Centrum voor Wiskunde en Informatica

REPORT*RAPPORT*

MAS

Modelling, Analysis and Simulation

Modelling, Analysis and Simulation

Een discontinue Galerkin methode toegepast op een eendimensionaal diffusieprobleem

L. Voort

NOTE MAS-N0201 APRIL 30, 2002

CWI is the National Research Institute for Mathematics and Computer Science. It is sponsored by the Netherlands Organization for Scientific Research (NWO).
CWI is a founding member of ERCIM, the European Research Consortium for Informatics and Mathematics.

CWI's research has a theme-oriented structure and is grouped into four clusters. Listed below are the names of the clusters and in parentheses their acronyms.

Probability, Networks and Algorithms (PNA)

Software Engineering (SEN)

Modelling, Analysis and Simulation (MAS)

Information Systems (INS)

P.O. Box 94079, 1090 GB Amsterdam (NL)
Kruislaan 413, 1098 SJ Amsterdam (NL)
Telephone +31 20 592 9333
Telefax +31 20 592 4199

ISSN 1386-3703

Een Discontinue Galerkin Methode Toegepast op een Eendimensionaal Diffusieprobleem

Lykle Voort
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
(4 september 2000 – 30 juni 2001)

ABSTRACT

De Discontinue Galerkin-methode is een methode waarmee randwaardenproblemen opgelost kunnen worden. Deze methode is gebaseerd op Galerkin-methoden waarbij in de numerieke oplossing discontinuïteiten worden toegelaten. In dit rapport wordt met behulp van een Discontinue Galerkin-methode een oplossing gezocht voor een één-dimensionaal diffusieprobleem.

In de praktijk blijkt dat bij het oplossen van het probleem, de randvoorwaarden aan strenge voorwaarden gebonden zijn. Correcte wiskundige gesteldheid van het probleem blijkt niet voldoende om een oplossing te verkrijgen. Er wordt aangetoond dat de stabiliteit van de oplossing samenhangt met de keuze van de randvoorwaarden. Indien aan de gestelde voorwaarden wordt voldaan, dan is de methode stabiel, convergent en tweede orde nauwkeurig.

De implementatie van de Discontinue Galerkin-methode vereist meer voorbereiding dan een eindige-volumenmethode of eindige-elementenmethode van vergelijkbare orde. Dit kan een overweging zijn bij de keuze van een oplosmethode.

De één-dimensionale Discontinue Galerkin-methode is geïmplementeerd in de programmeertaal C++, waarbij gebruik is gemaakt van een object-georiënteerd ontwerp van het programma. Door deze aanpak wordt de implementatie voor verschillende problemen bruikbaar, zonder dat de methode voor elk probleem op maat gemaakt hoeft te worden.

2000 Mathematics Subject Classification: 65K05, 65N12, 65N15, 65N22, 65N30
Keywords and Phrases: elliptische randwaardenproblemen, Discontinue Galerkin methoden, goede gesteldheid en stabiliteit, object-georiënteerd programmeren.
Note: Dit rapport is gemaakt in het kader van 'Voorbereidend Afstuderen', en is onderdeel van mijn afstudeerwerk aan de TU-Delft bij de faculteit Luchtvaart- en Ruimtevaarttechniek, leerstoel Aerodynamica.

Het in dit rapport beschreven onderzoek is uitgevoerd binnen CWI-project MAS2.1 "Computational Fluid Dynamics". Het is een zijspoor van onderzoek dat gedaan wordt door prof.dr. P.W. Hemker, verbonden aan het Centrum voor Wiskunde en Informatica (CWI) te Amsterdam. Het onderzoek is begeleid door dr.ir. Barry Koren, verbonden aan zowel de TU-Delft als aan het CWI, die dit rapport zeer nauwgezet heeft gecontroleerd en gecorrigeerd.

1. Inleiding

Een algemene manier om randwaardenproblemen voor convectie-diffusievergelijkingen, zoals de Navier-Stokes-vergelijkingen, te discretiseren, is d.m.v. een Galerkin-methode. Dit rapport behandelt een specialisatie van deze methoden, namelijk Galerkin-methoden waarbij de gezochte oplossingen discontinu worden verondersteld, de z.g. Discontinue Galerkin-methoden (DG-methoden).

DG-methoden zijn bij uitstek geschikt voor aerodynamische problemen met discontinue oplossingen (oplossingen met schokgolven en contactdiscontinuïteiten), problemen beschreven door de Eulervergelijkingen van de gasdynamica. DG-methoden zijn een mengsel van eindige-volumen- en eindige-elementenmethoden. In DG-methoden wordt de stromingsoplossing in iedere roostercel geschreven als een polynoom. Vergelijkingen kunnen worden afgeleid voor de polynoomcoëfficiënten. In een hogere-orde nauwkeurige fluxberekening over een celwandje is geen interpolatie nodig om de stromingstoestanden op de celwandjes te bepalen. Linker en rechter celwandtoestandjes kunnen

direct worden bepaald uit de polynomen in de bijbehorende linker en rechter cel. Omdat hogereorde DG-methoden zo lokaal zijn, blijven hun goede nauwkeurigheidseigenschappen behouden op onregelmatige roosters. Dit is een belangrijk voordeel van DG-methoden t.o.v. standaard eindige volumen- en eindige elementenmethoden. DG-methoden toegepast op de Euler-vergelijkingen bieden als extra voordelen t.o.v. deze standaard methoden: snellere convergentie en grotere nauwkeurigheid bij een zelfde roosterfijnheid. Een nadeel van DG-methoden is dat zij meer onbekenden per cel vereisen dan standaard eindige-volumen- en eindige-elementenmethoden. (In DG-methoden moet men niet alleen vergelijkingen oplossen voor de stromingsgrootheden, maar ook voor hun afgeleiden.) Een goed overzicht over DG-methoden, inclusief een uitgebreide literatuurlijst, wordt gegeven in [3].

Tot nu toe zijn DG-methoden niet of nauweljks toegepast op de Navier-Stokes-vergelijkingen. Een moeilijkheid is nl. de DG-discretisatie van diffusietermen. Recent is echter een veelbelovende discretisatie van diffusietermen voorgesteld [2]. Onderzoek in deze richting is nog volop gaande. Dit rapport behandelt een DG-discretisatie van een diffusievergelijking.

Er is een groot aantal artikelen over Discontinue Galerkin-methoden beschikbaar, waarbij in de meeste gevallen de nadruk ligt op het bewijs dat deze methoden stabiel, convergent, enz. zijn in zeer algemene zin. Weinig aandacht wordt besteed aan de daadwerkelijke discretisatie en implementatie voor een concreet randwaardenprobleem.

Dit rapport behandelt de discretisatie en implementatie van een ééndimensionaal diffusieprobleem, waarbij de gezochte functie $u(x)$ benaderd wordt met stuksgewijs lineaire functies. Dit is natuurlijk een drastische vereenvoudiging van problemen zoals die in werkelijkheid zullen voorkomen. De bedoeling is dan ook om inzicht te geven in de complexiteit van het wiskundig fundament van deze methoden, en de daaruit voortkomende hoeveelheid voorbereiding die nodig is om dit probleem te implementeren.

2. Beschrijving van het randwaardenprobleem

Gegeven is het één-dimensionale diffusieprobleem

$$-\Delta u(x) = S, \quad x \in \Omega. \tag{2.1}$$

Er worden Dirichlet- en Neumann-randvoorwaarden opgelegd, respectievelijk:

$$u(x) = f, \qquad x \in \Gamma_{\mathcal{D}}, \quad \text{en} \tag{2.2a}$$

$$\frac{\partial u}{\partial n}(x) = g, \qquad x \in \Gamma_{\mathcal{N}}, \tag{2.2b}$$

waarbij $\Gamma_{\mathcal{D}} \cap \Gamma_{\mathcal{N}} = \emptyset$ en $\Gamma_{\mathcal{D}} \cup \Gamma_{\mathcal{N}} = \{x_0, x_N\}$. In woorden staat er dat er twee randvoorwaarden opgelegd worden, één in x_0 en één in x_N, waarvan één een Dirichlet-randvoorwaarde is.

In [1] wordt bewezen dat als een functie $u(x)$ aan het één-dimensionale diffusieprobleem voldoet, dat dan ook de volgende *zwakke formulering* geldt voor alle functies $v(x)$:

$$B(u,v) = l(v), \quad \forall v(x), \tag{2.3}$$

waarin

$$\begin{aligned} B(u,v) = &\sum_{\Omega_i \in \Omega} \int_{\Omega_i} \frac{dv}{dx}\frac{du}{dx} dx + \\ &+ \sum_{x_i \in \Gamma_{\mathcal{D}}} \left(\frac{dv}{dn} u - v \frac{du}{dn} \right)(x_i) + \\ &+ \sum_{x_i \in \Gamma_{\text{int}}} \left(\left\langle \frac{dv}{dn} \right\rangle [u] - [v] \left\langle \frac{du}{dn} \right\rangle \right)(x_i), \end{aligned} \tag{2.4a}$$

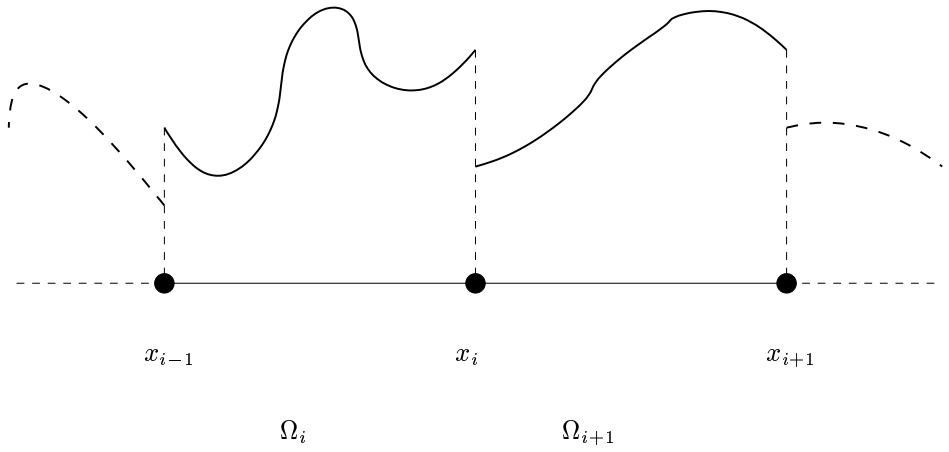
en

$$l(v) = \sum_{\Omega_i \in \Omega} \left(\int_{\Omega_i} vS dx \right) + \\ + \sum_{x_i \in \Gamma_{\mathcal{D}}} \left(\frac{dv}{dn} f \right)(x_i) + \sum_{x_i \in \Gamma_{\mathcal{N}}} (vg)(x_i), \tag{2.4b}$$

zoals afgeleid in [1] en [2]. Hierbij is de ruimte Ω gediscretiseerd door Ω op te delen in deelruimten Ω_i. De functie $u(x)$ is stuksgewijs continu, zoals geïllustreerd in figuur 1. Over de randen van de elementen Ω_i wordt een discontinuïteit toegestaan. Hierbij worden de sprongoperator $[\cdot]$ en de middelingsoperator $\langle \cdot \rangle$ geïntroduceerd, respectievelijk

$$[u](x_i) = \quad u(x_i)|_{\Omega_{i+1}} - u(x_i)|_{\Omega_i}, \tag{2.5a}$$
$$\langle u \rangle (x_i) = \quad \tfrac{1}{2} \left(u(x_i)|_{\Omega_{i+1}} + u(x_i)|_{\Omega_i} \right). \tag{2.5b}$$

Tot zover is de methode nog zeer algemeen: de benaderingen van $u(x)$ en van $v(x)$ zijn nog niet vastgelegd en zijn nog vrij te kiezen. In de volgende paragraaf worden deze gekozen.



Figuur 1: Discontinue definitie van een functie $u(x)$

Het is opvallend dat de eisen aan de randvoorwaarden, zoals deze in deze paragraaf naar voren zijn gekomen, strenger zijn dan nodig voor de gesteldheid van het probleem. Het probleem is goed gesteld, ook als $\Gamma_{\mathcal{D}}$ en $\Gamma_{\mathcal{N}}$ samenvallen. Dit blijkt in de implementatie echter niet mogelijk te zijn, wat nader verklaard wordt in paragraaf 6.

