```
> # Alexander Sage
> # Project 3
```

Newton's method was found by Isaac Newton and Joseph Raphson to find the roots of a function. Newton's method is an affective way to find the value of variables that make an associated function equal to the 0 vector for any number of dimensions. If the function is defined over the reals and is a smooth curve at least within a neighborhood of the initail values and the value for the desired root, then Newton's method can be considered.
Taking an analysis of a two dimensional system will show the basic process. The extension to an n dimension system is a simple one using the Jacobian matrix.

With any differentiable two dimensional function a tangent line can be taken
If F(x) is the function F'(x) is the derivative of the function and x_n is any value chosen close to a root of the function, then $F'(x_n)$ is defined as the slope of the tangent line to the function at x_n and $F(x_n)$ is the evaluated 'height' of the function.

If the function at x_n behaves a straight line, then the root of the function is equal to the root of the tangent line. To obtain the actual root of the function, the root of the tangent line can be found repeatedly until it is almost equivalent to the root of the function.

How does one go from x_n to x_r

by definition of a derivative

$$F'(x_n) = \frac{\Delta y}{\Delta x} \quad \Rightarrow \quad \frac{1}{F'(x_n)} = F'(x_n) \quad \Rightarrow \quad (F'(x_n))^{-1} = \frac{\Delta x}{\Delta y}$$

at x_n Δx is the distance from x_n to x_r

$$x_r = x_n + \Delta x$$

to *get* Δx from $\frac{\Delta x}{\Delta y}$ just multiply by $\Delta y = 0 - F(x_n) = -F(x_n)$

$$x_r = x_n - \frac{\Delta x}{\Delta y} \cdot F(x_n) = x_n - (F'(x_n))^{-1} \cdot F(x_n)$$

then the whole thing is repeated for x_r so it is redifined as the next x_n

$$x_{n+1} = x_n - (F'(x_n))^{-1} \cdot F(x_n)$$

An example of how this process works and how it doesn't is given below.

```
> with(plots):
```

The function $g = x^3$ is picked because it can be easily used as an example for when Newton's method works and doesn't work.

```
> g := x^3
```

$$g := x^3 \qquad (1)$$

The function is defined

```
> gf := x -> x^3
```

$$gf := x \rightarrow x^3 \qquad (2)$$

The derivative is taken

> $dgf := x \rightarrow 3 \cdot x^2$

$$dgf := x \rightarrow 3\ x^2 \tag{3}$$

A number close to the root is guessed ($x_0 = 2$) and the tangent line (s1) at x_0 to $g = x^3$ has an x-axis intercept equal to the next value x_1 .

> $x_0 := 2$

$$x_0 := 2 \tag{4}$$

> $s1 := dgf(x_0) \cdot (x - x_0) + gf(x_0)$

$$s1 := 12\ x - 16 \tag{5}$$

The guessed value x_0=2 is plugged into the formula $x_1 = x_0 - \frac{F(x)}{F'(x)}$ where, x_0 is the initial guess, x_1 is the next x value

> $x_1 := x_0 - \frac{(gf(x_0))}{dgf(x_0)}$

$$x_1 := \frac{4}{3} \tag{6}$$

These two steps are repeated for tangent lines s1 s2 and s3 with the formula $x_{n+1} = x_n - \frac{F(x)}{F'(x)}$

