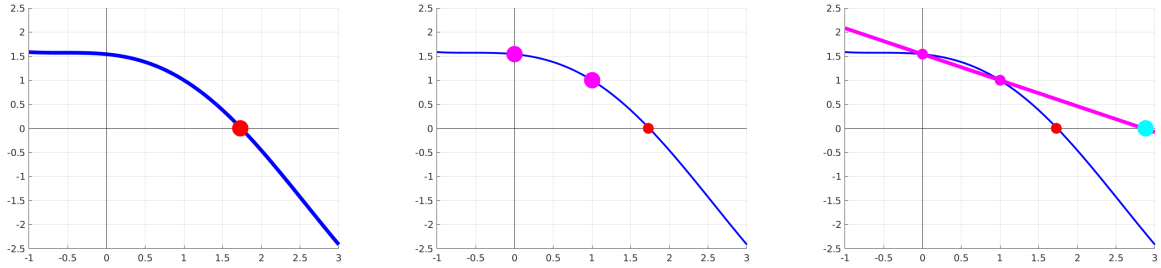
Nonlinear equations: The secant method

MATH2070: Numerical Methods in Scientific Computing I

Location: http://people.sc.fsu.edu/~jburkardt/classes/math2070_2019/nonlinear_secant/nonlinear_secant.pdf

---



Use two samples to guess where a function crosses the x axis

Rootfinding (Secant version)

Given a function $f(x)$ and two estimates for the root, use a linear model to predict a better root.

The bisection and regula falsi methods start and stay within a change-of-sign interval. This has the advantage of a guaranteed bracketing of the solution, but requires the user to supply such an interval at the beginning, and costs a certain amount of internal bookkeeping from step to step.

The secant method drops the change-of-sign requirement, resulting in simpler code, and faster convergence when the initial estimates are close to a root.

1 The secant method is a linear model for $f(x)$

For a given function $f(x)$ whose root we are seeking, suppose we have two sample data points, `(a,f(a))` and `(b,f(b))`. Then the linear function that matches this data has the form:

$$y(x) = \frac{f(a)\,(b-x) + f(b)\,(x-a)}{b-a}$$

If this linear function is a good model for $f(x)$ locally, then we can estimate the root of $f(x)$ by computing the root of $y(x)$:

$$c = \frac{f(a)\,b - f(b)\,a}{f(a) - f(b)}$$

If our new root estimate is not good enough, we can update our data by $a \leftarrow b$ and $b \leftarrow c$ and apply a new secant prediction. As the root estimates get better, the linear model approximates $f(x)$ better, and convergence may be expected to accelerate.

2 MATLAB code for a secant method

Compared to the codes for bisection and regula falsi, the code for the secant method should seem very similar, but also simpler:

```
function b = secant ( f, a, b, xtol, ftol, itmax )

  it = 0;

  while ( true )
    it = it + 1;
    c = ( a * f(b) - b * f(a) ) / ( f(b) - f(a) );
    alpha = abs ( c - b ) / abs ( b - a );
    a = b;
    b = c;
    if ( abs ( f(b) ) <= ftol && abs ( a - b ) <= xtol )
      return
    end
    if ( itmax <= it )
      return;
    end
  end

  return
end
```

Listing 1: secant.m

Because we no longer insist on a change of sign interval, we cannot guarantee that the new estimate `c` will be be between the old estimates `a` and `b`. We cannot guarantee that it will be closer to the root. In fact, in severe cases, the new estimate can be arbitrarily further away, even infinitely so.

