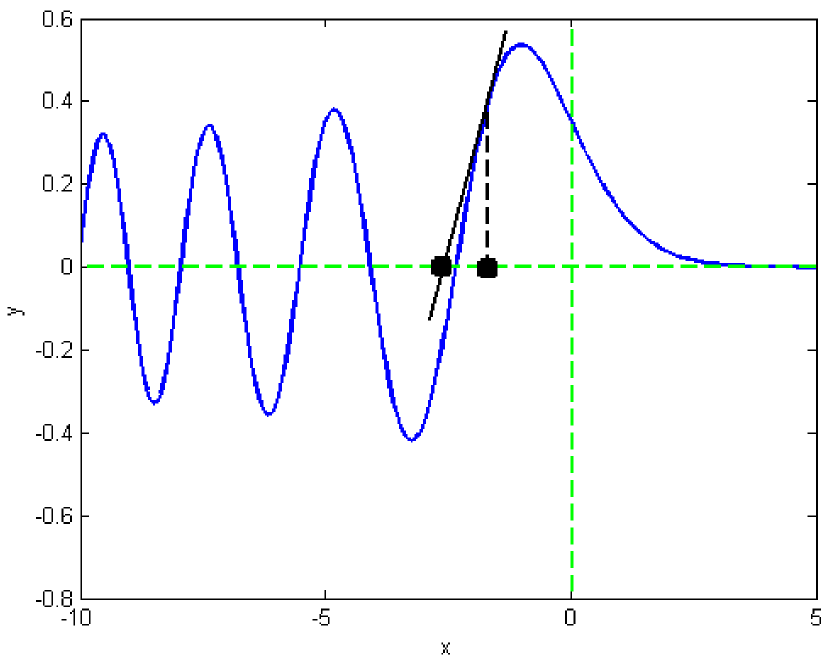
Chapter 4
Rootfinding



The root finding problem

- There are many times when we need to find one or more values of a variable that satisfy a nonlinear equations
- Roots of polynomials are one example: finding eigenvalues, applications in vibrations, control, and many other fields
- In that case, we need to find x such that $P_n(x) = 0$, with

$$P_n(x) = a_1 x^n + a_2 x^{n-1} + \cdots a_n x + a_{n+1}$$

when written like it's used in Matlab

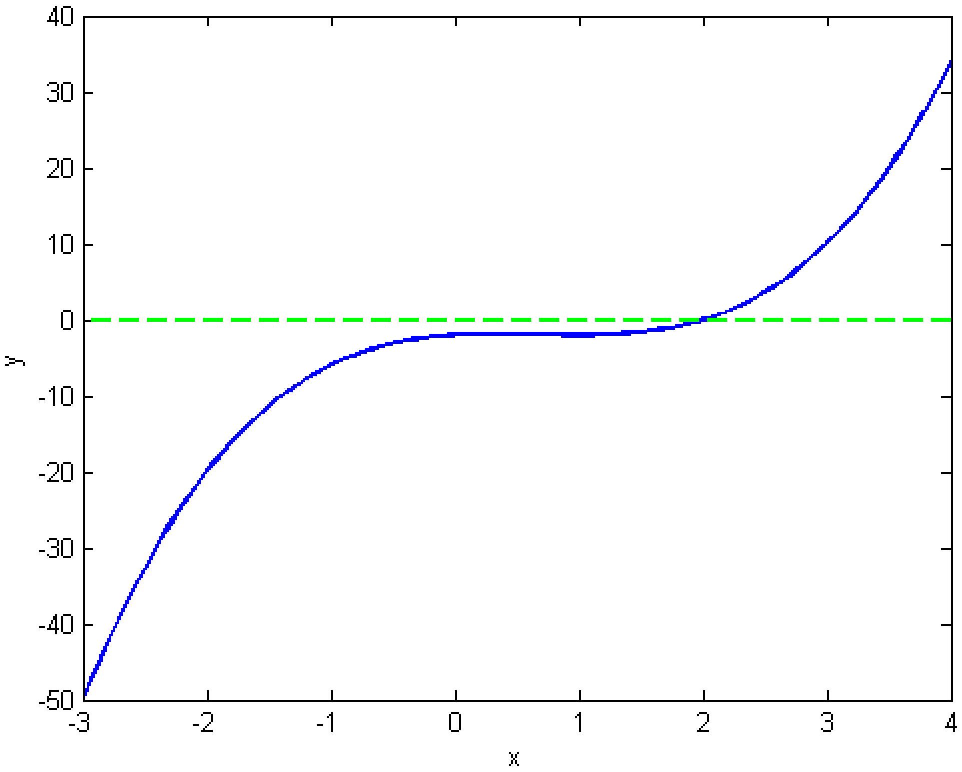
- Even for this simple case, we should, when possible, make a sketch or plot! You can see where the answers are, if any!