> $s2 := dgf(x_1) \cdot (x - x_1) + gf(x_1)$

$$s2 := \frac{16}{3}\ x - \frac{128}{27} \tag{7}$$

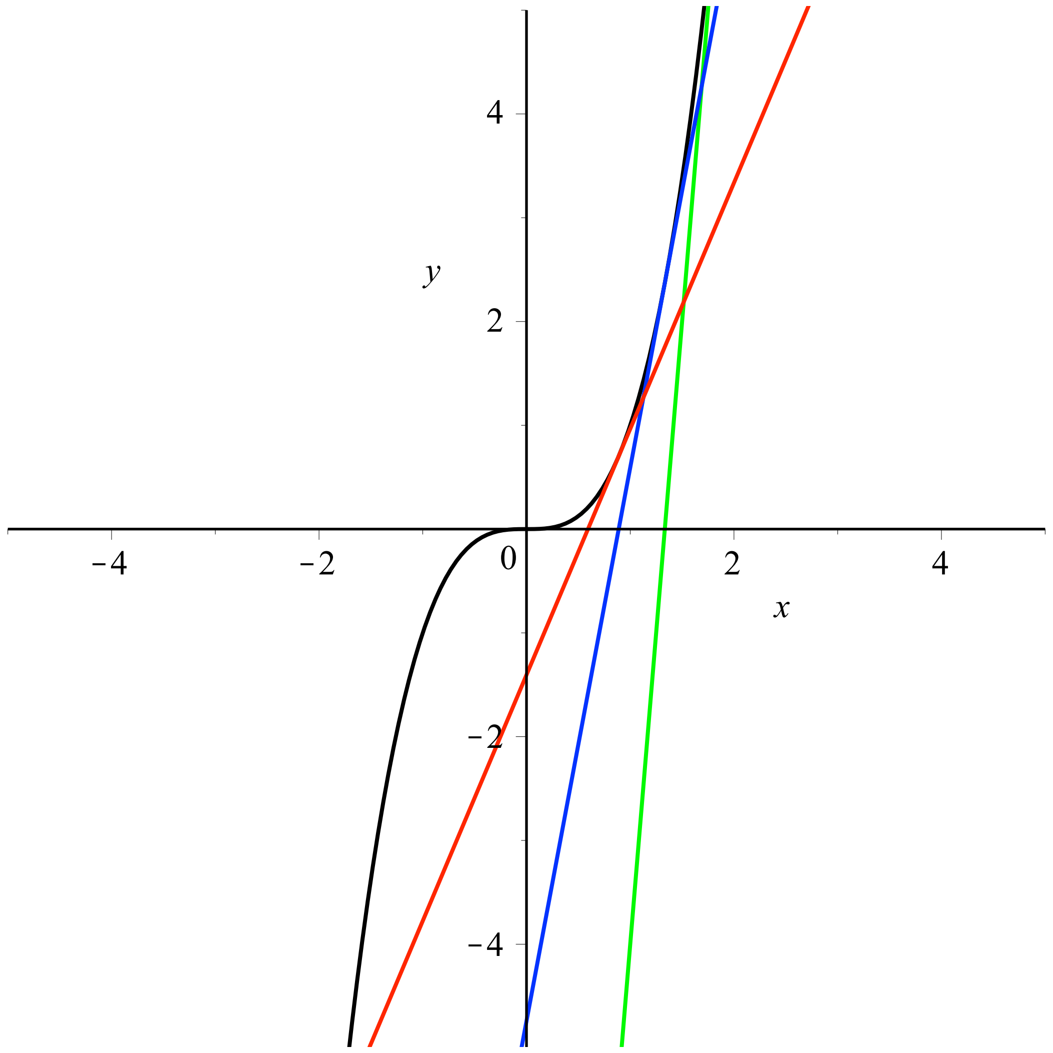
> $x_2 := x_1 - \frac{(gf(x_1))}{dgf(x_1)}$

$$x_2 := \frac{8}{9} \tag{8}$$

> $s3 := dgf(x_2) \cdot (x - x_2) + gf(x_2)$

$$s3 := \frac{64}{27}\ x - \frac{1024}{729} \tag{9}$$

```
> display({plot(s1, x=-5..5, y=-5..5, color=green), plot(s2, x=-5..5, y=-5
    ..5, color=blue), plot(g, x=-5..5, y=-5..5, color=black), plot(s3, x=-5
    ..5, y=-5..5, color=red)})
```



As can be seen from the graph, the first tangent line (green) comes close to the root, the next tangent line (blue) comes closer, and the next (red) comes even closer. The process continues until the x_{n+1} value when plugged into the Funciton is close to 0

