MATLAB
University of Puerto Rico

Domingo Rodríguez & Juan Valera

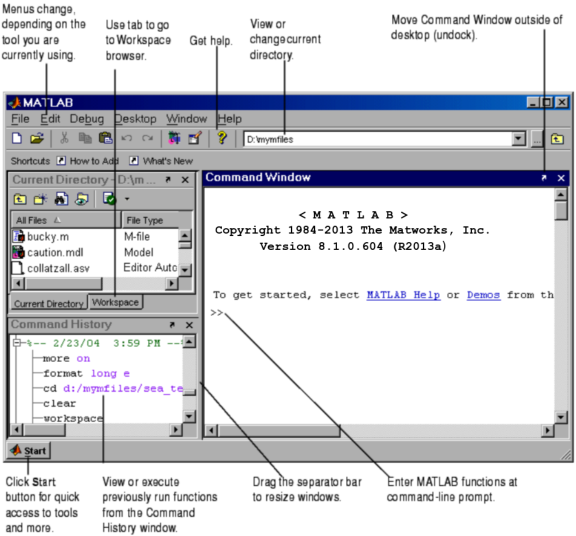
September 9, 2015

Outline

MATLAB Technical Language

AIPLaboratory at R&D Center

MATLAB - Main Screen User Interface



AIPLaboratory at R&D Center

Making Folders

- Use folders to keep your programs organized
- To make a new folder, click the Browse button next to Current Directory
- Click the Make New Folder button, and change the name of the folder. **Do NOT use spaces** in folder names. In the MATLAB folder, make two new folders: MATLAB
- Highlight the folder you just made and click OK
- The current directory is now the folder you just created
- To see programs outside the current directory, they should be in the Path. Choose menu option **File** and select the sub-option **Set Path** to add folders to the path

MATLAB Basics

- MATLAB can be considered a powerful graphics generator
- MATLAB is a programming language
  1. MATLAB is an interpreted language, like Java
  2. Commands can be executed line by line or using batch file called "Scripts"
  3. The extension of the "Scripts" of MATLAB is .m
  4. MATLAB has a "built-in editor" for creating or modifying "Scripts"

How get Help

- MATLAB has a commands called "help","doc", and "lookfor"
- To get info on how to use a function:
  1. `>> help sin`
  2. Help lists related functions at the bottom and links to the doc
- To get a nicer version of help with examples and easy-to- read descriptions:
  1. `>> doc sin`
- To search for a function by specifying keywords:
  1. `>> doc` *function_name*

Scripts

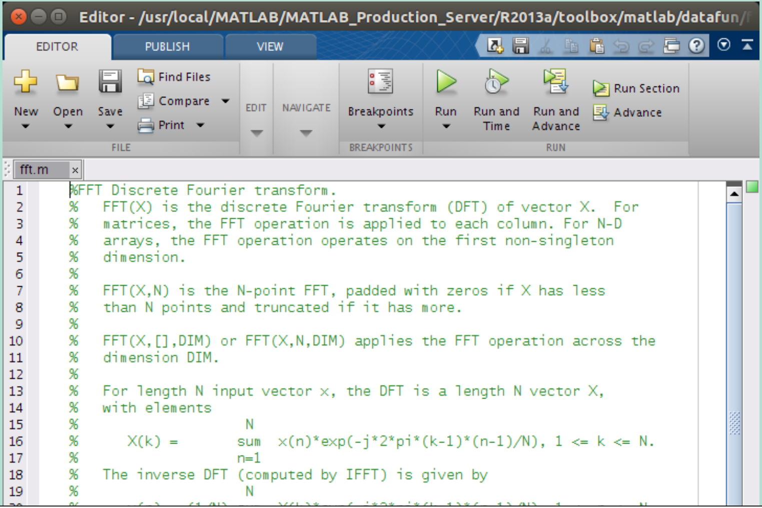
1. Scrips are:
   - collection of commands executed in sequence
   - written in the MATLAB editor
   - saved as MATLAB files (.m extension)
2. To create an MATLAB file from command-line
   - `>> edit myScript.m`

