Numerieke Methoden
3de Bachelor Fysica

# Practicum 2:
# stelsels van lineaire vergelijkingen

*Contact: jeroen.mulkers@uantwerpen.be U.305*

1. **'Sparse' matrices en MATLAB**

   We hebben reeds gezien dat als je rekening houdt met de eventuele speciale structuur van matrices je veel tijd kan uitsparen. Dit is ook het geval voor grote 'sparse' matrices, d.w.z. matrices met veel elementen gelijk aan nul. Vele matrices die men opstelt in de analyse van fysische systemen zijn 'sparse'. Alleen als men hiermee rekening houdt is het mogelijk om bv. grote stelsels met duizenden variabelen op te lossen. In dit practicum worden enkele van MATLABs krachtige mogelijkheden met 'sparse' matrices geïllustreerd. Rekening houden met de vele nullen in een matrix kan niet enkel de snelheid van je algoritme doen toenemen, maar dit bespaart ook op het ingenomen geheugen.

   MATLAB behandelt een matrix **niet** automatisch als 'sparse'. De gebruiker moet aangeven of de matrix tot de 'sparse class' of de 'full class' behoort. Dus

   ≫ b=sparse(a)

   zet een matrix b in een 'sparse' matrix. Hierna zal MATLAB er rekening mee houden dat b 'sparse' is door algoritmes te gebruiken die speciaal afgestemd zijn op 'sparse' matrices. Om b terug om te zetten naar een gewone matrix typen we

   ≫ c=full(b)

   De operatoren $*, +, -, /$ en $\backslash$ leveren enkel een 'sparse' matrix als oplossing als *beide* input operanda 'sparse' zijn. De functies chol(a) en lu(a) leveren ook als resultaat een 'sparse' matrix als a sparse is. Echter in gemengde gevallen waarbij één operand 'sparse' is en een andere 'full', levert als resultaat een 'full' matrix. Als je een eenheidsmatrix wil construeren die

MATLAB herkent als 'sparse' moet je gebruik maken van speye(n) i.p.v. eye(n). Gelukkig is het eenvoudig te testen of MATLAB een matrix als 'sparse' beschouwd of niet met het commando issparse.

Laten we nu een 'sparse' matrix construeren. Dat kunnen we ook doen met het commando sparse dat vele mogelijkheden kent (alle mogelijkheden zie je met help sparse). Het is het eenvoudigst als je volgend voorbeeld uitvoert:

```
≫ colpos = [1 2 1 2 5 3 4 3 4 5];
≫ rowpos = [1 1 2 2 2 4 4 5 5 5];
≫ val=[12 -4 7 3 -8 -13 11 2 7 -4];
≫ a=sparse(rowpos,colpos,val,5,5)
```

Bekijk het resultaat. Kijk ook wat

```
≫ b=full(a)
```

oplevert. Voer nu uit

```
≫ [issparse(a) issparse(b) nnz(a) nnz(b)]
```

en zie de betekenis van het commando nnz.

Construeer nu op een efficiënte manier een tridiagonale 'sparse' 3000×3000 matrix A met op de diagonaal allemaal 4'en, en op de twee off-diagonalen allemaal 2'en. Construeer ook een vector b van lengte 3000, met $b_i = i$. Los nu het stelsel $A\mathbf{x} = \mathbf{b}$ op met als vergrootte matrix:

$$\left[\begin{array}{ccccccc|c} 4 & 2 & 0 & & & & & 1 \\ 2 & 4 & 2 & & & & & 2 \\ & \ddots & \ddots & \ddots & & & & \vdots \\ & & & 2 & 4 & 2 & & 2999 \\ & & & 0 & 2 & 4 & & 3000 \end{array}\right]$$

Doe dit op twee manieren, nl. voor a 'sparse' en 'full', time de tijdsduur in beide gevallen en vergelijk (gebruik hiervoor het commando tic...toc).

Doe hetzelfde voor de LU decompositie met het commando lu.

