KTH Engineering Sciences

Laboration 1

Ekvationslösning, linjär algebra och minstakvadratmetoden

I denna lab ska ni använda numeriska metoder för att lösa olinjära och linjära ekvationer samt minstakvadratproblem. Relaterat till detta ska ni studera konvergensfrågor, störningskänslighet och komplexitet. Laborationen består av fyra uppgifter. Ni ska implementera era lösningar i MATLAB.

Laborationen examineras med ett individuellt datorprov. Före provet ska ni skicka in era MATLAB-program i Canvas. Till uppgift 3 ska ni också skicka in en text-fil som innehåller den efterfrågade tabellen med exekveringstider. På provet ska ni lösa uppgifter som är snarlika uppgifterna nedan. Ni kommer då ha tillgång till de filer ni skickat in. Tänk därför på att skriva så tydlig och generell kod som möjligt, så att det enkelt går att göra mindre förändringar i koden.

Frågepunkterna (•) i texten nedan är instuderingsfrågor som kopplar ihop de empiriska resultaten med teorin i kursen. Fundera gärna på dem!

Se Canvas för mer instruktioner, tips och rekommendationer.

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 $f(x) = x^2 - 8x - 12\sin(3x+1) + 19$. Se till att samtliga nollställen till f syns i plotten. Notera ungefärliga värden på nollställena. (Dessa kommer ni använda som startgissningar i metoderna nedan.)

b) Skriv MATLAB-kod som beräknar nollställena till (1) med hjälp av fixpunktsiterationen

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

Undersök empiriskt 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, vilka startgissningar som användes och antal iterationer som krävdes. Använd kommandot `format long` för att se alla decimaler.

Tips: Verifiera att din kod räknar rätt genom att studera differenserna $d_n := |x_{n+1} - x_n|$. Vid konvergens ska d_{n+1}/d_n konvergera mot en konstant mindre än ett.

- Motivera varför (2) är en fixpunktiteration för (1).
- Använd teorin för att förklara vilka nollställen fixpunktiterationen kan hitta.

c) Skriv MATLAB-kod 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, vilka startgissningar som användes och antal iterationer som krävdes.

Tips: Verifiera att din kod räknar rätt genom att studera differenserna $d_n := |x_{n+1} - x_n|$. Tumregeln för Newtons metod säger att antal ledande nollor i d_n approximativt ska dubblas i varje iteration. (Det betyder också att antal korrekta siffror i x_n approximativt dubblas.)

d) Ni ska nu jämföra konvergensen för de två metoderna noggrannare. Välj en av rötterna och gör först en mycket noggrann (15 korrekta siffror) referenslösning x^* med Newtons metod. Använd sedan x^* när ni beräknar felen nedan.

(i) 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.

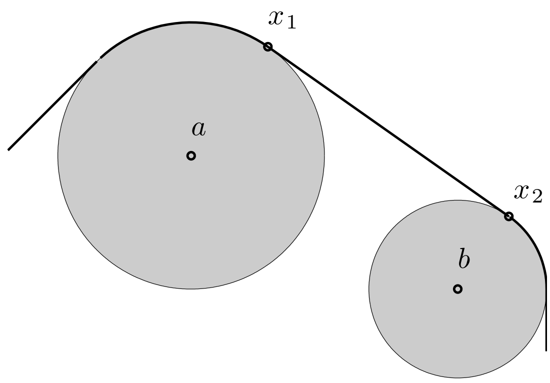
  - Vilken av metoderna bör konvergera snabbast? Stämmer det?

(ii) 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 den uppskattade konvergensordningen 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



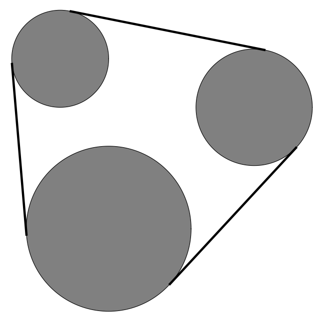
$$\begin{aligned} |\boldsymbol{x}_1 - \boldsymbol{a}|^2 &= r_a^2, \\ |\boldsymbol{x}_2 - \boldsymbol{b}|^2 &= r_b^2, \\ (\boldsymbol{x}_1 - \boldsymbol{x}_2) \cdot (\boldsymbol{x}_1 - \boldsymbol{a}) &= 0, \\ (\boldsymbol{x}_1 - \boldsymbol{x}_2) \cdot (\boldsymbol{x}_2 - \boldsymbol{b}) &= 0. \end{aligned}$$

(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, 17/9)*
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 ett MATLAB-program som löser ekvationssystemet med Newtons metod för system. Felet ska vara mindre än 10^{-10} i alla komponenter. Använd programmet 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 som användes. Plotta cirklarna och snöret i en figur. Hur många olika lösningar finns det? Hur ser de ut?

c) Använd nu koden från deluppgift (b) 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 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[1].

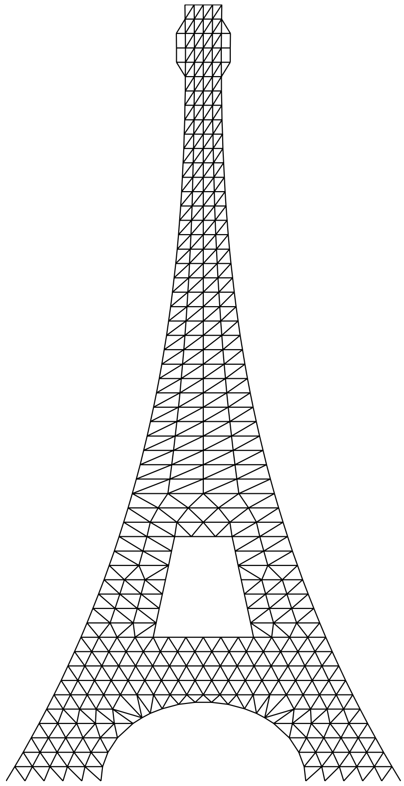
Ledning: Skriv om programmet i deluppgift (c) till en funktion som tar $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$, r_a, r_b och r_c som argument och returnerar snörets längd. Anropa funktionen högst 10 gånger för att få fram osäkerheten.

