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: $f(a) \cdot f(b) < 0 \to$ 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 $x^{(0)} = 1$:

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

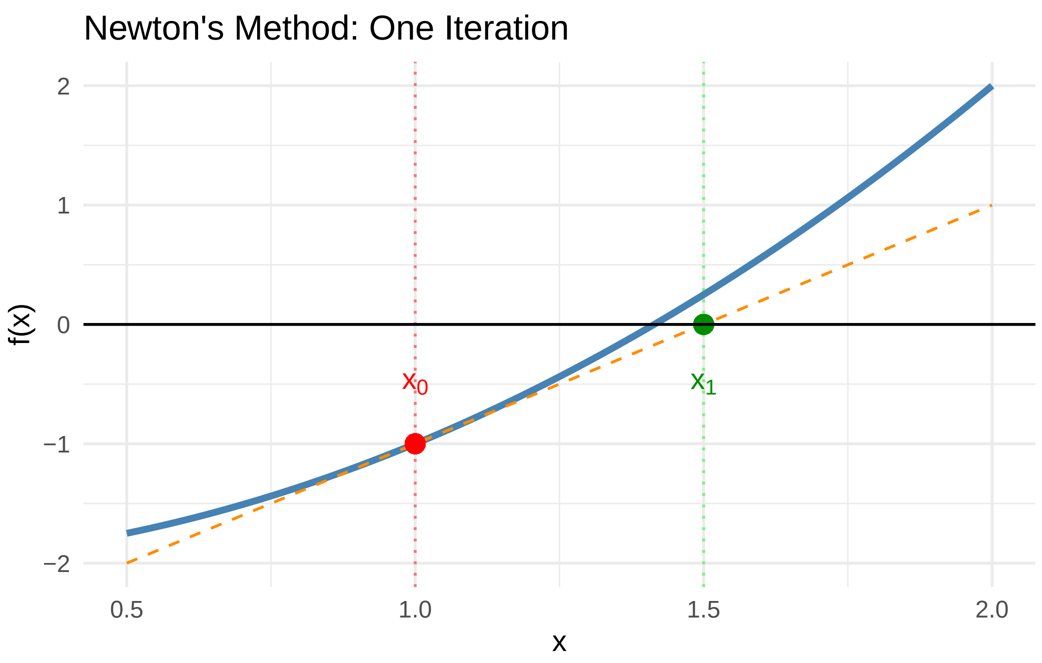
This converges very quickly to $\sqrt{2}$ — much faster than bisection.

```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.

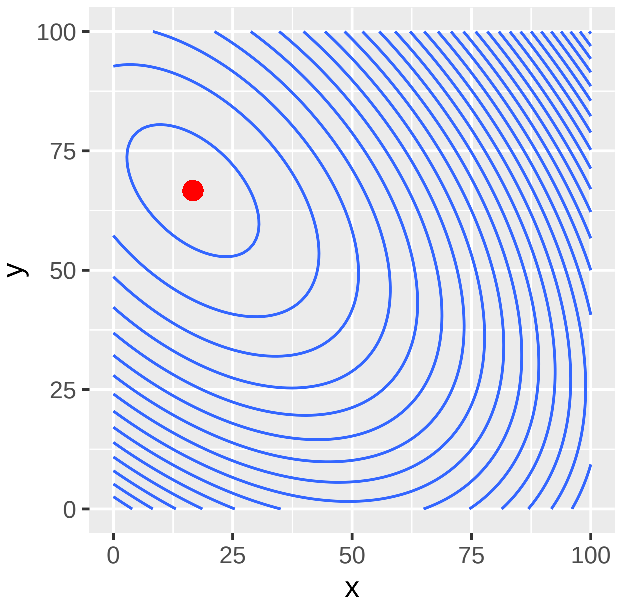
Alternatively, using eigenvalues, the characteristic polynomial is:

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

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

Contours of f(x,y) with Optimum Marked



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:

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

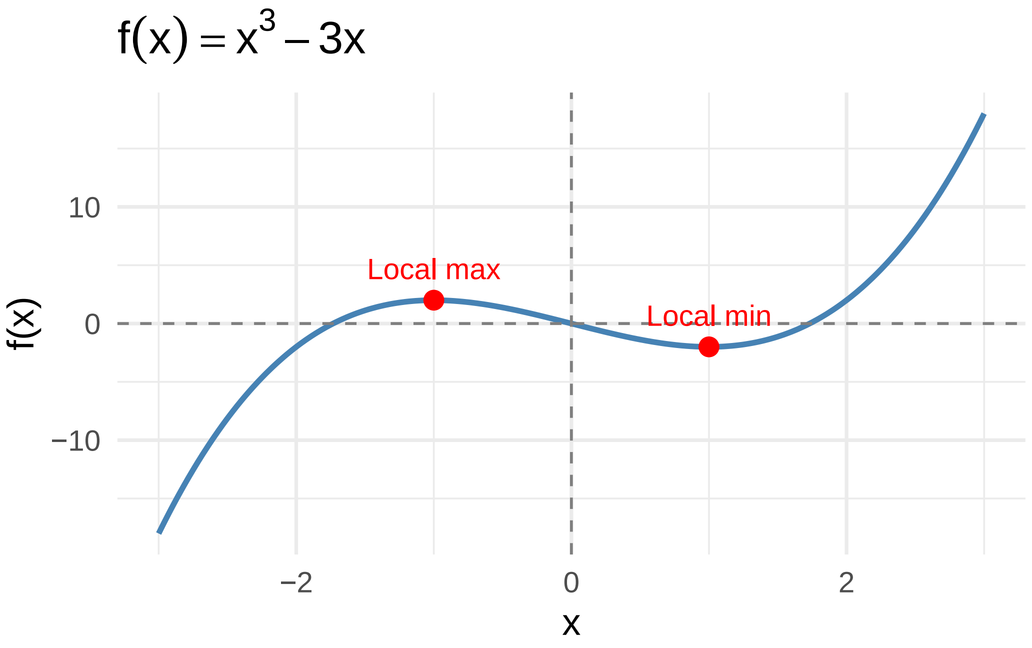
Global optimality requires stronger conditions about the **shape** of the objective function:

- 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.

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

3 Maximum Likelihood Estimation

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.

3.1 Likelihood function

Suppose we observe a sample of independent realizations $\{y_1, y_2, \dots, y_T\}$, 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**:

$$\ell(\theta) = \log L(\theta) = \sum_{t=1}^{T} \log f(y_t|\theta).$$

3.2 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 $\hat{\theta}_{MLE}$.

3.3 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:
$$H(\hat{\theta}) = \nabla^2_\theta \ell(\hat{\theta}) \prec 0.$$
  - $\prec$ 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**:

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

3.4 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.

3.5 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.

3.6 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.

3.6.1 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 $H^{(k)} = \nabla^2_\theta \ell(\theta^{(k)})$.
- Update parameters:
$$\theta^{(k+1)} = \theta^{(k)} - [H^{(k)}]^{-1} g^{(k)}.$$

3.6.2 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:

$$\theta^{(k+1)} = \theta^{(k)} + [B^{(k)}]^{-1} g^{(k)}.$$

3.6.3 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 updates gradually refine the Hessian approximation.
- As iterations proceed, the approximation converges to the true Hessian, so convergence accelerates.

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.

3.7 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,

$$\operatorname{logit}(p_i) = \log\left(\frac{p_i}{1 - p_i}\right) = x_i'\beta.$$

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

Let's start with the simple univariate case. The objective function has a single-dimension argument, x. Similarly, there is just one constraint and the function, $g(x)$ has a single dimension-argument, x.

4.1 Problem Setup

We consider the general problem:

$$\max_x f(x) \quad \text{s.t. } g(x) \leq b,$$