3 Example: Secant method for the quadratic equation

```
a = 1.0;
b = 2.0;
xtol = 0.000001;
ftol = 0.000001;
itmax = 50;

[ a, b, it ] = secant ( @(x)quadratic(x), a, b, xtol, ftol, itmax );
```

Listing 2: quadratic_secant.m

The convergence is significantly faster than we saw for the bisection method:

```
   x = estimated root  f(x)
   ------------------  ---------------------
a  0.0                 5.0
b 10.0                -6.08804
1  4.509362338266694  -1.468283915902688
2  2.764289450683286   2.972539956436820
3  3.932383262632530  -0.354202296279007
4  3.808014892908678  -0.044370277430026
5  3.790204406153088   0.001634270584124
6  3.790837107828249  -6.618464845509209e-06
7  3.790834555849615  -9.719784976880419e-10
8  3.790834555474779   1.110223024625157e-15
```

In this example, our two initial points formed a change of sign interval for $f(x)$. We could have chosen starting points for which this was not true. The most important thing is that the starting points are close to the root, so that a linear approximation to the function is reasonable.

4 Estimating the rate of convergence

Suppose that a function $f(x)$ has a root x^*, and that an iterative solution method produces a sequence x_i to this root. Define E_i to be the corresponding sequence of errors,

$$E_i = |x_i - x^*|$$

The method is said to have a convergence rate r if it is the case that

$$\lim_{i \to \infty} \frac{E_i}{(E_{i-1})^r} = C$$

for some finite nonzero constant C.

For the bisection method, we can take E_i to be simply the width of the change-of-sign interval, and with this view, we have that bisection has a linear rate of convergence ($r = 1$) with constant $C = \frac{1}{2}$. The value of r is of much more interest than the value of C.

While determining a convergence rate is a mathematical task, we are naturally interested in a way to computationally estimate r. When we are converging, we can approximate

$$\begin{aligned} E_i &= C\, E_{i-1}^r \\ E_{i-1} &= C\, E_{i-2}^r \end{aligned}$$

and hence

$$\frac{E_i}{E_{i-1}} = (\frac{E_{i-1}}{E_{i-2}})^r$$

from which we can solve for r:

$$r = \frac{\log \frac{E_i}{E_{i-1}}}{\log \frac{E_{i-1}}{E_{i-2}}}$$

Computationally, we can estimate E_i by $x_i - x_{i-1}$. But since we already compute the quantity $\alpha_i = \frac{|x_i - x_{i-1}|}{|x_{i-1} - x_{i-2}|}$ we can make the following approximation to the convergence rate:

$$r \approx \frac{\log(\alpha_i)}{\log(\alpha_{i-1})}$$

Making this calculation requires a few small modifications:

```
1   alpha = 0.0;
2   while ( true )
3     it = it + 1;
4     c = ( a * f(b) - b * f(a) ) / ( f(b) - f(a) );
5     alpha_old = alpha;
6     alpha = abs ( c - b ) / abs ( b - a );
7     if ( 0 < alpha_old )
8       r = log ( alpha ) / log ( alpha_old );
9       fprintf ( 'Estimated convergence rate = %g\n', r );
10    end
```

Listing 3: extract from secant2.m

Here are the sequence of convergence rate estimates that were computed during the treatment of the quadratic function:

```
 it  alpha       log(alpha)  estimated r
 --  ----------- ----------  -----------
  1  0.666667     -0.405465   ------
  2  0.100000     -2.30259    5.6789
  3  0.219512     -1.51635    0.658541
  4  0.028885     -3.54443    2.33748
  5  0.00502376   -5.29358    1.49349
  6  0.000148699  -8.81359    1.66496
```

The secant method seems to be converging at a rate between linear (r=1) and quadratic (r=2). Such behavior is termed *superlinear* convergence. The secant method generally has a convergence rate of $\frac{1+\sqrt{5}}{2} \approx 1.618$ and so our estimates towards the end of our iteration are actually quite reasonable.

The convergence rate does not guarantee that the method will converge, only that, if it does converge, then mathematically the behavior of the sequence of iterates will tend to this limiting pattern. This is called an *asymptotic limit*. So when the secant method works, it works significantly better than the bisection method, whose convergence rate is $r = 1$.

If we look at the entire sequence of estimates for r, then it is likely the early ones are poor estimates, because at that time, α is not a great estimate of the error. In the middle of the sequence, we may hope to see estimated convergence rates near 1.6, but if we take many iterations, the last few steps may be affected by roundoff error, and the corresponding values of r may break the pattern.

