Based on these observations, the use of tolerance and converging criteria must be done very carefully and in the context of the program that uses them.

2.2. Newton's Method for Root Approximation

Newton's (or Newton–Raphson) method can be used to approximate the roots of any linear or non–linear equation of any degree. This is an iterative (repetitive procedure) method and it is derived with the aid of Figure 2.1.

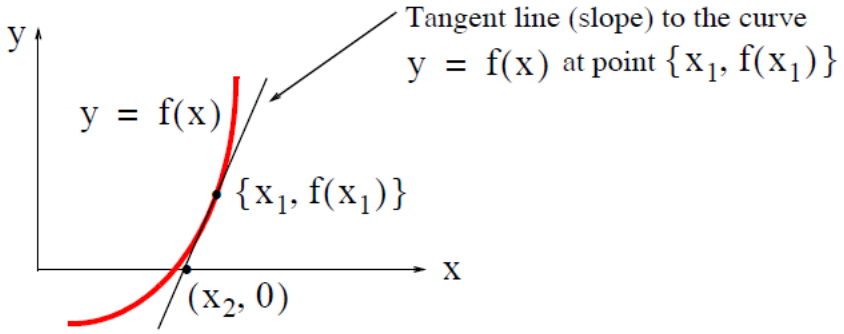

Figure 2.1.1: Newton's Method for approximating real roots of a function

We assume that the slope is neither zero nor infinite. Then, the slope (first derivative) at $x = x_1$ is

$$f'(x_1) = \frac{y - f(x_1)}{x - x_1}$$

$$y - f(x_1) = f'(x_1)(x - x_1) \tag{2.2}$$

The slope crosses the $x - axis$ at $x = x_1$ and y-0. Since this point lies in $[x_2, f(x_2)] = (x_2, 0)$ the slope line, it satisfies (2.1). By substitution,

$$0 - f(x_1) = f'(x_1)(x - x_1)$$

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} \tag{2.3}$$

Hence, the general equation is,

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{2.4}$$

Example 2.2.1

Use Newton's method to approximate the positive root of the following function

$$f(x) = x^2 - 5 \qquad (2.5)$$

To 4 decimal places

Solution

To solve this example, we shall use the following MATLAB built-in functions

Table 2.1.1: Some MATLAB functions

	MATLAB function	Description
1.	input ('string')	This command displays the text string and waits for an input from the user.
2.	polyder(f)	This function produces the coefficients of the derivative of a polynomial f.
3.	polyval (f, x)	This function evaluates the polynomial f at some value **x.**
4.	fprintf ('string')	This function prints the string in the command window.
5.	syms	This function is used to define one or more symbolic expressions.

We can compute the next iteration for approximating a root with Newton's method using these functions. Knowing the polynomial f and the first approximation x_0, we can use the following script for the next approximation x_1.

DON'T try it for now! just information

```
1. q=polyder(p)
2. x1=x0−polyval (p, x0)/polyval (q, x0)
```

To apply Newton's method, we must start with a reasonable approximation of the root value. In all cases, this can best be done by plotting $f(x)$ versus x with the familiar statements below. The following two lines of script will display the graph of the given equation in the interval $-4 \leq x \leq 4$.

Try it! In Editor Window

```
1. x=linspace (-4, 4, 100);      % Specifies 100 values between -4 and 4
2. y=x.^ 2 -5; plot (x, y); grid   % The dot exponentiation is a must
```

The plot is shown in Figure 2.1.2

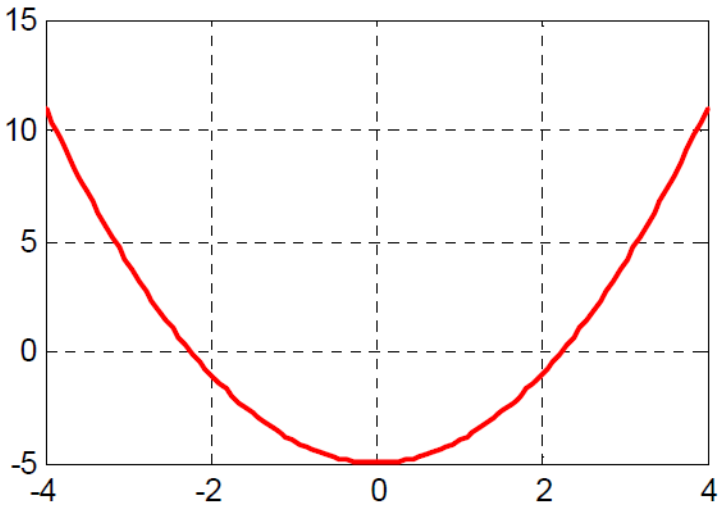

Figure 2.1.2: Plot of the curve of Example 2.1.1

As expected, the curve shows one crossing between x=2 and x=3 , so we take x0=2 as our first approximation, and we compute the next value x1 as

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 2 - \frac{2^2 - 5}{2(2)} = 2 - \frac{-1}{4} = 2.25 \qquad (2.6)$$

For the second approximation we have,

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 2.25 - \frac{(2.25)^2 - 5}{2(2.25)} = 2.25 - \frac{0.0625}{4.5} = 2.2361 \qquad (2.7)$$

To verify (2.5) and (2.6), we will use the following MATLAB script

Try it! In Editor window

```
% Approximation of a root of a polynomial function p(x)
% Do not forget to enclose the coefficients in brackets [ ]
p = input ('Enter coefficients of p(x) in descending order: ');
x0 = input ('Enter starting value: ');
q = polyder(p); % Calculates the derivative of p(x)
x1 = x0 – polyval (p, x0)/polyval(q,x0);
fprintf('\n');      % Inserts a blank line
% The next function displays the value of x1 in decimal format as indicated \n prints a
 % blank line before printing x1
fprintf ('The next approximation is: %9.6f \n', x1)
fprintf('\n');  % Inserts another blank line
%
fprintf ('Rerun the program using this value as your next approximation \n');
```

The following lines show MATLAB's inquiries and our responses (inputs) for the first two approximations. The following results are shown in the command window.

Try it! In the command window

```
Enter coefficients of P(x) in descending order:

[1 0 −5]

Enter starting value: 2

The next approximation is: 2.250000

Rerun the program using this value as your next approximation

Enter polynomial coefficients in descending order: [1 0 −5]

Enter starting value: 2.25

The next approximation is: 2.236111
```

We observe that this approximation is in close agreement with equation (2.6).

Example 2.2.2

Approximate one real root of the non-linear equation

$$f(x) = x^2 + 4x + 3 + \sin x - x\cos x \qquad (2.8)$$

to four decimal places using Newton's method.

Solution

We sketch the curve to find out where the curve crosses the x-axis. We generate the plot with the script below.

Try it! In Editor window

```
1. x=linspace (–pi, pi, 100);
2. y= x.^ 2 + 4. * x + 3 + sin(x) –x.* cos(x); plot (x, y); grid
```

The plot is shown in Figure 2.2.3.

Next, we generate the function funcnewt01, and we save it as an m−file. To save it, from the File menu of the command window, we choose New and click on M−File. This takes us to the Editor Window, where we type the following three lines, and we save it as funcnewt01.m.

Try it! In Editor window

```
1. function y=funcnewt01(x)
2. % Approximating roots with Newton's method
3. y= x.^ 2 + 4. * x + 3 + sin(x) – x .* cos(x);
```