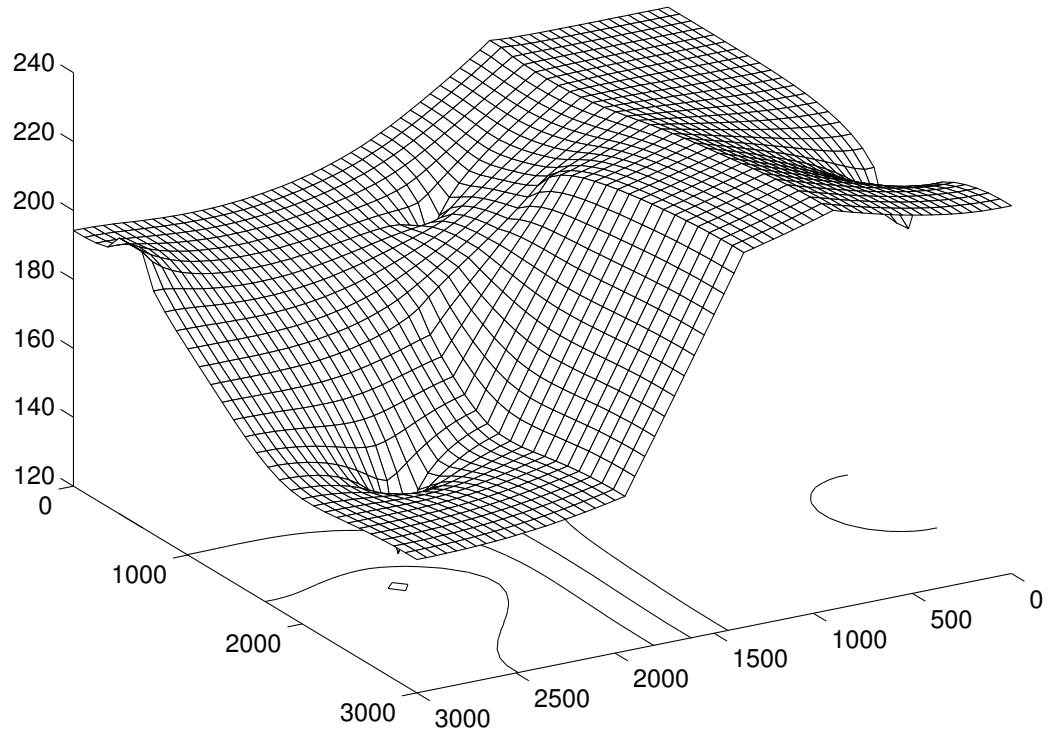
COMPUTATIONAL SCIENCE PRAKTIKUM
Iteratieve Lineaire Oplosmethoden, WISB354



Gerard Sleijpen

Universiteit Utrecht

Kamer 504, WG
Tel: 030-2531732
030-6373580 (privé)
`http://www.math.ruu.nl`
`/people/sleijpen/`
`E-mail:sleijpen@math.ruu.nl`

Wiskundig Instituut

7de herziene druk mei 2009

Werkschema

Sessies van 4 uur elk.

Sessie 1 en 2. t/m hoofdstuk 4
(discretisatie en matrix formulering).

Sessie 3 en 4. t/m hoofdstuk 7.3
(CGR en preconditionering).

Sessie 5 en 6. t/m hoofdstuk 9
(Expliciet preconditionering en herstart).

Sessie 7 en 8. Tweede methode (naar keuze).

Sessie 9 t/m 12. Afronding en verslag.

Werkbelasting. 7.5 ECTS = 160 uur werk.

Bij de "opdrachten" wordt gebruik gemaakt van de computer. De "*opgaven*" zijn theoretisch van aard en kunnen thuis gemaakt worden. Om geen kostbare praktikumtijd te verspillen wordt aanbevolen de tekst thuis te lezen en de opgaven ook voor thuis te bewaren. In de "(P)" opdrachten wordt gevraagd om een stukje code te schrijven, terwijl in de "(E)" opdrachten de nadruk ligt op experimenteren. Een deel van de code is, met name voor de eerste paar opdrachten, al geschreven. De code voor de met $*$ gemarkeerde opdrachten is zelfs helemaal kompleet.

The tekst in kleine letters bevat voor de liefhebbers wat extra theoretische achtergrond informatie.

De programma's worden in C of C++ geschreven.

1 Inleiding

In deze cursus staat de lineaire vergelijking

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{1}$$

centraal. Hierin is $\mathbf{A} = (a_{ij})$ een bekende $n \times n$ matrix, $\mathbf{b}$ een bekende n-vector en $\mathbf{x}$ een n-vector die bepaald moet worden. De dimensie n van het probleem is groot ($\geq 10^4$). De matrix heeft een *ijle* struktuur (sparse structure), d.w.z. het aantal niet nullen per rij is beperkt (denk aan 10). De *bandbreedte* (de maximale j waarvoor $a_{i,i+j} \neq 0$ of $a_{i,i-j} \neq 0$, waarbij het maximum genomen is over alle i; bandbreedte$\equiv \#\{j-1 \mid a_{ij} \neq 0\}$) is echter groot en loopt op met de dimensie van het probleem (typisch voorbeeld bandbreedte $\sim \sqrt{n}$).

Dit soort problemen komen in de praktijk vaak voor. In computer simulaties van praktische problemen waarbij (i) de ingangsgegevens verwerkt moeten worden, (ii) de matrix en het rechterlid moeten worden opgesteld, (iii) het lineaire probleem moet worden opgelost en (iv) de oplossing (na eventuele verdere bewerking) gerepresenteerd moet worden, gaat vaak meer dan 90% van de rekentijd zitten in stap (iii), in het oplossen van de lineaire vergelijking. Het loont dus zeer de moeite om een efficiënte oplosmethode voor het lineaire probleem op te sporen. De dimensie van het probleem dat gesimuleerd kan worden, wordt vaak bepaald door de "capaciteit" van de lineaire oplosmethode die gebruikt wordt. De dimensie bepaalt vaak de nauwkeurigheid van de oplossing: een grotere nauwkeurigheid, een "groter oplossend vermogen", vereist een grotere dimensie.

We zullen zien dat direkte oplosmethoden, die vrijwel allen gebaseerd zijn op Gauss eliminatie, voor grote ijle lineaire problemen onaantrekkelijk zijn. We zijn aangewezen op zogenaamde iteratieve methoden. In feite is er maar een direkte methode. Iteratieve methoden zijn er echter in diverse soorten die bovendien nog fundamenteel verschillend zijn. De kwaliteit van de iteratieve methode hangt af van eigenschappen van het lineaire probleem (en van de computer architektuur). Vaak zijn die eigenschappen niet bekend of moeilijk te achterhalen. Om tot een goede keus te komen is expertiese nodig betreffende de lineaire vergelijking, iteratieve methoden en hun samenhang. De krusiale eigenschappen van het lineaire probleem hangen vaak samen met eigenschappen van het onderliggende fysische probleem.

In deze cursus proberen we een zekere expertiese op te bouwen.

Uiteraard schenken we aandacht aan hele trajekt van fysisch, praktisch probleem, via discretisatie en implementatie naar het oplossen van het discrete stelsel en interpretatie van de oplossing. Maar onze aandacht gaat vooral uit naar iteratieve oplosmethoden.

2 Het fysisch probleem

2.1 Stroming grondwater

We bekijken de stroming van grondwater door een poureuse grondlaag. Voor het gemak nemen we aan dat de grondlaag 2-dimensionaal is en het grondwater in het xy-vlak stroomt (d.w.z. de grondlagen veranderen niet in de z richting; er is geen stroming in de z richting). Verder is de stroming stationair, d.w.z. het stromingsveld verandert niet in de tijd.

De stroming in het deel D van het xy-vlak waarin we geïnteresseerd zijn kan beschreven worden door de volgende partiële differentiaalvergelijking:

$$-\nabla \cdot (K \nabla \phi) = Q, \tag{2}$$

waarbij de grootheden de volgende betekenis hebben.
— $\phi = \phi(x, y)$ is de grondwater druk (*piëzometrische hoogte*) op plaats (x, y) en wordt gemeten in meters.

— $K = K(x, y)$ geeft de doorlaatbaarheid (*permeability*) aan van de grondlaag op plaats (x, y) en is een (symmetrische) 2×2 matrix die per plaats kan verschillen en afhankelijk is van de grondsoort, met als coëfficiënten $k_{ij}(x, y)$ de *doorlaadbaarheidscoëfficiënten* (in m^3/dag m^2 m, d.w.z., in kubieke meter per dag door een oppervlakte eenheid gemeten in m^2 per eenheid drukverschil).
— Het *snelheidsveld* (u, v) van de grondwaterstroming (in m^3/dag m^2, d.w.z., kubieke meter per dag door een vierkante meter oppervlakte) is het vectorveld gegeven door

$$(u, v) \equiv -K \nabla\phi \equiv -K \left(\frac{\partial\phi}{\partial x}, \frac{\partial\phi}{\partial y} \right)^T : \qquad (3)$$

$u(x, y)$, $v(x, y)$ is de snelheid waarmee het grondwater op plaats (x, y) stroomt in de x-, respektievelijk y-richting.
— De *bronterm* $Q = Q(x, y)$ (in m^3 water per dag per m^3 grond) vertelt hoeveel water er toegevoegd wordt (rivier) of weggehaald wordt (put).

Verder hebben we te maken met randvoorwaarden. De rand ∂D van het gebied D bestaat uit twee disjuncte stukken Γ_0 en Γ_1 ($\partial D = \Gamma_0 \cup \Gamma_1$ en $\Gamma_0 \cap \Gamma_1 = \emptyset$).
Op Γ_0 hebben we de *essentiële randvoorwaarde* (of *Dirichlet randvooraarde*) die de druk van het grondwater voorschrijven:

$$\phi = \phi_0, \qquad (4)$$

waarbij ϕ_0 een gegeven functie is op Γ_0.
Op Γ_1 hebben we de *natuurlijke randvoorwaarde* (of *Neumann randvoorwaarde*) die de uitstroomsnelheid van het grondwater vastlegt:

$$-(K \nabla\phi) \cdot n = \phi_0 \qquad (5)$$

waarbij ϕ_0 een gegeven functie is op Γ_1 en n de *normaal vector* is ($n(x, y)$ is een vector ter lengte 1 die t.o.v. D naar buiten wijst en in (x, y) loodrecht op Γ_1 staat). Een karakteristieke waarde voor ϕ_0 op Γ_1 is $\phi_0 = 0$: er stroomt geen water door Γ_1 weg (bijvoorbeeld als D aan Γ_1 begrensd wordt door een ondoorlaatbare laag).
Gemengde randvoorwaarde (of *Robin randvoorwaarde*) van het type

$$-(K \nabla\phi) \cdot n = \gamma(\phi - \phi_\infty) \qquad (6)$$

worden ook nogal eens gebruikt. Voor een referentie druk ϕ_∞ is de uitstroomsnelheid everedig met het drukverschil (je kunt hierbij denken aan de bron van een rivier); γ is een evenredigheidsconstante.

Het probleem is om de grondwaterdruk en het snelheidsveld te bepalen gegeven de doorlaadbaarheids coëfficiënten, de brontermen en de randvoorwaarden.

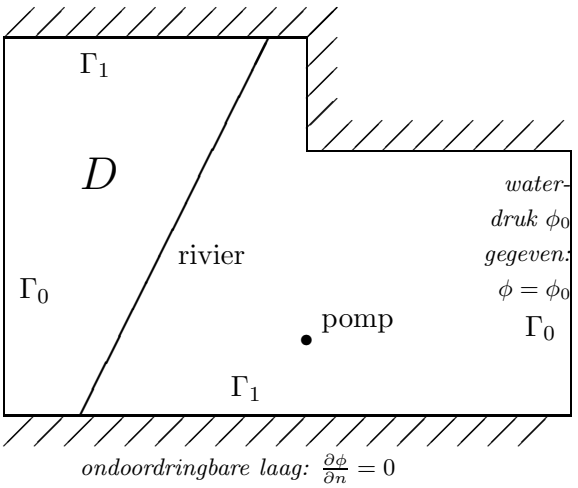


FIG.1: *Grondwaterstroming op een gebied D met rivier en pomp. Op de west – en de oost rand (Γ_0) Dirichlet randvoorwaarde, op de noord – en zuid rand (Γ_1) Neumann randvoorwaarde.*

2.2 Verwante problemen

Vergelijking (2), de *Poisson vergelijking*, speelt een belangrijke in zeer veel vakgebieden. In FIG.2 staan er een aantal opgesomd. Verder staat in het figuur de in het betreffende vakgebied gebruikelijke notatie en terminologie aangegeven met een indicatie voor een interpretatie van de termen.

De numerieke technieken die we in deze cursus zullen bekijken kunnen uiteraard ook toegepast worden om de problemen uit die vakgebieden op te lossen.

2.3 Bodemverontreiniging

Met een vergelijking verwant aan de Poisson vergelijking kan men bijvoorbeeld de verspreiding beschrijven van een in water oplosbare stof G in de grond via het grondwater.

Als ψ de concentratie van stof G aangeeft (in gr/m^3), dan voldoet ψ aan de differentiaalvergelijking

$$-\nabla \cdot (A\nabla\psi) + \nabla \cdot (V\psi) + c\,\psi = f, \qquad (7)$$

waarbij de grootheden A en $-A\nabla\psi$ een interpretatie hebben analoog aan die in ons grondwatermodel: A geeft de doorlaatbaarheid aan van de grondlaag met betrekking tot stof G (A zal niet evenredig zijn aan K; prima water-doorlaatbare lagen kunnen wellicht goed filteren), $-A\nabla\psi$ is de snelheid waarmee het gif zich verspreid door diffusie (in gr/(dag m^2), gram per dag door een m^2 grond); f de bronterm (in gr/(dag m^3)). $V = (u, v)$ is het snelheidsveld van het stromend grondwater (en is dus zelf de oplossing van een differentiaalvergelijking). Op een plaats (x, y) kan de concentratie veranderen
(i) door toevoeging van nieuwe stof G van buiten af

FIG.2: *Toepassingsgebieden van de vergelijking* $-\nabla\cdot(k\nabla u)=f$ *met essentiële randvoorwaarde,* $u=u_0$*, en/of natuurlijke randvoorwaarde,* $k\frac{\partial u}{\partial n}+h(u-u_\infty)=q$.

Toepassingsgebied	Functie u	Coëfficiëntfuncties k	Bronterm f	Overige grootheden q, $\frac{\partial u}{\partial x}$ $\frac{\partial u}{\partial y}$
Warmtetransport	Temperatuur T	Geleiding k	Warmtebron Q	Warmtestroom q geleiding als $h=0$ straling als $k=0$.
Rotatie vrije vloeistofstroming	Snelheidspotentiaal ϕ	Dichtheid ρ	Massaverandering σ	$\frac{\partial\phi}{\partial x}=u$, $\frac{\partial\phi}{\partial y}=v$
	Stromingsfunctie ψ	Dichtheid ρ	Massaverandering σ	$\frac{\partial\psi}{\partial x}=-v$, $\frac{\partial\psi}{\partial y}=u$
Grondwaterstroming	Piëzometrische hoogte ϕ	Doorlaatbaarheid K	Put ($Q>0$) of pomp ($Q<0$)	q lekkage $q=K\frac{\partial\phi}{\partial n}$, $-K(\frac{\partial\phi}{\partial x},\frac{\partial\phi}{\partial y})=(u,v)$
Torsie (mechanica)	Spanningsfunktie (Stress) ϕ	$k=1/G$ met G schuifmodulus	$Q=2\vartheta$, ϑ verdraaiingshoek per lengte eenheid	$\frac{\partial\psi}{\partial x}=-\tau_{zy}$, $\frac{\partial\psi}{\partial y}=\tau_{zx}$, τ_{zx}, τ_{zy} zijn schuifspanningen
Electrostatika	Potentiaal ϕ	Diëlektrische constante ε	Ladingsdichtheid ρ	Verplaatsing flux dichtheid D_n
Magnetostatika	Magnetische potentiaal ϕ	Permeabiliteit μ	Ladingsdichtheid ρ	Magnetische flux dichtheid B_n
Transversale uitwijking van elastische membranen	Transversale uitwijking u	T spanning membraan	Transversale verdeling belasting	Normaal kracht q

(gemodelleerd door de *bronterm* f),
(ii) door *diffusie* (gemodelleerd door de diffusie term $-\nabla\cdot(A\nabla\psi)$),
(iii) door *convectie* (ook wel *advectie* genoemd), d.w.z., door mee te stromen met het grondwater (gemodelleerd door de convectie term $\nabla\cdot(V\psi)$), en
(iv) door natuurlijke afbraak (die per plaats evenredig zal zijn met de concentratie; c is die evenredigheidsconstante, die afhangt van de grondsoort; c is niet negatief).

Bij dit model nemen we weer aan dat de aanvoer van stof G van buitenaf (de bronterm f en de randvoorwaarden) niet verandert in de tijd.

Differentiaal vergelijking (7) is een zogenaamde *convectie-diffusie vergelijking.*

3 Discretisatie

Om al te technische details te vermijden nemen we aan dat D een rechthoekig gebied is

$$D=(0,X)\times(0,Y),$$

waarop we de differentiaalvergelijking

$$-\frac{\partial}{\partial x}\left(a\,\frac{\partial\psi}{\partial x}\right)-\frac{\partial}{\partial y}\left(b\,\frac{\partial\psi}{\partial y}\right)+\frac{\partial(u\psi)}{\partial x}+\frac{\partial(v\psi)}{\partial y}+c\,\psi=f. \tag{8}$$

bekijken (zie (2) en (7)), met randvoorwaarde

$$-\mu\left(a\,\frac{\partial\psi}{\partial x},b\,\frac{\partial\psi}{\partial y}\right)\cdot n+(1-\mu)\,\psi=\psi_0, \tag{9}$$

waarbij ψ_0 gegeven is op de rand ∂D van D, $\mu=0$ op Γ_0 en $\mu=1$ op Γ_1 (zie (4) en (5): μ definieert in feite de partitie Γ_0, Γ_1 van ∂D. Neem $\mu=1/(1-\gamma)$ in geval (6)).

