KU LEUVEN

# MATLAB

basic mathematical applications

1

## Topics

- basic data analysis: summary statistics
- interpolation
- curve fitting
- polynomials
- solving linear equations
- function optimisation



3

## Workflow data analysis

**get the data**
- data files: different formats
- hardware: using interfaces, Data Acquisition Tbx, etc.

**preprocess / visualise**
- view the data
- scaling and averaging
- interpolating
- extracting sections of data

**analyse / visualise**
- basic statistics, curve fitting
- matrix analysis
- zero finding
- differential equation solvers

**share**
- publish the results

Based on Steve Lantz: Data Analysis with MATLAB, http://www.cac.cornell.edu/education/training



4

## Basic data analysis functions



6

# Summary Statistics functions

- `help datafun`

- Minimum of in a Data Set >> `min(v)`
- Maximum of in a Data Set >> `max(v)`
- Sum of a Data Set >> `sum(v)`
- Standard Deviation >> `std(v)`
- Mean of a Data Set >> `mean(v)`
- Sort a Data Set in ascending order >> `sort(v)`

remark:
MATLAB considers data sets stored in column-oriented arrays

KU LEUVEN

7

# Summary Statistics functions

- Cumulative functions
  - Cumulative sum >> `cumsum`
  - Cumulative product >> `cumprod`
- Integration
  - Trapezoidal numerical integration >> `trapz`
  - Cumulative trapezoidal numerical integration >> `cumtrapz`

8 KU LEUVEN

8

# mean

```
>> temp3city =

    12     8    18
    15     9    22
    12     5    19
    14     8    23
    12     6    22
    11     9    19
    15     9    15
     8    10    20
    19     7    18
    12     7    18
    14    10    19
    11     8    17
     9     7    23
     8     8    19
    15     8    18
     8     9    20
    10     7    17
    12     7    22
     9     8    19
    12     8    21
    12     8    20
    10     9    17
    13    12    18
     9    10    20
    10     6    22
    14     7    21
    12     5    22
    13     7    18
    15    10    23
    13    11    24
    12    12    22
```

```
>> mean(temp3city, 1)
ans =
   11.9677    8.2258   19.8710
>> mean(temp3city, 2)
ans =
   12.6667
   15.3333
   12.0000
   15.0000
   13.3333
   13.0000
   13.0000
   12.6667
   14.6667
   12.3333
   14.3333
   12.0000
   13.0000
   11.6667
   13.6667
   12.3333
   11.3333
   13.6667
   12.0000
   13.6667
   13.3333
   12.0000
   14.3333
   13.0000
…
```

- vectors: `mean(X)` is the mean value of the elements in X.
- matrices, `mean(X)` is a row vector containing the mean value of each column.
- `mean(X,DIM)` takes the mean along the dimension DIM of X.

KU LEUVEN

9

# median

- `median(x)` same as `mean(x)`, only returns the median value.

```
>> X = [0 1 2; 3 4 5]

X =

     0     1     2
     3     4     5

>> median(X,1)

ans =

    1.5000    2.5000    3.5000

>> median(X,2)

ans =

     1
     4
```

