University of Chicago Political Science Math Prefresher

Anton Strezhnev

9/12/2023

Table of contents

1 Overview **3**
1.1 Introduction . . . 3
1.2 Course Booklet . . . 3
1.3 Schedule . . . 3
1.4 Software . . . 4
1.5 Acknowledgments . . . 4

2 Sets, Operations, and Functions **5**
2.1 Sets . . . 5

Sets **7**

Sets **8**
2.2 Metric spaces . . . 8
2.3 Operators; Sum and Product notation . . . 9

Operators **12**

Operators **13**
2.4 Introduction to Functions . . . 13

Functions **15**

Functions **16**
2.5 Logarithms and Exponents . . . 16

Logarithms **19**
2.6 Graphing Functions . . . 19
2.7 Solving for Variables and Finding Roots . . . 20

Solving **21**

Roots **22**

3 Limits **23**
Example: The Central Limit Theorem . . . 23
Example: The Law of Large Numbers . . . 24

3.1 Sequences 25

Sequences **26**
3.2 The Limit of a Sequence 27

Ratios **28**

Limits **29**
3.3 Limits of a Function 29

Limits of a function **30**

Limits of a function **31**

Limits of ratios **32**

Limits of a function **33**
3.4 Continuity 33

Continuity **35**

Continuity **36**

4 Calculus **38**
Example: The Mean is a Type of Integral 38
4.1 Derivatives 39

Derivative **40**
Properties of derivatives 41

Power Rule **42**

Derivatives **44**
4.2 Higher-Order Derivatives (Derivatives of Derivatives of Derivatives) 44

Succession of derivatives **46**
4.3 Composite Functions and the Chain Rule 46

Composite functions **47**

Composite Exponent **48**
4.4 Derivatives of natural logs and the exponent 48

Derivative of Exponents/Logs **49**
Derivatives of natural exponential function (e) 49

Derivatives of exponents **51**
Derivatives of logarithms 51

Derivatives of logs **53**
Outline of Proof 53
4.5 Partial Derivatives 54

Partial derivatives **55**

Partial derivatives **56**
4.6 Taylor Series Approximation 56
4.7 The Indefinite Integration 57

Antiderivative **58**

Antiderivative **59**
4.8 Indefinite Integral 59

Graphing **60**
Common Rules of Integration 61

Common Integration **62**
4.9 The Definite Integral: The Area under the Curve 62

The Definite Integral (Riemann) **64**

First Fundamental Theorem of Calculus **65**

Second Fundamental Theorem of Calculus **66**

Definite Integral of a monomial **67**

Indefinite integrals **68**
Common Rules for Definite Integrals 68

Definite integrals **69**
4.10 Integration by Substitution 69

Integration by Substitutiton II **71**
4.11 Integration by Parts 71

Integration by parts **72**

Integration by parts **73**

5 Optimization **74**
Example: Meltzer-Richard . . . 74
5.1 Maxima and Minima . . . 76

Plotting a maximum and minimum **78**

Maxima and Minima by drawing **80**
5.2 Concavity of a Function . . . 80

Concave Function **81**

Convex Function **82**

Quasiconcave Function **83**

Quasiconvex Function **84**
Quadratic Forms . . . 84
Definiteness of Quadratic Forms . . . 85
5.3 FOC and SOC . . . 85
First Order Conditions . . . 86

Gradient **87**

Critical Point **88**
Second Order Conditions . . . 89

Hessian **90**

Max and min with two dimensions **91**
Definiteness and Concavity . . . 91
5.4 Global Maxima and Minima . . . 92

Optimization **93**
5.5 Constrained Optimization . . . 94
Equality Constraints . . . 96

Constrained optimization with two goods and a budget constraint **99**
5.6 Inequality Constraints . . . 100

Constrained optimization **103**
5.7 Kuhn-Tucker Conditions . . . 104

Kuhn-Tucker with two variables **106**

Kuhn-Tucker with logs **108**
5.8 Applications of Quadratic Forms 109

6 Probability Theory **111**
6.1 Counting rules 111

Counting Possibilities **112**

Counting **114**

Counting **115**
6.2 Probability 115
Probability Definitions: Formal and Informal 115

Probability **116**

Axioms of Probability **117**
Probability Operations 117

Probability **118**

Probability **119**
6.3 Conditional Probability and Bayes Rule 119

Conditional Probability 1 **120**

Conditional Probability 2 **121**

Multiplicative Law of Probability **122**

Law of total probability **123**

Bayes' Rule **125**

Conditional Probability **126**
6.4 Independence 126

Independence **127**
6.5 Random Variables 128

Random Variable **130**
6.6 Distributions 130

Distribution of a random variable **131**

Total Number of Occurrences **132**
Discrete Random Variables 132

Discrete Random Variable **133**

Probability Mass Function **134**
Continuous Random Variables 135

Continuous Random Variable **136**

Probability density function **137**

Continuous R.V. **138**
6.7 Joint Distributions 138

Discrete, Joint Distributions **140**
6.8 Expectation 140

Expectation of a discrete R.V. **141**

Expectation of a discrete R.V. **142**

Expectation of a continuous R.V. **143**
Expected Value of a Function 143
Properties of Expected Values 143
6.9 Variance and Covariance 144

Covariance **146**

Correlation **147**

Expectation and Variance 1 **148**

Expectation and Variance 2 **149**

Expectation and Variance 3 **150**
6.10 Distributions 150

Binomial Distribution **151**

Binomial distribution **152**

Poisson Distribution **153**

Poisson Distribution **154**

Uniform Distribution **155**

Uniform **156**

Normal Distribution **157**
6.11 Summarizing Observed Events (Data) 158

Sample Covariance and Correlation **160**
6.12 Asymptotic Theory 160
6.12.1 CLT and LLN 160

Central Limit Theorem **161**

Weak Law of Large Numbers (LLN) **162**
6.12.2 Big $\mathcal{O}$ Notation 162

7 Linear Algebra **164**
7.1 Working with Vectors 164

Vector Algebra **166**

Vector Algebra **167**
7.2 Linear Independence 167

Linear indepencence **168**

Linear indepencence **169**
7.3 Basics of Matrix Algebra 169

Matrix addition **171**

Scalar Multiplication **172**

Matrix multiplication **173**

Matrix Multiplication **175**
7.4 Systems of Linear Equations 175

Linear Equations **177**
7.5 Systems of Equations as Matrices 177
7.6 Finding Solutions to Augmented Matrices and Systems of Equations 178

Solving systems of equations **180**

Solving Systems of Equations **181**
7.7 Rank — and Whether a System Has One, Infinite, or No Solutions 181

Rank of Matrices **183**
7.8 The Inverse of a Matrix . . . 183

Matrix Inverse **185**

Matrix Inverse **186**
7.9 Linear Systems and Inverses . . . 186

Solve linear system using inverses **187**
7.10 Determinants . . . 187

8 Determinants **189**
8.1 Getting Inverse of a Matrix using its Determinant . . . 189

9 Calculate Inverse using Determinant Formula **191**

10 Programming: Orientation and Reading in Data **192**
Motivation: Data and You . . . 192
Where are we? Where are we headed? . . . 192
Check your understanding . . . 192
10.1 General Orientation . . . 193
10.2 But what is R . . . 194
10.3 The Computer and You: Giving Instructions . . . 195
10.4 Base-R vs. tidyverse . . . 195
Dataframe subsetting . . . 196
Read data . . . 197
Visualization . . . 197
10.5 A is for Athens . . . 198
10.5.1 Locating the Data . . . 198
10.5.2 Reading in Data . . . 199
10.5.3 Inspecting . . . 199
10.5.4 Finding observations . . . 201
Exercises . . . 202
1 . . . 202
2 . . . 202
3 . . . 202
4 . . . 202
5 . . . 203

11 Programming: Manipulating Vectors and Matrices **204**
Motivation . . . 204
Where are we? Where are we headed? . . . 204
11.1 Read Data . . . 205
11.2 data.frame vs. matricies . . . 207

11.3 Handling matricies in R . . . 208
11.4 Variable Transformations . . . 211
11.5 Linear Combinations . . . 212
11.6 Matrix Basics . . . 215
Checkpoint . . . 223
1 . . . 223
2 . . . 223
3 . . . 223
Exercises . . . 224
1 . . . 224
2 . . . 224
3 . . . 225
4 . . . 225
5 . . . 225

12 Objects, Functions, Loops **228**
Where are we? Where are we headed? . . . 228
12.1 What is an object? . . . 228
12.1.1 Lists . . . 229
12.2 Making your own objects . . . 231
12.2.1 Seeing R through objects . . . 233
12.2.2 Parsing an object by `str()`s . . . 234
12.3 Types of variables . . . 235
12.3.1 scalars . . . 235
12.3.2 numeric vectors . . . 236
12.3.3 characters (aka strings) . . . 236
12.4 What is a function? . . . 238
12.4.1 Write your own function . . . 238
Checkpoint . . . 240
1 . . . 240
2 . . . 240
3 . . . 240
12.5 What is a package? . . . 240
12.6 Conditionals . . . 241
12.7 For-loops . . . 242
12.8 Nested Loops . . . 244
Exercises . . . 245
Exercise 1: Write your own function . . . 245
Exercise 2: Using Loops . . . 245
Exercise 3: Storing information derived within loops in a global dataframe . . . 245

13 Joins and Merges, Wide and Long **248**
Motivation . . . 248

Where are we? Where are we headed? . . . 248
13.1 Setting up . . . 249
13.2 Create a project directory . . . 249
13.3 Data Sources . . . 249
13.4 Example with 2 Datasets . . . 250
13.5 Loops . . . 251
13.6 Merging . . . 252
13.7 Main Project . . . 254
Task 1: Data Input and Standardization . . . 254
Task 2: Data Merging . . . 255
Task 3: Tabulations and Visualization . . . 255

14 Functionals and Optimization **256**
14.1 More Tidyverse/Data Cleaning . . . 256
Functionals . . . 260
14.1.1 Replacing for loops . . . 261
14.2 Excerise: Optimization . . . 263

15 Simulation **267**
Motivation: Simulation as an Analytical Tool . . . 267
Where are we? Where are we headed? . . . 268
Check your Understanding . . . 268
15.1 Pick a sample, any sample . . . 268
15.2 The `sample()` function . . . 268
15.2.1 Sampling rows from a dataframe . . . 270
15.3 Random numbers from specific distributions . . . 273
`rbinom()` . . . 273
`runif()` . . . 274
`rnorm()` . . . 274
15.4 r, p, and d . . . 321
15.5 `set.seed()` . . . 322
15.6 Calculating an expectation using Monte Carlo . . . 323
Exercise . . . 325
Baccarat . . . 325
15.6.1 Part 1 (-) . . . 327
15.6.2 Part 2 (-) . . . 328
15.6.3 Part 3 (-) . . . 328
15.6.4 Part 4 (-) . . . 329
15.6.5 Part 5 (-) . . . 329
15.6.6 Challenge problem (-) . . . 329

1 Overview

1.1 Introduction

The 2023 UChicago Math Prefresher for incoming Political Science graduate students will be held from September 12-15 and September 18-20. The course is designed as a brief review of math fundamentals – calculus, optimization, probability theory and linear algebra among other topics – as well as an introduction to programming in the R statistical computing language. The course is entirely optional and there are no grades or assignments but we encourage all incoming graduate students to attend if they are able.

1.2 Course Booklet

The course notes for the math and programming sections as well as all practice problems are made available on this website and can be accessed by navigating the menus in the sidebar.

1.3 Schedule

The prefresher will run for a total of seven days September 12-15, and September 18-20 in *Pick Hall 506*. Each day will run from around 9am to 4pm with many breaks in between.

The morning will focus on math instruction. We will have two one hour sessions from 9:30am - 10:30am and 10:45am-11:45am, with a ~15 minute break in between. These sessions will involve a combination of lectures and working through practice problems.

We will break for lunch from 12:00pm-1:00pm.

The afternoon will focus on coding instruction with lecture/demonstration from 1:00pm-2:45pm. After a short break you will work together on a variety of coding exercises from 3:00-3:30pm. In the last 30 minutes we will regroup to wrap up and discuss any questions on the material.

1.4 Software

As the afternoons of the prefresher will involve instruction in coding, you should be sure to bring a laptop and a charge cable. In addition, prior to the start of the prefresher, please make sure to have installed the following on your computer.

- R (version 4.2.1 or higher)
- RStudio Desktop Open Source License (this is the primary IDE or integrated development environment in which we will be working)
- LaTeX: This is primarily to allow you to generate PDF documents using RMarkdown. We will use the TinyTeX LaTeX distribution which is designed to be minimalist and tailored specifically for R users. After installing R and RStudio, open up an instance of R, install the 'tinytex' package and run the `install_tinytex()` command

```
install.packages('tinytex')
tinytex::install_tinytex()
```

We will also spend some time discussing document preparation and typesetting using LaTeX and Markdown. For the former, we will be using the popular cloud platform Overleaf, which allows for collaborative document editing and streamlines a lot of the irritating parts of typesetting in LaTeX. You should register for an account using your university e-mail as all University of Chicago students and faculty have access to an Overleaf Pro account for free.

You are also welcome to install a LaTeX editor on your local machine to work alongside the TinyTeX distribution or any other TeX distribution that you prefer such as TexMaker

1.5 Acknowledgments

This prefresher draws heavily on the wonderful materials that have been developed by over 20 years of instructors at the Harvard Government Math Prefresher that have been so generously distributed under the GPL 3.0 License. Special thanks to Shiro Kuriwaki, Yon Soo Park, and Connor Jerzak for their efforts in converting the original prefresher materials into the easily distributed Markdown format.

2 Sets, Operations, and Functions

2.1 Sets

Sets are the fundamental building blocks of mathematics. Events are not inherently numerical: the onset of war or the stock market crashing is not inherently a number. Sets can define such events, and we wrap math around so that we have a transparent language to communicate about those events. Combining sets with operations, relations, metrics, measures, etc… allows us to define useful mathematical structures. For example, the set of *real numbers* ($\mathbb{R}$) has a notion of *order* as well as defined *operations* of addition and multiplication.

Set : A set is any well defined collection of elements. If x is an element of S, $x \in S$.

Examples:

1. The set of choices available to a player in Rock-Paper-Scissors $\{\text{Rock}, \text{Paper}, \text{Scissors}\}$
2. The set of possible outcomes of a roll of a six-sided die $\{1, 2, 3, 4, 5, 6\}$
3. The set of all natural numbers $\mathbb{N}$
4. The set of all real numbers $\mathbb{R}$

Common mathematical notation relevant to sets:

- $\in$ = "is an element of"; $\notin$ = "is not an element of"
- $\forall$ = "for all" (univeral quantifier)
- $\exists$ = "there exists" (existential quantifier)
- $:$ = "such that"

Subset: If every element of set A is also in set B, then A is a *subset* of B. $A \subseteq B$. If, in addition to being a subset of B, A is not equal to B, A is a *proper subset* $A \subset B$.

Empty Set: a set with no elements. $S = \{\}$. It is denoted by the symbol $\emptyset$.

Cardinality: The cardinality of a set S, typically written $|S|$ is the number of members of S.

Many sets are infinite. For example, $\mathbb{N}$ the set of natural numbers $\mathbb{N} = \{0, 1, 2, 3, 4, ...\}$ - Sets with cardinality less than $|\mathbb{N}|$ are *countable* - Sets with the same cardinality as *mathbbN*| are *countably infinite* - Sets with greater cardinality than $|\mathbb{N}|$ are *uncountably infinite* (e.g. the real numbers).

Set operations:

1. **Union**: The union of two sets A and B, $A \cup B$, is the set containing all of the elements in A or B. $A_1 \cup A_2 \cup \cdots \cup A_n = \bigcup_{i=1}^{n} A_i$
2. **Intersection**: The intersection of sets A and B, $A \cap B$, is the set containing all of the elements in both A and B. $A_1 \cap A_2 \cap \cdots \cap A_n = \bigcap_{i=1}^{n} A_i$
3. **Complement**: If set A is a subset of S, then the complement of A, denoted A^C, is the set containing all of the elements in S that are not in A.

Properties of set operations:

- **Commutative**: $A \cup B = B \cup A$; $A \cap B = B \cap A$
- **Associative**: $A \cup (B \cup C) = (A \cup B) \cup C$; $A \cap (B \cap C) = (A \cap B) \cap C$
- **Distributive**: $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$; $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
- **de Morgan's laws**: $(A \cup B)^C = A^C \cap B^C$; $(A \cap B)^C = A^C \cup B^C$
- **Disjointness**: Sets are disjoint when they do not intersect, such that $A \cap B = \emptyset$. A collection of sets is pairwise disjoint (**mutually exclusive**) if, for all $i \neq j$, $A_i \cap A_j = \emptyset$. A collection of sets form a partition of set S if they are pairwise disjoint and they cover set S, such that $\bigcup_{i=1}^{k} A_i = S$.

Example 2.1.

Sets

Let set A be $\{1, 2, 3, 4\}$, B be $\{3, 4, 5, 6\}$, and C be $\{5, 6, 7, 8\}$. Sets A, B, and C are all subsets of the S which is $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$

Write out the following sets:

1. $A \cup B$
2. $C \cap B$
3. B^c
4. $A \cap (B \cup C)$

Exercise 2.1.

Sets

Suppose you had a pair of four-sided dice. You sum the results from a single toss.

What is the set of possible outcomes?

Consider subsets $A = \{2, 8\}$ and $B = \{2, 3, 7\}$ of the sample space you found. What is

1. A^c
2. $(A \cup B)^c$

2.2 Metric spaces

A *metric space* is a set that has a notion of *distance* - called a "metric" - defined between any two elements (sometimes referred to as "points").

The distance function $d(x, y)$ defines the distance between element x and element y

- The real numbers $\mathbb{R}$ have a single distance function: $d(x, y) = |x - y|$
- In higher-dimensional real space (e.g. $\mathbb{R}^2$), we can define multiple distance metrics between $x = (x_1, x_2)$ and $y = (y_1, y_2)$
  - "Euclidean" distance: $d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$
  - "Taxicab" distance: $d(x, y) = |x_1 - y_1| + |x_2 - y_2|$
  - Chebyshev distance: $d(x, y) = \max\{|x_1 - y_1| + |x_2 - y_2|\}$
- All of these generalize to $\mathbb{R}^n$

A metric is a function that satisfies the following axioms

1. A distance between a point and itself is zero $d(x, x) = 0$
2. The distance between two points is strictly positive $d(x, y) > 0 \forall x \neq y$
3. Distance from x to y is the same as the distance from y to x $(d(x, y) = d(y, x))$
4. The "triangle inequality" holds: $d(x, z) \leq d(x, y) + d(y, z)$

Once we have a metric space, we can define some additional useful concepts

Ball: A ball of radius r centered at x_0 is a set that contains all points with a distance less than r from x_0.

Sphere: A sphere of radius r centered at x_0 is the set that contains all points with a distance exactly r from x_0.

Interior Point: The point x is an interior point of the set S if x is in S and if there is some ϵ-ball around x that contains only points in S. The **interior** of S is the collection of all interior points in S. The interior can also be defined as the union of all open sets in S.

- If the set S is circular, the interior points are everything inside of the circle, but not on the circle's rim.
- Example: The interior of the set $\{(x, y) : x^2 + y^2 \leq 4\}$ is $\{(x, y) : x^2 + y^2 < 4\}$.

Boundary Point: The point $\mathbf{x}$ is a boundary point of the set S if every ϵ-ball around $\mathbf{x}$ contains both points that are in S and points that are outside S. The **boundary** is the collection of all boundary points.

- If the set S is circular, the boundary points are everything on the circle's rim.
- Example: The boundary of $\{(x, y) : x^2 + y^2 \leq 4\}$ is $\{(x, y) : x^2 + y^2 = 4\}$.

Open: A set S is open if for each point $\mathbf{x}$ in S, there exists an open ϵ-ball around $\mathbf{x}$ completely contained in S.

- If the set S is circular and open, the points contained within the set get infinitely close to the circle's rim, but do not touch it.
- Example: $\{(x, y) : x^2 + y^2 < 4\}$

Closed: A set S is closed if it contains all of its boundary points.

- Alternatively: A set is closed if its complement is open.
- If the set S is circular and closed, the set contains all points within the rim as well as the rim itself.
- Example: $\{(x, y) : x^2 + y^2 \leq 4\}$
- Note: a set may be neither open nor closed. Example: $\{(x, y) : 2 < x^2 + y^2 \leq 4\}$

2.3 Operators; Sum and Product notation

Addition (+), Subtraction (-), multiplication and division are basic operations of arithmetic. In statistics or calculus, we will often want to add a *sequence* of numbers that can be expressed as a pattern without needing to write down all its components. For example, how would we express the sum of all numbers from 1 to 100 without writing a hundred numbers?

For this we use the summation operator $\sum$ and the product operator $\prod$.

Summation:

$$\sum_{i=1}^{100} x_i = x_1 + x_2 + x_3 + \cdots + x_{100}$$

The bottom of the $\sum$ symbol indicates an index (here, i), and its start value 1. At the top is where the index ends. The notion of "addition" is part of the $\sum$ symbol. The content to the right of the summation is the meat of what we add. While you can pick your favorite index, start, and end values, the content must also have the index.

A few important features of sums:

- $\sum_{i=1}^{n} cx_i = c \sum_{i=1}^{n} x_i$
- $\sum_{i=1}^{n} (x_i + y_i) = \sum_{i=1}^{n} x_i + \sum_{i=1}^{n} y_i$
- $\sum_{i=1}^{n} c = nc$

Product:

$$\prod_{i=1}^{n} x_i = x_1 x_2 x_3 \cdots x_n$$

Properties:

- $\prod_{i=1}^{n} cx_i = c^n \prod_{i=1}^{n} x_i$
- $\prod_{i=k}^{n} cx_i = c^{n-k+1} \prod_{i=k}^{n} x_i$
- $\prod_{i=1}^{n} (x_i + y_i) =$ a total mess
- $\prod_{i=1}^{n} c = c^n$

Other Useful Operations

Factorials!:

$$x! = x \cdot (x-1) \cdot (x-2) \cdots (1)$$

Modulo: Tells you the remainder when you divide the first number by the second.

- $17 \mod 3 = 2$
- $100 \ \% \ 30 = 10$

Example 2.2.

Operators

1. $\sum_{i=1}^{5} i =$
2. $\prod_{i=1}^{5} i =$
3. $14 \mod 4 =$
4. $4! =$

Exercise 2.2.

Operators

Let $x_1 = 4, x_2 = 3, x_3 = 7, x_4 = 11, x_5 = 2$

1. $\sum_{i=1}^{3}(7)x_i$
2. $\sum_{i=1}^{5} 2$
3. $\prod_{i=3}^{5}(2)x_i$

2.4 Introduction to Functions

A **function** is a mapping, or transformation, that relates members of one set to members of another set. For instance, if you have two sets: set A and set B, a function from A to B maps every value a in set A such that $f(a) \in B$. Functions can be "many-to-one", where many values or combinations of values from set A produce a single output in set B, or they can be "one-to-one", where each value in set A corresponds to a single value in set B. A function by definition has a single function value for each element of its domain. This means, there cannot be "one-to-many" mapping.

Dimensionality: $\mathbf{R}^1$ is the set of all real numbers extending from $-\infty$ to $+\infty$ — i.e., the real number line. $\mathbf{R}^n$ is an n-dimensional space, where each of the n axes extends from $-\infty$ to $+\infty$.

- $\mathbf{R}^1$ is a one dimensional line.
- $\mathbf{R}^2$ is a two dimensional plane.
- $\mathbf{R}^3$ is a three dimensional space.

Points in $\mathbf{R}^n$ are ordered n-tuples (just means an combination of n elements where order matters), where each element of the n-tuple represents the coordinate along that dimension.

For example:

- $\mathbf{R}^1$: (3)
- $\mathbf{R}^2$: (-15, 5)

- $\mathbf{R}^3$: (86, 4, 0)

Examples of mapping notation:

Function of one variable: $f : \mathbf{R}^1 \to \mathbf{R}^1$

- $f(x) = x + 1$. For each x in $\mathbf{R}^1$, $f(x)$ assigns the number $x + 1$.

Function of two variables: $f : \mathbf{R}^2 \to \mathbf{R}^1$.

- $f(x, y) = x^2 + y^2$. For each ordered pair (x, y) in $\mathbf{R}^2$, $f(x, y)$ assigns the number $x^2 + y^2$.

We often use variable x as input and another y as output, e.g. $y = x + 1$

Example 2.3.

Functions

For each of the following, state whether they are one-to-one or many-to-one functions.

1. For $x \in [0, \infty]$, $f : x \to x^2$ (this could also be written as $f(x) = x^2$).
2. For $x \in [-\infty, \infty]$, $f : x \to x^2$.

Exercise 2.3.

Functions

For each of the following, state whether they are one-to-one or many-to-one functions.

1. For $x \in [-3, \infty]$, $f : x \to x^2$.
2. For $x \in [0, \infty]$, $f : x \to \sqrt{x}$

Some functions are defined only on proper subsets of $\mathbf{R}^n$.

- **Domain**: the set of numbers in X at which $f(x)$ is defined.
- **Range**: elements of Y assigned by $f(x)$ to elements of X, or $f(X) = \{y : y = f(x), x \in X\}$ Most often used when talking about a function $f : \mathbf{R}^1 \to \mathbf{R}^1$.
- **Image**: same as range, but more often used when talking about a function $f : \mathbf{R}^n \to \mathbf{R}^1$.

Some General Types of Functions

Monomials: $f(x) = ax^k$

a is the coefficient. k is the degree.

Examples: $y = x^2$, $y = -\frac{1}{2}x^3$

Polynomials: sum of monomials.

Examples: $y = -\frac{1}{2}x^3 + x^2$, $y = 3x + 5$

The degree of a polynomial is the highest degree of its monomial terms. Also, it's often a good idea to write polynomials with terms in decreasing degree.

2.5 Logarithms and Exponents

Exponential Functions: Example: $y = 2^x$

Relationship of logarithmic and exponential functions:

$$y = \log_a(x) \iff a^y = x$$

The log function can be thought of as an inverse for exponential functions. a is referred to as the "base" of the logarithm.

Common Bases: The two most common logarithms are base 10 and base e.

1. Base 10: $y = \log_{10}(x) \iff 10^y = x$. The base 10 logarithm is often simply written as "log(x)" with no base denoted.
2. Base e: $y = \log_e(x) \iff e^y = x$. The base e logarithm is referred to as the "natural" logarithm and is written as "ln(x)".

Properties of exponential functions:

- $a^x a^y = a^{x+y}$
- $a^{-x} = 1/a^x$
- $a^x/a^y = a^{x-y}$
- $(a^x)^y = a^{xy}$
- $a^0 = 1$

Properties of logarithmic functions (any base):

Generally, when statisticians or social scientists write $\log(x)$ they mean $\log_e(x)$. In other words: $\log_e(x) \equiv \ln(x) \equiv \log(x)$

$$\log_a(a^x) = x$$

and

$$a^{\log_a(x)} = x$$

- $\log(xy) = \log(x) + \log(y)$
- $\log(x^y) = y\log(x)$
- $\log(1/x) = \log(x^{-1}) = -\log(x)$
- $\log(x/y) = \log(x \cdot y^{-1}) = \log(x) + \log(y^{-1}) = \log(x) - \log(y)$
- $\log(1) = \log(e^0) = 0$

Change of Base Formula: Use the change of base formula to switch bases as necessary:

$$\log_b(x) = \frac{\log_a(x)}{\log_a(b)}$$

Example:

$$\log_{10}(x) = \frac{\ln(x)}{\ln(10)}$$

You can use logs to go between sum and product notation. This will be particularly important when you're learning how to optimize likelihood functions.

$$\begin{aligned}
\log\left(\prod_{i=1}^{n} x_i\right) &= \log(x_1 \cdot x_2 \cdot x_3 \cdots \cdot x_n) \\
&= \log(x_1) + \log(x_2) + \log(x_3) + \cdots + \log(x_n) \\
&= \sum_{i=1}^{n} \log(x_i)
\end{aligned}$$

Therefore, you can see that the log of a product is equal to the sum of the logs. We can write this more generally by adding in a constant, c:

$$\begin{aligned}
\log\left(\prod_{i=1}^{n} cx_i\right) &= \log(cx_1 \cdot cx_2 \cdots cx_n) \\
&= \log(c^n \cdot x_1 \cdot x_2 \cdots x_n) \\
&= \log(c^n) + \log(x_1) + \log(x_2) + \cdots + \log(x_n) \\
&= n\log(c) + \sum_{i=1}^{n} \log(x_i)
\end{aligned}$$

Example 2.4.

Logarithms

Evaluate each of the following logarithms

1. $\log_4(16)$
2. $\log_2(16)$

Simplify the following logarithm. By "simplify", we actually really mean - use as many of the logarithmic properties as you can.

3. $\log_4(x^3y^5)$

Exercise 2.4. Evaluate each of the following logarithms

1. $\log_{\frac{3}{2}}(\frac{27}{8})$

Simplify each of the following logarithms. By "simplify", we actually really mean - use as many of the logarithmic properties as you can.

2. $\log(\frac{x^9y^5}{z^3})$
3. $\ln\sqrt{xy}$

2.6 Graphing Functions

What can a graph tell you about a function?

- Is the function increasing or decreasing? Over what part of the domain?
- How "fast" does it increase or decrease?
- Are there global or local maxima and minima? Where?
- Are there inflection points?
- Is the function continuous?
- Is the function differentiable?
- Does the function tend to some limit?
- Other questions related to the substance of the problem at hand.

2.7 Solving for Variables and Finding Roots

Sometimes we're given a function $y = f(x)$ and we want to find how x varies as a function of y. Use algebra to move x to the left hand side (LHS) of the equation and so that the right hand side (RHS) is only a function of y.

Example 2.5.

Solving

Solve for x:

1. $y = 3x + 2$
2. $y = e^x$

Solving for variables is especially important when we want to find the **roots** of an equation: those values of variables that cause an equation to equal zero. Especially important in finding equilibria and in doing maximum likelihood estimation.

Procedure: Given $y = f(x)$, set $f(x) = 0$. Solve for x.

Multiple Roots:

$$f(x) = x^2 - 9 \quad \Longrightarrow \quad 0 = x^2 - 9 \quad \Longrightarrow \quad 9 = x^2 \quad \Longrightarrow \quad \pm\sqrt{9} = \sqrt{x^2} \quad \Longrightarrow \quad \pm 3 = x$$

Quadratic Formula: For quadratic equations $ax^2 + bx + c = 0$, use the quadratic formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Exercise 2.5.

Roots

Solve for x:

1. $f(x) = 3x + 2 = 0$
2. $f(x) = x^2 + 3x - 4 = 0$
3. $f(x) = e^{-x} - 10 = 0$

3 Limits

Solving limits, i.e. finding out the value of functions as its input moves closer to some value, is important for the social scientist's mathematical toolkit for two related tasks. The first is for the study of calculus, which will be in turn useful to show where certain functions are maximized or minimized. The second is for the study of statistical inference, which is the study of inferring things about things you cannot see by using things you can see.

Example: The Central Limit Theorem

Perhaps the most important theorem in statistics is the Central Limit Theorem,

Theorem 3.1 (Central Limit Theorem)**.** *For any series of independent and identically distributed random variables $X_1, X_2, \cdots$, we know the distribution of its sum even if we do not know the distribution of X. The distribution of the sum is a Normal distribution.*

$$\frac{\bar{X}_n - \mu}{\sigma/\sqrt{n}} \xrightarrow{d} Normal(0,1)$$

where μ is the mean of X and σ is the standard deviation of X. The arrow is read as "converges in distribution to". $Normal(0,1)$ indicates a Normal Distribution with mean 0 and variance 1.

That is, the limit of the distribution of the lefthand side is the distribution of the righthand side.

The sign of a limit is the arrow "$\rightarrow$". Although we have not yet covered probability so we have not described what distributions and random variables are, it is worth foreshadowing the Central Limit Theorem. The Central Limit Theorem is powerful because it gives us a *guarantee* of what would happen if $n \rightarrow \infty$, which in this case means we collected more data.

Example: The Law of Large Numbers

A finding that perhaps rivals the Central Limit Theorem is the (Weak) Law of Large Numbers:

Theorem 3.2 ((Weak) Law of Large Numbers)**.** *For any draw of identically distributed independent variables with mean μ, the sample average after n draws, $\bar{X}_n$, converges in probability to the true mean as $n \to \infty$:*

$$\lim_{n\to\infty} P(|\bar{X}_n - \mu| > \varepsilon) = 0$$

A shorthand of which is $\bar{X}_n \xrightarrow{p} \mu$, where the arrow is read as "converges in probability to".

Intuitively, the more data, the more accurate is your guess. For example, Figure 3.1 shows how the sample average from many coin tosses converges to the true value : 0.5.

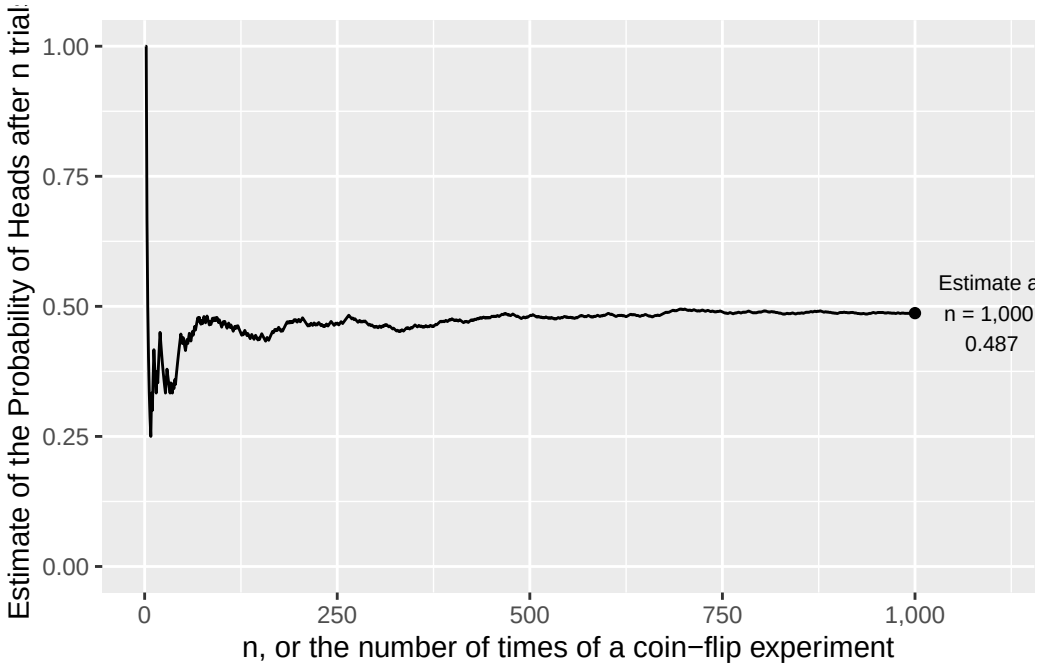


Figure 3.1: As the number of coin tosses goes to infinity, the average probabiity of heads converges to 0.5

3.1 Sequences

We need a couple of steps until we get to limit theorems in probability. First we will introduce a "sequence", then we will think about the limit of a sequence, then we will think about the limit of a *function*.

A **sequence** $\{x_n\} = \{x_1, x_2, x_3, \ldots, x_n\}$ is an ordered set of real numbers, where x_1 is the first term in the sequence and y_n is the nth term. Generally, a sequence is infinite, that is it extends to $n = \infty$. We can also write the sequence as $\{x_n\}_{n=1}^{\infty}$

where the subscript and superscript are read together as "from 1 to infinity."

Example 3.1.

Sequences

How do these sequences behave?

1. $\{A_n\} = \{2 - \frac{1}{n^2}\}$
2. $\{B_n\} = \left\{\frac{n^2+1}{n}\right\}$
3. $\{C_n\} = \{(-1)^n \left(1 - \frac{1}{n}\right)\}$

We find the sequence by simply "plugging in" the integers into each n. The important thing is to get a sense of how these numbers are going to change.

Graphing helps you make this point more clearly. See the sequence of $n = 1, ...20$ for each of the three examples in Figure 3.2.

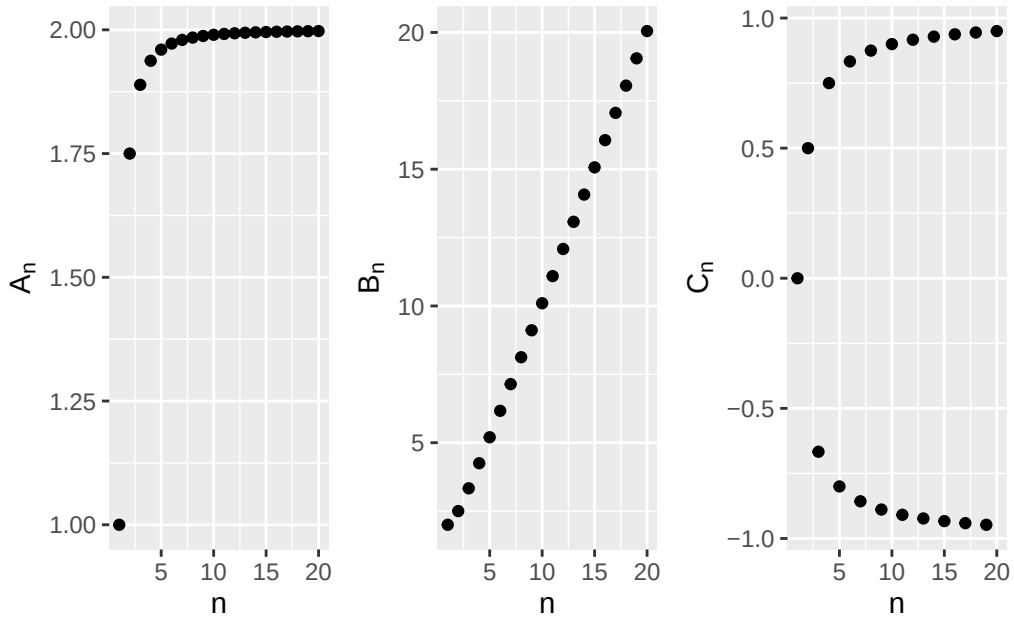


Figure 3.2: Behavior of Some Sequences

3.2 The Limit of a Sequence

The notion of "converging to a limit" is the behavior of the points in Example -@#exm-seqbehav. In some sense, that's the counterfactual we want to know. What happens as $n \to \infty$?

1. Sequences like 1 above that converge to a limit.
2. Sequences like 2 above that increase without bound.
3. Sequences like 3 above that neither converge nor increase without bound — alternating over the number line.

Definition: Limit The sequence $\{y_n\}$ has the limit L, which we write as $\lim_{n\to\infty} y_n = L$, if for any $\epsilon > 0$ there is an integer N (which depends on ϵ) with the property that $|y_n - L| < \epsilon$ for each $n > N$. $\{y_n\}$ is said to converge to L. If the above does not hold, then $\{y_n\}$ diverges.

We can also express the behavior of a sequence as bounded or not:

1. Bounded: if $|y_n| \leq K$ for all n
2. Monotonically Increasing: $y_{n+1} > y_n$ for all n
3. Monotonically Decreasing: $y_{n+1} < y_n$ for all n

A limit is *unique*: If $\{y_n\}$ converges, then the limit L is unique.

If a sequence converges, then the sum of such sequences also converges. Let $\lim_{n\to\infty} y_n = y$ and $\lim_{n\to\infty} z_n = z$. Then

1. $\lim_{n\to\infty} [ky_n + \ell z_n] = ky + \ell z$
2. $\lim_{n\to\infty} y_n z_n = yz$
3. $\lim_{n\to\infty} \frac{y_n}{z_n} = \frac{y}{z}$, provided $z \neq 0$

This looks reasonable enough. The harder question, obviously is when the parts of the fraction *don't* converge. If $\lim_{n\to\infty} y_n = \infty$ and $\lim_{n\to\infty} z_n = \infty$, What is $\lim_{n\to\infty} y_n - z_n$? What is $\lim_{n\to\infty} \frac{y_n}{z_n}$?

It is nice for a sequence to converge in limit. We want to know if complex-looking sequences converge or not. The name of the game here is to break that complex sequence up into sums of simple fractions where n only appears in the denominator: $\frac{1}{n}$, $\frac{1}{n^2}$, and so on. Each of these will converge to 0, because the denominator gets larger and larger. Then, because of the properties above, we can then find the final sequence.

Example 3.2.

Ratios

Find the limit of $\lim_{n\to\infty} \frac{n+3}{n}$

Solution. At first glance, $n+3$ and n both grow to ∞, so it looks like we need to divide infinity by infinity. However, we can express this fraction as a sum, then the limits apply separately:

$$\lim_{n\to\infty} \frac{n+3}{n} = \lim_{n\to\infty} \left(1 + \frac{3}{n}\right) = \underbrace{\lim_{n\to\infty} 1}_{1} + \underbrace{\lim_{n\to\infty} \left(\frac{3}{n}\right)}_{0}$$

so, the limit is actually 1.

After some practice, the key to intuition is whether one part of the fraction grows "faster" than another. If the denominator grows faster to infinity than the numerator, then the fraction will converge to 0, even if the numerator will also increase to infinity. In a sense, limits show how not all infinities are the same.

Exercise 3.1.

Limits

Find the following limits of sequences, then explain in English the intuition for why that is the case.

1. $\lim_{n\to\infty} \frac{2n}{n^2+1}$
2. $\lim_{n\to\infty} (n^3 - 100n^2)$

3.3 Limits of a Function

We've now covered functions and just covered limits of sequences, so now is the time to combine the two.

A function f is a compact representation of some behavior we care about. Like for sequences, we often want to know if $f(x)$ approaches some number L as its independent variable x moves to some number c (which is usually 0 or $\pm\infty$). If it does, we say that the limit of $f(x)$, as x approaches c, is L: $\lim_{x\to c} f(x) = L$. Unlike a sequence, x is a continuous number, and we can move in decreasing order as well as increasing.

For a limit L to exist, the function $f(x)$ must approach L from both the left (increasing) and the right (decreasing).

Definition 3.1.

Limits of a function

Let $f(x)$ be defined at each point in some open interval containing the point c. Then L equals $\lim_{x\to c} f(x)$ if for any (small positive) number ϵ, there exists a corresponding number $\delta > 0$ such that if $0 < |x - c| < \delta$, then $|f(x) - L| < \epsilon$.

A neat, if subtle result is that $f(x)$ does not necessarily have to be defined at c for $\lim_{x\to c}$ to exist.

Properties: Let f and g be functions with $\lim_{x\to c} f(x) = k$ and $\lim_{x\to c} g(x) = \ell$.

1. $\lim_{x\to c}[f(x) + g(x)] = \lim_{x\to c} f(x) + \lim_{x\to c} g(x)$
2. $\lim_{x\to c} kf(x) = k \lim_{x\to c} f(x)$
3. $\lim_{x\to c} f(x)g(x) = \left[\lim_{x\to c} f(x)\right] \cdot \left[\lim_{x\to c} g(x)\right]$
4. $\lim_{x\to c} \frac{f(x)}{g(x)} = \frac{\lim_{x\to c} f(x)}{\lim_{x\to c} g(x)}$, provided $\lim_{x\to c} g(x) \neq 0$.

Simple limits of functions can be solved as we did limits of sequences. Just be careful which part of the function is changing.

Example 3.3.

Limits of a function

Find the limit of the following functions.

1. $\lim_{x \to c} k$
2. $\lim_{x \to c} x$
3. $\lim_{x \to 2} (2x - 3)$
4. $\lim_{x \to c} x^n$

Limits can get more complex in roughly two ways. First, the functions may become large polynomials with many moving pieces. Second, the functions may become discontinuous.

The function can be thought of as a more general or "smooth" version of sequences. For example,

Example 3.4.

Limits of ratios

Find the limit of

$$\lim_{x\to\infty} \frac{(x^4+3x-99)(2-x^5)}{(18x^7+9x^6-3x^2-1)(x+1)}$$

Now, the functions will become a bit more complex:

Exercise 3.2.

Limits of a function

Solve the following limits of functions

1. $\lim_{x \to 0} |x|$
2. $\lim_{x \to 0} \left(1 + \frac{1}{x^2}\right)$

So there are a few more alternatives about what a limit of a function could be:

1. Right-hand limit: The value approached by $f(x)$ when you move from right to left.
2. Left-hand limit: The value approached by $f(x)$ when you move from left to right.
3. Infinity: The value approached by $f(x)$ as x grows infinitely large. Sometimes this may be a number; sometimes it might be ∞ or $-\infty$.
4. Negative infinity: The value approached by $f(x)$ as x grows infinitely negative. Sometimes this may be a number; sometimes it might be ∞ or $-\infty$.

The distinction between left and right becomes important when the function is not determined for some values of x. What are those cases in the examples below?

3.4 Continuity

To repeat a finding from the limits of functions: $f(x)$ does not necessarily have to be defined at c for $\lim_{x \to c}$ to exist. Functions that have breaks in their lines are called discontinuous. Functions that have no breaks are called continuous. Continuity is a concept that is more fundamental to, but related to that of "differentiability", which we will cover next in calculus.

Definition 3.2.

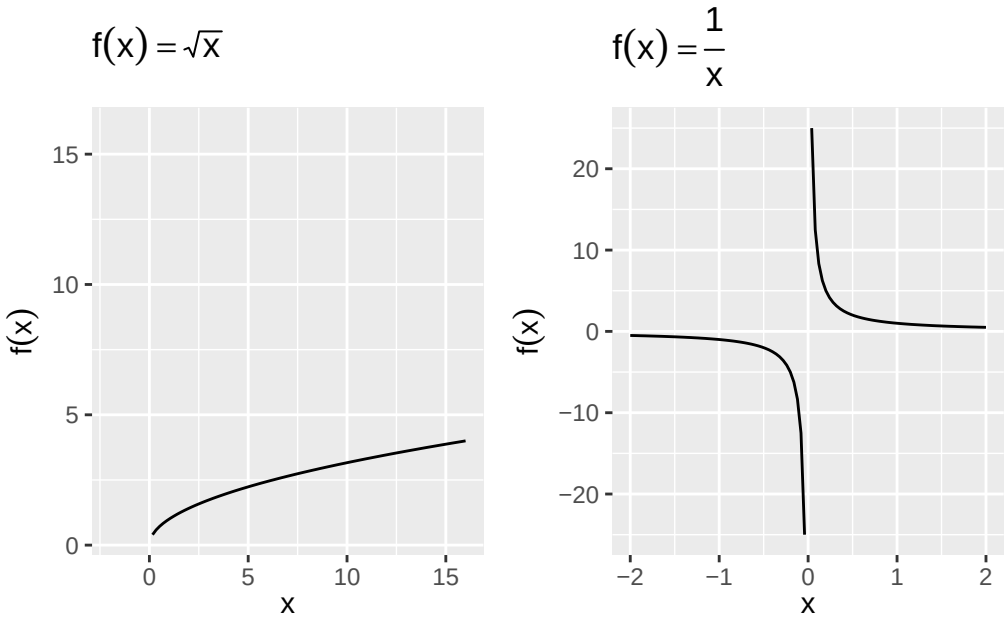


Figure 3.3: Functions which are not defined in some areas

Continuity

Suppose that the domain of the function f includes an open interval containing the point c. Then f is continuous at c if $\lim_{x \to c} f(x)$ exists and if $\lim_{x \to c} f(x) = f(c)$. Further, f is continuous on an open interval (a, b) if it is continuous at each point in the interval.

To prove that a function is continuous for all points is beyond this practical introduction to math, but the general intuition can be grasped by graphing.

Example 3.5.

Continuity

For each function, determine if it is continuous or discontinuous.

1. $f(x) = \sqrt{x}$
2. $f(x) = e^x$
3. $f(x) = 1 + \frac{1}{x^2}$
4. $f(x) = \text{floor}(x)$.

The floor is the smaller of the two integers bounding a number. So $\text{floor}(x = 2.999) = 2$, $\text{floor}(x = 2.0001) = 2$, and $\text{floor}(x = 2) = 2$.

Solution. In Figure 3.4, we can see that the first two functions are continuous, and the next two are discontinuous. $f(x) = 1 + \frac{1}{x^2}$ is discontinuous at $x = 0$, and $f(x) = \text{floor}(x)$ is discontinuous at each whole number.

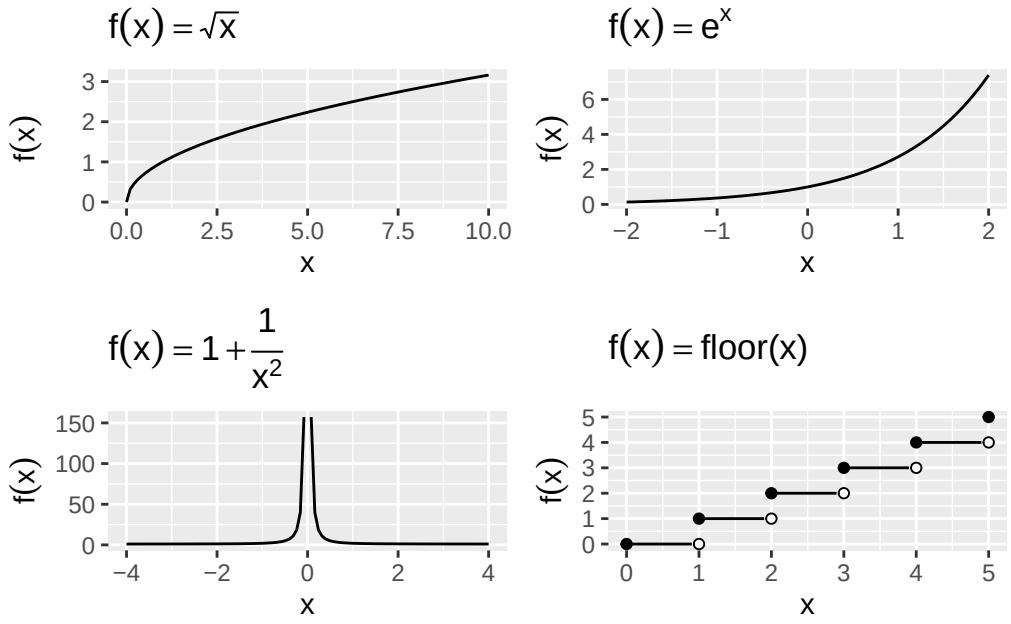


Figure 3.4: Continuous and Discontinuous Functions

Some properties of continuous functions:

1. If f and g are continuous at point c, then $f+g$, $f-g$, $f \cdot g$, $|f|$, and αf are continuous at point c also. f/g is continuous, provided $g(c) \neq 0$.
2. Boundedness: If f is continuous on the closed bounded interval $[a, b]$, then there is a number K such that $|f(x)| \leq K$ for each x in $[a, b]$.
3. Max/Min: If f is continuous on the closed bounded interval $[a, b]$, then f has a maximum and a minimum on $[a, b]$. They may be located at the end points.

Exercise

Let $f(x) = \frac{x^2+2x}{x}$.

1. Graph the function. Is it defined everywhere?
2. What is the functions limit at $x \to 0$?

4 Calculus

Calculus is a fundamental part of any type of statistics exercise. Although you may not be taking derivatives and integral in your daily work as an analyst, calculus undergirds many concepts we use: maximization, expectation, and cumulative probability.

Example: The Mean is a Type of Integral

The average of a quantity is a type of weighted mean, where the potential values are weighted by their likelihood, loosely speaking. The integral is actually a general way to describe this weighted average when there are conceptually an infinite number of potential values.

If X is a continuous random variable, its expected value $E(X)$ – the center of mass – is given by

$$E(X) = \int_{-\infty}^{\infty} x f(x) dx$$

where $f(x)$ is the probability density function of X.

This is a continuous version of the case where X is discrete, in which case

$$E(X) = \sum_{j=1}^{\infty} x_j P(X = x_j)$$

even more concretely, if the potential values of X are finite, then we can write out the expected value as a weighted mean, where the weights is the probability that the value occurs.

$$E(X) = \sum_{x} \quad \left(\underbrace{x}_{\text{value}} \cdot \underbrace{P(X = x)}_{\text{weight, or PMF}} \right)$$

4.1 Derivatives

The derivative of f at x is its rate of change at x: how much $f(x)$ changes with a change in x. The rate of change is a fraction — rise over run — but because not all lines are straight and the rise over run formula will give us different values depending on the range we examine, we need to take a limit (Section -Chapter 3).

Definition 4.1.

Derivative

Let f be a function whose domain includes an open interval containing the point x. The derivative of f at x is given by

$$\frac{d}{dx}f(x) = \lim_{h\to 0}\frac{f(x+h)-f(x)}{(x+h)-x} = \lim_{h\to 0}\frac{f(x+h)-f(x)}{h}$$

There are a two main ways to denote a derivate:

- Leibniz Notation: $\frac{d}{dx}(f(x))$
- Prime or Lagrange Notation: $f'(x)$

If $f(x)$ is a straight line, the derivative is the slope. For a curve, the slope changes by the values of x, so the derivative is the slope of the line tangent to the curve at x. See, For example, Figure -Figure 4.1

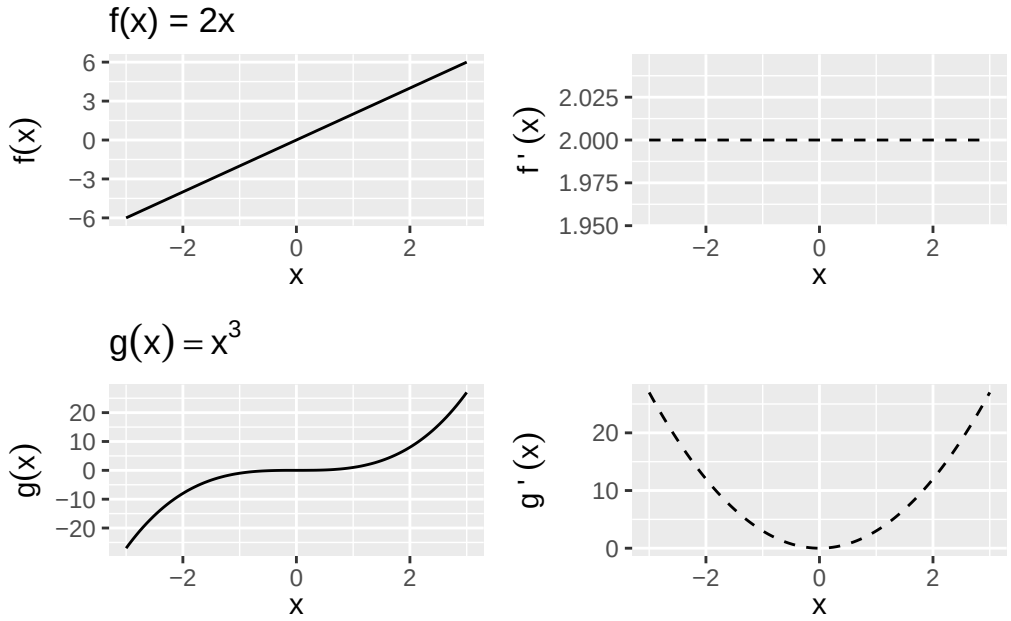


Figure 4.1: The Derivative as a Slope

If $f'(x)$ exists at a point x_0, then f is said to be **differentiable** at x_0. That also implies that $f(x)$ is continuous at x_0.

Properties of derivatives

Suppose that f and g are differentiable at x and that α is a constant. Then the functions $f \pm g$, αf, fg, and f/g (provided $g(x) \neq 0$) are also differentiable at x. Additionally,

Constant rule:

$$[kf(x)]' = kf'(x)$$

Sum rule:

$$[f(x) \pm g(x)]' = f'(x) \pm g'(x)$$

With a bit more algebra, we can apply the definition of derivatives to get a formula for of the derivative of a product and a derivative of a quotient.

Product rule:

$$[f(x)g(x)]' = f'(x)g(x) + f(x)g'(x)$$

Quotient rule:

$$[f(x)/g(x)]' = \frac{f'(x)g(x) - f(x)g'(x)}{[g(x)]^2},\ g(x) \neq 0$$

Finally, one way to think of the power of derivatives is that it takes a function a notch down in complexity. The power rule applies to any higher-order function:

Power rule:

$$\left[x^k\right]' = kx^{k-1}$$

For any real number k (that is, both whole numbers and fractions). The power rule is proved **by induction**, a neat method of proof used in many fundamental applications to prove that a general statement holds for every possible case, even if there are countably infinite cases. We'll show a simple case where k is an integer here.

Proposition 4.1.

Power Rule

$$\left[x^k\right]' = kx^{k-1}$$

for any integer k.

Proof. First, consider the first case (the base case) of $k = 1$. We can show by the definition of derivatives (setting $f(x) = x^1 = 1$) that

$$[x^1]' = \lim_{h \to 0} \frac{(x+h) - x}{(x+h) - x} = 1.$$

Because 1 is also expressed as $1x^{1-1}$, the statement we want to prove holds for the case $k = 1$.

Now, *assume* that the statement holds for some integer m. That is, assume

$$\left[x^m\right]' = mx^{m-1}$$

Then, for the case $m + 1$, using the product rule above, we can simplify

$$\begin{aligned}
\left[x^{m+1}\right]' &= [x^m \cdot x]' \\
&= (x^m)' \cdot x + (x^m) \cdot (x)' \\
&= mx^{m-1} \cdot x + x^m \quad \because \text{by previous assumption} \\
&= mx^m + x^m \\
&= (m+1)x^m \\
&= (m+1)x^{(m+1)-1}
\end{aligned}$$

Therefore, the rule holds for the case $k = m + 1$ once we have assumed it holds for $k = m$. Combined with the first case, this completes proof by induction – we have now proved that the statement holds for all integers $k = 1, 2, 3, \cdots$.

To show that it holds for real fractions as well, we can prove expressing that exponent by a fraction of two integers.

□

These “rules” become apparent by applying the definition of the derivative above to each of the things to be “derived”, but these come up so frequently that it is best to repeat until it is muscle memory.

Exercise 4.1.

Derivatives

For each of the following functions, find the first-order derivative $f'(x)$.

1. $f(x) = c$
2. $f(x) = x$
3. $f(x) = x^2$
4. $f(x) = x^3$
5. $f(x) = \frac{1}{x^2}$
6. $f(x) = (x^3)(2x^4)$
7. $f(x) = x^4 - x^3 + x^2 - x + 1$
8. $f(x) = (x^2 + 1)(x^3 - 1)$
9. $f(x) = 3x^2 + 2x^{1/3}$
10. $f(x) = \frac{x^2+1}{x^2-1}$

4.2 Higher-Order Derivatives (Derivatives of Derivatives of Derivatives)

The first derivative is applying the definition of derivatives on the function, and it can be expressed as

$$f'(x), \quad y', \quad \frac{d}{dx}f(x), \quad \frac{dy}{dx}$$

We can keep applying the differentiation process to functions that are themselves derivatives. The derivative of $f'(x)$ with respect to x, would then be

$$f''(x) = \lim_{h \to 0} \frac{f'(x+h) - f'(x)}{h}$$

and we can therefore call it the **Second derivative:**

$$f''(x), \quad y'', \quad \frac{d^2}{dx^2}f(x), \quad \frac{d^2y}{dx^2}$$

Similarly, the derivative of $f''(x)$ would be called the third derivative and is denoted $f'''(x)$. And by extension, the **nth derivative** is expressed as $\frac{d^n}{dx^n}f(x)$, $\frac{d^n y}{dx^n}$.

Example 4.1.

Succession of derivatives

$$
\begin{aligned}
f(x) &= x^3 \\
f'(x) &= 3x^2 \\
f''(x) &= 6x \\
f'''(x) &= 6 \\
f''''(x) &= 0
\end{aligned}
$$

Earlier, in Section -Section 4.1, we said that if a function differentiable at a given point, then it must be continuous. Further, if $f'(x)$ is itself continuous, then $f(x)$ is called continuously differentiable. All of this matters because many of our findings about optimization (Section @ref(optim)) rely on differentiation, and so we want our function to be differentiable in as many layers. A function that is continuously differentiable infinitly is called "smooth". Some examples: $f(x) = x^2$, $f(x) = e^x$.

4.3 Composite Functions and the Chain Rule

As useful as the above rules are, many functions you'll see won't fit neatly in each case immediately. Instead, they will be functions of functions. For example, the difference between $x^2 + 1^2$ and $(x^2 + 1)^2$ may look trivial, but the sum rule can be easily applied to the former, while it's actually not obvious what do with the latter.

Composite functions are formed by substituting one function into another and are denoted by

$$(f \circ g)(x) = f[g(x)].$$

To form $f[g(x)]$, the range of g must be contained (at least in part) within the domain of f. The domain of $f \circ g$ consists of all the points in the domain of g for which $g(x)$ is in the domain of f.

Example 4.2.

Composite functions

Let $f(x) = \log x$ for $0 < x < \infty$ and $g(x) = x^2$ for $-\infty < x < \infty$.

Then

$$(f \circ g)(x) = \log x^2, -\infty < x < \infty - \{0\}$$

Also

$$(g \circ f)(x) = [\log x]^2, 0 < x < \infty$$

Notice that $f \circ g$ and $g \circ f$ are not the same functions.

With the notation of composite functions in place, now we can introduce a helpful additional rule that will deal with a derivative of composite functions as a chain of concentric derivatives.

