Linköping institute of technology
Department of Computer and Information Science (IDA)
Torbjörn Jonsson, Klas Arvidsson, Erik Nilsson 2013-04-26

Lab MATLAB: Introduktion

Följande laboration introducerar MATLAB och hur man skriver skript och funktioner med några grundläggande kommandon samt hur man manipulerar vektorer och matriser.

Introduktion

Se hemsidan för hur mycket labtid som är tillägnad denna laboration. Du kommer troligtvis behöva jobba en del på egen hand, för att bli klar med uppgifterna i tid. Vissa uppgifter kan kräva mer tid än andra.

Mål

I denna laboration kommer du att lära dig :

- Hur man kör igång MATLAB på IDA:s datorer.
- Hur man skriver skript i MATLAB:s editor.
- Några grundläggande kommandon, t.ex. `input`, `disp`, `fprintf`, `mod`, `ceil`, `floor`, `randi`, `sum`, `prod`, `length`, `size`, `help` m.fl.
- Hur man skriver funktioner i MATLAB och vad som skiljer dessa från skript.
- Hur man skapar, indexerar i och manipulerar matriser med olika innehåll.
- Ett sätt att visualisera data i en matris.

För att komma igång:

1. Läs de generella tipsen på kurshemsidan. Följ instruktionerna på sidan ”Hur gör man?” på kurshemsidan. Informationen som står där är viktig för att klara av laborationerna. Använd gemener (små bokstäver) i dina filnamn. Filnamnen på dina lösningar till uppgifterna skall sluta med “.m”.

2. Använd kommandot `help` för att få information om några grundläggande matematiska kommandon, t.ex. `mod`, `sin`, `cosd`, `sqrt`, `abs`, `ceil`, och `floor`. Vad gör dessa kommandon? Hur använder man dem? Använd dem i MATLABs kommandoradsfönster (”command window”).

3. Testa på att skapa lite vektorer och matriser i kommandofönstret. För detta kan du använda kommandon så som `zeros`, `ones` och `eye`. Hur skapar man en tom vektor/matris?

Uppgift 1: `rand_int.m`

Skapa en funktion `rand_int` som genererar ett slumpat heltal i intervallet [A, B]. A och B är heltal som funktionen skall ta som parametrar. Du får använda den inbyggda funktionen `rand` (men inte `randi`) för att generera slumpade flyttal i intervallet [0.0, 1.0[. Distributionen av talen som genereras skall vara jämn över intervallet. D.v.s. sannolikheten att få A skall vara den samma som att få B och varje heltal däremellan. Du kan använda nästa uppgift för att testa din funktion.

Körexempel (funktionen anrop från MATLAB:s kommandoradsfönster):

```
>> rand_int(6, 10)
ans =      7
```

Uppgift 2: `roll_die_n_times.m`, `roll_dice_test.m`

Skapa funktionen `roll_die_n_times`, som tar två heltalsparametrar N och A. Funktionen skall slå en vanlig 6-sidig tärning N gånger och räkna hur många av dessa slag som gav resultatet A. Resultatet skall returneras från funktionen. Anropa din funktion `rand_int` med lämpliga parametrar för att simulera tärningsslagen.

Skapa ett sedan ett skript, `roll_dice_test`, som låter användaren mata in ett N (använd `input`) och som anropar `roll_die_n_times` 8 gånger, där A får vara värdena i intervallet [0, 7]. Skriv ut resultatet som en tabell där man kan se varje värde på A och hur många gånger som det blev A på N tärningsslag. Fundera på hur utskriften bör bli då N är stort.

Uppgift 3: `count_ones.m`

Skriv en funktion `count_ones`, som tar en parameter (ett heltal) och returnerar hur många ettor som finns i detta heltal. T.ex. har talet 11 (elva) 2 ettor, talet 12311 har 3 ettor, och talet 234 har 0 ettor.

Körexempel:

```
>> count_ones(8198191)
ans =
     3
```

Fråga: Varför är resultatet felaktigt för väldigt stora tal? Testa 1111111111111111 (som bör ge resultatet 17).

