# Linear algebra & polynomials: Scilab

Dr. Madhu N. Belur

Control & Computing group
Department of Electrical Engineering
Indian Institute of Technology Bombay
Email: belur@ee.iitb.ac.in

# Outline

# Today's focus

- Scilab is free.
- Matrix/loops syntax is same as for Matlab.
- Scilab provides all basic and many advanced tools.
- Today: linear algebra and polynomials: DFT/interpolation.

# Today's focus

- Scilab is free.
- Matrix/loops syntax is same as for Matlab.
- Scilab provides all basic and many advanced tools.
- Today: linear algebra and polynomials: DFT/interpolation.

# Today's focus

- Scilab is free.
- Matrix/loops syntax is same as for Matlab.
- Scilab provides all basic and many advanced tools.
- Today: linear algebra and polynomials: DFT/interpolation.

# Today's focus

- Scilab is free.
- Matrix/loops syntax is same as for Matlab.
- Scilab provides all basic and many advanced tools.
- Today: linear algebra and polynomials: DFT/interpolation.

# Defining a matrix

- `A=[1 3 4 6]`
- `B=[1 3 4 6;5 6 7 8]`
- `size(A), length(A), ones(A), zeros(B), zeros(3,5)`
- `quote ( ' ) for transpose of A`

# Defining a matrix

- `A=[1 3 4 6]`
- `B=[1 3 4 6;5 6 7 8]`
- `size(A), length(A), ones(A), zeros(B), zeros(3,5)`
- `quote ( ' ) for transpose of A`

# Defining a matrix

- `A=[1 3 4 6]`
- `B=[1 3 4 6;5 6 7 8]`
- `size(A), length(A), ones(A), zeros(B), zeros(3,5)`
- `quote ( ' ) for transpose of A`

# Defining a matrix

- `A=[1 3 4 6]`
- `B=[1 3 4 6;5 6 7 8]`
- `size(A), length(A), ones(A), zeros(B), zeros(3,5)`
- `quote ( ' ) for transpose of A`

# determinant/eigenvalues/trace

- `A=rand(3,3)`
- `det(A), spec(A), trace(A)`
- `sum(spec(A))`
- `if sum(spec(A))==trace(A) then`
  `disp('yes, trace equals sum')`
  `else`
  `disp('no, trace is not sum ')`
  `end`
  `(else disp('Trace is not sum within == tolerance')`
  `abs( sum(spec(A)) - trace(A) ) < 0.0001`
- `T_or_F = abs(prod(spec(A))-det(A))<0.0001`

# determinant/eigenvalues/trace

- `A=rand(3,3)`
- `det(A), spec(A), trace(A)`
- `sum(spec(A))`
- 
```
if sum(spec(A))==trace(A) then
      disp('yes, trace equals sum')
else
      disp('no, trace is not sum ')
end
(else   disp('Trace is not sum within ==
tolerance')
      abs( sum(spec(A)) - trace(A) ) < 0.0001
```
- `T_or_F = abs(prod(spec(A))-det(A))<0.0001`

# determinant/eigenvalues/trace

- A=rand(3,3)
- det(A), spec(A), trace(A)
- sum(spec(A))
- if sum(spec(A))==trace(A) then
  disp('yes, trace equals sum')
  else
  disp('no, trace is not sum ')
  end
  (else disp('Trace is not sum within == tolerance')
  abs( sum(spec(A)) - trace(A) ) < 0.0001
- T_or_F = abs(prod(spec(A))-det(A))<0.0001

# determinant/eigenvalues/trace

- A=rand(3,3)
- det(A), spec(A), trace(A)
- sum(spec(A))
- if sum(spec(A))==trace(A) then
  disp('yes, trace equals sum')
  else
  disp('no, trace is not sum ')
  end
  (else disp('Trace is not sum within == tolerance')
  abs( sum(spec(A)) - trace(A) ) < 0.0001
- T_or_F = abs(prod(spec(A))-det(A))<0.0001

