Stima & Filtraggio: Lab 1

Giacomo Baggio

Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Padova

✉ baggio@dei.unipd.it

March 29, 2017

General info

Instructor: Giacomo Baggio

@ DEI-A, 3rd floor, office 330

`baggio@dei.unipd.it`

`baggio.dei.unipd.it/~teaching` (slides + `.m` code)

Lab dates: 29/03/17 h. 16–18: Intro to MATLAB® + Static Estimation

TBD (≈ mid April) h. 16–18: Kalman Filtering and Applications

TBD (≈ mid May) h. 16–18: Wiener Filtering and Applications

How can I get MATLAB®? UniPD Campus License!

More info @: `csia.unipd.it` > `servizi` > `servizi-utenti-istituzionali` > `contratti-software-e-licenze` > `matlab`

Today's Lab

Part I: MATLAB® crash course

⇓

Part II: Static estimation

Today's Lab

Part I: MATLAB® crash course (1 h)

Part II: Static estimation (30 min)

Quick survey

Are you familiar with MATLAB®?

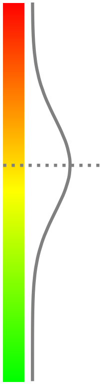
- *Mat...what!?*
- **Not so much**
- **More or less**
- **Yep**
- *I master it!*

Quick survey

Are you familiar with MATLAB®?

(my guess)

$\sim$ *1%*	🔴	***Mat...what!?***
$\sim$ *35%*	🟠	**Not so much**
$\sim$ *55%*	🟡	**More or less**
$\sim$ *9%*	🟢	**Yep**
$\sim$ *0%*	🟢	***I master it!***



• Part I •

MATLAB® crash course

basic commands & operations

functions & plots

some more advanced stuff...

• Part I •

MATLAB® crash course

basic commands & operations

functions & plots

some more advanced stuff...

Mat...what!?

MATLAB® stands for MATrix LABoratory and it's computing environment designed in the late 70s by Cleve Moler (C.S. prof @ UNM).

MATLAB® quickly became quite popular (especially among control theorist and practitioners) and used for both teaching and research. It was also *free*.

In the 80s an engineer, Jack Little, saw MATLAB® during a lecture by Moler at Stanford University. He rewrote MATLAB® in C and founded The MathWorks, Inc. to *market it*.

As a programming language MATLAB® ...

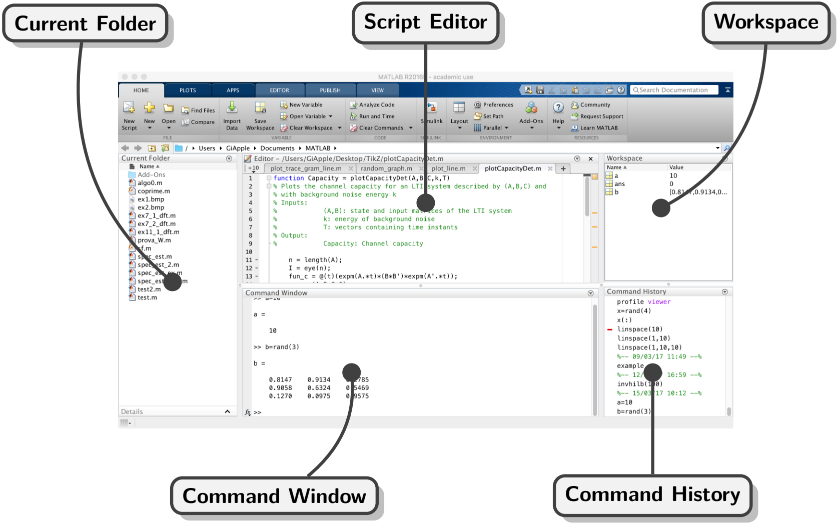
- ✓ can handle ***matrix*** and ***vector*** operation very easily (compare it with Python!)

- ✓ has an huge number of useful ***toolboxes*** (control system, identification, time series analysis, *etc.*)

- ✓ can do ***symbolic*** mathematics too!