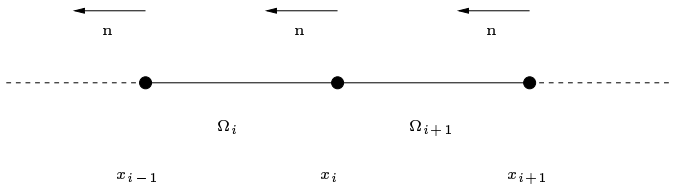
3. Ruimtediscretisatie en keuze testfuncties

De ruimte wordt opgedeeld in eindige volumina volgens figuur 2. Er zijn N elementen en $N+1$ knooppunten. Voor alle inwendige elementen wordt een vector n aangenomen die in de negatieve x-richting wijst; voor de knooppunten x_0 en x_N worden vectoren n aangenomen die naar buiten wijzen.

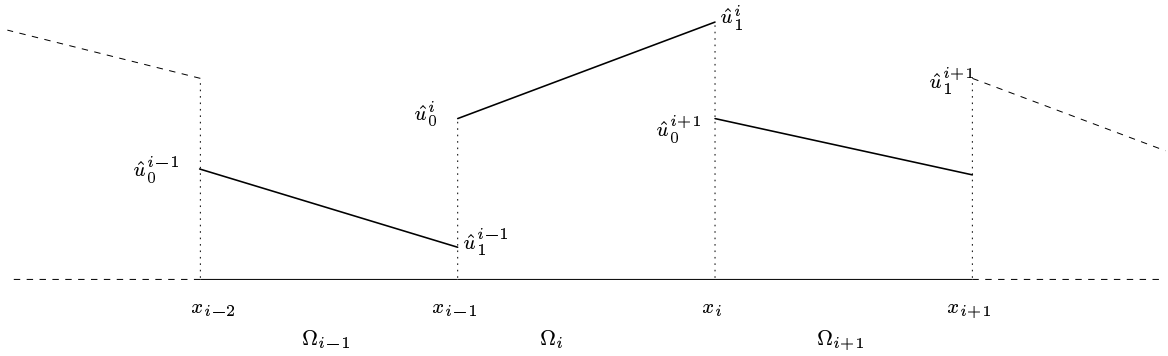
Per element Ω_i worden twee lineaire testfuncties aangenomen, $v_0(x)$ en $v_1(x)$ die de volgende eigenschappen hebben voor $x \in \Omega_i$:

$$v_0(x_{i-1}) = 1, \quad v_0(x_i) \quad = \quad 0, \tag{3.1a}$$
$$v_1(x_{i-1}) = 0, \quad v_1(x_i) \quad = \quad 1. \tag{3.1b}$$

Overal buiten Ω_i zijn $v_0(x)$ en $v_1(x)$ die bij dit element horen, gelijk aan 0.



Figuur 2: Discretisatie van het domein Ω



Figuur 3: Definitie van $u(x)$ op drie opeenvolgende elementen Ω_{i-1}, Ω_i en Ω_{i+1}

De gezochte functie $\tilde{u}(x)$ wordt benaderd met een stuksgewijs lineaire functie $u(x)$, zie figuur 3. Deze functie $u(x)$ wordt nu opgebouwd volgens

$$u(x) = \sum_{i=1}^{N} u_0^i(x) + u_1^i(x) = \sum_{i=1}^{N} \hat{u}_0^i v_0(x) + \hat{u}_1^i v_1(x) \tag{3.2}$$

Per knooppunt x_i zijn er dus twee verschillende waarden voor $u(x)$, namelijk $\hat{u}_1^i$ en $\hat{u}_0^{i+1}$, waardoor $u(x)$ dus discontinu gedefinieerd is. Er zijn dus per element twee onbekenden. In hoofdstuk 1 is reeds opgemerkt dat het aantal onbekenden per element groter is dan bij eindigevolumenmethoden. Dit is hier inderdaad het geval.

4. Uitwerking van de zwakke formulering

In hoofdstuk 2 is de zwakke formulering naar voren gekomen. In dit hoofdstuk wordt deze termsgewijs uitgewerkt. Dit is de vertaalslag van wiskundig model en discretisatie naar een model zoals dat door een computer uitgerekend kan worden.

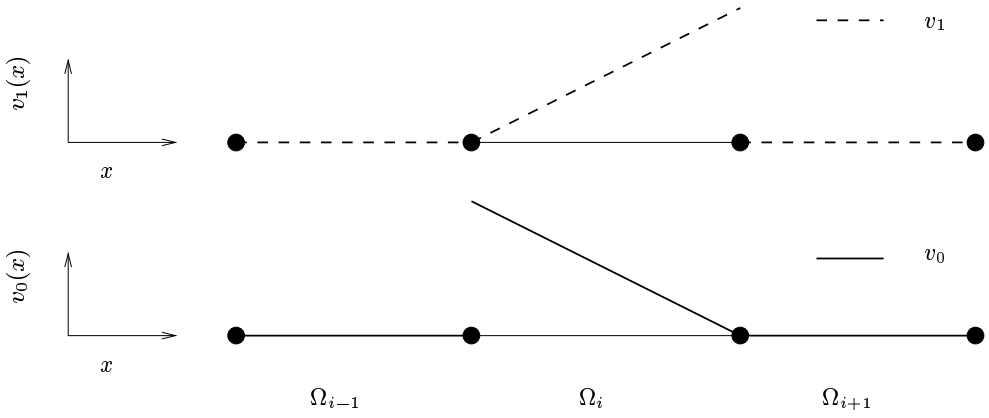
4.1 Samenvatting van de gegevens

In figuur 4 is een overzicht gegeven van alle relevante gegevens. Voor de eenvoud is aangenomen dat het rooster equidistant is. Deze gegevens worden tijdens de uitwerking van de diverse termen gebruikt.

Dit overzicht geeft een goede indruk hoeveel 'handwerk' nodig is om de methode uit te werken; voor hogere-orde testfuncties is deze hoeveelheid nog groter. Het kan dan lonen om dit handwerk te automatiseren.

4.2 Discretisatie van de bilineaire vorm

Het linkerlid van de bilineaire vergelijking (2.4a) heeft drie termen: een term die veel lijkt op een stijfheidsmatrix, een term die de flux over de inwendige wanden beschrijft, en een term die de invloed van een Dirichlet-rand weergeeft. Het linkerlid van de bilineaire vorm, $B(u, v)$, wordt hier termsgewijs uitgewerkt.



$u(x)$	$\hat{u}_0^{i-1}$ $\hat{u}_1^{i-1}$	$\hat{u}_0^{i}$ $\hat{u}_1^{i}$	$\hat{u}_0^{i+1}$ $\hat{u}_1^{i+1}$
$v_0(x)$	0 0	1 0	0 0
$v_1(x)$	0 0	0 1	0 0
$\frac{du}{dx}$	$\frac{\hat{u}_1^{i-1}-\hat{u}_0^{i-1}}{\Delta x}$	$\frac{\hat{u}_1^{i}-\hat{u}_0^{i}}{\Delta x}$	$\frac{\hat{u}_1^{i+1}-\hat{u}_0^{i+1}}{\Delta x}$
$\frac{dv_0}{dx}$	0	$-\frac{1}{\Delta x}$	0
$\frac{dv_1}{dx}$	0	$+\frac{1}{\Delta x}$	0

Figuur 4: Samenvatting van de relevante gegevens

4.2.1 De stijfheidsmatrix Met de gegevens uit figuur 4 volgt:

$$\int_{\Omega_i} \frac{dv_0}{dx}\frac{du}{dx}dx = \frac{1}{\Delta x}(+\hat{u}_0^i - \hat{u}_1^i), \tag{4.1a}$$

en

$$\int_{\Omega_i} \frac{dv_1}{dx}\frac{du}{dx}dx = \frac{1}{\Delta x}(-\hat{u}_0^i + \hat{u}_1^i), \tag{4.1b}$$

of geschreven in matrixvorm:

$$\int_{\Omega_i} \frac{dv}{dx}\frac{du}{dx}dx = \frac{1}{\Delta x}\begin{pmatrix} +1 & -1 \\ -1 & +1 \end{pmatrix}\begin{pmatrix} \hat{u}_0^i \\ \hat{u}_1^i \end{pmatrix}. \tag{4.2}$$

Dit heeft betrekking op alle elementen Ω_i. De termen die de numerieke flux verdisconteren hebben betrekking op óf inwendige knooppunten (derde term in (2.4a)), óf knooppunten op de rand (de tweede term in (2.4a)); in dit geval wil de rand zeggen: x_0 of x_N.

4.2.2 Numerieke flux over inwendige wanden De numerieke flux over de inwendige wanden wordt geleverd door de derde somterm uit de bilineaire vorm:

$$\sum_{x_i \in \Gamma_{\text{int}}} \left(\left\langle \frac{dv}{dn} \right\rangle [u] - [v] \left\langle \frac{du}{dn} \right\rangle \right)(x_i). \tag{4.3}$$

Deze term heeft betrekking op alle inwendige knooppunten; een element Ω_i heeft in het algemeen twee wanden gemeenschappelijk met de omliggende elementen, waarmee (4.3) reduceert tot

$$\sum_{x_j \in \{x_{i-1}, x_i\}} \left(\left\langle \frac{dv}{dn} \right\rangle [u] - [v] \left\langle \frac{du}{dn} \right\rangle \right)(x_j). \tag{4.4}$$

Voor Ω_1 ontbreekt de x_{i-1}-term, voor Ω_N ontbreekt de x_i-term.

Op inwendige wanden wijst de vector n in de negatieve x-richting. Omdat v_0 en v_1 alleen gedefinieerd zijn op Ω_i, zijn de richtingsafgeleiden van v_0 en v_1:

$$\frac{dv_0}{dn} = -\frac{dv_0}{dx} = \begin{cases} 0, & x \in \Omega_{i-1}, \\ +\frac{1}{\Delta x}, & x \in \Omega_i, \\ 0, & x \in \Omega_{i+1}, \end{cases} \tag{4.5a}$$

en

$$\frac{dv_1}{dn} = -\frac{dv_1}{dx} = \begin{cases} 0, & x \in \Omega_{i-1}, \\ -\frac{1}{\Delta x}, & x \in \Omega_i, \\ 0, & x \in \Omega_{i+1}. \end{cases} \tag{4.5b}$$

Er zijn nu twee gevallen te onderscheiden: een linker- en een rechterwand, die voor een element Ω_i horen bij respectievelijk de wanden x_{i-1} en x_i.