The root finding problem

- Consider this example:

$$P_3(x) = (x^2+1)(x-2) = 0.$$

- Call the root(s) p.
- Cubic polynomial, so three roots.
- But, only one is real-valued.
- If we only wanted real roots as in a possible engineering or physics problem, we wouldn't need to waste time looking for others
- Knowing where it is graphically is not solving the problem

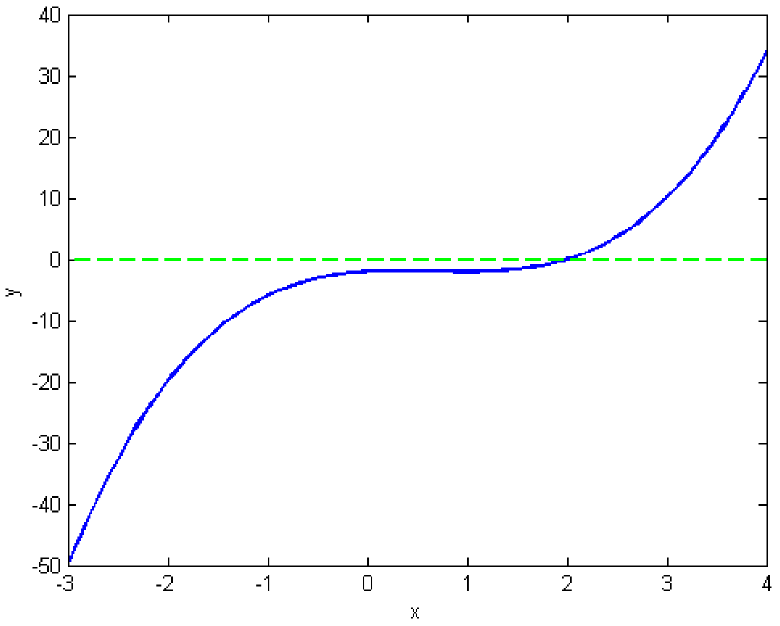


The root finding problem

- For this example:

$$P_3(x) \;=\; (x^2+1)(x-2) = 0.$$

- Call the root(s) p.
- The three roots are shown from the `roots` command in Matlab
- Note the form of the last two as $\pm i$
- Matlab's function does a good job of dealing with the poor conditioning of finding roots.




```
>> a = [1 -2 1 -2]
a =
    1   -2    1   -2
>> p=roots(a)
p =
   2.0000 + 0.0000i
   0.0000 + 1.0000i
   0.0000 - 1.0000i
```

The root finding problem

- Consider this example:

$$P(x) = \prod_{j=1}^{10} (x - j)$$

- Roots are integers 1 to 10
- However, look at the expanded form since we never get them like this

```
>> a = [1 -55 1320 -18150 157773 -902055 3416930 -8409500 12753576 -10628640 3628800];
>> roots(a)'
ans =
  Columns 1 through 8
   10.0000    9.0000    8.0000    7.0000    6.0000    5.0000    4.0000    3.0000
  Columns 9 through 10
    2.0000    1.0000
```

- This works great, but what if we perturb the coefficients a little?

The root finding problem

- Now modify the 9th degree coefficient a tiny bit ($a_2 = -55$)
- What happens?

```
>> b = a+[0 -1e-8 zeros(1,9)];
>> % tiny perturbation to ninth-degree coefficient
>> roots(b)
ans =
  10.0000
   8.9999
   8.0001
   6.9999
   6.0000
   5.0000
   4.0000
   3.0000
   2.0000
   1.0000
```

