

FINANCIAL OPTIMIZATION

Lecture 4: Solving Equations and Nonlinear Optimization: Univariate Case

Philip H. Dybvig
Washington University
Saint Louis, Missouri

Univariate Optimization and Finding Zeros

- Very common problems
- Often we find an optimum by looking at zeros of the derivative
- Simple enough to program ourselves
- Complex enough to illustrate tools and problems
- Good setting for developing intuition

Univariate Optimization and Finding Zeros: Overview

In practice, finding an optimum and solving equations often look the same because an optimization is often performed by solving the first-order conditions.

Drawing pictures:

- concave optimization
- maximum/minimum/inflection point
- second-order conditions
- constraints

The best technique for nonlinear optimization or finding a zero is problem-dependent and it is hard to give general rules of thumb for which technique will work the best.

Optimization and Finding Roots: Univariate Applications I

Many of the applications of optimization and numerical methods involve functions of a single variable. Sometimes this type of application involves an isolated calculation:

- Compute the IRR for a project
- Compute a break-even price for a purchase
- Compute the smallest future guaranteed rent that would make a property attractive
- Finding the price which maximizes profit, trading off margin against sales
- Compute a bond yield
- Compute the equity position that makes deposit insurance priced correctly
- Find the optimal trade-off between interest earned and credit risk for issuing mortgages.

Optimization and Finding Roots: Univariate Applications II

In other cases, an optimization or finding a root is part of a bigger project:

- Compute implied volatilities for a bunch of options for comparison across two term structure regimes
- Calculate the yield of each of many bonds at different dates for analysis of yield curve movements
- Tabulating the optimal responses to a number of different scenarios

Grid Search: A Primitive But Useful Technique

Example: find the maximum of $f(x) = (3x^3 + x^2 + 2x)e^{-x}$ on the interval $[1, 10]$.

Technique: choose a number N (e.g. $N = 10,000$) of intervals to use and compute $f(x_n)$ for all grid points $x_n = 1 + (10 - 1)n/N$ for $n = 0, \dots, N$. Then the approximate optimum is $\hat{x} = \operatorname{argmax}\{f(x_n) | n = 0, \dots, N\}$ and approximate value is $f(\hat{x}) = \max\{f(x_n) | n = 0, \dots, N\}$.

Example: find x such that $f(x) = 1$ on the interval $[0, 10]$ where $f(x) = x/2 + \sin(x)$.

Technique: choose a number N (e.g. $N = 10,000$) of intervals to use and compute $f(x_n)$ for all grid points $x_n = 10n/N$ for $n = 0, \dots, N$. Then an approximate zero of x is (1) any x_n for which $f(x_n) = 0$ (tolerance?) or (2) $(x_n + x_{n+1})/2$ where $f(x_n)f(x_{n+1}) < 0$ (i.e. $f(x_n)$ and $f(x_{n+1})$ have different signs).

good points: if f is continuous will find all zeros and a global optimum if the grid is fine enough. drawback: slow, which may not matter unless we are doing many optimizations or the objective function is “expensive” to compute.

Grid Search: Refinements

Grid search on a subinterval. Or, for smooth functions can do better:

Optimization: quadratic approximation near an optimum can make the procedure much more accurate. If x_n is a candidate interior optimum, fit a quadratic locally:

$$\begin{aligned}f(x_{n-1}) &= a + bx_{n-1} + cx_{n-1}^2 \\f(x_n) &= a + bx_n + cx_n^2 \\f(x_{n+1}) &= a + bx_{n+1} + cx_{n+1}^2\end{aligned}$$

which is easy to solve using matrix algebra (three equations in three unknowns). The maximum of the quadratic is $x = -b/(2c)$ and the value there is $a + bx + cx^2$.

Finding a zero: linear interpolation between two points where the sign flips. Fit $f(x) = a + bx$ to those two points and then $x = -a/b$ is a better estimate of the local zero.

Spreadsheet tips: can use max, sum, logical formulas to automate the process.

note: discrete problems: exhaustive search, also plotting can be useful

Binary search

First: do a search on a big level to bracket the solution

Second: keep subdividing the interval to isolate the solution

Maximization if you can compute the derivative (often can, even for complex functions, using the chain rule): First, find two points whose derivatives have opposite signs. Second, compute the derivative halfway between the two points. If it is zero (tolerance?) we are done. Otherwise, use this point to replace the point with the derivative of the same sign and repeat this step until we have sufficient accuracy (criterion?).

Maximization if you cannot compute the derivative (maybe it does not exist or it is so complicated it takes much longer to compute than the objective function): First, bracket the solution by finding three points such that the middle point has higher value than either of the outside points. Second, test additional points between the two outside points, each time eliminating the point furthest from the optimum. (The “golden section search” algorithm is a variant.)

Once near an optimum, the search can be refined as for a grid search.

Binary Search Example

Use a binary search to find the smallest value of $3x^2 - 4x + 12$ on the interval $[0, 1]$.

A. Compute the first derivative of $f(x) = 3x^2 - 4x + 12$.

B. Compute the second derivative, verify that it is positive so the objective function is convex and therefore this minimization is a convex optimization. Therefore, we will be looking for a zero (root) of $f'(x)$.

C. Do a binary search starting with points 0 and 1. After six function evaluations, what is the estimate for the optimal x ?

D. What is the exact optimal choice of x ? (Solve the equation $f'(x) = 0$.)

Speed and robustness

In numerical optimization, there can be a trade-off between speed and robustness. For example, a binary search is a lot faster than a grid search (because it has many fewer function calls), but a binary search can converge to a local optimum that is not a global optimum. If the optimization problem is known to be convex (maximizing a concave function or minimizing a convex function), this is not a problem, but often we do not know. Further speed can be obtained if we can compute first and second derivatives of the objective function, but these algorithms may also be somewhat fragile, especially far from the solution. For this reason, many popular algorithms have the flavor of the grid search refinements described above: start with a relatively robust algorithm and then switch to something known to be very fast near the optimum.

