KTH Engineering Sciences

Laboration 2

Egenvärden, approximation 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. En sammanställning ges här:*

Uppgift 1: `egenmoder.png`, `egentabell`, `invpot.m`

Uppgift 2: `baninterp.m`

Uppgift 3: `dollar.m`

Uppgift 4: `trapets.m`, `simpson.m`, `trapets2d.m`, `konvergens.m`

Uppgift 5: `mcinteg.m`

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!*

OBS! Era program ska inte använda MATLAB*s symboliska funktioner/variabler (*`syms`*) för beräkningar. Använd inte heller* MATLAB *Live Script (*`.mlx`*-filer).*

1. Egenvärden

I denna uppgift ska ni beräkna egenvärden och egenvektorer till matriserna i Eiffeltornsmodellen från Laboration 1. För dessa matriser motsvarar detta tornets svängningsmoder.

a) Bakgrund. Vi såg i Laboration 1 att förskjutningen från jämviktsläget gavs av ekvationen $A\boldsymbol{x} = \boldsymbol{F}$ när $\boldsymbol{F}$ var den yttre kraften på noderna. Allmänt, när systemet inte är jämvikt kommer kraften på varje nod vid förskjutningen $\boldsymbol{x}$ ges av $\boldsymbol{F}_{\text{nod}} = -A\boldsymbol{x}$. (Så att $\boldsymbol{F}+\boldsymbol{F}_{\text{nod}} = 0$ vid jämvikt.) I det tidsberoende fallet beror förskjutningarna på tiden $\boldsymbol{x} = \boldsymbol{x}(t)$, och de följer Newtons andra lag att kraften är lika med massan gånger accelerationen, dvs

$$\boldsymbol{F}_{\text{nod}} = m\frac{d^2\boldsymbol{x}}{dt^2} \quad \Rightarrow \quad m\frac{d^2\boldsymbol{x}}{dt^2} + A\boldsymbol{x} = 0, \tag{1}$$

där m är en effektiv massa för noderna (en liten förenkling av verkligheten) som vi antar är lika med ett, $m = 1$. En lösning till denna ODE ges av den oscillerande funktionen

$$\boldsymbol{x}(t) = \sin\left(t\sqrt{\lambda}\right)\boldsymbol{y}, \tag{2}$$

där λ, $\boldsymbol{y}$ är ett egenvärde respektive en egenvektor till A-matrisen. Detta gäller för alla par av egenvärden och egenvektorer. Egenmoderna till A beskriver därför möjliga svängningar i fackverket. Egenvärdet λ motsvarar svängningens *frekvens* (i kvadrat) och egenvektorn $\boldsymbol{y}$ förskjutningens *amplitud* från jämviktsläget i varje nod.[1]

- Visa att (2) är en lösning till (1). (Med papper och penna.) Ange också mer precist hur frekvensen f ges av egenvärdet, $f = \ldots$. (Detta samband behövs i (b) nedan.)

b) Visualisering För att visualisera egenmoderna, välj den minsta modellen och använd MATLAB-kommandot `eig` för att beräkna egenvärdena och motsvarande egenvektorer. Notera att `eig` inte returnerar egenvärden/egenvektorer i storleksordning. Använd därför `sort`-kommandot på lämpligt sätt för att få dem i rätt ordning. Var noga med att matcha rätt egenvektorer till egenvärdena efter sorteringen. [2]

Använd nu `trussplot` for att plotta tornet med noderna förskjutna enligt egenvektorerna. Om egenvektorn är `y` blir plottningen tex

```
>> trussplot(xnod+Y(1:2:end), ynod+y(2:2:end), bars);
```

Detta visar egenmodens karaktäristiska form.[3] Plotta de fyra lägsta egenmoderna på detta sätt och ange frekvensen för var och en av dem. De har ganska enkla former. Prova sedan också några högre moder, med mer komplicerade former. Med scriptet `trussanim.m`, som finns på kurshemsidan, kan du slutligen animera svängningsmoderna med förskjutningen $\boldsymbol{x}(t)$ där t är tiden. Syntaxen är

```
>> trussanim(xnod, ynod, bars, y);
```

c) Beräkning av egenvärden. Egenvärdena kan beräknas med MATLABs `eig`-kommando. För de större modellerna tar detta emellertid lång tid. Ni ska istället använda olika varianter av *potensmetoden* för att beräkna utvalda egenvärden.

