Principles of Linear Algebra With *Mathematica*® The Newton–Raphson Method

Kenneth Shiskowski and Karl Frinkle

Contents

Chapter 1

The Newton–Raphson Method for a Single Equation

1.1 The Geometry of the Newton–Raphson Method

In studying astronomy, Sir Isaac Newton (1642–1727) needed to solve an equation $f(x) = 0$ involving trigonometric functions such as sine. He could not do it by any algebraic technique he knew, and all he really needed was just a very good approximation to the equation's solution. To satisfy his requirements, Sir Newton developed the basic algorithm we now call the *Newton–Raphson Method*. Newton discovered this method in a purely algebraic format which was very difficult to use and understand. The general method and its geometric basis was actually first seen by Joseph Raphson (1648–1715) upon reading Newton's work; although Raphson only used it on polynomial equations to find their real roots.

The Newton–Raphson method is the true bridge between algebra (solving equations of the form $f(x) = 0$ and factoring) and geometry (finding tangent lines to the graph of $y = f(x)$). What follows is an exploration of the Newton–Raphson method and how tangent lines help us solve equations, both quickly and easily — although not for exact solutions, but approximate ones.

The reason that we are studying the Newton–Raphson method in this book is that it can also solve square non-linear systems of equations using matrices and their inverses as we shall see later. Part of the wonderful effectiveness of the Newton–Raphson method is that it can solve either a single equation, or a square system of equations, for its real or complex solutions, to as many decimal places as you desire.

Now we will discuss the important application of using tangent lines to solve a single equation of the form $f(x) = 0$ for approximate solutions, either real or complex.

Example 1.1.1. The best way to understand the simplicity of this method, and its geometric basis, is to look at an example. Let's say that we want to solve the equation

$$x^3 - 5x^2 + 3x + 5 = 0 \tag{1.1}$$

for an approximate solution x. We can easily estimate where the real solutions are by finding the x-intercepts of the graph of $y = x^3 - 5x^2 + 3x + 5$. Remember that the total number of real or complex roots to any polynomial is its degree (or order), which in this case is three. Also, when a polynomial has all real coefficients, as this one does, all of the complex roots (if there are any) must occur as complex conjugate pairs. This particular polynomial has exactly three real roots and no complex roots, as can be seen from its graph in Figure 1.1.

```
F[x_] = x^3 - 5 x^2 + 3 x + 5;
RootsF = NSolve[F[x] == 0, x]
```

```
{{x → −0.709275}, {x → 1.80606}, {x → 3.90321}}
```

```
RootsPlot = Graphics[{PointSize[0.025], Red, Point[{x, 0} /.
RootsF]}];
PlotF = Plot[F[x], {x, −3, 7}, PlotStyle→{Blue, Thickness[0.007]}];
Show[PlotF, RootsPlot, PlotRange→{{−2, 5}, {−5, 6}}]
```

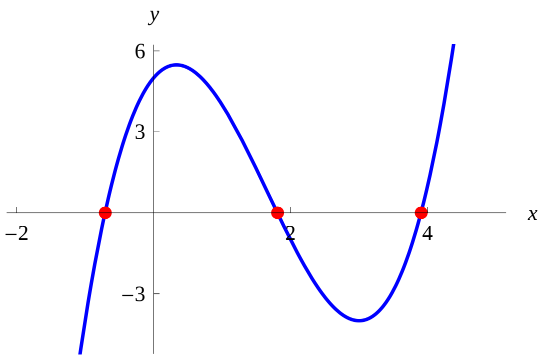

Figure 1.1: The three roots of the polynomial $x^3 - 5x^2 + 3x + 5$ are its three x-intercepts (circled)

After inspecting Figure 1.1, we see that the three real roots of the cubic, one negative and two positive, occur close to the values $x = -1$, $x = 2$, and $x = 4$. Let us first try to approximate the root located near $x = 4$ as accurately as we can.

First, let us see what **FindRoot** will give us for just the solution near $x = 4$. **FindRoot** uses many algorithms similar to, and including, the Newton–Raphson method, in combination, to approximate solutions both to a single equation or a square system of equations.

```
FindRoot[F[x] == 0, {x, 4}, WorkingPrecision→20]
```

```
{x → 3.9032119259115532876}
```

The idea that Raphson had was to take a value of x near $x = 4$, say $x = 5$, and compute the tangent line to the graph of our function $f(x)$ at $x = 5$. This tangent line goes through the point $(5, f(5)) = (5, 20)$ and has slope $\frac{df}{dx}(5)$, where $\frac{df}{dx}$ is the derivative function of $f(x)$. Now *Mathematica* can easily compute both $f(5)$ and $\frac{df}{dx}(5)$.

```
F[5]
```

```
20
```

```
DF[x_] = D[F[x], x]
```

```
3−10 x+3 x^2
```

```
DF[5]
```

```
28
```

So the slope of the tangent line to the graph of $y = f(x)$ at the point $(5, 20)$ is 28 and the equation of this tangent line at $x = 5$ is

$$y - f(5) = \frac{df}{dx}(5)(x - 5)$$

or

$$y = 28x - 120$$

Let us next graph together this tangent line and the original function $f(x)$.

```
ArrowPlots = Graphics[{Arrowheads[.05], Thickness[.010], Yellow,
Arrow[{{5, −10}, {5, −4.5}}], Black, Arrow[{{30/7, −10}, {30/7,
−4.5}}], Red, Arrow[{{3.9, 10}, {3.9, 2}}]}];
```

```
TangentF = Plot[28 x − 120, {x, −3, 7}, PlotStyle→{Black, Thick-
ness[0.007]}];
```

Show[ArrowPlots, PlotF, TangentF, RootsPlot, PlotRange→{{3.5, 5.5}, {−20, 30}}, Axes→True, AspectRatio→2/3]

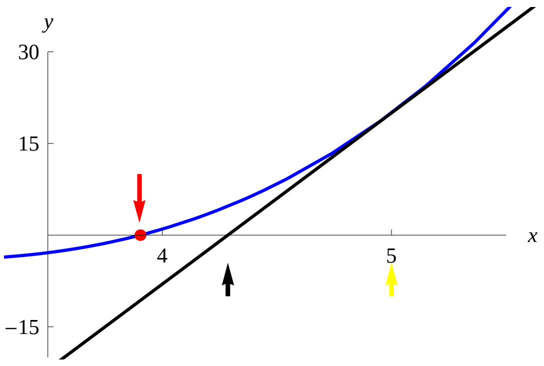

Figure 1.2: Tangent line to $x^3 - 5x^2 + 3x + 5$ at $x = 5$

Upon inspection of Figure 1.2, note that the tangent line at $x = 5$ crosses the x-axis much closer to our solution near $x = 4$ than our very rough estimate of $x = 5$, which was the x-value used to construct the tangent line.. The x-intercept of this tangent line is the solution for x to the tangent line's equation:

$$\begin{aligned} x &= 5 - \frac{f(5)}{\frac{df}{dx}(5)} \\ &= \frac{30}{7} \approx 4.285714286 \end{aligned} \tag{1.2}$$

So x-intercepts of tangent lines seem to move you closer to the x-intercepts of their function $f(x)$, which is what Raphson saw. Newton did not see this aspect of the method, as he forgot to look at the geometry of the situation and instead he concentrated on the algebra.

Raphson's next idea was to try to move even closer to the root of $f(x)$ (the x-intercept of $y = f(x)$ or solution to $f(x) = 0$) by repeating the tangent line, but now at the point given by this new value of $x = 4.285714286$, which is the x-intercept of the previous tangent line. Let uss do it repeating the above work and see if Raphson was correct to do this. To make this process more readily programmable, we will call our starting guess near the root $x_0 = 5$, and the x-intercept of the tangent line at $x = 5$ we will call $x_1 = 4.285714286$. Notice that the value of x_1 is closer to the root than the starting guess x_0.

```
x₀ = SetPrecision[5.00, 10];
x₁ = (x₀ − F[x₀]/DF[x₀])
```

```
4.28571429
```

F[x₁]

```
4.737609
```

DF[x₁]

```
15.244898
```

F[x₁] − x₁DF[x₁]

```
−60.597668
```

NextIntercept = Solve[F[x₁] + DF[x₁] (x − x₁) == 0, x]

```
{{x → 3.974947}}
```

ArrowPlots2 = Graphics[{Arrowheads[.05], Thickness[.010], Blue, Arrow[{{3.97495, −10}, {3.97495, −4.5}}]}];

TangentF2 = Plot[15.2449 x − 60.5977, {x, −3, 7}, PlotStyle→{Red, Thickness[0.007]}];