- $f(x)$: objective function
- $g(x) \leq b$: inequality constraint

4.2 The Lagrangian and KKT Conditions

The **Lagrangian** is

$$\mathcal{L}(x, \lambda) = f(x) + \lambda(b - g(x))$$

where λ are Lagrange multipliers (also called **shadow values**). The shadow price tells you the marginal value (in terms of the objective) of relaxing the resource constraint by a unit.

The **KKT conditions** are:

1. **Stationarity**

$$\frac{\partial \mathcal{L}}{\partial x} = \frac{\partial f(x^*)}{\partial x} - \lambda \frac{\partial g(x^*)}{\partial x} = 0$$

2. **Primal feasibility**: the solution x^* must lie in the feasible set. You cannot propose an "optimal" solution that violates the budget, land, time, or other resource limits.

$$g(x^*) \leq b$$

3. **Dual feasibility**: the Lagrange multipliers (shadow prices) must be nonnegative. Why? Because if a constraint is binding, the multiplier represents the increase in the objective from relaxing that constraint.

$$\lambda \geq 0$$

4. **Complementary slackness**: Either the constrain binds and holds with equality and the shadow price is nonnegative **or** the constraint does not bind and the shadow price is 0.

$$\lambda(b - g(x^*)) = 0$$

4.3 Bordered Hessian (Second Order Condition)

The SOCs use the **bordered Hessian** to check the **curvature of** f **along feasible directions**. With constraints, we can't just look at the full Hessian. Some directions are **forbidden** (they would leave the feasible set). The bordered Hessian modifies the Hessian so that curvature is evaluated only along directions that *respect the constraints*. Imagine standing at the tangent plane of the feasible set at x^*. The bordered Hessian checks if f bends downward (for a max) or upward (for a min) *within that tangent plane.*

For one variable and one active constraint, the bordered Hessian is a 2×2 matrix:

$$H_B = \begin{bmatrix} 0 & -g_x(x^*) \\ -g_x(x^*) & \mathcal{L}_{xx}(x^*, \lambda^*) \end{bmatrix}.$$

(The minus sign comes from writing the constraint as $b - g(x) \geq 0$ in the Lagrangian.)

The determinant is

$$\det(H_B) = -\big[g'(x^*)\big]^2.$$

- For a **constrained maximum** with one binding constraint, we require $\det(H_B) < 0$.
- For a **constrained minimum**, $\det(H_B) > 0$.

Suppose we have a vector x with n dimensions and q constraints.

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

- The **sign pattern of the principal minors** of H_B gives the curvature test:
  - For a maximum: determinants alternate in sign, starting with $(-1)^q$ starting from $q+1$.

  - For a minimum: determinants are all $(-1)^{q+r}$ for the rth minor.

In constrained optimization: You must "filter out" the directions that are not feasible. The **bordered Hessian** does that filtering — it's just a device to enforce that curvature is checked along the constraint surface.

In economics, global results often come from convexity assumptions:

```
-   Concave objective + convex feasible set → every local maximum is global.
-   This is why so much of microeconomic theory leans on convex analysis.
```

A set $C \subseteq \mathbb{R}^n$ is convex if, for any two points $x, y \in C$ and any $\theta \in [0,1]$, the convex combination

$$z = \theta x + (1-\theta) y$$

also belongs to C.

4.3.1 Example

Suppose we have a profit function $f(x_1, x_2)$ and a linear resource constraint:

$$\max_{x_1, x_2} f(x_1, x_2) = 10x_1 + 15x_2 - (x_1^2 + x_2^2) \quad \text{s.t. } x_1 + x_2 \leq 10$$

1. Find the optimum

The first-order conditions are

$$10 - 2x_1 - \lambda = 0,$$

$$15 - 2x_2 - \lambda = 0,$$

together with the binding constraint

$$x_1 + x_2 = 10.$$

From the first two equations:

$$2x_1 = 10 - \lambda, \qquad 2x_2 = 15 - \lambda.$$

Solving for λ, setting equal, and subbing into the constraint,

$$10 - 2x_1 = 15 - 2x_2 \Rightarrow x_1 = x_2 - 5/2; \quad x_2 - 5/2 + x_2 = 10 \Rightarrow x_2 = 12.5/2 = 6.25$$

Substituting back:

$$x_1 = 6.25 - 2.5 = 3.75,$$

2. Check the SOCs

$\nabla g(x) = [11]$ so the borders are 1. Then, $\nabla f(x_1, x_2)$:

$$H_B = \begin{bmatrix} 0 & 1 & 1 \\ 1 & -2 & 0 \\ 1 & 0 & -2 \end{bmatrix}.$$

Starting from $q + 1 = 1 + 1 = 2$, must alternate sign starting with $(-1)^q = 1$:

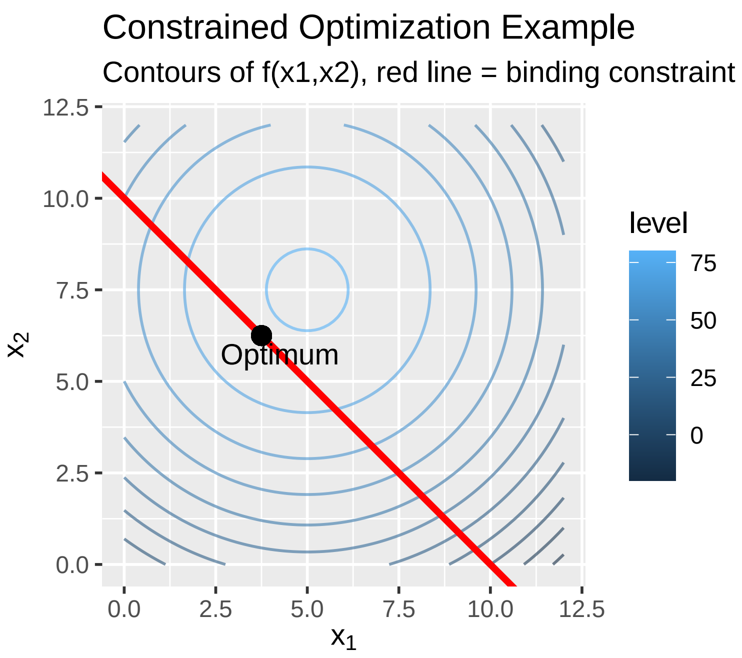
$$|H_B| = | \begin{bmatrix} 0 & 1 \\ 1 & -2 \end{bmatrix} | < 0$$

and the full H_B

