Example for Secant Method... start the chapter on linear algebra...

from last time...

Thus $e_{n+1} \approx \frac{f''(p)}{2f'(p)} e_n^2 = M e_n^2$ where $M = \frac{f''(p)}{2f'(p)}$

That is $e_{n+1} \approx M e_n^2$ quadratic convergence...

since $f'(p)$ is in the denominator, we need $f'(p) \neq 0$ for the quadratic convergence to hold.

Recall we also discussed why $e_{n+1} \approx M e_n^2$ implies the number of significant digits doubles at each iteration...

It is sometimes said that Newton's method doubles the number of significant digits at each iteration. This can be explained as follows: Let

$$\alpha = \log_{10}(5M|a|) \qquad \text{so that} \qquad 10^{\alpha} = 5M|a|.$$

Suppose x_n is accurate to k significant digits. By the definition this means

$$\frac{|x_n - a|}{|a|} \leq 5 \times 10^{-k}.$$

Now

$$\frac{|x_{n+1} - a|}{|a|} \leq \frac{M|x_n - a|^2}{|a|} = M|a|\left(\frac{|x_n - a|}{|a|}\right)^2$$
$$\leq M|a|(5^2 \times 10^{-2k}) = 5 \times 10^{\alpha - 2k}$$

implies x_{n+1} is accurate to $2k - \alpha$ significant digits. Provided k is large compared to α this is about twice the number of significant digits that were accurate in x_n. Since $k \to \infty$ as $x_n \to a$, it is natural to assume that k is very large compared to α. Therefore Newton's method about doubles the number of significant digits between each iteration.

reasonable to check the rate of convergence numerically to track down bugs in the code...

Check rate of convergence...

```
julia> include("newton.jl")
n=1, xn=2.5221887802138703
n=2, xn=2.5425691666820014
n=3, xn=2.5426413568569757
n=4, xn=2.5426413577735265
n=5, xn=2.5426413577735265
```

8
4
9
all

doubling in the significant digits as good as

converged to the digits available

Can see the doubling of significant digits better if using more precise arithmetic...

```
# newton.jl -- Perform five iterations of Newton's method
f(x)=5*x*exp(-x)-1
x0=big(2.0)

using Symbolics
@variables t
D(g)=expand_derivatives(Differential(t)(g))
as="df(t)="*string(D(f(t)))
eval(Meta.parse(as))

xn=x0
for n=1:7
   global xn=xn-f(xn)/df(xn)
   println("n=",n,", xn=",xn)
end
```

```
julia> include("newton.jl")
n=1, xn=2.5221887802138699545539145078849984373639368858896305351825744354954852407841752
n=2, xn=2.5425691666820012932906501991292208879669915256984204076710787757850013606376 7
n=3, xn=2.5426413568569757212300296254823981192300813136909872971407164071447532663889 5
n=4, xn=2.5426413577735264241460549916424281638000870501122156924436825125592045189049 2
n=5, xn=2.5426413577735264242938061566618482901576353626125931587701304088788632132035 8
n=6, xn=2.5426413577735264242938061566618482901614749075294317671169346997933525354831 7
n=7, xn=2.5426413577735264242938061566618482901614749075294317671169346997933525354831 7
```

converged

doubling of # of significant digits over a wider range of precision...

Note doubling # of significant digits yield lots of digits pretty fast...

Secant Method

n

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

· have assured convergence but the process is si

```
julia> include("secant.jl")
n=2, xn=2.5398155381501226386936018786308437552666027539875777841518601268248712017778242
n=3, xn=2.5424487759402535252198634276026955994303803245490337193198775811807085291058833
n=4, xn=2.5426412621910913879874925820184559411186733572902723049357471194421351436464137
n=5, xn=2.5426413577702891976836594028002932413118549008901771555214320932928223231360437
n=6, xn=2.5426413577735264242393848333904268920179933470954055522116317824568828849843513
n=7, xn=2.5426413577735264242938061566618173045462328585786348274556121661944142349777883
n=8, xn=2.5426413577735264242938061566618482901614749075294314705333254707220126912125773
n=9, xn=2.5426413577735264242938061566618482901614749075294317671169346997933525354831717
```

seek k such that $e_n \sim e_{n-1}^k$; then $e_{n+1} \sim e_n^k \sim e_{n-1}^{k^2}$ and $e_{n-1}e_n \sim e_{n-1}^{k+1}$, so that we deduce $k^2 \approx k+1$, whence $k \approx (1+\sqrt{5})/2 \approx 1.618$. The speed of convergence

expect $e_{n+1} \approx M e_n^{1.618}$

expect 60% more significant digits each iteration...

```
julia> 32/20
1.6

julia> 20/12
1.6666666666666667

julia> 12/7
1.7142857142857142
```

Since the quotients are close to 1.618 (especially 32/20) then the rate of convergence seems correct...

for next time start reading about Gaussian elimination and LU factorization in the the book Chapters 2.1 — 2.3.

Code for this output

```
# secant.jl -- Perform eight iterations of Secant method
f(x)=5*x*exp(-x)-1
x0=big(2.0)

xn=x0
xnm1=x0+big(1)/big(2)
xnp1=0
for n=2:9
   global xnp1=xn-f(xn)*(xn-xnm1)/(f(xn)-f(xnm1))
   global xnm1=xn
   global xn=xnp1
   println("n=",n,", xn=",xn)
end
```