Numerieke Wiskunde 3
Numerieke Lineaire Algebra

H.A.van der Vorst

12 december 1997

Hoofdstuk 1

Inleiding

Vrijwel ieder groot rekenprobleem maakt gebruik van technieken uit de numerieke lineaire algebra, bijvoorbeeld:

- Het oplossen van stelsels lineaire vergelijkingen. Deze stelsels kunnen in zeer uiteenlopende toepassingen ontstaan: oplossen van gediscretiseerde partiële differentiaal vergelijkingen, mechanische constructies, reservoir simulatie. Facetten die een rol spelen bij het ontwerp en de analyse van algoritmen zijn: efficiency ten aanzien van rekentijd en geheugengebruik; nauwkeurigheid en stabiliteit.

- Kleinste kwadraten problemen. Voorbeelden: het bepalen van parameters uit meetwaarden, beeldreconstructie. Nieuwe facetten: bepaling van de beste oplossing (uit vele mogelijkheden), rang van matrix.

- Lineaire eigenwaarden problemen. Deze kunnen we ook op vele plaatsen tegenkomen: energieniveaus van electronen rond een atoomkern, eigenfrequenties van mechanische constructies, stabiliteits analyses, onderzoek convergentiegedrag van iteratieve methoden.

- Matrixfuncties. Deze treden op in de netwerk theorie, systeemanalyse en bij het oplossen van stelsels gewone differentiaalvergelijkingen (bij hogere orde impliciete methoden).

De op te lossen problemen kunnen zeer uiteenlopend van aard zijn, hetgeen aanleiding geeft tot allerlei verschillende, op de probleemklasse toegesneden, algoritmen. De stelsels kunnen bovendien zeer groot zijn: 1000000 vergelijkingen met evenzovele onbekenden is al geen uitzondering meer. Dit legt heel bijzondere eisen op aan de te gebruiken methoden.
Vaak moet men een groot aantal gelijksoortige problemen oplossen (tijdsafhankelijke problemen, modelstudies, niet-lineaire stelsels), waarbij men zich kan afvragen of deze problemen geen gemeenschappelijk rekenwerk omvatten, en of er algoritmen bestaan die daar handig gebruik van maken.

Rekenfouten, ten gevolge van het rekenen in eindige precisie, en modelfouten spelen een zeer belangrijke rol. Zij kunnen er toe leiden dat stelsels singulier lijken te worden, of dat de gevonden oplossing belangrijk afwijkt van de exacte oplossing. Om practische redenen is het lang niet altijd mogelijk om het meest nauwkeurige of meest stabiele algoritme te gebruiken; men is vaak gebonden aan de grenzen die door de rekencapaciteit gesteld worden aan bijv. rekentijd en geheugenomvang.

Al dit soort overwegingen hebben aanleiding gegeven tot algoritmen die zeer specifiek zijn ontworpen voor klassen van problemen, bijvoorbeeld: grote ijle M-matrices, bandmatrices, symmetrische positief definiete matrices, Toeplitz matrices, etc.
Tegenwoordig moet men ook in toenemende mate rekening houden met de speciale architectuur van computers (vectorcomputers, parallelle systemen), immers deze maken het in principe mogelijk nog weer grotere problemen op te lossen. Soms staan de speciaal op deze machines toegesneden efficiënte algoritmen op gespannen voet met eisen ten aanzien van stabiliteit en nauwkeurigheid. Voor een uitvoerig overzicht van de aanpak van lineaire algebra problemen op dit soort computers zie [6].

Een belangrijke deelklasse van oplostechnieken wordt gevormd door de zogenaamde iteratieve methoden. De grondgedachte hierbij is dat men met weinig werk een verkregen benadering voor de oplossing wenst bij te werken. Hierbij doemen vragen op ten aanzien van convergentie en criteria om het proces op een geschikt moment af te breken.

Voor zeer veel rekenproblemen is software vervaardigd. Men moet er op bedacht zijn dat de kwaliteit daarvan zeer uiteenlopend kan zijn. Afgezien van programmeerfouten wordt de gebruiker slechts zelden in staat gesteld zich te vergewissen van de onzekerheden in de opgeleverde oplossing. Doel van dit college is ondermeer er voor te zorgen dat men beter in staat is een afgewogen keuze te maken uit de vele mogelijkheden en inzicht te doen verwerven in de mogelijkheden en problemen van diverse algoritmen. Aangezien het zelf vervaardigen van betrouwbare en robuuste software een moeilijke en tijdrovende affaire is, zal men daar alleen in strikt noodzakelijke gevallen zijn toevlucht toe moeten nemen.

Hoofdstuk 2

Een overzicht van numerieke software

In dit overzicht beperken we ons tot het noemen van de belangrijkste en meest gebruikte software collecties (libraries).
Grote programmatheken met wiskundige software worden geleverd door

- IMSL. Voor het gebruik van deze programmatheek is een licentie vereist. Bij de meeste wetenschappelijke rekencentra is de collectie beschikbaar voor de geregistreerde klanten.
- NAG. Dit is de Engelse tegenhanger van het Amerikaanse pakket IMSL. Ook deze programmatheek is alleen onder licentie te gebruiken. Voor de numeriek deskundige gebruiker bevat zij veel software waarin allerlei keuzen nog te maken zijn, in tegenstelling tot IMSL waar meestal een parameter keuze gemaakt is (die in de meeste gevallen dan ook goed voldoet).
- Harwell Subroutine Library. Ook hier is een licentie vereist, maar voor niet-commercieel gebruik zijn daar geen kosten aan verbonden. De programmatheek bevat ondere andere interessante subroutines voor het direct oplossen van lineaire stelsels met ijle matrices.
- MATLAB. Dit is een interactief pakket voor het manipuleren met matrices. Het pakket is bedoeld om de constructie van algoritmen, het toetsen van nieuwe ideeën en het testen van bestaande algoritmen te vergemakkelijken. Het pakket is onder licentie beschikbaar voor uiteenlopende systemen, van personal tot super computer.

Dan zijn er verder nog grote hoeveelheden software vrij verkrijgbaar. We noemen hier met name die software die van hoge kwaliteit is en die van zeer groot belang is voor problemen uit de numerieke lineaire algebra. Al deze software is eenvoudig te verkrijgen via een electronic mail service. Dat werkt in principe als volgt. Een gebruiker zendt vanaf zijn of haar workstation (of met het net verbonden pc) een elctronic mail boodschap naar

```
netlib@ornl.gov      (internet)
```

De te verzenden boodschap dient een van de volgende regels te bevatten:

```
send index

send index from {library}

send {routines} from {library}
```

Men krijgt dan het gevraagde automatisch, dat wil zeggen zonder menselijke tussenkomst, per electronic mail toegezonden naar het workstation waarvanaf men het verzoek verzonden heeft.

De volgende collecties zijn via "netlib" verkrijgbaar:

- LINPACK. Een zeer uitgebreide en goed uitgeteste verzameling FORTRAN subroutines, voor het oplossen van lineaire stelsels vergelijkingen. MATLAB heeft deze subroutines als kernen in haar systeem. De meeste in dit dictaat besproken directe oplostechnieken zijn in subroutine vorm in LINPACK aanwezig. LINPACK behoort bij een aantal computers standaard tot het systeem (o.a. CRAY, Convex, Alliant). Het pakket is zeer zorgvuldig beschreven in [5].

- EISPACK. Dit is de tegenhanger van LINPACK voor lineaire eigenwaarde problemen. Ook dit pakket is op een aantal computer systemen standaard beschikbaar. Het is zeer uitgebreid gedocumenteerd in [22].

- ITPACK. De iteratieve methoden, waaronder die welke in dit dictaat besproken worden, worden gedekt door dit pakket. Documentatie kan men vinden in [12].

- BLAS. De zogenaamde Basic Linear Algebra Subprograms zijn gedefinieerd om dienst te doen als rekenintensieve kernen in een groot aantal lineaire algebra subroutines. Als zodanig komt men ze dan ook tegen in pakketten als LINPACK, EISPACK, etc. We kennen de BLAS tegenwoordig in drie varianten: BLAS1 (operaties op vector-vector nivo, zoals een inprodukt), BLAS2 (operaties op matrix-vector nivo) en BLAS3 (operaties op matrix-matrix nivo). De BLAS2 en BLAS3 modulen zijn recentelijk in de plaats gekomen van de oorspronkelijke BLAS, om beter tegemoet te komen aan de mogelijkheden van moderne computer architecturen. Voor een overzicht van deze kernen, het gebruik ervan en verdere verwijzingen, zie [6].

- TOMS (ACM). In het vakblad Transactions on Mathematical Software worden regelmatig nieuwe subroutines voor recent ontworpen algoritmen gepubliceerd (vaak om het betreffende algoritme vast te leggen). Doorgaans is deze software ook via "netlib" te verkrijgen, dus wacht met overtikken van de gepubliceerde tekst.

- Templates [1]. Dit is een verzameling basis-routines voor het oplossen van lineaire stelsels met iteratieve methoden. Het boek waarin de templates worden besproken is electronisch verkrijgbaar via, bijvoorbeeld,
`http://www.math.ruu.nl/people/vorst` Klik op deze Web-pagina de Publications 94 aan, en dan krijgt men de toegang tot het boekje. In het boek zelf vindt men hoe de desbetreffende Fortran, Matlab en C++ codes te verkrijgen zijn.

Hoofdstuk 3

Aanbevolen Lectuur

De volgende boeken sluiten goed aan op de stof uit het dictaat en in vele gevallen is materiaal aan deze boeken ontleend voor deze tekst.

- Golub and Van Loan [10]. Het college is sterk op dit boek geïnspireerd en het is aanbevolen lectuur voor hen die vaker met problemen uit de numerieke lineaire algebra in aanraking komen. De tweede druk van het boek gaat ook nader in op de problemen die opdoemen bij het verwerken op parallelle en vector computers.

- Wilkinson [26]. Dit is het grote standaardwerk op numerieke lineaire algebra gebied, met name op het gebied van de foutenanalyse.

- Varga [25]. Alhoewel het boek de moderne iteratieve methoden natuurlijk niet behandelt, bevat het toch vrijwel al het basismateriaal voor de analyse van deze methoden.

- Parlett [17]. Een zeer leesbaar boek met zeer veel aandacht voor moderne iteratieve methoden voor het bepalen van eigenwaarden (machtsmethode, Lanczos).

- Dongarra et al. [6]. In dit boek wordt de nadruk gelegd op implementatie aspecten van numerieke lineaire algebra algoritmen voor moderne supercomputers. Het boek bevat veel voorbeelden van verwerking op vector computers (zoals CRAY, NEC, CONVEX en Alliant).

Hoofdstuk 4

Basis begrippen en Notaties

Een goed overzicht van de meest elementaire begrippen uit de numerieke lineaire algebra vindt men ook in [23].

Met $R^{m \times n}$ geven we de lineaire ruimte aan van alle m bij n reële matrices over $I\!R$:

$$A \in R^{m \times n} \Leftrightarrow A = (a_{i,j} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}.$$

In het bijzonder: een vector $x \in R^{m \times 1} = R^m$. Vectoren zullen we steeds met kleine letters aangeven, matrices met hoofdletters.
Eenvoudige matrix operaties:

- Optelling.
$$C = A + B \quad \text{met} \quad A, B, C \in R^{m \times n} \ : \quad c_{i,j} = a_{i,j} + b_{i,j}$$

- Scalaire vermenigvuldiging.
$$C = \alpha A \ : \ c_{i,j} = \alpha a_{i,j}$$

- Matrix-Matrix vermenigvuldiging.
$$C = AB \quad \text{met } A \in R^{m \times n}, B \in R^{n \times p}, C \in R^{m \times p} : \quad c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}$$

- I.h.b. Matrix-vector vermenigvuldiging.
$$y = Ax, \quad \text{met } A \in R^{m \times n}, x \in R^n, y \in R^m, \quad : \ y_i = \sum_{k=1}^{n} a_{i,k} x_k$$

- Spiegeling.
$$C = A^T \ : \ c_{i,j} = a_{j,i}$$

- Differentiëren.
$$C = (c_{i,j}(t)) \ : \ \frac{\partial C}{\partial t} = \left(\frac{\partial c_{i,j}(t)}{\partial t}\right)$$

Als $A \in I\!R^{n\times n}$ dan heet A *vierkant*. De (vierkante) eenheidsmatrix I_n wordt gedefinieerd door $I_{i,i} = 1$ en $I_{i,j} = 0$ als $i \neq j$. Als de orde uit de context blijkt dan wordt de index n meestal weggelaten.

Inverse: Als $A, B \in I\!R^{n\times n}$ en $AB = I$, dan is B de inverse van A en heet A *nietsingulier*. De inverse van A wordt genoteerd als A^{-1}. Met A^{-T} geven we aan $(A^{-1})^T$. Ga na dat $(A^{-1})^T = (A^T)^{-1}$.

Inprodukt van twee vectoren: Voor $x, y \in R^n$ wordt het inprodukt gedefinieerd als
$$x^T y = \sum_{i=1}^{n} x_i y_i$$
Dit is een zogenaamde BLAS-1 operatie (zie Hoofdstuk 2).
Voor $x, y \in C^n$ definiëren we het inprodukt als
$$x^H y = \sum_{i=1}^{n} \bar{x}_i y_i.$$

Het uitprodukt van 2 vectoren: Het uitproduct van $x \in R^m$ en $y \in R^n$ leidt tot de volgende rang 1 matrix:
$$xy^T = \begin{pmatrix} x_1y_1 & x_1y_2 & \cdots & x_1y_n \\ x_2y_1 & x_2y_2 & \cdots & x_2y_n \\ \vdots & \vdots & \ddots & \vdots \\ x_my_1 & x_my_2 & \cdots & x_my_n \end{pmatrix}$$

Voor rang 1 correcties van een matrix A geldt, mits alle inverses bestaan, de formule van *Sherman-Morrison* [10]:
$$(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u}.$$
Bewijs deze formule zelf.

Bij het opstellen en kiezen van geschikte algoritmen is een aantal matrix eigenschappen en matrixvormen van belang. We beginnen met het noemen van de belangrijkste matrixvormen.

- Bandmatrices: $A \in R^{m\times n}$ heeft onderbandbreedte r en bovenbandbreedte s als $a_{i,j} = 0$ voor $i > j + r$ en $j > i + s$.
- Diagonaal matrix: $a_{i,j} = 0$ als $i \neq j$. Handige notatie:
  $A = diag(\lambda_1, \ldots, \lambda_k)$, met $k = \min(m, n)$.
- Tridiagonaal matrix: bandbreedte $s = r = 1$.

- (Boven)bidiagonaal: $r = 0$, $s = 1$.
- Bovendriehoeksmatrix: $a_{i,j} = 0$ voor $i > j$.
- Strikt bovendriehoeksmatrix: $a_{i,j} = 0$ voor $i \geq j$.
- Boven Hessenberg matrix: $a_{i,j} = 0$ voor $i > j + 1$.
- Blokmatrix: Een in blokken gepartitioneerde matrix

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,q} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,q} \\ \vdots & \vdots & \ddots & \vdots \\ A_{p,1} & A_{p,2} & \cdots & A_{p,q} \end{pmatrix}.$$

  Hierbij geldt $A_{i,j} \in R^{m_i \times n_j}$, dus $A \in R^{\Sigma m_i \times \Sigma n_j}$.
  Men spreekt ook weer van bloktridiagonaal, etc.

- Een ijle matrix: Men noemt een matrix gewoonlijk ijl als elke rij slechts weinig elementen ongelijk aan nul bevat (bijvoorbeeld 5 procent of minder).

En dan nu een aantal van de belangrijkste eigenschappen (voor $A \in R^{n \times n}$):

symmetrie	:	$A = A^T$
scheef symmetrie	:	$A = -A^T$
positief definiet	:	$x^T A x > 0$ voor $x \neq 0$
niet-negatief definiet	:	$x^T A x \geq 0$, voor $x \neq 0$
indefiniet	:	$(x^T A x)(y^T A y) < 0$ voor een paar $x, y \neq 0$
orthogonaliteit	:	$A^T A = I$
nilpotent	:	er is een k zodat $A^k = 0$
idempotent	:	$A^2 = A$
positief	:	$A > 0$ als alle $a_{i,j} > 0$
diagonaal dominant	:	$\lvert a_{i,i}\rvert > \sum_{j \neq i} \lvert a_{i,j}\rvert$
kolom permutatie	:	willekeurige omwisseling van de kolommen van A
En voor $A \in C^{m \times n}$:		
geconjugeerde van A	:	$A^H = (\bar{a}_{j,i})$
unitaire A	:	$A^H A = I$
Hermietse A	:	$A^H = A$
positief definiet	:	$x^H A x > 0$ voor $x \neq 0$

Tenslotte nog een definitie: Zij $A, X \in R^{n \times n}$, en X nietsingulier, dan heten A en $X^T A X$ *congruent*.
Eigenschappen als symmetrie, scheefsymmetrie en positief definietheid blijven behouden onder congruentie.

Hoofdstuk 5

Vector en Matrix normen

Voor het bestuderen van de effecten van rekenfouten en meetfouten op de opgeleverde numerieke oplossingen hebben we een maat nodig. Pas als we een goede maat hebben, kunnen we een antwoord geven op de volgende vragen, die bijvoorbeeld ontstaan bij het oplossen van een stelsel $Ax = b$:

- Als A en b "niet exact" gegeven zijn, hoe nauwkeurig is dan x bepaald?
- Wat betekent het als A "bijna" singulier is?
- Als b niet opgespannen wordt door de kolommen van A, hoe kan je dan een x vinden zo dat Ax "dicht bij" b ligt?

Zo'n maat wordt gegeven door een norm.

5.1 Vectornormen

Een vectornorm op C^n is een afbeelding $\| \ \|$ van $C^n \to R$ met als eigenschappen

1. $\|x\| \geq 0$ voor alle $x \in R^n$, $\|x\| = 0 \Leftrightarrow x = 0$
2. $\|x + y\| \leq \|x\| + \|y\|$ voor alle $x, y \in R^n$
3. $\|\alpha x\| = |\alpha| \|x\|$, $\alpha \in R$, $x \in R^n$

Belangrijke normen zijn de zogenaamde *Höldernormen*:

$$\|x\|_p \equiv (|x_1|^p + \ldots + |x_n|^p)^{1/p}, \quad \text{voor } p \geq 1.$$

In het bijzonder:

$$\begin{aligned}
\|x\|_1 &= \sum |x_i| \\
\|x\|_2 &= \sqrt{\sum x_i^2} = \sqrt{x^T x} \\
\|x\|_\infty &= \max_i |x_i|
\end{aligned}$$

Uit de zogenaamde *Hölder ongelijkheid*:

$$|x^T y| \leq \|x\|_p \|y\|_q \quad \text{met } \frac{1}{p} + \frac{1}{q} = 1$$

volgt de ongelijkheid van *Cauchy-Schwarz*: $|x^T y| \leq \|x\|_2 \|y\|_2$.

Een belangrijke eigenschap is dat de 2-norm van x invariant is onder orthogonale transformaties Q:

$$\|Qx\|_2 = \sqrt{x^T Q^T Q x} = \sqrt{x^T x} = \|x\|_2.$$

Normen zijn equivalent: voor elk tweetal normen bestaan er constanten c_1 en c_2 zodat:

$$c_1 \|x\|_\alpha \leq \|x\|_\beta \leq c_2 \|x\|_\alpha.$$

Voor in dit dictaat belangrijke normen kunnen de volgende equivalenties bewezen worden:

$$\begin{aligned} \|x\|_2 &\leq \|x\|_1 \leq \sqrt{n}\|x\|_2 \\ \|x\|_\infty &\leq \|x\|_2 \leq \sqrt{n}\|x\|_\infty \\ \|x\|_\infty &\leq \|x\|_1 \leq n\|x\|_\infty \end{aligned}$$

Normen stellen ons in staat om over de afstand tussen x en y te spreken: $\|x - y\|$. Begrippen als convergentie, continuïteit en omgeving krijgen hierdoor zin en uitspraken hierover zijn norm onafhankelijk.
We definiëren nog de absolute fout van een benadering $\hat{x}$ ten opzichte van x als $\|\hat{x} - x\|$ en de relatieve fout als

$$\frac{\|\hat{x} - x\|}{\|x\|}.$$

Bij de $\| \ \|_\infty$-norm kan dat direct in een aantal decimalen vertaald worden:

$$\frac{\|\hat{x} - x\|_\infty}{\|x\|_\infty} \approx 10^{-p}$$

betekent dat de grootste component van $\hat{x}$ in ongeveer p decimalen correct is.

5.2 Matrixnormen

We zullen ook frequent een maat voor matrices nodig hebben: $\|A\|$ voor $A \in R^{m \times n}$. Een bekende norm is de *Frobenius-norm* (F-norm):

$$\|A\|_F \equiv \sqrt{\sum_i \sum_j a_{i,j}^2}.$$

Belangrijk zijn voorts de met de vectornormen geassocieerde matrixnormen:

$$\|A\|_p \equiv \sup_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p}.$$

Voor deze matrixnormen geldt:

$$\|AB\|_p \leq \|A\|_p \|B\|_p, \quad A \in R^{m\times n}, \quad B \in R^{n\times q}.$$

$$\|Ax\|_p \leq \|A\|_p \|x\|_p.$$

Een aantal belangrijke relaties:

$$\begin{array}{rcll} \|A\|_2 & \leq & \|A\|_F \leq \sqrt{n}\|A\|_2 & \\ \max|a_{i,j}| & \leq & \|A\|_2 \leq \sqrt{mn}\max|a_{i,j}| & \\ \|A\|_2 & \leq & \sqrt{\|A\|_1\|A\|_\infty} & \\ \|A\|_1 & = & \max\limits_j \sum\limits_i |a_{i,j}| & \text{max. kolomsom} \\ \|A\|_\infty & = & \max\limits_i \sum\limits_j |a_{i,j}| & \text{max. rijsom} \end{array}$$

Voor de 2-norm en de F-norm geldt weer normbehoud onder orthogonale transformaties:

$$\|QAZ\|_2 = \|A\|_2 \quad \text{en} \quad \|QAZ\|_F = \|A\|_F,$$

waarbij Q en Z orthogonale matrices zijn (van de juiste dimensies).

5.3 Conditie van een matrix

We bekijken het ten opzichte van $Ax = b$ verstoorde lineaire stelsel

$$(A + \epsilon F)x(\epsilon) = b + \epsilon f.$$

Als A nietsingulier is, dan is $A + \epsilon F$, voor ϵ klein genoeg, het ook, zodat $x(\epsilon)$ differentieerbaar is naar ϵ in de buurt van 0.
Met $\dot{x}(\epsilon) = \frac{\partial x(\epsilon)}{\partial \epsilon}$ krijgen we dan

$$Fx(\epsilon) + (A + \epsilon F)\dot{x} = f.$$

Dus voor $\dot{x}(0)$:

$$A\dot{x}(0) = f - Fx \quad \text{ofwel } \dot{x}(0) = A^{-1}(f - Fx).$$

Via Taylorreeks ontwikkeling: $x(\epsilon) = x + \epsilon\dot{x}(0) + \mathcal{O}(\epsilon^2)$ leidt dat tot

$$\frac{\|x(\epsilon) - x\|}{\|x\|} \leq \epsilon\|A^{-1}\|\left(\frac{\|f\|}{\|x\|} + \|F\|\right) = \epsilon\|A^{-1}\|\|A\|\left(\frac{\|f\|}{\|A\|\|x\|} + \frac{\|F\|}{\|A\|}\right) + \mathcal{O}(\epsilon^2).$$

Omdat $\|b\| \leq \|A\|\|x\|$:

$$\frac{\|x(\epsilon) - x\|}{\|x\|} \leq \epsilon\kappa(A)\left(\frac{\|f\|}{\|b\|} + \frac{\|F\|}{\|A\|}\right) + \mathcal{O}(\epsilon^2), \quad \text{met} \quad \kappa(A) \equiv \|A\|\|A^{-1}\|.$$

Het getal $\kappa(A)$ heet het *conditiegetal van* A ten opzichte van de norm $\| \ \|$. Veel gebruikt wordt $\kappa_2(A)$, dat is het conditiegetal t.o.v. $\| \ \|_2$.
Voor alle normen geldt $\kappa(A) \geq 1$.

Als $\kappa(A)$ relatief klein is ten opzichte van de inverse van de rekenprecisie u (zie volgend hoofdstuk), dan heet A goed-geconditioneerd; Als $\kappa(A)$ relatief groot is dan heet A slecht geconditioneerd. In dat geval kan een kleine verstoring in b of A tot een grote relatieve fout in x leiden. Orthogonale matrices zijn zeer goed geconditioneerd: $\kappa_2(Q) = 1$.

Soms ziet men wel $det(A)$ gebruikt worden als maat voor de conditie van A (omdat bijna singuliere A leidt tot "kleine" determinant). Dat is sterk af te raden, omdat "klein" geen betekenis heeft in de meeste gevallen.

Voorbeeld 1: De matrix

$$A = \begin{pmatrix} 1 & -1 & -1 & -1 & \ldots \\ & 1 & -1 & -1 & \ldots \\ & & 1 & -1 & \ldots \\ & & & 1 & \ldots \\ & & & & \ddots \end{pmatrix},$$

heeft conditiegetal $\kappa_\infty(A) = n2^{n-1}$, echter $det(A) = 1$.

Voorbeeld 2: $A = diag(\frac{1}{10}, \ldots, \frac{1}{10})$ heeft conditiegetal $\kappa_2(A) = 1$, echter $det(A) = 10^{-n}$.
Het is vaak aan te raden om A zodanig te schalen: $D_1 A D_2$, met D_1 en D_2 diagonaal matrices, dat de foutmatrix F ongeveer kolommen van gelijke lengte heeft (dit moet van geval tot geval apart bekeken worden, we zullen hier later nog op terug komen).

Voorbeeld 3: $A = diag(\frac{1}{\epsilon}, 1, 1, \ldots, 1, \epsilon)$ heeft als conditiegetal $\kappa_2(A) = \frac{1}{\epsilon^2}$, echter de oplossingen van $Ax = b$ zijn zeer goed bepaald, als we aannemen dat $\frac{1}{\epsilon}$ en ϵ zelf niet met extra fouten behept zijn.

Hoofdstuk 6

Afrondfouten en Basistransformaties

6.1 Floating point getallen

Vanwege eindige voorstelling van getallen zijn de meeste rekenkundige bewerkingen op een computer aan afrondfouten onderhevig.

Zij $\mathcal{F}$ de verzameling van floating point getallen f van de vorm:

$$f = \pm .d_1 d_2 \dots d_t \cdot \beta^e.$$

Hierin is β het *machinegrondtal* (of *machine base*), $d_1 \neq 0$, $0 \leq d_i < \beta$, $L \leq e \leq U$. Verder voegen we nog het getal 0 aan $\mathcal{F}$ toe. Bij de meeste huidige computers is $\beta = 2$, een uitzondering is bijvoorbeeld de IBM 370 waarvoor $\beta = 16$.

Er geldt, voor $f \neq 0$:

$$m \leq |f| \leq M,$$

met $m = \beta^{L-1}$ en $M = \beta^U(1 - \beta^{-t})$.

Om uitspraken over de effecten van afrondfouten te kunnen doen, introduceren we nu een model voor de computer arithmetiek op $\mathcal{F}$. Daarvoor definiëren we eerst de verzameling $\mathcal{G}$ als

$$\mathcal{G} = \{x \in R \mid m \leq |x| \leq M\} \cup \{0\}$$

en de afbeelding fl van $\mathcal{G}$ naar $\mathcal{F}$. De afbeelding fl voegt aan elk reëel getal uit $\mathcal{G}$ een floating point getal uit $\mathcal{F}$ toe. Bekende vormen voor fl zijn *afronden* en *afbreken*.
Er geldt:

$$fl(x) = x(1 + \xi),$$

voor $x \in \mathcal{G}$ en met

$$|\xi| \leq u \equiv \begin{cases} \frac{1}{2}\beta^{1-t} & \text{bij afronding} \\ \beta^{1-t} & \text{bij afbreken} \end{cases}$$

Zij $\diamond$ een der operaties $\times, /, +, -$. Als voor getallen a en b uit $\mathcal{F}$ het resultaat $a \diamond b$ geen element van $\mathcal{G}$ is, dan hebben we te maken met een *overflow* situatie (als $|a \diamond b| > M$) of een

underflow situatie (als $|a\diamond b| < m$). Overflow situaties resulteren doorgaans in een foutmelding bij computerverwerking, bij underflow wordt het resultaat meestal door 0 vervangen.
Als $a\diamond b \in \mathcal{G}$, dan hoeft het nog niet noodzakelijk element van $\mathcal{F}$ te zijn. Voor een gezonde arithmetiek moet gelden

$$fl(a\diamond b) = (a\diamond b)(1+\xi), \quad \text{met} \quad |\xi| \leq u,$$

ofwel

$$\frac{|fl(a\diamond b) - (a\diamond b)|}{|a\diamond b|} \leq u.$$

Zoals we al gezien hebben in het college Numerieke Wiskunde A, kunnen samengestelde operaties, of operaties op getallen die niet in $\mathcal{F}$ zitten, tot grote relatieve fouten leiden.

Voorbeelden:

- Stel dat $\beta = 10$ en $t = 4$, dat wil zeggen we rekenen in vier decimalen in het tientallig stelsel, en dat fl door afronding tot stand komt. Dan levert
$fl(fl(0.4 \cdot 10^{-4} + 1) - 1) = 0$ een relatieve fout 1 op en
$fl(0.4 \cdot 10^{-4} + fl(1-1)) = 0.4 \cdot 10^{-4}$ een relatieve fout 0.

- Als we voor grote positieve x de waarde van e^{-x} benaderen via de reeks $1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \ldots$, dan wordt het resultaat overspoeld met afrondfouten.

Voor de doorwerking van afrondfouten zal de volgende stelling van pas komen.

Stelling 1 *Als* $1+\alpha \equiv \prod_1^n (1+\alpha_i)$ *met* $|\alpha_i| \leq u$ *en* $n \times u \leq 0.01$, *dan geldt* $|\alpha| \leq 1.01 n \times u$.

Voor het bewijs zie [8].

Een floating point foutanalyse van het inprodukt
Omdat het inprodukt in de lineaire algebra een sleutelpositie inneemt, zullen we hiervoor apart de doorwerking van fouten in floating point arithmetiek bekijken.
Voor $x, y \in R^n$, met $x_i, y_i \in \mathcal{F}$, wordt in exacte aritmetiek het inprodukt gegeven door

$$x^T y = \sum_{i=1}^{n} x_i y_i.$$

Het resultaat dat ontstaat in floating point arithmetiek zullen we noteren als $fl(x^T y)$. Let op: met deze notatie zullen we steeds bedoelen het eindresultaat van een berekening waarvan de afzonderlijke operaties in floating point arithmetiek zijn uitgevoerd en **niet** het door fl afgebeelde exacte resultaat van $x^T y$ op $\mathcal{F}$.
Zij $S_p \equiv fl(\sum_1^p x_i y_i)$, dan volgt met de regels voor floating point arithmetiek

$$\begin{aligned} S_1 &= fl(x_1 y_1) = x_1 y_1 (1 + \xi_{1\times}) \\ S_k &= fl(S_{k-1} + fl(x_k y_k)) \\ &= (S_{k-1} + x_k y_k (1 + \xi_{k\times}))(1 + \xi_{1+}), \end{aligned} \tag{6.1}$$

waarbij $|\xi_{i\times}|, |\xi_{i+}| \le u$.
Hieruit volgt, met volledige inductie, dat $fl(S_n) = \sum_i^n x_i y_i (1+\gamma_i)$, met

$$(1+\gamma_i) = (1+\xi_{i\times}) \prod_{j=i}^{n} (1+\xi_{i+}).$$

Hierin is $\xi_{1+} = 0$.
Dus $|fl(x^T y) - x^T y| \le \sum_{i=1}^n |x_i y_i|\ |\gamma_i|$. Met gebruikmaking van stelling 1 krijgen we het resultaat

$$|fl(x^T y) - x^T y| \le 1.01 nu |x|^T |y|.$$

Dit resultaat betekent dat we dus een grote relatieve fout in het inprodukt mogen verwachten als $x \perp y$ of als x nagenoeg loodrecht staat op y.

Een grover resultaat wordt iets eenvoudiger bereikt via $|1+\alpha| = 1 + nu + \mathcal{O}(u^2)$, namelijk $|fl(x^T y) - x^T y| \le nu|x|^T|y| + \mathcal{O}(u^2)$.

