Queens College, CUNY, Department of Computer Science

Computational Finance

CSCI 365 / 765

Fall 2017

Instructor: Dr. Sateesh Mane

November 15, 2017

2 Lecture 2

2.1 Bond yield

- We previously saw the formula below, which relates the price of a bond B to its coupon c, face F and yield y:

$$B = \frac{\frac{1}{2}c}{1+\frac{1}{2}y} + \frac{\frac{1}{2}c}{(1+\frac{1}{2}y)^2} + \cdots + \frac{\frac{1}{2}c}{(1+\frac{1}{2}y)^{n-1}} + \frac{F+\frac{1}{2}c}{(1+\frac{1}{2}y)^n}\,. \tag{2.1.0.1}$$

- There are numerous questions to answer about this formula.
- Why are the coupon payments divided by 2? That is because I have assumed the bond pays coupons twice a year, which is typical in the USA. In other countries, bonds sometimes pay coupons quarterly, in which case there would be four cashflow per year, with coupon amounts $c/4$ (and the denominators would contain powers of $(1+\frac{1}{4}y)$).
- Why are the coupon amounts all equal? Answer: they do ***not*** have to be all equal. There are examples of "floating coupon" bonds, where the coupon may be tied to a floating interest rate. The above formula expresses the simplest case, where the coupons are all equal.
- The factors of $1/(1+\frac{1}{2}y)$, $1/(1+\frac{1}{2}y)^2$, etc. are obviously discount factors. But why should the discount factors of all the cashflows follow such a simple pattern? Answer: the above formula expresses the **definition** of the bond yield.
- What is the significance of the bond yield? The yield is an equivalent way of expressing the price of a bond. Unlike the price, the yield is a percentage (like an interest rate). Traders in the bond markets prefer to deal (or conceptualize things) in terms of interest rates or yields.
- If the yield increases, the bond price decreases. Prices and yields move in opposite directions.
- The face, coupon and maturity of a bond are all parameters of the bond, when it is issued. The yield changes from day to day, as trading in the bond markets takes place. Given the yield y, the bond price B can be easily calculated by summing the above series.
- However, we are usually more interested in the opposite calculation. The fact is that the market price B is what is observed in the bond markets. We must "invert" the above formula to obtain the yield y, given the bond price B. This is a more difficult calculation.

2.1.1 Yield from bond price

- We can regard the above sum as a function of the yield $B(y)$. Suppose the bond trades with a market orice B_{market}. We wish to solve the following equation for the yield y:

$$B(y) = B_{\text{market}}\,.$$

- In mathematical terms, this is an example of solving a nonlinear equation for a (real valued) variable x

$$f(x) = 0\,.$$

  In our case, "x" is the bond yield y and $f(y) = B(y) - B_{\text{market}}$.

- Hence we temporarily set aside finance and focus on numerical methods to solve nonlinear equations. This is part of the "computational" aspect of the course.

2.2 Numerical methods to solve nonlinear equations

The material in this section may span more than one lecture in class.

2.2.1 General remarks

- This section is on the solution of nonlinear equations $f(x) = 0$.
- *Why nonlinear?*
- Consider a linear function $f(x) = ax + b$. The solution of $f(x) = 0$ is obviously $x = -b/a$.
- Hence we focus on more complicated equations, for which a simple solution is not known.
- What about an equation such as $f(x) = 1000$? Define $f_1(x) = f(x) - 1000$ and solve the equation $f_1(x) = 0$. Hence without loss of generality we may consider only $f(x) = 0$.
- The solution, if it exists, is called a **root** of the equation.
- We shall consider only a real variable x. This has some limitations. Consider the quadratic function $f(x) = ax^2 + bx + c$. The equation $f(x) = 0$ has two roots

$$x_{\text{root1}} = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \qquad x_{\text{root2}} = \frac{-b - \sqrt{b^2 - 4ac}}{2a}.$$

  However, even if a, b and c are all real, *these roots can be complex numbers.*
- By restricting our attention to real values of x, we cannot find a solution for the equation $x^2 + 1 = 0$, even though it is a quadratic equation and an exact formula for the roots is known.
- Hence we restate that our goal is to find a *real root* of the nonlinear equation $f(x) = 0$.
- There may be no real root, one real root, or multiple real roots.
- In general, it is difficult to find all the roots. Our goal will be to find one real root, if it exists.

