FF505/FY505

Computational Science

Lecture 1

Introduction to Matlab

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

Outline

On the Course

Organization

1. Introduction to mathematical tools (Claudio) – weeks 5-8
2. Tutorial on numerical software, MATLAB (Marco) – weeks 5-8
3. Laboratories on applications in physics (Paolo) – weeks 7-11

Evaluation

Group project during the laboratory session + oral exam

MATLAB Section

The MATLAB Section will cover

- MATLAB interactive environment
- MATLAB vectorized operations
- MATLAB programming
- data input/output
- simple visualization.

More specifically, it should prepare you to carry out the exercises from the theory and laboratory sections.

MATLAB Section – Schedule

- Schedule for weeks 5-8 (4 weeks):
  - Lecture, Thursday, 10:15-12:00, U140, (weeks 05-08)
  - Tutorials:
    - S1 (FF505/FY505), Thursday, 12-14 U10 (weeks 05-08)
    - S6 (FF505), Friday, 12-14, U14, (week 05)
      S6 (FF505), Friday, 08-10, U14 (weeks 06-08)
    - S9 (FY505), Friday, 10-12, U49c (weeks 05-08)

From week 7 you'll start also laboratories

- Communication tools
  - BlackBoard (BB)
    (link to MATLAB Section `http://www.imada.sdu.dk/~marco/FF505`)
  - Announcements in BlackBoard
  - Personal email of instructors and Marco
  - Ask peers

Hands on Experience

Weekly exercises to be carried out in your study group outside of tutorial sessions.

Slides and exercises sheets are posted after lecture at
http://www.imada.sdu.dk/~marco/FF505

Getting MATLAB

- machines in IMADA terminal room and in U26B (12 PCs)
  (type `matlab` from command line)
- use a Matlab clone, eg, Octave
- buy the student edition of Matlab: 89$ (ca. 500 DDK)
  Link: http://www.mathworks.se/academia/student_version/
  Then click on "BUY NOW"

MATLAB

MATLAB (**ma**trix **la**boratory) is a high-level language and interactive environment to perform computationally intensive numerical computations faster than with low-level programming languages such as C, C++, and Fortran.

- Developed by a privately held company, MathWorks, 70% located at the company's headquarters in Massachusetts.
- Stable release: 2012b (we have 2008b)
- Written in C, Java
- License: Proprietary

Scientific vs Symbolic Computing

- scientific computing is based on numerical computation with approximate floating point numbers.

- symbolic computation manipulates mathematical expressions and other mathematical objects.
  emphasis on exact computation with expressions containing variables that have not any given value and are thus manipulated as symbols

  $\rightsquigarrow$ Try `http://www.wolframalpha.com`

Symbolic computation can be done in MATLAB with the Symbolic Math Toolbox and the MuPAD editor (not installed)

Other similar numerical computing environments with high-level programming language are:

- Maple `www.maplesoft.com` (symbolic) – Proprietary
- Mathematica `http://www.wolfram.com/mathematica` (discrete mathematics) – [Proprietary]
- Octave `www.gnu.org/software/octave` – [General Public License]
- R `www.r-project.org` (statistics) – [GPL]
- Sage `www.sagemath.org` (discrete mathematics) – [GPL]
- SciPy `www.scipy.org` (based on python) – [GPL]
- ...

Outline

MATLAB Desktop

- Command window
- Workspace
- Command history
- Current folder browser
- Variable editor
- MATLAB program editor
- Help
- Desktop menu
- Docking/Undocking, maximize by double click
- Current folder
- Search path (`File menu -> set path`)

Command line programming

```
%%% elementary operations
5+6
3-2
5*8
1/2
2^6
1 == 2 % false
1 ~= 2 % true. note, not "!="
1 && 0
1 || 0
xor(1,0)
```

Variable Assignment

The = sign in MATLAB represents the assignment or replacement operator.
It has a different meaning than in mathematics.