- ✗ it's not open source! (Open source alternative: GNU Octave)
- ✗ it's not very computationally efficient

The interface (R2016b)



Defining variables

Integer:

```
>> iValue = 2
```

Boolean:

```
>> bValue = true
```

String:

```
>> strHello = 'hello world!'
```

Row vector:

```
>> rvX = [1 2 3 5]
```

Column vector:

```
>> cvX = [1 2 3 5]'
```

Matrix:

```
>> mX = [1 2 3 5; 8 13 21 34]
```

Defining variables

Integer:

```
>> iValue = 2;
```

Boolean:

```
>> bValue = true;
```

String:

```
>> strHello = 'hello world!';
```

Row vector:

```
>> rvX = [1 2 3 5];
```

Column vector:

```
>> cvX = [1 2 3 5]';
```

Matrix:

```
>> mX = [1 2 3 5; 8 13 21 34];
```

`;` = ***suppress "echo"***

Managing variables

Workspace variables list:

`>> who`

Workspace variables info:

`>> whos`

Save workspace in `data.mat`:

`>> save data`

Save `iValue` in `data.mat`:

`>> save data iValue`

Load `data.mat`:

`>> load data`

Load `iValue` in `data.mat`:

`>> load data iValue`

Clear workspace:

`>> clear all`

Clear `iValue`:

`>> clear iValue`

Clear command window:

`>> clc`

Move cursor to the top:

`>> home`

Logical operations & building blocks

== (equality)	~ (negation)
&& (AND)	|| (OR)

Logical operations & building blocks

== (equality)

∼ (negation)

&& (AND)

|| (OR)

while loop

```
while iValue == 0
   ...
end
```

if/else statement

```
if iValue ≤0
   ...
else
   ...
end
```

for loop

```
for iValue = 1:10
   ...
end
```

Vector operations

```
>> rvX = [1 2 3 5]
```

Size:	`>> length(rvX)`
(Conjugate) transpose:	`>> rvX'`
Summing entries:	`>> sum(rvX)`
Multiplying entries:	`>> prod(rvX)`
Flipping entries:	`>> fliplr(rvX)` [row vec] `>> flipud(rvX')` [col vec]
Extract entries:	`>> rvX(2)` `>> rvX(1:3)`
Find entries satisfying condition:	`>> find(rvX == 5)`

Matrix operations

```
>> mX = [1 2; 3 5]
```

Dimension (#rows, #columns):	`>> length(mX)`
(Conjugate) transpose:	`>> mX'`
Eigenvalues:	`>> eig(mX)`
Inverse:	`>> inv(mX)`
Extract entries:	`>> mX(1,1)` [single] `>> mX(1,:)` [row] `>> mX(:,1)` [col]
Product:	`>> mX*mX`
Entrywise product:	`>> mX.*mX`

Polynomials

$p(x) = x^2 + 2x + 1 \quad \xrightarrow{\texttt{.m}} \quad$ `>> rvP = [1 2 1]`

polynomial $\quad \xrightarrow{\texttt{.m}} \quad$ vector of coefficients

Polynomials

N.B.

$q(x) = 3x^2 + 2x \quad \xrightarrow{\texttt{.m}} \quad$ `>> rvQ = [3 2 0]`

constant term!

Polynomials

$p(x) = x^2 + 2x + 1 \xrightarrow{.m}$ `>> rvP = [1 2 1]`

$q(x) = 3x^2 + 2x \xrightarrow{.m}$ `>> rvQ = [3 2 0]`

Evaluate $p(x)$ at $x = 3$:	`>> polyval(rvP,3)`
Roots of $p(x)$:	`>> roots(rvP)`
Product $p(x)q(x)$:	`>> conv(rvP,rvQ)`
Quotient (+rem) $p(x)/q(x)$:	`>> deconv(rvP,rvQ)`
Create $g(x)$ with roots in $3 \pm 2i$:	`>> r = [3+2*1i,3-2*1i]` `>> g = poly(r)`

How to create and run a script

Script = sequence of instructions

In MATLAB® → `.m` extension

Create it!

1. Highlight commands from the Command History, right-click, and select `Create Script`
2. Click the `New Script` button on the Home tab
3. Use the `edit` function
   `>> edit new_file_name`

Run it!

1. Type the script name on the command line and press `Enter`
   `>> new_file_name`
2. Click the `Run` ▶ button on the Editor tab
3. Use a shortcut (e.g. `F5`)

Help me please!

`>> help something`

↓

display help for the function/package `something`

Example:

```
>> help sin
 sin    Sine of argument in radians.
 sin(X) is the sine of the elements of X.

 See also asin, sind.
```

Help me please!

```
>> helpwin something
```

↓

display detailed documentation for the function/package `something` in the Help browser

✎ *Practice time 1!*

Ex 1.1. Create a 10-dim row vector of all 1's and then put to zero its last 3 entries.

$$[1\,1\,1\,1\,1\,1\,1\,1\,1\,1] \to [1\,1\,1\,1\,1\,1\,1\,0\,0\,0]$$

Ex 1.2. Create a 4×4 (uniformly) *random* matrix with entries in $[0, 1]$ and then flip the elements on its *diagonal*.

