Numerieke Wiskunde
Bachelor Informatica-Wiskunde

Academiejaar 2023-2024

1 LU-factorisatie

In de les hebben jullie de LU-factorisatie gezien om van een gegeven matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ een decompositie te maken zodat

$$\mathbf{A} = \mathbf{LU}$$

met $\mathbf{L}$ een onderdriehoeksmatrix en $\mathbf{U}$ een bovendriehoeksmatrix. Dit kan ons helpen bij het oplossen van het stelsel

$$\mathbf{Ax} = \mathbf{b}.$$

In het eerste deel van deze opgave bepalen we de LU-factorisatie van een matrix. Vervolgens bekijken we twee toepassingen van LU-factorisaties. In het laatste deel bekijken we kort de stabiliteit van het algoritme. We duiden een element (i, j) van een matrix aan met een onderindex. Zo wordt bijvoorbeeld het (i, j)-de element van een matrix $\mathbf{A}$ aangeduid met $\mathbf{A}_{ij}$. Gebruik deze notatie in je verslag.

Opdracht 1. We zoeken matrices $\mathbf{L}$ en $\mathbf{U}$ zodanig dat $\mathbf{A} = \mathbf{LU}$ met $\mathbf{L}$ een onderdriehoeksmatrix met $\mathbf{L}_{ii} = 1$ voor $i = 1, \ldots, n$ en $\mathbf{U}$ een bovendriehoeksmatrix. Toon aan dat volgende formules gelden voor $i = 1, \ldots, n$

$$\begin{cases} \mathbf{U}_{ik} & = \mathbf{A}_{ik} - \sum_{j=1}^{i-1} \mathbf{L}_{ij}\mathbf{U}_{jk} \qquad k = i, \ldots, n, \\ \mathbf{L}_{ki} & = \dfrac{\mathbf{A}_{ki} - \sum_{j=1}^{i-1} \mathbf{L}_{kj}\mathbf{U}_{ji}}{\mathbf{U}_{ii}} \quad k = i+1, \ldots, n. \end{cases} \tag{1}$$

Opdracht 2. Gebruik deze formules (1), en schrijf een Matlabfunctie die de LU-decompositie opstelt voor een gegeven matrix $\mathbf{A}$. Je dient dus geen rijpivoteringen te implementeren. For-lussen kunnen minder snel zijn in Matlab [1] [2], gebruik daarom matrix-vectoroperaties waar mogelijk. Deze functie kan geschreven worden met maar één for-lus die itereert over de i-index. Je functie dient te beginnen met de signatuur

```
[L,U] = lu_decomp(A)
% A: een inverteerbare matrix
% L: een onderdriehoeksmatrix met diagonaalelementen gelijk aan 1
% U: een bovendriehoeksmatrix
```

[1] Zie bv. het antwoord van Rob Patro op https://www.quora.com/Why-are-%E2%80%98for%E2%80%99-loops-so-slow-in-MATLAB

[2] De geïnteresseerde student kan zichzelf hiervan overtuigen door het eerste experiment "Vectorizing Code for General Computing" eens uit te voeren, te vinden op https://www.mathworks.com/help/matlab/matlab_prog/vectorization.html

Opdracht 3. We willen zeker zijn dat onze functie correct werd geïmplementeerd. Voer volgende test uit. Genereer een random $\mathbf{L}$ en $\mathbf{U}$, en bereken $\mathbf{A} = \mathbf{LU}$. Pas `lu_decomp` hierop toe en verifieer of je de oorspronkelijke matrices terugkrijgt.

Eénmaal we een LU-factorisatie van de matrix $\mathbf{A}$ hebben is het stelsel $\mathbf{Ax} = \mathbf{b}$ eenvoudig op te lossen door achtereenvolgens twee driehoeksstelsels op de lossen door middel van substitutie. We implementeren twee algoritmes. Deze kunnen respectievelijk stelsels met boven- of onderdriehoeksmatrices oplossen.

