B Linear Algebra Essentials

R. Heller · Date: 2026-05-08 · Last changed: 2026-11-03

Vectors, matrices, eigenvalues, and the linear algebra used throughout this book.

This appendix reviews the linear algebra that appears most often in game theory. The focus is on concepts rather than proofs, with R implementations alongside each topic. For a thorough mathematical treatment, see Strang (2023).

B.1 Vectors and vector operations

A vector in $\mathbb{R}^n$ is an ordered list of n real numbers. In game theory, vectors represent strategy profiles, probability distributions over actions, and payoff allocations.

```
# Define vectors
x ← c(3, 1, 4)
y ← c(2, 7, 1)

# Scalar multiplication and addition
2 * x
```

```
#> [1] 6 2 8
```

```
x + y
```

```
#> [1] 5 8 5
```

```
# Dot (inner) product
sum(x * y)
```

```
#> [1] 17
```

```
# Or equivalently:
x %*% y
```

```
#>      [,1]
#> [1,]   17
```

```
# Euclidean norm (length)
sqrt(sum(x^2))
```

```
#> [1] 5.1
```

The **dot product** $\mathbf{x} \cdot \mathbf{y} = \sum_i x_i y_i$ appears whenever we compute expected payoffs: if $\mathbf{p}$ is a probability vector and $\mathbf{u}$ is a payoff vector, then $\mathbf{p} \cdot \mathbf{u}$ is the expected payoff.

B.2 Matrices

A matrix $A \in \mathbb{R}^{m \times n}$ is a rectangular array of numbers. In game theory, the primary use is the payoff matrix of a normal-form game.

B.2.1 Matrix arithmetic in R

```
A ← matrix(c(3, 0, 5, 1), nrow = 2, byrow = TRUE)
B ← matrix(c(3, 5, 0, 1), nrow = 2, byrow = TRUE)

# Element-wise operations
A + B
```

```
#>      [,1] [,2]
#> [1,]    6    5
#> [2,]    5    2
```

```
A * B  # Hadamard (element-wise) product
```

```
#>      [,1] [,2]
#> [1,]    9    0
#> [2,]    0    1
```

```
# Matrix multiplication
A %*% B
```

```
#>      [,1] [,2]
#> [1,]    9   15
#> [2,]   15   26
```

```
# Transpose
t(A)
```

```
#>      [,1] [,2]
#> [1,]    3    5
#> [2,]    0    1
```

B.2.2 Key matrix operations

```
M ← matrix(c(4, 2, 1, 3), nrow = 2, byrow = TRUE)

# Determinant
det(M)
```

```
#> [1] 10
```

```
# Inverse (only if det ≠ 0)
solve(M)
```

```
#>      [,1] [,2]
#> [1,]  0.3 -0.2
#> [2,] -0.1  0.4
```

```
# Verify: M %*% M^{-1} = I
M %*% solve(M)
```

```
#>                          [,1] [,2]
#> [1,]  1.0000000000000000000    0
#> [2,] -0.0000000000000000278    1
```

B.3 Systems of linear equations

Many game-theoretic computations reduce to solving a system $A\mathbf{x} = \mathbf{b}$. For example, finding a mixed Nash equilibrium in a 2×2 game requires solving the indifference equations (see 5).

B.3.1 Solving $A\mathbf{x} = \mathbf{b}$

```
# System: 2x + y = 5, x + 3y = 7
A ← matrix(c(2, 1, 1, 3), nrow = 2, byrow = TRUE)
b ← c(5, 7)

x ← solve(A, b)
x
```

```
#> [1] 1.6 1.8
```

```
# Verify
A %*% x
```

```
#>      [,1]
#> [1,]    5
#> [2,]    7
```

B.3.2 Application: mixed Nash equilibrium

In a 2×2 game, finding the mixed Nash equilibrium reduces to making each player indifferent between their pure strategies. If Player 2 mixes with probability q on the first action, Player 1's indifference condition is:

$$a_{11}q + a_{12}(1-q) = a_{21}q + a_{22}(1-q)$$

This is a single linear equation in q. For larger games, mixed equilibria require solving systems of such equations simultaneously.

```
# Prisoner's Dilemma payoffs for Player 1
# Row player payoffs: A[i,j]
A ← matrix(c(3, 0, 5, 1), nrow = 2, byrow = TRUE)

# Indifference: A[1,1]*q + A[1,2]*(1-q) = A[2,1]*q + A[2,2]*(1-q)
# Rearranging: q*(A[1,1] - A[1,2] - A[2,1] + A[2,2]) = A[2,2] - A[1,2]
denom ← A[1, 1] - A[1, 2] - A[2, 1] + A[2, 2]
q_star ← (A[2, 2] - A[1, 2]) / denom
q_star
```

```
#> [1] -1
```

B.4 Eigenvalues and eigenvectors

An eigenvector $\mathbf{v}$ of a square matrix A satisfies $A\mathbf{v} = \lambda\mathbf{v}$ for some scalar λ (the eigenvalue). Eigenvalues appear in several places in this book:

- **Replicator dynamics** (20): the stability of fixed points is determined by the eigenvalues of the Jacobian matrix.
- **Markov chains**: the stationary distribution is the eigenvector corresponding to eigenvalue 1.

```
M ← matrix(c(4, 1, 2, 3), nrow = 2, byrow = TRUE)

eig ← eigen(M)
eig$values
```

```
#> [1] 5 2
```

```
eig$vectors
```

```
#>       [,1]   [,2]
#> [1,] 0.707 -0.447
#> [2,] 0.707  0.894
```

```
# Verify: M v = lambda v
lambda_1 ← eig$values[1]
v_1 ← eig$vectors[, 1]
M %*% v_1
```

```
#>      [,1]
#> [1,] 3.54
#> [2,] 3.54
```

```
lambda_1 * v_1
```

```
#> [1] 3.54 3.54
```