2.3 Bisection

2.3.1 Algorithm

- The "bisection algorithm" is easy to understand and implement.
- Suppose we have two values x_{low} and x_{high} such that $f(x_{\text{low}}) < 0$ and $f(x_{\text{high}}) > 0$, or alternatively $f(x_{\text{low}}) > 0$ and $f(x_{\text{high}}) < 0$.
- The essential property is that $f(x_{\text{low}})$ and $f(x_{\text{high}})$ have *opposite signs.*
- Then, *if $f(x)$ is continuous for all values $x \in [x_{\text{low}}, x_{\text{high}}]$*, the value of f will equal zero one or more times in the interval $[x_{\text{low}}, x_{\text{high}}]$. Hence there are one or more real roots in the interval $[x_{\text{low}}, x_{\text{high}}]$.
- We say that the interval $[x_{\text{low}}, x_{\text{high}}]$ has **bracketed a root**.
- However, $f(x)$ may not be continuous throughout the interval $[x_{\text{low}}, x_{\text{high}}]$. We shall discuss this situaton below.
- The bisection algorithm proceeds recursively. We first calculate the midpoint $x_{\text{mid}} = (x_{\text{low}} + x_{\text{high}})/2$ and the value $f(x_{\text{mid}})$.
- We test if $f(x_{\text{mid}}) == 0$, or more precisely, we test if its magnitude is less than some tolerance $|f(x_{\text{mid}})| \leq$ `tol_f`. If yes, then we return the value of x_{mid} as our approximation for the root.
- Else if $f(x_{\text{mid}})$ has the same sign as $f(x_{\text{low}})$ (i.e. $f(x_{\text{mid}})f(x_{\text{low}}) > 0$), then x_{mid} is on the same side of the root as x_{low}. Update and set $x_{\text{low}} := x_{\text{mid}}$.
- Else x_{mid} is on the same side of the root as x_{high}. Update and set $x_{\text{high}} := x_{\text{mid}}$.
- Note that after the update, the length of the bracketing interval has decreased by a factor of 2. We have bisected the original interval. This is the origin of the name "bisection" algorithm.
- Test if the length of the bracketing interval is less than a tolerance $|x_{\text{high}} - x_{\text{low}}| \leq$ `tol_x`. If yes, then we have located the real root to sufficient accuracy, so we return the value of x_{mid} as our approximation for the root.
- Else repeat the above steps and calculate a new value for x_{mid} using the updated values for x_{low} and x_{high}.
- The bisection algorithm will terminate after a finite number of steps, because we shall eventually meet the tolerance criterion.

2.3.2 Bisection part 2: caveats

- The above procedure will work when the function f behaves nicely.
- Suppose $f(x)$ is not continuous. Consider the example $f(x) = 1/x$ and choose $x_{\text{low}} = -2$ and $x_{\text{high}} = 1$. Then $f(x_{\text{low}}) = -\frac{1}{2}$ and $f(x_{\text{high}}) = 1$. Hence $f(x_{\text{low}})$ and $f(x_{\text{high}})$ have opposite signs, but $f(x)$ never equals zero in the interval $[x_{\text{low}}, x_{\text{high}}]$.
- The bisection algorithm will not blow up or fail. It will converge to the location of the discontinuity.
- Consider the "step function" $f(x) = 1$ for $x \geq 0$ and $f(x) = -1$ for $x < 0$. Again choose $x_{\text{low}} = -2$ and $x_{\text{high}} = 1$. Then $f(x_{\text{low}}) = -1$ and $f(x_{\text{high}}) = 1$. Hence $f(x_{\text{low}})$ and $f(x_{\text{high}})$ have opposite signs, but $f(x)$ never equals zero in the interval $[x_{\text{low}}, x_{\text{high}}]$.
- The bisection algorithm will again converge to the location of the discontinuity.
- If the initial bracket encloses more than one root, the bisection algorithm will converge to one of those roots, but in general there is no way to know which root will be found.