Opdracht 4. Het implementeren van driehoekige stelsels staat beschreven in je handboek. Bekijk Algoritme 5.1 en wijzig dit algoritme zodat het onderdriehoeksstelsels kan oplossen en implementeer dit. Leg uit wat je veranderd hebt. Implementeer ook Algoritme 5.1 zelf om stelsels $\mathbf{Uy} = \mathbf{b}$ op te lossen. Gebruik hiervoor de volgende signaturen:

```
y = solve_Lb( L, b)
% L: een inverteerbare onderdriehoeksmatrix
% b: het rechterlid
% y: de oplossing van het stelsel Ly = b
```

en

```
y = solve_Ub( U, b)
% U: een inverteerbare bovendriehoeksmatrix
% b: het rechterlid
% y: de oplossing van het stelsel Uy = b
```

Opdracht 5. Test de correctheid van de functies `solve_Lb` en `solve_Ub`. Tel vervolgens het aantal bewerkingen: enerzijds het aantal optellingen/aftrekkingen en anderzijds het aantal vermenigvuldigingen/delingen die er in totaal worden uitgevoerd in functie van n.

2 Spaarse stelsels en LU-factorisatie

Een spaarse matrix is er één die weinig niet-nul elementen bevat. Er is geen strikte definitie over welke proportie elementen dan al dan niet nul mogen zijn, maar vaak stellen we dat het aantal niet-nul elementen grofweg lineair is in het aantal rijen of kolommen (in tegenstelling tot kwadratisch voor een generieke matrix). Spaarse matrices zijn van groot belang in verschillende wetenschappelijke en technische toepassingen, zoals lineaire algebra, numerieke analyse, optimalisatie, en grafentheorie. Ze worden bijvoorbeeld vaak gebruikt om complexe netwerken, zoals sociale netwerken of transportnetwerken, te modelleren, waarbij de relaties tussen de elementen meestal schaars zijn. Idealiter kunnen we de spaarsheid van een matrix uitbuiten om efficiëntere algoritmes op te stellen. Niet alleen omdat we dan minder bewerkingen moeten uitvoeren vanwege de vele nullen, maar ook omdat we minder opslagcapaciteit nodig hebben: het volstaat immers de niet-nul elementen op te slaan. Het volgt dan dat niet-opgeslagen elementen nul zijn. We kunnen Matlab informeren dat we zo'n elementsgewijze opslag wensen door de commando's `sparse`, `spdiags` en `speye`. Dit zorgt voor een efficiënter gebruik van geheugen (nul-elementen worden niet expliciet opgeslagen). Doe dit voor alle spaarse matrices in het vervolg van dit practicum!

Het spreekt voor zich dat als we numerieke methodes willen toepassen op een spaarse matrix, we dit spaars karakter vaak willen behouden (indien niet zouden we plots veel meer elementen moeten opslaan!). Passen we bijvoorbeeld de LU-factorisatie toe op een spaarse matrix $\mathbf{A}$, dan zouden we willen dat de matrices $\mathbf{L}$ en $\mathbf{U}$ nog steeds spaars zijn. Volgende voorbeelden laten echter zien dat het soms moeilijk te voorspellen is hoe spaars de decompositie zal zijn.

Opdracht 6. Bepaal in Matlab een LU-factorisatie $\mathbf{A}_1 = \mathbf{L}_1\mathbf{U}_1$ en $\mathbf{A}_2 = \mathbf{L}_2\mathbf{U}_2$ met $\mathbf{A}_1, \mathbf{A}_2 \in \mathbb{R}^{n\times n}$ gegeven door

$$\mathbf{A}_1 = \begin{bmatrix} 1.1 & 0.01 & 0.01 & \dots & 0.01 \\ 0.01 & 1.1 & & & \\ 0.01 & & \ddots & & \\ \vdots & & & \ddots & \\ 0.01 & & & & 1.1 \end{bmatrix} \quad \text{en} \quad \mathbf{A}_2 = \begin{bmatrix} 1.1 & & & & 0.01 \\ & 1.1 & & & 0.01 \\ & & \ddots & & \vdots \\ & & & \ddots & 0.01 \\ 0.01 & 0.01 & \dots & 0.01 & 1.1 \end{bmatrix}. \quad (2)$$

voor $n = 5$ met behulp van je code `lu_decomp`. Print deze matrices in je verslag. Wat valt je op met betrekking tot het aantal nullen in de decompositie van $\mathbf{A}_1$ en $\mathbf{A}_2$? (Je mag aannemen dat dit ook geldt voor $n > 5$.) Je hoeft geen verklaring te geven.
Opmerking: gebruik `format long` zeker eens om je matrices te bekijken.