1. Börja med att beräkna det *största* egenvärdet med vanliga potensmetoden för de fyra modellerna. Använd toleransen 10^{-10} och ange egenvärdena samt antal iterationer som krävts.

2. Beräkna sedan det *minsta* egenvärdet med *inversa potensmetoden* för de fyra modellerna. Använd toleransen 10^{-10} och ange egenvärdena samt antal iterationer som krävts.

   Notera att inversa potensmetoden kan snabbas upp genom att använda `sparse`-format och LU-faktorisering på samma sätt som i Laboration 1. Gör här också en tidsstudie där du jämför tidsåtgången för att göra `eig`, naiv metod (ej gles+LU) och optimerad metod (med gles+LU).

3. Använd slutligen *inversa potensmetoden med skift* för att hitta egenvärdena till den minsta modellen (`eiffel1.mat`) som ligger närmast 10, 50 och 67. Använd toleransen 10^{-10} och ange egenvärdena samt antal iterationer som krävts.

[1] Den allmänna lösningen till differentialekvationen är en superposition av alla möjliga sådana svängningar:

$$\boldsymbol{x}(t) = \sum_{k=1}^{2N} \left[\alpha_k \sin\left(t\sqrt{\lambda_k}\right) + \beta_k \cos\left(t\sqrt{\lambda_k}\right)\right] \boldsymbol{y}_k,$$

där $\lambda_k, \boldsymbol{y}_k$ är egenvärdena/vektorerna till A och α_k, β_k är konstanter som bestäms av begynnelsedata.

[2] `sort`-kommandot returnerar två argument, där det andra ger en lista på index som är bra att använda.

[3] Om inte förskjutningen syns bra, prova att göra egenvektorn längre genom att multiplicera den med ett tal större än ett.

- Skiljer sig de högsta respektive de lägsta egenvärdena (egenfrekvenserna) mycket för de olika modellerna?
- Vad beror skillnaden i antal iterationer på? Hur hänger det ihop med avstånden mellan egenvärdena? Stämmer teorin?
- Hur skulle man kunna hitta egenvärdet nära 67 snabbare?

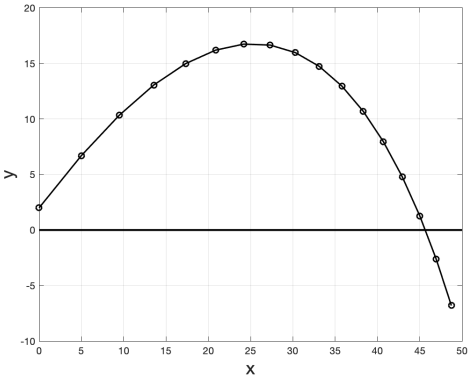
Skicka in: Figuren `egenmoder.png` som innehåller plotten med de fyra lägsta egenmoderna och frekvenser som efterfrågas i deluppgift (b). (Använd tex `subplot`-kommandot och `"print -dpng egenmoder.png"`.) För (1) och (2) i deluppgift (c), skicka in tabellen `egentabell` av typen[4]

	Egenvärde		Antal iterationer		Tid (minsta egenvärdet)		
	största	minsta	största	minsta	`eig`	Naiv	Gles+LU
`eiffel1.mat`							
`eiffel2.mat`							
`eiffel3.mat`							
`eiffel4.mat`							

För (3) i deluppgift (c), skicka in programmet `invpot.m` som löser uppgiften.

2. Interpolation

På hemsidan finns funktionen `kastbana.m` som ger banan för ett kast med en liten boll. Funktionen använder en mycket noggrann numerisk metod för att beräkna banan. Den returnerar bollens position och hastighet vid ett antal ekvidistanta tidpunkter $t_n = nh$. Funktionen anropas som `"[t,x,y,vx,vy]=kastbana(h)"` där `h` är steglängden mellan tidpunkterna och returvärdena är följande vektorer:



- `t` – vektor med tidpunkterna.
- `x, y` – vektorer med bollens koordinater vid tidpunkterna.
- `vx, vy` – vektorer med bollens hastigheter i $x-$ och y-led vid tidpunkterna.

