Lecture 6: MATLAB Functions / Coordinate Transformations

Professor Erik Cheever
Remember...

- 10/8: Lab 4/5 – MATLAB 2 – submitted on moodle
- 10/8: Learning styles questionnaire.
- 10/22: MATLAB 3 (not yet assigned)– submitted on moodle.

The big picture...

- Today: MATLAB functions and coordinate transformations
- Lab – Soldering.
- After break – some more SolidWorks.
Functions

Sometimes we would like to repeat a set of commands repeatedly.

An example:

Suppose we want to find the perimeter of a shape defined by a polygon, or “patch.”

Recall – distance from \((x_1,y_1)\) to \((x_2,y_2)\) is

\[
d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}
\]
Finding the perimeter  

Algorithm:
To find the perimeter we:

1. find the difference between consecutive points in x and in y,
2. square each difference in x and in y,
3. add the square for corresponding x and y differences.
4. take the square root of each of these to find distance between consecutive points, and
5. sum these distances.

\[ d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \]
Finding perimeter (1)

```matlab
>> %Define shape (a parallelogram)
>> x = [0 1 3 2]; y=[0 1 1 0];
>> myShape=patch(x,y,[0 0 1]);
>> axis([-3 3 -3 3],'square'); grid on

>> %Difference between x values
>> xd=diff(x)
xd = 1  2  -1

This is no good because it leaves off difference between last point and first point. Correct this by adding first point to end of vector.

>> x1=[x x(1)]
x1 = 0 1 3 2 0

>> xd=diff(x1)
xd = 1 2 -1 -2
```
Finding perimeter (2)

\[ d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \]

```matlab
>> x = [0 1 3 2];  y=[0 1 1 0];  define vertices of patch
>> x1=[x x(1)];  %Make new vectors, adding first point to end
>> y1=[y y(1)]
y1 = 0 1 1 0 0 0

>> xd=diff(x1);  %Find difference between consecutive points
>> yd=diff(y1)
yd = 1 0 -1 0

>> sqrd = xd.*xd + yd.*yd  % Square difference; add (x^2+y^2)
sqrd = 2 4 2 4

>> sqrtSqd=sqrt(sqrd)  % Get length of each side
sqrtSqd = 1.4142 2.0000 1.4142 2.0000

>> perim=sum(sqrtSqd)  %Get perimeter
perim = 6.8284
```

Step 1 in algorithm

Steps 2 and 3

Steps 4

Step 5
Aside: element-by-element operations in MATLAB

- Given the one by four (1x4) vector z, \( z = [1 \ 2 \ -1 \ 3] \).
  How can we calculate a vector in which each element is squared (i.e., the vector \([1 \ 4 \ 1 \ 9]\)).

  \[
  \text{>> } z^*z \quad \% \text{ Try multiplying the vectors.}
  \]

  \[
  \text{Error using } *
  \]

  \[
  \text{Inner matrix dimensions must agree.}
  \]

- Array multiplication is undefined for (1x4 vector)\((1x4\text{ vector})!

- We must use “element-by-element” multiplication, “.\*”

  \[
  \text{>> } z.\*z \quad \% \text{ Use element-by-element multiplication}
  \]

  \[
  \text{ans = } \begin{bmatrix} 1 & 4 & 1 & 9 \end{bmatrix}
  \]

  \[
  \text{>> } z.\^2 \quad \% \text{ ... or element-by-element exponentiation}
  \]

  \[
  \text{ans = } \begin{bmatrix} 1 & 4 & 1 & 9 \end{bmatrix}
  \]
Problems with this approach to finding perimeter

- If we want to calculate the perimeter of 10 different shapes, we need to enter the code 10 different times.
- If we find a mistake in the code, we need to change the code every place it appears.
- We can fix all of these issues by defining a “script” or a “function”.
- We’ll examine both – a function is generally superior.
A script to calculate perimeter