2.3.3 Bisection part 3: the initial bracket

- **How do we determine the values of x_{low} and x_{high} to obtain the initial bracket?**
- This is the weak point of the algorithm.
- There is no general procedure to determine suitable values for x_{low} and x_{high}.
- Consider the example $f(x) = (x - 0.7)^2$. This has a root $x = 0.7$, but the value of $f(x)$ never crosses zero. Bisection will not work for this function. The function has a real root, but we will never obtain a bracketing interval.
- Ultimately, we need to have some understanding of the properties of the function $f(x)$ we are dealing with. The function may arise from a problem in finance, physics or engineering, etc. The background context of the problem will tell us something about the properties of $f(x)$. We use that information to determine (or guess) the values of x_{low} and x_{high}.
- This is a common feature of many root finding algorithms: ***they are not self-starting.*** Some external information is required to supply a starting point for the algorithm. The algorithm can then iterate to return an improved approximation for the root.

2.3.4 Bisection part 4: tolerances

- Notice that I employed two tolerances: `tol_x` for the size of the interval and `tol_f` for the absolute value of f.
- Once again, it helps to have some information about the function f. For example, in many applications in finance, the value of f must be computed numerically. Furthermore, the value of $f(x)$ may only be known to limited precision. It might be unrealistic to set tolerances which are too tight.
- If x is the price of a stock, it may be unrealistic to demand `tol_x` $= 10^{-6}$. Stock prices are not known to an absolute precision of 10^{-6}.
- It may also be a good idea to employ relative tolerances. This goes back to the question: what if we wish to solve the equation $f(x) = 1000$? Setting `tol_f` $= 10^{-6}$ requires calculating the value of $f(x)$ to 9 significant figures. This may be unrealistic. It may be reasonable to compute the value of f to a relative accuracy of 0.1%, i.e. 10^{-3}. Then we demand a relative tolerance
$$\frac{|f(x) - f_{\text{target}}|}{|f_{\text{target}}|} \leq 10^{-3} \, .$$
Note that if $f_{\text{target}} = 1000$, then we have only computed the value of f to an absolute accuracy of $|f(x) - f_{\text{target}}| \simeq 1$. But, depending on context, this may be good enough.
- Hence we might have four tolerances: `abs_tol_x, rel_tol_x, abs_tol_f, rel_tol_f`.

2.4 Regula Falsi

2.4.1 Algorithm

- ***In my professional experience, I have never found Regula Falsi to be useful.***
- "Regula Falsi" is Latin for "false position" and is a variation on the bisection algorithm.
- Suppose we have bracketed a root. Suppose also the function values are $f(x_{\text{low}}) = -0.1$ and $f(x_{\text{high}}) = 0.9$. Bisection will set the next iterate at the midpoint of the interval. However, we can guess that the root is closer to x_{low} than to x_{high}. Regula Falsi attempts to use the function values $f(x_{\text{low}})$ and $f(x_{\text{high}})$ to perform a weighted interpolation.
- In the above example, we divide the interval into 10 equal parts and set the next iterate to

$$x_{\text{new}} = 0.9\, x_{\text{low}} + 0.1\, x_{\text{high}}\,.$$

- More generally, the weighted average is given by

$$x_{\text{new}} = \frac{f(x_{\text{high}})\, x_{\text{low}} - f(x_{\text{low}})\, x_{\text{high}}}{f(x_{\text{high}}) - f(x_{\text{low}})}\,.$$

- Regula Falsi is essentially an attempt to shortcut the bisection procedure. However, unlike bisection, which always reduces the length of the bracketing interval by a factor of 2, Regula Falsi may produce strange interpolations. Depending on the function f, the use of a weighted average may produce a wildly wrong guess for the location of the root. Regula Falsi will converge (since we have bracketed a root), but it may converge poorly.