Voor de linker- en rechterwand volgt nu:

$$\left\langle \frac{dv_0}{dn} \right\rangle (x_{i-1}) = \left\langle \frac{dv_0}{dn} \right\rangle (x_i) = +\frac{1}{2\Delta x}, \tag{4.6a}$$

en

$$\left\langle \frac{dv_1}{dn} \right\rangle (x_{i-1}) = \left\langle \frac{dv_1}{dn} \right\rangle (x_i) = -\frac{1}{2\Delta x}. \tag{4.6b}$$

Voor du/dn wordt gevonden:

$$\frac{du}{dn} = -\frac{du}{dx} = \begin{cases} \frac{1}{\Delta x}(u_0^{i-1} - u_1^{i-1}), & x \in \Omega_{i-1}, \\ \frac{1}{\Delta x}(u_0^{i} - u_1^{i}), & x \in \Omega_{i}, \\ \frac{1}{\Delta x}(u_0^{i+1} - u_1^{i+1}), & x \in \Omega_{i+1}, \end{cases} \tag{4.7}$$

zodat voor de linker- en rechterwand resp. volgt:

$$\left\langle \frac{du}{dn} \right\rangle (x_{i-1}) = \frac{1}{2\Delta x}(\hat{u}_0^{i-1} - \hat{u}_1^{i-1} + \hat{u}_0^{i} - \hat{u}_1^{i}), \tag{4.8a}$$

en

$$\left\langle \frac{du}{dn} \right\rangle (x_{i}) = \frac{1}{2\Delta x}(\hat{u}_0^{i} - \hat{u}_1^{i} + \hat{u}_0^{i+1} - \hat{u}_1^{i+1}). \tag{4.8b}$$

Voor de sprongoperatoren $[u]$ en $[v]$ wordt gevonden:

$$\begin{array}{lll} [u](x_{i-1}) & = \hat{u}_0^{i} & -\hat{u}_1^{i-1}, \\ [u](x_{i}) & = \hat{u}_0^{i+1} & -\hat{u}_1^{i}, \end{array}$$

$$\begin{array}{llll} [v_0](x_{i-1}) = 1 & [v_0](x_i) = & 0, & \text{en} \\ [v_1](x_{i-1}) = 0 & [v_1](x_i) = & -1. & \end{array}$$

Met deze termen is het nu mogelijk de bijdrage van de numerieke flux over de wanden te vinden. De somterm (4.3) levert voor $x = x_{i-1}$ met de testfuncties v_0 respectievelijk v_1:

$$\begin{array}{lll} \left(\left\langle \frac{dv_0}{dn} \right\rangle [u] - [v_0] \left\langle \frac{du}{dn} \right\rangle\right)(x_{i-1}) & = & \frac{1}{2\Delta x}(-\hat{u}_0^{i-1} + \hat{u}_1^{i}), \\ \left(\left\langle \frac{dv_1}{dn} \right\rangle [u] - [v_1] \left\langle \frac{du}{dn} \right\rangle\right)(x_{i-1}) & = & \frac{1}{2\Delta x}(\hat{u}_1^{i-1} - \hat{u}_0^{i}). \end{array} \tag{4.9}$$

De uitdrukkingen voor (4.3) zijn ook in matrixvorm te schrijven:

$$\left(\left\langle \frac{d}{dn}\begin{pmatrix} v_0 \\ v_1 \end{pmatrix} \right\rangle [u] - \begin{bmatrix} v_0 \\ v_1 \end{bmatrix} \frac{du}{dn}\right)(x_{i-1}) = \frac{1}{\Delta x}\begin{pmatrix} -\frac{1}{2} & 0 & 0 & +\frac{1}{2} \\ 0 & +\frac{1}{2} & -\frac{1}{2} & 0 \end{pmatrix} \begin{pmatrix} \hat{u}_0^{i-1} \\ \hat{u}_1^{i-1} \\ \hat{u}_0^{i} \\ \hat{u}_1^{i} \end{pmatrix}. \tag{4.10}$$

Wordt (4.3) voor $x = x_i$ geëvalueerd, dan wordt gevonden:

$$\left(\left\langle \frac{d}{dn}\begin{pmatrix} v_0 \\ v_1 \end{pmatrix} \right\rangle [u] - \begin{bmatrix} v_0 \\ v_1 \end{bmatrix} \frac{du}{dn}\right)(x_{i}) = \frac{1}{\Delta x}\begin{pmatrix} 0 & -\frac{1}{2} & +\frac{1}{2} & 0 \\ +\frac{1}{2} & 0 & 0 & -\frac{1}{2} \end{pmatrix} \begin{pmatrix} \hat{u}_0^{i} \\ \hat{u}_1^{i} \\ \hat{u}_0^{i+1} \\ \hat{u}_1^{i+1} \end{pmatrix}. \tag{4.11}$$

4.2.3 Termen uit Dirichlet-randvoorwaarden De invloed van de rand van het domein komt tot uitdrukking in de tweede term uit (2.4a):

$$\sum_{x_i \in \Gamma_D} \left(\frac{dv}{dn}u - v\frac{du}{dn}\right)(x_i). \tag{4.12}$$

Er moeten nu twee gevallen worden onderscheiden: de linkerrand en de rechterrand. Gedefinieerd zijn $n = -1$ in x_0 en $n = +1$ in x_N. Voor de linker- en rechterrand volgt dan respectievelijk:

- t.p.v. x_0:

$$\frac{dv_0}{dn}(x_0) = +\frac{1}{\Delta x}, \quad \frac{dv_1}{dn}(x_0) = -\frac{1}{\Delta x};$$

$$\frac{du}{dn}(x_0) = \frac{1}{\Delta x}(\hat{u}_0^1 - \hat{u}_1^1);$$

$$u(x_0) = \hat{u}_0^1(x_0) = \hat{u}_0^1;$$

$$v_0(x_0) = 1, \quad v_1(x_0) = 0.$$

In bovenstaande uitdrukkingen is $\hat{u}_0^1 = u(x_0)$ een Dirichlet-randvoorwaarde.

- t.p.v. x_N:

$$\frac{dv_0}{dn}(x_N) = -\frac{1}{\Delta x}, \quad \frac{dv_1}{dn}(x_N) = +\frac{1}{\Delta x};$$

$$\frac{du}{dn}(x_N) = \frac{1}{\Delta x}(\hat{u}_1^N - \hat{u}_0^N);$$

$$u(x_N) = \hat{u}_1^N(x_N) = \hat{u}_1^N;$$

$$v_0(x_N) = 0, \quad v_1(x_N) = 1.$$

Wederom is in bovenstaande uitdrukkingen $\hat{u}_1^N = u(x_N)$ een Dirichlet-randvoorwaarde.

Uit deze uitdrukkingen volgen de vergelijkingen die samenhangen met de randen. De uitdrukkingen voor element Ω_1 zijn:

- $x = x_0$, $v = v_0$: voor (4.12) wordt nu gevonden:

$$\left(\frac{dv_0}{dn}u - v_0\frac{du}{dn}\right)(x_0) = +\frac{1}{\Delta x}\hat{u}_1^1. \tag{4.13}$$

- $x = x_0$, $v = v_1$: (4.12) levert nu:

$$\left(\frac{dv_1}{dn}u - v_1\frac{du}{dn}\right)(x_0) = -\frac{1}{\Delta x}\hat{u}_0^1. \tag{4.14}$$

Nu kan (4.12) ook in matrixvorm geschreven worden:

$$\left(\frac{d}{dn}\begin{pmatrix} v_0 \\ v_1 \end{pmatrix}u - \begin{pmatrix} v_0 \\ v_1 \end{pmatrix}\frac{du}{dn}\right)(x_0) = \frac{1}{\Delta x}\begin{pmatrix} 0 & +1 \\ -1 & 0 \end{pmatrix}\begin{pmatrix} \hat{u}_0^1 \\ \hat{u}_1^1 \end{pmatrix}. \tag{4.15}$$

Voor element Ω_N volgt:

$$\left(\frac{d}{dn}\begin{pmatrix} v_0 \\ v_1 \end{pmatrix}u - \begin{pmatrix} v_0 \\ v_1 \end{pmatrix}\frac{du}{dn}\right)(x_N) = \frac{1}{\Delta x}\begin{pmatrix} 0 & -1 \\ +1 & 0 \end{pmatrix}\begin{pmatrix} \hat{u}_0^N \\ \hat{u}_1^N \end{pmatrix}. \tag{4.16}$$

Hiermee zijn alle termen uit (2.4a) gediscretiseerd.

4.3 Uitwerking van het rechterlid van de bilineaire vorm
Het rechterlid luidt nogmaals

$$\begin{aligned} l(v) = &\sum_{\Omega_i \in \Omega}\left(\int_{\Omega_i} vS dx\right) + \\ &+ \sum_{x_i \in \Gamma_D}\left(\frac{dv}{dn}f\right)(x_i) + \sum_{x_i \in \Gamma_N}(vg)(x_i). \end{aligned} \tag{4.17}$$

De termen zijn te identificeren als respectievelijk een elementsvector, een term die samenhangt met een Dirichlet-randvoorwaarde en een term die samenhangt met een Neumann-randvoorwaarde. De drie termen zullen hier afzonderlijk worden uitgewerkt.

4.3.1 Uitwerking van de elementsvector In veel gevallen zal de integraal in (4.17) niet analytisch uit te rekenen zijn. In plaats daarvan wordt de integraal met behulp van een kwadratuurformule uitgerekend, zie hiervoor appendix A.

Met de keuze van de eerder beschreven lineaire testfuncties is met behulp van kwadratuurformules de elementsvector dan in matrixvorm te schrijven:

$$\int_{\Omega_i} \begin{pmatrix} v_0 \\ v_1 \end{pmatrix} S \, dx = \frac{\Delta x}{6} \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} S(x_{i-1}) \\ S(x_i) \end{pmatrix}. \tag{4.18}$$

4.3.2 Rechterlid, Dirichlet-randvoorwaarde De tweede term uit (4.17) wordt toegepast voor de rand waarop een Dirichlet-randvoorwaarde wordt opgelegd; dit is of op $x = x_0$ of op $x = x_N$. Voor x_0 resp. x_N geldt:

$$\frac{dv_0}{dn}(x_0) = +\frac{1}{\Delta x}, \quad \frac{dv_1}{dn}(x_0) = -\frac{1}{\Delta x},$$

en

$$\frac{dv_0}{dn}(x_N) = -\frac{1}{\Delta x}, \quad \frac{dv_1}{dn}(x_N) = +\frac{1}{\Delta x}.$$

Een randvoorwaarde volgens $u(x_0) = f(x_0)$ levert de volgende term in het rechterlid:

$$\left(\frac{d}{dn} \begin{pmatrix} v_0 \\ v_1 \end{pmatrix} f \right)(x_0) = \frac{f(x_0)}{\Delta x} \begin{pmatrix} +1 \\ -1 \end{pmatrix}. \tag{4.19a}$$

Een randvoorwaarde volgens $u(x_N) = f(x_N)$ levert

$$\left(\frac{d}{dn} \begin{pmatrix} v_0 \\ v_1 \end{pmatrix} f \right)(x_N) = \frac{f(x_N)}{\Delta x} \begin{pmatrix} -1 \\ +1 \end{pmatrix}. \tag{4.19b}$$

Uitdrukking (4.19a) hoort bij de testfuncties v gedefinieerd op Ω_1, uitdrukking (4.19b) hoort bij Ω_N.

4.3.3 Rechterlid, Neumann-randvoorwaarde De laatste term uit (4.17) is van toepassing op de rand waarop de Neumann-randvoorwaarde wordt opgelegd. Met

$$v_0(x_0) = 1, \quad v_1(x_0) = 0, \quad v_0(x_N) = 0, \quad v_1(x_N) = 1 \tag{4.20}$$

volgen direct de rechterleden voor het geval $u'(x_0) = g$ respectievelijk $u'(x_N) = g$ wordt opgelegd:

$$\left(\begin{pmatrix} v_0 \\ v_1 \end{pmatrix} g \right)(x_0) = g(x_0) \begin{pmatrix} -1 \\ 0 \end{pmatrix} \tag{4.21}$$

voor testfuncties $v(x) \in \Omega_1$, en

$$\left(\begin{pmatrix} v_0 \\ v_1 \end{pmatrix} g \right)(x_1) = g(x_1) \begin{pmatrix} 0 \\ +1 \end{pmatrix} \tag{4.22}$$

voor testfuncties $v(x) \in \Omega_N$.

5. Implementatie

In deze paragraaf worden de stappen behandeld die bij de implementatie zijn gemaakt.

5.1 Werkwijze
Bij de implementatie van de methode is de uitwerking zoals in de vorige paragrafen beschreven, precies gevolgd. De hoofdlijn is als volgt:

1. Bereken de deelmatrices die bij de term $B(u, v)$ horen;
2. bereken de termen die bij het rechterlid horen;
3. voeg de deelmatrices samen tot het stelsel vergelijkingen;
4. los het stelsel vergelijkingen op.

Het stelsel vergelijkingen wordt direct opgelost. Een alternatieve oplossingsmethode die onderzocht kan worden, is Gauss-Seidel relaxatie.

5.2 Numerieke aspecten
In paragraaf 6 wordt gesteld dat met name in het geval van een slechte conditie van de matrix die het stelsel vergelijkingen weergeeft, een nauwkeurige oplosmethode nodig is. Om deze reden wordt in het hele programma gebruik gemaakt van dubbele precisie decimale getallen ('floating point'), het type '`double`', zie hiervoor ook het programma zelf in appendix E.

Naast numerieke precisie speelt de conditie een zeer belangrijke rol bij het uiteindelijke oplossen van het stelsel vergelijkingen. Hiervoor zijn twee procedures geschreven: één geoptimaliseerd voor bandmatrices, één met partiële pivotering. Dit gaat niet samen; voor slecht geconditioneerde problemen wordt dus gekozen voor de methode met pivotering, ten koste van rekentijd. Beide procedures heten '`linSolve`', zie hiervoor ook E.3.