Show[ArrowPlots, ArrowPlots2, PlotF, TangentF, TangentF2, RootsPlot, PlotRange→{{3.5, 5.5}, {−20, 30}}, AspectRatio→1, Axes→True]

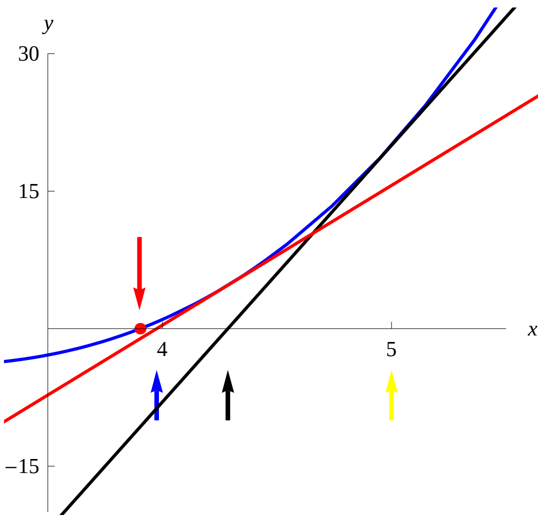

Figure 1.3: Tangent lines at $x = 5$ and $x = 4.28571$

Upon inspection of Figure 1.3, it should be clear that Raphson had a very good idea when he used tangent lines to approximately solve single equations of the form $f(x) = 0$. This is, of course, assuming that our picture above is generally true. Happily for both Raphson and us, this picture is typical, and successive tangent lines and their x-intercepts do move closer and closer to the x-intercept of the underlying function $f(x)$.

The general formula for the x-intercept of the tangent line to the graph of $y = f(x)$ at the point where $x = a$ is

$$x = a - \frac{f(a)}{\frac{df}{dx}(a)}, \tag{1.3}$$

since the equation of the tangent line at $x = a$ is

$$y = f(a) + \frac{df}{dx}(a)(x-a)$$

If you now solve

$$f(a) + \frac{df}{dx}(a)(x-a) = 0$$

for x, you will get equation (1.3).

Let us use this formula to get the successive x-intercepts to the corresponding tangent lines. From the output, we see that the values of the x-intercepts are moving towards the root of our polynomial which is located approximately at $x = 3.9$, and is depicted as the red dot on the x-axis, located below the red arrow, in Figure 1.3.

In the following *Mathematica* code, we set the number of digits to twelve, instead of the default of ten, since we want to get ten accurate digits when we round down to ten.

```
x0 = N[5, 30];
For[k = 1, k ≤ 8, k++, xk = (xk−1 − F[xk−1]/DF[xk−1]);]
Table[{ "x"k, "=", N[xk, 12]}, {k, 1, 8}] // MatrixForm
```

$$\begin{pmatrix} x_1 & = & 4.28571428571 \\ x_2 & = & 3.97494740868 \\ x_3 & = & 3.90652292594 \\ x_4 & = & 3.90321950278 \\ x_5 & = & 3.90321192595 \\ x_6 & = & 3.90321192591 \\ x_7 & = & 3.90321192591 \\ x_8 & = & 3.90321192591 \end{pmatrix}$$

The value of the x-intercept after six iterations converges to a value which agrees with the next two iterations of the method to ten decimal places. Notice that this also agrees to ten decimal places with the **FindRoot** value of $x = 3.9032119259115532876$. We may have even arrived at this value in the same way that **FindRoot** did, since the Newton–Raphson method is part of **FindRoot**'s root finding procedures. To summarize, we arrived at an approximation to the root accurate to ten decimal places after only six repeated applications of the Newton–Raphson method. This is a very fast procedure even when our starting guess of $x = 5$ is not very close to the root nearest to it.

It should be noted that you stop the Newton–Raphson method when you get a repetition in the value for two consecutive x-intercepts of your tangent lines accurate to the number of digits you desire. This is the reason we could could stop after x_6 was computed, since $|x_6 - x_5| < 10^{-10}$, and thus they agree to ten decimal places.

Newton's method will, in general, solve equations of the form $f(x) = 0$, for the solution nearest a starting estimate of $x = x_0$. It then creates a list of x_n values, where each x_n (the nth element of this list) is the x-intercept of the tangent line to $y = f(x)$ at the previous value in the list, which is x_{n-1}. This gives the general formula for x_n to be

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{\frac{df}{dx}(x_{n-1})} \tag{1.4}$$

starting with x_0. Each x_n is usually closer than the previous x_{n-1} to being a solution to $f(x) = 0$. The x_n values can all be gotten from iterating the starting guess x_0 in the *iteration function*

$$g(x) = x - \frac{f(x)}{\frac{df}{dx}(x)} \tag{1.5}$$

This means that $x_{n+1} = g(x_n)$ for all n starting with 0.

The method we are using is called *iteration* since it begins with a starting guess value of x_0 and then finds the iteration values $x_1, x_2, \ldots$ thereafter using the same formula $g(x)$ based on the previous value. The function $g(x)$ which computes these iteration values is called the *iterator*.

The following procedure will create a table of values of this Newton sequence of iterates from Example 1.1.1 and x_0 values near our three roots. Each column in this table corresponds to Newton's method applied to one of our initial guesses. We will carry out 10 iterations of the Newton–Raphson method to generate the table. We suggest trying different values for the three x_0 initial values to see how convergence to each root is affected. The three initial x_0 values below were chosen from the graph of $f(x)$ so that the tangents at these points will intersect the x-axis closer to the root than the initial guess. We stop iterating with the iterator $g(x)$ when two consecutive values in the sequence of x_n values are the same to the accuracy desired.

We will use the three starting guesses for x_0 of 0, 1, and 5 or the list **N[{0, 1, 5}, 30]**, where the command **N** is used to tell *Mathematica* that we want decimal approximations and not exact values, since, to *Mathematica* any number with a decimal point in it is considered an approximation. Without these decimal points *Mathematica* would compute exact values giving us horrendous fractions for the values of our iterates instead of decimal approximations.

```
x0 = N[{0, 1, 5}, 30];
Newt[x_] = x − F[x]/DF[x]
```

$$x-\frac{5+3x-5x^2+x^3}{3-10x+3x^2}$$

```
N[NestList[Newt[#] &, x0, 10], 12] // MatrixForm
```

$$\begin{pmatrix} 0 & 1.00000000000 & 5.00000000000 \\ -1.66666666667 & 2.00000000000 & 4.28571428571 \\ -1.00529100529 & 1.80000000000 & 3.97494740868 \\ -0.751331029986 & 1.80606060606 & 3.90652292594 \\ -0.710320317972 & 1.80606343352 & 3.90321950278 \\ -0.709276029620 & 1.80606343353 & 3.90321192595 \\ -0.709275359437 & 1.80606343353 & 3.90321192591 \\ -0.709275359437 & 1.80606343353 & 3.90321192591 \\ -0.709275359437 & 1.80606343353 & 3.90321192591 \\ -0.709275359437 & 1.80606343353 & 3.90321192591 \\ -0.709275359437 & 1.80606343353 & 3.90321192591 \end{pmatrix}$$

```
NSolve[F[x] == 0, x, WorkingPrecision→15]
```

$$\{\{x\to -0.709275359436923\},\{x\to 1.80606343352537\},\{x\to 3.90321192591155\}\}$$

The **NSolve** command has found the roots to fifteen decimal place accuracy and agrees with, to 12 decimal places, the answer obtained by the Newton iterator. You have just seen the Newton–Raphson method solve for the three real roots of our polynomial $f(x)$ simultaneously, working on finding each root based on three different starting guesses near them.

In the next section, we will continue looking at more examples of the uses of the Newton–Raphson algorithm — finding complex roots to polynomials, solving for thirteenth roots of a number, and finding inverse trigonometric function values.

Before we conclude this section, let's animate the tangent lines to our cubic polynomial $f(x)$ and see them moving along the polynomial's graph. This might help you understand why the Newton–Raphson method works geometrically if you watch where these tangent lines' x-intercepts are going. We will plot tangent lines for equally spaced points from $x = -2$ to $x = 6$. Figure 1.4 shows a frame in the animation, corresponding to the tangent line at $x = 0.9$.

