Nonlinear Programming

Nonlinear programming (NLP) is a branch of mathematical optimization that deals with problems where the objective function or constraints are nonlinear. NLP is widely used in various fields such as engineering, economics, finance, and machine learning. The motivation for studying NLP arises from the need to solve complex real-world problems that cannot be adequately addressed using linear programming techniques.

1 Numeric Foundations

This section covers the numeric foundations necessary for understanding and solving nonlinear programming problems. It includes topics such as numerical optimization techniques, gradient descent methods, and convergence criteria.

1.1 Motivation

Many economic problems involve solving nonlinear equations. For example, utility maximization and cost minimization often lead to nonlinear optimization problems. Understanding the numeric foundations of NLP is crucial for effectively addressing these challenges.

Example 1: Inverting a Nonlinear Demand Function

Consider a nonlinear demand function given by:

$$q(p) = a - bp^2$$

Find the price that corresponds to a given quantity $q = 2$. Now consider a more complex demand function which is the sum of domestic and foreign demand:

$$q(p) = 0.5p^{-0.2} + 0.5p^{-0.5}$$

Finding the price for a given quantity in this case may not have a closed-form solution, necessitating numerical methods to approximate the solution. Realistic models are often nonlinear and rarely have closed-form solutions.

1.2 Types of Problems

Many optimization and equilibrium problems in economics reduce to **solving equations**. These are the three core problem types:

1.2.1 Nonlinear Rootfinding

We want to find a solution x^* such that:

$f(x^*) = 0$

Where:

- $f : \mathbb{R}^n \to \mathbb{R}^n$
- $x^* \in \mathbb{R}^n$

This is the most general formulation. Economic examples include:

- Satisfying first-order conditions in maximization problems
- Finding market-clearing prices

1.2.2 Fixed-Point Problems

A **fixed point** of a function $g(x)$ is a solution to:

$x = g(x)$

This is equivalent to rootfinding if we define $f(x) = x - g(x)$. Many economic models are naturally written this way:

- Value function iteration in dynamic programming
- Nash equilibria in games

Fixed-point methods often rely on contraction mappings, which guarantee convergence under certain conditions.

> **i** What is a contraction mapping?
>
> Many economic problems — especially dynamic ones — are solved by **iterating a function** until it stops changing. For example:
> $x^{(k+1)} = g(x^{(k)})$
> This process works well **if** the function g pulls guesses closer together at each step.
> A **contraction mapping** is a function that does exactly that:
> it **shrinks the distance** between inputs as you iterate, so that no matter where you start, you end up at the same place.
> - Think of it like repeatedly zooming in on the true solution.
> - Each step gets you closer, and you're guaranteed to eventually land on the fixed point.
>
> #### 1.2.2.1 Example
> Suppose we are solving:
> $x = g(x) = \frac{1}{2}(x + 3)$
> Let's verify if g is a contraction:
> $|g(x) - g(y)| = \left|\frac{1}{2}(x - y)\right| = \frac{1}{2}|x - y|$
> So $\lambda = 0.5 < 1 \to g$ is a contraction.
> Then starting from any $x^{(0)}$, say $x^{(0)} = 0$, the iterates:
> $x^{(1)} = g(0) = 1.5$
> $x^{(2)} = g(1.5) = 2.25$
> $x^{(3)} = g(2.25) = 2.625$
> converge to the fixed point $x^* = 3$.

Also complementarity problems are a more general form of fixed point problem, but we won't cover those here.

1.3 Bisection Method

The **bisection method** is a simple and robust way to solve nonlinear equations of the form:

$f(x) = 0$

It is only applicable to **univariate** problems but serves as an excellent illustration of how numerical solvers work.

Requirements

To apply bisection:

- The function $f(x)$ must be **continuous** on the interval $[a, b]$
- The function must change sign over the interval: $f(a) \cdot f(b) < 0 \rightarrow$ This guarantees a root exists in $[a, b]$ by the **Intermediate Value Theorem**.
- How does it gaurantee this? either the a or b evaluates to negative and the other positive, so by the IVT there must be a point in between where it crosses zero.