Scripts: Miscellaneous

1. Comment
   - Anything following a % is seen as a comment
   - The first contiguous comment becomes the script's help file
   - Comment thoroughly to avoid wasting time later
2. All variables created and modified in a script exist in the workspace even after it has stopped running

Scripts: The editor

Example



save/clear/load

1. Use `save` to save variables to a file:
   - `>> save myFile a b`
   - saves variables a and b to the file myfile.mat
   - myfile.mat file is saved in the current directory
   - Default working directory is `MATLAB`
2. Use `clear` to remove variables from environment
   - `>> clear a b`
   - look at workspace, the variables a and b are gone
3. Use `load` to load variable bindings into the environment
   - `>> load myFile`
   - look at workspace, the variables a and b are back
4. Can do the same for entire environment:
   - `>> save myenv; clear all; load myenv;`

MATLAB Technical Language

Scalar numbers

1. A variable can be given a value explicitly:
   - `>> a = 10` (shows up in workspace)
2. Or as a function of explicit values and existing variables:
   - `>> c = sqrt(a^2 + b^2)`
3. To suppress output, end the line with a semicolon
   - `>> e=exp(1);`

Scalar Number Operations

Arithmetic operations (+,-,*,/)

Example

```
>> 7/45
ans =
0.1556
>> (1 - i)*(3 +2*i)
ans =
5.000 - 1.0000i
>> 1/0
ans =
Inf
>> 0/0
ans =
NaN
```

AIPLaboratory
at R&D Center

Scalar Number Operations

Exponentiation (^) and Complicated expressions, use parentheses

Example

```
>> 3^2
ans =
9
>> ((2.11+3.43)*5)^0.2
ans =
1.9431
>> (3+4*j)^2
ans =
-7.0000 + 24.0000i
>> 3(1+0.7)
3(1+0.7)
Error: Unbalanced or unexpected parenthesis or
bracket.
```

AIPLaboratory
at R&D Center

MATLAB Technical Language

AIPLaboratory
at R&D Center

Variables and Data Types

1. MATLAB is a weakly typed language
   - No need to initialize variables!
   - No need explicit declaration of variables!
2. MATLAB supports various types, the most often used are:
   - `>> 3.14159265` (64-bits double)
   - `>> 'a'` (16-bits char)
3. Most variables you will deal with will be vectors or matrices of doubles or chars
4. Other types are also supported: complex, symbolic, 16-bit and 8 bit integers, etc. You will be exposed to all these types through the homework

Creating Variables

1. To create a variable, simply assign a value to a name:
   - `>> varPI=3.1415927`
   - `>> myIdentityMatrix = [1 0 0;0 1 0;0 0 1]`
2. Variables names
   - first character must be a LETTER
   - after that, any combination of letters, numbers and _
   - CASE SENSITIVE! (var1 is different from Var1)
3. MATLAB has Built-in variables. Dont use these names!
   - i and j can be used to indicate complex numbers
   - pi has the value 3.1415926...
   - ans stores the last unassigned value (like on a calculator)
   - Inf and -Inf are positive and negative infinity
   - NaN represents "Not a Number"

MATLAB Technical Language

Built-in Functions

1. MATLAB has an enormous library of built-in functions
2. Call using parentheses passing parameter to function
   - `>> sqrt(2)`
   - `>> log(2), log10(0.23)`
   - `>> cos(pi), atan(1)`
   - `>> exp(-i*pi/4)`
   - `>> round(1.3), floor(4.5), ceil(4.5))`
   - `>> angle(i), abs(1+i)`

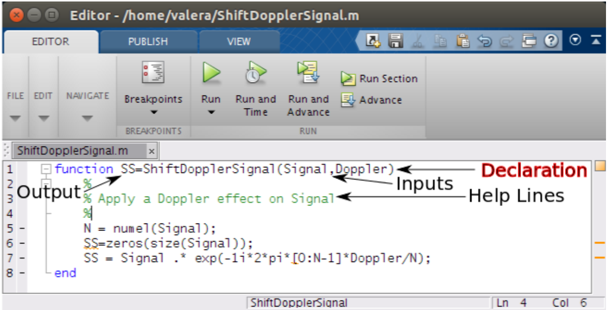
AIPLaboratory
at R&D Center

Element-Wise Functions

1. All the functions that work on scalars also work on vectors
   - `>> t = [3 4 5];`
   - `>> f = exp(t);`
     is the same as
   - `>> f = [exp(t(1)) exp(t(2)) exp(t(3))];`
2. If in doubt, check a functions help file to see if it handles vectors element-wise
3. Operators have two modes of operation:
   - element-wise (.* ./ .^)
   - standard (* / ^)