2.5 Fixed point iteration

2.5.1 Algorithm

- Let us digress slightly to compute the solution of the equation

$$g(x) = x\,. \tag{2.5.1.1}$$

- The iteration procedure is

$$x_{i+1} = g(x_i)\,. \tag{2.5.1.2}$$

- If the iteration converges, say to x_{fix}, then by definition $x_{\text{fix}} = g(x_{\text{fix}})$ and we have solved the desired equation. Additional iterations will simply yield the same answer.

- We say that x_{fix} is a **fixed point** of $g(x)$.

- Fixed point iteration is related to the problem of solving the equation $f(x) = 0$. Set $f(x) = g(x) - x$. Then if x_{fix} is a fixed point of $g(x)$, we see that

$$f(x_{\text{fix}}) = g(x_{\text{fix}}) - x_{\text{fix}} = x_{\text{fix}} - x_{\text{fix}} = 0\,.$$

  Hence x_{fix} is a root of f.

- We can also set $g(x) = x+f(x)$. In fact we can use any constant $c \neq 0$ and set $g(x) = x+cf(x)$. Then if x_{root} is a root of f, we see that

$$g(x_{\text{root}}) = x_{\text{root}} + cf(x_{\text{root}}) = x_{\text{root}} + 0 = x_{\text{root}}\,.$$

- Just as the equation $f(x) = 0$ can have multiple real roots or no real roots, the equation $g(x) = x$ can have multiple fixed points or no fixed points.

- Unlike bisection, fixed point iteration requires only *one* initial input value x_0. Similar to bisection, there is no simple way to determine a suitable value for the initial iterate x_0: fixed point iteration is also not self-starting.

- The fact that we do not have an initial bracketing interval has some consequences. Unlike bisection, fixed point iteration is not guaranteed to converge.

- The bisection algorithm guarantees that the length of the bracketing interval decreases by a factor of 2 at each step. For fixed point iteration, there is no simple constant such as 2, but if the fixed point iteration procedure converges, the rate of convergence is about the same as the bisection algorithm.

2.5.2 Fixed point iteration: square root

- How to compute the (positive) square root of a positive real number a? We can set $f(x) = x^2 - a$ and solve for $f(x) = 0$ using bisection. For example if $a = 2$ then a simple initial bracketing interval is $x_{\text{low}} = 1$ and $x_{\text{high}} = 2$.

- There is an alternative method, which was known in ancient times. Some call it the Babylonian method. It is an example of fixed point iteration.

- Observe that if $x < \sqrt{a}$ then $a/x > \sqrt{a}$ and vice versa. Hence take the average and set $g(x) = (x + a/x)/2$. Then $x = \sqrt{a}$ is the solution of the fixed point equation

$$x = \frac{1}{2}\left(x + \frac{a}{x}\right).$$

  Hence iterate

$$x_{i+1} = \frac{1}{2}\left(x_i + \frac{a}{x_i}\right).$$

- Consider $a = 2$ and choose $x_0 = 1$. Then we obtain the sequence of iterates

$$\begin{aligned} x_0 &= 1 \\ x_1 &= 1.5 \\ x_2 &= 1.416666666\ldots \\ x_3 &= 1.414215686\ldots \\ x_4 &= 1.414213562\ldots \end{aligned}$$

  After 5 iterates $\left|x_5 - \sqrt{2}\right|$ is good to machine precision on a 32 bit computer. We shall see later why this converges so rapidly, after we study the Newton-Raphson algorithm.