The enclosed subsection contains the same process with a new function

> $h := x^3 + 2$

$$h := x^3 + 2 \tag{1.1}$$

> $hf := x \to x^3 + 2$

$$hf := x \to x^3 + 2 \tag{1.2}$$

> $dhf := x \to 3 \cdot x^2$

$$dhf := x \to 3\ x^2 \tag{1.3}$$

> $t1 := dhf(2) \cdot (x - 2) + hf(2)$

$$t1 := 12\ x - 14 \tag{1.4}$$

> $a0 := 2 - \frac{(hf(2))}{dhf(2)}$

$$a0 := \frac{7}{6} \tag{1.5}$$

> $t2 := dhf(a0) \cdot (x - a0) + hf(a0)$

$$t2 := \frac{49}{12}\ x - \frac{127}{108} \tag{1.6}$$

> $a1 := a0 - \frac{(hf(a0))}{dhf(a0)}$

$$a1 := \frac{127}{441} \tag{1.7}$$

> $t3 := dhf(a1) \cdot (x - a1) + hf(a1)$

$$t3 := \frac{16129}{64827}\ x + \frac{167435476}{85766121} \tag{1.8}$$

> $a2 := a1 - \frac{(hf(a1))}{dhf(a1)}$

$$a2 := -\frac{167435476}{21338667} \tag{1.9}$$

> $t4 := dhf(a2) \cdot (x - a2) + hf(a2)$

$$t4 := \frac{28034638623346576}{151779569778963}\ x + \frac{940741876695753666 1434278}{971632109074966 5186963} \tag{1.10}$$