Funktionen beräknar banan fram till tiden $t = 5$. En typisk kastbana $y = y(x)$ visas i bilden ovan, där $h = 0.25$ användes. För mer information gör `help kastbana`. Er uppgift är att skriva en funktion `baninterp.m` som med hjälp av `kastbana.m` och (styckvis) interpolation beräknar tre värden:

- x-koordinaten för nedslagsplatsen (där $y = 0$).
- x-koordinaten för banans högsta punkt.
- y-koordinaten för banans högsta punkt.

[4]Formatet kan vara vanligt textformat eller PDF (men ej .doc, .rtf eller .odt).

Funktionen ska kunna göra detta för ett generellt h, med både *linjär* och *kvadratisk* interpolation. Ni får inte använda MATLABs inbyggda interpolations-funktioner (`polyfit`, `polyval`, `interp1`,...) eller ekvationslösare (`roots`, `fzero`, `fminsearch`, ...) i denna uppgift.[5] Interpolationen kan göras på flera olika sätt (speciellt för högsta punkten). Notera att både punktvärdena (x, y) och hastighetsvärdena (v_x, v_y) kan användas.

- Prova att använda olika h-värden och jämför resultaten för metoderna.

Frivillig utökning: Implementera också *Hermite-interpolation.* Ni behöver då använda hastighetsvärdena; tänk på att $dy/dx = (dy/dt)/(dx/dt)$. (I utökningen får ni använda det inbyggda kommandot `roots`.)

Skicka in: Funktionsfilen `baninterp.m`. som ska kunna anropas som `"[xn,xm,ym]=baninterp(h,typ)"` där `xn`, `xm`, `ym` är nedslagsplats och högsta punkt, `h` är steglängden och `typ` är 1 för linjär interpolation, 2 för kvadratisk interpolation (och 3 för Hermite-interpolation).

3. Minstakvadratmetoden

På kurshemsidan finns finns datafilen `dollarkurs.mat` som innehåller dollarkursen per dag från 1:a jan 2009 till 31:a dec 2010. Läs in filen i MATLAB med kommandot `load`.

a) Anpassa en linjär modell, $f(t) = c_1 + c_2 t$, i minstakvadratmening till dollarkursen genom att ställa upp och lösa lämpligt linjärt ekvationssystem för de okända koefficienterna c_i. Plotta data tillsammans med den anpassade kurvan.

b) Plotta felet mellan den linjära modellen och data. Beräkna medelkvadratfelet

$$E = \frac{1}{N} \sum_{i=1}^{N} (X_i - f(i))^2,$$

där X_i betecknar de N värdena på dollarkursen.

c) Felet i den linjära modellen tycks vara periodiskt. Uppskatta periodlängen L från plotten i förra deluppgiften. Anpassa därefter följande modell till dollarkursen:

$$f(t) = d_1 + d_2 t + d_3 \sin(2\pi t/L) + d_4 \cos(2\pi t/L).$$

Plotta resultatet och felet, samt ange medelkvadratfelet.

d) Låt nu även L vara en okänd parameter. Modellen blir då olinjär. Hitta hela den parameteruppsättningen d_1, d_2, d_3, d_4, L som ger bäst approximation av data i minstakvadratmening. Använd Gauss-Newtons metod med värdena från deluppgift (c) som startgissning. Plotta data och de tre modellernas anpassning i samma figur. Beräkna medelkvadratfelet.

- Hur mycket steg eller föll dollarkursen per dag under perioden enligt den linjära modellen? Ger de andra modellerna väsentligt annorlunda resultat för den långsiktiga trenden?
- Jämför medelkvadratfelen i de olika modellerna. Vilket är störst/minst?

Skicka in: Programmet `dollar.m` som genererar de figurer och skriver ut de värden som efterfrågas i uppgiften.