- The above is still a good algorithm today, to numerically compute square roots.

2.5.3 Fixed point iteration: square root alternative schemes

- Recall that to solve $f(x) = 0$ we can equivalently set $g(x) = x + cf(x)$ for any constant $c \neq 0$ and solve a fixed point equation $g(x) = x$.
- Let us try this, to compute the square root of a positive real number a. Set $f(x) = x^2 - a$. Let us use $a = 2$.
- Let us begin with $c = 1$. Then $g(x) = x + (x^2 - 2)$. As with the previous calculation, let us choose an initial iterate of $x_0 = 1$. Then we obtain the sequence of iterates

$$\begin{aligned} x_0 &= 1 \\ x_1 &= 0 \\ x_2 &= -2 \\ x_3 &= 0 \\ x_4 &= -2 \qquad \text{etc}\,. \end{aligned}$$

  This is no good. You can verify numerically that other values for x_0 also do not lead to convergence. If $x_0 = 2$ (and for any value $x_0 > \sqrt{2}$), the iteration is unstable.
- Let us see if $c = -1$ works better. Then $g(x) = x - (x^2 - 2)$. We again set $x_0 = 1$. Then we obtain the sequence of iterates

$$\begin{aligned} x_0 &= 1 \\ x_1 &= 2 \\ x_2 &= 0 \\ x_3 &= 2 \\ x_4 &= 0 \qquad \text{etc}\,. \end{aligned}$$

  This is also no good.
- Next let us try $c = 0.5$. Then $g(x) = x + (x^2 - 2)/2$. We again set $x_0 = 1$. Then we obtain the sequence of iterates

$$\begin{aligned} x_0 &= 1 \\ x_1 &= 0.5 \\ x_2 &= -0.375 \\ x_3 &= -1.3046875 \\ &\vdots \\ x_{12} &= -1.4142188 \end{aligned}$$

  The iteration converges, but to the *negative* square root $-\sqrt{2}$.

- Next let us try $c = -0.5$. Then $g(x) = x - (x^2 - 2)/2$. We again set $x_0 = 1$. Then we obtain the sequence of iterates

$$\begin{aligned} x_0 &= 1 \\ x_1 &= 1.5 \\ x_2 &= 1.375 \\ x_3 &= 1.4296875 \\ &\vdots \\ x_{12} &= 1.4142079 \end{aligned}$$

  The iteration converges to the positive square root $\sqrt{2}$.

- **Just because the bisection algorithm converges does not mean the fixed point iteration will also converge. And even when the fixed point iteration does converge, it may converge to a different root.**

2.6 Newton-Raphson

2.6.1 Algorithm

- Newton is Sir Isaac Newton. Joseph Raphson is an obscure person. In any case, the Newton-Raphson method is an alternative algorithm for finding real roots of nonlinear equations.
- Newton-Raphson is a very good method, when it works. Of course there are caveats.
- Suppose we have a guess x_i for the root, and by hypothesis it is close to the root x_{root}.
- Expand $f(x)$ in a Taylor series around x_i up to two terms plus a remainder. This yields (here c is a constant)

$$f(x_{\text{root}}) = f(x_i) + (x_{\text{root}} - x_i)f'(x_i) + \frac{(x_{\text{root}} - x_i)^2}{2!}\, c\,. \tag{2.6.1.1}$$

  Of course we do not know the value of x_{root}. If the value of x_i is close to x_{root}, we estimate that the magnitude of the remainder term is small and can be neglected. In addition, $f(x_{\text{root}}) = 0$ by definition. All of this yields the approximation

$$0 \simeq f(x_i) + (x_{\text{root}} - x_i)f'(x)\,. \tag{2.6.1.2}$$

  Rewrite this as

$$x_{\text{root}} \simeq x_i - \frac{f(x_i)}{f'(x_i)}\,. \tag{2.6.1.3}$$

  The Newton-Raphson algorithm therefore computes the next iterate for the root by setting

