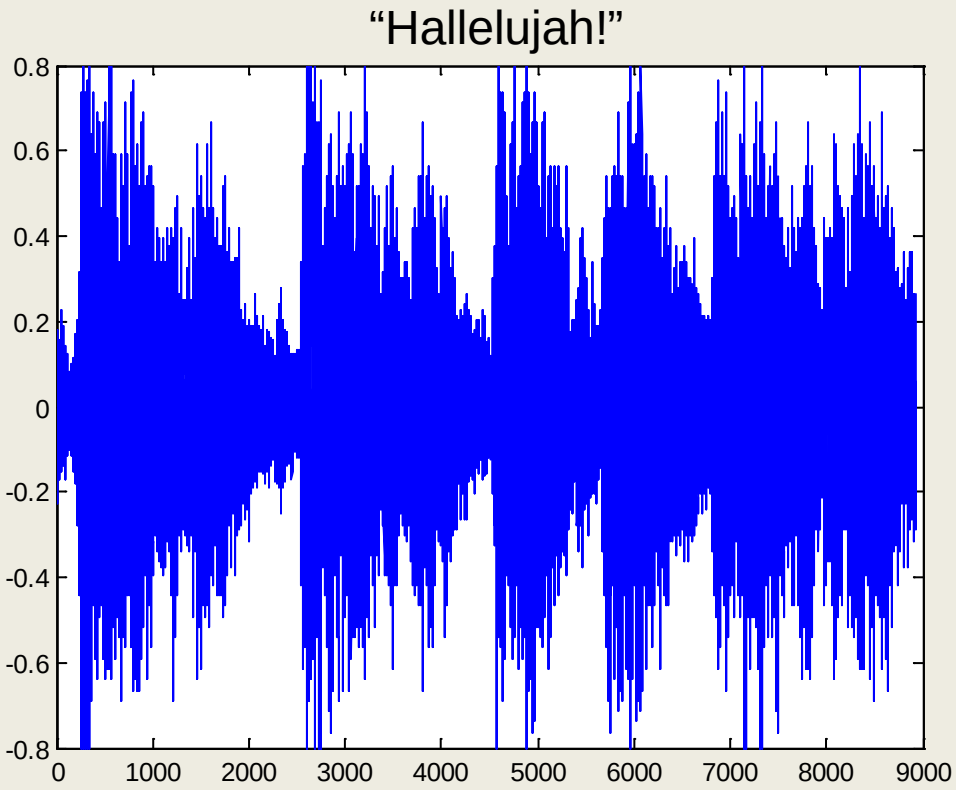
Basic Ingredients For Your Matrix Laboratory – Vectors and Matrices In Matlab

Olaf Hauk

MRC Cognition and Brain Sciences Unit
olaf.hauk@mrc-cbu.cam.ac.uk

A Sound File Contains a Vector



An Image File Contains a Matrix

Data Come In Matrices

Example: EEG data

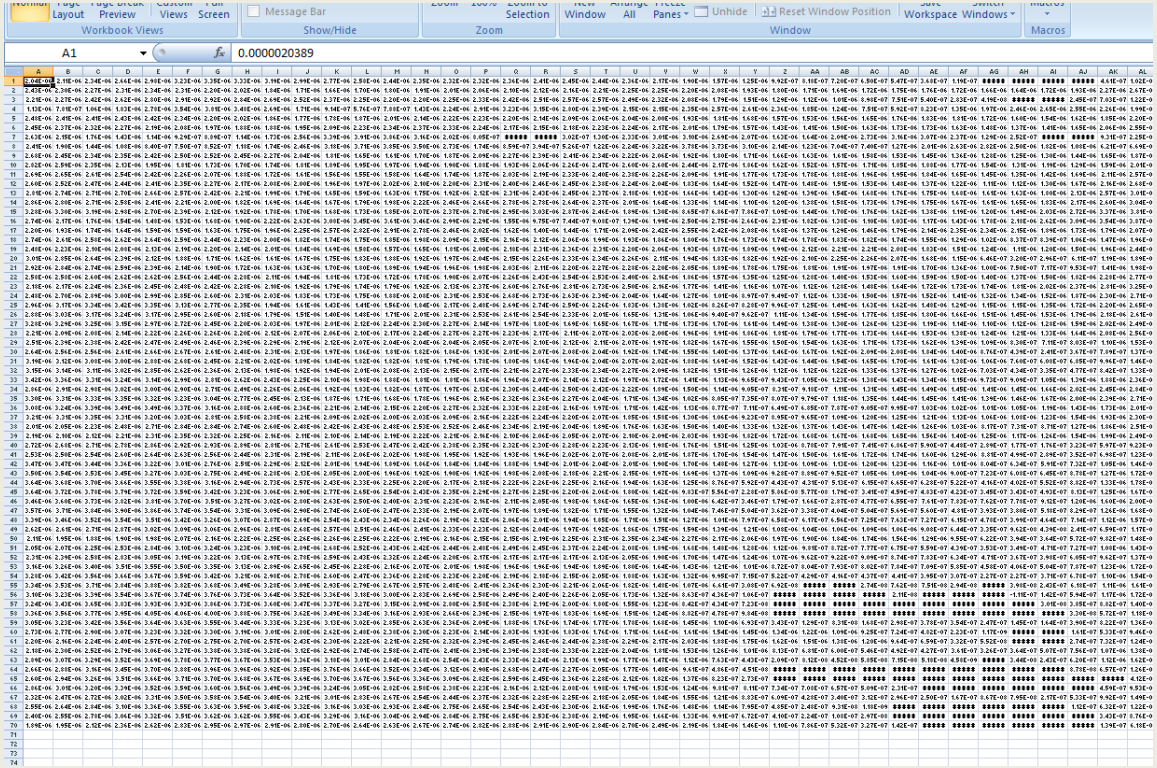
Electrodes
Rows

Time Points
Columns

=> Almost all data come in matrices!