```
     [,1] [,2] [,3]
[1,]    0    1    1
[2,]    1   -2    0
[3,]    1    0   -2
```

```
[1] -1
```

```
[1] 4
```



What is the shadow price of our constraint? $2(3.75) - 10 = 2.5 = \lambda$

What is the direction of steepest ascent?

- $f_{x_1}(x_1^*, x_2^*) = 10 - 2(3.75) = 2.5$
- $f_{x_2}(x_1^*, x_2^*) = 15 - 2(6.25) = 2.5$
- $\nabla g(x)$ points perpendicular to the constraint boundary (the "normal vector" to the feasible set).
- $\nabla f(x)$ points in the unconstrained steepest ascent direction.
- At the optimum on the boundary, these must line up — otherwise there would be a feasible direction in which f could still increase. One is just a rescaled version of the other, with the multiplier λ giving the scaling factor.

4.4 Envelope Theorem

- Once you set up the Lagrangian, the effect of parameter changes on the optimal value of the objective is given by multipliers.
- Example: In a consumer problem, $\frac{\partial V}{\partial I} = \lambda$ (marginal utility of income).
- Economists use this all the time in comparative statics, welfare analysis, and derivations of demand.

4.5 Duality

- The problem you solved is the primal: maximize the objective subject to constraints.
- Every primal problem has a dual problem, where the variables are the multipliers (the λ's).
- The dual is not just a mathematical trick — it has deep economic meaning: it's a price system that supports the allocation chosen in the primal.

The Lagrangian itself is the bridge:

$$\mathcal{L}(x, \lambda) = f(x) + \lambda(b - g(x)).$$

The dual function is

$$q(\lambda) = \max_x \mathcal{L}(x, \lambda).$$

The dual problem is

$$\min_{\lambda \geq 0} q(\lambda).$$

Economic interpretation: The dual chooses shadow prices λ so that the cost of resources (valued at λ) supports the primal allocation.

4.5.1 Example

The dual function is obtained by maximizing the Lagrangian over (x_1, x_2) for a given $\lambda \geq 0$:

$$q(\lambda) = \max_{x_1, x_2 \geq 0} \Big[10x_1 + 15x_2 - x_1^2 - x_2^2 + \lambda(10 - x_1 - x_2) \Big].$$

Stationarity conditions for the inner maximization:

$$\frac{\partial \mathcal{L}}{\partial x_1} = 10 - 2x_1 - \lambda = 0 \quad \Rightarrow \quad x_1 = \frac{10 - \lambda}{2},$$

$$\frac{\partial \mathcal{L}}{\partial x_2} = 15 - 2x_2 - \lambda = 0 \quad \Rightarrow \quad x_2 = \frac{15 - \lambda}{2}.$$

Plugging these $x_1(\lambda), x_2(\lambda)$ back into $\mathcal{L}$ gives the dual function. Compute each piece:
$x_1 = \frac{10-\lambda}{2}, x_2 = \frac{15-\lambda}{2}$.

$$q(\lambda) = 10\frac{10-\lambda}{2} - \left(\frac{10-\lambda}{2}\right)^2 + 15\frac{15-\lambda}{2} - \left(\frac{15-\lambda}{2}\right)^2 + \lambda(\lambda - 2.5).$$

The dual problem

$$\min_{\lambda \geq 0} q(\lambda).$$

Takeaways

- The dual problem says: find the resource price so that when agents maximize net benefit subject to that price, total demand for the resource just equals 10.
- The primal says: given 10 units of the resource, allocate across x_1 and x_2 to maximize net benefit.

Both yield the same value function, which is exactly the welfare–equilibrium duality economists care about.

---

5 Numeric Methods for Constrained Optimization

We will introduce constrained optimization numerical methods with the portfolio problem. In resource economics we often face **allocation problems under risk**. A manager (or policymaker) must decide how to allocate a fixed resource across several uses, where each use has uncertain payoffs.

Examples:

- An agricultural producer allocates land across crops with uncertain yields.
- A fishery regulator allocates harvest quotas across regions with uncertain stock growth.
