I dag

Vektorisering och effektivitet

- Tidtagning
- Fördimensionering av matriser
- Vektorisering
- Vektoriserade operationer under villkor
- Lokalisering av element i en vektor

Vektorisering och effektivitet: Intro

- MATLAB är ett *interpreterande* programmeringsspråk.
- Vid körning av program som man själv har skrivit, läser och tolkar MATLAB programkoden och utför instruktionerna samtidigt som programmet körs.
- Detta går i allmänhet långsammare än att köra ett kompilerat program, där instruktionerna i förväg omvandlats till maskinkod.
- För att göra M-filer så snabba och effektiva som möjligt bör man därför i så hög grad som möjligt utnyttja MATLAB:s inbyggda kommandon och funktioner (som är kompilerade).
- Andra sätt att snabba upp sina M-filer inkluderar *fördimensionering av matriser* och *vektorisering*.

Tidtagning

- Tiden det tar att köra ett antal kommandon kan mätas med kommandona `tic` och `toc`.
- Kommandot `tic` startar en tidtagare och kommandot `toc` stänger av densamma.
- Vi jämför nu hur lång tid det tar att addera samtliga element i en 1000×1000-matris, dels genom att skriva en nästlad loop, dels genom att utnyttja MATLAB:s inbyggda funktion `sum`:

```
% test1.m
A = 2*rand(1000,1000) - 1;  % Skapa en matris med en miljon
                            % likformigt fördelade slumptal
                            % mellan -1 och 1.
% Ta tid för summaberäkning med nästlad loop:
tic                         % Starta tidtagning.
s = 0;                      % Initiera variabel för summa.
for i = 1:1000              % Loopa över rader.
    for j = 1:1000          % Loopa över kolonner.
        s = s + A(i,j);     % Lägg till ett element till summan.
    end
end
toc                         % Läs av tidtagare.
disp(s)                     % Skriv ut summan.
% Ta tid för summaberäkning med inbyggda funktionen sum:
tic
s2 = sum(sum(A));           % Notera att sum(A) ger en vektor
                            % med kolonnsummor. Dessa summeras
                            % sedan i sin tur.
toc
disp(s2)
```

- Vi kör programmet `test1.m`:

```
>> test1
Elapsed time is 0.040826 seconds.
  260.0673

Elapsed time is 0.005982 seconds.
  260.0673

>> 0.040826/0.005982    % Jämför tiderna.

ans =

    6.8248
```

- Notera att det med den inbyggda funktionen går cirka 7 gånger snabbare.

Fördimensionering av matriser

- I motsats till många andra programspråk behöver man i MATLAB inte ange dimensionen av vektorer och matriser i förväg:

```
>> x(1) = 3
x =
     3
>> x(2) = 6
x =
     3     6
>> x(3) = -4
x =
     3     6    -4
```

- MATLAB reserverar minnesutrymme för de nya elementen allteftersom det behövs, men denna minneshantering är resurskrävande.
- Man kan spara tid genom att i förväg reservera minnet som behövs. Detta kallas att *fördimensionera matrisen*.

- Vi skapar som exempel en 1000×1000-matris A där elementet på plats (i, j) ges av $a_{ij} = (i - j)/(i + j)$:

```
% test2.m
clear all                       % Rensa alla variabler.
tic                             % Starta klocka.
for i = 1:1000                  % Loopa över rader.
    for j = 1:1000              % Loopa över kolonner.
        A(i,j) = (i-j)/(i+j);   % Beräkna och tilldela element på plats (i,j).
    end
end
toc                             % Stäng av klockan.

clear all                       % Exakt samma men med fördimensionering:
tic
A = zeros(1000,1000);           % Fördimensionera matrisen.
for i = 1:1000
    for j = 1:1000
        A(i,j) = (i-j)/(i+j);
    end
end
toc
```

- Vi kör programmet `test2.m`:

```
>> test2
Elapsed time is 15.349701 seconds.

Elapsed time is 0.048077 seconds.

>> 15.349701/0.048077

ans =

  319.2733
```

- Notera att det med fördimensionering går mer än 300 gånger snabbare!

