KTH Engineering Sciences

Laboration 1

Ekvationslösning

I denna lab ska ni använda numeriska metoder för att lösa olika olinjära och linjära ekvationer. Relaterat till detta ska ni studera konvergensfrågor, störningskänslighet och komplexitet.

Laborationen består av tre uppgifter. För varje uppgift finns en MATLAB-fil i Canvas som är en mall för era program. Filerna heter `uppg1.m`, `uppg2.m` och `uppg3.m`. Ni ska programmera de delar som saknas och skicka en den slutgiltiga filen i Canvas före redovisningen av laborationen. Till uppgift 3 ska ni också skicka in en text-fil i PDF-format som innehåller den efterfrågade tabellen med exekveringstider.

- De inlämnade uppgifts-filerna ska gå att köra utan några speciella inställningar eller andra hjälpfiler (utöver `trussplot.m` i uppgift 3). De ska producera de resultat som efterfrågas i respektive uppgift (skriva ut värden, generera plottar, etc.). De behöver inte ge svar på •-frågorna dock.
- I varje uppgifts-fil finns någon/några fördefinierad funktion(er) som ska implementeras. Följ de givna specifikationerna för funktionernas argument och returvärden. Ändra inte syntaxen.
- När flera figurer ska plottas i samma uppgift, använd MATLABs kommando `figure(n)` med olika `n` för att få flera figurfönster.
- Använd två procent-tecken `%%` i början av en rad för att dela in era programfiler i sektioner. Programmet kan då köras sektionsvis med *Run Section* i MATLAB-klienten. Se `uppg1.m` för exempel.
- Använd inte MATLABs symboliska funktioner/variabler (`syms`) för beräkningar. Dessa gör programmen betydligt långsammare än vanliga Matlab-funktioner.
- Lämna inte in MATLAB Live Script (`.mlx`-filer). Det introducerar mycket overhead som gör exekvering, och framförallt plottning, långsam.

På redovisningen ska ni (båda individuellt om ni är två i gruppen) kunna redogöra för teori och algoritmer som ni använt. Ni ska kunna svara på frågepunkterna (•) och förklara hur era MATLAB-program fungerar. Kom väl förberedda!

1. Olinjär skalär ekvation

Vi vill bestämma samtliga rötter till följande skalära ekvation,

$$x^2 - 8x - 12\sin(3x+1) + 19 = 0. \tag{1}$$

a) Plotta $f(x) = x^2 - 8x - 12\sin(3x+1) + 19$. Samtliga nollställen till f skall synas i plotten. Notera ungefärliga värden på nollställena. (Dessa kommer ni använda som startgissningar i metoderna nedan.)

- Hur många nollställen finns det?

b) Skriv MATLAB-funktionen `fixpunkt()` som beräknar nollställena till (1) med hjälp av fixpunktsiterationen

$$x_{n+1} = \frac{1}{19}\left[x_n^2 + 11x_n - 12\sin(3x_n+1)\right] + 1. \tag{2}$$

Använd sedan funktionen för att empiriskt undersöka vilka nollställen till (1) som ni kan bestämma med fixpunktiterationen. Beräkna dessa nollställen med ett fel mindre än 10^{-10}. Programmet ska skriva ut värdena på de rötter som kan bestämmas, startgissningar och antal iterationer som krävdes. För en av rötterna (välj själva) ska även de sista tio värdena på $|x_{n+1} - x_n|$ skrivas ut. Använd `format long` för att se alla decimaler.

- Motivera varför (2) är en fixpunktiteration för (1).
- Använd teorin för att förklara vilka nollställen fixpunktiterationen kan hitta.
- Hur ska skillnaderna $|x_{n+1} - x_n|$ avta enligt teorin? Verifiera att det stämmer i praktiken.

c) Skriv MATLAB-funktionen `newton()` som beräknar nollställena till (1) med hjälp av Newtons metod. Använd den för att beräkna samtliga nollställen med ett fel mindre än 10^{-10}. Programmet ska skriva ut rötternas värden, startgissningar och antal iterationer som krävdes. För en av rötterna (välj själva) ska även värdena på $|x_{n+1} - x_n|$ skrivas ut.

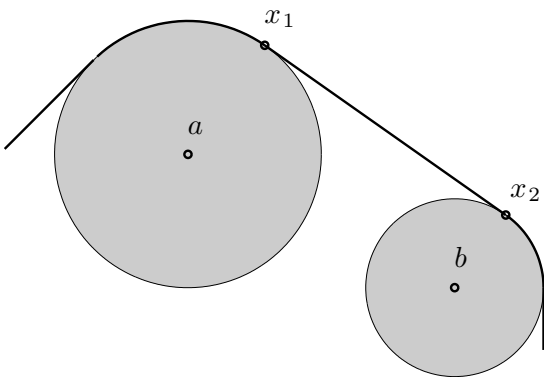
- Stämmer tumregeln för Newtons metod att antalet korrekta siffror dubblas i varje iteration?
- Hur ska skillnaderna $|x_{n+1} - x_n|$ avta enligt teorin? Verifiera att det stämmer i praktiken.

d) Ni ska nu jämföra konvergensen för de två metoderna noggrannare. Välj den minsta av rötterna där bägge metoderna fungerar. Gör först en mycket noggrann (15 korrekta siffror) referenslösning x^* med Newtons metod som ni använder för att beräkna felen nedan.

