Problem Find the square root of N (i.e. solve $x^2 = N$)

One possible approach is the Babylonian Method for finding square roots. First we rearrange the equation as follows:

$$x^2 = N \Rightarrow x = \frac{N}{x} \Rightarrow \frac{1}{2}x = \frac{1}{2}\frac{N}{x} \Rightarrow$$

$$x - \frac{1}{2}x = \frac{1}{2}\frac{N}{x} \Rightarrow x = \frac{1}{2}x + \frac{1}{2}\frac{N}{x} \Rightarrow x = \frac{1}{2}\left(x + \frac{N}{x}\right)$$

Then we pick an arbitrary first guess (x_0) and repeatedly refine it by applying:

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

```
function [ x ] = Babylon( N )
%BABYLON Uses Babylonian technique to find the square root of a value.
% Inputs: N = value of interest
% Outputs: x = square root of N

xold = N;
for k = 1 : 20
   x = 0.5*(xold + N/xold);
   change = abs(x - xold);
   fprintf ('x(%d) = %f, absolute change = %f\n', k, x, change );
   if change < 1e-6
      fprintf ('Babylonian technique has converged.\n');
      return;
   end
   xold = x; %get ready for next iterations
end;

error ('Babylonian technique has not converged.');

end
```

When $x_0 = N$ the approach rapidly converges on the solution:

```
>> Babylon (7)
x(1) = 4.00000000, absolute change = 3.00000000
x(2) = 2.87500000, absolute change = 1.12500000
x(3) = 2.65489130, absolute change = 0.22010870
x(4) = 2.64576704, absolute change = 0.00912426
x(5) = 2.64575131, absolute change = 0.00001573
x(6) = 2.64575131, absolute change = 0.00000000
Babylonian technique has converged.
ans =
   2.6458
```

Iteration is terminated when the approximate error = $E_A = | x_i - x_{i-1} | < 0.000001$

It is assumed that when the changes become small the error becomes correspondingly small.

In general this approach does not guarantee that the final answer will be within the chosen tolerance of the correct answer (although for this particular algorithm I believe that it can be proved this will always be the case).

When x_0 = *0.1* the initial step gives an x_1 that is further away from the correct answer and convergence takes a little longer:

```
>> Babylon (7)
x(1) = 35.05000000, absolute change = 34.95000000
x(2) = 17.62485735, absolute change = 17.42514265
x(3) = 9.01101184, absolute change = 8.61384551
x(4) = 4.89391957, absolute change = 4.11709227
x(5) = 3.16213296, absolute change = 1.73178661
x(6) = 2.68791431, absolute change = 0.47421865
x(7) = 2.64608200, absolute change = 0.04183231
x(8) = 2.64575133, absolute change = 0.00033067
x(9) = 2.64575131, absolute change = 0.00000002
Babylonian technique has converged.
ans =
   2.6458
```

This particular approach will always converge provided that x_0 is not zero (negatives values for x_0 give the negative root). This is not always the case.

Two key issues: Will iteration converge? How fast will it converge?

Iteration and Casio Calculators:

Enter the right hand side of the iterative formula using memory A as *x*.

Example: 0.5 * (ALPHA A + 7 / ALPHA A)

Enter CALC followed by initial value and '='

Example: CALC 7 =

Enter CALC followed by ANS (use last answer as value for A) and '='

Repeat last step until answer stops changing.

There is absolutely no guarantee that algebraic manipulation will produce an iterative formula that works.

$$x^2 = N \implies x^2 + x = N + x \implies x = N + x - x^2$$

$$x_{i+1} = N + x_i - x_i^{\,2}$$

x(1) = -35.000000, absolute change = 42.000000

x(2) = -1253.000000, absolute change = 1218.000000

x(3) = -1571255.000000, absolute change = 1570002.000000

x(4) = -2468843846273.000000, absolute change = 2468842275018.000000

x(5) = -6095189937282529100000000.000000,

absolute change = 6095189937280060500000000.000000

"old" Notes: Fixed point iteration "almost useless as a real problem solving method"

Something more systematic is required.