User-defined Functions

1. Functions look exactly like scripts, but for ONE difference
   Functions must have a function declaration



(1)

2. No need `return`: MATLAB "returns" the variables whose names match those in the function declaration
3. Variable scope: Any variables created within the function but not

AIPLaboratory
at R&D Center

Functions: overloading

1. MATLAB functions are generally overloaded
   - Can take a variable number of inputs
   - Can return a variable number of outputs
2. What would the following commands return:

```
>> A=zeros(2,4,8); % n-dimensional matrices are OK
>> [x,y,z]=size(A)
>> [m,n]=size(A)
>> D=size(A)
```

3. You can overload your own functions by having variable input and output arguments (see `varargin`, `nargin`, `varargout`, `nargout`)

Using Built-In Functions

1. MATLAB provides a large number of built-in functions. The following script uses some of them.

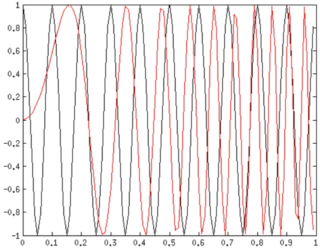
```
% using built-in functions
t = 0:0.01:1; % time vector
x = cos(2 * pi * t / 0.1);
% cos processes each of the entries in
% vector t to get the corresponding value in x
% plotting the function x
figure(1) % numbers the figure
plot(t, x) % interpolated continuous plot
xlabel('t (sec)') % label of x-axis
ylabel('x(t)') % label of y-axis
```

MATLAB Technical Language

Plot Parameters

1. Keeping the variables in memory

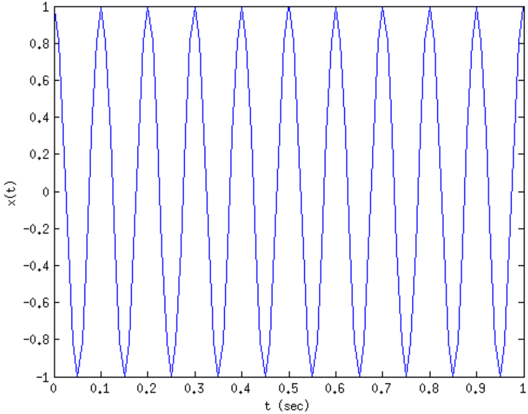
```
y = sin(2 * pi * t.^2 / .1); % notice the dot in ^
% t was defined before
sound(1000 * y, 10000) % to listen to the sinusoid
figure(2) % numbering of the figure
plot(t(1:100),x(1:100),'k',t(1:100),y(1:100),'r')
% plotting x and y on same plot
```



(2)

AIPLaboratory at R&D Center

Visualization Previous Example



(3)

AIPLaboratory at R&D Center

Saving and Loading Data

1. In many situations you would like to either save some data or load some data. The following is one way to do it. Suppose you want to build and save a table of sine values for angles between 0 and 360 degrees in intervals of 3 degrees. This can be done as follows:

```
>> x = 0:3:360;
>> y = sin(x * pi/180); % argument in radians
>> xy = [x' y']; % vector with 2 columns
```

2. Lets now save these values in a file "sine.mat":

```
>> save sine.mat xy
```

3. we use the function load to recover the table "sine"

```
>> clear
>> load sine
>> whos
```

MATLAB Technical Language

AIPLaboratory
at R&D Center

Vectors

1. A vector can be "row vector" or "column vector"
2. Row vector: comma or space separated values between brackets
   - `>> row_vector1 = [1 5 6 7.12]`
   - `>> row_vector2 = [2,5,-4.33,9]`
3. In command window:
   - ```
     >> row_vector1 = [1 5 6 7.12]
     row_vector1 =
     1.0000 2.0000 6.0000 7.1200
     ```
4. In workspace
   - ```
     Name            Size  Bytes  Class
     row_vector1     1x4    32    double array
     ```

