KTH Engineering Sciences

Laboration 1

Ekvationslösning och numerisk integration

Före redovisningen ska ni skicka in MATLAB*-filer och vissa resultatfiler i Canvas. Vad som ska skickas in står angivet efter respektive uppgift. På redovisningen ska ni (båda i laborationsgruppen om ni är två) kunna redogöra för teori och algoritmer som ni använt. Ni ska också 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,

$$x^2 - 9x - 12\sin(3x+1) + 20 = 0. \qquad (1)$$

a) Plotta $f(x) = x^2 - 9x - 12\sin(3x+1) + 20$. Samtliga nollställen till f skall vara med. Notera ungefärliga värden på nollställena och använd dem som startgissningar i metoderna nedan.

- Hur många nollställen finns det?

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

$$x_{n+1} = \frac{1}{20}x_n^2 + \frac{11}{20}x_n - \frac{3}{5}\sin(3x_n+1) + 1. \qquad (2)$$

Funktionen ska ta som indata en startgissning x_0 och en feltolerans τ. Den ska skriva ut x_n och skillnaden $|x_{n+1} - x_n|$ efter varje iteration, samt returnera ett svar med ett fel som är mindre än τ. (Använd lämpligt avbrottsvillkor för att säkerställa detta.)

Använd 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}. Vad blir värdena? 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.

Skicka in: Funktionsfilen `fixpunkt.m`. Den ska kunna anropas som "`xrot = fixpunkt(x0,tau)`". Variablerna `xrot`, `x0` och `tau` ska alla vara skalärer (ej vektorer).

c) Skriv en MATLAB-funktion som beräknar nollställena till (1) med hjälp av Newtons metod. Funktionen ska i övrigt bete sig på samma sätt som funktionen i (b) ovan. (Samma indata, utdata och utskrifter.) Beräkna samtliga nollställen med ett fel mindre än 10^{-10}. Vad blir värdena?

- Jämför konvergensen för fixpunktiteration och Newtons metod. Vilken metod konvergerar snabbast? Hur hänger det ihop med metodernas konvergensordningar?
- Stämmer tumregeln för Newtons metod att antalet korrekta siffror dubblas i varje iteration?

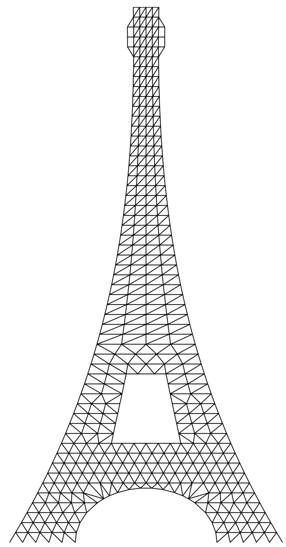
Skicka in: Funktionsfilen `newton.m`. Den ska kunna anropas som `"xrot = newton(x0,tau)"`. Variablerna `xrot`, `x0` och `tau` ska alla vara skalärer (ej vektorer).

2. 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.

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,

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



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

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 ekvationssystemet $A\boldsymbol{x} = \boldsymbol{b}$ med ett godtyckligt valt högerled $\boldsymbol{b}$ 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?

Skicka in: Figuren `tidsplot`, antingen i `.fig`-format (gör Save i figur-fönstret) eller i `.png`-format (gör `print -dpng tidsplot.png` vid prompten).

c) Ni ska nu skriva ett MATLAB-program som räknar ut i vilka noder fackverket är mest respektive minst känslig för *vertikal* belastning. Använd den minsta modellen, `eiffel1.mat`. Tag en nod i taget. Belasta den med kraften $F_j^y = -1$ (istället för F_j^x) och räkna ut resulterande förskjutningar $\boldsymbol{x}$. Notera storleken på den totala förskjutningen i euklidiska normen,

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

Fortsätt med nästa nod, etc. Systematisera beräkningarna med en `for`-slinga i ert MATLAB-program och spara storleken på förskjutningen för varje nod. Ta sedan reda på vilken nod som ger störst respektive minst total förskjutning. Programmet ska slutligen plotta tornet med `trussplot` och markera dessa mest och minst känsliga noder i figuren med cirkel `o` respektive asterisk *.