Een andere manier om 'sparse' matrices te construeren is met sprand en sprandsym. Deze commando's genereren random 'sparse' matrices en symmetrische random 'sparse' matrices. Experimenteer zo bv. met

```
≫ A=sprand(m,n,d)
```

dat een $m \times n$ random matrix genereert met dichtheid $d$ aan niet-nulelementen. Om nog even de kracht van MATLAB te tonen volgend voorbeeldje.

$\gg$ A=sprandsym(6,.4,[1 2.5 6 9 2 4.3])

Dit commando contrueert een symmetrische, random en 'sparse' matrix met een dichtheid van 40 % aan niet nul-elementen, en met eigenwaarden [1 2.5 6 9 2 4.3].

2. De kleinste-kwadratenmethode is een methode om een polynoom te construeren die een goede fit is voor experimentele datapunten. De polynoom van graad 3: $y = a_0 + a_1 x + a_2 x^2 + a_3 x^3$, die het beste $n$ datapunten fit vindt men door volgend stelsel op te lossen (zie les 5):

$$\begin{array}{ccccccccc} a_0 n & + & a_1 \sum x_i & + & a_2 \sum x_i^2 & + & a_3 \sum x_i^3 & = & \sum y_i \\ a_0 \sum x_i & + & a_1 \sum x_i^2 & + & a_2 \sum x_i^3 & + & a_3 \sum x_i^4 & = & \sum y_i x_i \\ a_0 \sum x_i^2 & + & a_1 \sum x_i^3 & + & a_2 \sum x_i^4 & + & a_3 \sum x_i^5 & = & \sum y_i x_i^2 \\ a_0 \sum x_i^3 & + & a_1 \sum x_i^4 & + & a_2 \sum x_i^5 & + & a_3 \sum x_i^6 & = & \sum y_i x_i^3 \end{array}$$

Vind, door bovenstaand stelsel op te lossen, de kleinste-kwadratenpolynoom van graad 3 die het best volgende data fit:

| $x$ | 1.0 | 1.5 | 2.0 | 2.5 | 3.0 | 3.5 | 4.0 | 4.5 |
|---|---|---|---|---|---|---|---|---|
| $y$ | 4.3 | 6.2 | 10.1 | 13.5 | 19.8 | 22.6 | 24.7 | 29.2 |

Maak ook een figuur van je resultaat (de datapunten + de fit). Maak hierbij gebruik van het MATLAB commando polyval. Let erop om de fit in een fijner grid over de x-as te evalueren.

3. We hebben gezien dat het belangrijk is om formules met de inverse van een matrix te interpreteren in termen van LU decompositie (wat MATLAB ook doet als je de \ operator gebruikt). Schrijf een MATLAB functie die $\alpha$ berekent $\alpha = \mathbf{c}^T A^{-1} \mathbf{d}$ met $\mathbf{c}, A$ en $\mathbf{d}$ gegeven, zonder $A^{-1}$ expliciet te berekenen.

4. In de les hebben we de Jacobi-aanpak afgeleid om een stelsel iteratief op te lossen. Het volgende voorbeeld illustreert Jacobi's methode:

$$\begin{array}{rcrcrcl} -7x + 2y + 0z & = & 5 & & \hat{x}_{i+1} & = & -\frac{1}{7}(5 + 0\hat{x}_i - 2\hat{y}_i - 0\hat{z}_i) \\ 0x + 16y + 8z & = & 7 & \rightarrow & \hat{y}_{i+1} & = & \frac{1}{16}(7 + 0\hat{x}_i + 0\hat{y}_i - 8\hat{z}_i) \\ 6x - 11y + 26z & = & -3 & & \hat{z}_{i+1} & = & \frac{1}{26}(-3 - 6\hat{x}_i + 11\hat{y}_i + 0\hat{z}_i) \end{array}$$

Dit voorbeeld suggereert ook een andere aanpak die men de Gauss-Seidel methode noemt. Als we $x$ geupdate hebben, dus als we $\hat{x}_{i+1}$ berekend hebben, kunnen we dit resultaat reeds gebruiken in de volgende berekeningen.