Over D leggen we een rechthoekig rooster (grid) R: voor

$$h_x=\frac{X}{n_x+1}\quad\text{en}\quad h_y=\frac{Y}{n_y+1}$$

met $n_x,n_y\in\mathbb{N}$, bestaat R uit de punten

$$(x_i,y_j)\equiv(ih_x,jh_y)\quad\text{in } D$$

$(i,j\in\mathbb{N},i\le n_x,j\le n_y)$; zie FIG.3.

We korten $\psi(x_i, y_j)$ af tot ψ_{ij} en bekijken differentiebenaderingen van $\frac{\partial\psi}{\partial x}$ in (x_i, y_j):

$$\partial_x^\circ \psi_{ij} = \partial_x^\circ \psi(x_i, y_j) \equiv \frac{\psi_{i+1,j} - \psi_{i-1,j}}{2h_x} \tag{10}$$

en

$$\partial_x^\bullet \psi_{i,j} = \partial_x^\bullet \psi(x_i, y_j) \equiv \frac{\psi_{i+\frac{1}{2},j} - \psi_{i-\frac{1}{2},j}}{h_x} \tag{11}$$

waarbij $\psi_{i+\frac{1}{2},j}$ de waarde is van ψ in het tussenpunt $(x_{i+\frac{1}{2}}, y_j) \equiv (ih_x + \frac{1}{2}h_x, jh_y)$. We benaderen $\frac{\partial\psi}{\partial x}$ ook in dit tussenpunt:

$$\partial_x^\bullet \psi_{i-\frac{1}{2},j} = \partial_x^\bullet \psi(x_{i-\frac{1}{2}}, y_j) = \frac{\psi_{i,j} - \psi_{i-1,j}}{h_x}. \tag{12}$$

We gebruiken analoge definities voor de *centrale differenties* ∂_y° en $\partial_y^\bullet$ in de y-richting.

We *discretiseren* (8) als volgt in de roosterpunten (x_i, y_j):

$$\begin{aligned} -\partial_x^\bullet\left(a\,\partial_x^\bullet\psi\right) - \partial_y^\bullet\left(b\,\partial_y^\bullet\psi\right) \\ + \partial_x^\circ u\psi + \partial_y^\circ v\psi + c\,\psi = f + \delta, \end{aligned} \tag{13}$$

waarbij de term $\delta(x_i, y_j)$ de *discretisatiefout* voorstelt en $f(x_i, y_j)$ de bronterm. Merk op dat, hoewel we partiële afgeleiden van ψ ook in tussenpunten benaderen (in $(x_{i+\frac{1}{2}}, y_j)$, etc.), we in (13) toch, vanwege (12), alleen te maken hebben met functiewaarden $\psi_{i'j'}$ van ψ in de roosterpunten $(x_{i'}, y_{j'})$ (geen waarden in de tussenpunten!). De functies a en b moeten we wel berekenen in tussenpunten: $a_{i+\frac{1}{2},j} = a(x_{i+\frac{1}{2}}, y_j)$, etc.; voor de overige bekende functies u, v, c, f, hoeven we alleen de waarde in de $(x_{i'}, y_{j'})$ te kennen: $u_{i'j'} = u(x_{i'}, y_{j'})$, etc..

Voor punten (x_i, y_j) nabij de rand van D (de punten (x_i, y_1), (x_1, y_j), etc.) hebben we de functiewaarden ψ_{ij} buiten D nodig (ψ_{i0}, ψ_{0j}, etc.; de rand van D is niet bevat in D!) die we moeten bepalen met behulp van de randvoorwaarden, die we als volgt discretiseren

$$\begin{aligned} \mu_{i0}\left(b\,\partial_y^\bullet\psi\right)_{i,\frac{1}{2}} + (1-\mu_{i0})\,\psi_{i0} \\ = \psi_0(x_i, 0) + \delta_{i0}, \\ \mu_{0j}\left(a\,\partial_x^\bullet\psi\right)_{\frac{1}{2},j} + (1-\mu_{0j})\,\psi_{0j} \\ = \psi_0(0, y_j) + \delta_{0j}, \\ \vdots \end{aligned} \tag{14}$$

met δ_{i0}, $\delta_{0,j}$, de discretisatiefouten van de randdiscretisatie.

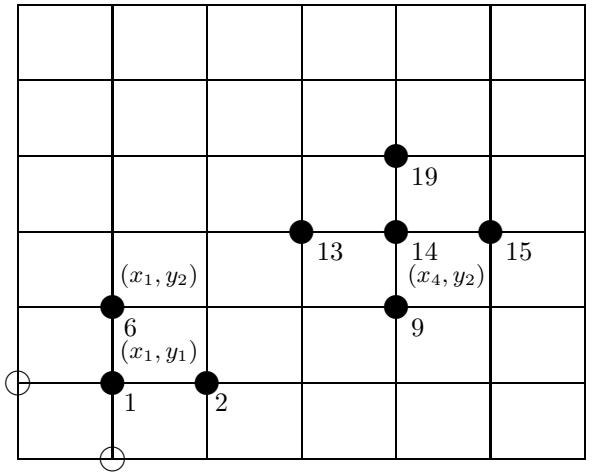


FIG.3: *Het rooster voor een rechthoekig gebied. De punten • behoren tot D, de punten ∘ liggen op de rand ∂D van D.*

Als we de discretisatiefouten negeren, kunnen we de functiewaarden ψ_{ij} op de rand uitdrukken in bekende functiewaarden $\psi_0(x_i, y_j)$ en functiewaarden $\psi_{i,j+1}$, $\psi_{i+1,j}$, etc., op het rooster: als $\mu_{i0} = 0$ dan $\psi_{i0} = \psi_0(x_i, 0)$, als $\mu_{i0} = 1$ dan $(b\,\partial_y^\bullet\psi)_{i,\frac{1}{2}} = \psi_0(x_i, 0)$, etc.. Hierdoor kunnen we de functiewaarden op de rand *elimineren* uit de vergelijkingen (13).

Negeren we de discretisatiefouten δ_{ij} dan hebben we hiermee een stelsel lineaire vergelijkingen

$$\begin{aligned} \alpha_{ij}^{\text{cent}}\psi_{i,j} + \alpha_{ij}^{\text{west}}\psi_{i-1,j} + \alpha_{ij}^{\text{oost}}\psi_{i+1,j} \\ + \alpha_{ij}^{\text{zuid}}\psi_{i,j-1} + \alpha_{ij}^{\text{noord}}\psi_{i,j+1} = \widehat{f}_{ij} \end{aligned} \tag{15}$$

met als onbekenden de functiewaarden ψ_{ij} op het rooster R; elk roosterpunt correspondeert met een vergelijking. De functies α^{cent}, α^{west}, ..., op het rooster R hangen af van de functies a, b, ..., op D en μ op ∂D; de functie $\widehat{f}$ op het rooster R hangt af van f op D en van μ en ψ_0 op ∂D.

De waarden ψ_{ij} van de oplossing van (15) geven in zekere nauwkeurigheid de exacte oplossing ψ van (8)&(9) in de roosterpunten (x_i, y_j). Met een programma als GNU-Plot of MATLAB kunnen we nu bijvoorbeeld de contourlijnen (de lijnen van gelijke druk of gelijke concentratie) plotten.

Met de ψ_{ij} kunnen we (nauwkeurig) het vectorveld $-A\nabla\psi$ in de punten (x_i, y_j) berekenen door

$$(-a_{ij}\partial_x^\circ\psi_{ij}, -b_{ij}\partial_y^\circ\psi_{ij}). \tag{16}$$

Dit vectorveld (op R of een deel van R) kan weer geplot worden met GNU-Plot of MATLAB.

De collectie van *koppelings coëfficiënten* α's (voor de vergelijking bij het (i,j)-de roosterpunt) wordt

vaak gerepresenteerd als een *stencil*:

$$\begin{bmatrix} 0 & \alpha_{ij}^{\text{noord}} & 0 \\ \alpha_{ij}^{\text{west}} & \alpha_{ij}^{\text{cent}} & \alpha_{ij}^{\text{oost}} \\ 0 & \alpha_{ij}^{\text{zuid}} & 0 \end{bmatrix}. \tag{17}$$

Een stencil representeerd één vergelijking met referentie naar het onderliggende rooster, terwijl een matrix een *stelsel* vergelijkingen voorstelt zonder roosterreferentie. Wordt het stelsel (15) in een matrix gezet dan correspondeert een stencil met een rij uit de matrix.

Opdracht 1 (P).

Schrijf een routine `diskretiseer` (in dubbele precisie) die, de rooster functies α^{cent}, α^{west}, α^{oost}, α^{noord}, α^{zuid} en $\widehat{f}$ produceert uit de functies a, b, c, u, v en f die gegeven zijn op D en uit μ en ψ_0 die gegeven zijn op de rand ∂D.

3A. Merk op dat we de benadering, die we voorstellen in (14) voor de normaal afgeleide, beter die afgeleide benadert in $(x_i, y_{\frac{1}{2}})$ dan in (x_i, y_0): voor $\mu_{i0} = 1$ is $\delta_{i0} = \mathcal{O}(h_y)$. De functiewaarde ψ_{i0} wordt daarentegen foutloos berekend: als $\mu = 0$ dan $\delta_{i0} = 0$.

We kunnen de normaal afgeleide op randpunten nauwkeuriger benaderen.
Voor ons rechthoekig rooster kan dat, als de natuurlijke randvoorwaarden langs hele zijden van de rechthoek gelden, door tussenpunten op de rand te laten uitkomen (als bijvoorbeeld $\mu = 1$ voor $y = 0$ en $\mu = 0$ voor $y = Y$, kan je $y_j = (j - \frac{1}{2})h_y$ nemen met $h_y = Y/(n_y + \frac{1}{2})$).

Andere mogelijkheid is om nauwkeuriger te interpoleren. De waarde $(b\,\partial_y^\bullet\psi)_{i,\frac{1}{2}}$ benadert $\psi_y \equiv \frac{\partial\psi}{\partial y}$ in $(x_i, y_{\frac{1}{2}})$ en zal bijvoorbeeld goed lijken op $\frac{1}{3}\psi_y(x_i, y_{\frac{3}{2}}) + \frac{2}{3}\psi_y(x_i, 0)$. Deze laatste waarde wordt weer goed benaderd door $\frac{1}{3}\partial_y^\circ\psi_{i,\frac{3}{2}} + \frac{2}{3}\psi_0(x_i, 0)$.

Voor een ingewikkelder gebied met kromme rand moeten de functies op de rand door interpolatie leiden tot functiewaarden op de roosterpunten.

3B. Voor de discretisatiefout in de benadering $\partial_x^\circ\psi$ en $\partial_x^\bullet\psi$ van $\frac{\partial\psi}{\partial x}$ kan gemakkelijk met behulp van Taylorreeksen een uitdrukking gevonden waarmee het mogelijk is deze fouten te schatten.

3.1 Het discretiseren van pompen

Stel dat in het roosterpunt $P = (x_{i_0}, y_{j_0})$ een pomp staat die het water wegpompt met een snelheid van p m^3 per dag. In de grondwatervergelijking (2) geeft dit een bijdrage in de bronterm. We geven hier een mogelijkheid om het effect van de pomp in het discrete model te verwerken.

In de bronterm moeten we aangeven hoeveel m^3 water er per dag per m^3 grond afgetapt wordt. In de rechthoek B_P van h_x bij h_y met midden P wordt per dag p m^3 water afgetapt. Omdat de functiewaarde van de functies die we bekijken in het midden P min of meer het gemiddelde van de functie over B_P representeren lijkt $p/(h_x h_y d)$ de correcte waarde van de pomp in de bronterm. Hierbij is d de dikte van de grondlaag (in de z-richting).

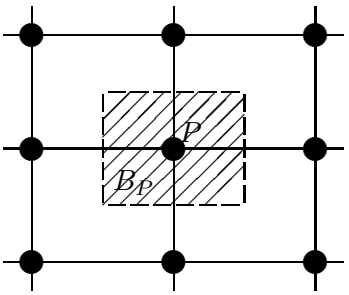


FIG.4: *De pomp gediscretiseerd*

Opdracht 2 (P).

Bouw in je discreet model de pomp(en) in met coördinaten $(x_{\text{pomp}}, y_{\text{pomp}})$ en met tabsnelheid p_{pomp}: pas in `diskretiseer` de bronterm $\widehat{f}$ op een geschikte manier aan.

* **Opdracht 3 (P)**.

Bouw rechtlijnige rivier(en) in. Een rivier wordt beschreven door de punten (x, y) waarvoor $ax + by = c$ met a, b en c bekende constanten, en voegt per dag en per strekkende meter rivier q m^3 water toe aan het grondwater (zie opgave A).

Opgave A. Door een $h_x \times h_y$ rechthoek B_P met midden P stroomt een rivier die q m^3 water per dag per m rivier aan het grondwater toevoegt. We nemen aan dat de rivier een lijn is en l meter lang is in B_P. Geef de bijdrage aan van de rivier in de bronterm in punt P.

3.2 Testproblemen

Testprobleem 0. A. $a = b = 1$, $u = v = c = 0$, $f = 2$ op $D = [0,1] \times [0,1]$. $\mu = 0$ & $\psi_0 = 0$ als $x = 0$ of $x = 1$, $\mu = 1$ & $\psi_0 = 0$ als $y = 0$ of $y = 1$ (dan is $\psi(x, y) = x(1 - x)$).
B. $a = 1 + y$, $b = 1 + x$, $u = x$, $v = y$, $c = 4$, $f = 8xy$ op $D = [0,1] \times [0,1]$. $\mu = 0$ & $\psi_0 = 0$ als $x = 0$, $\mu = 0$ & $\psi_0 = y$ als $x = 1$, $\mu = 1$ & $\psi_0 = x(1 + x)$ als $y = 0$, en $\mu = 1$ & $\psi_0 = -x(1 + x)$ als $y = 1$ (dan is $\psi(x, y) = xy$).

Testprobleem I. We modelleren de grondwaterstroom ($c = u = v = 0$).

$X = 3000$ m, $Y = 1500$ m,
$\mu = 0$ & $\psi_0 = 200$ m. als $x = 0$ of $x = X$,
$\mu = 1$ & $\psi_0 = 0$ als $y = 0$ of $y = Y$,
$a = b = 40$ m^3/(dag m^2),
in $(1000, 700)$ staat een pomp die 1200 m^3 per dag weghaalt,
$n_x = 29$, $n_y = 14$ $(d = 1)$.

Testprobleem II. Als I met als extra term een rechte rivier die van $(2500, 0)$ naar $(1000, 1500)$ stroomt met een toevoer van 0.24 m^3/dag m.

Testprobleem III. Als II maar nu met $a = b = 60$ m^3/(dag m^2) ten westen van de rivier, $a = b = 40$ m^3/(dag m^2) aan de oostkant, behalve als $|x - 2500| < 300$ en $y > 1000$. Voor deze laatste waarden van (x, y) is $a = b = 1$ m^3/(dag m^2).

Testprobleem IV. We bekijken de concentratie ψ van een stof G die zich verspreidt door het grondwater waarvan de stroming in III beschreven is. De stof wordt door een afvoer pijp in $(1900, 900)$ met 240 gram/dag de grond in gepompt. Op de noordelijke – en de zuidelijke rand van D hebben we natuurlijke randvoorwaarden (dus $\mu = 1$). In oost en west hebben we essentiële randvoorwaarden. Verder is ψ_0 gelijk aan 0 op de hele rand ∂D en is $a = b = 6$ ten westen van de rivier en $a = b = 4$ ten oosten; $c = 0$.
A. Voor het vectorveld (u, v) nemen we in eerste instantie $u = (y - 1000)/\kappa$, $v = (1500 - x)/\kappa$, met $\kappa = 5000$. Door κ te variëren kan je de sterkte van de stroming variëren. (Bekijk ook eens hetzelfde probleem maar neem nu, op de noord– en op de zuidrand, gemengde randvoorwaarden waarbij de uitstroom evenredig is met de concentratie ($\mu = 0.5$) (Testprobleem IV A′). Maak je nu de resolutie hoger (meer roosterpunten) dan zal je instabiliteiten aan de rand zien ontstaan. Deze instabiliteiten deden zich bij natuurlijke randvoorwaarden niet voor. De instabiliteiten ontstaan doordat de randvoorwaarden niet 'passen' bij de huidige convectie term. [1])
B. Neem tenslotte voor het vectorveld (u, v) het vectorveld geassocieerd aan de oplossing van III (berekend m.b.v. (16)) met natuurlijke randvoorwaarden op noordelijke – en zuidelijke rand. Merk op dat je hiervoor de routine `diskretiseer` moet aanpassen. Immers de routine `diskretiseer` berekent u en v uit functie-subroutines en niet uit array waarden als verkregen door III op te lossen.

Testprobleem V (Illustratie op de voorkant). $X = Y = 3000$. $a = b = 5$ voor $y > 1200$ en $1350 < x < 1950$ en $a = b = 40$ elders. $u = v = c = f = 0$ overal. Op de west rand $(x = 0)$: $\mu = -1$ & $\psi_0 = 400$ voor $1900 \le y \le 2100$, $\mu = 1$ & $\psi_0 = 0$ elders. Op de oostrand: $\mu = 0$ & $\psi_0 = 200$ voor $400 \le y \le 600$, $\mu = 1$ & $\psi_0 = 0$ elders. Op de zuid– en de noordrand: $\mu = 1$ en $\psi_0 = 0$. Pompen in $(2400, 1800)$ á 2400 m^3/dag en $(1550, 600)$ á 1200 m^3/dag. Een rivier stroomt over de lijn $x = 900$ met een toevoer van 1.4 m^3/dag m.

