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.

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.

1.3.3 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.

1.3.4 Multiple Roots

If the function has multiple roots in the interval, bisection will find one of them, but which one depends on the initial interval. If you need a specific root, you may need to adjust your interval accordingly. You could sample the domain to find intervals that bracket different roots.

Note also that if the function just touches zero (e.g., $f(x) = (x-1)^2$ at $x = 1$), bisection may fail since there is no sign change.

1.4 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**:

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

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.
A function $g(x)$ is a contraction on a set S if there exists a constant $\lambda \in [0, 1)$ such that:

$$|g(x) - g(y)| \leq \lambda |x - y| \quad \text{for all } x, y \in S$$

- 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.

Can we determine ex ante if a function is a contraction? Yes!
In the univariate case, a sufficient condition is:

$$|g\text{'}(x)| < 1 \quad \text{for all } x \in S$$

This guarantees that $g(x)$ is a contraction on S.

1.4.0.1 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 $\lambda = 0.5 < 1 \rightarrow g$ 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 $x^* = 3$.

1.4.1 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 $\sqrt{2} \approx 1.4142$.

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:
$|g(x^{(k)}) - x^{(k)}| < \epsilon$

1.5 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.

1.5.1 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.

1.5.2 Newtons method assumptions

1. $f(x)$ is **differentiable** in the neighborhood of the root.
2. The derivative $f'(x)$ is **not zero** at the root (to avoid division by zero).
3. The initial guess $x^{(0)}$ is **sufficiently close** to the actual root for convergence.
4. $f(x)$ is sufficiently **smooth** (e.g., continuous second derivative) near the root to ensure good approximation by the tangent line.
5. The root is **simple**. The root is simple if $f'(x^*) \neq 0$. If $f'(x^*) = 0$), the newtons method converges **linearly**.

1.5.3 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 $x^{(0)} = 1$:

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

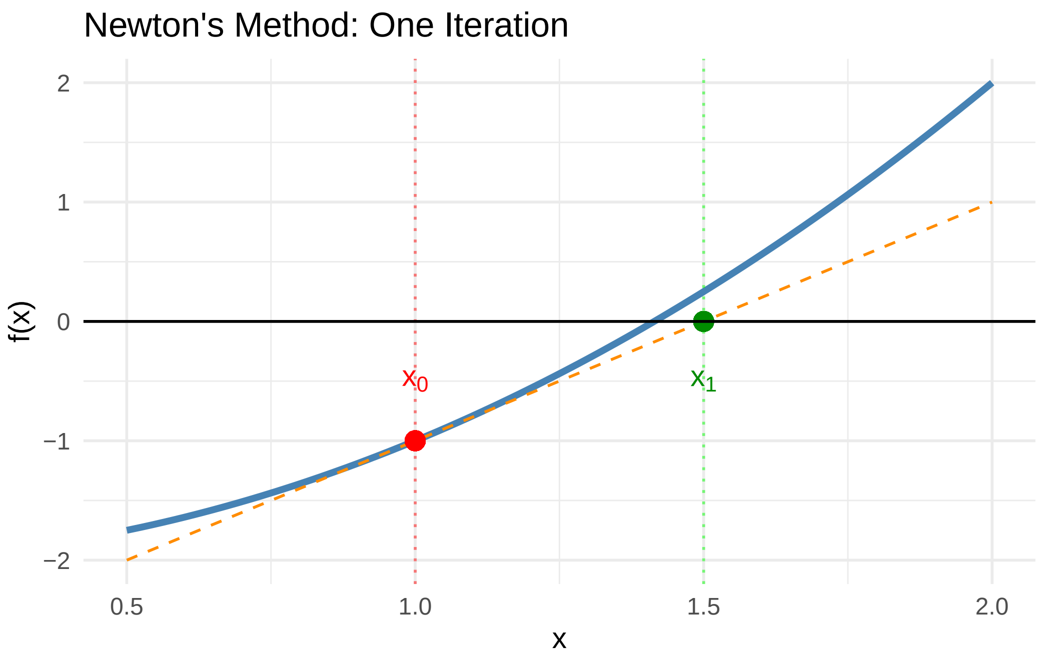
This converges very quickly to $\sqrt{2}$ — much faster than bisection.

```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()`?
```



1.5.4 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

1.5.5 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.

1.5.6 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

---

1.6 Quasi-Newton Methods

When the derivative $f'(x)$ is not available or too costly to compute, we can use **quasi-Newton methods**. These methods approximate the derivative using finite differences or other techniques.

1.6.1 Finite Difference Approximation

A simple way to approximate the derivative is using finite differences: $f'(x) \approx \frac{f(x^k)-f(x^{k-1})}{x^k-x^{k-1}}$

You can then plug this approximation into the Newton update formula: $x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}$

This approach is called the **secant method** in one dimension. You can substitute the finite difference approximation for the derivative in the Newton update.

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
Newton's Method	Fast (quadratic)	Yes ($f'(x)$)	Local (if well-started)	Less robust	Solving FOCs, likelihood equations
Quasi-Newton Methods	Fast (superlinear)	No	Local (if well-started)	More robust than Newton	Large-scale optimization, machine learning

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.

---

2 Unconstrained Optimization

2.1 Motivation

Unconstrained optimization involves finding the maximum or minimum of a function without any restrictions on the variable values. This is a common problem in economics, where we often want to maximize utility or profit, or minimize cost.