Floating point matrix operaties
We gebruiken de volgende notaties: $|A| = (|a_{i,j}|)$, en $|A| \le |B|$ betekent $|a_{i,j}| \le |b_{i,j}|$. Men kan nu voor de situatie dat $a_{i,j}, b_{i,j} \in \mathcal{F}$ eenvoudig de volgende relaties verifiëren

$$fl(\alpha A) = \alpha A + E, \quad \text{met} \quad |E| \le u|\alpha||A| \tag{6.2}$$

$$fl(A+B) = (A+B) + E, \quad \text{met} \quad |E| \le u|A+B| \tag{6.3}$$

$$fl(AB) = AB + E, \quad \text{met} \quad |E| \le 1.01nu|A||B| \quad \text{voor} \quad A \in I\!R^{m\times n},\ B \in I\!R^{n\times q}. \quad (\text{mits} \quad nu \le .01) \tag{6.4}$$

Merk op dat $|AB|$ veel kleiner kan zijn dan $|A||B|$, zodat de relatieve fout in het matrix product aanzienlijk kan zijn.
We hebben hier in feite voorbeelden gezien van hoe rekenfouten kunnen doorwerken in het resultaat. Men noemt een dergelijke analyse ook wel *voorwaartse fouten analyse*, in tegenstelling tot de zogenaamde *achterwaartse fouten analyse* waarbij men rekenfouten terugvoert op de gegeven begindata. We laten deze laatste techniek zien aan de hand van een eenvoudig voorbeeld.
Zij

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} \\ 0 & a_{2,2} \end{pmatrix} \qquad B = \begin{pmatrix} b_{1,1} & b_{1,2} \\ 0 & b_{2,2} \end{pmatrix}$$

Dan geldt

$$fl(AB) = \begin{pmatrix} a_{1,1}b_{1,1}(1+\xi_1) & \{a_{1,1}b_{1,2}(1+\xi_2) + a_{1,2}b_{2,2}(1+\xi_3)\}(1+\xi_4) \\ 0 & a_{2,2}b_{2,2}(1+\xi_5) \end{pmatrix} \quad \text{met} \quad |\xi_i| \le u.$$

Men rekent dan simpelweg na dat $fl(AB) = \hat{A}\hat{B}$ met

$$\hat{A} = \begin{pmatrix} a_{1,1} & a_{1,2}(1+\xi_3)(1+\xi_4) \\ 0 & a_{2,2}(1+\xi_5) \end{pmatrix} \qquad \hat{B} = \begin{pmatrix} b_{1,1}(1+\xi_1) & b_{1,2}(1+\xi_2)(1+\xi_4) \\ 0 & b_{2,2} \end{pmatrix}$$

Hieruit volgt dat $\hat{A} = A + E$ en $\hat{B} = B + F$ met $|E| \le 2u|A| + \mathcal{O}(u^2)$ en $|F| \le 2u|B| + \mathcal{O}(u^2)$. Men kan het resultaat van dit matrix product dus ook interpreteren als het exacte produkt van twee licht verstoorde matrices. We zullen nog vele voorbeelden van deze interpretatie tegenkomen.

Opgave 1 *Geef een 2 bij 2 voorbeeld van een matrix product in floating point arithmetiek waarbij een grote relatieve fout optreedt.*

Opgave 2 *Neem aan dat de $\sqrt{}$-berekening voldoet aan $fl(\sqrt{\alpha}) = \sqrt{\alpha}(1+\xi)$, met $|\xi| \leq u$. Geef aan hoe $\|x\|_2$ berekend kan worden en geef een bovengrens voor de afrondfouten in het resultaat ($x \in R^n$).*

6.2 Householder transformaties

Definitie 1 *Voor $v \in R^n$ en $v \neq 0$ heet*

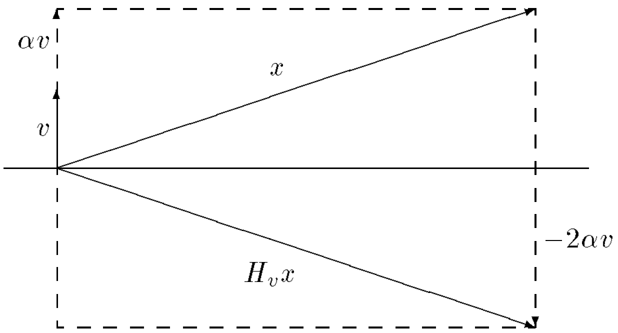
$$H_v = I - \frac{2}{v^T v} v v^T$$

een Householder transformatie.

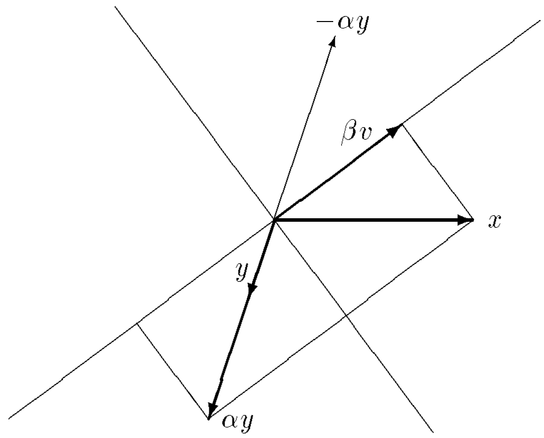
Voor $x \neq 0$ geldt

$$H_v x = (I - \frac{2}{v^T v} v v^T)x = x - \frac{2v^T x}{v^T v} v = x - \alpha v$$

Bedenk hierbij dat $\frac{1}{\sqrt{v^T v}} v$ een genormeerde vector voorstelt.



We zien dat $H_v x$ de gespiegelde van x oplevert ten opzichte van $v^\perp$. De operator H_v heet daarom ook wel een *Householder reflectie*. Door v geschikt te kiezen, kunnen we een gegeven vector x overvoeren in een gegeven andere vector μy, met μ zodanig dat $\|x\|_2 = \|\mu y\|_2$ (normbehoud). Merk nog op dat H_v een orthogonale matrix is: $H_v^T H_v = I$.
De constructie van een vector v die de gewenste transformatie tot stand brengt laat zich met een figuur eenvoudig toelichten.



De vector v is kennelijk een vector in de richting van de bissectrice van x en $\pm y$. Er moet dus gelden $H_v x = \mu y$ met $\mu = \pm\frac{\|x\|_2}{\|y\|_2}$. Vanwege het feit dat v in de richting van de bissectrice moet liggen, geldt dus: $v = x \pm \frac{\|x\|_2}{\|y\|_2} y$ (of een veelvoud daarvan).

Een wat formelere constructie van v verloopt als volgt. We zullen aannemen dat we x willen overvoeren naar een vector in de richting van de eerste eenheidsvector e_1. Er geldt dan dus

$$\mu e_1 = H_v x = x - \frac{2v^T x}{v^T v} v \Longrightarrow v = x + \beta e_1,$$

en dus $v^T x = x^T x + \beta x_1$ (met x_1 de eerste coördinaat van x), en $v^T v = x^T x + \beta^2 + 2\beta x_1$. We vullen de verkregen uitdrukking voor v in in de expressie voor $H_v x$:

$$H_v x = x - \frac{2v^T x}{v^T v}(x + \beta e_1) = (1 - \frac{2v^T x}{v^T v})x - \frac{2v^T x}{v^T v}\beta e_1.$$

Omdat $H_v x$ in de richting van e_1 moet liggen, moet de coëfficient van x in de laatste expressie kennelijk gelijk aan 0 worden. Daaruit volgt dat $v^T v = 2v^T x \Longrightarrow x^T x + \beta^2 + 2\beta x_1 = 2x^T x + 2\beta x_1$. Dus

$$\beta = \pm\|x\|_2 \quad \text{en} \quad v = x \pm \|x\|_2 e_1.$$

We hebben nog de vrijheid om het teken in de expressie voor v te kiezen. Als x ongeveer al in de richting van e_1 ligt, dan zijn de eerste coördinaten van x en $\|x\|_2 e_1$ nagenoeg gelijk en kan er door de aftrekking een grote relatieve fout in v ontstaan. We kiezen daarom het teken zodanig dat dit zoveel mogelijk voorkomen wordt: $v = x + sign(x_1)\|x\|_2 e_1$. Bij overvoering naar een vector y (in plaats van e_1) kiezen we het teken zo dat $\|v\|_2 \geq \|x\|_2$.

Householder transformaties kunnen gebruikt worden om bijvoorbeeld een opeenvolgend stel vector componenten op nul te zetten. We lichten dit weer toe met een voorbeeld.

Zij $x = (x_1, \ldots, x_{k-1}, x_k, x_{k+1}, \ldots, x_m, x_{m+1}, \ldots, x_n)^T$. Dan met

$$v = (0, 0, 0, \ldots, x_k + sign(x_k)\alpha, x_{k+1}, \ldots, x_m, 0, 0, \ldots)^T,$$

en

$$\alpha^2 = x_k^2 + x_{k+1}^2 + \ldots + x_m^2$$

geldt $H_v x = (x_1, \ldots, x_{k-1}, -sign(x_k)\alpha, 0, 0, \ldots, 0, x_{m+1}, \ldots, x_n)^T$.
Merk op dat H_v nu de gedaante

$$H_v = \begin{pmatrix} I_{k-1} & \emptyset & \emptyset \\ \emptyset & \bar{H}_v & \emptyset \\ \emptyset & \emptyset & I_{n-m} \end{pmatrix}$$

heeft, waarbij $\bar{H}|v$ een householder transformatie is beperkt tot de ruimte opgespannen door de k-de tot en met m-de basisvector.
Om problemen met under- of overflow te voorkomen, wordt vaak gerekend met de vector $\frac{x}{\|x\|_\infty}$ in plaats van x om v te bepalen (ga na dat de berekening van $H_v x$, afgezien van de under- en overflow problematiek, niet verandert als v berekend wordt uit αx in plaats van x). Men rekent in dit geval liever met de $\| \; \|_\infty$-norm in plaats van de 2-norm omdat de eerste zonder afrondfouten te berekenen is.

Opgave 3 *Kan men ook willekeurige, dat wil zeggen niet aaneengesloten, elementen van een vector via een geschikt gekozen Householder transformatie naar nul transformeren?*

Algoritme 1 $x \in R^n$, $1 \leq k < j \leq n$. *Het algoritme berekent een stabiele v zo dat de componenten $k+1$ tot en met j van de vector $H_v x$ gelijk zijn aan 0 (we nemen aan dat niet al deze componenten van x gelijk aan nul zijn).*

$\|x\|_\infty := \max\{|x|_k, \ldots, |x|_j\}$
$\alpha := 0$
voor $i = k, \ldots, j$
  $v_i := x_i / \|x\|_\infty$
  $\alpha := \alpha + v_i^2$
end i
$\alpha := \sqrt{\alpha}$
$\beta := 1/(\alpha(\alpha + |v_k|))$
$v_k := v_k + sign(v_k)\alpha.$

Ga na dat $\beta = \frac{2}{v^T v}$.

Algoritme 1 vereist ongeveer $2(j-k)$ vermenigvuldigingen (delingen) en $(j-k)$ optellingen (aftrekkingen). Vaak worden deze basis rekenoperaties als flops aangeduid (**flo**ating **p**oint operation**s**).
We kunnen H_v ook "loslaten" op een matrix, om bijvoorbeeld een gedeelte van een gegeven kolom "op nul te zetten":

$$H_v A = (I - \frac{2vv^T}{v^T v})A = A - \beta v(A^T v)^T, \quad \beta = \frac{2}{v^T v}.$$

We constateren dat H_v losgelaten op A aanleiding geeft tot een rang 1 correctie (rank 1 update). De inverse van de matrix H_vA zouden we dus via de formule van Sherman-Morrison uit die van A kunnen berekenen.

Het volgende algoritme berekent H_vA, zonder eerst H_v expliciet te vormen.

Algoritme 2 *Met* $A \in R^{n\times q}$, $v^T = (0, \ldots, v_k, \ldots, v_j, 0, \ldots, 0)$ *en* $\beta = 2/(v^Tv)$ *wordt* A *als volgt door* H_vA *overschreven:*

$$
\begin{array}{l}
\textit{voor } p = 1, \ldots, q \\
\quad s := v_k a_{k,p} + \ldots + v_j a_{j,p} \\
\quad s := \beta s \\
\quad \textit{voor } i = k, \ldots, j \\
\qquad a_{i,p} := a_{i,p} - s \cdot v_i
\end{array}
$$

Ga na dat het algoritme ongeveer $4q(j-k+1)$ flops vergt.

Opgave 4 *Maak zelf een algoritme dat* A *overschrijft met* AH_v*.*

Eigenschappen t.a.v. afrondfouten.
a. Algoritme 1 levert een $\hat{\beta} = fl(\beta)$ en $\hat{v} = fl(v)$ op waarbij voor $\hat{H}_v$ geldt

$$\|H_v - \hat{H}_v\|_2 \leq 10u$$

(voor het bewijs zie [26])
b. Als $\hat{H}_vA$ berekend wordt volgens algoritme 2, dan geldt:

$$fl(\hat{H}_vA) = H_v(A+E), \quad \|E\|_2 \leq c(j-k+1)^2 u \|A\|_2, \text{ met } c = \mathcal{O}(1).$$

Produkten van Householder transformaties.
Bij veel algoritmen wordt er een aantal verschillende Householder transformaties op A losgelaten, om er verschillende kolommen mee te bewerken. De resulterende orthogonale matrix waarmee A dan uiteindelijk bewerkt is, schrijven we als

$$Q = H_{v^{(r)}} H_{v^{(r-1)}} \cdots H_{v^{(1)}}, \quad \text{met} \quad H_{v^{(i)}} = I - \beta_i v^{(i)} v^{(i)^T}.$$

Het is vaak aan te raden om voor elke $H_{v^{(i)}}$ uitsluitend de β_i en $v^{(i)}$ op te slaan en niet de produktmatrix Q te vormen. Voor de berekening van QA gebruiken we dan het volgende algoritme, waarbij A door QA wordt overschreven.

Algoritme 3

$$
\begin{array}{l}
\textit{voor } i = 1, 2, \ldots, r \\
\quad A := H_{v^{(i)}} A,
\end{array}
$$

(via algoritme 2).

Ga na dat dit algoritme ongeveer $4nrq$ flops vergt voor $A \in R^{n\times q}$, $H_{v^{(i)}} \in R^{n\times m}$.
Zouden we eerst Q expliciet berekenen, dan krijgen we in feite algoritme 4:

Algoritme 4

$$
\begin{array}{l}
Q := I \\
\textit{voor } i = r, r-1, \ldots, 1 \\
\quad Q := QH_{v^{(i)}} \\
\quad A := QA
\end{array}
$$

Ga na dat dit algoritme ongeveer $2n^2r+n^2q$ flops kost. Wanneer is dat voordeliger dan algoritme 3?
Bij sommige toepassingen is het nodig de matrix Q expliciet in handen te hebben. We kunnen daarvoor algoritme 4 gebruiken (zonder de laatste opdracht $A := QA$). In deze gevallen hebben de vectoren $v^{(i)}$ doorgaans de eigenschap dat de eerste $i-1$ coördinaten gelijk aan nul zijn. De verwerkings volgorde in algoritme 4 is dan veel economischer dan de volgorde

$$\begin{array}{l} Q := I \\ \text{voor } i = 1, 2, \ldots, r \\ \quad Q := H_{v^{(i)}} Q, \end{array}$$

dankzij de eigenschap dat $H_{v^{(i+1)}} \cdots H_{v^{(r)}} = diag(I_i, Q_i)$, met $Q_i \in R^{(n-i)\times(n-i)}$. De laatst genoemde verwerkings volgorde rekent al vanaf de eerste stap ($i = 1$) met een volle matrix.

Opgave 5 *Als we aanemen dat de r vectoren $v^{(i)}$ opgeslagen staan in een $n \times n$ matrix V en als de eerste $(i-1)$ coördinaten van $v^{(i)}$ nul zijn, laat dan zien dat Q zonder gebruik te maken van extra werkruimte opgebouwd kan worden op de plaats van V.*

Householder transformaties worden ook vaak geschreven in de vorm $H_v = I - wy^T$, waarbij er voor gezorgd is dat de eerste niet-nul component van y gelijk is aan 1:

$$y^T = (0, \ldots, 0, 1, y_{k+1}, \ldots, y_j, 0, \ldots, 0), \quad y_i = v_i/v_k,$$

$$w^T = (0, \ldots, 0, w_k, w_{k+1}, \ldots, w_j, 0, \ldots, 0), \quad w_i = \beta v_k \cdot v_i,$$

Men doet dit om overflow en underflow problemen bij de transformatie van A te voorkomen. In de bovenstaande vorm spreekt men wel van *modified Householder matrices*.

Opgave 6 *Zij $x = (3, 1, 5, 1)^T$. Construeer H_v zodanig dat $H_v x = (\alpha, 0, 0, 0)^T$. Wat is de waarde van α?*

Opgave 7 *Zij $A \in R^{n\times n}$ symmetrisch. Zij H_{v_1} de Householder transformatie die $a_{3,1}, a_{4,1}, \ldots, a_{n,1}$ overvoert in nul en $a_{1,1}$ ongewijzigd laat. Laat zien dat in $A_1 \equiv H_{v_1} A H_{v_1}^T$ de elementen $a_{i,1}$ en $a_{1,i}$ gelijk aan nul zijn voor $i \geq 3$. Zij verder H_{v_2} de Householder transformatie die $a_{4,2}, a_{5,2}, \ldots, a_{n,2}$ overvoert in nul. Laat zien dat $H_{v_2} A_1 H_{v_2}^T$ in de eerste kolom en rij niet van A_1 verschilt en dat bovendien $a_{i,2}$ en $a_{2,i}$ nul zijn voor $i \geq 4$. Ga na dat men op deze wijze voortgaande de matrix A via een orthogonale Q kan overvoeren in een tridiagonale matrix A_{n-2} door $A_{n-2} = QAQ^T$.*

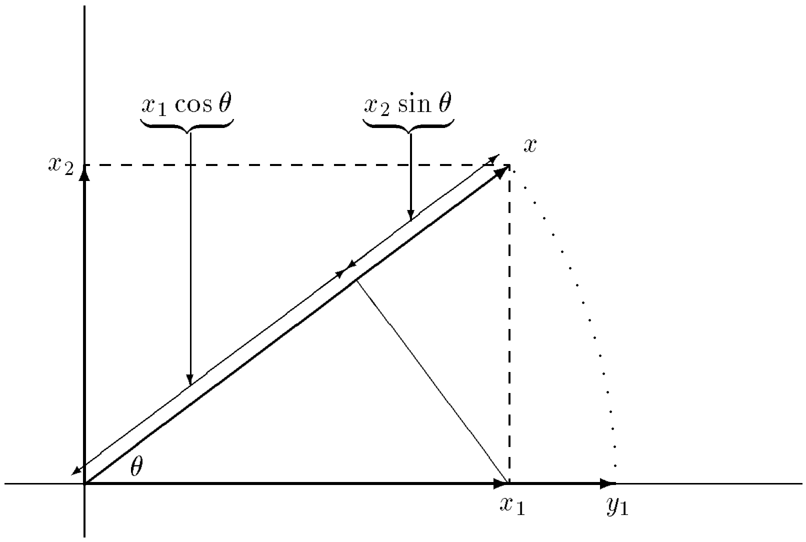
6.3 Givens transformaties

Met Givens transformaties (ook wel Givens rotaties) kan men in een vector selectief een element naar nul transformeren. We beginnen met een voorbeeld. Zij $x = (x_1, x_2)^T \in R^2$. Zij verder $y = (y_1, 0)^T$ met $y_1^2 = x_1^2 + x_2^2$. Dan geldt:

$$\begin{pmatrix} y_1 \\ 0 \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix},$$

met

$$\cos\theta = \frac{x_1}{\sqrt{x_1^2 + x_2^2}}.$$



Definitie 2 *Een Givens transformatie is een matrix van de vorm:*

$$G(i,k,\theta) = \begin{pmatrix} 1 & & & & & & & & & & \\ & \ddots & & & & & & & & & \\ & & 1 & & & & & & & & \\ & & & \cos\theta & & & & \sin\theta & & & \\ & & & & 1 & & & & & & \\ & & & & & \ddots & & & & & \\ & & & & & & 1 & & & & \\ & & & -\sin\theta & & & & \cos\theta & & & \\ & & & & & & & & 1 & & \\ & & & & & & & & & \ddots & \\ & & & & & & & & & & 1 \end{pmatrix}$$

voor zekere θ, waarbij de $\cos\theta$ de diagonaal posities (i,i) en (k,k) bezetten.

Transformatie van een vector x tot $G(i,k,\theta)x$ komt neer op een rotatie over een hoek θ in het vlak opgespannen door de i-de en de k-de eenheidsvector. Met $c = \cos\theta$ en $s = \sin\theta$ geldt voor $y = G(i,k,\theta)x$:

$$y_i = cx_i + sx_k$$

$$y_k = -sx_i + cx_k$$

$$y_j = x_j \quad \text{voor } j \neq i,k.$$

De coördinaat y_k wordt gelijk aan 0 door de keuze

$$c = \frac{x_i}{\sqrt{x_i^2 + x_k^2}}, \quad s = \frac{x_k}{\sqrt{x_i^2 + x_k^2}}.$$

Om problemen met overflow en underflow te voorkomen gaat men in de praktijk bij het berekenen van c en s als volgt te werk.
Als $x_i > x_k$ dan

$$c = \frac{1}{\sqrt{1 + (\frac{x_k}{x_i})^2}},$$

$$s = (\frac{x_k}{x_i}) \cdot c.$$

Voor $x_k > x_i$ analoog.
Dit leidt tot het volgende algoritme voor het berekenen van c en s.

Algoritme 5 *Zij $x \in R^n$, en $1 \leq i < k \leq n$. Dit algoritme berekent een c en een s zodanig dat de k-de component van de Givens getransformeerde $G(i,k,\theta)x$ gelijk is aan 0.*

if $x_k = 0$ *then* $(c := 1,\ s := 0)$
  else
    if $|x_k| \geq |x_i|$ *then*
    $(t := \frac{x_i}{x_k},\ s := \frac{1}{\sqrt{1+t^2}},\ c := s \cdot t)$
    else $(t := \frac{x_k}{x_i},\ c := \frac{1}{\sqrt{1+t^2}},\ s := c \cdot t)$

Door het volgende algoritme wordt de matrix A overschreven door $G(i,k,\theta)A$.

Algoritme 6 *Zij $A \in R^{n \times q}$ en $G(i,k,\theta)$ een Givens matrix. Dan wordt A op de volgende wijze overschreven door $G(i,k,\theta)A$:*

voor $j = 1, \ldots q$
  $v := a_{i,j},\ w := a_{k,j}$
  $a_{i,j} := c \cdot v + s \cdot w$
  $a_{k,j} := -s \cdot v + c \cdot w$

Het algoritme 6 vereist ongeveer $6q$ flops. Om een hele kolom van A, op het eerste element na, naar nul te transformeren zijn ruwweg $6nq$ flops nodig. Het algoritme is voor dat doel dus twee keer zo duur als de Householder transformatie.
Op soortgelijke wijze als in algoritme 6 kan men elementen in een rij van A naar nul transformeren via de berekening van $AG(i,k,\theta)$.

Eigenschappen t.a.v. afrondfouten
Men gaat gemakkelijk na dat de berekende c en s voldoen aan

$$\hat{c} = fl(c) = c(1 + \varepsilon_c), \quad \hat{s} = fl(s) = s(1 + \varepsilon_s), \quad \varepsilon_c, \varepsilon_s = \mathcal{O}(u).$$

Met behulp hiervan kan men weer aantonen dat

$$fl(\hat{G}(i,k,\theta)A) = G(i,k,\theta)(A + E), \quad \frac{\|E\|_2}{\|A\|_2} = \mathcal{O}(u).$$

Voor details zie [26].
Voor samengestelde Givens transformaties, werkend op $A \in R^{n \times q}$:

$A_0 = A$
voor $k = 1, \ldots, p$
$\quad A_k = fl(\hat{G}_\ell(i_k, j_k, \theta_k) A_{k-1} \hat{G}_r(i'_k, j'_k, \theta'_k))$

is aangetoond dat

$$A_p = G_\ell(i_p, j_p, \theta_p) \cdots G_\ell(i_1, j_1, \theta_1)(A + E) G_r(i'_1, j'_1, \theta'_1) \cdots G_r(i'_p, j'_p, \theta'_p)$$

met $\|E\|_2 \leq \gamma u \|A\|_2$, waarbij γ een niet al te wilde functie van n, q en p is, en G_ℓ en G_r de exacte orthogonale transformaties. Met andere woorden: A_p is te beschouwen als de exact getransformeerde van een matrix die maar weinig van A afwijkt.
Opmerking: De Jacobi methode voor het bepalen van de eigenwaarden van een symmetrische matrix berust ook op Givens transformaties.

6.4 Gauss transformaties

Men kan ook via niet-orthogonale transformaties de elementen van een vector naar 0 transformeren, bijvoorbeeld via:

$$\begin{pmatrix} 1 & & & & \\ \alpha_2 & 1 & & & \\ \alpha_3 & & 1 & & \\ \vdots & & & \ddots & \\ \alpha_n & & & & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 + \alpha_2 x_1 \\ x_3 + \alpha_3 x_1 \\ \vdots \\ x_n + \alpha_n x_1 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{pmatrix}.$$

Als $x_1 \neq 0$ dan is, met de keuze $\alpha_j = -x_j / x_1$, de vector y een vector met $y_j = 0$ voor $j = 2, \ldots, n$.

Definitie 3 *De matrix M_k, waarvoor*

$$M_k(x_1, \ldots, x_k, x_{k+1}, \ldots, x_n)^T = (x_1, \ldots, x_k, 0, \ldots, 0)^T,$$

is een Gauss transformatie:

$$M_k = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & 1 & & & \\ & & & \alpha_{k+1} & 1 & & \\ & & & \vdots & & \ddots & \\ & & & \alpha_n & & & 1 \end{pmatrix}.$$

De k-de kolom van deze matrix heet ook wel een Gauss vector.

Het volgende algoritme berekent de Gauss vector voor M_k, als $x_k \neq 0$, zo dat van $M_k x = (I - \alpha e_k^T)x$ de coördinaten $k+1$ tot en met n gelijk zijn aan 0 ($\alpha = (0, \ldots, 0, \alpha_{k+1}, \ldots, \alpha_n)^T$).

Algoritme 7 *Het algoritme berekent een Gauss vector:*

voor $i = k+1, \ldots, n$
$\quad \alpha_i := x_i / x_k$
end i

Het algoritme vergt $n-k$ flops. Het uitvoeren van een Gauss transformatie op een vector x is wel heel eenvoudig:

$$M_k x = (I - \alpha e_k^T) x = x - \alpha e_k^T x = x - x_k \alpha.$$

Het nu volgende algoritme overschrijft de matrix A door $(I - \alpha e_k^T)A$.

Algoritme 8 *Met* $A \in R^{n \times q}$ *en* α *als in algoritme 7, wordt* A *als volgt door* $(I - \alpha e_k^T)A$ *overschreven:*

voor $j = 1, \ldots, q$
$\quad$ *voor* $i = k+1, \ldots, n$
$\quad a_{i,j} := a_{i,j} - a_{k,j}\alpha_i$
end i

Dit algoritme kost ongeveer $2(n-k)q$ flops.
We zien hieruit dat een Gauss transformatie ongeveer 2/3 keer zo duur is als een Householder transformatie. Dat is zeker aantrekkelijk, echter de vraag rijst nu hoe het met het gedrag ten aanzien van afrondfouten staat (de numerieke stabiliteit). Uit algoritme 7 volgt meteen dat

$$\hat{\alpha} \equiv fl(\alpha) = \alpha + e_\alpha, \quad |e_\alpha| \leq |\alpha|.$$

De volgende stelling doet een uitspraak over de foutvoortplanting ten gevolge van algoritme 8.

Stelling 2 *Voor de door algoritme 8 opgeleverde matrix geldt*

$$fl((I - \hat{\alpha} e_k^T)A) = (I - \alpha e_k^T)A + E,$$

met

$$|E| \leq u\{|A| + 3|\alpha||a_k^T|\} + \mathcal{O}(u^2), \quad a_k = (a_{k,1}, \ldots, a_{k,q})^T.$$

Bewijs: Er geldt $fl((I - \hat{\alpha} e_k^T)A) = fl(A - fl(\hat{\alpha} a_k^T))$, met

$$\alpha = (0, \ldots, 0, \frac{a_{k+1,k}}{a_{k,k}}, \ldots, \frac{a_{n,k}}{a_{k,k}})^T, \quad , \hat{\alpha} = fl(\alpha).$$

Dus $\hat{\alpha}_j = fl(\alpha_j) = \alpha_j(1 + e_j^{(1)})$ met $|e_j^{(1)}| \leq u$ en $(fl(\hat{\alpha} a_k^T))_{i,j} = \hat{\alpha}_i a_{k,j}(1 + e_{i,j}^{(2)})$ met $|e_{i,j}^{(2)}| \leq u$.
Nu moeten we nog de fout tengevolge van de aftrek operatie bekijken:

$$\begin{aligned}
(fl(A - fl(\hat{\alpha} a_k^T)))_{i,j} &= (a_{i,j} - (fl(\hat{\alpha} a_k^T))_{i,j})(1 + e_{i,j}^{(3)}), \text{ met } |e_{i,j}^{(3)}| \leq u, \\
&= (a_{i,j} - \alpha_i a_{k,j}(1 + e_i^{(1)})(1 + e_{i,j}^{(2)}))(1 + e_{i,j}^{(3)}) \\
&= (a_{i,j} - \alpha_i a_{k,j}(1 + e_i^{(1)} + e_{i,j}^{(2)} + \mathcal{O}(u^2)))(1 + e_{i,j}^{(3)}) \\
&= a_{i,j} - \alpha_i a_{k,j} + a_{i,j} e_{i,j}^{(3)} - \alpha_i a_{k,j}(e_i^{(1)} + e_{i,j}^{(2)} + e_{i,j}^{(3)}) + \mathcal{O}(u^2) \\
&= a_{i,j} - \alpha_i a_{k,j} + e_{i,j} + \mathcal{O}(u^2),
\end{aligned}$$

waarbij $|e_{i,j}| \leq |a_{i,j}|u + 3|\alpha_i a_{k,j}|u$. Hieruit volgt het te bewijzene direct. $\square$

Merk op dat de fouten in $M_k A$ groot kunnen zijn in vergelijking met $|A|$ als α grote elementen heeft, dat wil zeggen als een der $a_{i,k}/a_{k,k}$ groot is.

Samengestelde Gauss transformaties

Een samengestelde Gauss transformatie M ontstaat door een aantal enkelvoudige Gauss transformaties achtereen toe te passen:

$$M = M_k \cdots M_1, \quad \text{met} \quad M_j = (I - \alpha^{(j)} e_j^T),$$

$$\alpha^{(j)} = (0, \ldots, 0, \ell_{j+1,j}, \ldots, \ell_{n,j})^T.$$

Men slaat vaak de Gauss vectoren van de M_j's op in plaats van M expliciet te berekenen. Deze wijze van opslag laat een handige berekening van M^{-1} toe, zoals uit de volgende opgave blijkt.

Opgave 8 *Zij $M_k = I - \alpha^{(k)} e_k^T$ een Gauss transformatie. Laat zien dat*

$$M_k^{-1} = I + \alpha^{(k)} e_k^T.$$

Vervolgens zien we dat voor M^{-1} geldt

$$\begin{aligned} M^{-1} &= M_1^{-1} \cdots M_k^{-1} \\ &= (I + \alpha^{(1)} e_1^T) \cdots (I + \alpha^{(k)} e_k^T) \\ &= I + \sum_{i=1}^{k} \alpha^{(i)} e_i^T \end{aligned}$$

De laatste uitdrukking ontstaat door op te merken dat $e_j^T \alpha^{(i)} = 0$ voor $j < i$.
Hieruit volgt dat M^{-1}, zonder enige verdere bewerking, uit de Gauss vectoren ontstaat:

$$M^{-1} = \begin{pmatrix} 1 & & & & & & \\ \ell_{2,1} & 1 & & & & & \\ \ell_{3,1} & \ell_{3,2} & \ddots & & & & \\ \vdots & \vdots & \vdots & 1 & & & \\ \vdots & \vdots & \vdots & \ell_{k+1,k} & 1 & & \\ \vdots & \vdots & \vdots & \vdots & & \ddots & \\ \ell_{n,1} & \ell_{n,2} & \cdots & \ell_{n,k} & & & 1 \end{pmatrix}.$$

Subroutines uit LINPACK (en andere libraries/pakketten) maken gebruik van deze eigenschap.

Hoofdstuk 7

Gauss eliminatie en LU decompositie

De hier behandelde theorie kan men uitgebreider, en voorzien van vele literatuur verwijzingen, terugvinden in ([10]:Chapter 4).

7.1 Driehoeksstelsels

Het uitvoeren van achtereenvolgende Gauss transformaties op een stelsel $Ax = b$ staat bekend onder de naam Gauss eliminatie. Zowel bij Gauss eliminatie als bij QR decompositie wordt een gegeven stelsel overgevoerd in een stelsel waarvan de matrix een bovendriehoeksmatrix is. We kijken daarom eerst in meer detail naar het oplossen van driehoeksstelsels. In het nu volgende algoritme wordt een benedendriehoeksstelsel opgelost (voorwaartse eliminatie).

