Några tentamenstips

- Anmäl dig till tentan.
- Tentamenstiden är fyra timmar från och med 2011.
- Bekanta dig med lathunden i god tid (dvs. inte under tentan). Vad betyder det, vad står med, vad saknas?
- Lathunden är enda hjälpmedlet (räknare är ej tillåten). Ordböcker och lexikon är tillåtna.
- Läs igenom alla uppgifterna, vad jag tror är svårt tycker du kanske är enkelt.
- Fråga om något är oklart. När tentan är inlämnad är det för sent att korrigera en lösning.
- Följ specifikationerna. Ändra inte på funktionsnamn, antal in/ut-parametrar, typer på in/ut-parametrar. Funktioner skall vara "tysta" om ej annat anges i problemet.
- Arbeta inte i onödan. Du får inte extra poäng om du skriver en funktion som kan mer än jag begärt.
- Förklara hur du har tänkt, jag är ingen tankeläsare. Kommentera koden (svenska eller engelska), förklara idéer. Det innebär dock inte att du behöver skriva långa uppsatser.
- Om du inte kan skriva en mindre del av koden förklara hur du har tänkt, vad variablerna skall innehålla efter det kodsegment som saknas. Du kan då kanske få lite poäng för uppgiften.
- Om en uppgift består av flera delar, där del b) använder kod från del a), kan du (normalt) göra b-uppgiften. Skriv då att du antar att a-uppgiften är löst.
- Skriv läsligt. Skilj på olika typer av parenteser t.ex. Om jag inte kan läsa koden blir det poängavdrag.
- Det kommer inte C++, rekursion, shell-script, awk, tr.
- Uppgift 2 och 8 från äldre tentor utgår.

Här följer början på en typisk tentamenstes:

Tentamen: Programmering med Matlab, MVGF30, MVG300, GU, 2010-03-13, Hörsalslängan

Skrivtid: 08.30-12.30.

Vakt: Magnus Goffeng.

Betygsgränser MVGF30:
10 poäng av maximalt 24 räcker för godkänt, 16 poäng för VG.

Betygsgränser MVG300:
12 poäng av maximalt 24 räcker för godkänt, 18 poäng för VG.

Uppgiftspoäng: alla uppgifter ger fyra poäng.

Hjälpmedel:
"Lathund för Matlab" av Thomas Ericsson (fyra A4-sidor).
Över- och understrykningar är tillåtna i lathunden, men egna anteckningar är inte tillåtna. Ordlistor och lexikon.

Iakttag följande:
- Programmen skall skrivas i MATLAB.
Du får inte utnyttja MATLABs symboliska del.
- Alltför ineffektiva lösningar kan ge poängavdrag.
Använd vettiga variabelnamn.
- Fullständiga lösningar och motiveringar krävs!
Kommentera och indentera programmen.
- Skriv tydligt och disponera papperet på ett lämpligt sätt.
Börja varje ny uppgift på nytt blad.
- Sortera Dina lösningar i nummerordning. Läs igenom alla uppgifterna. De är inte sorterade efter svårighetsgrad.

Här följer problemformuleringar och lösningsförslag till tentan 2010-03-13.

Skriv ett MATLAB-program som beräknar och skriver ut alla heltal, n, med $1 \leq n \leq 10^4$, för vilka gäller att:

$$0.01 \cdot n \leq \sum_{k=1}^{n} \frac{\sin(0.1 \cdot k)}{k^2} \leq 1.1 \cdot n$$

Här följer två lösningsförslag:

```
s = 0;
for k = 1:10000
  s = s + sin(0.1 * k) / k^2;
  if 1e-2 * k <= s && s <= 1.1 * k
    disp(k)
  end
end

% A vectorized alternative
n = (1:1e4)';
s = cumsum(sin(0.1 * n) ./ n.^2);
find(0.01 * n <= s & s <= 1.1 * n)
```


Skriv en funktion `r = unique(v)` som tar en rad- eller kolonnvektor `v` av tal som inparameter. Du kan anta att `v` innehåller minst ett tal. Funktionen skall returnera en vektor `r` som innehåller talen i `v` men utan upprepningar enligt följande exempel:

```
v = [7 7 -5 5 5 5 5 7 1 2 1 1 7 7 7] skall ge
r = [7 -5 5 7 1 2 1 7]
```

