Wetenschappelijk Rekenen - Opdracht

Tobias Ocula

April 2025

1 Givens rotaties

1.1 Inleiding

Een Givens rotatie is in de Lineaire Algebra een lineaire afbeelding waarbij deze afbeelding een vector roteert in een vlak gespannen door twee coördinaatassen. De matrix van een Givens rotatie G wordt gedefiniëerd als een matrix met op de diagonaal 1, behalve op positie (i, i) en (j, j), waar deze $\cos(\theta)$ heeft staan, op positie (i, j) $\sin(\theta)$ en op positie (j, i) $-\sin(\theta)$ en de rest nullen. Als bijvoorbeeld $i = 2, j = 3$ en G is een 3×3 matrix, dan is dit een gewone rotatie in het yz-vlak:

$$G(3, 2, \theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{pmatrix}$$

Deze matrix beschrijt dus een rotatie van θ radialen in het (i, j) vlak.

1.2 Methode

De bedoeling is om Givens rotaties te gebruiken om nullen te vormen onder de hoofddiagonaal van de gegeven matrix A, zodat deze uiteindelijk een bovendriehoeksmatrix wordt. Dit gaat men waarmaken door de rotaties zo te kiezen dat de coördinaten van de kolomvectoren van A onder de hoofddiagonaal 0 worden.
Hoe gaat men nu precies de matrices G construeren? Men weet dat indien men een vector vermenigvuldigt met deze matrix, deze operatie enkel de twee componenten gaat veranderen die in het rotatievlak liggen (de andere componenten blijven dus ongewijzigd). Men kan dus een waarde onder de diagonaal kiezen die men nul gaat maken, en men kiest als tweede waarde die mag veranderen bijvoorbeeld de waarde op de hoofddiagonaal. Zo kan men waarde per waarde de kolomvector afgaan en al zijn waarden onder de hoofddiagonaal op nul zetten.
We krijgen dus een stelsel waarbij wanneer we de eerste kolomvector $\vec{v}$ vermenigvuldigen met de gekozen rotatiematrix G_1, we een nieuwe kolomvector $\vec{v'}$ uitkomen die de oude kolomvector in A zal vervangen. Wat moeten nu de waarden zijn van de componenten die veranderen van de nieuwe vector? Eén component moet sowieso nul worden (dit was de bedoeling), en de andere moeten we zo

kiezen dat de norm van de kolomvector dezelfde blijft. Als we dus de componenten v_i en v_j noteren voor de kolomvector, dan moeten de nieuwe componenten respectievelijk $\sqrt{v_i^2 + v_j^2}$ en 0 zijn. Omdat enkel maar twee componenten in de bekomen vector zullen veranderen, krijgen we het volgende stelsel:

$$\begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} v_i \\ v_j \end{pmatrix} = \begin{pmatrix} \sqrt{v_i^2 + v_j^2} \\ 0 \end{pmatrix}$$

Als we dit nu herschrijven naar onze onbekenden c en s krijgen we

$$\begin{pmatrix} v_i & -v_j \\ v_i & v_j \end{pmatrix} \begin{pmatrix} c \\ s \end{pmatrix} = \begin{pmatrix} \sqrt{v_i^2 + v_j^2} \\ 0 \end{pmatrix}$$