Chain Rule:

Let $y = (f \circ g)(x) = f[g(x)]$. The derivative of y with respect to x is

$$\frac{d}{dx}\{f[g(x)]\} = f'[g(x)]g'(x)$$

We can read this as: "the derivative of the composite function y is the derivative of f evaluated at $g(x)$, times the derivative of g."

The chain rule can be thought of as the derivative of the "outside" times the derivative of the "inside", remembering that the derivative of the outside function is evaluated at the value of the inside function.

- The chain rule can also be written as

$$\frac{dy}{dx} = \frac{dy}{dg(x)}\frac{dg(x)}{dx}$$

  This expression does not imply that the $dg(x)$'s cancel out, as in fractions. They are part of the derivative notation and you can't separate them out or cancel them.)

Example 4.3.

Composite Exponent

Find $f'(x)$ for $f(x) = (3x^2 + 5x - 7)^6$.

The direct use of a chain rule is when the exponent of is itself a function, so the power rule could not have applied generaly:

Generalized Power Rule:

If $f(x) = [g(x)]^p$ for any rational number p,

$$f'(x) = p[g(x)]^{p-1}g'(x)$$

4.4 Derivatives of natural logs and the exponent

Natural logs and exponents (they are inverses of each other; see Section @ref(logexponents)) crop up everywhere in statistics. Their derivative is a special case from the above, but quite elegant.

Theorem 4.1.

Derivative of Exponents/Logs

The functions e^x and the natural logarithm $\log(x)$ are continuous and differentiable in their domains, and their first derivative is

$$(e^x)' = e^x$$

$$\log(x)' = \frac{1}{x}$$

Also, when these are composite functions, it follows by the generalized power rule that

$$\left(e^{g(x)}\right)' = e^{g(x)} \cdot g'(x)$$

$$(\log g(x))' = \frac{g'(x)}{g(x)}, \quad \textit{if} \quad g(x) > 0$$

Derivatives of natural exponential function (e)

To repeat the main rule in Theorem @ref(thm:derivexplog), the intuition is that

1. Derivative of e^x is itself: $\frac{d}{dx}e^x = e^x$ (See Figure 4.2)
2. Same thing if there were a constant in front: $\frac{d}{dx}\alpha e^x = \alpha e^x$
3. Same thing no matter how many derivatives there are in front: $\frac{d^n}{dx^n}\alpha e^x = \alpha e^x$
4. Chain Rule: When the exponent is a function of x, remember to take derivative of that function and add to product. $\frac{d}{dx}e^{g(x)} = e^{g(x)}g'(x)$

Example 4.4.

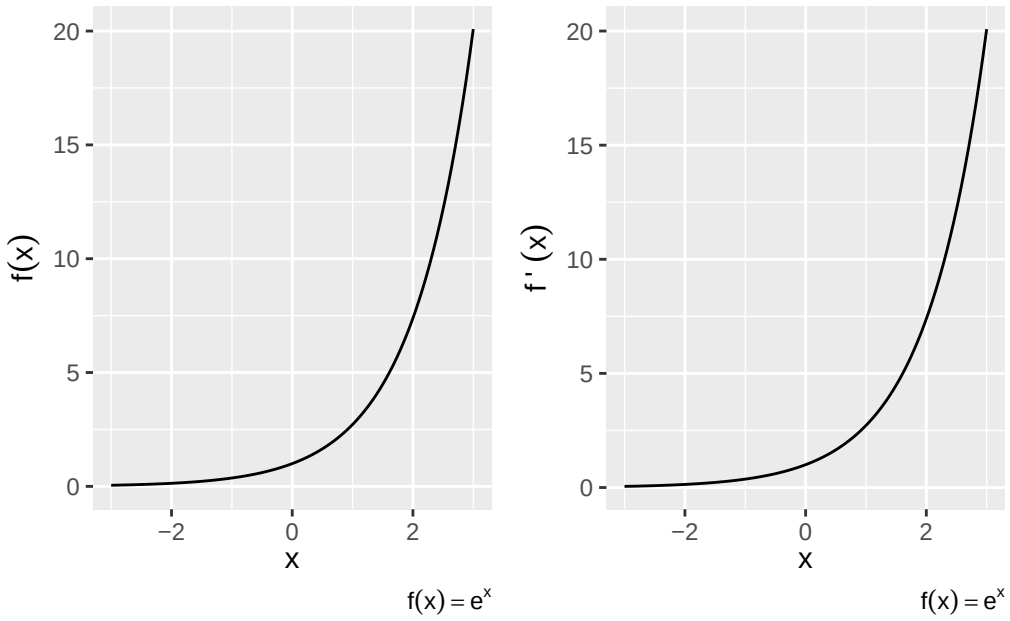


Figure 4.2: Derivative of the Exponential Function

Derivatives of exponents

Find the derivative for the following.

1. $f(x) = e^{-3x}$
2. $f(x) = e^{x^2}$
3. $f(x) = (x-1)e^x$

Derivatives of logarithms

The natural log is the mirror image of the natural exponent and has mirroring properties, again, to repeat the theorem,

1. log prime x is one over x: $\frac{d}{dx}\log x = \frac{1}{x}$ (Figure 4.3)
2. Exponents become multiplicative constants: $\frac{d}{dx}\log x^k = \frac{d}{dx}k\log x = \frac{k}{x}$
3. Chain rule again: $\frac{d}{dx}\log u(x) = \frac{u'(x)}{u(x)}$
4. For any positive base b, $\frac{d}{dx}b^x = (\log b)\,(b^x)$.

Example 4.5.

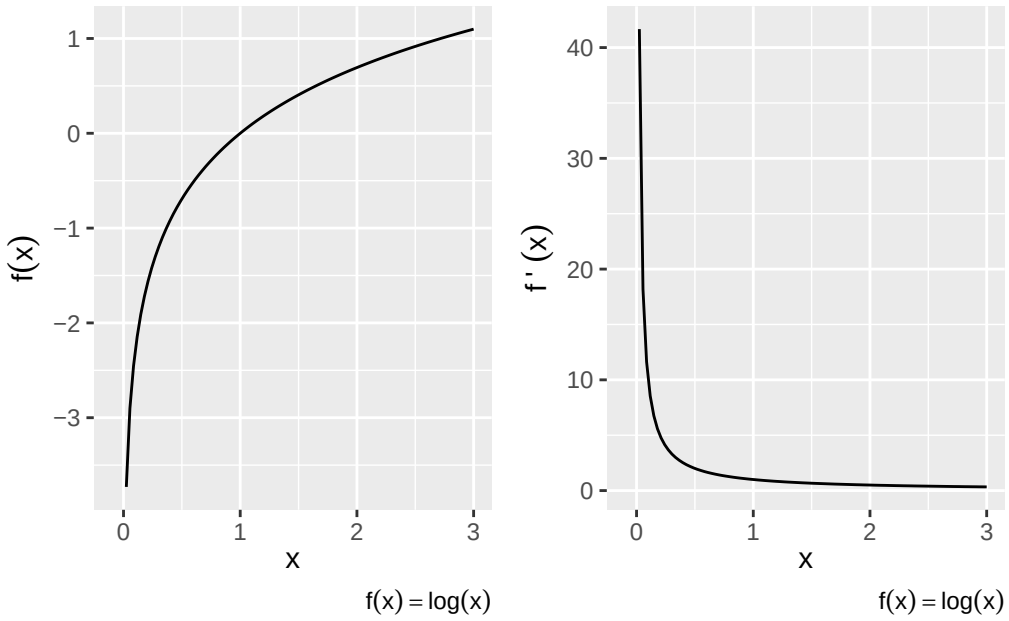


Figure 4.3: Derivative of the Natural Log

Derivatives of logs

Find dy/dx for the following.

1. $f(x) = \log(x^2 + 9)$
2. $f(x) = \log(\log x)$
3. $f(x) = (\log x)^2$
4. $f(x) = \log e^x$

Outline of Proof

We actually show the derivative of the log first, and then the derivative of the exponential naturally follows.

The general derivative of the log at any base a is solvable by the definition of derivatives.

$$(\log_a x)' = \lim_{h \to 0} \frac{1}{h} \log_a \left(1 + \frac{h}{x}\right)$$

Re-express $g = \frac{h}{x}$ and get

$$\begin{aligned}(\log_a x)' &= \frac{1}{x} \lim_{g \to 0} \log_a (1+g)^{\frac{1}{g}} \\ &= \frac{1}{x} \log_a e\end{aligned}$$

By definition of e. As a special case, when $a = e$, then $(\log x)' = \frac{1}{x}$.

Now let's think about the inverse, taking the derivative of $y = a^x$.

$$\begin{aligned} y &= a^x \\ \Rightarrow \log y &= x \log a \\ \Rightarrow \frac{y'}{y} &= \log a \\ \Rightarrow y' &= y \log a \end{aligned}$$

Then in the special case where $a = e$,

$$(e^x)' = (e^x)$$

4.5 Partial Derivatives

What happens when there's more than variable that is changing?

> If you can do ordinary derivatives, you can do partial derivatives: just hold all the other input variables constant except for the one you're differentiating with respect to. (Joe Blitzstein's Math Notes)

Suppose we have a function f now of two (or more) variables and we want to determine the rate of change relative to one of the variables. To do so, we would find its partial derivative, which is defined similar to the derivative of a function of one variable.

Partial Derivative: Let f be a function of the variables $(x_1, \dots, x_n)$. The partial derivative of f with respect to x_i is

$$\frac{\partial f}{\partial x_i}(x_1, \dots, x_n) = \lim_{h \to 0} \frac{f(x_1, \dots, x_i + h, \dots, x_n) - f(x_1, \dots, x_i, \dots, x_n)}{h}$$

Only the ith variable changes — the others are treated as constants.

We can take higher-order partial derivatives, like we did with functions of a single variable, except now the higher-order partials can be with respect to multiple variables.

Example 4.6.

Partial derivatives

Notice that you can take partials with regard to different variables.

Suppose $f(x, y) = x^2 + y^2$. Then

$$\frac{\partial f}{\partial x}(x, y) =$$
$$\frac{\partial f}{\partial y}(x, y) =$$
$$\frac{\partial^2 f}{\partial x^2}(x, y) =$$
$$\frac{\partial^2 f}{\partial x \partial y}(x, y) =$$

Exercise 4.2.

Partial derivatives

Let $f(x, y) = x^3y^4 + e^x - \log y$. What are the following partial derivatives?

$$\frac{\partial f}{\partial x}(x, y) =$$
$$\frac{\partial f}{\partial y}(x, y) =$$
$$\frac{\partial^2 f}{\partial x^2}(x, y) =$$
$$\frac{\partial^2 f}{\partial x \partial y}(x, y) =$$

4.6 Taylor Series Approximation

A common form of approximation used in statistics involves derivatives. A Taylor series is a way to represent common functions as infinite series (a sum of infinite elements) of the function's derivatives at some point a.

For example, Taylor series are very helpful in representing nonlinear (read: difficult) functions as linear (read: manageable) functions. One can thus **approximate** functions by using lower-order, finite series known as **Taylor polynomials**. If $a = 0$, the series is called a Maclaurin series.

Specifically, a Taylor series of a real or complex function $f(x)$ that is infinitely differentiable in the neighborhood of point a is:

$$\begin{aligned} f(x) &= f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots \\ &= \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n \end{aligned}$$

Taylor Approximation: We can often approximate the curvature of a function $f(x)$ at point a using a 2nd order Taylor polynomial around point a:

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + R_2$$

R_2 is the remainder (R for remainder, 2 for the fact that we took two derivatives) and often treated as negligible, giving us:

$$f(x) \approx f(a) + f'(a)(x-a) + \frac{f''(a)}{2}(x-a)^2$$

The more derivatives that are added, the smaller the remainder R and the more accurate the approximation. Proofs involving limits guarantee that the remainder converges to 0 as the order of derivation increases.

4.7 The Indefinite Integration

So far, we've been interested in finding the derivative $f = F'$ of a function F. However, sometimes we're interested in exactly the reverse: finding the function F for which f is its derivative. We refer to F as the antiderivative of f.

Definition 4.2.

Antiderivative

The antiverivative of a function $f(x)$ is a differentiable function F whose derivative is f.

$$F' = f.$$

Another way to describe is through the inverse formula. Let DF be the derivative of F. And let $DF(x)$ be the derivative of F evaluated at x. Then the antiderivative is denoted by D^{-1} (i.e., the inverse derivative). If $DF = f$, then $F = D^{-1}f$.

This definition bolsters the main takeaway about integrals and derivatives: They are inverses of each other.

Exercise 4.3.

Antiderivative

Find the antiderivative of the following:

1. $f(x) = \frac{1}{x^2}$
2. $f(x) = 3e^{3x}$

We know from derivatives how to manipulate F to get f. But how do you express the procedure to manipulate f to get F? For that, we need a new symbol, which we will call indefinite integration.

:::{#def-indefint}

4.8 Indefinite Integral

The indefinite integral of $f(x)$ is written

$$\int f(x)dx$$

and is equal to the antiderivative of f.

Example 4.7.

Graphing

Draw the function $f(x)$ and its indefinite integral, $\int f(x)dx$

$$f(x) = (x^2 - 4)$$

Solution. The Indefinite Integral of the function $f(x) = (x^2 - 4)$ can, for example, be $F(x) = \frac{1}{3}x^3 - 4x$. But it can also be $F(x) = \frac{1}{3}x^3 - 4x + 1$, because the constant 1 disappears when taking the derivative.

Some of these functions are plotted in the bottom panel of Figure 4.4 as dotted lines.

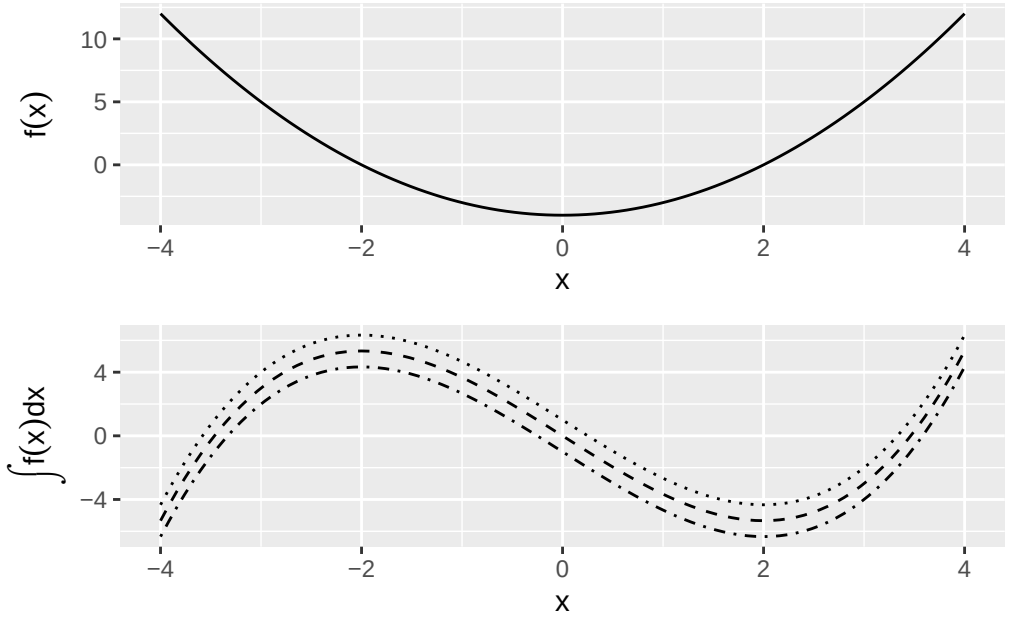


Figure 4.4: The Many Indefinite Integrals of a Function

Notice from these examples that while there is only a single derivative for any function, there are multiple antiderivatives: one for any arbitrary constant c. c just shifts the curve up or down on the y-axis. If more information is present about the antiderivative — e.g., that it passes through a particular point — then we can solve for a specific value of c.

Common Rules of Integration

Some common rules of integrals follow by virtue of being the inverse of a derivative.

1. Constants are allowed to slip out: $\int af(x)dx = a\int f(x)dx$
2. Integration of the sum is sum of integrations: $\int[f(x)+g(x)]dx = \int f(x)dx + \int g(x)dx$
3. Reverse Power-rule: $\int x^n dx = \frac{1}{n+1}x^{n+1} + c$
4. Exponents are still exponents: $\int e^x dx = e^x + c$
5. Recall the derivative of $\log(x)$ is one over x, and so: $\int \frac{1}{x}dx = \log x + c$
6. Reverse chain-rule: $\int e^{f(x)}f'(x)dx = e^{f(x)} + c$
7. More generally: $\int[f(x)]^n f'(x)dx = \frac{1}{n+1}[f(x)]^{n+1} + c$
8. Remember the derivative of a log of a function: $\int \frac{f'(x)}{f(x)}dx = \log f(x) + c$

Example 4.8.

Common Integration

Simplify the following indefinite integrals:

- (3x^2 dx)
- ((2x+1)dx)
- (e^x e{e}x} dx)

4.9 The Definite Integral: The Area under the Curve

If there is a indefinite integral, there *must* be a definite integral. Indeed there is, but the notion of definite integrals comes from a different objective: finding the are a under a function. We will find, perhaps remarkably, that the formula we find to get the sum turns out to be expressible by the anti-derivative.

Suppose we want to determine the area $A(R)$ of a region R defined by a curve $f(x)$ and some interval $a \leq x \leq b$.

One way to calculate the area would be to divide the interval $a \leq x \leq b$ into n subintervals of length Δx and then approximate the region with a series of rectangles, where the base of each rectangle is Δx and the height is $f(x)$ at the midpoint of that interval. $A(R)$ would then be approximated by the area of the union of the rectangles, which is given by

$$S(f, \Delta x) = \sum_{i=1}^{n} f(x_i)\Delta x$$

and is called a **Riemann sum**.

As we decrease the size of the subintervals Δx, making the rectangles "thinner," we would expect our approximation of the area of the region to become closer to the true area. This allows us to express the area as a limit of a series:

$$A(R) = \lim_{\Delta x \to 0} \sum_{i=1}^{n} f(x_i)\Delta x$$

Figure 4.5 shows that illustration. The curve depicted is $f(x) = -15(x-5) + (x-5)^3 + 50$. We want approximate the area under the curve between the x values of 0 and 10. We can do

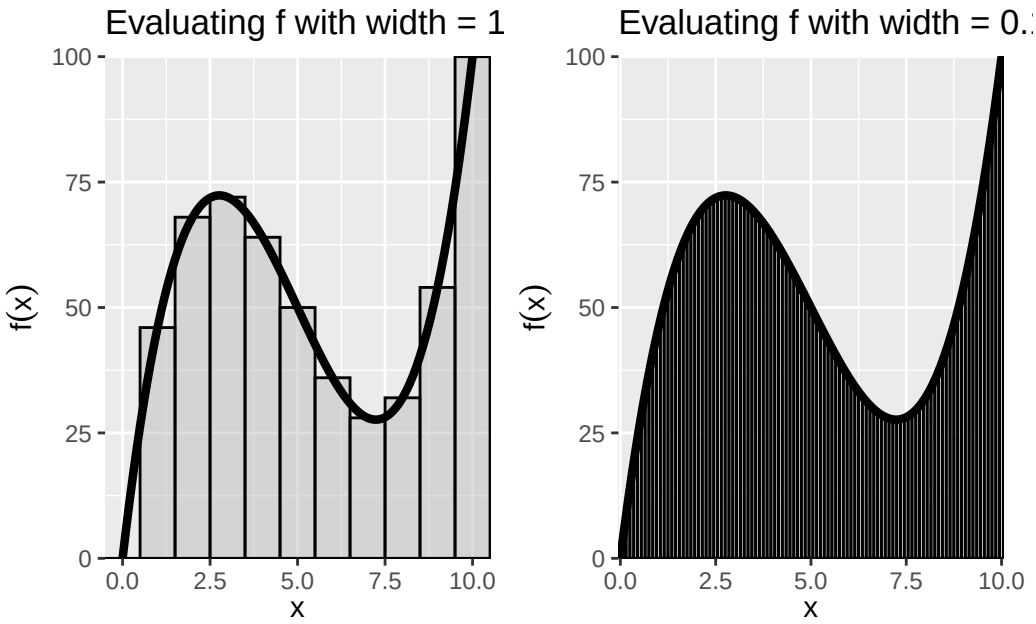


Figure 4.5: The Riemann Integral as a Sum of Evaluations

this in blocks of arbitrary width, where the sum of rectangles (the area of which is width times $f(x)$ evaluated at the midpoint of the bar) shows the Riemann Sum. As the width of the bars Δx becomes smaller, the better the estimate of $A(R)$.

This is how we define the "Definite" Integral:

Definition 4.3.

The Definite Integral (Riemann)

If for a given function f the Riemann sum approaches a limit as $\Delta x \to 0$, then that limit is called the Riemann integral of f from a to b. We express this with the $\int$, symbol, and write

$$\int_a^b f(x)dx = \lim_{\Delta x \to 0} \sum_{i=1}^{n} f(x_i)\Delta x$$

The most straightforward of a definite integral is the definite integral. That is, we read

$$\int_a^b f(x)dx$$

as the definite integral of f from a to b and we defined as the area under the "curve" $f(x)$ from point $x = a$ to $x = b$.

The fundamental theorem of calculus shows us that this sum is, in fact, the antiderivative.

Theorem 4.2.

First Fundamental Theorem of Calculus

Let the function f be bounded on $[a, b]$ and continuous on (a, b). Then, suggestively, use the symbol $F(x)$ to denote the definite integral from a to x:

$$F(x) = \int_a^x f(t)dt, \quad a \leq x \leq b$$

Then $F(x)$ has a derivative at each point in (a, b) and

$$F'(x) = f(x), \quad a < x < b$$

That is, the definite integral function of f is *the one of the antiderivatives of some f.*

This is again a long way of saying that that differentiation is the inverse of integration. But now, we've covered definite integrals.

The second theorem gives us a simple way of computing a definite integral as a function of indefinite integrals.

Theorem 4.3.

Second Fundamental Theorem of Calculus

Let the function f be bounded on $[a, b]$ and continuous on (a, b). Let F be any function that is continuous on $[a, b]$ such that $F'(x) = f(x)$ on (a, b). Then

$$\int_a^b f(x)dx = F(b) - F(a)$$

So the procedure to calculate a simple definite integral $\int_a^b f(x)dx$ is then

1. Find the indefinite integral $F(x)$.
2. Evaluate $F(b) - F(a)$.

Example 4.9.

Definite Integral of a monomial

Solve $\int_1^3 3x^2 dx$.

Let $f(x) = 3x^2$.

Exercise 4.4.

Indefinite integrals

What is the value of $\int_{-2}^{2} e^x e^{e^x} dx$?

Common Rules for Definite Integrals

The area-interpretation of the definite integral provides some rules for simplification.

1. There is no area below a point:
$$\int_a^a f(x)dx = 0$$
2. Reversing the limits changes the sign of the integral:
$$\int_a^b f(x)dx = -\int_b^a f(x)dx$$
3. Sums can be separated into their own integrals:
$$\int_a^b [\alpha f(x) + \beta g(x)]dx = \alpha \int_a^b f(x)dx + \beta \int_a^b g(x)dx$$
4. Areas can be combined as long as limits are linked:
$$\int_a^b f(x)dx + \int_b^c f(x)dx = \int_a^c f(x)dx$$

Exercise 4.5.

Definite integrals

Simplify the following definite intergrals.

1. $\int_1^1 3x^2 dx =$
2. $\int_0^4 (2x+1)dx =$
3. $\int_{-2}^0 e^x e^{e^x} dx + \int_0^2 e^x e^{e^x} dx =$

4.10 Integration by Substitution

From the second fundamental theorem of calculus, we now that a quick way to get a definite integral is to first find the indefinite integral, and then just plug in the bounds.

Sometimes the integrand (the thing that we are trying to take an integral of) doesn't appear integrable using common rules and antiderivatives. A method one might try is **integration by substitution**, which is related to the Chain Rule.

Suppose we want to find the indefinite integral

$$\int g(x)dx$$

but $g(x)$ is complex and none of the formulas we have seen so far seem to apply immediately. The trick is to come up with a *new* function $u(x)$ such that

$$g(x) = f[u(x)]u'(x).$$

Why does an introduction of yet another function end of simplifying things? Let's refer to the antiderivative of f as F. Then the chain rule tells us that

$$\frac{d}{dx}F[u(x)] = f[u(x)]u'(x)$$

. So, $F[u(x)]$ is the antiderivative of g. We can then write

$$\int g(x)dx = \int f[u(x)]u'(x)dx = \int \frac{d}{dx}F[u(x)]dx = F[u(x)] + c$$

To summarize, the procedure to determine the indefinite integral $\int g(x)dx$ by the method of substitution:

1. Identify some part of $g(x)$ that might be simplified by substituting in a single variable u (which will then be a function of x).
2. Determine if $g(x)dx$ can be reformulated in terms of u and du.
3. Solve the indefinite integral.
4. Substitute back in for x

Substitution can also be used to calculate a definite integral. Using the same procedure as above,

$$\int_a^b g(x)dx = \int_c^d f(u)du = F(d) - F(c)$$

where $c = u(a)$ and $d = u(b)$.

Example 4.10. Integration by Substitution I

Solve the indefinite integral

$$\int x^2\sqrt{x+1}dx.$$

For the above problem, we could have also used the substitution $u = \sqrt{x+1}$. Then $x = u^2 - 1$ and $dx = 2udu$. Substituting these in, we get

$$\int x^2\sqrt{x+1}dx = \int (u^2-1)^2 u 2udu$$

which when expanded is again a polynomial and gives the same result as above.

Another case in which integration by substitution is is useful is with a fraction.

Example 4.11.

Integration by Substitutiton II

Simplify

$$\int_0^1 \frac{5e^{2x}}{(1+e^{2x})^{1/3}}dx.$$

4.11 Integration by Parts

Another useful integration technique is **integration by parts**, which is related to the Product Rule of differentiation. The product rule states that

$$\frac{d}{dx}(uv) = u\frac{dv}{dx} + v\frac{du}{dx}$$

Integrating this and rearranging, we get

$$\int u\frac{dv}{dx}dx = uv - \int v\frac{du}{dx}dx$$

or

$$\int u(x)v'(x)dx = u(x)v(x) - \int v(x)u'(x)dx$$

More easily remembered with the mnemonic "Ultraviolet Voodoo":

$$\int udv = uv - \int vdu$$

where $du = u'(x)dx$ and $dv = v'(x)dx$.

For definite integrals, this is simply

$$\int_a^b u\frac{dv}{dx}dx = uv|_a^b - \int_a^b v\frac{du}{dx}dx$$

Our goal here is to find expressions for u and dv that, when substituted into the above equation, yield an expression that's more easily evaluated.

Example 4.12.

Integration by parts

Simplify the following integrals. These seemingly obscure forms of integrals come up often when integrating distributions.

$$\int xe^{ax}dx$$

Solution. Let $u = x$ and $\frac{dv}{dx} = e^{ax}$. Then $du = dx$ and $v = (1/a)e^{ax}$. Substituting this into the integration by parts formula, we obtain

$$\begin{aligned}\int xe^{ax}dx &= uv - \int vdu \\ &= x\left(\frac{1}{a}e^{ax}\right) - \int \frac{1}{a}e^{ax}dx \\ &= \frac{1}{a}xe^{ax} - \frac{1}{a^2}e^{ax} + c\end{aligned}$$

Exercise 4.6.

Integration by parts

1. Integrate
$$\int x^n e^{ax} dx$$
2. Integrate
$$\int x^3 e^{-x^2} dx$$

5 Optimization

To optimize, we use derivatives and calculus. Optimization is to find the maximum or minimum of a functon, and to find what value of an input gives that extremum. This has obvious uses in engineering. Many tools in the statistical toolkit use optimization. One of the most common ways of estimating a model is through "Maximum Likelihood Estimation", done via optimizing a function (the likelihood).

Optimization also comes up in Economics, Formal Theory, and Political Economy all the time. A go-to model of human behavior is that they optimize a certain utility function. Humans are not pure utility maximizers, of course, but nuanced models of optimization – for example, adding constraints and adding uncertainty – will prove to be quite useful.

Example: Meltzer-Richard

A standard backdrop in comparative political economy, the Meltzer-Richard (1981) model states that redistribution of wealth should be higher in societies where the median income is much smaller than the average income. More to the point, typically income distributions where the median is very different from the average is one of high inequality. In other words, the Meltzer-Richard model says that highly unequal economies will have more re-distribution of wealth. Why is that the case? Here is a simplified example that is not the exact model by Meltzer and Richard[1], but adapted from Persson and Tabellini[2]

We will set the following things about our model human and model democracy.

- Individuals are indexed by i, and the total population is normalized to unity ("1") without loss of generality.
- $U(\cdot)$, u for "utility", is a function that is concave and increasing, and expresses the utility gained from public goods. This tells us that its first derivative is *positive*, and its second derivative is **negative**.
- y_i is the income of person i
- W_i, w for "welfare", is the welfare of person i
- c_i, c for "consumption", is the consumption utility of person i

[1] Allan H. Meltzer and Scott F. Richard. "A Rational Theory of the Size of Government". *Journal of Political Economy* 89:5 (1981), p. 914-927

[2] Adapted from Torsten Persson and Guido Tabellini, *Political Economics: Explaining Economic Policy*. MIT Press.

Also, the government is democratically elected and sets the following redistribution output:

- τ, t for "tax", is a flat tax rate between 0 and 1 that is applied to everyone's income.
- g, "g" for "goods", is the amount of public goods that the government provides.

Suppose an individual's welfare is given by:

$$W_i = c_i + U(g)$$

The consumption good is the person's post-tax income.

$$c_i = (1 - \tau)y_i$$

Income varies by person (In the next section we will cover probability, by then we will know that we can express this by saying that y is a random variable with the cumulative distribution function F, i.e. $y \sim F$.). Every distribution has a mean and an median.

- $E(y)$ is the average income of the society.
- $\text{med}(y)$ is the **median income** of the society.

What will happen in this economy? What will the tax rate be set too? How much public goods will be provided?

We've skipped ahead of some formal theory results of democracy, but hopefully these are conceptually intuitive. First, if a democracy is competitive, there is no slack in the government's goods, and all tax revenue becomes a public good. So we can go ahead and set the constraint:

$$g = \sum_i \tau y_i P(y_i) = \tau E(y)$$

We can do this trick because of the "normalizes to unity" setting, but this is a general property of the average.

Now given this constraint we can re-write an individual's welfare as

$$\begin{aligned} W_i &= \left(1 - \frac{g}{E(y)}\right) y_i + U(g) \\ &= (E(y) - g)\,\frac{1}{E(y)} y_i + U(g) \\ &= (E(y) - g)\,\frac{y_i}{E(y)} + U(g) \end{aligned}$$

When is the individual's welfare maximized, **as a function of the public good**?

$$\frac{d}{dg}W_i = -\frac{y_i}{E(y)} + \frac{d}{dg}U(g)$$

$\frac{d}{dg}W_i = 0$ when $\frac{d}{dg}U(g) = \frac{y_i}{E(y)}$, and so after expressing the derivative as $U_g = \frac{d}{dg}U(g)$ for simplicity,

$$g_i^\star = {U_g}^{-1}\left(\frac{y_i}{E(y)}\right)$$

Now recall that because we assumed concavity, U_g is a negative sloping function whose value is positive. It can be shown that the inverse of such a function is also decreasing. Thus an individual's preferred level of government is determined by a single continuum, the person's income divided by the average income, and the function is **decreasing** in y_i. This is consistent with our intuition that richer people prefer less redistribution.

That was the amount for any given person. The government has to set one value of g, however. So what will that be? Now we will use another result, the median voter theorem. This says that under certain general electoral conditions (single-peaked preferences, two parties, majority rule), the policy winner will be that preferred by the median person in the population. Because the only thing that determines a person's preferred level of government is $y_i/E(y)$, we can presume that the median voter, whose income is med(y) will prevail in their preferred choice of government. Therefore, we wil see

$$g^\star = {U_g}^{-1}\left(\frac{\text{med}(y)}{E(y)}\right)$$

What does this say about the level of redistribution we observe in an economy? The higher the average income is than the median income, which often (but not always) means *more* inequality, there should be *more* redistribution.

5.1 Maxima and Minima

The first derivative, $f'(x)$, quantifies the slope of a function. Therefore, it can be used to check whether the function $f(x)$ at the point x is increasing or decreasing at x.

1. **Increasing:** $f'(x) > 0$
2. **Decreasing:** $f'(x) < 0$
3. **Neither increasing nor decreasing**: $f'(x) = 0$ i.e. a maximum, minimum, or saddle point

So for example, $f(x) = x^2 + 2$ and $f'(x) = 2x$

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

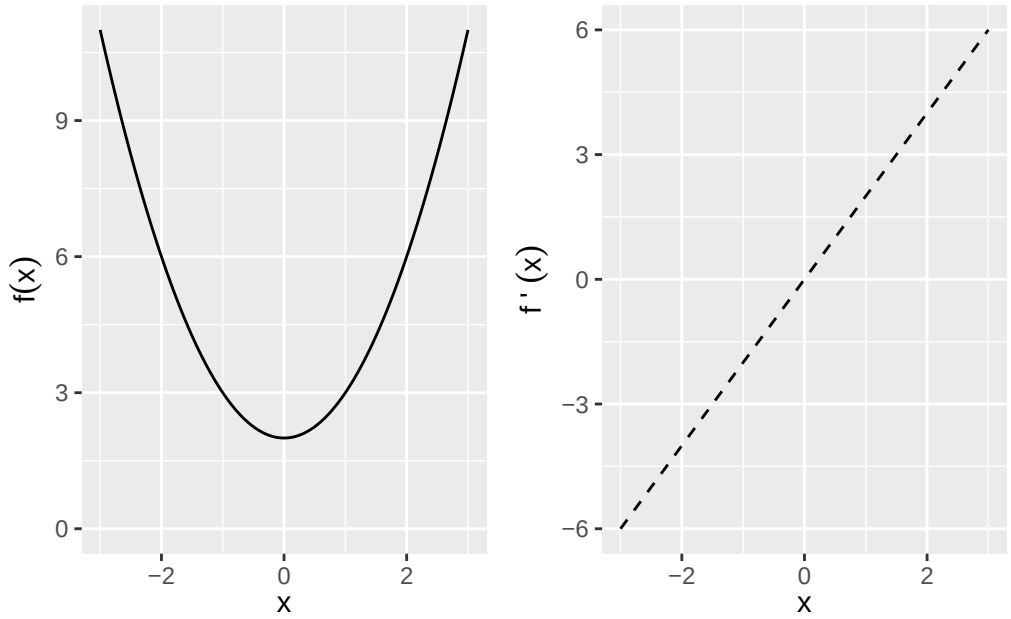


Figure 5.1: Maxima and Minima

Exercise 5.1.

Plotting a maximum and minimum

Plot $f(x) = x^3 + x^2 + 2$, plot its derivative, and identifiy where the derivative is zero. Is there a maximum or minimum?

The second derivative $f''(x)$ identifies whether the function $f(x)$ at the point x is

1. Concave down: $f''(x) < 0$
2. Concave up (convex): $f''(x) > 0$

Maximum (Minimum): x_0 is a **local maximum (minimum)** if $f(x_0) > f(x)$ ($f(x_0) < f(x)$) for all x within some open interval containing x_0. x_0 is a **global maximum (minimum)** if $f(x_0) > f(x)$ ($f(x_0) < f(x)$) for all x in the domain of f.

Given the function f defined over domain D, all of the following are defined as **critical points**:

1. Any interior point of D where $f'(x) = 0$.
2. Any interior point of D where $f'(x)$ does not exist.
3. Any endpoint that is in D.

The maxima and minima will be a subset of the critical points.

Second Derivative Test of Maxima/Minima: We can use the second derivative to tell us whether a point is a maximum or minimum of $f(x)$.

1. Local Maximum: $f'(x) = 0$ and $f''(x) < 0$
2. Local Minimum: $f'(x) = 0$ and $f''(x) > 0$
3. Need more info: $f'(x) = 0$ and $f''(x) = 0$

Global Maxima and Minima Sometimes no global max or min exists — e.g., $f(x)$ not bounded above or below. However, there are three situations where we can fairly easily identify global max or min.

1. **Functions with only one critical point.** If x_0 is a local max or min of f and it is the only critical point, then it is the global max or min.
2. **Globally concave up or concave down functions.** If $f''(x)$ is never zero, then there is at most one critical point. That critical point is a global maximum if $f'' < 0$ and a global minimum if $f'' > 0$.

3. **Functions over closed and bounded intervals** must have both a global maximum and a global minimum.

Example 5.1.

Maxima and Minima by drawing

Find any critical points and identify whether they are a max, min, or saddle point:

1. $f(x) = x^2 + 2$
2. $f(x) = x^3 + 2$
3. $f(x) = |x^2 - 1|$, $x \in [-2, 2]$

5.2 Concavity of a Function

Concavity helps identify the curvature of a function, $f(x)$, in 2 dimensional space.

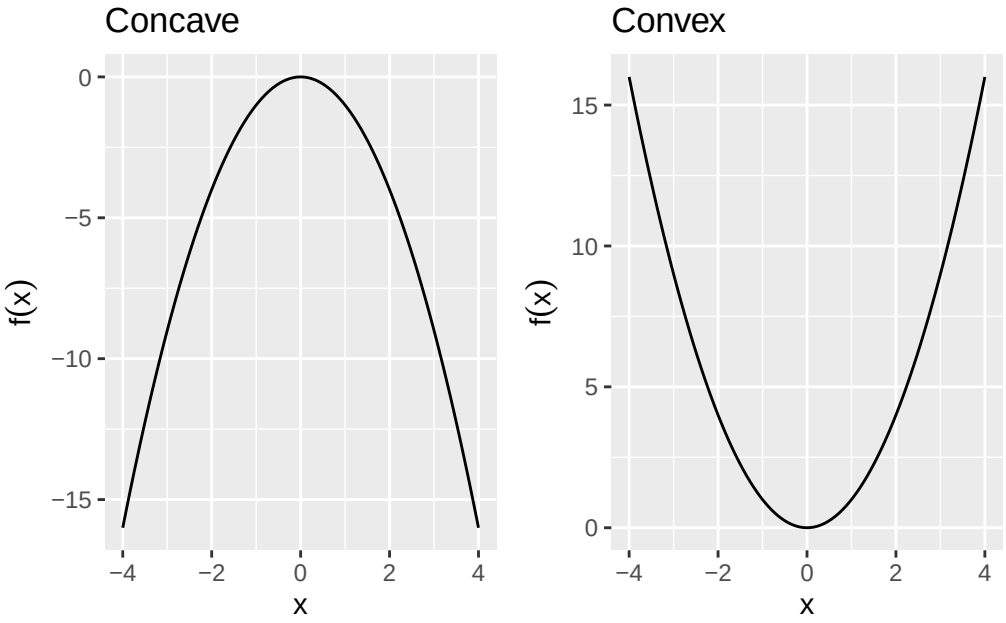
Definition 5.1.

Concave Function

A function f is strictly concave over the set S if $\forall x_1, x_2 \in S$ and $\forall a \in (0,1)$,

$$f(ax_1 + (1-a)x_2) > af(x_1) + (1-a)f(x_2)$$

Any line connecting two points on a concave function will lie *below* the function.



Definition 5.2.

Convex Function

Convex: A function f is strictly convex over the set S if $\forall x_1, x_2 \in S$ and $\forall a \in (0, 1)$,

$$f(ax_1 + (1-a)x_2) < af(x_1) + (1-a)f(x_2)$$

Any line connecting two points on a convex function will lie above the function.

Sometimes, concavity and convexity are strict of a requirement. For most purposes of getting solutions, what we call quasi-concavity is enough.

Definition 5.3.

Quasiconcave Function

A function f is quasiconcave over the set S if $\forall x_1, x_2 \in S$ and $\forall a \in (0, 1)$,

$$f(ax_1 + (1 - a)x_2) \geq \min(f(x_1), f(x_2))$$

No matter what two points you select, the *lowest* valued point will always be an end point.

Definition 5.4.

Quasiconvex Function

A function f is quasiconvex over the set S if $\forall x_1, x_2 \in S$ and $\forall a \in (0, 1)$,

$$f(ax_1 + (1-a)x_2) \leq \max(f(x_1), f(x_2))$$

No matter what two points you select, the *highest* valued point will always be an end point.

Second Derivative Test of Concavity: The second derivative can be used to understand concavity.

If

$$\begin{aligned} f''(x) < 0 &\Rightarrow \text{Concave} \\ f''(x) > 0 &\Rightarrow \text{Convex} \end{aligned}$$

Quadratic Forms

Quadratic forms is shorthand for a way to summarize a function. This is important for finding concavity because

1. Approximates local curvature around a point — e.g., used to identify max vs min vs saddle point.
2. They are simple to express even in n dimensions:
3. Have a matrix representation.

Quadratic Form: A polynomial where each term is a monomial of degree 2 in any number of variables:

$$\begin{aligned} \text{One variable: } & Q(x_1) = a_{11}x_1^2 \\ \text{Two variables: } & Q(x_1, x_2) = a_{11}x_1^2 + a_{12}x_1x_2 + a_{22}x_2^2 \\ \text{N variables: } & Q(x_1, \cdots, x_n) = \sum_{i \leq j} a_{ij}x_ix_j \end{aligned}$$

which can be written in matrix terms:

One variable

$$Q(\mathbf{x}) = x_1^\top a_{11} x_1$$

N variables:

$$\begin{aligned} Q(\mathbf{x}) &= \begin{pmatrix} x_1 & x_2 & \cdots & x_n \end{pmatrix} \begin{pmatrix} a_{11} & \frac{1}{2}a_{12} & \cdots & \frac{1}{2}a_{1n} \\ \frac{1}{2}a_{12} & a_{22} & \cdots & \frac{1}{2}a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{2}a_{1n} & \frac{1}{2}a_{2n} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \\ &= \mathbf{x}^\top \mathbf{A} \mathbf{x} \end{aligned}$$

For example, the Quadratic on $\mathbf{R}^2$:

$$\begin{aligned} Q(x_1, x_2) &= \begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} a_{11} & \frac{1}{2}a_{12} \\ \frac{1}{2}a_{12} & a_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \\ &= a_{11}x_1^2 + a_{12}x_1x_2 + a_{22}x_2^2 \end{aligned}$$

Definiteness of Quadratic Forms

When the function $f(\mathbf{x})$ has more than two inputs, determining whether it has a maxima and minima (remember, functions may have many inputs but they have only one output) is a bit more tedious. Definiteness helps identify the curvature of a function, $Q(\mathbf{x})$, in n dimensional space.

Definiteness: By definition, a quadratic form always takes on the value of zero when $x = 0$, $Q(\mathbf{x}) = 0$ at $\mathbf{x} = 0$. The definiteness of the matrix $\mathbf{A}$ is determined by whether the quadratic form $Q(\mathbf{x}) = \mathbf{x}^\top \mathbf{A} \mathbf{x}$ is greater than zero, less than zero, or sometimes both over all $\mathbf{x} \neq 0$.

5.3 FOC and SOC

We can see from a graphical representation that if a point is a local maxima or minima, it must meet certain conditions regarding its derivative. These are so commonly used that we refer these to "First Order Conditions" (FOCs) and "Second Order Conditions" (SOCs) in the economic tradition.

First Order Conditions

When we examined functions of one variable x, we found critical points by taking the first derivative, setting it to zero, and solving for x. For functions of n variables, the critical points are found in much the same way, except now we set the partial derivatives equal to zero. Note: We will only consider critical points on the interior of a function's domain.

In a derivative, we only took the derivative with respect to one variable at a time. When we take the derivative separately with respect to all variables in the elements of $\mathbf{x}$ and then express the result as a vector, we use the term Gradient and Hessian.

Definition 5.5.

Gradient

Given a function $f(\mathbf{x})$ in n variables, the gradient $\nabla f(\mathbf{x})$ (the greek letter nabla) is a column vector, where the ith element is the partial derivative of $f(\mathbf{x})$ with respect to x_i:

$$\nabla f(\mathbf{x}) = \begin{pmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \frac{\partial f(\mathbf{x})}{\partial x_2} \\ \vdots \\ \frac{\partial f(\mathbf{x})}{\partial x_n} \end{pmatrix}$$

Before we know whether a point is a maxima or minima, if it meets the FOC it is a "Critical Point".

Definition 5.6.

Critical Point

$\mathbf{x}^*$ is a critical point if and only if $\nabla f(\mathbf{x}^*) = 0$. If the partial derivative of f(x) with respect to x^* is 0, then $\mathbf{x}^*$ is a critical point. To solve for $\mathbf{x}^*$, find the gradient, set each element equal to 0, and solve the system of equations.

$$\mathbf{x}^* = \begin{pmatrix} x_1^* \\ x_2^* \\ \vdots \\ x_n^* \end{pmatrix}$$

Example 5.2. Example: Given a function $f(\mathbf{x}) = (x_1 - 1)^2 + x_2^2 + 1$, find the (1) Gradient and (2) Critical point of $f(\mathbf{x})$.

Solution. Gradient

$$\begin{aligned} \nabla f(\mathbf{x}) &= \begin{pmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \frac{\partial f(\mathbf{x})}{\partial x_2} \end{pmatrix} \\ &= \begin{pmatrix} 2(x_1 - 1) \\ 2x_2 \end{pmatrix} \end{aligned}$$

Critical Point $\mathbf{x}^* =$

$$\begin{aligned} &\frac{\partial f(\mathbf{x})}{\partial x_1} = 2(x_1 - 1) = 0 \\ &\Rightarrow x_1^* = 1 \\ &\frac{\partial f(\mathbf{x})}{\partial x_2} = 2x_2 = 0 \\ &\Rightarrow x_2^* = 0 \end{aligned}$$

So

$$\mathbf{x}^* = (1, 0)$$

Second Order Conditions

When we found a critical point for a function of one variable, we used the second derivative as a indicator of the curvature at the point in order to determine whether the point was a min, max, or saddle (second derivative test of concavity). For functions of n variables, we use *second order partial derivatives* as an indicator of curvature.

Definition 5.7.

Hessian

Given a function $f(\mathbf{x})$ in n variables, the hessian $\mathbf{H}(\mathbf{x})$ is an $n \times n$ matrix, where the (i,j)th element is the second order partial derivative of $f(\mathbf{x})$ with respect to x_i and x_j:

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

Note that the hessian will be a symmetric matrix because $\frac{\partial f(\mathbf{x})}{\partial x_1 \partial x_2} = \frac{\partial f(\mathbf{x})}{\partial x_2 \partial x_1}$.

Also note that given that $f(\mathbf{x})$ is of quadratic form, each element of the hessian will be a constant.

These definitions will be employed when we determine the **Second Order Conditions** of a function:

Given a function $f(\mathbf{x})$ and a point $\mathbf{x}^*$ such that $\nabla f(\mathbf{x}^*) = 0$,

1. Hessian is Positive Definite $\implies$ Strict Local Min
2. Hessian is Positive Semidefinite $\forall \mathbf{x} \in B(\mathbf{x}^*, \epsilon)\}$ $\implies$ Local Min
3. Hessian is Negative Definite $\implies$ Strict Local Max
4. Hessian is Negative Semidefinite $\forall \mathbf{x} \in B(\mathbf{x}^*, \epsilon)\}$ $\implies$ Local Max
5. Hessian is Indefinite $\implies$ Saddle Point

Example 5.3.

Max and min with two dimensions

We found that the only critical point of $f(\mathbf{x}) = (x_1 - 1)^2 + x_2^2 + 1$ is at $\mathbf{x}^* = (1, 0)$. Is it a min, max, or saddle point?

Solution. The Hessian is

$$\mathbf{H}(\mathbf{x}) = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$$

The Leading principal minors of the Hessian are $M_1 = 2; M_2 = 4$. Now we consider Definiteness. Since both leading principal minors are positive, the Hessian is positive definite.

Maxima, Minima, or Saddle Point? Since the Hessian is positive definite and the gradient equals 0, $x^\star = (1, 0)$ is a strict local minimum.

Note: Alternate check of definiteness. Is $\mathbf{H}(\mathbf{x}^*) \geq\leq 0 \quad \forall \quad \mathbf{x} \neq 0$

$$\begin{aligned} \mathbf{x}^\top H(\mathbf{x}^*)\mathbf{x} &= \begin{pmatrix} x_1 & x_2 \end{pmatrix} \\ &= \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \\ \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} &= 2x_1^2 + 2x_2^2 \end{aligned}$$

For any $\mathbf{x} \neq 0$, $2(x_1^2 + x_2^2) > 0$, so the Hessian is positive definite and $\mathbf{x}^*$ is a strict local minimum.

Definiteness and Concavity

Although definiteness helps us to understand the curvature of an n-dimensional function, it does not necessarily tell us whether the function is globally concave or convex.

We need to know whether a function is globally concave or convex to determine whether a critical point is a global min or max. We can use the definiteness of the Hessian to determine whether a function is globally concave or convex:

1. Hessian is Positive Semidefinite $\forall \mathbf{x}\}$ $\implies$ Globally Convex
2. Hessian is Negative Semidefinite $\forall \mathbf{x}\}$ $\implies$ Globally Concave

Notice that the definiteness conditions must be satisfied over the entire domain.

5.4 Global Maxima and Minima

Global Max/Min Conditions: Given a function $f(\mathbf{x})$ and a point $\mathbf{x}^*$ such that $\nabla f(\mathbf{x}^*) = 0$,

1. $f(\mathbf{x})$ Globally Convex $\implies$ Global Min
2. $f(\mathbf{x})$ Globally Concave $\implies$ Global Max

Note that showing that $\mathbf{H}(\mathbf{x}^*)$ is negative semidefinite is not enough to guarantee $\mathbf{x}^*$ is a local max. However, showing that $\mathbf{H}(\mathbf{x})$ is negative semidefinite for all $\mathbf{x}$ guarantees that x^* is a global max. (The same goes for positive semidefinite and minima.)\

Example: Take $f_1(x) = x^4$ and $f_2(x) = -x^4$. Both have $x = 0$ as a critical point. Unfortunately, $f_1''(0) = 0$ and $f_2''(0) = 0$, so we can't tell whether $x = 0$ is a min or max for either. However, $f_1''(x) = 12x^2$ and $f_2''(x) = -12x^2$. For all x, $f_1''(x) \geq 0$ and $f_2''(x) \leq 0$ — i.e., $f_1(x)$ is globally convex and $f_2(x)$ is globally concave. So $x = 0$ is a global min of $f_1(x)$ and a global max of $f_2(x)$.

Exercise 5.2.

Optimization

Given $f(\mathbf{x}) = x_1^3 - x_2^3 + 9x_1x_2$, find any maxima or minima.

1. First order conditions.
   a) Gradient $\nabla f(\mathbf{x}) =$

   b) Critical Points $\mathbf{x}^* =$

2. Second order conditions.
   a) Hessian $\mathbf{H}(\mathbf{x}) =$

   b) Hessian $\mathbf{H}(\mathbf{x_1^*}) =$

   c) Leading principal minors of $\mathbf{H}(\mathbf{x_1^*}) =$

d) Definiteness of $\mathbf{H}(\mathbf{x_1^*})$?

e) Maxima, Minima, or Saddle Point for $\mathbf{x_1^*}$?

f) Hessian $\mathbf{H}(\mathbf{x_2^*}) =$

g) Leading principal minors of $\mathbf{H}(\mathbf{x_2^*}) =$

h) Definiteness of $\mathbf{H}(\mathbf{x_2^*})$?

i) Maxima, Minima, or Saddle Point for $\mathbf{x_2^*}$?

3. Global concavity/convexity.
   a) Is f(x) globally concave/convex?

   b) Are any $\mathbf{x}^*$ global minima or maxima?

5.5 Constrained Optimization

We have already looked at optimizing a function in one or more dimensions over the whole domain of the function. Often, however, we want to find the maximum or minimum of a function over some restricted part of its domain.

ex: Maximizing utility subject to a budget constraint

Types of Constraints: For a function $f(x_1, \ldots, x_n)$, there are two types of constraints that can be imposed:

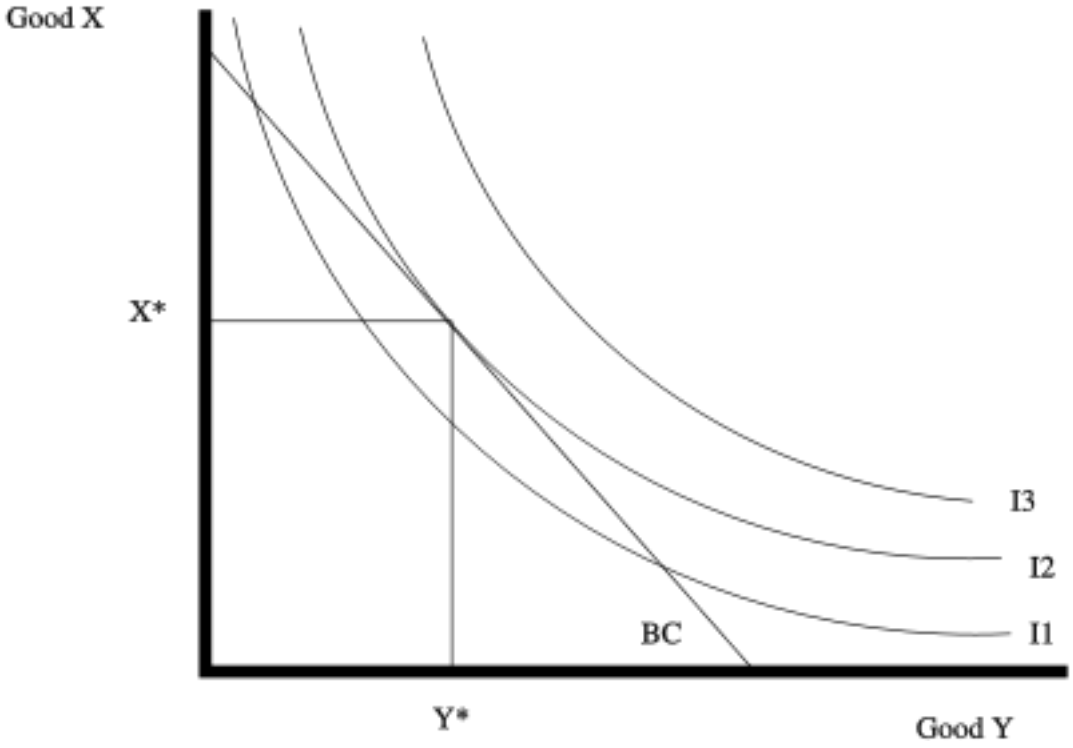


Figure 5.2: A typical Utility Function with a Budget Constraint

1. **Equality constraints:** constraints of the form $c(x_1, \ldots, x_n) = r$. Budget constraints are the classic example of equality constraints in social science.
2. **Inequality constraints:** constraints of the form $c(x_1, \ldots, x_n) \leq r$. These might arise from non-negativity constraints or other threshold effects.

In any constrained optimization problem, the constrained maximum will always be less than or equal to the unconstrained maximum. If the constrained maximum is less than the unconstrained maximum, then the constraint is binding. Essentially, this means that you can treat your constraint as an equality constraint rather than an inequality constraint.

For example, the budget constraint binds when you spend your entire budget. This generally happens because we believe that utility is strictly increasing in consumption, i.e. you always want more so you spend everything you have.

Any number of constraints can be placed on an optimization problem. When working with multiple constraints, always make sure that the set of constraints are not pathological; it must be possible for all of the constraints to be satisfied simultaneously.

Set-up for Constrained Optimization:

$$\max_{x_1, x_2} f(x_1, x_2) \text{ s.t. } c(x_1, x_2)$$

$$\min_{x_1, x_2} f(x_1, x_2) \text{ s.t. } c(x_1, x_2)$$

This tells us to maximize/minimize our function, $f(x_1, x_2)$, with respect to the choice variables, x_1, x_2, subject to the constraint.

Example:

$$\max_{x_1, x_2} f(x_1, x_2) = -(x_1^2 + 2x_2^2) \text{ s.t. } x_1 + x_2 = 4$$

It is easy to see that the *unconstrained* maximum occurs at $(x_1, x_2) = (0, 0)$, but that does not satisfy the constraint. How should we proceed?

Equality Constraints

Equality constraints are the easiest to deal with because we know that the maximum or minimum has to lie on the (intersection of the) constraint(s).

The trick is to change the problem from a constrained optimization problem in n variables to an unconstrained optimization problem in $n + k$ variables, adding *one* variable for *each* equality constraint. We do this using a lagrangian multiplier.

Lagrangian function: The Lagrangian function allows us to combine the function we want to optimize and the constraint function into a single function. Once we have this single function, we can proceed as if this were an *unconstrained* optimization problem.

For each constraint, we must include a **Lagrange multiplier** (λ_i) as an additional variable in the analysis. These terms are the link between the constraint and the Lagrangian function.

Given a *two dimensional* set-up:

$$\max_{x_1,x_2} / \min_{x_1,x_2} f(x_1, x_2) \text{ s.t. } c(x_1, x_2) = a$$

We define the Lagrangian function $L(x_1, x_2, \lambda_1)$ as follows:

$$L(x_1, x_2, \lambda_1) = f(x_1, x_2) - \lambda_1(c(x_1, x_2) - a)$$

More generally, in *n dimensions*:

$$L(x_1, \dots, x_n, \lambda_1, \dots, \lambda_k) = f(x_1, \dots, x_n) - \sum_{i=1}^{k} \lambda_i(c_i(x_1, \dots, x_n) - r_i)$$

Getting the sign right: Note that above we subtract the lagrangian term *and* we subtract the constraint constant from the constraint function. Occasionally, you may see the following alternative form of the Lagrangian, which is *equivalent*:

$$L(x_1, \dots, x_n, \lambda_1, \dots, \lambda_k) = f(x_1, \dots, x_n) + \sum_{i=1}^{k} \lambda_i(r_i - c_i(x_1, \dots, x_n))$$

Here we add the lagrangian term *and* we subtract the constraining function from the constraint constant.

Using the Lagrangian to Find the Critical Points: To find the critical points, we take the partial derivatives of lagrangian function, $L(x_1, \dots, x_n, \lambda_1, \dots, \lambda_k)$, with respect to each of its variables (all choice variables $\mathbf{x}$ *and* all lagrangian multipliers). At a critical point, each of these partial derivatives must be equal to zero, so we obtain a system of $n + k$ equations in $n + k$ unknowns:

$$\begin{aligned}
\frac{\partial L}{\partial x_1} &= \frac{\partial f}{\partial x_1} - \sum_{i=1}^{k} \lambda_i \frac{\partial c_i}{\partial x_1} = 0 \\
&\vdots = \vdots \\
\frac{\partial L}{\partial x_n} &= \frac{\partial f}{\partial x_n} - \sum_{i=1}^{k} \lambda_i \frac{\partial c_i}{\partial x_n} = 0 \\
\frac{\partial L}{\partial \lambda_1} &= c_1(x_i, \dots, x_n) - r_1 = 0 \\
&\vdots = \vdots \\
\frac{\partial L}{\partial \lambda_k} &= c_k(x_i, \dots, x_n) - r_k = 0
\end{aligned}$$

We can then solve this system of equations, because there are $n + k$ equations and $n + k$ unknowns, to calculate the critical point $(x_1^*, \dots, x_n^*, \lambda_1^*, \dots, \lambda_k^*)$.

Second-order Conditions and Unconstrained Optimization: There may be more than one critical point, i.e. we need to verify that the critical point we find is a maximum/minimum. Similar to unconstrained optimization, we can do this by checking the second-order conditions.

Example 5.4.

Constrained optimization with two goods and a budget constraint

Find the constrained optimization of

$$\max_{x_1,x_2} f(x) = -(x_1^2 + 2x_2^2) \text{ s.t. } x_1 + x_2 = 4$$

Solution. 1. Begin by writing the Lagrangian:

$$L(x_1, x_2, \lambda) = -(x_1^2 + 2x_2^2) - \lambda(x_1 + x_2 - 4)$$

2. Take the partial derivatives and set equal to zero:

$$\begin{aligned}
\frac{\partial L}{\partial x_1} &= -2x_1 - \lambda &&= 0 \\
\frac{\partial L}{\partial x_2} &= -4x_2 - \lambda &&= 0 \\
\frac{\partial L}{\partial \lambda} &= -(x_1 + x_2 - 4) &&= 0
\end{aligned}$$

3. Solve the system of equations: Using the first two partials, we see that $\lambda = -2x_1$ and $\lambda = -4x_2$ Set these equal to see that $x_1 = 2x_2$. Using the third partial and the above equality, $4 = 2x_2 + x_2$ from which we get

$$x_2^* = 4/3, x_1^* = 8/3, \lambda = -16/3$$

4. Therefore, the only critical point is $x_1^* = \frac{8}{3}$ and $x_2^* = \frac{4}{3}$
5. This gives $f(\frac{8}{3}, \frac{4}{3}) = -\frac{96}{9}$, which is less than the unconstrained optimum $f(0, 0) = 0$

Notice that when we take the partial derivative of L with respect to the Lagrangian multiplier and set it equal to 0, we return exactly our constraint! This is why signs matter.

5.6 Inequality Constraints

Inequality constraints define the boundary of a region over which we seek to optimize the function. This makes inequality constraints more challenging because we do not know if the maximum/minimum lies along one of the constraints (the constraint binds) or in the interior of the region.