We tonen ook even aan dat we de spaarsheid van een LU-factorisatie kunnen uitbuiten. Zo zal uiteindelijk substitutie voor een niet-spaarse LU-factorisatie meer tijd vergen dan substitutie voor een factorisatie die wel spaars is. Dit doen we door een aantal stelsels op te lossen met $\mathbf{L}_1$, $\mathbf{L}_2$, $\mathbf{U}_1$ en $\mathbf{U}_2$. We passen hiertoe de in Opdracht 4 geïmplementeerde algoritmes aan zodat deze op meer efficiënte wijze onze spaarse stelsels kunnen oplossen.

Opdracht 7. Als alles correct is verlopen, resulteerden de LU-factorisatie van $\mathbf{A}_1$ en $\mathbf{A}_2$ in enerzijds een koppel niet-spaarse matrices $\mathbf{L}_1, \mathbf{U}_1$ en anderzijds een koppel spaarse matrices $\mathbf{L}_2, \mathbf{U}_2$. Gebruik het feit dat je weet welke elementen in de spaarse factorisatie van $\mathbf{A}_2$ nul zijn, om je algoritme `solve_Ub` resp. `solve_Lb` efficiënter te maken en noem dit `solve_Ub_special` resp. `solve_Lb_special` (cfr. Opdracht 6). Gebruik hiervoor de volgende signaturen. Gebruik ook zeker `sparse` om je matrices te definiëren.

```
y = solve_Lb_special( L, b)
% L: een onderdriehoeksmatrix die de bovenstaande
%    spaarse structuur heeft
% b: het rechterlid
% y: de oplossing van het stelsel Ly = b
```

en

```
y = solve_Ub_special( U, b)
% U: een bovendriehoeksmatrix die de bovenstaande
%    spaarse structuur heeft
% b: het rechterlid
% y: de oplossing van het stelsel Uy = b
```

Opdracht 8. Tel het aantal bewerkingen in `solve_Lb_special` en `solve_Ub_special`. Vergelijk dit met je bevindingen in Opdracht 5 voor `solve_Lb` en `solve_Ub`.

Opdracht 9. Gebruik nu je implementaties uit de vorige opdrachten om de stelsels $\mathbf{L}_i\mathbf{x} = \mathbf{b}, i = 1, 2$ en $\mathbf{U}_i\mathbf{x} = \mathbf{b}, i = 1, 2$ uit te rekenen met $\mathbf{b} = [1, \dots, 1]^T$ en $n = 1000, 1500, 2000, \dots, 7000$. I.e. gebruik `solve_Ub_special`/ `solve_Lb_special` voor het spaarse koppel, en `solve_Ub`/ `solve_Lb` voor het niet-spaarse koppel. Omdat de berekeningen ietwat kunnen aanslepen is het goed georganiseerd te werk te gaan.

- Bereken eerst binnen een for-lus de LU-decomposities van A_1 voor de verschillende n. Sla deze dan op in twee Matlab arrays (met accolades in plaats van vierkante haakjes).

```
all_L1{n} = L1;
all_U1{n} = U1;
```

Het is goed de data ook op te slaan in een `.mat`-bestand. Dan moet je de decomposities maar één keer berekenen en heb je deze verder ter beschikking door deze gewoon opnieuw in te laden. Opslaan kan gewoon via rechtermuisklik op de gewenste data in je Matlab Workspace.

- Los nu de verschillende stelsels op met `solve_Ub`/ `solve_Lb`. Voer timings uit via de commando's `tic` en `toc`. Neem een gemiddelde over 20 runs. Laat per n de eerste timing weg, want die is vaak veel hoger [3].
- Zijn het de timings die je verwacht te hebben? Maak een figuur met de timings en gebruik deze om je resultaten te verklaren. Maak een schatting van de orde op basis van je experimenten. Leg uit hoe je dit doet.

