Linjär algebra E, vt 06
Vecko–PM läsvecka 6

Lay:6.1-6.6 Ortogonalitet, projektion och minstakvadratmetoden

I kapitel **6.1** införs *skalärprodukt, dot product* och *längd*, eller *norm* som kanske är ett bättre namn. Dessa motsvarar skalärprodukt och längd för geometriska vektorer då vektorerna ges i en ON-bas. Begreppet *ortogonalitet* är viktigt. Ortogonala komplementet till ett underrum i $\mathbb{R}^n$ är ett nytt begrepp jämfört med det vi studerade i inledande kursen. Tänk på att ett underrum kan vara t.ex ett plan genom origo. Ortogonala komplementet är i så fall den linje som går genom origo och är vinkelrät mot planet.

I avsnitt **6.2** är målet att definiera vad som menas med en ON-bas, en ortonormerad bas. **Sats 4** säger att ortogonalitet garanterar linjärt oberoende. Sats 5 visar hur lätt det är att bestämma koordinater i en sådan bas. Ännu enklare är det i en ON-bas, då är $x_k = \boldsymbol{x} \cdot \boldsymbol{e}_k$ Projektionsformeln som ger projektionen av en vektor på en annan ser likadan ut i $\mathbb{R}^n$ som den ”gamla” från inledande kursen, motiveringen till formeln kan emellertid inte se likadan ut (vi har inga vinklar i $\mathbb{R}^n$). En vidareutveckling kommer i sats 8 och sats 10 i **6.3**. Begreppet ortogonala matriser som nämns i förbigående i 6.2 kommer att användas mycket i kapitel 7.
I **6.4** ges Gram-Schmidt processen för att stegvis bestämma en ortogonal bas för ett underrum W då man har en annan bas för W. Metoden är enkel, det gäller att i varje steg använda projektionsformeln för att ersätta en basvektor med en som är ortogonal mot de redan bestämda basvektorerna. Gram-Schmidt processen leder till den numeriskt intressanta QR-faktoriseringen av matriser.
I avsnitt **6.5** ges den mycket använda *minstakvadrat-metoden* för att finna *bästa möjliga lösning* till ett ekvationssystem *som saknar lösning*. I synnerhet handlar det då om överbestämda system, sådana med fler ekvationer än obekanta. I Matlab ges denna lösning tll `Ax = b` av `x=A\b`.
Sats 13 ger metoden i en liten ask. Däremot har kanske inte sats 14 direkt räknemässigt bruk, det viktiga är att veta att om kolonnerna i A är linjärt oberoende så har ekvationen i sats 13 entydig lösning.
I boken definieras *minstakvadrat-felet* som $\|A\hat{\boldsymbol{x}} - \boldsymbol{b}\|$ då $\hat{\boldsymbol{x}}$ är minsta-kvadrat lösningen. Ofta använder man istället *kvadratiska medelfelet* som är $\|A\hat{\boldsymbol{x}} - \boldsymbol{b}\| / \sqrt{n}$ då n är antalet ekvationer i systemet. I **6.6** tillämpas metoden på t.ex anpassning av linje till mätpunkter. Minsta kvadrat-felet ökar då i allmänhet om man lägger till mätpunkter under det att kvadratiska medelfelet t.o.m. kan minska.

Rekommenderade uppgifter

(PP är förkortning av Practice problems. Här menas att du bör inleda med att göra alla dessa. Du hittar dem direkt före övningarna till respektive avsnitt.)

Avsnitt	Instuderingsuppgifter	Träningsuppgifter	Teoretiska uppgifter
6.1	PP, 1, 6, 8, 11	15, 17, 24, 34	19, 26, 29
6.2	PP, 3, 5	9, 17, 21	23, 27, 29
6.3	PP, 1, 3, 5	9 11, 15, 25	21, 23
6.4	PP, 1, 3, 5	9, 11, 13, 15, 24	17, 19, 23
6.5	PP, 1, 3, 5	7, 9, 11, 15	13, 17
6.6	PP, 1, 4	7, 10, 11	

Laborationsuppgift 4: (ingen redovisning)
Rotationer av objekt i 3d

Beräkning av rotationsmatriser

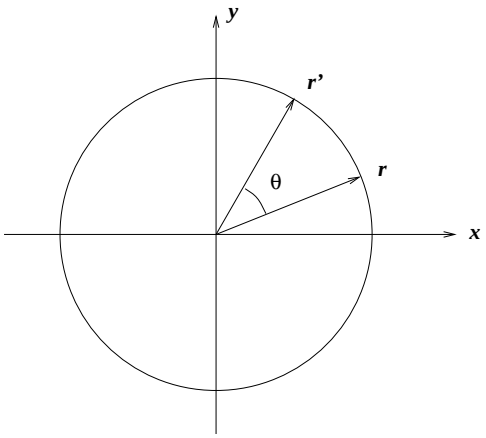
Matematisk bakgrund:

Låt $\boldsymbol{r} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$ vara ortsvektorn för en punkt i rummet. Sambandet $\boldsymbol{r}' = M\boldsymbol{r}$, där

$$M = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

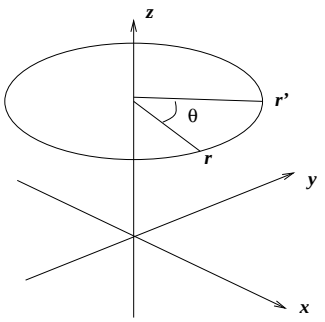
ger då en vridning av $\boldsymbol{r}$ en vinkel θ kring z-axeln (fig). Vi vill nu vrida en punkt kring en axel genom origo med riktningsvektor

$$\boldsymbol{v} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}.$$



Vi byter till en ny lämplig (högerorienterad) *ON-bas* $\mathcal{B} = \{\boldsymbol{f_1}, \boldsymbol{f_2}, \boldsymbol{f_3}\}$ som väljs så att $\boldsymbol{f_1} = \boldsymbol{u}/|\boldsymbol{u}|$, $\boldsymbol{f_2} = \boldsymbol{w}/|\boldsymbol{w}|$, och $\boldsymbol{f_3} = \boldsymbol{v}/|\boldsymbol{v}|$ där

$$\boldsymbol{u} = \begin{bmatrix} b \\ -a \\ 0 \end{bmatrix}, \ \boldsymbol{w} = \begin{bmatrix} ac \\ bc \\ -(a^2+b^2) \end{bmatrix}$$

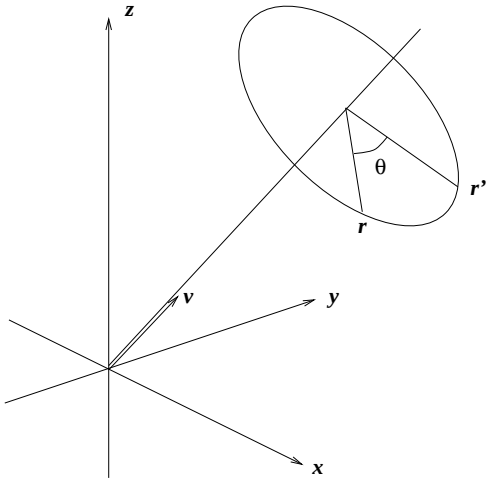


Med $P = [\boldsymbol{f_1} \ \boldsymbol{f_2} \ \boldsymbol{f_3}]$ (basbytesmatrisen) så ges sambandet mellan nya och gamla koordinater av $\boldsymbol{r} = P\boldsymbol{\rho}$, där $\boldsymbol{\rho} = [\boldsymbol{r}]_{\mathcal{B}}$ (koordinaterna för $\boldsymbol{r}$ i basen $\mathcal{B}$). I denna bas så ges matrisen för rotationen av matrisen M ovan. Vi får

$$\boldsymbol{\rho}' = M\boldsymbol{\rho}$$

och

$$\boldsymbol{r}' = P\boldsymbol{\rho}' = PM\boldsymbol{\rho} = PMP^{-1}\boldsymbol{r}.$$



Notera att eftersom kolonnerna i P utgör en ON-bas så är $P^{-1} = P^T$. Multiplikation med matrisen $R = PMP^T$ ger således en vridning med vinkeln θ kring vektorn $\boldsymbol{v}$. En samtidig vridning av flera ortsvektorer $\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_n$ åstadkommes genom matrismultiplikation

$$R[\boldsymbol{r}_1 \ \boldsymbol{r}_2 \ \ldots \ \boldsymbol{r}_n] = [\boldsymbol{r}'_1 \ \boldsymbol{r}'_2 \ \ldots \ \boldsymbol{r}'_n]$$

.

Uppgifter

1. **Rotationsmatrisen**
   Skriv en funktionsfil R=*rotation*(v,t) där
   v är den vektor som ger rotationsaxeln,
   t är den skalär som ger rotationsvinkeln och där
   R är den beräknade 3x3 rotationsmatrisen.
   Eventuell finputsning: Om v är parallell med z-axeln, blir vektorn u enligt proceduren ovan lika med nollvektorn . Tillfoga en villkorssats som gör att programmet klarar även detta fall!

2. **En roterande kub**
   Skriv en funktionsfil *kub*(v,t) som åstadkommer en roterande kub i grafikfönstret till matlab där
   v är den vektor som ger rotationsaxeln och där
   t är vinkeln som ger hur mycket kuben roterar mellan varje bild.
   Någon utvariabel behövs inte, filen ska ju bara åstadkomma plottning.