Vectors

1. Now we see "column vectors"
2. Column vector: semicolon separated values between brackets
   - `>> column_vector1 = [1;5;6;7.12]`
   - `>> column_vector2 = [2;5;-4.33;9]`
3. In command window:
   - ```
     >> column_vector1 = [1;5;6;7.12]
     column_vector1 =
     1.0000
     2.0000
     6.0000
     7.1200
     ```
4. In workspace
   - ```
     Name             Size Bytes Class
     column_vector1   4x1   32   double array
     ```

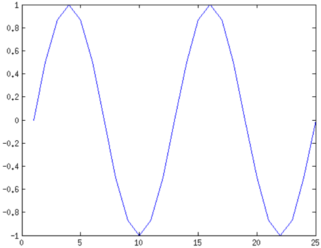
AIPLaboratory
at R&D Center

Vector Indexing

1. MATLAB indexing starts with 1, not 0
2. `>> a(n)` returns the n^{th} element
   ```
   >> a = [4 7 3 9]
   >> a(1) return 4
   >> a(2) return 7
   >> a(3) return 3
   >> a(4) return 9
   ```
3. The index argument can be a vector. In this case, each element is looked up individually, and returned as a vector of the same size as the index vector.
   `>> a(2:3)` return [7 3]
   `>> a(1:end-1)` return [4 7 3]

Examples

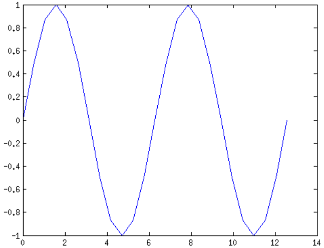
1. `>> x=linspace(0,4*pi,25);`
   `>> y=sin(x);`
2. Plot values against their index:
   `>> plot(y);`



(4)

Examples

1.
```
>> x=linspace(0,4*pi,25);
>> y=sin(x);
```
2. Usually we want to plot y versus x:
```
>> plot(x,y);
```

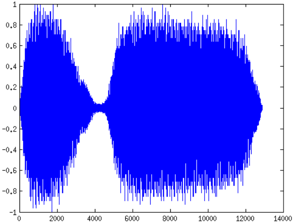


(5)

Train Signals

1. MATLAB provides some data files for experimentation and you only need to load them. The following ''train.mat'' is the recording of a train whistle, sampled at the rate of F_s samples/sec, which accompanies the sampled signal $y(n)$

```
>> clear all
>> load train
>> sound(y, Fs)
>> plot(y)
```



(6)

AIPLaboratory at R&D Center

Saving a Signal as WAV files

1.
```
>> load train
>> audiowrite('y.wav',y,44100) % Save y as y.wav
```
2. 44100 represents the frequency of sampling
3. Other formats are supported:
   - FLAC
   - MP4
   - OGG

Loading a Signal from WAV files

1. `>> clear`
   `>> [y,FS]=audioread('y.wav') % Load y.wav in y`
2. FS represents the frequency of sampling
3. Partial loading is supported:
   - `>> [Y, FS]=audioread(FILENAME, [START END])`
   - START and END represent the initial and final samples
4. Extensions .flac,.mp3,.mp4,.ogg,.m4a are supported
5. `audioread` and `audiowrite` commands leave obsolete `wavread` and `wavwrite` commands

MATLAB Technical Language

Basics on Matrices

1. Element by element:
   - `>> a = [1 2 3;4 5 6]`

$$a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

2. By concatenating vectors or matrices (dimension matters)
   - `>> a = [1 2]`
   - `>> b = [3 4]`
   - `>> c = [5;6]`
   - `>> d = [a;b]`

$$d = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

   - `>> e = [d c]`

$$e = \begin{bmatrix} 1 & 2 & 5 \\ 3 & 4 & 6 \end{bmatrix}$$

Transpose

1. The transpose operators turns a column vector into a row vector and vice versa
   - `>> a = [2 5 7 1-i]`
   - `>> transpose(a)`
   - `>> a'`
   - `>> a.'`
2. The ' gives the Hermitian-transpose, i.e. transposes and conjugates all complex numbers
3. For vectors of real numbers `transpose()` and ' give same result

