Iteratieve Lineaire Oplosmethoden

Arthur van Dam

13 april 1999

Inhoudsopgave

Hoofdstuk 1

Inleiding

Dit verslag geeft een beschrijving van het eerste deel van het tweedejaars CS Practicum.

Hierin wordt een grondwatersysteem bekeken waarbij een gebied als rechthoekig rooster wordt gediskretiseerd. De druk van het grondwater (of eventueel gifconcentratie) op de verschillende roosterpunten en de onderlinge samenhang hiervan tussen naburige roosterpunten, levert een (groot) stelsel van lineaire vergelijkingen op.

Voor het snel en goedkoop oplossen van dit stelsel zijn verschillende oplosmethoden onderzocht. Dit zijn iteratieve methoden, aangezien het probleem te complex is om analytisch op te lossen.

Voor dit verslag is onder andere gebruik gemaakt van de practicumhandleiding van G.L.G. Sleijpen ([2]). De Testproblemen waarmee de verschillende oplosmethoden zijn onderzocht staan in appendix B.

Hoofdstuk 2

Modellering

Het eigenlijke probleem dat opgelost dient te worden, is van fysische aard. Om hier mathematische methoden op toe te kunnen passen, moet het probleem in een model worden gezet.

2.1 Grondwaterstroming

We beschouwen een stationaire stroming in het xy-vlak. De stroming over een gebied D is te beschrijven door de partiële differentiaalvergelijking:

$$-\nabla \cdot (K \nabla \phi) = Q, \tag{2.1}$$

waarin:

- $\phi = \phi(x, y)$ is de *grondwaterdruk* op plaats (x, y), gemeten in meters.
- $K = K(x, y)$ geeft de doorlaatbaarheid aan van de grondlaag en is een symmetrische 2×2 matrix van *doorlaatbaarheidscoëfficiënten*, gemeten in m^3 dag^{-1} m^{-2}.
- Het *snelheidsveld* $(u, v) = -K \nabla \phi$ geeft de stroomsnelheid aan in de x richting $(u(x, y))$ en de y richting $(v(x, y))$ ter plaatse (x, y), gemeten in m^3 dag^{-1} m^{-2}.
- De *bronterm* $Q = Q(x, y)$ geeft de instroom of uitstroom van water aan in (x, y), gemeten in m^3 per dag per m^3 grond. (Respectievelijk pomp of put).

Op de rand van het gebied onderscheiden we de volgende *randvoorwaarden*:

Essentiële of Dirichlet randvoorwaarde legt de druk op de rand nadrukkelijk vast in een functie ϕ_0, zodat $\phi = \phi_0$.

Natuurlijke of Neumann randvoorwaarde legt de uitstroomsnelheid vast: $-(K \nabla \phi) \cdot n = \phi_0$, waarin n de normaalvector op de rand is en ϕ_0 een vastgelegde functie.

Gemengde of Robin randvoorwaarde is een combinatie van de twee voorgaande: $-(K \nabla \phi) \cdot n = \gamma(\phi - \phi_\infty)$, waarin γ een evenredigheidsconstante is en ϕ_∞ een referentiedruk. De uitsroomsnelheid is dus evenredig met het drukverschil (denk aan de bron van een rivier).

Het probleem is nu om met gegeven doorlaatbaarheid, brontermen en randvoorwaarden het snelheidsveld en de grondwaterdruk te bepalen.

De modellering en de oplosmethoden die we voor dit grondwaterprobleem gebruiken, zijn ook in veel andere fysische contexten bruikbaar, zoals:

- Warmtetransport
- Rotatie vrije vloeistofstroming
- Torsie (mechanica)
- Magnetostatica
- Electrostatica
- Transversale uitwijking van elastische membranen

2.2 Bodemverontreiniging

Naast grondwaterstroming, kunnen we met een vergelijking verwant aan de Poisson vergelijking de verspreiding van een in water oplosbare stof G beschrijven. Dit komt er als volgt uit te zien:

$$-\nabla \cdot (A\nabla\psi) + \nabla \cdot (V\psi) + c\psi = f, \tag{2.2}$$

waarin:

- $\psi = \psi(x, y)$ de concentratie van de stof G ter plaatse (x, y).
- A geeft, net als K, de doorlaatbaarheid van de grond aan, maar nu voor de stof G. A is in het algemeen niet evenredig met K.
- $-A\nabla\psi$ is de snelheid, waarmee het gif zich door diffusie verspreidt.
- V is het snelheidsveld van de grondwaterstroming en is dus zelf een oplossing van een vergelijking als (2.1).
- f is de *bronterm* en geeft aan hoeveel stof G er aan het grondwater wordt toegevoegd.

Wat betekenen al deze grootheden nu voor het fysische probleem? Welnu, de gifconcentratie wordt erdoor beïnvloed en kan dus op de volgende manieren veranderen:

1. door toevoeging van extra stof van buitenaf (gemodelleerd door *bronterm* f),
2. door *diffusie* (gemodelleerd door $-\nabla \cdot (A\nabla\psi)$),
3. door *konvektie*, oftewel door meestromen met het grondwater (gemodelleerd door $\nabla \cdot (V\psi)$),
4. door *natuurlijke afbraak* (gemodelleerd door $c\psi$). Hierin is c een evenrdigheidsconstante, afhankelijk van de grondsoort.

Ook in dit model beschouwen we de bronterm f en de randvoowaarden als tijdsonafhankelijk.

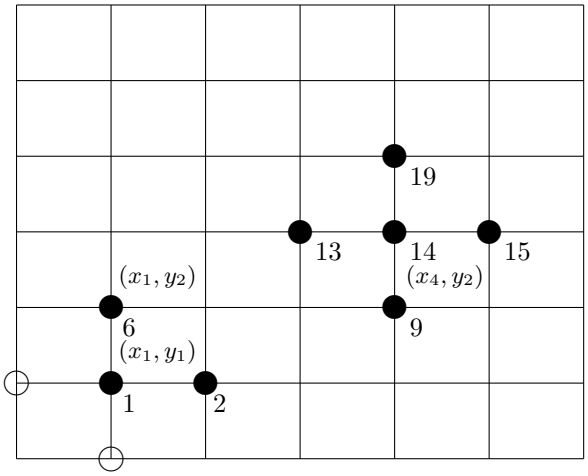

FIG.1: *Het rooster voor een rechthoekig gebied ($n_x = n_y = 5$). De punten $\bullet$ behoren tot D, de punten $\circ$ liggen op de rand ∂D van D.*

2.3 Discretisatie

Zoals altijd in de Computational Science is het beschreven probleem te ingewikkeld om analytisch op te lossen. We zullen het domein (het gebied D) discretiseren als een $n_x \times n_y$ rooster R. De rand ∂D die hier nog extra omheen ligt, is niet bevat in D. Als het gebied $D = \partial D$ afmetingen $X \times Y$ heeft, geldt:

$$h_x = \frac{X}{n_x + 1} \text{ en } h_y = \frac{Y}{n_y + 1} \tag{2.3}$$

Nu bestaat R uit de punten $(x_i, y_j) \equiv (ih_x, jh_y)$ in D met $(i, j \in \mathbb{N}, i \leq n_x, j \leq n_y)$. Zie figuur 1.

We zullen nu de differentiaalvergelijking

$$-\frac{\partial}{\partial x}\left(a\frac{\partial \psi}{\partial x}\right) - \frac{\partial}{\partial y}\left(b\frac{\partial \psi}{\partial y}\right) + \frac{\partial (u\psi)}{\partial x} + \frac{\partial (v\psi)}{\partial y} + c\psi = f \tag{2.4}$$

moeten discretiseren. De partiële afgeleiden benaderen we door een differentiequotiënt rond het punt (x_i, y_j). Voor $\frac{\partial \psi}{\partial x}(x_i, y_j)$ geldt:

$$\partial_x^\circ \psi_{i,j} \equiv \frac{\psi_{i+1,j} - \psi_{i-1,j}}{2h_x} \qquad \text{en} \qquad \partial_x^\bullet \psi_{i,j} \equiv \frac{\psi_{i+\frac{1}{2},j} - \psi_{i-\frac{1}{2},j}}{h_x} \tag{2.5}$$

Hierbij is $\psi_{i+\frac{1}{2},j} = \psi(x_{i+\frac{1}{2}}, y_j)$. Ook in dit 'tussenpunt' kunnen we $\frac{\partial \psi}{\partial x}$ bepalen:

$$\partial_x^\bullet \psi_{i-\frac{1}{2},j} = \frac{\psi_{i,j} - \psi_{i-1,j}}{h_x} \tag{2.6}$$

In (2.5) en (2.6) is h_x de afstand in x richting tussen twee aangrenzende roosterpunten. De definities voor de *centrale differenties* $\partial_y^\bullet$ en ∂_y° in de y-richting zijn analoog. Overigens hoeven

vanwege (2.6) de ψ's in de tussenpunten in (2.5) niet te worden bepaald bij dubbele afgeleiden. We kunnen (2.4) nu discretiseren met (2.5) en (2.6):

$$-\partial_x^\bullet(a\partial_x^\bullet\psi) - \partial_y^\bullet(b\partial_y^\bullet\psi) + \partial_x^\circ u\psi + \partial_y^\circ v\psi + c\psi = f + \delta, \tag{2.7}$$

waarin $\delta(x_i, y_j)$ de discretisatiefout voorstelt, die we voortaan zullen verwaarlozen.

Wanneer we dit nauwkeurig uitschrijven en sorteren op ψ, levert dit:

$$\begin{aligned} f_{i,j} = &\left(\frac{a_{i+\frac{1}{2},j} + a_{i-\frac{1}{2},j}}{h_x^2} + \frac{b_{i,j+\frac{1}{2}} + b_{i,j-\frac{1}{2}}}{h_y^2} + c_{i,j}\right)\psi_{i,j} \\ &+ \left(-\frac{a_{i-\frac{1}{2},j}}{h_x^2} - \frac{u_{i-1,j}}{2h_x}\right)\psi_{i-1,j} + \left(-\frac{a_{i+\frac{1}{2},j}}{h_x^2} + \frac{u_{i+1,j}}{2h_x}\right)\psi_{i+1,j} \\ &+ \left(-\frac{b_{i,j-\frac{1}{2}}}{h_y^2} - \frac{v_{i,j-1}}{2h_y}\right)\psi_{i,j-1} + \left(-\frac{b_{i,j+\frac{1}{2}}}{h_y^2} + \frac{v_{i,j+1}}{2h_y}\right)\psi_{i,j+1} \end{aligned} \tag{2.8}$$

Uitdrukking (2.8) geldt voor ieder punt in D en kan globaal worden beschreven door:

$$\alpha_{ij}^{\text{cent}}\psi_{i,j} + \alpha_{ij}^{\text{west}}\psi_{i-1,j} + \alpha_{ij}^{\text{oost}}\psi_{i+1,j} + \alpha_{ij}^{\text{zuid}}\psi_{i,j-1} + \alpha_{ij}^{\text{noord}}\psi_{i,j+1} = \widehat{f}_{ij}. \tag{2.9}$$

Hierin worden de α's gegeven door de coëfficiënten voor de ψ's uit (2.7) en $\widehat{f}$ door de bronterm f. De α's worden ook wel *koppelingscoëfficiënten* genoemd, omdat zij de koppeling beschrijven tussen naburige roosterpunten.

Voor punten nabij de rand hebben we de functiewaarden op de rand nodig. Deze rand ∂D is echter niet bevat in D, dus moeten we hiervoor de randvoorwaarden gebruiken:

$$-\mu\left(a\frac{\partial\psi}{\partial x}, b\frac{\partial\psi}{\partial y}\right)\cdot n + (1-\mu)\psi = \psi_0 \tag{2.10}$$

Bij $\mu = 0$ levert dit een Dirichlet randvoorwaarde op, voor $\mu = 1$ een Neumann randvoorwaarde, en tenslotte $\mu = 1/(1-\gamma)$ voor een Robin randvoorwaarde. Ook deze vergelijking kunnen we discretiseren:

$$\begin{aligned} \mu_{i,0}(b\partial_y^\bullet\psi)_{i,\frac{1}{2}} + (1-\mu_{i,0})\psi_{i,0} &= \psi_0(x_i, 0) + \delta_{i,0} \\ \mu_{0,j}(a\partial_x^\bullet\psi)_{\frac{1}{2},j} + (1-\mu_{0,j})\psi_{0,j} &= \psi_0(0, y_j) + \delta_{0,j} \\ \mu_{i,n_y+1}(b\partial_y^\bullet\psi)_{i,n_y+\frac{1}{2}} + (1-\mu_{i,n_y+1})\psi_{i,n_y+1} &= \psi_0(x_i, y_{n_y+1}) + \delta_{i,n_y+1} \\ \mu_{n_x+1,j}(a\partial_x^\bullet\psi)_{n_x+\frac{1}{2},j} + (1-\mu_{n_x+1,j})\psi_{n_x+1,j} &= \psi_0(x_{n_x+1}, y_j) + \delta_{n_x+1,j} \end{aligned} \tag{2.11}$$

Hierin zijn de δ's weer discretisatiefouten, die we zullen verwaarlozen.

Ook (2.11) schrijven we weer uit m.b.v. (2.6) en analoge uitdrukkingen. De funktiewaarden ψ_{ij} op de rand staan dan uitgedrukt in bekende funktiewaarden $\psi_0(x_i, y_j)$ en funktiewaarden $\psi_{i,j-1}$, $\psi_{i-1,j}$, etc., op het rooster. Als we die vier vergelijkingen (voor de vier randen) invullen in (2.8), de constante termen bij de $\widehat{f}$ onderbrengen en rechts van het =-teken weer groeperen op verschillende ψ dan krijgen alle α's een correctieterm, evenals $\widehat{f}$. Door het mengen van de termen langs de rand en de randvoorwaarden, worden de termen op de rand geëlimineerd hetgeen een kleiner probleem, en dus tijdswinst oplevert.

