EL2310 – Scientific Programming

Lecture 2: Matlab as a Tool

Yasemin Bekiroglu (yaseminb@kth.se)

Royal Institute of Technology – KTH

Overview

Lecture 2: Matlab as a Tool

- Wrap Up
- Matrices (continued)
- Linear Algebra
- Plotting & Visualization
- Tasks for Home

Last time

- To get help
  `help, lookfor, helpdesk`
- To check defined variables
  `who, whos`
- To load/save variables in workspace
  `save, load`
- To clear variables
  `clear`
- To “write” a diary
  `diary`

Last time, too

- Initialize a vector
  `v = [1 2 3];`
- Initialize a matrix
  `M = [1 2 3; 4 5 6; 7 8 9];`
- Simple operations on scalars, matrices and vectors
  `w = M * v; w = v';`
- Access values of vectors and matrices
  `w(0);`
  `M(2,2); M(5);`

Element-by-element Operations

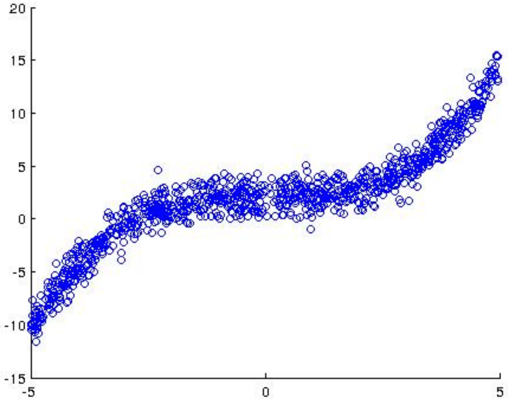
- Often we want to perform operations on independent elements of arrays
- Use the operator `.*  ./  .^`
- Examples:

```
>>A=[1 2]; B=[2; 2];
>>A * B
ans = 6
>>A .* B'
ans = [2 4]
```

Elementary matrices

- Many of the elementary matrices are predefined
- See more information with
  `help elmat`
- Examples
  Identity matrix: `I = eye(n);`
  Zero-matrix: `Z = zeros(n,m);`
  One-matrix: `O = ones(n,m);`
- If the second dimension is omitted, creates a rectangular matrix.

Some tools to deal with real data:

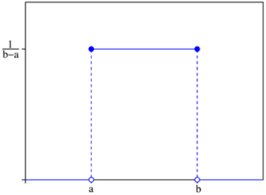


Solving linear systems

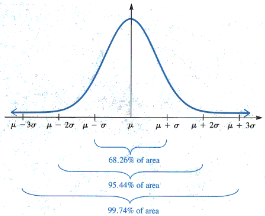
- You can solve a matrix equation $AX = B$ using $X = A\backslash B$. If B is invertible, this is the same as $X = A^{-1}B$, otherwise the solution is a solution in the least squares sense.

Random matrices

- Can easily create random matrices in [0, 1]
- Uniform distribution
  `rand(n,m)`
- Normal distribution
  `randn(n,m)`



- How to get a (2x2) matrix with uniformly distributed values in [3, 4] or [3, 10]?
- How to generate 100 values
  from a normal distribution
  with mean 1 and standard deviation 2?



Sequences

- Enumerate
  Ex: `v = [1 3 7];`
- Colon notation (function `colon`)
  Ex: `v = 1:9;`
  Ex: `v = 1:2:9;`
- More general linearly spaced vectors
  - `v = linspace(start_value, end_value, N);`
  - Generates *N* values between `start_value` and `end_value`
  - Do not have to calculate the step yourself
- Logarithmically spaced vector
  - `v = logspace(start_exp, end_exp, N);`
  - Calculates N linearly spaced values between `start_exp` and `end_exp` and 10 to the power of these values.
  - $\text{logspace}(x_1, x_2, \text{N}) = 10^{\text{linspace}(x_1, x_2, N)}$

Size of matrices

- You get the size of a matrix (rows and columns) with
  `size(A)`
- Number of rows
  `size(A,1)`
- Number of columns
  `size(A,2)`
- For a vector you get the length with
  `length(v)`
- For matrices `length(A)` gives “largest” dimension
- Often convenient to use `end` for index `v(4:end) = 0;`
  (you do not need to know the size)