AIPLaboratory
at R&D Center

Automatic Initialization

1. Initialize a vector of ones, zeros, or random numbers
   - `>> A=ones(1,10)`
     row vector with 10 elements, all 1
   - `>> B=zeros(23,1)`
     column vector with 23 elements, all 0
   - `>> C=rand(10,45)`
     Matrix 10x45 with 450 elements (uniform [0,1])
   - `>> D=nan(1,69)`
     row vector of NaNs (useful for representing uninitialized variables)

Automatic Initialization

1. To initialize a linear vector of values use `linspace`
    - `>> a=linspace(0,10,5)`
      starts at 0, ends at 10 (inclusive), 5 values
    - `>> b=0:2:10`
      starts at 0, increments by 2, and ends at or before 10
      increment can be decimal or negative
    - `>> c=1:5`
      if increment isnt specified, default is 1
    - To initialize logarithmically spaced values use `logspace`

Matrix Indexing

- Matrices can be indexed in two ways:
  - using subscripts (row and column)
  - using linear indices (as if matrix is a vector)
- Subscripts:

```
>> A = [7 3;6 1]
>> A(1,1) return 7
>> A(1,2) return 3
>> A(2,1) return 6
>> A(2,2) return 1
```

- Linear indices:

```
>> A(1) return 7
>> A(2) return 6
>> A(3) return 3
>> A(4) return 1
```

Advanced Indexing

1. `>> A = [1 2 3 4;5 6 7 8;9 10 11 12;13 14 15 16]`

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

   - `>> B = A(1:2,2:3) % return [2 3;6 7]`

     $$B = \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix}$$

   - `>> C = A([1 4 3],[2 4]) % return [2 4;14 16;10 12]`

     $$C = \begin{bmatrix} 2 & 4 \\ 14 & 16 \\ 10 & 12 \end{bmatrix}$$

   - `>> D = A(2,:)  % return [5 6 7 8]`

     $$D = \begin{bmatrix} 5 & 6 & 7 & 8 \end{bmatrix}$$

2. `>> A(:,1) = [-1;-2;-3;-4] % Replace the column 1`

Advanced Indexing

1. MATLAB contains functions to help you find desired values within a vector or matrix
   - `>> vec = [5 3 1 9 7]`
2. To get the minimum value and its index:
   - `>> [Val,Ind] = min(vec); % Val = 1, Ind = 3`
3. To find any the indices of specific values or ranges
   - `>> ind = find(vec == 9); % ind = 4`
   - `>> ind = find(vec > 2 & vec <= 7); % ind = [1 2 5]`
4. To convert between subscripts and indices, use `ind2sub`, and `sub2ind`. Look up `help` to see how to use them.

MATLAB Technical Language

Linear Transformation

Let $x \in \mathbb{R}^N$, $y \in \mathbb{R}^M$

Definition: Linear Transformation:

$$\begin{aligned} \mathcal{G} : \mathbb{R}^N &\to \mathbb{R}^M \\ x &\mapsto y = \mathcal{G}\{x\} \end{aligned}$$

Matrix-Vector Operation:

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{bmatrix} = G.x = \begin{bmatrix} g_{1,1}x_1 & g_{1,2}x_2 & \cdots & g_{1,N}x_N \\ g_{2,1}x_1 & g_{2,2}x_2 & \cdots & g_{2,N}x_N \\ \vdots & \vdots & \ddots & \vdots \\ g_{M,1}x_1 & g_{M,2}x_2 & \cdots & g_{M,N}x_N \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

Linear Transformation: System of Equations

$$y_1 = g_{1,1}x_1 + g_{1,2}x_2 + \ldots + g_{1,N}x_N$$
$$y_2 = g_{2,1}x_1 + g_{2,2}x_2 + \ldots + g_{2,N}x_N$$
$$\vdots$$
$$y_M = g_{M,1}x_1 + g_{M,2}x_2 + \ldots + g_{M,N}x_N$$