Så, om man har en sekvens av samma tal, tas kopiorna bort, men samma tal kan få förekomma flera gånger i `r` förutsatt att talet omges av andra tal. Om `v` är en radvektor så skall också `r` vara en radvektor och om `v` är en kolonnvektor så skall också `r` vara en kolonnvektor.

```
function r = unique(v)
% Problem 2.

r = v(1);

for k = 2:length(v)
  if v(k) ~= v(k-1)
    r = [r; v(k)];   % becomes a column vector
  end
end

if size(v, 2) > size(v, 1) % row vector
  r = r';
end
```

MATLAB har de två logiska funktionerna **all** och **any**. Antag att så inte vore fallet så att du var tvungen att skriva egna varianter. Så, skriv en logisk funktion, **all**, som tar en logisk vektor som inparameter och som returnerar **true** om alla elementen i vektorn är sanna och som returnerar **false** annars. Skriv också en logisk funktion, **any**, som tar en logisk vektor som inparameter och som returnerar **true** om något element i vektorn är sant och som returnerar **false** annars. Dina rutiner får givetvis inte använda MATLAB:s **all** eller **any**. Skriv också ett huvudprogram som testar **all** och **any** på vektorerna **1:5 > 2.5** samt **1:5 > 0**.

```
function all_any
% Problem 3.
% Main program for testing all and any

some_true = 1:5 > 2.5;
all_true  = 1:5 > 0;

disp('all')
all(some_true)
all(all_true)

disp('any')
any(some_true)
any(all_true)

disp('all')
all_alternative(some_true)
all_alternative(all_true)

disp('any')
any_alternative(some_true)
any_alternative(all_true)

% Here are two solutions for each problem
% (one solution is enough on the exam).
```



```
function result = all(vec)
% a vectorized solution
result = sum(vec) == length(vec);

function result = all_alternative(vec)
% a loop solution
result = true;
for k = 1:length(vec)
  if ~vec(k)
    result = false;
    return
  end
end

function result = any(vec)
% a vectorized solution
result = sum(vec) > 0;

function result = any_alternative(vec)
% a loop solution
result = false;
for k = 1:length(vec)
  if vec(k)
    result = true;
    return
  end
end
```


Vi vill skriva en funktion

```
plot_data(X, Y, syms, xtxt, ytxt, header)
```

X, **Y** är två matriser som skall ha samma antal kolonner, n säg.

Funktionen skall öppna ett nytt plotfönster och i detta rita n kurvor som definieras av parvisa kolonner i **X** och **Y** (så första kolonnen i **X** och första kolonnen i **Y** definierar den första kurvan etc).

Kurvan skall ritas med den färg och linjetyp som definieras av cellvektorn **syms** som skall ha minst n element, där varje element är en sträng.

Ett element skulle kunna vara **'r-'** för en röd heldragen kurva, ett annat element skulle kunna vara **'k:'** för en svart prickad kurva. Första kurvan skall använda första elementet ur **syms**, andra kurvan andra elementet etc.

Plotten skall förses med text utmed x-axeln, som definieras av strängvariabeln **xtxt**, analogt för y-axeln. **header** är en strängvariabel som innehåller den sträng som skall sättas som rubrik. Om **X** och **Y** har olika antal kolonner eller om antalet element i **syms** är mindre än antalet kolonner i **X** och **Y**, så skall funktionen skriva ut ett felmeddelande (och inte öppna något plotfönster eller plotta något).


```
function plot_data(X, Y, syms, xtxt, ytxt, header)
% Problem 4.

% Some basic error checking
n_cols = size(X, 2);
if n_cols ~= size(Y, 2)
  warning('Different number of columns in X and Y.')
  return
elseif length(syms) < n_cols
  warning('Wrong number of elements in syms.')
  return
else     % Seems OK, start to plot
  figure
  for k = 1:n_cols
    plot(X(:, k), Y(:, k), syms{k})
    hold on
  end
  xlabel(xtxt)
  ylabel(ytxt)
  title(header)
end
```

Vi har en uppsättning datafiler vars namn alla slutar på **`.dat`**. Varje fil innehåller ett (varierande) antal rader med ett reellt tal per rad. Skriv en rutin, **`data = comp_data`**, som går igenom alla datafilerna (i aktuell katalog) och som, för varje fil, beräknar det minsta värdet och det största värdet av talen i filen.

Minsta- och största värdet skall, tillsammans med filnamnet, returneras i utvariablen **`data`** som är en vektor av poster konstruerad på följande sätt.

