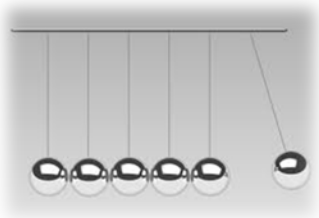
CEE 212—Dynamics

# Newton's Method

*solving nonlinear algebraic equations*



Keith D. Hjelmstad

**The Mechanics Project**
*Arizona State University*

**Contents.**

# Newton's Method

*The basic idea*


**The basic idea**. To get going with the concept of Newton's method for solving *nonlinear algebraic equations* let us consider a single equation in a single unknown. We write this equation as

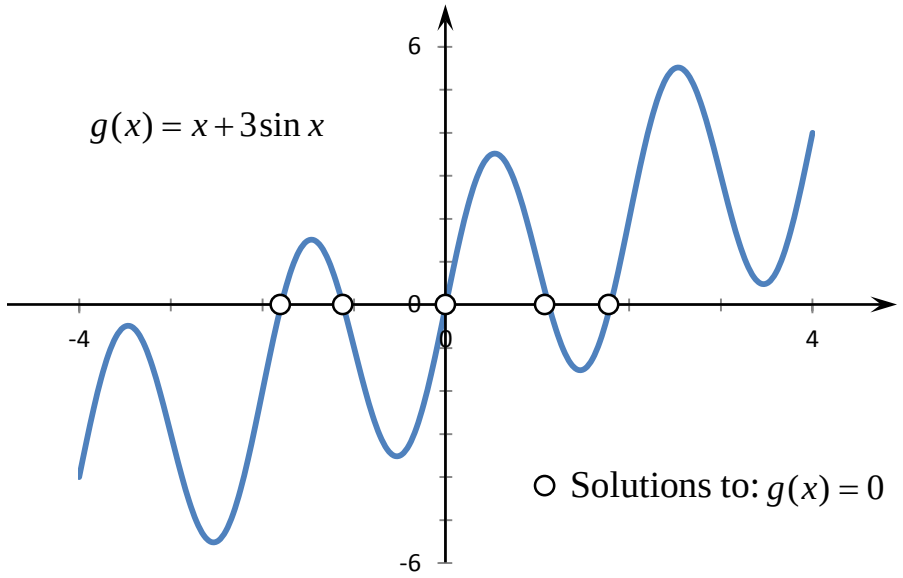
$$g(x) = 0$$

This shorthand notation basically covers all equations of this type. An example of an equation (which we will see later in dynamics) is

$$g(x) = x + c\sin x$$

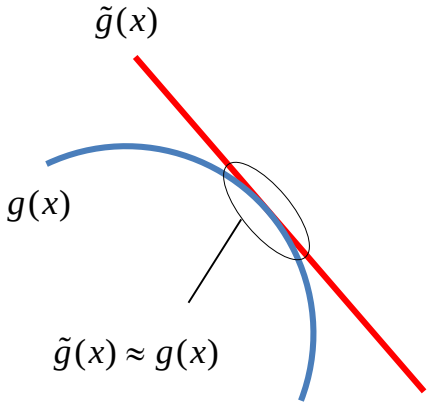
where $c$ is some constant. Notice that there is no algebraic manipulation that allows you to "solve for $x$." So what do we do?

A good way to start the quest of finding solutions is to plot the function and see where it is equal to zero (as shown at right for the example function with $c$ = 3). To do so gives you a clear picture of how many solutions there are and where (roughly) the are located. This approximate location might prove very useful in starting Newton's method as we shall soon see.