Notice where the tangent line intersects the x-axis, and how close to the root this point of intersection is.

```
Manipulate[TangentLine = F[a] + DF[a] (x − a);
 AxisIntercept = Solve[F[a] + DF[a] (x − a) == 0, x];
 PlotIntercept = Graphics[{PointSize[0.025], Black, Point[{x, 0} /.
   AxisIntercept[[1]]]}];
 TangentPlot = Plot[TangentLine, {x, −3, 7}, PlotStyle→{Red,
   Thickness[0.007]}, PlotRange→{{−2, 5}, {−5, 6}}, AspectRatio
   →1];
 Show[TangentPlot, PlotF, PlotIntercept, PlotRange→{{−2, 5},
   {−5, 6}}, AspectRatio→1],
 {{a, −1, "a"}, −1, 4.5, 0.09}]
```

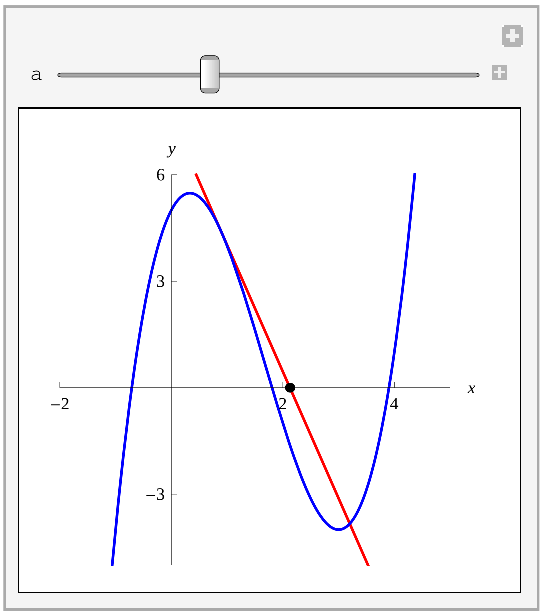

Figure 1.4: Tangent lines to $f(x) = x^3 - 5x^2 + 3x + 5$ and corresponding x-intercepts

1.2 Examples of the Newton–Raphson Method

In this section, we will use the machinery developed in Section 1.1 and apply the Newton–Raphson method to specific problems.

Example 1.2.1. As example of the power of Newton's method, we will use it to find all the roots of the polynomial $H(x) = 8x^5 - 3x^4 + 2x^3 + 9x - 5$. This polynomial has both real and complex roots. We will then use these five roots to factor the polynomial completely. Since this polynomial has all real coefficients, the complex roots must occur in complex conjugate pairs. This fact about complex roots usually means we only need find roughly half as many roots as the degree of the polynomial.

H[x_] = 8 x^5 − 3 x^4 + 2 x^3 + 9 x − 5;
Plot[H[x], {x, −10, 10}, PlotStyle→{Blue, Thickness[0.007]}]

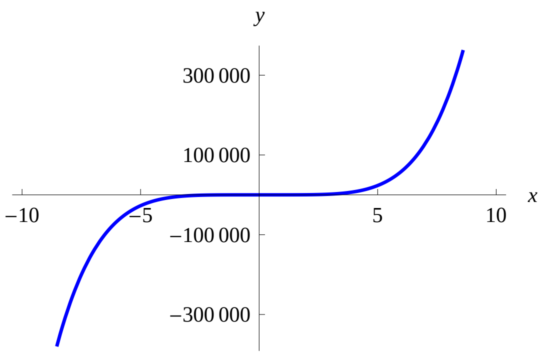

Figure 1.5: Graph of $f(x) = 8x^5 - 3x^4 + 2x^3 + 9x - 5$.

Plot[H[x], {x, −3, 3}, PlotStyle→{Blue, Thickness[0.007]}, PlotRange→{{−3, 3}, {−30, 30}}]

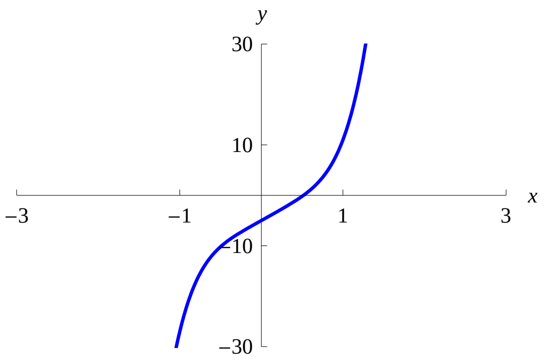

Figure 1.6: Graph of $f(x) = 8x^5 - 3x^4 + 2x^3 + 9x - 5$ zoomed in near the real root.

If we graph this polynomial (see Fig. 1.5) for $x \in [-10, 10]$, it is difficult to tell how many real roots the polynomial has. To convince ourselves that there is indeed only one real root, we need to shorten the domain of the graph. This is done in Figure 1.6 and we now know that this polynomial has only one real root. Why does an odd degree polynomial with all real coefficients have to have at least one real root? It is because there are going to be an even number of complex roots since they occur in complex conjugate pairs while the total number of all the roots must be the degree of the polynomial, which we assumed to be odd.

```
DH[x_] = D[H[x], x]
```

$$9+6\,x^2-12\,x^3+40\,x^4$$

```
x0 = N[{1, 1 + I, −1 − I}, 40];
Newt[x_]= x − H[x]/DH[x]
```

$$x-\frac{-5+9\,x+2\,x^3-3\,x^4+8\,x^5}{9+6\,x^2-12\,x^3+40\,x^4}$$

```
(NewtMat = N[NestList[Newt[#] &, x0, 6], 9]) // MatrixForm
```

$$\begin{pmatrix}
1.00000000 & 1.00000000+1.00000000\,i & -1.00000000-1.00000000\,i \\
0.744186047 & 0.829902292+0.866465925\,i & -0.835030231-0.857491933\,i \\
0.569703018 & 0.708086470+0.808269082\,i & -0.747580713-0.782085408\,i \\
0.518612910 & 0.651804769+0.816692621\,i & -0.723136047-0.760517831\,i \\
0.516111667 & 0.650719047+0.825354992\,i & -0.721409234-0.758895715\,i \\
0.516106685 & 0.650847675+0.825217125\,i & -0.721401098-0.758887036\,i \\
0.516106685 & 0.650847755+0.825217163\,i & -0.721401098-0.758887036\,i
\end{pmatrix}$$

```
RootsH = {NewtMat[[6, 1]], NewtMat[[6, 2]], Conjugate[NewtMat[[
6, 2]]], NewtMat[[6, 3]], Conjugate[NewtMat[[6, 3]]]}
```

$$\{0.516106685, 0.650847675+0.825217125\,i, 0.650847675-0.825217125\,i, -0.721401098-0.758887036\,i, -0.721401098+0.758887036\,i\}$$

```
NSolve[H[x], x]
```

$$\{\{x\to -0.721401-0.758887\,i\}, \{x\to -0.721401+0.758887\,i\}, \{x\to 0.516107\}, \{x\to 0.650848-0.825217\,i\}, \{x\to 0.650848+0.825217\,i\}\}$$

```
P = Coefficient[H[x], x, 5] Product[x − RootsH[[k]], {k, 1, 5}]
```

$$8\,((-0.650847675-0.825217125\,i)+x)((-0.650847675+0.825217125\,i)+x)(-0.516106685+x)((0.721401098-0.758887036\,i)+x)((0.721401098+0.758887036\,i)+x)$$

```
Chop[Expand[P], 10^-6]
```

$-4.9999992+8.999999\,x+2.000000\,x^3-2.9999987\,x^4+8\,x^5$

```
H[x]
```

$-5+9\,x+2\,x^3-3\,x^4+8\,x^5$

The polynomial P, defined as **P** above, gives the complete factoring of $f(x)$ using the roots found from Newton's method. When we expand P we do not get $f(x)$ back exactly due to rounding error in our roots.

