C Probability Essentials

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

Probability distributions, expectations, Bayes' rule, and stochastic processes.

This appendix reviews the probability concepts that underpin mixed strategies, Bayesian games, and the simulation chapters. The emphasis is on intuition and R implementation rather than measure-theoretic rigour.

C.1 Sample spaces and events

A **sample space** Ω is the set of all possible outcomes of a random experiment. An **event** $E \subseteq \Omega$ is a subset of outcomes. A **probability measure** P assigns a number $P(E) \in [0, 1]$ to every event, with $P(\Omega) = 1$.

In game theory, the sample space often corresponds to the set of action profiles or the set of player types.

C.2 Discrete distributions

A **discrete random variable** X takes values in a countable set $\{x_1, x_2, \ldots\}$ with probability mass function $p(x_i) = P(X = x_i)$.

C.2.1 Common discrete distributions

```
# Bernoulli: coin flip (used in mixed strategies)
rbinom(10, size = 1, prob = 0.5)
```

```
#>  [1] 1 1 0 1 1 1 1 0 1 1
```

```
# Binomial: number of cooperators in n rounds
dbinom(3, size = 10, prob = 0.6)  # P(X = 3)
```

```
#> [1] 0.0425
```

```
# Uniform: equally likely strategies
sample(c("Rock", "Paper", "Scissors"), size = 1)
```

```
#> [1] "Scissors"
```

```r
# Poisson: arrival counts (used in some mechanism design models)
rpois(10, lambda = 3)
```

```
#>  [1] 4 6 2 3 6 7 1 3 3 5
```

C.2.2 Probability mass function and CDF

```r
# PMF and CDF of Binomial(10, 0.4)
x ← 0:10
pmf ← dbinom(x, size = 10, prob = 0.4)
cdf ← pbinom(x, size = 10, prob = 0.4)

tibble(x = x, pmf = pmf, cdf = cdf) ▷
  head(6)
```

```
#> # A tibble: 6 × 3
#>       x     pmf     cdf
#>   <int>   <dbl>   <dbl>
#> 1     0 0.00605 0.00605
#> 2     1 0.0403  0.0464
#> 3     2 0.121   0.167
#> 4     3 0.215   0.382
#> 5     4 0.251   0.633
#> 6     5 0.201   0.834
```

C.3 Continuous distributions

A **continuous random variable** X has a probability density function $f(x)$ such that $P(a \leq X \leq b) = \int_a^b f(x)\,dx$.

C.3.1 Common continuous distributions

```r
# Uniform on [0, 1]: used for types in Bayesian games
runif(5, min = 0, max = 1)
```

```
#> [1] 0.1387 0.9889 0.9467 0.0824 0.5142
```

```r
# Normal: payoff noise, error terms
rnorm(5, mean = 0, sd = 1)
```

```
#> [1] -0.279 -0.133  0.636 -0.284 -2.656
```

```r
# Exponential: waiting times, discount factors
rexp(5, rate = 1)
```

```
#> [1] 4.996 0.224 1.211 0.719 1.308
```

C.3.2 Density, CDF, and quantiles

R uses a consistent naming convention: `d` for density, `p` for CDF, `q` for quantile, and `r` for random draws.

```
# Normal distribution
dnorm(0, mean = 0, sd = 1)   # density at 0
```

```
#> [1] 0.399
```

```
pnorm(1.96, mean = 0, sd = 1) # P(X ≤ 1.96)
```

```
#> [1] 0.975
```

```
qnorm(0.975, mean = 0, sd = 1) # 97.5th percentile
```

```
#> [1] 1.96
```

```
rnorm(3, mean = 0, sd = 1)    # three random draws
```

```
#> [1]  0.358  0.302 -0.394
```

C.4 Expected value and variance

The **expected value** of a discrete random variable is $E[X] = \sum_i x_i \, p(x_i)$. For a continuous variable, $E[X] = \int x \, f(x) \, dx$. In game theory, the expected payoff under a mixed strategy $\mathbf{p}$ is:

$$E[u_i] = \sum_{a \in A} p(a) \, u_i(a)$$

The **variance** $\mathrm{Var}(X) = E[(X - E[X])^2] = E[X^2] - (E[X])^2$ measures the spread of outcomes.

```r
# Expected payoff under a mixed strategy
payoffs ← c(3, 0, 5, 1)
probs   ← c(0.3, 0.2, 0.1, 0.4)

# Expected value
ev ← sum(payoffs * probs)
ev
```