We must introduce more variables in order to turn the problem into an unconstrained optimization.

Slack: For each inequality constraint $c_i(x_1, \dots, x_n) \leq a_i$, we define a slack variable s_i^2 for which the expression $c_i(x_1, \dots, x_n) \leq a_i - s_i^2$ would hold with equality. These slack variables capture how close the constraint comes to binding. We use s^2 rather than s to ensure that the slack is positive.

Slack is just a way to transform our constraints.

Given a two-dimensional set-up and these edited constraints:

$$\max_{x_1,x_2} / \min_{x_1,x_2} f(x_1, x_2) \text{ s.t. } c(x_1, x_2) \leq a_1$$

Adding in Slack:

$$\max_{x_1,x_2} / \min_{x_1,x_2} f(x_1, x_2) \text{ s.t. } c(x_1, x_2) \leq a_1 - s_1^2$$

We define the Lagrangian function $L(x_1, x_2, \lambda_1, s_1)$ as follows:

$$L(x_1, x_2, \lambda_1, s_1) = f(x_1, x_2) - \lambda_1(c(x_1, x_2) + s_1^2 - a_1)$$

More generally, in n dimensions:

$$L(x_1, \dots, x_n, \lambda_1, \dots, \lambda_k, s_1, \dots, s_k) = f(x_1, \dots, x_n) - \sum_{i=1}^{k} \lambda_i(c_i(x_1, \dots, x_n) + s_i^2 - a_i)$$

Finding the Critical Points: To find the critical points, we take the partial derivatives of the lagrangian function, $L(x_1, \dots, x_n, \lambda_1, \dots, \lambda_k, s_1, \dots, s_k)$, with respect to each of its variables (all choice variables x, all lagrangian multipliers λ, and all slack variables s). At a critical point, *each* of these partial derivatives must be equal to zero, so we obtain a system of $n + 2k$ equations in $n + 2k$ unknowns:

$$
\begin{aligned}
\frac{\partial L}{\partial x_1} &= \frac{\partial f}{\partial x_1} - \sum_{i=1}^{k} \lambda_i \frac{\partial c_i}{\partial x_1} = 0 \\
&\vdots = \vdots \\
\frac{\partial L}{\partial x_n} &= \frac{\partial f}{\partial x_n} - \sum_{i=1}^{k} \lambda_i \frac{\partial c_i}{\partial x_n} = 0 \\
\frac{\partial L}{\partial \lambda_1} &= c_1(x_i, \dots, x_n) + s_1^2 - b_1 = 0 \\
&\vdots = \vdots \\
\frac{\partial L}{\partial \lambda_k} &= c_k(x_i, \dots, x_n) + s_k^2 - b_k = 0 \\
\frac{\partial L}{\partial s_1} &= 2 s_1 \lambda_1 = 0 \\
&\vdots = \vdots \\
\frac{\partial L}{\partial s_k} &= 2 s_k \lambda_k = 0
\end{aligned}
$$

Complementary slackness conditions: The last set of first order conditions of the form $2s_i\lambda_i = 0$ (the partials taken with respect to the slack variables) are known as complementary slackness conditions. These conditions can be satisfied one of three ways:

1. $\lambda_i = 0$ and $s_i \neq 0$: This implies that the slack is positive and thus *the constraint does not bind*.
2. $\lambda_i \neq 0$ and $s_i = 0$: This implies that there is no slack in the constraint and *the constraint does bind*.
3. $\lambda_i = 0$ and $s_i = 0$: In this case, there is no slack but the *constraint binds trivially*, without changing the optimum.

Example: Find the critical points for the following constrained optimization:

$$\max_{x_1, x_2} f(x) = -(x_1^2 + 2x_2^2) \text{ s.t. } x_1 + x_2 \leq 4$$

1. Rewrite with the slack variables:

$$\max_{x_1, x_2} f(x) = -(x_1^2 + 2x_2^2) \text{ s.t. } x_1 + x_2 \leq 4 - s_1^2$$

2. Write the Lagrangian:

$$L(x_1, x_2, \lambda_1, s_1) = -(x_1^2 + 2x_2^2) - \lambda_1(x_1 + x_2 + s_1^2 - 4)$$

3. Take the partial derivatives and set equal to 0:

$$\begin{aligned}\frac{\partial L}{\partial x_1} &= -2x_1 - \lambda_1 = 0\\ \frac{\partial L}{\partial x_2} &= -4x_2 - \lambda_1 = 0\\ \frac{\partial L}{\partial \lambda_1} &= -(x_1 + x_2 + s_1^2 - 4) = 0\\ \frac{\partial L}{\partial s_1} &= -2s_1\lambda_1 = 0\end{aligned}$$

4. Consider all ways that the complementary slackness conditions are solved:

Hypothesis	s_1	λ_1	x_1	x_2	$f(x_1, x_2)$
$s_1 = 0$ $\lambda_1 = 0$	No solution				
$s_1 \neq 0$ $\lambda_1 = 0$	2	0	0	0	0
$s_1 = 0$ $\lambda_1 \neq 0$	0	$\frac{-16}{3}$	$\frac{8}{3}$	$\frac{4}{3}$	$-\frac{32}{3}$
$s_1 \neq 0$ $\lambda_1 \neq 0$	No solution				

This shows that there are two critical points: $(0, 0)$ and $(\frac{8}{3}, \frac{4}{3})$.

5. Find maximum: Looking at the values of $f(x_1, x_2)$ at the critical points, we see that $f(x_1, x_2)$ is maximized at $x_1^* = 0$ and $x_2^* = 0$.

Exercise 5.3.

Constrained optimization

Example: Find the critical points for the following constrained optimization:

$$\max_{x_1,x_2} f(x) = -(x_1^2 + 2x_2^2) \text{ s.t. } \begin{array}{l} x_1 + x_2 \leq 4 \\ x_1 \geq 0 \\ x_2 \geq 0 \end{array}$$

1. Rewrite with the slack variables:

2. Write the Lagrangian:

3. Take the partial derivatives and set equal to zero:

4. Consider all ways that the complementary slackness conditions are solved:

Hypothesis	s_1	s_2	s_3	λ_1	λ_2	λ_3	x_1	x_2	$f(x_1, x_2)$
$s_1 = s_2 = s_3 = 0$									
$s_1 \neq 0, s_2 = s_3 = 0$									
$s_2 \neq 0, s_1 = s_3 = 0$									
$s_3 \neq 0, s_1 = s_2 = 0$									
$s_1 \neq 0, s_2 \neq 0, s_3 = 0$									
$s_1 \neq 0, s_3 \neq 0, s_2 = 0$									
$s_2 \neq 0, s_3 \neq 0, s_1 = 0$									
$s_1 \neq 0, s_2 \neq 0, s_3 \neq 0$									

5. Find maximum:

5.7 Kuhn-Tucker Conditions

As you can see, this can be a pain. When dealing explicitly with *non-negativity constraints*, this process is simplified by using the Kuhn-Tucker method.

Because the problem of maximizing a function subject to inequality and non-negativity constraints arises frequently in economics, the **Kuhn-Tucker conditions** provides a method that often makes it easier to both calculate the critical points and identify points that are (local) maxima.

Given a *two-dimensional set-up*:

$$\max_{x_1,x_2} / \min_{x_1,x_2} f(x_1, x_2) \text{ s.t. } \begin{array}{l} c(x_1, x_2) \leq a_1 \\ x_1 \geq 0 \\ gx_2 \geq 0 \end{array}$$

We define the Lagrangian function $L(x_1, x_2, \lambda_1)$ the same as if we did not have the non-negativity constraints:

$$L(x_1, x_2, \lambda_2) = f(x_1, x_2) - \lambda_1(c(x_1, x_2) - a_1)$$

More generally, in n dimensions:

$$L(x_1, \dots, x_n, \lambda_1, \dots, \lambda_k) = f(x_1, \dots, x_n) - \sum_{i=1}^{k} \lambda_i(c_i(x_1, \dots, x_n) - a_i)$$

Kuhn-Tucker and Complementary Slackness Conditions: To find the critical points, we first calculate the Kuhn-Tucker conditions by taking the partial derivatives of the lagrangian function, $L(x_1, \dots, x_n, \lambda_1, \dots, \lambda_k)$, with respect to each of its variables (all choice variables x and all lagrangian multipliers λ) and we calculate the *complementary slackness conditions* by multiplying each partial derivative by its respective variable *and* include non-negativity conditions for all variables (choice variables x and lagrangian multipliers λ).

Kuhn-Tucker Conditions

$$\frac{\partial L}{\partial x_1} \leq 0, \dots, \frac{\partial L}{\partial x_n} \leq 0$$
$$\frac{\partial L}{\partial \lambda_1} \geq 0, \dots, \frac{\partial L}{\partial \lambda_m} \geq 0$$

Complementary Slackness Conditions

$$x_1 \frac{\partial L}{\partial x_1} = 0, \ldots, x_n \frac{\partial L}{\partial x_n} = 0$$
$$\lambda_1 \frac{\partial L}{\partial \lambda_1} = 0, \ldots, \lambda_m \frac{\partial L}{\partial \lambda_m} = 0$$

Non-negativity Conditions

$$x_1 \geq 0 \quad \ldots \quad x_n \geq 0$$
$$\lambda_1 \geq 0 \quad \ldots \quad \lambda_m \geq 0$$

Note that some of these conditions are set equal to 0, while others are set as inequalities!

Note also that to minimize the function $f(x_1, \ldots, x_n)$, the simplest thing to do is maximize the function $-f(x_1, \ldots, x_n)$; all of the conditions remain the same after reformulating as a maximization problem.

There are additional assumptions (notably, f(x) is quasi-concave and the constraints are convex) that are sufficient to ensure that a point satisfying the Kuhn-Tucker conditions is a global max; if these assumptions do not hold, you may have to check more than one point.

Finding the Critical Points with Kuhn-Tucker Conditions: Given the above conditions, to find the critical points we solve the above system of equations. To do so, we must check *all* border and interior solutions to see if they satisfy the above conditions.

In a two-dimensional set-up, this means we must check the following cases:

1. $x_1 = 0, x_2 = 0$ Border Solution
2. $x_1 = 0, x_2 \neq 0$ Border Solution
3. $x_1 \neq 0, x_2 = 0$ Border Solution
4. $x_1 \neq 0, x_2 \neq 0$ Interior Solution

Example 5.5.

Kuhn-Tucker with two variables

Solve the following optimization problem with inequality constraints

$$\max_{x_1,x_2} f(x) = -(x_1^2 + 2x_2^2)$$

$$\text{s.t.} \begin{cases} x_1 + x_2* \leq 4 \\ x_1* \geq 0 \\ x_2* \geq 0 \end{cases}$$

1. Write the Lagrangian:

$$L(x_1, x_2, \lambda) = -(x_1^2 + 2x_2^2) - \lambda(x_1 + x_2 - 4)$$

2. Find the First Order Conditions:

Kuhn-Tucker Conditions

$$\frac{\partial L}{\partial x_1} = -2x_1 - \lambda \leq 0$$
$$\frac{\partial L}{\partial x_2} = -4x_2 - \lambda \leq 0$$
$$\frac{\partial L}{\partial \lambda} = -(x_1 + x_2 - 4) \geq 0$$

Complementary Slackness Conditions

$$x_1 \frac{\partial L}{\partial x_2} = x_1(-2x_1 - \lambda) = 0$$
$$x_2 \frac{\partial L}{\partial x_2} = x_2(-4x_2 - \lambda) = 0$$
$$\lambda \frac{\partial L}{\partial \lambda} = -\lambda(x_1 + x_2 - 4) = 0$$

Non-negativity Conditions

$$x_1 \geq 0$$
$$x_2 \geq 0$$
$$\lambda \geq 0$$

3. Consider all border and interior cases:

Hypothesis	λ	x_1	x_2	$f(x_1, x_2)$
$x_1 = 0, x_2 = 0$	0	0	0	0
$x_1 = 0, x_2 \neq 0$	-16	0	4	-32
$x_1 \neq 0, x_2 = 0$	-8	4	0	-16
$x_1 \neq 0, x_2 \neq 0$	$-\frac{16}{3}$	$\frac{8}{3}$	$\frac{4}{3}$	$-\frac{32}{3}$

4. Find Maximum: Three of the critical points violate the requirement that $\lambda \geq 0$, so the point $(0, 0, 0)$ is the maximum.

Exercise 5.4.

Kuhn-Tucker with logs

$$\max_{x_1,x_2} f(x) = \frac{1}{3}\log(x_1+1) + \frac{2}{3}\log(x_2+1) \text{ s.t. } \begin{array}{l} x_1 + 2x_2 \leq 4 \\ x_1 \geq 0 \\ x_2 \geq 0 \end{array}$$

1. Write the Lagrangian:

2. Find the First Order Conditions:
   Kuhn-Tucker Conditions

   Complementary Slackness Conditions

Non-negativity Conditions

3. Consider all border and interior cases:

Hypothesis	λ x_1 x_2	$f(x_1, x_2)$
$x_1 = 0, x_2 = 0$		
$x_1 = 0, x_2 \neq 0$		
$x_1 \neq 0, x_2 = 0$		
$x_1 \neq 0, x_2 \neq 0$		

4. Find Maximum:

5.8 Applications of Quadratic Forms

Curvature and The Taylor Polynomial as a Quadratic Form: The Hessian is used in a Taylor polynomial approximation to $f(\mathbf{x})$ and provides information about the curvature of $f(\mathbf{x})$ at $\mathbf{x}$ — e.g., which tells us whether a critical point $\mathbf{x}^*$ is a min, max, or saddle point.

1. The second order Taylor polynomial about the critical point $\mathbf{x}^*$ is

$$f(\mathbf{x}^* + \mathbf{h}) = f(\mathbf{x}^*) + \nabla f(\mathbf{x}^*)\mathbf{h} + \frac{1}{2}\mathbf{h}^\top \mathbf{H}(x^*)\mathbf{h} + R(\mathbf{h})$$

2. Since we're looking at a critical point, $\nabla f(\mathbf{x}^*) = 0$; and for small $\mathbf{h}$, $R(\mathbf{h})$ is negligible. Rearranging, we get

$$f(\mathbf{x}^* + \mathbf{h}) - f(\mathbf{x}^*) \approx \frac{1}{2}\mathbf{h}^\top \mathbf{H}(x^*)\mathbf{h}$$

3. The Righthand side here is a quadratic form and we can determine the definiteness of $\mathbf{H}(x^*)$.

6 Probability Theory

Probability and Inferences are mirror images of each other, and both are integral to social science. Probability quantifies uncertainty, which is important because many things in the social world are at first uncertain. Inference is then the study of how to learn about facts you don't observe from facts you do observe.

6.1 Counting rules

Probability in high school is usually really about combinatorics: the probability of event A is the number of ways in which A can occur divided by the number of all other possibilities. This is a very simplified version of probability, which we can call the "counting definition of probability", essentially because each possible event to count is often equally likely and discrete. But it is still good to review the underlying rules here.

Fundamental Theorem of Counting: If an object has j different characteristics that are independent of each other, and each characteristic i has n_i ways of being expressed, then there are $\prod_{i=1}^{j} n_i$ possible unique objects.

Example 6.1.

Counting Possibilities

Suppose we are given a stack of cards. Cards can be either red or black and can take on any of 13 values. There is only one of each color-number combination. In this case,

1. $j =$
2. $n_{\text{color}} =$
3. $n_{\text{number}} =$
4. Number of Outcomes $=$

We often need to count the number of ways to choose a subset from some set of possibilities. The number of outcomes depends on two characteristics of the process: does the order matter and is replacement allowed?

It is useful to think of any problem concretely, e.g. through a **sampling table**: If there are n objects which are numbered 1 to n and we select $k < n$ of them, how many different outcomes are possible?

If the order in which a given object is selected matters, selecting 4 numbered objects in the following order (1, 3, 7, 2) and selecting the same four objects but in a different order such as (7, 2, 1, 3) will be counted as different outcomes.

If replacement is allowed, there are always the same n objects to select from. However, if replacement is not allowed, there is always one less option than the previous round when making a selection. For example, if replacement is not allowed and I am selecting 3 elements from the following set $\{1, 2, 3, 4, 5, 6\}$, I will have 6 options at first, 5 options as I make my second selection, and 4 options as I make my third.

1. So if ***order matters*** AND we are sampling ***with replacement***, the number of different outcomes is n^k.
2. If ***order matters*** AND we are sampling ***without replacement***, the number of different outcomes is $n(n-1)(n-2)...(n-k+1) = \frac{n!}{(n-k)!}$.
3. If ***order doesn't matter*** AND we are sampling ***without replacement***, the number of different outcomes is $\binom{n}{k} = \frac{n!}{(n-k)!k!}$.

Expression $\binom{n}{k}$ is read as "n choose k" and denotes $\frac{n!}{(n-k)!k!}$. Also, note that $0! = 1$.

Example 6.2.

Counting

There are five balls numbered from 1 through 5 in a jar. Three balls are chosen. How many possible choices are there?

1. Ordered, with replacement =
2. Ordered, without replacement =
3. Unordered, without replacement =

Exercise 6.1.

Counting

Four cards are selected from a deck of 52 cards. Once a card has been drawn, it is not reshuffled back into the deck. Moreover, we care only about the complete hand that we get (i.e. we care about the set of selected cards, not the sequence in which it was drawn). How many possible outcomes are there?

6.2 Probability

Probability Definitions: Formal and Informal

Many things in the world are uncertain. In everyday speech, we say that we are *uncertain* about the outcome of random events. Probability is a formal model of uncertainty which provides a measure of uncertainty governed by a particular set of rules. A different model of uncertainty would, of course, have a set of rules different from anything we discuss here. Our focus on probability is justified because it has proven to be a particularly useful model of uncertainty.

Sample Space (S): A set or collection of all possible outcomes from some process. Outcomes in the set can be discrete elements (countable) or points along a continuous interval (uncountable).

Probability Distribution Function: a mapping of each event in the sample space S to the real numbers that satisfy the following three axioms (also called Kolmogorov's Axioms).

Formally,

Definition 6.1.

Probability

Probability is a function that maps events from a sample space to a real number, obeying the axioms of probability.

The axioms of probability make sure that the separate events add up in terms of probability, and – for standardization purposes – that they add up to 1.

Definition 6.2.

Axioms of Probability

1. For any event A, $P(A) \geq 0$.
2. $P(S) = 1$
3. The Countable Additivity Axiom: For any sequence of *disjoint* (mutually exclusive) events $A_1, A_2, ...$ (of which there may be infinitely many),

$$P\left(\bigcup_{i=1}^{k} A_i\right) = \sum_{i=1}^{k} P(A_i)$$

The last axiom is an extension of a union to infinite sets. When there are only two events in the space, it boils down to:

$$P(A_1 \cup A_2) = P(A_1) + P(A_2) \quad \text{for disjoint } A_1, A_2$$

Probability Operations

Using these three axioms, we can define all of the common rules of probability.

1. $P(\emptyset) = 0$
2. For any event A, $0 \leq P(A) \leq 1$.
3. $P(A^C) = 1 - P(A)$
4. If $A \subset B$ (A is a subset of B), then $P(A) \leq P(B)$.
5. For *any* two events A and B, $P(A \cup B) = P(A) + P(B) - P(A \cap B)$
6. Boole's Inequality: For any sequence of n events (which need not be disjoint) $A_1, A_2, ..., A_n$, then $P\left(\bigcup_{i=1}^{n} A_i\right) \leq \sum_{i=1}^{n} P(A_i)$.

Example 6.3.

Probability

Assume we have an evenly-balanced, six-sided die.

Then,

1. Sample space S =
2. $P(1) = \cdots = P(6) =$
3. $P(\emptyset) = P(7) =$
4. $P(\{1, 3, 5\}) =$
5. $P(\{1, 2\}^C) = P(\{3, 4, 5, 6\}) =$
6. Let $A = \{1, 2, 3, 4, 5\} \subset S$. Then $P(A) = 5/6 < P(S) =$
7. Let $A = \{1, 2, 3\}$ and $B = \{2, 4, 6\}$. Then $A \cup B$? $A \cap B$? $P(A \cup B)$?

Exercise 6.2.

Probability

Suppose you had a pair of four-sided dice. You sum the results from a single toss. Let us call this sum, or the outcome, X.

1. What is $P(X = 5)$, $P(X = 3)$, $P(X = 6)$?
2. What is $P(X = 5 \cup X = 3)^C$?

6.3 Conditional Probability and Bayes Rule

Conditional Probability: The conditional probability $P(A|B)$ of an event A is the probability of A, given that another event B has occurred. Conditional probability allows for the inclusion of other information into the calculation of the probability of an event. It is calculated as

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Note that conditional probabilities are probabilities and must also follow the Kolmagorov axioms of probability.

Example 6.4.

Conditional Probability 1

Assume A and B occur with the following frequencies:

	A	A^c
B	n_{ab}	n_{a^cb}
B^C	n_{ab^c}	$n_{(ab)^c}$

and let $n_{ab} + n_{a^Cb} + n_{ab^C} + n_{(ab)^C} = N$. Then

1. $P(A) =$
2. $P(B) =$
3. $P(A \cap B) =$
4. $P(A|B) = \frac{P(A \cap B)}{P(B)} =$
5. $P(B|A) = \frac{P(A \cap B)}{P(A)} =$

Example 6.5.

Conditional Probability 2

A six-sided die is rolled. What is the probability of a 1, given the outcome is an odd number?

You could rearrange the fraction to highlight how a joint probability could be expressed as the product of a conditional probability.

Definition 6.3.

Multiplicative Law of Probability

The probability of the intersection of two events A and B is $P(A \cap B) = P(A)P(B|A) = P(B)P(A|B)$ which follows directly from the definition of conditional probability. More generally,

$$P(A_1 \cap \cdots \cap A_k) = P(A_k|A_{k-1} \cap \cdots \cap A_1) \times P(A_{k-1}|A_{k-2} \cap \cdots A_1) \times ... \times P(A_2|A_1) \times P(A_1)$$

Sometimes it is easier to calculate these conditional probabilities and sum them than it is to calculate $P(A)$ directly.

Definition 6.4.

Law of total probability

Let S be the sample space of some experiment and let the disjoint k events $B_1, ..., B_k$ partition S, such that $P(B_1 \cup ... \cup B_k) = P(S) = 1$. If A is some other event in S, then the events $A \cap B_1, A \cap B_2, ..., A \cap B_k$ will form a partition of A and we can write A as

$$A = (A \cap B_1) \cup \cdots \cup (A \cap B_k)$$

.

Since the k events are disjoint,

$$\begin{aligned} P(A) &= \sum_{i=1}^{k} P(A \cap B_i) \\ &= \sum_{i=1}^{k} P(B_i)P(A|B_i) \end{aligned}$$

Bayes Rule: Assume that events $B_1, ..., B_k$ form a partition of the space S. Then by the Law of Total Probability

$$P(B_j|A) = \frac{P(A \cap B_j)}{P(A)} = \frac{P(B_j)P(A|B_j)}{\sum_{i=1}^{k} P(B_i)P(A|B_i)}$$

If there are only two states of B, then this is just

$$P(B_1|A) = \frac{P(B_1)P(A|B_1)}{P(B_1)P(A|B_1) + P(B_2)P(A|B_2)}$$

Bayes' rule determines the posterior probability of a state $P(B_j|A)$ by calculating the probability $P(A \cap B_j)$ that both the event A and the state B_j will occur and dividing it by the probability that the event will occur regardless of the state (by summing across all B_i). The states could be something like Normal/Defective, Healthy/Diseased, Republican/Democrat/Independent, etc. The event on which one conditions could be something like a sampling from a batch of components, a test for a disease, or a question about a policy position.

Prior and Posterior Probabilities: Above, $P(B_1)$ is often called the prior probability, since it's the probability of B_1 before anything else is known. $P(B_1|A)$ is called the posterior probability, since it's the probability after other information is taken into account.

Example 6.6.

Bayes' Rule

In a given town, 40% of the voters are Democrat and 60% are Republican. The president's budget is supported by 50% of the Democrats and 90% of the Republicans. If a randomly (equally likely) selected voter is found to support the president's budget, what is the probability that they are a Democrat?

Exercise 6.3.

Conditional Probability

Assume that 2% of the population of the U.S. are members of some extremist militia group. We develop a survey that positively classifies someone as being a member of a militia group given that they are a member 95% of the time and negatively classifies someone as not being a member of a militia group given that they are not a member 97% of the time. What is the probability that someone positively classified as being a member of a militia group is actually a militia member?

6.4 Independence

Definition 6.5.

Independence

If the occurrence or nonoccurrence of either events A and B provides no information about the occurrence or nonoccurrence of the other, then A and B are independent.

If A and B are independent, then

1. $P(A|B) = P(A)$
2. $P(B|A) = P(B)$
3. $P(A \cap B) = P(A)P(B)$
4. More generally than the above, $P(\bigcap_{i=1}^{k} A_i) = \prod_{i=1}^{K} P(A_i)$

Are mutually exclusive events independent of each other?

No. If A and B are mutually exclusive, then they cannot happen simultaneously. If we know that A occurred, then we know that B couldn't have occurred. Because of this, A and B aren't independent.

Pairwise Independence: A set of more than two events $A_1, A_2, \ldots, A_k$ is pairwise independent if $P(A_i \cap A_j) = P(A_i)P(A_j)$, $\forall i \neq j$. Note that this does **not** necessarily imply joint independence.

Conditional Independence: If A and B are independent once you know the occurrence of a third event C, then A and B are conditionally independent (conditional on C):

1. $P(A|B \cap C) = P(A|C)$
2. $P(B|A \cap C) = P(B|C)$
3. $P(A \cap B|C) = P(A|C)P(B|C)$

Just because two events are conditionally independent does not mean that they are independent. Actually it is hard to think of real-world things that are "unconditionally" independent. That's why it's always important to ask about a finding: What was it conditioned on? For example, suppose that a graduate school admission decisions are done by only one professor, who picks a group of 50 bright students and flips a coin for each student to generate a class of about 25 students. Then the the probability that two students get accepted are conditionally independent, because they are determined by two separate coin tosses. However, this does not mean that their admittance is not completely independent. Knowing that student A got in gives us information about whether student B got in, if we think that the professor originally picked her pool of 50 students by merit.

Perhaps more counter-intuitively: If two events are already independent, then it might seem that no amount of "conditioning" will make them dependent. But this is not always so. For example, imagine that you own a house with a lawn (a very extreme hypothetical!) Let A be the event that it rained yesterday and B the event that your sprinkler system went off yesterday. Suppose that your sprinkler system is set to randomly go off and so A and B are independent of one another. $P(A \mid B) = P(A)$. But now let C be the event that the grass is wet. The grass can be wet *either* due to the rain or due to the sprinkler. For conditional independence to hold here, then $P(A \mid C)$ must be equal to $P(A \mid B \cap C)$. But this is not true.

Let $P(A) = .5$ and $P(B) = .5$.

The marginal probability $P(C)$ is

$$P(C) = P(A \cup B) = P(A) + P(B) - P(A \cap B) = .75$$

The conditional probability $P(A|C)$ is

$$P(A|C) = \frac{P(C|A)P(A)}{P(C)} = \frac{1 \times .5}{.75} = \frac{2}{3}$$

The conditional probability $P(A|B \cap C)$ is

$$P(A|B \cap C) = \frac{P(C \cap B|A)P(A)}{P(C \cap B)} = \frac{P(C \cap B|A)P(A)}{P(C|B)P(B)} \frac{.5 \times .5}{.5} = \frac{1}{2}$$

Intuitively, given that the grass is wet, knowing that it rained yesterday tells us that it is *less* likely that the sprinkler also went off!

6.5 Random Variables

Most questions in the social sciences involve events, rather than numbers per se. To analyze and reason about events quantitatively, we need a way of mapping events to numbers. A random variable does exactly that.

Definition 6.6.

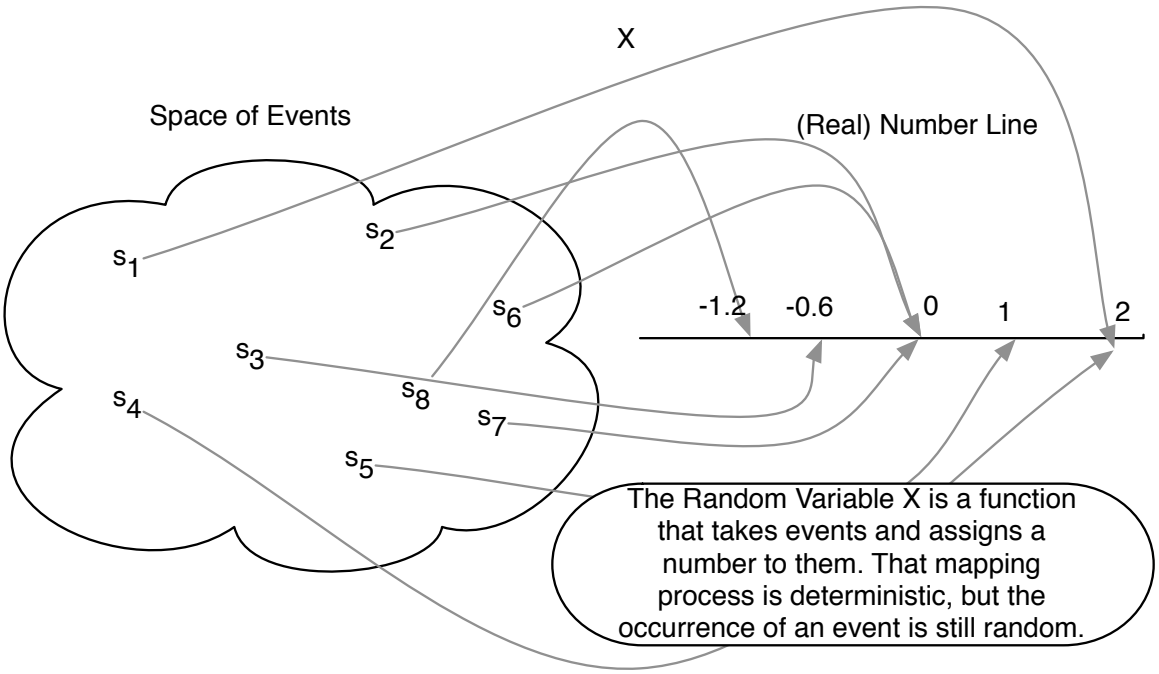


Figure 6.1: The Random Variable as a Real-Valued Function

Random Variable

A random variable is a measurable function X that maps from the sample space S to the set of real numbers R. It assigns a real number to every outcome $s \in S$.

Figure 6.1 shows a image of the function. It might seem strange to define a random variable as a function – which is neither random nor variable. The randomness comes from the realization of an event from the sample space s.

Randomness means that the outcome of some experiment is not deterministic, i.e. there is some probability $(0 < P(A) < 1)$ that the event will occur.

The support of a random variable is all values for which there is a positive probability of occurrence.

Example: Flip a fair coin two times. What is the sample space?

A random variable must map events to the real line. For example, let a random variable X be the number of heads. The event (H, H) gets mapped to 2 $X(s) = 2$, and the events $\{(H, T), (T, H)\}$ gets mapped to 1 $(X(s) = 1)$, the event (T, T) gets mapped to 0 $(X(s) = 0)$.

What are other possible random variables?

6.6 Distributions

We now have two main concepts in this section – probability and random variables. Given a sample space S and the same experiment, both probability and random variables take events as their inputs. But they output different things (probabilities measure the "size" of events, random variables give a number in a way that the analyst chose to define the random variable). How do the two concepts relate?

The concept of distributions is the natural bridge between these two concepts.

Definition 6.7.

Distribution of a random variable

A distribution of a random variable is a function that specifies the probabilities of all events associated with that random variable. There are several types of distributions: A probability mass function for a discrete random variable and probability density function for a continuous random variable.

Notice how the definition of distributions combines two ideas of random variables and probabilities of events. First, the distribution considers a random variable, call it X. X can take a number of possible numeric values.

Example 6.7.

Total Number of Occurrences

Consider three binary outcomes, one for each patient recovering from a disease: R_i denotes the event in which patient i ($i = 1, 2, 3$) recovers from a disease. R_1, R_2, and R_3. How would we represent the total number of people who end up recovering from the disease?

Solution. Define the random variable X be the total number of people (out of three) who recover from the disease. Random variables are functions, that take as an input a set of events (in the sample space S) and deterministically assigns them to a number of the analyst's choice.

Recall that with each of these numerical values there is a class of *events*. In the previous example, for $X = 3$ there is one outcome (R_1, R_2, R_3) and for $X = 1$ there are multiple $(\{(R_1, R_2^c, R_3^c), (R_1^c, R_2, R_3^c), (R_1^c, R_2^c, R_3), \})$. Now, the thing to notice here is that each of these events naturally come with a probability associated with them. That is, $P(R_1, R_2, R_3)$ is a number from 0 to 1, as is $P(R_1, R_2^c, R_3^c)$. These all have probabilities because they are in the sample space S. The function that tells us these probabilities that are associated with a numerical value of a random variable is called a distribution.

In other words, a random variable X *induces a probability distribution* P (sometimes written P_X to emphasize that the probability density is about the r.v. X)

Discrete Random Variables

The formal definition of a random variable is easier to given by separating out two cases: discrete random variables when the numeric summaries of the events are discrete, and continuous random variables when they are continuous.

Definition 6.8.

Discrete Random Variable

X is a discrete random variable if it can assume only a finite or countably infinite number of distinct values. Examples: number of wars per year, heads or tails.

The distribution of a discrete r.v. is a PMF:

Definition 6.9.

Probability Mass Function

For a discrete random variable X, the probability mass function (Also referred to simply as the "probability distribution.") (PMF), $p(x) = P(X = x)$, assigns probabilities to a countable number of distinct x values such that

1. $0 \leq p(x) \leq 1$
2. $\sum_{y} p(x) = 1$

Example: For a fair six-sided die, there is an equal probability of rolling any number. Since there are six sides, the probability mass function is then $p(y) = 1/6$ for $y = 1, \ldots, 6$, 0 otherwise.}

In a discrete random variable, **cumulative distribution function** , $F(x)$ or $P(X \leq x)$, is the probability that X is less than or equal to some value x, or

$$P(X \leq x) = \sum_{i \leq x} p(i)$$

Properties a CDF must satisfy:

1. $F(x)$ is non-decreasing in x.
2. $\lim_{x \to -\infty} F(x) = 0$ and $\lim_{x \to \infty} F(x) = 1$
3. $F(x)$ is right-continuous.

Note that $P(X > x) = 1 - P(X \leq x)$.

Definition 6.10. For a fair six-sided die with its value as Y, What are the following?

1. $P(Y \leq 1)$
2. $P(Y \leq 3)$
3. $P(Y \leq 6)$

Continuous Random Variables

We also have a similar definition for *continuous* random variables.

Definition 6.11.

Continuous Random Variable

X is a continuous random variable if there exists a nonnegative function $f(x)$ defined for all real $x \in (-\infty, \infty)$, such that for any interval A, $P(X \in A) = \int_A f(x)dx$. Examples: age, income, GNP, temperature.

Definition 6.12.

Probability density function

The function f above is called the probability density function (pdf) of X and must satisfy

$$f(x) \geq 0$$

$$\int_{-\infty}^{\infty} f(x)dx = 1$$

Note also that $P(X = x) = 0$ — i.e., the probability of any point y is zero.

While continuous random variables do not have a PMF (since the PMF would be 0 at every point), the cumulative distribution function is defined in the exact same way. The cumulative distribution gives the probability that Y lies on the interval $(-\infty, y)$ and is defined as

$$F(x) = P(X \leq x) = \int_{-\infty}^{x} f(s)ds$$

We can also make statements about the probability of Y falling in an interval $a \leq y \leq b$.

$$P(a \leq x \leq b) = \int_{a}^{b} f(x)dx$$

The PDF and CDF are linked by the integral: The CDF of the integral of the PDF:

$$f(x) = F'(x) = \frac{dF(x)}{dx}$$

Example 6.8.

Continuous R.V.

For $f(y) = 1, \quad 0 < y < 1$, find: (1) The CDF $F(y)$ and (2) The probability $P(0.5 < y < 0.75)$.

6.7 Joint Distributions

Often, we are interested in two or more random variables defined on the same sample space. The distribution of these variables is called a joint distribution. Joint distributions can be made up of any combination of discrete and continuous random variables.

Joint Probability Distribution: If both X and Y are random variable, their joint probability mass/density function assigns probabilities to each pair of outcomes

Discrete:

$$p(x, y) = P(X = x, Y = y)$$

such that $p(x, y) \in [0, 1]$ and

$$\sum\sum p(x, y) = 1$$

Continuous:

$$f(x, y); P((X, Y) \in A) = \iint_A f(x, y)dxdy$$

s.t. $f(x, y) \geq 0$ and

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x, y)dxdy = 1$$

If X and Y are independent, then $P(X = x, Y = y) = P(X = x)P(Y = y)$ and $f(x, y) = f(x)f(y)$

Marginal Probability Distribution: probability distribution of only one of the two variables (ignoring information about the other variable), we can obtain the marginal distribution by summing/integrating across the variable that we don't care about:

- Discrete: $p_X(x) = \sum_i p(x, y_i)$
- Continuous: $f_X(x) = \int_{-\infty}^{\infty} f(x, y)dy$

Conditional Probability Distribution: probability distribution for one variable, holding the other variable fixed. Recalling from the previous lecture that $P(A|B) = \frac{P(A \cap B)}{P(B)}$, we can write the conditional distribution as

- Discrete: $p_{Y|X}(y|x) = \frac{p(x,y)}{p_X(x)}, \quad p_X(x) > 0$
- Continuous: $f_{Y|X}(y|x) = \frac{f(x,y)}{f_X(x)}, \quad f_X(x) > 0$

Exercise 6.4.

Discrete, Joint Distributions

Suppose we are interested in the outcomes of flipping a coin and rolling a 6-sided die at the same time. The sample space for this process contains 12 elements:

$$\{(H,1),(H,2),(H,3),(H,4),(H,5),(H,6),(T,1),(T,2),(T,3),(T,4),(T,5),(T,6)\}$$

We can define two random variables X and Y such that $X = 1$ if heads and $X = 0$ if tails, while Y equals the number on the die.

We can then make statements about the joint distribution of X and Y. What are the following?

1. $P(X = x)$
2. $P(Y = y)$
3. $P(X = x, Y = y)$
4. $P(X = x|Y = y)$
5. Are X and Y independent?

6.8 Expectation

We often want to summarize some characteristics of the distribution of a random variable. The most important summary is the expectation (or expected value, or mean), in which the possible values of a random variable are weighted by their probabilities.

Definition 6.13.

Expectation of a discrete R.V.

The expected value of a discrete random variable Y is

$$E(Y) = \sum_y yP(Y = y) = \sum_y yp(y)$$

In words, it is the weighted average of all possible values of Y, weighted by the probability that y occurs. It is not necessarily the number we would expect Y to take on, but the average value of Y after a large number of repetitions of an experiment.

Example 6.9.

Expectation of a discrete R.V.

What is the expectation of a fair, six-sided die?

Expectation of a Continuous Random Variable: The expected value of a continuous random variable is similar in concept to that of the discrete random variable, except that instead of summing using probabilities as weights, we integrate using the density to weight. Hence, the expected value of the continuous variable Y is defined by

$$E(Y) = \int_y yf(y)dy$$

Example 6.10.

Expectation of a continuous R.V.

Find $E(Y)$ for $f(y) = \frac{1}{1.5}, \quad 0 < y < 1.5$.

Expected Value of a Function

Remember: An Expected Value is a type of weighted average. We can extend this to composite functions. For random variable Y,

If Y is Discrete with PMF $p(y)$,

$$E[g(Y)] = \sum_y g(y)p(y)$$

If Y is Continuous with PDF $f(y)$,

$$E[g(Y)] = \int_{-\infty}^{\infty} g(y)f(y)dy$$

Properties of Expected Values

Dealing with Expectations is easier when the thing inside is a sum. The intuition behind this that Expectation is an integral, which is a type of sum.

1. Expectation of a constant is a constant
$$E(c) = c$$
2. Constants come out
$$E(cg(Y)) = cE(g(Y))$$
3. Expectation is Linear
$$E(g(Y_1) + \cdots + g(Y_n)) = E(g(Y_1)) + \cdots + E(g(Y_n)),$$
regardless of independence

4. Expected Value of Expected Values:

$$E(E(Y)) = E(Y)$$

(because the expected value of a random variable is a constant)

Finally, if X and Y are independent, even products are easy:

$$E(XY) = E(X)E(Y)$$

Conditional Expectation: With joint distributions, we are often interested in the expected value of a variable Y if we could hold the other variable X fixed. This is the conditional expectation of Y given $X = x$:

1. Y discrete: $E(Y|X = x) = \sum_y y p_{Y|X}(y|x)$
2. Y continuous: $E(Y|X = x) = \int_y y f_{Y|X}(y|x)dy$

The conditional expectation is often used for prediction when one knows the value of X but not Y

6.9 Variance and Covariance

We can also look at other summaries of the distribution, which build on the idea of taking expectations. Variance tells us about the "spread" of the distribution; it is the expected value of the squared deviations from the mean of the distribution. The standard deviation is simply the square root of the variance.

Definition 6.14. The Variance of a Random Variable Y is

$$\text{Var}(Y) = E[(Y - E(Y))^2] = E(Y^2) - [E(Y)]^2$$

The Standard Deviation is the square root of the variance :

$$SD(Y) = \sigma_Y = \sqrt{\text{Var}(Y)}$$

Example 6.11. Given the following PMF:

$$f(x) = \begin{cases} \frac{3!}{x!(3-x)!}(\frac{1}{2})^3 & x = 0, 1, 2, 3 \\ 0 \quad otherwise \end{cases}$$

What is Var(x)?

Hint: First calculate $E(X)$ and $E(X^2)$

Definition 6.15.

Covariance

The covariance measures the degree to which two random variables vary together; if the covariance between X and Y is positive, X tends to be larger than its mean when Y is larger than its mean.

$$\text{Cov}(X, Y) = E[(X - E(X))(Y - E(Y))]$$

We can also write this as

$$\begin{aligned}\text{Cov}(X, Y) &= E\left(XY - XE(Y) - E(X)Y + E(X)E(Y)\right) \\ &= E(XY) - E(X)E(Y) - E(X)E(Y) + E(X)E(Y) \\ &= E(XY) - E(X)E(Y)\end{aligned}$$

The covariance of a variable with itself is the variance of that variable.

The Covariance is unfortunately hard to interpret in magnitude. The correlation is a standardized version of the covariance, and always ranges from -1 to 1.

Definition 6.16.

Correlation

The correlation coefficient is the covariance divided by the standard deviations of X and Y. It is a unitless measure and always takes on values in the interval $[-1, 1]$.

$$\text{Corr}(X, Y) = \frac{\text{Cov}(X, Y)}{\sqrt{\text{Var}(X)\text{Var}(Y)}} = \frac{\text{Cov}(X, Y)}{SD(X)SD(Y)}$$

Properties of Variance and Covariance:

1. $\text{Var}(c) = 0$
2. $\text{Var}(cY) = c^2\text{Var}(Y)$
3. $\text{Cov}(Y, Y) = \text{Var}(Y)$
4. $\text{Cov}(X, Y) = \text{Cov}(Y, X)$
5. $\text{Cov}(aX, bY) = ab\text{Cov}(X, Y)$
6. $\text{Cov}(X + a, Y) = \text{Cov}(X, Y)$
7. $\text{Cov}(X + Z, Y + W) = \text{Cov}(X, Y) + \text{Cov}(X, W) + \text{Cov}(Z, Y) + \text{Cov}(Z, W)$
8. $\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y) + 2\text{Cov}(X, Y)$

Exercise 6.5.

Expectation and Variance 1

Suppose we have a PMF with the following characteristics:

$$P(X = -2) = \frac{1}{5}$$
$$P(X = -1) = \frac{1}{6}$$
$$P(X = 0) = \frac{1}{5}$$
$$P(X = 1) = \frac{1}{15}$$
$$P(X = 2) = \frac{11}{30}$$

1. Calculate the expected value of X

Define the random variable $Y = X^2$.

2. Calculate the expected value of Y. (Hint: It would help to derive the PMF of Y first in order to calculate the expected value of Y in a straightforward way)
3. Calculate the variance of X.

Exercise 6.6.

Expectation and Variance 2

1. Find the expectation and variance

Given the following PDF:

$$f(x) = \begin{cases} \frac{3}{10}(3x - x^2) & 0 \leq x \leq 2 \\ 0 & otherwise \end{cases}$$

Exercise 6.7.

Expectation and Variance 3

1. Find the mean and standard deviation of random variable X. The PDF of this X is as follows:

$$f(x) = \begin{cases} \frac{1}{4}x & 0 \leq x \leq 2 \\ \frac{1}{4}(4-x) & 2 \leq x \leq 4 \\ 0 & otherwise \end{cases}$$

2. Next, calculate $P(X < \mu - \sigma)$ Remember, μ is the mean and σ is the standard deviation

6.10 Distributions

A distribution is defined by its cumulative distribution function. There are many common distributions that have useful properties that appear in probability and statistics.

Two *discrete* distributions used often are:

Definition 6.17.

Binomial Distribution

Y is distributed binomial if it represents the number of "successes" observed in n independent, identical "trials," where the probability of success in any trial is p and the probability of failure is $q = 1 - p$.

For any particular sequence of y successes and $n - y$ failures, the probability of obtaining that sequence is $p^y q^{n-y}$ (by the multiplicative law and independence). However, there are $\binom{n}{y} = \frac{n!}{(n-y)!y!}$ ways of obtaining a sequence with y successes and $n - y$ failures. So the binomial distribution is given by

$$p(y) = \binom{n}{y} p^y q^{n-y}, \quad y = 0, 1, 2, \ldots, n$$

with mean $\mu = E(Y) = np$ and variance $\sigma^2 = \mathrm{Var}(Y) = npq$.

Example 6.12.

Binomial distribution

Republicans vote for Democrat-sponsored bills 2% of the time. What is the probability that out of 10 Republicans questioned, half voted for a particular Democrat-sponsored bill? What is the mean number of Republicans voting for Democrat-sponsored bills? The variance? 1. $P(Y = 5) = 1$. $E(Y) = 1$. $\text{Var}(Y) = 6$

Definition 6.18.

Poisson Distribution

A random variable Y has a Poisson distribution if

$$P(Y = y) = \frac{\lambda^y}{y!} e^{-\lambda}, \quad y = 0, 1, 2, \ldots, \quad \lambda > 0$$

The Poisson has the unusual feature that its expectation equals its variance: $E(Y) = \text{Var}(Y) = \lambda$. The Poisson distribution is often used to model rare event counts: counts of the number of events that occur during some unit of time. λ is often called the "arrival rate."

Example 6.13.

Poisson Distribution

Border disputes occur between two countries through a Poisson Distribution, at a rate of 2 per month. What is the probability of 0, 2, and less than 5 disputes occurring in a month?

Two *continuous* distributions used often are:

Definition 6.19.

Uniform Distribution

A random variable Y has a continuous uniform distribution on the interval (α, β) if its density is given by

$$f(y) = \frac{1}{\beta - \alpha}, \quad \alpha \leq y \leq \beta$$

The mean and variance of Y are $E(Y) = \frac{\alpha+\beta}{2}$ and $\text{Var}(Y) = \frac{(\beta-\alpha)^2}{12}$.

Example 6.14.

Uniform

For Y uniformly distributed over $(1, 3)$, what are the following probabilities?

1. $P(Y = 2)$
2. Its density evaluated at 2, or $f(2)$
3. $P(Y \leq 2)$
4. $P(Y > 2)$

Definition 6.20.

Normal Distribution

A random variable Y is normally distributed with mean $E(Y) = \mu$ and variance $\text{Var}(Y) = \sigma^2$ if its density is

$$f(y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y-\mu)^2}{2\sigma^2}}$$

See Figure 6.2 are various Normal Distributions with the same $\mu = 1$ and two versions of the variance.

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

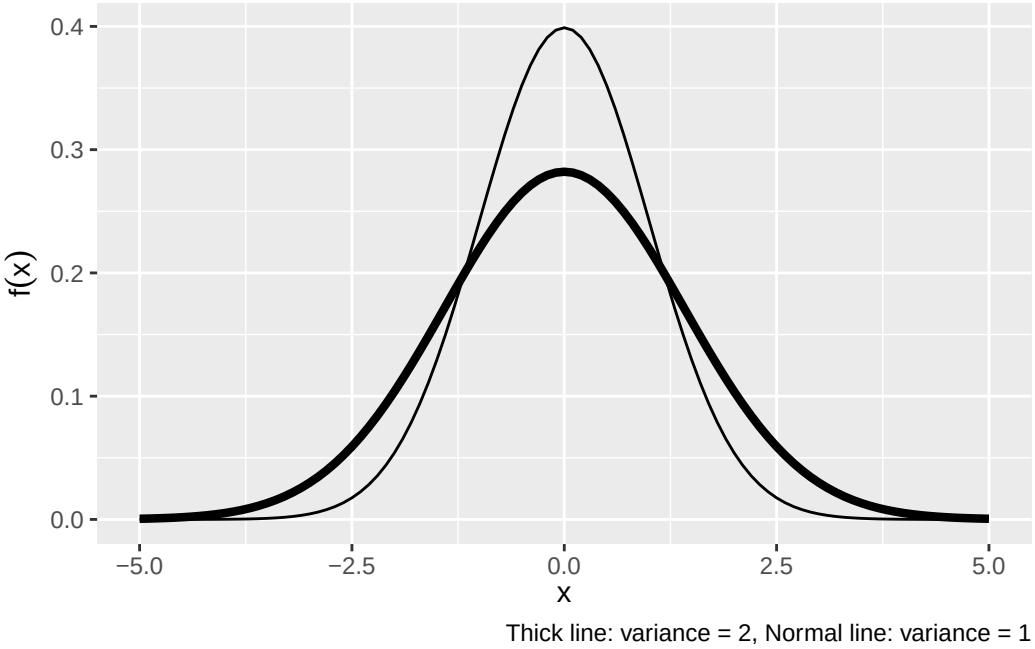


Figure 6.2: Normal Distribution Density

6.11 Summarizing Observed Events (Data)

So far, we've talked about distributions in a theoretical sense, looking at different properties of random variables. We don't observe random variables; we observe realizations of the random variable. These realizations of events are roughly equivalent to what we mean by "data". We'll spend more time in the intro class talking about this from the standpoint of *estimands*, *estimators* and *estimates*.

Sample mean: This is the most common measure of central tendency, calculated by summing across the observations and dividing by the number of observations.

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Example:

X	6	3	7	5	5	5	6	4	7	2
Y	1	2	1	2	2	1	2	0	2	0

1. $\bar{x} =$ $\bar{y} =$
2. median(x) = median(y) =
3. $m_x =$ $m_y =$

Dispersion: We also typically want to know how spread out the data are relative to the center of the observed distribution. There are several ways to measure dispersion.

Sample variance: The sample variance is the sum of the squared deviations from the sample mean, divided by the number of observations minus 1.

$$\widehat{\text{Var}}(X) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

Again, this is an *estimator* of the variance of a random variable; we divide by $n-1$ instead of n in order to get an unbiased estimator.

Standard deviation: The sample standard deviation is the square root of the sample variance.

$$\widehat{SD}(X) = \sqrt{\hat{\text{Var}}(X)} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Covariance and Correlation: Both of these quantities measure the degree to which two variables vary together, and are estimates of the covariance and correlation of two random variables as defined above.

1. **Sample covariance**: $\hat{\text{Cov}}(X, Y) = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})$
2. **Sample correlation**: $\hat{\text{Corr}} = \frac{\hat{\text{Cov}}(X,Y)}{\sqrt{\hat{\text{Var}}(X)\hat{\text{Var}}(Y)}}$

Example 6.15.

Sample Covariance and Correlation

Example: Using the above table, calculate the sample versions of:

1. $\text{Cov}(X, Y)$
2. $\text{Corr}(X, Y)$

6.12 Asymptotic Theory

In theoretical and applied research, asymptotic arguments are often made. In this section we briefly introduce some of this material.

What are asymptotics? In probability theory, asymptotic analysis is the study of limiting behavior. By limiting behavior, we mean the behavior of some random process as the number of observations gets larger and larger.

Why is this important? We rarely know the true process governing the events we see in the social world. It is helpful to understand how such unknown processes theoretically must behave and asymptotic theory helps us do this.

6.12.1 CLT and LLN

We are now finally ready to revisit, with a bit more precise terms, the two pillars of statistical theory we motivated Section @ref(limitsfun) with.

Theorem 6.1.

Central Limit Theorem

Let $\{X_n\} = \{X_1, X_2, ...\}$ be a sequence of i.i.d. random variables with finite mean (μ) and variance (σ^2). Then, the sample mean $\bar{X}_n = \frac{X_1+X_2+\cdots+X_n}{n}$ increasingly converges into a Normal distribution.

$$\frac{\bar{X}_n - \mu}{\sigma/\sqrt{n}} \xrightarrow{d} Normal(0, 1),$$

Another way to write this as a probability statement is that for all real numbers a,

$$P\left(\frac{\bar{X}_n - \mu}{\sigma/\sqrt{n}} \leq a\right) \to \Phi(a)$$

as $n \to \infty$, where

$$\Phi(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}\, dx$$

is the CDF of a Normal distribution with mean 0 and variance 1.

This result means that, as n grows, the distribution of the sample mean $\bar{X}_n = \frac{1}{n}(X_1 + X_2 + \cdots + X_n)$ is approximately normal with mean μ and standard deviation $\frac{\sigma}{\sqrt{n}}$, i.e.,

$$\bar{X}_n \approx \mathcal{N}\left(\mu, \frac{\sigma^2}{n}\right).$$

The standard deviation of $\bar{X}_n$ (which is roughly a measure of the precision of $\bar{X}_n$ as an estimator of μ) decreases at the rate $1/\sqrt{n}$, so, for example, to increase its precision by 10 (i.e., to get one more digit right), one needs to collect $10^2 = 100$ times more units of data.

Intuitively, this result also justifies that whenever a lot of small, independent processes somehow combine together to form the realized observations, practitioners often feel comfortable assuming Normality.

Theorem 6.2.

Weak Law of Large Numbers (LLN)

For any draw of independent random variables with the same mean μ, the sample average after n draws, $\bar{X}_n = \frac{1}{n}(X_1 + X_2 + ... + X_n)$, converges in probability to the expected value of X, μ as $n \to \infty$:

$$\lim_{n\to\infty} P(|\bar{X}_n - \mu| > \varepsilon) = 0$$

A shorthand of which is $\bar{X}_n \xrightarrow{p} \mu$, where the arrow is read as "converges in probability to" as $n \to \infty$. In other words, $P(\lim_{n\to\infty} \bar{X}_n = \mu) = 1$. This is an important motivation for the widespread use of the sample mean, as well as the intuition link between averages and expected values.

More precisely this version of the LLN is called the *weak* law of large numbers because it leaves open the possibility that $|\bar{X}_n - \mu| > \varepsilon$ occurs many times. The *strong* law of large numbers states that, under a few more conditions, the probability that the limit of the sample average is the true mean is 1 (and other possibilities occur with probability 0), but the difference is rarely consequential in practice.

The Strong Law of Large Numbers holds so long as the expected value exists; no other assumptions are needed. However, the rate of convergence will differ greatly depending on the distribution underlying the observed data. When extreme observations occur often (i.e. kurtosis is large), the rate of convergence is much slower. Cf. The distribution of financial returns.

6.12.2 Big $\mathcal{O}$ Notation

Some of you may encounter "big-OH' '-notation. If f, g are two functions, we say that $f = \mathcal{O}(g)$ if there exists some constant, c, such that $f(n) \leq c \times g(n)$ for large enough n. This notation is useful for simplifying complex problems in game theory, computer science, and statistics.

Example 6.16. What is $\mathcal{O}(5\exp(0.5n) + n^2 + n/2)$? Answer: $\exp(n)$. Why? Because, for large n,

$$\frac{5\exp(0.5n) + n^2 + n/2}{\exp(n)} \leq \frac{c\exp(n)}{\exp(n)} = c.$$

whenever $n > 4$ and where $c = 1$.

7 Linear Algebra

Topics: • Working with Vectors • Linear Independence • Basics of Matrix Algebra • Square Matrices • Linear Equations • Systems of Linear Equations • Systems of Equations as Matrices • Solving Augmented Matrices and Systems of Equations • Rank • The Inverse of a Matrix • Inverse of Larger Matrices

7.1 Working with Vectors

Vector: A vector in n-space is an ordered list of n numbers. These numbers can be represented as either a row vector or a column vector:

$$\mathbf{v}\begin{pmatrix} v_1 & v_2 & \dots & v_n \end{pmatrix}, \mathbf{v} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}$$

We can also think of a vector as defining a point in n-dimensional space, usually $\mathbf{R}^n$; each element of the vector defines the coordinate of the point in a particular direction.

Vector Addition and Subtraction: If two vectors, $\mathbf{u}$ and $\mathbf{v}$, have the same length (i.e. have the same number of elements), they can be added (subtracted) together:

$$\mathbf{u} + \mathbf{v} = \begin{pmatrix} u_1 + v_1 & u_2 + v_2 & \cdots & u_k + v_n \end{pmatrix}$$

$$\mathbf{u} - \mathbf{v} = \begin{pmatrix} u_1 - v_1 & u_2 - v_2 & \cdots & u_k - v_n \end{pmatrix}$$

Scalar Multiplication: The product of a scalar c (i.e. a constant) and vector $\mathbf{v}$ is:

$$c\mathbf{v} = \begin{pmatrix} cv_1 & cv_2 & \dots & cv_n \end{pmatrix}$$

Vector Inner Product: The inner product (also called the dot product or scalar product) of two vectors $\mathbf{u}$ and $\mathbf{v}$ is again defined if and only if they have the same number of elements

$$\mathbf{u} \cdot \mathbf{v} = u_1 v_1 + u_2 v_2 + \cdots + u_n v_n = \sum_{i=1}^{n} u_i v_i$$

If $\mathbf{u} \cdot \mathbf{v} = 0$, the two vectors are orthogonal (or perpendicular).

Vector Norm: The norm of a vector is a measure of its length. There are many different ways to calculate the norm, but the most common is the Euclidean norm (which corresponds to our usual conception of distance in three-dimensional space):

$$||\mathbf{v}|| = \sqrt{\mathbf{v} \cdot \mathbf{v}} = \sqrt{v_1 v_1 + v_2 v_2 + \cdots + v_n v_n}$$

Example 7.1.

Vector Algebra

Let $a = \begin{pmatrix} 2 & 1 & 2 \end{pmatrix}$, $b = \begin{pmatrix} 3 & 4 & 5 \end{pmatrix}$. Calculate the following:

1. $a - b$
2. $a \cdot b$

Exercise 7.1.

Vector Algebra

Let $u = \begin{pmatrix} 7 & 1 & -5 & 3 \end{pmatrix}$, $v = \begin{pmatrix} 9 & -3 & 2 & 8 \end{pmatrix}$, $w = \begin{pmatrix} 1 & 13 & -7 & 2 & 15 \end{pmatrix}$, and $c = 2$. Calculate the following:

1. $u - v$
2. cw
3. $u \cdot v$
4. $w \cdot v$

7.2 Linear Independence

Linear combinations: The vector $\mathbf{u}$ is a linear combination of the vectors $\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_k$ if

$$\mathbf{u} = c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_k\mathbf{v}_k$$

For example, $\begin{pmatrix} 9 & 13 & 17 \end{pmatrix}$ is a linear combination of the following three vectors: $\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$, $\begin{pmatrix} 2 & 3 & 4 \end{pmatrix}$, and $\begin{pmatrix} 3 & 4 & 5 \end{pmatrix}$. This is because $\begin{pmatrix} 9 & 13 & 17 \end{pmatrix} = (2)\begin{pmatrix} 1 & 2 & 3 \end{pmatrix} + (-1)\begin{pmatrix} 2 & 3 & 4 \end{pmatrix} + 3\begin{pmatrix} 3 & 4 & 5 \end{pmatrix}$

Linear independence: A set of vectors $\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_k$ is linearly independent if the only solution to the equation

$$c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_k\mathbf{v}_k = 0$$

is $c_1 = c_2 = \cdots = c_k = 0$. If another solution exists, the set of vectors is linearly dependent.

A set S of vectors is linearly dependent if and only if at least one of the vectors in S can be written as a linear combination of the other vectors in S.

Linear independence is only defined for sets of vectors with the same number of elements; any linearly independent set of vectors in n-space contains at most n vectors.

Since $\begin{pmatrix} 9 & 13 & 17 \end{pmatrix}$ is a linear combination of $\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$, $\begin{pmatrix} 2 & 3 & 4 \end{pmatrix}$, and $\begin{pmatrix} 3 & 4 & 5 \end{pmatrix}$, these 4 vectors constitute a linearly dependent set.

Example 7.2.

Linear indepencence

Are the following sets of vectors linearly independent?

1. $\begin{pmatrix} 2 & 3 & 1 \end{pmatrix}$ and $\begin{pmatrix} 4 & 6 & 1 \end{pmatrix}$
2. $\begin{pmatrix} 1 & 0 & 0 \end{pmatrix}$, $\begin{pmatrix} 0 & 5 & 0 \end{pmatrix}$, and $\begin{pmatrix} 10 & 10 & 0 \end{pmatrix}$

Exercise 7.2.