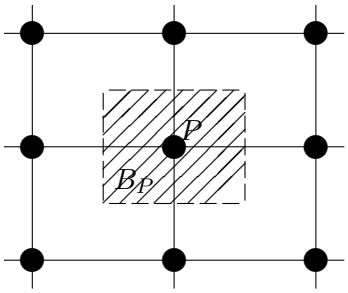

FIG.2: *De pomp gediskretiseerd*

De waarden $\psi_{i,j}$ van de oplossing van (2.9) zijn een benadering van de werkelijke grondwaterdruk of gifconcentratie. Het door de druk- of concentratieverschillen onstane stromingsveld $-A\nabla\psi$ kunnen we nauwkeurig berekenen door

$$(-a_{ij}\partial_x^\circ\psi_{ij}, -b_{ij}\partial_y^\circ\psi_{ij}) \tag{2.12}$$

In het algoritme hebben we eerst de 'gewone' α's en $\widehat{f}$'s bepaald en deze vervolgens in een tweede lus over de rand gecorrigeerd voor de randpunten. Zo hebben we een volledig gedefinieerd stelsel van vergelijkingen, wat we moeten gaan oplossen.

In sectie 2.4 zal worden besproken hoe we dit stelsel van vergelijkingen in een matrix zetten. Representatie van één vergelijking – dus van één roosterpunt – kan met een zogenaamd *stencil*:

$$\begin{bmatrix} 0 & \alpha_{ij}^{\text{noord}} & 0 \\ \alpha_{ij}^{\text{west}} & \alpha_{ij}^{\text{cent}} & \alpha_{ij}^{\text{oost}} \\ 0 & \alpha_{ij}^{\text{zuid}} & 0 \end{bmatrix}. \tag{2.13}$$

Een stencil representeert dus één rij uit de matrix van het stelsel.

2.3.1 Het discretiseren van pompen

Beschouwen we het resulterende stromingsveld in de oplossing van (2.4) op de rechthoek rondom één roosterpunt, dan is er een bepaalde instroom en uitstroom van water (of gif). De netto-instroom wordt gemodelleerd door de bronterm. Deze is 0 in systemen zonder pompen of rivieren; bij dit model gaan we er immers van uit dat er geen water (of gif) kan verdwijnen (instroom=uitstroom).

Als er echter een pomp is, die water of gif in het systeem brengt, of daaraan onttrekt, of een rivier loopt door het gebied, dan is dit netto-effect ongelijk aan 0.

Wanneer gegeven is dat een pomp in het roosterpunt $P = (x_{i_p}, y_{j_p})$ het water wegpompt met een snelheid van p m^3 per dag dan moeten we dit omrekenen naar de bronterm. De bronterm geeft aan hoeveel m^3 er per dag per m^3 grond wordt weggepompt. Om de p te middelen over de hele rechthoek van h_x bij h_y stellen we de bronterm gelijk aan $p/(h_x h_y d)$, met d de dikte van de grondlaag (in de z-richting, in meters).

Rivier; een rechtlijnige pomp

Een speciale pomp is de rivier. Deze kan worden voorgesteld door een lijn van pompen. Een rivier wordt beschreven door de punten (x, y) waarvoor $ax + by = c$, met a, b en c

bekende constanten en de capaciteit. De capaciteit wordt gegeven door q m^3 water per dag per strekkende meter.

Wanneer we nu weer een $h_x \times h_y$ rechthoek B_P met middelpunt P nemen die doorsneden wordt door een rivier R, dan bepalen we de lengte l (int meters) van het stukje rivier dat in B_p ligt. Ook hier middelen we de capaciteit over de hele rechthoek. De bijdrage van de rivier aan de bronterm in P wordt hiermee $lq/(h_x h_y d)$.

In het algoritme lopen we eerst in stappen alle rivieren af om te bepalen *welke* roosterpunten effect ondervinden van de rivier. Vervolgens wordt bepaald *welk deel van het volume* rondom die roosterpunten wordt doorsneden.

Na aanpassing van de brontermen aan alle pompen, is het systeem volledig gediscretiseerd.

2.4 Een matrix formulering

Uit de discretisatie is een stelsel van $n_x n_y$ vergelijking ontstaan. Nummeren we de roosterpunten als in figuur 1 dan kunnen we het stelsel beschrijven als de matrix-vector vergelijking

$$\mathbf{A}\mathbf{x} = \mathbf{b}. \tag{2.14}$$

Hierin bevat $\mathbf{A}$ de koppelingscoëfficiënten, b de brontermen en is $\mathbf{x}$ de oplossing voor ψ op alle roosterpunten.

Uit (2.9) blijkt echter dat er in iedere vergelijking hoogstens 5 koppelingscoëfficiënten staan. In de bewuste rij van matrix $\mathbf{A}$ is dus de rest van de elementen gelijk aan 0. We hebben hier dus te maken met een zogenaamde *sparse matrix* en we zullen $\mathbf{A}$ in *compressed row format* opslaan om veel tijd en ruimte te besparen. Hiervoor zijn verschillende technieken; wij gebruiken hiervoor de *row-indexed sparse storage mode* zoals beschreven in Ch. 2(p.78-79) van "Numerical Recipes" ([1]). Het globale idee is alle diagonaal elementen in een vector op te slaan, gevolgd door alle niet-nul buitendiagonaal elementen. Daarnaast wordt een tweede, even lange, vector bijgehouden die informatie bevat over de positie van de elementen in de oorspronkelijke matrix $\mathbf{A}$.

In het programma zijn voor de α^{cent}, α^{west} etc. vijf aparte vectoren gereserveerd. Als we weten waar alle verschillende α's in de matrix staan (zie Appendix A), dan kunnen we eenvoudig een methode implementeren die $\mathbf{A}$ in *compressed row format* opslaat.

Eigenschappen van A

We hebben reeds gezien dat $\mathbf{A}$ een bijzondere vorm heeft. Wanneer is $\mathbf{A}$ eigenlijk symmetrisch? Met de matrix uit appendix A is goed te zien dat voor een symmetrische matrix $\mathbf{A}$ geldt:

$$\alpha_{i,j}^{\text{zuid}} = \alpha_{i,j-1}^{\text{noord}} \quad \text{en} \quad \alpha_{i,j}^{\text{west}} = \alpha_{i-1,j}^{\text{oost}} \tag{2.15}$$

Met andere woorden: de koppelingsfactor van een punt p op een punt q is gelijk aan de koppelingsfactor van q op p in tegenovergestelde richting.

Wanneer we nog eens naar (2.8) kijken, zien we dat de voorfactoren opgebouwd zijn uit diffusie-termen (met a en b) en convectietermen (met u en v).
Voor een punt (x_i, y_j) is $\alpha_{i,j}^{\text{noord}} = -b_{i,j+\frac{1}{2}}/h_y^2 + v_{i,j+1}/2h_y$. De noorderbuur van dit punt (x_i, y_{j+1}) heeft een koppelingsfactor naar omlaag van $\alpha_{i,j+1}^{\text{zuid}} = -b_{i,j+\frac{1}{2}}/h_y^2 - v_{i,j}/2h_y$. Nu is te zien dat de diffusietermen van deze twee α's gelijk zijn maar de convectietermen niet. Willen

we dus een symmetrische matrix **A** hebben, dan moet de convectieterm $\nabla \cdot (V\psi)$ gelijk zijn aan 0. Iets dergelijks geldt in de oost-west-richting.

Hoofdstuk 3

Iteratieve oplosmethoden

We hebben het systeem nu gediscretiseerd en effectief opgeslagen. Nu moet er een keuze gemaakt worden voor de oplosmethode; gaan we het stelsel direct oplossen (*LU-decompositie*) of passen we iteratieve methoden toe?

3.1 De noodzaak tot iteratie

We zagen al dat de matrix $\mathbf{A}$ ijl is. Gaan we nu LU-dcompositie gebruiken, dan loopt de matrix tussen de diagonaal en de onderste of bovenste band (resp. $\mathbf{L}$ of $\mathbf{U}$) vol. dit levert een 'gevulde band' op ter breedte n_x, zodat het geheugengebruik $\mathcal{O}(n_x^2 n_y)$ is. Dit levert al snel grote problemen op.

Naast geheugengebruik, moeten we ook op de looptijd letten. Wanneer we in de k-e kolom alle niet-diagonaal-elementen op nul moeten zetten, kost dit n_x delingen en telkens voor elke rij n_x aftrekkingen (vanwege bandbreedte n_x). In totaal zijn er $n_x \cdot n_y$ kolommen, dus is de vereiste arbeid $\mathcal{O}(n_x^2 \cdot n_x \cdot n_y)$. In sectie 3.3.1 wordt hier nog dieper op ingegaan.

Toch mogen we nu al stellen dat het gebruikt van iteratieve methoden voordeel zou kunnen opleveren.

3.2 iteratieve oplosmethoden

Iteratieve oplosmethoden nemen een startwaarde $\mathbf{x}_0$ voor de oplossing en gaan deze vervolgens iteratief verbeteren; uit een benadering $\mathbf{x}_k$ wordt een betere benadering $\mathbf{x}_{k+1}$ gemaakt. Hiervoor zijn in iedere stap alleen eenvoudige rekenoperaties nodig (*vector-update* (AXPY): $\alpha\mathbf{x}+\mathbf{y}$; *inprodukt* (DOT): $\mathbf{x}^T\mathbf{y}$; *matrix-vector produkt* (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 produkt van dit soort matrices).

Hoe 'goed' een benadering is, wordt aangegeven door het *residu*: $\mathbf{r}_k := \mathbf{b} - \mathbf{A}\mathbf{x}_k$. De grootte van het residu is bepalend voor de voortgang van het iteratieve proces; de iteratie stopt als $\| \mathbf{r}_k \| \leq tol$.

De residuen die we hier bekijken, behoren tot de zogenaamde *Krylov deelruimte*:

$$\mathcal{K}_{k+1}(\mathbf{A}; \mathbf{r}_0) \equiv \text{span}(\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \ldots, \mathbf{A}^k\mathbf{r}_0) \tag{3.1}$$

```
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 (drie) verschillende oplosmethoden die we zullen bekijken, heten dan ook wel *Krylov deelruimte methoden.*

3.3 Generalized Conjungate Residuals (GCR)

We gebruiken de volgende recursie:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k(\mathbf{b} - \mathbf{A}\mathbf{x}_k) = \mathbf{x}_k + \alpha_k\mathbf{r}_k, \tag{3.2}$$

zodat

$$\mathbf{x}_{k+1} - \mathbf{x} = (\mathbf{I} - \alpha_k\mathbf{A})(\mathbf{x}_k - \mathbf{x}) \tag{3.3}$$

Om de norm van dit residu zo snel mogelijk naar 0 te laten convergeren, moeten we een geschikt α_k vinden. Deze zijn niet direct uit 3.3 te halen. Vermenigvuldiging van (3.3) met $\mathbf{A}$ levert echter: $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k\mathbf{A}\mathbf{r}_k$ en hieruit kan de optimale α_k wel bepaald worden door $\mathbf{r}_k$ en $\mathbf{A}\mathbf{r}_k$ loodrecht op elkaar te zetten: $\alpha_k = \mathbf{c}_k^T\mathbf{r}_k/\mathbf{c}_k^T\mathbf{c}_k$, met $\mathbf{c}_k = \mathbf{A}\mathbf{r}_k$.
Als we nu het residu niet alleen minimaliseren ten opzichte van de laatste basisvector $\mathbf{c}_k$, maar ook nog ten opzichte van alle reeds bekende $\mathbf{c}_1, \ldots, \mathbf{c}_k$ dan bereiken we een nog snellere convergentie. In het algoritme wordt een zoekrichting $\mathbf{u}_k$ 'gekozen' en vervolgens wordt een nieuwe basisvector bepaald volgens

$$\mathbf{c}_k = \mathbf{A}\mathbf{u}_k \tag{3.4}$$

Vervolgens wordt deze nieuwe basisvector in een lus loodrecht gezet op alle voorgaande basisvectoren.

Samengevat: GCR genereert een orthogonale basis $(\mathbf{c}_0, \ldots, \mathbf{c}_k)$ voor $\mathcal{K}_{k+1}(\mathbf{A}; \mathbf{r}_0)$ en bepaald vervolgens $\mathbf{r}_k$ als het orthogonale complement van deze ruimte:

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

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

In Alg.1 staat het gebruikte algoritme voor GCR.

3.3.1 Iteratief vs. direct

Tijdsanalyse

Bij directe methoden wordt een LU-decompositie van de matrix $\mathbf{A}$ gemaakt. Hierdoor gaat de ijlheid van de oorspronkelijke matrix verloren. Om de factoren te bepalen in LU-decompositie zijn ongeveer $2n_x^2n$ flops nodig. Hierin is $n = n_xn_y$. Dit is dan ook de grootste 'kostenpost'. Het oplossen van het verkregen stelsel vervolgens, kost circa $2n_x \cdot n_xn_y$ flops.

GCR is op beide punten goekoper: in de methode `GCR` wordt een lus van k stappen doorlopen, waarin een ID, een MV, drie DOTS, twee AXPY's en een binnenste lus worden uitgevoerd. De genoemde basisoperaties kosten $21kn$ flops. In de binnenste lus wordt nog een DOT en twee AXPY's uitgevoerd. De opdrachten in deze geneste lus worden $\frac{1}{2}k(k+1)$ maal uitgevoerd en kosten dus $3k(k_1)n$ flops.

Dit betekent dat LU $\mathcal{O}(n_x^2n)$ kost en GCR $\mathcal{O}(k^2n)$. Wanneer $n_x \gg k$, is het raadzaam om over te stappen op iteratieve methoden. Het is natuurlijk nooit helemaal duidelijk wat k is, maar k wordt in ieder geval begrensd door k_{max}. Gedacht moet worden aan k in $\mathcal{O}(100)$.

Geheugenanalyse

Zoals al bleek in sectie 3.1 is er voor LU-decompositie $\mathcal{O}(n_xn)$ geheugen nodig. De matrix is immers tussen de diagonaal en de onderste band 'volgelopen' en die elementen moeten allemaal opgeslagen worden.

In GCR heeft de matrix in de *row-indexed sparse storage mode* $5n$ doubles aan geheugen nodig. De bijbehorende 'positie-vector' bevat n integers (Zie sectie 2.4). Verder is voor $\mathbf{x}, \mathbf{b}$ en $\mathbf{r}$ 3 maal n doubles nodig en voor de basisvectoren en zoekrichtingen $\mathbf{c}$ en $\mathbf{u}$ twee maal n doubles. Deze laatste twee moeten echter allen bewaard blijven in de lus, dus dat waordt nog met een factor k vemenigvuldigd. In totaal levert dit een geheugenbehoefte van $\mathcal{O}(kn)$. Ook hier weer is te zien dat voor $n_x \gg k$ het verstandig is om op GCR over te stappen.

3.3.2 Experimenten met GCR

Om te kijken hoe GCR zich gedraagt ten aanzien van verschillende stelsels, plotten we op 10-log schaal de norm $\|\mathbf{r}_k\|$ van de residuen tegen de iteratiestap.

De testproblemen die we hiervoor zullen gebruiken (en ook bij de komende andere oplosmethoden) staan in appendix B.

Welke invloed zal $\mathbf{b}$ (brontermen en randfuncties) hebben op de convergentie. Vermoedelijk zal dit weinig invloed hebben. We hebben op dit probleem GCR losgelaten en vervolgens telkens één parameter veranderd en weer GCR uitgevoerd. De resultaten staan in tabel 3.1.

Het blijkt dat de pomp het probleem aardig kan verstoren: wanneer deze wordt weggelaten, is de convergentie duidelijk sneller. De doorlaatbaarheidscoëfficiënten hebben duidelijk

probleem	aant.stappen	$\|\mathbf{r}\|$	tijd (s)
Testprobleem 1	146	6.1e-8	5.29e0
I zonder pomp	25	2.0e-8	1.60e0
I b=400 ipv 40	280	6.4e-8	1.95e1
I a=400 ipv 40	105	5.6e-7	4.15e0
I x_0=100 ipv 0	151	3.2e-8	6.68e0
I $\mu_n = \mu_z = 0, \mu_0 = \mu_w = 1$	117	1.3e-7	4.66e0
I $h_x = h_y = 15$	78	1.2e-8	6.30e-1
I $h_x = 30h_y = 15$	93	4.4e-8	1.64e-0
I $h_x = 15h_y = 30$	144	1.6e-8	2.97e-0

TABEL 3.1: *Experimenten op symmetrisch probleem I met aanpassingen*

invloed, maar vermoedelijk hangt dit samen met de randvoorwaarden: vertienvoudiging van a (x-richting) levert tijdswinst op, terwijl dit bij b een vertaging oplevert. Bij verandering van de randvoorwaarden, is het verschil niet zo heel groot. Dit komt vermoedelijk door het feit dat de doorlaatbaarheidscoëfficiënten a en b gelijk zijn en dus in beide richtingen (x en y) weinig verschil zit.
Halvering van het rooster in beide richtingen blijkt een tijdswinst van ongeveer een factor $\frac{1}{2}$ op te leveren. Wanneer echter in één van de richtingen het rooster wordt gehalveerd, blijkt dit voor de y-richting een verbetering van een factor 0.6 op te leveren, terwijl in de x-richting nauwelijks verandering optreedt. Ook dit heeft vermoedelijk te maken met de randvoorwaarden.

De invloed van alle betrokken parameters in een probleem, blijkt vooral invloed te ondervinden van de randvoorwaarden. Verandering in andere parameters worden hierdoor beïnvloed. De startwaarde x_0 heeft weinig invloed: na een aantal stappen is reeds een beter startwaarde bereikt, waarmee verder geïtereerd kan worden.

In hoofdstuk B staan de verkregen oplossingen en stromingsvelden voor enkele testproblemen afgebeeld. Opvallend is dat bij asymmetrisch probleem IVa de discretisatiefouten groter zijn dan bij de andere testproblemen. In figuur 2 is dit te zien aan de 'golfjes' die rechts van de piek te zien zijn.

3.4 Preconditionering

3.4.1 Impliciet preconditioneren

De verbetering van de oplossingsvector $\mathbf{x}_k$ wordt in GCR in een bepaalde zoekrichting $\mathbf{u}_k$ gezocht. Naarmate de zoekrichting beter is, zal de benadering $\mathbf{x}_k$ ook steeds beter (sneller) naar de werkelijke oplossing van (2.14) convergeren. Het residu convergeert zo naar 0. Wanneer het residu meteen in de eerste slag 0 zou opleveren, hebben we direct de exacte oplossing van (2.14) gevonden. Als we naar Alg. 1 kijken, is te zien dat dit zo is, indien $\mathbf{c}_k = \mathbf{r}_k$, oftewel $\mathbf{A}\mathbf{u}_k = \mathbf{r}_k$. We zouden dus regel 5 van Alg. 1 moeten vervangen door : "`solve exactly` $\mathbf{A}\mathbf{u}_k = \mathbf{r}_k$". Dit oplossen is echter net zo moeilijk als het oplossen van het oorspronkelijke probleem.

Wat we wel kunnen doen is een matrix $\mathbf{M}$ vinden, die een beetje op $\mathbf{A}$ lijkt. We nemen dan als benadering $\mathbf{u}_k = \mathbf{M}^{-1}\mathbf{r}_k$. In de praktijk berekenen we niet $\mathbf{M}^{-1}$, maar lossen we $\mathbf{M}\mathbf{u}_k = \mathbf{r}_k$