%% perimScript.m
% This script calculates perimeter of an object with vertices stored
% in vectors "x" and "y" and stores the result in variable "perim"

x1=[x x(1)];    % Repeat first point of x at end of vector
y1=[y y(1)];    % ... and for y

xd=diff(x1);    % Find difference between x and y values.
yd=diff(y1);

perim = sum(sqrt(xd.^2 + yd.^2));  % Find perimeter
Using script to calculate perimeter (1)

```matlab
>> x1 = [0 1 3 2];     % Define parallelogram
>> y1 = [0 1 1 0];

>> theta = 0:45:360;  % 8 equally spaced angles on circle
>> x2 = 0.5*cosd(theta); % Define octagon (8 point on circle).
>> y2 = 0.5*sind(theta);

>> % Use script to find perimeter of parallelogram.
>> x = x1;  y=y1;  % Assign x and y values (used in script).
>> perimScript % Perform calculation (1st time)
>> disp(['Perimeter of parallelogram = ' num2str(perim)]);

Perimeter of parallelogram = 6.8284

Note: 1) we are required to reassign vertices to “x” and”y”
2) the result is always assigned to a variable called “perim”

>> x = x2;  y=y2;  PerimScript;  % Find perimeter of octagon.
>> perimScript % Perform calculate (2nd time)
>> disp(['Perimeter of parallelogram = ' num2str(perim)]);

Perimeter of octagon = 3.0615
```
Using script to calculate perimeter (2)

>> % Recalculate perimeter of parallelogram.
>> x = x1;  y=y1;  % Assign x and y values (used in script).
>> perimScript     % Perform calculation (3rd time)
>> disp(['Perimeter of parallelogram = ' num2str(perim)]);

Perimeter of parallelogram = 3.0615

This is clearly wrong (it is different than first calculation, 6.8284). Why?

Pertinent lines of script file

```matlab
%% perimScript.m
... commands not shown
x1=[x x(1)];    % Repeat first point of x at end of vector
y1=[y y(1)];    % ... and for y
... commands not shown
```

The values of x1 and y1 are redefined in script!
When we called script for octagon (2nd time) , x1 and y1 are changed!!
When we call the 3rd time we get incorrect result!!
Functions

- For our purposes a function is a special MATLAB file that does a calculation(s) and returns a result(s)
- It is different from a script file:
  - it returns a result
  - it can’t accidentally change other variables that you are using in the MATLAB workspace
Anatomy of a Function

Keyword “function” tells us this is a function file (called “perim.m”)
Output variable(s) name(s); in this case only one: the variable “p”
Function name; also name of “.m” file (i.e., “perimFunction.m”)

```
function p = perimFunction(xvals,yvals)
% This function calculates perimeter of a polygon defined by
% vertices in xvals and yvals.

x1=[xvals xvals(1)];    % Form augmented vector
y1=[yvals yvals(1)];

xd=diff(x1);            % Calculate differences
yd=diff(y1);

sqrd = xd.^2+yd.^2;     % Square them
sqrtSqd=sqrt(sqrd);     % Find distances

p=sum(sqrtSqd);         % Find perimeter and assign to variable "p"
```

Input variables

```
x1=[xvals xvals(1)];   % Form augmented vector
y1=[yvals yvals(1)];

xd=diff(x1);           % Calculate differences
yd=diff(y1);
```

Calculations

```
sqrd = xd.^2+yd.^2;    % Square them
sqrtSqd=sqrt(sqrd);    % Find distances

p=sum(sqrtSqd);        % Find perimeter and assign to variable "p"
```

Output variable (must be same variable name as is in function declaration)
Variables in function are local to function alone.
Using Functions

Now we can call function as if it was built in:

```matlab
%% Redefine two polygons
x1 = [0 1 3 2]; y1 = [0 1 1 0]; % Parallelogram
x2 = 0.5*cosd(theta); y2 = 0.5*sind(theta); % Octagon

p1 = perimFunction(x1,y1) % Perimeter of parallelogram
p1 =
    6.8284

p2 = perimFunction(x2,y2) % Perimeter of octagon
p2 =
    3.0615

perimFunction(x1,y1) % Recalculate parallelogram
ans =
    6.8284
```

Note: 1) we didn’t need to reassign vertices to “x” and “y”
2) result can be assigned to any variable, not just “perim”

This time we got the correct answer!

A trick: if you add comments to the very top of the function file, they will appear if you ask for documentation (i.e., “>>doc perim”).
MATLAB Function Caveats

- Function name may not be identical to a variable name
- Function name (and file names) may have no spaces
- Function must be in MATLAB directory so MATLAB can find it (or in "MATLAB path").
- You shouldn’t run function from debugger – call it from the command line (or another function).
- If you edit a function, you must save the file before the changes will take effect in subsequent calls.

**If you edit a function, you must save the file before the changes will take effect in subsequent calls!**
Coordinate transformations

- Robot arm: transformation between joint angles and position of robot
- In Lab after break: we will use coordinate transformations to move around objects.