# Newton's Method

*Linearization*


**Linearization**. Newton had a great idea. Since we know how to solve linear problems, let's create a linear problem that is close to the nonlinear one. Then solve the linear problem. How do we do that? We know from calculus that every function can be expanded in a Taylor series. Let us expand $g(x)$ around some (known) value $x_o$

$$g(x) = g(x_o) + g'(x_o)\left(x - x_o\right) + \tfrac{1}{2} g''(x_o)\left(x - x_o\right)^2 + \\ + \cdots + \tfrac{1}{n!} g^n(x_o)\left(x - x_o\right)^n + \cdots$$

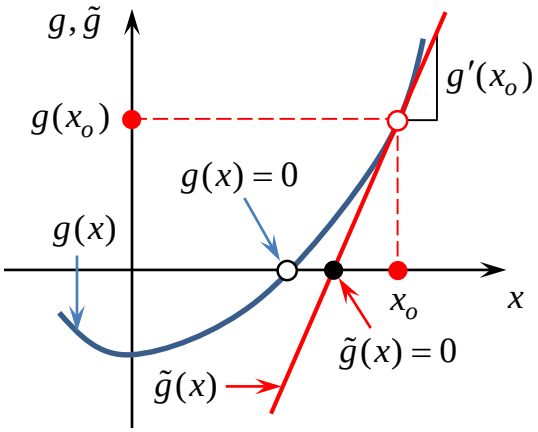
Where a prime denotes differentiation of the function with respect to $x$ (and the superscript $n$ on $g$ means the $n$th derivative). The first term is constant, the second term is linear in $x$, the third term is quadratic, etc. So we can create a linear function "close" to the original nonlinear one by just taking the first two terms of the Taylor series expansion. Let us call that linear function

$$\tilde{g}(x) = g(x_o) + g'(x_o)\left(x - x_o\right)$$

It is a geometric fact that any curve hugs pretty close to a line tangent to it over some distance (as shown at left). So in this neighborhood the linear function will do a good job of representing its nonlinear progenitor. Outside of the region circled? The representation is not so good (and can get really bad, depending on the nonlinear function).

# Newton's Method

*The algorithm*




**The algorithm**. You can get an idea of the relationship between the original function $g(x)$ and the associated linear function

$$\tilde{g}(x) = g(x_o) + g'(x_o)\left(x - x_o\right)$$

from the sketch at left. The original function is blue, the linear function is red. The arbitrarily selected point $x_o$ is shown as a red dot, on the abscissa and the value of the original function $g(x_o)$ is shown as a red dot on the ordinate. The open circle with black edge is the solution we seek. The open circle with red edge is the place where the linear function touches the original nonlinear one. Note that the linear function is tangent to the nonlinear one at that point!

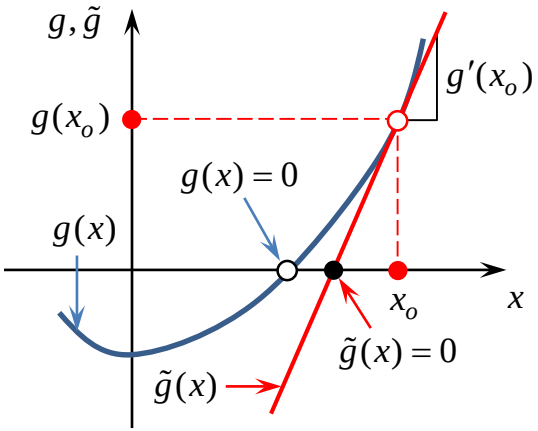
Newton's great idea was to recognize that we could actually solve the linear equation that results from setting the linear function equal to zero:

$$\tilde{g}(x) = 0 \quad \Rightarrow \quad g(x_o) + g'(x_o)\left(x - x_o\right) = 0 \quad \Rightarrow \quad \boxed{x = x_o - \frac{g\left(x_o\right)}{g'(x_o)}}$$

The value of $x$ found in this way is the solid black dot on the abscissa. Newton hypothesized that this guess at the solution we seek was actually better than $x_o$ was. He also realized that you could do this calculation repeatedly taking the newly computed $x$ and using it in place of $x_o$, thereby generating a sequence of values $\{x_o, x_{1\backslash}, x_{2\backslash}, x_{3\backslash}, \ldots, x_n\}$ and that the sequence would converge to the solution to the original nonlinear equation. That is *Newton's method*.