1. Plotta felet $e_n = |x_n - x^*|$ efter iteration n som funktion av n för båda metoderna i samma figur när de initieras med samma startgissning x_0. Använd `semilogy`-plot för att få logskala på y-axeln.
   - Vilken av metoderna konvergerar snabbast?
2. Ett precist sätt att kvantifiera hur snabbt metoden konvergerar är konvergensordningen. Den beskriver hur mycket mindre felet e_{n+1} är jämfört med e_n; konvergensordningen är p om $e_{n+1} \sim e_n^p$ när $n \to \infty$. Plotta därför e_{n+1} som en funktion av e_n i en `loglog`-plot för båda metoderna i samma figur.
   - Uppskatta metodernas konvergensordning med hjälp av figuren. Stämmer det med teorin?

2. Snöre runt cirklar

Ett snöre spänns runt två cirkelskivor enligt figuren. Cirklarna har radierna r_a respektive r_b och är centrerade i punkterna $\boldsymbol{a}$ respektive $\boldsymbol{b}$. Låt $\boldsymbol{x}_1$ och $\boldsymbol{x}_2$ vara de punkter (markerade i figuren) där snöret släpper från cirkeln. Dessa punkter kommer att satisfiera ekvationssystemet



$$
\begin{aligned}
|\boldsymbol{x}_1 - \boldsymbol{a}|^2 &= r_a^2, \\
|\boldsymbol{x}_2 - \boldsymbol{b}|^2 &= r_b^2, \\
(\boldsymbol{x}_1 - \boldsymbol{x}_2) \cdot (\boldsymbol{x}_1 - \boldsymbol{a}) &= 0, \\
(\boldsymbol{x}_1 - \boldsymbol{x}_2) \cdot (\boldsymbol{x}_2 - \boldsymbol{b}) &= 0.
\end{aligned}
$$

(Med $\boldsymbol{x} \cdot \boldsymbol{y}$ menas här skalärprodukten mellan vektorerna $\boldsymbol{x}$ och $\boldsymbol{y}$.) De två första ekvationerna kommer från kravet att $\boldsymbol{x}_1$ och $\boldsymbol{x}_2$ ligger på respektive cirkelrand. De två sista ekvationerna ges av att snöret är tangentiellt med cirkelranden vid punkterna, så att vektorn $\boldsymbol{x}_1 - \boldsymbol{x}_2$ är vinkelrät mot både $\boldsymbol{x}_1 - \boldsymbol{a}$ och $\boldsymbol{x}_2 - \boldsymbol{b}$.

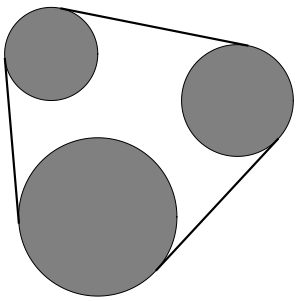
a) Förberedande arbete (tas även upp på Övning 2, 2/2).
Låt $\boldsymbol{x}_1 = (x_1, y_1)$ och $\boldsymbol{x}_2 = (x_2, y_2)$. Skriv om systemet på formen $\boldsymbol{F}(x_1, y_1, x_2, y_2) = 0$ där $\boldsymbol{F}$ är en vektorvärd funktion med fyra komponenter. Radierna r_a, r_b och cirklarnas medelpunkter $\boldsymbol{a} = (x_a, y_a)$, $\boldsymbol{b} = (x_b, y_b)$ kommer in som parametrar i ekvationerna. Beräkna också jakobimatrisen till $\boldsymbol{F}$.

b) Skriv MATLAB-funktionen `punkter()` som löser ekvationssystemet med Newtons metod för system. Felet ska vara mindre än 10^{-10} i alla komponenter. Notera: För enkelhetens skulle sätts här toleransen inne i funktionen, och funktionen returnerar enbart den sista, bästa, approximationen av lösningen, inte alla iterationer.

Använd funktionen för att lösa ekvationssystemet för fallet $\boldsymbol{a} = (-1.5,\, 3.0)^T$, $\boldsymbol{b} = (1.0,\, 1.0)^T$, $r_a = 1.5$ och $r_b = 0.8$. Skriv ut värdena på lösningen (x_1, y_1), (x_2, y_2) och antal iterationer. Plotta cirklarna och snöret i en figur.

