# 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 if: >0.5 and false 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
```

## 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(a:b, c:d, 1) = r;     % modify red value of (a:b, c:d)
A(a:b, c:d, 2) = g;     % modify green value of (a:b, c:d)
A(a:b, c:d, 3) = b;     % modify blue value of (a:b, c:d)
```

### Editing an Image

```
theImage = imread('image.png');     % access image
% Mask a colour range to be modified
mask = theImage(:, :, 1) > r & theImage(:, :, 2) > g & theImage(:, :, 3) > b;
```

### Create and Save an Animation

```
f = figure;
set(f, 'Visible', 'on');
video = VideoWriter('video.avi');   % create video object; write to video.avi
x = [...];                          % x values
y = [...];                          % y values
```

```
p = plot(x(1), y(1));               % create plot object
for i = 1:length(x)
    % 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]
% time vector using two different methods
t = 0 : 1 / Fs : duration;                  % using the colon operator
t = linspace(0, duration, length(y) + 1);   % using the linspace function
```

# Random Walks

## Initialisation

```
M = 10000;
N = 200;
delta = 1;
p = 0.5;
x = zeros(N + 1, M);
for i = 1:N
    r = rand(1, M);
    leftMask = r < p;
    x(i + 1, leftMask) = x(i, leftMask) - delta;
    x(i + 1, ~leftMask) = x(i, ~leftMask) + delta;
end
f = figure;
plot(x, '.-');
```