Als we dit oplossen, krijgen we dat $c = -v_i/r$ en $s = v_j/r$, met $r = \sqrt{v_i^2 + v_j^2}$. Met andere woorden, r is precies de lengte van de schuine zijde van de driehoek gevormd door deze componenten van de kolomvector, en s en c zijn op een teken na respectievelijk de sinus en cosinus van een van de hoeken van de driehoek. Men kan nu dus de matrix G_1 construeren. Om de nieuwe matrix te bekomen met de nul op de gewenste plek, vermenigvuldigt men deze matrix met A om een nieuwe matrix $A_2 = G_1 A$ te bekomen.
Nu herhalen we dit procedure voor de andere waarden in dezelfde kolomvector, dus v_i blijft dezelfde (waarde op hoofddiagonaal) en v_j schuift één plaats naar onderen. We zoeken een gepaste matrix G_2 op dezelfde manier als eerst om v'_j nul te maken. Men bekomt de matrix $A_3 = G_2 A_2 = G_2 G_1 A$.
Vervolgens blijft men dit proces herhalen totdat men aan de laatste kolomvector komt (met deze hoeft men niets te doen uiteraard).
Uiteindelijk bekomt men voor A een bovendriehoeksmatrix, noteer deze R. De bekomen orthogonale matrix Q is het product van de getransponeerde van alle gekozen matrices G_i. Om dit in te zien, ziet men dat men bij de eerste stap een matrix $A_1 = G_1 A$ bekomt, bij de tweede stap $A_2 = G_2 A_1 = G_2 G_1 A$, enzovoort. Uiteindelijk bekomt men dus de matrix $R = G_k G_{k-1} ... G_2 G_1 A$. Als men de gelijkheid $A = QR$ wilt hebben, moet de matirx Q dus gelijk zijn aan $(G_k G_{k-1} ... G_2 G_1)^T = G_1^T G_2^T ... G_k^T$. Dit is inderdaad een orthogonale matrix, want het product van een aantal orthogonale matrices is opnieuw een orthogonale matrix.
We maken nu nog een optimalisatie aan ons algoritme: uiteraard moeten we niet de hele matrixvermenigvuldiging uitvoeren van A en G, omdat de matrices G_i enkel één bepaalde rij van A verandert (namelijk rij i en j). Dit wordt een lineaire combinatie van de i-de en j-de rij (de coëfficiënten zijn c en s). Hetzelfde voor de vermenigvuldiging van $G_{i-1} G_i$: hier verandert enkel de i-de en j-de kolom. De kolommen worden ook lineaire combinaties van de i-de en j-de kolom. De exacte berekening kan men terugvinden in de implementatie.

1.3 Floating point operaties

In deze sectie zullen we het aantal floating point operaties berekenen van deze methode. Hiervoor heb ik pseudocode (in dit geval geen echte pseudocode) geschreven die dient om het aantal numerieke operaties te onderzoeken (later volgt uiteraard een implementatie in MATLAB).

```
def QR_givens_rotation(A: np.ndarray):
    n = A.shape[0]
    R = A.copy()
    Q = np.identity(n)
    for i in range(n-1):
        for j in range(i+1,n):
            vi = R[i,i]
            vj = R[j,i]

            r = np.hypot(vi, vj)
            c = vi / r
            s = -vj / r

            R_COPY = R.copy()
            Q_COPY = Q.copy()

            R[i,:] = R_COPY[i,:]*c - R_COPY[j,:]*s
            R[j,:] = R_COPY[i,:]*s + R_COPY[j,:]*c
            Q[:,i] = Q_COPY[:,i]*c - Q_COPY[:,j]*s
            Q[:,j] = Q_COPY[:,i]*s + Q_COPY[:,j]*c

    return R, Q
```

We berekenen het aantal FLOPS die nodig zijn om het algoritme te kunnen uitvoeren.
We nemen om te beginnen een vaste i en j en bekijken de code binnenin de twee for-loops. Dit zijn hier het aantal FLOPS:
-r: 2 kwadraten + één optelling + vierkantswortel geven 4 FLOPS
-c: 1 deling geeft 1 FLOP
-s: 1 deling geeft 1 FLOP

-$R[i,:]$: voor elke waarde in de rij hebben we 2 vermenigvuldigingen + 1 optelling (aftrekking in dit geval) en we hebben n rijen, dus dit geeft $3n$ FLOPS
-$R[j,:]$, $Q[:,i]$ en $Q[:,j]$: identiek hetzelfde, dus in het totaal $4 * 3n = 12n$ FLOPS
Voor een bepaalde i en j-index hebben we dus $4 + 1 + 1 + 12n = 6 + 12n$ FLOPS. Voor een bepaalde i en waarbij de j loopt van $i + 1$ tot n hebben we dus $\sum_{j=i+1}^{n}(6+12n) = (6+12n)(n-i)$. Voor i lopende van 1 tot $n-1$ hebben we $\sum_{i=1}^{n-1}(6+12n)(n-i) = (6+12n)\sum_{i=1}^{n-1}(n-i) = (6+12n)(n(n-1)-\sum_{i=1}^{n-1} i) = (6+12n)(n^2-n-n(n-1)/2) = 6n^3-3n^2-3n$.
Dit is dus een algoritme van kubische complexiteit.

