Random Numbers

Let M, N and P define a $M \times N \times P$ array.

```
rand([M, N, P], 'datatype')    % uniformly distributed random numbers between 0 & 1
randn([M, N, P], 'datatype')   % normally distributed random numbers
```

Random numbers between a & b

```
rand([M, N, P]) * (b - a) + a  % uniformly distributed random numbers
randn([M, N, P]) * (b - a) + a % normally distributed random numbers
randi([a, b], [M, N, P])       % uniformly distributed random integers
```

Data Types

Name	Description	Range
`logical`	boolean values	`0 & 1`
`uint8`	unsigned 8-bit integers	`0 ... 2^8`
`int8`	unsigned 8-bit integers	`-2^8 ... 2^8`
`single`	single precision "real" numbers	`-realmax ... realmax`
`double`	double precision "real" numbers	`-realmax ... realmax`

(un)signed 16, 32, 64-bit storage for integer data is created by appending the size to "(u)int".

Operators and Special Characters

Arithmetic Operators

MATLAB uses standard mathematical symbols: `+`, `-`, `*`, `/`, `^`.
For element-wise operations, prepend the mathematical operator with a dot (`.`).

Relational Operators

Symbol	Role
`==`	Equal to
`~=`	Not equal to
`>`	Greater than
`>=`	Greater than or equal to
`<`	Less than
`<=`	Less than or equal to

Logical Operators

Symbol	Role
`&`	logical `AND`
`|`	logical `OR`
`~`	logical `NOT`

Special Characters

Symbol	Role
`,`	Separator for row elements
`:`	Index all subscripts in array dimension; create unit-spaced vector
`;`	Separator for column elements; suppress output
`( )`	Operator precedence
`[ ]`	Array creation, multiple output argument assignment
`%`	Comment
`""`	String constructor
`~`	Argument placeholder (suppress specific output)
`=`	Assignment

Special Arrays

```
zeros(M, N) % zero array
false(M, N) % logical false array
```

Array Comparisons

```
A = rand(M, N); % random array
mask = A > 0.5; % logical array, true (1) if: >0.5 and false (0) if: <=0.5
```

Other Functions

```
who                     % list workspace variables
who -file <mat file>    % list variables in .mat file
pause(x)                % pause procedure for x seconds
```

Image Processing

Finding Area

```
f = figure;                                         % create a figure object
imshow('file.png');                                 % display image
p = drawpolygon(f.Children)                         % trace polygon on image
cP = p.Position;                                    % n by 2 array of (x, y) coordinates
areaPxSquared = polyarea(cP(:, 1), cP(:, 2));       % area [px^2]
l = drawline(f.Children)                            % trace scale bar on image
cL = l.Position;                                    % 2 by 2 array of (x, y) coordinates
scalePx = sqrt((cL(2, 1) - cL(1, 1))^2 + ...
               (cL(2, 2) - cL(1, 2))^2);            % scale length [px]
mPerPx = actualScaleLength / scalePx;               % [m] per [px]
mSquaredPerPxSquared = mPerPx^2;                    % [m^2] per [px^2]
areaMSquared = mSquaredPerPxSquared * areaPxSquared; % area [m^2]
```

Geolocation

```
longitudes = [...];                                             % e.g. 153.02
latitudes = [...];                                              % e.g. -27.46
origin = [mean(longitudes), mean(latitudes)];                   % arbitrary origin
radius = 6373.6;                                                % radius of Earth
circumference = 2 * pi * radius;                                % circumference of Earth
kmPerDegLatitude = circumference / 360;
kmPerDegLongitude = kmPerDegLatitude * cos(deg2rad(-27.5));     % near Brisbane
x = (longitudes - origin(1)) * kmPerDegLongitude;               % x coordinates
y = (latitudes - origin(2)) * kmPerDegLatitude;                 % y coordinates
plot(x, y, '.');                                                % plot locations
```

Images from Arrays

```
imshow(A)   % Display image
image(A)    % Display image, recommended if combining with other plots
```

Random Images

```
randi([0, 255], M, N, 'uint8');      % greyscale image
randi([0, 255], M, N, 3, 'uint8');   % colour image
```

Creating Colour Images by Modifying Array Entries

```
A = 255 * zeros(M, N, 3, 'uint8');  % black image
A = 255 * ones(M, N, 3, 'uint8');   % white image
% Access individual channels
rMask = A(:, :, 1);                 % red channel
gMask = A(:, :, 2);                 % green channel
bMask = A(:, :, 3);                 % blue channel
% Access specific region and change its colour to rgb(r, g, b)
A(x1:x2, y1:y2, 1) = r;             % modify red value of (x1:x2, y1:y2)
A(x1:x2, y1:y2, 2) = g;             % modify green value of (x1:x2, y1:y2)
A(x1:x2, y1:y2, 3) = b;             % modify blue value of (x1:x2, y1:y2)
```