- It establishes the **mathematical foundation** for more general optimization problems (e.g., constrained optimization, dynamic programming).
- Many economic models can first be understood in a frictionless, unconstrained environment before constraints are introduced.
- It provides intuition for how **marginal conditions** (equating marginal benefits and costs) determine optimal decisions.
- It introduces key concepts like **first-order conditions (FOCs)** and **second-order conditions (SOCs)** that are essential for understanding optimality.

2.1.1 Example

- **Profit maximization by a firm**:
  A monopolist choosing output q to maximize profits:

  $$\max_{q}\ \pi(q) = p(q)q - C(q)$$

  where $p(q)$ is the inverse demand function and $C(q)$ is the cost function. If there are no capacity or policy restrictions, this is an unconstrained problem.

2.2 Mathematical Setup

The general unconstrained optimization problem can be written as:

$$\max_{x \in \mathbb{R}^n} f(x)$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is the objective function.
The solution x^* is called an **optimum** if $f(x^*) \geq f(x)$ for all x in some neighborhood of x^*.

2.2.1 Assumptions for tractability

- $f(x)$ is continuous and differentiable at least once (so gradients exist).
- For stronger results, $f(x)$ is twice differentiable (so the Hessian exists).
- The function is well-behaved: bounded above (for maximization problems) or below (for minimization problems).

  Note: In economics, many standard functional forms (Cobb-Douglas, CES, quadratic, log-linear) satisfy these assumptions, making unconstrained optimization analytically tractable.

2.3 Univariate case

If $f : \mathbb{R} \to \mathbb{R}$, the problem reduces to finding the optimal point x^* such that:

1. First-order condition (FOC):

   $$f'(x^*) = 0$$

2. Second-order condition (SOC):

- $f''(x^*) < 0 \implies$ local maximum
- $f''(x^*) > 0 \implies$ local minimum

2.3.1 Example: A quadratic profit function

$$\pi(q) = aq - bq^2, \quad a, b > 0$$

The first-order condition is:

$$\pi'(q) = a - 2bq = 0 \implies q^* = \frac{a}{2b}$$

so $\pi(q)$ has optimum at $q^* = \frac{a}{2b}$. We can verify this is a maximum by checking the second-order condition.

The second-order condition is:

$$\pi''(q) = -2b < 0 \quad \text{(since } b > 0\text{)}$$

2.3.2 Multivariate case

If $f : \mathbb{R}^n \to \mathbb{R}$, the optimization problem is:

$$\max_{x \in \mathbb{R}^n} f(x_1, x_2, \ldots, x_n)$$

1. First-order condition:

$$\nabla f(x^*) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(x^*) \\ \frac{\partial f}{\partial x_2}(x^*) \\ \vdots \\ \frac{\partial f}{\partial x_n}(x^*) \end{bmatrix} = 0$$

   $\nabla f(x^*)$ is the **gradient vector** of first derivatives. It is also called the **Jacobian** in the multivariate case.

2. Second-order condition:
   Let $H(x)$ denote the Hessian matrix of second derivatives:

$$H(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

One way to classify the Hessian is by examining the **leading principal minors** — the determinants of the top-left $k \times k$ submatrices ($\det(H) = f_{xx}f_{yy} - (f_{xy})^2$, more than two vars gets complicated):

- If $H(x^*)$ is **negative definite** $\implies$ local maximum. The sequence of leading principal minors alternates in sign, starting with negative. $det(H) > 0$ and $f_{xx} < 0$.
- If $H(x^*)$ is **negative semi-definite** $\implies$ local maximum or saddle point. All odd-order leading principal minors are non-positive, and all even-order leading principal minors are non-negative. $det(H) \geq 0$ and $f_{xx} \leq 0$.
- If $H(x^*)$ is **positive definite** $\implies$ local minimum. All leading principal minors are positive. $det(H) > 0$ and $f_{xx} > 0$.
- If $H(x^*)$ is **positive semi-definite** $\implies$ local minimum or saddle point. All leading principal minors are non-negative. $det(H) \geq 0$ and $f_{xx} \geq 0$.
- If $H(x^*)$ is **indefinite** $\implies$ saddle point. The leading principal minors do not follow a consistent sign pattern. $det(H) < 0$.

2.3.2.1 Eigenvalues

An alternative (and often more intuitive) test is to examine the **eigenvalues** of H. Each eigenvalue corresponds to curvature along a principal direction (elements of the vector x).

- If all eigenvalues > 0 strictly convex (local minimum).
- If all eigenvalues < 0 strictly concave (local maximum).
- Mixed signs   saddle point.
- Zero eigenvalues   flat directions, inconclusive.

Eigenvalue Intuition

For a square matrix $H \in \mathbb{R}^{n\times n}$, an **eigenvalue** λ and corresponding **eigenvector** $v \neq 0$ satisfy:

$$Hv = \lambda v.$$

- λ is a scalar that tells how the matrix H scales the vector v.
- v is the direction in which the scaling occurs.