EEG Data Matrix

Electrodes (70)
Rows

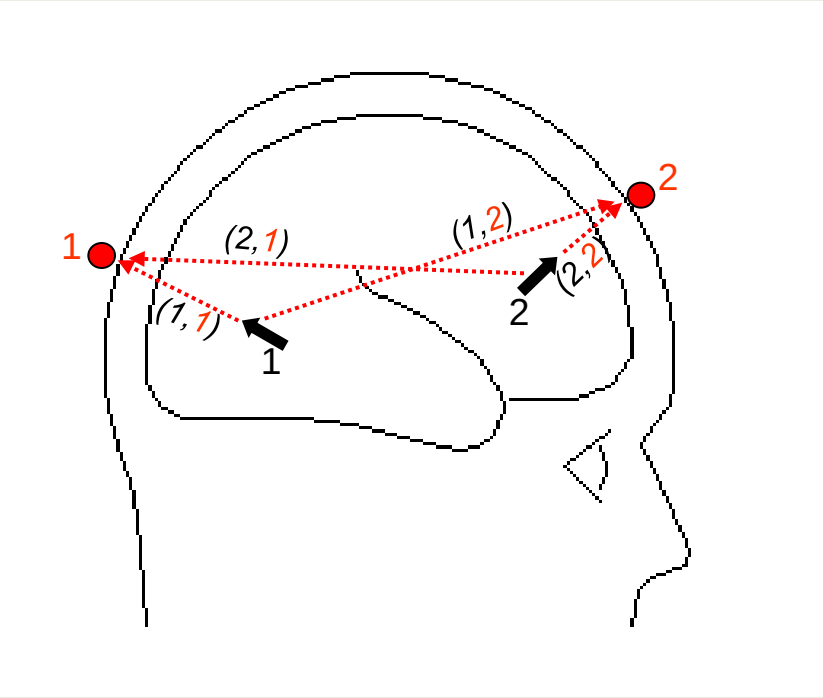


Time Points (200)
Columns

Lots of data manipulations will take a form like this:

Result = Matrix*Data

EEG/MEG Inverse Problem



data "leadfield" dipoles

$$\begin{pmatrix} d_1 \\ d_2 \end{pmatrix} = \begin{pmatrix} 0.5 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} j_1 \\ j_2 \end{pmatrix}$$

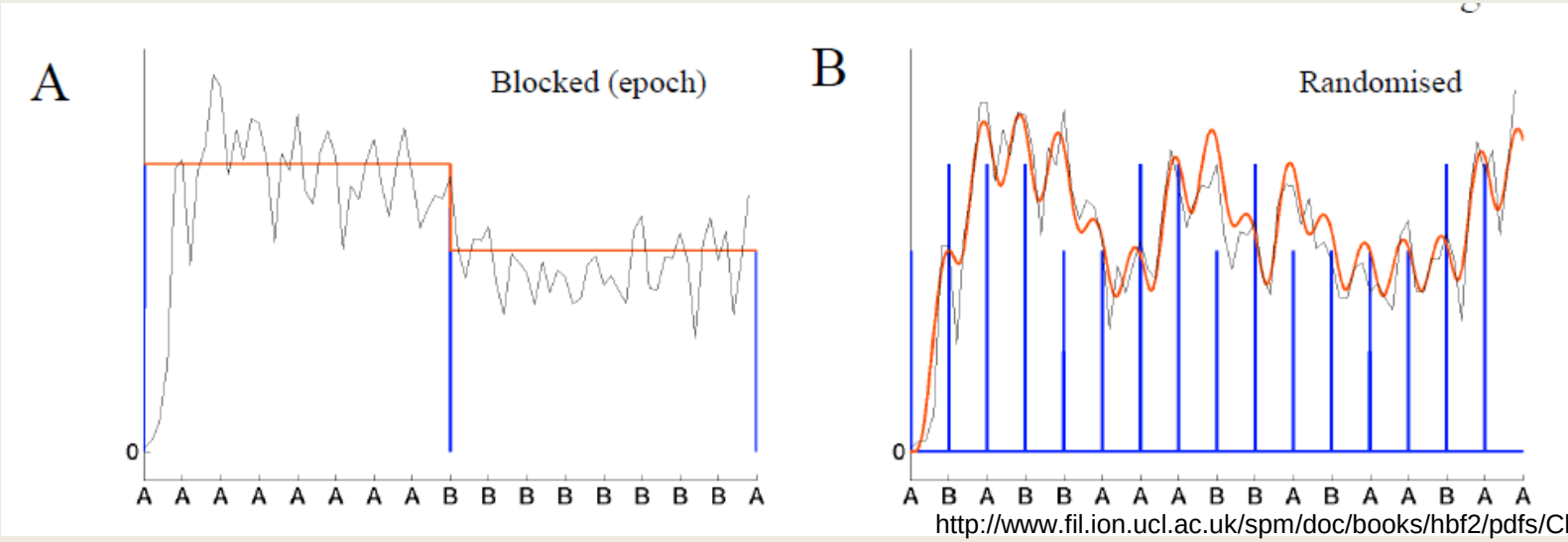
inversion

dipoles inverse data

$$\begin{pmatrix} \hat{j}_1 \\ \hat{j}_2 \end{pmatrix} = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \end{pmatrix}$$

Many models are described using matrices!

fMRI General Linear Model



http://www.fil.ion.ucl.ac.uk/spm/doc/books/hbf2/pdfs/Ch10.pdf

General Linear Model…

- *f*MRI time series: $Y_1,\ldots,Y_s,\ldots,Y_N$
  - acquired at times $t_1,\ldots,t_s,\ldots,t_N$
- Model: Linear combination of basis functions

$$Y_s = \beta_1 f^1(t_s) + \ldots + \beta_L f^1(t_s) + \ldots + \beta_L f^L(t_s) + \varepsilon_s$$