[Hint: Use built-in functions `rand` and `diag`]

$$\begin{bmatrix} 0.39 & 0.62 & 0.12 & 0.62 \\ 0.21 & 0.74 & 0.58 & 0.77 \\ 0.26 & 0.28 & 0.20 & 0.14 \\ 0.22 & 0.98 & 0.81 & 0.83 \end{bmatrix} \to \begin{bmatrix} 0.83 & 0.62 & 0.12 & 0.62 \\ 0.21 & 0.20 & 0.58 & 0.77 \\ 0.26 & 0.28 & 0.74 & 0.14 \\ 0.22 & 0.98 & 0.81 & 0.39 \end{bmatrix}$$

Ex 1.3. Create a polynomial $p(x)$ with roots in $\left\{-\frac{1}{3}, \frac{3}{4} \pm \mathrm{i}\right\}$. Compute the product $g(x) := p(x)q(x)$, with $q(x) := x^2 - \frac{1}{2}x$. Is $g(x)$ Schur stable?

• Part I •

MATLAB® crash course

basic commands & operations

functions & plots

some more advanced stuff...

How to define a function

```
function [dM,dS] = statVec(rvX)
% STATVEC Returns the mean and standard ...
    deviation of an input vector
% Input:
%               rvX: input vector
% Output:
%               dM: mean
%               dS: standard deviation

    iN = length(rvX);
    dM = sum(rvX)/iN;
    dS = sqrt(sum((rvX-dM).^2/iN));

end
```

How to define a function

outputs

`statVec.m`

inputs

```
function [dM,dS] = statVec(rvX)
% STATVEC Returns the mean and standard ...
    deviation of an input vector
% Input:
%                rvX: input vector
% Output:
%                dM: mean
%                dS: standard deviation

    iN = length(rvX);
    dM = sum(rvX)/iN;
    dS = sqrt(sum((rvX-dM).^2/iN));

end
```

Plotting

```
>> plot(rvX,rvY)
```

Example: Plotting sine in the interval $[0, 2\pi]$

Plotting

```
>> plot(rvX,rvY)
```

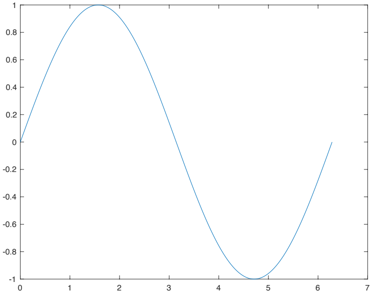
Example: Plotting sine in the interval $[0, 2\pi]$

```
dT = 0.001;            % Sampling period
rvX = 0:dT:2*pi;       % X vector
rvY = sin(rvX);        % Y vector
plot(rvX,rvY);
```

Plotting

```
>> plot(rvX,rvY)
```

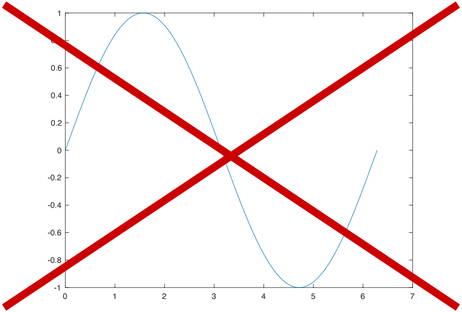
Example: Plotting sine in the interval $[0, 2\pi]$



Plotting

```
>> plot(rvX,rvY)
```

Example: Plotting sine in the interval $[0, 2\pi]$



Nice plotting

```
>> plot(rvX,rvY)
```

Example: *Nice* plotting sine

```
dT = 0.001;           % Sampling period
rvX = 0:dT:2*pi;      % X vector
rvY = sin(rvX);       % Y vector
plot(rvX,rvY, ...
     'LineStyle', '-',...
     'LineWidth', 2.5,...
     'Color', [0 0 0]);
```

Nice plotting

```
>> plot(rvX,rvY)
```

Example: *Nice* plotting sine

```
ax = gca;                   % get the current axes
ax.FontUnits = 'points';
ax.FontSize = 22;
ax.Title.Interpreter = 'latex';
ax.Title.String = '$f(t) = \sin(t)$';
ax.XLabel.Interpreter = 'latex';
ax.XLabel.String = '$t$ [sec]';
ax.YLabel.Interpreter = 'latex';
ax.YLabel.String = '$f(t)$ [Volt]';
```