Opdracht 4 (P).
Voer de testproblemen in.

4 Een matrix formulering

Nummeren we de roosterpunten (of equivalent hiermee, nummeren we de onbekende functiewaarden en de vergelijkingen) dan kunnen we het stelsel vergelijkingen (15) representeren als een matrix-vector vergelijking als in (1): de vectorcoördinaten van $\mathbf{x}$ corresponderen met roosterfunctiewaarden ψ_{ij}; de rechterlid vector $\mathbf{b}$ correspondeert met $\widehat{f}_{ij}$ en bevat brontermen en gegeven randfuncties.

We nummeren de roosterpunten lexicografisch: per rij, van links naar rechts, en vervolgens de rijen van onder naar boven (zie FIG.3).

Opdracht 5 (P).
Schrijf een (dubbele precisie) routine

`vectoriseer`$(\alpha^{\text{cent}}, \ldots, \alpha^{\text{noord}}, \widehat{f}, \mathbf{A}, \mathbf{b})$

die de matrix $\mathbf{A}$ (en de rechterlidvector $\mathbf{b}$) produceert uit de bekende rooster functies α^{cent}, α^{west}, α^{oost}, α^{noord}, α^{zuid} (en $\widehat{f}$). $\mathbf{A}$ is een ijle matrix en moet in een *"compressed format"* opgeslagen worden; we kiezen hiervoor de *"row-indexed sparse storage mode"* als beschreven in Ch. 2 (p.78–79) van "Numerical Recipees" (NR) [4]. (zie 4B voor de relatie met het MATLAB format).

∗ **Opdracht 6 (P).**
Schrijf een routine

`continueer`$(\mathbf{x}, \psi, u, v)$

die uit de vectoroplossing $\mathbf{x}$ de oplossing ψ op het rooster produceert (inclusief de waardes voor ψ op de rand) met het bijbehorend (discreet) snelheidsveld $(u, v) = (-a\partial_x^\circ\psi, -b\partial_x^\circ\psi)$ (zie (16)).

Opdracht 7 (E).
Je kunt nu, als je de matrix en de rechterlidvector doorgeeft aan MATLAB, testen of de code die je tot dusver hebt correct is. Voor daartoe de testproblemen

[1] Wellicht komt dit doordat de randvoorwaarden voor een convectie probleem eigenlijk niet goed gemodelleerd worden door (9). Immers het stromingsveld is $-A\nabla\psi + V\psi$ en niet $-A\nabla\psi$: de termen $u\psi$ en $v\psi$ ontbreken in (9). Een perfecte filter houdt alle verontreiniging tegen ook als die met het grondwater meestroomt: dus dan $(-A\nabla\psi + V\psi) \cdot n = 0$.

FIG.5: *Schema: van het continue probleem naar het discrete en terug.*

Vergelijking	Bekende functies	Onbekende functie	Vorm representatie & definitie gebied
$-\nabla\cdot(A\nabla\psi)+\nabla\cdot(V\psi)+c\,\psi=f$	A, V, c en f	ψ	continu, op D
$-\mu\,A\nabla\psi\cdot n+(1-\mu)\psi=\psi_0$	μ, ψ_0	ψ	continu, op rand ∂D
	Discretisatie		
$\alpha_{ij}^{\text{cent}}\psi_{i,j}+\alpha_{ij}^{\text{west}}\psi_{i-1,j}$ $+\alpha_{ij}^{\text{oost}}\psi_{i+1,j}+\alpha_{ij}^{\text{zuid}}\psi_{i,j-1}$ $+\alpha_{ij}^{\text{noord}}\psi_{i,j+1}=\widehat{f}_{ij}$	α^{cent}, α^{west}, α^{oost}, α^{noord}, α^{zuid}, $\widehat{f}$	ψ	discreet, op rooster R van $n_x\times n_y$ punten
	Vectorisatie: $k=i+(j-1)\cdot n_x$		
$\mathbf{Ax}=\mathbf{b}$	$\mathbf{A}$, $\mathbf{b}$ met $\mathbf{A}_{kk}=\alpha_{ij}^{\text{cent}}$, $\mathbf{A}_{k,k+1}=\alpha_{ij}^{\text{oost}}$, $\mathbf{A}_{k,k-1}=\alpha_{ij}^{\text{west}}$, $\mathbf{A}_{k,k-n_x}=\alpha_{ij}^{\text{zuid}}$, …, en $\mathbf{b}_k=\widehat{f}_{ij}$	$\mathbf{x}$ met $\mathbf{x}_k=\psi_{ij}$	matrix-vector vergl., op vectorruimte $\mathbb{R}^n$ van dimensie $n=n_x\cdot n_y$
	Los op: $\mathbf{Ax}=\mathbf{b}$		
	Naar rooster: $i=((k-1)\bmod n_x)+1$, $j=\lceil k/n_x\rceil$		
	ψ met $\psi_{ij}=\mathbf{x}_k$, $(u,v)=(-a\partial_x^\circ\psi,-b\partial_y^\circ\psi)$		rooster

in. Met name testprobleem 0 is nuttig, omdat je daarvan de exacte oplossingen kent.

Opgave B. Er is een eenduidig verband tussen de rooster functies α^{cent}, α^{west}, … en de matrix $\mathbf{A}$. *Matrix eigenschappen kunnen dus vertaalt worden in termen van de α's (en in termen van de differentiaalvergelijking).*

Wat betekent "$\mathbf{A}$ *is symmetrisch" in termen van de α's? Ga na dat, dankzij de discretisering die we gekozen hebben,* $\mathbf{A}$ *symmetrisch is precies dan als de convectieterm nul is.*
Zouden we ook een symmetrische matrix gevonden hebben voor de vergelijking $-a\nabla\cdot\nabla\psi=f$ (met a $\mathbb{R}$-waardig)?
Zouden we ook een symmetrische matrix gevonden hebben voor het convectievrije probleem als we de vergelijking (14) *hadden toegevoegd aan het stelsel* (13)*, i.p.v. de functiewaarden ψ_{ij} op de rand te elimineren uit* (13)*?*

4A. The functiewaarden in de rooster punten van de exacte oplossing van (8)&(9) voldoen aan een stelsel vergelijkingen als in (15); we moeten alleen bij de waarden $\widehat{f}_{ij}$ in rechterlid een term $\widehat{\delta}_{ij}$ op tellen waarin de locale discretisatiefouten van de differentiaalvergelijking en van de randvoorwaarden verwerkt zijn. De functiewaarden van de exacte oplossing kunnen ook een vector gezet worden; noem deze vector $\mathbf{x}^*$. Zetten we de discretisatiefouten δ_{ij} in een vector $\mathbf{d}$ dan geldt $\mathbf{Ax}^*=\mathbf{b}+\mathbf{d}$ en dus $\mathbf{A}(\mathbf{x}^*-\mathbf{x})=\mathbf{d}$. We kunnen de *globale* discretisatiefouten schatten door

$$\|\mathbf{x}^*-\mathbf{x}\|\le\|\mathbf{A}^{-1}\|\,\|\mathbf{d}\|.$$

4B. In het format van MATLAB (zie MATLAB "`sparse`" and "`find`"): wordt $\mathbf{A}$ gerepresenteerd door een tripel $[i,j,S]$ van drie kolom vectoren van gelijke lengte, waarbij de vectoren i en j uit positieve integers bestaat terwijl de vector S reële (of complexe) getallen bevat. De niet-nullen van $\mathbf{A}$ staan in S. De

elementen van i en j geven aan waar die niet nullen te vinden zijn: $S(k)$ is precies $\mathbf{A}_{i(k),j(k)}$.

Matlab hanteert niet het meest kompakte format. De "row-indexed sparse storage mode" uit NR [4] is veel kompakter en laat efficiënter rekenen toe. Het kan toch nuttig zijn om het NR-format om te zetten in het format van MATLAB. We zouden dan met MATLAB onze programma's kunnen testen (we kunnen bijvoorbeeld met MATLAB de vergelijking oplossen).

De relatie tussen het MATLAB format en het row-indexed sparse storage mode j', S van NR kan als volgt beschreven worden.

Stel dat de MATLAB vectoren als volgt geordend zijn. Eerst worden de diagonaal elementen van $\mathbf{A}$ geordend gegeven (ook als ze identiek 0 zijn): $i(k) = j(k) = k$ voor $k = 1, \ldots, n$ en $i(k) < i(k+1)$ voor $k = 1, \ldots, n-1$. Verder is i geordend in stijgende grootte: $i(k) \leq i(k+1)$ voor $k = n+1, \ldots, n_z$ met n_z het totaal aantal niet nullen van $\mathbf{A}$ (n_z is de lengte van de i, j en S vectoren).

Spreken we af dat we het MATLAB tripel $[i, j, s]$ altijd zo ordenen dan hoeven we de eerste n coördinaten van i en j niet meer expliciet te beschrijven. We kunnen dan de eerste n j-coördinaten gebruiken om aan te geven waar i opgehoogd wordt en de hele i vector is overbodig. Dus $j'(1) = n+1$ en vertelt (i) dat de matrix $n \times n$ is en geeft (ii) aan waar de 1-ste rij van $\mathbf{A}$ begint; $j'(2) = k$ betekent dat $S(k) = \mathbf{A}_{2\,j(k)}$, etc.:

$$S(k) = \mathbf{A}_{i\,j(k)} \quad \text{voor} \quad j(i) \leq k \leq j(i+1) - 1.$$

Verder veranderen we j niet: $j'(k) = j(k)$ voor $k > n+1$.

Om bij de bouw van de j en de S in de "row-indexed sparse storage mode" een "if"-statement te vermijden wordt een dummy $j(n+1)$ en een dummy $S(n+1)$ toegevoegd: de eerste rij van $\mathbf{A}$ is dus te vinden vanaf $k = n+2$ en $j(1) = n+2$.

5 Iteratieve oplosmethoden

Iteratieve oplosmethoden berekenen uit een *benadering* $\mathbf{x}_k$ van de exacte oplossing $\mathbf{x}$ een nieuwe (hopelijk) betere benadering $\mathbf{x}_{k+1}$. Voor zo'n methode moet een *beginbenadering* $\mathbf{x}_0$ opgegeven worden. Vaak wordt daarvoor $\mathbf{x}_0 = 0$ genomen. In de *k-de iteratiestap*, in de berekening van $\mathbf{x}_{k+1}$ uit $\mathbf{x}_k$, wordt alleen gebruik gemaakt van eenvoudige rekenoperaties (*vector-update* (AXPY): $\alpha\mathbf{x} + \mathbf{y}$; *inproduct* (DOT): $\mathbf{x}^T\mathbf{y}$; *matrix-vector product* (MV): $\mathbf{Ax}$; plus eventueel het oplossen van eenvoudige stelsels $\mathbf{Mx} = \mathbf{y}$ (*Preconditionering*) met $\mathbf{M}$ bijvoorbeeld een diagonaal matrix, een bovendriehoeks matrix, een benedendriehoeks matrix of een product van dit soort matrices). De bedoeling is dat de $\mathbf{x}_k$ naar $\mathbf{x}$ convergeren en liefst al na een beperkt aantal stappen $\mathbf{x}$ in goede nauwkeurigheid benaderen. Voor een gemotiveerde introductie van iteratieve methoden, zie de Appendix.

De *residuen* $\mathbf{r}_k \equiv \mathbf{b} - \mathbf{Ax}_k$ spelen een grote rol: de exacte oplossing $\mathbf{x}$ kennen we namelijk niet en men mag hopen dat de *fout* $\mathbf{x} - \mathbf{x}_k$ klein is als het residu $\mathbf{r}_k = \mathbf{A}(\mathbf{x} - \mathbf{x}_k)$ klein is. De grootte van vectoren wordt gemeten in een norm, in bijvoorbeeld de Euclidische norm, $\|\mathbf{y}\|_2 \equiv \sqrt{\mathbf{y}^T\mathbf{y}}$ (of in een gewogen 2-norm: $\|\mathbf{y}\| = \sqrt{h_x h_y\, \mathbf{y}^T\mathbf{y}}$). Het residu kunnen we altijd berekenen. Men gebruikt de grootte van het residu vaak om de voortgang van het iteratief proces te beoordelen: men stopt de iteratie bijvoorbeeld als $\|\mathbf{r}_k\| \leq tol$ is, voor zekere $tol \ll 1$. De iteratieve methoden die wij bekijken streven er naar om het residu in zekere zin zo klein mogelijk te krijgen.

We bekijken zogenaamde *Krylov deelruimte methoden*. Het residu $\mathbf{r}_k$ van zo'n methode behoort tot de *Krylov deelruimte* $\mathcal{K}_{k+1}(\mathbf{A}; \mathbf{r}_0)$ (van dimensie $k+1$ voortgebracht door $\mathbf{A}$ en $\mathbf{r}_0$):

$$\mathcal{K}_{k+1}(\mathbf{A}; \mathbf{r}_0) \equiv \text{span}(\mathbf{r}_0, \mathbf{Ar}_0, \ldots, \mathbf{A}^k\mathbf{r}_0),$$

We bekijken drie verschillende (soorten) methoden.

* **Opdracht 8 (P)**.
Schrijf een (dubbele precisie) subroutine
`mv(A,u,r)`
voor de MV $\mathbf{r} = \mathbf{Au}$
(of neem de routines `atimes` en `sprsax` uit NR [4]).

5A. Omdat $\mathbf{A}(\mathbf{x} - \mathbf{x}_0) = \mathbf{b} - \mathbf{Ax}_0 = \mathbf{r}_0$ is, in exacte arithmetiek, het oplossen met een (lineaire) iteratieve solver van

$$\mathbf{Ax} = \mathbf{b} \quad \text{met startwaarde} \quad \mathbf{x}_0 \tag{18}$$

equivalent met het oplossen van

$$\mathbf{Ay} = \mathbf{r}_0 \quad \text{met startwaarde} \quad \mathbf{y}_0 = 0 : \tag{19}$$

de k-de benadering $\mathbf{x}_k$ van (18) is precies $\mathbf{x}_0 + \mathbf{y}_k$ met $\mathbf{y}_k$ de k-de benadering van (19). We mogen dus gerust $\mathbf{x}_0 = 0$ en $\mathbf{r}_0 = \mathbf{b}$ veronderstellen (anders verschuiven we het probleem als in (19)).

6 GCR: minimale residuen

De eerste methode, GCR (Generalized Conjugate Residuals; zie ALG.1), bepaalt de benadering $\mathbf{x}_k$ met minimaal residu in de Krylov deelruimte $\mathcal{K}_{k+1}(\mathbf{A}; \mathbf{r}_0)$. Hiertoe (zie 6A) wordt, om efficiëntie – en stabiliteits redenen, een orthogonale basis $\mathbf{c}_0, \ldots, \mathbf{c}_k$ van $\mathbf{A}\mathcal{K}_k(\mathbf{A}; \mathbf{r}_0)$ iteratief uitgebreid (met gemodificeerd Gram-Schmidt) en wordt $\mathbf{r}_{k+1}$ berekend als het orthogonale komplement van $\mathbf{r}_0$ op deze ruimte:

$$\mathbf{r}_{k+1} = \mathbf{r}_0 - \alpha_0\mathbf{c}_0 - \ldots - \alpha_k\mathbf{c}_k = \mathbf{r}_k - \alpha_k\mathbf{c}_k$$

$$\text{met} \quad \alpha_k = \mathbf{c}_k^T\mathbf{r}_0 / \mathbf{c}_k^T\mathbf{c}_k = \mathbf{c}_k^T\mathbf{r}_k / \mathbf{c}_k^T\mathbf{c}_k.$$

De benadering $\mathbf{x}_{k+1}$ en de *zoekrichting* $\mathbf{u}_k$ worden gelijktijdig bijgehouden. De zoekrichting $\mathbf{u}_k$ is gerelateerd aan $\mathbf{c}_k$ volgens

$$\mathbf{c}_k = \mathbf{Au}_k. \tag{20}$$

```
Choose an x_0.
Compute r_0 = b − Ax_0.
For k = 0, 1, 2, ..., k_max do
      stop if ‖r_k‖ ≤ tol
      u_k = r_k
      compute c_k = Au_k
      for i = 0, ..., k − 1 do
            β_{i+1} = c_i^T c_k / σ_i
            u_k = u_k − β_{i+1} u_i
            c_k = c_k − β_{i+1} c_i
      end for
      σ_k = c_k^T c_k ,  α_k = c_k^T r_k / σ_k
      x_{k+1} = x_k + α_k u_k
      r_{k+1} = r_k − α_k c_k
end for
```

ALG.1: *Generalized Conjugate Residuals met residuale nauwkeurigheid tol en maximaal aantal iteraties k_{max}.*

De residuen in GCR zijn, wat men noemt, *geconjugeerd*: $\mathbf{r}_{k+1} \perp \mathbf{A}\mathbf{r}_j$ voor $j \leq k$.
Dit volgt uit de volgende feiten:

(i) $\mathbf{r}_{k+1} \perp \mathbf{c}_j \quad (j \leq k)$,
(ii) de $\mathbf{c}_j$ spannen $\mathbf{A}\mathcal{K}_k(\mathbf{A}; \mathbf{r}_0)$ op
(iii) $\mathbf{A}\mathcal{K}_k(\mathbf{A}; \mathbf{r}_0) \subset \mathbf{A}\mathcal{K}_{k+1}(\mathbf{A}; \mathbf{r}_0)$ en
(iv) $\mathbf{r}_j \in \mathcal{K}_k(\mathbf{A}; \mathbf{r}_0) \quad (j \leq k)$.