5 Class exercise: A scorecard for the secant method

To get a feeling for what can happen, let's try the secant method on some test problems, printing the number of iterations it, and the first and last estimates for the convergence rate r.
Each student should pick a function. Download copies of *secant2.m* and your function to your MATLAB directory. Then issue a command like

```
1  [ a, b, it ] = secant2 ( @(x)f(x), a, b, xtol, ftol, itmax )
```

where you want to replace:

- $f(x)$ by your function;
- a and b by two starting values, between 0 and 10;
- $xtol$ and $ftol$ by small toleranced, say 0.000001 each;
- $itmax$ by an iteration limit, say 50;

Then run the program. If it fails to converge, try using a different pair of starting values a and b. Once you get convergence, please record the initial values of a and b, the number of iterations it, and the final value of r, the estimated convergence rate.

function	a	b	it	estimated r
cubic()	..	..	..	....
hump()	..	..	..	....
kepler()	..	..	..	....
lambert()	..	..	..	....
quadratic()	..	..	..	....
trig()	..	..	..	....
wiggle()	..	..	..	....

Unlike the bisection method, which ruthless approaches the root, the secant iteration can diverge, especially if the starting points are far away from the root, or if the function has oscillations or the derivative is zero near the starting points. However, if the method is successful, we expect to see a convergence rate near 1.6.

6 Rate of convergence for fixed point

The convergence rate of a fixed point method can also be computed. Here is pseudocode for a fixed point function, arranged to look as much as possible like the other codes we have been using. In particular, the pseudocode computes the values of `alpha` and `alpha_old` so that it can estimate the convergence exponent r.

I want you to use this pseudocode to create a MATLAB function that solves fixed point problems, and apply it to several test cases. You will save a lot of time by starting with a code such as *secant2.m* and making the necessary modifications.

```
function x = fixed_point ( f, g, x, xtol, ftol, itmax )

it <-- 0
xold <-- 0
old <-- 0
new <-- 0
alpha <- 0

Loop

  it <-- it + 1
  xold <-- x
  x <-- g(x)
  old <-- new
  new <-- abs ( x - xold )

  alpha_old <- alpha

  if old is not 0, THEN
    alpha <- new / old
    if alpha_old is not 0 THEN
      r <- log ( alpha ) / log ( alpha_old );
      print r

  if new <= xtol and |f(x)| <= ftol, break from loop with success
  if it > itmax, break from loop with failure

End loop

Print x, f(x)
```

Listing 4: extract from fixed_point2_pseudocode.txt

7 Calling your fixed point code

Briefly, once you have written your fixed point code, you can test it with commands like:

```
x <-- Your starting point
xtol <-- Your x tolerance
ftol <-- Your f tolerance
itmax <-- Maximum iterations
x = fixed_point ( @(x)f1(x), @(x)g1(x), x, xtol, ftol, itmax );
```

where you have written function files *f1.m* and *g1.m* for the first case, and similarly *f2.m* and *g2.m* for the second case.

When you run your code, you want to note the solution and the final convergence rate estimate.

8 Assignment #4: Rate of convergence for fixed point

Using the pseudocode above as a guide, write a file *fixed_point.m* that can solve a general fixed point problem for a root of $f(x) = 0$ using the iteration $x = g(x)$.

Consider the following two problems:

1. $f_1(x) = x^3/8 - x^2 + x + 1 = 0$ with $g_1(x) = x^3/8 - x^2 + 2x + 1$ and starting $x = 1.75$;
2. $f_2(x) = -x^3 + 5x^2 - 4x - 6 = 0$ with $g_2(x) = -x^3 + 5x^2 - 3x - 6$ and starting $x = 2.75$;

For each problem, prepare the appropriate input, and then call your fixed point code for a solution. For both problems, note answers to the following questions:

1. What is the solution x?
2. What is your (final) estimate for the rate of convergence r?
3. Can you explain why one problem has a significantly higher rate of convergence? In class, the derivative of $g(x)$ was discussed.

Turn in: your file *fixed_point.m* and your answers to the questions, by Friday, September 20.