Skicka in: Filen `kanslighet.m` med programmet.

d) I c) löser ni samma stora linjära ekvationssystem med många olika högerled (n stycken). För de större modellerna blir detta mycket tidskrävande. Optimera programmet genom att använda LU-faktorisering av A (MATLAB-kommandot `lu`). Lös problemet i c) för var och en av de fyra modellerna, med och utan LU-faktorisering. (Totalt 8 fall[2].) Bestäm tidsåtgången för varje fall. (Exklusive tiden för `load` och plottning!) *Sammanställ tiderna i en tabell.*

- Varför går det snabbare att lösa problemet med LU-faktorisering?
- För vilken modell blir tidsvinsten störst? Varför?

Skicka in: Filen `kanslighetLU.m` med programmet för den minsta modellen.

e) När en matris är *gles* kan man lösa ekvationssystemet med effektivare metoder än standard gausseliminering. Använd kommandot `spy(A)` för att förvissa er om att styvhetsmatrisen 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 anropas. Gå igenom beräkningarna i d) igen och undersök tidsåtgången när MATLAB använder dessa glesa lösare. *Komplettera tabellen i d) med två nya kolumner för de uppmätta tiderna.*

[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.

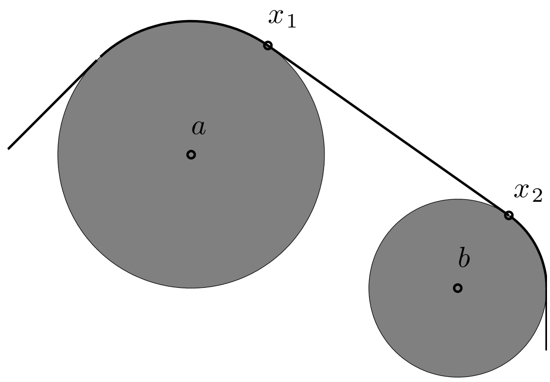
[2] Ni kan hoppa över de fall som tar orimligt lång tid.

- Vilken metod löser problemet snabbast? (Med/utan LU? Full/gles lösare?)
- Hur stor blir tidsvinsten mellan snabbaste och långsammaste metoden för minsta modellen?
- Vilket tar minst tid: minsta modellen med långsammaste metoden, eller största modellen med snabbaste metoden?

Skicka in: Tabellen `tidtabell` över de uppmätta tiderna för alla fyra metoderna, för de modeller som inte tar orimligt mycket tid. Det blir 4 kolumner (Full, Full LU, Sparse, Sparse LU) och 4 rader (Modell 1-4). Formatet kan vara vanligt textformat eller PDF (men ej `.doc`, `.rtf` eller `.odt`).

3. 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, 29/1)

- Låt $\boldsymbol{x}_1 = (x_1, y_1)$ och $\boldsymbol{x}_2 = (x_2, y_2)$. Skriv om ekvationssystemet 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 jakobian-matrisen till $\boldsymbol{F}$.

b) Skriv en MATLAB-funktion som löser ekvationsystemet med Newtons metod. Funktionen ska ta som indata en startgissning på vektorn $\mathbf{X} = (x_1, y_1, x_2, y_2)^T$, en feltolerans τ och värden på parametrarna för det aktuella fallet, dvs radierna r_a, r_b och mittpunkterna $\boldsymbol{a} = (x_a, y_a)^T$, $\boldsymbol{b} = (x_b, y_b)^T$. Funktionen ska returnera en lösningsvektor $\mathbf{X}_{\text{rot}}$ med ett fel mindre än τ. Funktionen ska också skriva ut mellanresultat (tex skillnaden mellan successiva iterationer) som visar att implementationen har kvadratisk konvergensordning.

Lös ekvationssystemet för fallet $r_a = 1.5$, $r_b = 0.8$, $\boldsymbol{a} = (-1.5,\ 0.5)^T$, $\boldsymbol{b} = (2,\ 0)^T$ och toleransen 10^{-10}. Plotta cirklarna och snöret i en figur och skriv ut svaret (punkternas koordinater).

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

Skicka in: Funktionsfilen `punkter.m`. Den ska kunna anropas som "`Xrot=punkter(X0,ra,rb,a,b,tau)`", där `Xrot, X0` är en kolumnvektorer med 4 element, `a, b` är kolumnvektorer med två element och `ra, rb, tau` är skalärer. (OBS! Var noga med att använda kolumnvektorer, inte radvektorer.)