KU LEUVEN

10

# std

Standard deviation

- There are two common textbook definitions for the standard deviation s of a data vector X.
- Def_1 $$s=\left(\frac{1}{n-1}\sum_{i=1}^{n}(x_i-\bar{x})^2\right)^{\frac{1}{2}}$$
- Def_2 $$s=\left(\frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})^2\right)^{\frac{1}{2}}$$
- `s = std(X)`
  X is a vector, returns the standard deviation using (1) above.
- `s = std(X,flag)`
  flag = 0, is the same as std(X).
  flag = 1, std(X,1) returns the standard deviation using (2)
- `s = std(X,flag,dim)` computes the standard deviations along the dimension of X specified by scalar dim.

```
>> dat =

    12     8    18
    15     9    22
    12     5    19
    14     8    23
    12     6    22
    11     9    19
    15     9    15
     8    10    20
    19     7    18
    12     7    18
    14    10    19
    11     8    17
     9     7    23
     8     8    19
    15     8    18
     8     9    20
    10     7    17
    12     7    22
     9     8    19
    12     8    21
    12     8    20
    10     9    17
    13    12    18
     9    10    20
    10     6    22
    14     7    21
    12     5    22
    13     7    18
    15    10    23
    13    11    24
    12    12    22
```

```
>> std(dat)
ans =
    2.5098    1.7646
2.2322
>> std(dat,1)
ans =
    2.4690    1.7360
2.1959
>> std(dat,0,2)
ans =
    5.0332
    6.5064
    7.0000
    7.5498
    8.0829
    5.2915
    3.4641
    6.4291
    6.6583
    5.5076
    4.5092
    4.5826
    8.7178
    6.3509
    5.1316
    6.6583
    5.1316
    7.6376
    4.3589
...
```

KU LEUVEN

11

# max (min)

- `max` Largest component.
- vectors: `MAX(X)` is the largest element in X.
- matrices: `MAX(X)` is a row vector containing the maximum element from each column.
- `[Y,I] = max(X)` returns the indices of the maximum values in vector I.
- `max(X,Y)` returns an array the same size as X and Y with the largest elements taken from X or Y. Either one can be a scalar.


```
>> dat =

    12     8    18
    15     9    22
    12     5    19
    14     8    23
    12     6    22
    11     9    19
    15     9    15
     8    10    20
    19     7    18
    12     7    18
    14    10    19
    11     8    17
     9     7    23
     8     8    19
    15     8    18
     8     9    20
    10     7    17
    12     7    22
     9     8    19
    12     8    21
    12     8    20
    10     9    17
    13    12    18
     9    10    20
    10     6    22
    14     7    21
    12     5    22
    13     7    18
    15    10    23
    13    11    24
    12    12    22
```


```
>> max(dat)

ans =

    19    12    24

>> max(max(dat))

ans =

    24

>> max(dat(:))

ans =

    24
>> [Y, I] = max(dat)

Y =

    19    12    24


I =

     9    23    30
```

KU LEUVEN

12

# diff

- `Y = diff(X)` calculates differences between adjacent elements of X.
- vector: returns a vector, 1 element shorter than X, of differences between adjacent elements: [X(2)-X(1) X(3)-X(2) ... X(n)-X(n-1)]
- matrix, then diff(X) returns a matrix of row differences: [X(2:m,:)-X(1:m-1,:)]
- `Y = diff(X,n)` applies diff recursively n times,
  diff(X,2) is same as diff(diff(X)).
- `Y = diff(X,n,dim)` is the nth difference function calculated along the dimension specified by scalar dim.


```
>> dat =

    12     8    18
    15     9    22
    12     5    19
    14     8    23
    12     6    22
    11     9    19
    15     9    15
     8    10    20
    19     7    18
    12     7    18
    14    10    19
    11     8    17
     9     7    23
     8     8    19
    15     8    18
     8     9    20
    10     7    17
    12     7    22
     9     8    19
    12     8    21
    12     8    20
    10     9    17
    13    12    18
     9    10    20
    10     6    22
    14     7    21
    12     5    22
    13     7    18
    15    10    23
    13    11    24
    12    12    22
```


```
>> Y
Y =
    19    12    24
>> diff(Y)
ans =
    -7    12
>> diff(Y,2)
ans =
    19
>> diff(dat)
ans =
     3     1     4
    -3    -4    -3
     2     3     4
    -2    -2    -1
    -1     3    -3
     4     0    -4
    -7     1     5
    11    -3    -2
    -7     0     0
     2     3     1
    -3    -2    -2
    -2    -1     6
    -1     1    -4
     7     0    -1
    -7     1     2
     2    -2    -3
     2     0     5
    -3     1    -3
     3     0     2
     0     0    -1
    -2     1    -3
     3     3     1
    -4    -2     2
     1    -4     2
     4     1    -1
    -2    -2     1
     1     2    -4
     2     3     5
    -2     1     1
    -1     1    -2
```