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

```r
# Variance
variance ← sum(probs * (payoffs - ev)^2)
variance
```

```
#> [1] 2.36
```

```r
# Using simulation to verify
simulated ← sample(payoffs, size = 100000, replace = TRUE, prob = probs)
mean(simulated)
```

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

```r
var(simulated)
```

```
#> [1] 2.37
```

C.4.1 Linearity of expectation

A key property: $E[aX + bY] = aE[X] + bE[Y]$, regardless of whether X and Y are independent. This is used constantly when computing expected payoffs in multi-player games.

C.5 Conditional probability

The **conditional probability** of A given B is:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}, \quad P(B) > 0$$

Two events are **independent** if $P(A \cap B) = P(A)P(B)$, equivalently $P(A \mid B) = P(A)$.

```r
# Simulation: P(both cooperate | Player 1 cooperates)
n ← 100000
p1 ← sample(c("C", "D"), n, replace = TRUE, prob = c(0.6, 0.4))
p2 ← sample(c("C", "D"), n, replace = TRUE, prob = c(0.5, 0.5))

# Joint and conditional (assuming independence here)
p_both_C ← mean(p1 == "C" & p2 == "C")
p_p1_C   ← mean(p1 == "C")
p_both_C / p_p1_C  # should be approx 0.5
```

```
#> [1] 0.501
```

C.6 Bayes' theorem

Bayes' theorem is the foundation of Bayesian games (7). It tells us how to update beliefs about a player's type after observing their action:

$$P(\theta \mid a) = \frac{P(a \mid \theta)\, P(\theta)}{P(a)}$$

where θ is the player's type, a is the observed action, $P(\theta)$ is the prior, $P(a \mid \theta)$ is the likelihood, and $P(a) = \sum_{\theta'} P(a \mid \theta')P(\theta')$ is the marginal likelihood.

```r
# Signaling game: worker chooses Education or No Education
# Types: High (prob 0.6), Low (prob 0.4)
# P(Education | High) = 0.9, P(Education | Low) = 0.2

prior_high ← 0.6
prior_low  ← 0.4

p_edu_given_high ← 0.9
p_edu_given_low  ← 0.2

# P(Education)
p_edu ← p_edu_given_high * prior_high + p_edu_given_low * prior_low
p_edu
```

```
#> [1] 0.62
```

```r
# P(High | Education) by Bayes' theorem
p_high_given_edu ← (p_edu_given_high * prior_high) / p_edu
p_high_given_edu
```

```
#> [1] 0.871
```

```r
# P(High | No Education)
p_no_edu ← 1 - p_edu
p_high_given_no_edu ← ((1 - p_edu_given_high) * prior_high) / p_no_edu
p_high_given_no_edu
```

```
#> [1] 0.158
```

C.6.1 Sequential updating

When multiple signals are observed, Bayes' theorem can be applied sequentially: the posterior from one update becomes the prior for the next. This is used in repeated Bayesian games where players learn about opponents over time.

```
# Sequential Bayesian updating: observe actions round by round
# Prior: P(Cooperator type) = 0.5
# P(C action | Cooperator type) = 0.9
# P(C action | Defector type) = 0.1

update_belief ← function(prior, likelihood_coop, likelihood_defect, action) {
  if (action == "C") {
    l_coop ← likelihood_coop
    l_defect ← likelihood_defect
  } else {
    l_coop ← 1 - likelihood_coop
    l_defect ← 1 - likelihood_defect
  }
  numerator   ← l_coop * prior
  denominator ← l_coop * prior + l_defect * (1 - prior)
  numerator / denominator
}

prior ← 0.5
actions ← c("C", "C", "D", "C", "C")

beliefs ← numeric(length(actions))
for (i in seq_along(actions)) {
  prior ← update_belief(prior, 0.9, 0.1, actions[i])
  beliefs[i] ← prior
}

tibble(round = seq_along(actions), action = actions, belief = beliefs)
```

```
#> # A tibble: 5 × 3
#>   round action belief
#>   <int> <chr>   <dbl>
#> 1     1 C       0.9
#> 2     2 C       0.988
#> 3     3 D       0.9
#> 4     4 C       0.988
#> 5     5 C       0.999
```