```
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.

op (regel 5 in Alg. 2). De preconditioneerder is nu *impliciet* verwerkt. Het hieruit volgende algoritme voor *gepreconditioneerd GCR* staat in Alg. 2.

3.4.2 Expliciet preconditioneren

Bij impliciet preconditioneren is verder niet aangegeven hoe $\mathbf{u}_k$ opgelost moet worden uit $\mathbf{M}\mathbf{u}_k = \mathbf{r}_k$. We kunnen ook *expliciet preconditioneren*; hierbij is het belangrijk te realiseren dat het k-de residu van het gepreconditioneerde GCR algoritme minimaal is in de Krylov-deelruimte $\mathcal{K}_{k+1}(\mathbf{A}\mathbf{M}^{-1}; \mathbf{r}_0)$. Wanneer we namelijk een hulpvariabele $\mathbf{y} = \mathbf{M}\mathbf{x}$ introduceren, hebben we een nieuw op te lossen stelsel:

$$\mathbf{A}\mathbf{M}^{-1}\mathbf{y} = \mathbf{b} \tag{3.5}$$

Wanneer we hierop GCR toepassen en de $\mathbf{y}_k$ vermenigvuldigen met $\mathbf{M}^{-1}$, ontstaat het precies gepreconditioneerde GCR algoritme, wat $\mathbf{x}_k$ berekent. Dit achteraf terugwerken naar $\mathbf{x}$ (vermenigvuldiging met $\mathbf{M}^{-1}$) heet *post-processen*, De vermenigvuldiging $\mathbf{A}\mathbf{M}^{-1}$ heeft *pre-processen*.

In (3.5) wordt (2.14) *expliciet rechts gepreconditioneerd*. We kunnen ook *expliciet links preconditioneren*:

$$\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b} \tag{3.6}$$

Hierbij hoeven we de uiteindelijke oplossing niet te post-processen, maar in het algoritme worden wel gepreconditioneerde residuen bepaald en gebruikt. Je weet dus niet zeker of het algoritme wel gestopt is bij een residu dat je vereist had.

3.4.3 Incomplete decomposities

De kunst is nu om een goede preconditioneerder $\mathbf{M}$ te vinden. We zullen een aantal mogelijkheden bespreken, waarbij we de volgende notaties gebruiken:

$\mathbf{L}_A$ is de strikte benedendriehoek van $\mathbf{A}$
$\mathbf{D}_A$ is de diagonaal van $\mathbf{A}$
$\mathbf{U}_A$ is de strikte bovendriehoek van $\mathbf{A}$, zodat

$$\mathbf{A} = \mathbf{L}_A + \mathbf{D}_A + \mathbf{U}_A \tag{3.7}$$

Een eenvoudige preconditioneerder, waarmee efficiënt gerekend kan worden is

$$\mathbf{M} = \mathbf{D}_A \tag{3.8}$$

Ook zullen we kijken naar preconditioneerders van de vorm

$$\mathbf{M} = (\mathbf{D} + \mathbf{L}_A)\mathbf{D}^{-1}(\mathbf{D} + \mathbf{U}_A), \tag{3.9}$$

waarbij $\mathbf{D}$ een diagonaalmatrix is, die hierna verder gespecificeerd zal worden.
Door de structuur van $\mathbf{M}$ als in (3.9) is $\mathbf{M}\mathbf{u}_k = \mathbf{r}_k$ efficiënt op te lossen. In sectie C.1 staat de listing van de routine `msolve(`$\mathbf{D}, \mathbf{A}, \mathbf{r}, \mathbf{u}$`)`, die deze oplossing bepaalt. Deze wordt dus gebruikt bij impliciete preconditionering.

Eerst moeten we nog een $\mathbf{D}$ uit (3.9) vinden, zodat $\mathbf{M}$ op $\mathbf{A}$ lijkt, oftewel:

$$\mathbf{R} \equiv \mathbf{A} - \mathbf{M} = \mathbf{D}_A - \mathbf{D} - \mathbf{L}_A\mathbf{D}^{-1}\mathbf{U}_A \approx 0 \tag{3.10}$$

$\mathbf{R}$ kan op verschillende manieren klein zijn:

- De ILU-benadering; diag($\mathbf{R}$)=0;
- De MILU-benadering; $\mathbf{R1} = 0$, waarbij $\mathbf{1}$ de vector is met louter enen als coördinaten.
- De RILU-benadering; combinatie van ILU en MILU, bepaald door een parameter ω.

Als we de structuur van $\mathbf{A}$ bekijken (sectie 2.4), dan blijkt dat de diagonaalelementen $\mathbf{D}_k \equiv \mathbf{D}_{kk}$ iteratief uit $\mathbf{D}_j$ te bepalen zijn (voor $j < k$).

$$\begin{aligned}\mathbf{D}_k = \mathbf{A}_{kk} &- \left(\frac{\mathbf{A}_{k,k-1}\mathbf{A}_{k-1,k}}{\mathbf{D}_{k-1}} + \frac{\mathbf{A}_{k,k-n_x}\mathbf{A}_{k-n_x,k}}{\mathbf{D}_{k-n_x}}\right) \\ &- \omega\left(\frac{\mathbf{A}_{k,k-1}\mathbf{A}_{k-1,k+n_x-1}}{\mathbf{D}_{k-1}} + \frac{\mathbf{A}_{k,k-n_x}\mathbf{A}_{k-n_x,k-n_x+1}}{\mathbf{D}_{k-n_x}}\right).\end{aligned} \tag{3.11}$$

Met $\omega = 0$ is dit de ILU-benadering, met $\omega = 1$ de MILU-benadering. In sectie C.2 staat de listing van de routine `rilu(`$\mathbf{D}, \mathbf{A}, \omega$`)`, die de matrix $\mathbf{D}$ opbouwt.

3.4.4 Experimenten met impliciet preconditioneren

De vraag is nu, welke combinatie van ILU en MILU de beste resultaten oplevert. We gaan hiervoor weer naar een symmetrisch (5) en een niet-symmetrisch (4A) probleem kijken en gebruiken we verschillende $\mathbf{M}$ op een 30 bij 30 rooster met tol =1.0e-8. De resultaten staan in tabel 3.2 en 3.3.
hierbij geeft p net als in het programma het type preconditioneerder aan:

- $p = 0$: geen preconditionering, $\mathbf{M} = \mathbf{I}$ (GCR);

p	ω	**aant. stap.**	**tijd**
0	-	205	1.11e+1
1	-	172	8.46e+0
2	0	52	2.08e+0
2	0.5	46	2.00e+0
2	0.97	33	1.67e+0
2	1.0	33	1.70e+0

TABEL 3.2: *Resultaten symmetrisch probleem V voor verschillende (impliciete) preconditioneerders*

p	ω	**aant. stap.**	**tijd**
0	-	398	3.71e+1
1	-	269	1.82e+1
2	0	29	1.61e+0
2	0.5	35	1.73e+0
2	0.97	134	5.93e+0
2	1.0	104	4.08e+0

TABEL 3.3: *Resultaten asymmetrisch probleem IVa voor verschillende (impliciete) preconditioneerders*

- $p = 1$: diagonaal preconditionering, $\mathbf{M} = \mathbf{D}_A$;
- $p = 2$: impliciet met $\mathbf{M}$ uit (3.9).

Uit de resultaten blijkt dat voor een symmetrisch probleem MILU bijzonder effectief is: bij $\omega = 0.97$ is de convergentie het snelst. Voor asymmetrische $\mathbf{A}$ lijkt ILU de snelste methode; echter voor asymmetrisch probleem 0b werd de snelste convergentie bereikt bij $\omega = 1.0$. Het is dus niet helemaal duidelijk wat hier de beste keus is. Overigens blijkt dat alle vormen van preconditionering, zoals verwacht, een duidelijke verbetering leveren ten opzicht van gewone GCR. Tussen de verschillende preconditioneerders onderling, kunnen behoorlijke verschillen optreden.
Voor het symmetrische probleem zijn de convergentie-plotjes voor diagonaal preconditionering en preconditionering met $\mathbf{M}$ uit (3.9) ($\omega = 0.97$) in figuur 1 gezet. Beide lijnen vertonen het zelfde verloop dat we al eerder bij ongepreconditioneerd GCR zagen, maar de gepreconditioneerde versie is duidelijk sneller.