MATLAB Code: $M = N = 4$

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

Linear Transf.: Matrix-Vector Operation

```
>>G=[1 4 3 2;2 1 4 3; 3 2 1 4; 4 3 2 1]
G =
 1 4 3 2
 2 1 4 3
 3 2 1 4
 4 3 2 1
>>x=[1;1;1;1];
>>y=G*x
y =
 10
 10
 10
 10
```

Linear Transformation: Matrix Composition

```
>> G=[1 4 3 2;2 1 4 3;3 2 1 4;4 3 2 1];
>> H=[2 0 0 0;0 2 0 0;0 0 2 0;0 0 0 2];
>> z=H*y;% y = G*x
>> T=H*G;% Matrix Composition
>> w=T*x;%(H*G)*x=H*(G*x)
>> z==w
ans =
 1
 1
 1
 1
```

AIPLaboratory
at R&D Center

MATLAB Technical Language

Finite Discrete Signal Filtering

Discrete Fourier Transform (DFT):

- It is an algorithm for the numeric computation of the *Fourier Transform* of a finite discrete signal.
- Let $x_p \in \mathbb{C}^N$.
  Fourier Transform of x_p:
  $\hat{x}_p = \mathcal{F}\{x_p\}$,

$$(\hat{x}_p)[k] = \hat{x}_p[k] = (\mathcal{F}\{x_p\})[k] = \sum_{n=0}^{N-1} x_p[n] W_N^{k.n}, \; k \in \mathbb{Z}_N$$

$$W_N = e^{-j\frac{2\pi}{N}}, \; j = \sqrt{-1}$$

AIPLaboratory at R&D Center

Finite Discrete Signal Filtering
Matrix-Vector DFT Computation

$\hat{x} = F_N . x,$

$$F_N = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & W_N & W_N^2 & \dots & W_N^{N-1} \\ 1 & W_N^2 & W_N^4 & \dots & W_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & \dots & W_N^{(N-1)(N-1)} \end{bmatrix}$$

$$X = (\hat{x})^{\vee} = F_N^{-1} \hat{x} = \frac{1}{N} F_N^{*} . \hat{x}$$

The symbol "*" denotes complex conjugation

MATLAB Code:

```
>> FN=dftmtx(4) % Fourier Matrix of 4th Order
```

AIPLaboratory
at R&D Center

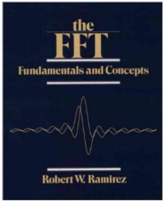
Finite Discrete Signal Filtering

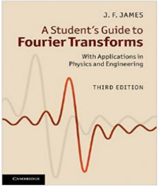
Fast Fourier Transform (FFT):

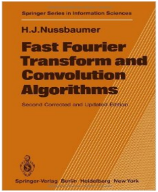
- It is an algorithm for the efficient computation of the DFT.
- MATLAB Code:

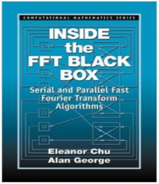
```
> x_hat=fft(x) % x̂ = F{x}
> x=ifft(x_hat) % x = F^-1{x̂}
```

Discrete Signal Filtering
Books on the FFT









(7)

MATLAB Technical Language

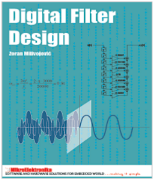
Finite Discrete Signal Filtering

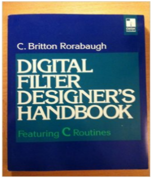
1. It deals with *algorithm treatment* of finite signals in order to extract information relevant to a user.

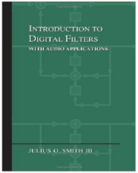
2. The algorithm takes the form of a *cyclic convolution* operation between the signal to be processed, and the signal containing the filtering attributes, the *impulse response signal*.

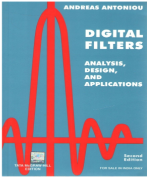
3. "Digital Filters" is the discipline that deals with the analysis, design, and implementation of impulse response signals.