Om eigenschappen van de iteratieve oplosmethoden te achterhalen plotten we op 10-log schaal de norm $\| \mathbf{r}_k \|$ van de residuen (dit levert een zogenaamd *convergentie-plaatje* (convergence history)).

Opdracht 9 (P).
Schrijf een (dubbele precisie) subroutine

`GCR(`$\mathbf{A}, \mathbf{b}, \mathbf{x}_0, k_{\max}, tol, \mathbf{x}$`)`

voor het oplossen van $\mathbf{Ax} = \mathbf{b}$ met GCR.
Als je je routines wilt testen kan je voor een door jou gekozen vector $\mathbf{x}$ het rechterlid $\mathbf{b}$ berekenen, immers $\mathbf{b} = \mathbf{Ax}$.

Opdracht 10 (E).
Los voor de testproblemen de grondwatervergelijking op. (Je kunt de contourlijnen plotten van de oplossing ψ of het snelheidsveld (u, v) (zie opdracht 6).

Opdracht 11 (E).
Onderzoek experimenteel de volgende vragen.

— Hangt het convergentie-gedrag essentieel af van de rechterlid vector $\mathbf{b}$ (brontermen en randfuncties) of van de beginbenadering $\mathbf{x}_0$?
— Hoe is de afhankelijkheid van de doorlaatbaarheidscoëfficiënten, en van de randvoorwaarden?
— Hoe neemt het aantal iteratiestappen toe als je $h_x = h_y$ verkleint? Hoe zit dat als je h_x vast houdt en h_y verkleint? Wat is de relatie met de hoeveelheid rekentijd?

Opgave C. Vergelijk de hoeveelheid geheugenruimte die nodig is om met GCR *een voldoend nauwkeurig antwoord te krijgen met de hoeveelheid geheugenruimte voor de direkte oplosmethode met Gauss eliminatie. Vergelijk ook de hoeveelheid werk (geef 'n orde van grootte als functie van n_x, n_y en k, waarbij k zo is dat $\| \mathbf{r}_k \| < tol$).*

Opgave D. Testprobleem V levert nogal wat problemen (zowel met betrekking tot de nauwkeurigheid van de discretisatie als met betrekking tot de convergentiesnelheid van de iteratieve methoden). Beschouw, om te begrijpen waar de moeilijkheid zit, de volgende twee 1-d problemen.

$$-\phi'' = f \ \textit{op}\ [0,1], \quad \phi(0) = 1, \quad \phi'(1) = 0,$$

waarbij f een bron is in $x = 1/3$ met capaciteit 1.

$$-\psi'' + (\phi'\psi)' = g \ \textit{op}\ [0,1], \quad \psi(0) = \psi(1) = 0,$$

waarbij g een put is in $x = 2/3$ met capaciteit 1 *en ϕ de oplossing van het vorige 1-d. probleem.*

Bepaal analytisch de oplossing van beide problemen.

6A. In een Krylov ruimte methode, met $\mathbf{x}_0 = 0$, behoort $\mathbf{x}_k$ tot $\mathcal{K}_k(\mathbf{A}; \mathbf{r}_0)$ en is $\mathbf{r}_k = \mathbf{r}_0 - \mathbf{A}\mathbf{x}_k$.
Het residu $\mathbf{r}_k$ is minimaal in $\mathcal{K}_{k+1}(\mathbf{A}; \mathbf{r}_0)$ als

$$\| \mathbf{r}_k \| = \min\{ \| \mathbf{r}_0 - \mathbf{s} \| \mid \mathbf{s} \in \mathbf{A}\mathcal{K}_k(\mathbf{A}; \mathbf{r}_0) \}.$$

Dit geldt precies dan als $\mathbf{r}_k \perp \mathbf{A}\mathcal{K}_k(\mathbf{A}; \mathbf{r}_0)$.

6B. GCR kan afbreken doordat $\mathbf{c}_k^T \mathbf{c}_k$ de waarde 0 kan krijgen. Door in het GCR-algorithme de regel

$$\text{“}\mathbf{u}_k = \mathbf{r}_k\text{”}$$

te vervangen door

$$\text{“}\mathbf{u}_k = \mathbf{c}_{k-1}\text{”}$$

en

$$\mathbf{c}_{-1} = \mathbf{r}_0$$

te nemen, vermijdt je het afbreekgevaar. Het algorithme dat je zo krijgt heet ORTHODIR.

6C. Je kunt nagaan dat iedere $\mathbf{u}_k$ na de “`for i=`”-loop een lineaire combinatie is van de $\mathbf{r}_j$ $(j \leq k)$. De scalairen voor de

combinatie kunnen berekend worden uit de β_j, Dus in plaats van de $\mathbf{u}_j$ te bewaren in GCR kunnen we ook de $\mathbf{r}_j$ bewaren. Er geldt ook

$$\mathbf{r}_{k+1} = \mathbf{r}_0 - \alpha_0\mathbf{c}_0 - \ldots - \alpha_k\mathbf{c}_k.$$

Dus kunnen we de $\mathbf{r}_j$ berekenen uit de $\mathbf{c}_i$ $(i < j)$ en $\mathbf{c}_{-1} \equiv \mathbf{r}_0$: we hoeven ook niet de $\mathbf{r}_j$ te bewaren. Kortom uit de $\mathbf{c}_i$ en de scalairen α_i en β_i kunnen we alle andere vectoren reconstrueren. Deze benadering levert een besparing in geheugenruimte op van 50% en ook, als je alles listig implementeert, een besparing van 50% in rekentijd.

Ook voor ORTHODIR (zie 6B) kan deze besparingsstrategie gevolgd worden. Het resulterend algorithme is robuust (breekt niet af) en is, onder de minimale-residu-methoden, een van de goedkoopste.

6D. Het GMRES algorithme, dat een orthogonale basis voor $\mathcal{K}_k(\mathbf{A};\mathbf{r}_0)$ berekent (en niet, zoals GCR doet, van $\mathbf{A}\mathcal{K}_k(\mathbf{A};\mathbf{r}_0)$) is in de praktijk populairder dan GCR. GMRES is even robuust en duur als de robuuste goedkope ORTHODIR variant die we in 6B gezien hebben. De bezuinigingsstrategiën, die we in §9 zullen bespreken, werken echter niet voor GMRES. We hebben daarom gekozen voor GCR. Het GCR algorithme ziet er ook beduidend eenvoudiger uit dan het GMRES algorithme.

7 Preconditioneren

7.1 Impliciet preconditioneren

De vector $\mathbf{u}_k$ wordt zoekrichting genoemd omdat GCR de verbetering van de benadering $\mathbf{x}_k$ in de richting $\mathbf{u}_k$ zoekt. In die richting is de verbetering zelfs optimaal in de zin dat het residu geminimaliseerd wordt: $\mathbf{c}_k = \mathbf{A}\mathbf{u}_k$ en

$$\|\mathbf{r}_{k+1}\| = \min_\alpha \|\mathbf{r}_k - \alpha\mathbf{A}\mathbf{u}_k\|$$

Naarmate de zoekrichting beter is zal $\mathbf{x}_{k+1}$ een betere benadering zijn van $\mathbf{x}$. Je kunt bewijzen dat de oplossing van de vergelijking $\mathbf{A}\mathbf{u} = \mathbf{r}_k$ de beste zoekrichting is (dan is $\mathbf{x}_{k+1} = \mathbf{x}$; zie opgave E). Het bepalen van deze beste zoekrichting is echter net zo moeilijk als het oplossen van het oorspronkelijke probleem. Maar natuurlijk mogen we wel hopen dat we efficiënt een zoekrichting kunnen vinden die (iets) beter is dan de "naieve" keuze $\mathbf{u}_k = \mathbf{r}_k$. Als $\mathbf{M}$ een $n \times n$ matrix is die $\mathbf{A}$ in enige mate benadert dan zal $\mathbf{u}_k = \mathbf{M}^{-1}\mathbf{r}_k$ wel beter zijn. Kunnen we $\mathbf{M}^{-1}\mathbf{r}_k$ ook nog efficiënt uitrekenen dan hebben we een mogelijkheid om ons convergentie-proces te versnellen. De matrix $\mathbf{M}$ met deze twee (heuristische) eigenschappen noemt men een *preconditioneerder*. De praktische berekening van $\mathbf{u}_k$ loopt gewoonlijk niet door $\mathbf{r}_k$ te vermenigvuldiging met de expliciet berekende inverse $\mathbf{M}^{-1}$, maar door $\mathbf{M}\mathbf{u}_k = \mathbf{r}_k$ op te lossen. De

```
Choose an x_0.
Compute r_0 = b − Ax_0.
For k = 0, 1, 2, ..., k_max do
    stop if ‖r_k‖ ≤ tol
    solve Mu_k = r_k
    compute c_k = Au_k
    for i = 0, ..., k − 1 do
        β_{i+1} = c_i^T c_k / σ_i
        u_k = u_k − β_{i+1} u_i
        c_k = c_k − β_{i+1} c_i
    end for
    σ_k = c_k^T c_k ,  α_k = c_k^T r_k / σ_k
    x_{k+1} = x_k + α_k u_k
    r_{k+1} = r_k − α_k c_k
end for
```

ALG.2: *Preconditioned* GCR.

gepreconditioneerde algorithmen ontstaan door de regel "$\mathbf{u}_k = \mathbf{r}_k$" te vervangen door "`solve` $\mathbf{M}\mathbf{u}_k = \mathbf{r}_k$" (zie ALG.2 voor het gepreconditioneerde GCR algorithme). De preconditioneerder wordt op deze manier *impliciet* verwerkt.

Het volgende argument moge duidelijk maken waarom preconditioneren nuttig (zo niet noodzakelijk is).
Stel dat $\mathbf{r}_0$ (als roosterfunctie geïnterpreteerd) maar alleen $\neq 0$ is in (x_{n_x}, y_j). Dan hebben alle vectoren in $\mathcal{K}_k(\mathbf{A};\mathbf{r}_0)$ de waarde 0 in de punten (x_1, y_j) voor $k < n_x$: een vermenigvuldiging met $\mathbf{A}$ brengt de niet-triviale waarden slechts een roosterpunt verder. Als de vector $\mathbf{x}$ in geen enkel roosterpunt $= 0$ is, lijken Krylov ruimte methoden niet zo snel een nauwkeurig antwoord te kunnen leveren. Door te preconditioneren ($\mathbf{M}^{-1}\mathbf{r}_0$) kunnen we wellicht de niet-nul in een stap over het hele rooster "uitsmeren".

Opgave E. Als je in GCR *de regel*

"$\mathbf{u}_k = \mathbf{r}_k$"

vervangt door

"`solve exactly` $\mathbf{A}\mathbf{u}_k = \mathbf{r}_k$"

heb je na één slag de exacte oplossing van $\mathbf{A}\mathbf{x} = \mathbf{b}$. *Zie dat in.*

7.2 Expliciet preconditioneren

Het k-de residu $\mathbf{r}_k$ van het gepreconditioneerde GCR algorithme is minimaal in de Krylov deelruimte

$\mathcal{K}_{k+1}(\mathbf{AM}^{-1}; \mathbf{r}_0)$. Dit volgt gemakkelijk uit de volgende alternatieve manier om tegen preconditioneren aan te kijken.

Merk op dat $\mathbf{x} = \mathbf{M}^{-1}\widetilde{\mathbf{x}}$ als $\widetilde{\mathbf{x}}$ de exacte oplossing is van

$$\mathbf{AM}^{-1}\widetilde{\mathbf{x}} = \mathbf{b}. \tag{21}$$

Passen we GCR toe op (21) en voeren we de vermenigvuldiging met $\mathbf{AM}^{-1}$ uit in twee stappen en vermenigvuldigen we de $\widetilde{\mathbf{x}}_k$ met $\mathbf{M}^{-1}$ dan krijgen we precies ons gepreconditioneerd GCR algorithme.

In (21) wordt (1) *expliciet rechts gepreconditioneerd*. We kennen ook het *expliciet links preconditioneren*:

$$\mathbf{M}^{-1}\mathbf{Ax} = \mathbf{M}^{-1}\mathbf{b}. \tag{22}$$

$\mathbf{M}^{-1}\mathbf{A}$ en $\mathbf{AM}^{-1}$ zijn *gepreconditioneerde matrices*.

Om uit (ongepreconditioneerd) GCR toegepast op het expliciet rechts gepreconditioneerd stelsel (21) de benaderingen $\mathbf{x}_k$ te halen moeten we de $\widetilde{\mathbf{x}}_k$ met $\mathbf{M}_k^{-1}$ vermenigvuldigen: GCR levert op het stelsel dat expliciet rechts gepreconditioneerd is "*gepreconditioneerde benaderende oplossingen*", terwijl expliciet links preconditioneren tot "*gepreconditoneerde residuen*" leidt. De impliciete variant levert daarentegen ware benaderingen en ware residuen.

Opgave F. *Veronderstel* $\mathbf{x}_0 = 0$.
Toon aan dat GCR met impliciet preconditioneren dezelfde residuen $\mathbf{r}_k^{impl}$ *levert als GCR met expliciet rechts preconditioneren:* $\mathbf{r}_k^{impl} = \mathbf{r}_k^{rechts}$.
Laat zien dat

$$\mathcal{K}_{k+1}(\mathbf{M}^{-1}\mathbf{A}; \mathbf{M}^{-1}\mathbf{b}) = \mathbf{M}^{-1}\mathcal{K}_{k+1}(\mathbf{AM}^{-1}; \mathbf{b})$$

Concludeer dat de gepreconditioneerde residuen $\mathbf{r}_k^{links}$*, die bij expliciet links preconditioneren door GCR geproduceerd worden, liggen in* $\mathbf{M}^{-1}$ *maal de ruimte waarin de residuen* $\mathbf{r}_k^{rechts}$ *liggen voor expliciet rechts preconditioneren. Is* $\mathbf{r}_k^{links} = \mathbf{M}^{-1}\mathbf{r}_k^{rechts}$*?*

7.3 Incomplete decomposities

Het vinden van een goede preconditioneerder is bijzonder belangrijk. Vaak "hangt" de convergentie, d.w.z. de mogelijkheid om het probleem op te lossen, op de preconditioneerder. We bespreken hieronder een paar preconditioneerders, die hoewel ze eenvoudig zijn toch vaak (soms in gegeneraliseerde vorm) gebruikt worden.

We gebruiken de volgende notatie:
$\mathbf{L}_A$ is de strikte benedendriehoek van $\mathbf{A}$;
$\mathbf{D}_A$ is de diagonaal van $\mathbf{A}$;
$\mathbf{U}_A$ is de strikte bovendriehoek van $\mathbf{A}$.
Dus

$$\mathbf{A} = \mathbf{L}_A + \mathbf{D}_A + \mathbf{U}_A. \tag{23}$$

De meest voor de hand liggende preconditioneerder waarmee ook nog eens efficiënt gerekend kan worden is de diagonaal van $\mathbf{A}$:

$$\mathbf{M} = \mathbf{D}_A. \tag{24}$$

De wat ingewikkeldere preconditioneerders $\mathbf{M}$, die we hier ook zullen onderzoeken, zijn van de vorm

$$\mathbf{M} = (\mathbf{D} + \mathbf{L}_A)\,\mathbf{D}^{-1}(\mathbf{D} + \mathbf{U}_A), \tag{25}$$

waarbij $\mathbf{D}$ een diagonaal matrix is die we nu nader zullen specificeren.

Merk op dat de vergelijking $\mathbf{Mu}_k = \mathbf{r}_k$ voor deze keuze van $\mathbf{M}$, als $\mathbf{D}$ bekend is, efficiënt op te lossen is. Verder is

$$\mathbf{R} \equiv \mathbf{A} - \mathbf{M} = \mathbf{D}_A - \mathbf{D} - \mathbf{L}_A\mathbf{D}^{-1}\mathbf{U}_A. \tag{26}$$

Vanwege onze tweede heuristische eis behoort $\mathbf{R} \approx 0$ is zekere zin. We geven een paar methoden om $\mathbf{D}$ te bepalen zodanig dat $\mathbf{R}$ klein te noemen is:
— de ILU benadering, $\mathrm{diag}(\mathbf{R}) = 0$;
— de MILU benadering, $\mathbf{R1} = 0$ waarbij $\mathbf{1}$ is de vector met op iedere coördinaat 1;
— RILU: combinaties van ILU en MILU.
Voor een parameter ω, berekenen we de diagonaal elementen $\mathbf{D}_k \equiv \mathbf{D}_{kk}$ van $\mathbf{D}$ iteratief ($\mathbf{D}_k$ uit $\mathbf{D}_j$ voor $j < k$).

$$\begin{aligned}\mathbf{D}_k = \mathbf{A}_{kk} &\\ - \left(\frac{\mathbf{A}_{k,k-1}\mathbf{A}_{k-1,k}}{\mathbf{D}_{k-1}} + \frac{\mathbf{A}_{k,k-n_x}\mathbf{A}_{k-n_x,k}}{\mathbf{D}_{k-n_x}}\right) &\\ - \omega\left(\frac{\mathbf{A}_{k,k-1}\mathbf{A}_{k-1,k+n_x-1}}{\mathbf{D}_{k-1}}\right. &\\ \left. + \frac{\mathbf{A}_{k,k-n_x}\mathbf{A}_{k-n_x,k-n_x+1}}{\mathbf{D}_{k-n_x}}\right).&\end{aligned} \tag{27}$$

Wellicht dat de beschrijving op het rooster wat duidelijker is. Als d_{ij} de rooster representatie is van $\mathbf{D}_k$ ($\mathbf{D}_{kk} = d_{ij}$ met $k = (j-1)\cdot n_x + i$) dan geldt:

$$\begin{aligned} d_{ij} &= \alpha_{ij}^{\mathrm{cent}} \\ &- \left(\frac{\alpha_{ij}^{\mathrm{west}}\alpha_{i-1,j}^{\mathrm{oost}}}{d_{i-1,j}} + \frac{\alpha_{ij}^{\mathrm{zuid}}\alpha_{i,j-1}^{\mathrm{noord}}}{d_{i,j-1}}\right) \\ &- \omega\left(\frac{\alpha_{ij}^{\mathrm{west}}\alpha_{i-1,j}^{\mathrm{noord}}}{d_{i-1,j}} + \frac{\alpha_{ij}^{\mathrm{zuid}}\alpha_{i,j-1}^{\mathrm{oost}}}{d_{i,j-1}}\right). \end{aligned} \tag{28}$$

Met $\omega = 0$ hebben we ILU en met $\omega = 1$ hebben we MILU.