Linear indepencence

Are the following sets of vectors linearly independent?

1. $\mathbf{v}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \mathbf{v}_2 = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}, \mathbf{v}_3 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$
2. $\mathbf{v}_1 = \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix}, \mathbf{v}_2 = \begin{pmatrix} -4 \\ 6 \\ 5 \end{pmatrix}, \mathbf{v}_3 = \begin{pmatrix} -2 \\ 8 \\ 6 \end{pmatrix}$

7.3 Basics of Matrix Algebra

Matrix: A matrix is an array of real numbers arranged in m rows by n columns. The dimensionality of the matrix is defined as the number of rows by the number of columns, $m \times n$.

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

Note that you can think of vectors as special cases of matrices; a column vector of length k is a $k \times 1$ matrix, while a row vector of the same length is a $1 \times k$ matrix.

It's also useful to think of matrices as being made up of a collection of row or column vectors. For example,

$$\mathbf{A} = \begin{pmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_m \end{pmatrix}$$

Matrix Addition: Let $\mathbf{A}$ and $\mathbf{B}$ be two $m \times n$ matrices.

$$\mathbf{A} + B = \begin{pmatrix} a_{11} + b_{11} & a_{12} + b_{12} & \cdots & a_{1n} + b_{1n} \\ a_{21} + b_{21} & a_{22} + b_{22} & \cdots & a_{2n} + b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} + b_{m1} & a_{m2} + b_{m2} & \cdots & a_{mn} + b_{mn} \end{pmatrix}$$

Note that matrices **A** and **B** must have the same dimensionality, in which case they are **conformable for addition**.

Example 7.3.

Matrix addition

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}, \qquad \mathbf{B} = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 1 & 2 \end{pmatrix}$$

$\mathbf{A} + B =$

Scalar Multiplication: Given the scalar s, the scalar multiplication of $s\mathbf{A}$ is

$$s\mathbf{A} = s \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} = \begin{pmatrix} sa_{11} & sa_{12} & \cdots & sa_{1n} \\ sa_{21} & sa_{22} & \cdots & sa_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ sa_{m1} & sa_{m2} & \cdots & sa_{mn} \end{pmatrix}$$

Example 7.4.

Scalar Multiplication

$s = 2, \qquad \mathbf{A} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$

$s\mathbf{A} =$

Matrix Multiplication: If $\mathbf{A}$ is an $m \times k$ matrix and $\mathbf{B}$ is a $k \times n$ matrix, then their product $\mathbf{C} = AB$ is the $m \times n$ matrix where

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{ik}b_{kj}$$

Example 7.5.

Matrix multiplication

1. $\begin{pmatrix} a & b \\ c & d \\ e & f \end{pmatrix} \begin{pmatrix} A & B \\ C & D \end{pmatrix} =$

2. $\begin{pmatrix} 1 & 2 & -1 \\ 3 & 1 & 4 \end{pmatrix} \begin{pmatrix} -2 & 5 \\ 4 & -3 \\ 2 & 1 \end{pmatrix} =$

Note that the number of columns of the first matrix must equal the number of rows of the second matrix, in which case they are **conformable for multiplication**. The sizes of the matrices (including the resulting product) must be

$$(m \times k)(k \times n) = (m \times n)$$

Also note that if **AB** exists, **BA** exists only if $\dim(\mathbf{A}) = m \times n$ and $\dim(\mathbf{B}) = n \times m$.

This does not mean that $\mathbf{AB} = \mathbf{BA}$. $\mathbf{AB} = \mathbf{BA}$ is true only in special circumstances, like when **A** or **B** is an identity matrix or $\mathbf{A} = \mathbf{B}^{-1}$.

Laws of Matrix Algebra:

1. Associative: $(A + B) + C = A + (B + C)$
   $(AB)C = A(BC)$
2. Commutative: $\mathbf{A} + B = B + A$
3. Distributive: $\mathbf{A}(B + C) = AB + AC$
   $(A + B)C = AC + BC$

Commutative law for multiplication does not hold – the order of multiplication matters:

$$\mathbf{A}B \neq BA$$

For example,

$$\mathbf{A} = \begin{pmatrix} 1 & 2 \\ -1 & 3 \end{pmatrix}, \qquad \mathbf{B} = \begin{pmatrix} 2 & 1 \\ 0 & 1 \end{pmatrix}$$

$$\mathbf{A}B = \begin{pmatrix} 2 & 3 \\ -2 & 2 \end{pmatrix}, \qquad \mathbf{B}A = \begin{pmatrix} 1 & 7 \\ -1 & 3 \end{pmatrix}$$

Transpose: The transpose of the $m \times n$ matrix $\mathbf{A}$ is the $n \times m$ matrix $\mathbf{A}^T$ (also written $\mathbf{A}'$) obtained by interchanging the rows and columns of $\mathbf{A}$.

For example,

$$\mathbf{A} = \begin{pmatrix} 4 & -2 & 3 \\ 0 & 5 & -1 \end{pmatrix}, \qquad \mathbf{A}^T = \begin{pmatrix} 4 & 0 \\ -2 & 5 \\ 3 & -1 \end{pmatrix}$$

$$\mathbf{B} = \begin{pmatrix} 2 \\ -1 \\ 3 \end{pmatrix}, \qquad \mathbf{B}^T = \begin{pmatrix} 2 & -1 & 3 \end{pmatrix}$$

The following rules apply for transposed matrices:

1. $(\mathbf{A} + B)^T = \mathbf{A}^T + \mathbf{B}^T$
2. $(\mathbf{A}^T)^T = \mathbf{A}$
3. $(s\mathbf{A})^T = s\mathbf{A}^T$
4. $(\mathbf{A}B)^T = \mathbf{B}^T\mathbf{A}^T$; and by induction $(\mathbf{A}BC)^T = \mathbf{C}^T\mathbf{B}^T\mathbf{A}^T$

Example of $(\mathbf{A}B)^T = \mathbf{B}^T\mathbf{A}^T$:

$$\mathbf{A} = \begin{pmatrix} 1 & 3 & 2 \\ 2 & -1 & 3 \end{pmatrix}, \qquad \mathbf{B} = \begin{pmatrix} 0 & 1 \\ 2 & 2 \\ 3 & -1 \end{pmatrix}$$

$$(\mathbf{A}B)^T = \left[\begin{pmatrix} 1 & 3 & 2 \\ 2 & -1 & 3 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 2 & 2 \\ 3 & -1 \end{pmatrix} \right]^T = \begin{pmatrix} 12 & 7 \\ 5 & -3 \end{pmatrix}$$

$$\mathbf{B}^T\mathbf{A}^T = \begin{pmatrix} 0 & 2 & 3 \\ 1 & 2 & -1 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & -1 \\ 2 & 3 \end{pmatrix} = \begin{pmatrix} 12 & 7 \\ 5 & -3 \end{pmatrix}$$

Exercise 7.3.

Matrix Multiplication

Let

$$A = \begin{pmatrix} 2 & 0 & -1 & 1 \\ 1 & 2 & 0 & 1 \end{pmatrix}$$

$$B = \begin{pmatrix} 1 & 5 & -7 \\ 1 & 1 & 0 \\ 0 & -1 & 1 \\ 2 & 0 & 0 \end{pmatrix}$$

$$C = \begin{pmatrix} 3 & 2 & -1 \\ 0 & 4 & 6 \end{pmatrix}$$

Calculate the following:

1. $$AB$$
2. $$BA$$
3. $$(BC)^T$$
4. $$BC^T$$

7.4 Systems of Linear Equations

Linear Equation: $a_1x_1 + a_2x_2 + \cdots + a_nx_n = b$

a_i are parameters or coefficients. x_i are variables or unknowns.

Linear because only one variable per term and degree is at most 1.

We are often interested in solving linear systems like

$$\begin{array}{rcrcr} x & - & 3y & = & -3 \\ 2x & + & y & = & 8 \end{array}$$

More generally, we might have a system of m equations in n unknowns

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \cdots & + & a_{2n}x_n & = & b_2 \\ \vdots & & & & \vdots & & & \vdots & \\ a_{m1}x_1 & + & a_{m2}x_2 & + & \cdots & + & a_{mn}x_n & = & b_m \end{array}$$

A **solution** to a linear system of m equations in n unknowns is a set of n numbers $x_1, x_2, \cdots, x_n$ that satisfy each of the m equations.

Example: $x = 3$ and $y = 2$ is the solution to the above 2×2 linear system. If you graph the two lines, you will find that they intersect at $(3, 2)$.

Does a linear system have one, no, or multiple solutions? For a system of 2 equations with 2 unknowns (i.e., two lines): _
One solution: The lines intersect at exactly one point.

No solution: The lines are parallel.

Infinite solutions: The lines coincide.

Methods to solve linear systems:

1. Substitution
2. Elimination of variables
3. Matrix methods

Exercise 7.4.

Linear Equations

Provide a system of 2 equations with 2 unknowns that has

1. one solution
2. no solution
3. infinite solutions

7.5 Systems of Equations as Matrices

Matrices provide an easy and efficient way to represent linear systems such as

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \cdots & + & a_{2n}x_n & = & b_2 \\ \vdots & & & & \vdots & & & \vdots & \\ a_{m1}x_1 & + & a_{m2}x_2 & + & \cdots & + & a_{mn}x_n & = & b_m \end{array}$$

as

$$\mathbf{A}x = b$$

where

The $m \times n$ **coefficient matrix A** is an array of mn real numbers arranged in m rows by n columns:

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

The unknown quantities are represented by the vector $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$.

The right hand side of the linear system is represented by the vector $\mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}$.

Augmented Matrix: When we append $\mathbf{b}$ to the coefficient matrix $\mathbf{A}$, we get the augmented matrix $\widehat{\mathbf{A}} = [\mathbf{A}|b]$

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} & | & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & | & b_2 \\ \vdots & & \ddots & \vdots & | & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} & | & b_m \end{pmatrix}$$

Exercise 7.5. Create an augmented matrix that represent the following system of equations:

$$2x_1 - 7x_2 + 9x_3 - 4x_4 = 8$$
$$41x_2 + 9x_3 - 5x_6 = 11$$
$$x_1 - 15x_2 - 11x_5 = 9$$

7.6 Finding Solutions to Augmented Matrices and Systems of Equations

Row Echelon Form: Our goal is to translate our augmented matrix or system of equations into row echelon form. This will provide us with the values of the vector $\mathbf{x}$ which solve the system. We use the row operations to change coefficients in the lower triangle of the augmented matrix to 0. An augmented matrix of the form

$$\begin{pmatrix} a'_{11} & a'_{12} & a'_{13} & \cdots & a'_{1n} & | & b'_1 \\ 0 & a'_{22} & a'_{23} & \cdots & a'_{2n} & | & b'_2 \\ 0 & 0 & a'_{33} & \cdots & a'_{3n} & | & b'_3 \\ 0 & 0 & 0 & \ddots & \vdots & | & \vdots \\ 0 & 0 & 0 & 0 & a'_{mn} & | & b'_m \end{pmatrix}$$

is said to be in row echelon form — each row has more leading zeros than the row preceding it.

Reduced Row Echelon Form: We can go one step further and put the matrix into reduced row echelon form. Reduced row echelon form makes the value of $\mathbf{x}$ which solves the system very obvious. For a system of m equations in m unknowns, with no all-zero rows, the reduced row echelon form would be

$$\left(\begin{array}{ccccc|c} \boxed{1} & 0 & 0 & 0 & 0 & b_1^* \\ 0 & \boxed{1} & 0 & 0 & 0 & b_2^* \\ 0 & 0 & \boxed{1} & 0 & 0 & b_3^* \\ 0 & 0 & 0 & \ddots & 0 & \vdots \\ 0 & 0 & 0 & 0 & \boxed{1} & b_m^* \end{array}\right)$$

Gaussian and Gauss-Jordan elimination: We can conduct elementary row operations to get our augmented matrix into row echelon or reduced row echelon form. The methods of transforming a matrix or system into row echelon and reduced row echelon form are referred to as Gaussian elimination and Gauss-Jordan elimination, respectively.

Elementary Row Operations: To do Gaussian and Gauss-Jordan elimination, we use three basic operations to transform the augmented matrix into another augmented matrix that represents an equivalent linear system – equivalent in the sense that the same values of x_j solve both the original and transformed matrix/system:

Interchanging Rows: Suppose we have the augmented matrix

$$\widehat{\mathbf{A}} = \left(\begin{array}{cc|c} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \end{array}\right)$$

If we interchange the two rows, we get the augmented matrix

$$\left(\begin{array}{cc|c} a_{21} & a_{22} & b_2 \\ a_{11} & a_{12} & b_1 \end{array}\right)$$

which represents a linear system equivalent to that represented by matrix $\widehat{\mathbf{A}}$.

Multiplying by a Constant: If we multiply the second row of matrix $\widehat{\mathbf{A}}$ by a constant c, we get the augmented matrix

$$\left(\begin{array}{cc|c} a_{11} & a_{12} & b_1 \\ ca_{21} & ca_{22} & cb_2 \end{array}\right)$$

which represents a linear system equivalent to that represented by matrix $\widehat{\mathbf{A}}$.

Adding (subtracting) Rows: If we add (subtract) the first row of matrix $\widehat{\mathbf{A}}$ to the second, we obtain the augmented matrix

$$\left(\begin{array}{cc|c} a_{11} & a_{12} & b_1 \\ a_{11}+a_{21} & a_{12}+a_{22} & b_1+b_2 \end{array}\right)$$

which represents a linear system equivalent to that represented by matrix $\widehat{\mathbf{A}}$.

Example 7.6.

Solving systems of equations

Solve the following system of equations by using elementary row operations:

$$\begin{array}{rcrcr} x & - & 3y & = & -3 \\ 2x & + & y & = & 8 \end{array}$$

Exercise 7.6.

Solving Systems of Equations

Put the following system of equations into augmented matrix form. Then, using Gaussian or Gauss-Jordan elimination, solve the system of equations by putting the matrix into row echelon or reduced row echelon form.

$$1. \begin{cases} x + y + 2z = 2 \\ 3x - 2y + z = 1 \\ y - z = 3 \end{cases}$$

$$2. \begin{cases} 2x + 3y - z = -8 \\ x + 2y - z = 12 \\ -x - 4y + z = -6 \end{cases}$$

7.7 Rank — and Whether a System Has One, Infinite, or No Solutions

To determine how many solutions exist, we can use information about (1) the number of equations m, (2) the number of unknowns n, and (3) the **rank** of the matrix representing the linear system.

Rank: The maximum number of linearly independent row or column vectors in the matrix. This is equivalent to the number of nonzero rows of a matrix in row echelon form. For any matrix $\mathbf{A}$, the row rank always equals column rank, and we refer to this number as the rank of $\mathbf{A}$.

For example

$$\begin{pmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{pmatrix}$$

Rank = 3

$$\begin{pmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 0 \end{pmatrix}$$

Rank = 2

Exercise 7.7.

Rank of Matrices

Find the rank of each matrix below:

(Hint: transform the matrices into row echelon form. Remember that the number of nonzero rows of a matrix in row echelon form is the rank of that matrix)

1.
$$\begin{pmatrix} 1 & 1 & 2 \\ 2 & 1 & 3 \\ 1 & 2 & 3 \end{pmatrix}$$

2.
$$\begin{pmatrix} 1 & 3 & 3 & -3 & 3 \\ 1 & 3 & 1 & 1 & 3 \\ 1 & 3 & 2 & -1 & -2 \\ 1 & 3 & 0 & 3 & -2 \end{pmatrix}$$

7.8 The Inverse of a Matrix

Identity Matrix: The $n \times n$ identity matrix $\mathbf{I}_n$ is the matrix whose diagonal elements are 1 and all off-diagonal elements are 0. Examples:

$$\mathbf{I}_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad \mathbf{I}_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Inverse Matrix: An $n \times n$ matrix $\mathbf{A}$ is **nonsingular** or **invertible** if there exists an $n \times n$ matrix $\mathbf{A}^{-1}$ such that

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}_n$$

where $\mathbf{A}^{-1}$ is the inverse of $\mathbf{A}$. If there is no such $\mathbf{A}^{-1}$, then $\mathbf{A}$ is singular or not invertible.

Example: Let

$$\mathbf{A} = \begin{pmatrix} 2 & 3 \\ 2 & 2 \end{pmatrix}, \qquad \mathbf{B} = \begin{pmatrix} -1 & \frac{3}{2} \\ 1 & -1 \end{pmatrix}$$

Since

$$\mathbf{AB} = \mathbf{BA} = \mathbf{I}_n$$

we conclude that $\mathbf{B}$ is the inverse, $\mathbf{A}^{-1}$, of $\mathbf{A}$ and that $\mathbf{A}$ is nonsingular.

Properties of the Inverse:

- If the inverse exists, it is unique.
- If $\mathbf{A}$ is nonsingular, then $\mathbf{A}^{-1}$ is nonsingular.
- $(\mathbf{A}^{-1})^{-1} = \mathbf{A}$
- If $\mathbf{A}$ and $\mathbf{B}$ are nonsingular, then $\mathbf{AB}$ is nonsingular
- $(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$
- If $\mathbf{A}$ is nonsingular, then $(\mathbf{A}^T)^{-1} = (\mathbf{A}^{-1})^T$

Procedure to Find $\mathbf{A}^{-1}$: We know that if $\mathbf{B}$ is the inverse of $\mathbf{A}$, then

$$\mathbf{AB} = \mathbf{BA} = \mathbf{I}_n$$

Looking only at the first and last parts of this

$$\mathbf{AB} = \mathbf{I}_n$$

Solving for $\mathbf{B}$ is equivalent to solving for n linear systems, where each column of $\mathbf{B}$ is solved for the corresponding column in $\mathbf{I}_n$. We can solve the systems simultaneously by augmenting $\mathbf{A}$ with $\mathbf{I}_n$ and performing Gauss-Jordan elimination on $\mathbf{A}$. If Gauss-Jordan elimination on $[\mathbf{A}|\mathbf{I}_n]$ results in $[\mathbf{I}_n|\mathbf{B}]$, then $\mathbf{B}$ is the inverse of $\mathbf{A}$. Otherwise, $\mathbf{A}$ is singular.

To summarize: To calculate the inverse of $\mathbf{A}$

1. Form the augmented matrix $[\mathbf{A}|\mathbf{I}_n]$
2. Using elementary row operations, transform the augmented matrix to reduced row echelon form.
3. The result of step 2 is an augmented matrix $[\mathbf{C}|\mathbf{B}]$.
    a. If $\mathbf{C} = \mathbf{I}_n$, then $\mathbf{B} = \mathbf{A}^{-1}$.
    b. If $\mathbf{C} \neq \mathbf{I}_n$, then $\mathbf{C}$ has a row of zeros. This means $\mathbf{A}$ is singular and $\mathbf{A}^{-1}$ does not exist.

Example 7.7.

Matrix Inverse

Find the inverse of the following matricies:

1. $\mathbf{A} = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 2 & 3 \\ 5 & 5 & 1 \end{pmatrix}$

Exercise 7.8.

Matrix Inverse

Find the inverse of the following matrix:

1. $\mathbf{A} = \begin{pmatrix} 1 & 0 & 4 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

7.9 Linear Systems and Inverses

Let's return to the matrix representation of a linear system

$$\mathbf{Ax} = \mathbf{b}$$

If $\mathbf{A}$ is an $n \times n$ matrix,then $\mathbf{Ax} = \mathbf{b}$ is a system of n equations in n unknowns. Suppose $\mathbf{A}$ is nonsingular. Then $\mathbf{A}^{-1}$ exists. To solve this system, we can multiply each side by $\mathbf{A}^{-1}$ and reduce it as follows:

$$\begin{aligned} \mathbf{A}^{-1}(\mathbf{Ax}) &= \mathbf{A}^{-1}\mathbf{b} \\ (\mathbf{A}^{-1}\mathbf{A})\mathbf{x} &= \mathbf{A}^{-1}\mathbf{b} \\ \mathbf{I}_n\mathbf{x} &= \mathbf{A}^{-1}\mathbf{b} \\ \mathbf{x} &= \mathbf{A}^{-1}\mathbf{b} \end{aligned}$$

Hence, given $\mathbf{A}$ and $\mathbf{b}$ and given that $\mathbf{A}$ is nonsingular, then $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ is a unique solution to this system.

Exercise 7.9.

Solve linear system using inverses

Use the inverse matrix to solve the following linear system:

$$\begin{aligned} -3x + 4y &= 5 \\ 2x - y &= -10 \end{aligned}$$

Hint: the linear system above can be written in the matrix form

$\mathbf{Az} = \mathbf{b}$

given $\mathbf{A} = \begin{pmatrix} -3 & 4 \\ 2 & -1 \end{pmatrix}$

$\mathbf{z} = \begin{pmatrix} x \\ y \end{pmatrix}$ and $\mathbf{b} = \begin{pmatrix} 5 \\ -10 \end{pmatrix}$

7.10 Determinants

Singularity: Determinants can be used to determine whether a square matrix is nonsingular.

A square matrix is nonsingular if and only if its determinant is not zero.

Determinant of a 1×1 matrix, $\mathbf{A}$, equals a_{11}

Determinant of a 2×2 matrix, $\mathbf{A}$, $\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}$:

$$\begin{aligned} \det(\mathbf{A}) &= |\mathbf{A}| \\ &= a_{11}|a_{22}| - a_{12}|a_{21}| \\ &= a_{11}a_{22} - a_{12}a_{21} \end{aligned}$$

We can extend the second to last equation above to get the definition of the determinant of a 3×3 matrix:

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11}\begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12}\begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13}\begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix}$$
$$= a_{11}(a_{22}a_{33} - a_{23}a_{32}) - a_{12}(a_{21}a_{33} - a_{23}a_{31}) + a_{13}(a_{21}a_{32} - a_{22}a_{31})$$

Let's extend this now to any $n \times n$ matrix. Let's define $\mathbf{A}_{ij}$ as the $(n-1) \times (n-1)$ submatrix of $\mathbf{A}$ obtained by deleting row i and column j. Let the (i, j)th **minor** of $\mathbf{A}$ be the determinant of $\mathbf{A}_{ij}$:

$$M_{ij} = |\mathbf{A}_{ij}|$$

Then for any $n \times n$ matrix $\mathbf{A}$

$$|\mathbf{A}| = a_{11}M_{11} - a_{12}M_{12} + \cdots + (-1)^{n+1}a_{1n}M_{1n}$$

For example, in figuring out whether the following matrix has an inverse?

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 2 & 3 \\ 5 & 5 & 1 \end{pmatrix}$$

1. Calculate its determinant.

$$\begin{aligned} &= 1(2-15) - 1(0-15) + 1(0-10) \\ &= -13 + 15 - 10 \\ &= -8 \end{aligned}$$

2. Since $|\mathbf{A}| \neq 0$, we conclude that $\mathbf{A}$ has an inverse.

8 Determinants

Determine whether the following matrices are nonsingular:

1. $\begin{pmatrix} 1 & 0 & 1 \\ 2 & 1 & 2 \\ 1 & 0 & -1 \end{pmatrix}$

2. $\begin{pmatrix} 2 & 1 & 2 \\ 1 & 0 & 1 \\ 4 & 1 & 4 \end{pmatrix}$

8.1 Getting Inverse of a Matrix using its Determinant

Thus far, we have a number of algorithms to

1. Find the solution of a linear system,
2. Find the inverse of a matrix

but these remain just that — algorithms. At this point, we have no way of telling how the solutions x_j change as the parameters a_{ij} and b_i change, except by changing the values and "rerunning" the algorithms.

With determinants, we can provide an explicit formula for the inverse and therefore provide an explicit formula for the solution of an $n \times n$ linear system.

Hence, we can examine how changes in the parameters and b_i affect the solutions x_j.

Determinant Formula for the Inverse of a 2×2:

The determinant of a 2×2 matrix $\mathbf{A}$ $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ is defined as:

$$\frac{1}{\det(\mathbf{A})} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

For example, Let's calculate the inverse of matrix A from Exercise @ref(exr:invlinsys) using the determinant formula.

Recall,

$$A = \begin{pmatrix} -3 & 4 \\ 2 & -1 \end{pmatrix}$$

$$\det(\mathbf{A}) = (-3)(-1) - (4)(2) = 3 - 8 = -5$$

$$\frac{1}{\det(\mathbf{A})} \begin{pmatrix} -1 & -4 \\ -2 & -3 \end{pmatrix}$$

$$\frac{1}{-5} \begin{pmatrix} -1 & -4 \\ -2 & -3 \end{pmatrix}$$

$$\begin{pmatrix} \frac{1}{5} & \frac{4}{5} \\ \frac{2}{5} & \frac{3}{5} \end{pmatrix}$$

9 Calculate Inverse using Determinant Formula

Calculate the inverse of A

$$A = \begin{pmatrix} 3 & 5 \\ -7 & 2 \end{pmatrix}$$

10 Programming: Orientation and Reading in Data[1]

Motivation: Data and You

The modal social science project starts by importing existing datasets. Datasets come in all shapes and sizes. As you search for new data you may encounter dozens of file extensions – csv, xlsx, dta, sav, por, Rdata, Rds, txt, xml, json, shp … the list continues. Although these files can often be cumbersome, its a good to be able to find a way to encounter any file that your research may call for.

Reviewing data import will allow us to get on the same page on how computer systems work.

Where are we? Where are we headed?

Today we'll cover:

- What's what in RStudio
- What R is, at a high level
- How to read in data
- Comment on coding style on the way

Check your understanding

- What is the difference between a file and a folder?
- In the RStudio windows, what is the difference between the "Source" Pane and the "Console"? What is a "code chunk"?
- How do you read a R help page? What is the `Usage` section, the `Values` section, and the `Examples` section?
- What use is the "Environment" Pane?
- How would you read in a spreadsheet in R?
- How would you figure out what variables are in the data? size of the data?
- How would you read in a `csv` file, a `dta` file, a `sav` file?

[1] Special thanks to Shiro Kuriwaki for developing the original version of this tutorial

10.1 General Orientation

1. RStudio is a **GUI** and an IDE for the programming language R. A Graphical User Interface allows users to interface with the software (in this case R) using graphical aids like buttons and tabs. Often we don't think of GUIs because to most computer users, everything is a GUI (like Microsoft Word or your "Control Panel"), but it's always there! A Integrated Development Environment just says that the software to interface with R comes with useful useful bells and whistles to give you shortcuts.

The **Console** is kind of a the core window through which you see your GUI actually operating through R. It's not graphical so might not be as intuitive. But all your results, commands, errors, warnings.. you see them in here. A console tells you what's going on now.

2. Via the GUI, you the analyst needs to send instructions, or **commands**, to the R application. The verb for this is "run" or "execute" the command. Computer programs ask users to provide instructions in very specific formats. While a English-speaking human can understand a sentence with a few typos in it by filling in the blanks, the same typo or misplaced character would halt a computer program. Each program has its own requirements for how commands should be typed; after all, each of these is its own language. We refer to the way a program needs its commands to be formatted as its **syntax**.

3. Theoretically, one could do all their work by typing in commands into the Console. But that would be a lot of work, because you'd have to give instructions each time you start your data analysis. Moreover, you'll have no record of what you did. That's why you need a **script**. This is a type of **code**. It can be referred to as a **source** because that is the source of your commands. Source is also used as a verb; "source the script" just means execute it. RStudio doesn't start out with a script, so you can make one from "File > New" or the New file icon.

4. You can also open scripts that are in folders in your computer. A script is a type of File. Find your Files in the bottom-right "Files" pane.

To load a dataset, you need to specify where that file is. Computer files (data, documents, programs) are organized hiearchically, like a branching tree. Folders can contain files, and also other folders. The GUI toolbar makes this lineaer and hiearchical relationship apparent. When we turn to locate the file in our commands, we need another set of syntax. Importantly, denote the hierarchy of a folder by the `/` (slash) symbol. `data/input/2018-08` indicates the `2018-08` folder, which is included in the `input` folder, which is in turn included in the `data` folder.

Files (but not folders) have "file extensions" which you are probably familiar with already: `.docx`, `.pdf`, and `.pdf`. The file extensions you will see in a stats or quantitative social science class are:

- `.pdf`: PDF, a convenient format to view documents and slides in, regardless of Mac/Windows.
- `.csv`: A comma separated values file
- `.xlsx`: Microsoft Excel file
- `.dta`: Stata data
- `.sav`: SPSS data
- `.R`: R code (script)
- `.Rmd`: Rmarkdown code (text + code)
- `.do`: Stata code (script)

5. In R, there are two main types of scripts. A classic `.R` file and a `.Rmd` file (for Rmarkdown). A .R file is just lines and lines of R code that is meant to be inserted right into the Console. A .Rmd tries to weave code and English together, to make it easier for users to create reports that interact with data and intersperse R code with explanation.

Rmarkdown facilitates is the use of **code chunks**, which are used here. These start and end with three back-ticks. In the beginning, we can add options in curly braces (`{}`). Specifying `r` in the beginning tells to render it as R code. Options like `echo = TRUE` switch between showing the code that was executed or not; `eval = TRUE` switch between evaluating the code. More about Rmarkdown in later sections. For example, this code chunk would evaluate `1 + 1` and show its output when compiled, but not display the code that was executed.

10.2 But what is R

R is a programming language primarily used for statistical computing. It's free, open source and has an extensive community that is constantly developing new tools and packages that extend its functionality in a lot of different ways (for example, the `tidyverse` project).

One feature of many programming languages is that they allow for "object-oriented" programming. In an object-oriented programming paradigm, we work with "objects" – some sort of structure that exists in the computer memory – that contains attributes and on which we can execute code (methods). R's object-oriented support has some interesting quirks compared to other languages like C or Python, but

Everything in R is an object, including its most basic *data types*. R has six basic data types:

- character
- numeric (real or decimal)
- integer
- logical

- complex

These *data types* make up the basic data structures of R

- atomic vectors
- lists
- matrix
- data frame
- factors

Beyond that, many R routines

10.3 The Computer and You: Giving Instructions

We'll do the Peanut Butter and Jelly Exercise in class as an introduction to programming for those who are new.

Assignment: Take 5 minutes to write down on a piece of paper, how to make a peanut butter and jelly sandwich. Be as concise and unambiguous as possible so that a robot (who doesn't know what a PBJ is) would understand. You can assume that there will be loaf of sliced bread, a jar of jelly, a jar of peanut butter, and a knife.

Simpler assignment: Say we just want a robot to be able to tell us if we have enough ingredients to make a peanut butter and jelly sandwich. Write down instructions so that if told how many slices of bread, servings of peanut butter, and servings of jelly you have, the robot can tell you if you can make a PBJ.

Now, translate the simpler assignment into R code using the code below as a starting point:

```
n_bread <- 8
n_pb <- 3
n_jelly <- 9

# write instructions in R here
```

10.4 Base-R vs. tidyverse

One last thing before we jump into data. Many things in R and other open source packages have competing standards. A lecture on a technique inevitably biases one standard over

another. Right now among R users in this area, there are two families of functions: base-R and tidyverse. R instructors thus face a dilemma about which to teach primarily.[2]

In this prefresher, we try our best to choose the one that is most useful to the modal task of social science researchers, and make use of the tidyverse functions in most applications. but feel free to suggest changes to us or to the booklet.

Although you do not need to choose one over the other, for beginners it is confusing what is a tidyverse function and what is not. Many of the tidyverse *packages* are covered in this 2017 graphic below, and the cheat-sheets that other programmers have written: https://www.rstudio.com/resources/cheatsheets/

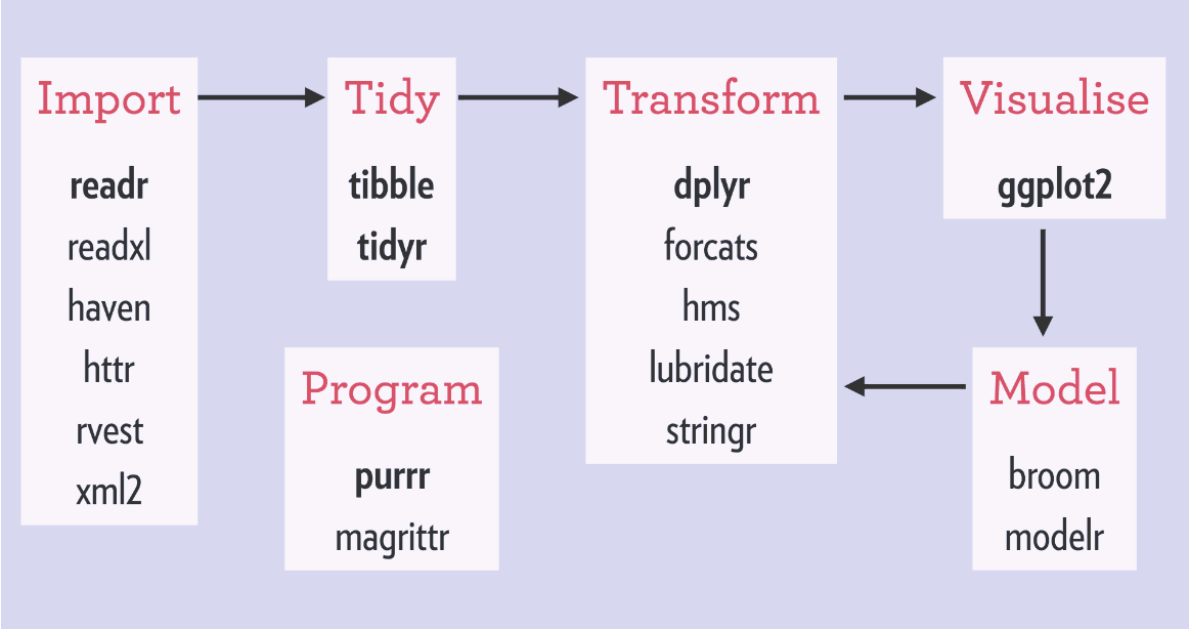


Figure 10.1: Names of Packages in the tidyverse Family

The following side-by-side comparison of commands for a particular function compares some tidyverse and non-tidyverse functions (which we refer to loosely as base-R). This list is not meant to be comprehensive and more to give you a quick rule of thumb.

Dataframe subsetting

[2]See for example this community discussion: https://community.rstudio.com/t/base-r-and-the-tidyverse/2965/17

In order to …	in tidyverse:	in base-R:
Count each category	`count(df, var)`	`table(df$var)`
Filter rows by condition	`filter(df, var == "Female")`	`df[df$var == "Female", ]` or `subset(df, var == "Female")`
Extract columns	`select(df, var1, var2)`	`df[, c("var1", "var2")]`
Extract a single column as a vector	`pull(df, var)`	`df[["var"]]` or `df[, "var"]`
Combine rows	`bind_rows()`	`rbind()`
Combine columns	`bind_cols()`	`cbind()`
Create a dataframe	`tibble(x = vec1, y = vec2)`	`data.frame(x = vec1, y = vec2)`
Turn a dataframe into a tidyverse dataframe	`tbl_df(df)`	

Remember that tidyverse applies to *dataframes* only, not vectors. For subsetting vectors, use the base-R functions with the square brackets.

Read data

Some non-tidyverse functions are not quite "base-R" but have similar relationships to tidyverse. For these, we recommend using the *tidyverse* functions as a general rule due to their common format, simplicity, and scalability.

In order to …	in tidyverse:	in base-R:
Read a Excel file	`read_excel()`	`read.xlsx()`
Read a csv	`read_csv()`	`read.csv()`
Read a Stata file	`read_dta()`	`read.dta()`
Substitute strings	`str_replace()`	`gsub()`
Return matching strings	`str_subset()`	`grep(., value = TRUE)`
Merge `data1` and `data2` on variables `x1` and `x2`	`left_join(data1, data2, by = c("x1", "x2"))`	`merge(data1, data2, by.x = "x1", by.y = "x2", all.x = TRUE)`

Visualization

Plotting by ggplot2 (from your tutorials) is also a tidyverse family.

In order to …	in tidyverse:	in base-R:
Make a scatter plot	`ggplot(data, aes(x, y)) + geom_point()`	`plot(data$x, data$y)`
Make a line plot	`ggplot(data, aes(x, y)) + geom_line()`	`plot(data$x, data$y, type = "l")`
Make a histogram	`ggplot(data, aes(x, y)) + geom_histogram()`	`hist(data$x, data$y)`

10.5 A is for Athens

For our first dataset, let's try reading in a dataset on the Ancient Greek world. Political Theorists and Political Historians study the domestic systems, international wars, cultures and writing of this era to understand the first instance of democracy, the rise and overturning of tyranny, and the legacies of political institutions.

This POLIS dataset was generously provided by Professor Josiah Ober of Stanford University. This dataset includes information on city states in the Ancient Greek world, parts of it collected by careful work by historians and archaeologists. It is part of his recent books on Greece (Ober 2015), "The Rise and Fall of Classical Greece"[3] and Institutions in Ancient Athens (Ober 2010) , "Democracy and Knowledge: Innovation and Learning in Classical Athens."[4]

10.5.1 Locating the Data

What files do we have in the `data/input` folder?

```
data/input/CES Guide 2022.pdf
data/input/Nunn_Wantchekon_sample.dta
data/input/ces22_subset.dta
data/input/gapminder_wide.tab
data/input/justices_court-median.csv
data/input/sample_mid.csv
data/input/upshot-siena-polls.csv
data/input/usc2010_001percent.csv
data/input/Nunn_Wantchekon_AER_2011.dta
data/input/acs2015_1percent.csv
data/input/gapminder_wide.Rds
data/input/german_credit.sav
data/input/ober_2018.xlsx
data/input/sample_polity.csv
data/input/usc2010_001percent.Rds
```

A typical file format is Microsoft Excel. Although this is not usually the best format for R because of its highly formatted structure as opposed to plain text, recent packages have made this fairly easy.

[3] Ober, Josiah (2015). *The Rise and Fall of Classical Greece.* Princeton University Press.
[4] Ober, Josiah (2010). *Democracy and Knowledge: Innovation and Learning in Classical Athens.* Princeton University Press.

10.5.2 Reading in Data

In Rstudio, a good way to start is to use the GUI and the Import tool. Once you click a file, an option to "Import Dataset" comes up. RStudio picks the right function for you, and you can copy that code, but it's important to eventually be able to write that code yourself.

For the first time using an outside package, you first need to install it.

```
install.packages("readxl")
```

After that, you don't need to install it again. But you **do** need to load it each time.

```
library(readxl)
```

The package `readxl` has a website: https://readxl.tidyverse.org/. Other packages are not as user-friendly, but they have a help page with a table of contents of all their functions.

```
help(package = readxl)
```

From the help page, we see that `read_excel()` is the function that we want to use.

Let's try it.

```
library(readxl)
ober <- read_excel("data/input/ober_2018.xlsx")
```

Review: what does the `/` mean? Why do we need the `data` term first? Does the argument need to be in quotes?

10.5.3 Inspecting

For almost any dataset, you usually want to do a couple of standard checks first to understand what you loaded.

```
ober
```

```
# A tibble: 1,035 x 10
   polis_number Name  Latitude Longitude Hellenicity  Fame Size  Colonies Regime
          <dbl> <chr>    <dbl>     <dbl> <chr>       <dbl> <chr>    <dbl> <chr>
 1            1 Alal~     42.1      9.51 most Greek   1.12 100-~        0 <NA>
 2            2 Empo~     42.1      3.11 most barba~  2.12 25-1~        0 <NA>
```

```
 3            3 Mass~     43.3      5.38 most Greek   4    25-1~        2 no ev~
 4            4 Rhode     42.3      3.17 most Greek   0.87 <NA>         0 <NA>
 5            5 Abak~     38.1     15.1  most barba~  1    <NA>         0 <NA>
 6            6 Adra~     37.7     14.8  most Greek   1    <NA>         0 <NA>
 7            7 Agyr~     37.7     14.5  most Greek   1.25 <NA>         0 no ev~
 8            8 Aitna     38.2     15.6  most Greek   3.25 200-~        1 no ev~
 9            9 Akra~     37.3     13.6  most Greek   6.37 500 ~        0 evide~
10           10 Akrai     37.1     14.9  most Greek   1.25 <NA>         0 <NA>
# i 1,025 more rows
# i 1 more variable: Delian <chr>
```

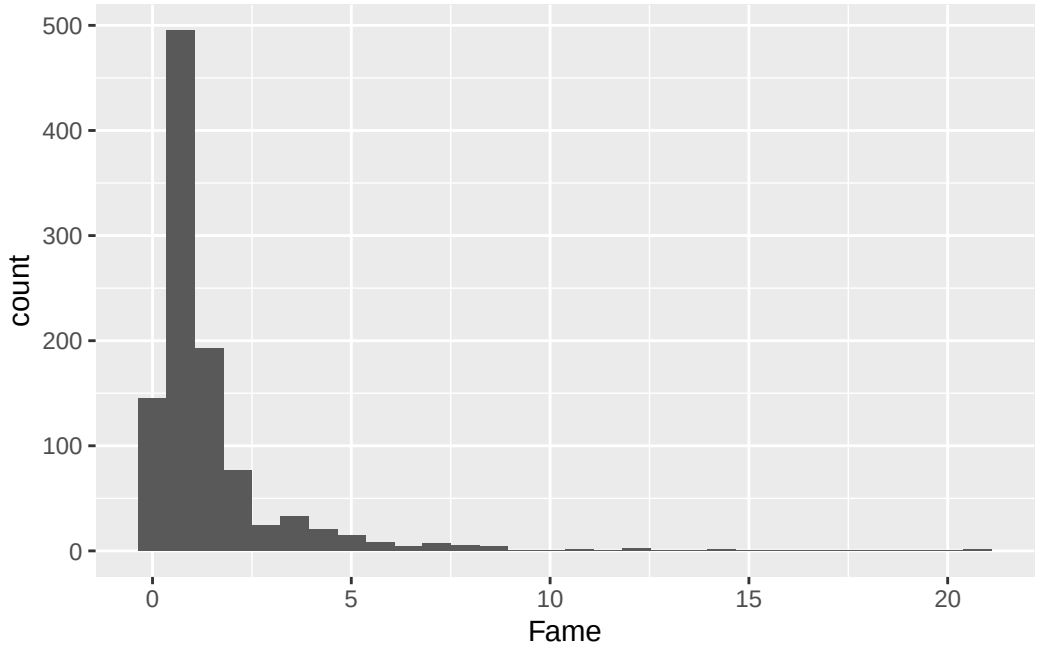
```
dim(ober)
```

```
[1] 1035   10
```

Graphics are useful for grasping your data - we will cover them more deeply later on.

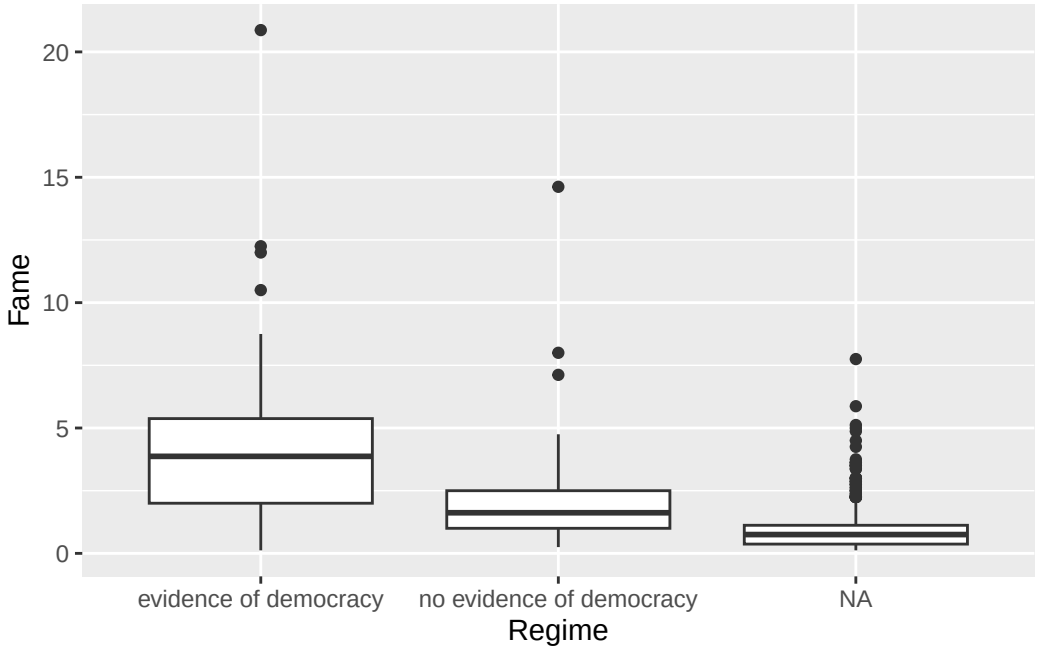
```
ggplot(ober, aes(x = Fame)) + geom_histogram()
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



What about the distribution of fame by regime?

```
ggplot(ober, aes(y = Fame, x = Regime, group = Regime)) +
  geom_boxplot()
```



What do the 1's, 2's, and 3's stand for?

10.5.4 Finding observations

These `tidyverse` commands from the `dplyr` package are newer and not built-in, but they are one of the increasingly more popular ways to wrangle data.

- 80 percent of your data wrangling needs might be doable with these basic `dplyr` functions: `select`, `mutate`, `group_by`, `summarize`, and `arrange`.
- These verbs roughly correspond to the same commands in SQL, another important language in data science.
- The `%>%` symbol is a pipe. It takes the thing on the left side and pipes it down to the function on the right side. We could have done `count(cen10, race)` as `cen10 %>% count(race)`. That means take `cen10` and pass it on to the function `count`, which will count observations by race and return a collapsed dataset with the categories in its own variable and their respective counts in `n`.

Exercises

1

What is the Fame value of Delphoi?

```
# Enter here
```

2

Find the polis with the top 10 Fame values.

```
# Enter here
```

3

Make a scatterplot with the number of colonies on the x-axis and Fame on the y-axis.

```
# Enter here
```

4

Find the correct function to read the following datasets into your R instance.

- `data/input/acs2015_1percent.csv`: A one percent sample of the American Community Survey
- `data/input/gapminder_wide.tab`: Country-level wealth and health from Gapminder[5]
- `data/input/gapminder_wide.Rds`: A Rds version of the Gapminder (What is a Rds file? What's the difference?)
- `data/input/Nunn_Wantchekon_sample.dta`: A sample from the Afrobarometer survey (which we'll explore tomorrow). `.dta` is a Stata format.
- `data/input/german_credit.sav`: A hypothetical dataset on consumer credit. `.sav` is a SPSS format.

Our Recommendations: Look at the packages `haven` and `readr`

```
# Enter here, perhaps making a chunk for each file.
```

[5]Formatted and taken from https://doi.org/10.7910/DVN/GJQNEQ

5

Read Ober's codebook and find a variable that you think is interesting. Check the distribution of that variable in your data, get a couple of statistics, and summarize it in English.

```
# Enter here
```

11 Programming: Manipulating Vectors and Matrices[1]

Motivation

Nunn and Wantchekon (2011) – "The Slave Trade and the Origins of Mistrust in Africa"[2] – argues that across African countries, the distrust of co-ethnics fueled by the slave trade has had long-lasting effects on modern day trust in these territories. They argued that the slave trade created distrust in these societies in part because as some African groups were employed by European traders to capture their neighbors and bring them to the slave ships.

Nunn and Wantchekon use a variety of statistical tools to make their case (adding controls, ordered logit, instrumental variables, falsification tests, causal mechanisms), many of which will be covered in future courses. In this module we will only touch on their first set of analysis that use Ordinary Least Squares (OLS). OLS is likely the most common application of linear algebra in the social sciences. We will cover some linear algebra, matrix manipulation, and vector manipulation from this data.

Where are we? Where are we headed?

Up till now, you should have covered:

- R basic programming
- Data Import
- Statistical Summaries.

Today we'll cover

- Matrices & Dataframes in R
- Manipulating variables
- And other `R` tips

[1] Special thanks to Shiro Kuriwaki and Yon Soo Park for developing the original module

[2] Nunn, Nathan, and Leonard Wantchekon. 2011. "The Slave Trade and the Origins of Mistrust in Africa." American Economic Review 101(7): 3221–52.

11.1 Read Data

```
library(haven)
nunn_full <- read_dta("data/input/Nunn_Wantchekon_AER_2011.dta")
```

Nunn and Wantchekon's main dataset has more than 20,000 observations. Each observation is a respondent from the Afrobarometer survey.

```
head(nunn_full)
```

```
# A tibble: 6 x 59
  respno  ethnicity murdock_name isocode region    district townvill location_id
  <chr>   <chr>     <chr>        <chr>   <chr>     <chr>    <chr>          <dbl>
1 BEN0001 fon       FON          BEN     atlnatiq~ KPOMASSE TOKPA-D~          30
2 BEN0002 fon       FON          BEN     atlnatiq~ KPOMASSE TOKPA-D~          30
3 BEN0003 fon       FON          BEN     atlnatiq~ OUIDAH   3ARROND           31
4 BEN0004 fon       FON          BEN     atlnatiq~ OUIDAH   3ARROND           31
5 BEN0005 fon       FON          BEN     atlnatiq~ OUIDAH   PAHOU             32
6 BEN0006 fon       FON          BEN     atlnatiq~ OUIDAH   PAHOU             32
# i 51 more variables: trust_relatives <dbl>, trust_neighbors <dbl>,
#   intra_group_trust <dbl>, inter_group_trust <dbl>,
#   trust_local_council <dbl>, ln_export_area <dbl>, export_area <dbl>,
#   export_pop <dbl>, ln_export_pop <dbl>, age <dbl>, age2 <dbl>, male <dbl>,
#   urban_dum <dbl>, occupation <dbl>, religion <dbl>, living_conditions <dbl>,
#   education <dbl>, near_dist <dbl>, distsea <dbl>, loc_murdock_name <chr>,
#   loc_ln_export_area <dbl>, local_council_performance <dbl>, ...
```

```
colnames(nunn_full)
```

```
 [1] "respno"                    "ethnicity"
 [3] "murdock_name"              "isocode"
 [5] "region"                    "district"
 [7] "townvill"                  "location_id"
 [9] "trust_relatives"           "trust_neighbors"
[11] "intra_group_trust"         "inter_group_trust"
[13] "trust_local_council"       "ln_export_area"
[15] "export_area"               "export_pop"
[17] "ln_export_pop"             "age"
[19] "age2"                      "male"
```

```
[21] "urban_dum"                          "occupation"
[23] "religion"                           "living_conditions"
[25] "education"                          "near_dist"
[27] "distsea"                            "loc_murdock_name"
[29] "loc_ln_export_area"                 "local_council_performance"
[31] "council_listen"                     "corrupt_local_council"
[33] "school_present"                     "electricity_present"
[35] "piped_water_present"                "sewage_present"
[37] "health_clinic_present"              "district_ethnic_frac"
[39] "frac_ethnicity_in_district"         "townvill_nonethnic_mean_exports"
[41] "district_nonethnic_mean_exports"    "region_nonethnic_mean_exports"
[43] "country_nonethnic_mean_exports"     "murdock_centr_dist_coast"
[45] "centroid_lat"                       "centroid_long"
[47] "explorer_contact"                   "railway_contact"
[49] "dist_Saharan_node"                  "dist_Saharan_line"
[51] "malaria_ecology"                    "v30"
[53] "v33"                                "fishing"
[55] "exports"                            "ln_exports"
[57] "total_missions_area"                "ln_init_pop_density"
[59] "cities_1400_dum"
```

First, let's consider a small subset of this dataset.

```
nunn <- read_dta("data/input/Nunn_Wantchekon_sample.dta")
```

```
nunn
```

```
# A tibble: 10 x 5
   trust_neighbors exports ln_exports export_area ln_export_area
             <dbl>   <dbl>      <dbl>       <dbl>          <dbl>
 1               3   0.388      0.328     0.00407        0.00406
 2               3   0.631      0.489     0.0971         0.0926
 3               3   0.994      0.690     0.0125         0.0124
 4               0 183.         5.21      1.82           1.04
 5               3   0          0         0              0
 6               2   0          0         0              0
 7               2 666.         6.50     14.0            2.71
 8               0   0.348      0.298     0.00608        0.00606
 9               3   0.435      0.361     0.0383         0.0376
10               3   0          0         0              0
```

11.2 data.frame vs. matricies

This is a `data.frame` object.

```
class(nunn)
```

```
[1] "tbl_df"     "tbl"        "data.frame"
```

But it can be also consider a matrix in the linear algebra sense. What are the dimensions of this matrix?

```
nrow(nunn)
```

```
[1] 10
```

`data.frame`s and matrices have much overlap in `R`, but to explicitly treat an object as a matrix, you'd need to coerce its class. Let's call this matrix `X`.

```
X <- as.matrix(nunn)
```

What is the difference between a `data.frame` and a matrix? A `data.frame` can have columns that are of different types, whereas — in a matrix — all columns must be of the same type (usually either "numeric" or "character").

You can think of data frames maybe as matrices-plus, because a column can take on characters as well as numbers. As we just saw, this is often useful for real data analyses.

Another way to think about data frames is that it is a type of list. Try the `str()` code below and notice how it is organized in slots. Each slot is a vector. They can be vectors of numbers or characters.

```
# enter this on your console
str(cen10)
```

11.3 Handling matricies in R

You can easily transpose a matrix

```
X
```

```
      trust_neighbors     exports ln_exports  export_area ln_export_area
 [1,]               3   0.3883497  0.3281158  0.004067405    0.004059155
 [2,]               3   0.6311236  0.4892691  0.097059444    0.092633367
 [3,]               3   0.9941893  0.6902376  0.012524694    0.012446908
 [4,]               0 182.5891266  5.2127004  1.824284434    1.038255095
 [5,]               3   0.0000000  0.0000000  0.000000000    0.000000000
 [6,]               2   0.0000000  0.0000000  0.000000000    0.000000000
 [7,]               2 665.9652100  6.5027380 13.975566864    2.706419945
 [8,]               0   0.3476418  0.2983562  0.006082553    0.006064130
 [9,]               3   0.4349871  0.3611559  0.038332380    0.037615947
[10,]               3   0.0000000  0.0000000  0.000000000    0.000000000
```

```
t(X)
```

```
                     [,1]        [,2]        [,3]        [,4] [,5] [,6]
trust_neighbors 3.000000000 3.00000000 3.00000000   0.000000    3    2
exports         0.388349682 0.63112360 0.99418926 182.589127    0    0
ln_exports      0.328115761 0.48926911 0.69023758   5.212700    0    0
export_area     0.004067405 0.09705944 0.01252469   1.824284    0    0
ln_export_area  0.004059155 0.09263337 0.01244691   1.038255    0    0
                     [,7]        [,8]        [,9] [,10]
trust_neighbors   2.000000 0.000000000 3.00000000     3
exports         665.965210 0.347641766 0.43498713     0
ln_exports        6.502738 0.298356235 0.36115587     0
export_area      13.975567 0.006082553 0.03833238     0
ln_export_area    2.706420 0.006064130 0.03761595     0
```

What are the values of all rows in the first column?

```
X[, 1]
```

```
 [1] 3 3 3 0 3 2 2 0 3 3
```

What are all the values of "exports"? (i.e. return the whole "exports" column)

```
X[, "exports"]
```

```
 [1]   0.3883497   0.6311236   0.9941893 182.5891266   0.0000000   0.0000000
 [7] 665.9652100   0.3476418   0.4349871   0.0000000
```

What is the first observation (i.e. first row)?

```
X[1, ]
```

```
trust_neighbors         exports      ln_exports     export_area  ln_export_area
    3.000000000     0.388349682     0.328115761     0.004067405     0.004059155
```

What is the value of the first variable of the first observation?

```
X[1, 1]
```

```
trust_neighbors
              3
```

Pause and consider the following problem on your own. What is the following code doing?

```
X[X[, "trust_neighbors"] == 0, "export_area"]
```

```
[1] 1.824284434 0.006082553
```

Why does it give the same output as the following?

```
X[which(X[, "trust_neighbors"] == 0), "export_area"]
```

```
[1] 1.824284434 0.006082553
```

Some more manipulation

```
X + X
```

```
      trust_neighbors      exports ln_exports  export_area ln_export_area
 [1,]               6    0.7766994  0.6562315  0.008134809     0.00811831
 [2,]               6    1.2622472  0.9785382  0.194118887     0.18526673
 [3,]               6    1.9883785  1.3804752  0.025049388     0.02489382
 [4,]               0  365.1782532 10.4254007  3.648568869     2.07651019
 [5,]               6    0.0000000  0.0000000  0.000000000     0.00000000
 [6,]               4    0.0000000  0.0000000  0.000000000     0.00000000
 [7,]               4 1331.9304199 13.0054760 27.951133728     5.41283989
 [8,]               0    0.6952835  0.5967125  0.012165107     0.01212826
 [9,]               6    0.8699743  0.7223117  0.076664761     0.07523189
[10,]               6    0.0000000  0.0000000  0.000000000     0.00000000
```

```
X - X
```

```
      trust_neighbors exports ln_exports export_area ln_export_area
 [1,]               0       0          0           0              0
 [2,]               0       0          0           0              0
 [3,]               0       0          0           0              0
 [4,]               0       0          0           0              0
 [5,]               0       0          0           0              0
 [6,]               0       0          0           0              0
 [7,]               0       0          0           0              0
 [8,]               0       0          0           0              0
 [9,]               0       0          0           0              0
[10,]               0       0          0           0              0
```

```
t(X) %*% X
```

```
                trust_neighbors    exports ln_exports export_area
trust_neighbors       62.000000   1339.276   18.61181    28.40709
exports             1339.276369 476850.298 5283.76294  9640.42990
ln_exports            18.611811   5283.763   70.50077   100.46202
export_area           28.407085   9640.430  100.46202   198.65558
ln_export_area         5.853106   1992.047   23.08189    39.72847
                ln_export_area
trust_neighbors       5.853106
exports            1992.046502
ln_exports           23.081893
export_area          39.728468
ln_export_area        8.412887
```

```
cbind(X, 1:10)
```

```
      trust_neighbors     exports ln_exports  export_area ln_export_area
 [1,]               3   0.3883497  0.3281158  0.004067405    0.004059155  1
 [2,]               3   0.6311236  0.4892691  0.097059444    0.092633367  2
 [3,]               3   0.9941893  0.6902376  0.012524694    0.012446908  3
 [4,]               0 182.5891266  5.2127004  1.824284434    1.038255095  4
 [5,]               3   0.0000000  0.0000000  0.000000000    0.000000000  5
 [6,]               2   0.0000000  0.0000000  0.000000000    0.000000000  6
 [7,]               2 665.9652100  6.5027380 13.975566864    2.706419945  7
 [8,]               0   0.3476418  0.2983562  0.006082553    0.006064130  8
 [9,]               3   0.4349871  0.3611559  0.038332380    0.037615947  9
[10,]               3   0.0000000  0.0000000  0.000000000    0.000000000 10
```

```
cbind(X, 1)
```

```
      trust_neighbors     exports ln_exports  export_area ln_export_area
 [1,]               3   0.3883497  0.3281158  0.004067405    0.004059155 1
 [2,]               3   0.6311236  0.4892691  0.097059444    0.092633367 1
 [3,]               3   0.9941893  0.6902376  0.012524694    0.012446908 1
 [4,]               0 182.5891266  5.2127004  1.824284434    1.038255095 1
 [5,]               3   0.0000000  0.0000000  0.000000000    0.000000000 1
 [6,]               2   0.0000000  0.0000000  0.000000000    0.000000000 1
 [7,]               2 665.9652100  6.5027380 13.975566864    2.706419945 1
 [8,]               0   0.3476418  0.2983562  0.006082553    0.006064130 1
 [9,]               3   0.4349871  0.3611559  0.038332380    0.037615947 1
[10,]               3   0.0000000  0.0000000  0.000000000    0.000000000 1
```

```
colnames(X)
```

```
[1] "trust_neighbors" "exports"         "ln_exports"      "export_area"
[5] "ln_export_area"
```

11.4 Variable Transformations

`exports` is the total number of slaves that were taken from the individual's ethnic group between Africa's four slave trades between 1400-1900.

What is `ln_exports`? The article describes this as the natural log of one plus the `exports`. This is a transformation of one column by a particular function

```
log(1 + X[, "exports"])
```

```
 [1] 0.3281158 0.4892691 0.6902376 5.2127003 0.0000000 0.0000000 6.5027379
 [8] 0.2983562 0.3611559 0.0000000
```

Question for you: why add the 1?

Verify that this is the same as `X[, "ln_exports"]`

11.5 Linear Combinations

In Table 1 we see "OLS Estimates". These are estimates of OLS coefficients and standard errors. You do not need to know what these are for now, but it doesn't hurt to getting used to seeing them.

TABLE 1—OLS ESTIMATES OF THE DETERMINANTS OF TRUST IN NEIGHBORS

Dependent variable: Trust of neighbors	Slave exports (thousands) (1)	Exports/ area (2)	Exports/ historical pop (3)	ln (1 + exports) (4)	ln (1 + exports/ area) (5)	ln (1 + exports/ historical pop) (6)
Estimated coefficient	−0.00068	−0.019	−0.531	−0.037	−0.159	−0.743
	[0.00014]	[0.005]	[0.147]	[0.014]	[0.034]	[0.187]
	(0.00015)	(0.005)	(0.147)	(0.014)	(0.034)	(0.187)
	{0.00013}	{0.005}	{0.165}	{0.015}	{0.034}	{0.212}
Individual controls	Yes	Yes	Yes	Yes	Yes	Yes
District controls	Yes	Yes	Yes	Yes	Yes	Yes
Country fixed effects	Yes	Yes	Yes	Yes	Yes	Yes
Number of observations	20,027	20,027	17,644	20,027	20,027	17,644
Number of ethnicities	185	185	157	185	185	157
Number of districts	1,257	1,257	1,214	1,257	1,257	1,214
R^2	0.16	0.16	0.15	0.15	0.16	0.15

Notes: The table reports OLS estimates. The unit of observation is an individual. Below each coefficient three standard errors are reported. The first, reported in square brackets, is standard errors adjusted for clustering within ethnic groups. The second, reported in parentheses, is standard errors adjusted for two-way clustering within ethnic groups and within districts. The third, reported in curly brackets, is T. G. Conley (1999) standard errors adjusted for two-dimensional spatial autocorrelation. The standard errors are constructed assuming a window with weights equal to one for observations less than five degrees apart and zero for observations further apart. The individual controls are for age, age squared, a gender indicator variable, five living conditions fixed effects, ten education fixed effects, 18 religion fixed effects, 25 occupation fixed effects, and an indicator for whether the respondent lives in an urban location. The district controls include ethnic fractionalization of each district and the share of the district's population that is the same ethnicity as the respondent.

A very crude way to describe regression is through linear combinations. The simplest linear combination is a one-to-one transformation.

Take the first number in Table 1, which is -0.00068. Now, multiply this by `exports`

```
-0.00068 * X[, "exports"]
```

```
[1] -0.0002640778 -0.0004291640 -0.0006760487 -0.1241606061  0.0000000000
[6]  0.0000000000 -0.4528563428 -0.0002363964 -0.0002957912  0.0000000000
```

Now, just one more step. Make a new matrix with just exports and the value 1

```
X2 <- cbind(1, X[, "exports"])
```

name this new column "intercept"

```
colnames(X2)
```

```
NULL
```

```
colnames(X2) <- c("intercept", "exports")
```

What are the dimensions of the matrix `X2`?