The eigenvalues are solutions to the **characteristic equation**:

$$\det(H - \lambda I) = \begin{vmatrix} a-\lambda & b \\ b & c-\lambda \end{vmatrix} = 0.$$

Expanding:

$$\lambda^2 - (a+c)\lambda + (ac - b^2) = 0.$$

Thus,

$$\lambda_{1,2} = \frac{(a+c) \pm \sqrt{(a-c)^2 + 4b^2}}{2}.$$

The eigenvalue approach is more diagnostic because it tells you the curvature in every independent direction. The determinant only summarizes them into a single product, which can hide cases in higher dimensions. That's why Miranda & Fackler (and others in numerical optimization) emphasize eigenvalues for curvature checks

Economic intuition

- The determinant test is like asking: *"Does curvature bend in the same direction along both variables, or in opposite directions?"*

- The eigenvalue test is like asking: *"If I walk in any direction in input space, is the function bending upwards, downwards, or differently depending on the direction?"*

2.3.3 Multivariate Example: Two–Variable Unconstrained Maximization

Consider the problem:

$$\max_{x,y} f(x,y) = 100x + 150y - x^2 - y^2 - xy$$

Step 1: First–Order Conditions

Compute the partial derivatives:

$$\frac{\partial f}{\partial x} = 100 - 2x - y$$

$$\frac{\partial f}{\partial y} = 150 - 2y - x$$

Set both equal to zero:

$$\begin{cases} 100 - 2x - y = 0 \\ 150 - 2y - x = 0 \end{cases}$$

Solve the system:

From the first equation, $y = 100 - 2x$.
Substitute into the second:

$150 - 2(100 - 2x) - x = 0 \implies 150 - 200 + 4x - x = 0 \implies 3x - 50 = 0.$

So $x^* = \frac{50}{3} \approx 16.67$.
Then $y^* = 100 - 2(16.67) = 66.67$.

Step 2: Second–Order Conditions

Hessian matrix:

$$H = \begin{bmatrix} f_{xx} & f_{xy} \\ f_{yx} & f_{xx} \end{bmatrix} = \begin{bmatrix} -2 & -1 \\ -1 & -2 \end{bmatrix}$$

Compute its leading principal minors:

- Determinant $= |H| = (-2)(-2) - (-1)^2 = 4 - 1 = 3 > 0$
- Top-left element $= -2 < 0$

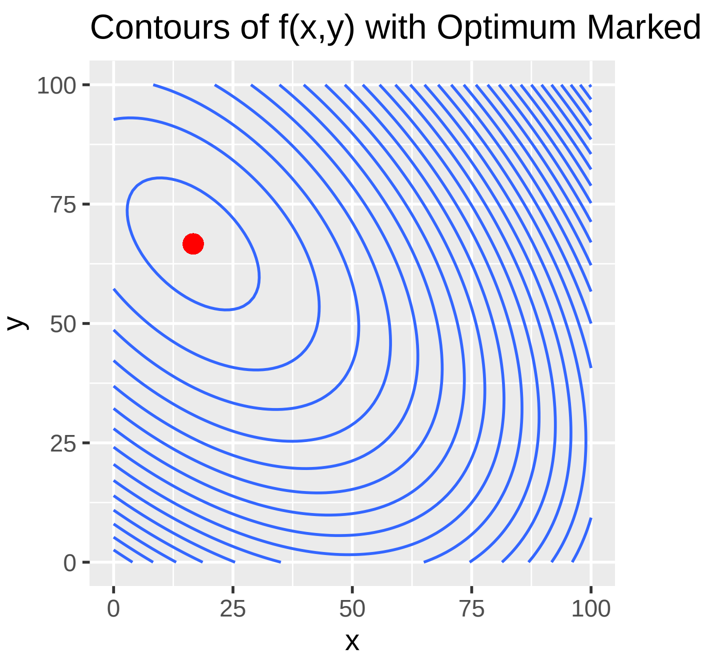
So H is **negative definite** $\rightarrow$ local maximum.

Alternatively, using eigenvalues, the characteristic polynomial is:

$$\lambda_{1,2} = \frac{-4 \pm \sqrt{(2+2)^2 - 4(-1)^2}}{2} = \frac{-4 \pm \sqrt{4}}{2} = \frac{-4 \pm 2}{2} = -1, -3$$

The eigenvalues are $\lambda_1 = -1$ and $\lambda_2 = -3$, both negative, indicating that the curvature is downward sloping in both dimensions and confirming a local maximum.

Step 4: Visualizing in R



2.4 Global vs. Local Optimality

When analyzing optimization problems, it is essential to distinguish between **local** and **global** optima.

2.4.1 Local optimum

A point x^* is a **local maximum** if there exists a neighborhood $\mathcal{N}(x^*)$ such that:

$$f(x^*) \geq f(x) \quad \forall x \in \mathcal{N}(x^*).$$

Similarly, x^* is a **local minimum** if:

$$f(x^*) \leq f(x) \quad \forall x \in \mathcal{N}(x^*).$$

Local optima are determined by **first- and second-order conditions**:

- FOC: $\nabla f(x^*) = 0$
- SOC: Hessian conditions (positive/negative definiteness)

These tests, however, only ensure optimality relative to nearby points.

