Tekniska Högskolan i Linköping
Institutionen för Datavetenskap (IDA)
Torbjörn Jonsson 2010-11-19

# Plot och rekursion

*I denna laboration skall du lära dig lite om hur plot i MatLab fungerar samt använda rekursion i vissa uppgifter.*

## Tidsåtgång

Denna laboration beräknas ta ca 6 timmar i anspråk (2 timmar bokade med assistent).

## Mål

Du ska efter denna laboration känna till

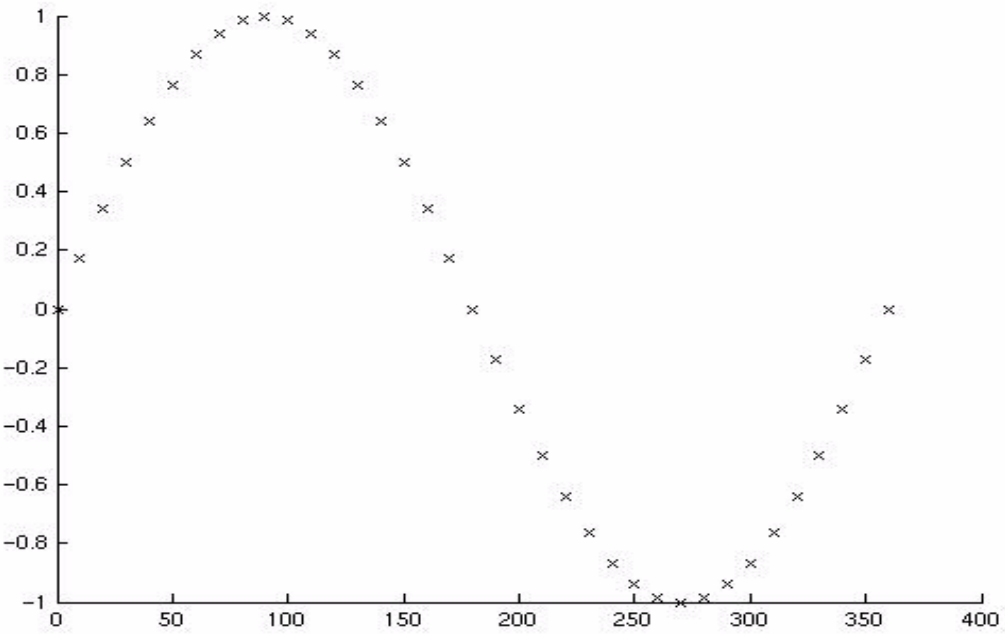
- hur plot fungerar i dess enklaste form
- när man behöver använda *hold*-kommandot
- vad *figure* är för något
- hur rekursion ser ut i MatLab
- hur man lägger flera funktioner i samma fil
- vilken räckvidd olika delar får i ett program (lokala/globala variabler/funktioner m.m.)

## Uppgift 1 (ca 30 min)

Skriv en funktion som ritar ut hur en matematisk funktion ser ut. Indata till denna funktion skall vara den matematiska funktionen, start- och slutvärde samt steglängd på x (d.v.s. 4 parametrar).

Ett exempel på hur det ser ut när man kör din utritningsfunktion (vi antar att du kallat den plot_function) och ritar ut funktionen sind i intervallet [0, 360] med steglängd 10:

```
>> plot_function(@sind, 0, 360, 10)
```



## Uppgift 2 (ca 45 min)

Skriv ett program som låter användaren mata in ett värde på ’x’ (där ’x’ skall ligga i intervallet [1, 7]) och sen beräknar resultatet enligt följande formler:

$$f(0) = 1$$
$$f(1) = 1$$

Samt för x > 1:

$$f(x) = \sum_{i=1}^{x} \left( i + \frac{\sum_{j=1}^{i} \left( \frac{f(j-1) + f(x-j))}{j} \right)}{i} \right)$$

Körexempel 1:

```
Ange värdet på x: 2
Resultatet blir : 6.50
```

Körexempel 2:

```
Ange värdet på x: 3
Resultatet blir : 21.4167
```

Körexempel 3:

```
Ange värdet på x: 5
Resultatet blir : 185.4417
```

Körexempel 4:

```
Ange värdet på x: 7
Resultatet blir : 1660.8183
```

Krav: Funktionen f(x) skall vara rekursiv. Beräkningen av summorna bör dock inte vara rekursiva.