Nice plotting

```
>> plot(rvX,rvY)
```

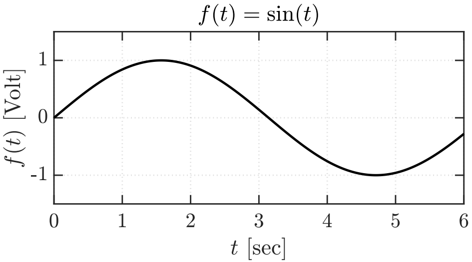
Example: *Nice* plotting sine

```
ax.XLim = [0 6];
ax.YLim = [-1.5 1.5];
ax.XGrid = 'on';
ax.YGrid = 'on';
ax.GridLineStyle = ':';
ax.TickLabelInterpreter = 'latex';
ax.TickLength = [0.02 0.02];
ax.LineWidth = 1.5;
ax.TickDir = 'in';
```

Nice plotting

```
>> plot(rvX,rvY)
```

Example: *Nice* plotting sine



Multiple plots

```
>> plot(rvX1,rvY1)
>> hold on
>> plot(rvX2,rvY2)
```

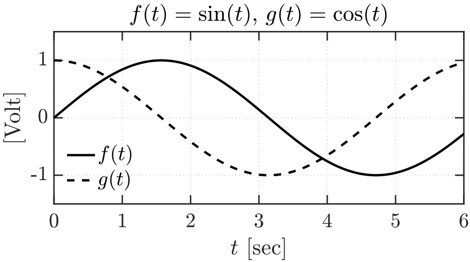
Setting legend

```
legend({'$f(t)$','$g(t)$'});
ax.Legend.Interpreter = 'latex';
ax.Legend.FontSize = 22;
ax.Legend.Location = 'southwest';
ax.Legend.Orientation = 'vertical';
ax.Legend.Box = 'off';
```

Multiple plots

```
>> plot(rvX1,rvY1)
>> hold on
>> plot(rvX2,rvY2)
```

Setting legend



✎ *Practice time 2!*

Ex 2.1. Create a function `rvY = zeroTail(rvX)` which has as input an n-dim ($n \geq 3$) vector `rvX` and as output a vector `rvY` equal to `rvX` except for its last 3 entries which are set to 0.

Ex 2.2. Create a function `mY = flipDiag(mX)` which has as input an $n \times n$ matrix `mX` and returns a matrix `mY` equal to `mX` but with a flipped diagonal.

Ex 2.3. Create a function `bT = testSchur(rvP,rvQ)` which has as inputs two arbitrary polynomials `rvP` and `rvQ`. This function:
i) plots the product of the two polynomials in the interval $[-10, 10]$,
ii) returns boolean `true` if the latter product is Schur stable and boolean `false` otherwise.

• Part I •

MATLAB® crash course

basic commands & operations

functions & plots

some more advanced stuff...

Commenting / Documenting

```
% This is a comment
```

```
%{ ... This is a comment block ... %}
```

When documenting a function it is a good habit to ***reference*** other nested functions as:

```
function [dM,dS] = statVec(rvX)
% STATVEC ...
% ...
% SEE ALSO
% MEANVEC, STDVEC
```

Their ***hyperlinks*** will appear together with the function description when typing

```
>> help statVec
```

Sectioning

```
%% This creates a new section
```

Sections are parts of a script that you can run ***independently*** from the whole script (button `Run Section` or shortcut `Ctrl+Enter`)

```
%% Section 1
...
some code
...
%% Section 2
...
some other code
...
```

Debugging

Debugging = locating and fixing program errors!

$$\text{In MATLAB}^{®}$$

Graphical

Programmatic

Debugging

Debugging = locating and fixing program errors!

In MATLAB®

$$\underbrace{\hspace{6em}}$$

Graphical   Programmatic

Set breakpoint = pause the execution of the program so you can examine the value or variables where you think a problem could be.

Debugging

Debugging = locating and fixing program errors!

In MATLAB®

Graphical

Programmatic

Set breakpoint

click the breakpoint alley (–) at an executable line where you want to set the breakpoint.

$- \longmapsto$ ●

file name

line

```
>> dbstop in test.m at 3
```

```
>> dbstop error
```

automatically puts you in debug mode stopped at the line that triggers an error

Debugging

Debugging = locating and fixing program errors!

In MATLAB®

Graphical

Programmatic

Resume and step through file = resume the execution of the code after a breakpoint. It can be done until completion or step-by-step.

Debugging

Debugging = locating and fixing program errors!

In MATLAB®

Graphical

Programmatic

Resume and step

use the `continue` button ▶▶ or `step` ↪ button.

```
>> dbcont
>> dbstep
```

Debugging

Debugging = locating and fixing program errors!

In MATLAB®

Graphical

Programmatic

Quit debugging = exit debug mode.

Debugging

Debugging = locating and fixing program errors!