2.4.2 Global optimum

A point x^* is a **global maximum** if:

$$f(x^*) \geq f(x) \quad \forall x \in \mathbb{R}^n.$$

A point x^* is a **global minimum** if:

$$f(x^*) \leq f(x) \quad \forall x \in \mathbb{R}^n.$$

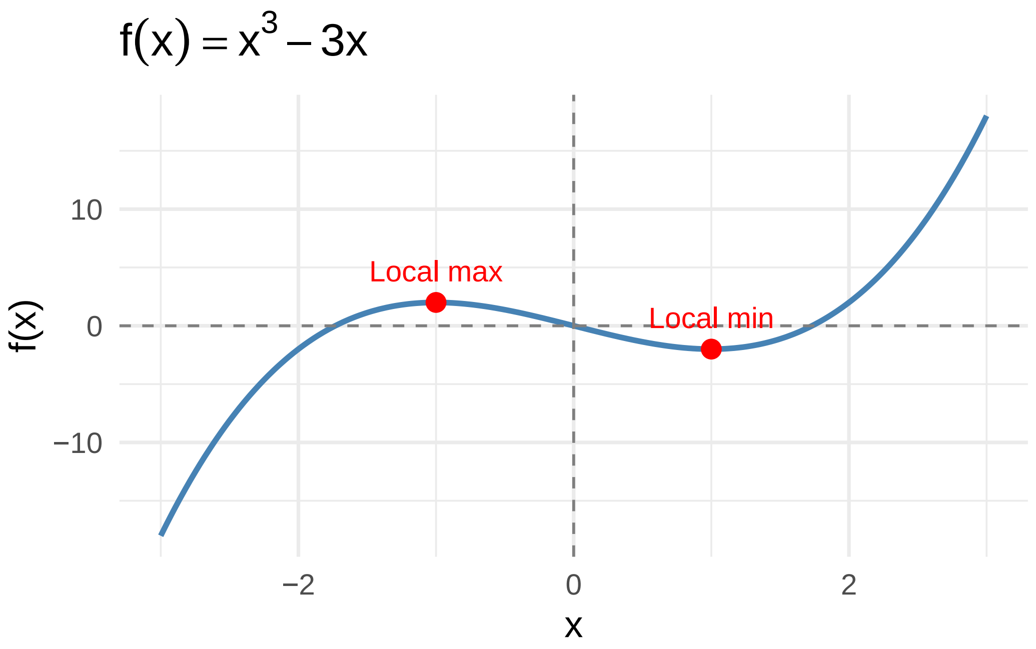
Global optimality requires stronger conditions about the **shape** of the objective function:

- If $f(x)$ is **strictly concave**, then any local maximum is also the global maximum.
- If $f(x)$ is **strictly convex**, then any local minimum is also the global minimum.

This is why assumptions of concavity and convexity are central in economics: they allow us to equate **marginal conditions** (first-order conditions) with **global results**.

2.4.3 Illustration

- A quadratic profit function
$$\pi(q) = aq - bq^2, \quad a, b > 0$$
is globally concave. The FOC $a - 2bq = 0$ yields $q^* = a/2b$, which is both a local and global maximum.
- A non-concave function such as
$$f(x) = x^3 - 3x$$
has two local extrema ($x = -1, x = 1$), but only one global maximum (as $x \to -\infty$, $f(x) \to -\infty$ and as $x \to \infty$, $f(x) \to \infty$). The FOC and SOC identify local points, but without concavity the global solution is not guaranteed.



2.4.4 Economic interpretation

- **Local optimality** corresponds to an agent being at a point where no *small deviation* is profitable.
- **Global optimality** ensures that the agent is making the *best possible choice* over the entire feasible set.

Economists typically assume concavity (utility, production, profit functions) to avoid the difficulty of distinguishing between local and global solutions. This ensures that solving the FOC is sufficient for finding the true economic optimum.

---

3 Maximum Likelihood Estimation

This section didn't work well for the class - too much for MS students and redundant for PhD students that have taken 735

New structure

- Start with simple distribution that gives a closed form solution
  - Bernoulli -> MLE is the sample mean. No need for solver because the LL has a closed form solution
  - Introduce simple score equation

3.1 Maximum Likelihood Estimation: Step 1 — Bernoulli Distribution

We begin with the simplest possible case: a **coin flip**.
Suppose we observe n independent draws $y_1, y_2, \dots, y_n$ from a Bernoulli distribution with probability of success p:

$$\Pr(y_i = 1) = p, \quad \Pr(y_i = 0) = 1 - p.$$

3.1.1 Likelihood function

The likelihood of observing the sample is

$$L(p) = \prod_{i=1}^{n} p^{y_i}(1-p)^{1-y_i}.$$

Taking logs gives the log-likelihood:

$$\ell(p) = \sum_{i=1}^{n} \left[y_i \log(p) + (1 - y_i)\log(1-p) \right].$$

3.1.2 First-order condition (Score equation)

Differentiate with respect to p:

$$\frac{\partial \ell(p)}{\partial p} = \sum_{i=1}^{n} \left[\frac{y_i}{p} - \frac{1-y_i}{1-p} \right] = 0$$

Set equal to zero:

$$\sum_{i=1}^{n} \frac{y_i}{p} = \sum_{i=1}^{n} \frac{1-y_i}{1-p}.$$

Note that $\sum_{i=1}^{n}(1-y_i) = n - \sum_{i=1}^{n} y_i$ and that you can factor out terms without an i.
Solve for p:

$$\hat{p} = \frac{1}{n}\sum_{i=1}^{n} y_i.$$

Thus, the **MLE of p is simply the sample mean** of the y_i's.

3.1.3 Second-order condition (Information)

The second derivative is

$$\frac{\partial^2 \ell(p)}{\partial p^2} = -\sum_{i=1}^{n} \left[\frac{y_i}{p^2} + \frac{1-y_i}{(1-p)^2} \right] < 0,$$

Note that $\mathbb{E}[y_i] = p$ and that $\sum_{i=1}^{n} 1 = n$. So the log-likelihood is concave and the solution is indeed a maximum.