```
dim(X2)
```

```
[1] 10  2
```

Now consider a new matrix, called `B`.

```
B <- matrix(c(1.62, -0.00068))
```

What are the dimensions of `B`?

```
dim(B)
```

```
[1] 2 1
```

What is the product of `X2` and `B`? From the dimensions, can you tell if it will be conformable?

```
X2 %*% B
```

```
          [,1]
 [1,] 1.619736
 [2,] 1.619571
 [3,] 1.619324
 [4,] 1.495839
 [5,] 1.620000
 [6,] 1.620000
 [7,] 1.167144
 [8,] 1.619764
 [9,] 1.619704
[10,] 1.620000
```

What is this multiplication doing in terms of equations?

11.6 Matrix Basics

Let's take a look at Matrices in the context of R

```
cen10 <- read_csv("data/input/usc2010_001percent.csv")
head(cen10)
```

```
# A tibble: 6 x 4
  state         sex      age race
  <chr>         <chr>  <dbl> <chr>
1 New York      Female     8 White
2 Ohio          Male      24 White
3 Nevada        Male      37 White
4 Michigan      Female    12 White
5 Maryland      Female    18 Black/Negro
6 New Hampshire Male      50 White
```

What is the dimension of this dataframe? What does the number of rows represent? What does the number of columns represent?

```
dim(cen10)
```

```
[1] 30871     4
```

```
nrow(cen10)
```

```
[1] 30871
```

```
ncol(cen10)
```

```
[1] 4
```

What variables does this dataset hold? What kind of information does it have?

```
colnames(cen10)
```

```
[1] "state" "sex"   "age"   "race"
```

We can access column vectors, or vectors that contain values of variables by using the $ sign

```
head(cen10$state)
```

```
[1] "New York"      "Ohio"          "Nevada"        "Michigan"
[5] "Maryland"      "New Hampshire"
```

```
head(cen10$race)
```

```
[1] "White"       "White"       "White"       "White"       "Black/Negro"
[6] "White"
```

We can look at a unique set of variable values by calling the unique function

```
unique(cen10$state)
```

```
 [1] "New York"            "Ohio"                "Nevada"
 [4] "Michigan"            "Maryland"            "New Hampshire"
 [7] "Iowa"                "Missouri"            "New Jersey"
[10] "California"          "Texas"               "Pennsylvania"
[13] "Washington"          "West Virginia"       "Idaho"
[16] "North Carolina"      "Massachusetts"       "Connecticut"
[19] "Arkansas"            "Indiana"             "Wisconsin"
[22] "Maine"               "Tennessee"           "Minnesota"
[25] "Florida"             "Oklahoma"            "Montana"
[28] "Georgia"             "Arizona"             "Colorado"
[31] "Virginia"            "Illinois"            "Oregon"
[34] "Kentucky"            "South Carolina"      "Kansas"
[37] "Louisiana"           "Alabama"             "District of Columbia"
[40] "Mississippi"         "Utah"                "Delaware"
[43] "Nebraska"            "Alaska"              "New Mexico"
[46] "South Dakota"        "Hawaii"              "Vermont"
[49] "Rhode Island"        "Wyoming"             "North Dakota"
```

How many different states are represented (this dataset includes DC as a state)?

```
length(unique(cen10$state))
```

```
[1] 51
```

Matrices are rectangular structures of numbers (they have to be numbers, and they can't be characters).

A cross-tab can be considered a matrix:

```
table(cen10$race, cen10$sex)
```

```
                                  Female  Male
American Indian or Alaska Native     142   153
Black/Negro                         2070  1943
Chinese                              192   162
Japanese                              51    26
Other Asian or Pacific Islander      587   542
Other race, nec                      877   962
Three or more major races             37    51
Two major races                      443   426
White                              11252 10955
```

```
cross_tab <- table(cen10$race, cen10$sex)
dim(cross_tab)
```

```
[1] 9 2
```

```
cross_tab[6, 2]
```

```
[1] 962
```

But a subset of your data – individual values– can be considered a matrix too.

```
# First 20 rows of the entire data
# Below two lines of code do the same thing
cen10[1:20, ]
```

```
# A tibble: 20 x 4
   state         sex      age race
   <chr>         <chr>  <dbl> <chr>
 1 New York      Female     8 White
 2 Ohio          Male      24 White
 3 Nevada        Male      37 White
 4 Michigan      Female    12 White
 5 Maryland      Female    18 Black/Negro
 6 New Hampshire Male      50 White
 7 Iowa          Female    51 White
 8 Missouri      Female    41 White
 9 New Jersey    Male      62 White
10 California    Male      25 White
11 Texas         Female    23 White
12 Pennsylvania  Female    66 White
13 California    Female    57 White
14 Texas         Female    73 Other race, nec
15 California    Male      43 White
16 Washington    Male      29 White
17 Texas         Male       8 White
18 Missouri      Male      78 White
19 West Virginia Male      10 White
20 Idaho         Female     9 White
```

```
cen10 %>% slice(1:20)
```

```
# A tibble: 20 x 4
   state         sex      age race
   <chr>         <chr>  <dbl> <chr>
 1 New York      Female     8 White
 2 Ohio          Male      24 White
 3 Nevada        Male      37 White
 4 Michigan      Female    12 White
 5 Maryland      Female    18 Black/Negro
 6 New Hampshire Male      50 White
 7 Iowa          Female    51 White
 8 Missouri      Female    41 White
 9 New Jersey    Male      62 White
10 California    Male      25 White
11 Texas         Female    23 White
12 Pennsylvania  Female    66 White
```

```
13 California    Female    57 White
14 Texas         Female    73 Other race, nec
15 California    Male      43 White
16 Washington    Male      29 White
17 Texas         Male       8 White
18 Missouri      Male      78 White
19 West Virginia Male      10 White
20 Idaho         Female     9 White
```

```
# Of the first 20 rows of the entire data, look at values of just race and age
# Below two lines of code do the same thing
cen10[1:20, c("race", "age")]
```

```
# A tibble: 20 x 2
   race              age
   <chr>           <dbl>
 1 White               8
 2 White              24
 3 White              37
 4 White              12
 5 Black/Negro        18
 6 White              50
 7 White              51
 8 White              41
 9 White              62
10 White              25
11 White              23
12 White              66
13 White              57
14 Other race, nec    73
15 White              43
16 White              29
17 White               8
18 White              78
19 White              10
20 White               9
```

```
cen10 %>% slice(1:20) %>% select(race, age)
```

```
# A tibble: 20 x 2
   race              age
   <chr>           <dbl>
 1 White               8
 2 White              24
 3 White              37
 4 White              12
 5 Black/Negro        18
 6 White              50
 7 White              51
 8 White              41
 9 White              62
10 White              25
11 White              23
12 White              66
13 White              57
14 Other race, nec    73
15 White              43
16 White              29
17 White               8
18 White              78
19 White              10
20 White               9
```

A vector is a special type of matrix with only one column or only one row

```
# One column
cen10[1:10, c("age")]
```

```
# A tibble: 10 x 1
     age
   <dbl>
 1     8
 2    24
 3    37
 4    12
 5    18
 6    50
 7    51
 8    41
 9    62
10    25
```

```
  cen10 %>% slice(1:10) %>% select(c("age"))
```

```
# A tibble: 10 x 1
     age
   <dbl>
 1     8
 2    24
 3    37
 4    12
 5    18
 6    50
 7    51
 8    41
 9    62
10    25
```

```
  # One row
  cen10[2, ]
```

```
# A tibble: 1 x 4
  state sex     age race
  <chr> <chr> <dbl> <chr>
1 Ohio  Male     24 White
```

```
  cen10 %>% slice(2)
```

```
# A tibble: 1 x 4
  state sex     age race
  <chr> <chr> <dbl> <chr>
1 Ohio  Male     24 White
```

What if we want a special subset of the data? For example, what if I only want the records of individuals in California? What if I just want the age and race of individuals in California?

```
  # subset for CA rows
  ca_subset <- cen10[cen10$state == "California", ]
```

```
ca_subset_tidy <- cen10 %>% filter(state == "California")

all_equal(ca_subset, ca_subset_tidy)
```

```
Warning: `all_equal()` was deprecated in dplyr 1.1.0.
i Please use `all.equal()` instead.
i And manually order the rows/cols as needed
```

```
[1] TRUE
```

```
# subset for CA rows and select age and race
ca_subset_age_race <- cen10[cen10$state == "California", c("age", "race")]

ca_subset_age_race_tidy <- cen10 %>% filter(state == "California") %>% select(age, race)

all_equal(ca_subset_age_race, ca_subset_age_race_tidy)
```

```
[1] TRUE
```

Some common operators that can be used to filter or to use as a condition. Remember, you can use the unique function to look at the set of all values a variable holds in the dataset.

```
# all individuals older than 30 and younger than 70
s1 <- cen10[cen10$age > 30 & cen10$age < 70, ]
s2 <- cen10 %>% filter(age > 30 & age < 70)
all_equal(s1, s2)
```

```
[1] TRUE
```

```
# all individuals in either New York or California
s3 <- cen10[cen10$state == "New York" | cen10$state == "California", ]
s4 <- cen10 %>% filter(state == "New York" | state == "California")
all_equal(s3, s4)
```

```
[1] TRUE
```

```r
# all individuals in any of the following states: California, Ohio, Nevada, Michigan
s5 <- cen10[cen10$state %in% c("California", "Ohio", "Nevada", "Michigan"), ]
s6 <- cen10 %>% filter(state %in% c("California", "Ohio", "Nevada", "Michigan"))
all_equal(s5, s6)
```

```
[1] TRUE
```

```r
# all individuals NOT in any of the following states: California, Ohio, Nevada, Michigan
s7 <- cen10[!(cen10$state %in% c("California", "Ohio", "Nevada", "Michigan")), ]
s8 <- cen10 %>% filter(!state %in% c("California", "Ohio", "Nevada", "Michigan"))
all_equal(s7, s8)
```

```
[1] TRUE
```

Checkpoint

1

Get the subset of cen10 for non-white individuals (Hint: look at the set of values for the race variable by using the unique function)

```r
# Enter here
```

2

Get the subset of cen10 for females over the age of 40

```r
# Enter here
```

3

Get all the serial numbers for black, male individuals who don't live in Ohio or Nevada.

```r
# Enter here
```

Exercises

1

Let

$$\mathbf{A} = \begin{bmatrix} 0.6 & 0.2 \\ 0.4 & 0.8 \end{bmatrix}$$

Use R to write code that will create the matrix A, and then consecutively multiply A to itself 4 times. What is the value of A^4?

```
## Enter yourself
```

Note that R notation of matrices is different from the math notation. Simply trying `X^n` where `X` is a matrix will only take the power of each element to `n`. Instead, this problem asks you to perform matrix multiplication.

2

Let's apply what we learned about subsetting or filtering/selecting. Use the `nunn_full` dataset you have already loaded

a) First, show all observations (rows) that have a `"male"` variable higher than 0.5

```
## Enter yourself
```

b) Next, create a matrix / dataframe with only two columns: `"trust_neighbors"` and `"age"`

```
## Enter yourself
```

c) Lastly, show all values of `"trust_neighbors"` and `"age"` for observations (rows) that have the "male" variable value that is higher than 0.5

```
## Enter yourself
```

3

Find a way to generate a vector of "column averages" of the matrix `X` from the Nunn and Wantchekon data in one line of code. Each entry in the vector should contain the sample average of the values in the column. So a 100 by 4 matrix should generate a length-4 matrix.

4

Similarly, generate a vector of "column medians".

5

Consider the regression that was run to generate Table 1:

```
form <- "trust_neighbors ~ exports + age + age2 +  male + urban_dum + factor(education) +
lm_1_1 <- lm(as.formula(form), nunn_full)

# The below coef function returns a vector of OLS coefficiants
coef(lm_1_1)
```

```
          (Intercept)                   exports
         1.619913e+00             -6.791360e-04
                  age                      age2
         8.395936e-03             -5.473436e-05
                 male                  urban_dum
         4.550246e-02             -1.404551e-01
 factor(education)1        factor(education)2
         1.709816e-02             -5.224591e-02
 factor(education)3        factor(education)4
        -1.373770e-01             -1.889619e-01
 factor(education)5        factor(education)6
        -1.893494e-01             -2.400767e-01
 factor(education)7        factor(education)8
        -2.850748e-01             -1.232085e-01
 factor(education)9       factor(occupation)1
        -2.406437e-01              6.185655e-02
factor(occupation)2       factor(occupation)3
         7.392168e-02              3.356158e-02
factor(occupation)4       factor(occupation)5
         7.942048e-03              6.661126e-02
```

```
       factor(occupation)6        factor(occupation)7
             -7.563297e-02              1.699699e-02
       factor(occupation)8        factor(occupation)9
             -9.428177e-02             -9.981440e-02
      factor(occupation)10       factor(occupation)11
             -3.307068e-02             -2.300045e-02
      factor(occupation)12       factor(occupation)13
             -1.564540e-01             -1.441370e-02
      factor(occupation)14       factor(occupation)15
             -5.566414e-02             -2.343762e-01
      factor(occupation)16       factor(occupation)18
             -1.306947e-02             -1.729589e-01
      factor(occupation)19       factor(occupation)20
             -1.770261e-01             -2.457800e-02
      factor(occupation)21       factor(occupation)22
             -4.936813e-02             -1.068511e-01
      factor(occupation)23       factor(occupation)24
             -9.712205e-02              1.292371e-02
      factor(occupation)25      factor(occupation)995
              2.623186e-02             -1.195063e-03
         factor(religion)2          factor(religion)3
              5.395953e-02              7.887878e-02
         factor(religion)4          factor(religion)5
              4.749150e-02              4.318455e-02
         factor(religion)6          factor(religion)7
             -1.787694e-02             -3.616542e-02
        factor(religion)10         factor(religion)11
              6.015041e-02              2.237845e-01
        factor(religion)12         factor(religion)13
              2.627086e-01             -6.812813e-02
        factor(religion)14         factor(religion)15
              4.673681e-02              3.844555e-01
       factor(religion)360        factor(religion)361
              3.656843e-01              3.416413e-01
       factor(religion)362        factor(religion)363
              8.230393e-01              3.856565e-01
       factor(religion)995 factor(living_conditions)2
              4.161301e-02              4.395862e-02
factor(living_conditions)3 factor(living_conditions)4
              8.627372e-02              1.197428e-01
factor(living_conditions)5       district_ethnic_frac
              1.203606e-01             -1.553648e-02
frac_ethnicity_in_district                isocodeBWA
```

```
 1.011222e-01             -4.258953e-01
   isocodeGHA               isocodeKEN
 1.135307e-02             -1.819556e-01
   isocodeLSO               isocodeMDG
-5.511200e-01             -3.315727e-01
   isocodeMLI               isocodeMOZ
 7.528101e-02              8.223730e-02
   isocodeMWI               isocodeNAM
 3.062497e-01             -1.397541e-01
   isocodeNGA               isocodeSEN
-2.381525e-01              3.867371e-01
   isocodeTZA               isocodeUGA
 2.079366e-01             -6.443732e-02
   isocodeZAF               isocodeZMB
-2.179153e-01             -2.172868e-01
```

First, get a small subset of the nunn_full dataset. This time, sample 20 rows and select for variables `exports`, `age`, `age2`, `male`, and `urban_dum`. To this small subset, add (`bind_cols()` in tidyverse or `cbind()` in base R) a column of 1's; this represents the intercept. If you need some guidance, look at how we sampled 10 rows selected for a different set of variables above in the lecture portion.

```
# Enter here
```

Next let's try calculating predicted values of levels of trust in neighbors by multiplying coefficients for the intercept, `exports`, `age`, `age2`, `male`, and `urban_dum` to the actual observed values for those variables in the small subset you've just created.

```
# Hint: You can get just selected elements from the vector returned by coef(lm_1_1)

# For example, the below code gives you the first 3 elements of the original vector
coef(lm_1_1)[1:3]
```

```
 (Intercept)      exports          age
1.619913146 -0.000679136  0.008395936
```

```
# Also, the below code gives you the coefficient elements for intercept and male
coef(lm_1_1)[c("(Intercept)", "male")]
```

```
(Intercept)        male
 1.61991315  0.04550246
```

12 Objects, Functions, Loops

Where are we? Where are we headed?

Up till now, you should have covered:

- R basic programming
- Data Import
- Statistical Summaries
- Visualization

Today we'll cover

- Objects
- Functions
- Loops

12.1 What is an object?

Now that we have covered some hands-on ways to use graphics, let's go into some fundamentals of the R language.

Let's first set up

```
library(dplyr)
library(readr)
library(haven)
library(ggplot2)
```

```
cen10 <- read_csv("data/input/usc2010_001percent.csv", col_types = cols())
```

Objects are abstract symbols in which you store data. Here we will create an object from `copy`, and assign `cen10` to it.

```
copy <- cen10
```

This looks the same as the original dataset:

```
copy
```

```
# A tibble: 30,871 x 4
   state         sex      age race
   <chr>         <chr>  <dbl> <chr>
 1 New York      Female     8 White
 2 Ohio          Male      24 White
 3 Nevada        Male      37 White
 4 Michigan      Female    12 White
 5 Maryland      Female    18 Black/Negro
 6 New Hampshire Male      50 White
 7 Iowa          Female    51 White
 8 Missouri      Female    41 White
 9 New Jersey    Male      62 White
10 California    Male      25 White
# i 30,861 more rows
```

What happens if you do this next?

```
copy <- ""
```

It got reassigned:

```
copy
```

```
[1] ""
```

12.1.1 Lists

Lists are one of the most generic and flexible type of object. You can make an empty list by the function `list()`

```
my_list <- list()
my_list
```

```
list()
```

And start filling it in. Slots on the list are invoked by double square brackets `[[]]`

```
my_list[[1]] <- "contents of the first slot -- this is a string"
my_list[["slot 2"]] <- "contents of slot named slot 2"
my_list
```

```
[[1]]
[1] "contents of the first slot -- this is a string"

$`slot 2`
[1] "contents of slot named slot 2"
```

each slot can be anything. What are we doing here? We are defining the 1st slot of the list `my_list` to be a vector `c(1, 2, 3, 4, 5)`

```
my_list[[1]] <- c(1, 2, 3, 4, 5)
my_list
```

```
[[1]]
[1] 1 2 3 4 5

$`slot 2`
[1] "contents of slot named slot 2"
```

You can even make nested lists. Let's say we want the 1st slot of the list to be another list of three elements.

```
my_list[[1]][[1]] <- "subitem 1 in slot 1 of my_list"
my_list[[1]][[2]] <- "subitem 1 in slot 2 of my_list"
my_list[[1]][[3]] <- "subitem 1 in slot 3 of my_list"

my_list
```

```
[[1]]
[1] "subitem 1 in slot 1 of my_list" "subitem 1 in slot 2 of my_list"
[3] "subitem 1 in slot 3 of my_list" "4"
[5] "5"

$`slot 2`
[1] "contents of slot named slot 2"
```

12.2 Making your own objects

We've covered one type of object, which is a list. You saw it was quite flexible. How many types of objects are there?

There are an infinite number of objects, because people make their own class of object. You can detect the type of the object (the class) by the function `class`

Object can be said to be an instance of a class.

Analogies:

class - Pokemon, **object** - Pikachu

class - Book, **object** - To Kill a Mockingbird

class - DataFrame, **object** - 2010 census data

class - Character, **object** - "Programming is Fun"

What is type (class) of object is `cen10`?

```
class(cen10)
```

```
[1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"
```

What about this text?

```
class("some random text")
```

```
[1] "character"
```

To change or create the class of any object, you can *assign* it. To do this, assign the name of your class to character to an object's `class()`.

We can start from a simple list. For example, say we wanted to store data about pokemon. Because there is no pre-made package for this, we decide to make our own class.

```
pikachu <- list(name = "Pikachu",
                number = 25,
                type = "Electric",
                color = "Yellow")
```

and we can give it any class name we want.

```
class(pikachu) <- "Pokemon"
str(pikachu)
```

```
List of 4
 $ name  : chr "Pikachu"
 $ number: num 25
 $ type  : chr "Electric"
 $ color : chr "Yellow"
 - attr(*, "class")= chr "Pokemon"
```

```
pikachu$type
```

```
[1] "Electric"
```

We can even define **class-specific** methods. For example, the `summary()` function is commonly used to summarize the output of a particular object (such as an `lm()` regression object). However, `summary()` will behave differently depending on what object you give it. Why? Because there is a version of summary defined specifically for `lm()` objects. Let's define a `summary()` method for the `Pokemon` class

```
# Input: an object of class "Pokemon"
# Output: A text summary of the Pokemon
summary.Pokemon <- function(x){
  out_text <- paste(x$name, " is a(n) ", x$type,
  " type Pokemon. Its PokeDex number is ",
  x$number, ". It is ", x$color, ".\n", sep="")
  cat(out_text)
}
```

Now let's call the generic `summary()` function on `pikachu`

```
summary(pikachu)
```

```
Pikachu is a(n) Electric type Pokemon. Its PokeDex number is 25. It is Yellow.
```

12.2.1 Seeing R through objects

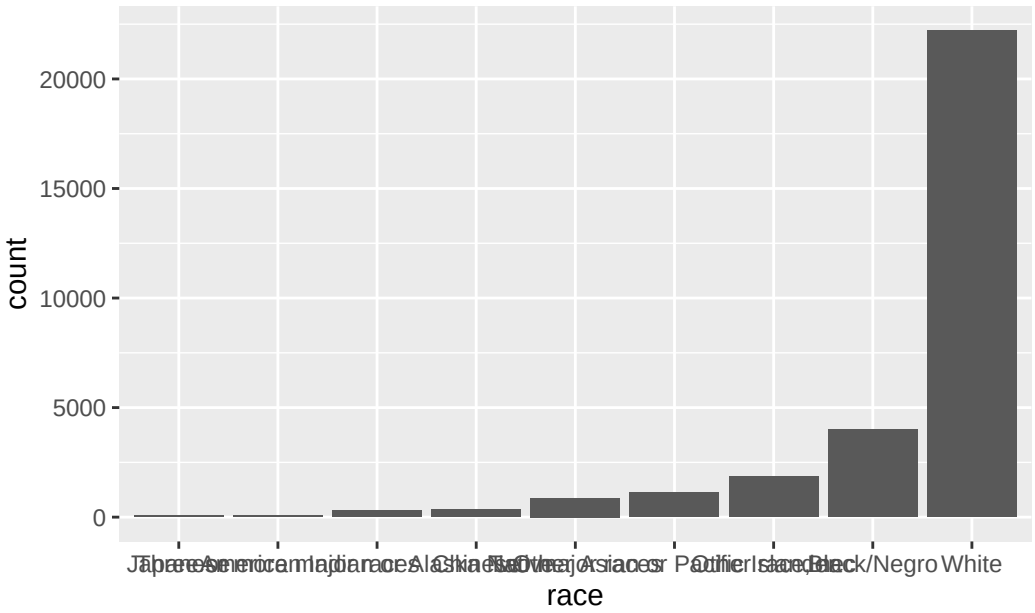
Most of the R objects that you will see as you advance are their own objects. For example, here's a linear regression object

```
ols <- lm(mpg ~ wt + vs + gear + carb, mtcars)
class(ols)
```

```
[1] "lm"
```

Anything can be an object! Even graphs (in `ggplot`) can be assigned, re-assigned, and edited.

```
grp_race <- group_by(cen10, race)%>%
  summarize(count = n())

grp_race_ordered <- arrange(grp_race, count) %>%
  mutate(race = forcats::as_factor(race))

gg_tab <- ggplot(data = grp_race_ordered) +
  aes(x = race, y = count) +
  geom_col() +
  labs(caption = "Source: U.S. Census 2010")

gg_tab
```



Source: U.S. Census 2010

You can change the orientation

```
gg_tab<- gg_tab + coord_flip()
```

12.2.2 Parsing an object by `str()`s

It can be hard to understand an `R` object because it's contents are unknown. The function `str`, short for structure, is a quick way to look into the innards of an object

```
str(my_list)
```

```
List of 2
 $       : chr [1:5] "subitem 1 in slot 1 of my_list" "subitem 1 in slot 2 of my_list" "subit
 $ slot 2: chr "contents of slot named slot 2"
```

```
class(my_list)
```

```
[1] "list"
```

Same for the object we just made

```
  str(pikachu)
```

```
List of 4
 $ name  : chr "Pikachu"
 $ number: num 25
 $ type  : chr "Electric"
 $ color : chr "Yellow"
 - attr(*, "class")= chr "Pokemon"
```

What does a `ggplot` object look like? Very complicated, but at least you can see it:

```
  # enter this on your console
  str(gg_tab)
```

12.3 Types of variables

In the social science we often analyze variables. As you saw in the tutorial, different types of variables require different care.

A key link with what we just learned is that variables are also types of R objects.

12.3.1 scalars

One number. How many people did we count in our Census sample?

```
  nrow(cen10)
```

```
[1] 30871
```

Question: What proportion of our census sample is Native American? This number is also a scalar

```
  # Enter yourself
  unique(cen10$race)
```

```
[1] "White"                                 "Black/Negro"
[3] "Other race, nec"                       "American Indian or Alaska Native"
[5] "Chinese"                               "Other Asian or Pacific Islander"
[7] "Two major races"                       "Three or more major races"
[9] "Japanese"
```

```
mean(cen10$race == "American Indian or Alaska Native")
```

```
[1] 0.009555894
```

Hint: you can use the function `mean()` to calcualte the sample mean. The sample proportion is the mean of a sequence of number, where your event of interest is a 1 (or `TRUE`) and others are 0 (or `FALSE`).

12.3.2 numeric vectors

A sequence of numbers.

```
grp_race_ordered$count
```

```
[1]    77    88   295   354   869  1129  1839  4013 22207
```

```
class(grp_race_ordered$count)
```

```
[1] "integer"
```

Or even, all the ages of the millions of people in our Census. Here are just the first few numbers of the list.

```
head(cen10$age)
```

```
[1]  8 24 37 12 18 50
```

12.3.3 characters (aka strings)

This can be just one stretch of characters

```
my_name <- "Anton"
my_name
```

```
[1] "Anton"
```

```
class(my_name)
```

```
[1] "character"
```

or more characters. Notice here that there's a difference between a vector of individual characters and a length-one object of characters.

```
my_name_letters <-  c("A","n","t","o","n")
my_name_letters
```

```
[1] "A" "n" "t" "o" "n"
```

```
class(my_name_letters)
```

```
[1] "character"
```

Finally, remember that lower vs. upper case matters in R!

```
my_name2 <- "anton"
my_name == my_name2
```

```
[1] FALSE
```

12.4 What is a function?

Most of what we do in R is executing a function. `read_csv()`, `nrow()`, `ggplot()` .. pretty much anything with a parentheses is a function. And even things like `<-` and `[` are functions as well.

A function is a set of instructions with specified ingredients. It takes an **input**, then **manipulates** it – changes it in some way – and then returns the manipulated product.

One way to see what a function actually does is to enter it without parentheses.

```
# enter this on your console
table
```

You'll see below that the most basic functions are quite complicated internally.

You'll notice that functions contain other functions. *wrapper* functions are functions that "wrap around" existing functions. This sounds redundant, but it's an important feature of programming. If you find yourself repeating a command more than two times, you should make your own function, rather than writing the same type of code.

12.4.1 Write your own function

It's worth remembering the basic structure of a function. You create a new function, call it `my_fun` by this:

```
my_fun <- function() {

}
```

If we wanted to generate a function that computed the number of men in your data, what would that look like?

```
count_men <- function(data) {

  nmen <- sum(data$sex == "Male")

  return(nmen)
}
```

Then all we need to do is feed this function a dataset

```
count_men(cen10)
```

```
[1] 15220
```

The point of a function is that you can use it again and again without typing up the set of constituent manipulations. So, what if we wanted to figure out the number of men in California?

```
count_men(cen10[cen10$state == "California",])
```

```
[1] 1876
```

Let's go one step further. What if we want to know the proportion of non-whites in a state, just by entering the name of the state? There's multiple ways to do it, but it could look something like this

```
nw_in_state <- function(data, state) {

  s.subset <- data[data$state == state,]
  total.s <- nrow(s.subset)
  nw.s <- sum(s.subset$race != "White")

  nw.s / total.s
}
```

The last line is what gets generated from the function. To be more explicit you can wrap the last line around `return()`. (as in `return(nw.s/total.s)`. `return()` is used when you want to break out of a function in the middle of it and not wait till the last line.

Try it on your favorite state!

```
nw_in_state(cen10, "Massachusetts")
```

```
[1] 0.2040185
```

Checkpoint

1

Try making your own function, `average_age_in_state`, that will give you the average age of people in a given state.

```
# Enter on your own
```

2

Try making your own function, `asians_in_state`, that will give you the number of `Chinese`, `Japanese`, and `Other Asian or Pacific Islander` people in a given state.

```
# Enter on your own
```

3

Try making your own function, 'top_10_oldest_cities', that will give you the names of cities whose population's average age is top 10 oldest.

```
# Enter on your own
```

12.5 What is a package?

You can think of a package as a suite of functions that other people have already built for you to make your life easier.

```
help(package = "ggplot2")
```

To use a package, you need to do two things: (1) install it, and then (2) load it.

Installing is a one-time thing

```
install.packages("ggplot2")
```

But you need to load each time you start a R instance. So always keep these commands on a script.

```
library(ggplot2)
```

12.6 Conditionals

Sometimes, you want to execute a command only under certain conditions. This is done through the almost universal function, `if()`. Inside the `if` function we enter a logical statement. The line that is adjacent to, or follows, the `if()` statement only gets executed if the statement returns `TRUE`.

For example,

For example,

```
x <- 5
if (x >0) {
  print("positive number")
} else if (x == 0)  {
  print ("zero")
} else {
  print("negative number")
}
```

```
[1] "positive number"
```

You can wrap that whole things in a function

```
is_positive <- function(number) {
  if (number >0) {
    print("positive number")
  } else if (number == 0)  {
    print ("zero")
  } else {
    print("negative number")
  }
}

is_positive(5)
```

```
[1] "positive number"
```

```
is_positive(-3)
```

```
[1] "negative number"
```

12.7 For-loops

Loops repeat the same statement, although the statement can be "the same" only in an abstract sense. Use the `for(x in X)` syntax to repeat the subsequent command as many times as there are elements in the right-hand object `X`. Each of these elements will be referred to the left-hand index `x`

First, come up with a vector.

```
fruits <- c("apples", "oranges", "grapes")
```

Now we use the `fruits` vector in a `for` loop.

```
for (fruit in fruits) {
  print(paste("I love", fruit))
}
```

```
[1] "I love apples"
[1] "I love oranges"
[1] "I love grapes"
```

Here `for()` and `in` must be part of any for loop. The right hand side `fruits` must be a thing that exists. Finally the `left-hand` side object is "Pick your favor name." It is analogous to how we can index a sum with any letter. $\sum_{i=1}^{10} i$ and `sum_{j = 1}^{10}j` are in fact the same thing.

```
for (i in 1:length(fruits)) {
  print(paste("I love", fruits[i]))
}
```

```
[1] "I love apples"
[1] "I love oranges"
[1] "I love grapes"
```

```r
states_of_interest <- c("California", "Massachusetts", "New Hampshire", "Washington")

for( state in states_of_interest){
  state_data <- cen10[cen10$state == state,]
  nmen <- sum(state_data$sex == "Male")

  n <- nrow(state_data)
  men_perc <- round(100*(nmen/n), digits=2)
  print(paste("Percentage of men in",state, "is", men_perc))

}
```

```
[1] "Percentage of men in California is 49.85"
[1] "Percentage of men in Massachusetts is 47.6"
[1] "Percentage of men in New Hampshire is 48.55"
[1] "Percentage of men in Washington is 48.19"
```

Instead of printing, you can store the information in a vector

```r
states_of_interest <- c("California", "Massachusetts", "New Hampshire", "Washington")
male_percentages <- c()
iter <-1

for( state in states_of_interest){
  state_data <- cen10[cen10$state == state,]
  nmen <- sum(state_data$sex == "Male")
  n <- nrow(state_data)
  men_perc <- round(100*(nmen/n), digits=2)

  male_percentages <- c(male_percentages, men_perc)
  names(male_percentages)[iter] <- state
  iter <- iter + 1
}

male_percentages
```

```
   California Massachusetts New Hampshire    Washington
        49.85         47.60         48.55         48.19
```

12.8 Nested Loops

What if I want to calculate the population percentage of a race group for all race groups in states of interest? You could probably use tidyverse functions to do this, but let's try using loops!

```
states_of_interest <- c("California", "Massachusetts", "New Hampshire", "Washington")
for (state in states_of_interest) {
  for (race in unique(cen10$race)) {
    race_state_num <- nrow(cen10[cen10$race == race & cen10$state == state, ])
    state_pop <- nrow(cen10[cen10$state == state, ])
    race_perc <- round(100*(race_state_num/(state_pop)), digits=2)
    print(paste("Percentage of ", race , "in", state, "is", race_perc))
  }
}
```

```
[1] "Percentage of  White in California is 57.61"
[1] "Percentage of  Black/Negro in California is 6.72"
[1] "Percentage of  Other race, nec in California is 15.55"
[1] "Percentage of  American Indian or Alaska Native in California is 1.12"
[1] "Percentage of  Chinese in California is 3.75"
[1] "Percentage of  Other Asian or Pacific Islander in California is 9.54"
[1] "Percentage of  Two major races in California is 4.62"
[1] "Percentage of  Three or more major races in California is 0.37"
[1] "Percentage of  Japanese in California is 0.72"
[1] "Percentage of  White in Massachusetts is 79.6"
[1] "Percentage of  Black/Negro in Massachusetts is 5.87"
[1] "Percentage of  Other race, nec in Massachusetts is 4.02"
[1] "Percentage of  American Indian or Alaska Native in Massachusetts is 0.77"
[1] "Percentage of  Chinese in Massachusetts is 2.32"
[1] "Percentage of  Other Asian or Pacific Islander in Massachusetts is 4.33"
[1] "Percentage of  Two major races in Massachusetts is 2.78"
[1] "Percentage of  Three or more major races in Massachusetts is 0"
[1] "Percentage of  Japanese in Massachusetts is 0.31"
[1] "Percentage of  White in New Hampshire is 93.48"
[1] "Percentage of  Black/Negro in New Hampshire is 0.72"
[1] "Percentage of  Other race, nec in New Hampshire is 0.72"
[1] "Percentage of  American Indian or Alaska Native in New Hampshire is 0.72"
[1] "Percentage of  Chinese in New Hampshire is 0.72"
[1] "Percentage of  Other Asian or Pacific Islander in New Hampshire is 2.17"
[1] "Percentage of  Two major races in New Hampshire is 0.72"
```

```
[1] "Percentage of  Three or more major races in New Hampshire is 0"
[1] "Percentage of  Japanese in New Hampshire is 0.72"
[1] "Percentage of  White in Washington is 76.05"
[1] "Percentage of  Black/Negro in Washington is 2.9"
[1] "Percentage of  Other race, nec in Washington is 5.37"
[1] "Percentage of  American Indian or Alaska Native in Washington is 2.03"
[1] "Percentage of  Chinese in Washington is 1.31"
[1] "Percentage of  Other Asian or Pacific Islander in Washington is 6.68"
[1] "Percentage of  Two major races in Washington is 4.79"
[1] "Percentage of  Three or more major races in Washington is 0.29"
[1] "Percentage of  Japanese in Washington is 0.58"
```

Exercises

Exercise 1: Write your own function

Write your own function that makes some task of data analysis simpler. Ideally, it would be a function that helps you do either of the previous tasks in fewer lines of code. You can use the three lines of code that was provided in exercise 1 to wrap that into another function too!

```
# Enter yourself
```

Exercise 2: Using Loops

Using a loop, create a crosstab of sex and race for each state in the set "states_of_interest"

```
states_of_interest <- c("California", "Massachusetts", "New Hampshire", "Washington")
# Enter yourself
```

Exercise 3: Storing information derived within loops in a global dataframe

Recall the following nested loop

```
states_of_interest <- c("California", "Massachusetts", "New Hampshire", "Washington")
for (state in states_of_interest) {
  for (race in unique(cen10$race)) {
    race_state_num <- nrow(cen10[cen10$race == race & cen10$state == state, ])
    state_pop <- nrow(cen10[cen10$state == state, ])
```

```
    race_perc <- round(100*(race_state_num/(state_pop)), digits=2)
    print(paste("Percentage of ", race , "in", state, "is", race_perc))
  }
}
```

```
[1] "Percentage of  White in California is 57.61"
[1] "Percentage of  Black/Negro in California is 6.72"
[1] "Percentage of  Other race, nec in California is 15.55"
[1] "Percentage of  American Indian or Alaska Native in California is 1.12"
[1] "Percentage of  Chinese in California is 3.75"
[1] "Percentage of  Other Asian or Pacific Islander in California is 9.54"
[1] "Percentage of  Two major races in California is 4.62"
[1] "Percentage of  Three or more major races in California is 0.37"
[1] "Percentage of  Japanese in California is 0.72"
[1] "Percentage of  White in Massachusetts is 79.6"
[1] "Percentage of  Black/Negro in Massachusetts is 5.87"
[1] "Percentage of  Other race, nec in Massachusetts is 4.02"
[1] "Percentage of  American Indian or Alaska Native in Massachusetts is 0.77"
[1] "Percentage of  Chinese in Massachusetts is 2.32"
[1] "Percentage of  Other Asian or Pacific Islander in Massachusetts is 4.33"
[1] "Percentage of  Two major races in Massachusetts is 2.78"
[1] "Percentage of  Three or more major races in Massachusetts is 0"
[1] "Percentage of  Japanese in Massachusetts is 0.31"
[1] "Percentage of  White in New Hampshire is 93.48"
[1] "Percentage of  Black/Negro in New Hampshire is 0.72"
[1] "Percentage of  Other race, nec in New Hampshire is 0.72"
[1] "Percentage of  American Indian or Alaska Native in New Hampshire is 0.72"
[1] "Percentage of  Chinese in New Hampshire is 0.72"
[1] "Percentage of  Other Asian or Pacific Islander in New Hampshire is 2.17"
[1] "Percentage of  Two major races in New Hampshire is 0.72"
[1] "Percentage of  Three or more major races in New Hampshire is 0"
[1] "Percentage of  Japanese in New Hampshire is 0.72"
[1] "Percentage of  White in Washington is 76.05"
[1] "Percentage of  Black/Negro in Washington is 2.9"
[1] "Percentage of  Other race, nec in Washington is 5.37"
[1] "Percentage of  American Indian or Alaska Native in Washington is 2.03"
[1] "Percentage of  Chinese in Washington is 1.31"
[1] "Percentage of  Other Asian or Pacific Islander in Washington is 6.68"
[1] "Percentage of  Two major races in Washington is 4.79"
[1] "Percentage of  Three or more major races in Washington is 0.29"
[1] "Percentage of  Japanese in Washington is 0.58"
```

Instead of printing the percentage of each race in each state, create a dataframe, and store all that information in that dataframe. (Hint: look at how I stored information about male percentage in each state of interest in a vector.)

13 Joins and Merges, Wide and Long[1]

Motivation

The "Democratic Peace" is one of the most widely discussed propositions in political science, covering the fields of International Relations and Comparative Politics, with insights to domestic politics of democracies (e.g. American Politics). The one-sentence idea is that democracies do not fight with each other. There have been much theoretical debate – for example in earlier work, Oneal and Russet (1999) argue that the democratic peace is not due to the hegemony of strong democracies like the U.S. and attempt to distinguish between realist and what they call Kantian propositions (e.g. democratic governance, international organizations)[2].

An empirical demonstration of the democratic peace is also a good example of a **Time Series Cross Sectional** (or panel) dataset, where the same units (in this case countries) are observed repeatedly for multiple time periods. Experience in assembling and analyzing a TSCS dataset will prepare you for any future research in this area.

Where are we? Where are we headed?

Up till now, you should have covered:

- R basic programming
- Counting.
- Visualization.
- Objects and Classes.
- Matrix algebra in R
- Functions.

Today you will work on your own, but feel free to ask a fellow classmate nearby or the instructor. The objective for this session is to get more experience using R, but in the process (a) test a prominent theory in the political science literature and (b) explore related ideas of interest to you.

[1] Module originally written by Shiro Kuriwaki, Connor Jerzak, and Yon Soo Park

[2] The Kantian Peace: The Pacific Benefits of Democracy, Interdependence, and International Organizations, 1885-1992. *World Politics* 52(1):1-37

13.1 Setting up

```
library(dplyr)
library(tidyr)
library(readr)
library(ggplot2)
```

13.2 Create a project directory

First start a directory for this project. This can be done manually or through RStudio's Project feature(`File > New Project...`)

Directories is the computer science / programming name for folders. While advice about how to structure your working directories might strike you as petty, we believe that starting from some well-tested guides will go a long way in improving the quality and efficiency of your work.

Chapter 4 of Gentzkow and Shapiro's memo, Code and Data for the Social Scientist] provides a good template.

13.3 Data Sources

Most projects you do will start with downloading data from elsewhere. For this task, you'll probably want to track down and download the following:

- **Correlates of war dataset (COW):** Find and download the Militarized Interstate Disputes (MIDs) data from the Correlates of War website: http://www.correlatesofwar.org/data-sets. Or a dyad-version on dataverse: https://dataverse.harvard.edu/dataset.xhtml?persistentId=hdl:1902.1/11489
- **PRIO Data on Armed Conflict:** Find and download the Uppsala Conflict Data Program (UCDP) and PRIO dyad-year data on armed conflict(https://www.prio.org) or this link to to the flat csv file (http://ucdp.uu.se/downloads/dyadic/ucdp-dyadic-171.csv).
- **Polity:** The Polity data can be downloaded from their website (http://www.systemicpeace.org/inscrdata.html). Look for the newest version of the time series that has the widest coverage.

13.4 Example with 2 Datasets

Let's read in a sample dataset.

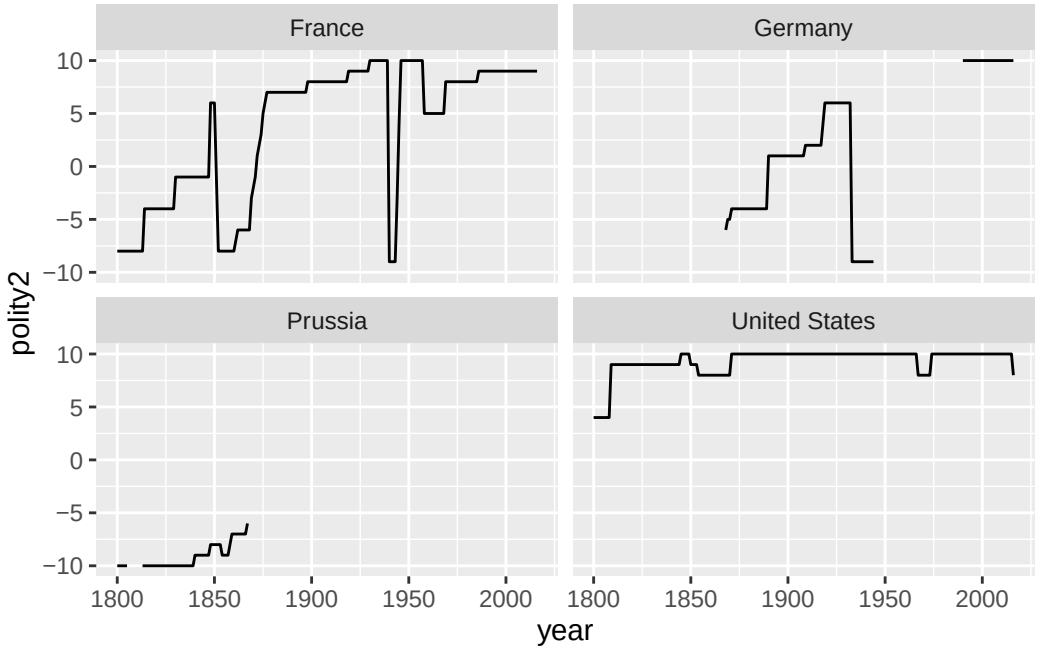
```
polity <- read_csv("data/input/sample_polity.csv")
mid <- read_csv("data/input/sample_mid.csv")
```

What does `polity` look like?

```
unique(polity$country)
```

```
[1] "France"        "Prussia"       "Germany"       "United States"
```

```
ggplot(polity, aes(x = year, y = polity2)) +
  facet_wrap(~ country) +
  geom_line()
```




```
head(polity)
```

```
# A tibble: 6 x 5
  scode ccode country  year polity2
  <chr> <dbl> <chr>   <dbl>   <dbl>
1 FRN     220 France   1800      -8
2 FRN     220 France   1801      -8
3 FRN     220 France   1802      -8
4 FRN     220 France   1803      -8
5 FRN     220 France   1804      -8
6 FRN     220 France   1805      -8
```

MID is a dataset that captures a `dispute` for a given country and year.

```
mid
```

```
# A tibble: 6,132 x 5
   ccode polity_code dispute StYear EndYear
   <dbl> <chr>         <dbl>  <dbl>   <dbl>
 1   200 UKG               1   1902    1903
 2     2 USA               1   1902    1903
 3   345 YGS               1   1913    1913
 4   300 <NA>              1   1913    1913
 5   339 ALB               1   1946    1946
 6   200 UKG               1   1946    1946
 7   200 UKG               1   1951    1952
 8   651 EGY               1   1951    1952
 9   630 IRN               1   1856    1857
10   200 UKG               1   1856    1857
# i 6,122 more rows
```

13.5 Loops

Notice that in the `mid` data, we have a start of a dispute vs. an end of a dispute.In order to combine this into the `polity` data, we want a way to give each of the interval years a row.

There are many ways to do this, but one is a loop. We go through one row at a time, and then for each we make a new dataset. that has `year` as a sequence of each year. A lengthy loop like this is typically slow, and you'd want to recast the task so you can do things with functions. But, a loop is a good place to start.

```
mid_year_by_year <- data_frame(ccode = numeric(),
                               year = numeric(),
                               dispute = numeric())
```

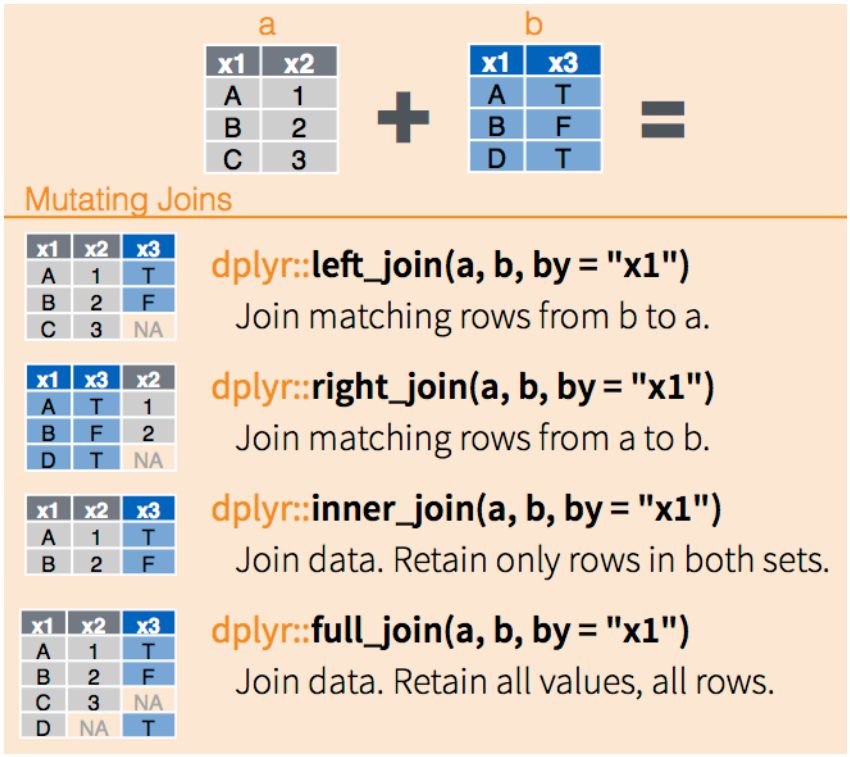
```
Warning: `data_frame()` was deprecated in tibble 1.1.0.
i Please use `tibble()` instead.
```

```
for(i in 1:nrow(mid)) {
  x <- data_frame(ccode = mid$ccode[i], ## row i's country
             year = mid$StYear[i]:mid$EndYear[i],  ## sequence of years for dispute in rou
             dispute = 1)
  mid_year_by_year <- rbind(mid_year_by_year, x)
}

head(mid_year_by_year)
```

```
# A tibble: 6 x 3
  ccode  year dispute
  <dbl> <int>   <dbl>
1   200  1902       1
2   200  1903       1
3     2  1902       1
4     2  1903       1
5   345  1913       1
6   300  1913       1
```

13.6 Merging

We want to combine these two datasets by merging. Base-R has a function called `merge`. `dplyr` has several types of `joins` (the same thing). Those names are based on SQL syntax.



Here we can do a `left_join` matching rows from `mid` to `polity`. We want to keep the rows in `polity` that do not match in `mid`, and label them as non-disputes.

```
p_m <- left_join(polity,
                 distinct(mid_year_by_year),
                 by = c("ccode", "year"))

head(p_m)
```

```
# A tibble: 6 x 6
  scode ccode country  year polity2 dispute
  <chr> <dbl> <chr>   <dbl>   <dbl>   <dbl>
1 FRN     220 France   1800      -8      NA
2 FRN     220 France   1801      -8      NA
3 FRN     220 France   1802      -8      NA
4 FRN     220 France   1803      -8      NA
5 FRN     220 France   1804      -8      NA
6 FRN     220 France   1805      -8      NA
```

Replace `dispute` = `NA` rows with a zero.

```
p_m$dispute[is.na(p_m$dispute)] <- 0
```

Reshape the dataset long to wide

```
p_m_wide <- pivot_wider(p_m,
                        id_cols = c(scode, ccode, country),
                        names_from = year,
                        values_from = polity2)

select(p_m_wide, 1:10)
```

```
# A tibble: 4 x 10
  scode ccode country       `1800` `1801` `1802` `1803` `1804` `1805` `1806`
  <chr> <dbl> <chr>          <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
1 FRN     220 France            -8     -8     -8     -8     -8     -8     -8
2 GMY     255 Prussia          -10    -10    -10    -10    -10    -10     NA
3 GMY     255 Germany           NA     NA     NA     NA     NA     NA     NA
4 USA       2 United States      4      4      4      4      4      4      4
```

13.7 Main Project

Try building a panel that would be useful in answering the Democratic Peace Question, perhaps in these steps.

Task 1: Data Input and Standardization

Often, files we need are saved in the `.xls` or `xlsx` format. It is possible to read these files directly into `R`, but experience suggests that this process is slower than converting them first to `.csv` format and reading them in as `.csv` files.

`readxl`/`readr`/`haven` packages(https://github.com/tidyverse/tidyverse) is constantly expanding to capture more file types. In day 1, we used the package `readxl`, using the `read_excel()` function.

Task 2: Data Merging

We will use data to test a version of the Democratic Peace Thesis (DPS). Democracies are said to go to war less because the leaders who wage wars are accountable to voters who have to bear the costs of war. Are democracies less likely to engage in militarized interstate disputes?

To start, let's download and merge some data.

- Load in the Militarized Interstate Dispute (MID) files. Militarized interstate disputes are hostile action between two formally recognized states. Examples of this would be threats to use force, threats to declare war, beginning war, fortifying a border with troops, and so on.
- Find a way to **merge** the Polity IV dataset and the MID data. This process can be a bit tricky.
- An *advanced* version of this task would be to download the dyadic form of the data and try merging that with polity.

Task 3: Tabulations and Visualization

1. Calculate the mean Polity2 score by year. Plot the result. Use graphical indicators of your choosing to show where key events fall in this timeline (such as 1914, 1929, 1939, 1989, 2008). Speculate on why the behavior from 1800 to 1920 seems to be qualitatively different than behavior afterwards.
2. Do the same but only among state-years that were invovled in a MID. Plot this line together with your results from 1.
3. Do the same but only among state years that were *not* involved in a MID.
4. Arrive at a tentative conclusion for how well the Democratic Peace argument seems to hold up in this dataset. Visualize this conclusion.

14 Functionals and Optimization

14.1 More Tidyverse/Data Cleaning

For our main example in this session, we'll take a look at the the 2022 Cooperative Election Survey (CES). The `ces22.dta` file contains a subset of questions asked to respondents in the 2022 wave of this annual survey stored in Stata's `dta` format.

```
ces <- haven::read_dta("data/input/ces22_subset.dta")
```

Stata files often contain metadata for each column. In the CES, responses are stored as numeric values that represent one of the discrete choices available to the respondent. As a result, while many of the columns have numeric data types, they should not be treated as numbers. This is something to be careful of as you work with any dataset – always read the codebook.

For example, while we could take the "mean" of the race variable, this is a nonsense quantity.

```
mean(ces$race)
```

```
[1] 1.696817
```

We can take a closer look at how the race variable is stored:

```
head(ces$race)
```

```
<labelled<double>[6]>: Race
[1] 1 1 1 1 1 7

Labels:
 value             label
     1             White
     2             Black
     3          Hispanic
     4             Asian
```

```
     5   Native American
     6 Two or more races
     7             Other
     8    Middle Eastern
    98           skipped
    99         not asked
```

And take a closer look at its class

```
class(ces$race)
```

```
[1] "haven_labelled" "vctrs_vctr"     "double"
```

`haven` reads in the Stata metadata and creates a column that has the `haven_labelled` class, which stores both the numeric labels and the metadata which denotes the mapping between the numeric label and its meaning. We'd like to use this to convert the variables to factors that we can work with in R. To do this, we'll be using the `as_factor()` function. This function is "generic" and has a specific implementation for `haven_labelled` objects in the `haven` package

```
head(as_factor(ces$race))
```

```
[1] White White White White White Other
10 Levels: White Black Hispanic Asian Native American ... not asked
```

Or in "tidy" style

```
ces %>% mutate(race = as_factor(race))
```

```
# A tibble: 60,000 x 28
     caseid commonweight commonpostweight gender4 educ    race  hispanic pid3
      <dbl>        <dbl>            <dbl> <dbl+l> <dbl+l> <fct> <dbl+lb> <dbl+l>
 1   1.98e9        3.65             3.53   1 [Man] 6 [Pos~ White 2 [No]   1 [Dem~
 2   1.98e9        0.780            0.819  1 [Man] 3 [Som~ White 2 [No]   3 [Ind~
 3   1.98e9        0.892            0.774  2 [Wom~ 5 [4-y~ White 2 [No]   1 [Dem~
 4   1.98e9        1.10             1.21   3 [Non~ 6 [Pos~ White 2 [No]   4 [Oth~
 5   1.98e9        0.543            0.328  1 [Man] 6 [Pos~ White 2 [No]   3 [Ind~
 6   1.98e9        0.114            0.0989 1 [Man] 5 [4-y~ Other 2 [No]   3 [Ind~
 7   1.98e9        0.899            1.21   2 [Wom~ 2 [Hig~ Black 2 [No]   1 [Dem~
```

```
 8   1.98e9        0.633           0.431  1 [Man] 6 [Pos~ White 2 [No]   1 [Dem~
 9   1.98e9        0.900           0.854  2 [Wom~ 5 [4-y~ White 2 [No]   1 [Dem~
10   1.98e9        0.725           0.586  1 [Man] 6 [Pos~ White 2 [No]   1 [Dem~
# i 59,990 more rows
# i 20 more variables: pid7 <dbl+lbl>, inputstate <dbl+lbl>,
#   CC22_306 <dbl+lbl>, CC22_320a <dbl+lbl>, CC22_320b <dbl+lbl>,
#   CC22_320c <dbl+lbl>, CC22_320d <dbl+lbl>, CC22_320e <dbl+lbl>,
#   CC22_320f <dbl+lbl>, CC22_320g <dbl+lbl>, CC22_320h <dbl+lbl>,
#   page_CC22_320grid_timing <dbl>, CC22_333 <dbl+lbl>, CC22_333a <dbl+lbl>,
#   CC22_333b <dbl+lbl>, CC22_333c <dbl+lbl>, CC22_333d <dbl+lbl>, ...
```

But we'd like to do this for every column in our dataset (except for columns that are numeric). We could manually go through every single column and convert it to a factor, or we could use some tidyverse tools.

The `across()` function allows us to select columns using the same custom selection syntax as in `select()`. We can combine this with the mutate function to mutate every column that we want to transform. Here, we want to select the columns that are of a certain **class** – the "labelled" class.

```
ces <- ces %>% mutate(across(where(is.labelled), as_factor))
```

`across()`, like `select()` is very flexible in how columns can be selected. In addition to selecting by number, you can select by partial string match, numerical sequence, or even regular expressions!

Now all of our columns operate correctly as factors.

```
ces
```

```
# A tibble: 60,000 x 28
   caseid commonweight commonpostweight gender4 educ  race  hispanic pid3  pid7
    <dbl>        <dbl>            <dbl> <fct>   <fct> <fct> <fct>    <fct> <fct>
 1 1.98e9        3.65             3.53  Man     Post~ White No       Demo~ Stro~
 2 1.98e9        0.780            0.819 Man     Some~ White No       Inde~ Lean~
 3 1.98e9        0.892            0.774 Woman   4-ye~ White No       Demo~ Stro~
 4 1.98e9        1.10             1.21  Non-bi~ Post~ White No       Other Inde~
 5 1.98e9        0.543            0.328 Man     Post~ White No       Inde~ Inde~
 6 1.98e9        0.114            0.0989 Man    4-ye~ Other No       Inde~ Inde~
 7 1.98e9        0.899            1.21  Woman   High~ Black No       Demo~ Not ~
 8 1.98e9        0.633            0.431 Man     Post~ White No       Demo~ Not ~
 9 1.98e9        0.900            0.854 Woman   4-ye~ White No       Demo~ Stro~
10 1.98e9        0.725            0.586 Man     Post~ White No       Demo~ Not ~
```

```
# i 59,990 more rows
# i 19 more variables: inputstate <fct>, CC22_306 <fct>, CC22_320a <fct>,
#   CC22_320b <fct>, CC22_320c <fct>, CC22_320d <fct>, CC22_320e <fct>,
#   CC22_320f <fct>, CC22_320g <fct>, CC22_320h <fct>,
#   page_CC22_320grid_timing <dbl>, CC22_333 <fct>, CC22_333a <fct>,
#   CC22_333b <fct>, CC22_333c <fct>, CC22_333d <fct>, CC22_333e <fct>,
#   page_CC22_333_timing <dbl>, page_CC22_333grid_timing <dbl>
```

We can do some more steps to make our data easier to work with. First, we can rename columns to be more informative. For example, `CC22_306` is a question on vaccination status. Let's `rename()` that column to make it easier to reference

```
ces <- ces %>% rename(vaccinated = CC22_306) #New aname = old name
```

`vaccinated` has a number of different levels

```
table(ces$vaccinated)
```

```
                                                I am fully vaccinated and have received at least one booster sh
                                                                                                            338
                                                      I am fully vaccinated but have not received a booster sh
                                                                                                            105
I am partially vaccinated (I have received the first of two shots for either Pfizer or Moderr
                                                                                                             19
                                                                                   I am not vaccinated at a
                                                                                                            134
                                                                                                          skipp

                                                                                                       not ask
```

Let's recode this categorical variable to a binary indicator for whether a respondent is fully vaccinated. Note that two of these categories have "fully vaccinated". We want to code these as a value of 1 and the rest as 0. To do this we'll use the `case_when()` function inside of a `mutate()`

```
ces <- ces %>% mutate(fullyvax = case_when(str_detect(vaccinated, "fully vaccinated") ~ 1,
                                           !str_detect(vaccinated, "fully vaccinated") ~ 0
                                           is.na(vaccinated) ~ NA_real_))
```

How many respondents are fully vaccinated?