In MATLAB®

Graphical | Programmatic

Quit

use the `quit debugging` button

`>> dbquit`

Improving code

- ***optimizing memory access***
  - preallocate arrays before accessing them within loops
  - avoid creating unnecessary variables

- ***vectorizing loops***

```
>> rvX = 0:0.01:10;
>> rvY = sin(rvX);
```

- ***inspecting performances*** `tic toc`, `profile`

Some useful tricks

- Extract elements of a vector from `iN` to the end: `rvX(iN:end)`
- Display string to video: `disp('Hello world!')`
- Vectorizing a matrix: `cvX = mX(:)`
- Create a multi-dim array: `cArrX = cell(iN1,iN2,...,iNp)`
- Quickly define functions: `@(x) 3*x.2 + 2*x + 7`
- Define symbolic variables: `syms x1 x2`

• **Part II** •

Static Estimation

Recap

Hands-on

• Part II •

Static Estimation

Recap

Hands-on

Quick recap

$$\mathbf{x} \sim \mathcal{N}(\mu_{\mathbf{x}}, \Sigma_{\mathbf{x}})$$

$$\mathbf{y} \sim \mathcal{N}(\mu_{\mathbf{y}}, \Sigma_{\mathbf{y}})$$

$$\mathbf{z} := \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_{\mathbf{x}} \\ \mu_{\mathbf{y}} \end{bmatrix}, \begin{bmatrix} \Sigma_{\mathbf{x}} & \Sigma_{\mathbf{xy}} \\ \Sigma_{\mathbf{xy}}^{\top} & \Sigma_{\mathbf{y}} \end{bmatrix} \right)$$

MAP estimate

$$\boxed{\mathbb{E}[\mathbf{x} \mid \mathbf{y}] = \mu_{\mathbf{x}} + \Sigma_{\mathbf{xy}} \Sigma_{\mathbf{y}}^{-1} (\mathbf{y} - \mu_{\mathbf{y}})}$$

$$\boxed{\mathrm{Var}[\mathbf{x} \mid \mathbf{y}] = \Sigma_{\mathbf{x}} + \Sigma_{\mathbf{xy}} \Sigma_{\mathbf{y}}^{-1} \Sigma_{\mathbf{xy}}^{\top}}$$

Quick recap

$$\mathbf{x} \sim \cancel{\mathcal{N}}(\mu_{\mathbf{x}}, \Sigma_{\mathbf{x}})$$

$$\mathbf{y} \sim \cancel{\mathcal{N}}(\mu_{\mathbf{y}}, \Sigma_{\mathbf{y}})$$

$$\mathbf{z} := \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} \sim \cancel{\mathcal{N}}\left(\begin{bmatrix} \mu_{\mathbf{x}} \\ \mu_{\mathbf{y}} \end{bmatrix}, \begin{bmatrix} \Sigma_{\mathbf{x}} & \Sigma_{\mathbf{xy}} \\ \Sigma_{\mathbf{xy}}^\top & \Sigma_{\mathbf{y}} \end{bmatrix} \right)$$

best linear MMSE estimate

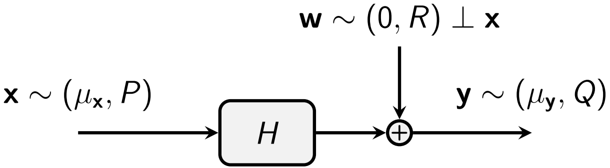
$$\boxed{\hat{\mathbb{E}}[\mathbf{x} \mid \mathbf{y}] = \mu_{\mathbf{x}} + \Sigma_{\mathbf{xy}} \Sigma_{\mathbf{y}}^{-1} (\mathbf{y} - \mu_{\mathbf{y}})}$$

$$\boxed{\mathrm{Var}[\tilde{\mathbf{x}}] = \Sigma_{\mathbf{x}} + \Sigma_{\mathbf{xy}} \Sigma_{\mathbf{y}}^{-1} \Sigma_{\mathbf{xy}}^\top}$$

$$\tilde{\mathbf{x}} := \mathbf{x} - \hat{\mathbb{E}}[\mathbf{x} \mid \mathbf{y}]$$

Quick recap

linear model



best linear MMSE estimate

$$\boxed{\hat{\mathbb{E}}[\mathbf{x} \mid \mathbf{y}] = \mu_{\mathbf{x}} + (P^{-1} + H^\top R^{-1} H)^{-1} H^\top R^{-1} (\mathbf{y} - \mu_{\mathbf{y}})}$$

$$\boxed{\mathrm{Var}[\tilde{\mathbf{x}}] = (P^{-1} + H^\top R^{-1} H)^{-1}}$$

$$\tilde{\mathbf{x}} := \mathbf{x} - \hat{\mathbb{E}}[\mathbf{x} \mid \mathbf{y}]$$