Discrete Signal Filtering
Books on Digital Filters









(8)

Finite Discrete Signal Filtering
Cyclic Convolution of two Signals:

- Given:
  1. Input Signal $x_p \in \mathbb{C}$
  2. Impulse Response Signal $h_p \in \mathbb{C}$
- Compute:
  1. Output Signal $y_p \in \mathbb{C}$
  2. $y_p[n] = \sum_{k=0}^{N-1} x_p[k].h_p[\langle n-k \rangle_N]$ for $n \in \mathbb{Z}_N$
- MATLAB Code:

```
1 > xp=[1;7;3;-5]; % A column vector xp
2 > hp=[1;1;-1;2]; % A column vector hp
3 > yp=cconv(xp,hp); % Cyclic Convolution
```

AIPLaboratory
at R&D Center

Signal Algebra: Binary Operation Cyclic Convolution: $\circledast_N$

$$\begin{aligned}\circledast_N : \mathbb{C}^N \times \mathbb{C}^N &\to \mathbb{C}^N \\ (x_p, h_p) &\mapsto y_p = x_p \circledast_N h_p,\end{aligned}$$

$$y_p[n] = \sum_{k=0}^{N-1} x_p[k].h_p[\langle n-k \rangle_N] \text{ for } n \in \mathbb{Z}_N$$

$$y_p[n] = \sum_{k=0}^{N-1} h_p[k].x_p[\langle n-k \rangle_N] \text{ for } n \in \mathbb{Z}_N$$

$$y_p = x_p \circledast_N h_p = h_p \circledast_N x_p$$

Signal Algebra: Unary Operation
Signal Filtering $\mathcal{T}_{h_p}$

$$\begin{aligned}\mathcal{T}_{h_p} : \mathbb{C}^N \times \mathbb{C}^N &\to \mathbb{C}^N \\ (x_p, y_p) &\mapsto y_p = \mathcal{T}_{h_p}\{x_p\},\end{aligned}$$

For a fixed $h_p \in \mathbb{C}_N$ we can redefine $\mathcal{T}_{h_p}$ as follows:

$$\begin{aligned}\mathcal{T}_{h_p} : \mathbb{C}^N &\to \mathbb{C}^N \\ x_p &\mapsto y_p = \mathcal{T}_{h_p}\{x_p\},\end{aligned}$$

$$y_p[n] = \sum_{k=0}^{N-1} x_p[k].h_p[\langle n-k \rangle_N] \text{ for } n \in \mathbb{Z}_N$$

AIPLaboratory
at R&D Center

Signal Algebra: Binary Operation
Hadamard Product: $\odot_N$

$$\odot_N : \mathbb{C}^N \times \mathbb{C}^N \to \mathbb{C}^N$$
$$(\hat{x}_p, \hat{h}_p) \quad \mapsto \hat{y}_p = \hat{x}_p \odot_N \hat{h}_p,$$

$$\hat{y}_p = \sum_{n=0}^{N-1} \hat{x}_p[n].\hat{h}_p[n]$$

$$\hat{y}_p = D_{\hat{h}_p}.x_p$$

$$D_{\hat{h}_p} \triangleq diag\{\hat{h}_p\} = \begin{bmatrix} \hat{h}_p[0] & 0 & \dots & 0 \\ 0 & \hat{h}_p[1] & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \hat{h}_p[N-1] \end{bmatrix}$$

AIPLaboratory
at R&D Center

Cyclic Convolution Theorems

Let $y_p = x_p \circledast_N h_p$; $x_p, h_p, y_p \in \mathbb{C}^N$

$$\hat{y}_p = \mathcal{F}\{y_p\} \; ; \; \hat{y}_p \in \mathbb{C}^N$$

1. Time-Domain Convolution Theorem:

$$\widehat{x_p \circledast_N h_p} = \hat{x}_p \odot_N \hat{h}_p$$

2. Frequency-Domain Convolution Theorem:

$$\widehat{x_p \odot_N h_p} = \frac{1}{N}\left(\hat{x}_p \circledast_N \hat{h}_p\right)$$