- $f^l(.)$: *basis* functions
  - *"reference waveforms"*
  - *dummy variables*
- β_l: *parameters* (*fixed effects*)
  - amplitudes of basis functions *(regression slopes)*
- ε_s: *residual errors*: $\varepsilon_s \sim N(0,\sigma^2)$
  - identically distributed
  - independent, or serially correlated *(Generalised Linear Model → GLM)*

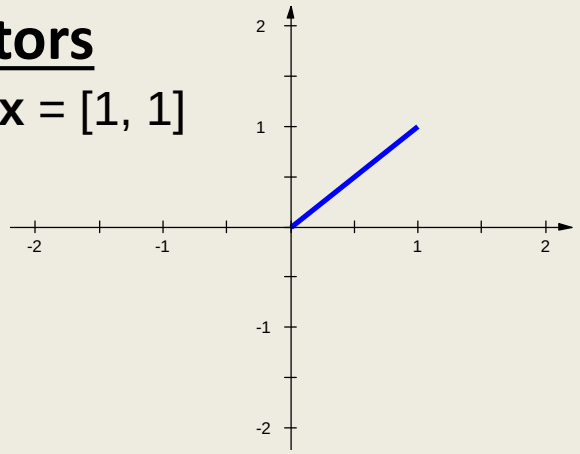
www.fil.ion.ucl.ac.uk/spm/course/slides02/ppt/glm.ppt

Example

2D Visualisation of Vectors

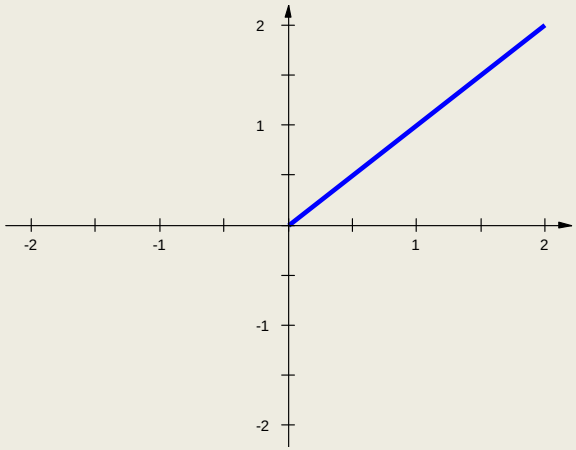
A “vector” is a list of numbers, e.g.: **x** = [1, 1]

x =



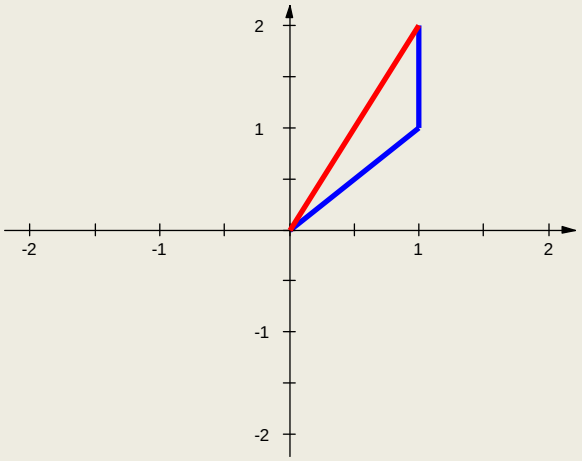
Multiplication with a scalar:

2*x = [2 2]



Vector addition/subtraction:

[1 1] + [0 1] = [1 2]



Vector Multiplication

Definition of "scalar product": $\mathbf{x} \bullet \mathbf{y} = \begin{pmatrix} x_1 \\ \dots \\ x_i \\ \dots \\ x_N \end{pmatrix} \bullet \begin{pmatrix} y_1 \\ \dots \\ y_i \\ \dots \\ y_N \end{pmatrix} = \sum_{i=1}^{N} x_i y_i$

So called because its result is a scalar

[1 1]• [1 1] = 1*1 + 1*1 = 2

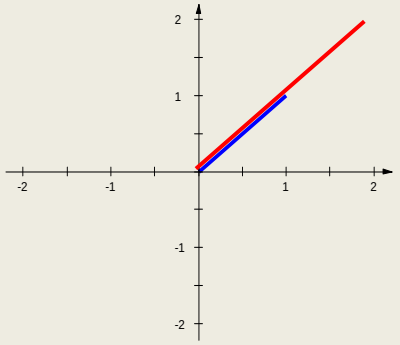
Some conventions:

Numbers/scalars are usually in normal/italic font: 12, *q*, λ

Vectors are usually bold: **x**, their elements are not: $\mathbf{x} = \begin{pmatrix} x_1 \\ \dots \\ x_i \\ \dots \\ x_N \end{pmatrix}$

Vector Multiplication

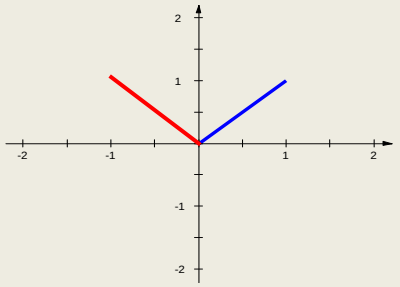
Parallel Vectors:
same concept as scalar multiplication, the “lengths multiply”



[1 1]• [2 2] = 1*2 + 1*2 = 4

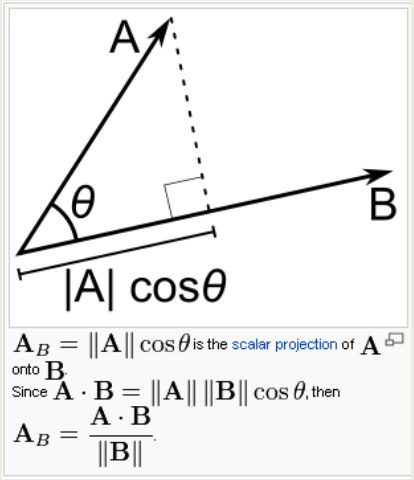
√2 • √8 = √16 = 4

Orthogonal Vectors:
nothing “covaries” -> 0



[1 1]• [-1 1] = 1*1 + 1*(-1) = 0

When vectors not parallel:
“correction” according to angle (cosine)