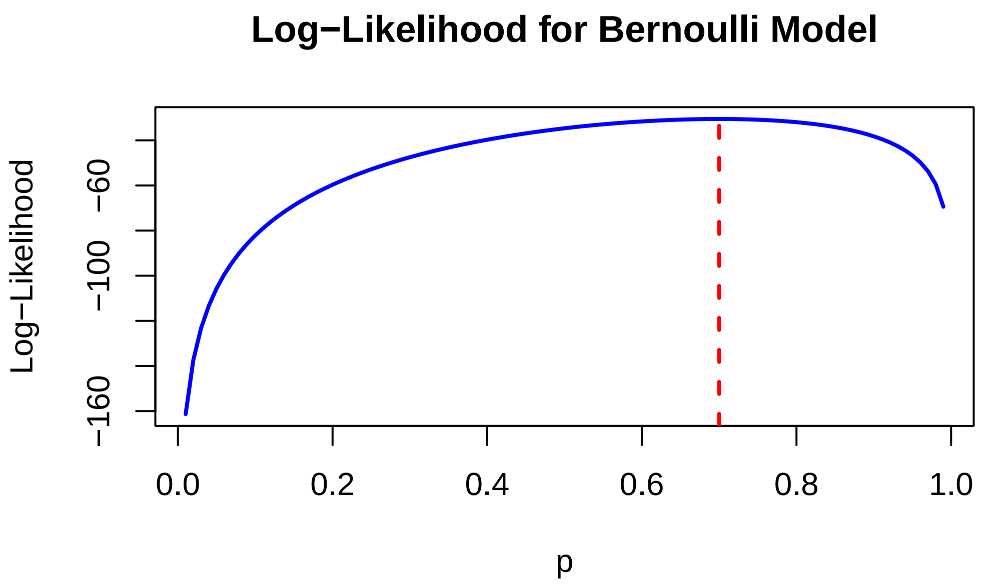
The **Fisher information** is

$$\mathcal{I}(p) = \mathbb{E}\left[-\frac{\partial^2 \ell(p)}{\partial p^2}\right] = \frac{n}{p(1-p)}.$$

Asymptotic variance:

$$\text{Var}(\hat{p}) \approx \frac{1}{\mathcal{I}(p)} = \frac{p(1-p)}{n}.$$

```r
set.seed(123)
n <- 50
p_true <- 0.7
y <- rbinom(n, 1, p_true)

# Closed-form MLE
p_hat <- mean(y)

# Log-likelihood function
ll <- function(p) sum(y*log(p) + (1-y)*log(1-p))

# Evaluate over grid
p_grid <- seq(0.01, 0.99, length.out=100)
ll_vals <- sapply(p_grid, ll)

plot(p_grid, ll_vals, type="l", lwd=2, col="blue",
     xlab="p", ylab="Log-Likelihood",
     main="Log-Likelihood for Bernoulli Model")
abline(v=p_hat, col="red", lty=2, lwd=2)
```



3.2 Maximum Likelihood Estimation: Step 2 — Logit Regression

We now generalize from the Bernoulli coin flip to a **binary regression model**.
Suppose we observe pairs (y_i, x_i), where $y_i \in \{0, 1\}$ and x_i is a covariate.

3.2.1 Model

We model the probability of success as

$$\Pr(y_i = 1|x_i) = p_i = \frac{1}{1 + \exp(-(\beta_0 + \beta_1 x_i))}.$$

Equivalently, the **log-odds** (logit) is linear in x_i:

$$\log\left(\frac{p_i}{1 - p_i}\right) = \beta_0 + \beta_1 x_i.$$

3.2.2 Likelihood and Log-Likelihood

The likelihood is

$$L(\beta) = \prod_{i=1}^{n} p_i^{y_i} (1 - p_i)^{1-y_i}.$$

Very close to the Bernoulli above but now the probability can vary by observation p_i Log-likelihood:

$$\ell(\beta) = \sum_{i=1}^{n} \Big[y_i \log(p_i) + (1 - y_i) \log(1 - p_i) \Big].$$

3.2.3 First-order condition (Score)

Differentiate with respect to β:

$$\nabla_\beta \ell(\beta) = \sum_{i=1}^{n} (y_i - p_i) x_i = 0,$$

where $x_i = (1,\ x_i)'$ includes the intercept.

- This equation **cannot be solved in closed form** for β.
- We must use **iterative optimization algorithms**.

3.2.4 Second-order condition (Hessian)

The Hessian is

$$\nabla^2_\beta \ell(\beta) = -\sum_{i=1}^{n} p_i (1 - p_i) x_i x_i'.$$

This is always **negative semi-definite**, so the log-likelihood is concave.
That means any local maximum is also the global maximum.

3.2.5 R Example

Let's simulate some data and compute the log-likelihood. See mle_example.r

- In the Bernoulli case, the MLE was the sample mean (closed form).
- In the logit regression, no closed form exists, so we need solvers.
- The likelihood surface is concave but curved: the solver iterates until it finds the peak.
- This illustrates the flexibility of MLE: as models grow more complex, the principle stays the same, but computation requires numerical optimization.

3.3 Maximum Likelihood Estimation: Step 3 — Dynamic Decision Problem

So far we considered static problems (Bernoulli, logit regression).
Now we extend to a **dynamic decision model**, where choices today affect both current payoffs and future states.