1.3.1 Algorithm

1. Check that $f(a) \cdot f(b) < 0$. If not, bisection cannot proceed.
2. Compute the midpoint: $c = \frac{a+b}{2}$
3. Evaluate $f(c)$:
    - If $f(c) = 0$, stop — you found the root
    - If $f(c)$ has the same sign as $f(a)$, replace a with c
    - Otherwise, replace b with c
4. Repeat steps 2–3 until the interval is sufficiently small.

1.3.2 Example

Solve: $f(x) = x^2 - 2 = 0$

We know $x = \sqrt{2} \approx 1.4142$ is the root.

Start with $a = 1$, $b = 2$:

- $f(1) = -1$, $f(2) = 2$, so the root is bracketed.

Each iteration cuts the interval in half:

Iteration	a	b	$c = \frac{a+b}{2}$	$f(c)$
1	1.0	2.0	1.5	0.25
2	1.0	1.5	1.25	-0.4375
3	1.25	1.5	1.375	-0.109375
...	...	...	...	...

After several iterations, we get very close to the root.

1.4 Convergence Criteria

The bisection method illustrates an important question in all numerical methods:

When do we stop iterating?

There's no exact answer in most cases — so we stop when we're "close enough."
There are two common ways to define this:

1. Interval is small enough

- We stop when: $|b - a| < \epsilon$. This ensures we're zoomed in tightly on the root. In bisection, this is the **primary** stopping rule.

2. Function value is small

- Alternatively, we can stop when: $|f(c)| < \epsilon$. This ensures the function value at our current guess is nearly zero — a general-purpose convergence test used in many methods.

Why Both?

- In some methods, the updates $x^{(k+1)} - x^{(k)}$ may become small **even when the function value isn't** (e.g. at flat spots).
- We'll use these same convergence criteria in more sophisticated methods throughout the course.

1.5 Fixed-Point Iteration

Fixed-point iteration is one of the simplest ways to solve a nonlinear equation.

Instead of solving $f(x) = 0$ directly, we **rearrange the equation** so it looks like:

$x = g(x)$

We then use this as a **rule for iteration**:

$x^{(k+1)} = g(x^{(k)})$

1.5.1 Example

Suppose we want to solve: $x^2 - 2 = 0$

Rewriting this as a fixed-point problem: $x = \sqrt{2} \quad \Rightarrow \quad x = g(x) = \frac{1}{2}(x + \frac{2}{x})$

This is the update formula behind **Heron's method** for square roots.

Start with $x^{(0)} = 1$ and apply the iteration:

- $x^{(1)} = \frac{1}{2}(1 + \frac{2}{1}) = 1.5$
- $x^{(2)} = \frac{1}{2}(1.5 + \frac{2}{1.5}) \approx 1.4167$
- $x^{(3)} = \frac{1}{2}(1.4167 + \frac{2}{1.4167}) \approx 1.4142$

The sequence quickly converges to $\sqrt{2} \approx 1.4142$.

> **i** Convergence Behavior
>
> Whether fixed-point iteration works depends on the properties of $g(x)$:
> - If g is a **contraction mapping**, then it **guarantees convergence** to a unique fixed point
> - If g is not a contraction, iteration may:
>   - Converge slowly
>   - Fail to converge
>   - Diverge entirely
>
> **Stopping Criteria**
> As with bisection, we stop when the guesses stop changing:
> $|x^{(k+1)} - x^{(k)}| < \epsilon$
> Or when we're sufficiently close to a fixed point:
> $|g(x^{(k)}) - x^{(k)}| < \epsilon$

1.6 Newton's Method

Newton's method is a fast and widely used approach to solving nonlinear equations of the form:

$f(x) = 0$

It uses both the function and its **derivative** to guide the search for the root. This makes it much faster than bisection or fixed-point iteration — but also more sensitive to where you start.

