Laboration 1

1.1 IEEE 754

IEEE 754 är den standard för flyttalsberäkning som används i de flesta datorer. Som bekant kan en dator generellt inte representera något annat än heltal exakt. Hur operationerna $+$, $\times$, $/$ och $\sqrt{\cdot}$ skall gå till för flyttal (vanligen av precisionen `double`) är specificerat i IEEE 754. I den här laborationen skall vi belysa några av de vanligaste felen som kan uppstå när man är oförsiktig i sitt hanterande av flyttalsaritmetik.

1.1.1 Hanterandet av singulära uttryck i Matlab

Kör följande kodsnutt i Matlab:

```
a  = 1.0/0.0;
b  = 1.0/a;
sa = sin(a);
sb = sin(b);
la = log(a);
lb = log(b);
c  = 0.0/0.0;
d  = 1.0/c;
sd = sin(d);
```

Vid en första anblick är det enkelt att tro att samtliga av dessa uttryck skall returnera `NaN` (Not a Number), då rent matematiskt utgör de allihopa icke-definierade uttryck. Så är dock inte fallet.

- Vilka får man en väldefinierade output av?
- Vad tror ni anledningen till det är?

1.1.2 Minsta möjliga precision vid flyttalsräkning

Kör följande kodsnutt i Matlab:

```
format long e
logspace(-301, -324, 24)'
```

Som output skall ni få en vektor $[10^{-301}, 10^{-302}, \ldots, 10^{-324}]^T$.

- Är det verkligen vad ni får?
- Vad är orsaken till att outputen skiljer sig från vad det borde vara?

1.1.3 Fullständing utskiftning

Antag att vi har två flyttal, x och δ, så att $x \gg \delta$. Vi säger att *fullständig utskiftning* sker när $x + \delta \to x$, det vill säga att enligt IEEE 754-standarden ger addition av talen x och δ enbart x som output. Detta är såklart matematiskt omöjligt för alla $\delta \neq 0$, men det kan ske i en dator som en konsekvens av flyttalsaritmetikens begränsningar.

- Sätt $x = 1$ och hitta sedan ett ungefärligt värde på δ så att $x+\delta$ resulterar i fullständig utskiftning.
- Gör sedan samma sak för $x = 10^{20}$ samt $x = 10^{-20}$. Hittar ni ett mönster i när utskiftning sker?

Ledning: Följande kod-stubb kan vara till hjälp:

```
x     = 1;
delta = x;
diff  = abs((x+delta)-x);
while diff > 0
    % Uppdatera delta och berakna diff igen
end
disp(delta)
```

1.1.4 Konsekvenser av utskiftning

I den här uppgiften skall vi studera vad som kan hända om man inte tar utskiftning i beaktning. Beräkna först summan $S_1 = \sum_{n=1}^{10^6} \frac{1}{n}$, förslagsvis med hjälp av en `for`-loop. Beräkna sedan summan $S_2 = \sum_{n=10^6}^{1} \frac{1}{n}$, det vill säga samma summa fast genomlöpt baklänges.

För att tydligare belysa utskiftningen skall summorna beräknas med `single`-precision till skillnad från den vanliga `double`-precisionen. Initialisera därför summorna som `S1 = single(0)` och `S2 = single(0)`. De två sätten att beräkna summan på kommer resultera i olika resultat.

- Varför det?
- Vilken av S_1 och S_2 är närmst det verkliga värdet?

1.1.5 Konsekvenser av utskiftning, fortsättning

Kör följande kodsnutt i Matlab:

```
x  = linspace(0.995,1.005,1001);
y1 = (x-1).^6;
y2 = 1-6*x+15*x.^2-20*x.^3+15*x.^4-6*x.^5+x.^6;
plot(x,y1)
hold on
plot(x,y2)
```

Notera att uttrycken $(x-1)^6$ och $1-6x+15x^2-20x^3+15x^4-6x^5+x^6$ är identiska rent matematiskt.

- Vad är det då som ger de synliga skillnaderna i plotten?
- Vilket sätt är det bättre att uttrycka funktionen på?

1.2 Matrisfaktoriseringar

I beräkningsmatematik stöter man ofta på mycket stora matrisproblem. Vid hantering av stora matriser stöter man på två huvudsakliga problem; det ena är att datorn får flyttalsaritmetiska bekymmer likt de i den föregående uppgiften. Det andra är att antalet beräkningar växer sig så stora att det blir orimligt för datorn att kunna lösa ens problem inom en rimlig tid. Som tur är så finns det en uppsjö av tekniker man kan använda för att förenkla beräkningsarbetet. Vi skall illustrera ett par av dessa i den här uppgiften.

1.2.1 Lösning av linjära ekvationssystem i Matlab

Kör följande kodsnutt i Matlab:

```
n = 3000;
A = randn(n);
b = randn(n, 1);
tic; x = inv(A) * b; toc
tic; y = A \ b; toc
```

Koden genererar ett slumpmässigt, stort ekvationssystem som vi sedan löser dels med hjälp av `inv(A)*b`, och dels med Matlab-rutinen \.

- Vilken av dem går snabbast?
- Varför gör den det?

Det kan hjälpa att kolla dokumentationen[1] för \.

1.2.2 Glesa matrisproblem

Vid numerisk lösning av differentialekvationer är det mycket vanligt att man får ut stora matriser där en överväldigande majoritet av elementen är lika med noll. Dessa matriser kallas för *glesa matriser*. Som ni strax kommer att se är det alldeles speciellt ineffektivt att arbeta med inverser av glesa matriser. Man brukar därför när det är möjligt utnyttja någon känd struktur hos matrisen till sin fördel.

Kör följande kodsnutt i Matlab:

```
n = 6000;
e = ones(n, 1);
A = spdiags([e 2*e e], -1:1, n, n);
b = randn(n, 1);
tic; x = inv(A) * b; toc
tic; y = A \ b; toc
```

[1] Här får ni en länk: `https://se.mathworks.com/help/matlab/ref/mldivide.html`.

Kommandot `spy(A)` markerar nollskilda element i matrisen A med en blå prick. A är konstruerad så att elementen på diagonalen, samt de två diagonalerna direkt över och under, är de enda nollskilda. En sådan matris kallas för *tridiagonal* och är mycket vanligt förekommande.

- Jämför återigen tidsskillnaden i att använda `inv(A)` kontra \-rutinen för att lösa linjära ekvationssystem. Kommentarer?
- Vilken matrisfaktorisering kommer \ att använda sig av?
- Avslutningsvis kan ni köra `spy(inv(A))`. Är inversen av A gles bara för att A är gles?

Inlämning

Skicka era svar på de samtliga frågor markerade i punktlistorna ovan som en pdf via Canvas, se deadline i CANVAS. Glöm inte att bifoga väl kommenterad kod till varje uppgift. Koden skall bifogas som en enda m-fil vid namn **namn1_namn2_MMG410Lab1.m**, och varje uppgift skall uppta en egen cell i filen.