Vektorisering

- MATLAB är ett "matris-språk", vilket innebär att det är utformat med tanke på vektor- och matrisoperationer. MATLAB är mycket effektivt på att utföra den typen av operationer.
- Man kan därför ofta snabba upp sina M-filer genom att *vektorisera* sina beräkningar.
- Att vektorisera innebär att omvandla `for` och `while`-loopar så att man istället utför vektor- och matrisoperationer.
- Det betyder att operationer på ett element i taget ersätts med operationer på hela vektorn/matrisen.
- Förutom att det går snabbare så blir ofta programmen mer överskådliga och lättlästa.
- Vi illustrerar med några exempel:

Vektorisering (forts.)

- För att beräkna $y = x^2$ för 101 stycken x-värden mellan 0 och 1 skulle vi kunna göra så här:

```
>> i = 0;
>> for x = 0:0.01:1
       i = i + 1;
       y(i) = x^2;     % Beräkna funktionsvärden ett åt gången.
   end
```

  men det här är ett bättre sätt:

```
>> x = 0:0.01:1;
>> y = x.^2;          % Beräkna samtliga funktionsvärden med
                      % en vektoroperation.
```

- För att addera vartannat element i en vektor x kan man göra så här:

```
>> x = [4 2 -4 5 -2 -7 3];
>> summa = 0;
>> for i = 1:2:length(x)
       summa = summa + x(i);
   end
>> summa
summa =
  1
```

  men det går också att göra så här:

```
>> x = [4 2 -4 5 -2 -7 3];
>> y = x(1:2:length(x))
y =
  4    -4    -2     3
>> summa = sum(y)
summa =
  1
```

- För att beräkna hur många element i en vektor x som är positiva kan man göra så här:

```
>> x = [4 2 -4 5 -2 -7 3]; antal = 0;
>> for i = 1:length(x)
      if x(i) > 0
         antal = antal + 1;
      end
   end
>> antal
antal =
     4
```

  men det går också att göra så här:

```
>> x = [4 2 -4 5 -2 -7 3];
>> y = x>0  % Det logiska uttrycket beräknas elementvis. 1 = sant, 0 = falskt.
y =
  1     1     0     1     0     0     1
>> antal = sum(y)
antal =
     4
```

- För att addera alla positiva element i en vektor x kan man göra så här:

```
>> x = [4 2 -4 5 -2 -7 3]; summa = 0;
>> for i = 1:length(x)
      if x(i) > 0
         summa = summa + x(i);
      end
   end
>> summa
summa =
 14
```

  men det går också att göra så här:

```
>> x = [4 2 -4 5 -2 -7 3]; y = x>0
y =
  1     1     0     1     0     0     1
>> z = x.*y
z =
  4     2     0     5     0     0     3
>> summa = sum(z)
summa =
 14
```

Vektoriserade operationer under villkor

- Ytterligare ett vektoriserat alternativ för att addera alla positiva element i en vektor x:

```
>> x = [4 2 -4 5 -2 -7 3];
>> x>0
ans =
  1     1     0     1     0     0     1
>> x(x>0)   % Ger delvektor med positiva element.
ans =
  4     2     5     3
>> summa = sum(x(x>0))
summa =
 14
```

- Allmän form:

```
kommando(vektor(logiskt villkor))
```

Lokalisering av element i en vektor

- Med kommandot `find` kan vi lokalisera de element i en vektor som uppfyller ett visst villkor.
- *Exempel.* Beräkna antalet negativa element i en vektor. Ersätt dessa med 0.

```
>> v = [4 2 -5 4 -2 -8 0 3 -5]
v =
     4     2    -5     4    -2    -8     0     3    -5
>> find(v<0)   % Ger en vektor med positioner för de negativa elementen.
ans =
     3     5     6     9
>> length(find(v<0))   % Ger antalet negativa element.
ans =
     4
>> v(find(v<0)) = 0    % Ersätter de negativa elementen med 0.
v =
     4     2     0     4     0     0     0     3     0
```