2 Householder reflexies

2.1 Inleiding

Een Householder reflexie is in de Lineaire Algebra een lineaire afbeelding die een vector spiegelt ten opzichte van een (hyper)vlak. Deze afbeelding wordt gedefinieerd als

$$H_{\vec{u}}(\vec{x}) = \vec{x} - 2\langle \vec{x}, \vec{u}\rangle \vec{u}$$

Met $\vec{u}$ de eenheidsvector die loodrecht staat op het hypervlak.
Men kan deze ook definiëren voor een niet-eenheidsvector $\vec{u}$:

$$H_{\vec{u}}(\vec{x}) = \vec{x} - 2\frac{\langle \vec{x}, \vec{u}\rangle}{\langle \vec{u}, \vec{u}\rangle}\vec{u}$$

De matrix van deze lineaire afbeelding is

$$H_{\vec{u}} = I - \frac{2}{\langle \vec{u}, \vec{u}\rangle}\vec{u} \cdot \vec{u}^T$$

Deze afbeelding heeft als bijzondere eigenschap dat deze zowel unitair als hermitisch is. Bijgevolg is $H = H^{-1} = H^T$.

2.2 Methode

De bedoeling is om deze lineaire afbeelding te gebruiken om de kolomvectoren van de gegeven matrix A om te zetten zodat deze onder de hoofddiagonaal enkel nullen bevatten. Maar hoe kiezen we de matrix H nu zodat $H\vec{u} = \vec{v}$, met $\vec{u}$ de te veranderen vector naar de vector $\vec{v}$ die onder de hoofddiagonaal nullen bevat?
Omdat de vectoren $\vec{u}$ en $\vec{v}$ dezelfde norm moeten hebben (we passen een orthogonale afbeelding toe) kiezen we deze zodat $\|\vec{u}\| = \|\vec{v}\|$ ($\vec{v}$ zal dus bestaan uit allemaal nullen behalve op de eerste positie, daar staat $\pm\|\vec{u}\|$). Men kan nu bewijzen dat indien men $\vec{x} = \vec{u} - \vec{v}$ neemt, dat dan $H_{\vec{x}}(\vec{u}) = \vec{v}$. Men kiest dus de matrix $H_1 = I - 2/\langle \vec{x}, \vec{x}\rangle \vec{x} \cdot \vec{x}^T$. Dit gebruiken we om de eerste kolomvector volledig te laten bestaan uit nullen behalve positie $(1,1)$.

Beschouw nu de matrix H_1A, dit is een matrix met als eerste kolom enkel nullen behalve de eerste positie. Beschouw nu de $(n-1)\times(n-1)$ matrix B die bestaat uit de matrix die men bekomt door de 1ste rij en kolom uit H_1A te schrappen. Beschouw ook de vector $\vec{u_1}$ als opnieuw de eerste kolomvector van B. We willen opnieuw de vector $\vec{v_1}$ bekomen die uit allemaal nullen bestaat behalve de eerste positie (opnieuw zoals bij de eerste stap). We passen nu opnieuw het algoritme toe op B door de matrix $\hat{H}_2$ te definiëren als $\hat{H}_2 = I - 2/\langle\vec{x_1},\vec{x_1}\rangle\vec{x_1}\cdot\vec{x_1}^T$ met $\vec{x_1} = \vec{u_1} - \vec{v_1}$. Dan hebben we opnieuw dat $\hat{H}_2\vec{u_1} = \vec{v_1}$. Definiëer nu de matrix