# determinant/eigenvalues/trace

- A=rand(3,3)
- det(A), spec(A), trace(A)
- sum(spec(A))
- if sum(spec(A))==trace(A) then
  disp('yes, trace equals sum')
  else
  disp('no, trace is not sum ')
  end
  (else disp('Trace is not sum within == tolerance')
  abs( sum(spec(A)) - trace(A) ) < 0.0001
- T_or_F = abs(prod(spec(A))-det(A))<0.0001

# determinant/eigenvalues/trace

- `A=rand(3,3)`
- `det(A), spec(A), trace(A)`
- `sum(spec(A))`
- ```
  if sum(spec(A))==trace(A) then
        disp('yes, trace equals sum')
  else
        disp('no, trace is not sum ')
  end
  (else   disp('Trace is not sum within ==
  tolerance')
        abs( sum(spec(A)) - trace(A) ) < 0.0001
  ```
- `T_or_F = abs(prod(spec(A))-det(A))<0.0001`
  ```

# (Block) diagonalize A?

Let $A$ be a square matrix $(n \times n)$ with distinct eigenvalues $\lambda_1, \ldots \lambda_n$. Eigenvectors (column vectors) $v_1$ to $v_n$ are then independent.

$$Av_1 = \lambda_1 v_1 \quad Av_2 = \lambda_2 v_2 \quad \ldots Av_n = \lambda_n v_n$$

$$A \begin{bmatrix} v_1 & v_2 & \ldots & v_n \end{bmatrix} = \begin{bmatrix} v_1 & v_2 & \ldots & v_n \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & \ldots & 0 \\ 0 & \lambda_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \lambda_n \end{bmatrix}$$

(Column scaling of vectors $v_1$, etc is just post-multiplication.)

- $[evectors, evalues] = spec(A)$
- `evalues=spec(A)`
- `inv(evectors)*A*evectors`

Inverse exists because of distint assumption on eigenvalues.
Use 'bdiag' command for block diagonalization

# (Block) diagonalize A?

Let $A$ be a square matrix $(n \times n)$ with distinct eigenvalues $\lambda_1, \ldots \lambda_n$. Eigenvectors (column vectors) $v_1$ to $v_n$ are then independent.

$$Av_1 = \lambda_1 v_1 \quad Av_2 = \lambda_2 v_2 \quad \ldots Av_n = \lambda_n v_n$$

$$A \begin{bmatrix} v_1 & v_2 & \ldots & v_n \end{bmatrix} = \begin{bmatrix} v_1 & v_2 & \ldots & v_n \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & \ldots & 0 \\ 0 & \lambda_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \lambda_n \end{bmatrix}$$

(Column scaling of vectors $v_1$, etc is just post-multiplication.)

- $[evectors, evalues] = spec(A)$
- `evalues=spec(A)`
- `inv(evectors)*A*evectors`

Inverse exists because of distint assumption on eigenvalues.
Use 'bdiag' command for block diagonalization

# (Block) diagonalize A?

Let $A$ be a square matrix $(n \times n)$ with distinct eigenvalues $\lambda_1, \ldots \lambda_n$. Eigenvectors (column vectors) $v_1$ to $v_n$ are then independent.

$$Av_1 = \lambda_1 v_1 \quad Av_2 = \lambda_2 v_2 \quad \ldots Av_n = \lambda_n v_n$$

$$A \begin{bmatrix} v_1 & v_2 & \ldots & v_n \end{bmatrix} = \begin{bmatrix} v_1 & v_2 & \ldots & v_n \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & \ldots & 0 \\ 0 & \lambda_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \lambda_n \end{bmatrix}$$

(Column scaling of vectors $v_1$, etc is just post-multiplication.)

- $[evectors, evalues] = spec(A)$
- `evalues=spec(A)`
- `inv(evectors)*A*evectors`

Inverse exists because of distint assumption on eigenvalues.
Use 'bdiag' command for block diagonalization

# Rank, SVD

- `rank(A)` `svd(A)`
- $[u, s, v] = svd(A)$
- check `u'-inv(u)` `u*s*v-A` `u*s*v'-A`
- `norm(u*s*v'-A)`

# Rank, SVD

- `rank(A)` `svd(A)`