$\mathbf{A}_B = \|\mathbf{A}\| \cos\theta$ is the scalar projection of $\mathbf{A}$ onto $\mathbf{B}$.
Since $\mathbf{A} \cdot \mathbf{B} = \|\mathbf{A}\| \, \|\mathbf{B}\| \cos\theta$, then
$\mathbf{A}_B = \dfrac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{B}\|}$.

$$\mathbf{A} \bullet \mathbf{B} = \|\mathbf{A}\| \bullet \|\mathbf{B}\| \cos\theta$$

$$\Rightarrow \cos\theta = \frac{\mathbf{A} \bullet \mathbf{B}}{\|\mathbf{A}\| \bullet \|\mathbf{B}\|}$$

Correlation between two vectors!

Row and Column Vectors

(Relevant for Matlab)

Column vector (2x1 “matrix”): $\begin{bmatrix}1\\2\end{bmatrix}$

Row vector (1x2 “matrix”): $\begin{bmatrix}1 & 2\end{bmatrix}$

$\begin{bmatrix}1 & 2\end{bmatrix} * \begin{bmatrix}1\\2\end{bmatrix} = 1*1+2*2=5$

is the same as the scalar product

$\begin{bmatrix}1 & 2\end{bmatrix} * \begin{bmatrix}1 & 2\end{bmatrix}$ $\quad \begin{bmatrix}1\\2\end{bmatrix} * \begin{bmatrix}1\\2\end{bmatrix}$

don’t work (dimensions don’t agree)

$\begin{bmatrix}1\\2\end{bmatrix} * \begin{bmatrix}1 & 2\end{bmatrix} = \begin{bmatrix}1*1 & 1*2\\2*1 & 2*2\end{bmatrix} = \begin{bmatrix}1 & 2\\2 & 4\end{bmatrix}$

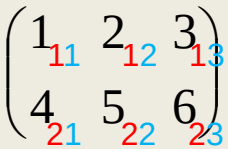
Is the “dyadic” product
(“tensor” product, $\mathbf{x} \otimes \mathbf{y}$)

$$\text{In Matlab}: \begin{bmatrix}1\\2\end{bmatrix}' = [1\ \ 2]$$

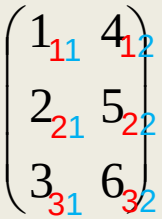
Example

Matrices

2 rows
3 columns



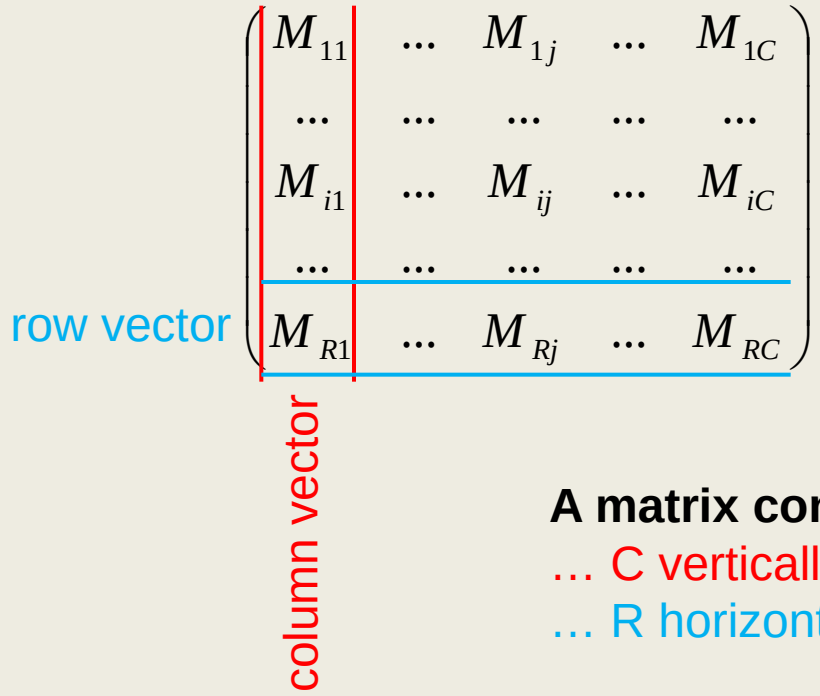
dimension 2x3



dimension (3x2)

3 rows
3 columns

Matrices Are “Stacked Vectors”



C: number of columns

R: number of rows

Matrix has dimension *R*x*C*

A matrix consists of…

… C vertically concatenated columns vectors

… R horizontally concatenated row vectors

A “square” matrix has the same number of rows and columns (*C*=*R*)

Matrix Transpose

$\mathbf{M} \rightarrow \mathbf{M}^T$ ($\mathbf{M} \rightarrow \mathbf{M}'$ in Matlab)

Rows of **M** become columns of **M**

Dimension changes from *R*x*C* to *C*x*R*

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

(2x3) *(3x2)*

$$\begin{pmatrix} M_{11} & \dots & M_{1j} & \dots & M_{1C} \\ \dots & \dots & \dots & \dots & \dots \\ M_{i1} & \dots & M_{ij} & \dots & M_{iC} \\ \dots & \dots & \dots & \dots & \dots \\ M_{R1} & \dots & M_{Rj} & \dots & M_{RC} \end{pmatrix} \rightarrow \begin{pmatrix} M_{11} & \dots & M_{i1} & \dots & M_{R1} \\ \dots & \dots & \dots & \dots & \dots \\ M_{1j} & \dots & M_{ij} & \dots & M_{Rj} \\ \dots & \dots & \dots & \dots & \dots \\ M_{1C} & \dots & M_{iC} & \dots & M_{RC} \end{pmatrix}$$