Compare:

`x = x + 3` In math it implies 0=2, which is an invalid statement
In MATLAB it adds 2 to the current value of the variable

```
%% variable assignment
a = 3; % semicolon suppresses output
b = 'hi';
c = 3>=1;

% Displaying them:
a = pi
disp(sprintf('2 decimals: %0.2f', a))
disp(sprintf('6 decimals: %0.6f', a))
format long % 16 decimal digits
a
format short e % 4 decimal digits +
     scientific notation
a
```

```
x + 2 = 20 % wrong statement
x = 5 + y % wrong if y unassigned
```

Variables are visible in the workspace

Names:

- [a-z] [A-Z] [0-9] _
- case sensitive
- max 63 chars

Managing the Work Session

```
who % lists variables currently in memory
whos % lists current variables and sizes
clear v % clear w/ no argt clears all
edit filename % edit a script file
clc % clears theCommand window
... % ellipsis; continues a line
help rand % returns help of a function
quit % stops MATLAB
```

Predefined variables

```
pi
Inf % 5/0
NaN % 0/0
eps % accuracy of computations
i,j % immaginary unit i=j=sqrt(-1)
3+8i % a complex number (no *)
Complex(1,-2)
```

Working with Files

MATLAB handles three types of files:

- M-files `.m`: Function and program files
- MAT-files `.mat`: binary files with name and values of variables
- data file `.dat`: ASCII files

```
%% loading data
load q1y.dat
load q1x.dat
save hello v; % save variable v into file
     hello.mat
save hello.txt v -ascii; % save as ascii
% fopen, fprintf, fscanf also work
% ls %% cd, pwd & other unix commands
      work in matlab;
% to access shell, preface with "!"
```

Files are stored and search in current directory and search path

Directories and paths

If we type `problem1`

1. seeks if it is a variable and displays its value
2. checks if it is one of its own programs and executes it
3. looks in the current directory for file `program1.m` and executes the file
4. looks in the search path for file `program1.m` and executes it

```
addpath dirname % adds the directory dirname to the search path
cd dirname % changes the current directory to dirname
dir % lists all files in the current directory
dir dirname % lists all files in dirname
path % displays the MATLAB search path
pathtool % starts the Set Path tool
pwd % displays the current directory
rmpath dirname % removes the directory dirname from the search path
what % lists MATLAB specific files in the current directory
what dirname % lists MATLAB specific files in dirname
which item % displays the path name of item
```

Arrays and Matrices

Arrays are the basic data structures of MATLAB
Types of arrays:
numeric • character • logical • cell • structure • function handle

```
%% vectors and matrices
A = [1 2; 3 4; 5 6]

v = [1 2 3]
v = [1; 2; 3]
v = [1:0.1:2] % from 1 to 2, with stepsize of 0.1. Useful for plot axes
v = 1:6 % from 1 to 6, assumes stepsize of 1

C = 2*ones(2,3) % same as C = [2 2 2; 2 2 2]
w = ones(1,3) % 1x3 vector of ones
w = zeros(1,3)
w = rand(1,3) % drawn from a uniform distribution
w = randn(1,3) % drawn from a normal distribution (mean=0, var=1)
w = -6 + sqrt(10)*(randn(1,10000)) % (mean = 1, var = 2)
hist(w) % histogram
e = []; % empty vector
I = eye(4) % 4x4 identity matrix
A = linspace(5,8,31) % equivalent to 5:0.1:8
```

```
%% indexing
A(3,2) % indexing is (row,col)
A(2,:) % get the 2nd row. %% ":" means every elt along that dimension
A(:,2) % get the 2nd col
A(1,end) % 1st row, last elt. Indexing starts from 1.
A(end,:) % last row

A([1 3],:) = [] % deletes 1st and 3rd rows
A(:,2) = [10 11 12]' % change second column
A = [A, [100; 101; 102]]; % append column vec
% A = [ones(size(A,1),1), A]; % e.g bias term in linear regression
A(:) % Select all elements as a column vector.
```

```
%% dimensions
sz = size(A)
size(A,1) % number of rows
size(A,2) % number of cols
length(v) % size of longest dimension
```