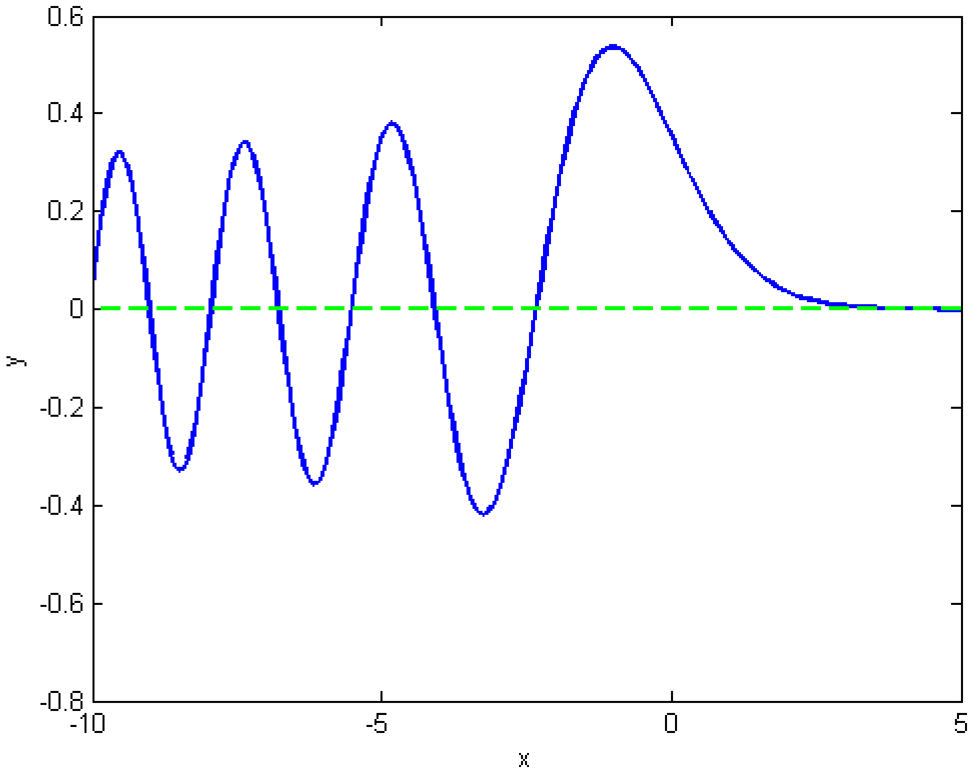
```
>> b = a+[0 -1e-7 zeros(1,9)];
>> roots(b)
ans =
  10.0003
   8.9990
   8.0013
   6.9991
   6.0004
   4.9999
   4.0000
   3.0000
   2.0000
   1.0000
```

```
>> b = a+[0 -1e-6 zeros(1,9)];
>> roots(b)
ans =
  10.0027
   8.9903
   8.0133
   6.9907
   6.0035
   4.9993
   4.0001
   3.0000
   2.0000
   1.0000
```

- You try it. Keep increasing the perturbation
- If uncertainty/noice in coefficients, use with caution!

The root finding problem

- There are lots of different functions that have zeros
- So-called "special functions" are solutions to variable coefficient ODE problems that arise typically from PDE problems
- Text example of Bessel function arises from vibration of circular drum head
- Another example is Airy function Ai(x) that satisfies y''-xy=0.
- Can you see why one side oscillates?



The root finding problem

- To have a baseline for testing accuracy, we will use Matlab's builtin function `fzero`
- Say we want to find the root p closest to zero such that $\mathrm{Ai}(p) = 0$
- To do this, we should give an initial guess near the root, specify the function (Matlab builtin `airy` here)
- We can see p, $f(p)$ and a "flag" telling us the answer status

```
>> [p,fval,exflag] = fzero(@airy,-2)
p =
  -2.3381
fval =
 -9.3595e-17
exflag =
     1
```

1	**fzero** found a zero X.
-1	Algorithm terminated by output function.
-3	NaN or Inf function value encountered during search for an interval containing a sign change.
-4	Complex function value encountered during search for an interval containing a sign change.
-5	**fzero** may have converged to a singular point.
-6	**fzero** can not detect a change in sign of the function.