KU LEUVEN

13

# filter

filters a data sequence using a digital filter

- `y = filter(b,a,X)` filters the data in vector X with the filter described by numerator coefficient vector b and denominator coefficient vector a.
  If X is a matrix, filter operates on the columns of X.
- Algorithm:
  a(1)y(n) = b(1)*x(n) + b(2)*x(n-1) + ... + b(nb+1)*x(n-nb) - a(2)*y(n-1) - ... - a(na+1)*y(n-na)
- more detail in `help filter`
  http://www.mathworks.nl/help/techdoc/data_analysis/bqm3i7m-1.html
- *File: demo_filter_movingaverage*

KU LEUVEN

14

## filter

Table 1. Filtering

| Year | Filter | Time Series | Filtered Values |
|---|---|---|---|
| 1 | | 12 | |
| 2 | .25 x | 17 | 14.00 |
| 3 | .50 x | 10 | 14.75 |
| 4 | .25 x | 22 | 17.25 |
| 5 | | 15 | 15.75 |
| 6 | | 11 | 13.75 |
| 7 | | 18 | 18.50 |
| 8 | | 27 | 21.50 |
| 9 | | 14 | |

```
>> time_series = [12 17 10 22 15 11 18 27 14]

time_series =

    12    17    10    22    15    11    18    27    14

>> a = [.25 .50 .25]

a =

    0.2500    0.5000    0.2500

>> filter(a,1,time_series)

ans =

  Columns 1 through 8

    3.0000   10.2500   14.0000   14.7500   17.2500   15.7500
  13.7500   18.5000

  Column 9

   21.5000
```

KU LEUVEN

15

## more functions

- `corrcoef`: correlation coefficients
- `cov`: covariance matrix
- `histc(x, edges)`: histogram count and bin locations using bins marked by edges
- `sort`: sorts in ascending or descending order
- `sortrows`: sort rows in ascending order (demo_sort_sortrow.m)

KU LEUVEN

16

## Hands-on: Data Preprocessing

- Missing values:
  - remove NaNs from the data before performing computations.
  - demo_check_NaN
- Removing outliers:
  - remove outliers or misplaced data points from a data set
    - Calculate the mean and standard deviation from the data set.
    - Get the column of points that lies outside the 3*std. (3σ-rule)
    - Remove these points
    - Check `isoutlier`
  - demo_check_outlier.m
- More examples:
  - demo_basic_DataAnalysis
  - data_treatment_cambridge

KU LEUVEN

20

## interpolation

21

KU LEUVEN

21

## Fitting/interpolation

Tips:

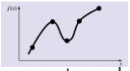
- Young/Mohlenkamp
  - http://www.ohiouniversityfaculty.com/youngt/IntNumMeth/
- Check Cleve Moler’s website
  - https://nl.mathworks.com/moler.html
- Help
  - Search in helpdesk data analysis
  - Interpolation
  - fitting

22

## Interpolation

- A way of estimating values of a function between those given by some set of data points
  - When you take data, how do you predict what other data points might be?
  - Two techniques are :
    - Linear Interpolation

    - Cubic Spline Interpolation



- Extrapolation: be careful, make sure extrapolation makes sense with your data

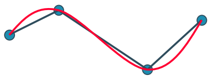
23

## Interpolation

• `YI = interp1(X,Y,XI)` interpolates to find YI, the values of the underlying function Y at the points in the vector or array XI

- Finding values between data points
  - •linear interpolation (default)
  - •cubic splines

```
yi = interp1 ( x, y, xi )
Yi = interp1 ( x, y, xi, 'spline' )
```



•*demo_interp1*
•*demo_interp2*
•*demo_interp3*
•*demo_interp4*

24

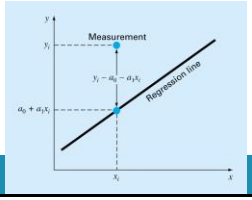
## Fitting

- Idea: modeling data with an equation
- We can estimate what equation represents the data by “eyeballing” a graph
  - there will be points that do not fall on the line we estimate
  - There is scatter in collected data
- Linear and cubic spline interpolations fit curves constrained to go through the data points
- A curve fitted using least squares may not pass through any data point but it will be “close” to all of them in a “least squares” sense
- Find a function,e.g. a polynomial of whatever order, that minimizes the mean square error