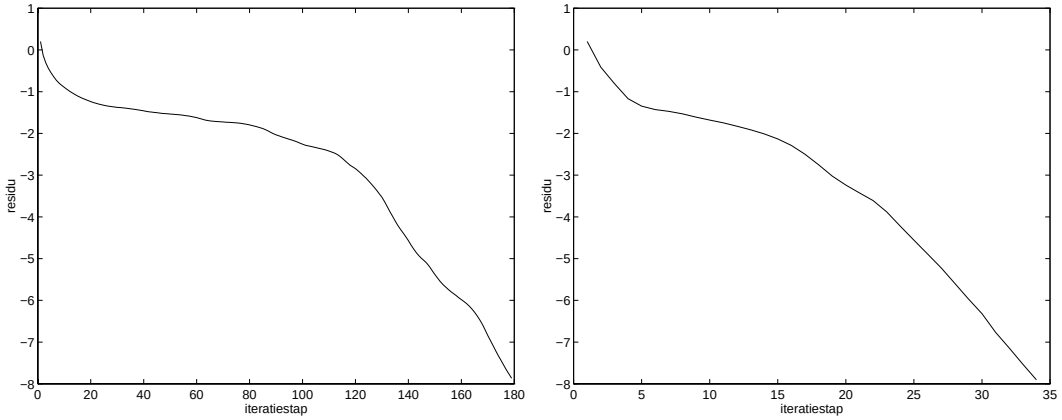

FIG.1: *Convergentie bij symmetrisch probleem V met diagonaal- resp. RILU-preconditionering*

3.4.5 Efficiënt rekenen

In sectie 3.4.2 werd besproken hoe links of rechts werd gepreconditioneerd. Het is ook mogelijk om *tweezijdig te preconditioneren* door $\mathbf{M}$ te factoriseren ($\mathbf{M} = \mathbf{M}_1\mathbf{M}_2$):

$$\mathbf{M}_1^{-1}\mathbf{A}\mathbf{M}_2^{-1}\mathbf{y} = \mathbf{M}_1^{-1}\mathbf{b}, \quad \mathbf{x} = \mathbf{M}_2^{-1}\mathbf{y} \tag{3.12}$$

We gebruiken $\mathbf{D}$ als in (3.9). De tweezijdige preconditionering bouwen we in twee stappen op:

1. We preconditioneren eerst met de diagonaal:

$$\mathbf{D}^{-1}\mathbf{A}\mathbf{x} = \mathbf{D}^{-1}\mathbf{b}. \tag{3.13}$$

   De matrix in het linkerlid, $\widetilde{\mathbf{A}} \equiv \mathbf{D}^{-1}\mathbf{A}$, wordt nog verder gepreconditioneerd. Alle grootheden die voortkomen uit het *diagonaal gepreconditioneerde stelsel* (3.13) voorzien we van een $\widetilde{\ }$.

2. Het stelsel (3.13) preconditioneren we als volgt:

$$(\mathbf{I} + \widetilde{\mathbf{L}}_A)^{-1}\widetilde{\mathbf{A}}(\mathbf{I} + \widetilde{\mathbf{U}}_A)^{-1}\mathbf{y} = \widetilde{\widetilde{\mathbf{b}}} \tag{3.14}$$

   met

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

   We verdelen dus als het ware $\widetilde{\mathbf{M}}^{-1}$, met

$$\widetilde{\mathbf{M}} \equiv (\mathbf{I} + \widetilde{\mathbf{L}}_A)(\mathbf{I} + \widetilde{\mathbf{U}}_A),$$

   over de linker- en rechterkant van $\widetilde{\mathbf{A}}$.

Het verkregen expliciet gepreconditioneerde systeem (3.14) kunnen we oplossen met ongepreconditioneerd GCR. Het enorme voordeel hier, is dat de basisvectoren

$$\mathbf{c}_k = (\mathbf{I} + \widetilde{\mathbf{L}}_A)^{-1}\widetilde{\mathbf{A}}(\mathbf{I} + \widetilde{\mathbf{U}}_A)^{-1}\mathbf{u}_k \tag{3.15}$$

zeer efficiënt berekend kunnen worden. Aangezien $\widetilde{\mathbf{A}} = \widetilde{\mathbf{L}}_A + \widetilde{\mathbf{D}}_A + \widetilde{\mathbf{U}}_A$ geldt namelijk:

$$(\mathbf{I} + \widetilde{\mathbf{L}}_A)^{-1}\widetilde{\mathbf{A}}(\widetilde{\mathbf{U}}_A + \mathbf{I})^{-1} = (\mathbf{I} + \widetilde{\mathbf{L}}_A)^{-1} + (\mathbf{I} + \widetilde{\mathbf{U}}_A)^{-1} + (\mathbf{I} + \widetilde{\mathbf{L}}_A)^{-1}(\widetilde{\mathbf{D}}_A - 2\mathbf{I})(\mathbf{I} + \widetilde{\mathbf{U}}_A)^{-1}.$$

Stap-voor-stap uitgewerkt levert dit met enkele hulpvariabelen, de bepaling van $\mathbf{c}_k$ uit de $\mathbf{u}_k$:

$$\begin{aligned}
&\texttt{solve } (\mathbf{I} + \widetilde{\mathbf{U}}_A)\,\mathbf{u}' = \mathbf{u}_k \\
&\texttt{compute } \mathbf{u}'' = \mathbf{u}_k + (\widetilde{\mathbf{D}}_A - 2\,\mathbf{I})\mathbf{u}' \\
&\texttt{solve } (\mathbf{I} + \widetilde{\mathbf{L}}_A)\,\mathbf{u}''' = \mathbf{u}'' \\
&\mathbf{c}_k = \mathbf{u}' + \mathbf{u}''',
\end{aligned} \tag{3.16}$$

Regel 1, 2, 3 en 4 kosten respectievelijk 2, 1, 2 en 1/2 AXPY, dus de kosten voor het berekenen van $\mathbf{c}_k$ zijn $11n$ flops. De matrixvermenigvuldiging $\mathbf{c}_k = \mathbf{A}\mathbf{u}_k$ kost $10n$ flops en kost dus iets minder. In het gepreconditioneerde GCR moest echter in regel 5 (zie Alg. 2) de $\mathbf{u}_k$ nog uit $\mathbf{M}\mathbf{u}_k = \mathbf{r}_k$ worden bepaald en dit is nu niet meer nodig. De zojuist beproken aanpak heet de *Eisenstat truuk*. In sectie C.3 staat de listing van de routine `pmv(`$\widetilde{\mathbf{A}}, \mathbf{u}, \mathbf{c}$`)`, waarin de Eisenstat truuk is geimplementeerd. De Eisenstat truuk is dus een manier om de preconditioneerder op een expliciete manier te verwerken, maar is zelf geen preconditioneerder. Hiervoor worden de in sectie 3.4.3 besproken preconditioneerders gebruikt.

p	ω	**aant. stap.**	**tijd**
3	0	34	1.65e+0
3	0.5	47	1.95e+0
3	0.97	34	1.75e+0
3	1.0	33	1.68e+0

TABEL 3.4: *Resultaten symmetrisch probleem V voor verschillende preconditioneerders met Eisenstat truuk*

p	ω	**aant. stap.**	**tijd**
3	0	29	1.62e+0
3	0.5	35	1.76e+0
3	0.97	95	3.57e+0
3	1.0	117	5.47e+0

TABEL 3.5: *Resultaten asymmetrisch probleem IV a voor verschillende preconditioneerders met Eisenstat truuk*

3.4.6 Experimenten met expliciet preconditioneren

Er zijn dus verschillende manieren om de preconditioneerder $\mathbf{M}$ te verwerken: impliciet, links of rechts expliciet of tweezijdig.

We kunnen nu impliciete en expliciete preconditionering vergelijken. Vermoedelijk zal het bij gelijke preconditioneerder weinig verschil maken of er expliciet of impliciet wordt geprecconditioneerd. Het bepalen van de $\mathbf{c}_k$'s kost namelijk vrijwel evenveel, behalve dan, dat bij impliciet preconditioneren nog een `msolve` nodig is om $\mathbf{u}_k$ te bepalen (regel 5 in Alg. 2). Vooral bij wat langere iteraties zal de binnenste lus echter domineren over dat klein beetje extra rekentijd dat de `msolve` met zich meebrengt. Hierdoor zullen beide methoden vrijwel even lang duren.

De experimenten, waarvan de resultaten in Tabel 3.2 en 3.3 staan (impliciete preconditionering) zijn hier nog eens met de Eisenstat-truuk uitgevoerd. De resultaten staan in Tabel 3.4 en 3.5. Hierbij staat $p = 3$ voor de expliciete preconditionering met de Eisenstat-truuk.

Bij vergelijkingen met de resultaten in Tabel 3.2 en 3.3 wordt snel duidelijk, dat het verschil tussen impliciete en expliciete preconditionering niet groot is: Bij een bepaald ω zijn meestal ongeveer evenveel iteraties nodig, ook de tijd komt redelijk overeen. Dit betekent dus dat inderdaad het type preconditioneerder veel invloed heeft op de convergentie, terwijl de verwerking ervan (impliciet of expliciet) weinig uitmaakt.

Hoofdstuk 4

Verfijningen

Naarmate de recursie in GCR vordert, wordt het algoritme langzamer. De nieuwe zoekrichting moet op steeds meer vectoren loodrecht gezet worden. In de k-e stap zijn $2k+2$ AXPY's en $k+2$ DOT's nodig om $\mathbf{r}_k$ te berekenen. Ook is er steeds meer geheugen nodig, want alle basisvectoren $\mathbf{c}_i$ en 'zoekrichtingen' $\mathbf{u}_i$ moeten worden opgeslagen. De benodigde hoeveelheid geheugen in de k-e iteratiestap is evenredig met k (nl. $2k+2$).

Om dit effect te verminderen, gebruiken we *korte recursies*. Hierbij worden de nieuwe zoekrichtingen en residuen uit slechts een aantal vorige zoekrichtingen en residuen bepaald. Hiervoor zijn verschillende methoden, waarvan er een aantal in de volgende secties aan bod zal komen.

4.1 Herstarten en afkappen

We kunnen op verschillende manieren gegevens uit slechts een aantal van de vorige iteraties gebruiken. Eén daarvan is *herstarten*.

Bij herstarten wordt niet de lus"`for` $i = 0, \ldots, k-1$ `do`" (regel 7 in Alg.1) uitgevoerd, maar "`for` $i = l_1\lfloor k/l_1 \rfloor, \ldots, k-1$ `do`". Wat er gebeurt is het volgende: k wordt in een lus opgehoogd en geeft de iteratieslag aan. De zoekrichtingen worden loodrecht gezet op alle voorgaand vectoren, totdat $k = l_1$. Dan worden de voorgaande l_1 vectoren 'vergeten' en wordt er begonnen met een nieuwe basis. Per iteratieslag kost de binnenste lus veel minder.

Een tweede techniek is *afkappen*. Hierbij wordt diezelfde regel 7 uit Alg. 1 vervangen door "`for` $i = \max(0, k-l_2), \ldots, k-1$ `do`". Afgezien van de eerste $l_2 - 1$ stappen wordt de nieuwe zoekrichting dus loodrecht gezet op de laatste l_2 gevonden basisvectoren.
Deze aanpak werkt vooral goed bij symmetrische problemen: in de k-e slag hoeft alleen β_k bepaald te worden en de rest niet. Dit wordt verder uitgewerkt in sectie 4.2.1 over *Conjungate Residuals*; de binnenste lus is dan niet meer nodig.

Overigens is de verwachting dat afkappen flink beter presteert dan herstarten. Bij afkappen is er immers steeds een terugkoppeling naar de laatste l_2 vectoren, terwijl herstarten telken opnieuw begint en dus telkens weer met grote fouten start.

Een derde techniek is de zogenaamde *nest-strategie*. Hierbij wordt net als bij preconditioneren een goede zoekrichting gekozen. Bij *nesten* wordt in een klein extra lusje de zoekrichting

met l_4 GCR-slagen bepaald. In het algoritme wordt "$\mathbf{u}_k = \mathbf{r}_k$" (regel 5 in Alg. 1) dus vervangen door: "`solve` $\mathbf{A}\mathbf{u}_k = \mathbf{r}_k$ `by` ℓ_4 `steps of GCR`". Wij hebben deze techniek gecombineerd met het herstarten na l_3 stappen. Dit wordt GMRESR(l_3,l_4) genoemd.

Het is de vraag of het nesten een snellere convergentie levert. Het nesten van lussen is doorgaans niet erg efficiënt! Wanneer we echter de convergentieplaatjes van 'gewone' GCR bekijken, zien we dat GCR langzaam op gang komt, maar wanneer de zoekrichting beter wordt in verdere slagen, de convergentie ineens ook snel gaat. Wanneer nu dus bij nesten meteen al een goed zoekrichting wordt 'aangeleverd', kan dit langzame stuk overgeslagen worden. De convergentie van nest-methode zal vermoedelijk dus toch goed zijn.

Voor alle drie de methoden geldt dat de iteratieslagen duidelijk sneller worden, maar – door het inperken van onze basis (verlies van informatie) – ook onnauwkeuriger. Dit betekent dat er meer iteratieslagen nodig zullen zijn om een klein residu te bereiken. De hoop is dat de korte tijd per iteratieslag domineert en dat deze aanpak dus tijdswinst oplevert.

4.1.1 Tijdsanalyse

Om een eerlijke vergelijking te kunnen maken tussen de drie methoden, zullen we moeten weten wat de kosten per iteratiestap zijn.

Herstarten In de buitenste lus is vóór de start van de binnenste lus 1 ID en 1 MV nodig. Na de binnenste lus nog 3 DOTS en 2 AXPY's. Dit alles gebeurt l_1 keer. In de binnenste lus zijn 1 DOT en 2 AXPY's nodig. Dit wordt $\frac{1}{2}l_1(l_1+1)$ keer uitgevoerd. Gemiddeld over l_1 iteratieslagen is dit dus:

$$\text{ID} + \text{MV} + 3\ \text{DOT} + 2\ \text{AXPY} + \frac{1}{2}(l_1+1)(\text{DOT} + 2\ \text{AXPY})$$

Afkappen Hierbij zijn de kosten in de buitenste lus gelijk aan die bij herstarten. De commando's in de binnenste lus worden nu niet $\frac{1}{2}l_2(l_2+1)$ maar l_2^2 keer uitgevoerd. De kosten voor afkappen voor niet-symmetrische problemen zijn dus:

$$\text{ID} + \text{MV} + 3\ \text{DOT} + 2\ \text{AXPY} + l_2(\text{DOT} + 2\ \text{AXPY})$$

Bij symmetrische problemen, zijn nog optimalisaties te maken, zoals besproken zal worden in sectie 4.2.1.

Herstarten en nesten Hierbij moeten we middelen over $l_3 \cdot l_4$ stappen, omdat voor elk van de l_3 slagen l_4 extra GCR-slagen zijn gemaakt in de geneste aanroep van GCR. De totale kosten zijn gelijk aan die van herstarten *plus* de kosten van de geneste lus. Gemiddeld over $l_3 \cdot l_4$ stappen levert dit:

$$\frac{\text{ID} + \text{MV} + 3\ \text{DOT} + 2\ \text{AXPY} + \frac{1}{2}(l_3+1)(\text{DOT} + 2\ \text{AXPY})}{l_4} + \text{ID} + \text{MV} + 3\ \text{DOT} + 2\ \text{AXPY} + \frac{1}{2}(l_4+1)(\text{DOT} + 2\ \text{AXPY})$$

4.1.2 Geheugengebruik

Zoals reeds gezegd, zijn de 'korte-iteraties-algoritmen' niet alleen sneller, maar vereisen ze ook minder geheugen. We zullen de bestaande algoritmen iets moeten aanpassen om hiervan gebruik te maken.
Bij herstarten laten we de binnenste lus niet lopen van $l_1\lfloor k/l_1 \rfloor$ tot k, maar van 0 tot `k%l1`. Dit heeft hetzelfde effect en de $\mathbf{c}_i$'s en $\mathbf{u}_i$'s kunnen in een array van slechts lengte l_1 opgeslagen worden. Iedere $\mathbf{c}_i, \mathbf{u}_i$ of $\mathbf{x}$ houdt natuurlijk wel lengte $n_x n_y + 1$ omdat de grootte van het probleem niet is veranderd.