- $[u, s, v] = svd(A)$
- check `u'-inv(u)` `u*s*v-A` `u*s*v'-A`
- `norm(u*s*v'-A)`

# Rank, SVD

- `rank(A)` `svd(A)`
- $[u, s, v] = svd(A)$
- check `u'-inv(u)` `u*s*v-A` `u*s*v'-A`
- `norm(u*s*v'-A)`

# Rank, SVD

- `rank(A)` `svd(A)`
- $[u, s, v] = svd(A)$
- check `u'-inv(u)` `u*s*v-A` `u*s*v'-A`
- `norm(u*s*v'-A)`

# The command 'find'

find([%T %F %T %F %T %F %F])
gives 'indices' of TRUE's.
x=[4 5 6 7]
$x < 5.5$
*true_indices_of _x*=find($x < 5.5$)
y=[5 6 7 8 9 0 -1]
y(*true_indices_of _x*)

# Defining polynomials

Polynomials play a very central role in control theory: the transfer function is a ratio of two polynomials.
Useful commands:

- `s=poly(0,'s')`
- `s=poly(0,'s','roots')`
- `p=s^2+3*s+2`
- `p=poly([2 3 1],'s','coeff')`
- `roots(p)`
- `horner(p,5)`
- $a = [1\ 2\ 3]$
- `horner(p,a)`
- `horner(p,a')`
- `w=poly(0,'w')`
- `horner(p,%i*w)`

# Differentiation

- `p=poly([1 3 4 -3],'s','coeff')`
- `cfp=coeff(p)` constant term first
- `diffpcoff=cfp(2:length(cfp)).*[1:length(cfp)-1]`
- `diffp=poly(diffpcoff,'s','coeff')`
- degree(p) can be used instead of `length(cfp)-1`
- Of course, derivat(p)

# Differentiation

- `p=poly([1 3 4 -3],'s','coeff')`
- `cfp=coeff(p)` `constant term first`
- `diffpcoff=cfp(2:length(cfp)).*[1:length(cfp)-1]`
- `diffp=poly(diffpcoff,'s','coeff')`
- degree(p) can be used instead of `length(cfp)-1`
- Of course, derivat(p)

# Differentiation

- `p=poly([1 3 4 -3],'s','coeff')`
- `cfp=coeff(p)` constant term first
- `diffpcoff=cfp(2:length(cfp)).*[1:length(cfp)-1]`
- `diffp=poly(diffpcoff,'s','coeff')`
- degree(p) can be used instead of `length(cfp)-1`
- Of course, derivat(p)

# Differentiation

- `p=poly([1 3 4 -3],'s','coeff')`
- `cfp=coeff(p)` constant term first
- `diffpcoff=cfp(2:length(cfp)).*[1:length(cfp)-1]`
- `diffp=poly(diffpcoff,'s','coeff')`
- degree(p) can be used instead of `length(cfp)-1`
- Of course, derivat(p)

# Differentiation

- `p=poly([1 3 4 -3],'s','coeff')`
- `cfp=coeff(p)` constant term first
- `diffpcoff=cfp(2:length(cfp)).*[1:length(cfp)-1]`
- `diffp=poly(diffpcoff,'s','coeff')`
- degree(p) can be used instead of `length(cfp)-1`
- Of course, derivat(p)

# Differentiation

- `p=poly([1 3 4 -3],'s','coeff')`
- `cfp=coeff(p)` constant term first
- `diffpcoff=cfp(2:length(cfp)).*[1:length(cfp)-1]`
- `diffp=poly(diffpcoff,'s','coeff')`
- degree(p) can be used instead of `length(cfp)-1`
- Of course, derivat(p)

# Coefficients and powers as vectors

A polynomial $p(s)$ of degree $n$ can be written as

$$p(s) = [1 \; s \; s^2 \; \cdots s^n] \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_n \end{bmatrix} = [p_0 \; p_1 \; \cdots \; p_n] \begin{bmatrix} 1 \\ s \\ \vdots \\ s^n \end{bmatrix}$$

Suppose $p(s)$ has value $b_1$ when $s = a_1$, i.e. $p(a_1) = b_1$. Then,

$$b_1 = [1 \; a_1 \; a_1^2 \; \cdots a_1^n] \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_n \end{bmatrix}$$

Similarly, for $b_2 = p(a_2)$, etc.

# Interpolation and Van der Monde matrix

Suppose $a_1$, $a_2$, $\ldots$, $a_n$ are $n$ (distinct?) numbers. Construct the $n \times n$ matrix:

$$V(a_1, \ldots, a_n) := \begin{bmatrix} 1 & a_1 & a_1^2 & \cdots & a_1^{n-1} \\ 1 & a_2 & a_2^2 & \cdots & a_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_n & a_n^2 & \cdots & a_n^{n-1} \end{bmatrix}$$