C: number of columns
R: number of rows

R: number of columns
C: number of rows

For a “symmetric” matrix: $\mathbf{M} = \mathbf{M}^T$

Matrix Addition/Subtraction

$$\mathbf{A}-\mathbf{B}=\begin{pmatrix}\dots & \dots & \dots\\ \dots & A_{ij} & \dots\\ \dots & \dots & \dots\end{pmatrix}-\begin{pmatrix}\dots & \dots & \dots\\ \dots & B_{ij} & \dots\\ \dots & \dots & \dots\end{pmatrix}=\begin{pmatrix}\dots & \dots & \dots\\ \dots & A_{ij}-B_{ij} & \dots\\ \dots & \dots & \dots\end{pmatrix}$$

A and **B** must have equal dimensions
The result has the same dimension as **A** and **B**

$$\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{pmatrix}-\begin{pmatrix}1 & 1 & 1\\ 2 & 2 & 2\end{pmatrix}=\begin{pmatrix}0 & 1 & 2\\ 2 & 3 & 4\end{pmatrix}$$

Basic applications: Subtract/Average data sets, contrasts etc.

Special Matrices

Identity Matrix

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Diagonal Matrix

$$\begin{pmatrix} a & 0 & 0 \\ 0 & c & 0 \\ 0 & 0 & b \end{pmatrix}$$

Upper Triagonal Matrix

$$\begin{pmatrix} a & d & f \\ 0 & c & e \\ 0 & 0 & b \end{pmatrix}$$

Symmetric Matrix

$$\begin{pmatrix} a & d & e \\ d & b & f \\ e & f & c \end{pmatrix}$$

Example

Multiplying Matrices with Vectors

Every element of **y** is the scalar product between a row of **M** and **x.**

$$\begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \end{pmatrix} * \begin{pmatrix} 3 \\ 4 \\ 5 \end{pmatrix} = \begin{pmatrix} 1*3+1*4+1*5 \\ 2*3+2*4+2*5 \end{pmatrix} = \begin{pmatrix} 12 \\ 24 \end{pmatrix}$$

M * **x** = **y**

Matrix*Vector or Vector*Matrix always yields a Vector.

Remember that earlier we multiplied row vectors with column vectors?

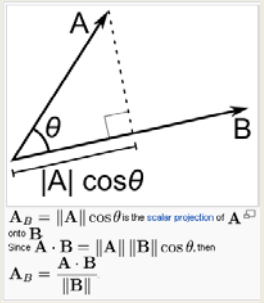
This makes sense now, because vectors are special cases of matrices.

Multiplying Matrices with Vectors

$$\mathbf{y} = \begin{pmatrix} y_1 \\ \dots \\ y_j \\ \dots \\ y_C \end{pmatrix} = \mathbf{M}\mathbf{x} = \begin{pmatrix} M_{11} & \dots & M_{1j} & \dots & M_{1C} \\ \dots & \dots & \dots & \dots & \dots \\ M_{i1} & \dots & M_{ij} & \dots & M_{iC} \\ \dots & \dots & \dots & \dots & \dots \\ M_{R1} & \dots & M_{Rj} & \dots & M_{RC} \end{pmatrix} \begin{pmatrix} x_1 \\ \dots \\ x_j \\ \dots \\ x_C \end{pmatrix} = \begin{pmatrix} \sum_{j=1}^{C} x_j * M_{1j} \\ \dots \\ \sum_{j=1}^{C} x_j * M_{ij} \\ \dots \\ \sum_{j=1}^{C} x_j * M_{Rj} \end{pmatrix} = \begin{pmatrix} \mathbf{M}_{1.} \bullet \mathbf{x} \\ \dots \\ \mathbf{M}_{i.} \bullet \mathbf{x} \\ \dots \\ \mathbf{M}_{R.} \bullet \mathbf{x} \end{pmatrix}$$

$\mathbf{M}_{i.}$ stands for the i - th row of $\mathbf{M}$.

Every element of **y** is the scalar product between a row of **M** and **x.**



$$\begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \end{pmatrix} * \begin{pmatrix} 3 \\ 4 \\ 5 \end{pmatrix} = \begin{pmatrix} 1*3+1*4+1*5 \\ 2*3+2*4+2*5 \end{pmatrix} = \begin{pmatrix} 12 \\ 24 \end{pmatrix}$$

M * x = y

Row and Column Vectors II

Column vector (2x1 “matrix”): $\begin{pmatrix} 1 \\ 2 \end{pmatrix}$

Row vector (1x2 “matrix”): $\begin{pmatrix} 1 & 2 \end{pmatrix}$

$$\begin{pmatrix} 1 & 2 \end{pmatrix} * \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 1*1+2*3 & 1*2+2*4 \end{pmatrix} = \begin{pmatrix} 7 & 10 \end{pmatrix}$$

matrix multiplication

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} * \begin{pmatrix} 1 & 2 \end{pmatrix}$$

doesn’t work (dimension don’t agree)

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 1 & 2 \end{pmatrix}^T = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 1*1+2*2 & 3*1+2*4 \end{pmatrix} = \begin{pmatrix} 5 \\ 11 \end{pmatrix}$$

uses transpose –
is different from above in values and dimensions

Multiplying Matrices with Vectors

matrix***vector** is not the same as **vector*****matrix**:

$$\begin{pmatrix} 3 \\ 4 \\ 5 \end{pmatrix} * \begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \end{pmatrix}$$ does not work!

$$\begin{pmatrix} 3 \\ 4 \\ 5 \end{pmatrix}^T * \begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \end{pmatrix} = \begin{pmatrix} 3 & 4 & 5 \end{pmatrix} * \begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \end{pmatrix}$$ does not work either!