Opdracht 12 (P).
Schrijf een procedure

$$\texttt{rilu}(\mathbf{D},\mathbf{A},\omega)$$

om $\mathbf{D}$ te bepalen met $\mathbf{D}$ als in (27). Je kunt hierbij ook (28) gebruiken.

Opdracht 13 (P).
Schrijf een procedure

$$\texttt{msolve}(\mathbf{D},\mathbf{A},\mathbf{r},\mathbf{u},p)$$

om $\mathbf{u}$ op te lossen uit $\mathbf{M}\mathbf{u} = \mathbf{r}$
met $\mathbf{M} = \mathbf{I}$ als $p = 0$,
$\mathbf{M} = \mathbf{D}_A$ als $p = 1$,
$\mathbf{M}$ als in (25) als $p = 2$.
In het laatste geval kan je in principe voor $\mathbf{D}$ iedere niet-singuliere diagonaal matrix nemen (mits van goede afmeting), dus, bijvoorbeeld, $\mathbf{D} = \mathbf{D}_A$ if de $\mathbf{D}$ uit `rilu`.

Opdracht 14 (P).
Bouw de preconditioneerder (op een impliciete manier) in in je GCR algorithme

$$\texttt{PGCR}(\mathbf{D},\mathbf{A},\mathbf{b},\mathbf{x}_0,k_{\max},p,tol,\mathbf{x}),$$

dat wil zeggen, bouw $\texttt{msolve}(\mathbf{D},\mathbf{A},\mathbf{r},\mathbf{u},p)$ in.

Opdracht 15 (E).
Onderzoek experimenteel hoe $\mathbf{M}$ de convergentie beïnvloedt. Laat ω variëren tussen 1 en 0. Vergelijk ook met $\mathbf{M} = \mathbf{I}$ en $\mathbf{M} = \mathrm{diag}(\mathbf{A})$.

Maak het verschil of $\mathbf{A}$ symmetrisch is?

Opgave G. Ga na dat voor het opslaan van $\mathbf{M}$ *als in* (25) *alleen een extra vector vereist is (nl. de coëfficiënten van de diagonaal van* $\mathbf{D}$*).*
Ga na dat het oplossen van een stelsel $\mathbf{M}\mathbf{u} = \mathbf{r}$ *met* $\mathbf{M}$ *als in* (25) *net zo duur is als vermenigvuldigen met* $\mathbf{M}$*:* $\widetilde{\mathbf{u}} = \mathbf{M}\mathbf{u}$.
Hoe duur is het expliciet berekenen van de inverse $\mathbf{M}\mathrm{inv} \equiv \mathbf{M}^{-1}$ *van* $\mathbf{M}$ *en het vervolgens uitrekenen van* $\mathbf{u}$ *door vermenigvuldiging:* $\mathbf{u} = \mathbf{M}\mathrm{inv}\,\mathbf{r}$*?*

7A. De ILU, MILU en RILU preconditioneringen zijn voorbeelden van zogenaamde incomplete LU-decomposities, die we hier voor een algemenere situatie kort zullen bespreken.

Bij incomplete decomposities kies je de plaatsen waarin je in de LU-decompositie "fill-in" toestaat: je kiest een deel $\mathcal{F}$ van de verzameling van paren (i,j) met $i,j \in \{1,\ldots,n\}$. $\mathcal{F}$ moet in ieder geval de diagonaal paren (i,i) bevatten. De verzameling van (i,j) waarvoor $\mathbf{A}_{ij} \neq 0$ is een populaire keuze voor $\mathcal{F}$. Voor een gekozen fill-in patroon $\mathcal{F}$ is een benedendriehoeks matrix $\mathbf{L} = (\mathbf{L}_{ij})$ met bovendriehoeks matrix $\mathbf{U}$ een *incomplete LU-decompositie* van $\mathbf{A}$ als voor $\mathbf{L}$, $\mathbf{U}$ en het product $\mathbf{M} = \mathbf{L}\mathbf{U}$ geldt

$$\mathbf{L}_{ij} = \mathbf{U}_{ij} = 0 \qquad \text{voor } i,j \notin \mathcal{F}$$
$$\& \quad \mathbf{R}_{ij} \equiv \mathbf{A}_{ij} - \mathbf{M}_{ij} = 0 \quad \text{voor } i,j \in \mathcal{F}, i \neq j.$$

De diagonaal elementen van $\mathbf{M}$ hoeven niet noodzakelijk samen te vallen met die van $\mathbf{A}$. Via de diagonaal probeert men extra eigenschappen te forceren. Door bijvoorbeeld $\mathbf{M1} = \mathbf{A1}$ te eisen hoopt men dat $\mathbf{M}^{-1}\mathbf{b}$ de oplossing $\mathbf{x}$ goed benadert als $\mathbf{x}$ niet veel varieert (men hoopt dat dan $\mathbf{x}$ — in ieder geval lokaal — min of meer konstant is: $\mathbf{x} \approx \alpha\mathbf{1}$).

```
M = A
for k = 1,...,n-1 do
  for i = k+1,...,n if (i,k) ∈ F do
    M_ik = M_ik/M_kk
    for j = k+1,...,n if (k,j) ∈ F do
      b = M_ik M_kj
      if (i,j) ∈ F then
        M_ij = M_ij - b
      else
        M_ii = M_ii - ω b
      end if
    end for
  end for
end for
```

ALG.3: *RILU(ω) met fill-in patroon* $\mathcal{F}$.

ALG.3 produceert een incomplete decompositie: als $\mathbf{L}\mathbf{U}$ de beoogde incomplete decompositie is van $\mathbf{A}$ met, voor een strikte benedendriehoeks matrix $\mathbf{L}'$, $\mathbf{L} = \mathbf{I} + \mathbf{L}'$ en $\mathbf{U}$ bovendriehoek dan is de strikte benedendriehoek van het resultaat $\mathbf{M}$ van ALG.3 precies $\mathbf{L}'$ en de bovendriehoek is $\mathbf{U}$: $\mathbf{L}_{ij} = \mathbf{M}_{ij}$ voor $i > j$ en $\mathbf{M}_{ij} = \mathbf{U}_{ij}$ voor $i \leq j$.
Voor $\omega = 0$ is $\mathrm{diag}(\mathbf{R}) = 0$ (ILU) en voor $\omega = 1$ is $\mathbf{R1} = 0$ (MILU). Voor andere ω's (RILU) werkt de preconditioneerder soms beter.

De decompositie in (25) met $\mathbf{D}$ uit (27) zijn, voor de vijfpunts discretisatie uit (15), precies de decomposities uit ALG.3 als we geen extra fill-in toelaten: d.w.z. als $\mathcal{F}$ uit de paren (i,j) bestaat waarvoor $\mathbf{A}_{ij} \neq 0$.

7.4 Efficiënt rekenen

Met gefactoriseerde $\mathbf{M}$, $\mathbf{M} = \mathbf{M}_1\mathbf{M}_2$, kan *tweezijdige expliciet* gepreconditioneerd worden:

$$\mathbf{M}_1^{-1}\mathbf{A}\mathbf{M}_2^{-1}\widetilde{\mathbf{x}} = \mathbf{M}_1^{-1}\mathbf{b}, \quad \mathbf{x} = \mathbf{M}_2^{-1}\widetilde{\mathbf{x}}. \tag{29}$$

We zullen hieronder zien dat hiermee voor de faktorisatie (25), met

$$\mathbf{M}_2 \equiv \mathbf{D}^{-1}(\mathbf{D} + \mathbf{U}_A) = \mathbf{I} + \mathbf{D}^{-1}\mathbf{U}_A, \tag{30}$$

en

$$\mathbf{M}_1 \equiv \mathbf{D} + \mathbf{L}_A = \mathbf{D}(\mathbf{I} + \mathbf{D}^{-1}\mathbf{L}_A), \tag{31}$$

bijzonder efficiënt gerekend kan worden.

Met $\widetilde{\mathbf{A}} \equiv \mathbf{D}^{-1}\mathbf{A}$ ziet (29) er uit als

$$(\mathbf{I} + \widetilde{\mathbf{L}}_A)^{-1}\widetilde{\mathbf{A}}\,(\mathbf{I} + \widetilde{\mathbf{U}}_A)^{-1}\,\widetilde{\widetilde{\mathbf{x}}} = \widetilde{\widetilde{\mathbf{b}}}, \qquad (32)$$

waarbij

$$\mathbf{x} = (\mathbf{I} + \widetilde{\mathbf{U}}_A)^{-1}\,\widetilde{\widetilde{\mathbf{x}}} \text{ en } \widetilde{\widetilde{\mathbf{b}}} \equiv (\mathbf{I} + \widetilde{\mathbf{L}}_A)^{-1}\mathbf{D}^{-1}\mathbf{b}.$$

Met $\widetilde{\mathbf{L}}_A$ en $\widetilde{\mathbf{U}}_A$ bedoelen we de strikte benedendriehoek, respectievelijk bovendriehoek van $\widetilde{\mathbf{A}}$.

Het expliciet gepreconditioneerde systeem in (32) kunnen we oplossen met (ongepreconditioneerd) GCR. De berekening van

$$\mathbf{c}_k = (\mathbf{I} + \widetilde{\mathbf{L}}_A)^{-1}\widetilde{\mathbf{A}}(\mathbf{I} + \widetilde{\mathbf{U}}_A)^{-1}\mathbf{u}_k \qquad (33)$$

kan zeer efficiënt uitgevoerd worden dankzij het feit dat

$$\widetilde{\mathbf{A}} = \widetilde{\mathbf{D}}_A + \widetilde{\mathbf{U}}_A + \widetilde{\mathbf{L}}_A.$$

Met $\Delta \equiv \widetilde{\mathbf{D}}_A - 2\,\mathbf{I}$ geldt (ga dat na)

$$(\mathbf{I} + \widetilde{\mathbf{L}}_A)^{-1}\widetilde{\mathbf{A}}(\widetilde{\mathbf{U}}_A + \mathbf{I})^{-1} =$$
$$(\mathbf{I} + \widetilde{\mathbf{L}}_A)^{-1} + (\mathbf{I} + \widetilde{\mathbf{U}}_A)^{-1} +$$
$$(\mathbf{I} + \widetilde{\mathbf{L}}_A)^{-1}\Delta(\mathbf{I} + \widetilde{\mathbf{U}}_A)^{-1}.$$

De volgende stappen leveren de $\mathbf{c}_k$ in (33) uit $\mathbf{u}_k$.

$$\begin{aligned}
&\texttt{solve } (\mathbf{I} + \widetilde{\mathbf{U}}_A)\,\mathbf{u}' = \mathbf{u}_k \\
&\texttt{compute } \mathbf{u}'' = \mathbf{u}_k + \Delta\mathbf{u}' \\
&\texttt{solve } (\mathbf{I} + \widetilde{\mathbf{L}}_A)\,\mathbf{u}''' = \mathbf{u}'' \\
&\mathbf{c}_k = \mathbf{u}' + \mathbf{u}''',
\end{aligned} \qquad (34)$$

Dit maakt de berekening van de $\mathbf{c}_k$ in (33) ongeveer net zo duur is als het berekenen van $\mathbf{A}\mathbf{u}_k$ (ga dat na). Deze aanpak noemt men de *Eisenstat truck*.

Een preconditioneerder $\mathbf{M}$ kan dus op verschillende manieren ingebouwd worden: impliciet, links – of rechts expliciet of tweezijdig. De manier van inbouwen heeft, voor niet al te sterk niet-symmetrische $\mathbf{A}$, nauwelijks effekt op het aantal iteratie stappen dat nodig is om het residue met een gegeven faktor te verkleinen (ga dit experimenteel na). Echter voor $\mathbf{M}$ van de vorm (25) kan de bereking per stap met de tweezijdige expliciete variant dankzij de Eisenstat truck zeer efficiënt gebeuren. De Eisenstat truck is dus geen prekonditonering maar een manier om met de preconditioneerder te rekenen.

Opdracht 16 (P).
Schrijf een subroutine

`pmv(D,A,u,c)`

die $\mathbf{c}_k = \mathbf{c}$ berekent uit $\mathbf{u}_k = \mathbf{u}$ volgens de Eisenstat truck (34).

Als je (al in het hoofdprogramma, of voor de GCR-loop) $\mathbf{A}$ vervangt door $\widetilde{\mathbf{A}}$ hoef je geen $\mathbf{D}$ mee te geven en is de `pmv` efficiënter!

Opdracht 17 (P).
Bouw de expliciete tweezijdige preconditionering, die de Eisenstat truck toelaat, in, in een GCR algorithme, bijvoorbeeld als:

EGCR($\mathbf{D}$,$\mathbf{A}$,$\mathbf{b}$,$\mathbf{x}_0$,$k_{\max}$,tol,$\mathbf{x}$).

Opdracht 18 (E).
Check de volgende bewering experimenteel.
Het aantal iteratiestappen van geprecondtioneerd GCR hangt sterk af van de gekozen preconditioneerder (zie opdracht 15), maar dat aantal hangt niet essentieel af van de *wijze* waarop gepreconditioneerd is (impliciet of expliciet; bij vaste $\mathbf{M}$ is het aantal stappen om een zekere residue-reductie te krijgen voor impliciet en expliciet preconditionering min of meer gelijk). De tijd per stap hangt wel af van de wijze van preconditionering.

Opgave H. Ga na dat (34) *inderdaad de* $\mathbf{c}_k$ *uit* (33) *oplevert en dat dit gebeurt min of meer tegen de kosten van de matrix-vector vermenigvuldiging* $\mathbf{A}\mathbf{u}_k$.

Opgave I. Laat zien dat met $\mathbf{M} = \mathbf{M}_1\mathbf{M}_2$,
$\mathbf{M}_1^{-1}\mathcal{K}_{k+1}(\mathbf{A}\mathbf{M}^{-1};\mathbf{b}) = \mathcal{K}_{k+1}(\mathbf{M}_1^{-1}\mathbf{A}\mathbf{M}_2^{-1};\mathbf{M}_1^{-1}\mathbf{b})$.

8 Korte recursies

Het aantal vectoren $\mathbf{u}_i$ en $\mathbf{c}_i$ dat opgeslagen moet worden in GCR is evenredig met k (nl. $2k+2$). Ook de kosten per stap nemen toe met k (nl. $2k+2$ AXPY en $k+2$ DOT om $\mathbf{r}_{k+1}$ te berekenen in de k-de stap).

In de volgende hoofdstukken bekijken we methoden waarvan alle stappen min of meer even efficiënt zijn. De nieuwe zoekrichtingen en de nieuwe residuen worden berekend uit een of een paar zoekrichtingen en residuen uit de vorige (of een paar) vorige stappen.

Het ligt voor de hand om te herstarten na een aantal iteratiestappen (d.w.z. na ℓ stappen beginnen we opnieuw maar nu met voor $\mathbf{x}_0$ de laatst verkregen benadering $\mathbf{x}_\ell$). In hoofdstuk 9 bekijken we de herstart strategie en strategiën die er aan verwand zijn: afkappen en nesten. Vaak werken die uitstekend, maar niet altijd. Het komt nogal eens voor

dat voor (snelle) convergentie toch een groot aantal vectoren $\mathbf{u}_i$ en $\mathbf{c}_i$ bewaard moet worden (denk aan $\ell > 50$) en oplossen is dan duur (of zelfs onmogelijk).

Voor symmetrische problemen blijkt de afkapstrategie wel uitstekend te werken (zie hoofdstuk 10) en ook nog inspiratie te geven voor efficiënte oplosmethoden voor niet-symmetrische problemen (zie hoofdstuk 12).

Opdracht 19 (P).
Werk met drie mensen samen. Een persoon implementeert en test herstart–, afkap–, en neststrategiën, een ander persoon doet hetzelfde voor de CG methode (voor symmetrische problemen) en Graig's methode voor niet-symmetrische problemen, de derde persoon concentreert zich op Bi-CGSTAB; al deze methoden worden in de volgende hoofdstukken besproken. Gebruik dezelfde testproblemen en gebruik de meest succesvolle preconditioneerders (welke dat is hangt af van het probleem!).

9 Herstarten, afkappen en nesten

Als het aantal iteratiestappen in GCR groter wordt neemt de hoeveelheid rekenwerk per iteratiestap toe. We kunnen deze toename op verschillende manieren proberen te beperken.

• Vervang de regel

"`for` $i = 0, \ldots, k-1$ `do`"

door

"`for` $i = j(k), \ldots, k-1$ `do`"

waarbij voor ℓ_1 en ℓ_2 (denk aan $\ell_1 = \ell_2 = 10$)
— $j(k) = \ell_1 \lfloor k/\ell_1 \rfloor$: we *herstarten* na ℓ_1 stappen;
— $j(k) = \max(0, k - \ell_2)$: we *kappen af* (truncation)
en neem alleen de laatste ℓ_2 vectoren $\mathbf{u}_j$ en $\mathbf{c}_j$ mee.
• Herstart na ℓ_1 stappen (of kap af) en vervang ook nog de regel

"$\mathbf{u}_k = \mathbf{r}_k$"

door

"`solve` $\mathbf{A}\mathbf{u}_k = \mathbf{r}_k$ `by` ℓ_3 `steps of GCR`"
(dus met `GCR(`$\mathbf{A}$`,`$\mathbf{r}_k$`,0,`ℓ_3`,0,`$\mathbf{u}_k$`)`'.