From this... ... to this
Trig review – basic functions

\[ x_0 = \ell \cos(\theta) \quad \cos(\theta) = \frac{x_0}{\ell} \]
\[ y_0 = \ell \sin(\theta) \quad \sin(\theta) = \frac{y_0}{\ell} \]
\[ \tan(\theta) = \frac{y_0}{x_0} \]

Inverse functions

\[ \theta = \arccos\left(\frac{x_0}{\ell}\right) \]
\[ \theta = \arcsin\left(\frac{y_0}{\ell}\right) \]
\[ \theta = \arctan\left(\frac{y_0}{x_0}\right) \]

But, be careful with arctangent…
Trig review – arctangents

Consider a point in the first quadrant, \((x_0, y_0) = (6,3)\)

\[
\tan(\theta) = \frac{y_0}{x_0} = \text{slope of line}
\]

\[
\gg \ \text{atan}(6/3) \times 180/\pi
\]
\[
\text{ans} = 63.4349
\]

\[
\gg \ \text{atan2}(6,3) \times 180/\pi
\]
\[
\text{ans} = 63.4349
\]

Now consider a point in the third quadrant, \((x_0, y_0) = (-6,-3)\).

\[
\gg \ \text{atan}(-6/-3) \times 180/\pi
\]
\[
\text{ans} = 63.4349
\]

\[
\gg \ \text{atan2}(-6,-3) \times 180/\pi
\]
\[
\text{ans} = -116.5651
\]
Trig functions in MATLAB

- \( \cos(\theta), \sin(\theta), \tan(\theta) \): assume angle, \( \theta \), is radians
- \( \cosd(\theta), \sind(\theta), \tand(\theta) \): assume angle, \( \theta \), is in degrees
- \( \arccos(z), \arcsin(z), \arctan(z) \): return an angle in radians
- \( \arccosd(z), \arcsind(z), \arctand(z) \): return an angle in degrees
- \( \text{atan2}(y,x) \): returns an angle in radians.
- \( \text{atan2d}(y,x) \): returns an angle in degrees.
Trig review – Pythagoras

\[ x_0^2 + y_0^2 = \ell^2, \quad \text{Pythagoras' theorem} \]

\[ x_0 = \ell \cos(\theta) \quad y_0 = \ell \sin(\theta) \]

\[ \ell^2 \cos^2(\theta) + \ell^2 \sin^2(\theta) = \ell^2 \]

\[ \cos^2(\theta) + \sin^2(\theta) = 1 \]
Rotation of a point (algebraic) 
(rotation around the origin)

\[
x = \ell \cos(\varphi), \quad y = \ell \sin(\varphi)
\]
\[
x' = \ell \cos(\theta + \varphi), \quad y' = \ell \sin(\theta + \varphi)
\]

Trig identities:
\[
\cos(\theta + \varphi) = \cos(\varphi)\cos(\theta) - \sin(\varphi)\sin(\theta)
\]
\[
\sin(\theta + \varphi) = \cos(\varphi)\sin(\theta) + \sin(\varphi)\cos(\theta)
\]

\[
x' = \ell \cos(\theta + \varphi) = \ell \cos(\varphi)\cos(\theta) - \ell \sin(\varphi)\sin(\theta) = x \cos(\theta) - y \sin(\theta)
\]
\[
y' = \ell \sin(\theta + \varphi) = \ell \cos(\varphi)\sin(\theta) + \ell \sin(\varphi)\cos(\theta) = x \sin(\theta) + y \cos(\theta)
\]
Rotation of a point (2x2 matrix)
(rotation around the origin)

\[
x' = \cos(\theta)x - \sin(\theta)y
\]
\[
y' = \sin(\theta)x + \cos(\theta)y
\]

\[
\begin{bmatrix}
x' \\
y'
\end{bmatrix} = \begin{bmatrix}
\cos(\theta) & -\sin(\theta) \\
\sin(\theta) & \cos(\theta)
\end{bmatrix} \begin{bmatrix}
x \\
y
\end{bmatrix}
\]

\[
\begin{bmatrix}
\cos(\theta) & -\sin(\theta) \\
\sin(\theta) & \cos(\theta)
\end{bmatrix} = \text{rotation matrix} = R
\]

\[
\begin{bmatrix}
x' \\
y'
\end{bmatrix} = R \begin{bmatrix}
x \\
y
\end{bmatrix}
\]
MATLAB (1):
Create a function that takes as input an “x” value, a “y” value and an angle (in degrees), and returns rotated values, xp and yp. Use the following shell (you don’t need to put in comments) and save it as “rot2.m.” Work as a group, discussing as you go – everybody should understand how it works.

```matlab
function [xp, yp] = rot2(x, y, theta) % Function rot2.
% Takes as input x, y and theta. Returns rotated values, xp, yp.

xy = [x; y]; % Make column vector
R = xxxx; % Make rotation matrix. Angle is in degrees.
xp_yp = xxxx; % Calculate x' and y'

xp = xxxx; % Find xp and yp.
yp = xxxx;
```