- An energy portfolio manager invests in renewable projects with uncertain output.

These problems share a common structure:

- Decision variables $x = (x_1, \ldots, x_n)$ represent fractions of the resource allocated to each activity.
- Each activity i has an **expected return** μ_i and **risk** Ω (variance and covariance with other activities).

5.1 Portfolio Problem

The manager wants to maximize expected return while managing risk, subject to a budget constraint $\sum_i x_i = 1$ and possibly non-negativity constraints $x_i \geq 0$ (no short selling).

We want to choose portfolio weights $x = (x_1, \dots, x_n)$ to maximize mean–variance utility:

$$\max_x ; U(x) = \mu^\top x - \tfrac{\lambda}{2} x^\top \Omega x$$

subject to:

- Budget constraint: $\sum_{i=1}^n x_i = 1$
- Non-negativity (no short selling, or negative land/harvest): $x_i \geq 0$

where:

- μ = vector of expected returns
- Ω = covariance matrix of returns
- λ = risk aversion parameter

This utility function is called **mean–variance utility** aka Markowitz utility. It captures the tradeoff between expected return and risk (variance of returns). The parameter λ controls how much the manager dislikes risk.

5.1.1 Solving the problem

Define the Lagrangian:

$$\mathcal{L}(x, \gamma, \nu) = \mu^\top x - \tfrac{\lambda}{2} x^\top \Omega x + \gamma \left(1 - \mathbf{1}^\top x\right) + \nu^\top x$$

where:

- γ is the multiplier for the budget constraint,
- $\nu \geq 0$ are multipliers for non-negativity constraints.

Intuition

- The problem is a **risk-return tradeoff**:
  - Higher return activities often come with higher variance.
  - Diversification reduces overall risk because covariance matters.
- In resource economics, this framework generalizes:
  - A farmer diversifying across crops is solving a **portfolio of land use** problem.
  - An energy planner investing in solar, wind, and hydro is solving a **portfolio of renewable energy projects**.
  - A conservation agency allocating budgets across programs faces the same tradeoff.
- The **shadow price** of the resource constraint tells us how much expected utility increases if we could expand the available resource slightly. In this case, it tells us about our marginal utility of wealth.

5.1.2 KKT Conditions

The KKT conditions are:

- Stationarity:

$$\nabla_x \mathcal{L} = \mu - \lambda \Omega x - \gamma \mathbf{1} + \nu = 0$$

- Primal feasibility:

$$\mathbf{1}^\top x = 1, \quad x \geq 0.$$

- Dual feasibility:
$$\nu \geq 0.$$
- Complementary slackness:
$$\nu_i x_i = 0 \quad \forall i.$$

Analytical Solution Without Inequality Constraints

Ignoring $x \geq 0$ (set $\nu = 0$), stationarity gives

$$\lambda \Omega x = \mu - \gamma \mathbf{1} \quad \Rightarrow \quad x(\gamma) = \tfrac{1}{\lambda}\Omega^{-1}(\mu - \gamma \mathbf{1})$$

Interpretation:

- First, note that the optimal allocation balances excess returns with the risk (weighted by risk aversion)
- allocate more to assets with higher excess returns $\mu_i - \gamma$ and lower variance (diagonal of Ω) and covariance (off-diagonal of Ω).
- Excess return is return above the shadow price γ of wealth. In this problem, the shadow price γ varies inversely with λ.
- If you are more risk averse (higher λ), the shadow price of wealth is lower because you value safety more. In other words, you need a higher expected return to compensate for risk.

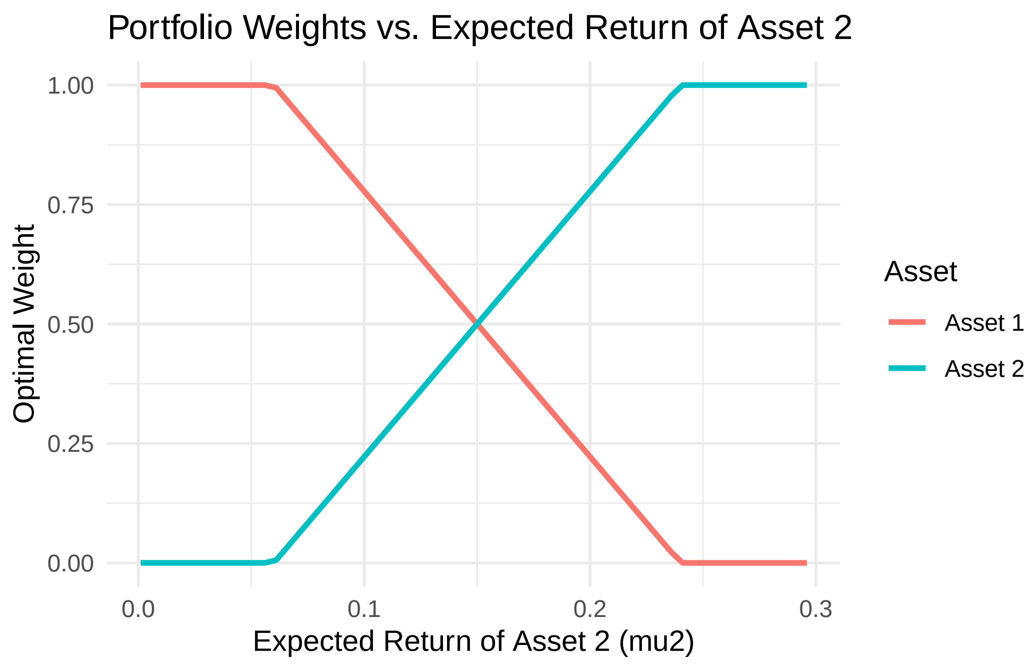
If we impose the constraint, $\mathbf{1}^\top x = 1$, we can solve for γ (shadow price of wealth):

$$\mathbf{1}^\top \Omega^{-1}(\mu - \gamma \mathbf{1}) = \lambda \;\Rightarrow\; \gamma = \frac{\mathbf{1}^\top \Omega^{-1}\mu - \lambda}{\mathbf{1}^\top \Omega^{-1}\mathbf{1}}.$$

5.2 Numerical Example in R

