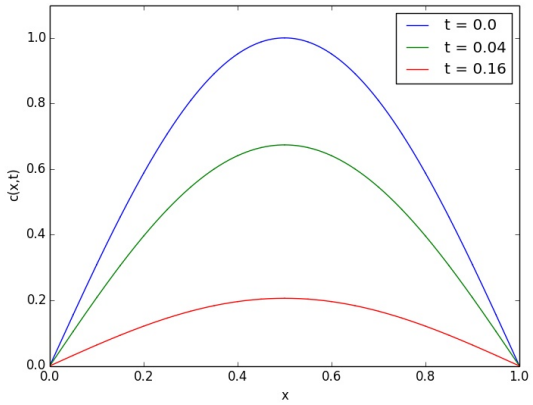
4.3. **De warmtevergelijking.** De exacte oplossing van de warmtevergelijking met randvoorwaarden zoals besproken in sectie 3.4, is

$$c(x,t) = \sin(\pi x)e^{-\pi^2 t}.$$

De oplossing zal dus steeds verder naar nul gaan, immers

$$\begin{aligned} 0 &\leq \lim_{t\to\infty} |\sin(\pi x)e^{-\pi^2 t}| \\ &= \lim_{t\to\infty} |\sin(\pi x)||e^{-\pi^2 t}| \\ &\leq \lim_{t\to\infty} |e^{-\pi^2 t}| \\ &= 0, \end{aligned}$$

wat ook ondersteund wordt door figuur 9. Dit alles echter komt niet overeen



FIGUUR 9. De exacte oplossing van de warmtevergelijking, op $t = 0.0, 0.04, 0.16$.

met de verkregen simulaties. Het runnen van de achterliggende python-code geeft een divergerende, discontinuë en verre van redelijke oplossing. Er zijn hiervoor verschillende redenen te bedenken:

(1) De integratiemethode is niet stabiel door de gekozen tijdstap;
(2) De DG-methode is verkeerd toegepast;
(3) Er zit een fout in de programmacode;
(4) Python kan niet goed omgaan met vermenigvuldigingen van matrices met hele kleine getallen.

Hierin zijn mogelijkheid (1) en (2) direct af te schrijven. Er kan geen spraken zijn van een te kleine tijdstap, volgens C. Vuik et al. (2016) is bij de Runge-Kutta methode, die hier gebruikt is, stabiliteit als $\Delta t < -\frac{2.8}{\lambda}$, met $\lambda = -\frac{\epsilon}{\Delta x^2} = \frac{-1}{\Delta x^2}$. Dus stabiliteit met $\Delta t < \frac{2.8\Delta x^2}{\epsilon}$. Er is gebruik gemaakt van $\Delta t = \frac{1}{10000}$ en $\Delta x = \frac{1}{20}$. Dus er is zeker aan deze voorwaarde voor stabiliteit voldaan.

Ook mogelijkheid (2) kan direct uitgesloten worden. De DG-methode is namelijk exact zo toegepast als bij de vorige problemen én toen kwam er ook een goed resultaat uit.

Dat betekent dat er dus ofwel een fout in de programmacode zit ofwel een narigheid in de programmeeromgeving. Al met al zijn er dus geen bruikbare resultaten behaald voor deze laatste vergelijking.

5. Conclusies

Resumerend: er is een aantal vergelijkingen in relatie tot de chemotaxis, dat het volgende stelsel partiële differentiaalvergelijkingen kent:

$$\begin{cases} 0 = \frac{\partial u}{\partial t} + \frac{\partial}{\partial x}(\zeta \frac{\partial c}{\partial x} u) & x \in (0,1), t > 0, \\ 0 = \frac{\partial c}{\partial t} - \epsilon \frac{\partial^2 c}{\partial x^2} & x \in (0,1), t > 0, \\ u(x,t) = u_0(x) & x \in (0,1). \end{cases}$$

De vergelijkingen die zijn bekeken zijn

(1) de transportvergelijking,
(2) een kleine variatie op de transportvergelijking,
(3) én de warmtevergelijking.

De DG-methode geeft voor de eerste twee vergelijkingen een goede benadering (zie figuur 2 3, 6 en 8). Hierin is de methode dus succesvol gebleken. Ook is laten zien dat hogere orde van de orde verkregen kan worden. Ook is te zien dat bij discontinue randvoorwaarde het gewenste resultaat verschijnt. Uiteraard is er wat 'overshoot'. Deze overshoot kan nog wel verkleind, en dus verbeterd, worden door zogenoemde filtering technieken en slope/flux limiting technieken. Er kan dus zeker geconcludeerd worden dat de discontinue Galerkinmethode hier succesvol blijkt.