Bij afkappen hebben we twee array's van lengte l_2 nodig. Hierbij kan in iedere array telkens één vector vervallen (afkappen) en één nieuwe ingevoegd worden (nieuwe basisvector/zoekrichting).

Bij het nesten zijn kleine array's nodig voor de l_4 extra GCR slagen. Bovendien kunnen deze constant opnieuw worden gebruikt, en dus als extra parameter worden meegegeven. Ze hoeven dan niet telkens opnieuw te worden gealloceerd.

4.1.3 Experimenten

We bekijken eerst een asymmetrisch probleem, Testprobleem IVa. Aangezien de oplosmethoden een stuk langzamer convergeren op asymmetrische problemen is de tolerantie voor het residu op 1.0e-4 gezet. Hierop passen we herstarten na 10 stappen toe op een 30 bij 30 rooster. Om een eerlijke vergelijking te maken tussen de verschillende methoden, moeten de stappen ongeveer evenveel kosten. Daarom nemen we voor afkappen: $l_2 = 7$ en voor het nesten: $l_3 = l_4 = 5$. De convergentieplaatjes voor herstarten en nesten staan in figuur 1. Opvallend is dat het verloop heel anders is, dan de meeste andere convergentieplaatjes van GCR. Dit komt doordat alleen het eerste stukje van GCR wordt uitgevoerd, waarna weer opnieuw wordt herstart. In feite wordt dus telkens het eerste stukje uit een 'gewoon GCR'-plaatje aan elkaar geplakt, zodat vrijwel een rechte lijn ontstaat. Aangezien het eerste stuk van GCR niet zo snel convergeert, is het hele verloop van herstarten niet zo snel: de helling is niet sterk negatief. Om toch nog een beetje goede resultaten te krijgen, moet een goed balans worden gevonden tussen het verkleinen van de lus voor tijdswinst en het vergroten van de lus voor toename van convergentiesnelheid.
Om de invloed van de (a)symmetrie van de matrix op de convergentie van de afkap-methode te bekijken hebben we deze techniek toegepast op Testprobleem 1 (symmetrisch) en 4a (asymmetrisch). De resultaten staan in figuur 2. Bij symmetrische problemen is afkappen zeer snel, maar bij asymmetrische problemen is de convergentie behoorlijk slecht.

Zoals gezegd wordt de snelheid van afkappen bij symmetrische problemen veroorzaakt door het 0 zijn van de β's (uitgezonderd de eerste). Het moet dus ook niet uitmaken op hoeveel basisvectoren de nieuwe zoekrichting loodrecht wordt gezet, aangezien alleen de eerste vector invloed heeft. (Alleen de eerste β is ongelijk aan nul). Uit de experimenten bleek dit ook: voor verschillende l_2 werd symmetrisch probleem I met afkappen opgelost. Voor iedere $l_2 > 0$ (getest is met $l_2 = 1, l_2 = 10$ en $l_2 = 100$) waren 146 stappen nodig bij een tolerantie van 1.0e-8. Natuurlijk wordt de benodigde tijd bij grotere l_2 ook groter en kan dus het beste $l_2 = 1$ worden genomen. Dit wordt verder uitgewerkt in sectie 4.2.1.

Wanneer we vergelijken met de oorspronkelijke GCR, blijkt de convergentie van alle "korte-iteratie-methoden" behoorlijk goed. GCR toegepast op testprobleem IVa met *tol*=1.0e-4 kost weliswaar slechts 301 iteraties, maar dit duurt 25 seconden. De 2000 iteraties van afkappen kosten slechts 8 seconden!

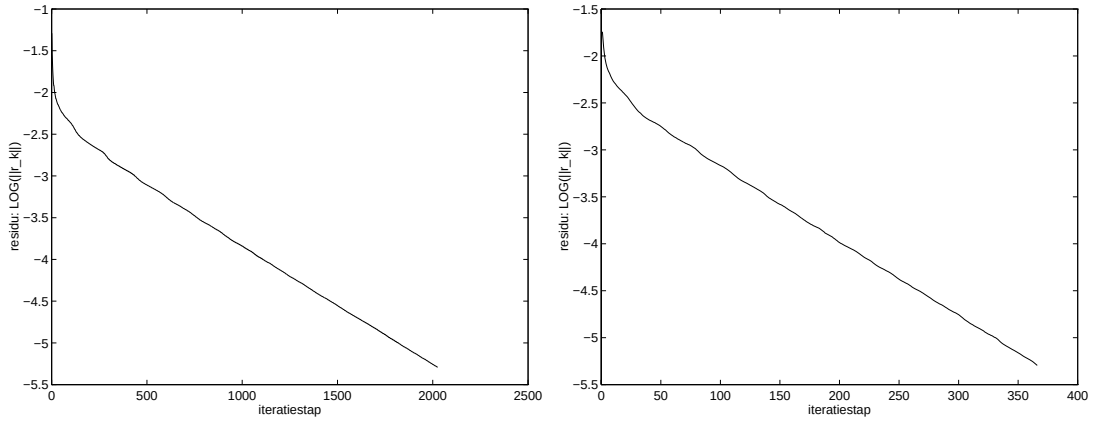

FIG.1: *Convergentie bij een asymmetrisch probleem met herstarten resp. nesten*

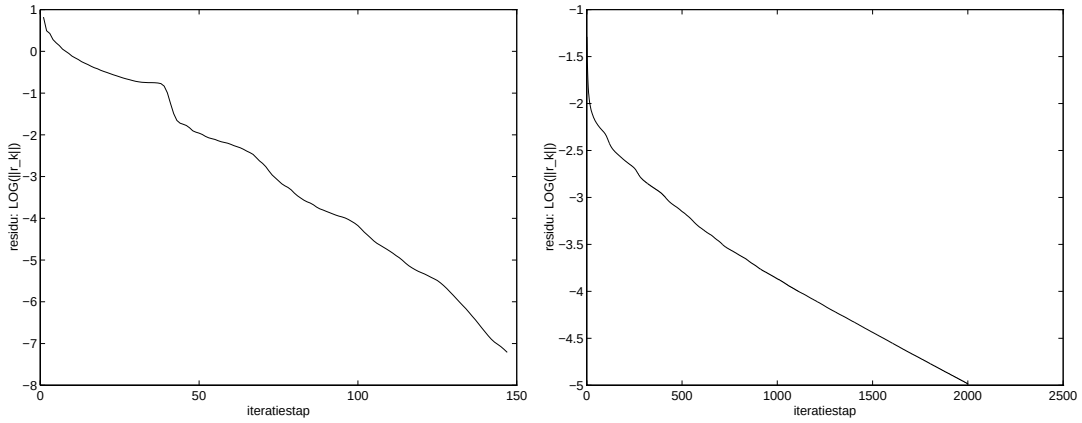

FIG.2: *Convergentie van afkappen bij symmetrisch, resp. asymmetrisch probleem*

De lage kosten per stap wegen dus duidelijk op tegen de toename van het aantal stappen en de techniek van *korte iteraties* blijkt dus veel winst op te leveren.

4.2 Symmetrische problemen

Wanneer[1] $\mathbf{A}$ symmetrisch is ($\mathbf{A}^T = \mathbf{A}$ en $\mathbf{M}$=$\mathbf{I}$ (of wanneer $\mathbf{AM}^{-1}$ symmetrisch is) kunnen er in GCR een aantal rekenstappen, om mathematische redenen, weggelaten worden. Dit omdat van te voren de uitkomst al bekend is. De methoden die zo ontstaan zullen hier behandeld worden.

[1]De verfijningen zijn in teamverband uitgewerkt; De symmetrisch problemen zijn door Gerben Wolterink gedaan ([4]).

4.2.1 Conjungate Residuals (CR)

Om de uitleg te vergemakkelijken wordt voor $\mathbf{M}$ de identiteit genomen, $\mathbf{M}$=$\mathbf{I}$. Dan geldt voor de vectoren uit GCR (na de "`for` $i = 0, \ldots$ `do`"-loop)

$$\mathbf{c}_k = \mathbf{A}\mathbf{r}_k - \beta_1\mathbf{c}_0 - \cdots - \beta_k\mathbf{c}_{k-1} \qquad \text{met} \qquad \beta_{j+1} = \mathbf{c}_j^T\mathbf{A}\mathbf{r}_k/\sigma_j.$$

Aangezien

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

blijkt nu dat, voor symmetrische $\mathbf{A}$,

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

Dit levert een aanzienlijke besparing op in het algoritme omdat in stap k de β_j $(j < k)$ niet meer uitgerekend hoeven worden. Ook de $\mathbf{c}_j$ en $\mathbf{u}_j$ $(j < k-1)$ zijn niet meer nodig. Vanwege de symmetrie en orthogonaliteit geldt verder

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

Hierdoor kan per stap nog een DOT extra bespaard worden. Dit alles leidt tot het CR algoritme (Conjugate Residuals) zoals in Algoritme 3, nu weer met preconditionering $\mathbf{M}$ (eventueel $\mathbf{M} \neq \mathbf{I}$). De preconditionering moet zo zijn dat $\mathbf{A}\mathbf{M}^{-1}$ symmetrisch is. Wanneer $\mathbf{M}$ symmetrisch en positief definiet is kan deze laatste eis vervallen. Er moet dan met een aangepast inproduct gewerkt worden. De $\mathbf{c}_j$'s vormen dan een basis die orthogonaal is t.o.v. het $\mathbf{M}^{-1}$-inproduct. Dan moet $\mathbf{r}_k^T\mathbf{c}_k$ vervangen worden 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.

```
Choose an x_0.
Compute r_0 = b - A x_0.
Put u_{-1} = c_{-1} = 0, ρ_{-1} = 1.
For k = 0, 1, 2, ..., k_max do
      stop if ‖r_k‖ ≤ tol
      solve M u_k = r_k
      compute c_k = A u_k
      ρ_k = r_k^T c_k, β_k = -ρ_k/ρ_{k-1}
      u_k = u_k - β_k u_{k-1}
      c_k = c_k - β_k c_{k-1}
      σ_k = c_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.3: *Conjugate Residuals.*

4.2.2 Conjungate Gradients (CG)

In het geval dat $\mathbf{A}$ positief definiet is (neem $\mathbf{M}$=$\mathbf{I}$) 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 wordt dan $\mathbf{r}_k\mathbf{A}^{-1}\mathbf{c}_k$ etc. bepaald. Uit (3.4) blijkt $\mathbf{A}^{-1}\mathbf{c}_k = \mathbf{u}_k$ en daarom geldt

$$\rho_k = \mathbf{r}_k^T\mathbf{A}^{-1}\mathbf{c}_k = \mathbf{r}_k^T\mathbf{u}_k \quad \text{en} \quad \sigma_k = \mathbf{c}_k^T\mathbf{A}^{-1}\mathbf{c}_k = \mathbf{c}_k^T\mathbf{u}_k.$$

In het geval $\mathbf{M}$=$\mathbf{I}$ geldt $\rho_k = \mathbf{r}_k^T\mathbf{r}_k = \|\mathbf{r}_k\|^2$. De norm van $\mathbf{r}_k$ wordt dus verkregen zonder extra rekenwerk. Omdat voor de berekening van ρ_k $\mathbf{c}_k$ niet meer nodig is kan $\mathbf{c}_k$ uit $\mathbf{u}_k$ berekend worden volgens $\mathbf{c}_k = \mathbf{A}\mathbf{u}_k$ (zie (3.4)), dit bespaart weer een AXPY. Dit leidt tot het beroemde en uiterst efficiënte CG (Conjugate Gradient) algoritme (zie Alg. 4). Voor CG moet $\mathbf{A}$ niet

```
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 = c_k^T u_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.4: *Conjugate Gradients.*

alleen symmetrisch zijn, maar ook positief definiet. De preconditioneerde $\mathbf{M}$ moet ditook zijn. In Alg. 4 is impliciete preconditionering toegevoegd.
In CR wordt, als $\mathbf{M}$=$\mathbf{I}$, het residu in de Euclidische norm geminimaliseerd. In CG is het residu minimaal ten opzichte van het zogenaamde $\mathbf{A}^{-1}$-norm:

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

In Alg. 4 wordt de benaderingskwaliteit beoordeeld in de Euclidische norm terwijl het residu minimaal is in de $\mathbf{A}^{-1}$-norm. Deze aanpak werkt goed en is goedkoop. Dit vanwege het feit dat $\|\mathbf{r}_k\|$ al berekend is en dus nu in feite voor niets verkregen wordt. Terwijl $\|\mathbf{r}_k\|_{\mathbf{A}^{-1}}$ nog bepaald zou moeten worden, hetgeen niet zo makkelijk is.

4.2.3 Preconditioneren

De preconditioneringen uit 3.4.3 zijn symmetrisch als $\mathbf{A}$ dat is. Echter $\mathbf{A}\mathbf{M}^{-1}$ is niet meer symmetrisch, daarom kan niet zomaar CG of CR toegepast worden. Met een tweezijdige preconditionering is de symmetrie te herstellen.
Stel $\mathbf{A} = \mathbf{A}^T$ en $\mathbf{M}$ als in (3.9). Dan geldt $\mathbf{L}_A = \mathbf{U}_A^T$. Wanneer alle elementen van $\mathbf{D}$ positief

zijn kan daarvan de wortel genomen worden. De resulterende matrix is $\mathbf{D}^{1/2}$. Preconditioneren gebeurt weer in twee stappen, alleen wordt de eerste stap, vanwege symmetriebehoud, anders uitgevoerd dan in sectie 3.4.5. De diagonaal-preconditionering gaat als volgt:

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

Het resulterende systeem met matrix $\widetilde{\mathbf{A}} := \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ wordt nog verder gepreconditioneerd. Alle grootheden geassocieerd aan het diagonaal-gepreconditioneerde stelsel worden weer voorzien van een $\widetilde{}$. De tweede stap wordt op dezelfde wijze uitgevoerd als in sectie 3.4.5.
Het resulterende systeem

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

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

4.2.4 Experimenten bij CG

CG kan impliciet en expliciet gepreconditioneerd worden. Bij impliciete preconditionering wordt $\mathbf{M}$ uit (3.9) gebruikt in Alg.4.

