while loop

- a `while` loop repeats a block of code as long as a logical condition is true
  - unlike a for loop
    - there is no loop variable
    - the number of times that the loop runs is not necessarily determined ahead of time

while loop

```
while
```

logical_condition

loop body: a sequence of MATLAB statements

```
end
```

if logical_condition is true then the loop body is run once

after the loop body is run, the loop restarts by checking the logical_condition

```
% repeat a loop until the user inputs 'y'

repeat = 1;
while (repeat)
  %
  % some code here that you want to repeat
  %

  % ask the user if they want to repeat again
  answer = input('Continue? (y / n)');
  repeat = strcmp(answer, 'y');
end
```

while loop: infinte loops

- observe that it is very easy to create an infinite loop using a **`while`** loop
  - you must ensure that whatever happens in the loop body eventually causes the logical condition to become false

- if you encounter an infinite loop in your program you can press **`Ctrl + c`** to stop your program
  - unfortunately this stops your entire program and not just your loop

```
% infinite loop example

repeat = 1;
while (repeat)
  %
  % some code here that you want to repeat
  %

  % ask the user if they want to repeat again
  answer = input('Continue? (y / n)');

  % comment out next line
  % repeat = strcmp(answer, 'y');
end
```

while loop: computing square root

- Hero's method
  - named after Hero of Alexandria (1st century Greek mathematician)
- to compute the square root of s

1. choose a starting value x_0
2. let x_1 be the average of x_0 and s/x_0
3. let x_2 be the average of x_1 and s/x_1
4. let x_3 be the average of x_2 and s/x_2, and so on

- how do you know when to stop?

while loop: computing square root

- Hero's method can be described mathematically as

$$x_0 \approx \sqrt{s}$$

$$x_{i+1} = \frac{1}{2}\left(x_i + \frac{s}{x_i}\right)$$

$$\sqrt{s} = \lim_{i \to \infty} x_i$$

```
% compute the square root of s

epsilon = 1e-9;
delta = Inf;
x = 0.5 * x;
while abs(delta) > epsilon
  xi = mean([x, s / x]);
  delta = xi – x;
  x = xi;
end
```

while loop: roots of functions

- Hero's method is a special case of Newton's method for finding roots of a real-valued function
- given a real-valued function

$$f(x)$$

find

$$x \text{ such that } f(x) = 0$$

while loop: Newton's method

- Newton's method can be described as

1. start with an initial estimate of the root x_0
2. $i = 0$
3. while $|f(x_i)| > \epsilon$

   $$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

   $$i = i + 1$$

```matlab
function [ root, xvals ] = newton(x0, epsilon)
%NEWTON Newton's method for x^2 - 1
%   ROOT = NEWTON(X0, EPSILON) finds a root of f(x) = x^2 - 1 using
%   Newton's method starting from an initial estimate X0 and a tolerance EPSILON
%
%   [ROOT, XVALS] = NEWTON(X0, EPSILON) also returns the iterative estimates
%   in XVALS

xvals = x0;
xi = x0;
while abs(f(xi)) > epsilon
   xj = xi - f(xi) / fprime(xi);
   xi = xj;
   xvals = [xvals xi];
end
root = xi;

end

function [ y ] = f(x)
y = x * x - 1;
end
```

local function: usable only inside **`newton.m`**

```matlab
function [ yprime ] = fprime(x)
yprime = 2 * x;
end
```

local function: usable only inside **`newton.m`**

while loop: Newton's method

- what happens if you call Newton's method with:
  - $x_0 = 1$
  - $x_0 = -1$
  - $x_0 = 2$
  - $x_0 = -2$
  - $x_0 = 100$
  - $x_0 = -100$
  - $x_0 = 0$
  - $x_0 = 1e - 6$

while loop: Newton's method

- the main idea in Newton's method
  - we cannot easily find a root of $f(x)$
  - we can approximate $f(x)$ around x_i by using the tangent line at x_i
  - we can easily compute the root of the tangent line as the x-intercept of the tangent line
  - we can use the root of the tangent line as an improved estimate of the root of $f(x)$

while loop: Newton's method

- the main idea in Newton's method
  - we cannot easily find a root of $f(x)$
  - we can approximate $f(x)$ around x_i by using the tangent line at x_i
  - we can easily compute the root of the tangent line as the x-intercept of the tangent line
  - we can use the root of the tangent line as an improved estimate of the root of $f(x)$
- see **`plotnewton.m`**

Nested loops

- a nested loop is a loop inside a loop
- often encountered when working with arrays of values
- consider matrix-vector multiplication

$$Ax = b$$

$$\begin{bmatrix} A(1,1) & A(1,2) & \cdots & A(1,n) \\ A(2,1) & A(2,2) & & A(2,n) \\ \vdots & & \ddots & \vdots \\ A(m,1) & A(m,2) & \cdots & A(m,n) \end{bmatrix} \begin{bmatrix} x(1) \\ x(2) \\ \vdots \\ x(m) \end{bmatrix} = \begin{bmatrix} b(1) \\ b(2) \\ \vdots \\ b(m) \end{bmatrix}$$