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

[1] Se anteckningar om Fel- och störningsanalys i Canvas.

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,

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

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

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

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

I Canvas 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 Canvas 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).[2] 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?

c) Ni ska nu räkna ut vilka noder i fackverket som är minst respektive mest känsliga för vertikal belastning. Ett par olika metoder ska användas och tidsåtgången för dem ska jämföras.

För att hitta de sökta noderna måste man lösa ekvationssystemet en gång för varje nod. Mer precist behöver man bestämma $\boldsymbol{x}_j$ så att

$$A\boldsymbol{x}_j = \boldsymbol{b}_j, \qquad j = 1, \ldots, n,$$

där $\boldsymbol{b}_j$ betecknar högerledsvektorn som motsvarar nedåtriktad belastning av nod j, dvs den där $F_j^y = -1$ och all övriga element är noll. Man definierar sedan den totala förskjutningen T_j som den euklidiska normen av $\boldsymbol{x}_j$,

$$T_j := ||\boldsymbol{x}_j||, \qquad ||\boldsymbol{x}|| = \left(\sum_{k=1}^{n} \Delta x_k^2 + \Delta y_k^2 \right)^{1/2}.$$

Den känsligaste noden är den som har störst T_j och den minst känsliga noden är den som har minst T_j.

(i) Skriv ett MATLAB-program som loopar igenom alla noderna i fackverket, beräknar $\boldsymbol{x}_j$ och T_j samt hittar den känsligaste och minst känsliga noden. Använd backslash här för att lösa ekvationssystemen och tex kommandot `norm` för att beräkna T_j. Testa först programmet på den minsta modellen `eiffel1.mat`. Plotta detta torn med `trussplot` och markera de minst och mest känsliga noderna i figuren med en cirkel `o` respektive asterisk *. Förvissa er om att resultatet verkar rimligt.

(ii) Lägg till kod som mäter tiden för att genomföra beräkningarna. Använd tex `tic` och `toc` som tidigare. Inkludera inte plottning och inladdning av filer i tidsmätningen.

(iii) Den naiva metoden att lösa ekvationssystemen med backslash varje gång blir mycket tidskrävande för de större modellerna. När man som här löser samma stora linjära ekvationssystem med många olika högerled kan man använda LU-faktorisering av matrisen A (MATLAB-kommandot `lu`) för att snabba upp beräkningarna. Gör det! Se till att LU-faktoriseringen bara görs en gång, utanför loopen, och att den inkluderas i tidsmätningen. Verifiera att metoden ger samma resultat som den naiva metoden och är snabbare.

(iv) När en matris är *gles* kan man lösa ekvationssystemet med effektivare metoder än standard gausseliminering. (Ni kan använda 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 den första naiva metoden och den som använder LU-faktorisering, när A är i glest format, och att beräkningarna är snabbare.

[2] 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.

Gå igenom de fyra modellerna `eiffel1.mat`, ..., `eiffel4.mat` och mät hur lång tid det tar att lösa problemet med de fyra olika metoderna som diskuterats ovan.[3]. Sammanfatta resultaten i en tabell enligt nedan. Tabellen ska skickas in.

	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?

4. Minstakvadratmetoden

På kurshemsidan finns finns datafilen `dollarkurs.mat` som innehåller dagsnoteringar för dollarkursen under tre och ett halvt år, från 1a januari 2020 till 30 juni 2023. Läs in filen i MATLAB med kommandot `load`. Kursvärdena och motsvarande dagar finns i variablerna `USDSEK` och `day`. (Dag 1 motsvarar 1 januari 2020. Helger är inte med bland dagarna.) Nedan används notationen X_i för kursen dag t_i. Antal datapunkter betecknas N.

a) Anpassa en linjär modell, $g(t) = c_0 + c_1 t$, i minstakvadratmening till dollarkursen genom att ställa upp och lösa lämpligt linjärt ekvationssystem för koefficienterna c_0 och c_1. Plotta data (t_i, X_i) tillsammans med den anpassade kurvan. Skriv ut värdena på koefficienterna och medelkvadratfelet

$$E = \frac{1}{N}\sum_{i=1}^{N}(X_i - g(t_i))^2.$$

b) Plotta felet mellan den linjära modellen och data. Felet tycks vara periodiskt. Uppskatta periodlängden L från plotten. Anpassa därefter följande modell till dollarkursen:

$$g(t) = d_0 + d_1 t + d_2 \sin(2\pi t/L) + d_3 \cos(2\pi t/L).$$

Plotta resultatet som ovan. Skriv ut värdena på koefficienterna och medelkvadratfelet.

c) Låt nu även L vara en okänd parameter. Modellen blir då olinjär. Hitta hela den parameteruppsättningen d_0, d_1, d_2, d_3, L som ger bäst approximation av data i minstakvadratmening. Använd Gauss-Newtons metod med resultatet från deluppgift (b) som startgissning. Skriv ut värdena på koefficienterna, inklusive L, och medelkvadratfelet. Plotta slutligen data (t_i, X_i) och alla de tre modellernas anpassning i samma figur.

- Jämför medelkvadratfelen i de olika modellerna. Vilket är störst/minst? Varför?

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