Example 1.2.2. Now let's use the Newton–Raphson method to solve the equation

$$e^x = \cos(2x) + 5 \tag{1.6}$$

for its single real solution where we actually solve

$$f(x) = e^x - \cos(2x) - 5 = 0$$

We will get that the solution is $x = 1.40054551401$ after five iterations starting with $x_0 = 2$. In order to see where the solution to this equation lies, we will plot each side of the equation separately and find their intersection point (see Figure 1.7).

```
Clear[H]
H[x_] = ⅇ^x; K[x_] = Cos[2 x] + 5;
Plot[{H[x], K[x]}, {x, 0, 3}, PlotStyle→{{Red, Thickness[0.007]},
{Blue, Thickness[0.007]}}, PlotRange→{{0, 3}, {0, 15}}]
```

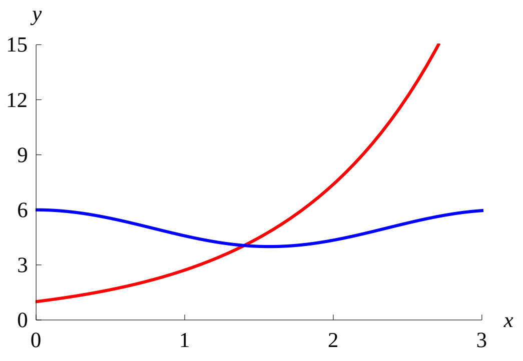

Figure 1.7: The intersection of $h(x) = e^x$ and $k(x) = \cos(2x) + 5$ occurs close to $x = 1.5$.

```
F[x_] = H[x] - K[x]
```

$-5 + e^{x} - \mathrm{Cos}[2\,x]$

```
DF[x_] = D[F[x], x]
```

$e^{x} + 2\,\mathrm{Sin}[2\,x]$

```
x0 = SetPrecision[2, 20];
Newt[x_] = x - F[x]/DF[x]
```

$$x - \frac{-5 + e^{x} - \mathrm{Cos}[2\,x]}{e^{x} + 2\,\mathrm{Sin}[2\,x]}$$

```
(NewtMat = N[NestList[Newt[#] &, x0, 5], 12]) // MatrixForm
```

$$\begin{pmatrix} 2.00000000000 \\ 1.48213342879 \\ 1.40082183928 \\ 1.40054551633 \\ 1.40054551401 \\ 1.40054551401 \end{pmatrix}$$

■

Example 1.2.3. For our next implementation of the Newton–Raphson method, we will find arctan(1.26195) from the function tan(x) alone. If we let $x =$ arctan(1.26195), then by taking tangent of both sides we have

$$\tan(x) = \tan(\arctan(1.26195)) = 1.26195$$

Rewriting this as $f(x) = \tan(x) - 1.26195 = 0$, the function $f(x)$ has the value arctan(1.26195) as a root. To pick an initial guess, we examine Figure 1.7 and note that since arctan(x) is always between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$, if we let $x_0 = 1 > 0$, then $\arctan(x) > 0$ as well.

```
F[x_] = SetPrecision[Tan[x] - 1.26195000000000000, 20]
```

$-1.2619500000000000000 + \mathrm{Tan}[x]$

```
DF[x_] = D[F[x], x]
```

$\mathrm{Sec}[x]^2$

```
Newt[x_] = x - F[x]/DF[x]
```

$x - \mathrm{Cos}[x]^2(-1.2619500000000000000 + \mathrm{Tan}[x])$

```
x0 = SetPrecision[1, 20];
(NewtMat = SetPrecision[NestList[Newt[#] &, x0, 5], 20]) // MatrixForm
```

$$\begin{pmatrix} 1.0000000000000000000 \\ 0.91374803639682598467 \\ 0.90090833143814183820 \\ 0.90069179844322014119 \\ 0.90069173923562329617 \\ 0.90069173923561887235 \end{pmatrix}$$

```
{ArcTan[1.26195], ArcTan[SetPrecision[1.26195, 20]]}
```

```
{0.900692, 0.90069173923561883576}
```

Example 1.2.4. As the last example of this section, we use Newton's method to find $\sqrt[13]{8319407225}$, or $8319407225^{1/13}$. This thirteenth root can be found by letting $x = 8319407225^{1/13}$ and then rewriting this equation without the root as

$$x^{13} - 8319407225 = 0 \tag{1.7}$$

We then have

$$f(x) = x^{13} - 8319407225 = 0 \tag{1.8}$$

Therefore, we now have our function $f(x)$ to apply Newton's method. Since

$$5^{13} = 1220703125 < 8319407225 < 13060694016 = 6^{13}$$

we can take the starting guess to be either $x_0 = 5$ or $x_0 = 6$, since choosing an integer for the starting value is an easy way to pick an initial value of x_0 (although this need not be required). You could also plot $y = f(x)$ and look for the x-intercept of this graph.

```
5^13
```

```
1 220 703 125
```

```
6^13
```

```
13 060 694 016
```

```
F[x_] = x^13 - 8 319 407 225;
DF[x_] = D[F[x], x];
Newt[x_] = x - F[x]/DF[x]
```

$$x - \frac{-8319407225 + x^{13}}{13\,x^{12}}$$

x_0 = SetPrecision[6, 20];
(NewtMat = SetPrecision[NestList[Newt[#] &, x_0, 5], 12]) // MatrixForm

$$\begin{pmatrix} 6.00000000000 \\ 5.83245253211 \\ 5.79678752512 \\ 5.79541014785 \\ 5.79540818028 \\ 5.79540818028 \end{pmatrix}$$

N[8 319 407 225$^{1/13}$, 12]

5.79540818028

Examples 1.2.3 and 1.2.4 help to illustrate that Newton's method may be used to find the values of inverse trigonometric functions using the regular trigonometric functions and to get roots of numbers using powers. In a similar fashion, Newton's method can also find the values of logarithms using exponentials. In chapter 2, we shall see that Newton's method also scales in dimension to to solve square systems of non-linear equations, not just a single equation.

1.3 An Example of When the Newton–Raphson Method Does the Unexpected

So far, all of our examples have worked out wonderfully. Our initially guesses converged to a root of the function in question, and did so remarkably fast. The following is an instance of Newton's method in which a very particular x_0 gives rise to convergence to a root not closest to x_0, and does so very slowly. We will attempt to determine exactly why this behavior occurs.

Example 1.3.1. If we attempt to solve the trigonometric equation $\sin(x) = 0$, we should first notice that the roots are easy to find, and are located at all integer multiples of π. One would expect that if we chose a starting value relatively close to $x = \pi$, Newton's method should converge to $x = \pi$. So, let us use $x_0 = 1.97603146838$ as our starting value. Before you look at the output of Newton's method, be sure to look at the graph of $\sin(x)$, as depicted in Fig. 1.8, near our starting value and try to determine for yourself what the end result should be.

F[x_] = Sin[x];

Plot[F[x], {x, 0, 2π}, PlotStyle→{Blue, Thickness[0.007]}]

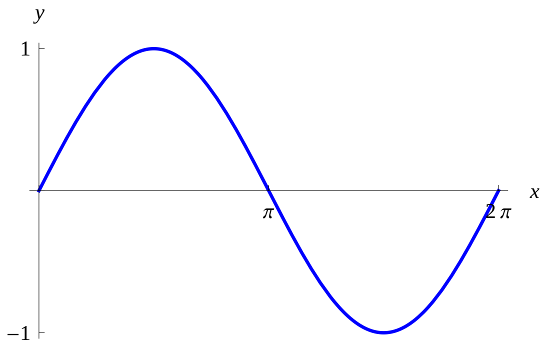

Figure 1.8: Graph of $f(x) = \sin(x)$ on the interval $[0, 2\pi]$.

```
x0 = SetPrecision[1.97603146838000000, 20];
DF[x_] = D[F[x], x];
Newt[x_] = SetPrecision[x − F[x]/DF[x], 20]
```

```
x − 1.0000000000000000000 Tan[x]
```

```
NewtMat = SetPrecision[NestList[Newt[#] &, x0, 30], 20]
```

```
{1.9760314683800000000, 4.3071538388110351013, 1.9760314683063380452,
 4.3071538392113238509, 1.9760314661311163369, 4.3071538510317647441,
 1.9760314018972838234, 4.3071542000869239695, 1.9760295050836902494,
 4.3071645076791808222, 1.9759734905665374138, 4.3074689482982437180,
 1.9743176314664917498, 4.3165112964510746414, 1.9238373689589648698,
 4.6376992990975918809, −8.7261251905529504699, −9.5661131584604639451,
 − 9.4238292930300700030, −9.4247779610539707931,
 − 9.4247779608704149723, −9.4247779608704149723,
 − 9.4247779608704149723, −9.4247779608704149723,
 − 9.4247779608704149723, −9.4247779608704149723,
 − 9.4247779608704149723, −9.4247779608704149723,
 − 9.4247779608704149723, −9.4247779608704149723,
 − 9.4247779608704149723}
```

```
N[−3π]
```

```
−9.42478
```

Now lets plot the first two tangent lines for Newton's method to $y = f(x) = \sin(x)$. It will help to illustrate what we see as this sequence of values move towards $x = -2\pi$ (see Figure 1.9).

