Projektuppgifter

IX1303 Algebra och Geometri, VT2023, period 4.

Målsättningen med denna inlämningsuppgift är att du ska få en ökad förståelse för kursinnehållet, bli ännu bättre på problemlösning och, inte minst, att du övar upp dina färdigheter i att använda datorn för att lösa problem.

Projektet innehåller 3 uppgifter som ska göras i Matlab. För godkänt på uppgiften ska ni göra alla tre programmerings uppgifter korrekt, svara på specifika frågor, och ladda upp era resultat i Canvas. Dessutom måste ni ladda upp den data ni använde i uppgift 2 under en separat inlämning ”***CO2 Data till projekt 2***”. Om ni får kommentarer på er lösning måste ni komplettera lösningen enligt kommentarerna för att få godkänt. Notera att era inlämnade svar rättas efter hand, i mån av tid, vilket betyder att det kommer att dröja till mitten av juni innan alla fått svar.

Lösningarna ska laddas upp senast 2023-06-05 om ni vill ha betyget godkänt i Ladok innan sommaren. Det går även att lämna in lösningar senare, men då kan vi inte garantera att de rättas innan sommaren.

Till varje uppgift får ni en ”template”-fil där delar av koden redan är given. Er uppgift är att fylla i vad som saknas, samt svara på olika frågor. Notera att frågorna redan är inskrivna.

Ni får naturligtvis arbeta tillsammans med andra, men alla som önskar få godkänt på kursen måste lämna in egna personliga lösningar. Om flera liknande svarsfiler lämnas in, där det finns anledning att misstänka att dessa har kopierats, riskerar samtliga som lämnat in sådana filer att anmälas för plagiering.

Betyget på inlämningsuppgiften blir godkänt eller inte godkänt (pass, or fail) och för godkänd inlämningsuppgift får ni 1.5 hsp (vilket motsvarar 1 arbetsvecka).

Lycka till!
/Thomas, Per och Henric

Uppgift 1: Linjära ekvationssystem med många obekanta

När man löser linjära ekvationssystem på formen $A\mathbf{x} = \mathbf{b}$ med många obekanta är det viktigt att man använder en effektiv metod. Här kommer vi lösa ekvationssystem med upp ett par tusen obekanta, vilket inte är jättestort, men tillräckligt stort för att man ska kunna se vilka metoder som är bra och dåliga.

Er uppgift är att lösa minst fyra ekvationssystem mellan 3 och 3000 obekanta. Ni ska mäta tiden det tar att lösa varje system och generera grafer som visar denna tid som funktion av antalet obekanta. För varje ekvationssystem ska ni:

A. Skapa en matris, A, full med slumptal. För att matrisen ska bli inverterbar använder vi oss av teorem 11 från sid 167 i Lay upplaga 6 (från kapitlet om Leontief modellen). Därmed ska vår matris ska vara på formen $A = I - C$, där I är identitetsmatrisen (se funktionen *eye*) och C ska innehålla slumptal. En funktion som genererar slumptal är funktionen *rand*, som skapar en matris av slumptal mellan 0 och 1. För att teoremet ska vara uppfyllt ska a summan av värdena i varje kolumn i C vara mindre än 1, samt att C inte får innehålla några negativa tal. För en $n \times n$ matris, kan vi skriva $C = kR$, där $R =$ *rand(n)* och k är en skalär. För vilka värden på k är summan av alla kolumner mindre är 1?
B. Använd *rand* för att generera en vektor $\mathbf{b}$ full med slumptal. Notera att enligt teorem 11, som vi använde ovan, får $\mathbf{b}$ inte innehålla några negativa tal.
C. Lös ekvationssystemet $A\mathbf{x} = \mathbf{b}$ med hjälp av funktionen *mldivide*, som kan köras med kortkommandot ”\”. Denna metod använder både matrisen och högerledet för att hitta lösningar, på ett sätt som påminner om radreducering (Gauss-eliminering). För att mäta tiden det tar för datorn att lösa ekvationen kan man använda Matlab’s inbyggda tidtagningssystem. Tidtagning startas med funktionen *tic*, därefter löser du ekvationen, och till sist stänger ni av tiden med funktionen *toc*. Notera att om ni skriver `T(i)=toc` så sparas tiden i element `i` av vektorn `T` och därmed kan ni sedan använda värdet för att rita en graf med de tider ni mätt upp.
D. Lös ekvationssystemet $A\mathbf{x} = \mathbf{b}$ med hjälp av inversen till A. Här kan ni använda funktionen *inv*. Även här ska ni använda *tic* och *toc* för att mäta tiden det tar att lösa ekvationssystemet.
E. Ni ska kontrollera att resultaten från *mldivide* och *inv* är desamma. Notera att när en dator räknar med reella tal så räknar den inte exakt. Matlab har en upplösning på ungefär 16 decimaler, så 17:e decimalen kan bli fel! Därmed kommer resultaten inte vara exakt lika. Som ett mått på skillanden mellan de två lösningarna kan man första skapa differensen mellan vektorerna. Om du nu har 3000 värden är det svårt att gå igenom alla, så istället kan vi ta fram längden på differens-vektor. Längden på en vektor kan man beräkna med funktionen *norm*.