Vervolgens doen we hetzelfde voor A_2. Vergeet niet dat je hier de functies `solve_Ub_special`/ `solve_Lb_special` dient te gebruiken.
Tip: Timings kunnen wel wat variabel zijn. Als het gedrag van je grafiek niet éénduidig genoeg is kan je altijd het aantal runs verhogen, en even niks anders doen op je computer.

Zoals vermeld willen we enerzijds een factorisatie $\mathbf{A} = \mathbf{LU}$, maar anderzijds willen we dat $\mathbf{L}$ en $\mathbf{U}$ spaars blijven. We zagen dat dit niet altijd mogelijk is. Een idee om dit toch te forceren zit vervat in de *incomplete LU-factorisatie*. Zoals de naam doet vermoeden, gaan we een fout toelaten op $\mathbf{L}$ en $\mathbf{U}$, dit wil zeggen $\mathbf{A} \approx \mathbf{LU}$, maar in ruil bewaren we het spaarse karakter van $\mathbf{L}$ en $\mathbf{U}$. De matrices $\mathbf{L}$ en $\mathbf{U}$ blijven respectievelijk onder- en bovendriehoeksmatrices maar nu met de extra voorwaarde dat als $\mathbf{A}_{ij} = 0$, dan geldt ook dat $\mathbf{U}_{ij} = 0$ en $\mathbf{L}_{ij} = 0$.

Dit doen we als volgt. Vlak voordat je een waarde toekent aan $\mathbf{L}_{ij}$ of $\mathbf{U}_{ij}$, verifiëren we eerst of $\mathbf{A}_{ij} = 0$. Enkel als $\mathbf{A}_{ij} \neq 0$, mag je een waarde verschillend van nul toekennen aan $\mathbf{L}_{ij}$ of $\mathbf{U}_{ij}$. Let op, het is niet toegelaten om eerst een volledige LU-factorisatie van $\mathbf{A}$ op te stellen en nadien de elementen die nul dienen te zijn, nul te maken. Dit geeft immers een ander (en fout!) resultaat.

Opdracht 10. Pas het algoritme `lu_decomp` dat je hebt opgesteld in Opgave 2 aan en voer dus de bovenstaande aanpassingen door. Het bekomen algoritme noem je `incompl_lu_decomp`. Gebruik volgende signatuur.

```
[L,U] = incompl_lu_decomp(A)
% A: een spaarse matrix
% L: een spaarse onderdriehoeksmatrix met diagonaalelementen
%    gelijk aan 1
% U: een spaarse bovendriehoeksmatrix
```

Hierbij zijn `A`, `L` en `U` spaarse matrices in Matlab.

We zullen dit algoritme gebruiken in het volgende onderdeel.

3 Preconditionering

In deze sectie geven we een strategie om grote, spaarse stelsels $\mathbf{Ax} = \mathbf{b}$ op te lossen. We zullen het in essentie hebben over:

1. Iteratieve methodes: Dit zijn methodes waarbij we starten vanuit een initiële waarde (in dit geval een vector), die we dan tijdens elke iteratie proberen te verbeteren tot we ons voldoende dicht bij een effectieve oplossing bevinden. We gebruiken dit om een oplossing voor het spaarse stelsel $\mathbf{Ax} = \mathbf{b}$ te vinden.

2. Preconditionering: We proberen de efficiëntie en robuustheid van de iteratieve methode te verbeteren met behulp van preconditionering. Dit wil zeggen dat we het oorspronkelijk lineair systeem omvormen tot een nieuw lineair systeem met dezelfde oplossing, maar dat sneller kan worden opgelost

[3] zie ook op deze link

door de gekozen iteratieve solver. In het bijzonder zal het conditiegetal van de matrix van het nieuwe stelsel kleiner zijn dan het conditiegetal van de matrix van het oorspronkelijke stelsel.