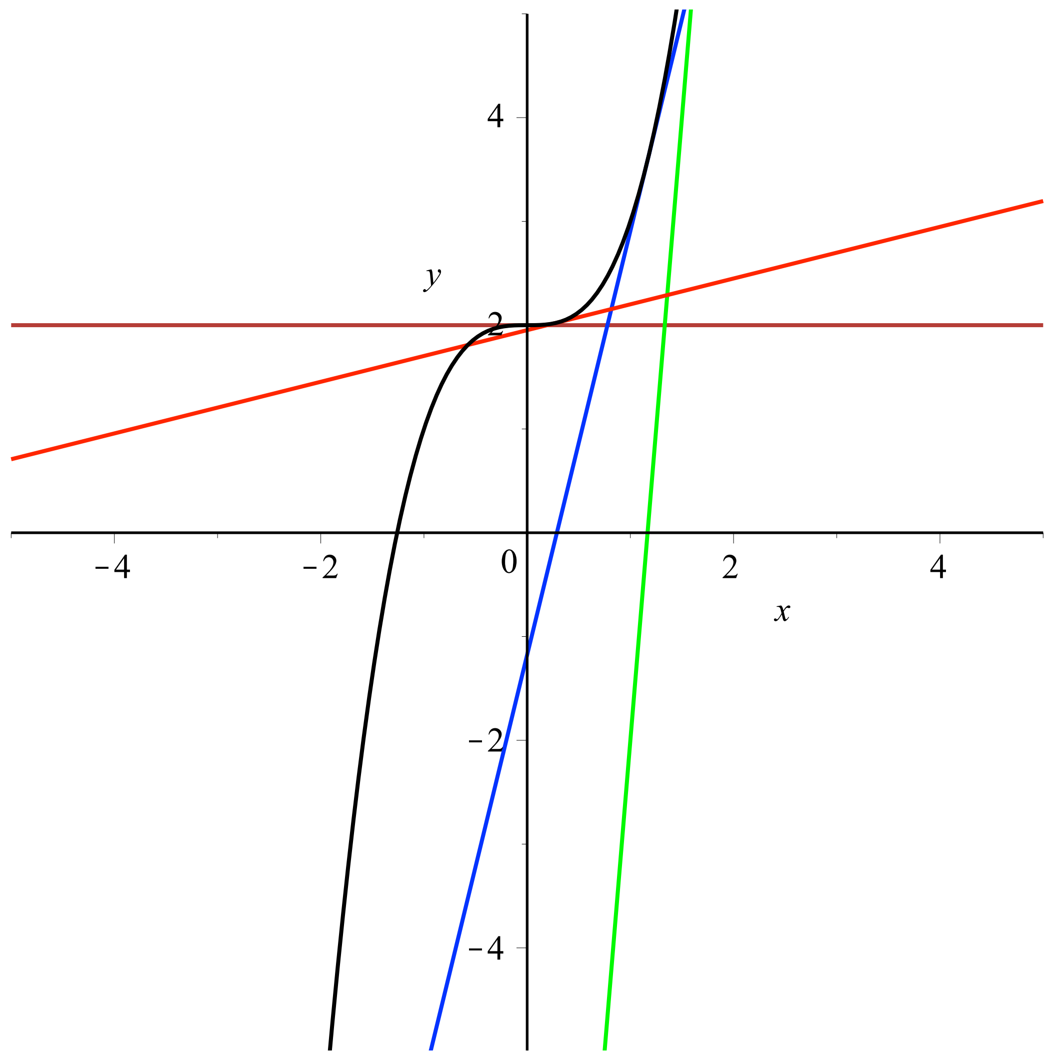
> $hor := 2$

$$hor := 2 \tag{1.11}$$

hor ⇒ is the perfectly horizontal brown line

Below is the function $h = x^3 + 2$ which is slightly different, but enough so that Newton's method is substantially less effective.

```
> display({plot(t1, x=-5..5, y=-5..5, color=green), plot(t2, x=-5..5, y=-5
    ..5, color=blue), plot(h, x=-5..5, y=-5..5, color=black), plot(t3, x=-5
    ..5, y=-5..5, color=red), plot(t4, x=-5..5, y=-5..5, color=yellow),
     plot(hor, x=-5..5, y=-5..5, color=brown)})
```



The last tangent line isn't visible on the screen because the previous tangent line (in red) deviates too far from the desired root. The x2 value (from the red line) is far on the left side of the graph and even though the code will continue until it finds the correct answer (now approaching from the left), this method becomes quite inifficient. This type of problem will occur everytime the tangent line is evaluated near an x_n value where the derivative is equal to 0. The brown line illistrates how if Newton's method attempts to run at that specific x_{sp} value (where F'(x_{sp})=0), then there will be no x-intercept and the code won't be able to divide by 0.

To extend this idea to a Multivariable system
such as the one described by these equations

$$> \left[3\cdot x1-\cos(x2\cdot x3)-\frac{1}{2}=0,\ x1^2-81\cdot(x2+0.1)^2+\sin(x3)+1.06=0,\ \exp(-x1\cdot x2)+20\cdot x3+\frac{10\cdot \mathrm{Pi}}{3}-1=0\right]:$$

with initial conditions are set and used as x_0

$$> \begin{bmatrix} 0.1 \\ 0.1 \\ -0.1 \end{bmatrix}$$

The equations are changed to functions in vector form for F(x)

$$F(x) := \begin{bmatrix} (x1,\ x2,\ x3)\rightarrow 3\ x1-\cos(x2\ x3)-\frac{1}{2} \\ (x1,\ x2,\ x3)\rightarrow x1^2-81\ (x2+0.1)^2+\sin(x3)+1.06 \\ (x1,\ x2,\ x3)\rightarrow e^{-x1\ x2}+20\ x3+\frac{10}{3}\ \pi-1 \end{bmatrix}$$

The Jacobian of this function is found for F'(x)

$$F'(x) := [[(x1,\ x2,\ x3)\rightarrow 3,\ (x1,\ x2,\ x3)\rightarrow \sin(x2\ x3)\ x3,\ (x1,\ x2,\ x3)\rightarrow \sin(x2\ x3)\ x2], [(x1,\ x2,\ x3)\rightarrow 2\ x1,\ (x1,\ x2,\ x3)\rightarrow -162\ x2-16.2,\ (x1,\ x2,\ x3)\rightarrow \cos(x3)], [(x1,\ x2,\ x3)\rightarrow -x2\ e^{-x1\ x2},\ (x1,\ x2,\ x3)\rightarrow -x1\ e^{-x1\ x2},\ (x1,\ x2,\ x3)\rightarrow 20]]$$

The Jacobian matrix is inverted and then the equation is used just as before
$x_{n+1}=x_n-\left[F'\left(x_n\right)\right]^{-1}\cdot F\left(x_n\right)$ where
x_n, x_{n+1}, and $F\left(x_n\right)$ *are vectors* and $\left[F'\left(x_n\right)\right]^{-1}$ *is a matrix.*