Creating matrices

- Diagonal matrices can be created with
  `diag(<vector>)`
- Creates a matrix with the vector on the diagonal, that is a square matrix of dimensions equal to the length of the vector argument
- You can shift the vector up and down from the diagonal
  `diag(<vector>, k)`
  where $k > 0$ means shifting up and $k < 0$ mean shifting down
- You can also create diagonal block matrices with
  `blkdiag(M1, M2, ...)`

Manipulating matrices

- Get lower triangular part `tril(A)`
- Get upper triangular part `triu(A)`
- Flip a matrix upside down `flipud(A)`
- Flip a matrix left/right `fliplr(A)`
- Rotate matrix 90° counter clock wise `rot90(A)`

```
>> A=[1 2 3;4 5 6;7 8 9];
>> tril(A)

ans =

     1     0     0
     4     5     0
     7     8     9

>> triu(A)

ans =

     1     2     3
     0     5     6
     0     0     9
```

Changing matrix shape

- Sometimes useful to change the shape of a matrix
- Ex: You have an array $x_1, y_1, x_2, y_2, \ldots, x_N, y_N$ and you want to make a matrix with (x, y) column vectors
- `reshape(A,n,m)`; goes through matrix/vector A column wise

```
A =
   1     4     7    10
   2     5     8    11
   3     6     9    12

B = reshape(A,2,6)

B =
   1     3     5     7     9    11
   2     4     6     8    10    12
```

Finding elements

- You can find non-zero elements
  `[ind] = find(A)`
  returns the linear index (single index)
- Can get the subscripts by providing two output arguments
  `[ii, jj] = find(A)`
- Can replace test for non-zero with a logic expression such as
  `[ii, jj] = find(A>3)`
- Note that `A>3` is a matrix of the same dimension as A and with 1-elements for each element in A that is $>$ 3 and 0 for the rest

Linear algebra (some examples)

- Easy to calculate basic linear algebra
- Inverse: `inv(A)`
- Determinant: `det(A)`
- Rank: `rank(A)`
- Trace: `trace(A)`

Linear algebra: Eigenvalues

- Finding eigenvalues
  `eig(A)`
- Getting eigenvalue and vectors
  `[V,D] = eig(A)`
  `V` full matrix contains the eigen vectors (columns) and `D` is a diagonal matrix with the eigenvalues on the diagonal
  Fulfills $AV = VD$

Linear algebra: Singular value decomposition (SVD)

- Calculating svd is simple
  `[U,S,V] = svd(A)`
- Fulfills $A = U * S * V^T$
- `s = svd(A)` gives the singular values

Square root matrix

- Square root matrix fulfills $A = XX$
- Calulated with `X = sqrtm(A);`
- Remember: Element wise multiplication with `.*`

```
>> A = [1 2;3 4]

A =

     1     2
     3     4

>> As = sqrtm(A)

As =

   0.5537 + 0.4644i   0.8070 - 0.2124i
   1.2104 - 0.3186i   1.7641 + 0.1458i

>> As*As

ans =

   1.0000 + 0.0000i   2.0000
   3.0000 + 0.0000i   4.0000

>> As.*As

ans =

   0.0909 + 0.5143i   0.6061 - 0.3428i
   1.3636 - 0.7714i   3.0909 + 0.5143i
```

More operations

- Easy to calculate mean, standard deviation, etc.
- Applies to a vector or columns of a matrix
- Mean value: `mean(v)`
- Standard deviation: `std(v)`
- Min value : `min(v)` (also `min(A,2)`)
- Max value : `max(v)` (also `max(A,2)`)
- Sum : `sum(v)`
- Difference : `diff(v)`
- Cumulative sum: `cumsum(v)`
- Covariance: `cov(X)`

More operations cont.

- Useful tip: Convert a matrix to column vector `A(:)`
  What's `min(A)` and `min(A(:))` if `A` is a matrix?
- Additional parameter specifies dimension:
  `mean(A, 1 or 2)`
  `min(A, [], 1 or 2)` Why `[]`?
  `max(A, [], 1 or 2)`
  `sum(A, 1 or 2)`

Plotting data

- Plotting data with
  `plot(x,y)`
- With one argument the x-axis will be the vector index and the y-axis the value of the input vector
- Can specify color and type of line/points, e.g.
  `plot(x,y,'r.')` to get a red dot for every data point
- For more information do
  `help plot`
- Example: Plot $\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$, i.e. a normal distribution with standard deviation σ and mean value μ.