Opdracht 11. Waarom is het beter dat de matrix in het stelsel een laag conditiegetal heeft? Verklaar dit vanuit de cursus.

Preconditionering kan op verschillende manieren. Stel $\mathbf{M}_1, \mathbf{M}_2$ twee inverteerbare matrices met dezelfde dimensies als $\mathbf{A}$, dan kunnen we in plaats van $\mathbf{Ax} = \mathbf{b}$ even goed één van de volgende stelsels oplossen

$$\mathbf{M}_1^{-1}\mathbf{A}\mathbf{M}_2^{-1}\mathbf{y} = \mathbf{M}_1^{-1}\mathbf{b}, \quad \mathbf{M}_2\mathbf{x} = \mathbf{y} \qquad \text{(Links-rechtse preconditionering)} \tag{3}$$

$$\mathbf{M}_1^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}_1^{-1}\mathbf{b} \qquad \text{(Linkse preconditionering)}$$

$$\mathbf{A}\mathbf{M}_1^{-1}\mathbf{y} = \mathbf{b}, \quad \mathbf{M}_1\mathbf{x} = \mathbf{y} \qquad \text{(Rechtse preconditionering)} \tag{4}$$

Opdracht 12. Toon aan dat de vector $\mathbf{x}$ als oplossing van (3) ook de oplossing is van het stelsel $\mathbf{Ax} = \mathbf{b}$.

We merken op dat we de inversen $\mathbf{M}_1^{-1}$ en $\mathbf{M}_2^{-1}$ nooit expliciet zullen uitrekenen, maar dat we in plaats daarvan stelsels oplossen (berekening van $\mathbf{z} = \mathbf{M}_1^{-1}\mathbf{u}$ is hetzelfde als het stelsel $\mathbf{M}_1\mathbf{z} = \mathbf{u}$ oplossen). De focus ligt in dit practicum enkel op rechtse preconditionering (4). Wat zou nu een goede keuze kunnen zijn voor $\mathbf{M}_1$? We maken twee relevante opmerkingen:

- Op het eerste zicht lijkt het alsof het omvormen van het oorspronkelijke stelsel juist meer werk vergt. Je moet immers extra stelsels oplossen. Daarom zal het belangrijk zijn dat de stelsels met $\mathbf{M}_1$ goedkoop (lees: spaars!) op te lossen zijn.
- Als we voor $\mathbf{M}_1$ de LU-factorisatie van $\mathbf{A}$ (dus $\mathbf{M}_1 = \mathbf{LU}$ met $\mathbf{L}, \mathbf{U}$ de LU-factorisatie van $\mathbf{A}$) gebruiken, dan is dat bijna een ideaal scenario. Immers stelsels met boven- en onderdriehoeksmatrices zijn eenvoudiger op te lossen. De snelheid van het oplossen van een stelsel met driehoeksmatrices hangt af van hoe spaars/ijl de matrices zijn. I.e. hoe meer nullen aanwezig zijn in de matrices, hoe goedkoper het is om tot een oplossing te komen.

Overlezen we dit, dan merken we op dat we reeds een aanpak bespraken die aan beide eisen voldoet: de incomplete LU-factorisatie!

We beschouwen een iteratieve methode om stelsels op te lossen. Hier wordt specifiek de *gmres*-methode gebruikt. Dit algoritme mogen jullie grotendeels zien als een 'black box'. Hetgeen we er van moeten weten is dat dit een iteratieve methode is om een stelsel op te lossen. We implementeren deze methode niet zelf, maar gebruiken de implementatie die in Matlab voor handen is.