• Part II •

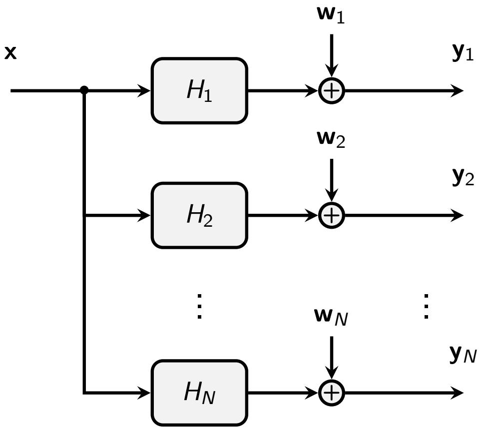
Static Estimation

- Recap
- Hands-on

Combining estimators (a.k.a. *sensor fusion*)

$$\mathbf{w}_i \sim (0, R_i) \perp \mathbf{x} \perp \mathbf{w}_j,\ i \neq j$$

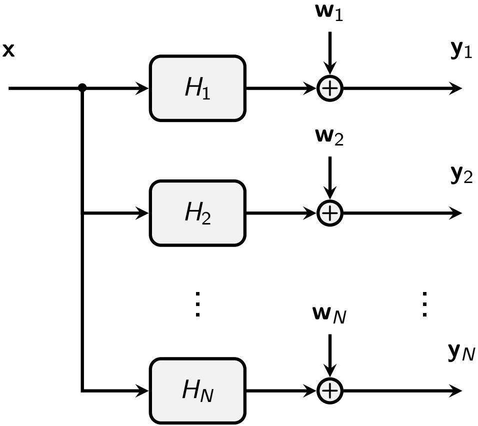
$\mathbf{x} \sim (0, P)$ $\qquad\qquad$ $\mathbf{y}_i \sim (0, Q_i)$



Combining estimators (a.k.a. *sensor fusion*)

$$\mathbf{w}_i \sim (0, R_i) \perp \mathbf{x} \perp \mathbf{w}_j,\ i \neq j$$

$\mathbf{x} \sim (0, P)$ $\mathbf{y}_i \sim (0, Q_i)$



$\hat{\mathbf{x}}_1 := \hat{\mathbb{E}}[\mathbf{x} \mid \mathbf{y}_1]$

$\hat{\mathbf{x}}_2 := \hat{\mathbb{E}}[\mathbf{x} \mid \mathbf{y}_2]$

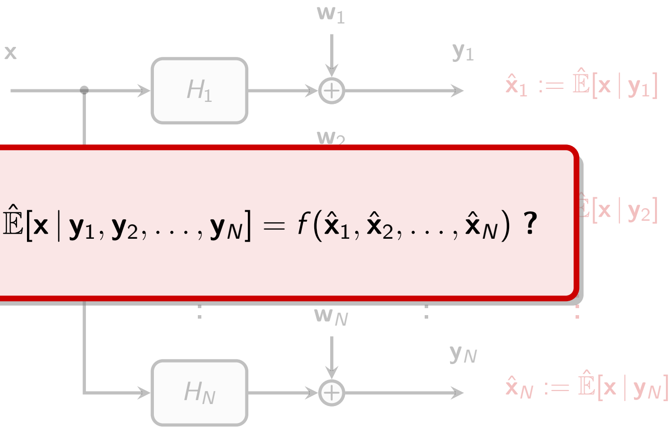
$\vdots$

$\hat{\mathbf{x}}_N := \hat{\mathbb{E}}[\mathbf{x} \mid \mathbf{y}_N]$

Combining estimators (a.k.a. *sensor fusion*)

$$\mathbf{w}_i \sim (0, R_i) \perp \mathbf{x} \perp \mathbf{w}_j,\ i \neq j$$

$\mathbf{x} \sim (0, P)$ $\mathbf{y}_i \sim (0, Q_i)$



$$\hat{\mathbf{x}} := \hat{\mathbb{E}}[\mathbf{x} \mid \mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_N] = f(\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, \dots, \hat{\mathbf{x}}_N)\ ?$$

Combining estimators (a.k.a. *sensor fusion*)

$$\hat{\mathbf{w}} := [\mathbf{w}_1, \dots, \mathbf{w}_N]^\top \sim (0, R), \qquad R = \mathrm{diag}(R_1, \dots, R_N)$$

$$H := \left[H_1^\top, \dots, H_N^\top\right]^\top$$

Combining estimators (a.k.a. *sensor fusion*)

$$\hat{\mathbf{w}} := [\mathbf{w}_1, \dots, \mathbf{w}_N]^\top \sim (0, R), \qquad R = \operatorname{diag}(R_1, \dots, R_N)$$

$$H := \left[H_1^\top, \dots, H_N^\top\right]^\top$$

$$\hat{\mathbf{x}} = (P^{-1} + H^\top R^{-1} H)^{-1} H^\top R^{-1} \mathbf{y}$$

$$(P^{-1} + H^\top R^{-1} H)\hat{\mathbf{x}} = H^\top R^{-1} \mathbf{y} \qquad \textit{(rearrange)}$$