Diagonal Matrices

Diagonal Matrix

$$\mathbf{D}*\mathbf{M}=\begin{pmatrix} D_{11} & \dots & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & D_{ii} & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & \dots & D_{CC} \end{pmatrix}\begin{pmatrix} M_{11} & \dots & M_{1j} & \dots & M_{1D} \\ \dots & \dots & \dots & \dots & \dots \\ M_{i1} & \dots & M_{ij} & \dots & M_{iD} \\ \dots & \dots & \dots & \dots & \dots \\ M_{R1} & \dots & M_{Rj} & \dots & M_{RD} \end{pmatrix}=\begin{pmatrix} D_{11}M_{11} & \dots & D_{11}M_{1j} & \dots & D_{11}M_{1D} \\ \dots & \dots & \dots & \dots & \dots \\ D_{ii}M_{i1} & \dots & D_{ii}M_{ij} & \dots & D_{ii}M_{iD} \\ \dots & \dots & \dots & \dots & \dots \\ D_{CC}M_{R1} & \dots & D_{CC}M_{Rj} & \dots & D_{CC}M_{RD} \end{pmatrix}$$

Identity Matrix

$$\mathbf{I}*\mathbf{M}=\begin{pmatrix} 1 & \dots & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & \dots & 1 \end{pmatrix}\begin{pmatrix} M_{11} & \dots & M_{1j} & \dots & M_{1D} \\ \dots & \dots & \dots & \dots & \dots \\ M_{i1} & \dots & M_{ij} & \dots & M_{iD} \\ \dots & \dots & \dots & \dots & \dots \\ M_{R1} & \dots & M_{Rj} & \dots & M_{RD} \end{pmatrix}=\mathbf{M}$$

Multiplying Matrices with Vectors

$$\mathbf{y} = \begin{pmatrix} y_1 \\ \dots \\ y_j \\ \dots \\ y_C \end{pmatrix} = \mathbf{Mx} = \begin{pmatrix} M_{11} & \dots & M_{1j} & \dots & M_{1C} \\ \dots & \dots & \dots & \dots & \dots \\ M_{i1} & \dots & M_{ij} & \dots & M_{iC} \\ \dots & \dots & \dots & \dots & \dots \\ M_{R1} & \dots & M_{Rj} & \dots & M_{RC} \end{pmatrix} \begin{pmatrix} x_1 \\ \dots \\ x_j \\ \dots \\ x_C \end{pmatrix} = \begin{pmatrix} \sum_{j=1}^{C} x_j * M_{1j} \\ \dots \\ \sum_{j=1}^{C} x_j * M_{ij} \\ \dots \\ \sum_{j=1}^{C} x_j * M_{Rj} \end{pmatrix} = \sum_{j=1}^{C} x_j * \begin{pmatrix} M_{1j} \\ \dots \\ M_{ij} \\ \dots \\ M_{Rj} \end{pmatrix} = \sum_{j=1}^{C} x_j \mathbf{M}_{.j}$$

$\mathbf{M}_{.j}$ stands for the j - th column of $\mathbf{M}$.

Each column of **M** is weighted by the corresponding element in **x**.

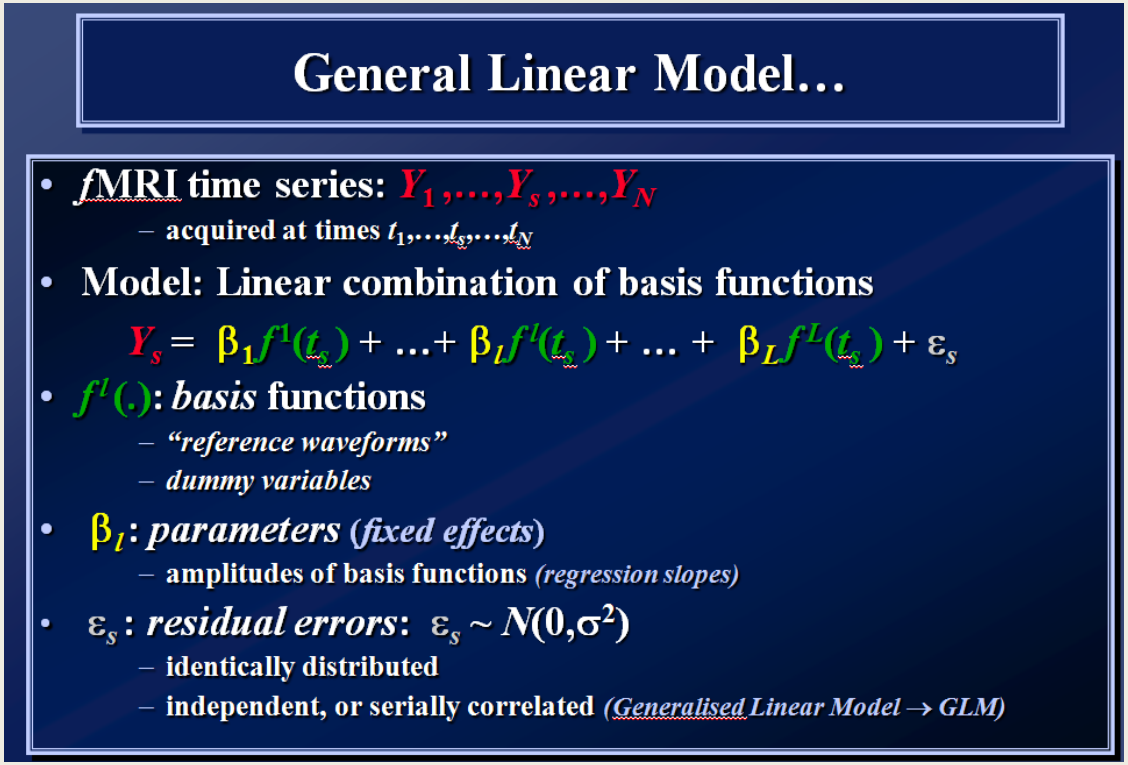
y is a “linear combination” of the columns of **M.**

$$\begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \end{pmatrix} * \begin{pmatrix} 3 \\ 4 \\ 5 \end{pmatrix} = 3 * \begin{pmatrix} 1 \\ 2 \end{pmatrix} + 4 * \begin{pmatrix} 1 \\ 2 \end{pmatrix} + 5 * \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 3 \\ 6 \end{pmatrix} + \begin{pmatrix} 4 \\ 8 \end{pmatrix} + \begin{pmatrix} 5 \\ 10 \end{pmatrix} = \begin{pmatrix} 12 \\ 24 \end{pmatrix}$$