# Newton's Method

*Example*




**Example**. Let us apply Newton's method to the simplest possible nonlinear equation that we can imagine. How about

$$g(x) = x^2 - 3$$

The beauty of the nonlinear equation that results from setting this function equal to zero is that we can actually solve it (spoiler alert—the answer is $\sqrt{3}$=1.732050808). Since we know the answer this will be a good problem to see what is going on. The Newton iteration looks like this

$$x = x_o - \frac{g(x_o)}{g'(x_o)} = x_o - \frac{x_o^2 - 3}{2x_o}$$

From any starting value we can compute a sequence of estimates.

Let's take a look at a specific case. The table below shows how the iteration goes starting from the value $x_o$ = 10. We could probably guess that 10 is not anywhere near the solution we seek, but this shows how quickly Newton's method can home in on the true solution.

| $n$ | $x_n$ | $g(x)$ | $g'(x)$ | $x_{n+1}$ | *error* |
|---|---|---|---|---|---|
| 0 | 10.000000000 | 97.000000000 | 20.000000000 | 5.150000000 | 9.700E+01 |
| 1 | 5.150000000 | 23.522500000 | 10.300000000 | 2.866262136 | 2.352E+01 |
| 2 | 2.866262136 | 5.215458632 | 5.732524272 | 1.956460732 | 5.215E+00 |
| 3 | 1.956460732 | 0.827738595 | 3.912921464 | 1.744920939 | 8.277E-01 |
| 4 | 1.744920939 | 0.044749084 | 3.489841878 | 1.732098271 | 4.475E-02 |
| 5 | 1.732098271 | 0.000164421 | 3.464196542 | 1.732050808 | 1.644E-04 |
| 6 | 1.732050808 | 0.000000002 | 3.464101616 | 1.732050808 | 2.253E-09 |

# Newton's Method

*Systems of equations*


**Systems of equations**. We can derive a similar result for systems of nonlinear equations (which is where Newton's method really has power). In such situations we will have multiple nonlinear equations with several unknowns. In fact, if we have *N* variables we will have *N* equations. The equations have the general form

$$\mathbf{g}(\mathbf{x}) = \mathbf{0}$$

Where the specific details of each of the *N* equations vary from one application to the next. Newton's method is constructed in a similar manner by recognizing that we can create a linear function related to the original function **g** using the Taylor series expansion of the function

$$\tilde{\mathbf{g}}(\mathbf{x}) = \mathbf{g}(\mathbf{x}_o) + \nabla\mathbf{g}(\mathbf{x}_o)\left(\mathbf{x} - \mathbf{x}_o\right)$$

where $\mathbf{x}_o$ is simply a specific value of the vector **x** and the gradient of the vector-value function **g** is simply the matrix of partial derivatives with the rows of the matrix associated with the function $g_i$ and the columns associated with the independent variable $x_i$. The matrix representations of **g** and its gradient are:

$$\mathbf{g}(\mathbf{x}) = \begin{pmatrix} g_1(x_1, x_2, \ldots, x_N) \\ g_2(x_1, x_2, \ldots, x_N) \\ \vdots \\ g_N(x_1, x_2, \ldots, x_N) \end{pmatrix} \qquad \nabla\mathbf{g} = \begin{pmatrix} \frac{\partial g_1}{\partial x_1} & \cdots & \frac{\partial g_1}{\partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_N}{\partial x_1} & \cdots & \frac{\partial g_N}{\partial x_N} \end{pmatrix}$$

# Newton's Method

*Example*


**Example (two equations in two unknowns)**. We can see what is at stake through a simple example. Consider the nonlinear set of equations.

$$g_1(x,y) = x^3 - 3xy + 2y^2 + x - 6 = 0$$
$$g_2(x,y) = y^3 + 2xy - 4x^2 + y + 10 = 0$$

The gradient of the function is

$$\nabla \mathbf{g}(\mathbf{x}) = \begin{bmatrix} \frac{\partial g_1}{\partial x} & \frac{\partial g_1}{\partial y} \\ \frac{\partial g_2}{\partial x} & \frac{\partial g_2}{\partial y} \end{bmatrix} = \begin{bmatrix} 3x^2 - 3y + 1 & -3x + 4y \\ 2y - 8x & 3y^2 + 2x + 1 \end{bmatrix}$$