- Hur många olika lösningar finns det? Hur ser de ut?

c) Skriv MATLAB-funktionen `langd()` som använder funktionen `punkter()` för att beräkna längden på ett snöre som spänts runt tre cirklar med medelpunkterna $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$ och radierna r_a, r_b, r_c. Bestäm sedan snörlängden för fallet när $\boldsymbol{a} = (-1.0,\, 1.5)^T$, $\boldsymbol{b} = (3.0,\, 0.5)^T$, $\boldsymbol{c} = (0.0,\, -2.0)^T$ och $r_a = 1.0$, $r_b = 1.2$, $r_c = 1.7$. (Jämför figuren till höger.) Programmet ska plotta cirklarna och snöret i en figur samt skriva ut snörets längd. Glöm inte att räkna in längden på den del av snöret som ligger an cirklarna. (Den delen behöver dock inte plottas.)

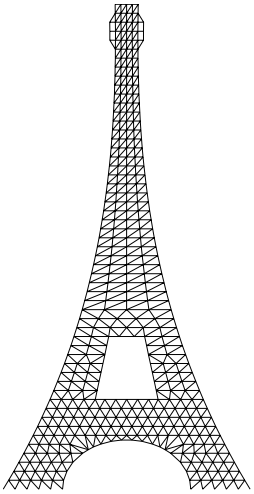


d) Bestäm osäkerheten i snörets längd när osäkerheten i alla parametrar (radier och mittpunkternas koordinater) är ± 0.01. Använd experimentell störningsanalys där ni anropar funktionen `langd()` högst 10 gånger. Programmet ska skriva ut osäkerheten.

- För vilka variabler bidrar osäkerheten i indata mest respektive minst till osäkerheten i snörets längd?

3. Stora matriser

I många realistiska tillämpningar måste man lösa *stora* linjära ekvationsystem, med miljontals obekanta. Det är i dessa fall som effektiva algoritmer blir viktiga att använda. Som exempel ska ni här räkna på ett komplicerat fackverk: en modell av Eiffeltornet. Ett fackverk består av stänger förenade genom leder i ett antal noder. Ni ska beräkna deformationen av fackverket när noderna belastas av yttre krafter. Ekvationerna för deformationen härleds i hållfasthetsläran, och baseras på att förskjutningarna i varje nod är små, och att Hookes lag gäller för förlängningen av varje stång.



Modell `eiffel2.mat`, med 399 noder (798 obekanta).

I slutändan får man ett linjärt ekvationssystem på formen $A\boldsymbol{x} = \boldsymbol{b}$. När antalet noder i fackverket är n kommer antalet obekanta vara $2n$ och $A \in \mathbb{R}^{2n\times 2n}$. Matrisen A brukar kallas *styvhetsmatrisen*. Högerledet $\boldsymbol{b}$ innehåller de givna yttre krafterna som verkar på noderna,

$$\boldsymbol{b} = \Big(F_1^x, F_1^y, F_2^x, F_2^y, \ldots, F_n^x, F_n^y\Big)^T, \qquad \boldsymbol{b} \in \mathbb{R}^{2n},$$

där $\boldsymbol{F}_j = (F_j^x, F_j^y)^T$ är kraften i nod j. Lösningen $\boldsymbol{x}$ innehåller de resulterande (obekanta) förskjutningarna,

$$\boldsymbol{x} = \Big(\Delta x_1, \Delta y_1, \Delta x_2, \Delta y_2, \ldots, \Delta x_n, \Delta y_n\Big)^T, \qquad \boldsymbol{x} \in \mathbb{R}^{2n}.$$

Här är alltså $(\Delta x_j, \Delta y_j)^T$ förskjutningen av nod j när fackverket belastas med krafterna i $\boldsymbol{b}$.

På kurshemsidan finns filerna `eiffel1.mat`, `eiffel2.mat`, `eiffel3.mat` och `eiffel4.mat`. De innehåller fyra olika modeller av Eiffeltornet med växande detaljrikedom ($n = 261$, 399, 561, 1592). Varje modell består av nodkoordinater i vektorerna `xnod, ynod`, stångindex i matrisen `bars` (används bara för plottningen) och styvhetsmatrisen `A`.

a) Ladda in en av modellerna i MATLAB med kommandot `load`. Hämta funktionsfilen `trussplot.m` från kurshemsidan och anropa den med `trussplot(xnod,ynod,bars)` för att plotta tornet. Välj en av noderna och belasta den med en kraft rakt högerut med beloppet ett. (Sätt $F_j^x = 1$ för något j, och resten av elementen i $\boldsymbol{b}$ lika med noll, dvs `b=zeros(2*n,1); b(j*2-1)=1;` i MATLAB.) Lös systemet $A\boldsymbol{x} = \boldsymbol{b}$ med backslash för att få fram förskjutningarna i alla punkter. Beräkna de nya koordinaterna för det belastade tornet, $x_j^{\text{bel}} = x_j + \Delta x_j$, etc.:

```
xbel = xnod + x(1:2:end); ybel = ynod + x(2:2:end);
```