```r
library(ggplot2)

# Parameters
mu1 <- 0.10
Omega <- matrix(c(0.04,0.02,
                  0.02,0.09), 2, 2)
lambda <- 2

# Function: returns weights and gamma
project_nn <- function(mu, Omega, lambda){
  OmegaInv <- solve(Omega); one <- rep(1, length(mu))
  gamma <- as.numeric((t(one)%*%OmegaInv%*%mu - lambda) /
                      (t(one)%*%OmegaInv%*%one))
  x <- as.numeric((1/lambda) * OmegaInv %*% (mu - gamma*one))

  # enforce nonnegativity
  if(all(x >= -1e-10)) {
    return(list(x = pmax(x,0)/sum(pmax(x,0)), gamma = gamma))
  }
  if(x[1] < 0) return(list(x = c(0,1), gamma = gamma))
  if(x[2] < 0) return(list(x = c(1,0), gamma = gamma))
  list(x = pmax(x,0)/sum(pmax(x,0)), gamma = gamma)
}

# Sweep mu2
```

```r
mu2_vals <- seq(0.001, 0.3, by=0.005)
results <- lapply(mu2_vals, function(m2) project_nn(c(mu1,m2), Omega, lambda))

weights <- t(sapply(results, function(r) r$x))
gammas  <- sapply(results, function(r) r$gamma)

df <- data.frame(mu2 = mu2_vals,
                 x1 = weights[,1],
                 x2 = weights[,2],
                 gamma = gammas)