När ni löst minst fyra olika ekvationssystem ska ni skapa två grafer som visar era resultat. I den först grafen ska ni rita antalet obekanta på x-axeln och tiden det tog att lösa ekvationssystemen på y-axeln. Ni ska rita resultaten från *mldivide* och *inv* i samma graf. I den andra grafen ska ni rita samma sak, men här ska ni skala om både x-axeln och y-axeln till log-skala genom att lägga till raderna:

```
set(gca,'xscale','log')
set(gca,'yscale','log')
```

Nu återstår bara att analysera resultaten och svara på följande frågor.

1. Antag att du ska lösa ett problem med tre obekanta en eller ett par gånger. Hur väljer du metod? Är det viktigt att välja rätt metod?
2. Antag att du ska lösa ett problem med tre obekanta 10 000 gånger. Hur väljer du metod? Är det viktigt att välja rätt metod?
3. Antag att du ska lösa ett problem med 3000 obekanta en eller ett par gånger. Hur väljer du metod? Är det viktigt att välja rätt metod?
4. Kör om alla räkningar tre gånger. Varför får du olika resultat varje gång du kör programmet?
5. Uppskatta den relativa skillnaden i beräkningstid mellan de två metoderna för 3000 obekanta?

Uppgift 2: Minstakvadratmetoden och CO_2 mätningar

Här ska vi analysera mätningar av halten av koldioxid i atmosfären med minstakvadratmetoden. För detta ska du själv hämta data från en öppen databas. Här rekommenderar jag att ni fokuserar på data från de senaste 60-70 åren, d.v.s. den data brukar kallas Keelingkurvan. Det finns även annan data över kortare eller längre perioder, men denna data inte kan approximeras med enkla polynom och passar därför inte lika bra in i här uppgiften. Om ni väljer att använda denna form av data så får ni som extrauppgift; ni ska hitta en funktions-bas som bättre beskriver er data.
Den mätdata ni ska använda kan ni hämta från till exempel https://scrippsco2.ucsd.edu. Här kan ni hitta filer på ”csv” format, vilka kan läsas med funktionen *readtable*, som returnerar ett table-objekt. Ett exempel på hur *readtable* kan användas är om man vill ta fram kolumnerna ”Date1” och ”CO2” från en fil *monthly_in_situ_co2_mlo.csv*:

```
TABLE=readtable('monthly_in_situ_co2_mlo.csv'); % Skapar table-objekt
t=[TABLE.Date1];
y=[TABLE.CO2];     % Hämtar kolumnen Date1 från TABLE och
                   % gör om till en vektor mha ”[…]”.
```

Notera att det allmänt kan vara krångligt att räkna med datum (tex, hur många dagar är det mellan 21 november 2013 och 1 maj 2023), men i vissa av csv filerna finns det datum på decimalform som kan vara användbara!