Multiplying Matrices with Vectors

The famous GLM equation:

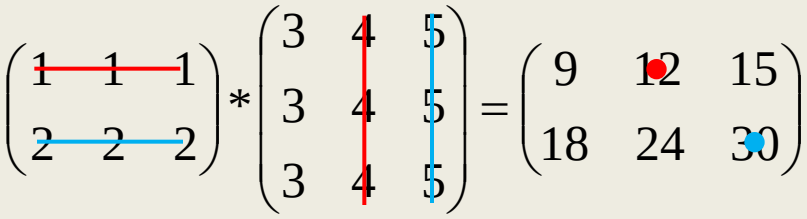
y = X*b



Example

Matrix Multiplication

Multiplication of Matrices with Matrices



M * **N** = **Y**

Every element *ij* of **Y** is the scalar product of the *i*-th row of **M** with the *j*-th column of **N**

M must have as many columns as **N** has rows.

Y has the number of rows of **M** and number of columns of **N** (i.e. dimension *R*x*C*).

Multiplying a Matrix with a Matrix always yields a Matrix.

Multiplication of Matrices with Matrices

$$\mathbf{Y} = \begin{pmatrix} Y_{11} & \dots & Y_{1j} & \dots & Y_{1C} \\ \dots & \dots & \dots & \dots & \dots \\ Y_{i1} & \dots & Y_{ij} & \dots & Y_{iC} \\ \dots & \dots & \dots & \dots & \dots \\ Y_{R1} & \dots & Y_{Rj} & \dots & Y_{RC} \end{pmatrix} = \mathbf{M} * \mathbf{N} = \begin{pmatrix} M_{11} & \dots & M_{1j} & \dots & M_{1D} \\ \dots & \dots & \dots & \dots & \dots \\ M_{i1} & \dots & M_{ij} & \dots & M_{iD} \\ \dots & \dots & \dots & \dots & \dots \\ M_{R1} & \dots & M_{Rj} & \dots & M_{RD} \end{pmatrix} \begin{pmatrix} N_{11} & \dots & N_{1j} & \dots & N_{1C} \\ \dots & \dots & \dots & \dots & \dots \\ N_{i1} & \dots & N_{ij} & \dots & N_{iC} \\ \dots & \dots & \dots & \dots & \dots \\ N_{D1} & \dots & N_{Dj} & \dots & N_{DC} \end{pmatrix}$$

M must have as many columns as **N** has rows.

Y has the number of rows of **M** and number of columns of **N** (i.e. dimension *R*x*C*).

Every element *ij* of **Y** is the scalar product of the *i*-th row of **M** with the *j*-th column of **N**

$$\begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \end{pmatrix} * \begin{pmatrix} 3 & 4 & 5 \\ 3 & 4 & 5 \\ 3 & 4 & 5 \end{pmatrix} = \begin{pmatrix} 9 & 12 & 15 \\ 18 & 24 & 30 \end{pmatrix}$$

M * **x** = **Y**

Multiplication of Matrices with Matrices

For matrix multiplication, some matrix dimensions must agree:
Number of columns of 1st, number of rows of 2nd, sometimes called “inner dimensions”)

$$\begin{pmatrix} 3 & 4 & 5 \\ 3 & 4 & 5 \\ 3 & 4 & 5 \end{pmatrix} * \begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \end{pmatrix}$$ doesn’t work.

For scalars, multiplication is commutative, i.e. *a***b* = *b***a*.
But for matrices it is not!

Even if it works, **A*****B** is usually not the same as **B*****A**:

$$\begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} * \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 0 \\ 0 & 0 \end{pmatrix}, \quad \text{but : } \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} * \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

Multiplication of Matrices with Matrices

EEG/MEG "inverse problem":

For a single time point, a solution can be obtained by multiplying the data vector with an inverse matrix:

j = G * d

dipoles = inverse * data

$$\begin{pmatrix} \hat{j}_1 \\ \hat{j}_2 \end{pmatrix} = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \end{pmatrix}$$

For multiple time points, data and sources turn into matrices, with one column per time point:

J = G * D

dipoles = inverse * data

$$\begin{pmatrix} \hat{j}_1 \ldots \\ \hat{j}_2 \ldots \end{pmatrix} = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} d_1 \ldots \\ d_2 \ldots \end{pmatrix}$$

Example

Exercise I

You've got a matrix **M**.

You want to compute the sum for each row.

How can you do this with matrix algebra?

$$\mathbf{M} = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix} * \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1*1+1*3+1*5 \\ 1*2+1*4+1*6 \end{pmatrix} = \begin{pmatrix} 9 \\ 12 \end{pmatrix}$$

How to compute the sum for each column?

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} * \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1*1+1*2 \\ 1*3+1*4 \\ 1*5+1*6 \end{pmatrix} = \begin{pmatrix} 3 \\ 7 \\ 11 \end{pmatrix}$$

How to compute the averages?

Note: The above is just for illustrating Matlab functionality. Matlab has commands sum() and mean() etc.

Exercise II

You've got a matrix **M** (e.g. EEG data).

You want to make sure the average of each column is zero ("average reference").

Possibility 1: Nested for-loops (complicated, inefficient and error-prone)

```
for cc = 1:nr_columns,     % column-by-column…
   MeanOfColumn = mean( M(:,cc) );   % compute mean of column
   for rr = 1:nr_rows,        % row-by-row
      M_new(rr,cc) = M(rr,cc) - MeanOfColumn;     % subtract mean from each element
   end;
end;
mean(M_new)     % check results
```

Exercise II: Average reference

You've got a matrix **M** (e.g. EEG data).