Matrix Operations

```
%% matrix operations
A * C % matrix multiplication
B = [5 6; 7 8; 9 10] % same dims as A
A .* B % element-wise multiplcation
% A .* C or A * B gives error - wrong dimensions
A .^ 2
1./v
log(v) % functions like this operate element-wise on vecs or matrices
exp(v) % e^4
abs(v)

-v % -1*v

v + ones(1,length(v))
% v + 1 % same

A' % (conjuate) transpose
```

Plots


```
%% plotting
t = [0:0.01:0.98];
y1 = sin(2*pi*4*t);
plot(t,y1);
y2 = cos(2*pi*4*t);
hold on; % "hold off" to turn off
plot(t,y2,'r--');
xlabel('time');
ylabel('value');
legend('sin','cos');
title('my plot');
close; % or, "close all" to close all figs

figure(2), clf; % can specify the figure number
subplot(1,2,1); % Divide plot into 1x2 grid, access 1st element
plot(t,y1);
subplot(1,2,2); % Divide plot into 1x2 grid, access 2nd element
plot(t,y2);
axis([0.5 1 -1 1]); % change axis scale
```

Control Flow

if

```
if w(1)==0
    % <statement>
elseif w(1)==1
    % <statement>
else
    % <statement>
end
```

switch

```
method = 'Bilinear';
switch lower(method)
   case {'linear','bilinear'}
      disp('Method is linear')
   case 'cubic'
      disp('Method is cubic')
   case 'nearest'
      disp('Method is nearest')
   otherwise
      disp('Unknown method.')
end
```

for

```
w = [];
z = 0;
is = 1:10
for i=is
    w = [w, 2*i] % Same as \/
% w(i) = 2*i
% w(end+1) = 2*i
    z = z + i;
    % break;
    % continue;
end
% avoid! same as w = 2*[1:10], z = sum([1:10]);
```

while

```
w = [];
while length(w) < 3
    w = [w, 4];
    % break
end
```

Vectorization

One way to make your MATLAB programs run faster is to vectorize the algorithms. A simple example involves creating a table of logarithms:

```
x = .01;
for k = 1:1001
   y(k) = log10(x);
   x = x + .01;
end
```

A vectorized version of the same code is

```
x = .01:.01:10;
y = log10(x);
```

Some functions are vectorized, hence with vectors must use element-by-element operators to combine them.
Eg: $z = e^y \sin x$, x and y vectors:

```
z=exp(y).*sin(x)
```

Script and Function Files

```
x=(1:1000)';
for k=1:5
y(:,k)=k*log(x);
end
plot(x,y)
```

command line `simple`

```
function y=simpl(maxLoop)
        % (smart indent)
        x=(1:1000)';
        for k=1:maxLoop
        y(:,k)=k*log(x);
        end
        plot(x,y)
```

command line `g=simple(10)`

Same name conventions for `.m` files as for variables.

```
exist("example1")
exist("example1.m","file")
exist("example1","builtin")
```

Debugging:
Two types of errors: (i) syntax errors (ii) runtime errors

- test on small examples whose result can be verified by hand
- display intermediate calculations
- use the debugger (not needed in this course)

Rapid Code Iteration

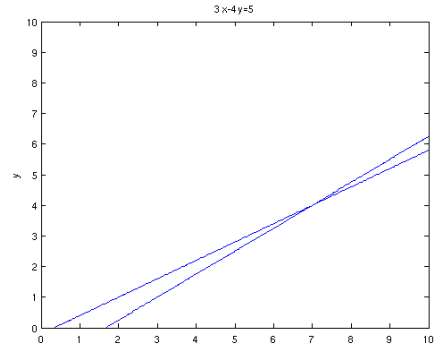
- Rapid code iterations using cells in the editor
- cells are small sections of code performing specific tasks
- they are separated by double %
- they can be executed independently, eg, CTRL+Enter and their parameters adjusted
- navigate by CTRL+SHIFT+Enter or by jumping
- publish in HTML or Word.