How the proc prcNewtonsys works:
It first takes in a system of equations, second a list of variables (e.g.[x1, x2, x3...]), third a list of initial values for the variables.
The actual number of dimensions passed in is irrelavent to the functions called in the code. The proc will find the root(s) for any number of dimension.
There are two optional arguements that can be passed in;
epsilon => once the proc evaluates a value of F(x_n) whose distance to the 0 vector is smaller than epsilon the proc stops.
maxlps => the maximum number of times the proc will loop.
The function will produce an error if it cannot reach a solution withing the maximum number of loops or if the jacobian matrix is singular.

```
> prcNewtonsys := proc( )
  ########################################################
  # arguments list
  # lstftnsys   a list of system RHS is 0
  # lstvars     a list of variables  [x1, x2...]
  # lststrpt    a list of start point like [x1=a, x2=b,...]
  # epsilon   ϵ    (option)  ϵ = 10^-7 by default
  # maxlps  maximun loop times (option)   maxlps =100 by default
  ########################################################
  local lstftnsys, lstftn, ftn, ftni, dftn, dftni, y, lstvars, lststrpt,
         epsilon, maxlps, i, lstxi, xi;

  ########################################################
  #  start arguments handling
  ########################################################
  if nargs > 5 then
      error "expecting four arguments at most";
  elif nargs < 3 then
      error "expecting three arguments at least";
  end if;

  lstftnsys := args[1];
  lstvars := args[2];
  lststrpt := args[3];

  if nargs = 3 then
       epsilon := 10^-7;
       maxlps := 100  ;
  elif nargs = 4 then
       epsilon := args[4];
       maxlps := 100  ;
  else
       epsilon := args[4];
       maxlps := args[5];
  end if;

   #####################################
  # end arguments handling
  ########################################

  with(VectorCalculus):
  with(LinearAlgebra):

 lstftn := map(lhs, lstftnsys);
  ftn := map(unapply , lstftn, op(lstvars));

  dftn := VectorCalculus[Jacobian](ftn(op(lstvars)), lstvars);
  dftn := map(unapply , dftn, op(lstvars));

  lstxi := lststrpt;
  xi := convert(lstxi, Vector);
  i := 0;

  do
    ftni :=   evalf(ftn(op(lstxi)));
    dftni := map(apply, dftn, op(lstxi));
    ftni := convert(ftni, Vector);
    dftni := convert(dftni, Matrix);

   #### add your code here #####################
    if (Determinant(dftni) ≠ 0) then
        y := -MatrixInverse(dftni).ftni;
        xi := xi − MatrixInverse(dftni).ftni;
     else

    error the matrix is "not" invertible please input different initail values;
     end if;


   lstxi := convert(xi, list);
   i := i + 1;
   if i > maxlps then    #if It taken too many steps then stop it
      error solution `not` found;

   end if;
   if norm(y, ∞) < epsilon then
      break;
   end if;
 end do;
  return convert(lstxi, Vector), ftni;
 end proc:
```

> lstftnsys := $\left[3 \cdot x1 - \cos(x2 \cdot x3) - \frac{1}{2} = 0,\right.$

$x1^{2} - 81 \cdot (x2 + 0.1)^{2} + \sin(x3) + 1.06 = 0,$

$$\exp(-x1 \cdot x2) + 20 \cdot x3 + \frac{10 \cdot \mathrm{Pi}}{3} - 1 = 0\Big]:$$

```
lstvars := [x1, x2, x3]:
lststrpt := [0.1, 0.1, -0.1]:
>
>
>
>
> prcNewtonsys(lstftnsys, lstvars, lststrpt);
```

$$\begin{bmatrix} 0.500000000000000 \\ 5.26191604463450\ 10^{-12} \\ -0.523598775499869 \end{bmatrix}, \begin{bmatrix} 2.12265760524133\ 10^{-11} \\ -1.25430799080561\ 10^{-8} \\ 1.79465331484607\ 10^{-11} \end{bmatrix} \qquad (10)$$

The first vector is the x1, x2, and x3 input values that cause the function to equal ≈0
The second vector is the value output of the function when x1, x2 and x3 are plugged in, and it is clearly close to the 0 vector.