TLx₀ = Expand[F[x₀] + (D[F[x], x] /. x→x₀) (x − x₀)]

```
1.6980302279867648055 − 0.3942348686703738045 x
```

TLx₁ = Expand[F[NewtMat[[2]]] + (D[F[x], x] /. x→NewtMat[[2]]) (x − NewtMat[[2]])]

```
0.779020506375484484 − 0.3942348686598524074 x
```

Plot[{F[x], TLx₀, TLx₁}, {x, 0, 2π}, PlotStyle→{{Blue, Thickness[0.008]}, {Red, Thickness[0.006]}, {Black, Thickness[0.006]}}]

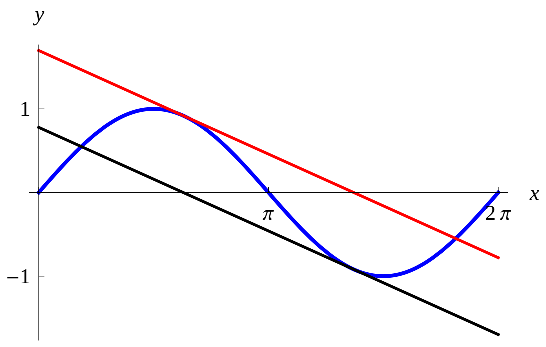

Figure 1.9: Graph of $f(x) = \sin(x)$ and parallel tangent lines.

For this starting value of $x_0 = 1.97603146838$, Newton's method accurately locates -2π to twelve decimal place accuracy. Clearly this root is not the closest solution to $\sin(x) = 0$ at x_0. The values in this sequence approximately alternate each other for a long time until suddenly they move off towards -2π. They do this because the tangent lines used in the method are approximately parallel lines where each has x-intercept approximately the x-coordinate value of the other. You should see what happens to this sequence of iterates if you increase the number of digits to sixteen.

In concluding this first chapter, we simply with to reiterate that the Newton–Raphson method works quickly to give very good accuracy in most instances. It normally takes no more than five to ten iterations from a reasonable initial guess x_0 to get the solution accurate to ten or more decimal places. Each iteration usually produces greater accuracy, and you reach the accuracy you want when two consecutive iterations agree in value to this accuracy.

Chapter 2

The Newton–Raphson Method for Square Systems of Equations

2.1 Newton–Raphson for Two Equations in Two Unknowns

In this section we will discuss the Newton–Raphson method for solving square (as many equations as variables) systems of two non-linear equations

$$f(x,y)=0,\ g(x,y)=0$$

in the two variables x and y. In order to do this we must combine these two equations into a single equation of the form $F(x,y)=0$ where F must give us a two component column vector and 0 is also the two component zero column vector, that is,

$$F(x,y)=\begin{bmatrix} f(x,y) \\ g(x,y) \end{bmatrix} \tag{2.1}$$

and $0=\begin{bmatrix} 0 \\ 0 \end{bmatrix}$. Then clearly the equation $F(x,y)=0$ is the same as

$$\begin{bmatrix} f(x,y) \\ g(x,y) \end{bmatrix}=\begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{2.2}$$

which is equivalent to the system of two equations $f(x,y)=0$ and $g(x,y)=0$.

Now let $F:\mathbb{R}^2\to\mathbb{R}^2$ be a continuous function which has continuous first partial derivatives where F is defined as in equation (2.1) for variables x, y and component functions $f(x,y)$, $g(x,y)$. We wish to solve the equation $F(x,y)=$

0 which is really solving simultaneously the square system of two non-linear equations given by $f(x, y) = 0$ and $g(x, y) = 0$.

In order to do this, we shall have to generalize the one variable Newton–Raphson method iterator formula for solving the equation $f(x) = 0$ given by the sequence p_k for p_0 the starting guess where

$$p_{k+1} = p_k - \frac{f(p_k)}{\frac{df}{dx}(p_k)} \tag{2.3}$$

or

$$p_{k+1} = g(p_k)$$

where

$$g(p_k) = p_k - \frac{f(p_k)}{\frac{df}{dx}(p_k)} \tag{2.4}$$

is the (single equation) Newton–Raphson iterator.

To see how this can be done, you must realize that now in the two equation case that

$$p_0 = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{2.5}$$

is our starting guess as a point in the xy-plane written as a column vector,

$$p_k = \begin{bmatrix} x_k \\ y_k \end{bmatrix} \tag{2.6}$$

is the kth iteration of our method, and

$$F(p_k) = \begin{bmatrix} f(x_k, y_k) \\ g(x_k, y_k) \end{bmatrix}$$

are all two component column vectors in $\mathbb{R}^2$ and not numbers as in the single variable case. Thus, dividing a two component column vector by a derivative requires that we be dividing by a 2×2 matrix, or multiplying by the inverse of this matrix. Thus, we need to replace $\frac{df}{dx}$ in our old iterator $g(x)$ by a 2×2 matrix consisting of the four partial derivatives of $F(x, y)$, which are $\frac{\partial f}{\partial x}$, $\frac{\partial f}{\partial y}$, $\frac{\partial g}{\partial x}$, and $\frac{\partial g}{\partial y}$.

The choice of this new derivative matrix is the 2×2 Jacobian matrix $J(x, y)$, of $F(x, y)$, given by the 2×2 matrix

$$J(x, y) = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \\ \frac{\partial g}{\partial x} & \frac{\partial g}{\partial y} \end{bmatrix} \tag{2.7}$$

Other choices for this 2×2 matrix of partial derivatives are possible and you should see if they will also work in place of this Jacobian matrix, but this

Jacobian matrix seems most logical if you think about the fact that for $F(x, y)$, the first row is $f(x, y)$ and $g(x, y)$ is in the second row while the variables are given as x first and y second in all these functions. Thus, the Newton–Raphson array (list or sequence) is now for a starting vector p_0, as defined in equation (2.5), given by

$$p_{k+1} = p_k - (J(p_k))^{-1}\, F(p_k) \tag{2.8}$$

where the vector $F(p_k)$ is multiplied on its left by the inverse of the Jacobian matrix $J(x, y)$ evaluated at p_k, i.e., $J(p_k) = J(x_k, y_k)$.

Note that finding $(J(p_k))^{-1}$ in each iteration is a formidable task if the system of equations is large, say 25×25 or more, and so at each iteration you can instead solve for p_{k+1} by solving the square linear system of equations

$$J(p_k)\, p_{k+1} = J(p_k)\, p_k - F(p_k)$$

where the components of p_{k+1} are the unknowns x_{k+1} and y_{k+1}. Of course, it is easy to find p_{k+1} directly, if you are have a small number of equations as in this case, using the inverse matrix.

Example 2.1.1. Let $F : \mathbb{R}^2 \to \mathbb{R}^2$ be given in the form of equation (2.1), for

$$f(x, y) = \frac{1}{64}(x - 11)^2 - \frac{1}{100}(y - 7)^2 - 1,\ g(x, y) = (x - 3)^2 + (y - 1)^2 - 400$$

We wish to apply the Newton–Raphson method to solve

$$F(x, y) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

or equivalently the square system

$$f(x, y) = 0,\ g(x, y) = 0$$

The solutions are the intersection points of these two curves $f(x, y) = 0$ which is a hyperbola, and $g(x, y) = 0$ which is a circle, and is depicted in Figure 2.1.

F[$x\_$, $y\_$] = $\frac{(x-11)^2}{64} - \frac{(y-7)^2}{100} - 1$;
G[$x\_$, $y\_$] = $(x-3)^2 + (y-1)^2 - 400$;
FGPlot = ContourPlot[{F[x, y] == 0, G[x, y] == 0}, {x, −25, 25}, {y, −25, 25}, ContourStyle→{{Red, Thickness[0.01]}, {Blue, Thickness[0.01]}}, PlotRange→{{−25, 25}, {−25, 25}}, Axes→True,

Frame→False, AspectRatio→1]

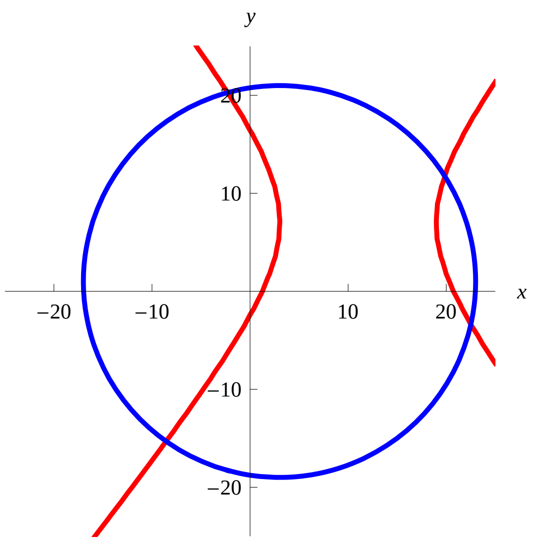

Figure 2.1: The hyperbola $f(x, y) = 0$ and circle $g(x, y) = 0$ intersect at four points.

It is clear from this plot of the circle and hyperbola that this system of equations has exactly four real solutions which are the intersection points of these two curves.