Ritning av godtyckliga figurer

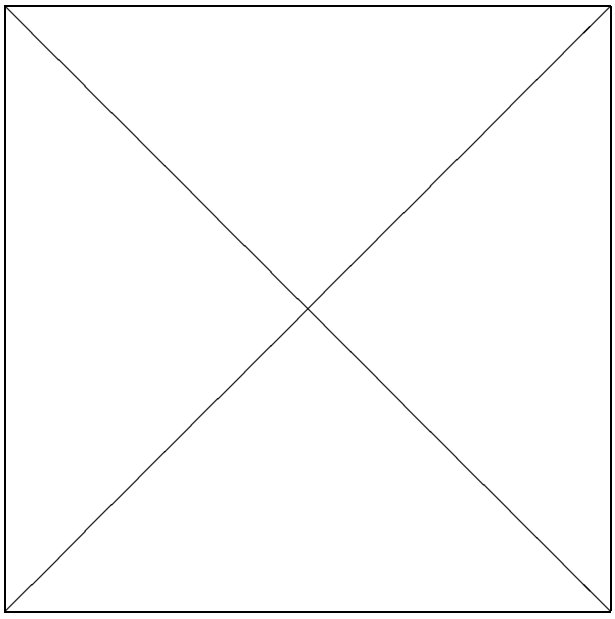
Vi vill kunna rita godtyckliga plana figurer med MATLAB, exempelvis en kurva $(x(t), y(t))$ på parameterform eller en polygon. Detta kan göras genom att skapa en koordinatmatris för de punkter vi vill rita. Denna matris kan skrivas på något av följande sätt

$$K = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix} = \begin{bmatrix} \boldsymbol{k}_1 & \boldsymbol{k}_2 & \dots & \boldsymbol{k}_n \end{bmatrix}$$

$\boldsymbol{x}$ och $\boldsymbol{y}$ är då radvektorer av längd n av x- resp y-koordinatern för punkterna, medan $\boldsymbol{k}_i$ är en kolonn vektor med kordinaterna för en punkt. Om vi vill rita en kurva på parameterform så väljer vi $\boldsymbol{k}_i = \begin{bmatrix} x(t_i) \\ y(t_i) \end{bmatrix}$ för tilläckligt täta t-värden. En polygon får vi om vi låter $\boldsymbol{k}_i$ vara hörnen uppräknade i en lämplig ordning.
Vi ritar nu figuren genom att plotta x-koordinaterna mot y-koordinaterna med kommandot
`>> plot(K(1,:),K(2,:))` .

Pröva genom att rita figuren nedan!

(0,1) (1,1)



(0,0) (1,0)

Plottning av 3d-objekt

Det handlar om att avbilda en projektion av objektet på ett plan. Enklaste sättet att göra detta i denna uppgift, är att välja xy-planet som projektionsplan. Man plottar då punkternas x-koordinater mot y-koordinaterna (z-koordinaterna lämnas därhän). Eftersom man här kan välja rotationsaxel fritt, kan man ändå åstadkomma alla perspektiv på en roterande kub om man nöjer sig med att projicera på xy-planet. Det finns i Matlab ett 3D-plot-kommando *plot3*, men uppgiften ska lösas med vanlig 2D-plot. (Testa gärna *plot3* också). Starta med en matris vars kolonner är koordinaterna för kubens olika hörn, skrivna i sådan ordning att man kan plotta alla önskade linjer. Det visar sig då att varje hörn måste tas med minst två gånger. Rita en kub, sätt namn på hörnen och planera matrisen. Animationen blir snyggast om man låter kubens centrum ligga i origo. Programmet kan t.ex. bygga på en loop där man i varje steg vrider kuben exempelvis 1/100 varv, ritar (plotta kubmatrisens första rad mot den andra, precis som ovan) och ger kommandot *drawnow* eller *pause(*Δt*)* (se Matlab help!). Ange ”axis square” efter plotkommandot. För vridningen behöver man generera en rotationsmatris (använd *rotation(v,t)*). Denna rotationsmatris skapas utanför loopen och återanvänds sedan i varje steg i loopen för att generera en ny kubmatris.

Den projektion som används här, innebär att linjer som är parallella också ser parallella ut i projektionen. I verkligheten ser man dock parallella linjer konvergera mot en punkt långt borta (t ex skenorna på ett järnvägsspår). Betraktar man kuben på lite håll, så påverkas parallelliteten ganska lite. Det ser alltså någorlunda realistiskt ut, om man tänker sig att kuben inte befinner sig alltför nära ögat. Däremot kan det vara svårt att se vad som är fram och bak på kuben, och en vanlig synvilla är att den plötsligt byter rotationsriktning. Detta kan avhjälpas genom att man märker en av kubens sidor, t ex genom att rita ut diagonalerna i den sidan. (Vill man åstadkomma nyss nämnda projektion med konvergerande linjer, kan detta åstadkommas med sk homogena koordinater. Man arbetar då i 4 dimensioner och med $4 \times 4-$matriser - se Lay, avsnitt 2.7!).