5.3 Keuze van de programmeertaal
Bij de implementatie is gekozen voor de programmeertaal C++; ten eerste omdat reeds bestaande Discontinue-Galerkin software in C++ is geschreven; ten tweede vanwege de erg goede uitbreidbaarheid van bestaande programmatuur. De object-georiënteerde mogelijkheden maken het zeer goed mogelijk te concentreren op de essentie van de methode.

6. Beperkingen in de randvoorwaarden

In paragraaf 2 is reeds gesteld dat de keuze van randvoorwaarden aan meer beperkingen gebonden is dan zoals te verwachten uit de wiskunde alleen. De reden hiervoor blijkt in de implementatie te liggen, wat in deze paragraaf uitgewerkt wordt.

Daarnaast blijken er in de praktijk meer beperkingen te zijn die samenhangen met de discretisatie en de keuze van de testfuncties. In deze paragraaf wordt op eenvoudige wijze aangetoond dat in het geval dat er twee Dirichlet-randvoorwaarden worden opgelegd, er geen oplossing gevonden wordt.

6.1 Onmogelijkheid van twee randvoorwaarden op één rand
Het randwaardenprobleem voor $u_{xx} = -S(x)$ is in wiskundige zin goed gesteld als er twee randvoorwaarden worden opgelegd, waarvan minstens één een Dirichlet-randvoorwaarde is; het doet er niet toe wáár de randvoorwaarden worden opgelegd; het probleem is ook goed gesteld als zowel u als u_x op één rand x_0 worden opgelegd. De enige beperking is dat er geen twee Neumann-voorwaarden mogen worden opgelegd.

In de praktijk echter blijkt het niet mogelijk te zijn om op één rand twee randvoorwaarden op te leggen. De reden hiervoor ligt in vergelijking (2.4a): de tweede term waarin de keuze van de randvoorwaarden tot uitdrukking komt laat niet toe dat een Dirichlet-rand en een Neumann-rand samenvallen.

Op het implementatieniveau komt dit terug wanneer het stelsel vergelijkingen wordt opgesteld: óf er worden wel extra vergelijkingen toegevoegd, in het geval van een Dirichlet-randvoorwaarde, óf de vergelijkingen ontbreken, in het geval van een Neumann-randvoorwaarde.

In de code is dit ook te zien als een keuze waarin beslist wordt of een deelmatrix al dan niet opgeteld wordt bij de matrix die het stelsel vergelijkingen weergeeft.

6.2 Gesteldheid bij het opleggen van twee Dirichlet-randvoorwaarden

Twee Dirichlet-randvoorwaarden mogen niet worden opgelegd bij deze keuze van de ruimtediscretisatie en testfuncties. In deze paragraaf wordt volstaan met het aantonen dat het mogelijk is een correcte oplossing te verstoren, terwijl de verstoorde oplossing nog steeds voldoet aan de gediscretiseerde zwakke formulering, inclusief randvoorwaarden.

Er wordt weer uitgegaan van het probleem

$$\frac{\partial^2 u}{\partial x^2} = -S, \tag{6.1}$$

waarvan de gediscretiseerde oplossing $\mathbf{u}$ voldoet aan de zwakke formulering:

$$B(\mathbf{u}, v) = l(v), \quad \forall v, \tag{6.2}$$

waarbij de testfuncties stuksgewijs lineair zijn.

Om aan te tonen dat de gezochte oplossing $\mathbf{u}$ niet uniek is, wordt een verstoring $\mathbf{u}^*$ gezocht, zó, dat

$$B(\mathbf{u}^*, v) = 0, \quad \forall v. \tag{6.3}$$

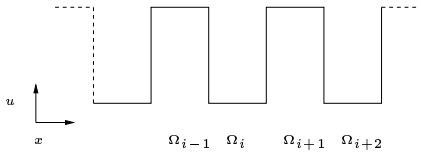
Omdat de bilineaire vorm lineair is, volgt uit optelling van (6.2) en (6.3) dat

$$B(\mathbf{u} + \mathbf{u}^*, v) = l(v), \quad \forall v, \tag{6.4}$$

ofwel de verstoorde oplossing voldoet ook aan (6.2). Indien er een dergelijke verstoring bestaat, is aangetoond dat het probleem niet meer goed gesteld is.

Er kan een verstoring geconstrueerd worden, die aan (6.3) voldoet. Het blijkt dat een conditie voor het bestaan van een niet-triviale verstoring is dat er twee Dirichlet-randvoorwaarden worden opgelegd.

Een niet-triviale verstoring die aan (6.3) voldoet, is een blokgolvige functie zoals geschetst in figuur 5.



Figuur 5: Blokgolvige verstoring van een oplossing

Direct is in te zien dat

$$[u]\,(x_i) = -\,[u]\,(x_{i+1}), \tag{6.5}$$

met als gevolg

$$[u^\star](x_1) = +[u^\star](x_{N-1}), \quad \text{als N even}, \tag{6.6a}$$

en

$$[u^\star](x_1) = -[u^\star](x_{N-1}), \quad \text{als N oneven}. \tag{6.6b}$$

In appendix B wordt aangetoond dat met stuksgewijs lineaire testfuncties (6.2) te vereenvoudigen is, waarbij drie gevallen worden onderscheiden: één Dirichlet-randvoorwaarde in x_0, één Dirichlet-randvoorwaarde in x_N, en het geval twee Dirichlet-randvoorwaarden, in respectievelijk x_0 en x_N. Hier volgt een bespreking van de drie gevallen, waaruit zal blijken dat in het geval er twee Dirichlet-randvoorwaarden worden opgelegd, er een niet-triviale oplossing mogelijk is.

Een Dirichlet-randvoorwaarde in x_0 In dit geval reduceert (6.3) tot

$$\Gamma_{\mathcal{D}} = \{x_0\}: \quad \begin{cases} \hat{u}_0^1 + [u](x_1) &= 0, \\ [u](x_{N-1}) &= 0. \end{cases} \tag{6.7}$$

Met (6.6a) of (6.6b) (afhankelijk van N even of oneven) blijft er één onafhankelijke onbekende over, bijvoorbeeld $[u](x_1)$. Met dit stelsel volgt dat $[u](x_1) = 0$. De gezochte stoorfunctie is dan overal nul.

Een Dirichlet-randvoorwaarde in x_N Het stelsel waartoe (6.3) reduceert, lijkt op de vorige situatie:

$$\Gamma_{\mathcal{D}} = \{x_N\}: \quad \begin{cases} [u](x_1) &= 0, \\ \hat{u}_N^1 + [u](x_{N-1}) &= 0. \end{cases} \tag{6.8}$$

Op analoge wijze volgt dat de stoorfunctie overal nul moet zijn.

Twee Dirichlet-randvoorwaarden Een andere situatie onstaat wanneer er twee Dirichlet-randvoorwaarden worden opgelegd; het stelsel dat in het geval van twee Dirichlet-randvoorwaarden verkregen wordt, luidt

$$\begin{cases} \hat{u}_1^0 + \frac{1}{2}[u](x_1) &= 0, \\ \frac{1}{2}[u](x_{N-1}) + \hat{u}_N^1 &= 0. \end{cases} \tag{6.9}$$

Met (6.6a) of (6.6b) volgt nu dat er één nog vrij te kiezen parameter is: de amplitude van de blokvorm, uitgedrukt in bijvoorbeeld $[u](x_1)$. De oplossing van het stelsel luidt dan:

$$\hat{u}_1^0 = -\frac{1}{2}[u](x_1), \quad \text{en} \tag{6.10}$$

$$\hat{u}_N^1 = -\frac{1}{2}[u](x_{N-1}). \tag{6.11}$$

Als de stoorfunctie aan deze twee voorwaarden voldoet, dan is de stoorfunctie een oplossing van (6.3), en is de verstoorde oplossing dus ook een oplossing. Alhoewel het probleem goed gesteld is, zal de methode niet in staat zijn een correcte oplossing te vinden.

Dit verschijnsel hangt volledig samen met de discretisatie en de keuze van de testfuncties, en is niet afhankelijk van de vorm van het rekenrooster.

7. Resultaten

Met de keuze van de ruimtediscretisatie en de testfuncties is het mogelijk de code te testen. Dit wordt gedaan in een aantal gevallen.

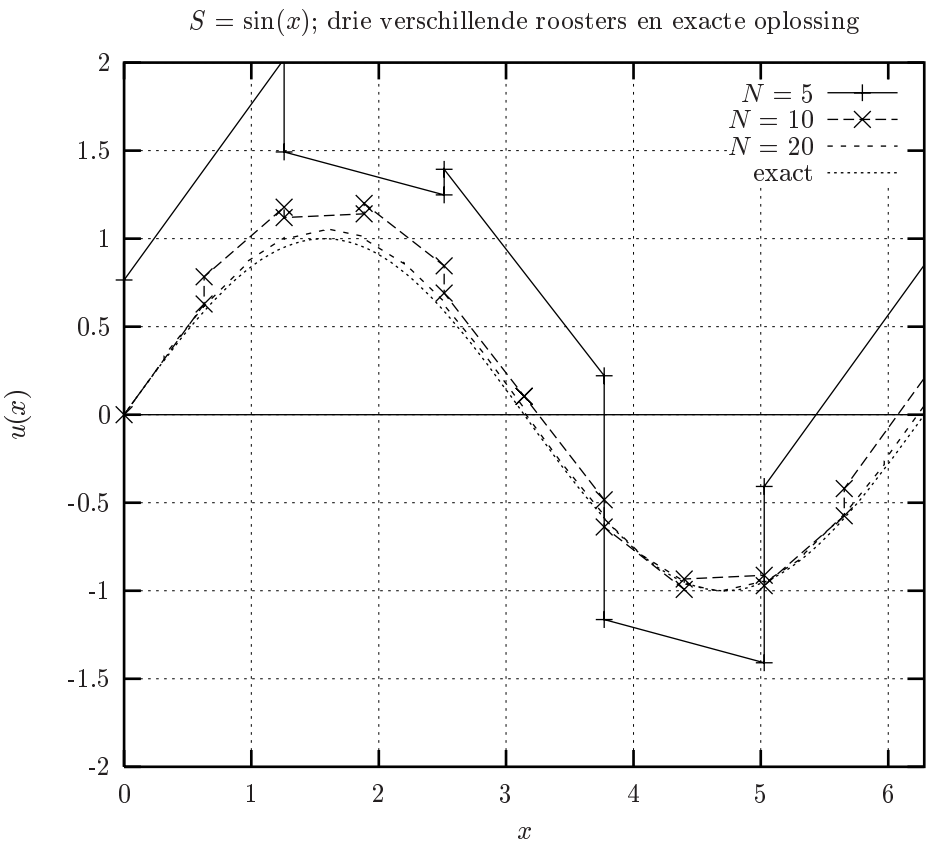
7.1 Sinusvormige bronterm

In de eerste twee testen wordt het geval onderzocht waarin $S(x) = \sin(x)$, $0 \leq x \leq 2\pi$. De randvoorwaarden zijn $u(0) = 0$, $u'(2\pi) = 1$. De exacte oplossing van dit probleem is $u(x) = \sin(x)$.

7.1.1 Plots voor drie discretisaties Allereerst is de code toegepast met $N = 5$, $N = 10$ en $N = 20$. In figuur 6 zijn de gevonden oplossingen gezet, tezamen met de exacte oplossing.

Uit de figuur blijkt duidelijk dat de gevonden oplossing discontinu is, en dat bovendien de gevonden oplossingen op de randen beter aansluiten naarmate het rooster fijner wordt, zie het geval $N = 20$. Dit is natuurlijk geheel volgens de verwachting. Op het oog lijkt het een tweede orde methode.

Overigens is in de grafiek te zien dat voor $N = 5$ de gevonden oplossing niet voldoet aan de gestelde randvoorwaarde $u(0) = 0$, terwijl voor $N = 10$ en $N = 20$ de oplossing wel aan deze randvoorwaarde voldoet. Hiervoor is geen sluitende verklaring; wel moet opgemerkt worden dat deze Discontinue Galerkin-methode de oplossing aan de randen niet vastlegt.