Tips: Tänk *inte för mycket* på vad som händer.

Följande körexempel skulle vi kunna åstadkomma, men vi rekommederar er inte att köra dessa. De tar längre tid än nödvändigt.

```
Ange värdet på x: 9
Resultatet blir : 16734.2009

Ange värdet på x: 10
Resultatet blir : 55497.835
```

## Uppgift 3 (ca 2 tim)

För att skriva en egen funktion **my_sin(x)** som approximerar den inbyggda sin(x) använder man sig t.ex. av Taylorutvecklingen för sin(x) som ser ut på följande sätt:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

Kuriosa: Felet i summan är alltid mindre än absolutbeloppet av den första utelämnade termen i serien. Skulle man t.ex. vilja ha ett resultat som har ett mindre fel än $10^{-6}$ skall man alltså avsluta när man hittar den första termen som har ett absolutbelopp som är mindre än just detta värde.

Använd Taylorutvecklingen ovan och beräkna sin(x) rekursivt på två olika sätt. Variant 1 där man beräknar summan framifrån (de stora termerna först) och variant 2 där man beräknar summan bakifrån (de små termerna först). Det är givet att ni bara behöver summera de första 6 termerna i serien beskriven ovan.

I huvudprogrammet skall ni anropa de två varianterna (kalla dem för **my_sin_1(x)** respektive **my_sin_2(x)** för att kunna skilja dem åt) och beräkna differensen. Utskriften från programmet skall vara de båda ”sinusvärdena”, den inbyggda sinusfunktionens sinusvärde samt differensen mellan dessa tre enligt exemplet nedan. Det är lämpligt att användaren får mata in vilket värde på x som skall testas.

Krav: Du skall dessutom visa att du förstår hur man har flera funktioner i samma fil och dessutom att du anropar det som går i denna fil från en funktion i en separat fil.

Tips: Det finns behov av en underfunktion till era sinusfunktioner. Underfunktionen är rekursiv. Själva **my_sin_N(x)** är inte rekursiv utan anropar underfunktionen med lämpliga data. Du kan även testa dina funktioner med *plot_function* som du gjorde tidigare om du lägger dem i separata filer.

Körexempel 1:

```
>> test_my_sin(0.1)
A=sin(x)      = 0.099833
B=my_sin_1(x) = 0.099833
C=my_sin_2(x) = 0.099833
B-A           = 0
C-A           = 1.3878e-17
B-C           = -1.3878e-17
```

Körexempel 2:

```
>> test_my_sin(0.25)
A=sin(x)      = 0.2474
B=my_sin_1(x) = 0.2474
C=my_sin_2(x) = 0.2474
B-A           = 0
C-A           = 0
B-C           = 0
```

Körexempel 3:

```
>> test_my_sin(1.4)
A=sin(x)      = 0.98545
B=my_sin_1(x) = 0.98545
C=my_sin_2(x) = 0.98545
B-A           = -1.2628e-08
C-A           = -1.2628e-08
B-C           = -1.1102e-16
```

Körexempel 4 (detta skall man reagera på!):

```
>> test_my_sin(6.28)
A=sin(x)      = -0.0031853
B=my_sin_1(x) = -3.1778
C=my_sin_2(x) = -3.1778
B-A           = -3.1746
C-A           = -3.1746
B-C           = 8.8818e-16
```