Voor ons voorbeeld wordt de Gauss-Seidelmethode:

$$\begin{aligned}\hat{x}_{i+1} &= -\tfrac{1}{7}(5 + 0\hat{x}_i - 2\hat{y}_i - 0\hat{z}_i)\\ \hat{y}_{i+1} &= \tfrac{1}{16}(7 + 0\hat{x}_{i+1} + 0\hat{y}_i - 8\hat{z}_i)\\ \hat{z}_{i+1} &= \tfrac{1}{26}(-3 - 6\hat{x}_{i+1} + 11\hat{y}_{i+1} + 0\hat{z}_i)\end{aligned}$$

Intuïtief verwachten we dat de Gauss-Seidelmethode sneller convergeert dan Jacobi's methode.

Formeel wordt de methode van Gauss-Seidel als volgt afgeleid:

$$A\mathbf{x} = \mathbf{b} \Rightarrow (L_A + M)\mathbf{x} = \mathbf{b} \Rightarrow \mathbf{x} = L_A^{-1}(\mathbf{b} - M\mathbf{x}),$$

met $L_A$ het benedendriehoeksdeel van $A$ (dus $L_{Aij} = A_{ij}$ als $i \leq j$ en 0 anders) en $M$ het bovendriehoeksdeel van $A$ maar ook met nullen op de diagonaal (dus $M_{ij} = A_{ij}$ als $i > j$ en 0 anders). Het algoritme is dus

$$\hat{\mathbf{x}}_{i+1} = L_A^{-1}(\mathbf{b} - M\hat{\mathbf{x}}_i), \text{ voor } i = 0, 1, 2, \ldots \text{ met gegeven } \hat{\mathbf{x}}_0.$$

Implementeer de Gauss-Seidelmethode door de functie jacobi.m aan te passen. Maak gebruik van de MATLAB commando's tril en triu die respectievelijk de beneden- en bovendriehoeksdelen van hun argument opleveren (in de help triu staat hoe je er ook voor kan zorgen dat $M$ enkel nullen op de diagonaal heeft). Hou ook rekening met wat er in de derde oefening van dit practicum gezegd werd.

Test uw functie op een voorbeeld en vergelijk de convergentiesnelheid met Jacobi's methode.

5. We komen nu terug op de opmerkingen aan het einde van het hoofdstuk over het bepalen van nulpunten. We hebben daar getoond hoe de oplossing van een stelsel van **niet-lineaire** vergelijkingen kan gevonden worden met een uitbreiding van de methode van Newton. Beschouw een set van $n$ niet-lineaire vergelijkingen met $n$ onbekenden. Het algoritme gaat als volgt:

   $\mathbf{x}_0 =$ Startkeuze
   for i=0:N
   Los op voor $\Delta\mathbf{x}^{(i)}$ : $\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \begin{bmatrix} \Delta x_1^{(i)} \\ \vdots \\ \Delta x_n^{(i)} \end{bmatrix} = - \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}$
   Update oplossing: $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} + \Delta\mathbf{x}^{(i)}$
   end

   waar de functies geëvalueerd worden in $\mathbf{x}^{(i)} = (x_1^{(i)}, \ldots, x_n^{(i)})$. Merk op dat dit een stelsel van **lineaire** vergelijkingen vormt.

Implementeer deze methode. Ga er vanuit dat er een functie J(x) bestaat die de matrix van eerste afgeleiden berekent (dit is de Jacobiaan).

Los daarna volgend niet-lineair stelsel op (bepaal de nulpunten):

$$\begin{cases} x_1 + 2x_2 - 2 = 0 \\ x_1^2 + 4x_2^2 - 4 = 0 \end{cases}$$

Programmeer dus eerst die functie J(x) voor dit stelsel.