The root finding problem

- For the Airy function we had a good approximation to the root
- How sensitive is it?
- Sensitivity for an ideal computer: condition number
- We can see p, $f(p)$ and a “flag” telling
- First, we need a norm: Use $||g||_\infty = \max_{x \in I} |g(x)|$
- The interval I will depend on context; for root finding it will most often be an interval containing the sequence of approximations to the answer, called iterates
- The same kinds of properties hold for this norm as for vectors: triangle inequality (add’n), Schwarz’ inequality (mult’n), etc

Conditioning of root finding

- Say we want to solve $f(x) = 0$, but f is perturbed by a function $h(x)$ so that the computed function is
$$\tilde{f}(x) = f(x) + \epsilon h(x)$$
- Say the root r is perturbed a little bit,
$$\tilde{r} = r + \delta s$$
- We assume that $\epsilon, \delta \ll 1$, and Taylor expand $\tilde{f}(\tilde{r}) = 0$
- We get $0 = f(r + \delta s) + \epsilon h(r + \delta s)$, and expand:

$$0 = f(r) + f'(r)\delta s + \epsilon[h(r) + h'(r)\delta s]$$

- Neglect the product term $\epsilon\delta$ as too small, $0 = f(r)$, then
$$\delta s = \epsilon h(r)/f'(r)$$

Conditioning of root finding

- We want the relative change in the answer
- Multiply both sides of

$$\delta s = \frac{\epsilon h(r)}{f'(r)}$$

- with

$$\frac{||f||_\infty}{|r|||\epsilon h||_\infty}$$

- Then after absolute value

$$\frac{\frac{|\delta s|}{|r|}}{\frac{||\epsilon h||_\infty}{||f||_\infty}} = \frac{\epsilon |h(r)|||f||_\infty}{|f'(r)||r|||\epsilon h||_\infty}$$

- Note that $\epsilon |h(r)| \leq ||\epsilon h||_\infty$, and substituting in the numerator gives

$$\kappa(f \mapsto r) = \frac{||f||_\infty}{|f'(r)||r|}$$

Conditioning of root finding

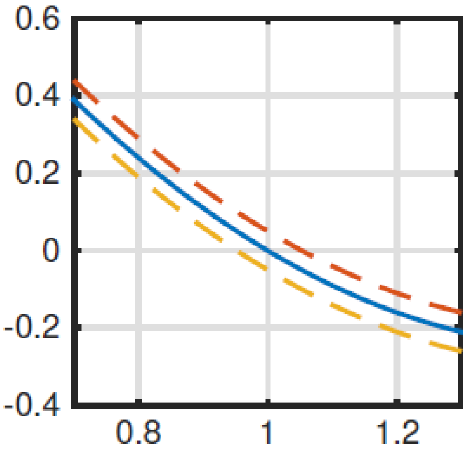
$$\kappa(f \mapsto r) = \frac{||f||_\infty}{|f'(r)||r|}$$

- r in the denominator gives the change relative to the result
- $||f||_\infty$ gives the size of the data (or function)
- If f'(r) is small, small changes from r may result in a big change in the answer.

```
f  = @(x) (x-1).*(x-2);
```

At the root $r = 1$, we have $f'(r) = -1$. If the val
0.05, we imagine finding the root of the function

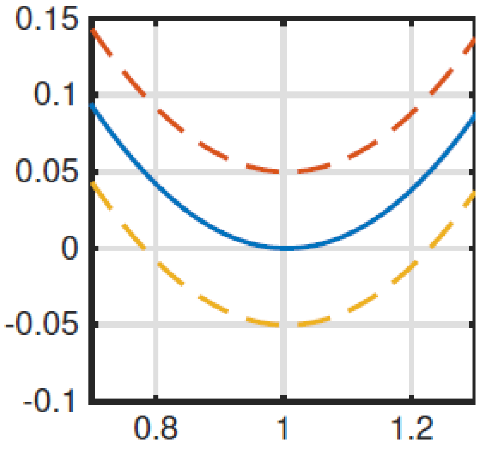
```
interval = [0.7 1.3];
fplot(f,interval), grid on
hold on, axis equal, axis square
fplot(@(x) f(x)+0.05,interval,'--')
fplot(@(x) f(x)-0.05,interval,'--')
```




```
f  = @(x) (x-1).*(x-1.01);
```

Now $f'(1) = -0.01$, and the graph of f will be m
on our thick rendering:

```
clf
fplot(f,interval), grid on
hold on, axis equal, axis square
fplot(@(x) f(x)+0.05,interval,'--')
fplot(@(x) f(x)-0.05,interval,'--')
```