3.3.1 Example: Dynamic Investment Decision

Imagine a firm managing a piece of equipment:

- If the equipment is **kept** ($y_t = 0$):
  - Profit depends on age s_t.
  - Next period, the equipment gets older: $s_{t+1} = s_t + 1$.
- If the equipment is **replaced** ($y_t = 1$):
  - The firm pays a replacement cost.
  - The equipment is reset: $s_{t+1} = 0$.

Thus the state evolves according to:

$$s_{t+1} = \begin{cases} 0, & y_t = 1 \quad \text{(replace)} \\ s_t + 1, & y_t = 0 \quad \text{(continue)}. \end{cases}$$

3.3.1.1 Mapping to the Data

What do we as econometricians observe?

- At each time t, we see:
  - The **state** (s_t, the age of equipment).
  - The **decision** (y_t, whether to replace or not).

Over T periods, the data consist of $\{(y_t, s_t)\}_{t=1}^{T}$.

3.3.2 Choice Probabilities

We assume decisions follow a **logit rule** based on value functions:

- Value of replacing: $V_1(s_t;\beta)$.
- Value of continuing: $V_0(s_t;\beta)$.

Then the probability of replacing is:

$$\Pr(y_t = 1|s_t) = \frac{\exp(V_1(s_t;\beta))}{\exp(V_1(s_t;\beta)) + \exp(V_0(s_t;\beta))}.$$

For simplicity, suppose:

- $V_1(s_t;\beta) = \beta_0$ (constant replacement payoff).
- $V_0(s_t;\beta) = \beta_1 s_t$ (continuation value declines with age).

So:

$$\Pr(y_t = 1|s_t) = \frac{\exp(\beta_0)}{\exp(\beta_0) + \exp(\beta_1 s_t)}.$$

3.3.3 Likelihood

Given data $\{(y_t, s_t)\}_{t=1}^T$, the likelihood is

$$L(\beta) = \prod_{t=1}^{T} \Pr(y_t|s_t;\beta).$$

Log-likelihood:

$$\ell(\beta) = \sum_{t=1}^{T} \Big[y_t \log \Pr(y_t = 1|s_t;\beta) + (1 - y_t) \log \Pr(y_t = 0|s_t;\beta) \Big].$$

3.3.4 Why is this different?

- The **likelihood depends on a state variable (s_t)** that evolves over time.
- The econometrician must **explicitly track states and transitions** to evaluate $\ell(\beta)$.
- Off-the-shelf functions like `glm()` cannot be used — we must write the likelihood ourselves.

3.3.5 R Example

```r
set.seed(123)

# Simulate dynamic decision data
T <- 200
beta0 <- -1  # replacement cost
beta1 <- 0.3 # depreciation effect

s <- numeric(T) # equipment age
```

```r
y <- numeric(T) # decision

for (t in 1:(T-1)) {
  p1 <- exp(beta0)/(exp(beta0) + exp(beta1*s[t]))
  y[t] <- rbinom(1, 1, p1)
  s[t+1] <- ifelse(y[t]==1, 0, s[t]+1)
}

# Log-likelihood function
loglik_dyn <- function(beta) {
  beta0 <- beta[1];
  beta1 <- beta[2]
  p1 <- exp(beta0)/(exp(beta0) + exp(beta1*s))
  p1 <- pmin(pmax(p1, 1e-8), 1-1e-8) # avoid log(0)
  sum(y*log(p1) + (1-y)*log(1-p1))
}

# Estimate with BFGS -- maybe switch to BHHH and show the evolution. Can we derive a gradient and dis
res <- optim(par=c(0,0.1),
             fn=function(b)-loglik_dyn(b),
             method="BFGS",
             control=list(maxit=1000,trace=1,REPORT=1))
```

```
initial  value 30.154258
iter   2 value 24.837166
iter   3 value 22.765082
iter   4 value 22.747756
iter   5 value 22.746884
iter   6 value 22.746641
iter   7 value 22.746606
iter   8 value 22.746562
iter   9 value 22.746552
iter   9 value 22.746551
iter   9 value 22.746551
final  value 22.746551
converged
```

```r
res$par
```

```
[1] -0.9136284  0.1741632
```

Intuition - In static logit, probabilities depend only on covariates. - In dynamic logit, probabilities depend on both covariates and the state evolution. - The econometrician must code the state transitions to correctly evaluate the likelihood. - Yet the MLE principle is unchanged: choose β to maximize $\ell(\beta)$.

Old

Many econometric problems can be expressed as optimization problems. In particular, **Maximum Likelihood Estimation (MLE)** is a method of estimating unknown parameters by maximizing the likelihood of observing the sample data. ## Likelihood function

Suppose we observe a sample of independent realizations $\{y_1, y_2, \ldots, y_T\}$, each drawn from a distribution with density $f(y|\theta)$ parameterized by $\theta \in \Theta$. For a concrete example, lets consider the Poisson. The **likelihood function** is $L(\theta) = \prod_{t=1}^{T} f(y_t|\theta)$. Because products of densities can be numerically unstable, we usually maximize the **log-likelihood function**:

$\ell(\theta) = \log L(\theta) = \sum_{t=1}^{T} \log f(y_t|\theta)$. ## Optimization problem The MLE is defined as $\hat{\theta}_{MLE} = \arg\max_{\theta \in \Theta} \ell(\theta)$. This is an **unconstrained optimization problem** whenever the parameter space $\Theta = \mathbb{R}^k$. In practice, standard optimization methods (Newton-Raphson, quasi-Newton, gradient ascent) are employed to find $\hat{\theta}_{MLE}$. ## First- and second-order conditions

- **FOC (Score equation):** $\nabla_\theta \ell(\hat{\theta}) = 0$.
- The term dates back to R.A. Fisher (1920s). He imagined the score as a running tally or "score card" of how much the data favors one parameter value over another.
- If the score is zero, the data have no incentive to move the estimate — you've "scored even."