Algoritme 9 *Zij $L \in R^{n \times n}$ een niet-singuliere benedendriehoeksmatrix en zij $b \in R^n$. Dan wordt de oplossing y van $Ly = b$ berekend via:*

voor $i = 1, \ldots, n$
 $y_i = (b_i - \sum_{j=1}^{i-1} y_j \ell_{i,j}) / \ell_{i,i}$
end i

Het algoritme vereist n^2 flops. Het leidt tot een oplossing $\hat{y}$, waarvoor volgens [8] geldt

$$(L + F)\hat{y} = b, \quad \text{met} \quad |F| \leq nu|L| + \mathcal{O}(u^2).$$

We krijgen dus de oplossing die exact voldoet aan een naburig stelsel.
Het oplossen van een bovendriehoeksstelsel heet terugwaartse eliminatie of terugsubstitutie. Het wordt beschreven door het volgende algoritme.

Algoritme 10 *Zij $U \in R^{n \times n}$ een niet-singuliere bovendriehoeksmatrix en zij $b \in R^n$, dan wordt de oplossing van $Uy = b$ berekend via:*

voor $i = n, \ldots, 1$
 $y_i = (b_i - \sum_{j=i+1}^{n} y_j u_{i,j}) / u_{i,i}$
end i

Ook dit algoritme kost n^2 flops en het leidt tot een oplossing waarvoor geldt [8]:

$$(U+F)\hat{y} = b, \quad \text{met} \quad |F| \leq nu|U| + \mathcal{O}(u^2).$$

Opgave 9 *Herschrijf de algoritmen 9 en 10 zo dat de rechterleden b overschreven worden door de oplossingen y.*

7.2 LU decompositie

We zullen in deze paragraaf aannemen dat $A \in R^{n\times n}$ een niet-singuliere matrix is en dat we het stelsel $Ax = b$ wensen op te lossen. Bijna altijd kunnen via Gauss transformaties niet-singuliere matrices op bovendriehoeksgedaante gebracht worden.:

$$M_{n-1}M_{n-2}\cdots M_1 A = U,$$

zodat het stelsel $Ax = b$ overgaat in $Ux = M_{n-1}M_{n-2}\cdots M_1 b$.

Voorbeeld.

$$A = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 11 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

Dan volgt rechtstreeks:

$$M_1 = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -3 & 0 & 1 \end{pmatrix}, \quad M_1A = \begin{pmatrix} 1 & 4 & 7 \\ 0 & -3 & -6 \\ 0 & -6 & -10 \end{pmatrix}, \quad M_1b = \begin{pmatrix} 1 \\ -1 \\ -2 \end{pmatrix},$$

en tenslotte

$$M_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -2 & 1 \end{pmatrix}, \quad M_2M_1A = \begin{pmatrix} 1 & 4 & 7 \\ 0 & -3 & -6 \\ 0 & 0 & 2 \end{pmatrix}, \quad M_2M_1b = \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix}.$$

De oplossing laat zich nu eenvoudig berekenen als $x_3 = 0$, $x_2 = \frac{1}{3}$ en $x_1 = -\frac{1}{3}$. Tenslotte geldt nog

$$M_1^{-1}M_2^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 2 & 1 \end{pmatrix}$$

(zie ook opgave 8 en de tekst daarop volgend).

We leiden nu de algemene gedaante van de voor oplossen van een stelsel met A benodigde M_i's af. Stel dat we $M_1, \ldots, M_{k-1} \in R^{n\times n}$ al hebben bepaald en wel zodanig dat

$$A^{(k-1)} \equiv M_{k-1}\cdots M_1 A = \begin{pmatrix} A_{1,1}^{(k-1)} & A_{1,2}^{(k-1)} \\ & A_{2,2}^{(k-1)} \end{pmatrix},$$

waarbij $A_{1,1}^{(k-1)} \in R^{(k-1)\times(k-1)}$ een bovendriehoeksmatrix is.
We schrijven $A_{2,2}^{(k-1)}$ als

$$A_{2,2}^{(k-1)} = \begin{pmatrix} a_{k,k}^{(k-1)} & \cdots & a_{k,n}^{(k-1)} \\ \vdots & \ddots & \vdots \\ a_{n,k}^{(k-1)} & \cdots & a_{n,n}^{(k-1)} \end{pmatrix}.$$

De matrix elementen $a_{k+1,k}^{(k-1)}, \ldots, a_{n,k}^{(k-1)}$ in de eerste kolom van $A_{2,2}^{(k-1)}$ worden naar 0 getransformeerd met

$$\alpha^{(k)} = \left(0, \ldots, \frac{a_{k+1,k}^{(k-1)}}{a_{k,k}^{(k-1)}}, \ldots, \frac{a_{k+1,k}^{(k-1)}}{a_{k,k}^{(k-1)}}\right)^T,$$

$$M_k = I - \alpha^{(k)} e_k^T,$$

tenzij $a_{k,k}^{(k-1)} = 0$, en

$$A^{(k)} \equiv M_k A^{(k-1)} = \begin{pmatrix} A_{1,1}^{(k)} & A_{1,2}^{(k)} \\ & A_{2,2}^{(k)} \end{pmatrix},$$

met $A_{1,1}^{(k)} \in R^{k\times k}$ een bovendriehoeksmatrix.
Dus als $a_{k,k}^{(k-1)} \neq 0$ voor $k = 1, \ldots, n-1$, dan geldt

$$A^{(n-1)} = M_{n-1} \cdots M_1 A = U,$$

en verder:

$$(M_{n-1} \cdots M_1)^{-1} = \prod_{i=1}^{n-1} (I + \alpha^{(i)} e_i^T) = L.$$

De volgende stelling doet een uitspraak over de existentie van de zogenaamde LU-ontbinding van A.

Stelling 3 *Zij A_k de leidende $k \times k$ ondermatrix van $A \in R^{n\times n}$. Als A_k niet-singulier is voor $k = 1, \ldots, n-1$, dan bestaat er een $L \in R^{n\times n}$ en een $U \in R^{n\times n}$ zo dat $A = LU$. Verder geldt nog dat $det(A_k) = u_{1,1} \cdots u_{k,k}$, voor $k = 1, \ldots, n$.*

Bewijs. Het bewijs loopt volgens volledige inductie. Stel dat de eerste $k-1$ Gauss eliminatie stappen konden worden uitgevoerd, zonder op een diagonaal element 0 te stuiten. Ga zelf na dat geldt $A_k = (L^{(k-1)})_k (A^{(k-1)})_k$. Omdat

$$det(A_k) = a_{k,k}^{(k-1)} \cdot det(A_{1,1}^{(k-1)}) = a_{k,k}^{(k-1)} \cdot det(A_{k-1})$$

volgt dat $a_{k,k}^{(k-1)} \neq 0$ en dus kan de k-de Gauss eliminatiestap worden uitgevoerd.
Verder volgt uit de inductiestap dat $det(A_k) = a_{1,1}^{(0)} \cdots a_{k,k}^{(k-1)}$, met $a_{1,1}^{(0)} = a_{1,1}$.
Met $L = (M_{n-1} \cdots M_1)^{-1}$ en $U = A^{n-1}$ volgt tenslotte de bewering. □

De diagonaal elementen van U, de $a_{k,k}^{(k-1)}$, waardoor steeds gedeeld moet worden om de Gauss transformatie matrices te bepalen, heten de *pivots*.
Het oplossen van $Ax = b$ is na het bepalen van L en U kinderspel. We lossen eerst $Ly = b$ op en daarna $Ux = y$.

Opgave 10 *Ga na dat voor het bepalen van de oplossing x van $Ax = b$ ongeveer $2n^2$ flops nodig zijn (als L en U gegeven zijn) en dat b door de oplossing kan worden overschreven (er zijn dus geen twee extra arrays nodig voor het opbergen van x en y).*

Het volgende algoritme berekent de LU-ontbinding van A, mits aan de voorwaarden van stelling 3 voldaan is.

Algoritme 11 *Zij $A \in R^{n\times n}$. Het algoritme berekent de LU-ontbinding van A. De strikte benedendriehoek van A wordt overschreven door de $\ell_{i,j}$, de rest van A wordt overschreven door de $u_{i,j}$ (de elementen van U).*

voor $k = 1, \ldots, n-1$
  als $a_{k,k} = 0$ *dan STOP* (A_k *is singulier!)*
  anders:
  voor $i = k+1, \ldots, n$
    $w_i := a_{k,i}$ *(een rij vanaf de diagonaal)*
    voor $i = k+1, \ldots, n$
      $\eta := a_{i,k}/a_{k,k}$ $(= \ell_{i,k})$
      $a_{i,k} := \eta$
      voor $j = k+1, \ldots, n$
        $a_{i,j} := a_{i,j} - \eta w_j$ *(transformatie slag)*
      end j
    end i
end k

Het algoritme is zeer zeker niet robuust, want het breekt al af voor de matrix

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

Als we van deze matrix de rijen verwisselen is er niets aan de hand. Er zijn dus nog ingrijpende veranderingen nodig om van algoritme 11 een robuust algoritme te maken. Verder zagen we al in stelling 2, en de conclusie daarop, dat de relatieve fouten in het algoritme per rekenslag groot kunnen zijn (als de pivots relatief zeer klein zijn). Ook daar moet wat aan gebeuren.
Ga zelf na dat uitvoering van het algoritme 11 ongeveer $2n^3/3$ flops vereist.

Opgave 11 *Ga na dat het ook mogelijk is de Gauss transformaties van "onder af aan" uit te voeren, zodanig dat $A = UL$, waarbij U een bovendriehoeksmatrix is met $u_{i,i} = 1$ en L benedendriehoeks. Merk op dat nu U de inverse is van de samengestelde Gauss transformaties.*

7.3 Foutenanalyse van de LU-decompositie

We hadden al eerder gezien dat voor ϵ klein genoeg, de relatieve fout in de oplossing x van $Ax = b$, ten gevolge van fouten in A (F) en b (f) begrensd wordt door:

$$\frac{\|x(\epsilon) - x\|}{\|x\|} \le \kappa(A)\epsilon\{\frac{\|F\|}{\|A\|} + \frac{\|f\|}{\|b\|}\} + \mathcal{O}(\epsilon^2), \text{ met } (A + \epsilon F)x(\epsilon) = b + \epsilon f.$$

Dit resultaat is in de praktijk niet erg bruikbaar, omdat er niet bij vermeld wordt wat "ϵ klein genoeg" betekent. We kunnen ook een stelling formuleren die daar wel een duidelijke uitspraak over doet. We bewijzen daarvoor eerst een stelling over de convergentie van Neumann reeksen.

Stelling 4 *Zij $E \in R^{n\times n}$ en zij, voor enige geschikte matrix p-norm $\|E\| < 1$, dan is $I - E$ niet singulier en*

$$(I-E)^{-1} = \sum_{k=0}^{\infty} E^k \quad , \quad \|(I-E)^{-1}\| \leq (I - \|E\|)^{-1}.$$

Bewijs: Neem aan dat $I - E$ singulier is, dan is er een $y \neq 0$ met $Ey = y$. Daaruit volgt $\|E\| = \sup_{x\neq 0} \frac{\|Ex\|}{\|x\|} \geq \frac{\|Ey\|}{\|y\|} = 1$, hetgeen tot een tegenspraak leidt.
Definieer

$$P \equiv (I-E)^{-1} - \sum_{k=0}^{N} E^k.$$

Dan geldt $I - (I-E)\sum_{k=0}^{N} E^k = (I-E)P \equiv B$. Dus $B = E^{N+1}$.
Verder $P = B + EP \Rightarrow \|B\| \geq \|P\|(1 - \|E\|)$.
Hieruit volgt:

$$\|E\|^{N+1} \geq \|E^{N+1}\| \geq \|P\|(1 - \|E\|)$$

en

$$\|P\| \leq \frac{\|E\|^{N+1}}{1 - \|E\|}.$$

Als $\|E\| < 1$, dan geldt dus $\lim_{N\to\infty} \|P\| = 0$ en daaruit volgt dat

$$(I-E)^{-1} = \sum_{k=0}^{\infty} E^k.$$

Tenslotte volgt nog

$$\|(I-E)^{-1}\| \leq \sum_{k=0}^{\infty} \|E\|^k = (1 - \|E\|)^{-1}.$$

$\square$

Nu dan de beloofde stelling.

Stelling 5 *Zij $b \in R^n$ en $A \in R^{n\times n}$ niet-singulier en zij x oplossing van het stelsel $Ax = b$.*
a. Als voor $\Delta A \in R^{n\times n}$ geldt $\|\Delta A\|\|A^{-1}\| = r < 1$, dan is het stelsel $(A + \Delta A)y = b + \Delta b$ niet-singulier.
b. Als bovendien $\|\Delta A\| \leq \epsilon\|A\|$ en $\|\Delta b\| \leq \epsilon\|b\|$ voor $\epsilon \geq 0$, dan

$$\frac{\|x-y\|}{\|x\|} \leq \frac{2\epsilon}{1-r}\kappa(A), \quad \textit{voor } x \neq 0.$$

Bewijs: $y = (A+\Delta A)^{-1}b + (A+\Delta A)^{-1}\Delta b$, gecombineerd met $x = A^{-1}b$ leidt tot

$$y - x = ((A+\Delta A)^{-1} - A^{-1})b + (A+\Delta A)^{-1}\Delta b.$$

Uit $A + \Delta A = A(I + A^{-1}\Delta A)$ volgt $(A+\Delta A)^{-1} = (I + A^{-1}\Delta A)^{-1}A^{-1}$ en dus

$$\begin{aligned}(A+\Delta A)^{-1} - A^{-1} &= \{(I + A^{-1}\Delta A)^{-1} - I\}A^{-1}\\ &= -A^{-1}\Delta A(I + A^{-1}\Delta A)^{-1}A^{-1}.\end{aligned}$$

Enig gegoochel met normen leidt dan tot

$$\begin{aligned}\|y - x\| &\le \|A^{-1}\|\|\Delta A\|\|(I + A^{-1}\Delta A)^{-1}\|\|x\|\\ &\quad+\|(I + A^{-1}\Delta A)^{-1}\|\|A^{-1}\|\|\Delta b\|\\ &\le \epsilon\|A\|\|A^{-1}\|\frac{1}{1-r}\|x\| + \frac{1}{1-r}\|A^{-1}\|\epsilon\|b\|\\ &\le \frac{2\epsilon}{1-r}\|A^{-1}\|\|A\|\|x\|.\end{aligned}$$

Als $x \neq 0$ dan geldt dus blijkbaar

$$\frac{\|y - x\|}{\|x\|} \le \frac{2\epsilon}{1-r}\kappa(A).$$

□

Voordat we het stelsel $Ax = b$ met Gauss eliminatie gaan oplossen, vragen we ons eerst af wat we aan foutbijdrage mogen verwachten, louter door het representeren van A en b door machine getallen. Dus zij $fl(b) = b + e$ en $fl(A) = A + E$ en neem aan dat $A + E$ niet-singulier is. Dan voldoet $\hat{x}$ aan

$$(A+E)\hat{x} = b + e, \quad \text{met } \|E\|_\infty \le u\|A\|_\infty,\ \|e\|_\infty \le u\|b\|_\infty. \tag{7.1}$$

Als nu het konditiegetal van A niet al te beroerd is, zeg $\kappa_\infty(A) \le \frac{1}{2u}$, dan geldt volgens stelling 5:

$$\frac{\|x - \hat{x}\|_\infty}{\|x\|_\infty} \le 4u\kappa_\infty(A).$$

We mogen dus niet verwachten dat enig algoritme, althans in $\| \|_\infty$-norm gemeten, tot een kleinere waarde voor de fout leidt.

Alvorens we nu tot een foutanalyse van het Gauss eliminatie algoritme overgaan, analyseren we eerst eens het foutgedrag voor een zeer eenvoudig probleem en zullen we zien dat een zorgvuldige analyse soms zeer nauwkeurige rekenresultaten kan verklaren.
We bekijken het volgende probleem. Los op $Ax = y$, met

$$A = \begin{pmatrix} 1 & & & & & \\ -a_2 & 1 & & & & \\ & -a_3 & 1 & & & \\ & & \ddots & \ddots & & \\ & & & \ddots & \ddots & \\ & & & & -a_n & 1 \end{pmatrix}.$$

We nemen aan dat de a_i en de y_i machinegetallen zijn.
Vraag: Hoe zou de relatieve fout eruit zien, als we rechtstreeks stelling 5 samen met het foutgedrag voor algoritme 9 toepassen?

Voor het rechtstreeks oplossen van het gegeven bidiagonale stelsel geldt in floating point arithmetiek:

$$\begin{aligned}\hat{x}_1 &= y_1 \equiv \hat{y}_1 \\ \hat{x}_i &= (y_i + a_i\hat{x}_{i-1}(1+\xi_{i\times}))(1+\xi_{i+}), \quad \text{met } |\xi_{i\times}|, |\xi_{i+}| \leq u, \\ &= \hat{y}_i + \hat{a}_i\hat{x}_{i-1},\end{aligned}$$

met $\hat{y}_i \equiv y_i(1+\xi_{i+})$ en $\hat{a}_i \equiv a_i(1+\xi_{i+})(1+\xi_{i\times})$.
Kennelijk kunnen we $\hat{x}$ opvatten als de exacte oplossing van het stelsel $\hat{A}\hat{x} = \hat{y}$.
Zij $\xi_+ \equiv diag(0, \xi_{2+}, \xi_{3+}, \ldots, \xi_{n+})$, dan geldt

$$\hat{y} = y + \xi_+ y = (I + \xi_+)y \equiv y + \Delta y,$$

en evenzo $\hat{A} = A + \Delta A$, met

$$\begin{aligned}\Delta A &= (I+\xi_+)(I+\xi_\times)(A-I) - (A-I) \\ &= (\xi_+ + \xi_\times + \xi_+\xi_\times)(A-I) \\ &\equiv \xi(A-I), \quad \text{met } \|\xi\|_\infty \leq 2u + u^2.\end{aligned}$$

Met $\hat{x} = x + \Delta x$ geldt

$$\begin{aligned}(A+\Delta A)(x+\Delta x) &= y + \Delta y \\ (A+\Delta A)\Delta x &= \Delta y - \Delta A x \\ &= \{\xi_+ - (\xi_\times + \xi_+\xi_\times)(A-I)\}x \equiv f.\end{aligned}$$

Zij nu $\|\Delta A\|_\infty \|A^{-1}\|_\infty = r < 1$. Dan geldt

$$\|(A+\Delta A)^{-1}\|_\infty \leq \|A^{-1}\|_\infty \frac{1}{1-r}.$$

Verder geldt nog

$$\begin{aligned}\|\xi_+ - (\xi_\times + \xi_+\xi_\times)(A-I)\|_\infty &\leq u + (u+u^2)\max_i |a_i| \\ &\leq (u+u^2)(1+\max_i |a_i|) = (u+u^2)\|A\|_\infty.\end{aligned}$$

Het resultaat wordt dus:

$$\frac{\|\Delta x\|_\infty}{\|x\|_\infty} \leq \frac{u+u^2}{1-r}\|A\|_\infty \|A^{-1}\|_\infty.$$

Als $|a_i| \leq 1$ dan volgt $\|A - I\|_\infty \leq 1$ en $\|A^{-1}\|_\infty \leq n$. De voorwaarde voor r luidt in dat geval

$$(2u + u^2)n = r < 1.$$

We zien dat de rechtstreekse oplossing van het stelsel tot de "beste x" leidt die we op grond van het conditiegetal van A hadden kunnen verwachten.

We gaan nu over tot de foutanalyse van het Gauss eliminatie proces zelf. De volgende stelling doet een uitspraak over de fout in de LU-ontbinding.

Stelling 6 *Zij $A \in R^{n\times n}$ een matrix van floating point getallen. Als alle pivots tijdens het eliminatieprocess ongelijk aan nul zijn, dan voldoen de resulterende driehoeksmatrices $\hat{L}$ en $\hat{U}$ aan*

$$\hat{L}\hat{U} = A + H, \quad met\ |H| \le 2(n-1)u\{|A| + |\hat{L}||\hat{U}|\} + \mathcal{O}(u^2).$$

Bewijs: Het bewijs gaat volgens volledige inductie. We tonen eerst aan dat het beweerde geldt voor $n = 2$:

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix},$$

en LU ontbinding in floating point arithmetiek leidt tot
$\hat{\ell}_{2,1} = fl(\frac{a_{2,1}}{a_{1,1}}) = \frac{a_{2,1}}{a_{1,1}} + f$, met $|f| \le |\frac{a_{2,1}}{a_{1,1}}|u$ en

$$\begin{aligned} \hat{u}_{2,2} &= fl(a_{2,2} - fl(\hat{\ell}_{2,1}a_{1,2})) \\ &= fl(a_{2,2} - \hat{\ell}_{2,1}a_{1,2}(1+\epsilon_1)) \\ &= (a_{2,2} - \hat{\ell}_{2,1}a_{1,2}(1+\epsilon_2))(1+\epsilon_1) \\ &= a_{2,2} - \hat{\ell}_{2,1}a_{1,2} + F \end{aligned}$$

met $|F| \le 2u(a_{2,2} + |\hat{\ell}_{2,1}||a_{1,2}|)$.
Nu geldt dat

$$\begin{pmatrix} 1 & 0 \\ \hat{\ell}_{2,1} & 1 \end{pmatrix} \begin{pmatrix} a_{1,1} & a_{1,2} \\ 0 & \hat{u}_{2,2} \end{pmatrix} \equiv \hat{L}\hat{U}$$

$$= \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{1,1}\hat{\ell}_{2,1} & a_{1,2}\hat{\ell}_{2,1} + \hat{u}_{2,2} \end{pmatrix} = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} + a_{1,1}f & a_{2,2} + F \end{pmatrix}.$$

Hieruit volgt het beweerde voor $n = 2$.

Neem nu aan dat de bewering geldt voor alle floating point matrices $A \in R^{(n-1)\times(n-1)}$. Zij vervolgens

$$A = \begin{pmatrix} \alpha & w^T \\ v & B \end{pmatrix} \quad \text{met } B \in R^{(n-1)\times(n-1)}.$$

Dan geldt voor de eerste stap van het Gauss eliminatie proces, toegepast op A:

$$\hat{z} \equiv fl(\frac{v}{\alpha}) \ \text{ en } \hat{B}_1 \equiv fl(B - \hat{z}w^T).$$

Dus $\hat{z} = \frac{1}{\alpha}v + f$ met $|f| \le \frac{u}{|\alpha|}|v|$ en $\hat{B}_1 = B - \hat{z}w^T + F$, waarbij

$$|F| \le 2u\{|B| + |\hat{z}||w^T|\} + \mathcal{O}(u^2).$$

Vervolgens wordt de LU decompositie van $\hat{B}_1$ berekend. Volgens de aanname geldt

$$\hat{L}_1\hat{U}_1 = \hat{B}_1 + H_1,$$

$$\text{met } |H_1| \le 2(n-2)u\{|\hat{B}_1| + |\hat{L}_1||\hat{U}_1|\} + \mathcal{O}(u^2).$$

Dus:

$$\hat{L}\hat{U} \equiv \begin{pmatrix} 1 & 0 \\ \hat{z} & \hat{L}_1 \end{pmatrix} \begin{pmatrix} \alpha & w^T \\ 0 & \hat{U}_1 \end{pmatrix} = \begin{pmatrix} \alpha & w^T \\ \alpha\hat{z} & \hat{z}w^T + \hat{L}_1\hat{U}_1 \end{pmatrix}$$

$$= \begin{pmatrix} \alpha & w^T \\ v + \alpha f & \hat{z}w^T + B - \hat{z}w^T + F + H_1 \end{pmatrix} = A + \begin{pmatrix} 0 & 0 \\ \alpha f & H_1 + F \end{pmatrix} = A + H.$$

Er geldt dat

$$|\hat{B}_1| \leq (1 + 2u)\{|B| + |\hat{z}||w^T|\}$$

en dus

$$|H_1 + F| \leq |H_1| + |F| \leq 2(n-2)u\{|B| + |\hat{z}||w^T| + |\hat{L}_1||\hat{U}_1|\}$$
$$+2u\{|B| + |\hat{z}||w^T|\} + \mathcal{O}(u^2)$$

zodat

$$|H| \leq 2(n-1)u\left\{\begin{pmatrix} |\alpha| & |w^T| \\ |v| & |B| \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ |\hat{z}| & |\hat{L}_1| \end{pmatrix} \times \begin{pmatrix} |\alpha| & |w^T| \\ 0 & |\hat{U}_1| \end{pmatrix}\right\} + \mathcal{O}(u^2).$$

□

In de volgende stelling combineren we de voorgaande resultaten tot een foutschatting voor de oplossing van het stelsel $Ax = b$ met het Gauss eliminatie algoritme.

Stelling 7 *Zij $\hat{L}\hat{U}$ de met algoritme 11 berekende ontbinding van de floating point matrix $A \in R^{n\times n}$, $\hat{y}$ de berekende oplossing van $\hat{L}y = b$ (via algoritme 9) en $\hat{x}$ de berekende oplossing van $\hat{U}x = \hat{y}$ (via algoritme 10).*
Dan geldt $(A + E)\hat{x} = b$, met

$$|E| \leq 2nu\{|A| + 2|\hat{L}||\hat{U}|\} + \mathcal{O}(u^2).$$

Bewijs: Voor de algoritmen 9 en 10 geldt:

$$(\hat{L} + F)\hat{y} = b \quad \text{met} \quad |F| \leq nu|\hat{L}| + \mathcal{O}(u^2)$$

$$(\hat{U} + G)\hat{x} = \hat{y} \quad \text{met} \quad |G| \leq nu|\hat{U}| + \mathcal{O}(u^2)$$

en dus $(\hat{L}\hat{U} + F\hat{U} + \hat{L}G + FG)\hat{x} = b$.
Volgens Stelling 6 geldt $\hat{L}\hat{U} = A + H$ met

$$|H| \leq 2(n-1)u\{|A| + |\hat{L}||\hat{U}|\} + \mathcal{O}(u^2).$$

Door E te definiëren als $E \equiv |H| + F\hat{U} + \hat{L}G + FG$ volgt $(A + E)\hat{x} = b$, met

$$|E| \leq 2nu\{|A| + 2|\hat{L}||\hat{U}|\} + \mathcal{O}(u^2).□$$

Even afgezien van het feit dat $|\hat{L}||\hat{U}|$ groot kan zijn, scheelt de bovengrens in stelling 7 "slechts" een factor $2n$ met die in (7.1), die gebaseerd was op louter de afrondfouten in A en b. Het zijn echter de factoren $|\hat{L}|$ en $|\hat{U}|$ die grote elementen kunnen hebben als er kleine pivots zijn opgetreden en daardoor roet in het eten gooien. We geven eerst een voorbeeld dat in zo'n geval de optredende fouten vrij realistisch worden afgeschat door de zojuist afgeleide bovengrens.

Voorbeeld.
In dit voorbeeld rekenen we in 3 decimalen en de (tussen) resultaten worden steeds afgebroken. We gaan het volgende stelsel oplossen:

$$\begin{pmatrix} 1.00 \cdot 10^{-3} & 1.00 \\ 1.00 & 8.00 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1.00 \\ 9.00 \end{pmatrix}.$$

Via Gauss eliminatie ontstaan de volgende factoren

$$\hat{L} = \begin{pmatrix} 1 & 0 \\ 1.00 \cdot 10^{3} & 1 \end{pmatrix}, \quad \hat{U} = \begin{pmatrix} 1.00 \cdot 10^{-3} & 1.00 \\ 0 & -1.00 \cdot 10^{3} \end{pmatrix}.$$

Omdat

$$\hat{L}\hat{U} = \begin{pmatrix} 1.00 \cdot 10^{-3} & 1.00 \\ 1.00 & 0 \end{pmatrix},$$

volgt dus dat

$$\hat{L}\hat{U} = A + \begin{pmatrix} 0 & 0 \\ 0 & -8 \end{pmatrix} = A + H.$$

Stelling 7 zegt dat

$$|H| \leq 2u \left\{ \begin{pmatrix} 1.00 \cdot 10^{-3} & 1.00 \\ 1.00 & 8.00 \end{pmatrix} + \begin{pmatrix} 1.00 \cdot 10^{-3} & 1.00 \\ 1.00 & 2.00 \cdot 10^{3} \end{pmatrix} \right\},$$

hetgeen met $u = 10^{-2}$ leidt tot

$$|H| \leq \begin{pmatrix} 2.00 \cdot 10^{-5} & 2.00 \cdot 10^{-2} \\ 2.00 \cdot 10^{-2} & 40.16 \end{pmatrix}$$

en dit is slechts een overschatting van de werkelijke fout met een factor 5. Oplossen van het gesplitste stelsel via de algoritmen 9 en 10 leidt dan tot $\hat{x} = (0, 1)^T$, hetgeen nogal afwijkt van de ware oplossing $x = (0.9989\cdots, 1.008\cdots)^T$.

Merk op dat kleine pivots niets te maken hoeven te hebben met een slechte conditie van A. Denk maar aan $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$. De tot dusverre behandelde Gauss eliminatie methode is dus in hoge mate instabiel.

7.4 LU decompositie met pivoting

We hebben tot nu toe steeds aangenomen dat de optredende pivots ongelijk aan nul zijn, maar we hebben ook waargenomen dat een kleine pivot een catastrofale doorwerking kan hebben. Een manier om kleine pivots te voorkomen is er voor te zorgen dat door geschikt verwisselen van rijen en/of kolommen van $A^{(1)}$ het in absolute waarde grootste element op de pivotplaats komt

te staan. Op deze manier houden we in elk geval de elementen van de Gauss transformatie matrix, nodig om $A^{(2)}$ te construeren, zo klein mogelijk. Deze techniek staat bekend onder de naam *complete pivoting*.
We moeten natuurlijk wel onthouden hoe er er tijdens het proces verwisseld is: verwisseling van rijen geeft aanleiding tot eenzelfde verwisseling van de elementen in het rechterlid b, terwijl verwisseling van kolommen weer goed gemaakt dient te worden door een overeenkomstige verwisseling van de elementen van de verkregen oplossing.

Men gaat gemakkelijk na dat bij compleet pivotten de elementen van $\hat{L}$ allemaal ≤ 1 zijn. Men kan eveneens zelf nagaan dat compleet pivotten aanleiding geeft tot een LU ontbinding van de matrix $\Pi_1 A \Pi_2$, waarbij Π_1 de rijen verwisselt en Π_2 de kolommen verwisselt van A (Π_1 en Π_2 zijn permutatie matrices).
Er geldt dan $Ly = \Pi_1 b$ en $Uz = y$, $x = \Pi_2 z$. Volgens stelling 7 geldt

$$(\Pi_1 A \Pi_2 + E)\hat{z} = \Pi_1 b, \quad \text{met } |E| \leq 2nu\{|\Pi_1 A \Pi_2| + 2|\hat{L}||\hat{U}|\} + \mathcal{O}(u^2).$$

Bij de operaties met Π_1 en Π_2 worden natuurlijk geen afrondfouten gemaakt.

Bij compleet pivotten geldt $\|\hat{L}\|_\infty \leq n$, en dus

$$\|E\|_\infty \leq nu\{2\|A\|_\infty + 4n\|\hat{U}\|_\infty\} + \mathcal{O}(u^2).$$

Een probleem blijft nu nog om $\|\hat{U}\|_\infty$ geschikt te begrenzen. We definiëren daarvoor de zogenaamde *groeifactor* ρ

$$\rho = \max_{i,j,k} \frac{|\hat{a}_{i,j}^{(k)}|}{\|A\|_\infty},$$

met $\hat{A}^{(k)}$ de berekende versie van $A^{(k)}$. Men kan bewijzen, zie [26], dat $\|E\|_\infty \leq 6n^3\rho\|A\|_\infty u + \mathcal{O}(u^2)$.
De factor n^3 is erg aan de pessimistische kant en het blijkt empirisch dat ρ beperkt blijft (het zg. wonder van Wilkinson): $\rho \leq 10$. Dit leidt ons tot de conclusie dat Gauss eliminatie met complete pivoting in de praktijk stabiel is (maar men blijve op zijn hoede).

Complete pivoting is een kostbare aangelegenheid, het betekent in de rekenpraktijk veelal dat het LU decompositie algoritme er een factor 2 duurder op wordt (en dan hadden we net zo goed Householder transformaties kunnen gebruiken), en op speciale machines (parallelle) kan het nog erger uitpakken. Daarom beperkt men in de praktijk het pivot zoeken meestal tot de eerste kolom van de aan de beurt zijnde submatrix. Dat heet *partial pivoting*. Er geldt dan nog steeds $\|\hat{L}\|_\infty \leq n$, echter de groeifactor ρ wordt nu groter. Empirisch blijkt evenwel deze groeifactor weer van dezelfde grootteorde te zijn als bij complete pivoting. Er is echter een voorbeeld geconstrueerd waarvoor $\rho \sim 2^n$.
Aangezien het voorkomen van dit soort stelsels in de praktijk uiterst onwaarschijnlijk is, kan men partial pivoting met enig vertrouwen toepassen. De beschikbare FORTRAN subroutines in LINPACK, NAG, IMSL en de codes in MATHLAB werken allen volgens dit principe. In sommige gevallen kan men de groeifactor bij laten houden, zodat men zich achteraf een beeld kan vormen van de eventueel aangerichte verwoestingen.