4. Numerisk integration

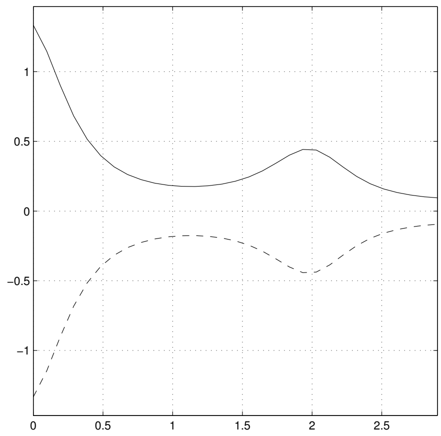
Konturen för en rotationssymmetrisk lur definieras av funktionskurvan

$$y(x) = \frac{8}{(2+x^2)(6-3\cos(\pi x))}, \quad 0 \le x \le L,$$

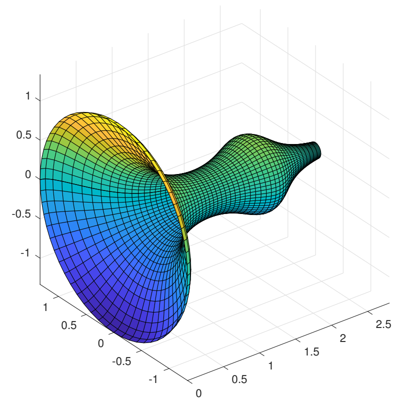
där L är lurens längd. Luren uppstår genom att kurvan roteras kring x-axeln och rotationsvolymen är

$$V = \pi \int_0^L y(x)^2\, dx.$$

Vi önskar beräkna volymen för en lur med längden $L = 2.8$.



a) Implementera de sammansatta versionerna av trapetsregeln och Simpsons formel. Skriv två MATLAB-funktioner som tar antal delintervall n som indata och returnerar respektive integralapproximation. (Steglängden $h = L/n$. Notera att n måste vara ett jämnt tal för Simpsons metod.) Beräkna integralen med båda metoderna, för $n = 40$. Gör sedan en tillförlitlighetsbedömning genom att också beräkna integralen för $n = 80$ med båda metoderna, och jämföra resultaten.



- Hur många decimaler verkar tillförlitliga i resultaten för de två metoderna?
- Vad hade kunnat sägas om tillförlitligheten om man bara beräknat integralen med *en* steglängd?

Skicka in: Funktionsfilerna `trapets.m` och `simpson.m`. De ska kunna anropas som "`V=trapets(n)`" respektive "`V=simpson(n)`". Variablerna `n` och `V` ska båda vara skalärer.

b) Definiera approximationsfelet $E_h = |V - V_h|$ där V_h är integralapproximationen med steglängd h. Undersök nu mer precist hur E_h för metoderna beror på steglängden h, på två olika sätt.

1. Plotta E_h som funktion av h för trapetsregeln och Simpsons metod i samma figur. Använd MATLABs kommando `loglog`, gärna tillsammans med `grid`-kommandot. (Gör först en mycket noggrann referenslösning med Simpsons metod[3] som ni använder för att beräkna felet.) Notera att antal delintervall måste vara jämnt i Simpson-fallet.

   - Uppskatta båda metodernas noggrannhetsordning med hjälp av figuren. Stämmer det med teorin?

   Skicka in: Figuren `konvergensplot`, antingen i `.fig`-format (gör Save i figur-fönstret) eller i `.png`-format (gör `print -dpng konvergensplot.png` vid prompten).

[3] Eller med MATLABs inbyggda funktion `integral`. Default-noggrannheten i `integral` är dock för låg i detta fall, och måste i så fall ökas.

2. Beräkna för en följd av små h kvoten

$$\frac{V_h - V_{h/2}}{V_{h/2} - V_{h/4}},$$

och uppskatta noggrannhetsordningen från dessa värden.[4] Sammanställ en tabell med kvoter och motsvarande noggrannhetsordning för trapetsregeln och för Simpsons metod. Notera att ingen referenslösning behövs.

Skicka in: Tabellen `kvottabell` över kvoterna och motsvarande noggrannhetsordning för båda metoderna. Formatet kan vara vanligt textformat eller PDF (men ej `.doc` eller `.odt`).

5. Sammanställning av filer som ska skickas in

Uppgift 1: `fixpunkt.m`, `newton.m`

Uppgift 2: `tidsplot` (som .`fig` eller .png), `kanslighet.m`, `kanslighetLU.m`, `tidtabell` (som .txt eller .pdf)

Uppgift 3: `punkter.m`

Uppgift 4: `trapets.m`, `simpson.m`, `konvergensplot` (som .`fig` eller .png), `kvottabell` (som .txt eller .pdf)

[4] Se föreläsningsanteckningarna om noggrannhetsordning.