1.6.1 The Iteration Rule

The Newton update is:

$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}$

This uses a **local linear approximation (first-order Taylor series expansion)**:

- At each step, approximate $f(x)$ by its tangent line
- Jump to where that line crosses zero
- Repeat until convergence

> **Iteration Rule Detail**
>
> Mathematically, the local linear approximation is:
> $L(x) = f(x^{(k)}) + f'(x^{(k)})(x - x^{(k)})$
> We find the zero of this line by solving $L(x) = 0$, which gives:
> $x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}$

This is the heart of Newton's method: each step uses the slope of $f(x)$ to make a **smart jump** toward the root, rather than guessing blindly.

Imagine standing on the curve of $f(x)$ at the point $(x^{(k)}, f(x^{(k)}))$. You draw the tangent line at that point. The next guess, $x^{(k+1)}$, is where this tangent line crosses the x-axis. This process is repeated, using the new point to draw a new tangent line, and so on, until you get sufficiently close to the root.

1.6.2 Example

Let's revisit the problem $f(x) = x^2 - 2$

We have:

- $f(x) = x^2 - 2$
- $f'(x) = 2x$

Then Newton's update is:

$x^{(k+1)} = x^{(k)} - \frac{x^{(k)2} - 2}{2x^{(k)}}$

Start with $x^{(0)} = 1$:

- $x^{(1)} = 1 - \frac{1^2 - 2}{2 \cdot 1} = 1 + 0.5 = 1.5$
- $x^{(2)} = 1.5 - \frac{1.5^2 - 2}{2 \cdot 1.5} \approx 1.4167$
- $x^{(3)} \approx 1.4142$

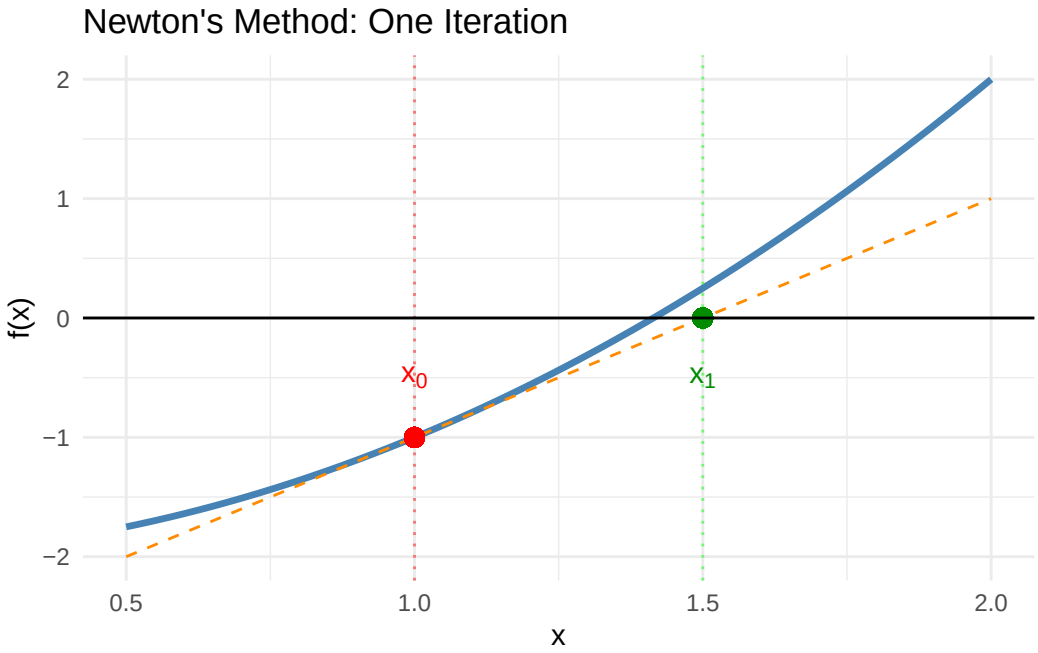
This converges very quickly to $\sqrt{2}$ — much faster than bisection.

```r
library(ggplot2)

# Define the function and its derivative
f <- function(x) x^2 - 2
f_prime <- function(x) 2 * x

# Choose initial guess
x0 <- 1.0
x1 <- x0 - f(x0) / f_prime(x0)  # first Newton update

# Build tangent line at x0
tangent <- function(x) f(x0) + f_prime(x0) * (x - x0)

# Create data for plotting
x_vals <- seq(0.5, 2, length.out = 300)
plot_df <- data.frame(x = x_vals, fx = f(x_vals), tangent = tangent(x_vals))

# Plot function and tangent
ggplot(plot_df, aes(x = x)) +
  geom_line(aes(y = fx), color = "steelblue", size = 1.2) +
  geom_line(aes(y = tangent), color = "darkorange", linetype = "dashed") +
  geom_vline(xintercept = x0, linetype = "dotted",color = "red",alpha=.5) +
  geom_vline(xintercept = x1, linetype = "dotted", color = "green",alpha=.5) +
  geom_point(aes(x = x0, y = f(x0)), color = "red", size = 3) +
  geom_point(aes(x = x1, y = 0), color = "green4", size = 3) +
  geom_hline(yintercept = 0, linetype = "solid") +
  annotate("text", x = x0, y = f(x0) + 0.5, label = "x[0]", parse = TRUE, color = "red") +
  annotate("text", x = x1, y = -0.5, label = "x[1]", parse = TRUE, color = "green4") +
  labs(
    title = "Newton's Method: One Iteration",
    y = "f(x)",
    x = "x"
  ) +
  theme_minimal()
```

```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
```

```
Warning in geom_point(aes(x = x0, y = f(x0)), color = "red", size = 3): All aesthetics have length 1, bu
i Did you mean to use `annotate()`?
```

```
Warning in geom_point(aes(x = x1, y = 0), color = "green4", size = 3): All aesthetics have length 1, but
i Did you mean to use `annotate()`?
```