7.5 Verbeteren en schatten van de fout in de oplossing

Als we $Ax = b$ oplossen via Gauss eliminatie met partial pivoting, dan vinden we dus blijkbaar de oplossing van een naburig stelsel $(A+E)\hat{x} = b$. Combinatie van de stellingen 5 en 7, met de "heuristiek" in de vorige paragraaf, leidt tot het inzicht dat

$$\frac{\|\hat{x}-x\|_\infty}{\|x\|_\infty} \cong 2\kappa_\infty(A) \cdot \frac{\|E\|_\infty}{\|A\|_\infty},$$

waarbij we aangenomen hebben dat $\|E\|_\infty \|A^{-1}\|_\infty \ll 1$.

Gauss eliminatie met partial pivoting is, practisch gezien, numeriek stabiel, en dus, rekenend met k decimalen floating point, hebben we dat

$$\frac{\|E\|_\infty}{\|A\|_\infty} \approx c10^{-k},$$

waarbij de constante c niet van A of b afhangt. Indien we verder aannemen dat $\kappa_\infty = 10^q$, dan geldt blijkbaar

$$\frac{\|\hat{x}-x\|_\infty}{\|x\|_\infty} \cong 2c10^{-k+q},$$

en

$$\|A\hat{x}-b\|_\infty = \|E\hat{x}_\infty\| \leq c10^{-k}\|A\|_\infty\|\hat{x}\|_\infty. \tag{7.2}$$

Dus Gauss eliminatie levert een $\hat{x}$ op met een relatief klein residu, terwijl de oplossing $\hat{x}$ zelf ongeveer $k-q$ correcte decimalen heeft.
De volgende vragen doemen op:

- Kunnen we A nog voorbewerken zodat we een betere oplossing $\hat{x}$ krijgen?
- Kunnen we de verkregen oplossing achteraf nog verbeteren?
- Is het mogelijk de fout in $\hat{x}$ te schatten, d.w.z., kunnen we op goedkope wijze een indruk krijgen van $\kappa_\infty(A)$?

In de volgende subparagrafen wordt op deze vragen ingegaan.

7.5.1 Schaling

Het voor de hand liggende antwoord op de vraag naar een voorbewerking van A met het oog op een betere oplossing, wordt gegeven door schaling. Om afrondfouten te vermijden schalen we A met matrices D die als diagonaal elementen gehele machten van het machine grondtal hebben:

$$D_1 = diag(\beta^{i_1}, \ldots, \beta^{i_n}),$$

$$D_2 = diag(\beta^{j_1}, \ldots, \beta^{j_n}).$$

In plaats van het gegeven stelsel $Ax = b$ lossen we het geschaalde stelsel

$$D_1 A D_2 y = D_1 b$$

op via $\Pi D_1 A D_2 = LU$, waarbij Π de permutatiematrix van het partial pivotten is, gevolgd door

$$Lw = \Pi D_1 b$$

$$Uy = w$$

en tenslotte $x = D_2 y$.
Merk op dat D_1 de vergelijkingen schaalt en D_2 de onbekenden. Er geldt

$$\frac{\|D_2^{-1}(\hat{x}-x)\|_\infty}{\|D_2^{-1}x\|_\infty} = \frac{\|\hat{y}-y\|_\infty}{\|y\|_\infty} \cong c10^{-k}\kappa_\infty(D_1 A D_2).$$

Een techniek om $\kappa_\infty(D_1 A D_2)$ redelijk binnen de perken te houden is rijschaling: D_2=I en D_1 zodanig dat $D_1 A$ rijen heeft van allemaal ongeveer dezelfde ∞-norm. Door rijschaling wordt de waarschijnlijkheid op het optellen van zeer kleine getallen bij zeer grote getallen (het "verdrinken" van resultaten) wat verminderd. Men dient zich echter wel te realiseren dat, door het kleiner maken van $\kappa_\infty(D_1 A D_2)$, we dan wel een kleinere relatieve fout bewerkstelligd hebben, maar deze fout is slechts gunstig, gemeten in een met D_2^{-1} geschaalde norm. Men moet dan ook met schaling oppassen en het is heel wel mogelijk om door dit soort schaling een slechtere oplossing $\hat{x}$ te verkrijgen. De beste taktiek lijkt nog te zijn er voor te zorgen dat de foutmatrix $\tilde{E}$, die de onzekerheden in A representeert, globaal elementen van dezelfde grootte-orde heeft (voor details zie [10]).

7.5.2 Iteratief naverfijnen

Stel dat we $Ax = b$ hebben opgelost via $\Pi A = LU$ en dat we nu $\hat{x}$ achteraf nog willen verbeteren. In principe kan dat door het residu $r = b - A\hat{x}$ te bepalen en een correctie op de oplossing te berekenen via

$$Ly = \Pi r, \quad Uz = y \text{ en } \tilde{x} = \hat{x} + z.$$

In exacte aritmetiek rekenend geldt dan

$$A\tilde{x} = A(\hat{x} + z) = A\hat{x} + Az = (b - r) + r = b.$$

Echter, omdat $fl(b - A\hat{x})$ relatief erg klein is ten opzichte van b, zie vergelijking (7.2), mogen we niet verwachten dat de elementen van $fl(b - A\hat{x})$ veel significante decimalen bevat: misschien wel geen. Dus $\tilde{x}$ hoeft in principe, wanneer we rekenen in machine arithmetiek, niet beter te zijn dan $\hat{x}$, immers $z = A^{-1}$"$ruis$" = "$ruis$".
Om echt een wezenlijke verbetering te verkrijgen moet daarom het residu $b - A\hat{x}$ in hogere (bijvoorbeeld dubbele) precisie worden uitgerekend. Het *iteratie naverfijnings algoritme* komt er dan als volgt uit te zien:

Algoritme 12 *Zij $A \in R^{n\times n}$. Het algoritme om de oplossing van $Ax = b$ met iteratief naverfijnen te bepalen, ziet er schematisch als volgt uit:*

begin met $x = 0$
Bepaal met partial pivotten de LU ontbinding van A:
  $\Pi A = LU$
Doe vervolgens zo vaak als nodig is:

$r = b - Ax$ (in dubbele precisie !)
Los op $Ly = \Pi r$
Daarna $Uz = y$
en overschrijf x door $x + z$.

Het is hierbij essentieel dat $r = b - Ax$ in dubbele precisie wordt uitgerekend met de **originele** A, en niet bijvoorbeeld via de factoren L en U waarmee we A wellicht overschreven hebben. Zowel de factoren als A moeten dus bewaard blijven.

Als $u\kappa_\infty(A) < 1$ dan kan het proces uiteindelijk een x opleveren, waarvoor $Ax - b$ componenten van de orde u (relatief t.o.v. b) heeft.
Het iteratie naverfijningsalgoritme is relatief goedkoop. Iedere iteratieslag kost $\mathcal{O}(n^2)$ flops, terwijl het decompositieproces $\Pi A = LU$ zelf $\mathcal{O}(n^3)$ flops kost. Een nadeel is wel dat de implementatie machineafhankelijk is, immers het rekenen in dubbele precisie is per machine anders geregeld. Op computers waarop het rekenen doorgaans toch al in dubbele precisie plaats vindt (bv IBM, Convex, ...), zou het residu in 4-dubbele precisie moeten worden uitgerekend. Het proces is zeer de moeite waard als men normaliter in weinig decimalen rekent.

7.5.3 Het schatten van het conditiegetal

Om een indruk te kunnen krijgen van het aantal goede decimalen in de verkregen oplossing is een schatting van het conditiegetal $\kappa_\infty(A) = \|A\|_\infty \|A^{-1}\|_\infty$ nodig.
Er geldt $\|A\|_\infty = \max_i \sum_j |a_{i,j}|$ en het probleem is gelegen in de berekening van $\|A^{-1}\|_\infty$. Berekening van A^{-1} via het oplossen van $Ax_i = e_i$, voor $i = 1, \ldots, n$, levert een benadering (nl. zij $\hat{X}$ de matrix met kolommen x_i, dan geldt $\|\hat{X}^{-1}\|_\infty \approx \|A^{-1}\|_\infty$, indien $\|A\|_\infty \|A^{-1}\|_\infty$ niet al te groot).
Echter, de berekening van $\hat{\kappa}_\infty(A) = \|A\|_\infty \|\hat{X}^{-1}\|_\infty$ kost dan drie keer zoveel als de berekening van $\hat{x}$.

We zullen nu een methode beschrijven om uitgaande van de ontbinding $\Pi A = LU$ goedkoop $\|A^{-1}\|_\infty$ te schatten [3].
Uit $Ay = d$ volgt $y = A^{-1}d$ en dus $\|y\|_\infty \leq \|A^{-1}\|_\infty \|d\|_\infty$. Dus

$$\|A^{-1}\|_\infty \geq \frac{\|y\|_\infty}{\|d\|_\infty},$$

en het idee is nu om vectoren y en d te construeren, zodat de norm van A^{-1} zo dicht mogelijk benaderd wordt.

We beginnen met de situatie dat $A = U$ een bovendriehoeksmatrix is te beschouwen. Dan geldt $y_n = d_n/u_{n,n}$ en we kunnen van het stelsel alvast 1 kolom elimineren. Het geëlimineerde gedeelte slaan we op in de vector p:

$$p_j = u_{j,n} y_n, \quad \text{voor } j = 1, \ldots, n-1.$$

Van het resterende stelsel kunnen we weer de laatste kolom elimineren, met $y_{n-1} = (d_{n-1} - p_{n-1})/u_{n-1,n-1}$. Dit leidt tot

$$p_j = p_j + u_{j,n-1} y_{n-1}, \quad \text{voor } j = 1, \ldots, n-2.$$

Zo kunnen we verder gaan en het hele eliminatieschema komt er als volgt uit te zien:

begin met $p_j = 0$, voor $j = 1, \ldots, n$
Voor $k = n, \ldots, 1$:
  $y_k = (d_k - p_k)/u_{k,k}$
  voor $i = 1, \ldots, k-1$:
    $p_i = p_i + u_{i,k} y_k$

De truc bestaat er nu uit de getallen d_k zo te kiezen dat y_k zo groot mogelijk is. Een eerste aanpak is d_k de waarde -1 of 1 te geven al naar gelang $(1-p_k)/u_{k,k}$ of $(-1-p_k)/u_{k,k}$ de grootste absolute waarde heeft.

Omdat $d = (\pm 1, \ldots, \pm 1)^T$, geldt $\|d\|_\infty = 1$ en de schatting voor κ_∞ wordt:

$$\hat{\kappa}_\infty = \|y\|_\infty \|A\|_\infty .$$

Een nog weer slimmere aanpak is de $d_k = \pm 1$ zo te kiezen dat zowel de geaccumuleerde sommen p_i als y_i beide zo groot mogelijk worden. Deze taktiek wordt uitgevoerd in het volgende algoritme, dat om later duidelijk wordende redenen, wordt toegepast op de situatie $A = L$.

Algoritme 13 *Zij $L \in R^{n \times n}$ een niet-singuliere benedendriehoeksmatrix. $w_1, \ldots, w_n$ zijn gegeven positieve weegfactoren (een gebruikelijke keuze is $w_i = 1/|\ell_{i,i}|$). Het algoritme berekent een $y \in R^n$ zodanig dat $\|y\|_\infty$ de waarde van $\|L\|_\infty$ benadert.*

begin met $p_j = 0$, *voor* $j = 1, \ldots, n$
Voor $k = 1, \ldots, n$:
  $y_k^+ = (1 - p_k)/\ell_{k,k}$
  $y_k^- = (-1 - p_k)/\ell_{k,k}$
  $S^+ = |y_k^+| + \sum_{k+1}^n w_i |p_i + \ell_{i,k} y_k^+|$
  $S^- = |y_k^-| + \sum_{k+1}^n w_i |p_i + \ell_{i,k} y_k^-|$
  Als $S^+ \geq S^-$ *dan* $y_k = y_k^+$ *anders* $y_k = y_k^-$
  voor $i = k+1, \ldots, n$:
    $p_i = p_i + \ell_{i,k} y_k$

Dit algoritme kost $\approx 5n^2$ flops. Als we $w_i = 0$ kiezen, dan hebben we het voorgaande algoritme te pakken. De keuze $w_i = 1/|\ell_{i,i}|$ ligt voor de hand, omdat de betreffende getallen "zodra ze aan de beurt zijn" door $\ell_{i,i}$ gedeeld moeten worden. Bij de normale LU decompositie geldt $\ell_{i,i} = 1$, maar we zullen ook nog versies tegenkomen waarbij $\ell_{i,i} \neq 1$.

De gang van zaken voor het schatten van $\|A^{-1}\|_\infty$ wordt nu:
(a) Pas algoritme 13 eerst toe op $U^T y = d$ om een grote y te verkrijgen.
(b) Los op $L^T r = y$.

Toelichting: Eigenlijk hadden we eerst op moeten lossen $Ly = d$ en dan pas $Ur = y$. De gevolgde filosofie is dat veelal U veel slechtere norm zal hebben dan L. Daarom passen we eerst het algoritme toe op U^T.
We hebben nu in feite het stelsel $A^T r = d$ opgelost. We hebben dat zo gedaan dat r groot is t.o.v. d, dus heeft r een stevige component in de richting van de links-singuliere vector horend

bij de kleinste singuliere waarde van A. Rechterleden met die eigenschap leiden meestal tot relatief grote oplossingen van het probleem $Az = r$.

Dus vervolgens lossen we op $Az = r$ via

$$\Pi Az = \Pi r \Rightarrow LUz = \Pi r$$

en in twee stappen wordt z verkregen:

$$\text{los } w \text{ op uit } Lw = \Pi r \quad \text{en } z \text{ uit } Uz = w.$$

Er geldt dus dat $z = A^{-1}r \to \|z\|_\infty \leq \|A^{-1}\|_\infty \|r\|_\infty$, en de conditie schatter wordt uiteindelijk:

$$\hat{\kappa}_\infty = \|A\|_\infty \frac{\|z\|_\infty}{\|r\|_\infty}.$$

Deze conditie schatter wordt bijvoorbeeld in de subroutine SGECO uit LINPACK toegepast.

7.6 Numerieke voorbeelden

Voor een geschikt geconstrueerde matrix A zullen we met de FORTRAN subroutine SGECO (uit LINPACK) een LU-decompositie bepalen. Volgens algoritme 13 (met de daarbij gegeven toelichting) wordt een schatting voor het conditie getal bepaald. Schaling van de matrix A zullen we achterwege laten. Tenslotte zullen we de oplossing van een stelsel $Ax = b$, die we verkregen hebben met SGESL, iteratief naverfijnen op de door ons besproken manier.

Ons eerste probleem is geschikte testmatrices A te construeren, zodat we achteraf nog iets kunnen interpreteren en zodat we vooraf het conditie getal ruwweg in de hand hebben. We hebben dat hier als volgt opgelost. Begin met $\tilde{A} = diag(\lambda_1, \ldots, \lambda_n)$ en laat meerdere willekeurige Householder transformaties tweezijdig op $\tilde{A}$ los (waarom?).
We hebben bij het rekenen het volgende FORTRAN programma gebruikt:

```
      program test
      dimension a(100,100),b(100),x(100),aa(100,100),r(100),
     1     ipvt(100),w(100)
c  programma om iteratief naverfijnen te toetsen
      n=100
      small=5.e-7
      do 10 i=1,n
         do 10 j=1,n
            a(i,j)=0.
  10  continue
      do 20 i=1,n
         a(i,i)=1.e0/i
  20  continue
      a(100,100)=small
```

```
      a(90,90)=small
      a(70,70)=small
      a(50,50)=small
      irand=1989
      do 100 kk=1,5
      call rando(n,irand,w)
      beta=2./sdot(n,w,1,w,1)
         do 40 i=1,n
            s=sdot(n,w,1,a(1,i),1)*beta
            do 35 j=1,n
               a(j,i)=a(j,i)-s*w(j)
 35         continue
 40      continue
 100  continue
      do 200 kk=1,5
         call rando(n,irand,w)
         beta=2./sdot(n,w,1,w,1)
         do 140 i=1,n
            s=sdot(n,w,1,a(i,1),n)*beta
            do 135 j=1,n
               a(i,j)=a(i,j)-s*w(j)
 135        continue
 140     continue
 200  continue
c  a(i,j) is nu voor- en achtervermenigvuldigd met een random orthogonale
c  matrix.
      do 50 i=1,n
         r(i)=0.
         x(i)=1.
 50   continue
      do 60 i=1,n
         b(i)=sdsdot(n,a(i,1),n,x(1),1,r(1))
 60   continue
 1002 format('  de eerste rij van A:  ')
      print 1002
      print 1000,(a(1,i),i=1,n)
      lda=n
      call sgeim(a,lda,n,b,x,relerr,aa,r,ipvt,info)
      print 998
 998  format('  de pivot nummers: ')
      print 999,ipvt
 999  format(10('  ',i4))
      print 1001
 1001 format('  de oplossing na het naverfijnen: ')
      print 1000,x
```

```
 1000 format(5('  ',e13.6))
      end

      function sdsdot(n,x,incx,y,incy,c)
      dimension x(incx,1),y(incy,1)
      double precision sum
      sum=0.d0
      if (n.le.0) goto 20
      do 10 i=1,n
         sum=sum+dble(x(1,i))*dble(y(1,i))
 10   continue
 20   sum=sum+dble(c)
      sdsdot=sngl(sum)
      return
      end

     subroutine sgeim(a,lda,n,b,x,relerr,aa,r,ipvt,info)
      integer lda,n,info
      real a(lda,1),b(n),x(n),relerr,aa(lda,1),r(n)
      integer ipvt(n)
      real z(300),rcond
c
c  example illustrating iterative improvement
c  uses sdsdot,sasum,sgefa,sgesl
c
     real xnorm,rnorm,sasum,sdsdot
      do 20 j=1,n
         do 10 i=1,n
            aa(i,j)=a(i,j)
 10      continue
          x(j)=b(j)
 20   continue
      call sgeco(aa,lda,n,ipvt,rcond,z)
      print 1002,rcond
      if (1.0+rcond.eq.1.0) goto 60
 1002 format(' schatting inverse conditie getal: ',e13.6)
      call sgesl(aa,lda,n,ipvt,x,0)
      print 998
      print 999,x
 999  format(5('  ',e13.6))
 998  format(' de oplossing voor het naverfijnen:')
      xnorm=sasum(n,x,1)
      print 1000,xnorm
 1000 format(' de 1-norm van x: ',e13.6)
      if (xnorm.eq.0.0) goto 60
```

```
      do 50 iter=1,20
         do 30 i=1,n
            r(i)=sdsdot(n,a(i,1),lda,x(1),1,-b(i))
 30      continue
       rnorm=sasum(n,r,1)
      print 1007,iter,rnorm
 1007 format('  iteratie nr ',i4,' norm van residu= ',e13.6)
         call sgesl(aa,lda,n,ipvt,r,0)
         do 40 i=1,n
            x(i)=x(i)-r(i)
 40      continue
         rnorm=sasum(n,r,1)
         print 1001,iter,rnorm
 1001    format('  iteratie nr. ',i4,'  norm van correctie= ',e13.6)
         t=xnorm+rnorm
         if (t.eq.xnorm) goto 60
 50   continue
      info =-1
 60   continue
      return
      end

      subroutine rando(n,irand,x)
c  genereert n pseudo random getallen in de vector x
      dimension x(n)
      ihalf=4096
      do 10 i=1,n
         irand=mod(irand*125,8192)
         x(i)=(irand-ihalf)/8192.0
 10   continue
      return
      end
```

Dit programma is op een ALLIANT FX/4 verwerkt in 23-bits single precision arithmetiek, met de volgende resultaten:

De eerste rij van A (om een indruk te krijgen van de grootte van de elementen):

0.811148E+00	0.184460E-02	0.618240E-01	-0.490350E-01	-0.365124E-01
-0.208534E-01	-0.180291E-01	0.135506E-01	0.189307E-01	0.865788E-01
0.262478E-01	0.936603E-02	0.142738E-01	0.330311E-01	0.265444E-01
-0.451030E-01	-0.177659E-01	-0.305330E-01	0.108433E-01	-0.489587E-01
-0.158370E-01	-0.343402E-01	-0.430869E-01	-0.235022E-02	0.604627E-01
0.167107E-02	-0.258459E-01	0.247579E-01	-0.170595E-02	-0.254588E-01
-0.344061E-01	0.317477E-01	-0.194548E-01	0.702694E-01	-0.298854E-01
-0.365133E-01	0.214646E-02	-0.311946E-01	0.521882E-01	0.195483E-01
0.298697E-01	-0.875536E-02	0.578267E-01	-0.249963E-02	-0.204104E-01
-0.633841E-02	-0.398395E-01	-0.131272E-01	0.213588E-01	0.490030E-01
-0.628474E-02	-0.302965E-01	-0.542292E-01	0.371433E-01	0.277947E-01
0.348350E-02	0.719888E-01	-0.903884E-01	0.250103E-01	-0.302957E-01
-0.795218E-02	-0.471695E-01	-0.132278E-01	0.687551E-01	0.332839E-02
0.161940E-01	0.694973E-02	-0.769513E-01	0.629029E-02	-0.143797E-01
0.139973E-01	-0.214325E-02	-0.263053E-01	-0.338156E-01	-0.574105E-01
0.522309E-01	0.394492E-01	0.581042E-01	0.400410E-01	0.238625E-01
-0.255763E-01	0.551162E-01	-0.515859E-02	0.142638E-01	0.574700E-01
-0.356776E-01	0.578779E-01	0.903151E-02	-0.671395E-02	-0.536576E-02
-0.227302E-01	0.453714E-01	0.722093E-01	-0.216243E-02	0.646866E-01
0.144772E-01	0.387646E-02	0.645698E-01	0.940985E-02	0.280795E-01

Schatting voor de inverse van het conditie getal: $0.100284E-06$

De oplossing vóór het naverfijnen:

0.100242E+01	0.999132E+00	0.997681E+00	0.100189E+01	0.999681E+00
0.999944E+00	0.100103E+01	0.997763E+00	0.100063E+01	0.998540E+00
0.999116E+00	0.999615E+00	0.998521E+00	0.998427E+00	0.998293E+00
0.100074E+01	0.999630E+00	0.100147E+01	0.998649E+00	0.100182E+01
0.100142E+01	0.999802E+00	0.100144E+01	0.100054E+01	0.997570E+00
0.100194E+01	0.100228E+01	0.997079E+00	0.999977E+00	0.100090E+01
0.100132E+01	0.999687E+00	0.999184E+00	0.998589E+00	0.999106E+00
0.100297E+01	0.100041E+01	0.100088E+01	0.997937E+00	0.998141E+00
0.100108E+01	0.999894E+00	0.998644E+00	0.999714E+00	0.100045E+01
0.100129E+01	0.999996E+00	0.999199E+00	0.999295E+00	0.102234E+01
0.100185E+01	0.100113E+01	0.100149E+01	0.999629E+00	0.998345E+00
0.100092E+01	0.998017E+00	0.100329E+01	0.998910E+00	0.100184E+01
0.100015E+01	0.100086E+01	0.999899E+00	0.999283E+00	0.100061E+01
0.100050E+01	0.100049E+01	0.100185E+01	0.100156E+01	0.990414E+00

0.999732E+00	0.998946E+00	0.999750E+00	0.100190E+01	0.100085E+01
0.999591E+00	0.100040E+01	0.998123E+00	0.999891E+00	0.998798E+00
0.100095E+01	0.998826E+00	0.100016E+01	0.998895E+00	0.998633E+00
0.100126E+01	0.999911E+00	0.999288E+00	0.999581E+00	0.100360E+01
0.100089E+01	0.998815E+00	0.997195E+00	0.100060E+01	0.999792E+00
0.100087E+01	0.998073E+00	0.997064E+00	0.998786E+00	0.100947E+01

$$\|x\|_1 = 0.100020E + 03$$

Het naverfijningsproces:
iteratie: 1: $\|r\|_1 = 0.263330E - 05$ $\|\Delta x\|_1 = 0.138716E + 00$
iteratie: 2: $\|r\|_1 = 0.295932E - 06$ $\|\Delta x\|_1 = 0.323022E - 03$
iteratie: 3: $\|r\|_1 = 0.380570E - 06$ $\|\Delta x\|_1 = 0.241957E - 05$

De pivot nummers:

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	44
31	32	33	44	48	54	37	38	39	48
41	42	68	63	54	46	47	68	49	68
51	57	53	54	55	56	63	64	59	60
61	62	85	76	65	66	67	76	69	75
71	85	73	74	85	83	77	78	79	80
81	82	98	84	99	86	87	88	89	92
91	94	93	95	97	96	97	98	99	100

Merk op dat er kennelijk een paar keer pivoting heeft plaatsgevonden. Hoe kan het gebeuren dat het rijnummer 97 twee keer voorkomt. Kan uit deze rij de oorspronkelijke volgorde gereconstrueerd worden?

De oplossing na het naverfijnen:

0.100018E+01	0.999996E+00	0.999892E+00	0.100041E+01	0.100028E+01
0.100015E+01	0.999881E+00	0.999863E+00	0.999763E+00	0.999924E+00
0.999871E+00	0.999513E+00	0.999809E+00	0.999892E+00	0.999753E+00
0.100002E+01	0.100000E+01	0.999886E+00	0.100002E+01	0.100008E+01
0.100004E+01	0.999860E+00	0.100033E+01	0.100008E+01	0.999722E+00
0.100009E+01	0.100037E+01	0.999678E+00	0.999707E+00	0.999716E+00
0.100031E+01	0.999965E+00	0.100008E+01	0.999972E+00	0.999907E+00
0.100034E+01	0.100043E+01	0.100015E+01	0.999710E+00	0.100018E+01
0.100005E+01	0.999974E+00	0.999814E+00	0.999966E+00	0.100012E+01
0.100039E+01	0.999635E+00	0.999843E+00	0.999879E+00	0.100115E+01
0.100036E+01	0.100013E+01	0.999965E+00	0.100020E+01	0.100016E+01

0.100040E+01	0.100001E+01	0.100014E+01	0.100033E+01	0.100000E+01
0.100013E+01	0.100023E+01	0.999539E+00	0.999874E+00	0.999845E+00
0.999753E+00	0.100024E+01	0.100021E+01	0.100033E+01	0.997251E+00
0.100009E+01	0.100025E+01	0.100002E+01	0.100025E+01	0.100024E+01
0.999741E+00	0.999863E+00	0.100025E+01	0.999942E+00	0.999753E+00
0.100033E+01	0.100000E+01	0.999716E+00	0.100033E+01	0.100023E+01
0.100012E+01	0.999860E+00	0.999728E+00	0.999955E+00	0.100493E+01
0.100008E+01	0.999799E+00	0.999889E+00	0.100008E+01	0.999846E+00
0.100035E+01	0.999652E+00	0.999751E+00	0.999601E+00	0.100161E+01

Enkele kanttekeningen bij de voorgaande resultaten:

- Voor de betreffende computer (ALLIANT FX/4) geldt:
  $\beta = 2$, $t = 23$, en er wordt afgerond: $u = 2^{-23} \approx 1.2 \cdot 10^{-7}$.
- Het geschatte conditiegetal is $\kappa_\infty \approx 10^7$.
  Op grond van stelling 5 mogen we dus verwachten dat de oplossing geen correcte decimalen meer bevat. Dat is een duidelijk te pessimistische voorstelling voor de werkelijke gang van zaken. Maar bedenk wel dat het conditiegetal bepaald wordt door $\epsilon = 10^{-7}$, en als de oplossing een flinke component had gehad in de bijbehorende eigenvector richting, dan was de relatieve fout in de oplossing vermoedelijk ook navenant groter geweest.
- Als we de oplossing na het naverfijnen vergelijken met die van voor het naverfijnen, dan krijgen we een indruk van de mate waarin het Gauss eliminatie proces de "intrinsieke" fout verergert. Zoals we zien valt dat nogal mee,

Opgave 12 *Gegeven is de matrix*

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & \dots & 1 & 1 & 1 & 1 & 2n \\ & 1 & -1 & -1 & \dots & -1 & -1 & -1 & 1 & n \\ & & 1 & -1 & \dots & -1 & -1 & -1 & 1 & n \\ & & & 1 & \dots & -1 & -1 & -1 & 1 & n \\ & & & & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ & & & & & 1 & -1 & -1 & 1 & n \\ & & & & & & 1 & -1 & 1 & n \\ & & & & & & & 1 & 1 & n \\ & & & & & & & & 1 & n \\ & & & & & & & & & 1 \end{pmatrix}.$$

a. Pas algoritme 13 tot op de matrix A^T met als weegfactoren $w_i = \frac{1}{|\ell_{i,i}|}$ en zie in dat $r = (1, 0, 0, \dots, 0, 1-2n)^T$. Bereken z en $\|z\|_\infty / \|r\|_\infty$.
b. Neem nu $\hat{r} = (1, 1, \dots, 1, 1, 0, 0)^T$ en bereken $\hat{z}$ uit $A\hat{z} = \hat{r}$, alsmede $\|\hat{z}\|_\infty / \|\hat{r}\|_\infty$. Welke conclusie kunt U hieruit trekken?

Hoofdstuk 8

Speciale lineaire stelsels; Choleski ontbinding

8.1 Choleski ontbinding

Alvorens we het belangrijke geval $A = A^T$ gaan bekijken, geven we eerst nog een kleine variant op de LU ontbinding. In de klassieke LU ontbinding heeft L diagonaalelementen 1. Bij de zogenaamde LDM^T ontbinding heeft zowel L als M diagonaalelementen 1. De constructie van zo'n ontbinding, over de existentie waarvan stelling 8 een uitspraak doet, is overigens vrij voor de hand liggend.

Stelling 8 *Als alle leidende hoofdondermatrices van $A \in R^{n\times n}$ niet-singulier zijn, dan bestaan er benedendriehoeksmatrices L en M met $diag(L) = diag(M) = I$ en een diagonaalmatrix $D = diag(d_1, \ldots, d_n)$ zo, dat $A = LDM^T$.*

Bewijs: Volgens stelling 3 bestaan er een L en een U zodat $A = LU$.
Wegens

$$det(A) = \prod_{i=1}^{n} u_{i,i} \neq 0,$$

zijn de $u_{i,i} \neq 0$.
Kies nu $d_i = u_{i,i}$, voor $i = 1, \ldots, n$ en definiëer $M^T = D^{-1}U$. $\square$

Als de niet-singuliere matrix A symmetrisch is en bovendien een LDM^T ontbinding heeft, dan geldt $M = L$. Zij namelijk $A = LDM^T$, dan geldt

$$M^{-1}AM^{-T} = M^{-1}LD.$$

De matrix $M^{-1}AM^{-T}$ is symmetrisch en dus moet de benedendriehoeksmatrix $M^{-1}LD$ ook symmetrisch zijn (ga na dat $M^{-1}L$ een benedendriehoeksmatrix is). Dus is $M^{-1}LD$, en daarmee ook $M^{-1}L$, een diagonaal matrix. Omdat $diag(M^{-1}L) = I$ geldt daardoor dat $M = L$.

Opgave 13 *Laat zien dat als de niet-singuliere A een LDM^T ontbinding heeft, deze uniek is.*

Opgave 14 *Zij* $A = LU$ *de ontbinding van* $A \in R^{n\times n}$*. Zij verder* $L = (\ell_{i,j})$, $\ell_{i,i} = 1$ *en* $U = (u_{i,j})$*. Leid een algoritme af voor de berekening van* $\ell_{i,j}$ *en* $u_{i,j}$*, door het produkt van* L *en* U *te vergelijken met* A*. Kan* A *tijdens het decompositie proces weer overschreven worden door de factoren* L *en* U*?*

Een speciale en zeer relevante deelklasse der symmetrische matrices is die der positief definiet symmetrische matrices. Een belangrijke eigenschap is vervat in de volgende opgave.

Opgave 15 *Zij* A *symmetrisch positief definiet. Laat zien dat alle hoofdondermatrices* A_k *eveneens symmetrisch positief definiet zijn.*

Als A symmetrisch positief definiet is, dan bestaat kennelijk de LDL^T decompositie van A. Vaak wordt D "over L en L^T" verspreid, hetgeen bekend staat als de *Choleski ontbinding.* De volgende stelling vat dit wat preciezer samen.