Titels, labels, etc

- Label the axes with
  ```
  xlabel('text on the x-axis')
  ylabel('text on the y-aixs')
  ```
- and give a title with
  ```
  title('Some nice title')
  ```
- You can change the font size by adding extra arguments
  ```
  xlabel('text on the x-axis', 'FontSize',20)
  ```

Handles and set/get

- Calls to graphics functions return a “handle”
- Can use this handle to set/get properties
- `h = title('Some nice title');`
- List properties with
  `get(h);`
- Set property with
  `set(h, 'FontSize', 20);`
- Get current handle:
  `gcf` - figure
  `gca` - axes

Plotting continued

- You can plot more than one thing at a time:
  `plot(x1, y1, x2, y2)`
  will plot x1 against y1 and x2 against y2 in the same graph
- Each pair assigned it own color automatically
- You can manually specify color/marker for each:
  `plot(x1, y1, 'r', x2, y2, 'b')`
- Every plot call will clear the figure
- Use `hold on` and `hold off` to stop from clearing

```
hold on
plot(x1,y1)
plot(x2,y2)
hold off
```

More plotting

- You can provide labels for your data with `legend`
```
plot(x1, y1, x2, y2)
legend('data set 1', 'data set 2')
```
- You can specify which figure window something goes to with
```
figure(n)
```
If specified window does not exist it will be created
- You can clear a figure (the current one) with `clf`
- Can get grid with `grid`
- Can plot with one or both axis in logarithmic scale
```
semilogx(x,y)
semilogy(x,y)
loglog(x,y)
```

Data Analysis

- Let's generate, plot and analyse data with Matlab

Exercises

- Generate a vector of normally distributed random samples
- Compute the mean and standard deviation from the samples
- Generate two sequences of random samples and compute covariance

Exercises

- Generate a ”data set” using $x = 5 - 10 * rand(1, 1000)$ $y = 2 + 3 * x + randn(1, 1000)$.
- Save the result in a file *data.mat*.

Exercises

- Assume someone hands you the data generated in the previous task without information about how it was generated.
- Load and plot the (x, y) data to understand it (try *scatter*).
- Assume that you don't know how the data was actually generated. Try to fit line to the data (x, y) using just the data samples.
- Plot your line approximation

Exercises

- Generated a ”data set” using
  x = 5-10*rand(1, 1000)
  y = 2+0.1*x.ˆ3+randn(1, 1000).
- Assume someone hands you the data above without any information about how it was generated.
- Plot the (x, y) data to understand it.
- Read about regression methods online and check useful matlab commands.
- Can you fit a non-linear function to the data?
- Quantify the error in your approximation compared to a simple line fit to this data?

Next time

- Finish up plotting
- Functions and scripts in detail

The First Presentation: PCA

- Explain what Principal Component Analysis (PCA) does, how it works and for what type of problems it is used.
- Implement it, compare your implementation with Matlab's built-in pca function on a dataset with different classes that has a large dimensionality. You can create your own data with multiple classes with random samples or use an already available dataset (from Matlab or another source).

The First Presentation: PCA

- Visualize the data in the new space and observe if data samples from the same classes are close to each other.
- How should we choose the number of eigen vectors to represent data without losing information?
- How can we implement a PCA-based face recognition method? (http://vision.ucsd.edu/content/yale-face-database)

The Second Presentation: Kmeans

- Explain what kmeans clustering algorithm does, how it works and for what type of problems it is used.
- Implement it and apply it on the IRIS dataset (load fisheriris)
- Compare your implementation with Matlab's built-in function. Do you get the same results?
- What are the factors that affect the performance of the algorithm?
- Apply your function to another dataset and evaluate the performance: e.g., kmeansdata.mat from Matlab

Matlab Project

- Coin Detection - Hough Transform
- Deadline 17 Sep, thursday 20:00
- Project description along with solutions to the exercises from the first lecture are available on course homepage.
- Help session on Wed 9/9 at 13:00 at 304 (22:an), Teknikringen 14.