25

## Linear regression

- Linear equation that is the best fit to a set of data points
- Minimize the sum of squared distances between the line and the data points
- Use `polyfit`
- *demo_linreg1.m*




KU LEUVEN



## Polynomial Fit

- A pair of vectors (x, y), representing the independent and dependent variables of a (possibly noisy) relationship.
- Get the curve that most closely fits the data.
- `polyfit` finds the coefficients of a polynomial representing the data
- generates a “best fit” polynomial (in the least squares sense) of a specified order for a given set of data.
- used to generate the n + 1 coefficients $a_j$ of the nth-degree polynomial used for fitting the data.

$$p_n(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_n x + a_0$$

```
as = polyfit(x,y,n)
```

KU LEUVEN



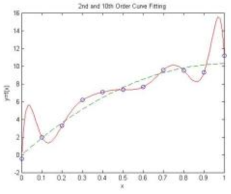
## Polynomial Fit

- `polyval:` use the coefficients to find new values of y, that correspond to the known values of x
- Create a new set of y points for the approximate curve with
- `yapprox = polyval(as,x)`

KU LEUVEN



## polyfit




*demo_polyfit*

```
>> x = [0 .1 .2 .3 .4 .5 .6 .7 .8 .9
  1];
>> y = [-.447 1.978 3.28 6.16 7.08 7.34
  7.66 9.56 9.48 9.30 11.2];
>> n = 2;
>> p = polyfit(x,y,n);
>> p =
  -9.8108   20.1293   -0.0317
xi = linspace(0,1,100);
yi = polyval(p,xi);
pp = polyfit(x,y,10);
y10 = polyval(pp,xi);
plot(x,y,'o',xi,yi,'--',xi,y10) % plot
  data
xlabel('x'), ylabel('y=f(x)')
title('2nd and 10th Order Curve
  Fitting')
```

KU LEUVEN

## polynomials

- Polynomials are described by using a row vector of the coefficients of the polynomial beginning with the highest power of *x* and inserting zeros for "missing" terms:

```
f=[9 -5 3 7];
g=[6  -1 2];
```

- add and subtract polynomial functions in MATLAB. To do this we must "zero" pad the polynomials so that their row vector representations are the same length:

```
h=f+[0 g];
```

- multiply and divide polynomials by using the `conv` and `deconv` functions

```
y=conv(f,g)
[q r]=deconv(y,f)
```

- evaluate a polynomial at any value of x:

```
p=[1 3 -4];
x=[-5:.1:5];
px=polyval(p,x);
```

KU LEUVEN



## polynomials

- `roots` finds polynomial roots.

```
roots( [ 1   6   11   6   ] )
ans =      -3.0000
      -2.0000
      -1.0000
```

- `polyder(P)` returns the derivative of the polynomial whose coefficients are the elements of vector P.

```
 polyder( [ 1   6   11   6 ] )
ans = [   3    12    11   ]
```

- `polyint(P)` returns the integral of the polynomial whose coefficients are the elements of vector P.

KU LEUVEN



## Linear equations



KU LEUVEN



## Linear equations

Tips:

- Check Cleve Moler's website
  - www.mathworks.com/moler
- Help
  - Systems of linear equations
  - Backslash operator



KU LEUVEN

## Simultaneous Linear Equations

- Basic form:

$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n = b_1$
$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ldots + a_{2n}x_n = b_2$
.........
$a_{m1}x_n + a_{m2}x_2 + a_{m3}x_2 + \ldots + a_{mn}x_n = b_n$