This square matrix is nonsingular if and only if $a_i$ are all distinct. Relation with interpolation?

# Polynomial interpolation

Suppose $a_1$, $a_2$, $\ldots$, $a_n$ are $n$ distinct numbers, and suppose $b_1$, $b_2$, $\ldots$, $b_n$ are desired values an interpolating polynomial is required to have. 'Assume' there exists some $n-1$ degree polynomial $p(s)$ with $p(s) = p_0 + p_1 s + \cdots + p_{n-1} s^{n-1}$ such that $p(a_1) = b_1$, etc.

$$\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = \begin{bmatrix} 1 & a_1 & a_1^2 & \cdots & a_1^{n-1} \\ 1 & a_2 & a_2^2 & \cdots & a_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_n & a_n^2 & \cdots & a_n^{n-1} \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_{n-1} \end{bmatrix}$$

Using nonsingularity, can find coefficients of $p(s)$ 'easily' by inverting the Van der Monde matrix.

# Lagrange polynomial interpolation

Same problem as before.
Define
$\ell_1(s) := (s - a_2)(s - a_3) \cdots (s - a_n)$ (all except $a_1$)
$\ell_2(s) := (s - a_1)(s - a_3) \cdots (s - a_n)$ (all except $a_2$) etc.
$\ell_i(s) := (s - a_1) \cdots (s - a_{i-1})(s - a_{i+1}) \cdots (s - a_n)$
Of course, $\ell_i(a_j) = 0$ if and only if $i \neq j$.
Now also normalize them so that $\ell_i(a_i) = 1$. (Use `horner` )
Then required interpolating polynomial is
$b_1\ell_1(s) + b_2\ell_2(s) + \cdots + b_n\ell_n(s)$

# Lagrange polynomial interpolation

Write a function that takes numbers $a_1, \ldots\ a_n$ and gives a polynomial with these numbers as roots.

Write a function that takes numbers $a_1, \ldots\ a_n$ and gives $n$ polynomials: first polynomial has $a_2$ upto $a_n$ as roots, second has $a_1$, $a_3$, $\ldots a_n$ as roots, etc.

Use horner to 'normalize' these polynomials.

# Lagrange polynomial interpolation

Write a function that takes numbers $a_1, \ldots\ a_n$ and gives a polynomial with these numbers as roots.
Write a function that takes numbers $a_1, \ldots\ a_n$ and gives $n$ polynomials: first polynomial has $a_2$ upto $a_n$ as roots, second has $a_1$, $a_3$, $\ldots a_n$ as roots, etc.
Use horner to 'normalize' these polynomials.

# Lagrange polynomial interpolation

Write a function that takes numbers $a_1, \ldots\ a_n$ and gives a polynomial with these numbers as roots.
Write a function that takes numbers $a_1, \ldots\ a_n$ and gives $n$ polynomials: first polynomial has $a_2$ upto $a_n$ as roots, second has $a_1$, $a_3$, $\ldots a_n$ as roots, etc.
Use horner to 'normalize' these polynomials.

# Discrete Fourier Transform

For a periodic sequence: DFT (Discrete Fourier Transform) gives the frequency content.
Linear transformation on the input sequence.
Take signal values of just one period: finite dimensional signal (due to periodicity of $N$).

$$X(k) := \sum_{n=0}^{N-1} x(n) e^{\frac{-2\pi i k}{N} n} \text{ for } k = 0, \ldots, N-1 \text{ (analysis equation)}$$

$e^{-\frac{2\pi i}{N}}$ is the $N^{\text{th}}$ root of unity.
Inverse DFT for the synthesis equation. Normalization constants vary in the literature.

# Discrete Fourier Transform

What is the matrix defining relating the DFT $X(k)$ of the signal $x(n)$? Define $\omega := e^{-\frac{2\pi i}{N}}$.