Editing an Image from a File

```
theImage = imread('image.png');     % access image
% Mask a colour range to be modified
mask = theImage(:, :, 1) > r & theImage(:, :, 2) > g & theImage(:, :, 3) > b;
% Modify channels of selected colour range - accessing individual channels shown in previous section
rMask(mask) = rNew;                 % modify red value in masked image
gMask(mask) = gNew;                 % modify green value in masked image
bMask(mask) = bNew;                 % modify blue value in masked image
theNewImage(:, :, 1) = rMask;       % assign red mask to new array
theNewImage(:, :, 2) = gMask;       % assign green mask to new array
theNewImage(:, :, 3) = bMask;       % assign blue mask to new array
```

Create and Save an Animation

```
f = figure;
set(f, 'Visible', 'on');
video = VideoWriter('video.avi');   % create video object
open(video);                        % open video for write access
x = [...];                          % x values
y = [...];                          % y values
p = plot(x(1), y(1));               % create plot object
for i = 1:length(x)                 % iterate through each frame
    % Update plot object data
    p.XData = x(i);
    p.YData = y(i);
    hold on;                        % use if previous points should remain on figure
    drawnow;                        % update figure
    frame = getFrame;               % get snapshot of current axes
    writeVideo(video, frame)        % write frame to video
end
hold off                            % use if hold on was used
close(video);                       % close the file
```

Sound Processing

Create pure tone

```
f = 523.251;                        % frequency of note
Fs = 8192;                          % sampling rate
l = 1;                              % length of tone [s]
t = 0: 1 / Fs : l;                  % vector of evenly-spaced times to sample at
y = sin(2 * pi * f * t);            % sine wave sampled at t
```

Processing sounds

```
[y1 + y2]                           % combine y1 and y2 (must be the same dimension)
[y1; y2]                            % append y2 after y1
soundsc(y, Fs)                      % play sound
resample(y, Fs, Q)                  % resample sound at the new sampling rate: Fs / Q
Fs / 2                              % half speed
Fs * 2                              % double speed
audiowrite('audio.wav', y, Fs)      % write sound to audio.wav
```

Let y be a column vector

```
duration = length(y) / Fs;                  % duration of sound [s]
% equivalent methods for defining a time vector
t = 0 : 1 / Fs : duration;                  % using the colon operator
t = linspace(0, duration, length(y) + 1);   % using the linspace function
```

Random Walks

Initialisation

```
M = 50;                                             % number of particles
N = 200;                                            % number of steps
delta = 1;                                          % size of step
p = 0.5;                                            % probability of jumping left
A = zeros(N + 1, M);                                % initialise particles at zero
for i = 1:N                                         % iterate through each step
    r = rand(1, M);                                 % random probability for each particle
    leftMask = r < p;                               % mask left-moving particles
    A(i + 1, leftMask) = A(i, leftMask) - delta;    % move leftMask left
    A(i + 1, ~leftMask) = A(i, ~leftMask) + delta;  % move ~leftMask right
end
```

Animated Step vs Position Plot

```
p = plot(A(1:2, :), '.-');                  % create plot object
L = max(abs(A(:)));                         % maximum distance reached
axis([0 N -L L]);                           % set axis bounds
for i = 1:N                                 % iterate through each step
    for j = 1:M                             % iterate through each particle
```

```
        p(j).YData = A(1:i, j);             % update plot object data
        hold on                             % hold until all particles trees have been plotted
    end
    hold off                                % hold off for next iteration
    drawnow                                 % update figure
end
```

1D Animated Position Plot

```
p = plot(A(1, :), zeros(1, M), '.');        % create plot object
L = max(abs(A(:)));                         % maximum distance reached
axis([-L L -1 1]);                          % set axis bounds
for i = 1:N                                 % iterate through each step
    p.XData = A(i, :);                      % update plot object data
    drawnow                                 % update figure
end
```

Cellular Automata

```
C = 100;                                        % number of cells
N = 50;                                         % number of steps
A = false(N + 1, C);                            % initialise all positions to be empty
A(1, :) = rand(1, C) > 0.5;                     % use random initial state
% Manually define initial state using an explicit assignment
A(1, :) = [...];                                % length of row vector must be C
for i = 1:N                                     % iterate through each step
    P = A(i, :)                                 % centre cells
    % Wrap-around ghost boundary cell
    L = [P(C), P(1:C - 1)];
    R = [P(2:C), P(1)];
    % Dead boundary cell
    L = [0, P(1:C - 1)];
    R = [P(2:C), 0];
    % List of all 1D configurations
    C000 = (L == 0 & P == 0 & R == 0);
    C001 = (L == 0 & P == 0 & R == 1);
    C010 = (L == 0 & P == 1 & R == 0);
    C011 = (L == 0 & P == 1 & R == 1);
    C100 = (L == 1 & P == 0 & R == 0);
    C101 = (L == 1 & P == 0 & R == 1);
    C110 = (L == 1 & P == 1 & R == 0);
    C111 = (L == 1 & P == 1 & R == 1);
    liveMask = C000 | ... | ...;                % cellular automation rule(s)
    A(i + 1, liveMask) = 1;                     % set live cells for next generation
end
imshow(~A, 'InitialMagnification', 'Fit');      % Display black live cells
```