Outline

Systems of Linear Equations

$$\begin{aligned} 6x- \quad 10y &= \quad 2 \\ 3x- \quad 4y &= \quad 5 \end{aligned}$$

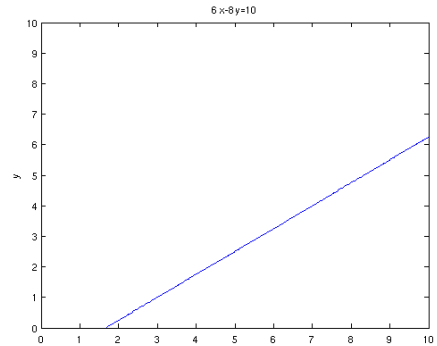
```
% plot functions in implicit form
ezplot('6*x-10*y=2',[0 10 0 10]),
hold,
ezplot('3*x-4*y=5',[0 10 0 10])
```



has one single solution

$$\begin{aligned} 3x- \quad 4y &= \quad 5 \\ 6x- \quad 8y &= \quad 10 \end{aligned}$$

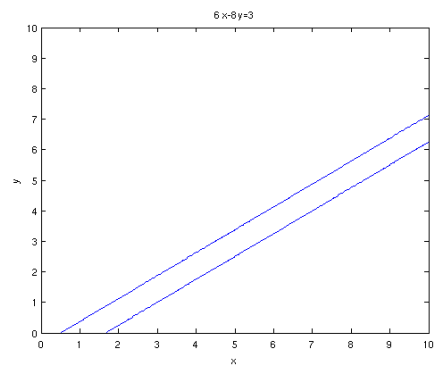
```
ezplot('3*x-4*y=5',[0 10 0 10]),
hold,
ezplot('6*x-8*y=10',[0 10 0 10])
```



has infinite solutions

$$\begin{aligned} 3x- \quad 4y &= \quad 5 \\ 6x- \quad 8y &= \quad 3 \end{aligned}$$

```
ezplot('3*x-4*y=5',[0 10 0 10]),
hold,
ezplot('6*x-8*y=3',[0 10 0 10])
```



has no solution

Matrix Form

The linear system:

$$\begin{aligned} 2x_1 + \quad 9x_2 &= \quad 5 \\ 3x_1 - \quad 4x_2 &= \quad 7 \end{aligned}$$

can be expressed in vector-matrix form as:

$$\begin{bmatrix} 2 & 9 \\ 3 & -4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 5 \\ 7 \end{bmatrix}$$

In general, a set of m equations in n unknowns can be expressed in the form $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A}$ is $m \times n$, $\mathbf{x}$ is $n \times 1$ and $\mathbf{b}$ is $m \times 1$.

The inverse of $\mathbf{A}$ is $\mathbf{A}^{-1}$ and has property that

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$$

Hence the solution to our system is:

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

A matrix is singular if $\det(\mathbf{A}) = |\mathbf{A}| = 0$

Inverse of a square matrix $\mathbf{A}$ is defined only if $\mathbf{A}$ is nonsingular.

If $\mathbf{A}$ is singular, the system has no solution

```
>> A=[3 -4; 6 -8];
>> det(A)
ans =
     0
>> inv(A)
Warning: Matrix is singular to working precision.
ans =
   Inf Inf
   Inf Inf
```

Calculating the inverse

$$\mathbf{A}^{-1} = \frac{1}{|\mathbf{A}|}\mathrm{adj}(\mathbf{A})$$

$\mathrm{adj}(\mathbf{A})$ is the adjugate matrix of $\mathbf{A}$:

1. Calculate the (i, j) minor of $\mathbf{A}$, denoted $\mathbf{M}_{ij}$, as the determinant of the $(n-1) \times (n-1)$ matrix that results from deleting row i and column j of $\mathbf{A}$.
2. Calculate the cofactor matrix of $\mathbf{A}$, as the $n \times n$ matrix $\mathbf{C}$ whose (i, j) entry is the (i, j) cofactor of $\mathbf{A}$

$$\mathbf{C}_{ij} = (-1)^{i+j}\mathbf{M}_{ij}$$

3. set $\mathrm{adj}(\mathbf{A})_{ij} = C_{ji}$

For a 2×2 matrix the matrix inverse is

$$\mathbf{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \qquad \mathbf{A}^{-1} = \frac{1}{|\mathbf{A}|} \begin{bmatrix} d & -c \\ -b & a \end{bmatrix}^T = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

For a 3×3 matrix

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

the matrix inverse is

$$\mathbf{A}^{-1} = \frac{1}{|\mathbf{A}|} \begin{bmatrix} + \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} & - \begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} & + \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix} \\ - \begin{vmatrix} a_{12} & a_{13} \\ a_{32} & a_{33} \end{vmatrix} & + \begin{vmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{vmatrix} & - \begin{vmatrix} a_{11} & a_{12} \\ a_{31} & a_{32} \end{vmatrix} \\ + \begin{vmatrix} a_{12} & a_{13} \\ a_{22} & a_{23} \end{vmatrix} & - \begin{vmatrix} a_{11} & a_{13} \\ a_{21} & a_{23} \end{vmatrix} & + \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} \end{bmatrix}^T$$

Left Division Method

- $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ rarely applied in practice because calculation is likely to introduce numerical inaccuracy
- The inverse is calculated by LU decomposition, the matrix form of Gaussian elimination.

```
% left division method
x = A\b
```

$$\mathbf{A} = \mathbf{LU}$$
$$\mathbf{PA} = \mathbf{LU}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

- for a matrix $\mathbf{A}$, $n \times n$, $\det(\mathbf{A}) \neq 0 \Leftrightarrow$ rank of $\mathbf{A}$ is n
  - for a system $\mathbf{Ax} = \mathbf{b}$ with m equations and n unknowns a solution exists iff $\mathbf{rank}(\mathbf{A}) = \mathbf{rank}([\mathbf{Ab}]) = r$
    - if $r = n \rightsquigarrow$ unique
    - if $r < n \rightsquigarrow$ infinite sol.
  - for a homogeneous system $\mathbf{Ax} = \mathbf{0}$ it is always $\mathbf{rank}(\mathbf{A}) = \mathbf{rank}([\mathbf{Ab}])$ and there is a nonzero solution iff $\mathbf{rank}(\mathbf{A}) < n$

- `A\b` works for square and nonsquare matrices. If nonsquare ($m < n$) then the system is undetermined (infinite solutions). `A\b` returns one variable to zero
- `A\b` does not work when $\det(\mathbf{A}) = 0$.

```
>> A=[2, -4,5;-4,-2,3;2,6,-8];
>> b=[-4;4;0];
>> rank(A)
ans =
     2
>> rank([A,b])
ans =
     2
>> x=A\b
Warning: Matrix is singular to working
      precision.
x =
   NaN
   NaN
   NaN
```

However since

$$\mathbf{rank}(\mathbf{A}) = \mathbf{rank}([\mathbf{Ab}])$$

an infinite number of solutions exist (undetermined system).
`x=pinv(A)b` solves with pseudoinverse and `rref([A,b])` finds the reduced row echelon form