Opdracht 13. We passen de gmres-methode toe op de stelsels $\mathbf{Ax} = \mathbf{b}$, met $\mathbf{A}$ de matrix in de bestanden `matrix_B1.mat` respectievelijk `matrix_B2.mat`, en $\mathbf{b} = [1, 1, \ldots, 1]^T$. Matrices inladen in de workspace, doe je met `load('matrix_B1.mat')` (analoog voor het andere bestand). Bepaal eerst een incomplete LU-factorisatie $\mathbf{L}, \mathbf{U}$ en los dan het stelsel $\mathbf{Ax} = \mathbf{b}$ op aan de hand van de volgende commando's

- Eerst gebruiken we geen preconditioner, maar passen gmres zuiver toe op $\mathbf{Ax} = \mathbf{b}$:

```
[x0,~,~,~,resvec0] = gmres(A,b)
```

- Vervolgens lossen we het stelsel op met een rechtse preconditioner (4) met $\mathbf{M}_1 = \mathbf{LU}$.

```
[y,~,~,~,resvec1] = gmres(@(x) A*( solve_Ub( U, solve_Lb(L,x))),b);
x1 = solve_Ub( U, solve_Lb(L,y));
```

Lees zeker even de info-pagina van de functie gmres voor een goed begrip van bovenstaande lijnen.

De vectoren $\mathbf{x}_0, \mathbf{x}_1$ zijn slechts benaderingen van de oplossingen. Je zal merken dat bij één van de twee matrices, de preconditioner ervoor zal zorgen dat de solver convergeert en bij de andere zal je zien dat het geen effect heeft. Verklaar dit.
Maak ook een figuur die de residuen van de opeenvolgende iteraties weergeeft. Deze vind je in de vector `resvec`. Als laatste geef je hier een schatting voor de relatieve voorwaartse fout. Leg kort uit hoe je dit berekent.

4 Ad hoc LU-factorisatie

ɹǝiИ elke methode dient algemeen te zijn. Wanneer je vaak éénzelfde stelsel dient op te lossen met een verschillend rechterlid kan het nuttig zijn het oplossen van dit specifieke probleem te optimaliseren. Je kan dan bijvoorbeeld de LU-factorisatie van de beschouwde matrices op voorhand berekenen en ergens opslaan. In deze sectie lossen we stelsels op met de matrix $\mathbf{A}_1$ uit Opdracht 6 via de LU-factorisatie van $\mathbf{A}_2$. De matrix $\mathbf{A}_1$ is gelinkt met de matrix $\mathbf{A}_2$, er zijn immers permutatiematrices $\mathbf{P}$ en $\mathbf{Q}$ zodat

$$\mathbf{P}\mathbf{A}_1\mathbf{Q} = \mathbf{A}_2.$$

We geven je cadeau dat $\mathbf{P}$ van de vorm

$$\mathbf{P} = \begin{bmatrix} 0 & 0 & 0 & \dots & 1 \\ \vdots & \vdots & \vdots & \iddots & \vdots \\ 0 & 0 & 1 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 1 & 0 & 0 & \dots & 0 \end{bmatrix}.$$

is.

Opdracht 14. Bepaal wat de matrix $\mathbf{Q}$ moet zijn voor algemene n en toon aan. Je kan testen doen in Matlab maar dit volstaat niet als bewijs.

Jullie zagen in de cursus dat LU-factorisaties kunnen gebruikt worden om stelsels goedkoper op te lossen. We willen het stelsel $\mathbf{A}_1\mathbf{x} = \mathbf{b}$ oplossen. Als je eerst een LU-decompositie $\mathbf{L}_1\mathbf{U}_1$ van $\mathbf{A}_1$ maakt, dan kan je het stelsel oplossen door volgende twee stelsels met driehoeksmatrices op te lossen:

$$\begin{cases} \mathbf{L}_1\mathbf{y} = \mathbf{b}, \\ \mathbf{U}_1\mathbf{x} = \mathbf{y}. \end{cases} \tag{5}$$

We hebben gezien dat de decompositie van de matrix $\mathbf{A}_2$ andere eigenschappen heeft dan deze van $\mathbf{A}_1$. Volgende opdracht linkt de LU-decompositie van $\mathbf{A}_2$ met de oplossing van het stelsel $\mathbf{A}_2\mathbf{x} = \mathbf{b}$.

Opdracht 15. Stel $\mathbf{x}$ zodat $\mathbf{A}_1\mathbf{x} = \mathbf{b}$. Laat $\mathbf{z}$ de oplossing zijn van het stelsel

$$\mathbf{A}_2\mathbf{z} = \mathbf{P}\mathbf{b}, \tag{6}$$

dan is $\mathbf{x} = \mathbf{Q}\mathbf{z}$. Toon dit aan.