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & & \vdots \\ \vdots & & & \vdots \\ \vdots & & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ \vdots \\ \vdots \\ b_m \end{bmatrix}$$

$$-or-$$

$$Ax = b$$

- Where:
  - $a_{ij}$ are known coefficients
  - $x_i$ are unknowns
  - $b_i$ are known right hand side values

KU LEUVEN

36

## Simultaneous Linear Equations

- The coefficient matrix A need not be square.
  If A is *m*n*, there are three cases.
  - *m = n*
    - Square system.
    - Seek an exact solution.
  - *m > n*
    - Overdetermined system.
    - Find a least squares solution.
  - *m < n*
    - Underdetermined system.
    - Find a basic solution with at most *m* nonzero components.
- Check *rank* of a matrix (number of independent rows or columns). : `rank`

KU LEUVEN

37

## Linear Equations

- Set of linear equations Ax = b can be solved using the Inverse operation on A
  `x=inv(A)*b`
- Advice:
  - Do not use this method
  - Inefficient

basicMath_
38

KU LEUVEN

38

## backslash \

- MATLAB has a bunch of methods available to solve a system of linear equations
- If you desire the solution of Ax = b, then the simplest method using Matlab to find x is to set x = A\b

```
A = [ 1 5 6; 7 9 6; 2 3 4]
b = [29; 43; 20]
x=A\b
```

- Use backslash operator **** (LU decomposition + pivoting)
- For non-square and singular systems, the operation A\b gives the solution in the least squares sense. No error message
- The " \ " backslash operator uses a combination of numerical methods including LU decomposition.
- `doc mldivide`

KU LEUVEN

39

## Steps in A\b

Stop if successful

- If A is upper or lower triangular, solve by back/forward substitution
- If A is permutation of triangular matrix, solve by permuted back substitution (useful for [L,U]=lu(A) since L is permuted)
- If A is symmetric/hermitian
  - Check if all diagonal elements are positive
  - Try Cholesky, if successful solve by back substitutions
- If A is Hessenberg (upper triangular plus one subdiagonal), reduce to upper triangular then solve by back substitution
- If A is square, factorize PA = LU and solve by back substitutions
- If A is not square, run Householder QR, solve least squares problem
  - demo_lineqN
  - check
  - scicomp.stackexchange.com/questions/1001/how-does-the-matlab-backslash-operator-solve-ax-b-for-square-matrices
  - http://www.mathworks.nl/support/solutions/en/data/1-172BD/index.html?product=ML&solution=1-172BD

KU LEUVEN

40

## More

42

KU LEUVEN

42

## Basic optimization

- Optimization deals with finding the maxima and minima of a function that depends on one or more variables.
- Basic functions available in Matlab + Optimization toolbox
- Minimization in 1 dimension
  - `X = fminbnd(FUN,x1,x2)` attempts to find a local minimizer X of the function FUN in the interval x1 < X < x2. FUN accepts scalar input X and returns a scalar function value F evaluated at X.
    - *demo_fminbnd*
    - *demo_fminbnd_2*
    - *demo_fminbnd_3*
- Minimization in N dimensions
  - `X = fminsearch(FUN,X0)` starts at X0 and attempts to find a local minimizer X of the function FUN. FUN accepts input X and returns a scalar function value F evaluated at X. X0 can be a scalar, vector or matrix.
  - Multidimensional unconstrained nonlinear minimization (Nelder-Mead).

KU LEUVEN

43

## Zero finding

- `fzero`: Find zero of a function of one variable
  find *x* satisfying *f(x)* = 0
- An implementation of T. Dekker's algorithm
- Combination of
  - Secant method / quadratic fit extension of Secant method
  - Bisection
- While not universally effective, combining techniques can improve the reliability and speed of convergence
- fzero uses bisection to avoid large steps, other methods for fast convergence
- syntax:
  `x = fzero(fun,x0)`
  - ***fun*** user supplied function specifying *f(x)*
  - ***x0*** is an initial guess of an *x*

KU LEUVEN

44

## Even More

- Solving ODE
- Eigenvalues
- Solving PDE
- Sparse systems
- …

KU LEUVEN