Eran uppgift är:

A. Att hämta data från CO_2 mätningar från en öppen databas. Notera att denna data ska laddas upp på Canvas under uppgiften ”CO2 Data till projekt 2”
B. Läs in datafilen i Matlab och finn en tidsvektor, **t**, samt en datavektor, **y**, med resultat från CO_2 mätningar. Notera att om ni använder ett annat format än csv har ni själva ansvar för att läsa och processa den data ni använder. Ni måste dessutom gå igenom filen själva för att se till att den data ni använder motsvarar just en tidsvektor, samt en relevant datavektor.
C. Skapa en minstakvadrat anpassning av $\mathbf{y}(\mathbf{t})$ till en rät linje. Rita både mätdata och anpassningen i samma graf.
   VIKTIGT: Här är det inte tillåtet att använda färdiga funktioner för

kurvanpassning eller optimering. Anpassningen ska istället skapas genom multiplikationer, additioner och subtraktioner av matriser, samt lösning av ekvationssystem med hjälp av någon av funktionerna *mldivide*, eller *inv*.

D. Skapa en minstakvadrat anpassning av $\mathbf{y(t)}$ till ett andragradspolynom som funktion av tiden. Rita både mätdata och anpassningen i samma graf.
VIKTIGT: Här är det inte tillåtet att använda färdiga funktioner för kurvanpassning eller optimering. Anpassningen ska istället skapas genom multiplikationer, additioner och subtraktioner av matriser, samt lösning av ekvationssystem med hjälp av någon av funktionerna *mldivide*, eller *inv*.

E. Skapa en minstakvadrat anpassning av $\mathbf{y(t)}$ till ett tredjegradspolynom som funktion av tiden. Rita både mätdata och anpassningen i samma graf.
VIKTIGT: Här är det inte tillåtet att använda färdiga funktioner för kurvanpassning eller optimering. Anpassningen ska istället skapas genom multiplikationer, additioner och subtraktioner av matriser, samt lösning av ekvationssystem med hjälp av någon av funktionerna *mldivide*, eller *inv*.

F. (*Bara för er som inte använder CO_2 data från de senaste 60-70 åren*)
Om dessa anpassningar inte ger en rimlig uppskattning av er CO_2 data så får ni föreslå en annan funktions-bas som ger en bättre beskrivning av er data (kanske lägga till en exponential- eller en sinus funktion). Använd den nya basen och gör en anpassning, samt visa resultat i en graf.

Frågor:

1. Beskriv med egna ord hur de tre kurvorna beskriver mätdata. Framför allt, blir lösningen lite eller mycket bättre när vi går från en rät linje till en andragradsfunktion? Blir den mycket bättre när vi går från en andragradsfunktion till en tredjegradsfunktion?
2. Kan man använda dessa anpassningar för att uppskatta utsläppen om 2 år? Motivera ditt svar.
3. Kan man använda dessa anpassningar för att uppskatta utsläppen om 50 år? Motivera ditt svar.

Uppgift 3: Datorgrafik och avbildningar

Här kommer ni skapa en film med en sträckgubbe, *Dash-man*, som förflyttas på skärmen med hjälp av linjära avbildningar. Mer specifikt ska ni hitta avbildningar så att det ser ut som om gubben faller från himlen. Här har ni fått lite hjälp och en del kod är redan skriven; för att lösa uppgiften behöver ni filerna DashMan.m, plotDashMan.m and addFrameToGif.m. I dessa filer finns det mesta av vad som ska programmeras. Det som återstår är framför allt att skapa matriser och att göra avbildningar.