```
table(ces$fullyvax)
```

```
    0     1
15423 44424
```

`CC22_320` contains approval ratings of various political institutions. Let's pull the three big ones (President, Legislature, Supreme Court) and create indicators for whether the respondent approves or disapproves of that institution.

```
ces <- ces %>% mutate(approvePresident = case_when(str_detect(CC22_320a, "disapprove") ~ 0
                                                   str_detect(CC22_320a, "approve") ~ 1,
                                                   str_detect(CC22_320a, "Not sure") ~ 0)
                      approveCongress = case_when(str_detect(CC22_320b, "disapprove") ~ 0,
                                                   str_detect(CC22_320b, "approve") ~ 1,
                                                   str_detect(CC22_320b, "Not sure") ~ 0)
                      approveCourt = case_when(str_detect(CC22_320c, "disapprove") ~ 0,
                                                   str_detect(CC22_320c, "approve") ~ 1,
                                                   str_detect(CC22_320c, "Not sure") ~ 0)
```

Let's pull those three columns, party ID and the survey weights and analyze them further.

```
ces_approval <- ces %>% select(commonweight, pid3, approvePresident, approveCongress, appr
```

Functionals

Many functions in R act on functions as input. You've already seen a lot of `tidyverse` functions that do this. For example, the `summarize()` function takes as input a dataset and a function or functions describing what operations should be done on the dataset. Above, we used `mutate` which took the functional arguments `across` and `as_factor`.

R is a *functional programming* language in that functions are "first-class citizens" and can be treated as any other data type. They can be given a name, passed as inputs to other functions, and returned as output.

This allows programs to be written in terms of compositions of functions. In fact, this is one of the principles of the `tidy` project as outlined by Hadley Wickham, and many of the constructs provided by the tidyverse encourage you to work this way. See the tidy manifesto for more.

14.1.1 Replacing for loops

R programmers tend to favor functional replacements for `for` loops. Rather than iterate over elements of a vector or list and execute some operation for each iteration, you can use one of the `apply` (Base-R) or `map` (tidyverse) functions.

To illustrate, let's take a look at our approval rating data from before. How would we write a for-loop to calculate the (weighted) mean for each column?

```
approval_data <- ces_approval %>% select(approvePresident, approveCongress, approveCourt)
for_time <- proc.time() # Store current time -- used for timing speed of function
approval <- rep(NA, 3)
names(approval) <- c("approvePresident", "approveCongress", "approveCourt")
for (name in names(approval)){
  approval[name] <- weighted.mean(approval_data[[name]], ces_approval$commonweight, na.rm=
}
print(proc.time() - for_time)
```

```
   user  system elapsed
  0.014   0.000   0.014
```

```
print(approval)
```

```
approvePresident  approveCongress     approveCourt
       0.4131196        0.2635909        0.3600206
```

This is pretty terrible looking. We've already seen the `summarize()` function work for this task, but let's illustrate another operation in base-R, the `apply` function. `apply()` operates on matrices and applies a function to each row or column.

```
for_time <- proc.time() # Store current time -- used for timing speed of function
approval <- apply(approval_data, 2, function(x) weighted.mean(x, ces_approval$commonweight
print(proc.time() - for_time)
```

```
   user  system elapsed
  0.008   0.000   0.008
```

```
print(approval)
```

```
approvePresident  approveCongress     approveCourt
       0.4131196        0.2635909        0.3600206
```

Note that there aren't really substantial speed benefits. Rather, the code just looks cleaner. There are some common functions that **are** faster, but they have been specifically optimized for speed. Consider `colMeans` or `rowMeans` for simple (unweighted) means.

Also note how we defined an "in-line" function within the `apply()` call. This is sometimes called a "lambda" or "anonymous" function. We won't be able to call it outside of the `apply()` call since it has no name, but there's not ever a reason why we would want to.

`sapply()` works on generic lists while `tapply()` can also apply a function based on the value of an index.

However, if you are using `tidyverse`, you probably should prefer the equivalent "generic" version of `apply()` – the `map` family of functions. They vary primarily in how they return their output. `map()` always returns a list while other forms (like `map_dbl()` or `map_chr()`) will force this list to be a vector of the specified type.

```
approval_data %>% map_dbl(function(x) weighted.mean(x, ces_approval$commonweight, na.rm=T)
```

```
approvePresident  approveCongress     approveCourt
       0.4131196        0.2635909        0.3600206
```

You can use `map()` on a vector to iterate over a function repeatedly

```
map(1:10, function(x) sample(1:100, 1)) # Sample a number from 1:100 ten times.
```

```
[[1]]
[1] 3

[[2]]
[1] 70

[[3]]
[1] 65

[[4]]
[1] 32

[[5]]
```

```
[1] 90

[[6]]
[1] 12

[[7]]
[1] 40

[[8]]
[1] 15

[[9]]
[1] 32

[[10]]
[1] 86
```

Note that when applying functions to grouped columns of a data frame, you should probably still be using `summarize()` for most cases. `group_map()` has more flexibility, but can be a bit more difficult to implement.

```
ces_approval %>% group_by(pid3) %>% summarize_at(vars(contains("approve")), ~ weighted.mea
```

```
# A tibble: 5 x 4
  pid3        approvePresident approveCongress approveCourt
  <fct>                  <dbl>           <dbl>        <dbl>
1 Democrat              0.837           0.507        0.205
2 Republican            0.0646          0.113        0.607
3 Independent           0.338           0.185        0.346
4 Other                 0.270           0.121        0.343
5 Not sure              0.248           0.150        0.170
```

14.2 Excerise: Optimization

In this section, we'll illustrate another valuable use of functional programming – optimization routines. Many tasks require finding maxima or minima of functions. Most functions of interest are very difficult to analyze analytically and rarely does a **closed-form solution** for the optima exist. However, a large number of methods exist that allow you to find a numerical solution to this problem.

Consider the following function.

$$f(x_1, x_2) = -x_1^2 + 2x_1 - 2x_2^2 + 3x_2 + x_1x_2 + 2$$

Let's it's maximum using numerical optimization

```
polynom <- function(x){
  return(-x[1]^2 + 2*x[1] - 2*x[2]^2 + 3*x[2] + x[1]*x[2] + 2)
}
```

We will use an implementation of the `BFGS` algorithm. This is an extension of the classic Newton-Raphson method. To find the maximum of a function, this approach starts with an initial guess x_n. Then, each subsequent step updates x_n by taking steps in the direction of the gradient, scaled by the Hessian. In a single dimension, the update step is:

$$x_{n+1} = x_n + \frac{f'(x_n)}{f''(x_n)}$$

For minimization, we replace the plus with a minus (as we want to move **away** from the gradient).

`BFGS` is implemented alongside many other algorithms in the `optim` function that is part of Base-R. `optim` takes as input a set of initial parameters, the name of the function to be optimized. A function that returns the gradient can also be specified, but this is optional. If not provided, `optim` routines that use the gradient will approximate it using a finite difference method (evaluating the function at small changes in the inputs).

```
max_1 <- optim(c(0,0), polynom, method = "BFGS", hessian=T, control=list(fnscale=-1))
```

Note that we have set `hessian=T` to return an evaluation of the hessian matrix at the critical point. We have also set this as a **maximization** problem by using the `fnscale` argument in `control`. By default, `optim()` minimizes the function. `fnscale` flips the function so that minimization is maximization of the original function.

Let's see the output

```
max_1
```

```
$par
[1] 1.571269 1.142798

$value
[1] 5.285714
```

```
$counts
function gradient
      10        6

$convergence
[1] 0

$message
NULL

$hessian
     [,1] [,2]
[1,]   -2    1
[2,]    1   -4
```

The x solution is stored in `$par`, the value at the maximum is stored in `$value`. `$convergence` describes whether the optimization routine converged or not (0 = success). We can confirm that this is a maximum by showing that the Hessian is negative definite and that all the eigenvalues of the Hessian are negative.

```
eigen(max_1$hessian)
```

```
eigen() decomposition
$values
[1] -1.585786 -4.414214

$vectors
           [,1]       [,2]
[1,] -0.9238795 -0.3826834
[2,] -0.3826834  0.9238795
```

Note that starting values matter. Choosing a starting point far away from the true solution means that more steps are required to reach convergence. Here, the function is well-behaved enough that even choosing $x_0 = \{1000, -100\}$ doesn't increase the number of steps too much, but some functions can be very poorly behaved (near-zero gradients) for some inputs.

```
max_2 <- optim(c(1000,-100), polynom, method = "BFGS", hessian=T, control=list(fnscale=-1)
max_2
```

```
$par
[1] 1.571428 1.142848

$value
[1] 5.285714

$counts
function gradient
      25       10

$convergence
[1] 0

$message
NULL

$hessian
     [,1] [,2]
[1,]   -2    1
[2,]    1   -4
```

Challenge problem:

Consider the following function.

```
myfun <- function(x, mu=5, sigma=10){
  return(((x*sigma*sqrt(2*pi))^-1)*exp((-(log(x) - mu)^2)/(2*sigma^2)))
}
```

Using `optim`, find the maximum of `myfun` for parameters `mu=5` and `sigma=10`.

Hint: `myfun` is defined only over the positive reals, but `optim` assumes the inputs are unbounded (at least for `BFGS`). Try to transform the inputs such that `optim` will work (you can do this with a lambda function).

Now, find the maximum for parameters `mu=7` and `sigma=5`

Challenge Problem 2:

Write a function that returns the gradient of `polynom`. Pass this function as as an argument to `optim`. Compare the speed of the optimizer when the gradient is known in closed form vs. when it is approximated.

15 Simulation[1]

Motivation: Simulation as an Analytical Tool

An increasing amount of political science contributions now include a simulation.

- Axelrod (1977) demonstrated via simulation how atomized individuals evolve to be grouped in similar clusters or countries, a model of culture.[2]
- Chen and Rodden (2013) argued in a 2013 article that the vote-seat inequality in U.S. elections that is often attributed to intentional partisan gerrymandering can actually attributed to simply the reality of "human geography" – Democratic voters tend to be concentrated in smaller area. Put another way, no feasible form of gerrymandering could spread out Democratic voters in such a way to equalize their vote-seat translation effectiveness. After demonstrating the empirical pattern of human geography, they advance their key claim by simulating thousands of redistricting plans and record the vote-seat ratio.[3]
- Gary King, James Honaker, and multiple other authors propose a way to analyze missing data with a method of multiple imputation, which uses a lot of simulation from a researcher's observed dataset.[4] (Software: Amelia[5])

Statistical methods also incorporate simulation:

- The bootstrap: a statistical method for estimating uncertainty around some parameter by re-sampling observations.
- Bagging: a method for improving machine learning predictions by re-sampling observations, storing the estimate across many re-samples, and averaging these estimates to form the final estimate. A variance reduction technique.
- Statistical reasoning: if you are trying to understand a quantitative problem, a wonderful first-step to understand the problem better is to simulate it! The analytical solution is often very hard (or impossible), but the simulation is often much easier :-)

[1] Module originally written by Connor Jerzak and Shiro Kuriwaki

[2] Axelrod, Robert. 1997. "The Dissemination of Culture." *Journal of Conflict Resolution* 41(2): 203–26.

[3] Chen, Jowei, and Jonathan Rodden. "Unintentional Gerrymandering: Political Geography and Electoral Bias in Legislatures. *Quarterly Journal of Political Science*, 8:239-269"

[4] King, Gary, et al. "Analyzing Incomplete Political Science Data: An Alternative Algorithm for Multiple Imputation". *American Political Science Review*, 95: 49-69.

[5] James Honaker, Gary King, Matthew Blackwell (2011). Amelia II: A Program for Missing Data. Journal of Statistical Software, 45(7), 1-47.

Where are we? Where are we headed?

Up till now, you should have covered:

- `R` basics
- Visualization
- Matrices and vectors
- Functions, objects, loops
- Joining real data

In this module, we will start to work with generating data within R, from thin air, as it were. Doing simulation also strengthens your understanding of Probability (Section @ref{probability}).

Check your Understanding

- What does the `sample()` function do?
- What does `runif()` stand for?
- What is a `seed`?
- What is a Monte Carlo?

Check if you have an idea of how you might code the following tasks:

- Simulate 100 rolls of a die
- Simulate one random ordering of 25 numbers
- Simulate 100 values of white noise (uniform random variables)
- Generate a "bootstrap" sample of an existing dataset

We're going to learn about this today!

15.1 Pick a sample, any sample

15.2 The `sample()` function

The core functions for coding up stochastic data revolves around several key functions, so we will simply review them here.

Suppose you have a vector of values `x` and from it you want to randomly sample a sample of length `size`. For this, use the `sample` function

```
sample(x = 1:10, size = 5)
```

```
[1]  6  9  2  1 10
```

There are two subtypes of sampling – with and without replacement.

1. Sampling without replacement (`replace = FALSE`) means once an element of `x` is chosen, it will not be considered again:

```
sample(x = 1:10, size = 10, replace = FALSE) ## no number appears more than once
```

```
 [1]  1  4  2  9  5  6  7  3 10  8
```

2. Sampling with replacement (`replace = TRUE`) means that even if an element of `x` is chosen, it is put back in the pool and may be chosen again.

```
sample(x = 1:10, size = 10, replace = TRUE) ## any number can appear more than once
```

```
 [1] 10  1  7 10  4  1  7  5  2  2
```

It follows then that you cannot sample without replacement a sample that is larger than the pool.

```
sample(x = 1:10, size = 10, replace = FALSE)
```

```
 [1]  9  6  1 10  8  5  4  7  3  2
```

So far, every element in `x` has had an equal probability of being chosen. In some application, we want a sampling scheme where some elements are more likely to be chosen than others. The argument `prob` handles this.

For example, this simulates 20 fair coin tosses (each outcome is equally likely to happen)

```
sample(c("Head", "Tail"), size = 20, prob = c(.5, .5), replace = TRUE)
```

```
 [1] "Tail" "Head" "Head" "Tail" "Head" "Tail" "Head" "Head" "Tail" "Tail"
[11] "Tail" "Tail" "Tail" "Head" "Head" "Head" "Tail" "Tail" "Tail" "Tail"
```

But this simulates 20 biased coin tosses, where say the probability of Tails is 4 times more likely than the number of Heads

```r
sample(c("Head", "Tail"), size = 20, prob = c(0.2, 0.8), replace = TRUE)
```

```
 [1] "Tail" "Tail" "Tail" "Tail" "Tail" "Tail" "Tail" "Tail" "Tail" "Tail"
[11] "Tail" "Tail" "Tail" "Head" "Head" "Tail" "Tail" "Tail" "Head" "Tail"
```

15.2.1 Sampling rows from a dataframe

In tidyverse, there is a convenience function to sample rows randomly: `slice_sample()`

For example, load the dataset on cars, `mtcars`, which has 32 observations.

```r
mtcars
```

```
                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Merc 450SE        16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 450SL        17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Merc 450SLC       15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Fiat 128          32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic       30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla    33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Toyota Corona     21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
Dodge Challenger  15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
AMC Javelin       15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
Camaro Z28        13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
Pontiac Firebird  19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
Fiat X1-9         27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
```

```
Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

sample_n picks a user-specified number of rows from the dataset:

```
mtcars[sample(1:ncol(mtcars), size=3, replace=F),]
```

```
              mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Valiant      18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Mazda RX4 Wag 21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Merc 240D    24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
```

```
slice_sample(mtcars, n=3)
```

```
                  mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Fiat X1-9        27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Merc 280C        17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
```

Sometimes you want a X percent sample of your dataset. In this case use the `prop` argument to `slice_sample`

```
mtcars %>% slice_sample(prop=.5)
```

```
                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Ferrari Dino      19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
Lotus Europa      30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
Maserati Bora     15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
Toyota Corolla    33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Merc 450SLC       15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Volvo 142E        21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
```

```
Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
```

A common task in statistics is to generate "bootstrap" samples of your data. This involves sampling a new dataset of equivalent size (`n` rows) but **with replacement** (so some observations are dropped; others might appear 2 or 3 times). You can do this with `slice_sample` as well

```
mtcars %>% slice_sample(prop=1, replace=T)
```

```
                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Mazda RX4...3       21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Merc 280...4        19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
Fiat 128...6        32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
Porsche 914-2...8   26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
AMC Javelin...9     15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
Chrysler Imperial...10 14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Porsche 914-2...15  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
Merc 450SL...17     17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Fiat 128...18       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Merc 450SL...19     17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Porsche 914-2...20  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
AMC Javelin...21    15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
Mazda RX4...22      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Chrysler Imperial...23 14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Mazda RX4...25      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
```

```
Chrysler Imperial...26 14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Merc 230               22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Ford Pantera L         15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
Merc 280...29          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Toyota Corona...30     21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
Toyota Corona...31     21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
Maserati Bora          15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
```

As a side-note, these functions have very practical uses for any type of data analysis:

- Inspecting your dataset: using `head()` all the same time and looking over the first few rows might lead you to ignore any issues that end up in the bottom for whatever reason.
- Testing your analysis with a small sample: If running analyses on a dataset takes more than a handful of seconds, change your dataset upstream to a fraction of the size so the rest of the code runs in less than a second. Once verifying your analysis code runs, then re-do it with your full dataset (by simply removing the `sample_n` / `sample_frac` line of code in the beginning). While three seconds may not sound like much, they accumulate and eat up time.

15.3 Random numbers from specific distributions

`rbinom()`

`rbinom` builds upon `sample` as a tool to help you answer the question – what is the *total number of successes* I would get if I sampled a binary (Bernoulli) result from a test with `size` number of trials each, with a event-wise probability of `prob`. The first argument `n` asks me how many such numbers I want.

For example, I want to know how many Heads I would get if I flipped a fair coin 100 times.

```
rbinom(n = 100, size = 1, prob = 0.1)
```

```
  [1] 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 0 0
 [38] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [75] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0
```

Now imagine this I wanted to do this experiment 10 times, which would require I flip the coin 10 x 100 = 1000 times! Helpfully, we can do this in one line

```
rbinom(n = 10, size = 100, prob = 0.5)
```

```
 [1] 46 41 50 50 59 53 57 49 51 56
```

runif()

`runif` also simulates a stochastic scheme where each event has equal probability of getting chosen like `sample`, but is a continuous rather than discrete system. We will cover this more in the next math module.

The intuition to emphasize here is that one can generate potentially infinite amounts (size `n`) of noise that is a essentially random

```
runif(n = 5, min=0, max=1)
```

```
[1] 0.01051054 0.57140770 0.35731914 0.36474572 0.28763365
```

rnorm()