Root finding: Newton’s method

- You have no doubt seen this method somewhere, but we will analyze it in a bit more depth
- We seek $f(p) = 0$ for $x = p$.
- We want to use Taylor’s theorem to linearize the problem near p.
- If we Taylor expand about x near p, we obtain

$$f(p) = f(x) + \frac{f'(x)}{1!}(p - x) + \frac{f''(\xi(p))}{2!}(p - x)^2$$

- The number $\xi(p)$ makes the formula exact.
- To solve the problem approximately, we neglect the quadratic term, which may be expected to work if $|p - x| \ll 1$

Root finding: Newton's method

- Also use $f(p) = 0$ to obtain

$$0 \approx f(x) + \frac{f'(x)}{1!}(p - x)$$

- This is the equation for a line tangent at x, which crosses the x-axis near p, but not at it (if things work right)
- Solving for p,

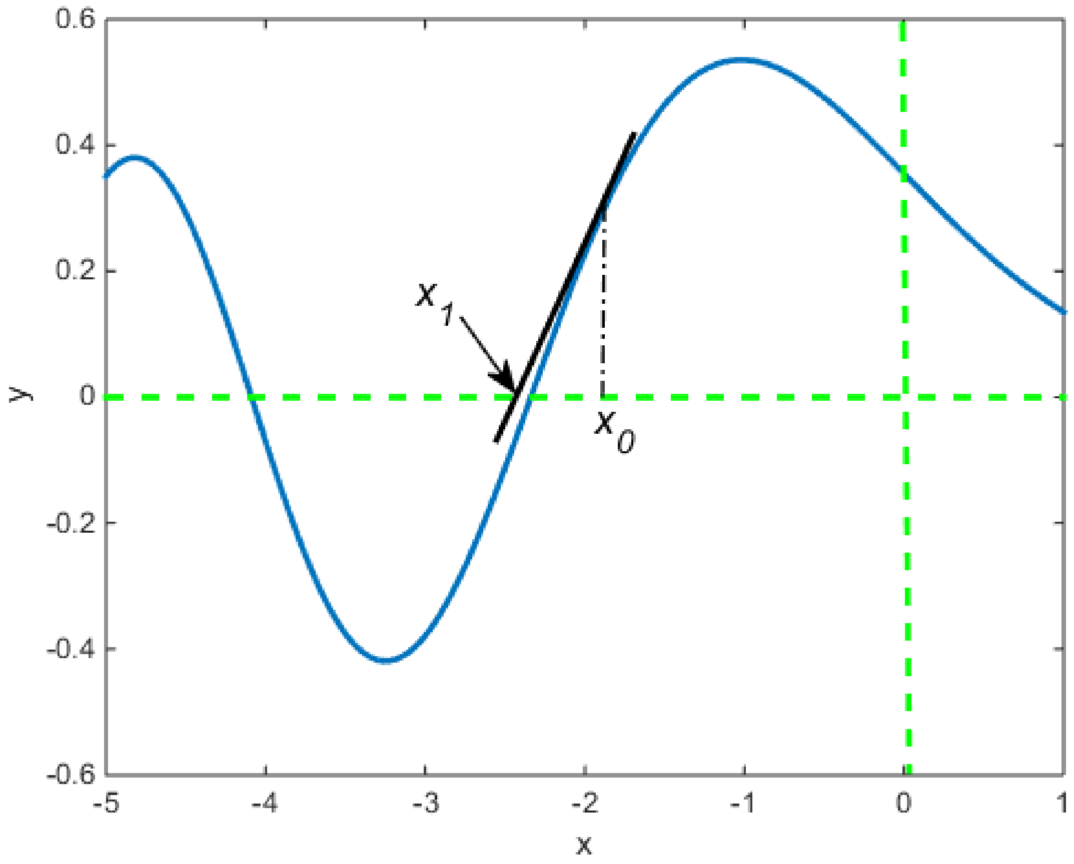
$$p \approx x - \frac{f(x)}{f'(x)}$$

- Because we aren't at the root, we turn this into an iteration. The x we expanded about becomes x_0; the approximate root into x_1:

$$x_1 \approx x_0 - \frac{f(x_0)}{f'(x_0)}$$

Root finding: Newton's method

- x_1 is an approximation as well.
- Using it on the right side, we can generate a new approximation.