(FGMat = {{F[x, y]}, {G[x, y]}}) // MatrixForm

$$\begin{pmatrix} -1 + \frac{1}{64}(-11 + \mathrm{x})^2 - \frac{1}{100}(-7 + \mathrm{y})^2 \\ -400 + (-3 + \mathrm{x})^2 + (-1 + \mathrm{y})^2 \end{pmatrix}$$

Flatten[N[FGMat /. {{x→−2, y→20}}]]

$\{-0.049375, -14.\}$

(DFGMat = {{D[F[x, y], x], D[F[x, y], y]}, {D[G[x, y], x], D[G[x, y], y]}}) // MatrixForm

$$\begin{pmatrix} \frac{1}{32}(-11 + \mathrm{x}) & \frac{7-\mathrm{y}}{50} \\ 2(-3 + \mathrm{x}) & 2(-1 + \mathrm{y}) \end{pmatrix}$$

The system iterator for Newton–Raphson will be called **NewtIter** and it is a function of x and y which outputs a two element list instead of a two component column vector.

The iterates p_k are given by **NewtIter[[k]]** with starting guess p_0 given by **NewtIter[[1]]**. In this case, since we graphed the two equations, we can find our starting guesses for the four solutions from this graph. Without a graph, you would have to plug guesses into $F(x, y)$ until you got a result close to the zero column vector; graphing is so much faster if we have a machine to do it for us. The number of iterations we will do is five.

(NewtIter = (Factor[Simplify[{{x}, {y}} − Inverse[DFGMat]. FGMat]])) // MatrixForm

$$\begin{pmatrix} -\frac{43039+137\,x^2-5599\,y-41\,x^2\,y+96\,y^2}{2(611-137\,x-323\,y+41\,x\,y)} \\ \frac{-109173+10391\,x-200\,x^2-323\,y^2+41\,x\,y^2}{2(611-137\,x-323\,y+41\,x\,y)} \end{pmatrix}$$

NewtIter /. {x→−2, y→20}

$\{\{-\frac{11091}{4810}\},\{\frac{19517}{962}\}\}$

x_0 = −2; y_0 = 20;
(Seq1 = N[NestList[Flatten[NewtIter] /. {x→#[[1]] , y→#[[2]]} &, {x_0, y_0}, 5], 15]) // MatrixForm

$$\begin{pmatrix} -2.00000000000000 & 20.0000000000000 \\ -2.30582120582121 & 20.2879417879418 \\ -2.30204186914601 & 20.2844076598497 \\ -2.30204129069382 & 20.2844071247155 \\ -2.30204129069380 & 20.2844071247155 \\ -2.30204129069380 & 20.2844071247155 \end{pmatrix}$$

root1 = Seq1[[5]]

{−2.30204129069380, 20.2844071247155}

So the root of the system of equations closest to the point $(-2, 20)$ is **Seq1[[5]]** which we have called **root1**. We also check this root using **FindRoot** where we must tell it where to look in order to get back just this single root.

FindRoot[{F[x, y] == 0, G[x, y] == 0}, {{x, −2}, {y, 20}}, WorkingPrecision→15]

{x → −2.30204129069382, y → 20.2844071247155}

We will now find the other roots using Newton's method, and will verify our results with the **FindRoot** command.

x_0 = 20; y_0 = 13;
(Seq2 = N[NestList[Flatten[NewtIter] /. {x→#[[1]] , y→#[[2]]} &, {x_0, y_0}, 5], 15]) // MatrixForm

$$\begin{pmatrix} 20.0000000000000 & 13.0000000000000 \\ 19.8434903047091 & 11.8467220683287 \\ 19.8595722538244 & 11.7593086742885 \\ 19.8596601049843 & 11.7588039022428 \\ 19.8596601079565 & 11.7588038853852 \\ 19.8596601079565 & 11.7588038853852 \end{pmatrix}$$

root2 = Seq2[[5]]

{19.8596601079565, 11.7588038853852}

FindRoot[{F[x, y] == 0, G[x, y] == 0}, {{x, 20}, {y, 13}}, WorkingPrecision→15]

{x → 19.8596601079565, y → 11.7588038853852}

x_0 = 20; y_0 = −4;
(Seq3 = N[NestList[Flatten[NewtIter] /. {x→#[[1]] , y→#[[2]]} &, {x_0, y_0}, 5], 15]) // MatrixForm

$$\begin{pmatrix} 20.0000000000000 & -4.00000000000000 \\ 22.7557687636629 & -3.23038620354627 \\ 22.5208918105057 & -3.35966313263437 \\ 22.5190258519629 & -3.35977445340876 \\ 22.5190257453235 & -3.35977453011049 \\ 22.5190257453235 & -3.35977453011049 \end{pmatrix}$$

root3 = Seq3[[5]]

{22.5190257453235, −3.35977453011049}

FindRoot[{F[x, y] == 0, G[x, y] == 0}, {{x, 20}, {y, −4}}, WorkingPrecision→15]

{x → 22.5190257453235, y → −3.35977453011049}

x_0 = −10; y_0 = −16;
(Seq4 = N[NestList[Flatten[NewtIter] /. {x→#[[1]] , y→#[[2]]} &, {x_0, y_0}, 5], 15]) // MatrixForm

$$\begin{pmatrix} -10.0000000000000 & -16.0000000000000 \\ -8.62568385732001 & -15.3450652855788 \\ -8.56457038737749 & -15.3176346035177 \\ -8.56444944110762 & -15.3175828216490 \\ -8.56444944063497 & -15.3175828214536 \\ -8.56444944063497 & -15.3175828214536 \end{pmatrix}$$

root4 = Seq4[[5]]

{−8.56444944063497, −15.3175828214536}

FindRoot[{F[x, y] == 0, G[x, y] == 0}, {{x, −10}, {y, −16}}, Working-Precision→15]

{x → −8.56444944063497, y → −15.3175828214536}

Now that we have all four real roots of this system, we can plot these four points with the hyperbola and circle to see that they are the correct intersection points (see Figure 2.2).

RootPlots = Graphics[{PointSize[0.03], Black, Point[{root1, root2, root3, root4}]}];
Show[FGPlot, RootPlots]

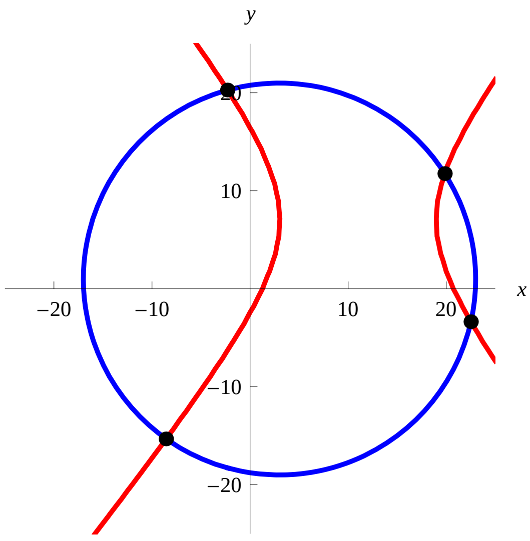

Figure 2.2: Intersection of the hyperbola $f(x, y) = 0$ and circle $g(x, y) = 0$ are the four points found by Newton–Raphson.

Example 2.1.2. Let $F : \mathbb{R}^2 \to \mathbb{R}^2$ be given in the form of equation (2.1), for

$$f(x, y) = 3x^2y - y^3 + 5x - 8,\ g(x, y) = 3xy^2 - x^3 - 4y + 2$$

We wish to apply Newton's method to solve

$$F(x, y) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

or equivalently the square system

$$f(x,y)=0,\ g(x,y)=0$$

The real solutions are the real intersection points of these two curves.

F[*x*_, *y*_] = 3 x² y − y³ + 5 x − 8;
G[*x*_, *y*_] = 3 x y² − x³ − 4 y + 2;
FGPlot = ContourPlot[{F[x, y] == 0, G[x, y] == 0}, {x, −25, 25}, {y, −25, 25}, ContourStyle→{{Red, Thickness[0.01]}, {Blue, Thickness[0.01]}}, PlotRange→{{−25, 25}, {−25, 25}}, Axes→True, Frame→False, AspectRatio→1]

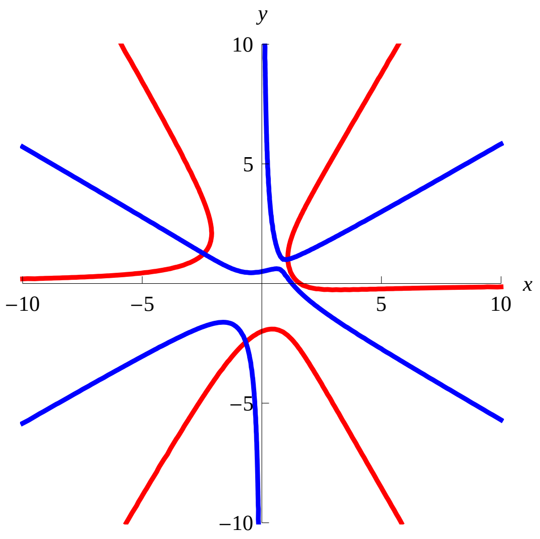

Figure 2.3: Intersection of the level curves $f(x,y)=0$ and $g(x,y)=0$.