$$H_2 = \begin{pmatrix} 1 & \vec{0}^T \\ \vec{0} & \hat{H}_2 \end{pmatrix}$$

Dan krijgen we de matrix H_2H_1A waarbij de eerste en tweede kolom uit enkel nullen onder de hoofddiagonaal bestaat.
Dit proces herhaalt men voor alle kolommen van A zodat men de matrix $H_nH_{n-1}...H_2H_1A = R$ bekomt. De gezochte matrix Q moet dan gelijk zijn aan $(H_nH_{n-1}...H_2H_1)^T = H_1^TH_2^T...H_{n-1}^TH_n^T = H_1H_2...H_{n-1}H_n$ (omdat telkens $H = H^T = H^{-1}$).
We zullen nu een wat efficiëntere en betere methode bespreken die in de praktijk gebruikt wordt. Het is niet efficiënt om telkens weer het product $H := HR$ en $Q := QH^T$ te berekenen. In de praktijk zullen we eerder het volgende doen:
-Initialiseer de matrix $R = A$.
-Zij $\vec{u}$ de eerste kolom van de blokmatrix in R van de huidige iteratiestap en $\vec{v}$ de gewilde vector (met allemaal nullen behalve op de eerste positie $\|\vec{u}\|$). Omdat de normen van $\vec{v}$ en $\vec{u}$ dezelfde moeten zijn, zijn we vrij om het teken van de eerste component van $\vec{v}$ te kiezen. We kiezen dit nu in functie van het teken van de eerste component van $\vec{u}$, namelijk hetzelfde teken van deze component van $\vec{u}$. Dit kiezen we zo omdat we willen vermijden dat we twee bijna identieke getallen met hetzelfde teken van elkaar aftrekken, omdat dit niet numeriek stabiel is. Zij nu $\vec{x} = \vec{u} + \vec{v}$ (nu is dit een plusteken omdat de tekens van $\vec{u}$ en $\vec{v}$ gelijk zijn). Dan moeten we R gelijkstellen aan

$$H_{\vec{x}}R = (I - \frac{2}{\langle\vec{x},\vec{x}\rangle}\vec{x}\cdot\vec{x}^T)R = R - \frac{2}{\langle\vec{x},\vec{x}\rangle}\vec{x}\cdot\vec{x}^TR$$

Met andere woorden we moeten

$$\frac{2}{\langle\vec{x},\vec{x}\rangle}\vec{x}\cdot\vec{x}^TR$$

van R aftrekken. Maar omdat $H_{\vec{x}}$ enkel werkt op het onderste rechtse deel van de matrix R (omdat linksboven H gelijk is aan de eenheidsmatrix en omdat R op de eerste paar kolommen onder de diagonaal enkel nullen heeft staan), hoeven we deze dus enkel uit te voeren op het onderste deel van R.
-We gaan nu Q updaten door dit gelijk te stellen aan

$$QH_{\vec{x}} = Q(I - \frac{2}{\langle\vec{x},\vec{x}\rangle}\vec{x}\cdot\vec{x}^T) = Q - \frac{2}{\langle\vec{x},\vec{x}\rangle}Q\vec{x}\cdot\vec{x}^T$$

Omdat nu ook H enkel linksonder bestaat uit iets dat niet de eenheidsmatrix is, zal deze bewerking enkel invloed hebben op de laatste $n-i$ kolommen van Q. Bijgevolg moeten we enkel rekening houden met de vermenigvuldiging van deze kolommen.
-Itereer naar de volgende kolom van R en ga terug naar de eerste stap, totdat men aan de laatste kolom komt.

2.3 Floating point operaties

In deze sectie zullen we het aantal floating point operaties berekenen van deze methode. Hiervoor heb ik opnieuw wat (geen echte) pseudocode geschreven in Python:

```python
def QR_household(A: np.ndarray):
    n = A.shape[0]
    R = A.copy()
    Q = np.identity(n)
    for i in range(n-1):
        u = R[i:,i]
        norm_u = np.linalg.norm(u)
        v = np.zeros(n-i)
        v[0] = np.copysign(norm_u, u[0])
        x = u + v
        x /= np.linalg.norm(x)

        R[i:, i:] -= 2 * np.outer(x, x @ R[i:, i:])
        Q[:, i:] -= 2 * np.outer(Q[:, i:] @ x, x)

    return Q, R
```