Combining estimators (a.k.a. *sensor fusion*)

$$\hat{\mathbf{w}} := [\mathbf{w}_1, \ldots, \mathbf{w}_N]^\top \sim (0, R), \qquad R = \mathrm{diag}(R_1, \ldots, R_N)$$

$$H := \left[H_1^\top, \ldots, H_N^\top\right]^\top$$

$$\hat{\mathbf{x}} = (P^{-1} + H^\top R^{-1} H)^{-1} H^\top R^{-1} \mathbf{y}$$

$$(P^{-1} + H^\top R^{-1} H)\hat{\mathbf{x}} = H^\top R^{-1} \mathbf{y} \qquad \textit{(rearrange)}$$

$$\left(P^{-1} + \sum_{i=1}^{N} H_i^\top R_i^{-1} H_i\right) \hat{\mathbf{x}} = \sum_{i=1}^{N} H_i^\top R_i^{-1} \mathbf{y}_i \qquad \textit{(reduce)}$$

Combining estimators (a.k.a. *sensor fusion*)

$$\hat{\mathbf{w}} := [\mathbf{w}_1, \dots, \mathbf{w}_N]^\top \sim (0, R), \qquad R = \operatorname{diag}(R_1, \dots, R_N)$$

$$H := \left[H_1^\top, \dots, H_N^\top\right]^\top$$

$$\hat{\mathbf{x}} = (P^{-1} + H^\top R^{-1} H)^{-1} H^\top R^{-1} \mathbf{y}$$

$$(P^{-1} + H^\top R^{-1} H)\hat{\mathbf{x}} = H^\top R^{-1} \mathbf{y} \qquad \textit{(rearrange)}$$

$$\left(P^{-1} + \sum_{i=1}^N H_i^\top R_i^{-1} H_i\right)\hat{\mathbf{x}} = \sum_{i=1}^N H_i^\top R_i^{-1} \mathbf{y}_i \qquad \textit{(reduce)}$$

$$\left(P^{-1} + \sum_{i=1}^N H_i^\top R_i^{-1} H_i\right)\hat{\mathbf{x}} = \sum_{i=1}^N (P^{-1} + H_i^\top R_i^{-1} H_i)\hat{\mathbf{x}}_i \qquad \textit{(replace)}$$

Combining estimators (a.k.a. *sensor fusion*)

$$\hat{\mathbf{w}} := [\mathbf{w}_1, \ldots, \mathbf{w}_N]^\top \sim (0, R), \qquad R = \operatorname{diag}(R_1, \ldots, R_N)$$

$$H := \left[H_1^\top, \ldots, H_N^\top\right]^\top$$

$$\hat{\mathbf{x}} = (P^{-1} + H^\top R^{-1} H)^{-1} H^\top R^{-1} \mathbf{y}$$

$$(P^{-1} + H^\top R^{-1} H)\hat{\mathbf{x}} = H^\top R^{-1} \mathbf{y} \qquad \textit{(rearrange)}$$

$$\left(P^{-1} + \sum_{i=1}^{N} H_i^\top R_i^{-1} H_i\right)\hat{\mathbf{x}} = \sum_{i=1}^{N} H_i^\top R_i^{-1} \mathbf{y}_i \qquad \textit{(reduce)}$$

$$\left(P^{-1} + \sum_{i=1}^{N} H_i^\top R_i^{-1} H_i\right)\hat{\mathbf{x}} = \sum_{i=1}^{N} (P^{-1} + H_i^\top R_i^{-1} H_i)\hat{\mathbf{x}}_i \qquad \textit{(replace)}$$

$$\boxed{\hat{\mathbf{x}} = \left(P^{-1} + \sum_{i=1}^{N} H_i^\top R_i^{-1} H_i\right)^{-1} \sum_{i=1}^{N} (P^{-1} + H_i^\top R_i^{-1} H_i)\hat{\mathbf{x}}_i}$$