- **SOC (Information matrix):** For $\hat{\theta}$ to be a maximum, the Hessian of $\ell(\theta)$ must be negative definite: $H(\hat{\theta}) = \nabla_{\theta}^{2}\ell(\hat{\theta}) \prec 0$. - $\prec$ denotes negative definiteness. - The sharper the log-likelihood curve, the more information the sample contains about the true parameter. - A steep peak $\rightarrow$ small variance of $\hat{\theta}$. - A flat peak $\rightarrow$ large variance. - $\mathcal{I}(\theta)$ thus quantifies the precision of the estimator.

In large samples, the covariance matrix of $\hat{\theta}$ is approximated by the inverse of the **Fisher information matrix**:

$$\mathrm{Var}(\hat{\theta}) \approx \mathcal{I}(\hat{\theta})^{-1}, \quad \mathcal{I}(\theta) = -\mathbb{E}[\nabla_\theta^2 \ell(\theta)].$$

Example: Normal distribution

Let $y_t \sim \mathcal{N}(\mu, \sigma^2)$ i.i.d. The log-likelihood is

$$\ell(\mu, \sigma^2) = -\frac{T}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{t=1}^{T}(y_t - \mu)^2.$$

- Maximizing with respect to μ yields $\hat{\mu} = \frac{1}{T}\sum_{t=1}^{T} y_t$.
- Maximizing with respect to σ^2 yields $\hat{\sigma}^2 = \frac{1}{T}\sum_{t=1}^{T}(y_t - \hat{\mu})^2$.

Both estimators emerge naturally from solving an optimization problem. ## Economic applications MLE is widely used in economics:
- Estimation of discrete choice models (logit, probit).
- Structural estimation of dynamic models - Calibration of production or policy models where likelihood-based estimation can incorporate uncertainty.

MLE links optimization methods with statistical inference: the **same algorithms** (Newton, quasi-Newton, gradient descent) used to find optima in deterministic models also underpin parameter estimation in econometrics. ## Numerically Solving MLE

In most economic applications, the log-likelihood function $\ell(\theta)$ does not admit a closed-form solution. We therefore rely on iterative optimization algorithms to find $\hat{\theta}_{MLE}$. These algorithms search over parameter values, guided by the slope (gradient) and sometimes curvature (Hessian) of the log-likelihood surface. Gradient-based solvers:

- Newton-Raphson: Uses both the gradient and the exact Hessian. Converges quickly near the optimum but requires computing and inverting the Hessian at each step, which can be costly or unstable if the Hessian is poorly conditioned. - Quasi-Newton methods (e.g. BFGS, BHHH): Approximate the Hessian rather than computing it directly. More stable and widely used in practice.

Newton-Raphson 1. Initialization: Start from an initial guess $\theta^{(0)}$ of length m. 2. Iterative update: - Compute the $m \times 1$ gradient $g^{(k)} = \nabla_\theta \ell(\theta^{(k)})$ and $m \times m$ Hessian $H^{(k)} = \nabla_\theta^2 \ell(\theta^{(k)})$. - Update parameters: $\theta^{(k+1)} = \theta^{(k)} - [H^{(k)}]^{-1} g^{(k)}$. ### Berndt-Hall-Hall-Hausman (BHHH)

The BHHH algorithm is a quasi-Newton method that approximates the Hessian using the outer product of gradients. It is particularly useful for MLE because it leverages the structure of the likelihood function. 1. Initialization: Start from an initial guess $\theta^{(0)}$.

2. Iterative update: - Compute the gradient $g^{(k)} = \nabla_\theta \ell(\theta^{(k)})$ which is an $m \times 1$. - Compute the outer product of gradients for each observation t: $B^{(k)} = \sum_{t=1}^{T} g_t^{(k)} g_t^{(k)\prime}$, where $g_t^{(k)}$ is the gradient contribution from observation t. The dimensions of $B^{(k)}$ are $m \times m$.. - Update parameters: $\theta^{(k+1)} = \theta^{(k)} + [B^{(k)}]^{-1} g^{(k)}$. ### Broyden–Fletcher–Goldfarb–Shanno (BFGS)

The BFGS algorithm is a quasi-Newton method. It updates an approximation of the inverse Hessian at each iteration using only gradient information. 1. Initialization: - Start from an initial guess $\theta^{(0)}$. - Initialize an approximate inverse Hessian $H^{(0)}$ (often the identity matrix).

2. Iterative update: - Compute the gradient $g^{(k)} = \nabla_\theta \ell(\theta^{(k)})$. - Choose a search direction: $d^{(k)} = -H^{(k)} g^{(k)}$. - Perform a line search to select a step size $\alpha^{(k)}$ that increases $\ell(\theta)$ sufficiently. - Update parameters: $\theta^{(k+1)} = \theta^{(k)} + \alpha^{(k)} d^{(k)}$.
3. Update Hessian approximation: - Define $s^{(k)} = \theta^{(k+1)} - \theta^{(k)}$ and $y^{(k)} = g^{(k+1)} - g^{(k)}$. - Update inverse Hessian: $H^{(k+1)} = \left(I - \frac{s^{(k)} y^{(k)\prime}}{y^{(k)\prime} s^{(k)}}\right) H^{(k)} \left(I - \frac{y^{(k)} s^{(k)\prime}}{y^{(k)\prime} s^{(k)}}\right) + \frac{s^{(k)} s^{(k)\prime}}{y^{(k)\prime} s^{(k)}}$.