Opdracht 16. We hebben nu twee manieren gezien hoe we het stelsel $\mathbf{A}_1\mathbf{x} = \mathbf{b}$ kunnen oplossen. Enerzijds via (5) en anderzijds door eerst het stelsel $\mathbf{A}_2\mathbf{z} = \mathbf{P}\mathbf{b}$ op te lossen en dan $\mathbf{x} = \mathbf{Q}\mathbf{z}$ uit te rekenen (zie vorige opgave). Vergelijk nu beide methodes door de oplossing van het stelsel $\mathbf{A}\mathbf{x} = \mathbf{b}$ uit te rekenen met $\mathbf{b} = [1, \dots, 1]^T$ en $n = 1000, 1500, 2000, \dots, 7000$ aan de hand van beide methodes en voer timings uit via de commando's `tic` en `toc`. Gebruik hierbij je methodes `solve_Lb`, `solve_Ub`, `solve_Lb_special`, `solve_Ub_special`. Neem een gemiddelde over 20 runs maar laat ook hier de eerste timing weg. Het bepalen van de LU-decompositie van $\mathbf{A}_1$ en $\mathbf{A}_2$ neem

je niet mee in je timings. Je kan dus de decomposities die je reeds bepaalde (en als `.mat`-bestand opsloeg) hergebruiken!

Zijn dit de timings die je verwachtte? Link dit met het resultaat uit Opdracht 9.
Opmerking: Implementeer de matrices $\mathbf{P}$ en $\mathbf{Q}$ als spaarse matrices.

5 Stabiliteit

Passen we nu eens `[L,U] = lu_decomp(A)` toe op de matrix

$$\mathbf{A} = \begin{bmatrix} 10^{-20} & 1 \\ 1 & 2 \end{bmatrix} \tag{7}$$

dan krijgen we dat

$$\mathbf{LU} = \begin{bmatrix} 10^{-20} & 1 \\ 1 & 0 \end{bmatrix} \neq \mathbf{A}.$$

Het lijkt alsof we een fout hebben gemaakt, maar in feite heeft het alles te maken met de stabiliteit van ons algoritme. In de voorbeelden in het vorige onderdeel was dit geen probleem, de matrices waren immers zo gekozen dat er geen probleem was.

Opdracht 17. Het algoritme dat we hebben geïmplementeerd, is niet achterwaarts stabiel. Geef een verklaring aan de hand van het voorbeeld uit (7) en de cursus of slides.
Welke bewerking zorgt er in dit voorbeeld voor dat het algoritme instabiel is? (i.e. welke bewerking introduceert de fout bij de berekening van de LU-decompositie?)

Opdracht 18. In de les en oefenzitting zagen we een manier om dit probleem te omzeilen. Geef een korte beschrijving, en illustreer aan de hand van bovenstaand voorbeeld dat deze aanpak het probleem oplost.

Praktisch

Groepen

Je werkt in groepjes van twee personen. Ten laatste op **zondag 14 april** vermeld je per groepje de twee namen en de twee studentennummers van de leden van het groepje op het forum[4] op Toledo. Er is een forum '*Vind Groepslid*' aangemaakt. Dit kan helpen om een partner te vinden.

Indien je geen partner vindt laten we toe dat je alleen werkt. Je maakt dan enkel de eerste drie onderdelen van het practicum (LU-factorisatie, Spaarse stelsels en LU-factorisatie, en Preconditionering).

Begeleide oefenzitting

Volgens de voorlopige planning zal in de week van 22 tot 26 april in een oefenzitting aan het practicum kunnen werken en kan je ook vragen stellen aan de assistent. Deze oefenzitting hoort niet bij de belasting. Je mag je vragen stellen in de discussieruimte **"Vragen practicum"**. Gelieve je vraag zo duidelijk mogelijk te formuleren. We geven geen garantie dat vragen die een paar dagen voor de deadline worden gesteld nog worden beantwoord.