$$\begin{bmatrix} X(0) \\ X(1) \\ \vdots \\ X(N-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2N-2} \\ 1 & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{2N-2} & \cdots & \omega^{(N-1)^2} \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ \vdots \\ x(N-1) \end{bmatrix}$$

(Note: $\omega^N = 1$, etc.)
Check that the above $N \times N$ matrix has nonzero determinant.
(Change of basis.) Moreover, columns are orthogonal.
Orthonormal? (Normalization (by $\sqrt{N}$) not done yet.)

# Discrete Fourier Transform and interpolation

Van der Monde matrix: closely related to interpolation problems

Of course, inverse DFT is nothing but interpolation. Construct $p(s) := x_0 + x_1 s + x_2 s^2 \cdots + x_{N-1} s^{N-1}$

To obtain $X(k)$, evaluate $p$ at $s = \omega^k$.

$X(k) = p(\omega^k)$ `horner` command

Given values of $p(\omega^k)$ for various $\omega^k$ (i.e., $X(k)$), find the coefficients of the polynomial $p(s)$: inverse DFT: interpolation of a polynomial to 'fit' given values at specified (complex) numbers.

# Discrete Fourier Transform and interpolation

Van der Monde matrix: closely related to interpolation problems
Of course, inverse DFT is nothing but interpolation. Construct
$p(s) := x_0 + x_1 s + x_2 s^2 \cdots + x_{N-1} s^{N-1}$
To obtain $X(k)$, evaluate $p$ at $s = \omega^k$.
$X(k) = p(\omega^k)$ `horner` command
Given values of $p(\omega^k)$ for various $\omega^k$ (i.e., $X(k)$), find the coefficients of the polynomial $p(s)$: inverse DFT: interpolation of a polynomial to 'fit' given values at specified (complex) numbers.

# Discrete Fourier Transform and interpolation

Van der Monde matrix: closely related to interpolation problems
Of course, inverse DFT is nothing but interpolation. Construct
$p(s) := x_0 + x_1 s + x_2 s^2 \cdots + x_{N-1} s^{N-1}$
To obtain $X(k)$, evaluate $p$ at $s = \omega^k$.
$X(k) = p(\omega^k)$ `horner` command
Given values of $p(\omega^k)$ for various $\omega^k$ (i.e., $X(k)$), find the coefficients of the polynomial $p(s)$: inverse DFT: interpolation of a polynomial to 'fit' given values at specified (complex) numbers.

# Discrete Fourier Transform and interpolation

Van der Monde matrix: closely related to interpolation problems
Of course, inverse DFT is nothing but interpolation. Construct
$p(s) := x_0 + x_1 s + x_2 s^2 \cdots + x_{N-1} s^{N-1}$
To obtain $X(k)$, evaluate $p$ at $s = \omega^k$.
$X(k) = p(\omega^k)$ `horner` command
Given values of $p(\omega^k)$ for various $\omega^k$ (i.e., $X(k)$), find the coefficients of the polynomial $p(s)$: inverse DFT: interpolation of a polynomial to 'fit' given values at specified (complex) numbers.

# Discrete Fourier Transform and interpolation

Van der Monde matrix: closely related to interpolation problems
Of course, inverse DFT is nothing but interpolation. Construct
$p(s) := x_0 + x_1 s + x_2 s^2 \cdots + x_{N-1} s^{N-1}$
To obtain $X(k)$, evaluate $p$ at $s = \omega^k$.
$X(k) = p(\omega^k)$ `horner` command
Given values of $p(\omega^k)$ for various $\omega^k$ (i.e., $X(k)$), find the coefficients of the polynomial $p(s)$: inverse DFT: interpolation of a polynomial to 'fit' given values at specified (complex) numbers.

# Conclusions

- Matrices and polynomials provide rich source of problems
- Due to good computational tools available currently, the future lies in computational techniques
- Scilab provides handy tools
- We saw: `for horner poly coeff roots find`