Ook is nog gekeken naar het gedrag van deze partiële differentiaalvergelijkingen door te kijken naar het gedrag over de karakteristieken. Aan de hand daarvan is geconcludeerd dat het gedrag de numerieke oplossing overeenkomt met het gedrag dat bij deze vergelijking hoort.

Verder is er nog gekeken naar de laatste vergelijking, de warmtevergelijking, hoewel hier geen bruikbaar numeriek resultaat uitgekomen is - en het derhalve ook nog niet mogelijk is de resultaten van de eerste en de tweede vergelijking te combineren tot een resultaat voor het gehele model - is wel laten zien wat het gedrag van de oplossing is, het gaat richting nul.

Kortom, er is laten zien dat de DG-methode een hele nuttige methode is, onder andere in het bekijken van discontinuïteiten, maar ook zeker voor het oplossen van problemen als de transportvergelijking.

Literatuurlijst

Vuik, C., Vermolen, F., van Gijzen, M., Vuik, M. (2016). Numerical Methods for Ordanary Differential Equations (tweede editie). Delft, Nederland: Delft Academic Press.

Van Kan, J., Segal, A., Vermolen, F. (2014). Numerical Methods in Scientific Computing (tweede editie). Delft, Nederland: Delft Academic Press.

Biology Online (z.d.) dictionary/Chemotaxis. Geraadpleegd op 7 september 2017 van: http://www.biology-online.org/dictionary/Chemotaxis

Roberts, B., Chung, E., Yu, S., Li, S. (2012). Keller-Segel Models for Chemotaxis. Geraadpleegd op 16 april 2018, van http://isn.ucsd.edu/courses/Beng221/problems/2012/BENG221_Project%20-%20Roberts%20Chung%20Yu%20Li.pdf

britannica.com. (z.d.). Fibroblast. Geraadpleegd op 16 april 2018, van https://www.britannica.com/science/fibroblast

leef.nl. (z.d.) Wondgenezing. Geraadpleegd op 7 januari 2018, van https://www.leef.nl/kennisbank/wondgenezing/

Vermolen, F. J., Crapts, L. Y. D., Ryan, J. K. (2018). A Discontinuous Galerkin Model for the Simulation of Chemotaxis Processes: Application to Stem Cell Injection After a Myocardial Infarction.

Crapts, L. Y. D. (2012). Modeling an angiogenesis treatment after a myocardial infarction.