Varje post skall ha tre datamedlemmar, **`filename`** (för filnamnet), **`min`** (för minsta värdet) och **`max`** (för största värdet). Om ingen datafil ligger i katalogen så skall **`data`** bestå av en post där **`filename`** är den tomma strängen och där **`min`** och **`max`** är tomma vektorer.

```
function data = comp_data

files = dir('*.dat');     % all files ending in .dat
n_files = length(files); % number of files

if n_files == 0          % no files
  data.filename = '';
  data.min      = [];
  data.max      = [];
else
  for k = 1:n_files             % for each data file
    fid = fopen(files(k).name);
    [x, n] = fscanf(fid, '%e'); % read n numbers to
    fclose(fid);                % vector x

    data(k).filename = files(k).name; % save data
    data(k).min = min(x);
    data(k).max = max(x);
  end
end
```


Sudoku är ett pussel där det gäller att fylla en 9×9-matris, $\boldsymbol{S}$, med tal. $\boldsymbol{A}, \ldots, \boldsymbol{I}$ är 3×3-matriser nedan.

$$\boldsymbol{S} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} & \boldsymbol{C} \\ \boldsymbol{D} & \boldsymbol{E} & \boldsymbol{F} \\ \boldsymbol{G} & \boldsymbol{H} & \boldsymbol{I} \end{bmatrix}$$

$$\boldsymbol{L} = \begin{bmatrix} 5 & 4 & 8 & 1 & 6 & 7 & 2 & 9 & 3 \\ 1 & 2 & 9 & 3 & 5 & 4 & 6 & 7 & 8 \\ 3 & 7 & 6 & 2 & 9 & 8 & 4 & 1 & 5 \\ 2 & 1 & 3 & 7 & 8 & 5 & 9 & 4 & 6 \\ 6 & 8 & 5 & 9 & 4 & 2 & 7 & 3 & 1 \\ 4 & 9 & 7 & 6 & 3 & 1 & 8 & 5 & 2 \\ 7 & 5 & 1 & 8 & 2 & 9 & 3 & 6 & 4 \\ 9 & 3 & 2 & 4 & 1 & 6 & 5 & 8 & 7 \\ 8 & 6 & 4 & 5 & 7 & 3 & 1 & 2 & 9 \end{bmatrix}, \quad \boldsymbol{T} = \begin{bmatrix} 1.3 & -0.1 & 4 & 7 \\ 33 & -14 & 6 & -3.12 \end{bmatrix}$$

En lösning till pusslet skall uppfylla följande villkor: varje rad och varje kolonn i $\boldsymbol{S}$ liksom varje 3×3-matris, $\boldsymbol{A}, \ldots, \boldsymbol{I}$, skall innehålla siffrorna $1, 2, 3, 4, 5, 6, 7, 8, 9$ precis en gång. Matrisen $\boldsymbol{L}$ ovan är exempel på en lösning.

Skriv en *vektoriserad* funktion **`is_sudoku(M)`** som tar en matris som inparameter och returnerar **`true`** om **`M`** är en lösning (d.v.s. uppfyller villkoren ovan) och som returnerar **`false`** annars.

Du kan anta att din rutin anropas med en icke-tom matris med tal, men matrisen behöver inte vara en 9×9-matris och talen kan vara godtyckliga reella tal (flyttal). Din rutin kan, till exempel, anropas med matrisen $\boldsymbol{T}$ ovan (som alltså inte är en lösning, så att **`is_sudoku(T)`** skall returnera **`false`**).


```
function result = is_sudoku(S)

if any(size(S) ~= [9 9]) % Not a 9x9-matrix?
  result = false;
  return
end

% Test that each sub-matrix contains 1:9.
digits = (1:9)';
for row = 1:3:7    % row-start of sub-matrix
  for col = 1:3:7  % column-start
    t = S(row:row+2, col:col+2);% sub-matrix
    if any(sort(t(:)) ~= digits)
      result = false;
      return
    end
  end
end

% Test that 1:9 once in each column
for col = 1:9
  if any(sort(S(:, col)) ~= digits)
    result = false;
    return
  end
end

% and each row
digits = digits'; % transpose, since we are
for row = 1:9     % checking rows
  if any(sort(S(row, :)) ~= digits)
    result = false;
    return
  end
end

result = true; % S is a Sudoku-matrix
```