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}\,. \tag{2.6.1.4}$$

- Hence, starting from an initial guess x_0, we obtain a sequence of iterates

$$\begin{aligned} x_1 &= x_0 - \frac{f(x_0)}{f'(x_0)}\,, \\ x_2 &= x_1 - \frac{f(x_1)}{f'(x_1)}\,, \\ x_3 &= x_2 - \frac{f(x_2)}{f'(x_2)}\,, \qquad \text{etc.} \end{aligned}$$

- As with the bisection algorithm, at every step we test for tolerances, e.g. $|f(x_i)| \le$ `tol_f` and $|x_{i+1} - x_i| \le$ `tol_x`. We can also test for relative tolerances.

2.6.2 Newton-Raphson part 2: rate of convergence

- Recall the bisection algorithm divides the length of the bracketing interval by 2 at each step.
- Hence after n iterations the accuracy of the approximation for the root is symbolically

$$(\Delta x)_{\text{bisection}} = \frac{\text{const.}}{2^n} \,. \tag{2.6.2.1}$$

- For convenience replace 2 by 10, so we can illustrate using decimal arithmetic. Then if the initial iterate is accurate to one decimal place, the second iterate will be accurate to two decimal places, the third iterate will be accurate to three decimal places, the fourth iterate will be accurate to four decimal places, etc. At each step we gain one decimal place of accuracy. (In the real bisection algorithm we gain a factor of two at each step.)
- If the Newton-Raphson algorithm converges, then when the iterates are close to the root the accuracy of the approximation for the root is symbolically

$$(\Delta x)_{\text{NR}} = \frac{\text{const.}}{2^{2^n}} \,. \tag{2.6.2.2}$$

  Again replace 2 by 10, so we can illustrate using decimal arithmetic. If the initial iterate is accurate to one decimal place, the second iterate will be accurate to two decimal places, the third iterate will be accurate to *four* decimal places, the fourth iterate will be accurate to *eight* decimal places, etc. At each step, the number of decimal places of accuracy *doubles*. This is a much faster rate of convergence than bisection.
- Hence Newton-Raphson converges very quickly, when it works.
- The textbooks will give a rigorous derivation of the above statements.

2.6.3 Newton-Raphson part 3: general remarks

- Unlike bisection, Newton-Raphson requires us to compute both the function f and its first derivative f' at every iterate x_i.
- It may not be easy to compute $f'(x)$.
- When Newton-Raphson works, it usually converges to the root faster than bisection.
- However, unlike bisection, which is guaranteed to converge once an initial bracket is found, **Newton-Raphson is not guaranteed to converge.**
- Unlike bisection, Newton-Raphson requires only *one* initial value x_0. Newton-Raphson does not guarantee to bracket the root (upper and lower bounds for the root).
- Similar to bisection, Newton-Raphson is not self-starting. There is no general procedure to determine the initial guess x_0.

2.6.4 Newton-Raphson part 4: caveat examples

- Let us apply Newton-Raphson to some of the examples listed for bisection.
- Begin with $f(x) = (x - 0.7)^2$. We saw that bisection fails for this function because we cannot find an initial bracket. However Newton-Raphson works well for this function. First note that $f'(x) = 2(x - 0.7)$. Choose $x_0 = 0$. Then we obtain the sequence of iterates

$$\begin{aligned} x_0 &= 0\,, \\ x_1 &= 0.35\,, \\ x_2 &= 0.525\,, \\ x_3 &= 0.6125\,, \\ &\vdots \\ x_{10} &= 0.699316406\,. \end{aligned}$$

- **Newton-Raphson can find a root even if the function value does not cross zero.**
- Next consider $f(x) = 1/x$ and choose $x_0 = 1$. For this function $f'(x) = -1/x^2$. Hence

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} = x_i - \frac{1/x_i}{(-1/x_i^2)} = x_i - (-x_i) = 2x_i\,.$$