It is clear from Figure 2.3 that this system of equations has exactly three real solutions which are the intersection points of these two curves. How many total real and complex solutions are there to this system?

(FGMat = {{F[x, y]}, {G[x, y]}}) // MatrixForm

$$\begin{pmatrix} -8+5\,x+3\,x^2\,y-y^3 \\ 2-x^3-4\,y+3\,x\,y^2 \end{pmatrix}$$

Flatten[N[FGMat /. {x→−2 + I, y→5 − 3 I}]]

{1.+116. ⅈ, −22.+229. ⅈ}

```
(DFGMat = {{D[F[x, y], x], D[F[x, y], y]}, {D[G[x, y], x], D[G[x, y],
y]}}) // MatrixForm
```

$$\begin{pmatrix} 5+6\,x\,y & 3\,x^2-3\,y^2 \\ -3\,x^2+3\,y^2 & -4+6\,x\,y \end{pmatrix}$$

```
(NewtIter = (Factor[Simplify[{{x}, {y}} − Inverse[DFGMat].
FGMat]])) // MatrixForm
```

$$\begin{pmatrix} \frac{2\left(-16+3x^2+3x^5+24xy-12x^2y-3y^2+6x^3y^2+4y^3+3xy^4\right)}{-20+9x^4+6xy+18x^2y^2+9y^4} \\ \frac{2\left(-5+12x^2-5x^3-6xy+3x^4y-12y^2+15xy^2+6x^2y^3+3y^5\right)}{-20+9x^4+6xy+18x^2y^2+9y^4} \end{pmatrix}$$

```
NewtIter /. {x →−0.25 + 0.433 I, y→0.433 - 0.75 I}
```

$\{\{1.06721-0.313821\,i\},\{-0.0545072-0.4715\,i\}\}$

```
x0 = 7. − 10. I; y0 = −5. + 3. I;
(Seq1 = SetPrecision[NestList[Flatten[NewtIter] /. {x→#[[1]] , y→
#[[2]]} &, {x0, y0}, 12], 12]) // MatrixForm
```

```
 7.00000000000 − 10.0000000000 i      −5.00000000000 + 3.00000000000 i
 4.71607628904 − 6.60071522707 i      −3.38297490293 + 2.01217959092 i
 3.22416973697 − 4.30864937934 i      −2.32662679354 + 1.37263495340 i
 2.28314611858 − 2.75413325299 i      −1.64393687179 + 0.99073595274 i
 1.74285971034 − 1.71614567645 i      −1.17852503220 + 0.82200286336 i
 1.46529550929 − 1.09376769351 i     −0.770937911022 + 0.788281456980 i
 1.29241826622 − 0.76054701886 i     −0.399170553164 + 0.748552340339 i
1.204529252433 − 0.582511953350 i    −0.145124036298 + 0.681731736303 i
1.196760917973 − 0.515657328424 i    −0.052891607834 + 0.618883269338 i
1.202567804087 − 0.509474855682 i    −0.049752878796 + 0.603573708249 i
1.202681504457 − 0.509586110927 i    −0.050028196483 + 0.603512420464 i
1.202681462289 − 0.509586075656 i    −0.050028104126 + 0.603512445782 i
1.202681462289 − 0.509586075656 i    −0.050028104126 + 0.603512445782 i
```

```
root1 = Seq1[[12]];
FGMat /. {x→root1[[1]], y→root1[[2]]}
```

$\{\{0.\times10^{-11}+0.\times10^{-11}\,i\},\{0.\times10^{-11}+0.\times10^{-11}\,i\}\}$

This last example indicates that we have at least one complex solution to this real system of equations. We conclude this section with a few remarks.

(1) It should be clear from the above examples that even the system of equations version of the Newton–Raphson method is very fast! It also works to generate complex solutions as long as the starting value is also complex when your equations are real. Redo the above example to see if you can find another complex solution. Is the complex conjugate of a solution in the last example also a solution?

(2) Also, rewrite the code above so that no inverse of the Jacobian is needed since this is impractical for large matrices and systems of equations.

(3) Now try using the Newton–Raphson method to solve the system

$$(x-4)^2+(y-9)^2=25,\ (x-3)^2+(y-7)^2=36$$

for both solutions. The two real solutions here are the intersection points of these two circles. Are there any complex solutions?

2.2 Newton–Raphson for Three Equations in Three Unknowns

In this section we will look at an example of using the system version of the Newton–Raphson method to solve a 3 × 3 system of equations which are all spheres in space.

Example 2.2.1. Our three spheres have the equations

$$\begin{aligned}(x-5)^2+(y-9)^2+(z-4)^2&=49\\(x-2)^2+(y-7)^2+(z-13)^2&=100\\(x-6)^2+(y-11)^2+(z-10)^2&=64\end{aligned}\tag{2.9}$$

Let's now plot all three spheres and see if we can find any real intersection points of all three spheres (see Figure 2.4).

```
F[x_, y_, z_] = (x − 5)^2 + (y − 9)^2 + (z − 4)^2 − 49;
G[x_, y_, z_]= (x − 2)^2 + (y − 7)^2 + (z − 13)^2 − 100;
H[x_, y_, z_]= (x − 6)^2 + (y − 11)^2 + (z − 10)^2 − 64;
FGHPlot = ContourPlot3D[{F[x, y, z] == 0, G[x, y, z] == 0, H[x, y,
z] == 0}, {x,−10,25}, {y,−10,25}, {z,−10,25}, ContourStyle→{{Red,
Thickness[0.01]}, {Blue, Thickness[0.01]}, {Yellow, Thickness[0.01]}},
PlotRange→{{−10, 25}, {−10, 25}, {−10, 25}}, Axes→True, Aspect-
```

Ratio→1, Mesh→None, AxesLabel→{"x", "y", "z"}]

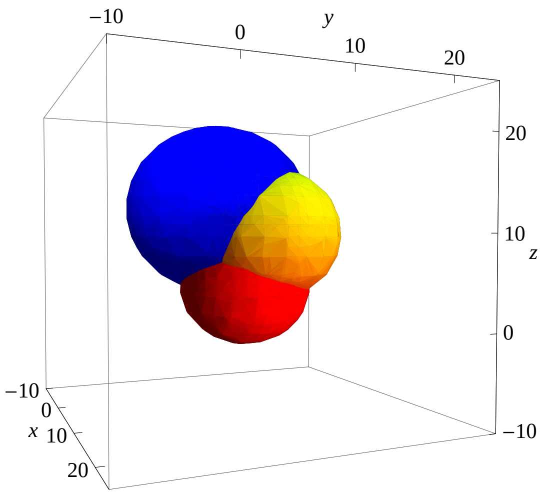

Figure 2.4: Intersection of three spheres.

It is clear from this plot of the three spheres that this system of equations has exactly two real solutions which are the intersection points of these three spheres near the points $(-2, 14, 6)$ and $(8, 5, 7)$.