Plotta det belastade tornet. Använd `hold on` för att plotta de två tornen ovanpå varandra i samma figur. Markera vilken nod ni valt med en asterisk * i figuren.

b) Backslash-kommandot i MATLAB använder normalt vanlig gausseliminering. Undersök hur tidsåtgången för gausseliminering beror på systemmatrisens storlek genom att lösa ekvationsystemet $A\boldsymbol{x} = \boldsymbol{b}$ med ett godtyckligt valt högerled $\boldsymbol{b}$ (tex med `b=randn(N,1)`) för var och en av de fyra modellerna. Använd MATLAB-kommandona `tic` och `toc` (`help tic` ger mer info).[1] Plotta tidsåtgången mot antal obekanta $N = 2n$ i en `loglog`-plot; använd också `grid on`.

- Hur ska tidsåtgången bero på N enligt teorin? Stämmer det överens med din plot?

[1] För att få bra noggrannhet i tidsmätningen (speciellt om den är kort) kan man mäta totaltiden för flera upprepade beräkningar, och sedan dividera tiden med antalet upprepningar.

c) Ni ska nu räkna ut i vilka noder fackverket är minst respektive mest känsligt för vertikal belastning. Ett par olika metoder ska användas.

Skriv MATLAB-funktionen `kanslighet()`. Funktionen ska loopa igenom alla noderna i fackverket. För nod j ska tornet belastas med en vertikal nedåtriktad kraft enbart i nod j, mer precist är $F_j^y = -1$ och all övriga element i $\boldsymbol{b}$ noll. De resulterande förskjutningarna $\boldsymbol{x}$ beräknas och storleken på den totala förskjutningen T_j i euklidiska normen sparas, där

$$T_j := ||\boldsymbol{x}|| = \left(\sum_{j=1}^{n} \Delta x_j^2 + \Delta y_j^2 \right)^{1/2}.$$

Den nod som har störst T_j är mest känslig och den som har minst T_j är minst känslig. Funktionen `kanslighet()` ska returnera indexen för dessa två T_j-värden. För varje j måste alltså ett linjärt ekvationssystem lösas. Börja med att implementera detta med en naiv metod (`metod=1` i funktionen) där ni använder backslash för varje j.

Använd `kanslighet()`-funktionen med `metod=1` för att hitta den minst och mest känsliga noden i modellen `eiffel1.mat`. Plotta detta torn med `trussplot` och markera de minst och mest känsliga noderna i figuren med en cirkel `o` respektive asterisk *.

d) I deluppgift (c) löser ni samma stora linjära ekvationssystem med många olika högerled (n stycken). För de större modellerna blir detta mycket tidskrävande. Implementera en bättre metod i `kanslighet()`-funktionen (`metod=2`) genom att använda LU-faktorisering av matrisen A (MATLAB-kommandot `lu`). Verifiera att metoden ger samma resultat som metod 1 och är snabbare.

När en matris är *gles* kan man lösa ekvationssystemet med effektivare metoder än standard gausseliminering. Ni kan använd kommandot `spy(A)` för att förvissa er om att styvhetsmatrisen i det här fallet faktiskt är gles (bandad). Tala om för MATLAB att matrisen är gles genom att konvertera format med kommandot `A=sparse(A)`. Bättre metoder kommer då automatiskt användas när backslash och `lu` anropas. Verifiera att ni får samma resultat som tidigare för både metod 1 och 2 är A är i glest format, och att beräkningarna är snabbare.

Lägg nu till kod som använder `kanslighet()` för att lösa problemet för alla fyra modellerna med alla fyra metoderna (med/utan LU, full/gles matris). Bestäm tidsåtgången för anropet till `kanslighet()` i varje fall[2]. Skapa en tabell av följande typ och fyll i tiderna. I `uppg3.m`-filen finns ett exempel på hur man gör tabeller i MATLAB. Tabellen ska skickas in som PDF-fil.

	Naiv	LU	Gles (ej LU)	Gles+LU
`eiffel1.mat`				
`eiffel2.mat`				
`eiffel3.mat`				
`eiffel4.mat`				

- Varför går det snabbare att lösa problemet med LU-faktorisering?
- Vilken metod löser problemet snabbast? (Med/utan LU? Full/gles lösare?)
- För vilken modell blir tidsvinsten störst? Varför?

[2]Ni kan hoppa över de fall som tar orimligt lång tid.