6. Bijlagen

Bijlage 1: Pythoncode behorend bij de transportvergelijking en de gemodificeerde transportvergelijking:

```
import numpy as np
import matplotlib as mpl
from math import *
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
from sympy import *
from matplotlib.pyplot import figure, show
from numpy import arange, sin, pi
import pylab
import scipy.integrate as integrates

def diff(y,n):
    #Differentieer y naar x n keer
    return y.diff(x,n)

def diff2(y,n):
    #Differentieer en vermenigvuldig met float(n-i) om extreem grote getallen te voorkomen.
    for i in range (0,n):
        y=y.diff(x)/(float(n-i))
    return y

x = Symbol('x')

def k(x, x0, eps):
    #de functie c' uit de gemodificeerde transportvgl.
    return ((-2*(x-x0)/(4*eps))*(exp(-(x-x0)*(x-x0)/(4*eps))))
    #of de andere die gebruikt is:
    #if (t==0.0):
    #    return 1.0
    #return (-2*(x-x0)/(4*eps))*(exp(-(x-x0)*(x-x0)/(4*eps))/sqrt(4*eps*pi))

def phi(n):
    #bereken de eigenfuncties
    y=x**2 - 1
    z= 1/(factorial(n)*2**(n))*diff(y**(n),n)
    return z

def phiAfgel(n):
    #bereken de afgeleide van phi
    y=x**2 - 1
    z= 1/(factorial(n)*2**n)*diff(y**n,n+1)
    return z

def Waarde(z,x0):
    #bereken de warde van f, waarbij voor x z wordt ingevuld
    f = lambdify(x, z, 'numpy')
    return f(x0)

#4 verschillende integraties:
def Int1(a,b):
    g = phi(a)*phiAfgel(b)
    f=integrate(g,x)
    return Waarde(f,1)-Waarde(f,-1)

def Int2(a,b):
    g = phi(a)*phi(b)
    f=integrate(g,x)
    return Waarde(f,1)-Waarde(f,-1)

def Int3(a,b):
    g = phi(a)*phiAfgel(b)*k(x, 0.0, 2.0)
    f= integrates.quad(lambda xv: g.subs(x,xv), -1.0,1.0)[0]
    return f

def Int4(a,b):
    g = phi(a)*phi(b)
    return Waarde(g,1)

def CreateM(Orde):
    #bereken M, waarbij bekend is dat alleen elementen
    #buiten de diagonaal een waarde ongelijk 0 hebben
    M=[]
    for i in range (0,Orde+1):
        a=[]
        for j in range (0,Orde+1):
            if (i==j):
                a.append(Int2(i,i)/2)
            else:
                a.append(0.0)
        M.append(a)
    M=np.array(M)
    return M
```

```
def CreateS(Orde):
    #bereken S, door het simpelweg uit te rekenen
    S=[]
    for i in range (0,Orde+1):
        a=[]
        for j in range (0,Orde+1):
            a.append(Int1(j,i))
        S.append(a)
    S=np.array(S)
    return S

def CreateS2(Orde):
    S=[]
    for i in range (0,Orde+1):
        a=[]
        for j in range (0,Orde+1):
            a.append(Int3(j,i))
        S.append(a)
    S=np.array(S)
    return S

def CreateB(Orde):
    #construeer B uit de gegevens bekend uit het verslag
    S=[]
    a=[]
    b=[]
    for i in range (0,Orde+1):
        a.append(-1.0)
        b.append(1.0)
    for j in range (0,Orde+1):
        if (j%2)==0:
            S.append(a)
        if (j%2)==1:
            S.append(b)
    S=np.array(S)
    return S

def CreateB2(Orde):
    S=[]
    a=[]
    b=[]
    for i in range (0,Orde+1):
        a.append(-1.0)
        b.append(1.0)
    for j in range (0,Orde+1):
        if (j%2)==0:
            S.append(a)
        if (j%2)==1:
            S.append(b)
    S=np.array(S)
    S=k(-1.0,0.0,2.0)*S
    return S

def CreateA(Orde):
    #construeer A uit de gegevens bekend uit het verslag
    S=[]
    for i in range (0,Orde+1):
        a=[]
        for j in range (0,Orde+1):
            a.append(Int4(j,i))
        S.append(a)
    S=np.array(S)
    return S

def CreateA2(Orde):
    S=[]
    for i in range (0,Orde+1):
        a=[]
        for j in range (0,Orde+1):
            a.append(Int4(j,i))
        S.append(a)
    S=np.array(S)
    S=S*k(1.0,0.0,2.0)
    return S
```

```
from math import *
import numpy as np
import matplotlib.pyplot as plt
import pylab
from Matrices import *

Stap = "discont"
#Stap = "cont"
#Stap = "mod"

def P(orde,x):
    #return het achtste orde Legendre-polynoom
    if (orde==8):
        return x**7 + 21*x**5*(x**2 - 1.0)/2 + 105*x**3*(x**2 - 1.0)**2/8 + 35*x*(x**2 - 1.0)**3/16
    if (orde==7):