We berekenen het aantal FLOPS voor dit algoritme.
We beginnen binnenin de loop:
-Voor de norm hebben we $n-i$ vermenigvuldigingen, $n-i-1$ optellingen en 1 vierkantswortel, dit geeft $2(n-i)$ FLOPS
-Voor de berekening van $\vec{x}$ hebben we 1 optelling nodig, dit geeft 1 FLOP.
-Voor de norm van x hebben we opnieuw $2(n-i)$ operaties nodig.
-Om de matrix R te updaten, is er om te beginnen 1 aftrekking, 1 product en 1 deling. Het product $\vec{x}^T R$ heeft per kolom van R $n-i$ vermenigvuldigingen en $n-i-1$ optellingen nodig, dus $2(n-i)-1$ operaties, en dit $n-i$ keer, dus

$2(n-i)^2-(n-i)$ operaties. Vervolgens moeten we links vermenigvuldigen met $\vec{x}$. Dit geeft $(n-i)^2$ vermenigvuldigingen.
-Om de matrix Q te updaten, zijn er opnieuw om te beginnen 1 optelling, 1 deling en 1 vermenigvuldiging nodig. Dan voor het product $Q\vec{x}$ zijn er $(n-i)(2(n-i)-1) = 2(n-i)^2-(n-i)$ operaties nodig. Vervolgens voor de vermenigvuldiging met $\vec{x}^T$ zijn er dan zoals voorheen $(n-i)^2$ berekeningen nodig.
Voor een bepaalde index i nemen we dus de som

$$\begin{aligned}
&\sum_{i=1}^{n-1}(2(n-i)+1+3+2(n-i)^2-(n-i)\\
&+(n-i)^2+3+2(n-i)^2-(n-i)+(n-i)^2\\
&=\sum_{i=1}^{n-1}(7+6(n-i)^2) = 7(n-1)+6\sum_{i=1}^{n-1}(n^2-2ni+i^2)\\
&=7n-7+6n^2(n-1)-12n^2(n-1)/2+6(n-1)n(2n-1)/6\\
&=7n-7+6n^3-6n^2-6n^3+6n^2+2n^3-3n^2+n\\
&=2n^3-3n^2+8n-7
\end{aligned}$$

Het aantal benodigde operaties is dus van kubische complexiteit.

2.3.1 Vergelijking FLOPS met Givens rotaties en Gramm-Schmidt

We zien dat de Householde relfecties methode voor de QR-ontbinding een lagere complexiteit heeft dan de Givens-rotaties ($2n^3$ versus $6n^3$). Het Gramm-Schmidt algoritme heeft dan weer een complexiteit van ongeveer $2n^3$. Bijgevolg is de Givens rotaties methode iets complexer.

3 Toepassing: implementatie op Hilbertmatrix

We passen nu de Gramm-Schmidt, de Givens rotaties en de Householder reflecties toe om de QR-ontbinding te berekenen op de Hilbertmatrix met posities $H(i,j) = \frac{1}{i+j-1}$. In het bestand 'hilbertmatrix_QR.m' kan u het algoritme hiervoor terugvinden. Hier passen we de drie QR-ontbindingen toe voor dimensies lopende van 5 tot 50 met stapgrootte 5. We willen nu bepalen hoe goed (accuraat) het QR-ontbindingsalgoritme is voor de ontbinding van de Hilbertmatrix. We bepalen dus voor elke methode, van de $n \times n$ matrix H de normen van $H-QR$ en Q^TQ-I. Hoe kleiner de norm, hoe beter. Hieronder het resultaat van de beide matrixnormen voor $n = 50$:

```
dimensie: 50
norm H-QR (GS):
    8.3880

norm QTQ-I (GS):
   11.1151

norm H-QR (GR):
   1.9832e-15

norm QTQ-I (GR):
   2.5641e-15

norm H-QR (HR):
   7.0060e-16

norm QTQ-I (HR):
   3.2024e-15
```