Stelling 9 *Als* $A \in R^{n\times n}$ *symmetrisch positief definiet is, dan bestaat er een benedendriehoeksmatrix* $\tilde{L}$ *met posititieve diagonaal elementen, zodanig dat* $A = \tilde{L}\tilde{L}^T$*.*

Bewijs: Alle leidende hoofdondermatrices van A zijn nietsingulier (zie opgave 15). Volgens stelling 8 bestaat er een LDM^T ontbinding van A en wegens $A = A^T$ geldt $M = L$ (zie ook de toelichting bij stelling 8).
Wegens $\det(A_k) = d_k \cdot \det(A_{k-1})$ (ga na!) geldt $d_k > 0$. De matrix $\tilde{L} \equiv LD^{\frac{1}{2}}$ met $D^{\frac{1}{2}} = diag(\sqrt{d_1}, \ldots, \sqrt{d_n})$ is de gezochte factor in $A = \tilde{L}\tilde{L}^T$. □

De ontbinding $A = \tilde{L}\tilde{L}^T$ heet de *Choleski decompositie* en $\tilde{L}$ heet de *Choleski factor.*
De elementen van $\tilde{L}$ kunnen berekend worden door $\tilde{L}\tilde{L}^T$ rechtstreeks met A te vergelijken. Dit leidt dan tot bijvoorbeeld het volgende algoritme, waarbij successievelijk de kolommen van $\tilde{L}^T$ berekend worden.

Algoritme 14 *Dit is de kolomversie van de Choleski ontbinding. We nemen aan dat* $A \in R^{n\times n}$ *symmetrisch positief definiet is. Het algoritme berekent de factor* $\tilde{L}$ *waarvoor geldt* $A = \tilde{L}\tilde{L}^T$ *en hierbij worden de elementen* $a_{i,j}$ *overschreven door de* $\tilde{\ell}_{i,j}$ *(voor* $i \geq j$*).*

Voor $k = 1, \ldots, n$
  $a_{k,k} = (a_{k,k} - \sum_{p=1}^{k-1} a_{k,p}^2)^{\frac{1}{2}}$
  Voor $i = k+1, \ldots, n$
    $a_{i,k} = (a_{i,k} - \sum_{p=1}^{k-1} a_{i,p}a_{k,p})/a_{k,k}$

Het bovenstaande algoritme kost ruwweg $n^3/3$ flops (ga na!).
De elementen van $\tilde{L}$ (en dus ook die van $\tilde{L}^T$) zijn eenvoudig naar boven af te schatten:

$$\tilde{\ell}_{i,j}^2 \leq \sum_{p=1}^{i} \tilde{\ell}_{i,p}^2 = a_{i,i}.$$

We kunnen nu stelling 7 toepassen en daarmee inzien dat de Choleski ontbinding (zonder enige vorm van pivoting!) gebruikt kan worden voor het oplossen van $Ax = b$. Daarbij kan nog bedacht worden dat $\|A\|_2 = \|\tilde{L}\|_2^2$, nl. $\|\tilde{L}\|_2^2 = \max \lambda(\tilde{L}\tilde{L}^T) = \|A\|_2$.

Zij $\hat{L}$ de berekende factor van $A = \hat{L}\hat{L}^T$, dan kan $Ax = b$ als volgt worden opgelost: $\hat{L}y = b$ en $\hat{L}^T x = y$. Zij vervolgens $\hat{x}$ de berekende oplossing, dan geldt volgens Wilkinson [26]:

$$(A + E)\hat{x} = b, \quad \text{met} \quad \|E\|_2 \leq c_n u \|A\|_2,$$

waarbij c_n een bescheiden getal is dat alleen van n afhangt. Verder heeft Wilkinson aangetoond dat als $q_n u \|A\|_2 \|A^{-1}\|_2 \leq 1$, met q_n niet veel groter dan 1, het Choleski algoritme 14 niet afbreekt doordat er een wortel uit een niet-positief getal getrokken zou moeten worden.

Opgave 16 *Toon aan dat de Choleski ontbinding van een symmetrisch positief definiete A uniek is.*

8.2 Bandmatrices

Bij veel toepassingen treden matrices op met een bandstructuur. In die gevallen kan veel rekenwerk en geheugenruimte bespaard worden omdat de factoren in de ontbindingen LU, LDM^T en $\tilde{L}\tilde{L}^T$ eveneens een bandstructuur hebben. De volgende stelling doet daar een uitspraak over.

Stelling 10 *Zij A een bandmatrix met bovenbandbreeedte q en onderbandbreedte p. Als alle leidende hoofdondermatrices van A nietsingulier zijn, dan geldt $A = LU$ (met $A, L, U \in R^{n \times n}$, waarbij L onderbandbreedte p en U bovenbandbreedte q heeft.*

Bewijs: Men gaat eenvoudig na dat M_1 benedenbandbreedte p heeft en de matrix $M_1 A$ bovenbandbreedte q. Met een inductie argument volgt dan het te bewijzene. □

Als er partial pivoting nodig is dan komen de zaken wat gecompliceerder te liggen. Men kan zelf nagaan dat in dat geval U hoogstens bandbreedte $p + q$ heeft en van L kan men zeggen dat er hooguit $p + 1$ elementen ongelijk aan 0 per kolom voorkomen.
Bij de methode van Choleski, voor symmetrische positief definiete bandmatrices, hoeft men niet te pivotten, zodat de Choleski factoren dezelfde bandbreedte hebben als A. Bij Choleski moet men wel n keer een wortel trekken. Voor kleine bandbreedte gaat de rekentijd die daarvoor nodig is een rol spelen. Men gebruikt dan vaak de zogenaamde *root-free* Choleski ontbinding $A = LDL^T$.
We laten dit zien voor een tridiagonale matrix (een veel voorkomende situatie bij bijv. gediscretiseerde partiële differentiaalvergelijkingen). Met

$$L = \begin{pmatrix} 1 & & & & \\ e_1 & 1 & & & \\ & e_2 & 1 & & \\ & & \ddots & \ddots & \\ & & & e_{n-1} & 1 \end{pmatrix} \quad \text{en} \quad D = \begin{pmatrix} d_1 & & & & \\ & d_2 & & & \\ & & d_3 & & \\ & & & \ddots & \\ & & & & d_n \end{pmatrix},$$

volgt uit de gelijkstelling van LDL^T met A:

$$d_1 = a_{1,1}$$

$$e_{k-1} d_{k-1} = a_{k,k-1}$$

$$d_k + e_{k-1}^2 d_{k-1} = d_k + e_{k-1} a_{k,k-1} = a_{k,k}$$

$$\text{voor} \quad k = 2, \ldots, n.$$

De stelsels $Ly = b$, $Dz = y$ en $L^T x = z$ kunnen elk met resp. $2n$, n en $2n$ flops worden opgelost.

Opgave 17 *Zij A een symmetrische positief definiete tridiagonale matrix, waarvan de diagonaalelementen zijn opgeslagen in $d_1, d_2, \ldots, d_n$ en de subdiagonaalelementen in $e_1, e_2, \ldots, e_{n-1}$. Schrijf een algoritme dat de LDL^T ontbinding van A berekent, waarbij de arrays d en e worden overschreven door de elementen van de factoren, en dat vervolgens $Ax = b$ oplost, waarbij b overschreven wordt door x.*

Opgave 18 *Zij A een symmetrisch positief definiete matrix. De getallen $f_i(A)$, voor $i = 1, \ldots, n$, worden gedefinieerd door $f_i(A) \equiv \min\{j | a_{i,j} \neq 0\}$. Ga na dat voor de Choleski factoren in $A = LL^T$ geldt $f_i(A) = f_i(L)$, voor $i = 1, \ldots, n$.*

8.2.1 Blok tridiagonale matrices

Deze klasse van matrices treedt veelvuldig op, onder andere bij het discretiseren van elliptische en parabolische partiële differentiaal vergelijkingen over rechthoekige roosters. De bijbehorende stelsels hebben schematisch de volgende vorm:

$$Ax = \begin{pmatrix} D_1 & F_1 & & & \\ E_2 & D_2 & F_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & F_{n-1} \\ & & & E_n & D_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{pmatrix},$$

waarbij $D_i, E_i, F_i \in R^{q \times q}$ en $x_i, b_i \in R^q$.

In voorkomende gevallen hebben de blokken zelf ook weer een speciale structuur, bijvoorbeeld E_i en F_i kunnen diagonaalmatrices zijn en de D_i tridiagonaal matrices. Als bijvoorbeeld de Poisson vergelijking wordt gediscretiseerd met 5-punts eindige differentie discretisatie over een rechthoek met een equidistant rooster, dan leidt dat tot

$$D_i = \begin{pmatrix} 4 & -1 & & & & \\ -1 & 4 & -1 & & & \\ & -1 & 4 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & \ddots & \ddots & -1 \\ & & & & -1 & 4 \end{pmatrix}, E_i = F_i = \begin{pmatrix} -1 & & & & & \\ & -1 & & & & \\ & & -1 & & & \\ & & & \ddots & & \\ & & & & \ddots & \\ & & & & & -1 \end{pmatrix}.$$

De waarde van q is in dat geval gelijk aan het aantal roosterpunten op een lijn van het rooster (waarop de oplossing onbekend is).

Een methode om dit soort speciale bloktridiagonale stelsels op te lossen is *block cyclic reduction*. We beschrijven deze methode en nemen daarbij ter vereenvoudiging aan dat $D_i = D$ en $E_i =$

$F_i = F$. Bovendien nemen we aan dat F en D commuteren: $FD = DF$. Tenslotte nemen we aan dat het aantal blokken langs de diagonaal gelijk is aan $n = 2^k - 1$. De orde van A is dus $q \cdot (2^k - 1)$.
We werken het idee uit voor het geval $k = 3$ (voor verdere details zie [2]):

$$
\begin{array}{ccccccccc}
Dx_1+ & Fx_2 & & & & & & = b_1 \\
Fx_1+ & Dx_2+ & Fx_3 & & & & & = b_2 \\
 & Fx_2+ & Dx_3+ & Fx_4 & & & & = b_3 \\
 & & Fx_3+ & Dx_4+ & Fx_5 & & & = b_4 \\
 & & & Fx_4+ & Dx_5+ & Fx_6 & & = b_5 \\
 & & & & Fx_5+ & Dx_6+ & Fx_7 & = b_6 \\
 & & & & & Fx_6+ & Dx_7 & = b_7
\end{array}
$$

Voor $i = 2, 4, 6$ vermenigvuldigen we de vergelijking $i - 1$ met F, vergelijking i met $-D$ en $i + 1$ met F en tellen de 3 resulterende vergelijkingen bij elkaar op. Dat leidt tot

$$(2F^2 - D^2)x_2 + F^2 x_4 = F(b_1 + b_3) - Db_2$$

$$F^2 x_2 + (2F^2 - D^2)x_4 + F^2 x_6 = F(b_3 + b_5) - Db_4$$

$$F^2 x_4 + (2F^2 - D^2)x_6 = F(b_5 + b_7) - Db_6$$

Merk op dat dit weer een bloktridiagonaal stelsel is, echter ditmaal van orde 3. Met $D^{(1)} \equiv 2F^2 - D^2$ en $F^{(1)} \equiv F^2$, $b_i^{(1)} \equiv F(b_{i-1} + b_{i+1}) - Db_i$, voor $i = 2, 4, 6$, kunnen we de voorgaande reductie truc weer uithalen, waarna we overhouden:

$$(2F^{(1)^2} - D^{(1)^2})x_4 = F^{(1)}(b_2^{(1)} + b_6^{(1)}) - D^{(1)}b_4^{(1)}.$$

Hieruit kan met weinig moeite (zie de voorgaande paragraaf) x_4 worden opgelost. Uit het 3 bij 3 bloktridiagonale stelsel kunnen dan x_2 en x_6 worden opgelost en de overige x_i's kunnen worden opgelost uit de oneven vergelijkingen van het oorspronkelijke stelsel.
Het zal duidelijk zijn dat het succes van deze methode grotendeels afhangt van de mate waarin de blokken $D^{(i)}$ en $F^{(i)}$ tijdens het proces vollopen. In ons modelprobleem blijven de $F^{(i)}$ diagonaal matrices, de blokken $D^{(i)}$ lopen echter steeds meer vol. Door nu voor, bijvoorbeeld, de gediscretiseerde Poisson vergelijking gebruik te maken van de bekende eigenvectoren en eigenwaarden van D kunnen ook de blokken $D^{(i)}$ naar diagonaal gedaante getransformeerd worden. Op dit idee is de zogenaamde *Fast Poisson Solver* [2] gebaseerd.

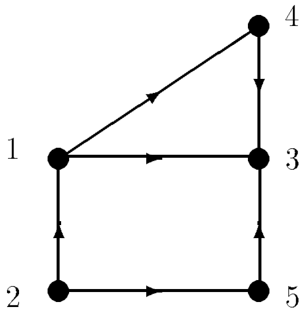
8.3 Directe methoden voor ijle matrices

De hier gepresenteerde aanpak is grotendeels gebaseerd op werk van Reid [18].
Alhoewel voor ijle matrices vaak gebruik gemaakt wordt van iteratieve methoden voor het oplossen van stelsels $Ax = b$ (waarover later nog uitvoeriger), kunnen directe methoden, gebaseerd op Gauss-eliminatie, ook zeer efficiënt zijn. Ijle matrices komen ondere andere voort uit de netwerktheorie. Onze aanpak is dan ook gebouwd op ideeën en begrippen uit die theorie.
In een netwerk zijn de knooppunten veelal slechts met weinig andere knooppunten verbonden en dat geeft aanleiding tot de ijlheid van de matrix die het verband tussen onbekenden in het

netwerk representeert. Ook voor anderssoortige problemen is het bij de eruit voortvloeiende matrices vaak nuttig ze afkomstig te denken van een netwerk of *graph*. In een graph worden de onbekenden waar de matrix op werkt, vertegenwoordigd door *knooppunten*. Als $a_{i,j} \neq 0$ dan krijgt knooppunt i een verbinding naar knooppunt j, weergegeven door een pijl (in de richting van knooppunt j).
Als A symmetrisch is, dan is met $a_{i,j} \neq 0$ ook $a_{j,i} \neq 0$ en dan worden de pijlpunten weggelaten. Een graph met pijlpunten (dus voor nietsymmetrische matrices) heet ook wel een *digraph*.

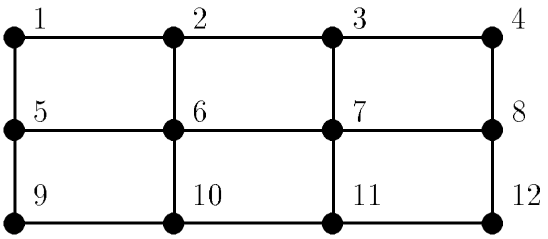
Voorbeelden.

$$\text{matrix:} \quad \begin{pmatrix} * & & * & * & \\ * & * & & & * \\ & & * & & \\ & & * & * & \\ & & * & & * \end{pmatrix} \qquad \text{digraph:}$$



$$\text{matrix:} \quad \begin{pmatrix} * & * & & & * & & & & & & & \\ * & * & * & & & * & & & & & & \\ & * & * & * & & & * & & & & & \\ & & * & * & & & & * & & & & \\ * & & & & * & * & & & * & & & \\ & * & & & * & * & * & & & * & & \\ & & * & & & * & * & * & & & * & \\ & & & * & & & * & * & & & & * \\ & & & & * & & & & * & * & & \\ & & & & & * & & & * & * & * & \\ & & & & & & * & & & * & * & * \\ & & & & & & & * & & & * & * \end{pmatrix}$$

De bijbehorende digraph is:



Directe methoden, voor het oplossen van lineaire stelsels, leiden meestal tot fill-in in de matrix, en dat wenst men zoveel mogelijk te beperken. In het algemeen kan men zeggen dat er minder fill-in plaatsvindt in gevallen waarin de graph door weglating van slechts een paar knooppunten uiteen valt in disjuncte onderdelen.

8.4 Het vermijden van fill-in

Men gebruikt partial pivoting doorgaans om de elementen van $|L|$ te begrenzen. Een andere reden voor pivoting zou kunnen zijn de hoeveelheid fill-in te vermijden:

$$\begin{pmatrix} x & x & x & x & x & x \\ x & x & & & & \\ x & & x & & & \\ x & & & x & & \\ x & & & & x & \\ x & & & & & x \end{pmatrix} \quad \text{versus} \quad \begin{pmatrix} x & & & & & x \\ & x & & & & x \\ & & x & & & x \\ & & & x & & x \\ & & & & x & x \\ x & x & x & x & x & x \end{pmatrix}.$$

Een prettig geval doet zich voor als A blokbenedendriehoeks is:

$$A = \begin{pmatrix} A_{1,1} & & & \\ A_{2,1} & A_{2,2} & & \\ \vdots & \vdots & \ddots & \\ A_{N,1} & A_{N,2} & \cdots & A_{N,N} \end{pmatrix},$$

waarbij de diagonaalblokken $A_{i,i}$ vierkante matrices zijn.
De vergelijking $Ax = b$ valt nu uiteen in de vergelijkingen

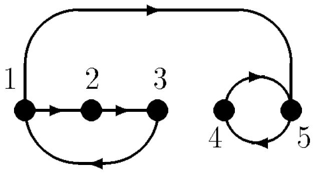
$$A_{i,i}x_i = b_i - \sum_{j=1}^{i-1} A_{i,j}x_j. \tag{8.1}$$

Hierin zijn de x_i en b_i vector segmenten overeenkomend met de afmetingen van de $A_{i,i}$. Het probleem is nu gereduceerd tot het oplossen van kleinere stelsels en we behoeven nog slechts bezorgd te zijn voor de fill-in die ontstaat in de $A_{i,i}$.

We geven in het kort weer hoe de blokbenedendriehoeksvorm gevonden kan worden. De eerste stap bestaat daaruit dat men door rij en/of kolom verwisselingen er zorg voor draagt dat alle diagonaal elementen $a_{i,i}$ van A ongelijk aan nul zijn. Als A nietsingulier is dan is dat altijd mogelijk (ga na). Als dat gebeurd is dan bekijken we de digraph van A. Als we geen gesloten pad kunnen vinden dat, de pijlen volgend, door alle knooppunten van de digraph gaat en weer op het uitgangspunt terug komt, dan kunnen we de digraph verdelen in twee delen en wel zodanig dat er geen verbinding is vanuit het eerste deel naar het tweede deel toe. De knooppunten worden nu zo hernummerd, dat de k punten in het eerste deel van de graph corresponderen met de eerste k onbekenden, dat wil zeggen we geven ze rangnummer 1 tot en met k.
Hetzelfde proces kan herhaald worden op het resulterende tweede deel. De digraph wordt door deze manier van werken opgebroken in zogenaamde *strong components*. Een strong component bestaat uit knooppunten die allemaal door een gesloten pad verbonden zijn, maar verliest deze eigenschap als er een willekeurig ander knooppunt aan wordt toegevoegd.

Voorbeeld 1:

$$\begin{pmatrix} * & * & & & * \\ & * & * & & \\ * & & * & & \\ & & & * & * \\ & & & * & * \end{pmatrix}$$

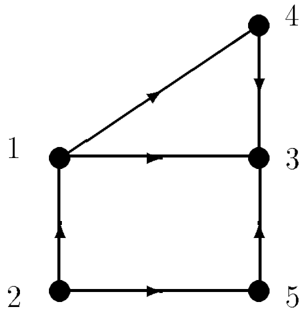


De set $(1,2,3)$ heeft een verbinding naar $(4,5)$; echter van $(4,5)$ kunnen we niet meer terug. We hernummeren nu $4 \to 1$, $5 \to 2$, $1 \to 3$,$2 \to 4$ en $3 \to 5$.
De eerste strong component wordt dan gevormd door de set $(1,2)$. De tweede component vormt ook een gesloten pad en is zelf dus ook een strong component. De hernummering levert de volgende matrix op:

$$\begin{pmatrix} * & * & & & \\ * & * & & & \\ & * & * & * & \\ & & & * & * \\ & & * & & * \end{pmatrix}.$$

Voorbeeld 2:

$$\begin{pmatrix} * & & * & * & \\ * & * & & & * \\ & & * & & \\ & & * & * & \\ & & * & & * \end{pmatrix}$$



Knooppunt 1 is verbonden met 3, maar 3 is verder nergens naar door verbonden. Knooppunt 3 wordt dus ons eerste knooppunt. In de resterende digraph is 1 verbonden met 4 en daar houdt het nu weer op: 4 is ons nieuwe tweede knooppunt. Vervolgens wordt 1 ons derde nieuwe knooppunt, 5 het vierde en tenslotte 2 het vijfde. Deze hernummering leidt tot de volgende matrix:

$$\begin{pmatrix} * & & & & \\ * & * & & & \\ * & * & * & & \\ * & & & * & \\ & & * & * & * \end{pmatrix},$$

en deze matrix staat zelfs al geheel op benedendriehoeksgedaante.
Algoritmen om de (blok) benedendriehoeksvorm te vinden, zijn beschreven in [20] en [24].

We nemen nu verder aan dat A niet te herleiden is tot blokbenedendriehoeks vorm (anders bekijken we gewoon de blokken $A_{i,i}$). Na k slagen Gauss eliminatie ziet het onderblok van A er als volgt uit:

$$\begin{pmatrix} a_{k,k}^{(k)} & a_{k,k+1}^{(k)} & \dots & a_{k,n}^{(k)} \\ a_{k+1,k}^{(k)} & \dots & & \\ \vdots & & & \vdots \\ a_{n,k}^{(k)} & \dots & & a_{n,n}^{(k)} \end{pmatrix}.$$

De elementen van $A^{(k+1)}$ ontstaan uit de berekening

$$a_{i,j}^{(k+1)} = a_{i,j}^{(k)} - \frac{a_{i,k}^{(k)}}{a_{k,k}^{(k)}} a_{k,j}^{(k)}, \quad k < i, j \leq n.$$

Bij partial pivoting zorgen we er voor dat $|a_{k,k}^{(k)}| \geq \max_{k \leq i \leq n} |a_{i,k}^{(k)}|$. Dat geeft ons echter te weinig vrijheid om te proberen fill-in te vermijden. We vervangen het criterium daarom door de zwakkere eis dat voor het leidende diagonaal element (de pivot) moet gelden:

$$|a_{k,k}^{(k)}| \geq p \max_{k \leq i \leq n} |a_{i,k}^{(k)}| \ , \ \text{ met } 0 < p \leq 1.$$

We vinden dat

$$|a_{i,j}^{(k+1)}| \leq (1 + \frac{1}{p}) \max_{k \leq i \leq n} |a_{i,j}^{(k)}|.$$

Merk op dat $a_{i,j}^{(k)}$ slechts verandert als de pivotrij op de j-de plaats een element ongelijk aan nul bevat. Als p_j het aantal niet-nul elementen gedurende het gehele proces in de j-de rij van de benedendriehoek weergeeft, dan is de groeifactor dus maximaal $(1 + \frac{1}{p})^{p_j}$.
Bij sommige toepassingen (bijv. lineaire programmering) is p_j erg klein, zodat men zich een kleine waarde voor p kan permitteren. Dat wil zeggen men kan dat benutten door een dusdanige pivot te kiezen dat de fill-in meevalt. Duff en Reid [7] vonden experimenteel dat de keuze $p = \frac{1}{4}$ goed voldoet. Markowitz [15] bedacht evenwel het volgende eenvoudige criterium om een geschikt pivotelement te bepalen.
Zoek in $A^{(k)}$ het niet-nul element waarvoor het produkt van n_k en n_r minimaal is. Hierbij is:
n_k = het aantal overige niet-nul elementen in de kolom van het betreffende element,
n_r = het aantal overige niet-nul elementen in de rij van het betreffende element.
Duff en Reid probeerden om telkens het niet-nul element te vinden waarvoor de totale hoeveelheid fill-in in $A^{(k+1)}$ minimaal is. In beide strategieën wordt het gevonden niet-nul element als pivot gekozen. De "Duff en Reid" strategie is relatief duur en beperkt bovendien lang niet altijd wezenlijk de fill-in in de daaropvolgende slagen, zoals blijkt uit een door Reid [18] gepubliceerde tabel (bepaald met $p = \frac{1}{4}$):

orde van de matrix	54	199
aantal niet-nullen in A	291	701
aantal niet-nullen in L en U via Markowitz	381	1387
idem via Duff en Reid	361	1372
CPU-tijd voor "analyse" Markowitz	140 msec	900
idem voor Duff en Reid	800 msec	6400

Voor details over implementatie en behandeling van speciale gevallen, zie [18].

Hoofdstuk 9

De QR-methode en kleinste kwadraten problemen

9.1 De normaalvergelijkingen en QR

In dit hoofdstuk wijden wij onze aandacht aan het bepalen van de kleinste kwadraten oplossing van het overbepaalde stelsel $Ax = b$, met $A \in R^{m \times n}$, waarbij $m > n$. De kleinste kwadraten oplossing is die x waarvoor $\|Ax - b\|_2$ minimaal is. Er zijn verschillende mogelijkheden om dit probleem aan te pakken: we noemen slechts de QR-methode, de SVD-methode en het oplossen van de zogenaamde normaalvergelijkingen. Software voor QR en SVD is in de standaardprogrammatheken als LINPACK, NAG en IMSL, alsmede in het MATHLAB pakket, beschikbaar.

Als een stelsel meer vergelijkingen heeft dan onbekenden, dan noemen we het stelsel *overbepaald*. In de meeste gevallen heeft zo'n stelsel geen exacte oplossing, immers dan zou het rechterlid b precies door de kolommen van A opgespannen worden. Men probeert daarom een zodanige x te vinden dat $\|Ax - b\|_p$, voor geschikt gekozen p-norm, minimaal is. Verschillende normen kunnen daarbij tot verschillende oplossingen x leiden.

Voorbeeld: Zij $A = (1, 1, 1)^T$, $b = (b_1, b_2, b_3)^T$ met $b_1 \geq b_2 \geq b_3 \geq 0$.
dan gelden de volgende oplossingen

$p = 1 \quad \Rightarrow \quad x = b_2$
$p = 2 \quad \Rightarrow \quad x = (b_1 + b_2 + b_3)/3$
$p = \infty \quad \Rightarrow \quad x = (b_1 + b_3)/2$

Minimalisatie in de 1-norm en de ∞-norm is gecompliceerd, omdat het residu in de bijbehorende normen niet differentieerbaar is. In de 2-norm is het residu wel differentieerbaar en deze norm heeft bovendien het voordeel dat zij invariant is onder orthogonale transformaties, zodat we eerst eens kunnen proberen het gegeven stelsel op een wat beter hanteerbare gedaante te brengen. Met orthogonale Q^T geldt:

$$\|Ax - b\|_2 = \|Q^T Ax - Q^T b\|_2 \quad \text{met} \quad Q^T Q = I_m.$$

Men kan Q^T zo bepalen dat $Q^T A$ bovendriehoeksvorm heeft. We komen hier nog op terug.
We definiëren eerst de verzameling X van oplossingen x.

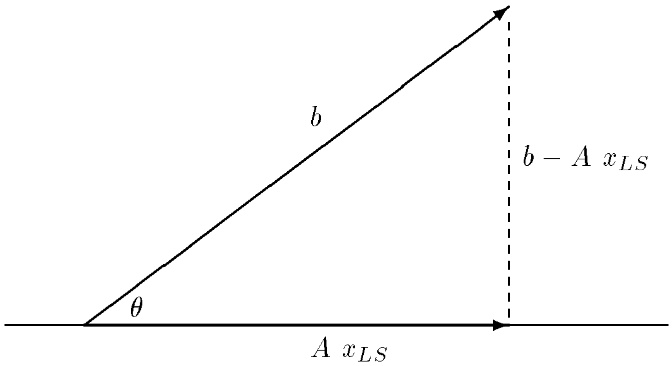
Definitie 4 *De verzameling X van kleinste kwadraten oplossingen van het stelsel $Ax = b$ wordt gedefinieerd als*

$$X \equiv \{x \in R^n | \ \|Ax - b\|_2 = \inf_{\xi \in R^n} \|A\xi - b\|_2\}.$$

Opmerkingen:

- $X \neq \{\emptyset\}$
- Uit het feit dat $\|Ax - b\|_2$ minimaal is, volgt dat $Ax - b \perp R(A)$.
  Dus $Ax - b \perp y$, voor $y \in R(A)$
  $\Rightarrow (Ax - b, Az) = 0$, voor alle $z \in R^n$
  $\Rightarrow (A^T(Ax - b), z) = 0$, voor alle $z \in R^n$,
  en omdat $A^T(Ax - b) \in R^n$ geldt dat x voldoet aan $A^TAx = A^Tb$.
- X bevat slechts één element x_{LS} d.e.s.d. $rang(A) = n$.
- X bevat een uniek element x_{LS} met minimale 2-norm.

De vergelijkingen $A^TAx = A^Tb$ heten de *normaalvergelijkingen*. De uniek gedefinieerde oplossing noteren we met x_{LS} en $\rho_{LS} \equiv \|Ax_{LS} - b\|_2$.



We schrijven $x_{LS} = A^+b$, A^+ heet de *pseudoinverse* van A en $\kappa_2(A) = \|A\|_2\|A^+\|_2$.
Als $rang(A) = n$ dan geldt $A^+ = (A^TA)^{-1}A^T$; als $m = n = rang(A)$ dan geldt $A^+ = A^{-1}$.

De volgende stelling zegt iets over de gevoeligheid van x_{LS} ten aanzien van afrondfouten.

Stelling 11 *Veronderstel dat x, r, $\hat{x}$ en $\hat{r}$ voldoen aan*

$\|Ax - b\|_2 = minimaal$	$r = b - Ax$
$\|(A + \partial A)\hat{x} - (b + \partial b)\|_2 = minimaal$	$\hat{r} = b + \partial b - (A + \partial A)\hat{x}$

met A, $\partial A \in R^{m\times n}$, $m \geq n$ en $rang(A) = n$.
Zij verder

$$\epsilon \equiv \max\{\frac{\|\partial A\|_2}{\|A\|_2}, \frac{\|\partial b\|_2}{\|b\|_2}\} < \sqrt{\frac{\lambda_n(A^TA)}{\lambda_1(A^TA)}},$$

waarbij λ_n, λ_1, de kleinste, resp. grootste, eigenwaarde van A^TA is, en verder

$$\sin\theta \equiv \frac{\rho_{LS}}{\|b\|_2} \neq 1, \quad \kappa_2(A) = \|A\|_2\|A^+\|_2.$$

Dan geldt

$$\frac{\|\hat{x}-x\|_2}{\|x\|_2} \leq \epsilon\{\frac{2\kappa_2(A)}{\cos\theta} + \tan\theta\kappa_2^2(A)\} + \mathcal{O}(\epsilon^2) \quad \textit{en}$$

$$\frac{\|\hat{r}-r\|_2}{\|b\|_2} \leq \epsilon(1+2\kappa_2(A))\min(1,m-n) + \mathcal{O}(\epsilon^2).$$

Voor het bewijs verwijzen we naar [10].

Een populaire, maar niet altijd even gunstige, aanpak om overbepaalde stelsels op te lossen loopt via het opstellen van de normaalvergelijkingen. Als A rang n heeft dan is de matrix A^TA symmetrisch positief definiet en dus kan men de normaalvergelijkingen oplossen met de methode van Choleski, hetgeen de oplossing x_{LS} oplevert.
Als we aannemen dat A^TA en A^Tb zonder afrondfouten berekend worden (door bijvoorbeeld de gewenste inprodukten in dubbele precisie te berekenen bereikt men dat de afrondfouten relatief kleiner of gelijk aan u zijn) dan volgt uit de foutanalyse voor het Choleski algoritme (zie paragraaf 8.1):

$$(A^TA+E)\hat{x} = A^Tb,$$

met $\hat{x}$ de berekende oplossing en $\|E\|_2 \leq c_n u\|A^TA\|_2$ met c_n een bescheiden constante.
Voor $\hat{x} - x_{LS}$ geldt dan

$$\frac{\|\hat{x}-x_{LS}\|_2}{\|x_{LS}\|_2} \approx c_n u\kappa_2(A^TA) = c_n u\kappa_2^2(A).$$

De nauwkeurigheid van de berekende oplossing wordt dus in hoge mate bepaald door het kwadraat van het conditiegetal van A. Volgens stelling 11 mogen we dat ook verwachten indien $\rho_{LS} \neq 0$ (dus als $\theta \neq 0$). Echter als ρ_{LS} klein is, dan is ook $\tan\theta$ klein en dan zou men, met een stabieler algoritme, in principe mogen hopen op een kleinere relatieve fout in $\hat{x}$.
Zo'n stabiel algoritme wordt verkregen via de op Householder transformaties gebaseerde QR-ontbinding. We hebben in Hoofdstuk 6 al gezien dat we met een (orthogonale) Householder transformatie de eerste kolom van A kunnen overvoeren in een veelvoud van de eerste eenheidsvector. De elementen in de 2^e kolom van A, beneden de diagonaal, kunnen naar nul getransformeerd worden via een Householder transformatie van de gedaante:

$$\begin{pmatrix} 1 & 0 & \ldots & 0 & 0 \\ 0 & & & & \\ \vdots & & I - \frac{2ww^T}{w^Tw} & & \\ 0 & & & & \end{pmatrix}$$

Deze transformatie laat de eerste kolom van A ongemoeid. Op deze wijze verder gaand kan men in A alle elementen beneden de diagonaal naar nul transformeren:

$$Q^TA = R, \text{ met } Q \in R^{m\times m},$$

waarbij R bovendriehoeksvorm heeft. We geven het bovenste n bij n stuk van R aan met R_1. Zij verder $Q^T b = (c, d)^T$, met $c \in R^n$ en $d \in R^{m-n}$, dan geldt

$$\|Ax - b\|_2^2 = \|Q^T Ax - Q^T b\|_2^2 = \|R_1 x - c\|_2^2 + \|d\|_2^2,$$

voor alle $x \in R^n$. Als $rang(A) = rang(R_1) = n$, dan geldt $x_{LS} = R_1^{-1} c$ en $\rho_{LS} = \|d\|_2$.
Dus als A volle rang heeft, dan kan het overbepaalde stelsel eenduidig in kleinste kwadraten zin worden opgelost via de QR-ontbinding van A. Uit de resultaten in Hoofdstuk 6 volgt dat de QR-ontbinding ongeveer $n^2(m - n/3)$ flops kost. Zij $P_1, \ldots, P_n$ de opeenvolgende Householder transformaties, dan kan men $Q^T b$ uitrekenen via $Q^T b = P_n \cdots P_1 b$, en dus is het niet nodig Q^T zelf ook uit te rekenen of op te slaan. Het berekenen van $Q^T b$ vergt nog eens $\mathcal{O}(mn)$ flops.