# For teaching: take gamma at baseline mu2 = 0.15
gamma_ref <- df$gamma[10]

# Plot weights vs mu2 with vertical line at gamma
ggplot(df, aes(x=mu2)) +
  geom_line(aes(y=x1, color="Asset 1"), size=1) +
  geom_line(aes(y=x2, color="Asset 2"), size=1) +
  # geom_vline(xintercept = gamma_ref, linetype="dashed") +
  # annotate("text", x=gamma_ref, y=1.05,
  #          label=paste0("gamma = ", round(gamma_ref,3)),
  #          hjust=-0.1, vjust=0, angle=90) +
  labs(title="Portfolio Weights vs. Expected Return of Asset 2",
       x="Expected Return of Asset 2 (mu2)", y="Optimal Weight",
       color="Asset") +
  theme_minimal()
```



5.3 Sequential Least Squares Quadratic Programming

For larger allocation problems with many assets and constraints, we often need to use numerical solvers. One popular algorithm is **Sequential Least Squares Quadratic Programming (SLSQP)**.

The SLSQP idea

1. Start from a feasible x^k.
2. At each step, approximate the problem locally with a quadratic program (QP).
3. Solve the QP $\rightarrow$ gives a search direction d^k.
4. Update $x^{k+1} = x^k + \alpha d^k$ with a line search.
5. Repeat until gradients + constraints satisfy tolerances.

More details

1. Nonlinear problem setup
   You have an objective $f(x)$ and constraints $g(x) = 0$ (budget) and $h(x) \geq 0$ (no short selling). Analytically, the KKT conditions require solving a system of equations involving gradients and multipliers.

2. Local quadratic approximation At the current guess x^k, SLSQP builds a quadratic model of the objective and linear models of the constraints:

- Objective approximated by a quadratic (second-order Taylor expansion).

$$f(x) \approx f(x^k) + \nabla f(x^k)^\top (x - x^k) * \tfrac{1}{2}(x - x^k)^\top B^k (x - x^k)$$

- where B^k is an approximation of the Hessian (e.g. BFGS update).
- Constraints approximated by linearizing their gradients.

$$g(x) \approx g(x^k) + \nabla g(x^k)^\top (x - x^k)$$

$$h(x) \approx h(x^k) + \nabla h(x^k)^\top (x - x^k)$$

- This reduces the nonlinear program to a Quadratic Program (QP) at each step.

3. Solve the QP subproblem
   The QP is much easier: quadratic objective + linear constraints.

Then find d to solve the QP subproblem:

$$\min_d ; \nabla f(x^k)^\top d + \tfrac{1}{2} d^\top B^k d$$

subject to linearized constraints:

$$g(x^k) + \nabla g(x^k)^\top d = 0, \; h(x^k) + \nabla h(x^k)^\top d \geq 0.$$

- The QP solution gives a step direction d^k.

4. Update with line search
   To ensure stability, SLSQP does a line search along d^k:

$$x^{k+1} = x^k + \alpha d^k, \quad 0 < \alpha \leq 1$$

- It chooses α to reduce the merit function (a mix of objective + penalties for constraint violation).
- Then it updates B^k (the Hessian approximation).

5. Iterate until convergence
   Repeat until the gradients are small and the constraints are satisfied (up to tolerance).

- Stationarity holds: $\nabla f(x^*) + \sum \lambda_j \nabla g_j(x^*) = 0$.
- Constraints satisfied: $g(x^*) = 0$, $h(x^*) \geq 0$.
- Complementary slackness: $\lambda_j h_j(x^*) = 0$.

5.3.1 Roles of gradient and Hessian

- Gradient term $\nabla f(x^k)$ gives the local direction of steepest descent/ascent.
- If we only used this term, the subproblem would be a linear program:

$$\min_d ; \nabla f(x^k)^\top d$$

- subject to linearized constraints. That would always push to an extreme corner of the feasible region (like simplex in LP).
- Hessian term B^k curves the model of the objective.
- It tells the solver how the slope changes as you move, i.e. how "sharp" or "flat" the function is in different directions.
- This ensures the step d is not just feasible and descending, but also "shaped" to anticipate the curvature of f.

In optimization language:

- Gradient = first-order information (direction of decrease).
- Hessian = second-order information (how big of a step you can take in each direction without overshooting).