Bij expliciete preconditionering wordt Alg.4 toegepast op $\widetilde{\mathbf{A}}$ en $\widetilde{\widetilde{\mathbf{b}}}$. Verder geldt dan $\mathbf{M}$=$\mathbf{I}$. Voor de berekening van compute c=Au wordt gebruik gemaakt van de Eisenstat truuk zoals beschreven in (3.16) (dictaat 33). Beide vormen van preconditionering zijn toegepast op het symmetrische probleem V. De resultaten staan in tabel 4.1 en 4.2.

Het verloop van de grootte van het residu bij impliciete- en expliciete preconditionering is

ω	stappen	tijd (s)
0.4	51	0.40
0.97	37	0.27

TABEL 4.1: *impliciete preconditionering*

ω	stappen	tijd (s)
0.4	51	0.25
0.97	37	0.21

TABEL 4.2: *expliciete preconditionering*

voor beide ω ongeveer gelijk. Daaruit blijkt, dat naarmate de oplossing beter is, de convergentie ook beter wordt. Wanneer deze resultaten vergeleken worden met de andere iteratieve oplosmethoden, blijkt dat PCG de snelste oplosmethode is voor positief definiete problemen. Dit is goed te begrijpen, wanneer men bedenkt dat in elke stap geen (overbodige) lus doorlopen moet worden, hetgeen bij de andere methoden wel het geval is.

4.3 Bi-orthogonale residuen

4.3.1 Bi-orthogonale residuen

Voor[2] symmetrische en positief definiete problemen zijn er verschillende methoden om snel en goedkoop een nauwkeurig resultaat te produceren. Te denken valt aan CG, voor én symmetrische én positief definiete problemen, en aan gepreconditioneerd GCR. Bij GCR nemen

[2]De verfijningen zijn in teamverband uitgewerkt; De bi-orthogonale residuen zijn door Joost Rommes gedaan ([3]).

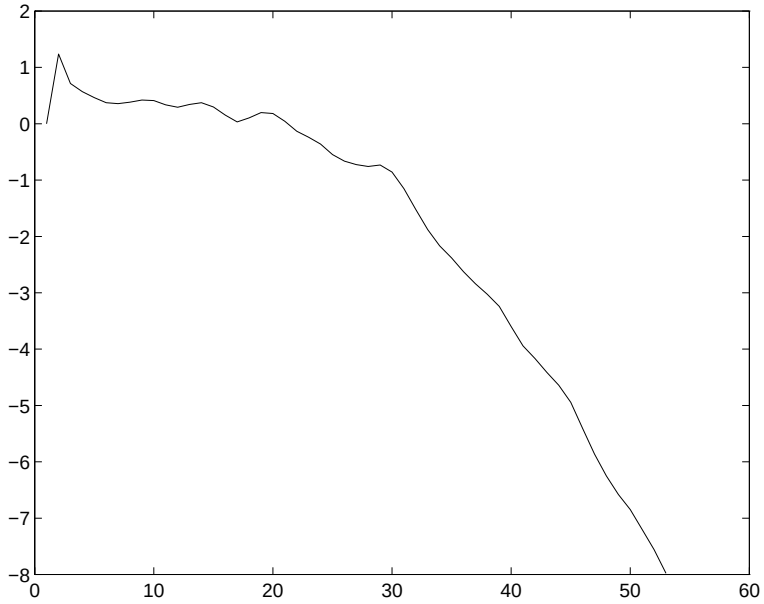

FIG.3: *Convergentieplaatje bij PCG*

echter de kosten toe als het aantal iteratiestappen toeneemt. Aangezien CG niet werkt voor niet-symmetrische problemen zou er dan een keuze gemaakt moeten worden uit per iteratiestap duurder worden GCR of minder nauwkeurigere varianten van GCR als GCR(l_1) en GMRESR(l_3,l_4). Er is echter een andere methode die, met goedkope stappen, niet direct minimale residuen produceert, maar residuen $\mathbf{r}_k$ die loodrecht staan op de k-dimensionale zogenaamde schaduw ruimten. Het idee hierachter is dat voor toenemende k de $\mathbf{r}_k$ in steeds kleinere ruimten worden geconstrueerd. De residuen $\mathbf{r}_k$ zullen op de al eerder besproken wijze worden gecorrigeerd, evenals het bijwerken van $\mathbf{x}_k$:

$$\begin{aligned} \mathbf{r}_{k+1} &= \mathbf{r}_k - \mathbf{A}\mathbf{v}_k, \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + \mathbf{v}_k. \end{aligned} \tag{4.1}$$

Aannemende dat $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$ volgt uit

$$\begin{aligned} \mathbf{r}_{k+1} &= \mathbf{b} - \mathbf{A}\mathbf{x}_{k+1} \\ &= \mathbf{b} - \mathbf{A}(\mathbf{x}_k + \mathbf{v}_k) \\ &= \mathbf{r}_k - \mathbf{A}\mathbf{v}_k = \mathbf{r}_{k+1} \end{aligned}$$

volgens inductie dat de inductie hypothese $\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k$ geldt voor iedere k.

Omdat de residuen $\mathbf{r}_k$ loodrecht gezet worden op de schaduw Krylov deelruimte $\mathcal{K}_k(\mathbf{A}^T; \widetilde{\mathbf{r}}_0)$, is er sprake van bi-orthogonaliteit. Het schaduw beginresidu $\widetilde{\mathbf{r}}_0$ kan op verschillende manieren gekozen worden, gebruikelijk is

$$\widetilde{\mathbf{r}}_0 = \mathbf{r}_0$$

of een random gekozen beginresidu. Zoals bekend behoren de residuen $\mathbf{r}_k$ en de zoekrichting $\mathbf{u}_k$ tot de Krylov deelruimte $\mathcal{K}_{k+1}(\mathbf{A}; \mathbf{r}_0)$. De loodrechtheid van $\mathbf{r}_k$ en de correctie $\mathbf{c}_k := \mathbf{A}\mathbf{u}_k$ op $\mathcal{K}_k(\mathbf{A}^T; \widetilde{\mathbf{r}}_0)$ is te verkrijgen door 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 $\mathcal{K}_k(\mathbf{A}^T; \widetilde{\mathbf{r}}_0) = \text{span}(\widetilde{\mathbf{r}}_0, \widetilde{\mathbf{r}}_1, \ldots, \widetilde{\mathbf{r}}_{k-1})$. Met inductie volgt dan de loodrechtheid op de vectoren $\widetilde{\mathbf{x}}_j$ voor $j < k-1$.

Uit het feit dat $\mathbf{u}_k = \mathbf{r}_k - \beta_k \mathbf{u}_{k-1}$ en dat $\mathbf{c}_k = \mathbf{A}\mathbf{u}_k$ loodrecht op $\widetilde{\mathbf{r}}_{k-1}$ moet komen te staan volgt dat

$$\mathbf{A}(\mathbf{r}_k - \beta_k \mathbf{u}_{k-1}) \cdot \widetilde{\mathbf{r}}_{k-1} = 0$$

en uit deze vorm laat β_k zich makkelijk oplossen:

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

Omdat $\mathbf{r}_k = \mathbf{r}_{k-1} - \alpha_{k-1}\mathbf{c}_{k-1}$ en $\mathbf{r}_k \perp \widetilde{\mathbf{r}}_{k-1}$ volgt op dezelfde wijze dat

$$\alpha_{k-1} = \frac{\widetilde{\mathbf{r}}_{k-1}^T \mathbf{r}_{k-1}}{\widetilde{\mathbf{r}}_{k-1} \mathbf{c}_{k-1}}.$$

Het algoritme om de orthogonale residuen te produceren is Alg.5.

Choose an $\mathbf{x}_0$ and an $\widetilde{\mathbf{r}}_0$.
Compute $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$.
Put $\mathbf{u}_{-1} = 0$, $\mathbf{x} = \mathbf{x}_0$, $\sigma_{-1} = \vartheta_{-1} = 1$.
For $k = 0, 1, 2, \ldots$ do
  stop if $\|\mathbf{r}_k\| \leq tol$
  $\rho_k = \widetilde{\mathbf{r}}_k^T \mathbf{r}_k$, $\beta_k = \rho_k/(\vartheta_{k-1}\sigma_{k-1})$
  $\mathbf{u}_k = \mathbf{r}_k - \beta_k \mathbf{u}_{k-1}$
  compute $\mathbf{c}_k = \mathbf{A}\mathbf{u}_k$
  $\sigma_k = \widetilde{\mathbf{r}}_k^T \mathbf{c}_k$, $\alpha_k = \rho_k/\sigma_k$
  $\mathbf{x} = \mathbf{x} + \alpha_k \mathbf{u}_k$
  $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{c}_k$
  select a $\widetilde{\mathbf{r}}_{k+1} \in \mathcal{K}_{k+2}(\mathbf{A}^T; \widetilde{\mathbf{r}}_0)$
  such that for some $\vartheta_k \neq 0$
  $\widetilde{\mathbf{r}}_{k+1} - \vartheta_k \mathbf{A}^T \widetilde{\mathbf{r}}_k \in \mathcal{K}_{k+1}(\mathbf{A}^T; \widetilde{\mathbf{r}}_0)$
end for

ALG.5: *Bi-orthogonal residuals.*

4.3.2 Bi-Conjugate Gradient (biCG)

Uit ALG.5 kan het Bi-Conjugate Gradient (BiCG) algoritme verkregen worden door voor de schaduw residuen de residuen van de schaduw vergelijking $\mathbf{A}^T\widetilde{\mathbf{x}} = \widetilde{\mathbf{r}}_0$ te nemen. Met β_k en α_k als in ALG.5, en $\vartheta_k = -\alpha_k$, kunnen deze residuen op de gebruikelijke wijze berekend worden:

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

Een nadeel van BiCG is dat er extra rekenwerk nodig is om de schaduw residuen expliciet te bepalen, terwijl deze eigenlijk niet interessant zijn. Al met al kost dat in ieder geval twee AXPY's, twee DOT's en nog eens een MV met $\mathbf{A}^T$ extra.

4.3.3 Bi-Conjugate Gradient Stabilized (BiCGSTAB)

4.3.3.1 Bi-Conjugate Gradient Stabilized: theorie

Om ertoe te komen dat iedere matrix vermenigvuldiging het residu verkleint, moet op de volgende manier tegen het schaduw residu worden aangekeken. Omdat het schaduw residu $\widetilde{\mathbf{r}}_k$ voor een polynoom q_k, van graad k, van de vorm $q_k(\mathbf{A}^T)\widetilde{\mathbf{r}}_0$ is, kan, met $Q_k := q_k(\mathbf{A})$, het volgende geschreven worden:

$$\rho_k = \widetilde{\mathbf{r}}_k^T \mathbf{r}_k = \widetilde{\mathbf{r}}_0^T Q_k \mathbf{r}_k, \tag{4.2}$$

$$\sigma_k = \widetilde{\mathbf{r}}_k^T \mathbf{A}\mathbf{u}_k = \widetilde{\mathbf{r}}_0^T \mathbf{A} Q_k \mathbf{u}_k. \tag{4.3}$$

De Q_k moeten nu zo gekozen worden dat $|Q_k\mathbf{r}_k|$ kleiner dan $|\mathbf{r}_k|$ is. Door $Q_k\mathbf{u}_{k-1}$ en $Q_k\mathbf{r}_k$ met bijbehorende $\mathbf{x}$ te berekenen, kan de verdere berekening in twee delen opgesplitst worden:

Eerst moeten de BiCG stappen met Q_k vermenigvuldigd worden, wat resulteert in

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

waarbij β_k en α_k nu met behulp van (4.2) en (4.3) worden berekend.

Vervolgens wordt in BiCG stabilized (BiCGSTAB) de graad van q_k één opgehoogd door $\omega_k \in \mathbb{R}$ zo te kiezen dat $|(I - \omega_k\mathbf{A})Q_k\mathbf{r}_k|$ minimaal is:

$$Q_{k+1} = (I - \omega_k\mathbf{A})Q_k.$$

Zodoende kunnen $Q_{k+1}\mathbf{u}_k, Q_{k+1}\mathbf{r}_k$ met bijbehorende $\mathbf{x}$ uit $Q_k\mathbf{u}_k, Q_k\mathbf{r}_k$ en $\mathbf{x}$ berekend worden.

Om het Bi-CGSTAB algorithme, met preconditioneerder $\mathbf{M}$, te verkrijgen, ALG.6, is voor $\mathbf{u}_{k-1}, \widehat{\mathbf{u}_k}, \mathbf{c}_k, \mathbf{r}_k$ en $\widehat{\mathbf{r}_k}$ respectievelijk $Q_k\mathbf{u}_{k-1}, Q_k\mathbf{u}_k, \mathbf{A}Q_k\mathbf{u}_k, Q_k\mathbf{r}_k, Q_k\mathbf{r}_{k+1}$ geschreven en zijn vervolgens de indices k verwijderd. Het aantal benodigde $\mathbf{M}$-solves kan naar 2 teruggebracht:

$$\begin{aligned} \widehat{\mathbf{u}} &= \mathbf{M}^{-1}\mathbf{r}; \\ \widehat{\mathbf{r}} &= \mathbf{r} - \alpha\mathbf{c}; \\ \widehat{\mathbf{c}} &= \mathbf{M}^{-1}\mathbf{c}; \\ \mathbf{s}' &= \mathbf{M}^{-1}\widehat{\mathbf{r}} \\ &= \mathbf{M}^{-1}\mathbf{r} - \alpha\mathbf{M}^{-1}\mathbf{c} = \widehat{\mathbf{u}} - \alpha\widehat{\mathbf{c}}. \end{aligned}$$

Wel moet dan het algorithme in een iets andere volgorde worden gezet en moet er opgelet worden dat niet de verkeerde vectoren worden overschreven. De routine PBCGSTAB is te vinden in Appendix C.4. Hierin is behalve de zojuist genoemde verbetering ook het geheugengebruik beperkt.