Opgave 19 *Verifiëer het aantal flops dat nodig is om de QR-ontbinding van A tot stand te brengen. Bepaal ook het aantal flops dat nodig is om daarna x_{LS} te berekenen. Vergelijk de totale kosten met de rekenkosten nodig om x_{LS} via de normaalvergelijkingen en Choleski's methode te bepalen.*

9.2 De Singular Value Decomposition

Tot nu toe hebben we overbepaalde stelsels bekeken waarbij de rang van de matrix gelijk is aan het aantal kolommen. Wat echter te doen als de rang kleiner is? Met deze situatie moeten we rekening houden als de relatieve fout in het rechterlid, of in A, groot is t.o.v. $\kappa_2(A)$, bijv. als

$$\kappa_2(A)\{\frac{\|\partial A\|_2}{\|A\|_2} + \frac{\|\partial b\|_2}{\|b\|_2}\} > 1$$

(zie bijvoorbeeld stelling 11).
Men kan dit soort problemen te lijf gaan via QR met pivot strategie (zie bijvoorbeeld LINPACK: Chapter 9), echter de zogenaamde *Singular Value Decomposition* (SVD) biedt meer mogelijkheden. We zullen deze aanpak, zij het enigszins oppervlakkig, gaan bekijken. De eigenwaarden van $A^T A$ spelen een belangrijke rol in de analyse van de afhankelijkheid van x_{LS} ten aanzien van het rechterlid. We merken alvast op dat de eigenwaarden $\lambda_1, \ldots, \lambda_n$ van $A^T A$ allen niet-negatief zijn.

Definitie 5 *De parameters $\sigma_i \equiv \sqrt{\lambda_i}$ heten de singuliere waarden van A. We zullen aannemen dat de singuliere waarden σ_i in dalende volgorde genummerd zijn:*

$$\sigma_n \geq \sigma_{n-1} \geq \cdots \geq \sigma_1 \geq 0.$$

De nu volgende stelling garandeert het bestaan van de SVD.

Stelling 12 *Zij $A \in R^{m \times n}$, dan bestaan er orthogonale matrices $U \in R^{m \times m}$ en $V \in R^{n \times n}$, met $U^T U = I_m$, $V^T V = I_n$, zodat $U^T A V = \Sigma \equiv diag(\sigma_p, \ldots, \sigma_1)$, met $p = \min(m, n)$ en $\Sigma \in R^{m \times n}$.*

Bewijs: Zij $\sigma_p = \|A\|_2$. Volgens de definitie geldt

$$\sigma_p = \max_{\|x\|_2 = 1} \frac{\|Ax\|_2}{\|x\|_2} = \frac{\|A\tilde{x}\|_2}{\|\tilde{x}\|_2},$$

met $p = \min(m, n)$.
De vector y wordt zo gekozen dat $\sigma_p y = A\tilde{x}$. Bovendien eisen we nog dat $\|y\|_2 = 1$. Merk op dat $x \in R^m$, $y \in R^n$.
Als $\sigma_p \neq 0 \Rightarrow y = \frac{1}{\sigma_p} A\tilde{x}$, en als $\sigma_p = 0$ dan kan y willekeurig gekozen worden.
We definiëren verder $V_p = (\tilde{x}, V_1) \in R^{n\times n}$ en $U_p = (y, U_1) \in R^{m\times m}$, waarbij U_p en V_p orthogonale matrices zijn. Dit is mogelijk omdat een vector $\tilde{x} \in R^n$ met $\|\tilde{x}\|_2 = 1$ altijd aan te vullen is tot een orthonormale basis voor R^n.
Daaruit volgt dat $U_p^T A V_p$ de volgende structuur heeft

$$A_1 \equiv U_p^T A V_p = \begin{pmatrix} \sigma_p & w^T \\ 0 & B \end{pmatrix}.$$

Er volgt dan

$$\|A_1 \begin{pmatrix} \sigma_p \\ w \end{pmatrix}\|_2^2 \geq (\sigma_p^2 + w^T w)^2$$

en dus, als $\|(\sigma_p, w)^T\|_2 \neq 0$ dan

$$\|A_1\|_2^2 \geq \frac{\|A_1(\sigma_p, w)^T\|_2^2}{\|(\sigma_p, w)^T\|_2^2} \geq \sigma_p^2 + w^T w,$$

en in het andere geval $\|A_1\|_2^2 \geq \sigma_p^2 + w^T w = 0$.
Echter $\|A_1\|_2^2 = \|U_p^T A V_p\|_2^2 = \|A\|_2^2 = \sigma_p^2 \Rightarrow w^T w = 0 \Rightarrow w = 0$.
Op dezelfde wijze gaan we verder met B. Dat leidt dan uiteindelijk tot

$$U_1^T \cdots U_p^T A V_p \cdots V_1 = U^T A V = diag(\sigma_p, \ldots, \sigma_1).$$

Dus:

$$V^T A^T U U^T A V = V^T A^T A V = diag(\sigma_p^2, \ldots, \sigma_1^2).$$

Daaruit volgt dat $\sigma_j^2 = \lambda_j$, waarbij λ_j de j-de eigenwaarde van $A^T A$ is. $\square$

Voor de goede orde, als $m > n$, dan heeft $U^T A V$ een staande rechthoeksvorm en anders een liggende.
Enige naamgeving: de σ_j heten de singuliere waarden van A, de kolommen u_j van U heten de *links singuliere vectoren* en de kolommen v_i van V heten de *rechts singuliere vectoren* van A.

Opgave 20 *Ga na dat* $Av_i = \sigma_i u_i$ *en* $A^T u_i = \sigma_i v_i$ *voor* $i = 1, \ldots, \min(m, n)$.

In de volgende stelling wordt een aantal belangrijke eigenschappen van de SVD op een rijtje gezet.

Stelling 13 *Zij* $U^T A V = diag(\sigma_p, \ldots, \sigma_1) \in R^{m\times n}$ *en zij*
$\sigma_p \geq \ldots \geq \sigma_r > \sigma_{r-1} = \cdots = \sigma_1 = 0$.
Dan geldt:

$$rang(A) = p - r + 1$$

$$N(A) = \{v_{r-1}, \ldots, v_1\}$$

$$R(A) = \{u_p, \ldots, u_r\}$$
$$A = \sum_{i=p}^{r} \sigma_i u_i v_i^T = U_r \Sigma_r V_r^T,$$
met $U_r = [u_p, \ldots, u_r]$, $V_r = [v_p, \ldots, v_r]$, $\Sigma_r = diag(\sigma_p, \ldots, \sigma_r)$ *en tenslotte* $\|A\|_2 = \sigma_p$, $p = \min(m, n)$.

Opgave 21 *Bewijs stelling 13.*

Met behulp van de SVD kunnen we nu ook x_{LS} bepalen. Over de berekening van de SVD zelf, via Householder transformaties, zullen we het hier verder niet hebben.

Er geldt, vanwege $x = Vy$ (ofwel $y = V^T x$), dat
$$\|Ax - b\|_2 = \|U^T Ax - U^T b\|_2 = \|U^T AVy - U^T b\|_2 = \|\Sigma y - U^T b\|_2.$$
Zij nu $\sigma_p, \ldots, \sigma_r \neq 0$. Dan is de vector y met minimale 2-norm waarvoor $\|Ax - b\|_2$ minimaal is, gegeven door:
$$\begin{aligned}
&y_p = (U^T b)_p / \sigma_p = (u_p^T b)/\sigma_p \\
&\vdots \\
&y_r = (U^T b)_r / \sigma_r = (u_r^T b)/\sigma_r \\
&y_{r-1} = 0 \\
&\vdots \\
&y_1 = 0
\end{aligned}$$
Bedenk nog dat $\|y\|_2 = \|x_{LS}\|_2$. Daaruit volgt dat
$$x_{LS} = Vy = \sum_{i=r}^{p} [(u_{p-i+1}^T b)/\sigma_i] v_i$$
$$\text{en} \quad \rho_{LS} = \sqrt{\sum_{i=p-r+1}^{m} (u_i^T b)^2}.$$
In de numerieke praktijk zal het maar zelden voorkomen dat één of meer singuliere waarden exact gelijk zijn aan nul. Men heeft aangetoond dat de berekende waarden $\hat{\sigma_j}$ van A dicht bij de exacte waarden σ_j liggen: $|\sigma_j - \hat{\sigma_j}| \leq \epsilon \sigma_p$, waarbij ϵ een bescheiden veelvoud van de machine precisie u is. Men vervangt daarom in de praktijk singuliere waarden $\sigma_j < \partial$ door nul, en verlaagt de numerieke rang van A dienovereenkomstig, waarbij ∂ een geschikt gekozen drempelwaarde is , bijvoorbeeld $u\|A\|_\infty$ of $u\|A\|_2$.
Preciezer: Men zegt dat A *numerieke rang* $p - r + 1$ heeft als de $\hat{\sigma_j}$ voldoen aan
$$\hat{\sigma_p} \geq \ldots \geq \hat{\sigma_r} > \partial \geq \hat{\sigma_{r-1}} \geq \ldots \geq \hat{\sigma_1} \geq 0.$$
Als de (meet) fouten in het rechterlid b aanleiding geven tot een fout ∂b met $\|\partial b\|_2 = \epsilon \|b\|_2$, dan zien we dat de onzekerheid in $x_{\hat{LS}}$ in de richting van v_j een grootte heeft van $[(u_j^T \partial b)/\sigma_j]$. Als $\sigma_j < \|\partial b\|_2 / \|b\|_2$, dan is het heel wel mogelijk dat
$$|\frac{u_j^T \partial b}{\sigma_j}| > |u_j^T b|$$

en dan is dus de onzekerheid in x_{LS} groter dan de bijdrage in de richting v_j aan ρ_{LS}.
Evenzo kan een onzekerheid ∂A in A, met $\|\partial A\|_2 = \epsilon \|A\|_2$ leiden tot onzekerheden $\mathcal{O}(\epsilon \sigma_p)$ in de σ_j.
Daarom kiest men ∂ veelal in de orde van $\epsilon \|A\|_2$. Het is evenwel raadzaam om in dit soort gevallen de singuliere waarden σ_j te inspecteren en te bekijken of ze wellicht uiteenvallen in een groep relatief "grote" en een groep relatief "kleine" waarden, d.w.z. waarden kleiner dan $\epsilon \|A\|_2$. In zulk soort gevallen ligt de keuze van ∂ dan wel voor de hand. Voor details zie bijvoorbeeld de aanwijzingen bij de SVD routines in LINPACK en de doorverwijzingen in [10].

9.3 Numerieke Voorbeelden

In deze paragraaf bekijken we numerieke resultaten die verkregen zijn door een geschikt geconstrueerd overbepaald stelsel $Ax = b$ met de diverse besproken technieken op te lossen.
We construeren daartoe een matrix $A \in R^{100 \times 50}$ met gegeven singuliere waarden, en wel zodanig dat de conditie van $A^T A$ relatief vrij slecht is. Alle berekeningen zijn uitgevoerd op een CONVEX C240 in (single precision) 23-bits floating point arithmetiek (d.w.z. $u \approx 1.2 \cdot 10^{-7}$.
De singuliere waarden zijn als volgt gekozen

$$\sigma_i = 1/i \qquad \text{voor} \quad i \neq 1, 11, 21, 31, 41$$

$$\sigma_i = 0.0003 \qquad \text{voor} \quad i = 1, 11, 21, 31, 41.$$

De konditie van A zou dus ongeveer 3000 moeten zijn, terwijl $A^T A$ een konditiegetal 10^7 zou moeten hebben. Dat houdt in dat $A^T A$ in de gebruikte rekenprecisie vrijwel niet te onderscheiden is van een singuliere matrix. Net zoals in Hoofdstuk 7, paragraaf 6, wordt A weer van een meer algemene gedaante voorzien door de diagonaalmatrix met diagonaal elementen σ_i, voor en na te vermenigvuldigen met 5 random gekozen Householder transformaties.
Vervolgens berekenen we de rechts singuliere vectoren v_i van A en we kiezen x als volgt

$$x = \sum_{i=1}^{n} \gamma_i v_i$$

$$\text{met} \quad \gamma_i = 0.0003 \quad \text{als} \quad \sigma_i = 0.0003$$

$$\text{en} \quad \gamma_i = 1 \quad \text{in de overige gevallen.}$$

De gedachte hierachter is als volgt: stel dat het exacte stelsel vergelijkingen de singuliere waarden $\sigma_i = 1/i$ heeft voor $i \neq 1, 11, \ldots$ en $\sigma_i = 0$ voor $i = 1, 11, \ldots$. Het door ons geconstrueerde stelsel vatten we nu op als een verstoring van dit exacte stelsel (zo'n verstoring zou bijvoorbeeld in de praktijk kunnen ontstaan door meet of modelfouten).
Stel voorts dat de oplossing van het exacte stelsel geen componenten heeft in de v_i richtingen behorend bij de singuliere waarden gelijk aan nul. Dan lijkt het redelijk aan te nemen dat ons verstoorde stelsel ook slechts kleine componenten heeft in de v_i richtingen corresponderend met de singuliere waarden gelijk aan 0.0003. Voor deze kleine componenten hebben we ook de waarde 0.0003 gekozen. In wezen zijn we dus niet in deze (verstoorde) componenten geïnteresseerd, maar deze componenten werken natuurlijk wel door in de oplossing die we te zien krijgen.

Tenslotte berekenen we een rechterlid $\tilde{b}$ door met de gegeven x de vector $\tilde{b} = Ax$ uit te rekenen. Nu zit $\tilde{b}$ dus in het opspansel van de kolommen van A en dat zal in de praktijk wel niet zo erg vaak voorkomen. Daarom verstoren we $\tilde{b} \in R^{100}$ nog als volgt

$$b_i = \tilde{b}_i(1 + 2 \cdot eps \cdot (r_i - 0.5)),$$

voor gegeven eps en met r_i uniform verdeelde random getallen uit $(0, 1)$.

Het aldus verkregen stelsel $Ax = b$ wordt op de volgende 3 manieren opgelost:

- Via de methode van Choleski, losgelaten op de de normaalvergelijkingen $A^T Ax = A^T b$, gevolgd door iteratief naverfijnen (vanwege de slechte conditie) (LINPACK routines SPOCO en SPOSL). De Choleski decompositie levert meteen een schatting op voor het conditiegetal $\kappa_\infty(A^T A)$.
- Via de QR ontbinding van A (LINPACK subroutines SQRDC en SQRSL).
- Via de SVD van A (LINPACK subroutine SSVDC). We kijken hierbij ook naar het effect van een geschikte keuze van ∂ (zie paragraaf 9.2).

Het Programma (FORTRAN77):

```
      program test2
      dimension a(100,50),b(100),jpvt(50),qraux(50),bs(100),
     1         qb(100),qtb(100),x(50),rsd(100),ax(100),ata(50,50)
     2         ,atb(50),ataa(50,50),w(100),as(100,50),us(100,100),
     3         vs(50,50),sv(50),work(100),e(50),qrb(100)
c
c  programma om overbepaald stelsel op te lossen via diverse methoden
c
      m=100
      n=50
      eps=.3e-3
      eps=0.
      small=.3e-3
      print 700,eps,small
 700  format('   eps=',e12.5,/,
     1  '  de kleine sing. waarden zijn : ',e13.6,//)
      do 10 i=1,m
         do 10 j=1,n
            a(i,j)=0.
  10  continue
c  slechts de diagonaal van de matrix wordt gevuld
      do 20 i=1,n
         a(i,i)=1.e0/i
  20  continue
```

```
c  een paar elementen wordt een kleine waarde toegekend, om zo een
c  slecht geconditioneerde matrix te simuleren
      do 30 i=1,n,10
        a(i,i)=small
  30  continue
c  nu wordt de oorspronkelijke matrix voor en na-vermenigvuldigd met 5
c  householder matrices, omdat gauss-eliminatie op een diagonaal matrix
c  niet de gewenste effecten laat zien. Deze householder matrices worden
c  met random vectoren gegenereerd.
      irand=1989
      do 50 kk=1,5
      call rando(m,irand,w)
c  sdot berekent het inproduct van twee vectoren in enkele precisie
      beta=2./sdot(m,w,1,w,1)
         do 40 i=1,n
            s=sdot(m,w,1,a(1,i),1)*beta
            do 35 j=1,m
               a(j,i)=a(j,i)-s*w(j)
  35        continue
  40     continue
  50  continue
      do 70 kk=1,5
         call rando(n,irand,w)
         beta=2./sdot(n,w,1,w,1)
         do 60 i=1,m
            s=sdot(n,w,1,a(i,1),m)*beta
            do 55 j=1,n
               a(i,j)=a(i,j)-s*w(j)
  55        continue
  60     continue
  70  continue
      print 1009
 1009 format('  de exacte oplossing ')
c  a(i,j) is nu voor en na-vermenigvuldigd met een random orthogonale
c  matrix.
      do 80 i=1,m
         do 80 j=1,n
            as(i,j)=a(i,j)
  80  continue
      job=11
      call ssvdc(as,m,m,n,sv,e,us,m,vs,n,work,job,info)
      do 81 i=1,n
         x(i)=0.
 81   continue
      do 85 i=1,n
```

```
        gamma=1.
        if (sv(i).le.2.*small) gamma=small
        do 84 j=1,n
           x(j)=x(j)+gamma*vs(j,i)
 84     continue
 85   continue
      print 1000,x
      do 86 i=1,m
         do 86 j=1,n
 86         as(i,j)=a(i,j)
c  het rechterlid b van het stelsel wordt nu in dubbele precisie berekend,
c  en daarna in enkele precisie opgeborgen.
      do 100 i=1,m
         b(i)=sdsdot(n,a(i,1),m,x(1),1,0.)
 100  continue
      call rando(m,irand,w)
      do 110 i=1,m
 110     b(i)=b(i)*(1.+2.*eps*(w(i)-0.5))
      do 120 i=1,m
 120     bs(i)=b(i)
      do 130 i=1,n
         do 130 j=1,n
            sum=0.
            do 125 k=1,m
              sum=sum+a(k,i)*a(k,j)
 125        continue
            ata(i,j)=sum
 130  continue
      do 140 i=1,n
 140  atb(i)=sdot(m,a(1,i),1,b,1)
      print 3000,m
      call sgeim(ata,n,n,atb,x,relerr,ataa,qrb,jpvt,info)
      print 1001
      print 1000,x
c  berekening van de 2-norm van het residu
      sum=0.
      print 3000,m
      print 3000,n
 3000 format('  m=',i5)
      do 150 i=1,m
         sas=sdot(n,as(i,1),m,x,1)
         sum=sum+(sas-bs(i))**2
 150  continue
      residu=sqrt(sum)
      print 1002,residu
```

```
      call sqrdc(a,m,m,n,qraux,jpvt,work,0)
      call sqrsl(a,m,m,n,qraux,b,qb,rsd,x,rsd,qb,110,info)
      rhols=sqrt(sdot(m,rsd,1,rsd,1))
      bnorm=sqrt(sdot(m,b,1,b,1))
      print 1003
      print 1000,x
      print 1004,rhols,rhols/bnorm
      do 160 i=1,m
         do 160 j=1,n
160         a(i,j)=as(i,j)
      sum=0.
      do 165 i=1,m
         sum=sum+(sdot(n,a(i,1),m,x,1)-bs(i))**2
165   continue
      residu=sqrt(sum)
      print 1002,residu
      print 1005,info
1005  format('  de singuliere waarden, info= ',i4)
      print 1000,sv
      print 1006
1006  format('  berekening van x via alle singuliere waarden ')
      do 170 i=1,n
170      x(i)=0.
      do 180 k=1,n
         pp=sdot(m,us(1,k),1,bs,1)/sv(k)
         do 175 i=1,n
175         x(i)=x(i)+pp*vs(i,k)
180   continue
      print 1000,x
      sum=0.
      do 190 i=1,m
190      sum=sum+(sdot(n,a(i,1),m,x,1)-bs(i))**2
      residu=sqrt(sum)
      print 1002,residu
      delta=1.e-3
      isv=0
      do 200 i=1,n
         if (sv(i).lt.delta) goto 210
200      isv=i
210   continue
      print 1007,isv
1007  format('  met grootste ',i4,' singuliere waarden')
      do 220 i=1,n
220      x(i)=0.
      do 230 k=1,isv
```

```
         pp=sdot(m,us(1,k),1,bs,1)/sv(k)
         do 225 i=1,n
 225        x(i)=x(i)+pp*vs(i,k)
 230  continue
      print 1000,x
      sum=0.
      do 240 i=1,m
 240     sum=sum+(sdot(n,a(i,1),m,x,1)-bs(i))**2
      residu=sqrt(sum)
      print 1002,residu
 1000 format(5('  ',e13.6))
 1001 format('  de oplossing van ATAx=ATb na het naverfijnen:')
 1003 format('  de oplossing via QR-ontbinding:')
 1002 format('  2-norm van residu=',e13.6)
 1004 format('  residue= ',e13.6,'  rhols/b=',e13.6)
      end
      function sdsdot(n,x,incx,y,incy,c)
      dimension x(incx,1),y(incy,1)
      double precision sum
c  inproduct berekening in dubbele precisie met enkele lengte resultaat.
      sum=0.d0
      if (n.le.0) goto 20
      do 10 i=1,n
         sum=sum+dble(x(1,i))*dble(y(1,i))
 10   continue
 20   sum=sum+dble(c)
      sdsdot=sngl(sum)
      return
      end
      subroutine sgeim(a,lda,n,b,x,relerr,aa,r,ipvt,info)
      integer lda,n,info
      real a(lda,1),b(n),x(n),relerr,aa(lda,1),r(n)
      integer ipvt(n)
      real z(300),rcond
c
c  routine om het effect van iteratief naverfijnen te illustreren,
c  de volgende BLAS en LINPACK routines worden gebruikt:
c  sdsdot,sasum,sgefa,sgesl
c
     real xnorm,rnorm,sasum,sdsdot
c  omdat het residu moet worden berekend met de oorspronkelijke matrix,
c  en omdat deze wordt overschreven (door sgeco) door zijn LU-ontbinding,
c  copieren we de matrix eerst naar aa( , ).
      do 20 j=1,n
         do 10 i=1,n
```

```
           aa(i,j)=a(i,j)
 10      continue
          x(j)=b(j)
 20   continue
c  berekening van de LU-decompositie met partial pivoting
      call spoco(aa,lda,n,rcond,z,info)
c  rcond bevat een schatting voor de inverse van het conditie getal
      print 1002,rcond
      if (1.0+rcond.eq.1.0) goto 60
 1002 format(' schatting inverse conditie getal: ',e13.6)
c  oplossen van het stelsel met inachtneming van de pivot volgorde (in ipvt):
      call sposl(aa,lda,n,x)
      print 998
      print 3000,n
 3000 format('  n(sgeim)=',i5)
      print 999,x
 999  format(5('  ',e13.6))
 998  format(' de oplossing voor het naverfijnen:')
c  berekening van de oneindig-norm van x
      xnorm=sasum(n,x,1)
      print 1000,xnorm
 1000 format(' de oneindig-norm van x: ',e13.6)
      if (xnorm.eq.0.0) goto 60
c  berekening van het nieuwe residu in dubbele precisie:
      rold=xnorm
      do 50 iter=1,20
         print 5000,n
 5000     format('  voor sdsdot ',i5)
         do 30 i=1,n
            r(i)=sdsdot(n,a(i,1),lda,x(1),1,-b(i))
  30     continue
       print 4000,n
 4000  format('  voor rnorm',i5)
       rnorm=sasum(n,r,1)
       rnorm=sdot(n,r,1,r,1)
       rnorm=sqrt(rnorm)
      print 1007,iter,rnorm
 1007 format('  iteratie nr ',i4,' norm van residu= ',e13.6)
c  het rechterlid r wordt overschreven door de oplossing
         call sposl(aa,lda,n,r)
c  correctie van de oplossing
         do 40 i=1,n
            x(i)=x(i)-r(i)
 40      continue
c  berekening van de oneindig-norm van de correctie:
```

```
        rnorm=sasum(n,r,1)
        print 1001,iter,rnorm
 1001   format('  iteratie nr. ',i4,'  norm van correctie= ',e13.6)
c  hier wordt gekeken of de correctie nog effect heeft op de oplossing:
        t=xnorm+rnorm
        if (t.eq.xnorm) goto 60
        if (rnorm.gt.rold) goto 60
        rold=rnorm
 50   continue
      info =-1
 60   continue
      return
      end
      subroutine rando(n,irand,x)
c  genereert n pseudo random getallen in de vector x
      dimension x(n)
      ihalf=4096
      do 10 i=1,n
         irand=mod(irand*125,8192)
         x(i)=(irand-ihalf)/8192.0
 10   continue
      return
      end
```

Experiment I:
$eps = 0$. De vector b zit nu dus in het opspansel van de kolommen van A.

De exacte oplossing:

-0.151004E+01	-0.835064E+00	-0.667678E+00	-0.627037E+00	0.305617E+00
0.731139E+00	0.113613E+01	0.215086E+00	0.990981E+00	0.149293E+01
0.141181E+00	0.249551E+00	-0.690287E+00	0.168589E+01	-0.860096E+00
0.656764E+00	0.638315E+00	-0.202423E+01	-0.180248E-01	0.379057E+00
-0.300674E+00	-0.156333E+01	-0.108887E+01	0.252356E+00	0.803267E+00
-0.368727E+00	0.133039E+01	-0.818489E+00	-0.103463E+01	0.865767E+00
-0.611070E+00	0.124585E+00	-0.150386E+01	-0.455283E+00	0.706082E+00
-0.121467E+01	0.168247E+01	0.944064E+00	-0.657883E+00	-0.446674E+00
0.629427E+00	-0.103219E+01	-0.926004E-02	0.251413E+00	0.121047E+01
-0.104013E+01	-0.748256E+00	-0.806859E+00	0.185722E+01	0.208722E+00

Schatting voor het inverse conditie getal: $0.108631E-06$

De oplossing voor het naverfijnen:

-0.149821E+01	-0.827691E+00	-0.659594E+00	-0.629316E+00	0.297896E+00
0.730947E+00	0.113218E+01	0.218145E+00	0.990059E+00	0.149981E+01
0.132815E+00	0.248121E+00	-0.688065E+00	0.169476E+01	-0.855294E+00
0.653968E+00	0.637436E+00	-0.203088E+01	-0.192133E-01	0.376209E+00
-0.248371E+00	-0.156611E+01	-0.109445E+01	0.249763E+00	0.806393E+00
-0.366769E+00	0.132999E+01	-0.814141E+00	-0.103458E+01	0.863228E+00
-0.614660E+00	0.131339E+00	-0.150546E+01	-0.451658E+00	0.703321E+00
-0.122055E+01	0.168472E+01	0.938973E+00	-0.654732E+00	-0.440877E+00
0.634521E+00	-0.103543E+01	-0.628313E-02	0.250236E+00	0.121222E+01
-0.104127E+01	-0.749073E+00	-0.807189E+00	0.185855E+01	0.216238E+00

Het naverfijnen:
iteratie nr. 1 : $\|\Delta x\|_2 = 0.724511E-01$
iteratie nr. 2 : $\|\Delta x\|_2 = 0.241435E-03$
iteratie nr. 3 : $\|\Delta x\|_2 = 0.196711E-05$
iteratie nr. 4 : $\|\Delta x\|_2 = 0.857619E-06$

De oplossing van $A^TAx = A^Tb$ na het naverfijnen:

-0.149986E+01	-0.828227E+00	-0.659447E+00	-0.629033E+00	0.295910E+00
0.730098E+00	0.113323E+01	0.217176E+00	0.991665E+00	0.150132E+01
0.136547E+00	0.248562E+00	-0.688101E+00	0.169361E+01	-0.855893E+00

0.654039E+00	0.635874E+00	-0.203047E+01	-0.205724E-01	0.376509E+00
-0.261199E+00	-0.156762E+01	-0.109551E+01	0.249025E+00	0.805632E+00
-0.365141E+00	0.133057E+01	-0.815070E+00	-0.103412E+01	0.863652E+00
-0.608437E+00	0.131523E+00	-0.150675E+01	-0.451252E+00	0.702409E+00
-0.121939E+01	0.168416E+01	0.938211E+00	-0.654299E+00	-0.442357E+00
0.624481E+00	-0.103509E+01	-0.539996E-02	0.249473E+00	0.121109E+01
-0.104117E+01	-0.749463E+00	-0.809035E+00	0.185908E+01	0.214533E+00

$$\rho_{LS} = 0.150758E - 04$$

De oplossing via QR-ontbinding:

-0.151003E+01	-0.835061E+00	-0.667674E+00	-0.627038E+00	0.305614E+00
0.731139E+00	0.113612E+01	0.215085E+00	0.990981E+00	0.149293E+01
0.141149E+00	0.249549E+00	-0.690285E+00	0.168590E+01	-0.860093E+00
0.656760E+00	0.638316E+00	-0.202424E+01	-0.180276E-01	0.379053E+00
-0.300691E+00	-0.156333E+01	-0.108887E+01	0.252353E+00	0.803271E+00
-0.368728E+00	0.133039E+01	-0.818483E+00	-0.103463E+01	0.865766E+00
-0.611052E+00	0.124592E+00	-0.150386E+01	-0.455278E+00	0.706082E+00
-0.121467E+01	0.168247E+01	0.944063E+00	-0.657879E+00	-0.446669E+00
0.629428E+00	-0.103220E+01	-0.925481E-02	0.251410E+00	0.121047E+01
-0.104013E+01	-0.748253E+00	-0.806856E+00	0.185722E+01	0.208731E+00

$$\rho_{LS} = 0.387006E - 07 \qquad \sin\theta = 0.494310E - 07$$

Het oplossen via de SVD:

a. de singuliere waarden:

0.500000E+00	0.333333E+00	0.250000E+00	0.200000E+00	0.166667E+00
0.142857E+00	0.125000E+00	0.111111E+00	0.100000E+00	0.833333E-01
0.769231E-01	0.714286E-01	0.666666E-01	0.624999E-01	0.588235E-01
0.555555E-01	0.526316E-01	0.500000E-01	0.454546E-01	0.434783E-01
0.416667E-01	0.400000E-01	0.384615E-01	0.370370E-01	0.357143E-01
0.344827E-01	0.333334E-01	0.312500E-01	0.303031E-01	0.294118E-01
0.285714E-01	0.277778E-01	0.270270E-01	0.263158E-01	0.256410E-01
0.250000E-01	0.238095E-01	0.232558E-01	0.227273E-01	0.222222E-01
0.217391E-01	0.212766E-01	0.208334E-01	0.204082E-01	0.200000E-01
0.300004E-03	0.300001E-03	0.300000E-03	0.299998E-03	0.299996E-03

b. Berekening van x via alle singuliere waarden :

-0.150997E+01	-0.835066E+00	-0.667683E+00	-0.627009E+00	0.305626E+00
0.731151E+00	0.113613E+01	0.215065E+00	0.990983E+00	0.149292E+01
0.141139E+00	0.249535E+00	-0.690297E+00	0.168589E+01	-0.860106E+00
0.656743E+00	0.638325E+00	-0.202425E+01	-0.180446E-01	0.379050E+00
-0.300686E+00	-0.156333E+01	-0.108885E+01	0.252349E+00	0.803273E+00
-0.368733E+00	0.133039E+01	-0.818483E+00	-0.103465E+01	0.865755E+00
-0.611168E+00	0.124611E+00	-0.150388E+01	-0.455261E+00	0.706089E+00
-0.121466E+01	0.168249E+01	0.944072E+00	-0.657880E+00	-0.446657E+00
0.629435E+00	-0.103220E+01	-0.924216E-02	0.251391E+00	0.121047E+01
-0.104012E+01	-0.748255E+00	-0.806852E+00	0.185724E+01	0.208753E+00

$$\rho_{LS} = 0.587191E - 06$$

c. Berekening van x nu alleen met de grootste 45 singuliere waarden:

-0.151029E+01	-0.835058E+00	-0.667684E+00	-0.627126E+00	0.305685E+00
0.731143E+00	0.113610E+01	0.215176E+00	0.990936E+00	0.149288E+01
0.140846E+00	0.249604E+00	-0.690261E+00	0.168592E+01	-0.860057E+00
0.656814E+00	0.638360E+00	-0.202420E+01	-0.179174E-01	0.379061E+00
-0.300678E+00	-0.156327E+01	-0.108889E+01	0.252402E+00	0.803303E+00
-0.368795E+00	0.133035E+01	-0.818453E+00	-0.103461E+01	0.865798E+00
-0.610783E+00	0.124504E+00	-0.150377E+01	-0.455347E+00	0.706122E+00
-0.121474E+01	0.168243E+01	0.944080E+00	-0.657893E+00	-0.446672E+00
0.629695E+00	-0.103217E+01	-0.932913E-02	0.251497E+00	0.121050E+01
-0.104018E+01	-0.748230E+00	-0.806797E+00	0.185716E+01	0.208697E+00

$$\rho_{LS} = 0.634447E - 06$$

Experiment II:
We kiezen nu $eps = 3 \cdot 10^{-4}$. Op grond van stelling 11 mogen we nu verwachten dat door een dermate grote fout de oplossing goeddeels onherkenbaar zal zijn t.o.v. de x waarmee het ongestoorde rechterlid was geconstrueerd.