Als je, in plaats van $\mathbf{u}_k = \mathbf{r}_k$ de exacte oplossing zou nemen van $\mathbf{A}\mathbf{u}_k = \mathbf{r}_k$ dan zou $\mathbf{x}_{k+1}$ precies de oplossing zijn van $\mathbf{A}\mathbf{x} = \mathbf{b}$. Met ℓ_3 slagen GCR ($k_{\max} = \ell_3$) vindt je natuurlijk slechts een benadering van $\mathbf{A}^{-1}\mathbf{r}_k$, maar je mag hopen dat die een beter resultaat produceert dan $\mathbf{u}_k = \mathbf{r}_k$. In deze laatste aanpak passen we GCR "*genest*" toe.

Deze laatste methode wordt GMRESR(ℓ_1, ℓ_3) genoemd, hoewel GCRr(ℓ_1, ℓ_3) (GCR repeated) wellicht een betere beschrijving zou zijn (voor het verband tussen GCR en GMRES zie 6D). Met GCR(ℓ_1) geeft men de GCR variant aan waarin na iedere ℓ_1-ste stap herstart wordt.

Het moge duidelijk zijn dat "herstarten", "afkappen" of "nesten" tot goedkopere stappen leidt. In GCR worden minimale residuen geproduceerd en we weten dan ook zeker dat (tenminste in exacte arithmetiek) GCR op den duur (in ieder geval na n stappen) de oplossing van het lineaire probleem produceert. Met de modificaties zijn we die zekerheid kwijt. De kans op (snelle) convergentie is voor grotere ℓ groter.

Opdracht 20 (P).
Schrijf een subroutine

GCRr($\mathbf{A}$,$\mathbf{b}$,$\mathbf{x}_0$,$k_{\max}$,ℓ_1,ℓ_3,tol,$\mathbf{x}$)

voor het oplossen van $\mathbf{A}\mathbf{x} = \mathbf{b}$ met GCR waarin je de afkap–, herstart– en nestparameters kan kiezen:
herstart wordt na ℓ_1 slagen als $\ell_1 > 0$,
afgekapt wordt na $\ell_2 = |\ell_1|$ slagen als $\ell_1 < 0$,
en alle vectoren worden meegenomen als $\ell_1 = 0$.
Hou de benodigde hoeveelheid geheugenruimte beperkt.

Bouw ook de preconditionering in:

PGCRr($\mathbf{D}$,$\mathbf{A}$,$\mathbf{b}$,$\mathbf{x}_0$,$k_{\max}$,ℓ_1,ℓ_3,p,tol,$\mathbf{x}$).

Opdracht 21 (P).
Vergelijk het effect van de drie strategiën met GCR en onderling. Let op tijd en het aantal iteratiestappen[2] dat nodig is om een voldoend nauwkeurig resultaat te krijgen. Neem hiertoe voor de herstartstrategie $\ell_1 = 10$. Neem voor het afkappen de ℓ_2 en voor GMRESR de ℓ_1 en ℓ_3 zo dat de stappen van de drie strategiën gemiddeld ongeveer evenveel operaties (=rekentijd) vergen.

Merk op dat voor symmetrische matrices (zonder preconditioneerder of met symmetrische precondioneerder) afkappen met $\ell_2 > 0$ hetzelfde effekt heeft als afkappen met $\ell_2 = 1$: kontroleer dat $\beta_0 = \ldots = \beta_{k-1} = 0$. Hoe komt dat (zie hoofdstuk 10).

Opgave J. Inventariseer de hoeveelheid rekenwerk in de k-de stap van GCR, GCR(ℓ_1), *ℓ_2-truncated* GCR *en* GMRESR(ℓ_1,ℓ_3). *Om tot een eerlijk vergelijk te komen moet je in de laatste drie methoden de gemiddelde hoeveelheid werk berekenen, waarbij je moet middelen over ℓ_1, resp. $\ell_1 \cdot \ell_3$, stappen (waarom?).*

[2] Het begrip iteratiestap is (zeker voor GMRESR) wat vaag. Het aantal MVs is gewoonlijk een betere maat om de convergentie aan te relateren.

Kies een positief geheel getal K. Inventariseer (schat) de totale hoeveelheid werk die ieder van de vier methodes nodig heeft in K rm MVs.

Stel dat GMRESR(ℓ_1,ℓ_3) *voor iedere ℓ_1 en ℓ_3 waarvoor $\ell_1\ell_3 = K$ ongeveer dezelfde residu reductie haalt. Voor welke keuze van ℓ_1 (en daarmee van ℓ_3) is* GMRESR *het efficiënst?*

Inventariseer ook de hoeveelheid geheugenruimte die nodig is voor het rekenproces. Hou daarbij alleen rekening met lange vectoren en negeer scalairen (waarom is dat redelijk?).

9A. In onze versie van GMRESR stelden we voor $\mathbf{A}\mathbf{u}_k = \mathbf{r}_k$ benaderend op te lossen met GCR. Uiteraard kan je het probleem met iedere andere methode proberen op te lossen. Het deel van het algorithme waarin $\mathbf{A}\mathbf{u}_k = \mathbf{r}_k$ iteratief wordt opgelost noemt men de *binnen-loop*; het overige deel van het algorithme heet de *buiten-loop*. In de binnen-loop kan men bijvoorbeeld de methoden uit 6C of 6D gebruiken. Voor de buiten-loop zijn die methoden echter *on*geschikt en lijkt GCR de aangewezen keuze. (In ORTHODIR zou geen residu-vergelijking opgelost worden maar $\mathbf{A}\mathbf{u}_k = \mathbf{c}_{k-1}$.)

10 Symmetrische problemen

In geval $\mathbf{A}$ *symmetrisch* is ($\mathbf{A}^T = \mathbf{A}$) en $\mathbf{M} = \mathbf{I}$ (of in geval $\mathbf{A}\mathbf{M}^{-1}$ symmetrisch) kunnen we in GCR een aantal rekenstappen weglaten omdat we, om mathematische redenen, van te voren de uitkomst al weten. De resulterende methoden bespreken we in dit hoofdstuk.

10.1 CR

We nemen, om het idee beter te kunnen uit leggen, voor $\mathbf{M}$ de identiteit: $\mathbf{M} = \mathbf{I}$.
Dan geldt voor de GCR vectoren (na de "`for` $i = 0, \ldots$ `do`"-loop).

$$\mathbf{c}_k = \mathbf{A}\mathbf{r}_k - \beta_1\mathbf{c}_0 - \ldots - \beta_k\mathbf{c}_{k-1}$$

$$\text{met} \quad \beta_{j+1} = \mathbf{c}_j^T\mathbf{A}\mathbf{r}_k/\sigma_j.$$

Omdat

$$\mathbf{r}_k \perp \mathbf{A}\mathbf{c}_j \quad (j < k-1)$$

volgt hieruit dat, voor symmetrische $\mathbf{A}$,

$$\beta_j = 0 \qquad (j < k).$$

In stap k hoeven de $\beta_j \quad (j < k)$ niet meer uitgerekend te worden, en hebben we de $\mathbf{c}_j$ en $\mathbf{u}_j \quad (j < k-1)$ niet meer nodig: dit levert een aanzienlijke besparing.

`Choose an` $\mathbf{x}_0$`.`
`Compute` $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$`.`
`Put` $\mathbf{u}_{-1} = \mathbf{c}_{-1} = 0$`,` $\rho_{-1} = 1$`.`
`For` $k = 0, 1, 2, \ldots, k_{\max}$ `do`
  `stop if` $\|\mathbf{r}_k\| \leq tol$
  `solve` $\mathbf{M}\mathbf{u}_k = \mathbf{r}_k$
  `compute` $\mathbf{c}_k = \mathbf{A}\mathbf{u}_k$
  $\rho_k = \mathbf{r}_k^T\mathbf{c}_k$`,` $\beta_k = -\rho_k/\rho_{k-1}$
  $\mathbf{u}_k = \mathbf{u}_k - \beta_k\mathbf{u}_{k-1}$
  $\mathbf{c}_k = \mathbf{c}_k - \beta_k\mathbf{c}_{k-1}$
  $\sigma_k = \mathbf{c}_k^T\mathbf{c}_k$`,` $\alpha_k = \rho_k/\sigma_k$
  $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{u}_k$
  $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k\mathbf{c}_k$
`end for`

ALG.4: *Conjugate Residuals.*

Door nog wat meer met de symmetrie en de orthogonaliteit te spelen kan men laten zien dat

$$\beta_k = -\frac{\rho_k}{\rho_{k-1}}, \quad \alpha_k = \frac{\rho_k}{\sigma_k} \quad \text{met} \quad \rho_k = \mathbf{r}_k^T\mathbf{A}\mathbf{r}_k,$$

waardoor er per stap nog een DOT extra bespaard kan worden. Het resulterend algorithme heet CR (Conjugate Residuals) en staat, nu weer met preconditionering $\mathbf{M}$ (eventueel $\mathbf{M} \neq \mathbf{I}$), in ALG.4 (in ALG.5 staat hetzelfde algorithme nu zonder overbodige indices). De preconditioneerder moet zo zijn dat $\mathbf{A}\mathbf{M}^{-1}$ symmetrisch is. Van deze laatste eis kunnen we af als

ALG.5: *Conjugate Residuals*

`Choose an` $\mathbf{x}_0$`.`
`Compute` $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}$`.`
`Put` $\mathbf{u} = \mathbf{c} = 0$`,` $\mathbf{x} = \mathbf{x}_0$`,` $\rho = 1$`.`
`For` $k = 0, 1, 2, \ldots, k_{\max}$ `do`
  `stop if` $\|\mathbf{r}\| \leq tol$
  `solve` $\mathbf{M}\mathbf{u}' = \mathbf{r}$
  `compute` $\mathbf{c}' = \mathbf{A}\mathbf{u}'$
  $\rho' = \rho$`,` $\rho = \mathbf{r}^T\mathbf{c}'$`,` $\beta = -\rho/\rho'$
  $\mathbf{u} = \mathbf{u}' - \beta\,\mathbf{u}$
  $\mathbf{c} = \mathbf{c}' - \beta\,\mathbf{c}$
  $\sigma = \mathbf{c}^T\mathbf{c}$`,` $\alpha = \rho/\sigma$
  $\mathbf{x} = \mathbf{x} + \alpha\,\mathbf{u}$
  $\mathbf{r} = \mathbf{r} - \alpha\,\mathbf{c}$
`end for`

$\mathbf{M}$ symmetrisch en positief definiet is. We moeten dan met een aangepast inproduct werken (zie 10A).

10A. Als $\mathbf{M}$ symmetrisch en positief definiet is kunnen we een $\mathbf{M}^{-1}$-orthonormale basis van $\mathbf{c}_i$'s opbouwen, d.w.z. een basis die orthonormaal is t.o.v. het $\mathbf{M}^{-1}$-inproduct. We moeten dan $\mathbf{r}_k^T\mathbf{c}_k$ vervangen door $\mathbf{r}_k^T\mathbf{M}^{-1}\mathbf{c}_k$ en $\mathbf{c}_k^T\mathbf{c}_k$ door $\mathbf{c}_k^T\mathbf{M}^{-1}\mathbf{c}_k$. De eis dat $\mathbf{A}\mathbf{M}^{-1}$ symmetrisch moet zijn t.o.v. het standaard inproduct vervalt dan, want $\mathbf{A}\mathbf{M}^{-1}$ is precies symmetrisch t.o.v. dit $\mathbf{M}^{-1}$-inproduct (ga dat na).

10.2 CG

Als $\mathbf{A}$ positieve definiet is (we nemen weer even aan dat $\mathbf{M}=\mathbf{I}$) dan kan het standaard inproduct vervangen worden door een inproduct met $\mathbf{A}^{-1}$: in plaats van $\mathbf{r}_k^T\mathbf{c}_k$ in CR nemen we $\mathbf{r}_k\mathbf{A}^{-1}\mathbf{c}_k$, etc.. Omdat $\mathbf{A}^{-1}\mathbf{c}_k=\mathbf{u}_k$ (zie (20)) vinden we

$$\rho_k=\mathbf{r}_k^T\mathbf{A}^{-1}\mathbf{c}_k=\mathbf{r}_k^T\mathbf{u}_k$$

en

$$\sigma_k=\mathbf{c}_k^T\mathbf{A}^{-1}\mathbf{c}_k=\mathbf{c}_k^T\mathbf{u}_k.$$

Merk op dat, in geval $\mathbf{M}=\mathbf{I}$, $\rho_k=\mathbf{r}_k^T\mathbf{r}_k=\|\mathbf{r}_k\|^2$: we krijgen de norm van $\mathbf{r}_k$ zonder extra inproduct. Omdat we nu niet meer $\mathbf{c}_k$ nodig hebben voor de berekening van ρ_k kunnen we $\mathbf{c}_k$ uit $\mathbf{u}_k$ berekenen volgens $\mathbf{c}_k=\mathbf{A}\mathbf{u}_k$ (zie (20)) hetgeen nog eens een AXPY bespaart. Dit leidt tot het beroemde en uiterst efficiënte CG (Conjugate Gradient) algorithme, dat te vinden is in ALG.6. In deze versie is impliciete preconditionering toegevoegd.

Convergentie voor CG is alleen gegarandeerd als zowel $\mathbf{A}$ als de preconditioneerder $\mathbf{M}$ positief definiet zijn. De gepreconditioneerde matrix $\mathbf{A}\mathbf{M}^{-1}$ hoeft echter niet symmetrisch te zijn (zie 10B en 10A voor een verklaring).

Overigens convergeert CG gewoonlijk ook, en zelfs behoorlijk snel, als $\mathbf{A}$ en $\mathbf{M}$ niet positief definiet zijn, maar wel symmetrisch (zie 10C voor een 'soort' verklaring), maar er is geen garantie op convergentie. De convergentie kan grillig zijn en het process kan *afbreken* (*'break-down'*): zowel ρ_k als σ_k kunnen 0 zijn voordat $\|\mathbf{r}_k\|$ klein is.

Voor niet symmetrische matrices $\mathbf{A}$ (of $\mathbf{M}$) werkt CG practisch nooit.

In CR minimalizeren we, als $\mathbf{M}=\mathbf{I}$, het residu $\mathbf{r}_k$ in de Euclidische norm. In CG is $\mathbf{r}_k$ minimaal ten opzichte van de de zogenaamde $\mathbf{A}^{-1}$-norm:

$$\|\mathbf{r}_k\|_{\mathbf{A}^{-1}}\equiv\mathbf{r}_k^T\mathbf{A}^{-1}\mathbf{r}_k.$$

In ALG.6 beoordelen we de benadering met behulp van de Euclidische norm $\|\mathbf{r}_k\|$ van het residu. Ook al

```
Choose an x_0.
Compute r_0 = b − Ax_0.
Put u_{-1} = 0, ρ_{-1} = 1.
For k = 0, 1, 2, ..., k_max do
    stop if ‖r_k‖ ≤ tol
    solve Mu'_k = r_k
    ρ_k = r_k^T u'_k, β_k = −ρ_k/ρ_{k−1}
    u_k = u'_k − β_k u_{k−1}
    compute c_k = Au_k
    σ_k = u_k^T c_k, α_k = ρ_k/σ_k
    x_{k+1} = x_k + α_k u_k
    r_{k+1} = r_k − α_k c_k
end for
```

ALG.6: *Conjugate Gradients.*

produceert CG een residu dat minimaal is in de $\mathbf{A}^{-1}$-norm we kunnen de benaderingskwaliteit toch prima beoordelen in een andere norm als de Euclidische. Deze wellicht wat onlogische aanpak is voordelig: we hebben immers, zoals al opgemerkt, $\|\mathbf{r}_k\|$ voor niets, terwijl $\|\mathbf{r}_k\|_{\mathbf{A}^{-1}}$ niet gemakkelijk te berekenen is. Zie ALG.7, waarin weer de overbodige indices weggelaten zijn en vectoren overschreven worden als dat kan.

Opgave K. *Stel dat $\mathbf{M}=\mathbf{K}^T\mathbf{K}$ voor zekere $\mathbf{K}$. Dan is CG zonder preconditionering toegepast op de vergelijking $\mathbf{K}^{-T}\mathbf{A}\mathbf{K}^{-1}y=\mathbf{K}^{-T}\mathbf{b}$ (waarbij $\mathbf{K}^{-T}\equiv(\mathbf{K}^T)^{-1}$) equivalent met CG met impliciete preconditionering $\mathbf{M}$ toegepast op $\mathbf{A}\mathbf{x}=\mathbf{b}$. In welke zin geldt die equivalentie en waarom?*

ALG.7: *Conjugate Gradients.*

```
Choose an x_0.
Compute r = b − Ax_0.
Put u = 0, x = x_0, ρ = 1.
For k = 0, 1, 2, ..., k_max do
    solve Mc = r
    ρ' = ρ, ρ = r^T c, β = −ρ/ρ'
    stop if ρ ≤ tol^2
    u = c − β u
    compute c = Au
    σ = u^T c, α = ρ/σ
    x = x + α u
    r = r − α c
end for
```

Als $\mathbf{M}$ *symmetrisch en positief definiet is, is er altijd een* $\mathbf{K}$ *waarvoor* $\mathbf{M} = \mathbf{K}^T\mathbf{K}$. *Zie dat in. Merk ook op dat we voor de praktijk zo'n* $\mathbf{K}$ *niet hoeven te bepalen.*

10B. We kunnen de impliete preconditionering inbouwen voor het geval $\mathbf{M}$ symmetrisch en positief definiet is door op te merken dat $\mathbf{AM}^{-1}$ symmetrisch is t.o.v. het $\mathbf{M}^{-1}$-inproduct. Omdat $\mathbf{M}^{-1}(\mathbf{AM}^{-1})^{-1} = \mathbf{A}^{-1}$ is het $\mathbf{A}^{-1}$-inproduct gelijk aan het $(\mathbf{AM}^{-1})^{-1}$-$\mathbf{M}^{-1}$-inproduct. Dus impliciet geprecondi-tioneerd CG volgt uit impliciet geprecondi-tioneerd CR t.o.v. het $\mathbf{M}^{-1}$-inproduct (zie 10A).