Ni har även fått en template fil IX1303_VT2023_Projekt3_DashMan.m som ni ska skriva klart. Läs igenom filen för att skapa en bild av hur programmet fungerar och vilka uppgifter ni ska göra. Om ni kör programmet kommer ni skapa ett object `D` som beskriver er Dash-man. Om ni därefter skriver `D` i kommandoprompten kommer ni se vilka kroppsdelar han har. Ni kan sedan se vilka punkter som beskriver kroppen genom att

skriva `D.body`, eller benen med `D.legs`, *osv*. Här innehåller varje kolumn en punkt i xy-planet samt en homogen koordinat.

Notera att ni kan köra programmet redan från början och den kommer att skapa en film, men i den filmen står stäckgubben stilla!

Här ska ni skapa en film där vi flyttar gubben en kort sträcka mellan varje bild. Alla förflyttningar ska beskrivas av samma sammansatta linjär avbildning. Denna avbildning består av tre delar; först roterar vi gubben med en liten vinkel, därefter translaterar ni gubben en kort sträcka i xy-planet, och till sist gör ni gubben lite större. Vi återkommer till hur värdena på rotationen, translationen och förstoringen ska väljas.

Eran uppgift:

A. Bekanta er med streckgubben så att ni förstår vad rader och kolumner betyder och därmed förstår hur man kan transformera gubben (dvs punkterna som beskriver gubben) genom linjära avbildningar.
B. Skapa en rotationsmatris. Notera också att Matlabs sin- och cos-funktioner förväntar sig att vinklar är givna i radianer och inte i grader.
   Här ska gubben rotera mellan 3 och 4 varv under filmen. Använd att filmen genereras i en loop som repeteras 50 gånger för att beräkna rotationen mellan två bilder.
C. Skapa en translationsmatris som flyttar objekt i xy-planet. Till att börja med bör ni flytta gubben en mycket kort sträcka; vi ska studera olika translationer senare.
D. Skapa en förstoringsmatris som förstorar objekt i xy-planet. Här bör figuren bli en faktor mellan 2 och 4 gånger större under filmen.
E. Skapa en sammansatt matris som beskriver resultatet om man först roterar, sen translaterar och till slut förstorar.
F. Inne i for-loopen ska ni flytta streckgubben med hjälp av den sammansatta matrisen. Här måste ni flytta varje kroppsdel för sig. Kör sedan koden och se hur gubben rör på sig. Roterar den rätt antal varv?
   Notera att om man ska avbilda flera kolumnvektorer, v_1, v_2 ... med en matris A så kan man skapa en matris $V = [v_1, v_2 \ldots]$ och avbilda hela matrisen, $V \rightarrow AV$.
G. Nu ska ni laborera med translationen. Jämför resultaten om ni translaterar uppåt, nedåt, åt vänster och höger.
   Notera att när du kör koden så ändrar sig koordinatsystemet hela tiden. För att undvika detta kan ni först specificera hur koordinat-axlarna ska se ut genom att aktivera ”`axis(BoundingBox)`” (dvs ta bort tecknet ”%” i början av raden) på två ställen i koden. Dessutom måste ni hitta rätt värden på BoundingBox. För att hitta detta värde kan ni se vilka de största värdena vi har på x- och y-axlarna under animationen. Om detta värde är 14 (eller -14) så sätter ni
   ```
   BoundingBox = [-1,1,-1,1]*14;
   ```
   När ni har satt bestämda koordinat-axlar kan ni gå vidare och slå av axlarna helt genom `set(gca,'visible','off')`.
H. Finn värden på translationen, rotationen, samt förstoringen så att ni får en snygg film där det ser ut som om Dash-man faller från himmelen.

Frågor:

1. Varför innehåller sista raden i D.head bara ettor?
2. Beskriv skillnaden i gubbens rörelse över flera varv (d.v.s banan gubben rör sig längs) när man translaterar uppåt, neråt, åt höger eller vänster?
3. Om man flyttar gubben en sträcka $dx = 0.1$ per steg, och vi tar 50 steg med kombinerad translation och rotation, varför har gubben inte flyttats 50*0.1 åt höger?