`rnorm` is also a continuous distribution, but draws from a Normal distribution – perhaps the most important distribution in statistics. It runs the same way as `runif`

```
rnorm(n=10000, mean=0, sd=2)
```

```
   [1]  1.9375519482 -0.9540334151 -0.7534708879 -1.4081970595  3.5492358192
   [6] -0.4155053782  0.8162373598  0.8342929864 -0.5619006484 -4.2741471637
  [11] -1.9476655404  2.5611890733  4.4058477775 -3.2815761498 -1.0610735991
  [16]  4.6971848164  0.7735608013 -0.5783967960 -3.8359939029 -0.0653062253
  [21]  0.1732250075  3.0798254468 -0.5603321520  0.6797845511 -1.3479874734
  [26]  0.4128801465 -3.1266007194  1.6878228770  1.1641032740 -0.1098841094
  [31] -0.6048624273 -2.3260148197  2.3663816274  0.7631114817 -3.1680263095
  [36] -0.0695743219 -1.0704521008 -2.0791962920 -3.3549409823  1.2845562252
  [41] -6.5520017890 -2.6993084219 -0.7367391651 -0.4651679936 -1.9338206109
  [46]  0.9096974920 -0.8915106367  0.1230721171 -1.4075471394  1.8191713075
  [51] -1.8847131815  0.4222355173 -1.7848302717 -2.6041597323  3.2300314733
  [56] -1.3520953614  0.4102561984 -0.3779930944 -0.3796896464 -2.4585258010
  [61] -1.3121744754 -0.4198547335  2.0257756216  0.1431198529 -0.4470239082
  [66]  2.1734726390  0.2391735211  1.4296845849  2.0626790180  0.9375847801
  [71]  1.6886017909 -0.4339843837  0.7848730825 -2.2479333648 -1.0438704864
  [76] -1.6567851412 -5.6079877980 -5.3271919237 -0.1974825494 -1.8080343019
  [81] -2.7890296449 -0.5264763941  2.5729469330 -1.5445478759 -4.6016363057
  [86]  0.6272035855  1.0339879749  0.0853034436  2.9253329576  0.2300794122
```

```
 [91] -5.2406570291 -3.2365860198  3.0330234700 -2.6055787567  1.9815945627
 [96] -0.8010812382  1.6253754138 -2.7625208350  0.8461988223  0.0213522201
[101]  0.1438321137  1.7551074462 -3.0135942270 -2.6321579755 -1.5499692586
[106]  2.0295556821  0.1527271453 -2.0021673814  2.7303155125 -4.6226688534
[111] -0.7655667970 -1.8669603555 -0.5610156687  2.4015248246  1.2118840037
[116]  2.8950430792 -0.1058957014 -2.0023969949  1.6954220271 -2.6493615431
[121] -0.1269152861 -2.1090428334  1.5990967983  0.9220079346  0.1869982691
[126]  2.1262473075 -3.3417038745  3.2802900481 -2.3909569810 -1.8910402874
[131]  0.7599058920 -1.6127018989 -1.9723127697 -2.2583793398  0.1592506061
[136]  2.9522045823 -0.7889590424 -1.9036359270  0.4664674183  1.8587322176
[141]  0.5012044684  2.7989786903  1.1512650364 -1.1751496188  0.0883863807
[146]  1.9298222803  3.8002731770  4.3519829748 -0.4928142498  1.0048878567
[151] -2.9383719289  0.7640221575  0.9347427731  4.7494556227 -2.8642498205
[156] -2.6679409076  1.0837169820 -1.8835173694 -0.1829837455  0.0330336129
[161]  0.9023969224  0.1855621760  2.5776543347  3.1910834721 -1.6547497269
[166]  1.1810091008  1.4029592271 -2.1628814089  1.8355351053  1.6320147806
[171]  0.2154173628 -3.7411595443  2.2742466109  2.0842002880  1.7845601080
[176]  0.7564505284 -0.6725914792  0.2112203253 -1.7452824794 -4.4256638217
[181]  3.4346935797 -1.0775546201  3.5013522689  0.3850494414  3.4930652396
[186] -0.6670379495  2.2769456044  3.5855572951 -0.9353746997  1.0282454508
[191] -0.6829984603 -1.7031158586 -2.8168913183 -2.4065460883 -2.4481741157
[196] -0.5251163728 -1.8252410915 -0.7367736184 -3.1732991568  2.9660503485
[201]  0.2108545736  2.8490119923  0.5099095698 -0.5946282050 -2.2859574906
[206]  0.2025519219  0.2654909929  0.3770217652 -2.9033472338 -1.9545739878
[211]  0.5860187632  2.5850224218 -5.3383940341 -3.4852959837 -4.2691022349
[216] -0.0793222914 -2.7604769240 -1.8257749681 -2.5813788660  0.1295375574
[221]  1.0994991824 -0.0519838640 -4.4449749839 -2.0752947690 -2.0022579840
[226] -1.5614822915 -0.4311067547 -1.2749709604 -2.4062663809  1.7709243853
[231] -1.8872831708  1.5893795258  1.0569160199  0.0875719645 -0.2666160958
[236]  1.8650617029  3.0332721997  0.8993540491 -0.2730765508  2.0404513252
[241] -1.1291950674 -0.8896783383 -2.1156137494  1.6256604752  0.2547582569
[246] -2.0474459966  2.7330183341  2.4868525328  0.9441151818 -1.1927228555
[251] -4.1293340388  0.9851407149  1.7427525871  4.4476947999 -1.5126782946
[256] -1.0500366354  0.0891290215 -1.4353091897  1.6926189589 -2.3475342701
[261]  0.2503115454 -1.1148784005  1.0470506468  0.5177279220 -1.4293647374
[266] -1.3684126453  0.2299728811  0.3527042827  0.2039131973  0.9694345561
[271]  1.9451120444  0.4798668250  3.2892903536  2.6759446562  3.4025726311
[276] -0.0168438828  0.8513404844 -2.2514249829 -0.6992332109 -0.8573035293
[281] -2.3080037839  0.4104704523  2.2001694002 -0.7986695264  0.4695348044
[286]  3.3328123709  2.6812737593 -0.2179384616 -1.2110450657  0.9348195251
[291]  0.8307989173  1.3534833720 -0.3466172208 -0.2489639133 -0.3437002265
[296]  2.1732947058 -2.9957698817  1.2922316987  2.1082736652  0.1068149976
[301]  1.1095845497 -0.8917112373 -3.2667344815  3.2987417583 -3.7647342290
```

```
[306]  2.6769625885 -0.2997453678 -1.7673831333 -2.1503191736  1.9182534434
[311] -4.0169292033  1.7202711232  0.4289589603  2.6867555186  0.7327525065
[316]  0.2541240892 -3.0647261400  0.3830260150  1.1963807586  1.3312469192
[321] -1.0789139458 -2.1654192620  1.3915682233  1.6537256455  0.6937066462
[326]  1.2336017839  3.9007790426  1.8971681207 -1.2216264143  1.8449113966
[331]  0.4491364100 -1.7580806014  3.8591417130 -0.1898995839 -2.6587378998
[336]  2.4667409573 -1.2793108408 -1.1967242769  0.9562779207 -1.6437380390
[341]  1.1743132949  0.3710191424 -0.0932713581 -0.2919188729  0.9116824494
[346] -2.4535709146  1.8079772272  3.6306147096 -0.8998827648 -1.1662314546
[351] -0.1806260889 -2.4002242197  3.0252548443 -0.1042774137  0.7235187809
[356] -0.6230245073  1.3193049169 -3.8703984846 -2.2438435389  4.5904504417
[361]  2.4530972789 -2.1989473532 -1.1519296447  2.1938101309 -0.1627613753
[366] -2.1037437854  2.2306442490  0.4631960384 -0.4889035181 -0.9188889213
[371]  2.9899107096 -0.1173444333  1.2666383410 -1.2515669860 -1.0301426788
[376] -1.0677562425 -2.1605907618  0.6813513524 -2.3816831550  2.0370061069
[381] -1.2717426222 -0.4069419205 -1.1546075375  3.6415992290  0.7626236252
[386]  2.0965335058  3.8282967611 -2.5082853097  2.4565705875  0.6327312178
[391]  3.1370499199 -1.8145625630  1.4623195016 -0.3787013964 -6.3218102615
[396]  1.6781256270  1.0203825710 -2.1525982738  1.2111608618  0.2640390201
[401]  4.5889860916 -0.1866694872 -0.6838013811  4.6467238871  0.6299037569
[406]  1.5867992140 -0.3273860812  0.8364895268  0.1548857835  1.6738819604
[411] -0.9844279434 -0.0974906376  1.2679357927  2.1728838220 -0.7697846758
[416]  0.7960631508 -1.4760725724 -2.5877871423  1.3228674727  0.8217398643
[421]  1.1068832562 -1.8405598906  2.8949540107 -0.2908328354  0.8779765437
[426]  3.2649532021 -1.2083187313  0.9754919601  2.7935028206 -1.7355948488
[431] -0.8518509465  0.1023892764 -1.9222004015  3.0868810787  1.6231406788
[436] -1.6250443350 -1.8068863636  2.4187479861  2.7309617523  1.7921560796
[441]  0.6270239473  3.7863308761  0.3459435574 -2.8433439447  3.5145549922
[446] -1.7227966115  5.1483590308  1.6144993815  0.5064387892  1.5656640500
[451] -2.2067997371  0.9959488174  0.7823957021 -1.4068340017  4.0446087880
[456] -1.7253674089 -0.6652301756  2.3341395149 -1.7061106629  1.6426498482
[461] -3.9400178597 -0.8515121997 -1.5862609171  4.5991457142 -0.8071605554
[466]  1.6508365372 -0.0645276138  2.0275884944 -1.9966796165  2.2648495928
[471]  1.0546578525 -1.7296848763 -2.6343873473 -0.2401317747  0.2239516598
[476]  6.7062139730  1.1022999910 -3.2346896463 -0.3912174497 -3.6931039811
[481] -1.0138429050 -0.2620367169  4.7013543385 -0.5725323662 -1.5394085922
[486]  2.4137735291  1.3533939354 -0.3423991146 -1.4185190643 -0.3301015347
[491]  3.1814424743  5.4376920810 -0.0761613999  1.4683516593  1.9432624200
[496]  3.0905460616 -0.8889828754 -1.6611998498 -2.9798570858 -1.7947010998
[501] -1.0704940601  1.0693259762 -0.0806591684  1.3938837048 -1.9455564778
[506] -0.1179718788  0.8662954493  1.2563394740 -0.1626640271  0.7724302333
[511]  0.8211812118 -4.0587665471 -2.1063445873 -2.1946146975  0.1907788741
[516]  2.4522303522  2.6727943762 -0.0792371912 -3.7196601624 -0.3822321560
```

```
[521]  3.3791810000 -1.4791792292  1.7477540206  0.3850595767 -0.0788140652
[526]  0.7876023949 -1.8398894881 -1.8423521858  1.9687080135 -4.8169359463
[531] -0.6866787976 -0.4076023243  0.3742866420 -0.5403272657  2.0836297311
[536]  3.0357936703 -2.1398653021  3.9052316822 -0.5399912044  2.0369682844
[541] -3.8504241433 -1.4503939749 -0.8360681903  0.7466578365  1.2596175019
[546]  0.9233189649  4.0046500081 -1.2795001263  2.7892801672 -2.3242421773
[551] -0.3205640800  0.2270119213  0.0568879502  0.8871420748 -0.0936461385
[556] -1.7192439125  2.1832802311 -3.4636771257  0.8091143891  0.4161382943
[561] -1.3670981525 -1.0865129444 -3.6377794896 -0.7285511346 -2.2392403473
[566] -1.0434563320 -1.2533771653 -2.7268828790 -2.7253279056 -1.7420594181
[571] -0.7082614789 -0.4753587037 -1.1382277622  3.1846219856  2.3200854080
[576]  0.7528564937  1.6250144142  1.8377818006  1.7678789941  2.1927487481
[581] -1.5900686350 -1.3113974323 -1.2150738234  0.7409961111 -0.9424167207
[586] -0.3717974755 -3.0265119215 -3.8315015091 -0.3579538116 -2.1824670149
[591]  0.5955120045 -0.8403241213  0.8190498649  2.1521533830  1.1453708875
[596]  1.0226721513 -0.9735747347  0.9684526722  0.0349973663 -1.2881895276
[601]  1.2064830980  2.0480273857 -2.8354026625  1.5349797122  1.7397546717
[606] -0.8139191203 -1.8660010276 -0.9981449535  2.3750672786 -1.5286680142
[611] -0.1780193409 -3.4250976889  2.0596770771 -2.3797759715  1.6853223307
[616] -0.1432630532 -2.0937851823  0.7277688287  0.2776566508 -2.5892125071
[621] -0.1187942842  1.0017500359 -1.6910859286  1.7125180505 -0.2072016461
[626]  0.0949226810 -1.8254772168 -1.1576550719  0.7178861643  2.1364484687
[631] -3.3362289226 -3.7914471256  0.8884004304 -1.4065075225  3.3946958439
[636]  0.7911513457 -3.5370734600  0.5010358909  1.4750204706 -3.5604509897
[641] -0.2276270543  2.8771490644 -1.8990379790 -3.4933146639 -0.1062186431
[646] -1.2543207920  1.2311603012 -1.0650139050 -0.9119555458  0.4167479602
[651]  1.0464983751  1.9107140451  1.0503193899 -1.1659796871  3.8017476383
[656] -4.9326588286 -3.0854576291 -3.3422461332 -0.8977664777 -1.4886693627
[661] -1.6211267182 -3.8403881934 -0.8210833742 -0.0162812611 -0.9381447218
[666] -0.1234736622  1.8751865683 -0.3474623915  0.7519905354 -2.1953005135
[671] -1.7673698595  2.5724246166  1.9889852224  4.3374040710 -2.2418697923
[676] -2.5058064157  0.5519068823  1.8925303019  1.0529876424 -1.0750109938
[681]  1.4883617628  0.4455718235 -1.7615801990 -0.6523387286 -0.2645818828
[686] -1.8524684749 -5.7446254872 -2.0334807922  0.9114442321  1.0801428534
[691] -2.4114352316 -1.6473333925 -0.5695421550  2.1587976598 -1.3370153632
[696] -0.5637252009 -3.6231919699  2.0828256174 -4.5413774863  0.7533798697
[701]  2.2488524748  4.9600064561  5.4988658225  2.1440885878  1.9894616673
[706]  1.4395378754  0.6607755012 -0.4024176327  0.4445071365 -2.1206231932
[711] -1.5111008802  2.4557455286  4.3471639729  0.2301513914  0.6554548939
[716] -1.7101072640 -1.1920991448 -1.3197442430 -0.2976215519  3.2295271356
[721] -2.9321084069  0.6581527791 -2.1654155739  0.9953368598 -0.0733718250
[726] -1.5451673511  1.0263913899 -0.2933060592 -0.9483970635 -0.8242822443
[731] -0.2739517506  0.5718645198  0.0500914880 -1.7149243884 -0.9552667304
```

```
[736]  0.6242573691  0.6366190263  2.4408171514  1.0180195467 -2.3174022272
[741] -0.6540634295  0.4836314232  0.7533919737  0.6687449505  3.2819701762
[746]  0.1140478363 -1.0319411137 -1.1260343220  2.7495333863 -0.3793238131
[751] -1.2328332819 -3.7805050843 -0.1977919310  3.5172695852  0.1774049513
[756]  1.4830955794  3.3598903578  0.0080504262 -0.8448232443  0.1064592869
[761] -1.4086621440  2.8174985253 -1.0649643027  1.6228249132  0.1042530867
[766] -1.3784672194 -1.6510185758 -0.8613501073 -0.2207473521  1.7076791318
[771] -1.1177984358 -3.5234127836  2.5021365825 -0.4020700311  1.5585709751
[776]  1.8474370050 -3.9668137667  1.1838631696 -1.7262175688 -2.1639483006
[781] -1.1463721381  2.2684559077 -0.2839795357  0.6166927777  0.9101150295
[786]  1.4199548935 -1.1188113972 -2.6654344619  1.9215375288 -0.8273385447
[791]  0.0650447131 -2.6936128340 -2.5508986430 -2.3769381382  0.7901334275
[796]  0.3927777976  2.3130277690 -1.7561234547  3.1517893404  0.2837779250
[801]  2.2991008433 -0.3565685475 -1.4018965243  3.4776548349 -0.4195628561
[806] -0.7946090247  2.1991605522  1.4090065003 -2.0719966241  0.1793722093
[811]  0.0672084435 -0.9901835563 -0.8151430578  1.8266203060  0.5037994523
[816] -2.9957326274 -0.3590301910  0.7176790788 -2.6194955962 -1.2754404818
[821]  1.3673903176  5.6692603556 -0.4405893563 -0.0273318931  2.4416375438
[826] -0.7203748149 -0.0394718777  1.1572389686 -0.1686920894  0.9914990894
[831] -3.7055174434 -3.6699738765  2.4739986625  1.3400526081  0.3915782908
[836] -2.9008666315 -1.4039053678 -3.8102609422  2.3571903093 -1.6471258373
[841]  2.4735883477  1.8044120092  1.6494534384 -1.1916021785  1.0250934613
[846] -1.5604816189  2.4599492231 -2.1549220464 -3.7017474526  0.9311839669
[851] -1.1534183865 -1.9386556256 -1.9659618088  1.3682719628  0.3904858790
[856] -3.0300586978  0.7600079179  1.4996217430  1.1867268700  1.3119824226
[861] -0.2762953514  1.4700657994  1.5603445377 -0.3774398633 -2.6310984355
[866] -2.5881055654  1.0002686715 -0.7055510530 -1.0880269799  4.9343269633
[871]  3.1231473653 -1.6078049548 -1.6780317083 -1.6191251246 -1.9161272464
[876]  3.4998282811  0.9014779530  0.1506638314  1.1267699600 -0.5443979883
[881] -3.3506852122 -0.8910567204  0.7671029996 -1.9448875011 -1.0309890816
[886]  0.7798109980 -0.7308992205 -0.8756491351 -0.5057115209 -1.1969948196
[891] -1.0806846205  0.0573953076  1.1731272074 -1.7725872302  2.2347480932
[896]  0.5607187992  3.3051322305  2.1375499588  2.3744430865  0.4391914048
[901]  4.4982508878 -2.5618893418 -1.3313026796 -0.3373879370  0.1948399844
[906]  0.3944784145  2.1776211043  1.1892407379 -0.5341230747 -0.9055224051
[911] -0.5343921225 -0.9170008139  0.0530116935  0.1629362239  0.1973118162
[916]  5.0035342007  0.4942143396  0.4201062254  0.5667661209 -0.6942121752
[921]  2.2838865652 -0.6704681285  1.6004714067 -0.8534240169 -0.8238768476
[926]  0.6464139054  0.4782397336 -4.2967711438  0.3129432653 -1.4424403821
[931] -1.6991114606  0.1531716198  1.4534666030  2.0122803215  2.5512454496
[936] -0.1807174496 -2.7954219370 -0.7640346805  0.0966779177 -3.3267887682
[941]  1.7487249631 -3.9120775576  1.5727618606 -5.3937995206 -1.8988506715
[946] -2.0910820101  1.7862009134 -0.1557136787 -0.5642322493  2.9654635194
```

```
 [951]  0.7058999412  2.4571915753  2.1700209691  0.7743146294  3.3034553022
 [956] -1.8073212800  0.3804658075  1.4369269097 -0.6343903060  0.6949027383
 [961]  2.0751901822  2.1702309027  1.2646168132 -0.5708190116 -3.7524751806
 [966]  1.2867762920  0.2937196520 -2.3951282857  4.0217006042  1.7319020356
 [971]  2.7272485226 -0.2678637573  2.2546177874 -1.9692630458  3.4011623546
 [976]  1.5086455587 -0.4180090093 -1.3936289696 -0.7297489545 -2.4793724732
 [981] -0.4788370173  0.7547289483 -0.8673554780  0.9088341889 -0.7044120200
 [986]  2.2497416312  0.3016355886 -0.6780244044 -0.6503797793  0.7528514126
 [991] -0.3649340191  2.5460526087 -0.3483457568  3.5275219593 -0.9522015373
 [996] -0.3890209598  2.5985839698  1.2436193373 -1.2196051546 -2.4229229693
[1001]  0.9373278016 -2.5862550207  0.1467812030 -0.6640304931 -0.0386544614
[1006]  4.2553345489 -1.6266407162 -0.5807897915  0.4756166027  0.4826802316
[1011]  0.6718648213  2.9437531473 -0.4802421159 -2.4998377034 -3.0561413356
[1016]  0.5165518497  3.7054997930 -0.2634337360  1.5201993280  0.2985784808
[1021]  0.6309401159 -1.9251479351  0.2819149693 -0.8361971727 -2.7797698457
[1026]  0.0964237359 -0.3716704270  0.6339126856 -1.8502569041  2.6043719949
[1031] -0.8926131618  0.1909364637 -2.3755163627  1.5475982249 -0.4590548269
[1036] -4.3077737406 -4.2633511378  1.6331389526  1.1834868081  1.0930738963
[1041]  1.8936236874  1.4769682122  0.8287467132 -4.3821896128 -0.8285171175
[1046]  1.2803569626  0.2163714605  3.3537556640 -1.2821010350  1.3376201092
[1051]  2.7541746815  1.5822867787  1.4004430213  1.5992085638  2.2592122567
[1056] -0.2093380225  1.2090653912 -1.5350729813  0.6371876757 -0.2814121988
[1061]  0.7367295492  2.5854197240  0.6232308523  3.3716178867 -1.2588772134
[1066] -0.1648861475  0.1637062166 -0.7518093839  3.0236712738 -2.1594928061
[1071]  2.1088402677 -1.2195402182  0.2480208452 -2.4281326827  0.3264274954
[1076] -1.8656788406  0.2472368534  0.4240519969 -1.9448457062 -2.4024204364
[1081]  0.0110317652  0.5323006494 -1.9124937980 -1.4949542966 -0.6650408173
[1086]  0.0431003738  0.6929435711  2.5143897380  0.2704690711  3.4252221489
[1091] -0.5065932734  1.4255418441 -0.5335897671 -1.7909521785  4.2687033703
[1096]  2.6248412639  2.4822244372  0.9639636321  1.1478959965  1.4735764960
[1101] -2.7293686828  2.6514090318 -2.5224590980  2.9207140157  2.4526029844
[1106]  1.0828285312  0.3611561838  4.7325274549 -0.8733940102 -0.8582768042
[1111] -0.8561471132  1.1150432167 -0.3718014479  0.9176519725  1.9705329908
[1116] -2.5150100032  5.4454952142 -1.8581315938 -2.1002676773 -1.5188143187
[1121] -0.8350757800 -1.4793950952  0.8141469071 -1.8723437848  1.5484118392
[1126]  1.2736293945 -0.7304738814 -2.1826951076 -0.9776175839 -0.6664998410
[1131] -0.6306744270 -0.5782672536 -0.6755491091 -1.0142010085 -1.5275910887
[1136] -1.5453425548 -0.9157791022  0.8784267608 -2.0799400628 -0.3502716335
[1141]  3.3350663639 -3.1230396021 -2.8351223529  2.1555935418  1.7290945495
[1146]  1.1998069878  0.9060235620 -0.8415465487  0.8193593468 -2.2210961564
[1151] -2.2354862493  0.0605920169  2.1957749116  0.2791598666 -4.6358717778
[1156] -1.8655589771 -0.0248889434  0.4071344424  1.6921712058  0.2341366155
[1161] -2.2053272877 -0.4564891272 -1.7486225909  3.5397330412 -0.2596207802
```

```
[1166]  0.3693265283  2.1999962935  2.5167500036  1.3061649945 -0.4614449116
[1171] -3.2486537764 -2.1093185411 -0.4190026730 -2.8294419482 -0.4207367405
[1176]  0.0873673265  3.6064367531  1.1791853087 -0.3353474457  5.1886371757
[1181] -2.7148786093 -1.2446775749  2.3750837544  3.3981634787 -0.4275008129
[1186] -1.8384628222 -0.2895907194 -0.5961000654  2.7914801878  1.0262689524
[1191] -0.3288754659  2.5302183757 -0.4896853183  2.3684918114 -0.1124746992
[1196] -0.4866508010 -0.5248394142  1.9848618334  1.4403570613 -0.4932348123
[1201]  3.2953617234  1.0466320620  2.8651911813  1.2136961599  0.9765643030
[1206]  0.3434433582 -1.9138790825 -1.4782506252  0.1898653301  1.9800098522
[1211] -2.7137049067  1.1772069802  2.6925938731 -2.0355424736 -0.5306690179
[1216] -2.1917669148  0.4580275090  0.7094422618 -0.2265336561 -2.0464034253
[1221]  4.8441449186  1.1742487755 -1.0683709616  3.1502353063  0.8016902850
[1226] -3.3358021619  1.5668957869 -0.7767996749  3.1880886084 -2.5919903842
[1231]  0.8704609381 -2.5684301395  0.2587282204  0.2737595919  1.9367100972
[1236]  0.4235275501 -3.2423676130 -0.0613506022  2.7218950286 -0.7082260178
[1241] -0.0818777748  1.2463654806  0.5248908385  0.5496912101  2.9249018728
[1246]  0.3260203120 -2.4394416127 -0.7347460631 -0.1064411781 -0.1995156969
[1251] -1.9091400670  2.1375660360 -2.4399705093 -0.9994420487 -1.5508966711
[1256]  0.0836378585 -0.5831608515  1.6109522795 -0.0656309996 -0.3752062951
[1261] -3.1197740530  1.7500096457  0.2552997846 -0.0867133456  2.1338155437
[1266] -1.2437660681 -2.7013602986  0.1286862251 -4.0558430210  0.0109638287
[1271] -2.5474147802  1.6473634931 -0.3740298914 -1.5550380701  3.0025211608
[1276] -0.7990780325 -1.3258314658 -1.2158520052  0.4292474182  1.4643001054
[1281]  0.1759286130 -1.2265808553 -1.5580299702 -0.0133399215 -2.6775625322
[1286]  0.8144682985 -1.0440380560  0.6341569426 -3.1131002715  0.0226358579
[1291]  0.8820423512  4.5969428275 -0.5700365119  5.5595099712 -4.3091585359
[1296] -0.7770955647 -1.5408412299 -1.2548570532 -5.4806021638  0.3786064974
[1301]  0.1652302039  4.6640258815  0.7503809487  0.3549599700 -1.6617313400
[1306]  4.0568315753  0.4335674624  2.1057415430 -1.3062757953 -2.7271547403
[1311] -0.2662597650 -0.5490422266 -2.5864849477  2.7905292373 -0.7201803563
[1316]  3.2129379145 -0.2747946937  1.2008343150  3.5359854620  2.7721657740
[1321] -1.5493487890 -0.5865980922 -0.8185997588 -3.5005389529 -0.3650905027
[1326] -3.7685725473  1.6835740903 -5.0253881844 -2.2712552980  0.9944672870
[1331]  3.5380468982  1.7028835458  3.0801407538 -0.6532546363 -1.5702083088
[1336] -3.3995855976  1.1645679613  0.7155151048 -2.1583997305 -3.4255402140
[1341] -2.0865416576 -5.3772308230 -0.2804505099 -0.0528955251 -2.0893251502
[1346]  1.3769533918 -0.1609775557 -1.3808392264  0.7139094673  0.6804623250
[1351]  1.7839215516  1.4102221196  1.1283281978 -2.2297240012  3.0377218868
[1356] -0.3894922537  0.3757603646  2.8794725512 -0.0721626050  1.3793607143
[1361]  0.8559877752 -1.7678690639 -0.7929001591 -3.8143360803 -0.7628309848
[1366] -3.7121683142 -1.7645971526  1.6550362554 -2.2331395313 -2.1448329425
[1371] -3.3287804977 -4.0584149879  3.5974309353 -2.3268970507  1.3589068341
[1376] -0.0797773746  0.9623623954 -1.6318535470 -0.1801530951  2.2250713629
```

```
[1381] -2.0087219808  1.7761597921  2.2876813492 -0.1726145497  0.1098040713
[1386] -0.9245637022  1.8028464983  0.5561746334  1.6290363704 -0.0265409753
[1391]  0.4629955083  1.6519054922  1.4251741985 -4.2017862728 -2.4731176879
[1396] -2.1652852647 -2.5812121550 -0.6139434233 -2.0188424010  1.2505603769
[1401] -2.7293002996  2.7115957547 -0.7050589941  1.9265603729  1.5231590388
[1406]  2.4262709325 -2.2835960210 -1.6186909867 -1.3073142895 -1.7196661398
[1411]  2.2777432731  3.2588488957 -0.4127721437 -3.2317759795 -1.0617227475
[1416] -0.0875438489  2.0959394769 -1.4468964136  4.6628234273  2.9371096583
[1421] -3.0003885720 -3.6354688525  0.4327943354 -2.5420053090 -2.2009325215
[1426] -3.2800908242  0.2719290650  2.7102748584 -0.1145006952  1.7871828506
[1431]  2.6092236606 -0.6464298555 -1.3923093470 -5.2381701721  2.8134949342
[1436] -1.2496618564 -2.1327807341 -0.3330905126 -0.2963460032 -1.0773946141
[1441] -1.2613456761 -1.0412424885 -1.6549315531  0.6687966529 -2.0682622247
[1446] -0.2296258479 -0.1974844113 -2.4827441881 -3.8739098329  1.6163779259
[1451] -2.6664729715  2.3848636691  1.4997379055  1.0771128423  3.4555621400
[1456]  0.2554426730  0.1727337064 -2.5365379531 -1.7451193639  2.7316357638
[1461]  0.0062956866 -1.3509662526  2.5413005989 -1.0757952230 -2.2397414519
[1466] -2.2104664125 -1.3146255069  1.1545218581 -1.2937527927 -3.3976643028
[1471]  0.2413401246  1.8013450657 -0.3605208326  0.4600959941  0.7625135123
[1476]  1.7218829258  2.4417783242 -2.7258809486 -1.4280700226  2.7575237374
[1481] -0.9447052399 -3.1209583276 -0.2306366321 -1.2417007142  1.0689066648
[1486]  2.6787970311  3.4641945315 -0.4373324519 -0.6050691648  3.8419120359
[1491]  1.1474242891  1.3462084904  2.7752242291  2.1964021819  3.6851465480
[1496] -1.1387391992  1.9123983629 -1.8204959286 -2.2112994196 -3.6111410514
[1501] -0.5721050175 -1.9470404805  1.4363322896  0.7403122295  0.4727221869
[1506]  2.3333356217 -0.1135283892  1.0165316325  2.4398203463  2.9020281577
[1511]  0.5546443678 -1.5433737646  0.2697015584  1.6994439882 -0.0255823842
[1516]  0.4933179296 -3.8290847905 -1.0475347849  1.3243588819 -2.6432543742
[1521] -0.0552948503 -1.4564908249  0.1467224848  0.6962288115  0.2107493655
[1526]  0.0638957460  0.1788156147  3.4236523230  1.4820567357 -0.9234772712
[1531]  1.1845813709  0.9302743988  1.0542285191 -1.0528397829  1.1975988726
[1536]  2.2222375577  0.0260536486 -1.1998049482  2.9137067089  2.3254978968
[1541] -2.0296544171  3.0902081092  0.0663614464 -2.7622101586 -0.0392145389
[1546] -0.0884219880  1.0742722676 -2.6859008951 -3.4250705322 -0.2856980620
[1551] -0.3173303694 -0.5850226882  3.9710832218  2.2666033197  1.7750773058
[1556]  3.2954343960  1.6739095595  0.2411069417  2.3440988358 -0.7031124753
[1561]  1.8989850919 -0.4749596050  0.5083532895  2.4857678529 -0.2869504838
[1566]  1.2487023585 -0.4670253096  1.7662324343  1.0107337525  1.6618360582
[1571]  0.3821915195 -1.6856509609 -3.2009002676 -2.3575880396  0.7043205821
[1576] -0.5084430490  4.0289993183 -1.5722680212 -2.2217711964  0.0952820501
[1581] -1.1752605797  1.4980911674 -1.7250074641 -2.2497045124 -4.6492142736
[1586]  2.4889757482 -2.5625777737 -1.1455722904  1.8223614229 -3.4967686574
[1591]  2.3821603962  2.4837819835  1.4271958654  0.4420756799  1.4378851529
```

```
[1596] -3.1224545575 -1.6077850044 -1.2718817572  0.2617765563 -1.4552626971
[1601]  0.4066695508  0.2913932982 -5.0060713055  0.2008581285 -0.9156841970
[1606] -0.0138752303  0.8530284866 -1.1565715453 -2.1218009337 -0.6801261507
[1611]  1.3074410100  1.3604927938 -0.7772054237  0.0517024795 -3.1093496377
[1616]  2.4223159877 -0.7205096349  1.1890476588 -2.1232061705  0.1889809266
[1621]  2.0121259738  0.5134177603  5.2528915966  1.6681869251  0.0294401916
[1626] -3.4727803749  2.4378849097 -0.0338510083  1.0720959779 -1.3036089562
[1631]  1.0086056572  0.6602250076  1.3339159161 -0.8487133590  0.3885424064
[1636]  2.5306678635  0.9166450197 -4.8579459337  0.4493865941 -1.6102367328
[1641] -2.2327472664  2.4191853816  1.3597719722  2.4172539163 -2.3416318739
[1646] -4.4273478024  0.7678620430  1.7145388076  2.0648578364  0.2628117034
[1651]  1.3475918946  2.3216902479  0.1425561629  0.5366731777  1.8588528706
[1656] -0.3032521296  1.8244123534 -0.1246882560 -0.5407023969  0.6880975713
[1661] -4.0387315265  0.0625525588  2.3491560866  1.2691739947  1.4341738717
[1666] -0.4903861183  0.7826629994 -2.7728790470 -1.0229079433 -1.2433730263
[1671] -0.9147015406 -3.8092052843  0.4800664523 -1.3635616636 -0.1456653840
[1676] -1.2370409202 -3.8720597047 -0.1741914239 -0.1647943047  0.0396013394
[1681]  1.6026780096 -0.7306724398 -0.2720853565  1.4116204348 -0.2634819897
[1686]  3.1436159091  1.1236265774 -0.6177377352 -3.5397436955 -0.8390642226
[1691] -0.0349443237  0.8289748484 -1.2635428497  1.4400199766  2.9004884947
[1696] -4.4998353465  1.0716737842  1.1712532530  2.1092156054  3.7143335041
[1701] -0.4555416536  2.8436234674 -0.3906318824  0.3211713415  1.2992259935
[1706] -2.2977128795  0.3476451697  0.2686647127  1.7252766497 -0.3016098887
[1711] -2.2667442215 -1.1054819086 -1.2935339941 -0.2763523266 -1.8492341767
[1716]  2.1537244446  0.5762182793  2.9186984294 -0.6788367499 -1.8389339231
[1721] -0.5805968589 -0.7566677580 -0.9058391275  2.3569308604 -2.1073513411
[1726]  3.1783869713 -1.9231640383 -3.7827180229  2.3948000149 -0.8247901690
[1731] -2.2076435001 -0.8970210991 -2.3134753797 -1.8297938570 -0.5179796716
[1736] -0.1961759054  0.1701928632  1.2242550851 -2.6833129287  2.6715662907
[1741] -0.8113254717  0.7859419296  0.1940708653  1.6132262372  1.2408667692
[1746] -2.5608278877  3.8215347229  1.3695631823 -0.9984094033 -1.2173401714
[1751]  2.5586909339  2.6190024616 -0.6109383135 -0.3808103885 -2.2851189976
[1756]  1.6985423873 -1.5662909009  0.1453047485  0.4230090323  1.3537223479
[1761]  3.1631018281  4.1487521127 -1.5052334231 -0.5854894769  0.6090741911
[1766]  1.2854202567  1.8785321892 -1.2264168159  2.7508076562  0.0408393039
[1771]  0.3502350585  0.2558830502 -0.6992064639  0.9007323405 -1.8441387910
[1776] -1.6596335487 -1.9789234101 -1.5233833036  0.1140633763  1.4807005066
[1781] -1.8224152921  4.0778968099  1.1836578358 -0.7828620578 -0.0864080590
[1786]  0.0069584456 -2.3144813507  0.0821233298 -1.5389380797 -0.4998517497
[1791] -0.6461969652  2.7850750036 -1.3980573735 -0.8149253195  1.1367720079
[1796]  2.2487912535 -1.0040997976  1.1426013465 -0.6780013001  2.5870861998
[1801]  0.1474660309 -0.7940415016  5.1431010441 -1.1690843125  0.8327499581
[1806]  1.2976093979  0.9278746925  0.9801879797  1.3461858269 -3.5057735920
```

```
[1811] -0.6481886743 -1.0177177332 -1.6600019642 -1.2615819453  0.2898582631
[1816]  0.9702144602 -3.2857474703  0.6293265865 -0.3803877891  0.4189255525
[1821] -2.3355143324  2.1977040858  2.1275328052  1.0672064588 -2.7960808647
[1826]  0.3865246157 -0.5065432697  0.9380882652 -2.8623867687 -2.9956120632
[1831]  0.3687545926 -1.1518884222  1.8696754748  2.7521972505 -3.6307186660
[1836]  1.4336921268  1.5010175274 -2.2163145521  0.0665293637  4.1070997949
[1841] -1.2344522616 -0.3941746417  0.0986008842 -1.6078592632  3.0630442821
[1846]  0.3647363392  0.3118346404  0.4656687991 -0.4657567639 -3.5575774902
[1851]  0.9396133172  0.0947088686  1.4616944497  0.8069555175  5.2501463536
[1856]  2.2228710713  0.8258306635  1.3086205089  5.0541031781 -2.5172157257
[1861] -5.6597822572 -0.3629927116 -2.1987837083  2.1924171299  1.2212759420
[1866] -2.3507378832  1.7080578564  1.0137748356 -0.0106262824 -2.7858296497
[1871]  1.4803605391  2.8324749258  1.7182326513 -4.7981432251  0.1938912557
[1876] -1.5591824295  1.3660671501 -2.1520883733  1.5270597142  1.3702035970
[1881] -0.1053923795  4.1266017979 -1.7700278868  0.2386625866 -2.1690773349
[1886]  2.8764044113 -5.1645581491 -1.7658853081  0.0578345083  2.8884143511
[1891]  2.5302084997 -1.6711989611  4.5080015869 -1.4741996956  1.8781691962
[1896]  0.6244405656  2.6693297543  3.6752595869  3.9053016890 -2.3463731157
[1901] -0.6219058209  2.4826941411 -0.8358692146  0.6109156412  0.2907451898
[1906]  0.1406117884 -3.0827271478 -0.4443644819  1.1609067799  0.6132377722
[1911] -1.0589005639 -0.3414026539 -1.5984874334  0.9056688444  1.0815996474
[1916]  1.0045057551  0.8450718519  4.0995125489  1.3694586501  0.2704057865
[1921]  2.0402225718  2.0197416059  0.1446682068 -1.7397746765 -0.0547260130
[1926]  1.6738701018 -3.2054164273 -1.1233737672  0.4720995257 -2.2405366910
[1931] -0.7096304187  0.2220256177 -0.4979130389  2.1264062542  3.6464318806
[1936] -2.6278290693 -2.4071689533  1.2658451492  3.2042999481  4.6452091182
[1941]  1.0889573252  0.7146392794  2.6778201294  2.2900261362 -3.2561006352
[1946] -2.1676095537  1.4511028993  2.1336551822 -0.7399980830 -1.3962100109
[1951]  2.8861982747 -4.5136873994  1.0319059132  0.2054826005 -0.5937480503
[1956]  2.6101449138 -2.5025436306  1.5119418752  3.5145001888  0.3624348909
[1961]  0.7873026280  0.1639700219 -0.1626997797 -1.2724766536  0.3924150290
[1966]  1.2523030647 -2.2760520534 -1.1907958102  2.7214395487 -5.2256428090
[1971]  2.8600085653 -0.5854847482  1.7085172501 -0.0395481479 -0.6809361153
[1976] -0.8598841844 -2.4772126533 -2.1300947112 -2.2404003388 -2.1407713263
[1981] -2.5151030424  1.2440723976 -0.6839536463 -2.7437155499  0.1796545445
[1986]  2.3429552995 -1.6999287177  0.1419507990  3.1496589260  4.1499516752
[1991] -1.9254250463  1.2633559584  1.9571871613  2.2429810517  1.3786719635
[1996] -1.2112022187  1.8024110195  4.2317470293 -0.5039345044 -0.1977781127
[2001]  0.0925759700 -2.4406737400 -2.8334122970  2.0487200136 -0.4962184626
[2006] -0.7407012838  0.0955082926  0.3524408022  1.4601585333 -1.5265541666
[2011] -0.1518943338  0.0098717353 -3.9376167201 -1.2201051161  1.1207521436
[2016] -1.6865844333 -0.2538410676  0.6140613986  1.8785139984  1.3157967044
[2021]  2.7795269769  1.5396444220  2.4177373002 -0.0013600565  1.4947758200
```

```
[2026]  1.2078858515 -1.0455597660  1.0453196160  0.8217344215 -1.4903922351
[2031] -1.4361430622  1.8371916235 -0.1949920940  0.4353049184 -1.2664861426
[2036] -4.2060147791 -2.1948194609  1.4376160400  0.0798709141 -2.0476511589
[2041] -2.6526375603 -0.5842991164 -2.9732650167 -0.4595831981  0.2300326086
[2046]  0.3825817228  2.7414360788 -0.5940370923  1.7262833645  0.0837647364
[2051] -0.0018110020 -0.4078975830  1.0383585356 -0.9856435892 -1.7785703300
[2056] -0.4591708525 -1.5581150637 -1.0842465151 -0.8908817662  0.3772947207
[2061] -0.2665529890 -3.2319054114  1.9379562525  0.5440382142 -0.6530916986
[2066] -0.4315008860  2.2582347203 -2.3217105525 -4.1852254951 -0.1871809133
[2071] -0.8376381432 -0.5284948501 -0.2947168447 -1.3361371227 -3.1801606669
[2076] -2.8318110676  0.4963097566 -0.4018214980  0.8314154985 -3.7450925899
[2081] -1.8233990089 -0.5668608273 -0.6480722174 -0.2281549517 -3.2072594340
[2086]  0.2770197136 -1.6003451684 -0.3979153308 -0.0830958795 -0.3623263500
[2091]  0.8123037087 -0.4992309392 -5.2224108437  2.7408517754  0.9601578133
[2096] -4.8521789700 -3.3272427647 -2.4443219948  2.3358935249  0.1233085927
[2101]  0.0601210588 -1.7582357389 -2.5945623326 -3.5433990917 -2.4286998626
[2106] -1.4826827492  1.5664517897 -1.3637332019 -1.0352790475  5.5638362369
[2111] -1.6951511529  4.0456993501  5.2841312775  1.0546281140  2.3874133958
[2116]  1.8372256457  0.1783071727 -0.1603695313 -0.5071626050  2.1390531106
[2121]  0.1177658011 -1.8493372124  1.6249087318  4.3946481444  0.7408414078
[2126] -0.4580097819  3.0433010056 -3.9293021291 -0.6393710571 -2.0404792670
[2131] -0.1018925955 -1.5613090913 -2.8173202833 -1.6338724728 -0.8234389763
[2136] -1.1384174995 -2.3359792256  2.0658293829 -3.2098124146 -1.3717944901
[2141] -4.6213072964 -1.4351099954  0.1221008960 -0.8438856234 -3.5101748998
[2146]  1.1591090036 -0.9385220728  2.0553471208  2.4115393747 -1.9291398039
[2151]  0.1529395349  2.5739491190 -2.3696113280 -2.0287116608 -0.7549673378
[2156] -0.3716822784  0.2062965904  0.1470928892  3.4465884297 -0.5482824054
[2161]  3.0607187219  0.9317005502 -1.1633321698  0.9766951870  0.6301509358
[2166] -1.3841748784 -1.1580455737  2.1062676732  1.2927550504  1.1470532278
[2171]  0.6362896899  0.4555084634 -0.4282121436  2.7370549976 -0.7598773167
[2176] -1.5753687915 -0.3397546438 -3.7074042749  0.9951604061 -1.4138541750
[2181] -0.6292890353  0.4544504496  0.5335392426  1.8722113504  0.6318620525
[2186]  0.8731914023  0.0135850701 -1.7302691334 -3.1571526988 -1.2015942331
[2191]  1.8369921958  0.1265674213  0.3783982110  0.1519836985 -0.0547952497
[2196]  0.9309759177 -2.6593114475 -2.7924383002 -1.5029662412 -1.7527143625
[2201]  1.5355317245  1.1382602914  0.4960290798  1.3057565685 -0.3611598638
[2206]  2.2996529624 -1.4325198482  0.0275403171  0.1708645433 -0.3928290490
[2211]  1.7969196191 -2.5179091211  1.9755876063 -3.2412081882 -0.8143982584
[2216] -1.2179735323  3.0615401405  0.3546426903 -0.0295278754 -1.2885152112
[2221] -1.6562162456 -0.3162402681 -0.7701863843  1.1711582257  0.7389916165
[2226] -1.9678610495 -1.3390395105  3.7548350859  0.6354517816 -0.1794006201
[2231]  7.2914339949 -0.6835581508 -2.5451820732  0.7059360713  0.8685300454
[2236] -0.8434137772 -2.0432448335 -4.6109778990 -1.3045045585  1.5727024182
```

```
[2241] -0.6320095224 -0.1404444988  0.3387320989 -1.6787430950  0.2104701320
[2246]  1.7428453725  1.3288913041 -1.6485161906 -1.9574378633 -4.7305225470
[2251]  1.0558454005 -0.8053109632 -3.3378956777 -1.6997507814  2.5491314819
[2256]  0.4670409826 -0.1262509148  0.5987950296 -0.6178797371  0.5251027509
[2261]  1.1659552493  1.3486700404 -2.4285858611  3.1554102493 -0.2135118600
[2266]  1.6510799464 -1.7771188588 -1.0753315863 -1.8353325970 -2.1265365174
[2271]  4.8328845721 -2.3172907206  1.6696863344  2.6595210344  2.5198646565
[2276]  1.5009528388 -0.2795820699 -0.5834616138  0.7748500665 -0.4032196184
[2281] -0.1010187265  0.7297383655 -0.6481024139  3.8822018483 -0.3373999170
[2286]  2.3349673659 -1.3573214252  0.7846282450 -2.2005597151  0.2018982623
[2291]  2.4001117587  2.5361533354  2.7460219333 -2.9725768786  0.1087280078
[2296]  0.2940120444  0.4842754967  1.8221844432 -2.1731123297  1.1306027123
[2301] -0.4742146134  0.5303920608 -1.9783970261 -0.0473626193 -0.4292624514
[2306] -4.4054334301  1.0655023990  2.7417299160 -2.8935267941  1.6627485424
[2311] -1.1353475834 -1.8203280561 -2.1582684630  0.3626577491 -1.1058267246
[2316]  0.5429016207  0.2300860659 -1.0209574811 -0.4276264090  0.6324780567
[2321]  0.4574859819  3.1872771464  1.0052897401 -2.8142319282  2.0211999097
[2326] -1.6133180226 -1.7418217000 -0.6241283601 -2.7273341370 -1.9553955454
[2331]  1.5694567991  4.7241084188 -2.7003259876 -1.7194346306  0.5101097718
[2336] -1.2494245893 -3.5365164783 -0.8881086574 -1.4945161072  0.7943925555
[2341] -3.6856880212 -0.9621248019  2.4334575128  3.3186963334 -1.3973161189
[2346] -0.4523494463 -1.3871419685  0.4953991201  2.0790505704 -1.1803114761
[2351]  2.0548485589  0.4018921733 -1.6741818312  1.9631750857  1.7288479005
[2356]  0.3385353157  3.9531549998  0.0961255088 -0.5605914475 -2.3293593516
[2361] -3.5710764064  0.3083389475  2.4182556112  2.8623773221  5.4803601869
[2366]  0.2436836542 -1.7015538372 -0.0013369650 -3.8469661921  0.7207342705
[2371]  1.5206865759  0.7584854962  3.7104580296  0.3126380019  3.1579393777
[2376]  0.4862428199 -0.4851490615 -1.5840232148 -3.2567781919 -1.1000290359
[2381]  0.9152259677 -1.4753645090  0.4384249391 -2.7508524306 -0.8880335394
[2386]  0.7668346218  2.1409303458  0.5871264956 -2.0721529232  1.2447784705
[2391]  3.7773633783  2.7736226158  0.6157223021  2.2423579979  1.1743488216
[2396] -1.1360229682 -1.2178708660 -2.7226212093  1.2619496249  2.1985501780
[2401] -3.5814725775  1.9220189331 -0.1880052209  0.1602119179  0.2937355091
[2406]  1.5512551501  0.3639876726  0.6126452686  0.6771418445 -1.8214261198
[2411]  0.0178693964 -0.2849534249  1.3106463683  0.4856058834 -0.3659312489
[2416]  1.0812099207 -2.0090858825 -2.0620517662 -0.9151442859  1.9312714662
[2421]  0.8178415458 -2.7363728765  5.0377868824 -0.3874156513 -0.9232144626
[2426] -0.1932257810 -1.0860002506  2.5414836553 -1.5024272973 -4.0357781779
[2431] -1.6445932847 -4.4108215083  1.1656353882  1.3377103075  2.4991951423
[2436] -0.4361654535 -0.5265738941  0.2172142365 -0.7977720225 -1.4538653728
[2441] -0.2467654284 -2.1743570569 -2.8706515190  1.7579435078  3.0698793273
[2446]  1.4184624241  2.9688226150 -1.8446485757  1.3306023421 -0.2632489394
[2451]  0.1135737997 -0.5019289355 -0.4967471532 -0.3655054871  1.7882152875
```

```
[2456]  1.5055667972 -0.2152051007 -0.4889924160 -2.0487821834 -1.3950321849
[2461] -0.8844753286  4.1469428501  0.2874812352 -1.7827652317  0.6925854839
[2466]  2.3420598843  1.6888794081  2.4572398366  2.9093647273  0.4616568644
[2471] -2.2324807094 -2.3321712046  1.2588433387  1.3208266339 -4.1750643259
[2476]  1.0444066807 -2.9419982555 -1.4190546241  6.7628857029  2.7108779713
[2481]  1.1479681331  1.6455890809 -0.0588417729  0.9266959219  1.8232480663
[2486]  0.3918524915  2.5049518534  0.9123133511 -0.3245474735  1.5420784067
[2491] -2.3160213382 -3.0795370129 -0.6202879810 -1.7607511431 -3.4970095741
[2496] -0.3961703247  0.9748202441  3.1603063373 -2.0616659407  5.6108112684
[2501] -3.2531597069 -0.8254674124  2.6829988405 -2.2790838349 -0.5397209195
[2506] -1.8016150304 -0.2043668901 -0.5100152961  4.3017467135  1.1606541546
[2511]  0.9261307727 -2.0044572504 -0.3656452237 -2.1450063395  0.3359991048
[2516] -0.3046089098 -0.8026798323 -0.3245944952 -0.5566564407  0.8405908209
[2521]  3.3123885458  2.1436750277  0.6896175871 -1.1913648035 -4.0109776390
[2526] -0.5336007792 -0.6476474551  2.4262233979 -0.8848768235 -2.3523658106
[2531] -1.6247928287  1.8428916349 -1.2455276446  1.4520532151 -1.9773416171
[2536] -0.2668950172 -4.9642186327 -0.2860356214 -0.7050155467 -1.0685523491
[2541] -2.5056581354 -0.1962681067 -0.0751468490 -2.2817351433 -0.5784177688
[2546] -0.3421235721 -1.3997598634  0.2801183610  2.2701731692 -4.2728712029
[2551] -2.5291278249  2.6918100384  2.6156975488  2.4209575409  0.1982128237
[2556] -3.1338023811  0.3863405901 -0.4519598642 -0.5257609290 -0.6762281775
[2561] -0.5255560077  1.3263622112  0.2789207618  4.0492333681 -2.9659270986
[2566]  0.6444345230  2.0098124409  0.3619885050 -1.1020150789 -1.2705333278
[2571]  1.3129428363  2.4274205216  0.1294507558  5.1987145437  1.6833845191
[2576]  1.2957729864 -0.6864226945 -1.1685593559  2.3570036807  1.3674884220
[2581] -2.4527896872  0.8467465409 -1.0463635959 -1.5093862933  0.1187238671
[2586] -0.1661744573  2.2978266731  0.4245825282  2.0524829909 -1.6807153940
[2591] -1.7330262673 -0.3162336182 -1.6623334545  1.7009670725  0.4987245027
[2596] -1.0609023276  0.1921957209  0.4756830715  0.3756677046 -3.1384630623
[2601] -1.7092500043 -3.0436188592  0.6301803018  3.6708301135 -2.2111795376
[2606] -1.9489205563  3.1163877722  1.2407031900  0.6888793787 -1.3355873039
[2611]  0.1952589362 -0.1953409416  1.3297604510  0.5074811091  1.7659869851
[2616]  0.5288113473  3.3140022428  1.6402670702  3.2024089753  0.8566282480
[2621]  0.2672375747  2.8254235958 -2.1134520167 -0.1297929060 -2.4064794045
[2626]  0.0334431192  0.7629599518  1.2514265605 -2.2677350766 -0.0571726772
[2631] -1.2827552614  1.3241582009 -1.4593419051  3.4397627891  3.4669305050
[2636]  0.3598083097 -0.9396467470  2.2593127951  0.4895010885 -1.4403045608
[2641] -1.5870050064 -1.8473043461  1.3886433528  1.0264147521  1.7419209415
[2646] -1.0111659912 -1.8122674425 -1.7320845082  1.1341712906  0.9083539185
[2651]  1.4354285001 -1.0624198290 -3.8467584099  1.9569750335  0.5170284312
[2656]  1.9362753701 -1.9505674033  4.4161225197 -0.8114944641  2.4272048944
[2661] -1.2079692853  1.0114385353 -1.0454893314  0.4445539645  0.5840983212
[2666] -2.1531127768  5.2221572804 -1.7477931063  2.1425413556  1.2746477593
```

```
[2671]  2.5565447559  1.7667138594  1.5403916606  2.3086910242  0.7191658443
[2676] -0.4376264218 -2.8649405730  3.4232152678 -3.3208075182 -4.4517343317
[2681]  2.7245828278 -1.7543454980 -0.2263420787  3.6670202748 -0.5593318995
[2686]  2.9471640328 -2.2465828455 -0.1708837891 -0.2560805461 -1.7035624856
[2691]  1.1018517244 -0.3045817339  0.3060932503 -2.9192054133 -0.9924492771
[2696]  2.2465458533 -0.2238380550 -2.7779224544  0.3614816424  1.7536764304
[2701]  4.0704905833 -0.6404115592  1.4462299295  1.7868955795 -0.3020310568
[2706]  0.5973454045  2.1210729933  2.0651157635  0.0644659606 -1.1112429468
[2711]  3.8213479271  1.4038283086 -3.2539081431  1.0652906519  3.1263069989
[2716]  0.3232125743 -1.3929796154  0.8554693526 -0.7259393478  1.0920610030
[2721]  0.0924687516  0.0222568534  3.5812044889  2.4849262746 -2.9126045574
[2726]  4.3190728553 -1.0846247100  0.6732052686  2.5335320676  1.5474337926
[2731]  5.1534563778 -0.5161734357  1.3249222102  2.7749290894 -2.4920954213
[2736]  1.5830283582  0.6469877015  3.2848559991  3.2513216787  0.0265481315
[2741] -0.7085856522  1.2699150351 -0.9465822377  2.9220083798  4.0499289604
[2746] -0.1226139602 -1.4413025446  0.6504541097 -1.8267272067 -0.2681898105
[2751]  3.8165417192 -0.8590291289 -0.3394301244  2.8972427732 -0.5725814257
[2756]  0.5233348610 -1.4019514677  0.8039420841 -3.0611445586 -0.7552612536
[2761] -0.9665367546  0.1230708943  1.9429586424  1.8725354941 -0.5720409714
[2766] -4.2503952507  2.0188506330 -2.4275431850  1.2155857053 -1.1622125645
[2771]  2.6531980276 -0.1827095919  2.3661162504  0.8734360831 -0.9742787458
[2776]  2.4409737834 -1.4957608862  2.5974455058 -0.3645265765  0.6191991939
[2781]  0.1161556178  0.7350241971 -0.3804923595  1.9611406671  0.5277081932
[2786]  0.0044528343 -2.5390361525 -0.2163896643 -0.0722457800 -3.3242115214
[2791]  1.0913943298 -0.1465115834  1.1067406180  0.6612746953  1.3087817067
[2796] -1.5417357882 -0.3398267162 -1.5159520511 -1.2077964248  1.5238141590
[2801] -2.0780913719  4.5316171618 -2.8050798674 -1.3367747555  3.7518900467
[2806]  1.7870665354 -0.8385861334 -0.8989092416 -0.8585568505  2.3581442337
[2811]  0.8487971538  1.4631256098  0.8974893119  3.2926197925  0.7846067980
[2816] -1.1104491427 -0.8149338295 -2.4906108436  2.3560905756 -0.1619187986
[2821]  1.3290142723  1.1060752453  0.7228823582 -4.0365849684 -3.7311438844
[2826] -0.9231677938  0.3796092891 -3.4651368180  0.2697879012  0.6184305953
[2831]  1.5572228506  0.1661878207  1.8619852036 -2.4386358433 -3.6959546732
[2836] -1.0016796912  2.1986550949 -0.0842737130  1.9761639187  2.3369532283
[2841] -0.6246753376  0.0582480381 -0.3744095288  4.0876869497 -0.0364501932
[2846]  3.2494674834  0.3510926780  1.5056597274 -1.8599484140  1.2858183761
[2851] -3.0659474737 -1.2702940009 -0.7422697686 -2.7336447260 -0.9915617478
[2856]  1.9256982885  1.3222840458  1.2929365148 -2.1104614702  0.7013243401
[2861]  1.0502273613  1.2822984072 -0.5553469900  1.5137877002  1.4339407147
[2866]  2.0009253697 -0.5515274208  2.1001345527 -1.2862566409 -0.8896016616
[2871] -1.3331741333 -4.1842468852  2.4312092799 -1.3219358817  0.5464533619
[2876] -1.7599620730  0.7535631693  1.3229816060 -1.1720895905 -3.0109296911
[2881] -2.9906307592 -0.1893436076 -0.6718888566 -0.6334278608  1.8076003605
```

```
[2886]  0.9165970884  4.6604125861 -0.2863355424 -0.6103194692  1.3192778490
[2891]  2.2843727643 -2.4048848915 -1.2175545386  2.3052467212  1.2035939683
[2896]  0.9795834597 -2.6669783207 -2.2090655527 -2.7634322259 -0.8124946143
[2901] -3.2378295249  0.2354105117  0.0115977966  0.6746168496 -0.5349543545
[2906] -0.9759664453  1.0559751114 -1.7603014139 -1.4028584290  1.5852590721
[2911]  1.7329475122 -2.8094866650 -1.9312512932 -1.3649379493  0.0927632108
[2916]  0.3146300878 -2.3327695167 -1.1250111902  3.7525594993 -0.6745812161
[2921]  0.0258504066 -0.9995639905 -0.0246025609 -0.1252875121  2.4125142650
[2926] -0.0545513040 -0.1278134252  0.4891836929  0.6950804184  2.2046744433
[2931]  0.1841138439 -1.8557627099  0.8536999188 -0.4480573979 -1.4190411677
[2936]  1.9590746816  2.4787562607 -0.3726079194 -1.9461822476  1.0805788475
[2941] -1.8639993204 -1.8318364667  1.0848433667 -0.9494664301  1.9685226243
[2946] -2.3523738668 -1.9945811952  2.3238144700 -1.1258460293 -0.7711847346
[2951] -5.5737253624 -0.0188460383 -1.0331917624  0.7308741043  4.6993428885
[2956]  0.1676889780  0.2805393385  0.0477204473 -5.1414208492  1.7535351882
[2961] -1.1246449362  1.6547555688  0.2970708833  0.7617626481 -0.8440396821
[2966] -0.7141875242  0.1450508630 -0.7708110311  1.2001264336  2.7022789896
[2971]  1.4251916521 -1.1294157860  1.5406711979 -0.1767548940 -3.5056584113
[2976]  0.4707758843 -1.2625743072  2.3533095407 -4.5712077164  1.2594058355
[2981] -2.7263060338  0.1125136631  0.3407883179 -1.7858925966  1.0747565261
[2986]  2.2939108289  1.6590368686  1.9217941761 -0.7657599181  2.1716505361
[2991] -2.3087492658 -1.7742957850 -0.2510945735  1.2084733060 -0.3645203349
[2996] -0.1149663198  0.8008753020  3.1660592477 -0.0449629171 -2.6271741426
[3001] -1.8181629534 -0.8433387388  3.9222538158 -0.5602377468  0.1969051062
[3006]  1.8531409439  1.3026365051 -0.7281759713 -2.5907775550 -1.4585068838
[3011]  1.5619683431  0.6683171863 -0.2500445209  1.5173124896 -1.4336554080
[3016] -3.2793362764  0.9916790828  1.5410305023  0.5386188980  1.0504348725
[3021]  0.2074349776  0.4290615901 -0.6413998211 -0.7703222434 -0.5946053100
[3026]  0.2116525230 -2.2109579280 -0.4899506743  0.5399203689  2.6116320418
[3031]  0.2803721282 -0.1944427378 -1.2457311391  0.7393056824 -3.8567755318
[3036]  0.3970823069  1.3463919908  0.5258465122  2.0836183364 -3.5102833984
[3041] -0.0590156125  0.3059855559  1.1986231832 -1.7431905936  4.3160694568
[3046]  0.3513055744 -1.8010172152 -0.1169905135  3.7018273299  1.9249389958
[3051]  0.8980597910 -1.2161338503  1.1265489967  0.6544207796 -6.9107938904
[3056]  1.1737882480 -1.0162901216 -0.8460741072  0.1776598355  0.0368982682
[3061] -4.2222862724 -1.0886059256  2.3157525867  0.8396210253 -6.6599139704
[3066] -1.8527217977 -0.9449983069 -1.1716145084  2.5396752098  3.0996125551
[3071] -1.0673060329  1.0616336056  1.2207400739  1.2087171280 -0.8516907029
[3076]  1.8440569927  1.7975254760  0.6555891990 -2.3040828103  0.9231731773
[3081] -2.0393634819 -0.6677380314 -1.2552949178  0.9097808373  1.2321425323
[3086] -1.3499702848  1.5887730766 -0.9022561143  1.7695730915 -0.8202473349
[3091]  0.3009839842  0.2165850048  1.8150293010 -2.2564252481 -0.8204726577
[3096] -1.1450824335 -4.1073130953  0.4097450598  0.3360994480 -0.5292058699
```

```
[3101]  3.9905807317 -2.9823673532  0.6034646908 -1.5076212370 -0.2204990168
[3106] -0.0609866502  3.5445671355 -0.8220625708 -3.2271757401  1.4769361456
[3111]  0.5744205122 -2.5730190075 -0.8121849696 -1.4624411535  0.1610867920
[3116] -1.6312961606 -0.9412500136  3.2581538573 -0.0893961698 -1.6318438875
[3121] -1.7598129430 -0.9845034682  4.6700112497 -2.5213036740  1.3306574749
[3126] -2.0379549777 -2.8686856100 -2.8715000758 -0.3891972486 -1.4816237201
[3131] -1.0245314696 -0.0046765299 -2.7948740687 -3.0639583925 -1.7232232366
[3136]  0.0918734047 -1.2288311356 -2.2900564653 -2.5971176825 -2.1128376859
[3141]  0.8666644773  1.8498325519 -1.2001057021  4.3793538062 -1.8042386525
[3146]  1.0775110763 -2.2256616936  4.4759814370 -3.5355882145 -0.2352979603
[3151]  2.9705490243  1.1618796692 -0.7080009288  4.8090164441 -1.3183386005
[3156]  2.6456656408  0.1618719297 -0.4269085821 -0.5786991338 -6.0357061552
[3161] -0.1778015649 -0.4157182462  2.4273702450 -3.7300843342  3.7890816407
[3166]  1.2838342494 -4.1010967957  2.5885864823  3.3228248835  0.8483449532
[3171] -0.9597416358  0.7572380148 -1.9555493065 -1.7578953044  2.9604728529
[3176]  0.2723843746 -1.8284540310 -0.3087504295  1.1454750324  1.0902359257
[3181] -0.3293160099  2.8179142709 -1.0887291430 -4.0637146780 -6.9381056020
[3186] -0.2542845765  0.1831844163 -1.5431647590  1.9018270702  2.7872710668
[3191]  1.2351286919  0.2922544585  2.0102159693 -2.5330272374  3.0546318915
[3196]  1.0970333006  0.5422521446  2.5742934414  0.9249008110 -0.4072105822
[3201]  1.6889359779 -2.4646203616  0.8047161093  2.2634982291 -0.5066530435
[3206]  2.9073448889 -0.7210915078  2.5489039803  0.6927598018  0.1668960230
[3211] -0.0031675605 -1.9399871825 -0.5030778465 -3.3844162269 -1.3977585498
[3216]  1.1111370070  0.2692144707 -2.9415840600 -0.7891637002 -1.4774443123
[3221] -0.2461457688 -0.3181971939 -0.6811257235  1.8147864041 -1.7392905005
[3226]  0.0842331424 -0.0218166535 -1.0612609571  2.5995956623 -1.3256961021
[3231] -0.3859198846  0.9003726780  0.0471911534  1.5612782451  1.4042811501
[3236]  0.8907577989 -0.7300646602 -0.2763727164 -1.6289922511 -1.1914578758
[3241] -2.5258997208  0.2184256348  5.3994196587  1.3840986776  2.5394249479
[3246] -0.4869129081  1.2855100102 -2.4199552321 -1.0184666216 -0.5873671618
[3251]  0.4386953576  0.6588747989 -0.9389509265 -1.2910936735  1.5688428094
[3256] -0.6149263395  2.4494219421  0.7842877020 -0.3080055861  2.3014446076
[3261]  2.5100077800  0.2764155086  3.2371222072  0.5776512342 -0.8171136213
[3266]  1.9043150898  2.4048938905  0.3824925348 -1.4844174245  3.2159149788
[3271] -0.7080847887 -0.8631618983 -0.2348088559 -0.1857302821 -1.2894155117
[3276] -1.3531631744  2.0359537210  1.9450642232 -1.1421871559  0.5400758701
[3281]  0.7472764676 -0.4847176299  0.8947675023  0.0530394370 -0.8880924599
[3286]  2.2823474476  0.2845018296  1.6622062378 -0.4527037321 -1.0012495049
[3291] -1.7520296310 -0.5818476476  0.9556442478 -2.1017240275  0.5541359623
[3296]  1.9039345712 -0.7921609277 -2.9355316202 -0.3031001037 -0.5563377771
[3301] -1.5407589204  1.9221870695  0.9610988519 -0.0111995000  1.0720295525
[3306] -0.0647314083 -0.6267118831 -0.6412831060  1.6396866401 -0.1279105154
[3311] -0.2868481886 -0.5186547972 -1.3695286236  0.1629345983  1.0981375767
```

```
[3316]  0.8539938396 -1.0117619079 -4.4251547468  1.0234452733  1.6546778221
[3321]  1.2789672501 -1.2713921411  1.1497801488 -0.1276621894 -0.2028866060
[3326]  1.2326600559  0.5044223410 -2.7978954704  0.6973478887  0.5975974675
[3331] -0.4098773423 -2.9294989970 -0.2701594712 -1.0956497243  0.8684262774
[3336] -2.1205973634  1.1957912658 -0.2033770156  1.5446403772  0.7954773147
[3341] -0.7436789989  2.2243684800 -5.0762604579  0.1129570487  1.6093137691
[3346]  1.1021259360  1.6509856484  1.5179361313 -2.8483511563  1.6319191483
[3351] -0.2477396150  2.6312602084  0.9125481057  0.5453304490 -2.1322555967
[3356] -0.3137769133  0.3485723039  2.2924236631  0.5743401918  0.8126543645
[3361]  0.9162715183  3.6407224054 -2.1920851797  2.7647904126 -0.1990505883
[3366]  0.6799031100 -1.3148647297  0.9999799246  3.5447691944  3.1019448996
[3371]  0.9074486224  0.9881225553  1.3291343411  0.5717530873 -2.0369419528
[3376]  1.3192471201 -1.5839393861 -1.3718240681 -0.7218107176 -1.7954104116
[3381]  3.1033387593 -0.1377061324  2.1313776530  1.6974207817 -0.1803021622
[3386] -0.9568634469 -0.6445645993  1.5355525565  2.2638508374 -0.0847294317
[3391]  2.6741026748  0.6978929102  2.1535985885 -0.7826710081  0.3449127566
[3396]  0.1675090798 -0.2280192111 -0.0060100263 -3.4751338146 -4.7338457423
[3401]  1.4718368188  0.2503753493 -0.2294210233 -1.2087825903  1.9591619060
[3406]  2.6753902172  1.2240992432 -0.9284392034 -1.0768512497  1.2168198719
[3411]  1.8883744891 -0.7285516743  1.2940810811  3.5108570342  2.0921755023
[3416] -1.4196491218 -1.7981110346  1.0311355837 -2.9616855243  0.5974933068
[3421] -2.5253395865  2.6857129377 -0.4257817297  0.0826610439 -0.1566398838
[3426]  1.6816278854  0.5298375574  5.7563000946  1.1265470350 -1.2263494534
[3431] -3.0453224531 -4.3558169618  1.4846595604 -3.7078609802  2.4179766410
[3436]  1.4960079101  4.2344134605 -0.4804881648  1.7763521592 -0.1023088238
[3441] -4.4411656776  5.0858880701 -4.0789855586  0.0684767536  1.6357385425
[3446]  2.4371051588  3.4739482506  1.0774864923  1.7051911917  0.1600734271
[3451] -1.5273140687 -1.0581766786  0.0417924105  2.2329277257 -1.5581008286
[3456] -2.1385117619  0.8606911077 -3.7301034498 -1.3803920526  0.2350053505
[3461] -0.6145309219 -0.1417863114  1.8722630688 -5.9478085760 -0.0226770455
[3466]  3.0138103021  2.8734294765 -0.4159294895  1.9206239973  1.6456623237
[3471]  1.1891047477 -4.0249000440 -1.0660739706  2.1827633542 -3.4947704109
[3476] -3.7210346900 -0.9815297220 -1.5352997722  2.4919720464  0.7148327323
[3481]  3.1922976550  0.3052819033  0.3976613603  1.2744971436  2.4471619069
[3486]  0.4453625363 -1.5922637555  2.7123521637 -1.4528392399 -1.5618655098
[3491] -1.4843077904  0.3253016013 -0.5693518284 -0.4888056862  0.0124635450
[3496] -1.3036555859  1.0289650672 -0.3297534407 -0.1396391896  2.0330985014
[3501]  1.9910983471 -2.5879830510 -1.7339889346  2.9687811796  2.5506527132
[3506]  1.5579476989  4.3367594547 -0.4300652447  1.2966080683  1.1733962522
[3511]  1.9574822768 -1.2280886547 -1.2422385018  1.5373921120  0.5206801562
[3516]  2.7053439486  1.7926792021 -3.7427481909 -1.8864440604  0.2835079609
[3521] -0.7543143852 -2.6712745789  1.8313703813 -2.9221751202  2.5779655093
[3526] -1.3516259151  0.1449723604  1.8892980001  0.7050575907 -0.8998322767
```

```
[3531] -0.0568298711  0.5699863999 -3.1674852392  2.5697986950 -2.1424633468
[3536] -0.1074941800  1.4646357684  2.4305798963 -0.4500836772 -2.3622070593
[3541] -0.6959307028  0.8554607857  1.1156757705  0.1685318420 -5.5631299908
[3546]  0.1037953635  1.9522719338 -2.7335535828  0.3652100204 -4.7170317642
[3551] -0.6262711035  1.1262566048 -1.2945426206  1.3458396568 -1.6436846622
[3556] -0.3559128429 -0.1440454671 -4.9862130997  4.2259505858  1.3171613662
[3561]  2.6777989908  1.4828200707 -3.1622048768  0.8832851563 -1.1226183886
[3566]  0.2050945538  0.9932214635 -2.3624983710  3.0424940133 -0.4356371701
[3571]  1.2979811792 -1.8236175945 -0.3444325117  2.9489559260  5.2421889036
[3576] -1.4974474829 -3.7991756638 -3.0885801252  1.2460330950 -3.7220528952
[3581] -2.6170974275 -2.3999474281  0.2323678873 -0.5658468159 -0.7163956734
[3586] -0.9622985367 -1.1183505267 -1.1753368441 -0.0287432148 -2.2264069575
[3591] -0.9649030974  0.9571894468  0.7403668588  2.1616597530 -2.5353164740
[3596] -1.6966906577 -2.9987265599  2.0341691379 -0.0092460141 -0.0958955351
[3601] -0.4378232488 -1.6902777137 -1.0898929504  4.5687966331 -1.5398350612
[3606] -2.9474487633  1.0730487234  1.2731807989  0.4045925045 -0.5797836427
[3611]  2.9273623716  0.1299917294  0.8690323215 -2.0154684896 -0.7812714546
[3616]  0.0916023451 -0.7860330779  1.2546378431 -3.6323911184  1.0235940452
[3621]  4.0051082828 -3.9143985084  2.7726525770  0.2486116471 -4.0200276444
[3626] -0.4192110436  5.1880057582 -3.8422260409 -1.2712722253  1.9148555178
[3631]  0.1215532273 -1.2412021742 -4.5115495298  0.7132577571 -1.0895310424
[3636] -1.6048879778  0.9147869153 -4.5549792653 -1.5073319244 -4.2943421188
[3641]  1.8136274422  0.0170110165  0.6927342391 -3.6183321879  3.0277209615
[3646]  1.1974241538 -1.5210370225  0.1022834722  3.0298337120 -0.6183829500
[3651] -1.2899865112  3.0090782248 -6.0602735674  1.2096706761 -1.5911189301
[3656]  1.1492044225 -2.3074480333  0.6021907629  3.0873126327  1.4638078874
[3661]  0.5462394053  4.9342419984 -1.7039038266 -1.6626200405 -0.5453370907
[3666] -1.0936383848 -0.6367589097 -1.1282658960  0.1036719246  0.1070917830
[3671] -2.2770784066 -0.2284428518  0.0169142760 -1.6977665659  0.1990179120
[3676] -3.5181378330  0.9781269035  0.3060044912 -1.6701626491 -2.3608799847
[3681] -1.6383697205  3.4183767397 -2.5159895215 -1.0689598004  4.0375837702
[3686]  0.0507617082 -2.9057040071  1.5033079117  0.5879843179  0.6453264822
[3691]  3.7001819495  3.0997168946 -0.0596366673 -1.0936923764  0.9082685462
[3696] -4.0182280102 -0.1842053484 -0.1038016743  4.7846565696  0.6321468437
[3701] -0.2745527717  1.8712966827 -3.2787575480 -3.9804682117  1.4515960728
[3706] -2.3691025868  1.4435884406 -0.1251318102  2.4550721135  1.9973917682
[3711] -0.1188957885 -1.7847101540  1.9441401332  1.3999970302  1.2244104267
[3716]  1.8274845917  1.4006217273  2.0123874120  1.4823973439 -1.4797908428
[3721] -0.9361442687 -0.9805224772 -3.0108978276 -6.0889082513  1.1637615758
[3726] -1.5734603136  1.6252573403 -0.3145376004  1.5232516065 -2.5174130599
[3731] -1.1249545867 -2.5709029660  2.7549517957  3.1063846497  1.9676948703
[3736]  0.3969227350 -3.3647209301 -4.1905368871 -0.1895757638  1.5018241512
[3741]  1.1093690656 -2.3547599212  0.5631887434  0.5942790262  0.0953156799
```

```
[3746]  0.6802876718 -2.3435615089  2.7547652882  0.7353523274 -1.2798590345
[3751] -1.6363225202 -0.7369474888 -1.7732235489  3.0134804806 -0.6536686310
[3756]  2.0608191548 -2.0008754393  1.5161032924  0.4857531183  1.5145781692
[3761] -2.3272216176 -0.6721608352 -0.2299567064 -0.2871439741  2.6131015436
[3766]  2.4001461858 -0.0547534118 -0.6032637496 -0.9008403943 -2.4579725508
[3771]  1.3312098033  0.9478723122 -0.9591144159 -0.3708216383  1.8828847144
[3776]  2.7132903363  1.5136009642 -3.4458385010  0.8896156863 -0.9855158051
[3781]  0.0215744831 -2.1089737359 -0.1649585007 -0.5396382587 -1.4748605802
[3786] -0.8503966865  1.3325265158 -3.3438481319  1.0251057241  2.6308867776
[3791] -0.8706486225 -0.8229304861  1.3731236924 -5.7718979070  2.9833386768
[3796]  0.6650132959  2.3547834851  1.3968310553  0.6922305371 -2.4906297306
[3801] -0.8595835997  0.1803796546  3.0085519472  1.3119096076  1.3496027752
[3806] -1.0989225366 -2.0619815228  2.7915164858 -1.3286497287 -0.8092106525
[3811]  2.2438789630  1.0382285219 -2.0985208968  4.3430705049 -2.8463753259
[3816] -0.7774268057 -1.7179641236 -2.5291783993  0.3966673481  2.5126907339
[3821]  2.3678691436 -0.5561310710  0.9835530747  2.1086164744  1.1911298934
[3826]  1.8534565700  4.6333570731 -1.5607625649 -2.1888038304 -2.2023703098
[3831] -3.4384442004  2.3578452021 -1.2198607554  0.7514627701 -1.0642876472
[3836] -0.1716384033 -0.0745071365  2.5851690282 -1.4959549468 -0.7936704606
[3841] -3.1466101076 -0.0714268462 -2.4231283800 -3.4297764571  1.7348742845
[3846]  1.2528280515 -2.3041028584  3.3316045016  2.5842196823 -0.6374583985
[3851]  0.5328916657  1.8806744234 -0.0667413696  4.3059203457  1.4129475702
[3856] -0.2752696561 -3.1476044317 -1.1236615065 -1.8407881430 -3.0909563459
[3861] -1.0429820144 -1.9347854752  0.0671663476 -1.3540437896  1.4188547200
[3866] -1.4323789164  1.1492370365  0.9470337833 -0.9333344331  1.0917930461
[3871] -0.7094465837 -0.0126242470  1.3557900534  1.6120557924  2.6129915016
[3876]  0.8968329837  3.3456474866 -1.2811685288  2.8061427525 -1.0371601224
[3881] -1.7099980588 -2.2167711792 -0.1834019058 -2.7900149324 -0.5794502647
[3886] -2.0744461795  0.6245392991 -2.2930751530 -0.4313205108  0.4571903190
[3891] -1.2345181078  1.0818948615  0.9847807108  1.5656545772 -1.1614103908
[3896]  1.4485536505  0.2699275158 -0.9409374754 -3.3132839662  2.2084113919
[3901] -0.7292758398  1.7430618375 -0.0631214387 -2.5726519147  3.7091201263
[3906] -0.9237351577 -1.8262150638 -1.2671448623 -2.7693312842 -0.5333675498
[3911]  1.1224584783  1.9769209572 -1.0853629063  2.5858741414  0.3924209046
[3916] -1.2474296398 -1.9669720433  1.5624803213  0.4826337800  0.9867464311
[3921] -1.9440387960 -0.2829245535  0.9279161135 -1.8083655549  2.0482739198
[3926] -3.5221690345  2.3358289633 -2.5481603837 -0.0650877079 -1.3472097162
[3931] -3.2455959865  2.3976963234  0.2136464931 -1.6937927800 -0.5124682942
[3936] -0.6785631440  0.2072745273 -2.1364486700 -2.9289924488 -0.2996223874
[3941]  4.9187861623 -3.4083852305 -1.8449946453  7.2413151020 -0.2025868570
[3946]  1.8291188755 -0.0206024542 -0.3735093252  0.5205600972  3.0115750863
[3951]  0.1352870479 -0.5998688275 -1.6858298409  1.4703680691 -2.0651416657
[3956] -2.7600907491 -0.1223020821  1.0240168657  0.8707640484  1.1999894248
```

```
[3961]  2.2483207629  0.2491829614 -0.8967830489 -3.6862066958 -0.3305048990
[3966] -2.1333314704  0.4858600034  1.9386706845 -1.1852463556 -1.0153214497
[3971] -1.4828698321 -0.7721164327 -1.4573361595 -2.0303186253  0.1442974048
[3976] -0.4968134618  1.0273134545  1.1765686086 -1.4452072550  0.7204559667
[3981] -1.1756690792 -3.2963033489  0.2013594829 -0.5129037940  0.5753729977
[3986] -0.4435667106  1.1319811473  0.5777778914  2.3736934984 -1.3386355822
[3991] -2.5232210815  0.6324830486 -0.9092699905 -3.0822138450 -1.6974922312
[3996]  0.9016326385 -0.7347043303  1.0148291818 -1.4344283249  2.2567957088
[4001] -1.2790360845 -1.8980979111  3.5293819719 -0.7429105899  1.1730803399
[4006] -3.5146290031  1.7685513322  2.6214357725 -0.2088310647  2.4678232308
[4011]  0.1607806067  0.8388233845 -0.0813483916 -1.6073679501 -0.9320008351
[4016]  0.6954278787 -1.3687060345 -4.4599264269 -1.3862619099  0.9330579488
[4021]  2.1503851124 -0.4888234115 -2.6421778172 -2.3588080757 -0.4324929892
[4026] -2.7924678777 -0.3460879344 -2.7500909842  0.6325813383  2.0385393890
[4031] -1.2397363971  0.5778740805  0.3192399067 -0.8905160961  2.4331785818
[4036] -0.5168484871 -3.4518887120  2.1155256680  0.2498841231 -2.4742045360
[4041]  0.8655020855  2.0515891242 -1.3671305126  2.5219195451  1.2896107368
[4046] -0.0818605140 -1.0242242333  1.7584543466  0.0112138949 -0.6462403755
[4051]  0.5014759901  1.1240748317 -1.5177211010  1.3061759027 -2.1042711224
[4056]  1.1136786054 -1.9142105838  1.1846489766  1.2853286702  0.5042681226
[4061]  1.4881759416 -1.7054739149 -1.3393101788  1.8716275204  2.3876242566