Finite Discrete Signal Filtering

Input Signal: $x_p \in \mathbb{C}^N$
Impulse Response Filter Signal: $h_p \in \mathbb{C}^N$
Filtered Output Signal: $y_p \in \mathbb{C}^N$

Matrix-Vector Filtering Operation:

$$y_p = h_p \circledast_N x_p = H_N . x_p$$

$$H_N = \begin{bmatrix} h_p[0] & h_p[N-1] & \dots & h_p[1] \\ h_p[1] & h_p[0] & \dots & h_p[2] \\ \vdots & \vdots & \ddots & \vdots \\ h_p[N-1] & h_p[N-2] & \dots & h_p[0] \end{bmatrix}$$

Efficient Signal Filtering

$$\mathcal{F}\{y_p\} = \mathcal{F}\{h_p \circledast_N x_p\} = \mathcal{F}\{h_p\} \odot_N \mathcal{F}\{x_p\} = \hat{h}_p \odot_N \hat{x}_p$$

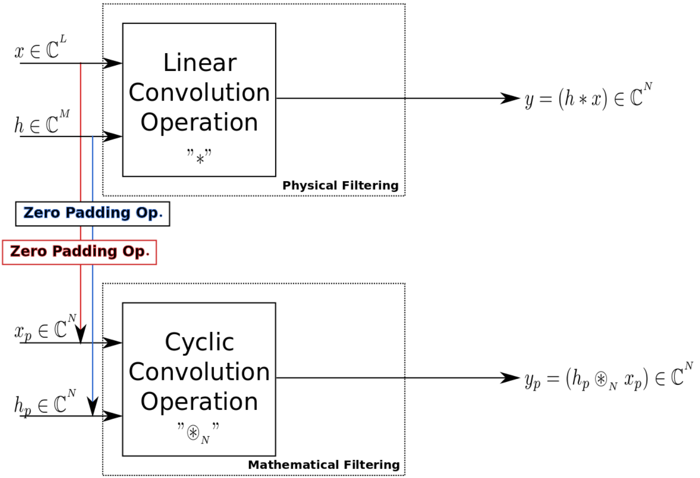
$$y_p = \mathcal{F}^{-1}\{\hat{y}_p\} = \mathcal{F}^{-1}\{\hat{h}_p \odot_N \hat{x}_p\} = \mathcal{F}^{-1}\{D_{\hat{h}_p}.\hat{x}_p\}$$

$$y_p = \mathcal{F}^{-1}\{D_{\hat{h}_p}.(\mathcal{F}\{x_p\})\}$$

$$y_p = F_N^{-1}.D_{\hat{h}_p}.F_N.x_p = \frac{1}{N}\left(F_N^{*} D_{\hat{h}_p} F_N\right).x_p$$

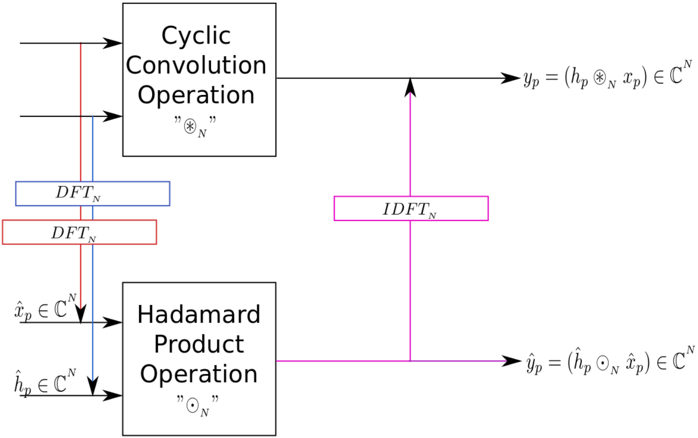
If F_N $\left(F_N^{-1}\right)$ is computed in an efficient manner; then, we have an efficient signal filtering procedure.

Physical Filtering vs. Mathematical Filtering



AIPLaboratory
at R&D Center

Spectral Signal Filtering



AIPLaboratory
at R&D Center

Sources

Main Information Source

MIT OpenCourseWare

http://ocw.mit.edu