Uppgift 4: `my_add.m`, `my_mtimes.m`

Skriv funktionerna `my_add` och `my_mtimes` som båda tar två matriser som parametrar och adderar respektive multiplicerar dessa två matriser. Kom ihåg att dimensionerna på matriserna måste matcha, eller så kan en av dem vara en skalär (1x1), för att operationen skall kunna genomföras. Om operationen inte går att genomföra skall funktionerna skriva ut ett felmeddelande och avbryta. Det är inte tillåtet att använda de inbyggda operatorerna (“+” och “*” för matriser) någonstans i dina funktioner. Du får självklart använda “+” och “*” för skalärer. Att addera/multiplicera en matris med en skalär medför att man adderar/multiplicerar skalären med varje element i matrisen. Dina funktioner får INTE ha några andra parametrar än de två matriserna.

Obs! Du kan inte anta att det är just en av parametrarna är en skalär. Oavsett om den första parametern är en skalär eller om det är den andra som är det så skall resultat bli detsamma.

Tips: Att addera [1, 2, 3] med 5 är samma sak som att addera 5 med [1, 2, 3]. Kan du t.ex. använda dig av kommandona `reshape` och `numel`?

Exempel:

```
my_add([1, 2, 3], [1, 2, 3])       => [2, 4, 6]

my_add([1, 2, 3], 2)               => [3, 4, 5]

my_add([1, 2, 3; 4, 5, 6], [5, 6, 7; 2, 3, 4]) =>
                                       [6, 8, 10 ;
                                       [6, 8, 10]


my_mtimes([1, 2, 3], [1 ; 2 ; 3]) => 14

my_mtimes(2, [1, 2, 3])            => [2, 4, 6]

my_mtimes([1 ; 2 ; 3], [1, 2, 3]) => [1, 2, 3 ;
                                       2, 4, 6 ;
                                       3, 6, 9]

my_mtimes([1, 2, 3; 3, 5, 6], [5, 2; 6, 3; 7, 4]) =>
                                      [38, 20 ;
                                       92, 47]
```

Uppgift 5: `my_life.m`

Skriv en funktion som genomför simuleringen *Game of Life* (skapad av Paul Callahan). Det är en enkel simulering över hur en population kan variera över tid. Varje ruta (cell) i ett rutnät kan innehålla en individ som föds, överlever eller dör, beroende på dess omgivning.

Låt din funktion generera en slumpad startpopulation. Låt sedan funktionen utföra simuleringen tills användaren avbryter den (med Ctrl-C). Följande regler gäller för en cell (och för dess 3x3-omgivning given med 8 grannar):

- En cell överlever om den har 2 eller 3 grannar.
- En cell föds om den har exakt 3 grannar.
- En cell dör, eller förblir död i alla andra fall.

Funktionen skall visa upp populationen för varje tidssteg. Kom ihåg att varje cell måste uppdateras innan en ny bild kan visas upp. För att hantera de celler som är på ”världens” kant så kan du anta att de ”grannar” som är utanför är döda. Eller så kan du låta cellerna på andra sidan vara dessa grannar (d.v.s världen ligger inte på en plan yta, utan har ”wrap-around”). Du kan titta på det inbyggda kommandot `life`, för att se hur cellerna kommer att bete sig.

Tips: När du uppdaterar världen kan du antingen arbeta på en ”kopia” eller använda samma värld. Om du använder samma värld måste du dock vara försiktig så du inte uppdaterar en cell utefter celler som redan blivit uppdaterade en gång detta tidssteg.

Krav 1: Funktionen skall ha två parametrar som anger hur stor världen skall vara (höjd och bredd).

Krav 2: När världen ritas ut skall det vara tydligt och snyggt. Varje cell skall inte bara representeras av en punkt. En ruta per cell kan vara lämpligt. Du kan t.ex. använda kommandot `imagesc`. (Men använd inte kommandot `image`). Du kan använda kommandot `pause(1)` för att få programmet att vänta i en sekund (lämpligt att göra detta mellan varje utritning).