10C. Als zowel $\mathbf{A}$ als $\mathbf{M}$ symmetrisch zijn, dan kan men aantonen dat

$$\mathbf{r}_k, \mathbf{A}\mathbf{u}_k \perp \mathbf{M}^{-1}\mathcal{K}_k(\mathbf{AM}^{-1}; \mathbf{r}_0).$$

Om de convergentie te verklaren kan men opmerken dat bij het vorderen van het iteratie-proces, d.w.z. bij toenemende k, de 'schaduw' ruimte $\mathbf{M}^{-1}\mathcal{K}_k(\mathbf{AM}^{-1}; \mathbf{r}_0)$ groeit en er steeds 'minder plaats' over blijft voor $\mathbf{r}_k$.

10.3 Preconditioneren

De preconditioneringen die we in 7.3 gezien hebben zijn symmetrisch als $\mathbf{A}$ dat is ($\mathbf{M} = \mathbf{M}^T$; ga dat na). Echter $\mathbf{AM}^{-1}$ is niet meer symmetrisch en CG of CR is niet meer toepasbaar op het expliciet geprecondi-tioneerd stelsel (d.w.z. een theoretische rechtvaardiging ontbreekt). We zullen zien dat, we met een tweezijdige variant de symmetrie kunnen herstellen.

Stel $\mathbf{A} = \mathbf{A}^T$ en beschouw $\mathbf{M}$ als in (25). Dan is $\mathbf{L}_A = \mathbf{U}_A^T$. Stel dat alle diagonaal elementen van $\mathbf{D}$ positief zijn. Dan kunnen we uit deze elementen de wortel trekken. Laat $\mathbf{D}^{1/2}$ de resulterende diagonaal matrix zijn. We kunnen nu weer preconditioneren in twee stappen. Alleen de eerste stap voeren we, om symmetrie te behouden, een beetje anders uit dan in 7.4.
De diagonaal-preconditionering gaat als volgt:

$$\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\widetilde{\mathbf{x}} = \mathbf{D}^{-1/2}\mathbf{b} \text{ met } \mathbf{x} = \mathbf{D}^{-1/2}\widetilde{\mathbf{x}}.$$

We preconditioneren het resulterend systeem met matrix $\widetilde{\mathbf{A}} \equiv \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ nog verder. We voorzien daartoe weer alle grootheden die geassocieerd zijn aan het diagonaal-geprekonditioneerd probleem van een $\widetilde{\ }$ en we gaan verder te werk als in 7.4.
Het resulterend systeem

$$(\mathbf{I} + \widetilde{\mathbf{L}}_A)^{-1}\widetilde{\mathbf{A}}\,(\mathbf{I} + \widetilde{\mathbf{U}}_A)^{-1}\widetilde{\widetilde{\mathbf{x}}} = \widetilde{\widetilde{\mathbf{b}}}$$

is symmetrisch (ga dat na) en CG (of CR) kan toegepast worden.

Opdracht 22 (P).

Implementeer gepreconditioneerd CG voor symmetrische problemen met symmetrische preconditionering:
PCG($\mathbf{D}$,$\mathbf{A}$,$\mathbf{b}$,$\mathbf{x}_0$,$k_{\max}$,p,tol,$\mathbf{x}$).

11 Graig's methode

Als $\mathbf{A}$ niet symmetrisch is kunnen we CR en CG toch gebruiken. Immers, als we de oplossing $\widetilde{\mathbf{x}}$ van

$$\mathbf{A}\mathbf{A}^T\widetilde{\mathbf{x}} = \mathbf{b} \tag{35}$$

hebben, dan kunnen we met $\mathbf{x} = \mathbf{A}^T\widetilde{\mathbf{x}}$ ook de oplossing van $\mathbf{Ax} = \mathbf{b}$ berekenen en $\mathbf{AA}^T$ is symmetrisch. Als $\widetilde{\mathbf{x}}_k$ de benadering is van $\widetilde{\mathbf{x}}$ in stap k van CG voor (35) dan is $\mathbf{x}_k \equiv \mathbf{A}^T\widetilde{\mathbf{x}}_k$ een benadering voor $\mathbf{x}$. Evenzo kan $\mathbf{u}_k \equiv \mathbf{A}^T\widetilde{\mathbf{u}}_k$ gezien worden als een zoekrichting voor $\mathbf{x}_k$ als $\widetilde{\mathbf{u}}_k$ een CG-zoekrichting is voor $\widetilde{\mathbf{x}}_k$. Verdrijven we de $\widetilde{\ }$'s in CG voor (35) dan vinden we de methode van *Graig*: zie ALG.8 (overbodige indices zijn weer onderdrukt). Merk op dat nu

$$\|\mathbf{b} - \mathbf{A}\mathbf{x}_k\|_{\left(\mathbf{AA}^T\right)^{-1}} = \|\,\mathbf{x} - \mathbf{x}_k\|\,:$$

dus in Graig's methode wordt de fout $\mathbf{x} - \mathbf{x}_k$ geminimaliseerd in de Euclidische norm (in een Krylov deelruimte voortgebracht door $\mathbf{AA}^T$).
Voor Graig's methode hebben we ook een routine nodig die met $\mathbf{A}^T$ vermenigvuldigd.
Graig's methode kan ook gepreconditioneerd worden. De preconditionering moet dan wel expliciet gebeuren (links, rechts, tweezijdig?). Merk op dat je dan ook de preconditioneerder met transporteren ($\mathbf{M}^T$).

ALG.8: *Graig's methode.*

```
Choose an x_0.
Compute r = b − Ax_0.
Put u = 0, x = x_0, ρ = 1.
For k = 0, 1, 2, ..., k_max do
      ρ' = ρ, ρ = r^T r, β = −ρ/ρ'
      stop if ρ ≤ tol^2
      compute c = A^T r
      u = c − β u
      compute c = Au
      σ = u^T u, α = ρ/σ
      x = x + α u
      r = r − α c
end for
```

Opdracht 23 (P).
Implementeer Graig's methode met preconditionering:
PGRAIG($\mathbf{D}$,$\mathbf{A}$,$\mathbf{b}$,$\mathbf{x}_0$,$k_{\max}$,p,tol,$\mathbf{x}$).

Opgave L. Een alternatief voor (35) *zijn de zogenaamde* normaal vergelijkingen*:*

$$\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b}. \tag{36}$$

Wat minimaliseert CG als we CG hierop toepassen?

12 Bi-orthogonale residuen

In dit hoofdstuk bekommeren we ons niet al te veel om $\mathbf{x}_k$ (we zullen de k in $\mathbf{x}_k$ weglaten). Net als in de vorige hoofdstukken, zullen we ernaar streven om $\mathbf{r}_k$ zo efficiënt mogelijk naar 0 te laten gaan en zullen we telkens $\mathbf{r}_k$ korrigeren met een vector van de vorm $-\mathbf{A}\mathbf{v}_k$, waarbij we de $\mathbf{v}_k$ expliciet kennen. We werken $\mathbf{x}_k$ gelijktijdig bij met $+\mathbf{v}_k$:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{v}_k, \quad \mathbf{r}_{k+1} = \mathbf{r}_k - \mathbf{A}\mathbf{v}_k. \tag{37}$$

Dan geldt (als we aannemen dat de evaluatiefouten verwaarloosbaar zijn) $\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k$.

GCR produceert de kleinste residuen in de Krylov deelruimten. Helaas worden de stappen van GCR duurder naarmate het aantal iteratie-stappen toeneemt. Dat is niet het geval voor CG: CG produceert minimale residuen met zeer goedkope stappen. Helaas is, zoals we al opmerkten, convergentie voor CG alleen gegarandeerd voor positief definiete problemen en blijkt CG niet te werken voor niet-symmetrische problemen. Aan deze situatie valt weinig te verbeteren: er is bewezen dat, voor niet-symmetrische problemen, geen enkele Krylov deelruimte methode met goedkope stappen minimale residuen produceert. Voor niet symmetrische problemen moeten we dus of toenemend dure stappen voor lief nemen (als in GCR) of residuen die kwalitatief minder zijn (als in GCR(ℓ_1) en GMRESR(ℓ_1,ℓ_3)).
Een andere aanpak om met goedkope stappen te werken streeft niet naar minimale residuen, maar naar residuen $\mathbf{r}_k$ die loodrecht staan op k-dimensionale "schaduw ruimten". Men hoopt op convergentie omdat bij toenemende k de $\mathbf{r}_k$ geconstrueerd worden in steeds kleinere ruimten. Deze aanpak zullen we hier bespreken.

De residuen $\mathbf{r}_k$ worden loodrecht gezet op de *schaduw Krylov deelruimten* $\mathcal{K}_k(\mathbf{A}^T;\widetilde{\mathbf{r}}_0)$ voortgebracht door de getransponeerde $\mathbf{A}^T$ van $\mathbf{A}$ en een schaduw beginresidu $\widetilde{\mathbf{r}}_0$. We spreken over *bi-orthogonaliteit*. Het schaduw residu dient gekozen te worden. Vaak kiest men hiervoor

$$\widetilde{\mathbf{r}}_0 = \mathbf{r}_0 \quad \text{of} \quad \widetilde{\mathbf{r}}_0 = \text{random}.$$

Het pseudo-algorithme ALG.9 produceert die orthogonale residuen:
$\mathbf{r}_k$ en zoekrichting $\mathbf{u}_k$ behoren tot $\mathcal{K}_{k+1}(\mathbf{A};\mathbf{r}_0)$ en
$\mathbf{r}_k$ en $\mathbf{c}_k \equiv \mathbf{A}\mathbf{u}_k$ staan loodrecht op $\mathcal{K}_k(\mathbf{A}^T;\widetilde{\mathbf{r}}_0)$.
Deze loodrechtheid wordt in het algorithme bewerkstelligd door de coëfficiënten α_{k-1} en β_k zo te bepalen dat $\mathbf{r}_k \perp \widetilde{\mathbf{r}}_{k-1}$ en $\mathbf{c}_k \perp \widetilde{\mathbf{r}}_{k-1}$ waarbij de $\widetilde{\mathbf{r}}_j$ $(j = 0,\ldots,k-1)$ de schaduw ruimte $\mathcal{K}_k(\mathbf{A}^T;\widetilde{\mathbf{r}}_0)$ opspannen. Loodrechtheid op de overige schaduw basis vectoren $\mathbf{r}_j$ $(j < k-1)$ volgt met een inductie argument waarin ondermeer gebruikt wordt dat

$$\mathbf{A}\mathbf{r}_k \perp \mathcal{K}_{k-1}(\mathbf{A}^T;\widetilde{\mathbf{r}}_0)$$

hetgeen volgt uit

$$\mathbf{r}_k \perp \mathbf{A}^T\mathcal{K}_{k-1}(\mathbf{A}^T;\widetilde{\mathbf{r}}_0) \subset \mathcal{K}_k(\mathbf{A}^T;\widetilde{\mathbf{r}}_0).$$

De vraag is natuurlijk hoe je de schaduw basis vectoren $\widetilde{\mathbf{r}}_k$ moet kiezen.

Opgave M. Bewijs dat $\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k$ *voor ieder* k *als dit correct is voor* $k = 0$ *en we* $\mathbf{x}_k$ *en* $\mathbf{r}_k$ *bijwerken als in* (37).

```
Choose an x_0 and an r̃_0.
Compute r_0 = b − Ax_0.
Put u_{−1} = 0, x = x_0, σ_{−1} = ϑ_{−1} = 1.
For k = 0, 1, 2, ... do
    stop if ‖r_k‖ ≤ tol
    ρ_k = r̃_k^T r_k, β_k = −ρ_k/(ϑ_{k−1}σ_{k−1})
    u_k = r_k − β_k u_{k−1}
    compute c_k = Au_k
    σ_k = r̃_k^T c_k, α_k = ρ_k/σ_k
    x = x + α_k u_k
    r_{k+1} = r_k − α_k c_k
    select a r̃_{k+1} ∈ K_{k+2}(A^T; r̃_0)
    such that for some ϑ_k ≠ 0
    r̃_{k+1} − ϑ_k A^T r̃_k ∈ K_{k+1}(A^T; r̃_0)
end for
```

ALG.9: *Bi-orthogonal residuals.*

Opgave N. *Met* $\mathbf{u}_k = \mathbf{r}_k - \beta_k \mathbf{u}_{k-1}$ *is* $\mathbf{A}\mathbf{u}_k \perp \widetilde{\mathbf{r}}_{k-1}$ *precies dan als*

$$\beta_k = \frac{\widetilde{\mathbf{r}}_{k-1}^T \mathbf{A}\mathbf{r}_k}{\widetilde{\mathbf{r}}_{k-1}^T \mathbf{A}\mathbf{u}_{k-1}}.$$

Bewijs dit en toon ook aan dat

$$\widetilde{\mathbf{r}}_{k-1}^T \mathbf{A}\mathbf{r}_k = (\mathbf{A}^T \widetilde{\mathbf{r}}_{k-1})^T \mathbf{r}_k = \frac{1}{\vartheta_{k-1}} \widetilde{\mathbf{r}}_k^T \mathbf{r}_k.$$

12.1 BiCG

In het BiCG (Bi-Conjugate Gradient) algorithme worden voor de schaduw residuen de residuen van de schaduw vergelijking $\mathbf{A}^T \widetilde{\mathbf{x}} = \widetilde{\mathbf{r}}_0$ gekozen, die op 'n zelfde manier berekend kunnen worden als de echte residuen:

$$\begin{aligned} \widetilde{\mathbf{u}}_k &= \widetilde{\mathbf{r}}_k - \beta_k \widetilde{\mathbf{u}}_{k-1} \\ \widetilde{\mathbf{c}}_k &= \mathbf{A}^T \widetilde{\mathbf{u}}_k \\ \widetilde{\mathbf{r}}_{k+1} &= \widetilde{\mathbf{r}}_k - \alpha_k \widetilde{\mathbf{c}}_k \end{aligned}$$

waarbij β_k en α_k als in ALG.9. Verder is dan

$$\vartheta_k = -\alpha_k.$$

Gewoonlijk convergeert Bi-CG. De convergentie is echter vaak onregelmatig en het proces kan zelfs *afbreken* (*'break-down'*): ρ_k and σ_k kunnen 0 worden. Echt afbreken komt zelden voor maar bijna afbreken ($\rho_k \approx 0$ of $\sigma_k \approx 0$) gebeurt wel en dat leidt tot pieken in de convergentie historie. Hierdoor worden rekenfouten dramatisch vergroot, gaat de convergentie snelheid verloren en is de uitkomst *onnauwkeurig* (d.w.z. $\|\mathbf{r}_k\|$ is klein terwijl $\|b - Ax_k\|$ helemaal niet klein is).

Merk op dat we de benaderingen $\widetilde{\mathbf{x}}_k$ van de oplossing van de schaduw vergelijking niet nodig hebben en ook niet berekenen. Desondanks wordt er in BiCG toch nog flink wat rekenwerk geïnvesteerd in het oplossen van een vergelijking die ons in feite niets interesseert (de schaduw residuen worden wel expliciet bepaald).

Opgave O. *Als* $\mathbf{A}^T = \mathbf{A}$ *en* $\widetilde{\mathbf{r}}_0 = \mathbf{r}_0$ *is BiCG precies CG* ($\widetilde{\mathbf{r}}_k = \mathbf{r}_k$, $\widetilde{\mathbf{u}}_k = \mathbf{u}_k$ *en* $\widetilde{\mathbf{c}}_k = \mathbf{c}_k$). *Ga dat na.*

12.2 Bi-CGSTAB

We zouden graag zien dat iedere matrix vermenigvuldiging (ook met $\mathbf{A}^T$) tot reductie van het residu leidt. De volgende opmerking helpt ons.