B.4.1 Stability and the Jacobian

For a dynamical system $\dot{\mathbf{x}} = f(\mathbf{x})$, a fixed point $\mathbf{x}^*$ is **stable** if all eigenvalues of the Jacobian matrix $J = \partial f / \partial \mathbf{x}$ evaluated at $\mathbf{x}^*$ have negative real parts.

```
# Example: Jacobian at a fixed point of a 2D system
J ← matrix(c(-2, 1, 0, -3), nrow = 2, byrow = TRUE)

eig_J ← eigen(J)
eig_J$values
```

```
#> [1] -3 -2
```

```
# Both eigenvalues are negative ⇒ stable fixed point
all(Re(eig_J$values) < 0)
```

```
#> [1] TRUE
```

B.5 Positive definite matrices

A symmetric matrix A is **positive definite** if $\mathbf{x}^T A \mathbf{x} > 0$ for all nonzero $\mathbf{x}$. Equivalently, all eigenvalues are positive. Positive definiteness arises in quadratic payoff functions and in verifying second-order conditions for optimisation problems.

```r
A ← matrix(c(4, 1, 1, 3), nrow = 2, byrow = TRUE)

# Check via eigenvalues
eigen(A)$values
```

```
#> [1] 4.62 2.38
```

```r
all(eigen(A)$values > 0)
```

```
#> [1] TRUE
```

```r
# Check via Cholesky decomposition (succeeds only for positive definite matrices)
chol(A)
```

```
#>      [,1] [,2]
#> [1,]    2 0.50
#> [2,]    0 1.66
```

B.6 Convex sets and the simplex

B.6.1 The probability simplex

A **mixed strategy** is a probability distribution over pure strategies. The set of all such distributions forms the **(probability) simplex**:

$$\Delta_n = \left\{ \mathbf{p} \in \mathbb{R}^n \;\middle|\; p_i \geq 0,\ \sum_{i=1}^{n} p_i = 1 \right\}$$

For $n = 2$, the simplex is a line segment; for $n = 3$, it is a triangle.

```r
# Check whether a vector is a valid mixed strategy
is_mixed_strategy ← function(p, tol = 1e-10) {
  all(p ⩾ -tol) && abs(sum(p) - 1) < tol
}

is_mixed_strategy(c(0.3, 0.5, 0.2))
```

```
#> [1] TRUE
```

```r
is_mixed_strategy(c(0.5, 0.6, -0.1))
```

```
#> [1] FALSE
```

B.6.2 Convex sets and convex combinations

A set C is **convex** if for any two points $\mathbf{x}, \mathbf{y} \in C$ and any $\lambda \in [0, 1]$, the point $\lambda \mathbf{x} + (1 - \lambda)\mathbf{y}$ is also in C. Important convex sets in game theory include:

- The simplex Δ_n (the set of mixed strategies).
- The **core** of a cooperative game (see 9).
- The **feasible payoff region** of a repeated game (see 8).

```
# Convex combination of two strategy vectors
p1 <- c(0.7, 0.2, 0.1)
p2 <- c(0.1, 0.5, 0.4)
lambda <- 0.6

p_mix <- lambda * p1 + (1 - lambda) * p2
p_mix
```

```
#> [1] 0.46 0.32 0.22
```

```
is_mixed_strategy(p_mix)
```

```
#> [1] TRUE
```

B.6.3 The support of a mixed strategy

The **support** of a mixed strategy $\mathbf{p}$ is the set of pure strategies that receive positive probability: $\text{supp}(\mathbf{p}) = \{i : p_i > 0\}$. In a Nash equilibrium, every pure strategy in the support must yield the same expected payoff (the indifference principle from 5).

```
support <- function(p, tol = 1e-10) {
  which(p > tol)
}

support(c(0.5, 0, 0.5, 0))
```

```
#> [1] 1 3
```

```
support(c(0.25, 0.25, 0.25, 0.25))
```

```
#> [1] 1 2 3 4
```

B.7 Matrix decompositions

B.7.1 LU decomposition

R does not expose LU decomposition directly in base, but `solve()` uses it internally. For most game-theoretic applications, `solve()` suffices.

B.7.2 Singular value decomposition (SVD)

The SVD decomposes any $m \times n$ matrix as $A = U\Sigma V^T$. It is useful for low-rank approximations and for diagnosing near-singular payoff matrices.

```
A ← matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, byrow = TRUE)

s ← svd(A)
s$d  # singular values
```

```
#> [1] 9.508 0.773
```

```
# Reconstruct: U %*% diag(d) %*% t(V) = A
s$u %*% diag(s$d) %*% t(s$v)
```

```
#>      [,1] [,2] [,3]
#> [1,]    1    2    3
#> [2,]    4    5    6
```

B.8 Summary of R linear algebra functions

Task	R function	Example
Matrix multiply	`%*%`	`A %*% B`
Transpose	`t()`	`t(A)`
Determinant	`det()`	`det(A)`
Inverse	`solve()`	`solve(A)`
Solve $Ax = b$	`solve()`	`solve(A, b)`
Eigenvalues	`eigen()`	`eigen(A)$values`
SVD	`svd()`	`svd(A)$d`
Cholesky	`chol()`	`chol(A)`
Cross product	`crossprod()`	`crossprod(A)` gives $A^T A$
Outer product	`outer()` or `%o%`	`x %o% y`

R functions for linear algebra