De exacte oplossing:

-0.151004E+01	-0.835064E+00	-0.667678E+00	-0.627037E+00	0.305617E+00
0.731139E+00	0.113613E+01	0.215086E+00	0.990981E+00	0.149293E+01
0.141181E+00	0.249551E+00	-0.690287E+00	0.168589E+01	-0.860096E+00
0.656764E+00	0.638315E+00	-0.202423E+01	-0.180248E-01	0.379057E+00
-0.300674E+00	-0.156333E+01	-0.108887E+01	0.252356E+00	0.803267E+00
-0.368727E+00	0.133039E+01	-0.818489E+00	-0.103463E+01	0.865767E+00
-0.611070E+00	0.124585E+00	-0.150386E+01	-0.455283E+00	0.706082E+00
-0.121467E+01	0.168247E+01	0.944064E+00	-0.657883E+00	-0.446674E+00
0.629427E+00	-0.103219E+01	-0.926004E-02	0.251413E+00	0.121047E+01
-0.104013E+01	-0.748256E+00	-0.806859E+00	0.185722E+01	0.208722E+00

De oplossing van $A^T A x = A^T b$ na het naverfijnen:

-0.147196E+01	-0.826223E+00	-0.654874E+00	-0.619230E+00	0.282562E+00
0.728512E+00	0.113572E+01	0.206903E+00	0.998659E+00	0.151127E+01
0.155416E+00	0.243308E+00	-0.690155E+00	0.169249E+01	-0.858618E+00
0.645517E+00	0.629022E+00	-0.203694E+01	-0.354814E-01	0.374010E+00
-0.279467E+00	-0.157629E+01	-0.109634E+01	0.241421E+00	0.803725E+00
-0.355859E+00	0.133390E+01	-0.816221E+00	-0.103586E+01	0.859360E+00
-0.620834E+00	0.145556E+00	-0.152019E+01	-0.440262E+00	0.695817E+00
-0.121318E+01	0.168870E+01	0.933704E+00	-0.649726E+00	-0.440849E+00
0.579080E+00	-0.103887E+01	0.689867E-02	0.237672E+00	0.120581E+01
-0.103673E+01	-0.752306E+00	-0.817295E+00	0.186820E+01	0.219741E+00

$$\rho_{LS} = 0.560065E-04$$

De oplossing via QR-ontbinding:

-0.149580E+01	-0.836463E+00	-0.669877E+00	-0.623670E+00	0.308163E+00
0.731323E+00	0.113628E+01	0.211871E+00	0.989437E+00	0.148996E+01
0.156658E+00	0.246726E+00	-0.690937E+00	0.168470E+01	-0.861293E+00
0.655542E+00	0.638772E+00	-0.202493E+01	-0.200503E-01	0.377789E+00
-0.292033E+00	-0.156243E+01	-0.108496E+01	0.251823E+00	0.802056E+00
-0.368938E+00	0.133066E+01	-0.818563E+00	-0.103566E+01	0.864953E+00
-0.630214E+00	0.126454E+00	-0.150385E+01	-0.454563E+00	0.706188E+00
-0.121285E+01	0.168449E+01	0.946939E+00	-0.658381E+00	-0.445087E+00

0.636965E+00	-0.103201E+01	-0.746145E-02	0.249613E+00	0.121075E+01
-0.103928E+01	-0.747391E+00	-0.804780E+00	0.185892E+01	0.211359E+00

$$\rho_{LS} = 0.481841E-04 \qquad \sin\theta = 0.615608E-04$$

Berekening van de oplossing via de SVD:

a. berekening van x via alle singuliere waarden

-0.149576E+01	-0.836466E+00	-0.669883E+00	-0.623643E+00	0.308168E+00
0.731334E+00	0.113629E+01	0.211852E+00	0.989444E+00	0.148995E+01
0.156639E+00	0.246714E+00	-0.690950E+00	0.168470E+01	-0.861307E+00
0.655525E+00	0.638778E+00	-0.202494E+01	-0.200685E-01	0.377789E+00
-0.292051E+00	-0.156243E+01	-0.108494E+01	0.251819E+00	0.802058E+00
-0.368940E+00	0.133067E+01	-0.818565E+00	-0.103567E+01	0.864941E+00
-0.630307E+00	0.126473E+00	-0.150388E+01	-0.454545E+00	0.706191E+00
-0.121284E+01	0.168450E+01	0.946943E+00	-0.658381E+00	-0.445079E+00
0.636944E+00	-0.103202E+01	-0.744890E-02	0.249593E+00	0.121074E+01
-0.103926E+01	-0.747395E+00	-0.804781E+00	0.185893E+01	0.211375E+00

$$\rho_{LS} = 0.481882E-04$$

b. berekening van x met de grootste 45 singuliere waarden

-0.150959E+01	-0.834770E+00	-0.667673E+00	-0.626795E+00	0.305710E+00
0.730805E+00	0.113532E+01	0.215372E+00	0.990290E+00	0.149227E+01
0.140838E+00	0.249589E+00	-0.690392E+00	0.168530E+01	-0.859879E+00
0.656512E+00	0.638060E+00	-0.202357E+01	-0.176954E-01	0.378757E+00
-0.300576E+00	-0.156295E+01	-0.108859E+01	0.252638E+00	0.803238E+00
-0.368835E+00	0.132970E+01	-0.818050E+00	-0.103449E+01	0.865531E+00
-0.610477E+00	0.124761E+00	-0.150300E+01	-0.455395E+00	0.705394E+00
-0.121426E+01	0.168204E+01	0.944524E+00	-0.657756E+00	-0.446304E+00
0.629309E+00	-0.103222E+01	-0.904194E-02	0.251689E+00	0.121003E+01
-0.104043E+01	-0.748343E+00	-0.806630E+00	0.185679E+01	0.208336E+00

$$\|b - Ax\|_2 = 0.491929E-04$$

Hoofdstuk 10

Iteratieve oplosmethoden voor lineaire stelsels

10.1 Basis iteratie methoden

We nemen in dit hoofdstuk verder aan dat A een vierkante matrix is: $A \in R^{n \times n}$ en dat A niet-singulier is.
Iteratieve methoden voor het oplossen van een stelsel lineaire vergelijkingen zijn gebaseerd op het idee om met weinig rekenwerk een gegeven benadering x_i, voor de oplossing x van $Ax = b$, te vervangen door een (hopelijk) betere benadering x_{i+1}.
In formule: $x_{i+1} = \phi(x_i)$. We beperken ons nu tot lineaire functies ϕ. Indien de rij $\{x_i\}$ convergeert naar x, moet dus gelden $x = \phi(x)$, en dus is ϕ te schrijven als $\phi = I + \theta$: $\phi(y) = y + \theta(y)$, met $\theta(x) = 0$. Omdat x moet voldoen aan het stelsel $Ax - b = 0$, ligt de keuze

$$\theta(y) = P^{-1}(b - Ay) \tag{10.1}$$

voor de hand, waarbij $P^{-1} \in R^{n \times n}$ nog een vrij te kiezen niet-singuliere matrix is.

De basis iteratie formule wordt nu

$$x_{i+1} = \phi(x_i) = x_i + P^{-1}(b - Ax_i). \tag{10.2}$$

Voor de iteratiefout $x_{i+1} - x$ geldt dan

$$x_{i+1} - x = (I - P^{-1}A)(x_i - x). \tag{10.3}$$

De iteratie convergeert in 1 iteratieslag, voor elke startvector x_0, indien $P = A$. Men moet dan echter A^{-1} loslaten op het beginresidu $b - Ax_0$, dat wil zeggen men moet een stelsel met A oplossen, en het was ons er juist om begonnen om dat werk te vermijden.
Men kan geschikte iteratieformules krijgen door voor P een of andere benadering voor A te kiezen, zodanig dat $Py = \tilde{b}$, voor gegeven $\tilde{b}$ eenvoudig en goedkoop op te lossen is. Er geldt dan $y = P^{-1}\tilde{b}$, maar we geven aan het oplossen vrijwel altijd de voorkeur boven het expliciet inverteren van P.

De iteratie formule (10.2) kan nog herschreven worden als

$$Px_{i+1} = (P - A)x_i + b = Rx_i + b, \tag{10.4}$$

met $A = P - R$.
We zullen nu een paar voor de hand liggende iteratie methoden gaan bekijken.

Gauss-Jacobi iteratie: We kiezen $P = diag(A)$ en als P niet-singulier is resulteert dit in de Gauss-Jacobi iteratie.

Gauss-Seidel iteratie: Schrijf $A = D - L - U$, met $D = diag(A)$, L het strikt benedendriehoeks gedeelte van A en U het strikt bovendriehoeks gedeelte. De keuze $P = D - L$ leidt, als P niet-singulier, is tot de Gauss-Seidel methode.

Zogenaamde blok (of lijn) varianten, voor matrices A met een geschikte blok structuur (zie Hoofdstuk 8), zijn op voor de hand liggende wijze te verkrijgen.

Uit vergelijking (10.4) volgt

$$x_{i+1} - x = P^{-1}R(x_i - x) = (P^{-1}R)^{i+1}(x_0 - x). \tag{10.5}$$

Om convergentie aan te kunnen tonen voor geschikte P gebruiken we de volgende stelling.

Stelling 14 *Als $A \in C^{n\times n}$ dan geldt $\lim_{m\to\infty} \|A^m\| = 0 \Leftrightarrow \rho(A) < 1$, met $\rho(A) \equiv \max |\lambda_i|$.*

Voor het bewijs zie [25]:blz 13.
Uit deze stelling halen we meteen een voorwaarde voor convergentie van het basis iteratie proces.

Stelling 15 *Zij A niet-singulier en zij $A = P - R$, met P eveneens niet-singulier. Dan convergeert de rij $\{x_i\}$, gedefinieerd door $Px_{i+1} = Rx_i + b$, naar de oplossing x van $Ax = b$ voor elke startvector x_0 mits $\rho(P^{-1}R) < 1$.*

Bewijs: $\|x_{i+1} - x\| \leq \|(P^{-1}R)^{i+1}\|\|x_0 - x\|$. □

Voor de Gauss-Jacobi iteratie methode is strikte diagonaal dominantie voldoende voor gegarandeerde convergentie, namelijk $P = D$, $R = L + U$, en er geldt

$$\rho(P^{-1}R) \leq \|P^{-1}R\|_\infty = \|D^{-1}(L+U)\|_\infty$$

$$= max_i \sum_{j\neq i} |\frac{a_{i,j}}{a_{i,i}}| < 1.$$

De laatste ongelijkheid volgt uit toepassing van de stelling van Gershgorin. We zien dat de methode sneller convergeert, naarmate de elementen $a_{i,i}$ meer overheersen.
De convergentie bewijzen voor de Gauss-Seidel methode zijn meestal ingewikkelder. Als voorbeeld het volgende resultaat.

Stelling 16 *Als $A \in R^{n\times n}$ symmetrisch positief definiet is en $A = D - L - L^T$, met $D = diag(A)$ en L strikt benedendriehoeks, dan convergeert de iteratie methode $(D - L)x_{i+1} = L^T x_i + b$ voor elke x_0.*

Bewijs: Vanwege stelling 14 is het voldoende te bewijzen dat

$$\rho((D-L)^{-1}L^T) < 1.$$

Zij $G \equiv (D-L)^{-1}L^T$ en $G_1 \equiv D^{\frac{1}{2}}GD^{-\frac{1}{2}}$. G_1 heeft dezelfde eigenwaarden als G, namelijk $Gx = \lambda x \Leftrightarrow D^{\frac{1}{2}}Gx = \lambda D^{\frac{1}{2}}x$, en met $x = D^{-\frac{1}{2}}y$ volgt $D^{\frac{1}{2}}GD^{-\frac{1}{2}}y = \lambda y$.
Blijft dus nog aan te tonen dat $\rho(G_1) < 1$.
$G_1 = D^{\frac{1}{2}}(D-L)^{-1}L^TD^{-\frac{1}{2}} = D^{\frac{1}{2}}[D^{\frac{1}{2}}(I - D^{-\frac{1}{2}}LD^{-\frac{1}{2}})D^{\frac{1}{2}}]^{-1}L^TD^{-\frac{1}{2}} = (I-L_1)^{-1}L_1^T$,
waarbij $L_1 \equiv D^{-\frac{1}{2}}LD^{-\frac{1}{2}}$.
Zij λ een eigenwaarde van G_1 met eigenvector x en $\|x\|_2^2 = x^Hx = 1$.
Dan geldt

$$G_1x = \lambda x \Rightarrow (I-L_1)^{-1}L_1^Tx = \lambda x \Rightarrow L_1^Tx = \lambda(I-L_1)x$$

en $x^HL_1^Tx = \lambda(1 - x^HL_1x)$.
Stel $x^HL_1^Tx = a + bi$, dan volgt uit

$$\lambda = \frac{\overline{x^HL_1x}}{1 - x^HL_1x} \quad \text{dat} \quad |\lambda|^2 = \frac{a^2+b^2}{1-2a+a^2+b^2}.$$

De matrix A is positief definiet, dus ook $D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ is positief definiet en $D^{-\frac{1}{2}}AD^{-\frac{1}{2}} = I - L_1 - L_1^T$.
Dus $x^HD^{-\frac{1}{2}}AD^{-\frac{1}{2}}x > 0 \Rightarrow 1 - x^HL_1x - x^HL_1^Tx = 1 - 2a > 0$.
Daaruit volgt dat $|\lambda|^2 < 1 \Rightarrow \rho(G_1) < 1$. □

Opgave 22 *Laat zien dat de Gauss-Jacobi iteratie methode geschreven kan worden in de vorm* $x_{i+1} = x_i + H(b - Ax_i)$.

Opgave 23 *Laat zien dat als A strikt diagonaal dominant is de Gauss-Seidel methode ook gegarandeerd convergeert.*

Allerlei tactische varianten zijn weer mogelijk. Als voorbeeld geven we de volgende succesvolle variant op Gauss-Seidel iteratie. Bij deze methode moet per iteratie slag een benedendriehoeks stelsel opgelost worden. Men kan proberen om zodra een nieuwe coördinaat van x_{i+1} is uitgerekend, deze eerst nog een duwtje in de goede richting te geven, alvorens hem te gebruiken voor de volgende vergelijking. Dat leidt tot het volgende iteratie schema:

Voor $j = 1, \ldots, n$
$(D\tilde{x})_j = (Lx_{i+1})_j + (Ux_i)_j + (b)_j$
$(x_{i+1})_j = (x_i)_j + \omega((\tilde{x})_j - (x_i)_j)$

In dit schema is n het aantal vergelijkingen, met $(y)_j$ geven we de j-de coördinaat aan van de vector y en ω is een geschikt gekozen parameter, de zogenaamde *relaxatiefactor*. De aldus verkregen methode heet de *Successive Over Relaxation* methode (SOR).
Voor een aantal specifieke problemen is bekend hoe groot ω moet zijn om versnelde convergentie te bewerkstelligen. Voor de meeste praktijk problemen evenwel houdt de bepaling van een goede ω een redelijke hoeveelheid werk in. Een bijkomend probleem is dat de convergentie vaak nogal

kritisch van ω afhangt, zowel een iets te grote als een iets te kleine waarde kan aanzienlijk efficiëntie verlies betekenen. Voor details zie [25].

Zoals we uit formule (10.5) zien, geldt

$$\begin{aligned} x_{i+1} - x &= (P^{-1}R)^{i+1}(x_0 - x) && (10.6)\\ &= [P^{-1}(P-A)]^{i+1}(x_0 - x) && (10.7)\\ &= [I - P^{-1}A]^{i+1}(x_0 - x) && (10.8)\\ &= p_{i+1}(P^{-1}A)(x_0 - x), && (10.9) \end{aligned}$$

waarbij $p_i(x)$ een i-de graads polynoom is met $p_i(0) = 1$.
Het polynoom $p_i(x)$, dat bij de basis iteratiemethode hoort, is een heel speciaal polynoom, namelijk

$$p_i(x) = (1 - x)^i.$$

Toch opent dit een nieuw gezichtspunt. We zouden bijvoorbeeld elke iteratie slag een andere splitsing van A kunnen gebruiken. Bij de SOR methode gebeurt dat als we elke slag een andere ω zouden kiezen. Voor de basis iteratie methode krijgen we een heel eenvoudige variant, door in plaats van $A = P - R$ in de i-de slag te kiezen

$$A = \frac{1}{\alpha_i}P - (\frac{1}{\alpha_i}P - P + R),$$

dat wil zeggen, we splitsen $\frac{1}{\alpha_i}P$ van A af in plaats van P. Dat leidt tot de iteratieformule

$$x_{i+1} = (I - \alpha_i P^{-1}A)x_i + \alpha_i P^{-1}b. \tag{10.10}$$

De formule geldt natuurlijk ook voor x zelf:

$$x = (I - \alpha_i P^{-1}A)x + \alpha_i P^{-1}b,$$

dus

$$\begin{aligned} x_{i+1} - x &= (I - \alpha_i P^{-1}A)(x_i - x)\\ &= \prod_{j=0}^{i}(I - \alpha_j P^{-1}A)(x_0 - x)\\ &= p_{i+1}(P^{-1}A)(x_0 - x). && (10.11) \end{aligned}$$

We zien dat $p_i(x)$ nu een willekeurig polynoom kan zijn uit de klasse Π_i^1 (dat zijn de polynomen van graad i en met constante term 1, immers $p_i(0) = 1$).
De vraag rijst nu of we een polynoom $p_i \in \Pi_i^1$ kunnen bedenken dat gunstiger convergentie biedt dan de "basiskeuze" $p_i(x) = (1 - x)^i$. Immers

$$\|x_i - x\| \leq \|p_i(P^{-1}A)\|\|x_0 - x\| \tag{10.12}$$

suggereert dat we een p_i moeten zoeken waarvoor $\|p_i(P^{-1}A)\|$ minimaal is. Als we zo'n p_i gevonden hebben, dat wil zeggen een set $\alpha_0^{(i)}, \ldots, \alpha_{i-1}^{(i)}$, waarmee de iteratie wordt uitgevoerd, dan kan het "optimale" polynoom p_{i+1} er opeens weer heel anders uitzien. Dat wil zeggen dat we in principe weer een geheel nieuwe set iteratie parameters moeten kiezen en we zouden misschien weer helemaal van voren af aan moeten beginnen om een in een of andere zin optimale x_{i+1} te bepalen.

Daarom, alvorens op zoek te gaan naar mogelijk optimale polynomen $p_i \in \Pi_i^1$, vragen we ons eerst af of we het werk verricht om x_i te berekenen, nog nuttig kunnen gebruiken om vervolgens x_{i+1} te berekenen. Als dat niet mogelijk zou blijken, dan zou dat zo'n polynoom methode wel zeer onpractisch maken.
Er geldt

$$\begin{aligned} x_i - x &= p_i(M)(x_0 - x) \quad \text{met } M = P^{-1}A, \quad p_i(0) = 1 \\ &= (I + q_i(M))(x_0 - x), \quad \text{met} \quad q_i(0) = 0 \\ &= x_0 - x + q_i(M)(x_0 - x). \end{aligned} \tag{10.13}$$

Dus

$$\begin{aligned} x_i &= x_0 + q_i(M)M^{-1}M(x_0 - x) \\ &= x_0 + q_i(M)M^{-1}\tilde{r}_0, \quad \text{met} \quad \tilde{r}_0 = P^{-1}(Ax_0 - b). \end{aligned} \tag{10.14}$$

Vanwege $q_i(0) = 0$ is $q_i(M)M^{-1}$ een $(i-1)$-ste graads polynoom in M. In principe kan men dus volstaan met de vectoren $M^j\tilde{r}_0$ uit te rekenen en te bewaren. Als de coëfficiënten van het polynoom $q_i(M)M^{-1}$ gegeven worden door $\beta_j^{(i)}$, dan volgt

$$x_i = x_0 + \sum_{j=0}^{i-1} \beta_j^{(i)} M^j \tilde{r}_0$$

en

$$x_{i+1} = x_0 + \sum_{j=0}^{i} \beta_j^{(i+1)} M^j \tilde{r}_0.$$

Als de coëfficiënten $\beta_j^{(i+1)}$ geen enkele relatie hebben met de $\beta_j^{(i)}$, dan is dat nog steeds een heleboel werk. Het houdt ook een geheugen probleem in, wegens de noodzaak tot opslag van alle vectoren $M^j\tilde{r}_0$. We zullen echter zien dat er geschikte keuzen voor $p_i(x)$ mogelijk zijn waarvoor dat allemaal reuze meevalt.

10.2 De Chebyshev iteratie methode

We nemen aan dat $M = P^{-1}A$ symmetrisch is (dat vergemakkelijkt de analyse aanzienlijk), met eigenwaarden $0 < \mu_1 \leq \mu_2 \leq \ldots, \leq \mu_n$.
Wegens relatie (10.12) geldt

$$\|x_i - x\|_2 \leq \|p_i(M)\|_2 \|x_0 - x\|_2, \tag{10.15}$$

en dus

$$\|p_i(M)\|_2 = \max_j |p_i(\mu_j)| \leq \max_{\mu_1 \leq \mu \leq \mu_n} |p_i(\mu)|. \tag{10.16}$$

We kunnen het zoeken beperken tot een polynoom zodanig dat de maximale waarde op het interval $[\mu_1, \mu_n]$ zo klein mogelijk is. Dat polynoom is bekend, het is het van $[-1, 1]$ naar $[\mu_1, \mu_n]$ getransformeerde Chebyshev polynoom (geschaald zodat de waarde in 0 gelijk is aan 1):

$$\tilde{T}(\mu) = T_i\left(\frac{2\mu - (\mu_n + \mu_1)}{\mu_n - \mu_1}\right) / T_i\left(-\frac{(\mu_n + \mu_1)}{\mu_n - \mu_1}\right), \tag{10.17}$$

waarbij T_i het i-de graads Chebyshev polynoom is.

Als μ_n en μ_1 bekend zijn dan zijn we in principe klaar en wordt het gezochte polynoom gegeven door $\tilde{T}_i(\mu)$. De volgende vraag is nu of we bij de bepaling van x_{i+1} nog nuttig gebruik kunnen maken van x_i en eventueel nog een paar voorgangers. We zullen laten zien dat dat heel goed mogelijk is en dat uiteindelijk een zeer efficiënte methode resulteert.
Er geldt

$$\begin{aligned} x_i - x &= p_i(M)(x_0 - x) \\ x_i &= p_i(M)x_0 + (1 - p_i(M))x. \end{aligned} \tag{10.18}$$

We vullen nu voor p_i het polynoom $\tilde{T}_i$ in, met de volgende notaties

$$\begin{aligned} \tilde{T}_i(M) &= \frac{1}{c_i} T_i(\tilde{M}), \\ \text{met } c_i &= T_i(-\frac{\mu_n + \mu_1}{\mu_n - \mu_1}), \\ \tilde{M} &= \frac{2}{\mu_n - \mu_1} M - \frac{\mu_n + \mu_1}{\mu_n - \mu_1} I \equiv \alpha M - \beta I. \end{aligned} \tag{10.19}$$

Verder gebruiken we de bekende recursie formule voor de Chebyshev polynomen:

$$T_i(\mu) = 2\mu T_{i-1}(\mu) - T_{i-2}(\mu), \tag{10.20}$$

met $T_0(\mu) = 1$, $T_1(\mu) = \mu$.
Dan volgt uit vergelijking (10.17):

$$\begin{aligned} \tilde{T}_i(M) = \frac{T_i(\tilde{M})}{c_i} &= \frac{2\tilde{M}T_{i-1}(\tilde{M}) - T_{i-2}(\tilde{M})}{c_i} \\ &= 2\frac{c_{i-1}}{c_i}\tilde{M}\tilde{T}_{i-1}(M) - \frac{c_{i-2}}{c_i}\tilde{T}_{i-2}(M) \end{aligned} \tag{10.21}$$

Dus gaat (10.18) met $p_i(M) = \tilde{T}_i(M)$ over in

$$\begin{aligned} x_i &= \tilde{T}_i(M)x_0 + (1 - \tilde{T}_i(M))x \\ &= 2\tilde{M}\frac{c_{i-1}}{c_i}\tilde{T}_{i-1}(M)x_0 - \frac{c_{i-2}}{c_i}\tilde{T}_{i-2}(M)x_0 + \\ &\quad (1 - 2\tilde{M}\frac{c_{i-1}}{c_i}\tilde{T}_{i-1}(M) + \frac{c_{i-2}}{c_i}\tilde{T}_{i-2}(M))x \\ &= 2\tilde{M}\frac{c_{i-1}}{c_i}\{x_{i-1} - (1 - \tilde{T}_{i-1}(M))x\} \end{aligned}$$

$$
\begin{aligned}
&-\frac{c_{i-2}}{c_i}\{x_{i-2}-(1-\tilde{T}_{i-2}(M))x\} \\
&+(1-2\tilde{M}\frac{c_{i-1}}{c_i}\tilde{T}_{i-1}(M)+\frac{c_{i-2}}{c_i}\tilde{T}_{i-2}(M))x \\
= \;& 2\tilde{M}\frac{c_{i-1}}{c_i}x_{i-1}-\frac{c_{i-2}}{c_i}x_{i-2}+(x+\frac{c_{i-2}}{c_i}x-2\tilde{M}\frac{c_{i-1}}{c_i}x).
\end{aligned} \tag{10.22}
$$

Er geldt $\tilde{M}=\alpha M-\beta I$, en dat levert voor x_i:

$$
x_i = 2\alpha\frac{c_{i-1}}{c_i}(Mx_{i-1}-Mx)-2\beta\frac{c_{i-1}}{c_i}x_{i-1}-\frac{c_{i-2}}{c_i}x_{i-2}+(2\beta\frac{c_{i-1}}{c_i}+\frac{c_{i-2}}{c_i}+1)x. \tag{10.23}
$$

Dat is nog geen erg prettige uitdrukking, immers de onbekende x komt er nog in voor. Echter wegens

$$
\begin{aligned}
1=\tilde{T}_i(0)=\frac{T_i(-\beta)}{c_i} &= -2\beta\frac{T_{i-1}(-\beta)}{c_i}-\frac{T_{i-2}(-\beta)}{c_i} \\
&= -2\beta\frac{c_{i-1}}{c_i}-\frac{c_{i-2}}{c_i},
\end{aligned} \tag{10.24}
$$

vervalt de coëfficiënt van x in (10.23).
Tenslotte vinden we met $M=P^{-1}A$:

$$
\begin{aligned}
x_i &= 2\alpha\frac{c_{i-1}}{c_i}P^{-1}(Ax_{i-1}-b)-2\beta\frac{c_{i-1}}{c_i}x_{i-1}+(2\beta\frac{c_{i-1}}{c_i}+1)x_{i-2} \\
&= -2\beta\frac{c_{i-1}}{c_i}\{\frac{\alpha}{\beta}P^{-1}r_{i-1}+x_{i-1}-x_{i-2}\}+x_{i-2} \\
&= -2\frac{c_{i-1}}{c_i}\frac{\mu_n+\mu_1}{\mu_n-\mu_1}\{\frac{2}{\mu_n+\mu_1}P^{-1}r_{i-1}+x_{i-1}-x_{i-2}\}+x_{i-2},
\end{aligned} \tag{10.25}
$$

met

$$
c_i=-2\frac{\mu_n+\mu_1}{\mu_n-\mu_1}c_{i-1}-c_{i-2},\quad c_0=1,\quad c_1=-\frac{\mu_n+\mu_1}{\mu_n-\mu_1},
$$

x_0 een willekeurig gekozen startvector, en

$$
x_1=\frac{2}{\mu_n+\mu_1}P^{-1}r_0+x_0.
$$

De aldus geconstrueerde iteratie methode staat bekend onder de naam *Chebyshev semi-iteratieve methode*, geassocieerd met het stelsel $P^{-1}Ax=P^{-1}b$.
Er kleven wel enige practische bezwaren aan deze methode. Allereerst moet de matrix $P^{-1}A$ symmetrisch positief definiet zijn, maar bovendien moeten we de kleinste eigenwaarde μ_1 en de grootste eigenwaarde μ_n van deze matrix kennen. Alvorens hier nader op in te gaan, onderzoeken we eerst het convergentiegedrag van deze methode. Dat zal met name van belang blijken voor de methode die we in de volgende paragraaf gaan beschrijven.

Wegens (10.15) geldt, met $p_i(M)=\tilde{T}_i(M)$, dat

$$
\begin{aligned}
\|x_i-x\|_2 &\le \|p_i(M)\|_2\|x_0-x\|_2 \\
&\le \max_{\mu_1\le\mu\le\mu_n}|\frac{T_i(\alpha\mu-\beta)}{T_i(-\frac{\mu_n+\mu_1}{\mu_n-\mu_1})}|\|x_0-x\|_2 \\
&\le \frac{1}{T_i(\frac{\mu_n+\mu_1}{\mu_n-\mu_1})}\|x_0-x\|_2.
\end{aligned} \tag{10.26}
$$

Chebyshev polynomen kunnen als volgt geschreven worden

$$T_i(x) = \frac{1}{2}[\left(x + \sqrt{x^2-1}\right)^i + \left(x + \sqrt{x^2-1}\right)^{-i}]. \tag{10.27}$$

Dus

$$\begin{aligned} T_i\left(\frac{\mu_n+\mu_1}{\mu_n-\mu_1}\right) &= T_i\left(\frac{1+\mu_1/\mu_n}{1-\mu_1/\mu_n}\right) \geq \frac{1}{2}\left(\frac{1+\mu_1/\mu_n}{1-\mu_1/\mu_n} + \sqrt{(\frac{1+\mu_1/\mu_n}{1-\mu_1/\mu_n})^2-1}\right)^i \\ &= \frac{1}{2}\left(\frac{1+\frac{\mu_1}{\mu_n}+2\sqrt{\frac{\mu_1}{\mu_n}}}{1-\frac{\mu_1}{\mu_n}}\right)^i = \frac{1}{2}\left(\frac{1+\sqrt{\frac{\mu_1}{\mu_n}}}{1-\sqrt{\frac{\mu_1}{\mu_n}}}\right)^i. \end{aligned} \tag{10.28}$$

In combinatie met (10.26) leidt dit tot de volgende bovengrens voor de fout in de i-de iteratie stap:

$$\|x_i - x\|_2 \leq 2(\frac{\sqrt{\frac{\mu_n}{\mu_1}}-1}{\sqrt{\frac{\mu_n}{\mu_1}}+1})^i\|x_0 - x\|_2. \tag{10.29}$$

We kunnen dus concluderen dat er altijd convergentie optreedt als de eigenwaarden positief zijn.