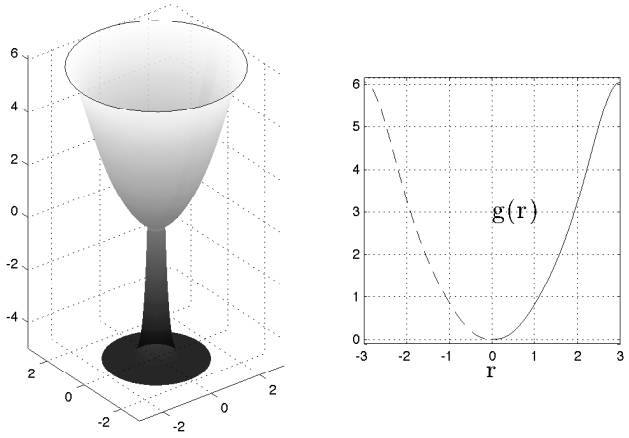
[5] Använd dem dock gärna för att kolla att er kod räknar rätt!

4. Numerisk integration

Ett glas har en cirkulär kropp och en öppning med radien $R = 3$ cm. Dess kontur beskrivs av funktionen

$$g(r) = \frac{3r^3 e^{-r}}{1 + \frac{1}{3}\sin\left(\frac{r\pi}{2}\right)},$$

där r är avståndet till öppningens mitt i centimeter; se figur. Glasets volym V [cm^3] ges av en dubbelintegral. Med polära koordinater reducerar den till en enkelintegral,



$$V = \int_0^R \int_0^{2\pi} (g(R) - g(r)) r d\theta dr = 2\pi \int_0^R [g(R) - g(r)] r dr = V_0 - 2\pi \int_0^R g(r) r dr.$$

där $V_0 = g(R)R^2\pi$.

a) Implementera de sammansatta versionerna av trapetsregeln och Simpsons formel för integralen ovan. Skapa funktionsfilerna `trapets.m` och `simpson.m` som ska kunna anropas som `"V=trapets(n)"` respektive `"V=simpson(n)"` där variablerna `n` och `V` båda är skalärer. Beräkna glasets volym med båda metoderna, för $n = 30$ och $n = 60$, (dvs $h = 0.1$ och $h = 0.05$ cm).

- 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?

b) Definiera approximationsfelet $E_h = |V - V_h|$ där V_h är integralapproximationen med steglängd h. Använd funktionerna från deluppgift (a) för att mer precist undersöka hur E_h beror på steglängden h, på två olika sätt.

1. Gör en mycket noggrann referenslösning $\tilde{V}$ med Simpsons metod[6] och låt $E_h = |\tilde{V} - V_h|$. Plotta sedan E_h som funktion av h för trapetsregeln och Simpsons metod i samma figur. Använd MATLABs kommando `loglog`, gärna tillsammans med `grid`-kommandot. Notera att antal delintervall måste vara jämnt i Simpson-fallet.

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

2. Bestäm noggrannhetsordningen *utan att använda en referenslösning* genom att beräkna kvoten

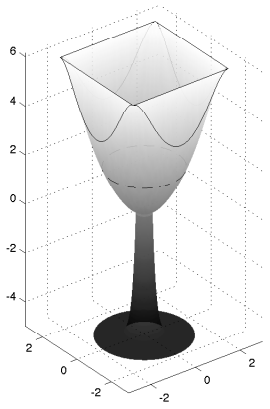
   $$\frac{V_h - V_{h/2}}{V_{h/2} - V_{h/4}},$$

   för en följd av små h och uppskatta noggrannhetsordningen från dessa värden.[7] Sammanställ en tabell med kvoter och motsvarande noggrannhetsordning för trapetsregeln och för Simpsons metod.

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

[7] Se föreläsningsanteckningarna om noggrannhetsordning.

c) I en alternativ design av glaset har man fasat av sidorna och gjort öppningen kvadratisk med sidan $L = 3\sqrt{2}$ cm; se figur. För att beräkna volymen måste man nu lösa hela dubbelintegralen numeriskt,



$$V = \int_{-L/2}^{L/2} \int_{-L/2}^{L/2} g(R) - g\left(\sqrt{x^2+y^2}\right)\, dxdy.$$

Implementera trapetsregeln i två dimensioner och beräkna volymen av glaset. Skapa funktionsfilen `trapets2d.m` som anropas `"V=trapets2d(n)"` där $h = L/n$. Verifiera att noggrannhetsordningen för implementationen är två, genom att plotta felet eller betrakta kvoterna som i deluppgift (b) ovan.