Combining estimators (a.k.a. *sensor fusion*)

$$\hat{\mathbf{w}} := [\mathbf{w}_1, \ldots, \mathbf{w}_N]^\top \sim (0, R), \qquad R = \mathrm{diag}(R_1, \ldots, R_N)$$

$$H := \left[H_1^\top, \ldots, H_N^\top\right]^\top$$

$$\hat{\mathbf{x}} = (P^{-1} + H^\top R^{-1} H)^{-1} H^\top R^{-1} \mathbf{y}$$

$$(P^{-1} + H^\top R^{-1} H)\hat{\mathbf{x}} = H^\top R^{-1} \mathbf{y} \qquad \textit{(rearrange)}$$

$$\left(P^{-1} + \sum_{i=1}^N H_i^\top R_i^{-1} H_i\right)\hat{\mathbf{x}} = \sum_{i=1}^N H_i^\top R_i^{-1} \mathbf{y}_i \qquad \textit{(reduce)}$$

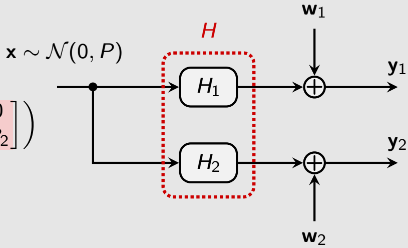
$$\left(P^{-1} + \sum_{i=1}^N H_i^\top R_i^{-1} H_i\right)\hat{\mathbf{x}} = \sum_{i=1}^N (P^{-1} + H_i^\top R_i^{-1} H_i)\hat{\mathbf{x}}_i \qquad \textit{(replace)}$$

$$\boxed{\hat{\mathbf{x}} = \mathrm{Var}[\tilde{\mathbf{x}}] \sum_{i=1}^N \mathrm{Var}[\tilde{\mathbf{x}}_i]^{-1} \hat{\mathbf{x}}_i}$$

✎ *Practice time 3!*

$$\mathbf{w} := \begin{bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \overset{R}{\begin{bmatrix} R_1 & 0 \\ 0 & R_2 \end{bmatrix}} \right)$$

$$H := \begin{bmatrix} H_1 \\ H_2 \end{bmatrix}, \quad \mathbf{y} := \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{bmatrix}$$



Ex 3.1. With reference to the above block diagram:

i) Create two (random) 500×2 measurement matrices `mH1` $:= H_1$, `mH2` $:= H_2$ and stack them in `mH` $:= H$.

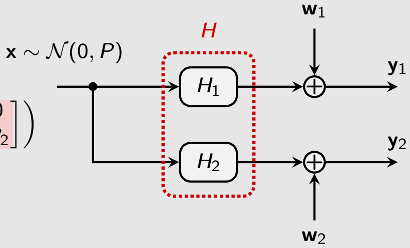
ii) Create a (random) 2×2 covariance matrix `mP` $:= P$, and two (random) 500×500 covariance matrices `mR1` $:= R_1$, `mR2` $:= R_2$. Use the latter matrices to build the matrix `mR` $:= R$.

iii) Generate a realization of the measurement vectors `cvY1` $:= \mathbf{y}_1$,`cvY2` $:= \mathbf{y}_2$ and stack them in `cvY` $:= \mathbf{y}$.

✎ *Practice time 3!*

$$\mathbf{w} := \begin{bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \overset{R}{\begin{bmatrix} R_1 & 0 \\ 0 & R_2 \end{bmatrix}} \right)$$

$$H := \begin{bmatrix} H_1 \\ H_2 \end{bmatrix}, \quad \mathbf{y} := \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{bmatrix}$$



Ex 3.2. Create the function

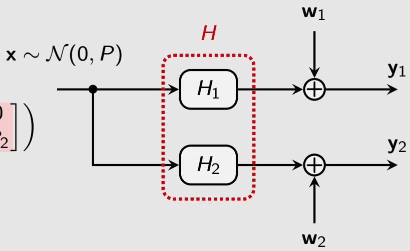
```
[cvE,mV] = centralMMSE(cvY,mP,mR,mH)
```

which has as input the generated realization `cvY`, the a priori covariance `mP`, the noise covariance `mR` and the measurement matrix `mH`, and returns the best linear MMSE estimate `cvE` and the variance of the estimation error `mV` using the "standard" formula.

✎ Practice time 3!

$$\mathbf{w} := \begin{bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \overbrace{\begin{bmatrix} R_1 & 0 \\ 0 & R_2 \end{bmatrix}}^{R} \right)$$

$$H := \begin{bmatrix} H_1 \\ H_2 \end{bmatrix}, \quad \mathbf{y} := \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{bmatrix}$$



Ex 3.3. Create the function

```
[cvE,mV] = distribMMSE(cvY1,cvY2,mP,mR1,mR2,mH1,mH2)
```

which as as inputs the single-system measurement vectors, noise covariance matrices and measurement matrices, and returns the best linear MMSE estimate `cvE` and the variance of the estimation error `mV` using the "distributed" formula.

Extra question: What is the more efficient solution?