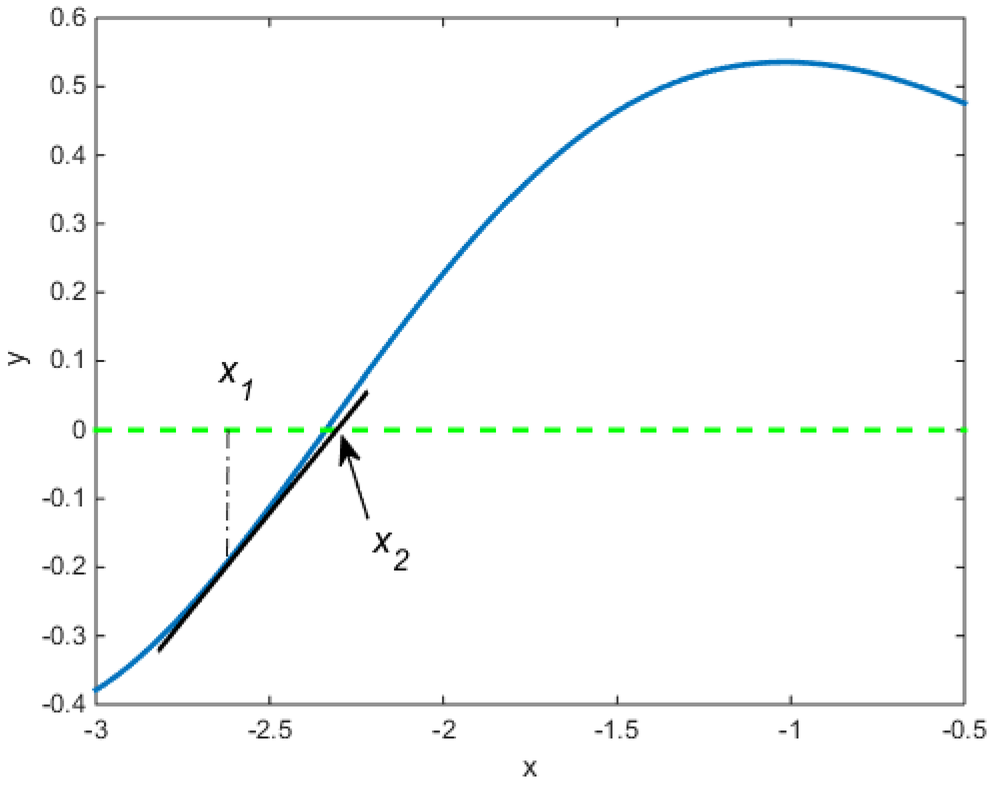


Root finding: Newton's method

- x_1 is an approximation as well. Using it on the right side, we can generate a new approximation.
- Using x_1 as input, we compute a new $f(x_1)$ and $f'(x_1)$, and find the new approximation x_2
- We can repeat this and make it into an iteration:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \qquad n = 0,1, \ldots$$

- Will it work? If so, how long to do this?



Newton’s method: convergence analysis

- Call the root r; we study what happens to the error $e_k = r - x_k$ for n big enough
- We assume that we can make e_n as small as we like
- Using $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \quad k = 0,1,\ldots$
- Subtract r from both sides, and eliminate x_k:

$$e_{k+1} = e_k - \frac{f(r - e_k)}{f'(r - e_k)}, \qquad n = 0,1,\ldots$$

- The arguments of f and f’ are small; Taylor expand them

$$e_{k+1} = e_k + \frac{f(r) - e_k f'(r) + \frac{1}{2}e_k^2 f''(r) + O(e_k^3)}{f'(r) - e_k f''(r) + O(e_k^2)}$$

Newton’s method: convergence analysis

- Now use f(r)=0 in

$$e_{k+1} = e_k + \frac{f(r) - e_k f'(r) + \frac{1}{2}e_k^2 f''(r) + O(e_k^3)}{f'(r) - e_k f''(r) + O(e_k^2)}$$

- Then factor out f’(r); result in denominator can be written as geometric series:

$$e_{k+1} = e_k - e_k \left[1 - \frac{1}{2}\frac{f''(r)}{f'(r)}e_k + O(e_k^2)\right] \left[1 - \frac{f''(r)}{f'(r)}e_k + O(e_k^2)\right]^{-1}$$