De rekenkosten van deze implementatie bestaan voor iedere iteratiestap uit 5 DOT's, 7 AXPY's, 2 MV's en 2 msolve's. Aangezien een msolve $11n$ flops kost, komen de totale rekenkosten voor k stappen uit op $k(66n)$ flops.

Bovenop het vaste geheugengebruik voor $\mathbf{A}, \mathbf{x}, \mathbf{b}, \mathbf{D}$ komt nog eens een hoeveelheid van $8n$ doubles voor de verschillende vectoren.

4.3.3.2 Bi-Conjugate Gradient Stabilized: praktijk

Op een rooster van 30 bij 30 roosterpunten is Bi-CGSTAB onderzocht voor symmetrisch probleem V en asymmetrisch probleem IVa. Verder is $\mathbf{M}$ gekozen als in (3.9). Volgens de theorie moet deze methode op beide problemen een goed resultaat geven. Vooral van belang is de eventuele winst in het aantal iteratiestappen, maar ook de benodigde tijd per iteratiestap moet vergeleken worden. In tabel (4.3) en tabel (4.4) staan de respectievelijke resultaten.

Voor probleem V valt op dat de tijd per stap in vergelijking met CG meer is, maar dat het aantal iteratiestappen in het beste geval ($\omega = 1.0$) zo'n 30% lager is. Verder is het tweede norm residu een factor 10 groter, wat komt door de methode van schaduw residuen. De convergentie-plaatjes in figuur (4) zijn ook anders. Wat vooral opvalt zijn de stagnaties: deze worden veroorzaakt door het nagenoeg nul zijn van één of meer van de drie coëfficiënten ρ, σ, ω.

Voor probleem IV A is ten koste van eveneens een grotere tijd per iteratiestap een grote reductie in het aantal iteratiestappen geconstateerd. Dit bevestigt dat Bi-CGSTAB ook

```
Choose an x = x_0 and an r̃_0.
Compute r = b − Ax.
u = 0, σ = ω = 1.
For k = 0, 1, 2, ..., k_max do
    stop if ‖r‖ ≤ tol
    ρ = r̃_0^T r, β = −ρ/(ωσ)
    û = r − β u
    compute c = Aû
    σ = r̃_0^T c, α = ρ/σ
    x̂ = x + α û
    r̂ = r − α c
    compute s = Ar̂
    ω = s^T r̂ / s^T s
    u = û − ω c
    x = x̂ + ω r̂
    r = r̂ − ω s
end for
```

ALG.6: Bi-CGSTAB.

ω (RILU)	# iteratiestappen	tijd (s)
0.0	41	0.62
0.4	35	0.54
0.97	22	0.34
1.0	27	0.42

TABEL 4.3: *Resultaten probleem V met karakteristieke 2de norm residu is* $3.2 \cdot 10^{-9}$

ω (RILU)	# iteratiestappen	tijd (s)
0.0	15	0.24
0.4	18	0.27
0.7	27	0.47

TABEL 4.4: *Resultaten probleem IV A met karakteristieke 2de norm residu is* $7.0 \cdot 10^{-9}$.

op niet-symmetrische problemen goed toepasbaar is. Het patroon in de ontwikkeling van het aantal iteratiestappen is hetzelfde als wanneer gepreconditioneerd GCR wordt toegepast: voor $\omega = 0.0$ wordt het beste resultaat geboekt. De karakteristieke convergentie-plaatjes in figuur (5) laten nog eens duidelijk de stagnaties zien, wat doet vermoeden dat Bi-CGSTAB nog verbeterd kan worden.

Verder was er voor $\omega = 0.97$ en $\omega = 1.0$ geen sprake van convergentie, wat verklaard kan worden met het argument van de genoemde stagnatie: het residu bleef lange tijd nagenoeg onveranderd.

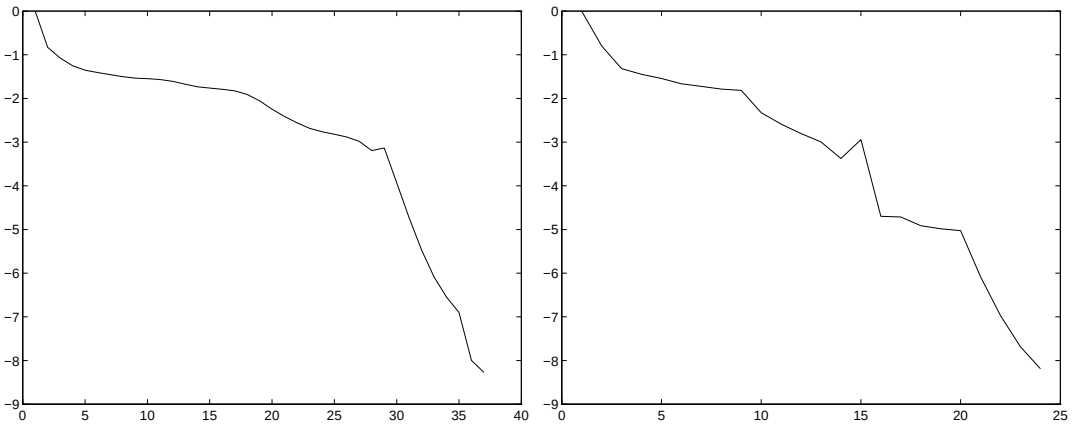

FIG.4: *Convergentieplaatjes voor Bi-CGSTAB toegepast op probleem V met* $\omega = 0.4$ *(l) en* $\omega = 0.97$

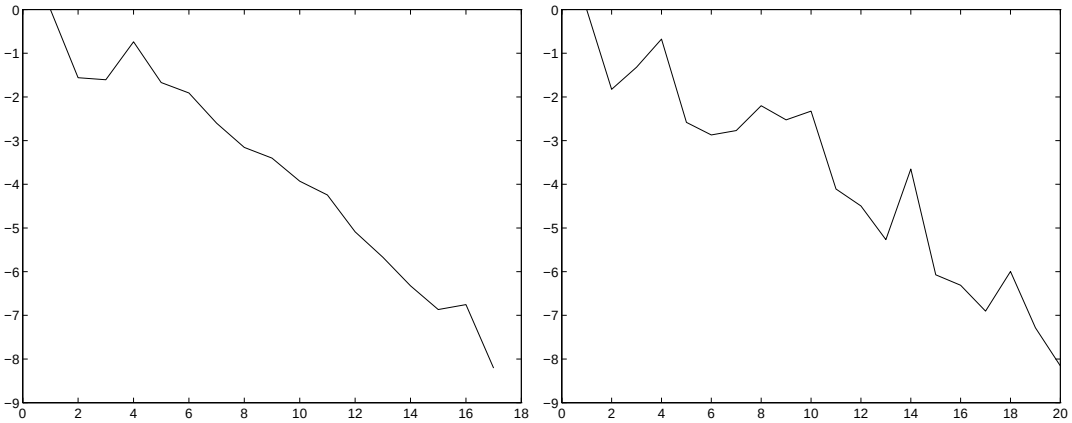

FIG.5: *Convergentieplaatjes voor Bi-CGSTAB toegepast op probleem IV a met* $\omega = 0.0$ *(l) en* $\omega = 0.4$

Hoofdstuk 5

Conclusie

Na bestudering van de vele technieken die allen aanpassingen van GCR zijn, mogen we concluderen dat GCR vooral voor grote problemen een redelijk efficiënte oplosmethode is. Voor kleinere systemen zouden directe oplosmethoden beter zijn, maar hiervoor is alleen een theoretisch voorspelling gedaan.

Het basisidee is eenvoudig: er wordt een basis van de Krylov deelruimte $\mathcal{K}_{k+1}(\mathbf{A}; \mathbf{r}_0)$ gegenereerd en de oplossing $\mathbf{x}_k$ wordt bepaald als het orthogonaal complement. Hierop zijn verschillende verbeteringen te maken.

Het kiezen van een goede startwaarde door preconditionering bleek veel tijdswinst op te leveren: Een factor 5 is goed haalbaar. Ook kon dit bereikt worden door de startwaarde met een extra genest lusje van GCR te bepalen (nesten). Nadeel van (eventueel gepreconditioneerd) GCR is echter, dat de kosten per iteratiestap toenemen naarmate de iteratie vordert. De zoekrichting $\mathbf{u}_k$ moet immers op steeds meer basisvectoren loodrecht gezet worden.

Hiervoor gebruiken we methode met kortere iteraties. Dit kan door zelf de eis op te leggen dat de iteratie korter moet zijn. Herstarten bleek snellere stappen op te leveren, maar er waren duidelijk veel meer stappen nodig door de verhoogde onnauwkeurigheid. Hierdoor is herstarten toch niet zo heel efficiënt. Afkappen werkt een stuk beter, mits het probleem symmetrisch is.

Voor symmetrisch problemen kon de symmetrie-eigenschap nog verder uitgebuit worden, zodat zelfs de binnenste lus uit de oorpsronkelijke GCR verdween. Een tijdswinst van minstens een factor 10 was het resultaat. (CR en CG).

Om niet terug te hoeven vallen naar de dure stappen van GCR, de onnauwkeurigheid van herstarten of afkappen of de symmetrie-eis van CR en CG, is het probleem iets anders bekeken. De bi-orthogonale residuen zijn het resultaat van het loodrecht zetten van de residuen op de schaduw-Krylov deelruimten (BiCG). De $\mathbf{x}_k$ worden nauwelijks beschouwd. De gestabiliseerde versie van BiCG (biCGSTAB),waarbij de reductie van het residu wordt gewaarborgd, blijkt een hele efficiënte en bovendien flexibele oplosmethode te zijn; er worden geen eisen gesteld aan de matrix, als symmetrie of positief definiet. Toch behoudt BiCGSTAB de snelheid die we al bij CG zagen. Hierdoor is BiCGSTAB voor alle problemen zeer snel.

Tot slot nog de opmerking dat hoewel de nadruk in dit practicum op de oplosmethoden lag, er toch wel een aantal toepassingen te bedenken zijn. De resultaten zijn nog niet altijd even accuraat, zeker gezien de vrij grove discretisering, maar toch geven de plaatjes, zoals ze te zien zijn in appendix B een aardig beeld van het fysische systeem.

Referenties

[1] William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery, *Numerical Recipes*, Cambridge University Press, 1992

[2] Gerard L.G. Sleijpen, Computational Science Praktikum*; Iteratieve Lineaire Oplosmethoden*, Universiteit Utrecht, Wiskundig Instituut, Utrecht, 3de herziene druk februari 1998

[3] Joost Rommes, *Iteratieve lineaire Oplosmethoden*, April 1999

[4] Gerben J. Wolterink, *Iteratieve Lineaire Oplosmethoden*, April 1999

Appendix A

De matrix A

Hieronder staat de matrix **A** van het oorspronkelijke stelsel van vergelijkingen uitgeschreven voor $n_x = 4$ en $n_y = 3$. Op alle lege plaatsen horen nullen. Er ontstaat een blokstructuur en **A** is opgebouwd uit $n_y \times n_y$ vierkante blokken met zij de n_x.
Dit werd gebruikt bij het implementeren van een methode, die het systeem vectoriseert.

$$\mathbf{A} = \begin{bmatrix}
\alpha^{c}_{11} & \alpha^{o}_{11} & & & \alpha^{n}_{11} & & & & & & & \\
\alpha^{w}_{21} & \alpha^{c}_{21} & \alpha^{o}_{21} & & & \alpha^{n}_{21} & & & & & & \\
 & \alpha^{w}_{31} & \alpha^{c}_{31} & \alpha^{o}_{31} & & & \alpha^{n}_{31} & & & & & \\
 & & \alpha^{w}_{41} & \alpha^{c}_{41} & & & & \alpha^{n}_{41} & & & & \\
\alpha^{z}_{12} & & & & \alpha^{c}_{12} & \alpha^{o}_{12} & & & \alpha^{n}_{12} & & & \\
 & \alpha^{z}_{22} & & & \alpha^{w}_{22} & \alpha^{c}_{22} & \alpha^{o}_{22} & & & \alpha^{n}_{22} & & \\
 & & \alpha^{z}_{32} & & & \alpha^{w}_{32} & \alpha^{c}_{32} & \alpha^{o}_{32} & & & \alpha^{n}_{32} & \\
 & & & \alpha^{z}_{42} & & & \alpha^{w}_{42} & \alpha^{c}_{42} & & & & \alpha^{n}_{42} \\
 & & & & \alpha^{z}_{13} & & & & \alpha^{c}_{13} & \alpha^{o}_{13} & & \\
 & & & & & \alpha^{z}_{23} & & & \alpha^{w}_{23} & \alpha^{c}_{23} & \alpha^{o}_{23} & \\
 & & & & & & \alpha^{z}_{33} & & & \alpha^{w}_{33} & \alpha^{c}_{33} & \alpha^{o}_{23} \\
 & & & & & & & \alpha^{z}_{43} & & & \alpha^{w}_{43} & \alpha^{c}_{43}
\end{bmatrix} \tag{A.1}$$

Voor $n_x = n_y = 5$ staat in figuur 1 een MatLab-plaatje van de niet-nul elementen van de matrix **A**.

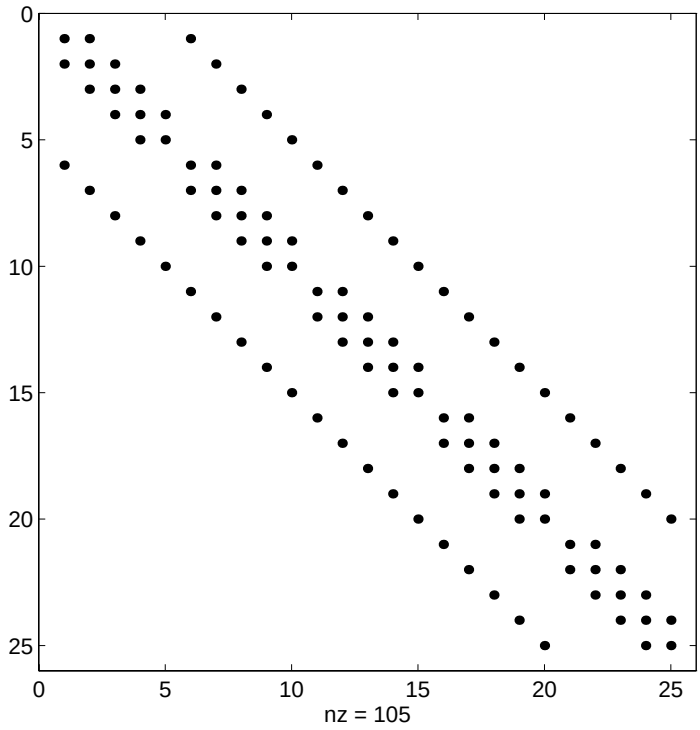

FIG.1: *Niet-nul elementen in de matrix* $\mathbf{A}$ *voor* $n_x = n_y = 5$

Appendix B

Testproblemen

Voor het experimenteren met de verschillende iteratieve oplosmethoden zijn onderstaande testproblemen gebruikt.