Men kan duidelijk zien dat voor grote matrices, de Givens rotaties en de Householder reflecties methode accurater zijn. Voor de Gramm-Schmidt methode ziet men dat beide normen (zowel $\|H-QR\|$ en $\|Q^TQ-I\|$) een stuk groter zijn dan voor beide anderen.
We doen nu hetzelfde, maar nemen voor de Gramm-Schmidt methode nu de versie waarbij we NIET reorthogonaliseren. In de vorige versie hebben we dit wel gedaan. Om dit te implementeren, vervangen we simpelweg de functie 'QR_Gramm_Schmidt' door de functie 'QRontbinding' in het bestand van de functie. We verwachten nu dat de fout (veel) groter zal zijn. Hierbeneden het resultaat voor $n = 50$:

```
dimensie: 50
norm H-QR (GS):
   4.5111e-17

norm QTQ-I (GS):
   41.9620

norm H-QR (GR):
   1.9832e-15

norm QTQ-I (GR):
   2.5641e-15

norm H-QR (HR):
   7.0060e-16

norm QTQ-I (HR):
   3.2024e-15
```

Opmerkelijk genoeg is de norm $\|H - QR\|$ nu kleiner, maar is de norm $\|Q^TQ - I\|$ veel groter geworden. Men kan nu besluiten dat het reorthogonaliseren wel degelijk een effect heeft gehad, alhoewel dit een impact heeft gehad op de nauwkeurigheid van de matrix QR.

4 Toepassing: stelsel oplossen

We zullen nu de drie versies van het QR-ontbindingsalgoritme gebruiken om het stelsel $Hx = b$ met $b = (1, 1, ..., 1)^T$ op te lossen. Om dit te doen gebruiken we het algoritme om $H = QR$ te berekenen. We krijgen dus $QRx = b$ of $Rx = Q^Tb$. Om het product Q^Tb te berekenen, moeten we dus gewoon voor elke kolom de som nemen van alle rijen van Q^T (dus de kolommen van Q). Vervolgens krijgen we het resultaat $x = R^{-1}Q^Tb$. Maar omdat R een bovendriehoeksmatrix is, hoeven we de inverse van R niet te berekenen, maar kunnen we gebruik maken van backwards-substitution. Voor de implementatie, zie het bestand 'backSub.m'. In de volgende afbeelding wordt het resultaat getoond voor de drie methoden (met de drie vectoren $X = (x_1, x_2, x_3)$ na elkaar):

```
X =

   1.0e+06 *

   -0.0000   -0.0000   -0.0000
    0.0010    0.0010    0.0010
   -0.0238   -0.0238   -0.0238
    0.2402    0.2402    0.2402
   -1.2612   -1.2611   -1.2612
    3.7836    3.7833    3.7836
   -6.7264   -6.7259   -6.7264
    7.0010    7.0005    7.0010
   -3.9381   -3.9378   -3.9381
    0.9237    0.9237    0.9237
```

We zien dat de drie algoritmen nagenoeg dezelfde oplossing geven (met een paar hele kleine verschillen in de tweede vector). Indien we nu geen reorthogonalisatie meer gebruiken voor het Gramm-Schmidt algoritme krijgen we:

```
X =

   1.0e+06 *

    0.0004   -0.0000   -0.0000
   -0.0192    0.0010    0.0010
    0.2280   -0.0238   -0.0238
   -1.1435    0.2402    0.2402
    2.8874   -1.2611   -1.2612
   -3.8633    3.7833    3.7836
    2.6126   -6.7259   -6.7264
   -0.6995    7.0005    7.0010
   -0.0025   -3.9378   -3.9381
   -0.0005    0.9237    0.9237
```

Men kan dus zien dat de eerste vector (die van het Gramm-Schmidt algoritme) nu een (heel) ander resultaat geeft! Reorthogonalisatie heeft dus wel degelijk een grote impact op de numerieke stabiliteit van het algoritme.

5 Bronnen

https://en.wikipedia.org/wiki/QR_decomposition
https://en.wikipedia.org/wiki/Householder_transformation
https://en.wikipedia.org/wiki/Givens_rotation
https://math.stackexchange.com/questions/501018/what-is-the-operation-count-for-qr-factorization-using-householder-transformatio
https://math.stackexchange.com/questions/4432126/computing-cost-for-givens-rotation-to-yield-qr
Cursus Wetenschappelijk Rekenen