- Multiply out the last two terms:

$$\begin{aligned} e_{k+1} &= e_k - e_k \left[1 - \frac{1}{2}\frac{f''(r)}{f'(r)}e_k + O(e_k^2)\right] \left[1 + \frac{f''(r)}{f'(r)}e_k + O(e_k^2)\right] \\ &= -\frac{1}{2}\frac{f''(r)}{f'(r)}e_k^2 + O(e_k^3). \end{aligned}$$

Newton's method: convergence analysis

- This means $|e_{k+1}| \approx C|e_k|^2$, with the approximation getting better as k increases
- This is "quadratic convergence" or "quadratic rate of convergence"
- Number of correct digits doubles with each iteration
- If we take the log of both sides, we get $\log|e_{k+1}| \approx 2\log|e_k| + K$
- This gives us an empirical way to detect rate of convergence
- Consider two sequences:
- $a_k = 2^{-k}, k = 0,1, \ldots$ each term in the sequence is half of previous
- $b_k = 2^{-2^k}, k = 0,1, \ldots$ this time, the exponent doubles every time

Newton’s method: convergence analysis

- Consider two sequences:
- $a_k = 2^{-k}, k = 0,1,\ldots$ each term in the sequence is half of previous
- $b_k = 2^{-2^k}, k = 0,1,\ldots$ this time, the exponent doubles every time
- Try RateOfConv.m for empirical test in log-log plot
- Also look at $\lim_{k\to\infty} \frac{a_{k+1}}{a_k}$ and $\lim\limits_{k\to\infty} \frac{b_{k+1}}{b_k}$

Newton’s method: example

- Consider solving for x where $2 = xe^x$
- We need to write it in standard form: $f(x) = xe^x - 2$
- Then we are solving $f(x) = 0$
- Define f and df/dx

```
f = @(x) x.*exp(x) - 2
dfdx = @(x) exp(x).*(x+1)
```

- Then, we can use fzero to find “exact” error

```
r = fzero(f,1)
```

```
r =
    0.8526
```

- Now use simple loop to iterate the formula:

```
x = 1;
for n = 1:6
    x(n+1) = x(n) - f(x(n))/dfdx(x(n))
end
```

Newton's method: example

- Computing $f(x) = xe^x - 2 = 0$
- The log of the error is at right
- Doubles, roughly for first five iterates
- Doubling stops because we ran out precision in the computer
- Computing the ratio of the logs shows that the slope is two, roughly, for the first five iterations; this is like looking at the ratio of b_n in model sequences

```
logerr = log(abs(e))
```

```
logerr =
  -1.9146e+00
  -4.1816e+00
  -8.6344e+00
  -1.7531e+01
  -3.5351e+01
  -3.6737e+01
  -3.6737e+01
```

```
ratios = logerr(2:end) ./ logerr(1:end-1)
```

```
ratios =
   2.1840e+00
   2.0649e+00
   2.0303e+00
   2.0165e+00
   1.0392e+00
   1.0000e+00
```

Newton's method: Code

- For function, we need f, df/dx, initial guess
- Tolerances in both x and $f(x)$ are set to 100eps
- A max number of iterations is set
- Iterate in while loop until tolerances aren't satisfied
- Stops if too many iterations

```
function x = newton(f,dfdx,x0)
% NEWTON   Newton's method for a scalar equation.
% Input:
%   f        function that outputs value of the function
%   dfdx     function that outputs values of the derivative
%   x0       initial root approximation
% Output
%   x        vector of root approximations (last is best)

% Operating parameters.
funtol = 100*eps;  xtol = 100*eps;  maxiter = 40;

x = x0;
y = f(x0);
dx = Inf;
k = 1;

while (abs(dx) > xtol) && (abs(y) > funtol)
  dydx = dfdx(x(k));
  dx = -y/dydx;   % Newton step
  x(k+1) = x(k) + dx;

  k = k+1;
  if k==maxiter
    warning('Maximum number of iterations reached.')
    break
  end

  y = f(x(k));
end
```

Newton’s method: observations and advice

- If mistakes in df/dx or if multiple roots, then rate of convergence falls to only a linear rate of convergence
- If you see linear convergence, check your functions for mistakes, or multiple roots
- The guess must be close enough to the root to avoid zero slope in the function and to converge quadratically