Newton –Raphson

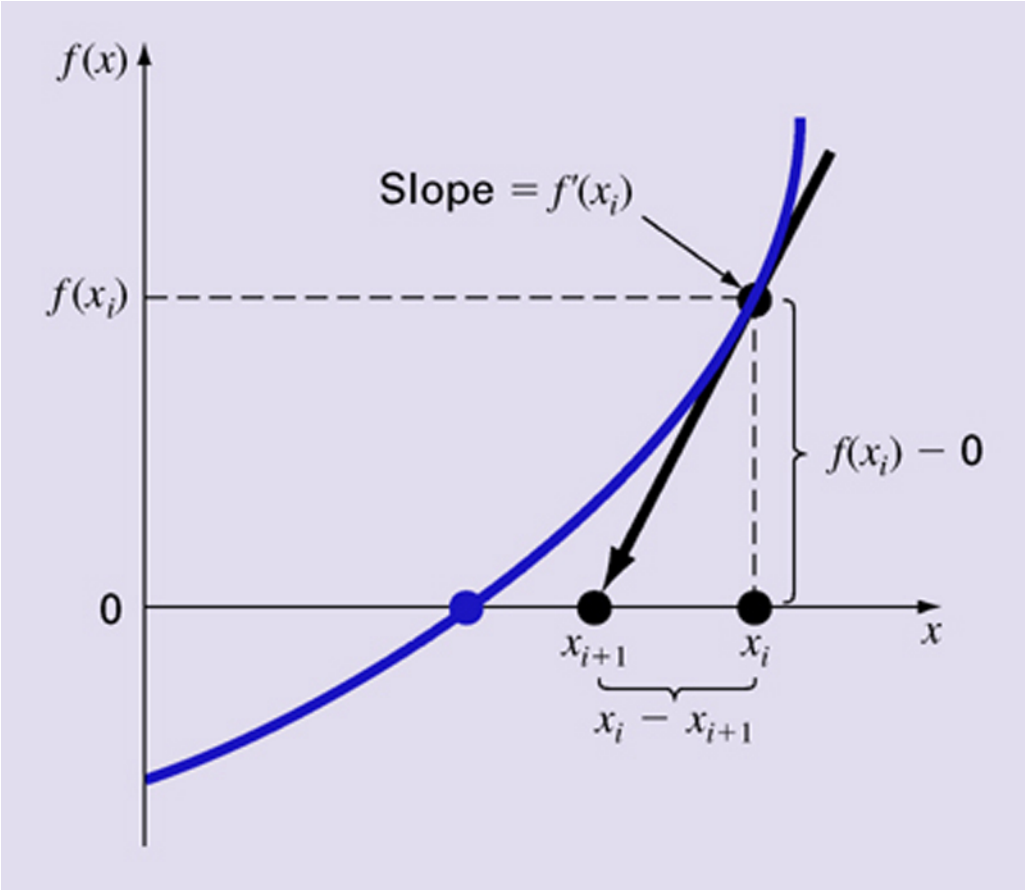
A root-finding technique (solves $f(x) = 0$)

Iterative Formula:

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

Popular, efficient

Quadratic convergence (number of significant figures approximately doubles with each iteration)



Newton-Raphson Disadvantages:

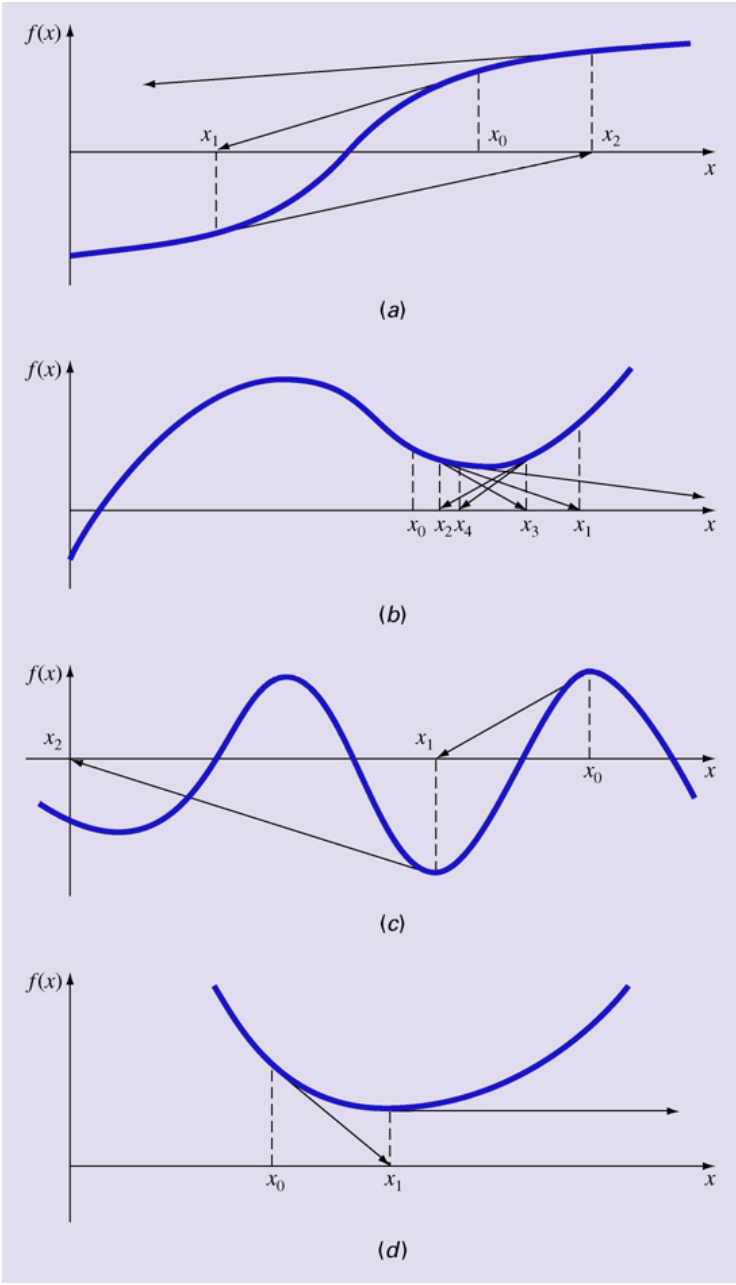
Requires derivative of function

Can diverge (e.g. case (a) on right)

Can oscillate between values of *x*

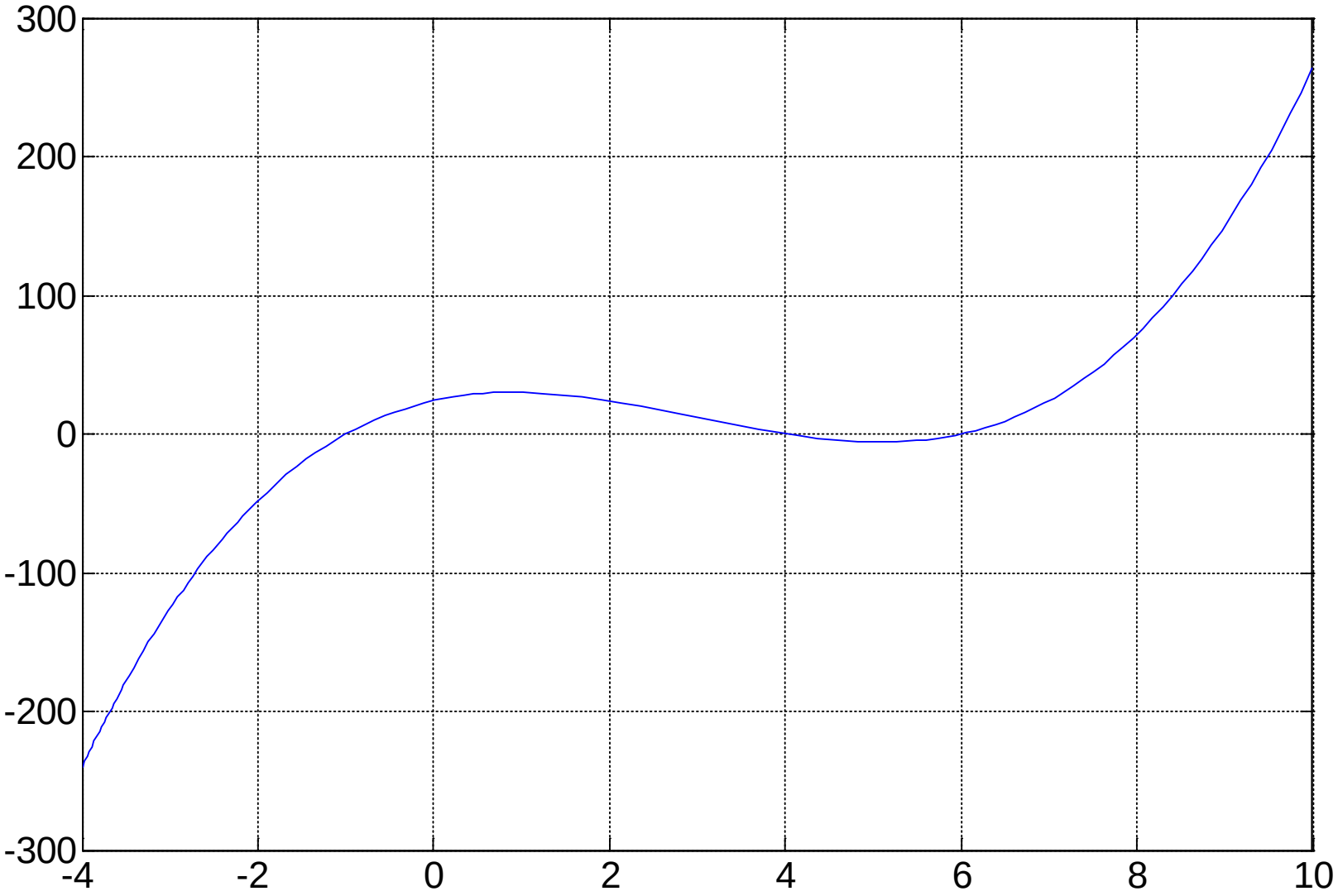
Can move away from root of interest (e.g. case (c) on right)