Figuur 6: Plots bij drie roosterfijnheden

7.1.2 Fout t.o.v. exacte oplossing bij roosterverfijning Een volgende test bestaat uit het berekenen van de (L_2-)norm van het verschil tussen de gevonden oplossing en de exacte oplossing:

$$\epsilon = \left[\int_0^{2\pi} \left(u(x) - u(x)_{\text{exact}} \right)^2 \quad dx \right]^{\frac{1}{2}}. \tag{7.1}$$

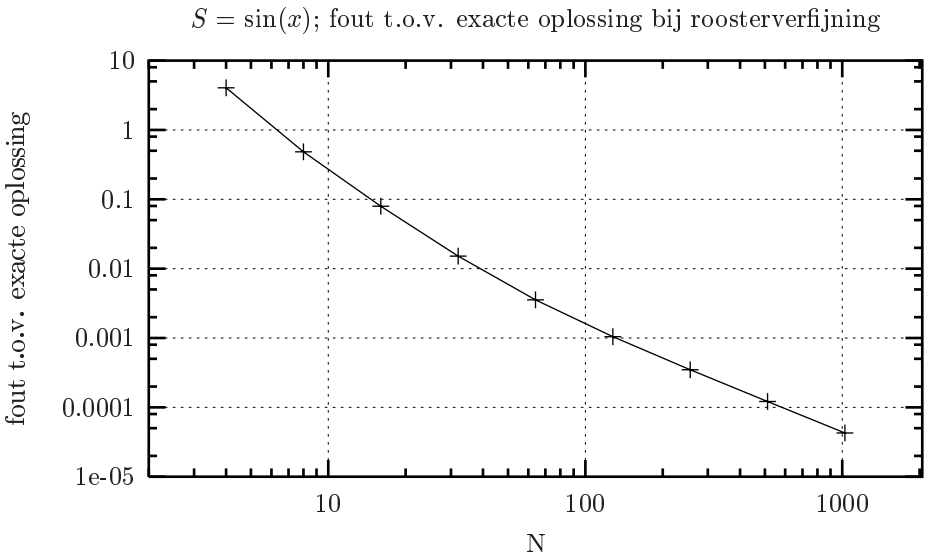
Met dezelfde bronfunctie en randvoorwaarden als in de vorige test, is voor een aantal roosterverfijningen de norm van de fout uitgezet tegen het aantal volumina, zie figuur 7. De schaal is dubbellogaritmisch uitgezet.

7.2 Roosterfijnheid bij constante fout: frequentieanalyse

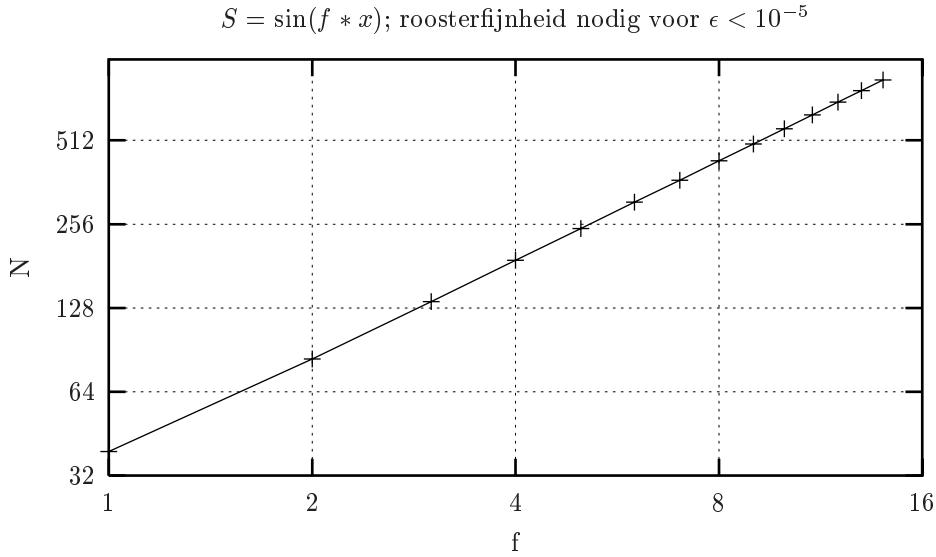
In de laatste test wordt weer een sinusvormige bronterm aangenomen, dit maal met een frequentie f, dat wil zeggen $S(x) = \sin(fx)$, met f een geheel getal. De reden hiervoor is dat zowel het probleem als de gevonden oplossingen lineair zijn. Met behulp van een Fourier-analyse kan een bronterm worden geschreven als een som van sinustermen, waarna elke sinusterm een oplossing heeft. Opgeteld vormen deze weer de oplossing van het totaalprobleem. Voor het gemak zijn cosinustermen weggelaten, aangezien de aandacht uitgaat naar frequenties, niet naar fasen.

Zoals gesteld is $S(x) = \sin(fx)$; de randvoorwaarden zijn $u(0) = 0$ en $u'(2\pi) = 1/f$; de exacte oplossing is $u(x) = \sin(fx)/f^2$. In deze test wordt de frequentie stapsgewijs vergroot, waarbij het aantal volumina N gezocht wordt zodanig, dat de genormeerde fout ϵf^2 kleiner is dan 10^{-2}.

In figuur 8 is voor frequenties $1 \leq f \leq 15$ dit aantal discretisatiestappen uitgezet. Duidelijk te zien is dat als de frequentie verdubbelt, dat dan ook het aantal benodigde stappen iets meer dan verdubbelt.



Figuur 7: Fout ten opzichte van exacte oplossing; dubbellogaritmische schaal



Figuur 8: Aantal stappen dat voor een bepaalde nauwkeurigheid vereist is

7.3 Probleem met twee Dirichlet-randvoorwaarden

Zoals besproken in paragraaf 6.2, kan de methode verkeerde resultaten opleveren als er twee Dirichlet-randvoorwaarden worden opgelegd. Om dit duidelijk te maken, wordt de volgende situatie beschouwd:

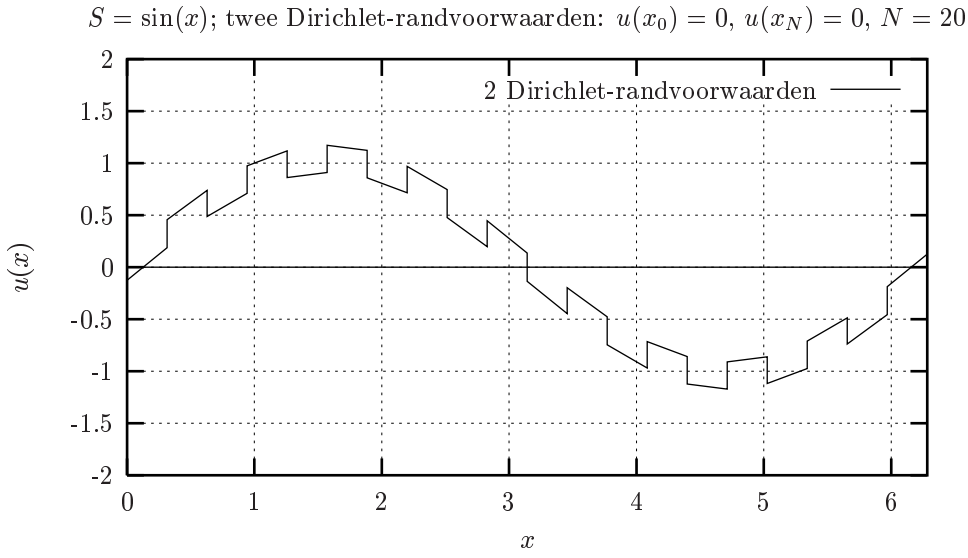
$$u_{xx} = -\sin(x), \tag{7.2a}$$

met randvoorwaarden

$$u(x_0) = 0, \quad u(x_N) = 0. \tag{7.2b}$$

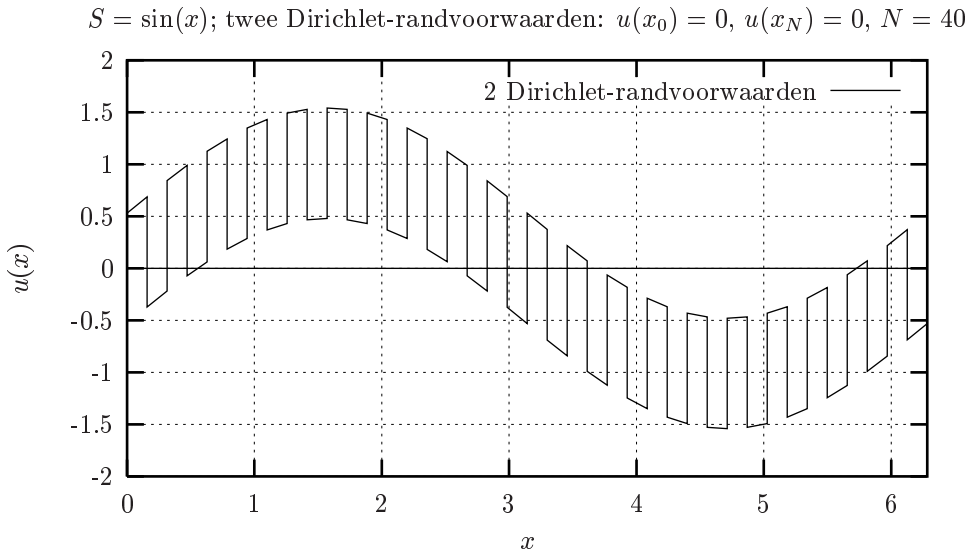
De exacte oplossing van dit probleem is weer $u(x) = \sin(x)$, dezelfde oplossing als in het geval er één Dirichlet-randvoorwaarde en één Neumann-randvoorwaarde worden opgelegd. Deze oplossing

wordt echter niet gevonden: zie figuur 9, waarin de gevonden oplossing voor $N = 20$ gegeven is.



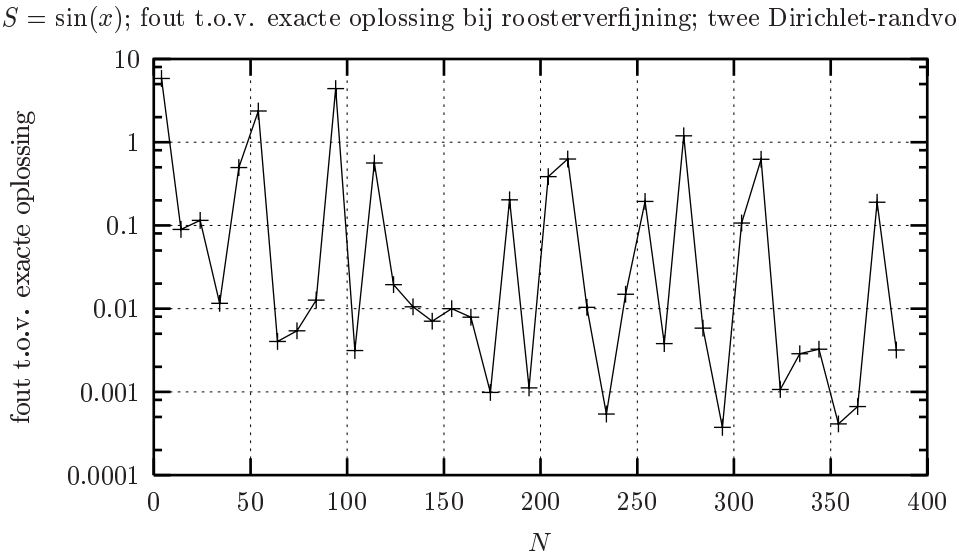
Figuur 9: Verstoorde oplossing bij twee Dirichlet-randvoorwaarden

De nauwkeurigheid van de oplossing blijkt in dit geval niet af te hangen van de roosterfijnheid: met verdere roosterverfijning wordt het resultaat in dit geval slechter: figuur 10 toont het geval $N = 40$.



Figuur 10: Verslechtering van het resultaat door roosterverfijning

Een plot van hoe de fout afhangt van de roosterfijnheid, is gegeven in figuur 11. Direct duidelijk is dat er geen sprake is van een trend: met het verfijnen van het rooster neemt de fout in sommige gevallen af, in andere gevallen neemt de fout toe.