1.6.3 Convergence Properties

Newton's method has **quadratic convergence** when it works:

- The number of correct digits roughly **doubles** each iteration

But it can also:

- **Diverge** if $f'(x)$ is small or zero
- Converge to the **wrong root** if the starting point is poorly chosen
- Be unstable near points of inflection

1.6.4 Stopping Criteria

Same logic as before, but more care needed due to speed and sensitivity:

- Change in x: $|x^{(k+1)} - x^{(k)}| < \epsilon$
- Residual is small: $|f(x^{(k)})| < \epsilon$

In practice, it's common to use both.

1.6.5 When to Use Newton's Method

- Works best when you have access to an **analytic derivative** $f'(x)$
- Very effective for well-behaved, smooth functions
- Poor choice if $f'(x)$ is expensive to compute or not available

1.7 Summary: Comparing Rootfinding Algorithms

Method	Speed	Derivative Needed	Convergence Guarantee	Robustness	Common Use Cases
Bisection	Slow (linear)	No	Yes (if sign change)	Very robust	Bracketing roots in univariate problems
Fixed-Point Iteration	Slow (linear)	No	Only if contraction	Medium	Dynamic programming, expectations

Method	Speed	Derivative Needed	Convergence Guarantee	Robustness	Common Use Cases
Newton's Method	Fast (quadratic)	Yes ($f'(x)$)	Local (if well-started)	Less robust	Solving FOCs, likelihood equations

Choosing the Right Method

- Use **bisection** when you need guaranteed convergence and can bracket the root.
- Use **fixed-point iteration** for problems naturally expressed as $x = g(x)$, especially in dynamic contexts.
- Use **Newton's method** for fast convergence when you have a good initial guess and can compute derivatives.
- In practice, hybrid methods that combine these approaches are often employed to balance speed and robustness.