Fails if $f'(x) = 0$ (e.g. case (d) on right)



```
function [xr] = Newton (f, fp, x0, Edes, display)
% NEWTON  Finds a root by performing a Newton-Raphson search.
% Inputs: f = a function of one variable
%         fp = derivative of function
%         x0 = initial guess at root
%         Edes = tolerance in x
%               (function stops when change in x <= Edes)
%         display = display option (0 = no output, 1 = output, defaults to 0)
% Outputs: xr = estimate of root

if nargin < 5; display = 0; end

if display
  fprintf ('  step       xr          yr          Ea\n');
end
```

```
xold = x0;

for k = 1:100  % 100 max iterations

   xr = xold - f(xold)/fp(xold); yr = f(xr);  Ea = abs(xr - xold);

   if display
      fprintf ('%5d   %12.6f %12.6f %12.6f\n', k, xr, yr, Ea);
   end

   if Ea <= Edes || yr == 0  % error acceptably small or direct hit
      return;
   end

   xold = xr;

end

error ('Newton-Raphson search has not converged.');

end
```



Find a solution of $f(x) = x^3 - 9x^2 + 14x + 24$

```
>> f = @(x) x^3 - 9 * x^2 + 14 * x + 24;
>> fp = @(x) 3 * x^2 - 18 * x + 14;

>> p = [ 1 -9 14 24 ];
>> roots(p)
ans =
    6.0000
    4.0000
   -1.0000

>> Newton (f, fp, 2, 1e-6, 1)
  step        xr            yr            Ea
    1       4.400000    -3.456000     2.400000
    2       3.914607     0.875186     0.485393
    3       3.998033     0.019678     0.083427
    4       3.999999     0.000012     0.001965
    5       4.000000     0.000000     0.000001
    6       4.000000     0.000000     0.000000
ans =
    4.0000
```

Problem: Find the square root of *N* (i.e. solve x^2 –N = 0) using the Newton-Raphson approach.

$$f(x) = x^2 - N$$

$$f'(x) = 2x$$

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} = x_i - \frac{x_i^{\,2} - N}{2x_i}$$

$$= \frac{1}{2x_i}\left(2x_i^{\,2} - x_i^{\,2} + N\right) = \frac{1}{2}\left(x_i + \frac{N}{x_i}\right)$$

This is exactly the same as the Babylonian method .

The Babylonian method is fact a Newton –Raphson search (which is why it works so well)