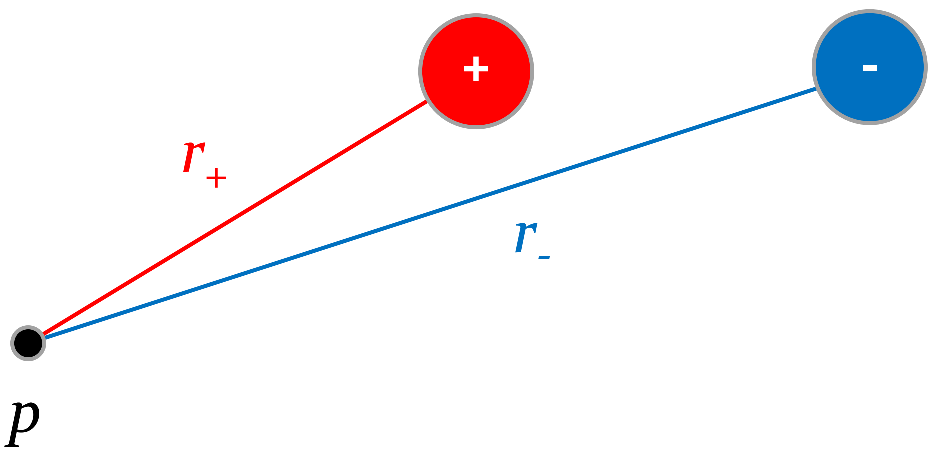
- to compute b we need to compute $m \times n$ multiplications

```
% for some (m x n) matrix A and (n x 1) vector x

[m, n] = size(A);

b = zeros(m, 1);
for row = 1:m
  for col = 1:n
    b(row) = b(row) + A(row, col) * x(col);
  end
end
```

Nested loops: dipole electric potential

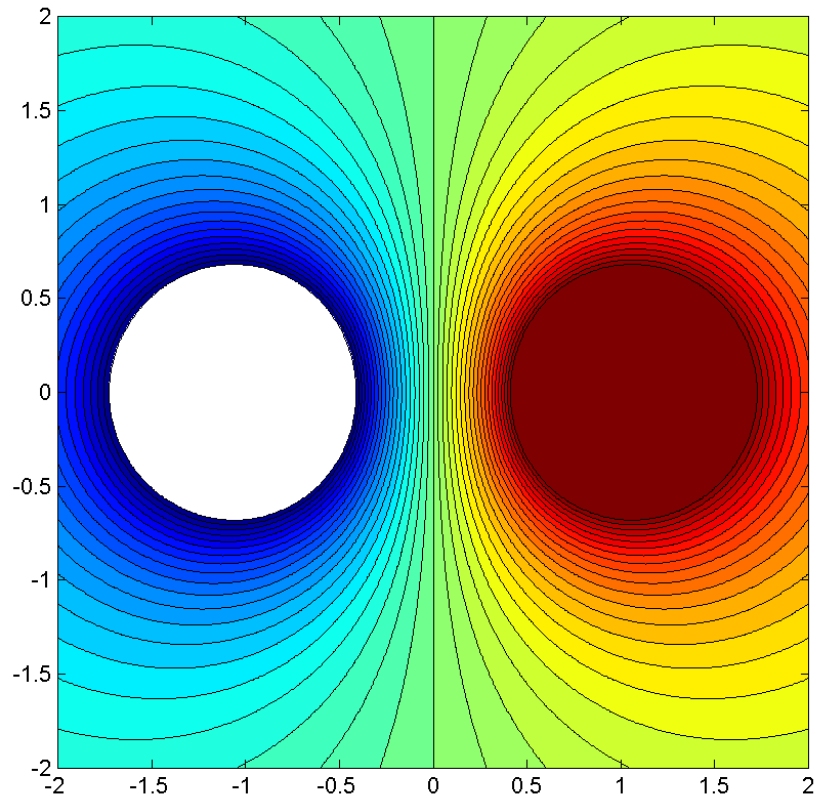
- the dipole electric potential at some point p is proportional to:

$$V \propto \frac{q_+}{r_+} + \frac{q_-}{r_-}$$



Nested loops: dipole electric potential

- the lines of equipotential



Nested loops: dipole electric potential

- to draw the lines of equipotential, we need to compute the dipole electric potential at discrete points (x_i, y_i)
- we can make a grid of equally spaced points using the **`meshgrid`** function

```
>> [X, Y] = meshgrid(-2:2);
>> [X, Y] = meshgrid(-2:2, 0:4);
```

```matlab
%% electric dipole potential

% charge 1 (negative)
p1 = [-0.995; 0];
q1 = -1;

% charge 2 (positive)
p2 = [0.995; 0];
q2 = 1;

% the grid to compute the potential on
[X, Y] = meshgrid([-2:0.01:2]);

% the electric potential
V = zeros(size(X));
for row = 1:size(X, 1)
    for col = 1:size(X, 2)
        p = [X(row, col); Y(row, col)];
        v1 = q1 / norm(p - p1);
        v2 = q2 / norm(p - p2);
        V(row, col) = v1 + v2;
    end
end

% show the electric potential
c = -1:0.05:1;
contourf(X, Y, V, c)
```