Test it from MATLAB command prompt.
```
>> [xnew, ynew] = rot2(1, 1, 45)
xnew =
   0
ynew =
  1.4142
```
function [xp, yp] = rot2(x,y,theta,tx,ty) % Function rot2.
% Takes as input x, y, theta2.
% Returns rotated values, xp, yp.

xy = [x;y]; % Make column vector called xy

% Rotation matrix
R = [cosd(theta) -sind(theta); sind(theta) cosd(theta)];

xyp = R*xy; % Perform rotation

xp=xyp(1); % Get rotated values in xp and yp.
yp=xyp(2);
Translation of a point (3x3)

\[ x' = x + t_x, \quad y' = y + t_y \]

How can we express this with matrices?

\[
\begin{bmatrix}
  x' \\
  y' \\
  1
\end{bmatrix}
= 
\begin{bmatrix}
  1 & 0 & t_x \\
  0 & 1 & t_y \\
  0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix}
\]

\[
\begin{bmatrix}
  1 & 0 & t_x \\
  0 & 1 & t_y \\
  0 & 0 & 1
\end{bmatrix}
= \text{translation matrix} = \mathbf{M}
\]

\[
\begin{bmatrix}
  x' \\
  y' \\
  1
\end{bmatrix}
= \mathbf{M}
\begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix}
\]

We can combine a rotation (first) with a translation (second).

This can be done in one operation if we redefine the rotation matrix to be 3x3

\[
\begin{bmatrix}
\cos(\theta) & -\sin(\theta) & 0 \\
\sin(\theta) & \cos(\theta) & 0 \\
0 & 0 & 1
\end{bmatrix} = \text{rotation matrix} = R
\]

\[
x' = \cos(\theta)x - \sin(\theta)y
\]

\[
y' = \sin(\theta)x + \cos(\theta)y
\]
Rotation and translation (1)  
(rotation around origin)

We can combine a rotation (first) with a translation (second).

\[
\begin{bmatrix}
x' \\
y' \\
1
\end{bmatrix} = \mathbf{M} \begin{bmatrix}
x \\
y \\
1
\end{bmatrix} = \mathbf{MR}
\]

\[
\begin{bmatrix}
x' \\
y' \\
1
\end{bmatrix} = \begin{bmatrix}
1 & 0 & t_x \\
0 & 1 & t_y \\
0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
\cos(\theta) & -\sin(\theta) & 0 \\
\sin(\theta) & \cos(\theta) & 0 \\
0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
x \\
y \\
1
\end{bmatrix}
\]

\[
\begin{bmatrix}
x' \\
y' \\
1
\end{bmatrix} = \begin{bmatrix}
\cos(\theta) & -\sin(\theta) & t_x \\
\sin(\theta) & \cos(\theta) & t_y \\
0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
x \\
y \\
1
\end{bmatrix}
\]

Rotation and translation (2)
(rotation around origin)

\[
\begin{bmatrix}
\cos(\theta) & -\sin(\theta) & t_x \\
\sin(\theta) & \cos(\theta) & t_y \\
0 & 0 & 1
\end{bmatrix} = \text{transformation matrix} = T
\]

\[
\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix}
\cos(\theta) & -\sin(\theta) & t_x \\
\sin(\theta) & \cos(\theta) & t_y \\
0 & 0 & 1
\end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = T \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}
\]

\[
x' = \cos(\theta)x - \sin(\theta)y + t_x
\]

\[
y' = \sin(\theta)x + \cos(\theta)y + t_y
\]

Extension to many points

If we have several points \((x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots (x_n, y_n)\), we can still use the transformation matrix to rotate them all through an angle \(\theta\) and translate by \(t_x\) and \(t_y\).