Intuition

- Think of BFGS as learning the curvature of the log-likelihood surface on the fly. - Newton's method jumps directly using the true curvature (the Hessian), but if that curvature estimate is noisy or expensive, the algorithm can diverge.
- BFGS instead builds a smoothed memory of past gradient steps to approximate curvature: - $s^{(k)}$ tells the algorithm how parameters moved. - $y^{(k)}$ tells the algorithm how gradients changed.

- Together, these

Why Economists Like BFGS - Robustness: Works well even when the likelihood is not globally concave. - Efficiency: Avoids direct inversion of the Hessian, which is costly in high dimensions. - Generality: Most econometric software (e.g., R's optim(), Stata's ml, MATLAB's fminunc) use BFGS as a default solver. ## MLE Example We'll estimate a simple logit model

$\Pr(y_i = 1 \mid x_i) = p_i = \frac{1}{1+\exp(-x_i'\beta)}$. Equivalently, $\text{logit}(p_i) = \log\left(\frac{p_i}{1-p_i}\right) = x_i'\beta$. Likelihood Given i.i.d. data $(y_i, x_i)_{i=1}^n$, the likelihood is $L(\beta) = \prod_{i=1}^n p_i^{y_i}(1-p_i)^{1-y_i}$. Log-likelihood: $\ell(\beta) = \sum_{i=1}^n \Big[y_i \log(p_i) + (1-y_i)\log(1-p_i)\Big]$.

Gradient (Score) $\nabla_\beta \ell(\beta) = \sum_{i=1}^n (y_i - p_i)x_i$.

Hessian $\nabla_\beta^2 \ell(\beta) = -\sum_{i=1}^n p_i(1-p_i)x_i x_i'$.

4 Constrained Optimization

4.0.1 Bordered Hessian

Consider the general equality-constrained problem:

$$\max_{x\in\mathbb{R}^n} f(x) \quad \text{s.t. } g_i(x) = 0,\ i = 1, \dots, m$$

Define the **Lagrangian**:

$$\mathcal{L}(x, \lambda) = f(x) + \sum_{i=1}^m \lambda_i g_i(x)$$

The **first order conditions (FOCs)** are:

- Stationarity: $\nabla_x \mathcal{L}(x, \lambda) = 0$
- Feasibility: $g_i(x) = 0$ for all i

A candidate (x^*, λ^*) that solves the FOCs is called a **critical point**.
We now ask: is x^* a maximum, a minimum, or a saddle?

4.1 2. The Bordered Hessian

The bordered Hessian is a matrix that extends the Hessian of the Lagrangian to account for constraints.
It is defined as:

$$H_B = \begin{bmatrix} 0 & \nabla g(x^*)^\top \\ \nabla g(x^*) & \nabla^2_{xx} \mathcal{L}(x^*, \lambda^*) \end{bmatrix}$$

- $\nabla g(x^*)$ is the $m \times n$ Jacobian of constraints.
- $\nabla^2_{xx} \mathcal{L}$ is the Hessian of the Lagrangian with respect to x.

This matrix has dimension $(m + n) \times (m + n)$.

4.2 3. Second Order Conditions

The SOCs are based on the **principal minors** (determinants of leading submatrices) of H_B.

- For a **maximum**:
  The determinants of the bordered Hessian must alternate in sign, starting with $(-1)^m$ for the first nonzero principal minor.
- For a **minimum**:
  All relevant principal minors must have sign $(-1)^{m+q}$ where q is the order of the minor (so they do *not* alternate).

Interpretation:
The bordered Hessian captures curvature of the Lagrangian restricted to the feasible set. It generalizes the unconstrained Hessian test.

4.3 4. Example: One Constraint, Two Variables

Suppose:

$$\max_{x_1, x_2} f(x_1, x_2) \quad \text{s.t. } g(x_1, x_2) = 0$$

The bordered Hessian is:

$$H_B = \begin{bmatrix} 0 & g_{x_1} & g_{x_2} \\ g_{x_1} & f_{x_1 x_1} + \lambda g_{x_1 x_1} & f_{x_1 x_2} + \lambda g_{x_1 x_2} \\ g_{x_2} & f_{x_2 x_1} + \lambda g_{x_2 x_1} & f_{x_2 x_2} + \lambda g_{x_2 x_2} \end{bmatrix}$$

At the candidate (x^*, λ^*):

- If $\det(H_B)$ has the correct sign pattern, x^* is a local maximum.

- If the pattern is reversed, x^* is a local minimum.

4.4 5. Geometric Intuition

- The FOCs flatten the gradient of f against the feasible set (the tangent space).
- The bordered Hessian checks whether curvature along feasible directions bends the objective **downward (max)** or **upward (min)**.
- Without the constraint, we would just check the Hessian of f. With constraints, the borders enforce feasibility.

4.5 6. Practical Notes

- For small n, m (2–3 variables, 1–2 constraints), computing the bordered Hessian is tractable and instructive.
- In applied work, we often rely instead on convexity assumptions (e.g., concave objective, convex feasible set $\Rightarrow$ global maximum).
- But for teaching, the bordered Hessian gives students a concrete test beyond FOCs.