[4] Zie Toledo > Numerieke Wiskunde[G0N90a] > Practicum > Groepjes practicum

Code

Maak één zip-bestand met jullie achternamen in de bestandsnaam, bv. `codeSmetVanIeper.zip`, die al jullie code bevat. Het zip-bestand moet minstens de volgende bestanden bevatten:

- `lu_decomp.m`
- `solve_Lb.m`
- `solve_Ub.m`
- `solve_Lb_special.m`
- `solve_Ub_special.m`
- `incompl_lu_decomp.m`
- `hoofdprogramma.m`

Vergeet mogelijke hulpfuncties niet toe te voegen die jullie geschreven hebben. Het hoofdprogramma is een Matlabscript dat alle overige code bundelt en die je gebruikte om te antwoorden op de verschillende opdrachten (dus ook code om figuren te genereren). Scheid verschillende opdrachten met een dubbel procentteken en de naam van de opdracht (`%% Opdracht 1`).

Verslag

Schrijf een duidelijk gestructureerd en bondig verslag van maximaal 10 pagina's waarin zeker het volgende staat:

- Alle figuren en tabellen. Gebruik doordachte figuren en tabellen om je bevindingen te verduidelijken. Gebruik logaritmische grafieken waar nodig (`semilogy`, `semilogx` of `loglog`). Kies de schaal van je figuren oordeelkundig, vooral als twee verschillende figuren vergeleken moeten worden. Voeg de gepaste legendes toe, en benoem je assen. Je hoeft geen uitgebreid onderschrift te voorzien, als je in de tekst duidelijk naar de figuur verwijst. **Dit punt is essentieel**: Figuren zijn een belangrijke tool om je argumentatie te staven. Gebruik je slordige/slechte/onduidelijke figuren dan is ook je argumentatie zelf waardeloos.
- Antwoord op alle opgaves en vragen die gesteld worden.
- Indien je dit nodig acht, kan je in het verslag bepaalde ontwerpkeuzes voor je algoritmes verduidelijken.
- Tijdsbesteding aan de verschillende onderdelen van het practicum:
  - Schrijven van de code;
  - Schrijven van het verslag.

Dien je verslag in als een pdf-bestand en geef deze een analoge naam als je code, bv. `verslagSmetVanIeper.pdf`. Zich niet houden aan bovenstaande instructies zal leiden tot penalisatie in de quotering.

Evaluatie en indienen

Het practicum telt mee voor 3 van de 20 punten van het eindexamen. De evaluatie van het practicum is gebaseerd op de ingediende code en het verslag. Je dient het zip-bestand met code en verslag in op toledo. Dit ten laatste op **zondag 5 mei**.

Veel succes!
Het team Numerieke Wiskunde

Appendix

Aangezien jullie nog niet vaak verslagen hebben geschreven, enkele algemene tips in willekeurige volgorde:

- Antwoord gericht op de vraag. Niet relevante "bla-bla" lezen we toch niet.
- Aangezien dit practicum een opeenvolging is van opdrachten, mag je verslag een opeenvolging zijn van antwoorden. Je hoeft geen roman te schrijven. Zorg gewoon dat het er netjes uitziet.
- Gebruik bij voorkeur korte zinnen (i.e. geen 5 geneste bijzinnen). Dit verhoogt de leesbaarheid.
- Indien je resultaten gebruikt die niet standaard zijn, dan refereer je daar naar, of bewijs je die.
- Foto's van tekeningen[5] mogen worden toegevoegd, maar zorg dat deze verzorgd zijn.
- Voorblad en inhoudstafel zijn overbodig. Je schrijft een verslag, geen boek.
- Geen foto's van berekeningen!
- Je gebruikt bij voorkeur LaTeX om je verslag te schrijven.
- Zorg dat de notatie duidelijk is, gebruik deze consistent.
- Structureer je code. Bv. hetgeen binnen een for-loop staat geef je een indent. Gebruik voldoende spaties om de leesbaarheid te vergroten.
- Geef de constanten en variabelen overzichtelijke namen.

[5] Waarschijnlijk niet van toepassing in dit practicum