\[
\begin{bmatrix}
   x'_1 \\
   x'_2 \\
   x'_3 \\
   \vdots \\
   x'_n
\end{bmatrix} =
\begin{bmatrix}
   \cos(\theta) & -\sin(\theta) & t_x \\
   \sin(\theta) & \cos(\theta) & t_y \\
   0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
   x_1 \\
   x_2 \\
   x_3 \\
   \vdots \\
   x_n
\end{bmatrix}
\]

\[
\begin{bmatrix}
   y'_1 \\
   y'_2 \\
   y'_3 \\
   \vdots \\
   y'_n
\end{bmatrix} =
\begin{bmatrix}
   \cos(\theta) & -\sin(\theta) & t_x \\
   \sin(\theta) & \cos(\theta) & t_y \\
   0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
   y_1 \\
   y_2 \\
   y_3 \\
   \vdots \\
   y_n
\end{bmatrix}
\]

\[
x'_1 = \cos(\theta) x_1 - \sin(\theta) y_1 + t_x \\
y'_1 = \sin(\theta) x_1 + \cos(\theta) y_1 + t_y
\]

etc…

If the points define a shape, we can rotate and translate that shape with a single matrix multiplication.
MATLAB (2):

Start with two vectors of the same length

```
>> x=1:3
x =
    1     2     3
>> y=5:7
y =
    5     6     7
```

Create an array with x in top row, y in middle row and ones on bottom row.

```
z =
    1     2     3
    5     6     7
    1     1     1
```

Hint:
```
>> ones(size(x))
ans =
    1     1     1
```
MATLAB (3):

Create a function that takes as input a row vector of x coordinates, a vector of y coordinates, an angle, tx, and ty and returns the rotated and translated vectors. Save as “rotTrans.m.” Start by discussing what needs to be done. Use a piece of paper to layout the steps (in English). Once everybody agrees on the procedure, convert to code.

The goal is not speed, but to work together so that everybody understands.

```matlab
function [xp, yp] = rotTrans(x,y,theta,tx,ty)  % Function rotTrans.
% Takes as input x, y, theta, tx and ty.  x and y can be vectors.
% Returns rotated and translated values, xp, yp.

% Finish this . . . Start by discussing what needs to be done.
% Use a piece of paper to layout the steps (in English).
% Once everybody agrees on the procedure, convert to code.
```
Test “rotTrans”:

>> [xp,yp]= rotTrans(1,1,45,0,0)  % Test case
xp =
    0
yp =
    1.4142

>> [xp,yp]= rotTrans(1,1,45,1,-1)  % Another test case
xp =
    1
yp =
    0.4142

>> % Vector input test case
>> [xp,yp]= rotTrans([1 0],[1 0],45,1,-1)
xp =
    1     1
yp =
    0.4142   -1.0000

Email the file to group members – we will use this later.
function [xp, yp] = rotTrans(x,y,theta,tx,ty) \% Function rotTrans. 
\% Takes as input x, y, theta, tx and ty. 
\% Returns rotated and translated values, xp, yp.

xy = [x;y;ones(size(x))]; \% Make column vector

T=[cosd(theta) -sind(theta) tx; 
   sind(theta) cosd(theta) ty; 
   0 0 1];

xyp = T*xy; \% Do transformation

xp = xyp(1,:); \% X values in first row
yp = xyp(2,:); \% Y values in second row
Other transformations...
(that we won’t be using)

Scaling

Matrix form:

\[
\begin{bmatrix}
    x' \\
    y'
\end{bmatrix} =
\begin{bmatrix}
    s_x & 0 \\
    0 & s_y
\end{bmatrix}
\begin{bmatrix}
    x \\
    y
\end{bmatrix}
\]

Shearing

x-direction shear of unit cube.

\[
SH_x =
\begin{bmatrix}
    1 & sh_x & 0 \\
    0 & 1 & 0 \\
    0 & 0 & 1
\end{bmatrix}
\]

Composite transformations (e.g., rotation about an arbitrary point $(x_1, y_1)$):

- Translate to origin
- Rotate
- Translate back

\[
T(x_1, y_1) \cdot R(\theta) \cdot T(-x_1, -y_1) = \begin{bmatrix} 1 & 0 & x_1 \\ 0 & 1 & y_1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_1 \\ 0 & 1 & -y_1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & x_1(1-\cos \theta) + y_1 \sin \theta \\ \sin \theta & \cos \theta & y_1(1-\cos \theta) - x_1 \sin \theta \\ 0 & 0 & 1 \end{bmatrix}
\]