In multidimensional problems, algorithms that do not require the computation of any derivatives usually fail on practical problems. That is especially true for algorithms that are approximating these derivatives by finite differences. Even convex optimization can give an algorithm trouble in high dimensions, especially if the objective function has a “ridge” .

In-Class Exercise: Relative Speed of Binary Search and Grid Search

Suppose we are looking for the zero of a continuous function f on $[0, 1]$ and we know from first principles that $f(0) < 0$ and $f(1) > 0$.

1. Roughly how many function calls are required to find a zero of f in $[0, 1]$ to a precision of .001 using a grid search?
2. Roughly how many function calls are required to find a zero of f in $[0, 1]$ to a precision of .001 using a binary search?
3. If there are two local solutions but only one global solution, which one will the grid search find if the grid is fine enough?
4. If there are two local solutions but only one global solution, which one will a binary search find if the grid is fine enough?

Newton's Algorithm

Suppose we are trying to solve $g(x) = 0$ and we are considering a point x_n and we know $g(x_n)$ and $g'(x_n)$. Then we can use Newton's method, which solves for $g(x) = 0$, assuming $g(x)$ is linear, expanding around x_n : $g(x) \approx g(x_n) + (x - x_n)g'(x_n) = 0$ and therefore $x_{n+1} = x_n - g(x_n)/g'(x_n)$. Applying this approximation again and again converges very quickly (number of correct digits roughly doubles each time) if (1) $g' \neq 0$ at the optimum and (2) we start close enough to the optimum. If either condition does not hold, optimization can fail.

For an optimization, $g(x) = f'(x)$ where $f(x)$ is the function to be maximized or minimized. Therefore, $x_{n+1} = x_n - f'(x)/f''(x)$.

Note: a rule for varying the step size can help solve the problems that may occur with this algorithm..

In-class Exercise: Newton's Algorithm

Consider minimizing $f(x) = x^2 + x^4$. Give the first three steps in Newton's algorithm (A) starting with $x = .5$ (B) starting with $x = 10$.

When is it Time To Quit: Tolerance

When to stop? We need a rule of thumb, but none of the popular techniques work well for all problems. Here are some ideas. Consider maximization of $f(x)$ and think of tol as a small number perhaps $1e - 06$ ($= 1 \times 10^{-6}$). Here are some typical types of convergence criteria

- Small slope: $|f'(x)| < tol$. Maybe also $f''(x) < 0$?
- Small change in x : $|x_{n+1} - x_n| < tol$.
- Small or negative change in value; $f(x_{n+1}) \leq f(x_n)$ (probably sensible only if we try smaller and smaller stepsizes to find an improvement)
- Maximum number of iterations (usually indicates not converging, but it can be used as a convergence criterion for an algorithm, e.g. binary search, that is guaranteed to give adequate precision in a fixed number of rounds). If you are supplying any derivatives, this can be a symptom that you calculated them incorrectly.

How to choose the Tolerance and Why Scaling Matters

If we pick tol too big, we may stop before we are close to the maximum, but if we pick tol too small, the algorithm may move around seemingly at random, chasing rounding error in the function evaluations. Scaling the problem by multiplying f by a constant does not change the theoretical solution, but it definitely changes the convergence criteria, which is why these algorithms are sensitive to scaling. Sometimes, it makes sense to make tol a fraction (of f or x or whatever) but not if what is in the denominator can be 0 or negative!

Bond Yield Application

Sometimes practitioners use “bond-equivalent yields” based on compounding twice a year with simple interest handling of any fractions of a half-year. However, it is more common now, especially in bond shops, to work with continuously-compounded yields. If the date now is s and the bond has future cash flows c_n (including principle and interest) at dates t_n for $n = 1, \dots, N$, the yield is the interest rate y that solves

$$P_s = \sum_{n=1}^N e^{-yt_n} c_n.$$

where the price P_s is adjusted for any accrued interest. Since the future cash flows from the bond are typically positive, the right-hand side is increasing in y so there is a unique solution. This can be calculated in solver using solve (instead of max or min). Do not click “assume linear model” and think first if you want to click “assume non-negative.”

In actual bond yield calculations, there are usual complications, e.g., how to handle call or convertible provisions. Usually, these are handled using *ad hoc* rules of thumb which are intended to be simple without doing too much violence to the underlying economics.

Numerical example of a Bond Yield

A standard 5-year bond with semi-annual coupons at a rate of 4%/year sells at \$97 per \$100 of face value (after adjustment for accrued interest). Compute its continuously compounded yield.

The bond has cash flows c_n at times $t_n = 0.5n$ for $n = 1, \dots, 9$ and a cash flow of 102 at $t_n = 5$. The input cell is the yield y and the spreadsheet has a target cell with the NPV

$$\left(\sum_{n=1}^N e^{-yt_n} c_n \right) - P_s$$

The solution given by solver is $y = 0.04625981$. If P_s were 100, the yield would be around 4%, but the yield is higher because the bond is selling at a discount.

Univariate Optimization and Search for Zeros: Summary

- Univariate Optimization is Common and Relatively Easy
- Grid search is robust plus slow
- Binary search is faster but may not find the global optimum
- Newton's method is very fast under good conditions but can be fragile
 - Very fast starting near the solution if g' (or f'') is not zero at the optimum
 - Starting with another method until near the solution can help
 - Adjusting the step size can help
- Choosing a tolerance for stopping is important and the same value will not work for all problems.