We now use the Newton–Raphson system method to solve our problem where $F : \mathbb{R}^3 \to \mathbb{R}^3$ be given by

$$F(x, y, z) = \begin{bmatrix} f(x, y, z) \\ g(x, y, z) \\ h(x, y, z) \end{bmatrix} \tag{2.10}$$

for

$$\begin{aligned} f(x, y, z) &= (x-5)^2 + (y-9)^2 + (z-4)^2 - 49 \\ g(x, y, z) &= (x-2)^2 + (y-7)^2 + (z-13)^2 - 100 \\ h(x, y, z) &= (x-6)^2 + (y-11)^2 + (z-10)^2 - 64 \end{aligned} \tag{2.11}$$

We wish to apply the Newton–Raphson method to solve

$$F(x, y, z) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

(FGHMat = {{F[x, y, z]}, {G[x, y, z]}, {H[x, y, z]}}) // MatrixForm

$$\begin{pmatrix} -49+(-5+\mathrm{x})^2+(-9+\mathrm{y})^2+(-4+\mathrm{z})^2 \\ -100+(-2+\mathrm{x})^2+(-7+\mathrm{y})^2+(-13+\mathrm{z})^2 \\ -64+(-6+\mathrm{x})^2+(-11+\mathrm{y})^2+(-10+\mathrm{z})^2 \end{pmatrix}$$

Flatten[N[FGHMat /. {{x→−2., y→14., z→6.}}]]

$\{29., 14., 25.\}$

(DFGHMat = {{D[F[x, y, z], x], D[F[x, y, z], y], D[F[x, y, z], z]}, {D[G[x, y, z], x], D[G[x, y, z], y], D[G[x, y, z], z]}, {D[H[x, y, z], x], D[H[x, y, z], y], D[H[x, y, z], z]}}) // MatrixForm

$$\begin{pmatrix} 2(-5+\mathrm{x}) & 2(-9+\mathrm{y}) & 2(-4+\mathrm{z}) \\ 2(-2+\mathrm{x}) & 2(-7+\mathrm{y}) & 2(-13+\mathrm{z}) \\ 2(-6+\mathrm{x}) & 2(-11+\mathrm{y}) & 2(-10+\mathrm{z}) \end{pmatrix}$$

The system iterator for Newton–Raphson will be called **NewtIter** and it is a function of x, y, and z which outputs a three element column vector. The p_k iterates are stored in the variable **Seq1**, with starting guess p_0 given by **Seq1**[[**1**]]. In this case, since we graphed the three equations, we can find our starting guesses for the two solutions from this graph. Without a graph, you would have to plug guesses into $F(x, y, z)$ until you got a result close to the zero column vector; graphing is so much faster. The number of iterations we will do below is five, at which time the root has been located to twelve decimal place accuracy.

(NewtIter = (Factor[Simplify[{{x}, {y}, {z}} − Inverse[DFGHMat]. FGHMat]])) // MatrixForm

$$\begin{pmatrix} \frac{3118+15\,\mathrm{x}^2-393\,\mathrm{y}+15\,\mathrm{y}^2-169\,\mathrm{z}+15\,\mathrm{z}^2}{77+30\,\mathrm{x}-27\,\mathrm{y}+4\,\mathrm{z}} \\ -\frac{3595-786\,\mathrm{x}+27\,\mathrm{x}^2+27\,\mathrm{y}^2-409\,\mathrm{z}+27\,\mathrm{z}^2}{2\,(77+30\,\mathrm{x}-27\,\mathrm{y}+4\,\mathrm{z})} \\ \frac{1699+338\,\mathrm{x}+4\,\mathrm{x}^2-409\,\mathrm{y}+4\,\mathrm{y}^2+4\,\mathrm{z}^2}{2\,(77+30\,\mathrm{x}-27\,\mathrm{y}+4\,\mathrm{z})} \end{pmatrix}$$

NewtIter /. {x→−2., y→14., z→6.}

$\{\{-0.421365\}, \{13.4792\}, \{5.57715\}\}$

x$_0$ = −2.; y$_0$ = 14.; z$_0$ = 6.;
(Seq1 = SetPrecision[NestList[Flatten[NewtIter] /.{x→#[[1]] , y→#

[[2]], z→#[[3]]} &, {x_0, y_0, z_0}, 5], 12]) // MatrixForm

$$\begin{pmatrix} -2.00000000000 & 14.0000000000 & 6.00000000000 \\ -0.421364985163 & 13.4792284866 & 5.57715133531 \\ -0.262201908681 & 13.3359817178 & 5.59837307884 \\ -0.259615579897 & 13.3336540219 & 5.59871792268 \\ -0.259614896621 & 13.3336534070 & 5.59871801378 \\ -0.259614896621 & 13.3336534070 & 5.59871801378 \end{pmatrix}$$

root1 = Seq1[[5]]

{−0.259614896621, 13.3336534070, 5.59871801378}

FGHMat /. {x→root1[[1]], y→root1[[2]], z→root1[[3]]}

$\left\{\left\{0.\times10^{-10}\right\},\left\{0.\times10^{-10}\right\},\left\{0.\times10^{-10}\right\}\right\}$

FindRoot[{F[x, y, z] == 0, G[x, y, z] == 0, H[x, y, z] == 0}, {{x, −2.}, {y, 14.}, {z, 6.}}, WorkingPrecision→15]

{x → −0.259614896620942, y → 13.3336534069588, z → 5.59871801378387}

So the root of the system of equations closest to the point $(-2, 14, 6)$ is **Seq1[[5]]** which we have called **root1**. We also checked this root in two ways by first plugging it back in the function F to see if we get very close to zero (which we do) and then by using **FindRoot** where we must tell it where to look in order to get back just this specific root.

x_0 = 8.; y_0 = 5.; z_0 = 7.;
(Seq2 = SetPrecision[NestList[Flatten[NewtIter] /.{x→#[[1]] , y→# [[2]], z→#[[3]]} &, {x_0, y_0, z_0}, 5], 12]) // MatrixForm

$$\begin{pmatrix} 8.00000000000 & 5.00000000000 & 7.00000000000 \\ 9.71428571429 & 4.35714285714 & 6.92857142857 \\ 9.53347012054 & 4.51987689151 & 6.90446268274 \\ 9.53013275167 & 4.52288052350 & 6.90401770022 \\ 9.53013161395 & 4.52288154745 & 6.90401754853 \\ 9.53013161395 & 4.52288154745 & 6.90401754853 \end{pmatrix}$$

root2 = Seq2[[5]]

{9.53013161395, 4.52288154745, 6.90401754853}

FGHMat /. {x→root2[[1]], y→root2[[2]], z→root2[[3]]}

$\left\{\left\{0.\times10^{-10}\right\},\left\{0.\times10^{-10}\right\},\left\{0.\times10^{-10}\right\}\right\}$

FindRoot[{F[x, y, z] == 0, G[x, y, z] == 0, H[x, y, z] == 0}, {{x, 8.}, {y, 5.}, {z, 7.}}, WorkingPrecision→15]

{x → 9.53013161394617, y → 4.52288154744845, z → 6.90401754852616}

Now we will start with a complex guess to see if our system might have any complex solutions.

x_0 = 15. − 20. I; y_0 = −14. + 8. I; z_0 = −50. − 9. I;

Seq3 = Chop[SetPrecision[NestList[Flatten[NewtIter] /.{x→#[[1]] , y→#[[2]], z→#[[3]]} &, {x_0, y_0, z_0},11], 12], 10^{-15}];

Part[Seq3[[All, 1]]] // MatrixForm

$$\begin{pmatrix} 15.0000000000 - 20.0000000000\,i \\ 30.7230149911 + 36.4439840743\,i \\ 17.8347214116 + 18.0046438095\,i \\ 11.5522658813 + 8.5695435957\,i \\ 8.77700411428 + 3.43829707054\,i \\ 8.41850568458 + 0.29760979388\,i \\ 9.67397046894 - 0.09876135064\,i \\ 9.53127161528 - 0.00279697097\,i \\ 9.53013094812 - 6.5163 \times 10^{-7}\,i \\ 9.53013161395 + 0. \times 10^{-14}\,i \\ 9.53013161395 \\ 9.53013161395 \end{pmatrix}$$

Part[Seq3[[All, 2]]] // MatrixForm

$$\begin{pmatrix} -14.0000000000 + 8.0000000000\,i \\ -14.5507134920 - 32.7995856669\,i \\ -2.9512492704 - 16.2041794286\,i \\ 2.70296070683 - 7.71258923612\,i \\ 5.20069629715 - 3.09446736348\,i \\ 5.52334488388 - 0.26784881449\,i \\ 4.39342657796 + 0.08888521557\,i \\ 4.52185554624 + 0.00251727388\,i \\ 4.52288214669 + 5.8647 \times 10^{-7}\,i \\ 4.52288154745 + 0. \times 10^{-14}\,i \\ 4.52288154745 \\ 4.52288154745 \end{pmatrix}$$

Part[Seq3[[All, 3]]] // MatrixForm

$$\begin{pmatrix}
-50.0000000000 - 9.0000000000\,i \\
9.72973533214 + 4.85919787657\,i \\
8.01129618821 + 2.40061917460\,i \\
7.17363545084 + 1.14260581276\,i \\
6.80360054857 + 0.45843960941\,i \\
6.75580075794 + 0.03968130585\,i \\
6.92319606253 - 0.01316818008\,i \\
6.90416954870 - 0.00037292946\,i \\
6.90401745975 - 8.688 \times 10^{-8}\,i \\
6.90401754853 + 0. \times 10^{-14}\,i \\
6.90401754853 \\
6.90401754853
\end{pmatrix}$$

With this complex starting value we have gotten back to the previous real solution in **root2** since all of the imaginary parts of the solution above are all very close to zero. You should try more complex starting guesses and see if they all give these two real solutions or not. Do you believe that this system of equations has only two real solutions and no complex ones?