Just to demonstrate how these things look, let us evaluate the function and the gradient at the (arbitrarily selected) point $\mathbf{x}_o = (1, 2)$, i.e., $x$=1, $y$=2. We get

$$g_1(1,2) = (1)^3 - 3(1)(2) + 2(2)^2 + (1) - 6 = -2$$
$$g_2(1,2) = (2)^3 + 2(1)(2) - 4(1)^2 + (2) + 10 = 20$$

$$\nabla \mathbf{g}(1,2) = \begin{bmatrix} 3(1)^2 - 3(2) + 1 & -3(1) + 4(2) \\ 2(2) - 8(1) & 3(2)^2 + 2(1) + 1 \end{bmatrix} = \begin{bmatrix} -2 & 5 \\ -4 & 15 \end{bmatrix}$$

The upshot of doing this little calculation is simply to show that at any given point, the function and gradient are just numbers. That is what makes it possible to program a computer to do Newton's method. At each step all we will need to do is evaluate the function and the gradient, and then solve a linear system of equations.

## Newton's Method

*Solve the linear equations*


**Solve the linear equations.** So we have found a linear function that hugs the original nonlinear function in the neighborhood of the point $\mathbf{x}_o$. This function is

$$\tilde{\mathbf{g}}(\mathbf{x}) = \mathbf{g}(\mathbf{x}_o) + \nabla\mathbf{g}(\mathbf{x}_o)\left(\mathbf{x} - \mathbf{x}_o\right)$$

If we *set the linear function equal to zero* then we a linear system of equations.

$$\tilde{\mathbf{g}}(\mathbf{x}) = 0 \quad \Rightarrow \quad \mathbf{g}(\mathbf{x}_o) + \nabla\mathbf{g}(\mathbf{x}_o)\left(\mathbf{x} - \mathbf{x}_o\right) = \mathbf{0}$$

The beauty of linear equations is that we can easily solve them. Solving those linear equations we can find a value of $\mathbf{x}$

$$\mathbf{x} = \mathbf{x}_o - \left[\nabla\mathbf{g}(\mathbf{x}_o)\right]^{-1}\mathbf{g}(\mathbf{x}_o)$$

As was true for the scalar version Newton's genius was to recognize that this new value of $\mathbf{x}$ is a better estimate of the solution than $\mathbf{x}_o$ was. With this new value you can repeat the process (i.e., use the newly found $\mathbf{x}$ where we had used $\mathbf{x}_o$ before, compute the residual $\mathbf{g}(\mathbf{x}_o)$ and the gradient, and solve again. This calculation can be done repeatedly, until we arrive at an estimate $\mathbf{x}_n$ after $n$ repetitions of the process.

When should we stop? We will never get the exact answer, but we will get close. Remember that the goal was to solve the equation $\mathbf{g}(\mathbf{x}) = \mathbf{0}$. So, if the norm of the residual vector $\mathbf{g}(\mathbf{x}_n)$ is small enough (less than some preset tolerance *tol*) then we are probably close enough. Newton's method is not guaranteed to converge from any arbitrary starting point, so in our programs we will probably want to specify a maximum number of Newton iterations we are willing to try (call it *maxit*). We stop if the iteration has converged or if we have exhausted the maximum number of iterations we have specified.

# Newton's Method

*Example*


**Example (two equations in two unknowns)**. The calculation can be done in MATLAB as shown in the following script.

```
%... Newton's method example
     Z = [1; 2]; n = 0;
     itmax = 20; tol = 1.e-8; err = 1.0;

     while ((err > tol) && (n < itmax))
         fprintf('%5i%18.12f%18.12f%15.3e\r', n, Z(1), Z(2), err)
         n = n + 1;
         x = Z(1); y = Z(2);
         g     = [   x^3 - 3*x*y + 2*y^2 + x - 6 ;...
                     y^3 + 2*x*y - 4*x^2 + y + 10 ];
         A     = [ 3*x^2 - 3*y + 1 , -3*x + 4*y  ;...
                     2*y - 8*x     ,  3*y^2 + 2*x + 1 ];
         Z  = Z - A\g;
         err   = norm(g);
     end % while
```

The variable Z is used for **x** (to keep it distinct from the scalar variable x in the program). The starting values are $x$=1 and $y$=2. The iteration converges in 14 steps to a tolerance of $1.\times10^{-8}$. The MATLAB program produces the results shown at right.