  Then we obtain the sequence of iterates

$$\begin{aligned} x_0 &= 1\,, \\ x_1 &= 2\,, \\ x_2 &= 4\,, \\ x_3 &= 8\,, \\ x_4 &= 16\,, \qquad \text{etc.} \end{aligned}$$

  Bisection converges to the location of the discontinuity at $x = 0$. Newton-Raphson goes roaring exponentially off to ∞.
- Next consider the step function $f(x) = 1$ for $x \geq 0$ and $f(x) = -1$ for $x < 0$. Choose $x_0 = 1$. For this function $f'(x) = 0$. Hence

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 1 - \frac{1}{0} = -\infty\,.$$

- Newton-Raphson contains a division in the calculation. In general, Newton-Raphson will have difficulty if the value of $|f'(x)|$ is small, and it will blow up if $f'(x) = 0$. Bisection never performs a division in the calculation (except by 2).
- Hence Newton-Raphson is a good algorithm, and can succeed in cases where bisection does not work, but Newton-Raphson must be used with care.

2.6.5 Newton-Raphson part 5: roots of positive numbers

- How to (efficiently) compute the (positive) square root of a positive real number a?
- Set $f(x) = x^2 - a$, but use Newton-Raphson not bisection. Now $f'(x) = 2x$. Hence

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} = x_i - \frac{x_i^2 - a}{2x_i} = \frac{1}{2}\Big(x_i + \frac{a}{x_i}\Big).$$

- This is the ancient Babylonian method. Newton-Raphson provides a rigorous derivation. Newton-Raphson also justifies that the rate of convergence of the Babylonian method (a fixed point iteration) is faster than that of bisection, for this problem.
- Using Newton-Raphson, we can devise a procedure to compute the real positive n^{th} root of any positive number a. Set $f(x) = x^n - a$. Then $f'(x) = nx^{n-1}$. Then

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} = x_i - \frac{x_i^n - a}{nx_i^{n-1}} = \frac{1}{n}\Big((n-1)x_i + \frac{a}{x_i^{n-1}}\Big).$$

2.7 Secant method

2.7.1 Algorithm

- Newton-Raphson requires the computation of the derivative $f'(x)$, which may be difficult.
- The secant method is an idea to get around this problem.
- Recall the Newton-Raphson iteration procedure

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}\,. \tag{2.7.1.1}$$

  Suppose we have two iterates x_i and x_{i-1} and the functon values $f(x_i)$ and $f(x_{i-1})$. Then we approximate the value of $f'(x_i)$ via the numerical difference

$$f'(x_i) \simeq \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}\,. \tag{2.7.1.2}$$

  Substitute this into the Newton-Raphson iteration procedure to obtain

$$x_{i+1} = x_i - \frac{f(x_i)(x_i - x_{i-1})}{f(x_i) - f(x_{i-1})}\,. \tag{2.7.1.3}$$

- This iteration procedure requires only the computation of $f(x)$ not the derivative $f'(x)$.
- Note that at each iteration step, the only new computation is to evaluate $f(x_i)$. The value of $f(x_{i-1})$ was calculated at the previous step. The iterates $x_0, x_1, x_2, \ldots$ and the function values $f(x_0), f(x_1), f(x_2), \ldots$ should be saved in arrays, for future use.
- The secant method does not resolve the problem of what happens if $f'(x_i) = 0$. The secant method also requires a division by a possibly small denominator. There can be problems if the value of $|f(x_i) - f(x_{i-1})|$ is very small, or if $|f(x_i) - f(x_{i-1})| \ll |x_i - x_{i-1}|$.
- To use the secant method, we need two initial iterates x_0 and x_1. However, unlike bisection, the values of x_0 and x_1 are not required to bracket a root.
- As always, the secant method is not self starting. There is no general procedure to determine suitable values for x_0 and x_1.