You want to make sure the average of each column is zero ("average reference").

Possibility 2: Use some matrix algebra

% create matrix "operator" that subtracts means from columns
MatrixOperator = eye(nr_rows) – ones(nr_rows)/nr_rows;

M_new = **MatrixOperator***M;

Mean(M_new) % check result

MatrixOperator **M**

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix} - \frac{1}{2}\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix} = \left[\underbrace{\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}}_{\text{eye()}} - \frac{1}{2}\underbrace{\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}}_{\text{ones()}}\right]\begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

The nice thing is: Any time a new matrix **M** comes along, you just need to multiply it with **MatrixOperator** (no additional for-loops required)

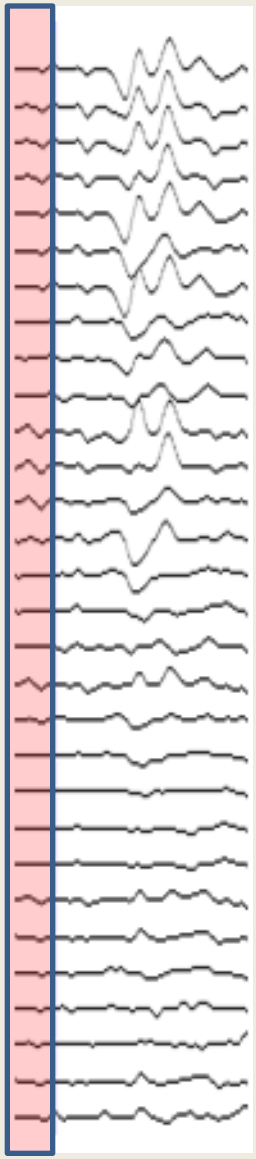
Exercise III: Baseline Correction

You've got a matrix **M** (e.g. EEG data).

The average of the "baseline interval" should be zero for each channel.

Each channel is one row of the matrix.

The baseline interval comprises the first 50 elements of the matrix.



Baseline interval

% compute mean baseline values for each channel
MeanBaseline = mean(M(:,1:50), 2);

% repeat vertically to fit the dimensions of **M**
RepMeanBaseline = repmat(MeanBaseline, 1, nr_columns);

% subtract baseline values from each element in **M**
M_new = M – RepMeanBaseline;

mean(M_new(:,1:50), 2) % check result

$$\text{repmat}\left(\begin{bmatrix}1\\2\end{bmatrix},1,3\right)=\begin{bmatrix}1&1&1\\2&2&2\end{bmatrix}$$

Exercise IV: Signal-to-Noise Ratio (SNR)

You've got a matrix **M** (e.g. EEG data).

You want to know the SNR at each sample in each channel.

SNR: Ratio of amplitude and standard deviation in baseline interval for each channel.

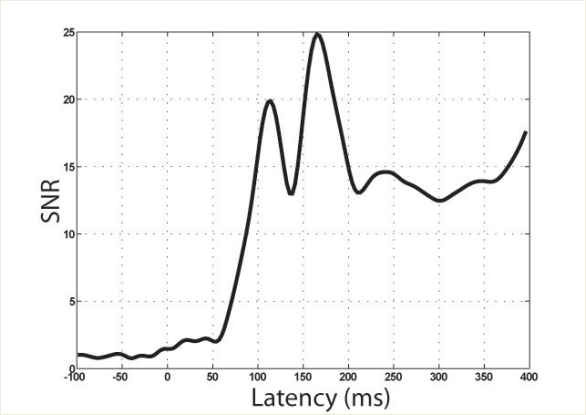
Each channel is one row of the matrix.
The baseline interval comprises the first 50 elements of the matrix.

% compute standard deviations of baseline intervals for each channel
StdBaseline = std(M(:,1:50), 0, 2);

% repeat vertically to fit the dimensions of **M**
RepStdBaseline = repmat(StdBaseline, 1, nr_columns);

% divide each element of **M** by baseline standard deviations
M_new = M ./ RepStdBaseline;

plot(M_new'); % check result



Reading and Writing Data in Matlab

There are convenient Matlab commands as long as you keep your data in Matlab format:

In order to save and load the complete workspace:

```
save <filename.mat>
load <filename.mat>
```

In order to save/load only specific variables:

```
save <filename.mat> <myvariable1> <myvariable2> …
load <filename.mat> <myvariable1> <myvariable2> …
```

Reading and Writing Data in Matlab I

Reading/writing matrices in text format is easy, too:

```
% writes mymatrix to filename.txt, elements separated by commas:
dlmwrite( filename, mymatrix, ', ' );

% …or separated by spaces:
dlmwrite( filename, mymatrix, ' ' );

% Read the data into Matlab:
newmatrix = dlmread( filename, ' ' );

% You can write data to Excel files (only Windows, not Linux):

% write my matrix to Excel worksheet 2:
xlswrite( filename, mymatrix, 2 );
```

Reading and Writing Data in Matlab II

Reading/Writing formatted text is a bit trickier:

You first need to initialise a file, and create a file identifier (FID)

```
MyFile_id = fopen('TestFile.txt', 'w');  % This file is for writing, hence ‘w’
```

Later, whenever you use MyFile_id, Matlab will read or write to ‘TestFile.txt’:

```
fprintf(MyFile_id, '%s : %d %f', 'Test', 1, 1.2);    % “format print file”
```

will write 'Test : 1 1.2' to the file 'TestFile.txt'

You can use '\n' for line breaks, '\t' for tabs, etc.

When you are done, close the file (it will not be deleted):

```
fclose(MyFile_id);
```

You can read formatted text using fscanf or textread:

```
[a,b,c,d] = textread('TestFile.txt', , '%s%c%d%f');
```

Enough for now?