Notice that things get worse before they get better, but Newton finally converges once it gets on the right track. We could have picked better starting values.

```
    0    1.000000000000    2.000000000000     1.000e+000
    1  -12.000000000000   -2.800000000000     2.010e+001
    2   -6.021091715729  -35.379440188591     1.904e+003
    3   -6.868172912488  -23.624905344337     4.406e+004
    4   -5.519256646876  -15.769369207358     1.307e+004
    5   -4.178715179621  -10.527310729493     3.875e+003
    6   -3.040506722518   -7.036276958119     1.149e+003
    7   -2.115792429438   -4.735015921002     3.396e+002
    8   -1.377426961569   -3.278499799339     9.881e+001
    9   -0.816000324097   -2.472691351051     2.715e+001
   10   -0.473272815076   -2.152159597364     6.331e+000
   11   -0.361829741041   -2.077810191156     1.047e+000
   12   -0.352113414041   -2.071714267736     7.470e-002
   13   -0.352044273689   -2.071670060041     5.317e-004
   14   -0.352044270172   -2.071670057777     2.719e-008
```

# Newton's Method

*In dynamics*


**In dynamics**. Most of the time the equations of motion that result from rigid body dynamics are nonlinear. The pendulum equation, for example, is

$$a + B\sin x = 0$$

where $a$ is "acceleration," $x$ is "position," and $B$ is a constant that depends upon the physical properties of the system. The function $\sin x$ is nonlinear. We will deal with the differential relationships through numerical integration and that will give us an equation that relates $x$ to $a$ that has the form

$$x = C + D\,a$$

where $C$ and $D$ are constants that depend upon the numerical integration scheme.

The interesting fact is that the equation that relates position to acceleration will always be a linear equation and we can always use substitution into the original equation to get a single equation in the unknown $a$. Once we have done that then we arrive at the computational task of finding $a$ , which means solving a nonlinear algebraic equation. Newton's method will help us here.

In a future set of notes we will take a look at the numerical integration process in more detail. But at this point what we need to know about it is that it will be a (seemingly magical) method to convert differential equations into algebraic equations. Newton's method will be the tool we pull out to deal with the resulting equations (and, as such, does not really have anything to do with numerical integration).

# Newton's Method

*Summary*


**Summary.** The solution of a nonlinear system of algebraic equations is a task at the core of (almost) every one of the problems that come up in rigid body dynamics. We can develop a systematic approach to problems that include Newton's method at the core of the calculation.

When used in conjunction with a time-stepping algorithm based on the trapezoidal rule for integrating for velocity and position the *equation of motion* is the only nonlinear equation that appears. If we substitute the trapezoidal integration equations into the equation of motion we reduce the calculation to an iterative search for the new accelerations that satisfy the equations of motion. We will explore this more in the next set of notes.

There are a few features of Newton's method that are *important for all implementations*:

1. Newton's method always involves the formation of a residual (**g**) and a tangent matrix (**A**=$\nabla$**g**). The tangent matrix is the derivative of **g** with respect to the arguments of **g**.
2. Newton's method must always be given a starting guess $\mathbf{x}_o$. Newton's method is not guaranteed to converge, but if the starting guess is close enough to the solution it will. Often there is a very good way to estimate a good starting guess.
3. Newton's method is an iteration. Therefore, it must have conditions set to tell it when to stop. We generally use two: (a) the norm of the residual being less that a preset tolerance, and (2) the number of iterations being fewer than a preset allowable number. Newton's method usually converges quickly and for most of the dynamics problems (where we can select a very good starting point) we will encounter only 3 or 4 iterations are usually needed.