Opgave 24 *Zij $\tilde{\mu}_1$ een schatting voor μ_1 en $\tilde{\mu}_n$ een schatting voor μ_n. Laat zien dat de Chebyshev methode in het algemeen langzamer convergeert, wanneer $0 < \tilde{\mu}_1 < \mu_1$ en $\tilde{\mu}_n > \mu_n$, indien de schattingen gebruikt worden voor de berekening van de iteratie parameters.*

Opgave 25 *Toon aan dat als de Chebyshev iteratie (10.25) wordt uitgevoerd met $\tilde{\mu}_1$, $\tilde{\mu}_n$, er divergentie kan optreden als $\tilde{\mu}_1 > \mu_1$ en $\tilde{\mu}_n < \mu_n$ (Hint: maak gebruik van (10.15), wanneer kan daarin het gelijkteken optreden?).*

Wanneer A symmetrisch is, dan volstaat de keuze $P = I$. In de praktijk kiest men echter vaak een geschikte positief definiete P in de vorm $P = LL^T$. De vergelijking $Ax = b$ wordt dan getransformeerd tot

$$L^{-1}AL^{-T}\tilde{x} = L^{-1}b, \quad \text{met} \quad \tilde{x} = L^T x.$$

We krijgen dus het stelsel $\tilde{A}\tilde{x} = \tilde{b}$ met

$$\tilde{A} = L^{-1}AL^{-T}, \quad \tilde{x} = L^T x \quad \text{en} \quad \tilde{b} = L^{-1}b.$$

Als A symmetrisch positief definiet is, dan is $\tilde{A}$ het ook en men kan het nieuwe stelsel in principe oplossen met de Chebyshev iteratie methode en achteraf x bepalen door x op te lossen uit het stelsel $L^T x = \tilde{x}$. We zullen later nog een methode tegenkomen om voor zekere A dusdanige P te construeren dat het conditiegetal van de (gepreconditioneerde) matrix $\tilde{A}$ flink kleiner is dan dat van A.

De Chebyshev iteratie methode is veel minder populair dan de in de volgende paragraaf beschreven geconjugeerde gradiënten methode. Sommige auteurs (zie [10]) zien evenwel voordelen in een combinatie van deze methoden, met name voor machines waar de berekening van inprodukten kostbaar is. Zo'n combinatie zou ook tot hogere efficiëntie kunnen leiden in bepaalde "parallel computing" omgevingen. Tenslotte zij nog vermeld dat Manteuffel [14] een bruikbare variant van de Chebyshev methode voor het oplossen van een klasse niet-symmetrische stelsels heeft ontworpen. Deze variant omvat ook een elegante methode voor het adaptief schatten van de benodigde iteratie parameters.

10.3 De geconjugeerde gradiënten methode

Ook in deze paragraaf wordt aangenomen dat A symmetrisch positief definiet is.
De vraag rijst nu of er een optimaal polynoom bestaat, zodanig dat $\|x_i - x\|$ minimaal is in een of andere geschikte norm. We zullen voorlopig, ter beperking van het schrijfwerk, P^{-1} weglaten en verder aannemen dat $x_0 = 0$ en dus $r_0 = b - Ax_0 = b$. Dit laatste is ook al geen wezenlijke beperking, immers, als $x_0 \neq 0$, dan geldt met $z \equiv x - x_0$ dat $Az = b - Ax_0 = \tilde{b}$. En voor het stelsel $Az = \tilde{b}$ geldt dan weer $z_0 = 0$ als startvector.

Wegens (10.15) zouden we kunnen proberen om het polynoom $p_i(\mu)$ te vinden waarvoor

$$\max_{j=1,\ldots,n} |p_i(\mu_j)|$$

minimaal is.
Dit levert niets bruikbaars op; enerzijds moeten we nu alle eigenwaarden van A kennen en verder bestaat er ook geen eenvoudige betrekking tussen de opeenvolgende polynomen van verschillende graad.

Voor de hand zou ook kunnen liggen de p_i zo te construeren dat $\|x_i - x\|_2$ minimaal is. Dat leidt voor $i = 1$ tot (zie ook (10.10)):

$$\begin{aligned} x_1 &= \alpha_0 r_0 \quad (= \alpha_0 b) \\ \text{en} \quad \|x - x_1\|_2^2 &= (x - \alpha_0 r_0, x - \alpha_0 r_0) \\ &= (x, x) - 2\alpha_0 (r_0, x) + \alpha_0^2 (r_0, r_0). \end{aligned} \tag{10.30}$$

Voor een minimum moet dus gelden

$$\frac{\partial}{\partial \alpha_0} \|x - x_1\|_2^2 = 0 \Rightarrow \alpha_0 = \frac{(r_0, x)}{(r_0, r_0)},$$

en aangezien α_0 nu van de ons nog onbekende x afhangt, levert dit ook niets bruikbaars op. Aangezien echter $Ax = b$, lijkt het wel de moeite waard het inprodukt iets anders te kiezen: $(\ ,\ )_A$, in plaats van het meer gangbare inprodukt, met $(x, y)_A \equiv (Ax, y)$ (ga zelf na dat dit inderdaad een inprodukt definieert).
Uit dit nieuwe inprodukt definiëren we de norm $\|x - x_1\|_A \equiv (A(x - x_1), x - x_1)$ en nu lukt het minimaliseren van de norm van de iteratie fout wel, immers

$$(A(x - x_1), x - x_1) = (Ax, x) - 2\alpha_0 (r_0, Ax) + \alpha_0^2 (r_0, Ar_0)$$

en voor de minimaliserende α_0 vinden we

$$\alpha_0 = \frac{(r_0, Ax)}{(r_0, r_0)} = \frac{(r_0, r_0)}{(r_0, Ar_0)}.$$

We gaan dit verder uitwerken. Eerst bekijken we (10.11) opnieuw:

$$x_i - x = p_i(A)(x_0 - x) \quad \text{met} \quad p_i(0) = 1, \tag{10.31}$$

en dus

$$\begin{aligned} x_i &= p_i(A)x_0 + (I - p_i(A))x \\ &= p_i(A)x_0 + (I - p_i(A))A^{-1}b \\ &= x_0 + (I - p_i(A))A^{-1}(b - Ax_0) \\ &= x_0 + A^{-1}(I - p_i(A))r_0. \end{aligned} \tag{10.32}$$

Omdat p_i een willekeurig polynoom is uit Π_i^1, is $A^{-1}(I - p_i(A))$ dus een willekeurig polynoom uit Π_{i-1} (merk op dat er nu geen restrictie meer geldt ten aanzien van de constante term). Dus we kunnen x_i ook als volgt schrijven:

$$x_i = q_{i-1}(A)r_0 = a_0 r_0 + a_1 A r_0 + \cdots + a_{i-1} A^{i-1} r_0, \tag{10.33}$$

en we constateren dat x_i in de ruimte zit opgespannen door de vectoren

$$\{r_0, Ar_0, \ldots, A^{i-1}r_0\}.$$

Deze ruimte staat bekend als de *Krylov ruimte* van dimensie i, voortgebracht door A en r_0, genoteerd als

$$K^i(A; r_0).$$

In deze Krylov ruimte moet die x_i gevonden worden, waarvoor $\|x_i - x\|_A$ minimaal is, dus

$$x_i - x \perp_A span\{r_0, Ar_0, \ldots, A^{i-1}r_0\}, \tag{10.34}$$

waarbij $x \perp_A y$ betekent dat $(x, y)_A = 0$.

Uit (10.34) volgt

$$b - Ax_i = r_i \perp K^i(A; r_0). \tag{10.35}$$

Stelling 17 *De $r_j = b - Ax_j$, passend bij de x_j waarvoor $\|x - x_j\|_A$ minimaal is, vormen een orthogonale basis voor $K^i(A; r_0)$, ($j = 0, 1, \ldots, i-1$).*

Bewijs: Uit (10.35) volgt $r_1 \perp r_0$. Vanwege (10.33) geldt

$$r_1 = b - Ax_1 = r_0 - A(a_0 r_0) = r_0 - a_0 A r_0.$$

Dus r_0 en r_1 vormen een orthogonale basis voor $K^2(A; r_0)$.
Zij nu $r_0, \ldots, r_{j-1}$ een orthogonale basis voor $K^j(A; r_0)$. Wegens (10.35) geldt dat $r_j \perp K^j(A; r_0)$ en via (10.33):

$$r_j = b - Ax_j = r_0 - Aq_{j-1}(A)r_0 \in K^{j+1}(A; r_0).$$

Dus $r_0, \ldots, r_j$ vormen een orthogonale basis voor $K^{j+1}(A; r_0)$. □

Opgave 26 *In het bovenstaande bewijs werd impliciet aangenomen dat $a_{i-1} \neq 0$ (immers anders is q_{i-1} hooguit een $(i-2)$-de graads polynoom). Laat zien dat als $a_{i-1} = 0$, dat dan $r_i = 0$, dat wil zeggen $x_i = x$ en dan zijn we klaar met itereren.*

Stelling 18 *Voor de residuen r_j uit stelling 17 geldt een 3-terms recursie*

$$\alpha_{i+1} r_{i+1} = A r_i - \beta_i r_i - \gamma_i r_{i-1}. \tag{10.36}$$

Bewijs: $r_1 \in K^2(A; r_0) \Rightarrow \alpha_1 r_1 = A r_0 - \beta_0 r_0$

$$r_2 \in K^3(A; r_0) \Rightarrow r_2 \in \{r_0, r_1, A^2 r_0\} = \{r_0, r_1, A r_1\}$$

en dus $\alpha_2 r_2 = A r_1 - \beta_1 r_1 - \gamma_1 r_0$.
We passen nu volledige inductie toe.

$$r_{j-1} \in K^j(A; r_0) \Rightarrow A r_{j-1} \in K^{j+1}(A; r_0) = span\{r_0, \ldots, r_j\}.$$

Daaruit volgt

$$\alpha_j r_j = A r_{j-1} - \beta_j r_{j-1} - \gamma_j r_{j-2} - \delta_j r_{j-3} - \ldots$$

Vanwege de inductie veronderstelling geldt:

$$(A r_{j-1}, r_k) = (r_{j-1}, A r_k) = (r_{j-1}, \alpha_{k+1} r_{k+1} + \beta_k r_{k-1} + \gamma_k r_{k-2}) = 0,$$

voor $k + 1 < j - 1$ (orthogonaliteit der r_j's).
Hieruit volgt de 3-terms recursie voor stap $j + 1$. □

We zijn nu in staat om op eenvoudige wijze, uitgaande van r_0, een basis te genereren voor de ruimte waarin de gezochte x_i ligt. De coëfficiënten β_j en γ_j volgen uit de orthogonaliteitseis en α_{j+1} volgt uit de eis dat het voortbrengend polynoom $p_{j+1} \in \Pi^1_{j+1}$. Dit leidt tot de relatie $\alpha_{j+1} + \beta_j + \gamma_j = 0$ (Ga dit zelf na).
Als we R_i schrijven voor de matrix met als kolommen $r_0, r_1, ..., r_{i-1}$, dan kan relatie (10.36) in matrixvorm herschreven worden in:

$$A R_i = R_i T_i + \alpha_i r_i e^T_{i+1}.$$

Ga zelf na dat T_i een tridiagonale matrix is van orde i, met als kolom $\gamma_j, \beta_j, \alpha_{j+1}$. We zoeken nu een x_i in $K^i(A; r_0)$, dus $x_i = R_i y$, zodanig dat $A x_i - b$ loodrecht staat op $K^i(A; r_0)$. Hieruit vinden we dat y moet voldoen aan

$$\begin{aligned} R_i^T A R_i y &= R_i^T b \\ \text{dus}\;\; R_i^T R_i T_i y &= \|r_0\|_2^2 e_1 \\ \text{en}\;\; T_i y &= e_1. \end{aligned} \tag{10.37}$$

Het bepalen van x_i is in principe een eenvoudig uitvoerbare zaak, echter de methode is alleen practisch uitvoerbaar indien met het werk gedaan voor de bepaling van x_{i-1} slechts relatief weinig extra gedaan hoeft te worden om x_i te bepalen. We zullen aangeven hoe dat in zijn werk gaat.
Voor

$$T_i = \begin{pmatrix} \beta_0 & \gamma_1 & & & \\ \alpha_1 & \beta_1 & \ddots & & \\ & \alpha_2 & \ddots & \ddots & \\ & & \ddots & \ddots & \gamma_{i-1} \\ & & & \alpha_{i-1} & \beta_{i-1} \end{pmatrix},$$

schrijven we $T_i = L_i U_i$, met

$$L_i = \begin{pmatrix} 1 & & & & \\ f_1 & 1 & & & \\ & f_2 & 1 & & \\ & & \ddots & \ddots & \\ & & & f_{i-1} & 1 \end{pmatrix} \quad \text{en} \quad U_i = \begin{pmatrix} d_0 & g_1 & & & \\ & d_1 & g_2 & & \\ & & d_2 & g_3 & \\ & & & \ddots & \ddots \\ & & & & d_{i-1} \end{pmatrix}.$$

Dan volgt

$$x_i = R_i y = R_i T_i^{-1} e_1 = (R_i U_i^{-1})(L_i^{-1} e_1).$$

Ga na dat de i-de kolom van $(R_i U_i^{-1})$ eenvoudig uit de $(i-1)$-ste kolom kan worden afgeleid:

$$d_{i-1}(R_i U_i)_i^{-1} = r_{i-1} - g_{i-1}(R_i U_i)_{i-1},$$

en dat de eerste $i-1$ kolommen van $(R_i U_i)$ gelijk zijn aan die van $(R_{i-1} U_{i-1})$. De matrix RU^{-1} wordt dus telkens met een kolom uitgebreid.
Ga ook zelf na dat de vector $\vec{k}_i = L_i^{-1} e_1$, met $\vec{k}_i = (k_0, \ldots, k_{i-1})^T$ uit $\vec{k}_{i-1}$ door aan deze vector de coördinaat k_{i-1} toe te voegen met $k_{i-1} = -f_{i-1} k_{i-2}$
Men kan vervolgens ook zelf verifiëren dat

$$x_i = x_{i-1} + k_{i-1}(R_i U_i^{-1})_i.$$

We constateren dat, nadat de recursie voor de r_i's tot stand is gekomen, we steeds alleen het laatste residu hoeven te bewaren voor het bijwerken van de matrix RU^{-1} (waarvan ook steeds slechts de laatste kolom bewaard hoeft te blijven). Tenslotte hoeft ook van de vector $\vec{k}$ alleen maar de laatste coördinaat bewaard te blijven om x_{i-1} te kunnen bijwerken tot x_i. Zo komt de zogenaamde *geconjugeerde gradiënten methode* tot stand.
Men kan inzien, dat vanwege $R^T AR = R^T RT = \tilde{T}$, de matrix T via een diagonaal transformatie over te voeren is in een symmetrische positief definiete matrix, zodat de LU-ontbinding mogelijk is (tenzij het orthogonalisatie proces afbreekt, maar dan zijn we ook al bij de oplossing gearriveerd).

Men kan het geconjugeerde gradiënten proces (CG) nog iets meer in elkaar schuiven, door ook de recursie voor de residuen met die voor de RU^{-1}-kolommen te combineren. Zo ontstaat het volgende populaire schema voor de CG-methode:

$x_0 = 0$, $r_0 = b$, $p_0 = r_0$
voor $i = 0, 1, 2, \ldots,$ tot convergentie
  $\tilde{\alpha}_i = (r_i, r_i)/(p_i, Ap_i)$
  $x_{i+1} = x_i + \tilde{\alpha}_i p_i$
  $r_{i+1} = r_i - \tilde{\alpha}_i Ap_i$
  $\tilde{\beta}_i = (r_{i+1}, r_{i+1})/(r_i, r_i)$
  $p_{i+1} = r_{i+1} + \tilde{\beta}_i p_i$

De x_i's die door het CG schema gegenereerd worden convergeren naar de oplossing x. Voor de richtings vectoren p_i, waarmee x_i wordt bijgewerkt, geldt de eigenschap

$$(p_i, Ap_j) = 0 \quad \text{voor} \quad i \neq j.$$

Hieraan ontleent het CG-schema zijn naam.

Opgave 27 *Ga na dat het CG schema inderdaad identiek is met het eerder beschreven projectie proces.*

Indien $x_0 \neq 0$, dan passen we CG toe op $Az = \tilde{b}$, met $\tilde{b} = b - Ax_0$. Voor dit proces geldt $z_0 = 0$, $\tilde{r}_0 = \tilde{b} = b - Ax_0 = r_0$ en $\tilde{p}_0 = r_0$.
Verder geldt $z_{i+1} = z_i + \tilde{\alpha}_i p_i$ en dus $x_{i+1} = x_i + \tilde{\alpha}_i p_i$, etc. We mogen dus de eerste regel van het algoritme vervangen door
"$x_0 =$ willekeurige startvector, $r_0 = b - Ax_0$, $p_0 = r_0$".

De CG methode minimaliseert $\|x_i - x\|_A$ over alle mogelijke iteratie methoden van de vorm

$$x_i - x = p_i(A)(x_0 - x), \quad \text{met} \quad p_i(0) = 1.$$

Bij de Chebyshev methode probeerden we $\|x_i - x\|_2$ zo goed mogelijk te minimaliseren.
Er geldt voor CG blijkbaar

$$\|x_i - x\|_A \leq \|\tilde{p}_i(A)(x_0 - x)\|_A,$$

waarbij $\tilde{p}_i(A)$ een willekeurig polynoom uit Π_i^1 is (bijvoorbeeld het Chebyshev polynoom).
Hierdoor kunnen we de fout bij CG afschatten:

Opgave 28 *Laat zien dat voor de CG iteranden x_i geldt*

$$\|x_i - x\|_A \leq 2 \left(\frac{\sqrt{\frac{\mu_n}{\mu_1}} - 1}{\sqrt{\frac{\mu_n}{\mu_1}} + 1} \right)^i \|x_0 - x\|_A.$$

Hint: gebruik de ongelijkheid $\|x_i - x\|_A \leq \|\tilde{T}_i(A)\|_2 \|x_0 - x\|_A$.

Meetkundige interpretatie van CG

Zij $\tilde{x}$ de oplossing van $Ax = b$. Voor $n = 2$ zijn de "hoogtelijnen": $(A(x - \tilde{x}), x - \tilde{x}) = c$ gegeven door de ellipsen in de figuur.

$(p_1, Ap_0) = 0$ en p_1 wijst in de richting van het minimum in de ruimte opgespannen door r_0

en r_1, in dit geval dus rechtstreeks naar $\tilde{x}$.

10.4 Het convergentiegedrag van de CG methode

De CG methode convergeert met minstens een factor $\frac{\sqrt{c}-1}{\sqrt{c}+1}$ per slag, waarbij c het conditiegetal van A is. Maar uit de constructie van de methode volgt ook dat de methode in hooguit n iteratieslagen bij de oplossing moet zijn, aangezien $r_0, \ldots, r_{n-1}$ de gehele R^n opspannen. In feite is CG dus een eindige methode. In de praktijk heeft men niet veel aan die wetenschap. Het is immers niet erg prettig om bij een stelsel van orde 100000 te weten dat men na evenzovele slagen de oplossing gevonden heeft. Men hoopt doorgaans met iteratieve methoden in veel minder slagen een redelijke benadering voor de oplossing gevonden te hebben. Bovendien blijkt in de praktijk dat, door de invloed van afrondfouten, de na n stappen gevonden oplossing ook nog vrij ver van de exacte oplossing verwijderd kan liggen.

Voorbeeld: (ontleend aan [9])
$Ax = b$, $n = 40$ en $b = (1, 0, \ldots, 0)^T$.

$$A = \begin{pmatrix} 5 & -4 & 1 & & & & & & \\ -4 & 6 & -4 & 1 & & & & & \\ 1 & -4 & 6 & -4 & 1 & & & & \\ & 1 & -4 & 6 & -4 & 1 & & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & & 1 & -4 & 6 & -4 & 1 \\ & & & & & 1 & -4 & 6 & -4 \\ & & & & & & 1 & -4 & 5 \end{pmatrix}.$$

Opgave 29 *Laat zien dat A symmetrisch positief definiet is en dat de eigenwaarden van A gegeven zijn door*
$\lambda_k \sin^4 \frac{k\pi}{82}$, $k = 1, 2, \ldots, 40$.
Hint: Schrijf A in de vorm $A = B^2$, met B tridiagonaal.

Voor het conditiegetal van A geldt

$$\kappa_2 \approx \frac{16}{16(\frac{\pi}{82})^4} = (\frac{82}{\pi})^4$$

en hieruit volgt, met de bovengrens uit opgave 28, een convergentie factor van ongeveer 0.997. Op grond hiervan kijken we er in het begin van het CG proces natuurlijk niet zo vreemd van op dat er maar weinig opgeschoten wordt (het proces lijkt dan nog op een Chebyshev proces). Maar, na 40 iteratie stappen zou de exacte oplossing toch gevonden moeten zijn.

Zie echter wat er gebeurt, rekenend in 11 decimalen floating point arithmetiek.

Dit soort ervaringen leidden er toe dat de CG methode, na haar introductie in 1952 door Hestenes en Stiefel [11], al snel in een ongunstig daglicht kwam te staan. In 1972 liet Reid [19] zien dat de methode, opgevat als iteratieve methode, voor een klasse van relevante problemen heel aantrekkelijk kan zijn. Deze problemen komen bijvoorbeeld voort uit de discretisatie van (elliptische) partiële differentiaal vergelijkingen (zie ook het college numerieke wiskunde C).
Nadien is geleidelijk aan duidelijk geworden dat het convergentie gedrag van CG vrijwel uitsluitend afhangt van de ligging van de buitenste eigenwaarden van A en in het geheel niet van de orde van de matrix.
Voor deze problemen kan men de convergentie snelheid heel wel afschatten met de bovengrens uit opgave 28 en dat leidt dan vaak tot een aantal iteratieslagen dat beduidend lager is dan de orde van A. De vraag rijst nu ook hoe goed die bovengrens het convergentie gedrag voorspelt. Om die bovengrens te verkrijgen werden de CG iteratiefouten simpelweg afgeschat via de Chebyshev polynomen. We krijgen op die manier dus nooit te zien dat CG wel eens een heel stuk efficiënter zou kunnen zijn dan de Chebyshev methode.

Bij een constante foutreductie zou men de volgende grafiek verwachten

Echter wat men experimenteel vaak waarneemt is meer van de vorm:

Dat wil zeggen CG gaat na een aantal iteratieslagen beduidend sneller convergeren. Het lijkt wel of in het iteratieproces het oorspronkelijke conditiegetal gaandeweg vervangen wordt door een (veel) gunstiger effectief conditiegetal.

Voorbeeld: A is nu de gediscretiseerde Poisson operator over een equidistant rooster van 30 bij 30 punten. Voor $n = 900$ worden de eigenwaarden gegeven door

$$\lambda_{k,\ell} = 4 - 2\cos\frac{\pi k}{31} - 2\cos\frac{\pi \ell}{31},$$
$$\text{met} \quad 1 \leq k, \ell \leq 30.$$

Met de bovengrens uit opgave 28 kan men narekenen dat er 280 iteratieslagen nodig zijn, wil men er zeker van zijn dat de initiële fout met een factor 10^{-12} gereduceerd wordt (in A-norm). Bij daadwerkelijke berekening blijkt evenwel dat bij een rechterlid $b = (1, 1, 1, \ldots, 1)^T$ en $x_0 = 0$ er al na 120 iteratieslagen zo'n reductie is opgetreden. De bovengrens geeft het ware convergentie gedrag nu dus niet zo best weer.

Laat het orthogonale eigensysteem van A gegeven zijn door $\{\lambda_j, v_j\}_{j=1,\ldots,n}$, dat wil zeggen $Av_j = \lambda_j v_j$, met $0 < \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$. Zij verder $x_0 - x = \sum \gamma_j v_j$, dan geldt

$$x_i - x = \sum_{j=1}^{n} p_i(\lambda_j)\gamma_j v_j, \tag{10.38}$$

met p_i het CG-polynoom.
Merk op dat zodra $p_i(\lambda) = 0$ voor alle *verschillende* λ_j, dan geldt $x_i = x$. Dus als de componenten

γ_j horen bij m verschillende eigenwaarden, dan is het proces na m stappen klaar. Voor de norm van de fout geldt

$$\|x_i - x\|_A = \sum_{k=1}^{m} \lambda_{j_k} p_i^2(\lambda_{j_k}) \gamma_{j_k}^2, \tag{10.39}$$

waarbij de sommatie alleen over de verschillende eigenwaarden, waarvoor de bijbehorende gewichten $\gamma_j \neq 0$ zijn gaat. Daarbij is γ_{j_k} gelijk aan de kwadraatsom der gewichten horend bij (de eventueel meervoudig voorkomende γ_{j_k}.

Opgave 30 *Zij voor een gegeven stelsel $Ax = b$, met A symmetrisch positief definiet, α en β gedefiniëerd door*

$$\alpha \equiv \min\{\lambda_j | \gamma_j \neq 0\}$$

$$\beta \equiv \max\{\lambda_j | \gamma_j \neq 0\}.$$

Laat zien dat de CG convergentie factor uit opgave 28 vervangen kan worden door

$$\frac{\sqrt{\frac{\beta}{\alpha}} - 1}{\sqrt{\frac{\beta}{\alpha}} + 1}.$$

Het getal β/α noemen we het **effectieve conditiegetal** *van A.*

De eigenwaarden van de tridiagonaal matrix T_i zullen ons van pas komen bij het begrijpen van het convergentiegedrag van CG. Deze eigenwaarden heten de *Ritz waarden* van A met betrekking tot $K^i(A; r_0)$. De vectoren $R_i y_i$, met y_i een eigenvector van T_i, heten de *Ritz vectoren* van A met betrekking tot $K^i(A; r_0)$. Zonder bewijs vermelden we een aantal opvallende eigenschappen van deze Ritz waarden. Voor details zie [17] en literatuur over het aan CG nauw verwante *Lanczos* proces.
Eigenschappen der Ritz waarden:

- De Ritz waarden zijn benaderingen voor de eigenwaarden van A, in die zin dat de j-de Ritz waarde van T_i, voor toenemende i, monotoon naar λ_j convergeert. Hierbij moet worden opgemerkt dat we alleen die (verschillende) eigenwaarden van A mee laten tellen, waarvoor geldt $(r_0, v_j) \neq 0$. Eenzelfde verhaal geldt voor de Ritz vectoren.

- De convergentie snelheid van een Ritz waarde naar "zijn" eigenwaarde hangt af van de afstand van die eigenwaarde tot de rest van het spectrum, relatief t.o.v. de lengte van dat spectrum. Hoe groter die verhouding hoe sneller de convergentie.

- De buitenste Ritz waarden convergeren in het algemeen het snelst (naar de buitenste "in de startmix vertegenwoordigde" eigenwaarden).

Als een Ritz vector een goede benadering vormt voor en eigenvector v_k van A, dan houdt dat dus in dat v_k "voor een belangrijk deel" in $K^i(A; r_0)$ ligt (denk aan de definitie van Ritz vector). Dat houdt in dat $x_i - x$ min of meer A-loodrecht op v_k staat, ofwel $(x_i, v_k) \approx (x, v_k)$, ofwel de k-de coördinaat van x_i ligt in de buurt van de k-de coördinaat van de gezochte oplossing x (t.o.v. de eigenvector basis). Dit heuristische verhaal geeft de achtergrond voor de volgende empirische observatie in [4]:

"De buitenste eigenwaarden van A in het bijzonder, worden op den duur in het CG proces goed benaderd door Ritz waarden. Zodra dat het geval is gedraagt het CG proces zich verder alsof deze eigenwaarden er niet meer zijn."
De achterliggende gedachte is dat het effectieve conditiegetal vanaf dat moment gereduceerd is en dat is een aanzet tot een verklaring voor de waargenomen versnelde convergentie van CG, de zogenaamde *superlineaire convergentie.*

In de volgende stelling wordt dit beter onderbouwd. We introduceren eerst een vergelijkings proces. Dit CG proces start met $\bar{x}_0$ die zo gekozen is dat $x - \bar{x}_0$ precies de projectie is van $x - x_i$ op v_2 tot en met v_n. Dat wil zeggen dat het vergelijkingsproces ontstaat door het CG proces na i stappen te herstarten nadat de coördinaat van x_i t.o.v. v_1 vervangen is door de coördinaat van x t.o.v. v_1. Voor dit proces kunnen we dus convergentie verwachten met een effectief conditiegetal λ_n/λ_2 in plaats van λ_n/λ_1 (zie opgave 30).

Stelling 19 *Zij x_i de i-de iterand van CG voor $Ax = b$. Zij, voor zekere i, de vector $\bar{x}_j$ de j-de iterand van het vergelijkings CG proces dat start met $\bar{x}_0$ (zie hiervoor).*
Dan geldt voor alle j:

$$\|x - x_{i+j}\|_A \leq F_i \|x - \bar{x}_j\|_A \leq F_i \frac{\|x - \bar{x}_j\|_A}{\|x - \bar{x}_0\|_A} \|x - x_i\|_A, \tag{10.40}$$

met

$$F_i \equiv \frac{\theta_1^{(i)}}{\lambda_1} \max_{k \geq 2} |\frac{\lambda_k - \lambda_1}{\lambda_k - \theta_1^{(i)}}|,$$

waarbij $\theta_1^{(i)}$ de bij λ_1 passende Ritz waarde is bij de i-de stap van het CG-proces.

Voor bewijs en verdere analyse zie [21].

Bibliografie

[1] R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, PA, 1994

[2] B.L.Buzbee, G.H.Golub and C.W.Nielsen, *On direct methods for solving Poisson's equations*, SIAM J.Numer.Anal., **7**, 627-656, 1970

[3] A.K.Cline, C.B.Moler, G.W.Stewart and J.H.Wilkinson, *An estimate for the condition number of a matrix*, SIAM J.Num.Anal., **16**, p368-375, 1979

[4] P.Concus, G.H.Golub and D.P.O'Leary, *A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations*, in: J.R.Bunch and D.J.Rose, *Sparse Matrix Computations*, Academic Press, New York, 1976

[5] J.J.Dongarra, J,R.Bunch, et al., *LINPACK users guide*, SIAM Publications, Philadelphia, 1978

[6] J.J.Dongarra, I.S.Duff, D.S.Sorenson and H.A.van der Vorst, *Solving Linear Systems on Vector and Shared Memory Computers*, SIAM Publications, Philadelphia, 1990

[7] I.S.Duff and J.K.Reid, *A comparison of sparsity orderings for obtaining a pivotal sequence in Gaussian elimination*, J.Inst.Math. Appl., **17**, p281-291, 1974

[8] G.E.Forsythe and C.B.Moler, *Computer solution of linear algebraic systems*, Prentice-Hall, Englewood Cliffs, NJ, 1967

[9] T.Ginsburg, *The conjugate gradient method*, in: J.H.Wilkinson and C.H.Reinsch (eds.), *Linear Algebra*, Springer Verlag, Berlin, 1971

[10] G.H.Golub and C.F.Van Loan, *Matrix Computations*, North Oxford Academic, Oxford, 1983

[11] M.Hestenes and E.Stiefel, *Methods of conjugate gradients for solving linear systems*, J.Research NBS, **49**, p409-436, 1952

[12] D.R.Kincaid, T.C.Oppe, J.R.Respess and D.M.Young, *ITPACKV 2C User's Guide*, Report CNA-191, Center for Numerical Analysis, Univ. of Texas, Austin, Tex., 1984

[13] C.L.Lawson and R.J.Hanson, *Solving Least Squares Problems*, Prentice Hall, Englewood Cliffs, NJ., 1974

[14] T.A.Manteuffel, *The Tchebychev iteration for nonsymmetric linear systems*, Numer.Math.,**28**, p307-327, 1977

[15] H.M.Markowitz, *The elimination form of the inverse and its application to linear programming*, Management Science, **3**, p255-269, 1957

[16] J.A.Meijerink and H.A.van der Vorst, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Math.of Comp., **31**, p148-162, 1977

[17] B.N.Parlett, *The Symmetric Eigenvalue Problem*, Prentice Hall, Englewood Cliffs, NJ., 1980

[18] J.K.Reid, *Solution of linear systems of equations: direct methods (general)*, in:V.A.Barker, ed., *Sparse Matrix Techniques*, Lecture Notes in Mathematics, **572**, Springer Verlag, Berlin, 1977

[19] J.K.Reid, *On the method of conjugate gradients for the solution of large sparse systems of linear equations*, Proc. Conf. on Large Sparse Sets of Linear Equations, Academic Press, New York, 1971

[20] R.W.H.Sargent and A.W.Westerberg, *"Speed-up" in chemical engineering design*, Trans.Inst.Chem.Engnrs., **42**, p190-197, 1964

[21] A.van der Sluis and H.A.van der Vorst, *The rate of convergence of Conjugate Gradients*, Numer.Math., **48**, p543-560, 1986

[22] B.T.Smith, J.N.Boyle et al., *Matrix eigensystem routines - EISPACK Guide*, 2-nd edition, Springer Verlag, Berlin, 1976

[23] G.Strang, *Linear Algebra and its Applications*, Academic Press, New York, 1976

[24] R.E.Tarjan, *Depth first search and linear graph algorithms*, SIAM J. Comput., **1**, p146-160, 1972

[25] R.S.Varga, *Matrix Iterative Analysis*, Prentice Hall, Englewood Cliffs, NJ, 1962

[26] J.H.Wilkinson, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1985