## Uppgift 4 (ca 2 tim)

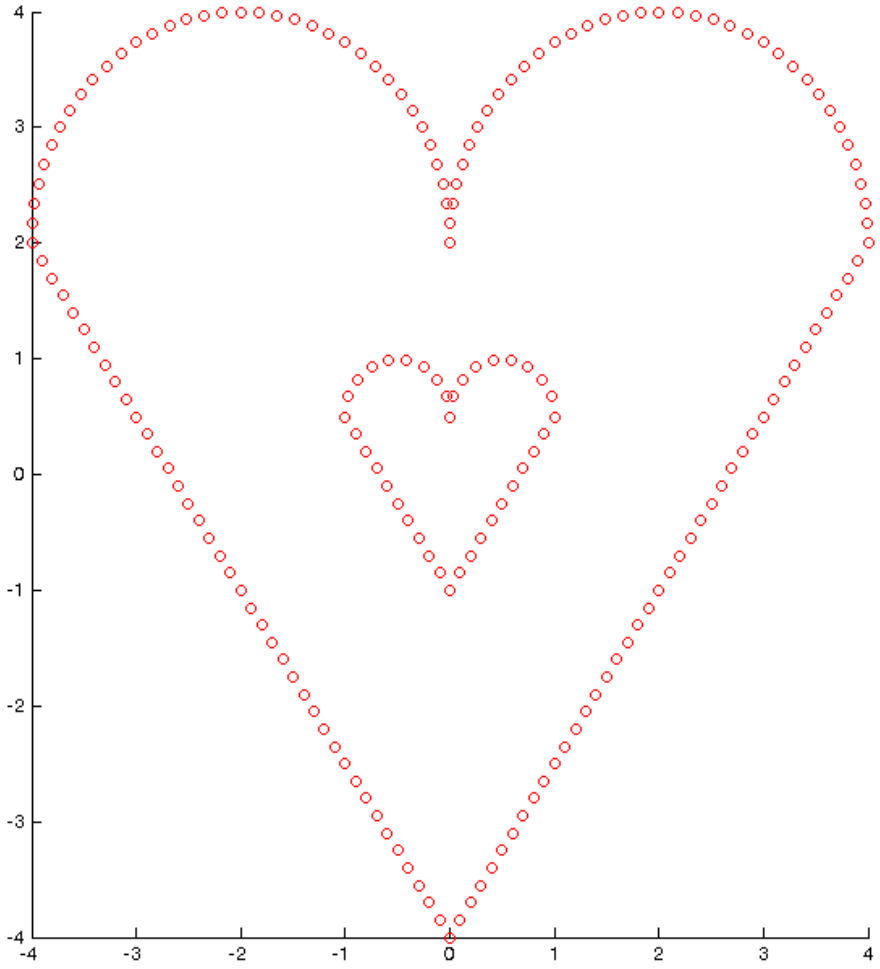
Skriv funktionen *plot_heart* som ritar ut ett hjärta enligt figurerna nedan. Till funktionen skall du skicka in storleken på hjärtat (en multipel av 0.1). För att rita ut hjärtat krävs fyra delar. Två halvcirklar (som man ritar ut position för position) samt två raka streck (som också ritas punkt för punkt). De olika positionerna som ritas ut skall vara små röda cirklar.

De två figurerna nedan har storlek 0.5 respektive 2.0 och det man direkt ser är att det är fler småcirklar i den andra figuren samt att graderingen på axlarna är olika (detta justeras i MatLab, som ni märkt, automatiskt). Antalet cirklar skall bero av storleken (detta för att man skall kunna rita två olika storlekar i samma figur, men ändå inte får tätare mellan cirklarna i det mindre hjärtat).

Tips1 : Se i figuren och mät för att få fram hur avstånd m.m. förhåller sig till storleken som matats in.

Tips 2: Det kan vara bra att tänka på i vilken riktning man utför olika saker. Det kan i vissa fall ge olika resultat om man börjar från ”fel” håll.

Krav: Om man anropar *plot_heart* flera gånger skall tidigare hjärtan ligga kvar. Man ser då tydligt avståndsskillnaderna i olka storlekar om man gjort galet.



## Uppgift 5 (ca 45 min)

Skriv funktionen *fill_heart* som ritar ut ett fyllt hjärta enligt figuren nedan. Till din hjälp antar jag att du kan använda *plot_heart* som du redan skapat. Även denna figur skall ritas ut givet en storlek som parameter (på samma sätt som föregående uppgift). I denna figur ser man varför det är rimligt att ha samma avstånd mellan de olika cirklarna som används för att rita figuren.

Krav 1: Ditt program skall rita ut figuren med fördröjning så att man ser hur hjärtat ritas ut. Åtminstone en ”ram” i taget. Behöver inte vara per cirkel.

Krav 2: Funktionen *fill_heart* skall vara rekursiv. Avbryt när storleken är 0.1.

Tips: Följande inbyggda kommandon kan vara bra att kolla in: *clf*, *figure*

Denna figur är ritad för storlek 2.