        return x**6 + 15*x**4*(x**2 - 1.0)/2 + 45*x**2*(x**2 - 1.0)**2/8 + 5*(x**2 - 1.0)**3/16
    if (orde==6):
        return x**5 + 5*x**3*(x**2 - 1.0) + 15*x*(x**2 - 1.0)**2/8
    if (orde==5):
        return x**4 + 3.0*x**2*(x**2 - 1.0) + 3*(x**2 - 1)**2.0/8.0
    if (orde==4):
        return x**3 + 3*x*(x**2 - 1.0)/2.0
    if (orde==3):
        return 3*x**2/2 - 1.0/2.0
    if (orde==2):
        return x
    if (orde==1):
        return 1.0

def e(x):
    #andere notatie voor de e-macht
    return exp(x)

def g(x):
    #definieer de beginvoorwaarde
    if (Stap == "discont"):
        if (x<0.5):
            return 1
        else:
            return 0
    elif (Stap == "cont" or Stap == "mod"):
        return cos(2*pi*x)
        #of
        #return sin(pi*x)

k=2    #=orde
n=40   #=aantal elementen
dx = 1.0/n #=grootte element

def x(j):
    return (j-1.0/2.0)*dx

r_s = [-0.93246951, -0.66120939, -0.23861918, 0.23861918, 0.66120939, 0.93246951]
w_s = [0.17132449, 0.36076157, 0.46791393, 0.46791393, 0.36076157, 0.17132449]

u=(np.indices((k,n+1))[0]).tolist()
#genereer startvector u
for j in range (0,n+1): # over elements
    for m in range (0,k): #over polynomen
        u[m][(j-1)] = 0.0
        for i in range (0,6):
            u[m][(j-1)%n] = u[m][(j-1)%n] + (2.0*(m)+1.0)/2.0*g(x(j)+dx*r_s[i]/2)*P(m+1,r_s[i])*w_s[i]

#genereer matrices
M = dx*CreateM(k-1)
if (Stap == "mod"):
    S=CreateS2(k-1)
    A=CreateA2(k-1)
    B= CreateB2(k-1)
else:
    S=CreateS(k-1)
    A=CreateA(k-1)
    B= CreateB(k-1)

#twee clones van u
uuu=np.transpose(np.array(u))
uuu1=np.transpose(np.array(u))
dt = 1.0/10000.0
niter = 100
invM = np.linalg.inv(M) #inverse van M
eye=np.dot(M,invM) #eenheidsmatrix
invMS_A = np.dot(invM,S-A) #de inverse van M keer (S-A)

def f(uuu,uuu_1):
```

```
    #één integratiestap
    eenh = list(uuu_1)
    eenh = np.array(eenh)
    if (Stap == "cont"):
        for i in range (0, len(eenh[0])):
            eenh[0][i]=0.0
        eenh[0][0]=1.0
        eenh = np.transpose(eenh)
    k = np.transpose(np.array(np.dot(invMS_A,np.matrix.transpose(uuu))-np.dot(np.dot(invM,B),np.matrix.transpose(eenh))))
    return k

#Runge-kutta
for it in range (0,niter):
    v=(np.array(uuu)).tolist()
    for i in range (0,len(v)):
        v[len(v)-i-1]=v[len(v)-i-2]
    v[0]=v[len(v)-1]
    v[len(v)-1]=((np.array(uuu)).tolist())[len(v)-1]
    uuu_1=np.array(v)
    k1=f(uuu,uuu_1)
######################################################################
    v=(np.array(uuu+dt/2.0*k1)).tolist()
    for i in range (0,len(v)):
        v[len(v)-i-1]=v[len(v)-i-2]
    v[0]=v[len(v)-1]
    v[len(v)-1]=((np.array(uuu)).tolist())[len(v)-1]
    uuu_1=np.array(v)
    k2=f(uuu+dt/2.0*k1,uuu_1)
######################################################################
    v=(np.array(uuu+dt/2.0*k2)).tolist()
    for i in range (0,len(v)):
        v[len(v)-i-1]=v[len(v)-i-2]
    v[0]=v[len(v)-1]
    v[len(v)-1]=((np.array(uuu)).tolist())[len(v)-1]
    uuu_1=np.array(v)
    k3=f(uuu+dt/2.0*k2,uuu_1)
######################################################################
    v=(np.array(uuu+dt*k3)).tolist()
    for i in range (0,len(v)):
        v[len(v)-i-1]=v[len(v)-i-2]
    v[0]=v[len(v)-1]
    v[len(v)-1]=((np.array(uuu)).tolist())[len(v)-1]
    uuu_1=np.array(v)
    k4=f(uuu+dt*k3,uuu_1)
    uuu=uuu+dt/6.0*(k1+2.0*k2+2.0*k3+k4)
u=(np.matrix.transpose(np.array(uuu)))

###plotten:
if (k!=1):
    for j in range (0,n):
        xx = np.linspace(x(j)+dx/2.0, x(j)+3.0*dx/2.0, 1000)
        uu = 0.0
        for p in range (0,k):
            uu=0.0
            for b in range(0,k):
                uu= uu + u[b][j]*P(b+1,2.0/dx*(xx-dx-x(j)))
        pylab.axis([0,1, -0.2,1.2])
        pylab.plot(xx,uu)
    pylab.show()
if (k==1):
    xx = []
    yy = []
    for j in range (0,n):
        for i in range(0,100):
            xx.append(float(j)/float(n)+(float(i)/(n*100.0)))
            yy.append(u[0][j])
    plt.axis([0,1,-0.2,1.2])
    plt.plot(xx, yy)
    plt.show()
```