Het schaduw residu $\widetilde{\mathbf{r}}_k$ is, voor zeker polynoom q_k van graad k, van de vorm $q_k(\mathbf{A}^T)\,\widetilde{\mathbf{r}}_0$. Dus

$$\rho_k = \widetilde{\mathbf{r}}_k^T \mathbf{r}_k = \widetilde{\mathbf{r}}_0^T\, q_k(\mathbf{A})\, \mathbf{r}_k \tag{38}$$

en

$$\sigma_k = \widetilde{\mathbf{r}}_k^T \mathbf{A}\mathbf{u}_k = \widetilde{\mathbf{r}}_0^T\, \mathbf{A}\, q_k(\mathbf{A})\, \mathbf{u}_k. \tag{39}$$

Hierin hebben we $\mathbf{A}$ en $q_k(\mathbf{A})$ verwisseld.
We schrijven $\mathbf{Q}_k \equiv q_k(\mathbf{A})$ en stellen voor $\mathbf{Q}_k\mathbf{u}_{k-1}$, en $Q_k\mathbf{r}_k$ met bijbehorende $\mathbf{x}$ te berekenen en $\mathbf{Q}_k$ zo te kiezen dan $\|\mathbf{Q}_k\mathbf{r}_k\|$ klein is (kleiner dan $\|\,\mathbf{r}_k\|$). De berekening verloopt in twee delen.
• In het eerste deel, het BiCG deel, vermenigvuldigen we de BiCG stappen met $\mathbf{Q}_k$ (vergl. ALG.9):

$$\begin{aligned} \mathbf{Q}_k\mathbf{u}_k &= \mathbf{Q}_k\mathbf{r}_k - \beta_k \mathbf{Q}_k\mathbf{u}_{k-1} \\ \mathbf{Q}_k\mathbf{c}_k &= \mathbf{A}\mathbf{Q}_k\mathbf{u}_k \\ \mathbf{Q}_k\mathbf{r}_{k+1} &= \mathbf{Q}_k\mathbf{r}_k - \alpha_k \mathbf{Q}_k\mathbf{c}_k. \end{aligned}$$

De ρ_k en de σ_k — en dus de β_k en α_k — zijn nu berekenbaar volgens (38) en (39).
• In het tweede deel, krikken we de graad van q_k op en berekenen we $\mathbf{Q}_{k+1}\mathbf{u}_k$, $\mathbf{Q}_{k+1}\mathbf{r}_{k+1}$ met bijbehorende $\mathbf{x}$ ($= \widehat{\mathbf{x}}$) uit $\mathbf{Q}_k\mathbf{u}_k$, $\mathbf{Q}_k\mathbf{r}_{k+1}$ en $\mathbf{x}$.
In CGS (CG Squared) worden de polynomen op een BiCG manier bepaald (doen wij hier niet).
In Bi-CGSTAB (BiCG stabilized) wordt

$$\mathbf{Q}_{k+1} = (\mathbf{I} - \omega_k \mathbf{A})\, \mathbf{Q}_k$$

gekozen met optimale reële scalair ω_k. Dan is

$$\vartheta_k = \omega_k$$

en

$$\begin{aligned} \mathbf{Q}_{k+1}\mathbf{u}_{k+1} &= (\mathbf{I} - \omega_k \mathbf{A})\, \mathbf{Q}_k\mathbf{u}_k \\ \mathbf{Q}_{k+1}\mathbf{r}_{k+1} &= (\mathbf{I} - \omega_k \mathbf{A})\, \mathbf{Q}_k\mathbf{r}_k. \end{aligned}$$

Met optimale ω_k bedoelen we dat

$$\|(\mathbf{I} - \omega_k \mathbf{A})\, \mathbf{Q}_k\mathbf{r}_k\|$$

minimaal is.

Schrijven we $\mathbf{u}_{k-1}$, $\widehat{\mathbf{u}}_k$, $\mathbf{c}_k$, $\mathbf{r}_k$ en $\widehat{\mathbf{r}}_k$ in plaats van achtereenvolgens $\mathbf{Q}_k\mathbf{u}_{k-1}$, $\mathbf{Q}_k\mathbf{u}_k$, $\mathbf{A}\mathbf{Q}_k\mathbf{u}_k$, $\mathbf{Q}_k\mathbf{r}_k$, $\mathbf{Q}_k\mathbf{r}_{k+1}$ en verwijderen we de indices k dan krijgen we het Bi-CGSTAB algorithme ALG.10 (met preconditioneerder $\mathbf{M}$. In ALG.10 kunnen we nog meer geheugenruimte besparen door de $\widehat{\ }$ te schrappen).

Het werken met een minimalizerende ω_k heeft nog een bijkomend voordeel. De convergentie van Bi-CGSTAB is niet alleen sneller in termen van residureductie per matrix-vector vermenigvuldiging (vaak ongeveer twee maal), maar ook veel 'gladder' dan die van Bi-CG. De methode is hierdoor minder gevoelig voor effecten van rekenfouten. Dit verklaart

```
Choose an x_0 and an r̃_0
Compute r = b − Ax_0.
Put u = 0, x = x_0, σ = ω = 1.
For k = 0, 1, 2, ..., k_max do
    stop if ‖r‖ ≤ tol
    ρ = r̃_0^T r, β = −ρ/(ωσ)
    solve Mû = r
    û = û − β u
    compute c = Aû
    σ = r̃_0^T c, α = ρ/σ
    x̂ = x + α û
    r̂ = r − α c
    solve Ms' = r̂
    solve Mĉ = c
    compute s = As'
    ω = s^T r̂ / s^T s
    u = û − ω ĉ
    x = x̂ + ω s'
    r = r̂ − ω s
end for
```

ALG.10: Bi-CGSTAB.

het woord 'stabilized' in de naam van de methode. Maar net als Bi-CG kan Bi-CGSTAB afbreken. Bi-CGSTAB heeft zelfs nog een extra afbreek mogelijkheid: niet alleen ρ en σ kan 0 zijn, maar ook ω.

Opdracht 24 (P).

Implementeer gepreconditioneerd Bi-CGSTAB:

PBCGSTAB($\mathbf{D}$,$\mathbf{A}$,$\mathbf{b}$,$\mathbf{x}_0$,$k_{\max}$,p,tol,$\mathbf{x}$).

Ga zo efficiënt mogelijk met geheugenruimte om en herrangschik het algorithme zo dat per stap maar twee "$\mathbf{M}$-solve"s nodig zijn.

12A. Bi-CGSTAB vindt vaak uiterst efficiënt de oplossing van de lineaire vergelijking. Soms *stagneert* Bi-CGSTAB (d.w.z. het residu wordt gedurende een flink aantal opeenvolgende stappen niet veel kleiner). Dat kan een aantal oorzaken hebben. Ieder van de drie coëfficiënten ρ_k, σ_k of ω_k kan 0 zijn. Precies gelijk aan 0 komt in de praktijk zelden voor (kans 0%), maar bijna nul kan al vervelend zijn en grote evaluatie fouten veroorzaken. De situatie waarin $\omega_k \approx 0$ is kan verholpen worden door $q_k(t)$ niet te schrijven als een product van k eerste graads polynomen $(1-\omega_k t)$ maar (voor $k = m\ell$) als een product van m ℓ-de graads polynomen. Het resulterend algorithme BiCGstab(ℓ) staat in ALG.11.
Voor $\ell = 2$ of $\ell = 4$ werkt BiCGstab(ℓ) vaak (veel) beter dan Bi-CGSTAB.

Voor $\ell > 1$ kan de preconditionering niet efficiënt *impliciet* verwerkt worden in BiCGstab(ℓ) en wordt aanbevolen om de

```
Choose an x_0 and an r̃_0.
Compute r_0 = b − Ax_0.
Put u_0 = 0, x = x_0, σ = ω_ℓ = 1
and U = [u_0], R = [r_0].
For k = 0, 1, 2, ..., k_max do
    stop if ‖r_0‖ ≤ tol
    σ = −ω_ℓ σ
    for j = 1, ..., ℓ do
        ρ = r̃_0^T r_{j−1}, β = ρ/σ
        U = R − β U
        compute u_j = A u_{j−1}
        U = [U, u_j]
        σ = r̃_0^T u_j, α = ρ/σ
        x = x + α u_0
        R = R − α [u_1, ..., u_j]
        compute r_j = A r_{j−1}
        R = [R, r_j]
    end for
    solve z = (ω_1, ..., ω_ℓ)^T
        from (S^T S) z = S^T r_0
        where S := [r_1, ..., r_ℓ]
    u_0 = u_0 − [u_1, ..., u_ℓ] z
    x = x + [r_0, ..., r_{ℓ−1}] z
    r_0 = r_0 − [r_1, ..., r_ℓ] z
    U = [u_0], R = [r_0]
end for
```

ALG.11: BiCGstab(ℓ); *$\mathbf{R}$ en $\mathbf{U}$ zijn matrices: aan het eind van 'n "`for` j"-loop is* $\mathbf{R} = [\mathbf{r}_0, \ldots, \mathbf{r}_j]$ *en* $\mathbf{U} = [\mathbf{u}_0, \ldots, \mathbf{u}_j]$.

preconditionering *expliciet* toe te passen.

Voor $\ell = 1$ hebben we precies Bi-CGSTAB: Bi-CGSTAB-=BiCGstab(1). Om dit in te zien kunnen we de $\widehat{\ }$'s in Bi-CGSTAB verwijderen en $\mathbf{M} = \mathbf{I}$ nemen. Dan hebben de grootheden in beide algorithmen de volgende relatie (BiCGstab(1) grootheden eerst): $\mathbf{r}_0 = \mathbf{r}$, $\mathbf{r}_1 = \mathbf{s}$, $\mathbf{u}_0 = \mathbf{u}$, $\mathbf{u}_1 = \mathbf{c}$, $z = \omega_1 = \omega$).

13 Tot slot

De nadruk in dit praktikum heeft gelegen op het oplossen van grote lineaire stelsels vergelijkingen. In het thema "grondwatervergelijkingen" passen een groot aantal andere aspekten die hier niet aan de orde geweest zijn:

• 3-d Model met zwaartekracht term.
• Tijdsafhankelijk model.
• Modelleren bron termen.
• Oneindig grote snelheden zijn niet mogelijk. Hoe moet je het model aanpassen om dit te voorkomen?

• Hogere orde rand discretisaties.
• Stabiele discretisatie voor convectie termen.

• Doorlaatbaarheid K (niet lineair) van de druk afhankelijk.
• Doorlaatbaarheid K richtings afhankelijk.

• Matrix A afhankelijkheid vervangen door subroutine afhankelijkheid. (`MV(in,out)`).

⋮

Referenties

[1] R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. van der Vorst, *Templates for the solution of linear systems: building blocks for iterative methods*, SIAM, Philadelphia, 1994. <http://www.netlib.org/templates/Templates.html>

[2] T.F. Chan and H.A. van der Vorst, *Approximate and incomplete factorizations*, Preprint 871, Dep. Math., University Utrecht (1994). <http://www.math.ruu.nl/people/vorst/>

[3] C. Pommerell, *Solution of large unsymmetric systems of linear equations*, Series in Micro-electronics 17, Hartung-Gorre Verlag Konstanz, 1992.

[4] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery, Numerical Recipes, Cambridge University Press, 1992.

[5] G. Sleijpen and H. van der Vorst, *Optimal iteration methods for large linear systems of equations*, in "Numerical Methods for Advection-Diffusion Problems", C.B. Vreugdenhil and B. Koren, eds., Notes on Numerical Fluid Mechanics, Vol. 45, Ch. 12, pp. 291–320, Vieweg, Braunschweig, 1993.

[6] G. Sleijpen and H. van der Vorst, *Krylov Subspace Methods for Large Linear Systems of Equations*, Preprint 803, Dep. Math., University Utrecht (1993). <http://www.math.ruu.nl/people/sleijpen/>

[7] H.A. van der Vorst, *Lecture notes on iterative methods*, <http://www.math.ruu.nl/people/vorst/>

[8] H.A. van der Vorst, *Diktaat numerieke lineaire algebra*, <http://www.math.ruu.nl/people/vorst/>

Appendix

Direkte methoden. Rechtstreekse methoden (*direkte methoden*) voor het oplossen van de vergelijking $\mathbf{Ax} = \mathbf{b}$ ontbinden $\mathbf{A}$ in factoren $\mathbf{L}$ en $\mathbf{U}$, $\mathbf{A} = \mathbf{LU}$, met $\mathbf{L}$ benedendriehoeks met 1-en op de diagonaal en $\mathbf{U}$ bovendriehoeks (*Gauss eliminatie*). Vervolgens lossen ze het benedendriehoeks stelsel $\mathbf{Ly} = \mathbf{b}$ op (*voorwaartse eliminatie*) en daarna het bovendriehoeks stelsel $\mathbf{Ux} = \mathbf{y}$ (*terugwaartse substitutie*).

In ons geval is matrix $\mathbf{A}$ ijl. Deze ijlheid gaat verloren in de L- en U-factoren. In onze voorbeelden heeft $\mathbf{A}$ 5 niet-nullen per rij, terwijl $\mathbf{L}$ en $\mathbf{U}$ ieder $\approx n_x$ niet-nullen per rij hebben. In vergelijking met de kosten van één MV met $\mathbf{A}$ (in onze vb. $10n$ flop met $n = n_x \cdot n_y$) zijn de kosten van, met name, het berekenen van de factoren duur ($\approx 2n_x^2 \cdot n$ flop). Ook het oplossen van de driehoeks stelsels is duurder ($\approx 2n_x n$ flop). Verder kan voor grote problemen het opslaan van de factoren op problemen stuiten (voor $\mathbf{A}$ moeten $5n$ reals worden opgeslagen, voor $\mathbf{U}$ en $\mathbf{L}$ is dat ongeveer $10n_x$ keer zoveel). Om deze redenen is er behoefte aan oplosmethoden die alleen gebruik maken van een beperkt aantal "goedkope" vermenigvuldigingen met $\mathbf{A}$; als het aantal vermenigvuldigingen ver onder de n_x^2 kan blijven is deze benadering zeer aantrekkelijk. Iteratieve methoden vallen in deze kategorie.

Iteratieve methoden. We bekijken, om het idee achter iteratieve methoden te begrijpen, eerst naar het flauwe geval waarin $\mathbf{A}$ een 1 bij 1 matrix is, $\mathbf{A} = (\lambda)$ en $\mathbf{b} = (\beta)$. De vergelijking $\lambda x = \beta$ zou voor een skaleir α als volgt opgelost kunnen worden: kies een x_0 en bereken iteratief ($k = 0, 1, 2, \ldots$)

$$x_{k+1} = x_k + \alpha(\beta - \lambda x_k).$$

Als $|1 - \alpha\lambda| < 1$ convergeren de x_n naar de oplossing (β/λ; zie diktaat Numerieke Wiskunde A). Als $\lambda > 0$ is kunnen we met de parameter α de convergentiesnelheid beïnvloeden.

Richardson. Het meer dimensionale geval kunnen we op dezelfde manier proberen op te lossen:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha\mathbf{r}_k \quad \text{met} \quad \mathbf{r}_k = \mathbf{b} - \mathbf{Ax}_k.$$

Dit zogenaamde Richardson iteratie proces (zie ALG.12) convergeert als α zo gekozen is dat voor iedere eigenwaarde λ van $\mathbf{A}$ geldt $|1 - \alpha\lambda| < 1$. Als alle eigenwaarden $\lambda > 0$ (en $< 10^5$) werkt $\alpha = 10^{-6}$, maar de convergentie zal dan bijzonder traag zijn (zeker als 'n $\lambda \leq 1$). De snelste convergentie krijgen we voor die α waarvoor $(1 - \max_\lambda |1 - \alpha\lambda|)$ maximaal is. In de praktijk is het praktisch onmogelijk om een geschikte α te vinden: de λ's zijn onbekend.

LMR. Het is echter mogelijk het proces zelf te laten uitzoeken wat in iedere stap de beste α is (α is dan afhankelijk van k). De α waarvoor de fout $\|\mathbf{x} - \mathbf{x}_{k+1}\| = \|\mathbf{x} - \mathbf{x}_k - \alpha\mathbf{r}_k\|$ minimaal is zou de beste zijn. Helaas kunnen we die α niet berekenen omdat die afhangt van de onbekende oplossing $\mathbf{x}$. Merk op dat het nieuwe residu $\mathbf{r}_{k+1} \equiv \mathbf{b} - \mathbf{Ax}_{k+1}$ gelijk is aan $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha\mathbf{Ar}_k$.

```
Choose a scalar α.
Choose an x_0.
Compute r_0 = b − Ax_0.
For k = 0, 1, 2, ..., k_max do
    stop if ‖r_k‖ ≤ tol
    u_k = r_k
    compute c_k = Au_k
    x_{k+1} = x_k + αu_k
    r_{k+1} = r_k − αc_k
end for
```

ALG.12: *Richardson iteratie.*

De α waarvoor het nieuwe residu minimaal is, waarvoor $\|\mathbf{r}_k - \alpha\mathbf{Ar}_k\|_2$ minimaal is, kan wel gemakkelijk bepaald worden: $\alpha = \mathbf{c}_k^T\mathbf{r}_k/\mathbf{c}_k^T\mathbf{c}_k$, hierbij is $\mathbf{r}_k$ bekend uit de vorige stap en $\mathbf{c}_k = \mathbf{Ar}_k$. Het resulterend LMR (Local Minimal Residual) algorithme staat in ALG.13. Dit algorithme convergeert altijd (zelfs voor matrices met niet-positieve eigenwaarden). De convergentie kan helaas nog steeds ontzettend traag zijn.

GCR. Een volgende verbetering krijgen we door het residu niet alleen te minimaliseren ten opzichte van de laatste correctie vector $\mathbf{c}_k$, maar tenopzichte van alle reeds berekende correctie vectoren $\mathbf{c}_1, \ldots, \mathbf{c}_k$: minimalizeer $\|\mathbf{r}_k - (\alpha_1\mathbf{c}_1 + \ldots + \alpha_k\mathbf{c}_k)\|$. De berekeningen blijven overzichtelijk en redelijk stabiel door eerst de correctievectoren te orthogonaliseren t.o.v. elkaar. Het proces dat dan ontstaat heet GCR. Door met orthogonal vectoren $\mathbf{c}_k$ te werken vermijden we, dat we met een correctie $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k\mathbf{c}_k$, opnieuw komponenten in richting $\mathbf{c}_j$ invoeren.

ALG.13: *Local Minimal Residuals.*

```
Choose an x_0.
Compute r_0 = b − Ax_0.
For k = 0, 1, 2, ..., k_max do
    stop if ‖r_k‖ ≤ tol
    u_k = r_k
    compute c_k = Au_k
    σ_k = c_k^T c_k ,  α_k = c_k^T r_k/σ_k
    x_{k+1} = x_k + α_k u_k
    r_{k+1} = r_k − α_k c_k
end for
```