Skicka in: Funktionsfilerna `trapets.m`, `simpson.m`, `trapets2d.m` samt ett script `konvergens.m` som, genom att anropa de andra filerna, genererar de figurer och skriver ut de värden som efterfrågas i uppgiften.

5. Högdimensionell numerisk integration

I denna uppgift ska ni beräkna följande tiodimensionella integral på två olika sätt,

$$I = \int_\Omega e^{x_1x_2x_3x_4x_5x_6x_7x_8x_9x_{10}} d\boldsymbol{x}, \qquad \boldsymbol{x} = (x_1, \ldots, x_{10})^T, \qquad \Omega = [0,\, L]^{10}, \qquad L = 1.2.$$

a) På hemsidan finns funktionen `trapets10d.m` som beräknar integralen med trapetsregeln generaliserad till tio dimensioner. Den anropas som `"I=trapets10d(n)"` där $h = L/n$. (Studera gärna koden.) Använd funktionen för att beräkna integralvärdet. Ni kommer inte kunna ha speciellt liten steglängd eftersom beräkningskostnaden ökar mycket snabbt med n.

- Hur litet fel kan ni få? (Inom rimlig tid.)
- Hur lång blir beräkningstiden?

Ledning: Ett bra närmevärde till I ges av $I \approx 6.231467927023725$.

b) Beräkna integralvärdet med Monte–Carlo-integration. Plotta det approximerade integralvärdet som funktion av antal slumptal $n = 1, \ldots, N$ (dvs antal funktionsevalueringar). Det största antalet slumptal N ska vara minst 10^6, men gärna större. Plotta också felet som funktion av n i `loglog`-diagram.

Integralapproximationen och felet beror på slumptalen. Upprepa därför förfarandet för flera olika följder (minst 5) av slumptal. Detta ger olika *realiseringar* av approximationen. Plotta alla realiseringar tillsammans: en figur för integralapproximationerna och en figur för felen. Använd `axis`-kommandot för att zooma in på den intressanta delen i figuren med integralapproximationerna!

Ledning: Det behövs bara *en* följd av slumptal för att skapa integral- och en felplottarna för en realisering. (Inte en följd per plottad punkt!) Generera alla $N \times 10$ nödvändiga tal på en gång med `rand`-kommandot. Beräkna sedan approximationerna för $1 \le n \le N$ från dessa värden. MATLAB-kommandona `prod` och `cumsum` kommer att vara användbara.

- Baserat på plottarna, uppskatta hur stort n man måste ta för att, med stor sannolikhet, få ett fel mindre än felet ni fick med trapetsregeln. Jämför beräkningstiderna.
- Hur litet fel kan ni få som bäst?
- Överensstämmer felets avtagande med teorin? Motivera!
- Förklara skillnaden i hur felet beror på antal funktionsevalueringar n för Monte–Carlo och trapetsregeln.

Skicka in: Programmet `mcinteg.m` som genererar de figurer och skriver ut de värden som efterfrågas i deluppgift (b).

6. Frivillig uppgift: Konvergensstudie för styckvis interpolation

Gå tillbaka till uppgift 2 och gör en konvergensstudie för den interpolerade nedslagsplatsen och för maxpunkten. Beräkna först noggrannna referensvärden på dessa genom att välja ett mycket litet h i `baninterp.m`. Bestäm sedan felen i de tre kvantiteterna för de två (tre) metoderna för *alla* värden på $h = 5/n$ när $n = 5, 6, 7, \ldots, 200$. Plotta felkurvor och försök förklara resultaten! En del överraskningar bör dyka upp. Vad blir noggrannhetsordningarna? Är felen "snälla" eller inte? Hur skulle felkurvorna se ut om man plottade max-felet i styckvis linjär interpolation?