Overdetermined Systems

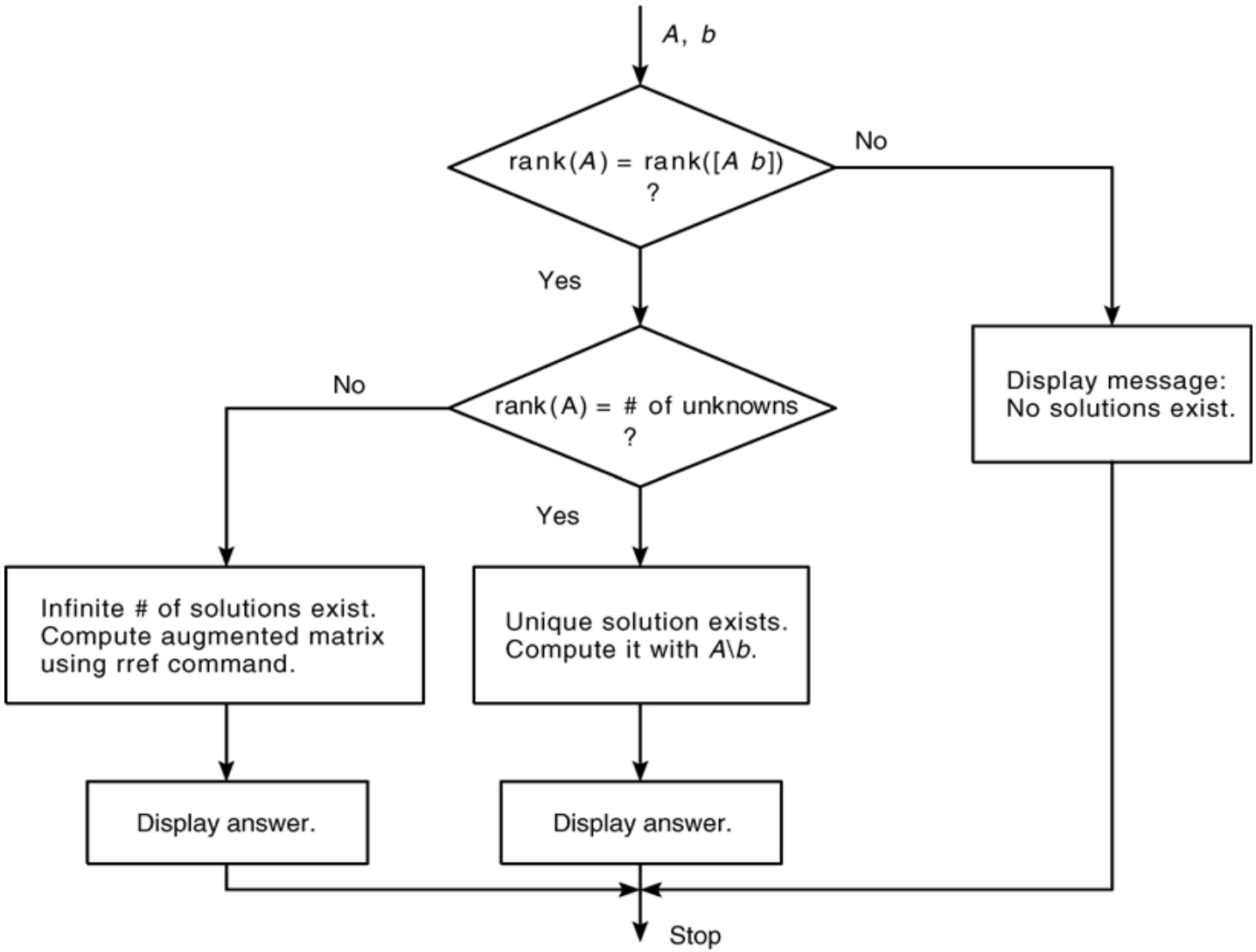
An overdetermined system is a set of equations that has more independent equations than unknowns.

For such a system the matrix inverse method will not work because the A matrix is not square.

However, some overdetermined systems have exact solutions, and they can be obtained with the left division method `x = A \ b`

For other overdetermined systems, no exact solution exists. We need to check the ranks of $\mathbf{A}$ and $[\mathbf{Ab}]$ to know whether the answer is the exact solution. If a solution should not exist the left-division answer is a least squares solution.

Flowchart for Linear System Solver



Resume

- Overview to MATLAB programming, environment and arrays
- Solving linear systems in MATLAB
- Work at the posted exercises in small groups