Figuur 11: Norm van de fout bij roosterverfijning als er twee Dirichlet-randvoorwaarden worden opgelegd

8. Conclusies

De ervaringen bij het in de praktijk toepassen van de Discontinue Galerkin-methode op een eenvoudig randwaardenprobleem worden in hoofdstuk samengevat. Vanuit wiskundig perspectief zijn de nauwkeurigheid en de stabiliteit van belang; aangezien de implementatie gecompliceerd is, wordt ook de praktische kant belicht.

8.1 Nauwkeurigheid en stabiliteit

Uit de resultaten in sectie 7 blijkt dat deze methode met eerste-orde testfuncties tweede-orde nauwkeurig is. Binnen de gestelde grenzen is deze methode dus bruikbaar.

Gebleken is echter dat de keuze van de randvoorwaarden aan beperkingen gebonden is. Daarom moet niet alleen een nauwkeurige oplosmethode worden gebruikt, ook moet altijd de conditie van het stelsel vergelijkingen worden onderzocht.

8.2 Implementatie

Een groot deel van de implementatie bestaat uit de discretisatie, en de discretisatie vergt dan ook een grotere inspanning dan bij bijvoorbeeld een eindige-volumenmethode. De testfuncties zijn in dit rapport lineair, in de praktijk zullen dit hogere-orde testfuncties zijn.

Dit rapport is een demonstratie van de hoeveelheid voorbereiding die nodig is om de methode te implementeren. Bij het gebruik van hogere-orde methoden worden de voorbereidingen zo snel ingewikkelder, dat dit slechts geautomatiseerd kan gebeuren.

Referenties

1. Daniël Benden. *The Discontinuous Galerkin Method in the One Dimensional Case*, MSc-thesis, interne CWI-note.
2. Carlos Erik Baumann, *An HP-Adaptive Discontinuous Finite Element Method for Computational Fluid Dynamics*, PhD-thesis, The University of Texas at Austin, December 1997.
3. B. Cockburn, *Discontinuous Galerkin Methods for Convection Dominated Problems*, http://www.math.umn.edu/~ cockburn/LectureNotes.html.

A. Kwadratuurregels

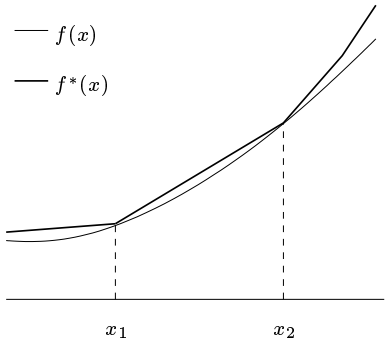
In de zwakke formulering wordt de bronterm $S(x)$ vermenigvuldigd met een testfunctie $v(x)$, waarna geïntegreerd wordt over Ω_i.

Slechts in eenvoudige gevallen zal het mogelijk zijn deze term exact te integreren. Als er hogere-orde methoden worden gebruikt, wordt het steeds moeilijker een exacte uitdrukking te vinden, en kost het exact uitrekenen steeds meer bewerkingen, terwijl uiteindelijk toch slechts een nauwkeurigheid van dezelfde orde als de overige termen nodig is. In veel gevallen is er echter alleen een functiewaarde beschikbaar, bijvoorbeeld vanuit andere programma's of uit tijdstappen. Het is dan heel goed mogelijk met kwadratuurmethoden (2.4b) te benaderen.

Er zijn diverse kwadratuurmethoden beschikbaar; in veel Discontinue Galerkin-codes wordt de methode van Lobatto gebruikt.

Het programma waarmee de resultaten in dit artikel verkregen zijn, gebruikt een eigen kwadratuurmethode die in dit hoofdstuk wordt afgeleid en toegelicht ter illustratie.

A.1 Een eenvoudige kwadratuurmethode voor lineaire testfuncties



Figuur 12: Benadering van een gladde functie met lineaire functies door de steunpunten

Er wordt uitgegaan van dezelfde twee testfuncties $v_0(x)$ en $v_1(x)$ die gedefinieerd zijn op een element Ω_i, waarbij Ω_i het interval (x_1, x_2) op de reële as is. De functies zijn gegeven door:

$$v_0(x) = (x_2 - x)/\Delta x, \text{ en} \tag{A.1}$$
$$v_1(x) = (x - x_1)/\Delta x. \tag{A.2}$$

Om nu voor een willekeurige functie $f(x)$ de integralen

$$\int_{x_1}^{x_2} v_0(x)\ f(x)\ dx, \quad \text{en} \int_{x_1}^{x_2} v_1(x)\ f(x)\ dx \tag{A.3}$$

uit te rekenen, wordt de functie $f(x)$ gelineariseerd gedacht, zie ook figuur 12,

$$f^*(x) = f(x_1) + \frac{f(x_2) - f(x_1)}{\Delta x}(x - x_1).$$

Wordt nu in (A.3) $f(x)$ vervangen door $f^*(x)$, dan geven de integralen een benadering voor redelijk gladde functies:

$$\int_{x_1}^{x_2} v_0(x)\ f^*(x)\ dx = \frac{\Delta x}{6}\left(2f(x_1) + f(x_2)\right),$$

$$\int_{x_1}^{x_2} v_1(x)\ f^*(x)\ dx = \frac{\Delta x}{6}\left(f(x_1) + 2f(x_2)\right).$$

Deze uitdrukkingen zijn zeer snel uit te rekenen en worden in de code in dit artikel gebruikt.

B. Uitwerking van de bilineaire term voor een blokvormige stoorfunctie

In sectie 6.2 wordt gezocht naar een niet-triviale stoorfunctie $u^*(x)$ zó, dat

$$B(u^\star, v) = 0. \tag{B.1}$$

Daartoe werd voor de stoorfunctie een blokgolvige functie aangenomen. Deze stoorfunctie wordt gekenmerkt door:

$$\frac{du^\star}{dx} = \frac{du^\star}{dn} = 0, \quad \forall \Omega_i \tag{B.2}$$

en daarnaast dat

$$[u^\star](x_i) = -[u^\star](x_{i+1}), \quad \forall x_i. \tag{B.3}$$

Een gevolg hiervan is dat

$$[u^\star](x_1) = +[u^\star](x_{N-1}), \quad \text{als N even,} \tag{B.4a}$$

en

$$[u^\star](x_1) = -[u^\star](x_{N-1}), \quad \text{als N oneven.} \tag{B.4b}$$

B.1 De stijfheidsmatrix

Wegens (B.2) is direct in te zien dat de bijdrage van de stijfheidsmatrix vervalt voor alle Ω_i:

$$\int_{\Omega_i} \frac{dv}{dx}\frac{du}{dx} dx = \begin{pmatrix} 0 \\ 0 \end{pmatrix}. \tag{B.5}$$

B.2 Inwendige elementen

Onder inwendige elementen wordt verstaan: elementen waarvan de twee wanden geen deel uitmaken van de rand van het domein. Omdat $\frac{dv_0}{dn}$ en $\frac{dv_1}{dn}$ op deze elementen constant zijn, is de term die de inwendige fluxen in rekening brengt, te vereenvoudigen tot:

$$\sum_{x_i \in \Gamma_{\text{int}}} \left(\left\langle \frac{dv}{dn} \right\rangle [u] - [v] \left\langle \frac{du}{dn} \right\rangle \right)(x_i) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}. \tag{B.6}$$

Voor wat betreft inwendige elementen voldoet de stoorfunctie dus aan het randwaardenprobleem.

B.3 Volumina aan de rand van het domein

Er zijn twee volumina aan de rand van het domein, Ω_i en Ω_N. De twee scharen testfuncties leveren elk één vergelijking; de term waarin de randvoorwaarden tot uitdrukking komen, is:

$$\sum_{x_i \in \Gamma_D} \left(\frac{dv}{dn} u - v \frac{du}{dn} \right)(x_i). \tag{B.7}$$

Als $x_0 \in \Gamma_{\mathcal{D}}$, dan levert Ω_1 de term:

$$\left(\frac{dv}{dn} u - v \frac{du}{dn} \right)(x_0) = \frac{1}{\Delta x} \begin{pmatrix} +1 \\ -1 \end{pmatrix} \hat{u}_1^0, \tag{B.8a}$$

De vergelijkingen horend bij Ω_N leveren:

$$\left(\frac{dv}{dn} u - v \frac{du}{dn} \right)(x_N) = \frac{1}{\Delta x} \begin{pmatrix} -1 \\ +1 \end{pmatrix} \hat{u}_N^1. \tag{B.8b}$$

Daarnaast is er de bijdrage van de numerieke fluxen; Ω_1 levert

$$\left(\left\langle \frac{dv}{dn}\right\rangle [u] - [v]\left\langle \frac{du}{dn}\right\rangle\right)(x_1) = \frac{1}{2}\frac{1}{\Delta x}\begin{pmatrix}+1\\-1\end{pmatrix}[u](x_1); \tag{B.9a}$$

Ω_N levert:

$$\left(\left\langle \frac{dv}{dn}\right\rangle [u] - [v]\left\langle \frac{du}{dn}\right\rangle\right)(x_{N-1}) = \frac{1}{2}\frac{1}{\Delta x}\begin{pmatrix}+1\\-1\end{pmatrix}[u](x_1). \tag{B.9b}$$

Met deze bijdragen worden de vergelijkingen opgesteld, waaraan de stoorfunctie moet voldoen.

Er zijn nu drie mogelijke situaties, elk van de drie levert een stelsel vergelijkingen op:

- $\Gamma_{\mathcal{D}} = x_0$; waarmee het volgende stelsel ontstaat:

$$\begin{cases} \hat{u}_1^0 + & \frac{1}{2}[u](x_1) & = 0, \\ & [u](x_{N-1}) & = 0. \end{cases} \tag{B.10}$$

  De oplossing van dit stelsel levert slechts een triviale stoorfunctie op: $u^*(x) = 0$.

- $\Gamma_{\mathcal{D}} = x_N$: hiermee ontstaat een vergelijkbaar stelsel:

$$\begin{cases} [u](x_1) & = & 0, \\ \hat{u}_N^1 + \frac{1}{2}[u](x_{N-1}) & = & 0. \end{cases} \tag{B.11}$$

  Ook dit stelsel levert slechts de stoorfunctie $u^*(x) = 0$ op.

- $\Gamma_{\mathcal{D}} = \{x_0, x_N\}$:

$$\begin{cases} \hat{u}_1^0 & +[u](x_1) & = 0, \\ [u]\,(x_{N-1}) & +\hat{u}_N^1 & = 0. \end{cases} \tag{B.12}$$

  Dit stelsel heeft één vrijheidsgraad, zodat een van nul verschillende stoorfunctie mogelijk is.

C. Plaatsing van de deelmatrices

In paragraaf 4.2 is (2.4a) met behulp van een ruimtediscretisatie en een keuze voor de testfuncties herschreven tot een stelsel vergelijkingen. Per element zijn twee gezochte waarden, $\hat{u}_0^i$ en $\hat{u}_1^i$, en twee vergelijkingen geleverd door voor $v(x)$, $v_0(x)$ dan wel $v_1(x)$ te kiezen. Er zijn dan voor N elementen $2N$ vergelijkingen met $2N$ onbekenden.

Het linkerlid van (2.4a) levert per element drie termen op, en voor elke term per testfunctie één vergelijking. In paragraaf 4.2 is het linkerlid van (2.4a) reeds termsgewijs in matrixnotatie weergegeven. In dit appendix wordt aangegeven hoe de matrices worden geplaatst in de grote matrix die het uiteindelijk op te lossen stelsel vergelijkingen representeert.

Net zoals de termen bij elkaar worden opgeteld, worden de deelmatrices 'over elkaar gelegd': de deelmatrices worden bij elkaar opgeteld.