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 koncentratie 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 gemengde randvoorwaarden waarbij de uitstroom evenredig is met de koncentratrie ($\mu = 0.5$). In oost en west hebben we essentiële randvoorwaarden. ψ_0 is 0 op de hele rand ∂D. Verder is $a = b = 6$ ten westen van de rivier en $a = b = 4$ ten oosten, $c = 0$.
A. Voor het vektorveld (u, v) nemen we in eerste instantie $u = (y - 1000)/1000$, $v = (1500 - x)/1000$. (Bekijk ook eens het probleem met natuurlijke randvoorwaarden op de noordelijke – en zuidelijke rand.)
B. Neem tenslotte voor het vektorveld (u, v) het vektorveld geassocieerd aan de oplossing van III (berekend m.b.v. (2.12)) (weer met gemengde randvoorwaarden op noordelijke – en zuidelijke rand).

Testprobleem V $X = Y = 3000$. $a = b = 5$ 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 $1900 \leq y \leq 2100$, $\mu = 1$ & $\psi_0 = 0$ elders. Op de oostrand: $\mu = 0$ & $\psi_0 = 200$ voor $400 \leq y \leq 600$, $\mu = 1$ & $\psi_0 = 0$ 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 0.7 m^3/dag m.

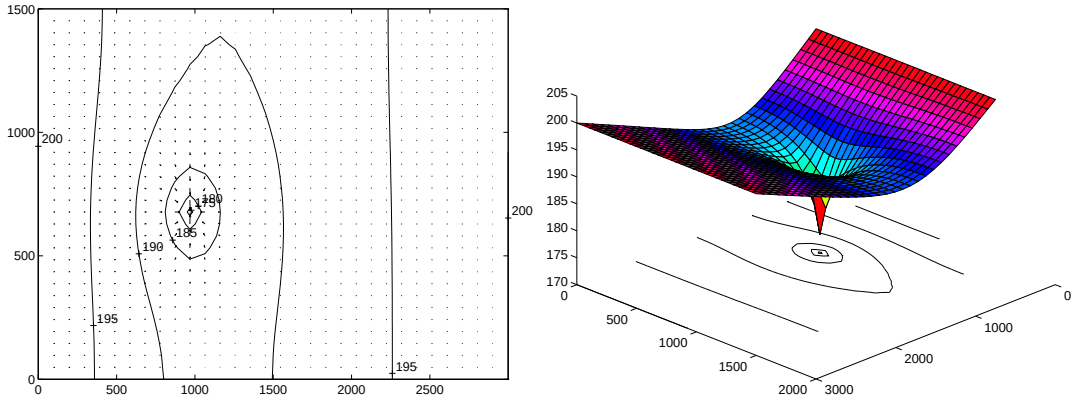

FIG.1: *Het stromingsveld (l) en de gifconcentratie (r) op het bekeken gebied voor probleem I.*

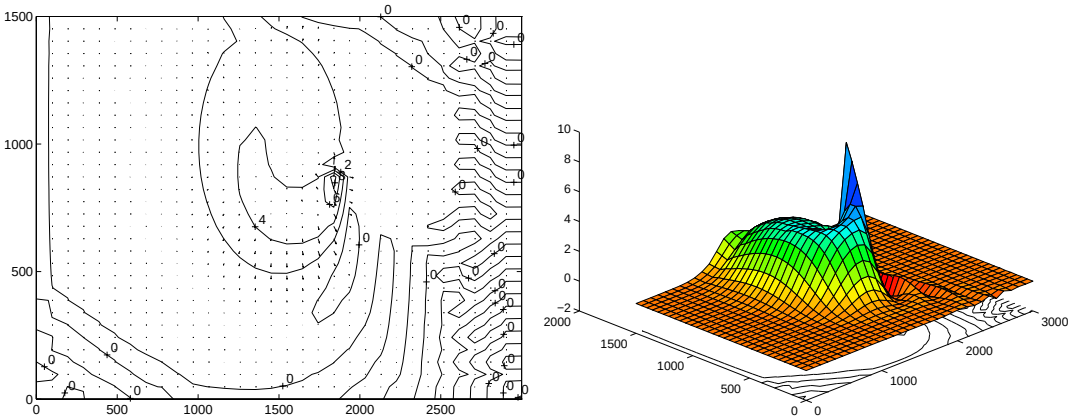

FIG.2: *Het stromingsveld (l) en de gifconcentratie (r) op het bekeken gebied voor probleem IVa.*

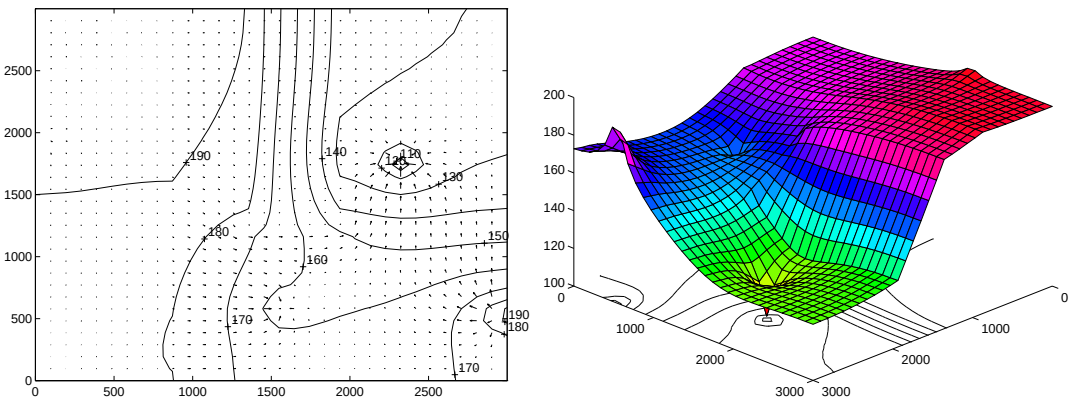

FIG.3: *Het stromingsveld (l) en de grondwaterdruk (r) op het bekeken gebied voor probleem V*

Appendix C

Listings

C.1 De routine `msolve`

De routine `msolve` lost $\mathbf{u}$ op uit $\mathbf{Mu} = \mathbf{r}$ met $\mathbf{M}$ als in (3.9) en gebruikt hiervoor enkele oplosroutines.

```
void msolve(double *D, double *A, int *ja, double *r, double *u){
  double *u1, *u2;
  u1=vecallocd(ja[1]-2+1);
  u2=vecallocd(ja[1]-2+1);
  lowerSolve(D, A, ja, r, u1);
  diagMV(D,ja[1]-2,u1,u2);
  upperSolve(D, A, ja, u2, u);
  free(u1);
  free(u2);
}

/* Solve (La+D)*u = r  */
void lowerSolve(double *D, double *A, int *ja, double *r, double *u){
  int k, jb, je, j;
  int n = ja[1]-2;
  u[1] = r[1]/D[1];
  for(k = 2; k <= n; k++){
    jb = ja[k];
    je = ja[k+1];
    u[k] = r[k];
    for(j = jb; j < je; j++){
      if(ja[j] < k)
        u[k] -= A[j]*u[ja[j]];
    }
    u[k] /= D[k];
  }
}

/* Solve (Ua+D)*u = r  */
void upperSolve(double *D, double *A, int *ja, double *r, double *u){
  int k, jb, je, j;
  int n = ja[1]-2;
  u[n] = r[n]/D[n];
  for(k = n-1; k >=1; k--){
    jb = ja[k];
```

```
    je = ja[k+1];
    u[k] = r[k];
    for(j = jb; j < je; j++){
      if(ja[j] > k)
       u[k] -= A[j]*u[ja[j]];
    }
    u[k] /= D[k];
  }
}

/* Compute r = D*u  (D is a diagonalmatrix) */
void diagMV(double *D, int n, double *u, double *r){
  int i;
  for(int i = 1; i <= n; i++)
    r[i] = D[i]*u[i];
}
```

C.2 De routine `rilu`

De routine `rilu` gebruikt de routine `mtxElement`. Hiervan is ook de listing afgedrukt.

```
/* Returnt matrix-element op i-e rij j-e kolom
   uit matrix A in row-indexed sparse storage mode */
double mtxElement(int i, int j, double *A, int *ja){
  int jb, je, n = ja[ja[1]-1]-1, k;
  if(i < 1 || j < 1 || i > n || j > n)
    return 0.0;

  jb = ja[i];
  je = ja[i+1]-1;

  for(k = jb; k <= je; k++){
    if(ja[k]==j)
      return A[k];
  }
  return 0.0;
}

/* Bepaalt matrix D, die nodig is voor preconditioneerder M */
void rilu(double *D, double *A, int *ja, double omega){
  int k, n=Grid.x*Grid.y, nx = Grid.x;
  double a1, a2, a3, a4, a5, a6;
  D[1] = A[1];
  for(k = 2; k <= n; k++){
    a1 = mtxElement(k,k-1,A,ja);
    a2 = mtxElement(k-1,k,A,ja);
    a3 = mtxElement(k,k-nx,A,ja);
    a4 = mtxElement(k-nx,k,A,ja);
    a5 = mtxElement(k-1,k+nx-1,A,ja);
    a6 = mtxElement(k-nx,k-nx+1,A,ja);
    D[k] = A[k];
    if(D[k-1]!=0.0){
      D[k] -= a1*a2/D[k-1];
      D[k] -= omega*a1*a5/D[k-1];
    }
    if(k > nx){
```

```
      if(D[k-nx] != 0.0){
        D[k] -= a3*a4/D[k-nx];
    D[k] -= omega*a3*a6/D[k-nx];
      }
    }
  }
}
```

C.3 De routine pmv

De routine pmv implementeert de Eisenstat-truuk (sectie 3.4.5) en gebruikt hiervoor de oplosroutines upperISolve en lowerISolve.

```
void pmv(double *At, int *ja, double *u, double *c){
  double *u1, *u2, *u3;
  int i, n = ja[1]-2;
  u1=vecallocd(ja[1]-2+1);
  u2=vecallocd(ja[1]-2+1);
  u3=vecallocd(ja[1]-2+1);

  upperISolve(At, ja, u, u1);
  for(i = 1; i <= n; i++)
    u2[i] = u[i] + (At[i] - 2)*u1[i];
  lowerISolve(At, ja, u2, u3);
  for(i = 1; i <= n; i++)
    c[i] = u1[i] + u3[i];
  free(u1);
  free(u2);
  free(u3);
}

/* Solve (Ua~ +I)*u = r  */
void upperISolve(double *At, int *ja, double *r, double *u){
  int k, jb, je, j;
  int n = ja[1]-2;
  u[n] = r[n];
  for(k = n-1; k >= 1; k--){
    jb = ja[k];
    je = ja[k+1];
    u[k] = r[k];
    for(j = jb; j < je; j++){
      if(ja[j] > k)
       u[k] -= At[j]*u[ja[j]];
    }
  }
}

/* Solve (La~ +I)*u = r  */
void lowerISolve(double *At, int *ja, double *r, double *u){
  int k, jb, je, j;
  int n = ja[1]-2;
  u[1] = r[1];
  for(k = 2; k <= n; k++){
    jb = ja[k];
    je = ja[k+1];
    u[k] = r[k];
```

```
    for(j = jb; j < je; j++){
      if(ja[j] < k)
        u[k] -= At[j]*u[ja[j]];
    }
  }
}
```

C.4 De routine BiCGSTAB

```
void BiCGSTAB(double *D, double *A,int *ja,double *b,int kmax, double tol,double *x)
{
   int k,i,n=ja[1]-2;
   double *r,*r0, *u, *ud, *c, *cd, *s, *sd, sigma, rho, alpha, beta, omega;
   FILE *fp;

   printf("\n maximaal %d cg slagen, nauwkeurigheid %9.2E\n",kmax,tol);
   fp=fopen("UITVOER/rho.m","w");
   fprintf(fp," rh=[ ");

   r=vecallocd(n+1);
   u=vecallocd(n+1);
   ud=vecallocd(n+1);
   r0=vecallocd(n+1);
   cd=vecallocd(n+1);
   s=vecallocd(n+1);
   sd=vecallocd(n+1);
   for(i=1;i<=n;i++)u[i]=0.0;
   i=1;
   c=vecallocd(n+1);

   MV(A,ja,x,r);
   axpy(n,-1.0,r,b,r);
   id(n,r,r0);
   rho=1.0;
     printf("\n |r_%d|=%9.2e ",k,sqrt(rho));
     fprintf(fp,"%9.2e ",sqrt(rho));
   tol=tol * tol * rho;
   sigma=omega=1.0;

   for (k=0;k<kmax;k++){
if (dot(n,r,r)<tol) break;

rho=dot(n,r0,r);
beta=-rho/(omega*sigma);
msolve(D,A,ja,r,ud);
axpy(n,-beta,u,ud,u);
MV(A,ja,u,c);
sigma=dot(n,r0,c);
alpha=rho/sigma;
axpy(n,alpha,u,x,x);
axpy(n,-alpha,c,r,r);

msolve(D,A,ja,c,cd);
axpy(n,-alpha,cd,ud,sd);
MV(A,ja,sd,s);
```

```
omega=dot(n,s,r)/dot(n,s,s);
axpy(n,-omega,cd,u,u);
axpy(n,omega,sd,x,x);
axpy(n,-omega,s,r,r);

rho=dot(n,r,r);
        printf("\n |r_%d|=%9.2e ",k,sqrt(rho));
        fprintf(fp,"%9.2e ",sqrt(rho));
   }

  fprintf(fp,"]; plot(log10(rh)) \n\n ");
  fclose(fp);
  free(r); free(u); free(c);
}
```