C.7 Law of large numbers and Monte Carlo

The **law of large numbers** guarantees that the sample mean converges to the true mean as the sample size grows. This justifies the Monte Carlo methods used extensively in 16.

```r
# Demonstrate convergence of sample mean
n_values ← c(10, 100, 1000, 10000, 100000)
true_mean ← 3.5  # E[Uniform die roll]

estimates ← vapply(n_values, function(n) {
  mean(sample(1:6, n, replace = TRUE))
}, numeric(1))

tibble(n = n_values, estimate = estimates, true_mean = true_mean)
```

```
#> # A tibble: 5 × 3
#>        n estimate true_mean
#>    <dbl>    <dbl>     <dbl>
#> 1     10     3.2         3.5
#> 2    100     3.58        3.5
#> 3   1000     3.48        3.5
#> 4  10000     3.49        3.5
#> 5 100000     3.50        3.5
```

C.7.1 Monte Carlo estimation

To estimate a quantity $\theta = E[g(X)]$, draw N independent samples $X_1, \ldots, X_N$ and compute:

$$\hat{\theta}_N = \frac{1}{N} \sum_{i=1}^{N} g(X_i)$$

The standard error of this estimate is $\mathrm{SE} = \sigma / \sqrt{N}$, where σ is the standard deviation of $g(X)$.

```r
# Estimate P(at least one cooperator in 5 players) via Monte Carlo
# Each player cooperates independently with probability 0.3

estimate_prob ← function(n_sims, n_players = 5, p_coop = 0.3) {
  counts ← replicate(n_sims, {
    actions ← rbinom(n_players, size = 1, prob = p_coop)
    as.integer(sum(actions) ⩾ 1)
  })
  c(estimate = mean(counts), se = sd(counts) / sqrt(n_sims))
}

# Exact answer: 1 - (1 - 0.3)^5
exact ← 1 - 0.7^5

set.seed(42)
mc ← estimate_prob(100000)
tibble(
  method   = c("Monte Carlo", "Exact"),
  estimate = c(mc["estimate"], exact)
)
```

```
#> # A tibble: 2 × 2
#>   method      estimate
#>   <chr>          <dbl>
#> 1 Monte Carlo    0.834
#> 2 Exact          0.832
```

C.8 Random sampling for games

Several chapters use random sampling to simulate strategic interactions. The key functions are:

```
# sample(): draw from a discrete set
sample(c("Hawk", "Dove"), size = 10, replace = TRUE, prob = c(0.4, 0.6))
```

```
#>  [1] "Dove" "Hawk" "Dove" "Dove" "Hawk" "Hawk" "Dove" "Hawk" "Dove" "Hawk"
```

```
# replicate(): repeat a simulation
replicate(5, sum(sample(1:6, 2, replace = TRUE)))
```

```
#> [1] 4 6 5 6 7
```

```
# set.seed(): ensure reproducibility
set.seed(123)
runif(3)
```

```
#> [1] 0.288 0.788 0.409
```

```
set.seed(123)
runif(3)  # identical output
```

```
#> [1] 0.288 0.788 0.409
```

C.9 Summary of R probability functions

Distribution	PMF/PDF	CDF	Quantile	Random
Binomial	`dbinom()`	`pbinom()`	`qbinom()`	`rbinom()`
Poisson	`dpois()`	`ppois()`	`qpois()`	`rpois()`
Uniform	`dunif()`	`punif()`	`qunif()`	`runif()`
Normal	`dnorm()`	`pnorm()`	`qnorm()`	`rnorm()`
Exponential	`dexp()`	`pexp()`	`qexp()`	`rexp()`

R functions for common probability distributions