C.1 Eerste term: de stijfheidsmatrix

De eerste term uit (2.4a) is met de gekozen ruimtediscretisatie en testfuncties in matrixvorm te schrijven als

$$\begin{pmatrix}
\bullet & \bullet & \cdot & \cdot & & & & & & & \\
\bullet & \bullet & \cdot & \cdot & & & & & & & \\
\cdot & \cdot & \bullet & \bullet & \cdot & \cdot & & & & & \\
\cdot & \cdot & \bullet & \bullet & \cdot & \cdot & & & & & \\
 & & \cdot & \cdot & \bullet & \bullet & \ddots & & & & \\
 & & \cdot & \cdot & \bullet & \bullet & & & & & \\
 & & & & \ddots & & \ddots & & & & \\
 & & & & & & & \cdot & \cdot & & \\
 & & & & & & & \cdot & \cdot & & \\
 & & & & & & \cdot & \cdot & \bullet & \bullet & \\
 & & & & & & \cdot & \cdot & \bullet & \bullet &
\end{pmatrix}
\begin{pmatrix}
\hat{u}_0^1 \\ \hat{u}_1^1 \\ \hat{u}_0^2 \\ \hat{u}_1^2 \\ \vdots \\ \\ \\ \vdots \\ \hat{u}_0^{N-1} \\ \hat{u}_1^{N-1} \\ \hat{u}_0^N \\ \hat{u}_1^N
\end{pmatrix}. \tag{C.1}$$

C.2 Tweede term: Dirichlet-randvoorwaarden
In de probleemstelling moeten één of twee Dirichlet-randvoorwaarden worden opgelegd. Elke Dirichlet-randvoorwaarde levert twee vergelijkingen op. De plaatsing van de bijbehorende matrices is dan weergegeven in (C.2). De termen horende bij een Dirichlet-randvoorwaarde in x_0 zijn aangegeven met '$\triangleleft$', de termen bij een randvoorwaarde in x_N met '$\triangleright$'.

$$\begin{pmatrix} \triangleleft & \triangleleft & \cdot & \cdot & & & & & & \\ \triangleleft & \triangleleft & \cdot & \cdot & & & & & & \\ \cdot & \cdot & \bullet & \bullet & \cdot & \cdot & & & & \\ \cdot & \cdot & \bullet & \bullet & \cdot & \cdot & & & & \\ & & \cdot & \cdot & \bullet & \bullet & \ddots & & & \\ & & \cdot & \cdot & \bullet & \bullet & & & & \\ & & & & \ddots & & \ddots & & & \\ & & & & & & & & \cdot & \cdot \\ & & & & & & & & \cdot & \cdot \\ & & & & & & \cdot & \cdot & \triangleright & \triangleright \\ & & & & & & \cdot & \cdot & \triangleright & \triangleright \end{pmatrix} \begin{pmatrix} \hat{u}_0^1 \\ \hat{u}_1^1 \\ \hat{u}_0^2 \\ \hat{u}_1^2 \\ \vdots \\ \\ \\ \vdots \\ \hat{u}_0^{N-1} \\ \hat{u}_1^{N-1} \\ \hat{u}_0^N \\ \hat{u}_1^N \end{pmatrix}. \tag{C.2}$$

C.3 Derde term: flux over de inwendige wanden
De gediscretiseerde vorm waarin de derde term in (2.4a) te schrijven is, ziet er voor de linkerwand als volgt uit:

$$\begin{pmatrix} \cdot & \cdot & \cdot & \cdot & & & & & & \\ \cdot & \cdot & \cdot & \cdot & & & & & & \\ * & * & * & * & \cdot & \cdot & & & & \\ * & * & * & * & \cdot & \cdot & & & & \\ * & * & * & * & \cdot & \cdot & \ddots & & & \\ * & * & * & * & \cdot & \cdot & & & & \\ & & & & \ddots & & \ddots & & & \\ & & & & & & & & \cdot & \cdot \\ & & & & & & & & \cdot & \cdot \\ & & & & & & * & * & * & * \\ & & & & & & * & * & * & * \end{pmatrix} \begin{pmatrix} \hat{u}_0^1 \\ \hat{u}_1^1 \\ \hat{u}_0^2 \\ \hat{u}_1^2 \\ \vdots \\ \\ \\ \vdots \\ \hat{u}_0^{N-1} \\ \hat{u}_1^{N-1} \\ \hat{u}_0^N \\ \hat{u}_1^N \end{pmatrix}, \tag{C.3}$$

waarin de relevante termen met een asterisk ('$*$') zijn weergegeven. De gediscretiseerde vorm voor de rechterwand is

$$\begin{pmatrix} * & * & * & * & & & & & & \\ * & * & * & * & & & & & & \\ \cdot & \cdot & * & * & * & * & & & & \\ \cdot & \cdot & * & * & * & * & & & & \\ & & \cdot & \cdot & * & * & \ddots & & & \\ & & \cdot & \cdot & * & * & & & & \\ & & & & \ddots & & \ddots & & & \\ & & & & & & & & * & * \\ & & & & & & & & * & * \\ & & & & & & * & * & \cdot & \cdot \\ & & & & & & \cdot & \cdot & \cdot & \cdot \end{pmatrix} \begin{pmatrix} \hat{u}_0^1 \\ \hat{u}_1^1 \\ \hat{u}_0^2 \\ \hat{u}_1^2 \\ \vdots \\ \\ \\ \vdots \\ \hat{u}_0^{N-1} \\ \hat{u}_1^{N-1} \\ \hat{u}_0^N \\ \hat{u}_1^N \end{pmatrix}. \tag{C.4}$$

De relevante termen zijn met een asterisk weergegeven.

D. Object-georiënteerd programmeren

De code is op verschillende manieren *object-georiënteerd.* Dit is een moderne programmeerwijze die in moderne programmeertalen (C++, Java) mogelijk is. Deze wijze van programmeren vervult een aantal wensen vanuit de softwareontwikkeling.

In dit hoofdstuk wordt een aantal eigenaardigheden van de code besproken. Aan de precieze grammatica wordt geen aandacht besteed, noch aan de technische achtergronden. In deze paragraaf worden de structuren verklaard zoals die in het programma gebruikt worden.

D.1 Terminologie

Een object is een verzameling van:

- gegevens, en
- functies die op deze gegevens betrekking hebben.

Per gegeven of functie kan de toegang gespecificeerd worden. Dit kan bijvoorbeeld gebeuren om gegevens te beschermen.

Een *klasse* is een *type*-aanduiding; wanneer er een variabele van wordt gemaakt, heet deze een object of een instantie (Engels: instance).

Twee begrippen spelen binnen het object-georiënteerd softwareontwerp een cruciale rol: *data-encapsulatie* en *abstractie*. De abstractie komt tot uiting zowel bij gewone gegevens als bij functies.

D.2 Data encapsulatie

Data encapsulatie wil letterlijk zeggen: het verpakken van gegevens. Het doel hiervan is om goed met grote hoeveelheden gegevens om te kunnen gaan. Dit is een concept dat heel goed te realiseren is met behulp van een object-geörienteerd ontwerp.

D.2.1 Achtergrond Binnen de *Computational Fluid Dynamics* (CFD) komen stelsels lineaire vergelijkingen zeer vaak voor. Voor complexe problemen nemen deze in het algemeen veel geheugenruimte in.

In traditionele programmeertalen zoals Fortran-77 wordt tijdens de ontwikkeling van een programma de benodigde hoeveelheid geheugenruimte vastgelegd. Dit maakt het nodig dat per probleem deze hoeveelheid steeds opnieuw moet worden bepaald en aangepast.

Minder gebruikelijke programmeertalen (C, Pascal) kennen dynamisch geheugen. Dit laat het toe dat het programma tijdens een berekening geheugen reserveert, in plaats van dat het tijdens de ontwikkeling moet worden vastgelegd. Het programma wordt hierdoor makkelijker uitbreidbaar. Een nadeel van het gebruik van dynamisch geheugen is dat de boekhouding van het geheugen ook aan de programmeur wordt overgelaten: geheugen moet worden aangevraagd wanneer dat nodig is, en weer worden vrijgegeven als het niet meer nodig is. Index-overschrijvingen of gebruik van geheugen dat verkeerd is aangevraagd, kunnen tot verkeerde resultaten leiden. Dit soort programmeerfouten zijn lastig op te sporen, en leiden de aandacht af van het op te lossen probleem.

Een object is een verzameling van gegevens met de belangrijke eigenschap dat het mogelijk is de toegang tot gegevens te beheersen; dat wil zeggen dat gegevens of volledig worden afgeschermd, of slechts benaderbaar worden gemaakt met daarvoor bestemde routines. Een tweede belangrijke eigenschap is dat het aanmaken, kopiëren en verwijderen van objecten altijd op een gecontroleerde manier gebeurt.

Deze twee eigenschappen maken het mogelijk dynamisch geheugen te gebruiken, waarmee het programma efficiënt wordt. Tegelijkertijd kan dit geheugen volledig beheerst worden, om bijvoorbeeld array-overschrijdingen te ontdekken en zo fouten sneller op te sporen.

Het voordeel van het gebruik van objecten om data in op te slaan is dan ook tweeledig: enerzijds hoeven bibliotheken en programma's niet herschreven te worden als blijkt dat de hoeveelheid benodigd geheugen groter wordt dan aanvankelijk gedacht, omdat de hoeveelheid geheugen die het programma gebruikt zich aanpast aan de behoefte. Anderzijds kan men met objecten werken zoals met gewone variabelen in bijvoorbeeld FORTRAN. Het gebruik van pointers wordt vermeden, wat programmeerfouten kan voorkomen.

D.2.2 Een object in de praktijk: de klasse 'Matrix' Een binnen de CFD veel voorkomende abstractie is een matrix. Door mij is een bibliotheek geschreven die een eenvoudige implementatie van het concept 'matrix' heeft. Geïmplementeerd zijn onder andere:

- volledige automatische beheersing van het geheugengebruik bij creatie, kopie, assignment en verwijdering;
- optelling, aftrekking en vermenigvuldiging van matrices;
- Gauss-eliminatie met partiële pivotering of optimalisatie voor bandmatrices.

Van deze eigenschappen zullen twee voorbeelden worden gegeven zoals deze in de code staan.

Automatisch geheugengebruik In de code is onder andere terug te vinden de volgende regel:

```
Matrix A(2*N,2*N);
```

Deze regel geeft aan dat `A` een matrix is van $2N$ bij $2N$ elementen. Er wordt nu voldoende (dynamisch) geheugen gereserveerd voor $4N^2$ reële getallen met dubbele precisie. Het weer vrijgeven van het geheugen gebeurt automatisch bij het verlaten van het programma.

Een andere regel ziet er als volgt uit:

```
Matrix A=B;
```

Hier wordt een exacte kopie gemaakt van matrix `B`. Wederom wordt (precies) genoeg geheugen gereserveerd en *tijdens de werking van het programma*, als dat nodig is, weer vrijgegeven. De benodigde hoeveelheid geheugen wordt dus niet tijdens het schrijven of compileren vastgelegd. .

Oplossing van een stelsel In de code wordt een stelsel vergelijkingen opgelost door middel van Gauss-eliminatie van een bandmatrix. De functie 'linSolve' lost het stelsel $\mathbf{Au} = \mathbf{l}$ op; twee argumenten zijn $\mathbf{A}$ en $\mathbf{l}$, het resultaat is $\mathbf{u}$. Een optioneel argument is de bandbreedte, in welk geval slechts een bandmatrix geveegd wordt. In het programma ziet het er zo uit:

```
Matrix u = linSolve(A,l,5);
```

Op het eerste gezicht ziet de functie 'linSolve' er uit als een gewone functie. De argumenten zijn echter matrices en het resultaat is ook een matrix.

De schijnbare eenvoud is opzettelijk: de zeer gecompliceerde boekhouding gebeurt volledig uit het zicht. Dit houdt de code zeer 'schoon': er is een strikte scheiding tussen wat de verschillende programmeurs 'zien'; de wijze waarop de bibliotheek geschreven is en de wijze waarop de bibliotheek gebruikt wordt, zijn onafhankelijk van elkaar.

D.3 Abstracte functies

Een tweede voorbeeld van hoe een object-georiënteerde programmeerwijze de ontwikkeling van uitbreidbare software kan vergroten, is het gebruik van abstracte functies. De theorie is gecompliceerd, daarom wordt geprobeerd aan de hand van een voorbeeld dit begrip toe te lichten.

Het gedeelte van het programma waar gebruik wordt gemaakt van abstracte functies is waar het rechterlid moet worden uitgerekend. Er moeten hier functiewaarden berekend worden. Om deze routine zo algemeen mogelijk te houden, is het wenselijk deze functie *buiten* de routine te schrijven. Ideaal zou het zijn als de *functie* als parameter van de routine zou kunnen worden meegegeven.

Dit is een abstractie die traditionele programmeertalen zoals FORTRAN niet kennen. In een flexibele taal zoals C is het in principe mogelijk, om met pointers naar functies te verwijzen. Naast het feit dat dit zeer nauwkeurig (en dus foutgevoelig) programmeren vereist, is C voor wetenschappelijke toepassingen minder geschikt. Met de ontwikkeling van C++ en Java is het met de toepassing van abstracte klassen veel makkelijker geworden functies als parameter te gebruiken.

Het idee is als volgt: zoek naar de overeenkomsten tussen functies in het algemeen. Voor ééndimensionale functies wordt nu gesteld dat zij alle gedefinieerd worden als

```
double f(x)  {
   ...
}
```

Het is nog steeds niet mogelijk functies als parameter te gebruiken, maar het is wel mogelijk objecten als parameter op te geven. Aangezien objecten functies kunnen bevatten, wordt nu een klasse `Functie` gedefinieerd, die als het ware om `f(x)` heen zit. De definitie ziet er op het eerste gezicht cryptisch uit:

```
class Functie  {
public:
   virtual double f(double x)=0;
};
```

Direct valt op dat `f(x)` als '0' gedefinieerd is. Dit wil zeggen dat de functie uitdrukkelijk *niet* gedefinieerd is; het heeft niets te maken met de functiewaarde.

Het is niet mogelijk een object te maken van dit type, maar het is wel mogelijk een *afgeleide klasse* te maken van deze klasse. De afgeleide klasse geeft dan de garantie dat de niet-gedefinieerde functies wel gedefinieerd worden. Een sinusfunctie ziet er dan zo uit:

```
class Sin : public Functie  {
public:
   virtual double f(double x) {
      return sin(x);
   }
};
```

Wederom ziet het er cryptisch uit, mede door de grote hoeveelheid C++-grammatica, maar duidelijk te zien is dat aan de functie die eerst expliciet als *niet bestaand* was gedefinieerd, nu een concrete invulling is gegeven. Van deze klasse kunnen wel objecten gemaakt worden.

Het grote belang van abstracte klassen (object-typen) is dat zij, alhoewel er geen objecten zelf mee gemaakt kunnen worden, wel het type van een parameter van een functie kunnen aangeven. In de declaratie van de routine `dg1d` is dit te zien:

```
Matrix dg1d(Functie & S, ...).
```

De parameter `S` is dus een object met de garantie dat de lidfunctie `f(x)` bestaat. De functie kan dus ook worden aangeroepen. Om de notatie nog eenvoudiger te maken, mag ook `S(x)` worden geschreven als afkorting van `S.f(x)`.

Op deze manier is de definitie van de bronterm volledig buiten de routine `dg1d` gehouden. Besluit men later een andere functie te gebruiken, dan hoeft deze routine dan ook niet aangepast te worden, wat fouten kan voorkomen.

E. Beknopte beschrijving van de software

In dit appendix wordt de kern van het programma, de procedure '`dg1d`' besproken. Deze procedure is de feitelijke implementatie van de methode; de keuze van de bronfunctie en de keuze van de ruimtediscretisatie worden niet in deze procedure vastgelegd, maar zijn parameters. Ook worden in deze procedure data noch ingelezen, noch weggeschreven.

Het programma is zo intuïtief mogelijk geschreven, specifieke C++-kennis is niet nodig om de werking van het programma te kunnen volgen.

De procedure heeft parameters als invoer, en een matrix als uitvoer. Hierna volgt een beschrijving van deze in- en uitvoer.

Daarnaast maakt de procedure gebruik van de routine `linSolve`. Het gebruik van deze routine wordt in dit hoofdstuk uitgelegd.

E.1 Parameters

De parameters hebben de volgende betekenis en type:

S Bronterm $S(x)$; type is een van `Functie` afgeleide klasse waarvoor de lidfunctie `f(x)` gedefinieerd is; zie hiervoor ook D.3.

x0 en xN Linker- en rechterrand van het domein; beide dubbele precisie decimaal.

N Aantal elementen; geheel getal

type_rvw_x0 en type_rvw_xN Het type van de randvoorwaarde in respectievelijk x_0 en x_N, Dirichlet of Neumann. De waarde kan `rvw_Dirichlet` of `rvw_Neumann` zijn (zoals vastgelegd in `dg1d.h`)

rvw_x0 en rvw_xN De randvoorwaarden in x_0 en x_N zoals opgegeven; afhankelijk van de keuze is dit dus $u(x)$ of $u'(x)$; decimaal getal in dubbele precisie.

E.2 Indexering van een matrix
Een matrix van $m \times n$ elementen heeft indices $0 \cdots m-1$ bij $0 \cdots n-1$. Dit heeft te maken met de array-indexering in C++.

De volgende aanroep creëert een matrix in het geheugen van $m \times n$ elementen, en noemt deze A:

```
Matrix A(m,n);
```

Wordt nu een element in een matrix A aangeduid met rij i, kolom j, dan wordt dit element aangeduid met `A[i-1][j-1]`, wat wederom te maken heeft met telling in C++.

Het optellen van een deelmatrix bij een grotere matrix gebeurt met het commando `add`; de eerste parameter is een deelmatrix, de twee daaropvolgende parameters geven de rij en de kolom in de grotere matrix aan vanaf waar de kleinere matrix wordt opgeteld.

E.3 Oplossen van een stelsel vergelijkingen
Het oplossen van een stelsel vergelijkingen in de vorm $\mathbf{Ax} = \mathbf{y}$ gebeurt met het commando `linSolve`. Er zijn daarvan twee varianten mogelijk:

```
Matrix y = linSolve(A,x);
```

of

```
Matrix y = linSolve(A,x,n);
```

In beide gevallen wordt het stelsel opgelost, waarbij de oplossing in een matrix `y` wordt opgeslagen; in het eerste geval wordt gebruik gemaakt van partiële pivotering om de nauwkeurigheid zo groot mogelijk te maken. In het tweede geval wordt aangenomen dat `A` een bandmatrix met bandbreedte `n` is. Voor grote ijle matrices is de tijdwinst van de bandmethode aanzienlijk.

E.4 De routine dg1d.cpp
Deze sectie bevat de feitelijke implementatie van de Discontinue Galerkin-methode: de routine dg1d.cpp.

```
Matrix dg1d(Functie & S, double x0, double xN, int N, type_rvw
            type_rvw_x0, type_rvw type_rvw_xN,
            double rvw_x0, double rvw_xN)   {

        Matrix B(2*N, 2*N);
        double dx = (xN-x0) / (double) N;

        Matrix B_1(2,2);      // eerste term uit de bilineaire term
        B_1[0][0] = +1.0/dx;            B_1[0][1] = -1.0/dx;
```

```
        B_1[1][0] = -1.0/dx;            B_1[1][1] = +1.0/dx;


        Matrix B_2_0(2,2);    // tweede term uit de bilineaire vorm;
                              // Dirichlet-randvoorwaarde voor x_0


        B_2_0[0][0] = 0.0/dx;   B_2_0[0][1] = +1.0/dx;
        B_2_0[1][0] =-1.0/dx;   B_2_0[1][1] =  0.0/dx;

        Matrix B_2_N(2,2);    // tweede term uit de bilineaire vorm;
                              // Dirichlet-randvoorwaarde voor x_N

        B_2_N[0][0] =  0.0/dx;  B_2_N[0][1] = -1.0/dx;
        B_2_N[1][0] = +1.0/dx;  B_2_N[1][1] =  0.0/dx;




        Matrix B_3_R(2,4);    // derde term uit de bilineaire vorm;
                              // numerieke flux voor x_i

/*  0    -0.5   +0.5  0
   +0.5   0      0   -0.5
*/

        B_3_R[0][0] = .0/dx; B_3_R[0][1] =-.5/dx;
        B_3_R[0][2] =+.5/dx; B_3_R[0][3] = .0/dx;
        B_3_R[1][0] =+.5/dx; B_3_R[1][1] = .0/dx;
        B_3_R[1][2] = .0/dx; B_3_R[1][3] =-.5/dx;

        Matrix B_3_L(2,4);  // derde term uit de bilineaire vorm;
                            // numerieke flux voor x_i-1
/*  -0.5  0      0   +0.5
     0   +0.5   -0.5 0
*/
        B_3_L[0][0] =-.5/dx; B_3_L[0][1] = .0/dx;
        B_3_L[0][2] = .0/dx; B_3_L[0][3] =+.5/dx;
        B_3_L[1][0] = .0/dx; B_3_L[1][1] =+.5/dx;
        B_3_L[1][2] =-.5/dx; B_3_L[1][3] = .0/dx;

/*
   Nu wordt de matrix B opgebouwd door de drie componenten in de matrix
   te plaatsen.
*/

        // eerste term geldt voor alle elementen

        for (int i=1; i<=N; i++)
           B.add(B_1, (i-1)*2, (i-1)*2);

/*
        tweede term leveren Dirichlet-randvoorwaarden; weglaten
        levert een Neumann-voorwaarde; het is dus niet mogelijk om
```

```
        op 1 punt zowel een Dirichlet- als een Neumann-
        randvoorwaarde te hebben, ook al is het probleem dan nog
        steeds goed gesteld!
*/

        if (type_rvw_x0 == rvw_Dirichlet)
           B.add(B_2_0, 0, 0);         // Dirichlet voor eerste element

        if (type_rvw_xN == rvw_Dirichlet)
           B.add(B_2_N, 2*N-2, 2*N-2); // Dirichlet voor laatste element

/*
         let op telling! laatste index is 2*N-1

         derde term rechts; geldt voor alle elementen met een inwendige x_i,
         dat zijn elementen 1 ... N-1
*/

        for (int i=1; i<=N-1; i++)
           B.add(B_3_R, (i-1)*2, (i-1)*2);

/*
        derde term links; geldt voor alle elementen met een
        inwendige x_(i-1), dat zijn elementen 2 ... N
*/
        for (int i=2; i<=N; i++)
                B.add(B_3_L, (i-1)*2, (i-2)*2); // let op de plaatsing
                                                // en de telling!


/*
        De bronterm wordt opgebouwd met behulp van de functie S
        en gediscretiseerd met behulp van een eenvoudige
        kwadratuurformule (zie rapport, app. A)
*/

        Matrix L(N*2,1);

        double x1,x2;
        for (int i=1;i<=N;i++)  {
          x1 = x0 +(i-1)*dx;
          x2 = x1 + dx;
          L[(i-1)*2][0]= dx / 6.0 * ( S(x1) + 2.0 * S(x2));
          L[(i-1)*2+1][0] = dx /6.0 * (2.0 * S(x1) + S(x2)) ;
        }

/*
        Randvoorwaarden:

        voor x0 Dirichlet:  -dv/dx * f(0)
        voor x0 Neumann:      v(0) * g(0)
*/

        Matrix L_rvw_x0(2,1), L_rvw_xN(2,1);
```

```
        if (type_rvw_x0 == rvw_Dirichlet)  {
                        L_rvw_x0[0][0] = +rvw_x0 / dx;
                        L_rvw_x0[1][0] = -rvw_x0 / dx;
        }
        else  {         // Neumann
                        L_rvw_x0[1][0] =  -    rvw_x0;
                        L_rvw_x0[0][0] =          0.0;
        }



/*
        voor xN Dirichlet: +dv/dx * f(xN)
        voor xN Neumann:     v(xN) * g(xN)
*/

        if (type_rvw_xN == rvw_Dirichlet)  {
                        L_rvw_xN[0][0] = -rvw_xN / dx;
                        L_rvw_xN[1][0] = +rvw_xN / dx;
        }
        else  {       // Neumann
                        L_rvw_xN[0][0] =          0.0;
                        L_rvw_xN[1][0] =  +    rvw_xN;
        }

        L.add(L_rvw_x0, 0,0);
        L.add(L_rvw_xN, 2*N-2,0);

/*
Tot zover de voorbereidingen. Nu kan het stelsel B u = l
opgelost worden. De '5' geeft de bandbreedte van de
matrix aan, waardoor tijdswinst geboekt wordt.
*/

        Matrix U = linSolve (B,L,5);

/*
Het resultaat wordt opgeslagen in een Vector U, en teruggegeven
aan het hoofdprogramma als resultaat.
*/

        return U;

}
```

Table of Contents