[4066] -0.2183105215  0.4631566852  0.0746995244 -1.4452812187 -0.0786925995
[4071]  0.1585346672  0.8341880799  1.4109040231 -0.1704649700  1.8241737879
[4076]  4.1909356948  0.0522362758 -0.2863970766  3.5373575373  0.5103577520
[4081] -0.4195716314 -2.2484818297 -0.1387195746  0.0518086107  5.8618314451
[4086]  1.4241154041 -1.3216806551  1.0674584144 -1.5677879152  3.3024115359
[4091]  1.7300302298 -4.7808376770  3.0825263709 -0.1455373075 -1.8759670039
[4096]  1.7668330109  3.1485796667  1.0948768774  1.2989443649 -1.8040978836
[4101] -2.1923871060 -4.2410271160  0.7553953099  1.2797270173  1.8146762302
[4106] -2.7051470444 -1.3973680016  1.2039951717 -1.0879493916  1.9432781760
[4111]  0.7212831693  0.9954832850  1.1890035749 -1.1557690961 -1.5524609242
[4116] -3.2312373052  2.6165247592 -0.6991858541 -2.6638445033 -2.4358422284
[4121]  1.1974527442  3.3565128195  1.0940459803 -3.4899204344 -0.4680882952
[4126] -0.6697295683 -4.8330160492 -0.8859863915 -2.4224830165  3.1505133017
[4131]  1.1208583674 -3.2199126693 -2.1017115618 -2.3262526606 -1.4317716617
[4136] -2.7822155963  0.6468008151 -1.1347736674  4.7128765191 -2.6678024175
[4141]  2.7864397601 -3.1311082191 -0.1800125954  1.2762150218 -1.0370822117
[4146] -1.9705136696  3.5882748203  3.0613680241  0.8203589488  1.2095697962
[4151] -1.9565226439 -1.0041039590  1.4384741146 -3.6399473905 -1.9273340699
[4156]  0.0529975654  1.2424414441 -0.7334483895 -1.4825154126 -1.9906774332
[4161]  1.2397901356  2.6075413962  0.0439217305 -0.7107289825  0.0185687517
[4166]  4.1284071879  0.5142003105  0.1877592565 -4.2913998528 -0.3375910008
[4171] -0.3374553822  0.3998875243  3.0563010296 -1.7800240398  2.1528596713
```

```
[4176] -1.5874224796 -5.3433771143  1.6868590097  2.5148240022 -0.4621133894
[4181]  0.5998474861  1.4832782679  0.8864614622 -4.6243504221 -0.2186093085
[4186] -0.2893049589 -3.1007158475 -1.0473378088 -0.3843844610  0.1038293682
[4191] -1.9687886774 -1.3611171487 -3.2028159483 -2.7152087969  0.0499433499
[4196]  1.9812050777 -1.1872101894 -1.9695480742  1.8684751916  4.8459540516
[4201]  0.0477268158 -0.5601190165 -3.7392102014 -3.0020428481  0.8633914308
[4206] -1.4926619004  0.0380193237  2.1581407766 -0.9160228932 -0.8591448264
[4211] -1.8664156521  0.3236569575  2.0648412701 -2.6616037316  1.0749498585
[4216]  0.4236997936  1.2779252016 -1.3940915560  0.8601771662  0.9729965837
[4221] -2.8925324599  0.6849653258 -3.0772390865 -0.1177013765  3.3921293102
[4226]  1.1335013365 -1.3099475699  2.5666444955 -1.1813570985  1.2969586346
[4231]  0.2894991318 -2.9837851797 -2.8764623926  1.8015274239  3.1950948683
[4236] -0.3523557083  0.5600188841  0.4911702275  2.3001617195 -2.1812458973
[4241]  0.7653705473  2.0284049979 -1.2954467355  2.0034448238 -1.2602245370
[4246] -1.7044377174 -1.5723269383 -1.8453269476  1.5904910300 -0.6610790559
[4251] -2.4239756749 -1.2750997435  2.3272167280 -2.8158414004  2.9450127645
[4256] -2.3676732287 -0.6095567061 -0.0539019369 -1.0300245214 -1.6020820504
[4261] -0.2811081761  4.0263510743 -0.0491582774  1.5353660393 -0.0417797144
[4266] -0.9291575916  1.2823421529  0.4533480241 -1.8851656460  3.1355094660
[4271]  2.5445940041 -4.1594892968  2.1691108672 -0.4638344852  0.5967723985
[4276]  4.1924399124 -0.0848121440  0.8506849366 -0.1444749951  0.8543528112
[4281]  1.8796861672  0.8168607930  0.7525130966  3.8118919120 -1.2187202663
[4286]  4.5821166404 -1.6076587379  0.9125472416  2.0361900413 -2.2315701061
[4291]  1.3742263004 -3.0760784868 -0.0015822265  1.9838712756 -1.9092330516
[4296] -4.3935646253  0.1719778783 -1.2272862131 -1.3671968987 -2.9309381397
[4301]  3.3940096768 -1.8714608690  2.2229103794  1.5195682761 -2.0701204882
[4306] -1.1037983946 -2.4539917770 -0.1042691813 -2.5352038888  1.5802113891
[4311]  1.3026376735 -0.8547607321 -2.1341520546 -1.0677855342  2.5608889903
[4316] -0.6939531795  0.6548022313  0.3463379778  1.3928559024  1.5047191693
[4321] -0.4710159157  1.0554769468 -4.6617677940 -3.1519521738  0.3002409198
[4326]  0.8362028540  0.1208649301 -2.5396855112  0.2530118625  0.4623590324
[4331] -1.5143541462  1.5129240087  3.0270045573  3.9559345413 -1.2349497814
[4336]  1.9294814559 -2.2551395402 -4.0130772982  1.6287002570  2.1373506958
[4341] -1.6089017393  0.0777409213  0.4288506695  0.3357918010 -2.5986016338
[4346] -2.3835583566 -1.0699243204 -2.1159407959  2.0917316282  5.5338424606
[4351] -3.9998338060 -1.3618004782  2.3234114837  0.6997856379 -0.6097161158
[4356]  1.5989344029  0.3936638836 -2.0048890956 -1.1779334124  3.3355787638
[4361] -0.0324445043 -0.3874105268  0.2371767314 -4.5883754881 -0.1143350149
[4366]  2.1220348238  1.9751418583  3.2670198998 -1.3266593526  1.9197737599
[4371] -0.8143801217  0.1724256696  0.8371086237 -1.6764058132  1.9817010702
[4376] -0.4825708002  5.6871832419  1.1274215813  1.3462063125 -0.1501587640
[4381]  0.8884271404 -1.9237648139 -1.7248040425 -4.0407344995 -1.8905465072
[4386] -1.4140103001  0.0107493925 -2.3714614604 -0.5636084362  1.2758815505
```

```
[4391] -0.2114486765  0.7221134935  0.3433632310  0.0190164563 -0.9676682191
[4396] -0.0725162404 -1.0434646446 -1.5987701297 -1.7797847322 -2.3244852296
[4401]  0.4333036328  2.4958072644  1.7959768326 -0.8080201149 -2.2207341501
[4406] -1.0379120079  0.8032673880  2.0944551876  0.3461600203 -1.1085323661
[4411] -0.8253923109  4.5622034566 -2.3434225130 -1.7736522091 -0.9383474477
[4416] -0.5822696721  0.3138160211 -0.8509481708 -0.8975533504 -4.0934846025
[4421]  2.0916958211 -0.8671858666 -0.2006773902  2.7552664673 -5.2323877314
[4426]  0.4417098211 -1.0287915279 -2.4134874729 -3.0698999210  0.0709438344
[4431] -4.3812169024  1.2168134791  0.8964248417 -2.0033296497  2.4835773583
[4436]  0.9073154840  3.4259108662  2.2318962077 -1.1621117427  0.0610718702
[4441]  0.8907697002 -1.4447293114  1.0726959428 -3.6300805313  1.8406161860
[4446] -0.0801371156 -0.0943175573  0.4704806312 -3.0791448483 -3.3705885160
[4451]  1.4677777259  4.3726606495  0.0236446576 -3.4842686753  1.1866933414
[4456] -1.5208806893  1.1030699789  0.5088317319 -3.6253819019  0.6061055037
[4461] -0.8931276772 -0.1726264513 -1.2479248359  0.0253571705 -1.4925094304
[4466] -3.9377823869  1.0569022704 -0.2990206499 -0.6308917113  0.2753606018
[4471] -0.5344406436  5.9789571817  0.8347837021  0.7414236937  1.9099563258
[4476] -3.2789903885 -5.8068444080 -3.5749356175  2.4015841490 -0.0307296321
[4481]  1.9159335333 -0.1597675690  0.7851565726  2.2834470252 -2.8996030114
[4486]  1.0160660864 -2.2689441801  0.5561822645  0.6324223146 -0.6649076878
[4491]  3.2875372896  3.2413531684 -2.1030751328  2.1864012829  0.3978435046
[4496] -2.0999766544  3.2954134140  1.2427817673 -0.6827098880 -0.6258931083
[4501]  4.3939975574  0.5350123650 -1.6515235773  2.4199775503 -0.0764398449
[4506]  2.5306648716  2.4772259894  1.0213499723  1.2437195471 -2.4045681350
[4511] -0.0951614930  1.1082428090 -0.8547993916  3.3391320908  2.2246718264
[4516]  1.9469985780 -1.4392320572  0.0746481638 -1.2083820910  2.0784710361
[4521] -2.8554516331  2.5907873414  0.3265854414  3.9599464888  0.6825115100
[4526]  0.9922447421 -1.3933955643 -2.2134112487  0.5161853713  3.0693248172
[4531] -1.7064048898  1.2620245611 -1.2862429148  5.0715103494 -0.1559726473
[4536] -0.0420409683 -1.2077693516  0.9467602576  2.6213287091 -0.6378480075
[4541] -1.5440092495 -1.3312257658 -0.8387991360 -2.5458899548  3.2943294389
[4546]  4.2165384740  0.8504312497  1.8099839594  0.8969035266 -1.7176242772
[4551]  2.3941607808 -1.0568434023  1.6152306767 -1.6231653562 -2.1108801373
[4556]  1.0330650117  1.9148883502  1.3514642441 -1.2286760535  1.9887423688
[4561] -1.8662782248 -0.7959305288  1.1293994733 -2.8676384961  0.4379865893
[4566] -1.7772025774  3.0558353019 -1.5773592871  0.1740948491  0.7414922496
[4571] -1.2997538781  2.2652361232 -4.0915625421  2.0479670136 -0.5612287938
[4576]  1.2750899439 -5.0431482393 -2.8195390120 -1.8198223029  2.4104612207
[4581]  1.1880986998  1.0311871572  0.1848070214  0.0682582762 -0.8678536783
[4586] -0.0344852582  0.8254896252 -0.5125158279 -2.6665233392  0.0158096549
[4591] -4.4364801854  0.1076584088  0.0855190398  3.3252686444 -1.4338950638
[4596]  0.8478285153 -0.7553910645  1.4407527395 -1.0447204628 -2.0681063195
[4601]  0.6630955716  3.0694910285 -1.6232974846 -0.1937852556  1.2488892848
```

```
[4606]  0.6672493381 -3.6538101167  0.0677146742  1.2838593923 -0.1407851446
[4611] -0.6840737453  0.7435521394 -2.6781423448 -1.7942998491  1.3637817713
[4616]  3.1376392734  1.0338625688  1.1277230824  1.2836820519 -1.6404687571
[4621]  0.1837994785 -0.8949209162  2.7151568580 -3.9072571992 -1.3372332907
[4626]  3.9376729279 -1.0270150942 -1.8747224646  0.4955788152 -1.5768758513
[4631]  1.7584785481  3.6651967349  3.0438600159 -1.3856353531 -2.0379387104
[4636] -0.1089704903  4.7758873738  0.0888110389 -2.0449836709 -2.7929739927
[4641]  0.7052939732  3.2729057891  2.4021979575  2.3125982417 -0.4600506382
[4646]  0.2791094723 -0.5103051303  0.4764813482 -2.2447985629 -3.2436192073
[4651]  1.8187460107 -1.8875594858  0.9176000799  1.4693626118  0.2996654019
[4656]  1.5757290124 -1.8919442271 -0.6070674109  1.9574114054 -1.8291230474
[4661]  2.8848376695 -1.2499197046 -0.1543188505 -0.9703176567 -0.2933158795
[4666] -2.3890204590  0.5704202120  2.4441837773 -1.0172170136  0.3873921666
[4671] -1.9268082189  1.6839728349  2.9071856650 -2.6301505648 -0.9826237669
[4676] -1.6405071114 -0.7789853874  2.3171201610  2.5124294487  1.8563578762
[4681]  4.2796660626  1.4078529446  0.5828711715 -1.4074902072 -0.0904627058
[4686]  1.2691133655  2.5859924937 -0.5828324489  0.7158714983  0.8974792247
[4691] -0.7029617543  2.6076285323 -2.5831229606  0.9136236599  1.5898757014
[4696] -2.3554153107 -1.7815685399  0.0280295823 -2.3366018598  1.1575457250
[4701]  2.1641765581  2.1602890441  1.3361086459 -1.5177437530  1.7328967181
[4706]  0.3998373914  4.0767335993  1.7260932184  4.2205415747  1.4541989462
[4711] -0.9922610806  4.4354050442 -1.9024190315  0.1568019049 -1.3885786015
[4716] -0.8646445083  0.0933599831  1.2022293572  1.9534584623  2.2405488222
[4721] -0.8199434303 -3.7840607356  3.9222165494  0.9044582640  0.2352729495
[4726] -0.6140777324 -0.9851254269  0.1913214651 -1.4781213313  0.6734995210
[4731]  1.1786549394 -1.5437682421  0.9271456941  1.5065106919  1.6218305730
[4736]  0.9387716594  1.3800890162  2.7448438737 -0.6446330645  3.2450008329
[4741]  2.5878735866 -0.4028490753  1.2916163833 -0.0466600579 -0.6830592813
[4746]  0.9890171729  3.2174006744 -1.1757999677  2.3457944940  2.6596160616
[4751]  1.7917037549  0.2871157131  1.1616020710 -0.0015726420 -1.1372007314
[4756] -0.2455836584  2.4150247339 -1.1241571262 -0.5408405010 -0.6923510057
[4761] -3.2601834751  1.1021930408  2.5205324156 -3.5118254335 -1.5173629173
[4766] -0.3959892053  0.0611492333 -0.8122615706  0.7766054042  5.2025212588
[4771] -0.8779004350  3.9408085691  1.8163733933 -0.1709960955 -0.0184357063
[4776] -0.8968384512  1.0377097557  0.5254357372  1.1245513458  1.3464871735
[4781] -0.3203808559 -2.0285919448 -1.0083944443  0.8514392347 -0.9514748102
[4786]  0.5177188195  1.0698319659  0.8607744166  1.0528016348 -0.0292971201
[4791]  0.2840635251  0.2510081199  1.3149424578  0.5855213899 -1.3154199530
[4796]  0.3643504270 -0.8241005494 -1.7756696956  0.6848986308  0.4162831355
[4801] -3.1018621522  0.8727863677  2.7525456384  1.4603161152  0.8251521921
[4806]  0.9282634630  0.7895144639 -1.0587519337  0.1240445670 -0.3910666604
[4811]  3.4520330397  1.1874934622 -0.3650213161  1.9311237761 -2.0598325675
[4816]  2.3415903614 -0.2273389165 -3.6506286819 -0.9507251368 -2.4782810427
```

```
[4821]  0.4158120008 -2.3999367991 -2.7698786857 -1.0233039917 -3.1989155181
[4826] -3.4346855585  5.3072259540 -2.3983040902 -0.8767182696  2.8606938747
[4831]  2.9989381810  0.9813019311  2.1084617315  1.2130941877 -0.6697272374
[4836] -0.5313606601  0.7709180803 -1.2356415261  1.9867210431  4.2984835671
[4841] -1.3973515861  0.8469169760  1.9031488222  0.0153301701 -1.3850592860
[4846] -1.2967720377  2.2910128231 -1.2324455888  3.0472882695 -1.1675739102
[4851] -0.6850685881 -0.8179562588 -3.6677986647  2.0111909758 -4.6963016553
[4856] -0.8457974315  3.3380938127 -0.9873688947 -2.9537999611  3.8851572119
[4861]  1.4351867785 -3.6581706816  3.7887616493 -0.2431147241 -1.5962152726
[4866]  1.0577763891 -1.6533124434  1.8977130716 -0.6191109589  0.4919179654
[4871]  0.2710233063 -3.3228529148  1.8266857587  2.0990658330 -0.3676777609
[4876] -0.3527192703 -2.4439788409  1.1481757446  2.2425157067  2.5523735167
[4881]  0.7138473931 -0.6511798001  0.0436862116 -3.8205291419  0.4196160995
[4886] -2.0997887391  0.6728431854  0.1374480050  1.0242154658  2.4046394455
[4891]  1.6799296612 -0.0028484292 -2.1744351297 -1.8588116993 -1.8484936156
[4896]  0.3471427987  0.2753719880 -2.1614225838 -1.1285989602 -0.3346831703
[4901]  1.4663540430 -0.9662055592  1.2910142508  1.0501836339  2.3824141230
[4906]  2.8609091069 -0.9208611067 -2.9582330182  0.1803268413  2.3912121021
[4911]  1.5558382879  2.6436549590 -0.5002702480 -2.8061173476 -3.7339295206
[4916]  0.3243575744 -1.0226749430 -0.4040345109  0.1016101757  4.1084373355
[4921] -0.8598272652  2.8527756497 -0.0893641362  1.4736677523  0.5693749003
[4926]  0.4497965342 -1.5833325937  0.5371326374  1.1201236413 -1.3775384309
[4931]  3.2248067709 -2.2183489355  2.3684395534 -2.7700467832  3.3334856184
[4936]  2.6099307604  0.1313585072 -0.9396821064 -3.2861721160 -3.9309631808
[4941] -1.4016531510  2.0744621724  0.2862793096 -0.1643722217 -0.9915189049
[4946]  6.1789851695 -1.2445888540 -2.2389401983 -0.5267145975 -2.0122738024
[4951]  4.1923418591  0.7158670954 -0.7343634973  3.5160141572  1.4965630990
[4956] -0.6447669077  2.0494326094 -1.3580724143  3.3156673754  5.4442127165
[4961]  0.3688477949  4.5235632232  1.6049198569 -0.4182363068  0.7443250557
[4966] -4.1071223360  0.9201361953 -3.1490936542  0.5525248684 -0.6439065748
[4971] -0.7859233494  1.6210379304  1.6204455797 -3.2282520300 -1.2136691342
[4976]  0.0582032299  2.7560467648  1.2657823255  2.4616889784 -0.9696867499
[4981]  3.6235952093 -0.0483860155  1.7161698846  1.8742059048  1.8177810014
[4986]  0.2139969822  4.2077161573  0.0171289559 -0.2544580422 -0.0498915588
[4991] -2.6260759128 -0.8595373215  0.1043967935  1.9861514995  2.2382676417
[4996]  0.4256177794  3.3752862902  0.7040136418  0.0808748462 -0.8181051263
[5001] -0.9821126659  1.0812881390 -1.4760359045 -0.7545566734  1.7732075019
[5006]  2.8117215007  1.1400716684  0.3730118073  3.4714505405 -0.5389705493
[5011]  2.2572790096 -1.9071662663 -1.4560928255 -1.7427485712  1.0029699896
[5016]  2.0271123913  1.5388515309  0.5520628300  0.8598366785  0.8915787591
[5021] -2.2496598708 -1.2993285315  1.7154088218  3.0910511506 -1.6407974047
[5026] -3.4897235170  0.7174830521 -3.7446804302 -2.4143298951  2.8524286905
[5031] -0.3292834204  1.2044484930  5.1695736322 -3.0180725583  4.0933690700
```

```
[5036] -1.4932947185 -1.3304429977  1.1649664701  2.5511935526  0.0177711320
[5041]  1.6482254184 -0.7896765219  2.0788594436  1.6130834330 -1.3301738345
[5046] -2.2674783791 -0.3646196618 -0.2208997739 -3.0504516978 -1.4655099095
[5051] -1.4247776646 -3.2443789673 -0.0742722421 -1.0761608699  0.5749129483
[5056] -1.5370489575  0.1121521107 -1.3872758712 -3.6399269263 -0.9109812696
[5061]  0.5265260256 -2.0067427463  0.9056428127 -3.0065433405  0.0670917785
[5066] -0.0865255733 -1.1723317786 -1.2949978585  1.2857889043 -3.5083858214
[5071]  0.4398925941  0.7876239723 -0.8404673428 -1.1773229136 -0.4595996077
[5076]  1.3340790974 -0.8860371194  0.6354823714 -0.4985000708  0.8218175517
[5081]  0.4513564133 -0.9302593744 -1.6548088392 -1.6213330173  3.1370046939
[5086] -0.9713573957  0.5425744945 -2.5039007568  0.4871717898 -0.7390797217
[5091]  1.2749826370 -0.3482542217 -0.6522462859  2.8574778235 -5.0053257649
[5096] -3.6815166166  0.9155349243 -1.1490220381 -0.1330664750 -2.2482495479
[5101]  4.2238350861  1.5605758632  0.4282369284  1.6442327148 -0.2886956430
[5106] -0.1445104444 -0.4499844984  3.6794368132 -0.3666626651  1.9378034406
[5111] -3.0954231653  2.5982561195  0.0186085163 -0.6263898288  1.9657528617
[5116] -1.0286259902  0.7317993639  0.0187264397 -1.1121455434  0.6210199558
[5121] -3.5872174346  1.0228916282 -0.7962482987  1.0822265290  3.1915198137
[5126] -2.7301491524 -1.0324090336  1.8127579244  0.1044316542  3.6290379221
[5131] -1.1761299076 -0.3403340023 -0.9243218945  1.8269831654  2.4388273146
[5136]  1.4592287071 -0.0588037544  1.9695493229 -3.9830995100  0.4858644067
[5141]  0.5542179749  1.6499466662 -1.0630262653  3.0487736713  0.2920892781
[5146] -0.1478096003  0.7231956949 -1.2056498229  1.3317627904  1.2863453710
[5151] -0.6004878877 -0.2462649448 -0.2384908712  3.4054566406 -0.1866123481
[5156] -0.2782524764 -2.5176393604  1.4242669241 -2.5477690019  1.0879284402
[5161] -0.1645273979 -1.1644312204 -3.6499465591  1.6662434376 -0.2138635480
[5166]  2.1748615406 -0.4605878387  0.1211431954 -2.7439116294 -5.8545959051
[5171] -2.3321625408  1.3662871576 -4.3896931233  0.7390492361  1.3231769635
[5176] -2.1974936622 -2.9998057812  0.4477793001 -1.1854460810  0.4547835179
[5181] -1.2127275503  1.0743372716  2.2970486898 -0.8145002684 -0.1698586180
[5186] -1.3978264824 -2.8033042457  4.9470523949  0.1469167980  0.4950823632
[5191]  2.4612340824 -1.5751329467 -0.1834359224  1.5605487636  1.5938431676
[5196]  3.0795484732  0.2585336620 -1.7297553322  0.7202766434 -1.2890798297
[5201]  0.9933547672 -0.7346687372  0.7941708228 -0.4971022393  0.5437625322
[5206]  1.2786446461 -1.4777638066 -2.1765045328 -0.4841601886  0.8050978399
[5211]  0.0912247786 -1.2407775352  2.5224886936  1.2662755291 -0.1509943795
[5216]  2.4155935239 -1.3690975129 -1.6554611098 -3.8469821194 -0.5992151679
[5221]  1.2288585678 -1.7966374017 -2.8966436923  1.8067426561 -0.9056671150
[5226] -2.4587254859  1.0167388295  1.4157161887 -0.5794689865  0.2124685520
[5231] -1.9373967647  2.8053716089 -0.5527393446  0.9667640287  0.3942905366
[5236]  0.9552174646  2.6225935466 -2.7763166630  0.8230002980 -0.5946667143
[5241]  2.0278151094  0.6226513815 -1.3374000785  2.3040438055  1.4125032250
[5246]  0.8505879134  4.9789093326  1.7138604907 -1.9556060225  1.0645568507
```

```
[5251] -0.0644478570  1.6443708908  0.0530962666  0.0357956403  0.4543590791
[5256]  0.8851848152 -1.6172460434  1.5113600028  2.0019513557  0.0002310473
[5261]  0.3877348846 -0.7729643996  3.5783027261 -0.8797080267  2.7129571376
[5266] -0.6393209217 -0.6144016517 -0.1880949772 -0.3501438611  2.2022026614
[5271] -2.9393047101 -2.2782040778  1.1105918571  1.6527444923 -1.4206898303
[5276] -1.5492531516  2.3251112775  3.3440274406  0.9348350785 -4.2377134723
[5281]  4.1696292129  1.5621966303 -1.0661205680 -2.4989683774 -0.1028698310
[5286]  1.0616527230  1.0993770322  0.1964788996 -0.0016609108  1.3331516923
[5291]  1.3114429838 -0.7502117122 -1.0850699913 -1.7215427331  1.2915114628
[5296]  0.4277661501 -2.4379200831 -2.4252786451  1.0873160290  1.9922495114
[5301]  2.6164222840 -2.7153292903 -2.0545014230  0.2368360908  0.2735231541
[5306] -3.0767084911  0.0072099202 -0.9029099970 -0.3104389079 -1.2539806203
[5311]  2.2948454514 -2.1392665457 -1.3131166588 -0.5880757528 -0.9419999207
[5316] -1.7466187913 -0.7532575251 -1.5085330277  2.2601529853  0.9171935316
[5321] -1.5260185523 -1.7803988439  1.1831961615 -2.4061032022 -3.9440097901
[5326] -1.5640720627  0.1537135474 -0.8930462439  3.6320154837  0.1136471426
[5331]  1.1486543281 -1.0156665439 -0.4413069493  1.6115795853  1.1160485062
[5336]  0.8469424815 -1.4844089609  2.3233393442  0.2620461751  2.8975156240
[5341] -5.9789071873 -0.2461350724 -0.4217640379 -0.3853664034 -0.2267052401
[5346] -1.9610382223  2.3470933397 -2.0860141295 -0.4041200987 -1.4613385584
[5351] -0.9276697121 -0.7385925497  2.1950738450 -1.0981896374 -1.0927875721
[5356]  1.7926094334 -0.1090264512 -0.6987885190  1.1397225451 -1.0220016964
[5361] -1.9505832246  0.8940377320  3.7281888196  1.8511972977 -2.6502337218
[5366]  1.2874115378 -1.8704707359 -2.3152906276  2.9633107338  0.8124240776
[5371]  1.7374737513 -0.7328948618 -0.6093113488 -2.2073025809  0.1018760842
[5376]  2.6876890514 -0.3742854733  0.3868664047 -1.3800277490 -0.3928883777
[5381] -0.8946816905 -2.1330479730 -1.9354338858 -2.9561436797 -0.9828283268
[5386] -2.8113654856  2.0045210214 -0.9087958997  0.9713999063  2.5686959583
[5391] -4.0211152078  1.4692934686 -2.0494912240 -0.0116987230  0.8110335668
[5396]  1.3458420164  1.6495997061 -2.4159099222 -0.4905474730  1.8828996973
[5401] -2.6466650175  0.5185460562  1.5447967276 -4.4103264285  1.3468605398
[5406] -2.1336117397 -1.1226310205  0.5572147731  2.0671281289  3.2962604874
[5411] -1.1341145455 -2.4992527445 -0.6643703993  3.0822219681  0.7248684571
[5416]  4.5565979505  4.3634662144 -1.2811568749 -3.3967763195  2.4754431715
[5421]  3.7347251269  4.5233388265 -0.6741017272  2.6658254100  2.9792320114
[5426]  1.6762545211 -4.2881747213  1.1538083802  3.4391381419 -2.4379860140
[5431] -1.1005428444 -2.0871439351 -3.1815500400  0.8355521812 -0.7513523637
[5436] -1.0610951006 -1.6786426204  3.0815072703 -1.8556942558  1.4022367376
[5441] -0.4813118621  0.6306474299  0.9270796775  2.3176005628  6.3171525003
[5446]  2.5719652178 -0.5732423396 -3.0011476379  1.2136282071  2.2774800752
[5451] -5.9860635409 -1.4335647749 -3.9893643660 -2.3149047687  2.1608619078
[5456]  0.8016988399 -2.1863615063  1.0191466446 -1.9696158098 -1.7842222369
[5461]  2.0141206974 -3.8102955555 -0.4778231579 -0.4835611082  0.1075681901
```

```
[5466] -0.3272132578 -2.8934761325 -1.0525650612  0.7246730610  3.5942066729
[5471] -0.7643635355 -0.2855884425 -0.4279582697  0.9304547588  2.1177296280
[5476] -0.4155962816 -2.0229923604 -3.0504553168  1.9008236909  3.4919421896
[5481]  0.1345169264 -1.1072179437 -0.2596362321  0.4407473370  0.1781161922
[5486]  2.4517809447 -0.6250558116 -1.0018750857  1.0690953486  3.5718696052
[5491] -1.1850864082  1.5251056187 -0.3883313310  0.8612668691  1.2048133003
[5496] -1.1127957731 -1.9325718105 -1.6997436702  0.4995931831 -2.9309449912
[5501]  0.1149174939 -0.7310976371  1.2603327735 -0.9011040872  4.5224137682
[5506] -0.9884045845  2.5430836017  4.8430567814  1.3734825430  1.4979369005
[5511]  1.3655651936  1.2062238897 -1.1140444900 -3.3131078287  0.5477280531
[5516]  3.6366605955  1.7777656710 -1.9938371112 -2.2021294071 -1.8274969968
[5521]  2.4295744031  0.4769619351 -2.1421893216 -1.2291036851  0.7159367534
[5526] -1.6310162859 -0.0283751970 -2.4866864613  2.1231760983 -1.1366945050
[5531]  3.0689923326 -1.0859637726  1.2293767445 -0.4775101972 -1.2858974575
[5536]  0.4371191619  2.1399215326 -1.3627767348 -2.2136519752 -0.8111274518
[5541] -0.4997694015  1.2410639571 -0.5290183804  5.0107700617  1.7355918691
[5546]  0.1326306946  0.6428968308 -2.1993387762 -1.3404638134 -2.5885643197
[5551]  1.2436300415  0.7968105985 -2.3252692788 -1.0445339239 -1.1719959420
[5556]  1.5305487448 -1.1956851039 -0.2121514307  1.5149589594  1.2470586785
[5561]  2.1065704889 -1.8757650640 -1.1021507414 -0.1674673384  2.3711035613
[5566]  3.4913621340 -2.4815767184  0.4714666079 -1.0980635484  1.3745453538
[5571] -1.8543182647 -1.6081947976  0.5686115689 -4.3526163725 -1.1057330385
[5576] -2.0182156892  1.4522676400 -1.1689693050  1.6736783753 -1.2055501728
[5581] -1.0629108763 -3.3674522469 -0.5072294777 -0.0365636637 -0.1104517324
[5586] -0.3266900907  0.3587339925  0.2692485941  2.8907510050 -1.3205796116
[5591] -3.8471326456 -2.3672733355  0.0890516934 -0.9023099607 -1.6462895271
[5596]  2.1820964697 -2.6082009804  0.7342288669 -1.3948240504 -1.5012170633
[5601] -0.7140736484  2.0508803353 -0.0155362461  0.2013620766 -1.0205641376
[5606] -0.9406246435 -0.0128780919 -2.7253694025 -0.4303109646  0.4446539019
[5611] -0.1695602049  1.0815995616  1.5182686664  1.1753901049 -0.6666656180
[5616] -2.5682740165  0.8118302897  0.2226687246  1.2130496734 -0.1501407240
[5621]  1.4888025445 -0.3368339824  1.5239534865 -0.6582307540 -0.6296724529
[5626] -0.9001297527 -2.3988501300  0.1187846097  2.7236119549  1.5715834661
[5631]  4.5830601229 -0.5806156645 -1.8534424964  0.1073239351 -1.6958843359
[5636]  0.3646143381  0.4488369801  1.4721661810  5.0402605987  2.3934273716
[5641] -1.0919581781  2.4257919029 -1.7280802131 -1.2660133270 -1.9518546345
[5646]  2.8279023941  2.6053560518 -1.4294819950  1.6843154719  0.7854462742
[5651]  0.5278528470 -1.3191181149  0.2898069784 -0.6555873445 -1.7995169000
[5656]  1.3129723840 -0.0362838358  6.1549545929  0.8972990685 -1.3759356711
[5661] -1.8776595142 -0.8811385000 -2.6659573402 -0.8861027212 -0.9180102635
[5666]  0.6919040778 -0.6368734147  0.0571908816  0.7894522012  4.7226420136
[5671] -2.0603567874 -3.2646925365 -0.0539761411 -0.1158125587  0.5691833198
[5676] -1.8254247857 -1.8025641461  1.2354431726  1.4616904889 -1.1668875652
```

```
[5681] -0.2049529363  0.1700544812  1.3672561688  0.6492252621 -1.0377702696
[5686] -2.1949450457 -0.2035282628 -0.1840282430  0.6987562919 -0.8645493018
[5691] -0.3773549355  2.0426993134  3.4073593483 -0.6472319322  0.5420255495
[5696]  2.7646000376 -1.0553520203  0.2844700316 -3.5060182242 -1.0547430722
[5701]  3.1973386608  0.5797654674  0.6364556656  1.9656383797  2.0690204494
[5706] -0.6399268217  0.8481141696  0.5374613598  0.2965417202 -0.1905699626
[5711] -0.2187808702  0.8229782209 -1.3892127872  1.9098253453  0.1404494307
[5716] -1.2237836275 -0.4804578261 -2.2310446316  2.4680273869  0.9197149666
[5721] -1.0630206217 -0.4610932597  0.8716710883  0.3187047225  2.4524190358
[5726]  0.7578566581  2.6878553234  1.0007398948  0.7347953653 -1.7883338853
[5731]  2.5193475818  0.7726743205 -0.0509560445  1.5323894716  0.2937720933
[5736]  1.4010887515  0.5182072300  0.7341127180 -1.3941866469  0.3997766209
[5741] -0.7705048775 -0.1591407975 -0.0770612199 -0.5811452266  3.9212769462
[5746]  0.3698749983 -1.8585415946  0.7300415277  1.4380597610 -1.7720586115
[5751] -1.6689948538 -2.4145522102  2.3605781286  1.5842622267  2.1253677463
[5756]  1.8528740182  0.1019570874 -1.3516577605 -1.4047842142 -0.7239949756
[5761]  1.8698802900 -1.2772123006 -2.3559788819  0.0100216504  0.0766960269
[5766]  0.6195025180  0.9980235262  0.8778615709 -3.2944735442 -1.9319224432
[5771] -0.9987299624 -0.3301497730 -0.7170090558  0.4825939521 -0.1085207994
[5776]  0.9031584119 -1.7093485942  1.7168675122  0.7804005427  1.4595366149
[5781] -1.0782076339  0.9762779128 -3.0173283941 -2.1652457872  1.1989928056
[5786]  1.6493772625 -2.2598835728 -0.6626567264  0.2570591358  2.3468020802
[5791] -0.7485821432  0.6227306612 -0.0986026640 -1.6392791628  0.1042198031
[5796] -1.6029479156 -0.8205744051 -0.4890415080  3.8021659606 -1.4198945321
[5801] -0.5266429687  0.6540613586  1.7680034926  2.7008409399  0.6857340969
[5806] -1.4563370585 -3.4117501466 -3.0488796484 -3.4243915620 -3.7081260484
[5811] -3.3748614076  0.0206234423 -1.9117898535  1.6219996157  0.1865912858
[5816]  1.4217916594  3.5055415274 -1.2110654240 -0.3394506764 -0.9891368878
[5821] -2.4107407927 -2.3333151203 -0.6201531858 -0.1246842766 -2.3857540681
[5826]  1.2570868836  0.9824234170  1.2553979972 -0.0216017981  0.0801050971
[5831] -1.6353952661  1.6114264858 -2.7290840852 -1.2105445616 -0.1015393562
[5836]  1.4738907413  0.1566385779 -1.8125833815  3.0407234418 -1.8775980748
[5841] -3.5927331000  2.7611487274 -0.1420250773 -2.2415578737 -1.0575898933
[5846]  1.5511720718 -2.1713270670 -1.0426527974  3.3605794320  2.1380706597
[5851]  0.7286722828 -1.2809998877 -0.0174286945 -2.5155944612 -0.2631450530
[5856] -0.4207057520  2.5786095781  2.2032998733  0.0037674774 -0.3810522406
[5861] -2.4185027083  3.1182873527  0.0796151882  2.7768674470 -1.7449965132
[5866]  2.1400475619 -0.6725144814  2.5874828490  0.7936388922 -0.6765843381
[5871] -1.2369955940 -4.6504550516  1.9867285071 -1.6161509959  1.6316733809
[5876] -1.9326907477  0.8965020275 -1.0714717364  2.6510502987 -0.1570561790
[5881] -2.6518430764  0.8290309171  1.8958463621 -4.2007429549  1.3617359966
[5886]  2.1787154035 -1.4225758782 -0.4399430197 -1.3989368432  3.5590881582
[5891] -0.2123447492 -0.5093136664 -0.5541134190  0.4986158058  0.2066509489
```

```
[5896]  2.0075362412 -1.1343837357 -1.5878170409  0.0252601590  0.9020530029
[5901]  1.1488866241  1.0815343319 -2.3192957447 -1.9654895396  0.4414376740
[5906] -1.3888849204  0.1999476707 -2.8979220200  2.9366334385  3.2596394521
[5911]  1.7526890238 -3.1184110220 -2.3685703444 -3.0960126676 -0.4855680323
[5916] -2.6658778852  3.0024797725  2.3769581413  0.0643953028 -5.0184813366
[5921]  1.7579046474 -0.4107982310  0.4611624974  4.0224989447 -0.4987690257
[5926] -2.5503010095 -1.8626414753 -0.6084724089  1.3522103613  0.2228407757
[5931] -1.5587333426  2.9130785406 -1.1132888893 -0.3411672939  0.4040282314
[5936]  1.8784470415 -2.5649684926 -0.8911699741  0.1970625228  1.2628160118
[5941]  2.2057659679 -0.0597342917 -0.4319581268 -2.9081981374 -2.4356754737
[5946] -1.9412448262  0.1185836699 -1.4033188125 -1.3780120197 -1.4347778064
[5951] -1.3599155898 -4.9624355821  1.4526537382  0.8084291767 -2.8488852469
[5956] -1.8163565162 -1.7719802115 -1.1968415842  0.1756579874  2.5099417872
[5961]  0.5855667617  2.7067133828 -0.3852624813 -4.0720494062 -2.0863917680
[5966]  0.2347824430  3.4550511336  1.5526159717  2.0100521602 -6.2978547323
[5971]  0.5879848434 -2.7653268524 -2.0782510660  1.9382782606 -1.6509763513
[5976]  0.0703167102  3.5417166224 -0.7743591473  0.9434785919  0.6957610434
[5981] -4.0949541731 -1.1422293445 -0.2844702172  0.9871354341 -1.7332580783
[5986] -1.0673096000 -0.7890356219  2.6718532444 -2.1531417474 -3.0761535915
[5991] -0.4889304483 -0.1200103940 -3.8240989971  2.8268580574 -0.5682750202
[5996] -0.7952840381 -1.0927808725 -0.7824669480  1.1853131736 -2.6322656796
[6001] -2.1134903946 -0.6955567185  1.3660495522 -0.6633101370  0.7426615887
[6006] -1.9957708447 -1.1662052462 -4.5950693074 -2.2249139917 -0.6593551625
[6011] -1.8728992023 -1.0720231617 -0.0288446526 -2.0910754585 -2.2186055844
[6016]  0.3184467935  1.6840562245 -0.3671423449  2.7420401507 -1.0987992986
[6021] -0.5618439029 -1.7509495527  0.2965106484 -0.4182524686 -0.2073545711
[6026] -1.3224382803  2.5603556023 -0.4040094184 -1.0760140720  2.8567621153
[6031]  0.4789876337 -2.3147757805  0.6549616701  0.7085822248 -3.9535703889
[6036] -1.8268116021 -2.6872480486  1.2973104300 -3.2521804879  0.2522116240
[6041] -0.1105057153 -0.4633595414  0.5792417126 -1.5606666858  0.5491866789
[6046] -3.3062969018 -0.9260305877  1.8242680475 -1.3888012220 -2.2691798935
[6051] -0.1063439909  1.8134839821 -3.3037630167  1.9430833221 -1.2093499625
[6056] -0.6303632595  0.2838464701 -1.3523641273  0.6060018842  2.4090076378
[6061]  3.6395652649 -1.7323426053  1.9578501701 -0.3059561361  1.3446528962
[6066] -0.2719280281  0.1737803172 -0.5322577589 -0.5734710706  1.3026570923
[6071] -0.8587002480 -2.7248692640  2.7495358482 -1.0650399240 -2.7658410907
[6076] -1.2057442120  1.0440169166  1.2329037210 -0.5740571402  0.9094417302
[6081] -2.9240373524 -0.4655612429  3.4480218853  0.5055854842 -1.6471835945
[6086] -1.5881587683  3.2953494991  2.6108949705  0.8988465627  3.1153928651
[6091] -1.9311580858 -1.9011496980  0.9289770508  0.6067672474 -0.8519680080
[6096] -1.1323817887 -1.5094587190  4.4066613370 -0.9637488584  1.5435294556
[6101]  1.5377085130  1.3942650045  2.1684098498  2.3186138525 -1.2202689863
[6106] -1.7396725889  2.0834009266  1.4247064353  0.0569608186  0.1522111399
```

```
[6111] -3.9138833371  1.2002223148  1.3806114658 -1.0429384495 -1.5953525835
[6116]  2.0700841564 -0.7401826545  1.7917040092 -0.3648306286  0.3783018718
[6121]  0.1414102941 -0.7964209305 -1.4634156286 -1.3010132405  1.9660819261
[6126]  1.9669252286  0.7304380778  1.1537999793 -3.2388104012  2.8139202573
[6131]  1.0485942121  1.8549246557  0.7988159261  0.5955802731  1.6956906429
[6136] -3.1286810333 -0.7496696197  0.3239742105 -1.6342416092 -1.7342873865
[6141] -1.9785395263  1.4811409501 -1.0670291451  0.8877401988  2.2870483058
[6146] -1.4108723162 -0.0671972246  0.9491956339  0.3424074234 -0.7102522239
[6151]  0.7760507133 -0.1553561102 -3.6526936977 -2.4305873060 -1.2373303522
[6156]  1.1674495063 -3.9070225460  0.7103122317  2.4476530873 -2.1530943582
[6161] -0.7788568965 -0.4604170463  2.3312187035  1.2629664888 -1.1804408992
[6166]  2.9527186500  1.6694849022 -0.5377090283  0.0841282461  0.3859001007
[6171] -1.2327321909  1.2133707024 -1.4471126209  1.0264836703  1.2811089607
[6176] -1.8974055420  0.2264745049 -3.0636663114 -0.6136213851 -1.5375381812
[6181]  1.8556578294 -0.5475078897 -0.2509243702 -2.2787402470 -0.9633578624
[6186] -2.6513715259 -2.5652366175  3.7570541042  1.5610710325  0.1665949945
[6191] -0.8937172690  0.7882934517 -0.5292468451  1.6497832837 -2.4396419571
[6196] -0.5427772617  2.3149514656 -0.0295652274  3.6890081036  1.6039213748
[6201]  0.8082350456 -1.5700093811  2.8386544575 -3.3272757212  1.8817794530
[6206] -0.4480661182  0.1352682487  1.1599730325 -0.4506174483  0.8658802077
[6211] -1.3902364715 -1.6759942083  0.3550611319 -1.4786953174  3.0470539138
[6216]  6.3822751437  1.7950469157  5.1199811991 -3.4325669126 -2.8008448179
[6221]  0.3187593194 -1.0962376651  0.8950414279  0.1085629110  1.5165629098
[6226]  0.0769128259  2.9396393388  3.2263825234  2.0449025558  0.4098383147
[6231]  0.8251971499  1.5369428683 -0.1789202117 -0.9475516790 -3.8346672919
[6236]  1.1308310254  2.7374058129 -0.6392615420  0.9876518500 -0.5854400198
[6241]  0.1577257483 -2.4981919063  3.1800211285  1.2670325207 -1.8239212299
[6246]  0.2077612357  0.0691492269  0.7858909789 -0.7554783114 -0.4966294381
[6251]  3.3419403901  0.2928173105  2.1525170118 -0.7651772124 -1.1693853582
[6256] -3.4100642091  2.3223750815 -1.7067426746 -2.4081886299  1.8357475336
[6261]  0.3962826501  0.5330383313  0.0820953783  2.7953722899 -2.3891807516
[6266]  0.8019574848 -0.8631132984 -1.5317802792  0.7211412182  1.7893989027
[6271] -2.6993468896  0.4402221825  0.5762470388  3.8126680812 -0.6303332677
[6276]  3.2916698426  0.4665599669  0.8582516851  1.3810697115  1.4617654839
[6281]  0.8359343668  2.1464731753  2.7197582944  0.1043497692  1.8198679730
[6286]  2.9394713746  0.3380453911  1.8588769801 -2.8409233694 -2.4714058447
[6291] -0.6133319138 -0.9844938285 -3.5082043372 -0.4948581970 -1.9653967265
[6296]  0.0714716174 -0.8038012574  1.3691001312 -2.0210585674 -1.4142195439
[6301] -4.4804815648 -0.4413982351  1.0155607908  1.3676359834 -0.8974935003
[6306] -1.1341646230 -1.3444666774 -1.6817644437  2.5604888410  0.6201498061
[6311] -2.5188824979 -0.2885790763  1.3689917428 -3.7576770730  1.1799372396
[6316]  0.5772767297 -1.2766817376 -0.0167752856  2.1890737537 -0.6627345988
[6321]  0.4907807741 -3.1195014477  1.0957959088  1.7961709710  0.8897534129
```

```
[6326] -2.6309543184  0.1902407166  0.3429547303 -1.8243281468  2.0163890860
[6331] -1.0354830991 -2.2412662678  1.7790531571  1.3679965149  0.9912415330
[6336] -0.7274207023  0.6901433896 -1.6683333795 -3.2716811706 -3.3372570408
[6341] -1.7857502390 -2.5050519077  4.4870394009 -3.0089135546  1.3061339953
[6346] -1.8623502611  0.4166418876 -0.5288439100  1.1187976429  1.3443808116
[6351]  2.7313535018 -1.0624125763  0.8304316117  2.0608578127 -2.0838501996
[6356] -2.2276552447 -0.4279096725 -1.1570822024 -2.9642141425  2.7273353276
[6361]  1.3128044596 -1.3462762591  2.8206877155  0.0153052443 -0.7398445020
[6366] -0.7497926230  0.3938837521 -0.2716706784 -0.1705452388  1.8206435389
[6371]  2.0930335028  4.2185768255  2.3058366120 -0.1922867945 -0.3246312419
[6376] -1.9538466006 -0.9867368944  0.8386884592 -0.7414375643  1.9902918479
[6381]  0.2879196825  0.3841679575  0.3244518286  3.7709284271 -4.1645007211
[6386]  1.6280087277 -1.7547379366 -3.0519941794  0.0963835664 -2.2825481767
[6391]  0.3001814390 -0.7677986614  1.3845917577  0.3841287389  1.9930710432
[6396]  2.4976117029 -4.7679748028 -0.6003969174  2.3142058222 -0.8745639147
[6401] -2.1094143891 -0.1342938890  1.2776651906 -0.8370179828  2.8529002975
[6406] -1.3679972012  2.0798033238  2.1095994155 -2.9180421918 -0.7184678307
[6411] -0.8691229027 -0.7657524828  1.6866910108  0.5275511056 -1.0040881827
[6416]  2.3841293527  1.1764328457  2.0118151905 -3.2564086976  1.1668954170
[6421]  1.2409969110  0.9120337686  3.0032635121  2.4546318069  1.8462190571
[6426]  1.6536005590  0.7178496187 -1.7122903090 -0.1279148349 -1.2328788790
[6431] -0.5463265745  1.9383209114 -2.3224358105  0.9626909319 -2.2613843921
[6436]  0.1954781081  1.1446532734  2.2068269857 -0.5436488441 -2.3332145442
[6441]  0.0514685257 -4.4706010587 -1.4376544735  2.8785431657  0.8259325904
[6446] -4.1623097548  1.0717142003  1.3007959315  1.3197682119  2.4895835247
[6451] -2.5109394188 -1.1636310892 -2.5460157215 -0.5202877834  1.1054354811
[6456] -0.3760715596 -2.6015166407 -0.4111794106  0.6410129133  2.6893096659
[6461] -2.3905381313 -1.3292764182 -0.0389769551  0.6344656005 -0.6763127340
[6466]  2.4037610643 -3.4020468779 -1.0666024012 -2.0125337982  1.0854279699
[6471]  1.7938746108  3.4940374524  3.7923536851 -1.0320262070 -4.7585246948
[6476] -2.1475089561  0.3949195765  0.1714128402 -0.3719151758  0.8816536466
[6481] -0.6503653203 -1.3551497065 -0.2231269880  0.8860019171  1.6707776845
[6486] -4.1271164934  2.2521968691  1.7153846288 -2.2693304577  1.3635636147
[6491]  5.1208349043 -0.9231496615 -2.3839658546  2.8949519923  1.2267955787
[6496] -0.2000204352 -2.1595223548 -0.9581068030  1.1968361963 -3.3034555685
[6501]  0.4806503813  1.7905348809 -2.0606120137 -1.8502327523  2.5466359204
[6506]  1.0369983761 -0.4832131451  4.1043028228 -1.8464819498  0.3706422581
[6511] -1.3364029514 -2.6915320629 -0.4284751205 -0.1193272234 -1.2462464231
[6516] -1.5977265761  1.9169460739  0.0989907487 -0.4981181533  2.8659969180
[6521]  2.1632396461 -1.8601162577  0.0533684839  2.3602850462  1.2602097524
[6526] -0.4036121355 -2.3707725930 -3.0458119981  1.8124162402 -0.0831988621
[6531] -0.6409970056  1.6514687103  0.0157315946 -1.4057907031  1.3528325225
[6536]  0.7690698685 -0.6443364240 -3.4215378375 -1.7293599546 -0.3980937929
```

```
[6541]  0.4226811701  0.8985984850  2.2805640084  2.4432400795  0.6765605145
[6546] -2.9678613763 -1.4409101476  1.5592695850 -2.5095506055  0.2853095359
[6551] -2.7835754681 -0.9024674599  2.0469094117 -0.2217123777 -2.0703384317
[6556] -0.9268232993  0.1289783271 -3.8486976745  0.2879003443  0.1546665375
[6561] -1.0584468032 -3.3791179592 -0.4450699549 -1.7979510729 -1.0788621173
[6566]  1.1155316597 -0.3663400508 -1.2903220879  0.3697754059 -2.3462204040
[6571]  1.1446739453  0.2479018653  4.7839263172 -4.6913225485 -1.1239959020
[6576] -0.0255006941  1.1025834632 -0.1682742041  0.9958140633 -0.2218979264
[6581] -1.5371618803  1.6795685567 -0.8953191185 -0.3544397048  3.1261217059
[6586]  3.4362770967 -0.9347036079  0.0376604535  1.2298725603  0.7147415771
[6591]  3.1435031879 -2.3393973743 -2.6649122110  1.3378309017 -0.4812454156
[6596]  1.2373972616 -1.6310801199 -1.1825971140 -0.3184072282 -0.3209200572
[6601]  2.3272234652  0.8691647573  4.8780863964  0.3140030698 -2.4155845533
[6606] -1.7763771697 -0.5977229297  1.1418373969 -1.2583946920  1.6730961334
[6611] -1.0816804956  1.8919084917 -1.9396013670 -1.3569571426 -1.9477375694
[6616] -1.7012467176 -3.3528084575 -0.7313380334  1.3153119490 -3.6913954549
[6621]  1.0264015716  2.6854399829  0.2025378062 -0.4720660168 -3.9373501260
[6626]  0.1443664228 -1.9045362184  1.2008499898  3.2696855724 -0.7180458168
[6631] -2.2370507350  0.7300925706  0.5572681546 -0.1810334369  0.4742498515
[6636]  1.2984500341 -1.0691296292 -0.1812210131 -1.0371775934 -1.9825205326
[6641] -0.7293312969  0.5924142682 -1.1782216565 -1.5612774843 -0.0570452777
[6646] -0.4553420872 -0.9447092202  2.1112244059  2.1788136763 -0.3732101859
[6651] -1.7465302544 -1.5448433114  0.9620698881  1.6632355971  0.0816196488
[6656]  1.7836391282  0.0452830850 -2.0836931044 -0.0817798310 -3.1297964146
[6661]  1.5934259634  0.3601794027 -2.2748168706 -1.2654119025 -1.8206309692
[6666] -0.3124857690  1.2224455626  0.6452560745 -1.4504798338 -0.2397916561
[6671]  0.7714970264 -3.4670041312  2.5047278826  0.4477690607 -1.9951503511
[6676]  1.1000304491 -0.6753006038  0.1047575455  1.4073924978 -0.6343911791
[6681] -3.0258549539  3.5799980024 -1.2718280464 -0.4734533100 -2.0118576728
[6686] -0.1985729965 -1.0047635375  1.4870661027 -0.4664718643  0.7930146105
[6691] -0.1678008950 -3.9826199243 -1.3255765152  0.8296474753 -1.8739734374
[6696] -0.8404031317  2.2187593877 -2.1944641066  2.6870091448  1.2168327246
[6701]  1.5478788095  2.3107070570 -2.5677014551 -1.2610536526 -0.7305564105
[6706]  1.5356686117 -0.3774758217 -0.1223609265  0.0850596002  0.6836525550
[6711] -1.4345295626 -0.9342629063  1.7624647652 -1.2504970762  0.6940061606
[6716]  2.4607208794 -2.8496150460 -2.5064504212  2.1124326375  1.9117376210
[6721]  2.3571898483  2.5896418951 -2.1140809724 -1.2021058721 -1.7479838832
[6726] -0.3219111499  2.9075775430  3.1645888730 -2.9075359525  3.1350413048
[6731]  1.9462771323 -2.2912007323 -4.1463799180 -0.5682448043 -0.5305995880
[6736] -1.2955836268 -0.8614509845 -2.0540132496  2.2600270999 -0.8557507143
[6741] -0.8968091762 -2.3097251562  1.6072296391  2.4988478386  1.0315631630
[6746]  0.4758244004 -2.9140203333 -0.5617563384  0.6234605811  1.3905383572
[6751]  4.5334251586  1.6058680484  0.0614292716  0.9610637554  0.0212916640
```

```
[6756]  3.4630158435  5.1272000111 -1.1805951441  1.4633914018 -1.6295673707
[6761]  1.3113779175 -1.4418383797  1.7902094599 -1.7544303472 -0.2170320078
[6766] -1.9436701344 -3.4357299522  1.1018201239 -0.5803224936 -0.1798434103
[6771]  3.0350536473  2.4555731923  1.1422239993 -1.0155519425  0.9854485133
[6776] -2.6988164687 -1.0401993017  1.9862759966 -3.1212101040 -0.8331977993
[6781] -0.0280917301 -5.1466268313 -0.7496760955  1.4509023932 -0.5926603767
[6786]  2.6205867111  0.4717847194 -4.1053016966  0.7995624160  2.5085732754
[6791] -2.7200470900 -3.2874640667 -1.1382464622  0.0385175195 -0.2014842772
[6796] -3.6061451722 -4.0342353483  1.9511674589  1.3047751729 -1.2987239382
[6801] -4.7429090009  0.1101816425 -0.5504996211 -0.6986273852 -0.6628827425
[6806] -1.3444070649 -1.9167741864 -5.3137453523 -0.3285785160  1.0450052099
[6811]  2.5666807738  0.2002022703  2.2761453112 -0.3733612635 -3.8819785217
[6816]  1.2788644302  2.0881618302  4.0051231904 -3.9575190525 -0.6776352059
[6821]  0.9323296313 -0.3617775855 -0.2063568253  0.8492111552  1.5848803607
[6826]  1.8072925080 -1.0401231097 -3.7995398864  3.0769400532 -1.6843045716
[6831] -3.6827419235  2.6273970122  0.6107213947 -1.1642172326 -1.7776256604
[6836] -1.1824369071 -0.6875896641  0.2037445452  0.0153014805 -1.2705529861
[6841] -1.9725707587 -3.9409248798  4.4996611280  0.5848744992  1.5233758145
[6846]  0.8291947140  1.2972950298 -0.7242575192 -0.8852897667  0.4129622093
[6851]  1.9243169067  0.1085821722  2.9257059198 -0.9517501171  3.0042390805
[6856] -0.4799866713  0.8737978653 -0.2676676609 -2.3409229703 -3.0080489507
[6861]  2.0568619433 -1.1771797050  3.8562343206 -2.1422379882  2.0636684279
[6866]  0.4931147618  0.5719493341 -0.7700553434 -1.7864596076  2.4139723772
[6871]  0.7619389515 -1.5202486990 -2.0666881623 -0.4668599965  2.0268368117
[6876] -1.6406048807  0.5539938215  1.5322075356 -1.6100178925 -2.3108919591
[6881]  2.8386874898  0.8096439216  0.6332740886  3.6334515988  0.3737615494
[6886]  0.4097635839  0.8570319309 -0.4130773981  1.3632891904 -1.5993770669
[6891] -1.7093905817 -3.9730893437 -1.1428620737 -3.0567987853 -0.6210350141
[6896] -1.8635127366 -0.0160973157  0.5391686820 -1.7506350798  0.5537414910
[6901] -2.3772317988 -1.7838414414  1.2122302088  2.3510711753  5.1170472840
[6906] -5.0743319957  1.7519571619  1.3710400677 -2.1764745396 -2.9231958148
[6911]  1.1707357194  2.3755286682 -1.7332200491  1.2418947016  0.1996025730
[6916] -1.0016875828  0.1063126190  0.1231098551 -0.6090752952  2.4115154617
[6921] -2.5760939556  1.9245735672  0.2428653498  1.8254181236  0.1703286836
[6926] -0.5531751292  4.2616844219 -3.2480596088 -0.4393096902 -2.2908725067
[6931]  1.6073400419 -0.2988785097  2.3274987167  2.1156758842  4.1047584940
[6936] -1.4140863219  3.7406030359 -0.1770499504  3.3715410125  1.4366804051
[6941]  4.2248445425  2.6506099829 -2.0640030705  1.6488424942  2.0254253837
[6946]  2.6415599713 -1.5535283324  2.3947844812  0.6324168377 -7.2671214771
[6951] -1.9390935786 -3.1314060778  2.5614653556 -0.1820904114 -1.3112183622
[6956]  1.6682862723  0.2766878027  1.3323270746  4.8210996174  1.1756907374
[6961]  4.8335781409  1.0357948170 -3.1480989949 -0.7285115796 -2.7092802417
[6966]  0.6188475688  1.0629907544  3.5276624592 -0.9003695165 -2.2696689609
```

```
[6971]  1.4705616151 -1.0020014398  3.5655702442 -3.9163670878 -1.7559738864
[6976]  1.7836665341 -3.9876418712  0.4066862293 -3.8775154757 -1.3402856123
[6981]  0.6025705681  0.2118383176  1.9245079746 -1.6256197890  2.5136187523
[6986]  0.7570384700  1.0761389619  1.6540092347  1.4548871709 -2.8731026185
[6991]  0.4010736272 -3.2045940769 -2.8561422937 -0.9197881138  1.4221300214
[6996] -0.1051363257 -1.4174090018 -1.0407742831  3.3603991927 -0.5172172816
[7001]  1.0168179266  2.5780365156  1.6739639724 -3.0317549537 -1.2002378051
[7006] -0.6759303962 -1.9780375487  4.2532133732  2.4251968560  0.1494527106
[7011] -2.6042910405  2.0137436626 -2.6601234931 -0.8606181694 -0.4028299026
[7016] -0.0054592062 -2.8353656400 -1.1881312222 -2.3518476769  0.2326748779
[7021]  2.0805024439  0.3201096315 -3.0154769219  0.4787684104  2.0441813382
[7026] -1.0940586024 -4.6942062476 -4.6038090396 -1.9251289745  5.1929642074
[7031]  0.8506488185  0.6862268462 -1.2219680319  0.2565105091  1.4724391497
[7036]  1.9675620876 -0.2768785137 -0.6438189514 -0.7717279942 -0.4957892224
[7041]  0.6863653606 -0.5707786191 -1.4228740365  1.3024423850  0.2095359547
[7046]  1.5408747167 -3.8265873852  2.5852413898 -0.5916193659 -0.0451179748
[7051] -1.9309679611 -0.9367223545  0.9117576405 -1.7227559055  2.4474182931
[7056] -1.4346921514  1.5642288232  1.2046443481 -0.8798235772 -1.1624080650
[7061] -2.0576014956  2.5632371059 -0.0775472010  1.7794526110  2.4431523227
[7066]  0.6499219567  2.0885505038  1.2795901958  0.6697520242  1.0486097883
[7071]  2.9937571716  1.4129065221  1.9805388038 -2.9844500573 -1.4744849855
[7076]  1.4033802412  1.6921929849  1.8953704614  1.0638252201 -0.9258381215
[7081]  0.5333489994  1.7429577666 -0.1251244319  3.0936575533  0.4748021142
[7086]  1.4043155322 -0.1813922154 -0.9473165106  0.0237063683 -1.8794579466
[7091] -1.2433467335  1.4764934766 -2.3597008529  2.0175003168 -0.3422378724
[7096]  0.9267873260  0.9707061779  1.8694425849 -2.4833697590 -3.0227730962
[7101]  1.2123616810 -2.5985719339  4.8217662453 -1.0859525083  1.5170118193
[7106] -1.1044464742  1.1149370202  3.3122311458 -0.2196802941 -0.4256611901
[7111]  2.8607750107  0.5775382582 -1.2810765543  1.2743974832 -0.0214823817
[7116] -1.6856759460 -2.6895572834  1.5766256828  1.3284632379 -2.2119957426
[7121] -1.7228814946  1.9439334484  1.0368617335 -0.3809054652  1.8533383947
[7126] -3.6764680803  2.4352999294 -0.1260139271 -1.8447505560 -0.1176303042
[7131]  3.1468246445 -4.7457423672 -1.6096695018 -2.8327326536  1.6258841238
[7136] -2.2887277720 -1.1427676465  1.8729280306  0.5050338373  0.6056648240
[7141] -0.0275773131  0.6589584996 -0.5388909707 -0.1235564590  3.4292561965
[7146] -0.2161536595 -4.6197039166  1.4661600335  0.8751302088 -1.5891524007
[7151]  0.4684887599 -1.7682837430  0.4172846670 -0.7880202903  0.5925125287
[7156]  1.4683304770  1.0969758816 -2.1424555663 -3.1560244304  2.6382543347
[7161]  0.9710684533 -1.3188283481  0.4670115787  1.0949714028  1.9968599466
[7166]  0.7305984326 -1.3164451841 -2.2868162670 -1.5932242268 -4.7584457427
[7171]  0.2904372893 -6.2845097308 -1.3140856623  3.6524955993 -1.3778060722
[7176]  1.9075602381  2.0702624367 -0.2640619327 -0.3956919701  0.7638406343
[7181]  1.0405344636  2.3834133689  0.8106041832  0.1658483600 -0.4628979157
```

```
[7186] -0.1171922747  1.7265212815 -0.3177228200 -0.6916178547  4.3240182713
[7191] -3.0417649120 -4.7545908910 -2.0802725320  2.1274682238 -1.1153699768
[7196]  1.1971990508 -3.8781062682 -3.1893690273  1.7447199022 -2.2568447520
[7201] -5.2501715380  4.5403235472 -1.6699133259  0.8769572920  1.2188788318
[7206] -2.3731229270  2.1536179430  1.8986705337  0.7307835208  0.6291878835
[7211]  3.1861001326  3.4995915628  1.6318010081 -2.7639917870 -1.1247172650
[7216] -0.3606046318 -1.7850897786 -3.9492917346  1.9871870084 -1.5436530801
[7221] -0.5693952266  4.4607618507 -0.9000811981 -0.3072030769 -0.3777683271
[7226]  2.8530361986  0.1105334341 -0.1228264474  0.2280680328 -4.0520661205
[7231]  0.1787466865 -3.6161371890 -1.7836377348 -2.0202630853 -0.2099305872
[7236]  1.0940674518  0.9139241593 -1.2605062044 -1.3063867354 -2.7962190219
[7241]  1.5637949906 -2.5001178328 -2.5870678231  2.3102007223  2.3001897158
[7246] -1.3972966425  0.3033881337 -1.2808807096  4.6701424396 -0.1927994928
[7251] -1.2080650274 -0.6090869484 -0.6783341201  1.5230834127  2.1889643829
[7256]  3.0204755619  2.4045234463 -0.4005998348 -1.4213494028 -3.0347530507
[7261]  1.5569863094  3.0282585387 -0.1827917415  2.0567966467  0.8878991569
[7266]  2.2103392975 -1.0224804475 -0.3579891799 -1.1326267275 -0.8289444265
[7271]  3.2313854722 -0.2958180272  0.0161397294  1.9464373807  0.1651565671
[7276] -1.8266829167 -0.8950806609  3.0742643179 -0.3513510356  0.7916025222
[7281] -2.2402561733  0.9522432665 -1.1629775885  0.4416824272  4.1058415000
[7286]  0.3372861628 -2.4425529762 -3.5977528781  1.7209904655  1.4526915028
[7291]  0.6604581492  2.5480373665 -0.4244772730  0.4337719451  4.3164530170
[7296]  1.9056563146  3.5689172061  1.5655099984 -0.8306997345 -0.7941286400
[7301]  5.5476312494  3.5294442487  2.3117084325  0.9574695150 -2.8115058904
[7306]  3.7420636497 -1.8646690071 -1.5078994077 -4.4900935686  0.5826394944
[7311] -3.0579290319  1.1716774471  0.4348859056 -0.7498169482  1.8273310425
[7316]  0.4489376922 -2.9555133150 -1.7236539082 -0.1119647895 -1.6987796983
[7321]  3.3315521193 -0.1051232630  2.3488498961 -0.5217016236  0.3998507283
[7326]  0.4307515529  2.1162010975  0.9808505914 -0.1593425798  0.2762413267
[7331] -0.9469911750  2.8888311402  2.1044233746 -1.7475315987 -0.4896528057
[7336]  0.0524328139  1.0487333649 -0.2959726679 -0.1766913253  1.0622727740
[7341] -1.1311528890  0.0894483849 -1.5250061718  2.9151473245 -0.9495891910
[7346] -1.3203385724 -1.5332242060  0.9872945882  2.3884264298 -0.6219947572
[7351]  2.4603492864 -1.8710717325  1.3218189162  0.0686976419 -3.7255250518
[7356] -1.3802082777 -0.3583679871 -3.1520534464 -1.1009614868 -6.1890684814
[7361] -0.4924701047 -1.1689579525  3.7836999460 -3.3914673102  0.6603062541
[7366]  4.2435362219 -0.1310858779 -0.1388921005 -1.5848066365  2.9036406091
[7371]  0.9065702981  2.6174703138 -1.8365923403 -1.8692018256  0.5592912042
[7376] -2.4433282855 -2.2087311170  0.5767242677  2.1805528202 -1.1201021484
[7381]  3.1911948623  0.9243292571 -2.4784712216  1.8332217569  0.8069451749
[7386]  2.4902153021  1.1472238738  1.4264933333  2.4398794132 -1.9017504131
[7391]  1.3832795553 -1.3942107500 -2.7444722557 -0.6320277384  0.8227429208
[7396]  0.9866985218  2.2651587533  1.3684711114  3.1235975057  0.7810612497
```

```
[7401] -1.3158790130  1.4566734446  0.0624480739  0.7925363116  0.0127364846
[7406]  0.0290743057  1.2061285143 -0.1887065033 -0.6956870894 -1.5056303247
[7411] -1.5921804100 -2.3293189712 -3.9664986026 -0.9060194448  1.5792506184
[7416] -0.5013040041  1.4977415362  1.8096121415  3.0915646400 -0.1426992148
[7421] -2.5037561422  2.8717403428  2.8512153398 -1.2744881403  1.4108759012
[7426]  1.6940443259 -0.8548796791  4.2398782446  0.1197234492 -4.1524531404
[7431] -1.0886506469  0.8507694249  0.1307740111  0.2184936680  3.1338753326
[7436] -0.8070308081  1.1144163369 -0.2812863445 -0.9414917766  4.1358667137
[7441]  1.7089912434  1.7730097101 -0.6298483151 -0.5391045526  1.4098479162
[7446]  0.8389698218 -0.2310174549  1.0672963457  3.7674920710  0.6233399563
[7451]  2.0338942810  0.9200903471 -0.7858642923  0.0737276225  0.4257673981
[7456] -1.9030996076  2.4799368930  0.5922226684  0.1187995804  0.8609158058
[7461] -2.0109914099  1.2099329726 -1.7676521275 -2.9065098803  3.0046816563
[7466]  3.0358639310 -1.7647759202  0.2824530389 -0.2542431923 -1.3370872912
[7471]  2.2935427410 -4.2476769466  2.3283698504 -1.6438855377  0.5881081716
[7476] -0.6142115851 -1.5101988742  2.4246098858  1.0280147718 -3.7382535636
[7481] -4.3805774345  1.7482256961 -3.4700570680  1.1723073905 -3.1655814925
[7486] -0.0461225704  0.1418611715  1.0639639583 -1.0592306360 -2.3363821539
[7491] -0.0249705114 -2.0354827063 -1.9094318406 -0.7991525730  3.8917928222
[7496]  1.3949758614  0.9887164857 -2.7312829017 -1.3355875282  0.4607303324
[7501] -4.1163543700 -2.7568879598 -2.1454028527 -2.3324062158  0.4672317276
[7506]  0.4335945635 -1.1424171659 -1.9423666605 -1.5922811218  0.8339050938
[7511]  1.9408071137  1.1960434082  1.9500872611  1.1443796629 -1.3092189449
[7516] -2.1006907034 -0.8408017144  4.1835893809 -1.4772607630 -1.2263114058
[7521]  2.3285160838 -0.2743744389 -0.4095125536  0.6521383951  0.9294735861
[7526] -0.3587795129 -1.6824468063  1.9051005266 -0.0387708847  2.3697909735
[7531] -1.6119246816  1.1151636539  0.1417402356  1.0153956833 -0.2192708452
[7536]  0.6596004404  0.7452261506  1.4465660010 -2.0565708480 -4.3054637473
[7541] -3.5625455229  0.0021269117  0.1028905420  2.5124914923  0.1708335220
[7546]  3.3791821646 -4.1049197100  2.6049894283 -0.0668660960  5.2756551552
[7551]  3.5395562836 -0.3607554869 -0.2274565603 -3.0320112338  0.8482952198
[7556] -0.2812836425 -0.5078274216  3.1399488797  0.4258376834 -2.2945001386
[7561] -3.2443904175 -2.0432454713  4.5528710202  0.2348707585  2.6531461431
[7566] -1.3335357508  0.0422174616  0.7692153080 -2.2817476709 -1.3234819990
[7571] -5.6503581340  1.9017823610 -1.5689535828  1.1052831526  0.8158173753
[7576] -2.6119290202  1.1926538641  0.1586121330  4.0597428417  0.9122718781
[7581] -0.8618133584 -1.6670766730  0.4538090611  1.1385003009  3.2093154135
[7586] -1.8123928888 -1.1896024829  1.9044153867  2.4514140096  1.3244514599
[7591]  1.7161968951 -1.7871687305 -0.3945924546  0.1996115655  0.0159268197
[7596] -1.1763378585  1.8536141524 -1.0175175173  1.2374999365 -0.4485155188
[7601]  1.3860601788 -2.5033609265 -4.4537645175 -4.4513617451  2.2139961294
[7606]  0.8549265644 -1.6230748630 -0.2872014988  2.5398798623 -1.8077445646
[7611]  0.6401403667 -1.1751562351 -1.1345800745 -0.6479408826 -2.8976873701
```

```
[7616]  4.1227775597 -2.7071951216  0.8273837820 -0.1137977917  1.9203065076
[7621]  0.2988450424  1.5642457253  3.4118672862  1.2277076833  0.1488254061
[7626]  0.3416072210  0.1475417857  0.8825154425  0.8329795347  1.8900695110
[7631] -3.4106343079 -0.4872592116  1.5600121544 -0.4140527278  0.2222979574
[7636]  1.3699363863 -2.1308078368  1.5259287664 -0.5273725673 -1.5959203261
[7641]  2.1758259753 -2.8002585434 -2.5140235770  2.0812916081 -2.3966780716
[7646]  1.0469912695 -2.6250819001 -4.8706679106  1.2169270498  1.0861964124
[7651]  2.2373917510 -0.5161903656  2.3008672859 -0.0020433579 -2.7374212201
[7656]  0.4740430829 -2.6055068151  1.5230802978  0.8161380101  0.5404543516
[7661]  0.9293655103  1.9609209077 -1.5293479333  0.5598454357  0.6049800809
[7666] -1.3125445287  3.5741121441 -1.1499691961 -0.1380323824 -0.5735667539
[7671]  1.2980113781 -1.1272683294  1.0831676881  1.7744430015  0.9089486414
[7676]  0.2830316417 -0.3368646688  1.2015531600  1.2697585300 -2.9279960695
[7681] -1.1847612497  0.4691538278  0.2823944698  1.7098262267 -0.2399497726
[7686] -0.5519540156  2.5215414392  0.4347565820 -2.9707505159 -0.8043667255
[7691]  0.0666029459 -3.2428042810 -0.0485930322 -2.1992624232  2.3386859980
[7696] -4.4617406206  0.7458516309 -2.5190897175 -2.1057727977  0.1028417952
[7701] -4.0884584724 -0.9047615885 -0.5078464027  3.0255710933 -3.7775898072
[7706] -1.7364558247  1.3789761671  1.1557173430 -0.3452702173 -0.6889048375
[7711]  0.6408572266 -3.2067086292  1.5558926836 -4.9322925292 -2.0992941180
[7716]  0.4052579186  0.4022447773 -1.4392122907 -2.5896790979 -4.1947721956
[7721]  0.4754394427 -1.9420916085 -4.7636448721 -0.1322316913  2.5582788840
[7726]  2.1482237780 -3.1916551092 -0.8564500859  1.3428348076 -0.3515931720
[7731]  0.8516741329  0.2686373571  0.2008039526 -0.4993937437 -0.7166096887
[7736]  1.9963237613 -1.4985341956  0.0019586462  1.4842343819 -1.7248747144
[7741] -1.4366858678 -3.5905868257  1.0332576708 -0.5708404317 -0.6239758994
[7746] -0.0248493818  0.6617528487 -0.4799345545  0.9953630127 -1.8955320099
[7751] -3.7360814915  0.8250962649  2.3302207418 -2.1356733040 -1.0140356021
[7756]  0.6685969091 -1.6848159782  1.0400004783 -0.0853583158 -0.2959851973
[7761]  0.9851554990 -0.4181998645  2.1047352787  1.4184766593 -0.9722427857
[7766] -1.3065165658  0.5657354052  3.9131761149  0.5883289090  0.4939594535
[7771]  0.2183105003  3.4539366309  1.6909600103 -0.5035510156 -2.7183290666
[7776]  2.8059710972 -4.3688731175  2.5188782228 -1.4956835314  0.4108180278
[7781]  1.7829428521 -1.0013303175 -1.7850151864  2.8286261413  4.1074684450
[7786]  0.8665806669 -1.6798958611  3.3593774256 -2.2283657385 -0.0250405147
[7791]  1.9660056312  4.2221415561  1.4785049454  0.4003868793  0.8137257517
[7796] -0.2595814756  1.4036953246 -0.4194955988 -1.1783319535 -0.6435292098
[7801]  0.0496695979  1.4270107568 -0.9526674127 -0.9127857698 -1.0094140871
[7806] -2.0140346371 -1.6696746638 -1.2071924210  0.3078939669  0.7378536701
[7811]  0.5103682065  0.1322693628  1.5643057513  1.1154219964 -0.6774464000
[7816] -0.9206907792 -0.9746858499  0.0005393142  0.3424453522 -1.1413487989
[7821] -1.9623902522 -0.2928935598  3.1009051164  2.2903862981  1.3667195546
[7826]  0.0158944019  2.0475173547 -1.0054210617  1.6670830590  1.8694161400
```

```
[7831] -0.3758439013  3.8740195621  2.6651104254 -3.6418220053 -1.0157636249
[7836] -3.9245695212 -1.2037341288 -2.3212637641  1.4006222422  1.2728335086
[7841]  0.4194331385  3.0983731081  1.7287586713 -4.1980718256 -1.1829433974
[7846]  2.6866394652  0.9519913007 -4.4418350005  1.8306014295  1.0318282543
[7851] -0.0237591952  0.0412606576 -0.4877322063 -2.2472363449  1.3403846944
[7856] -2.7101092105 -1.7875629400 -0.5318152099 -0.8245214220  1.1806480061
[7861] -1.4571619014 -0.1453705146 -0.4612489360  0.1415049708  2.4846985123
[7866]  1.2282527416 -3.6044204596  1.0907928880 -3.0623624819 -0.2618708651
[7871] -2.1928468038 -2.0854607423 -1.0726610837  0.3673866495  0.6395512883
[7876] -1.1120539357  2.3227854908 -0.5439042274 -4.3935952400  2.8926118360
[7881]  2.0771465135 -1.4903975623 -0.3838159080 -1.7218361462 -1.2249333244
[7886]  0.0465207559  0.6419916694 -1.2251881539 -3.4459835465  3.1384380744
[7891]  2.1004683863 -1.9836157625  3.2805888761 -4.3890364890  2.2139053573
[7896]  0.1912241986  1.1769084225  0.4159502155  2.5125285373 -0.8323159370
[7901] -3.7112712110 -0.7591356464  1.9717345674 -2.9463245600 -5.0449026021
[7906] -0.8275509651 -0.7905852383 -0.7310152924  1.5843870466 -3.6602733621
[7911] -2.0511366244  1.1806745563  2.9097554149 -1.8192081085 -0.2018058502
[7916]  1.7027987798 -2.5661933596  0.8960957670 -0.0630168583  4.0089580225
[7921] -0.3020593951 -2.4966158318  1.8790550745 -2.1796800043 -0.3640401930
[7926] -0.1670550037 -0.4376626488 -1.9594760288 -0.1710894579  0.9194546229
[7931] -1.3370645558  1.0565367691  1.6687158658  2.4058854589  2.0713404878
[7936]  4.6626419544 -0.4109391554 -4.7040547531 -1.7255656293  0.3151213949
[7941]  0.0182679803 -0.7076967863  2.5608846875 -2.2229219325  0.0612311745
[7946]  0.4220380717 -2.7376273398  1.4389765727  2.0502688811 -1.3984491405
[7951]  1.9638661080 -4.3872216339  3.4508336278  0.3941851040  6.0698932750
[7956] -0.1304250324 -4.9582826810 -3.3515768831 -0.9375290334 -0.2143547855
[7961]  3.4336252455 -2.4247807829 -1.6450003771 -2.0801570857  0.8479029812
[7966] -0.0916367062 -0.7316872920 -1.8977348489  2.6125406639 -1.6514613312
[7971]  0.3143017066 -1.5034937799 -2.7031944652  0.1440092137 -2.6892678314
[7976]  0.3592778055  0.7716793906 -0.3548018781 -2.1249895166  1.5185804751
[7981]  0.4023775493 -1.3416608183  0.7477393115 -2.1391447014 -2.8340629595
[7986] -1.4852854591 -0.5922441037  0.0443632218  0.1953892402 -2.1487768454
[7991] -4.0961335478  0.1350367615  1.1663320928  1.5921363418  1.9258878262
[7996] -1.8896011605 -0.4957260933  0.5683508872  0.2334670030 -1.4390609709
[8001] -2.4493383717  2.2762427144  2.3948335461  2.5790204093  3.5117650874
[8006] -1.3672759270 -2.1956083397 -0.4188308518  1.4945413340  1.1211845071
[8011] -1.2604771979 -3.2123961851 -4.7735894795 -0.8034061834  2.7066706384
[8016]  0.1048972428 -3.2193101343 -0.1951996423 -0.9842068685  0.1010912432
[8021] -0.7265322277 -2.6076514853  2.1512522524  3.5907736659 -0.9031367569
[8026] -0.5364945971 -0.9968271008  0.1258965887  0.3397317126  0.7799662827
[8031] -3.4390311650  1.4571738820  3.4040149480 -2.7437272102  0.6969449713
[8036] -0.6882784631  0.0676595840 -1.7802315847  0.3768898349  0.8534969655
[8041]  1.4530793527  0.2026506166  1.5078196246 -1.6641219140  1.2983359444
```

```
[8046]  1.3098217729 -2.9370765849 -0.6327238972 -0.2391041302 -3.5614628960
[8051] -3.4994178859  2.0737995098 -2.8940681502  2.5092196903  0.4639216245
[8056]  1.9962180383 -2.3143559038 -0.9115143063  1.2967131732 -0.0559500898
[8061]  3.3064905350 -2.9995288936 -0.2789465597 -3.2564306497 -0.2300373845
[8066] -0.6206690631 -0.1264713149  1.5237863759  0.4940829431  0.8945584042
[8071] -1.2541970893  2.3642710958  2.3104303259  1.2225544104  3.0664237457
[8076] -1.2301396125 -1.3440970854  0.9183687492  0.3429146489  4.3251228248
[8081] -1.4059044074  0.2289586196 -0.3973116497  2.6209389335  1.0739203327
[8086] -0.6193777136  0.6872134855  1.3494417708  0.2706271992 -0.1760667558
[8091] -3.1302518809  2.1012107562 -0.3845652787 -0.5532040075 -0.8680642748
[8096] -1.7271205566  0.0454687486 -0.9547077219 -0.7699754985 -0.6623865721
[8101] -0.6933736342  3.8534512960 -2.6159859668 -1.5197222677  2.8990837284
[8106]  3.7605329391 -0.4202413916 -0.8434366275  2.7371988971  1.0074798475
[8111] -0.7528690895 -0.6717071032  0.8765061977 -0.9005715717  0.8238130187
[8116] -0.7912234111 -0.5533500772  4.4613269931  1.2497037574  0.4267421107
[8121]  2.4069497694 -0.7175817019 -3.4476279497  1.0915235483  0.9206015369
[8126]  1.2190726928  2.6236580263 -1.4079575037 -0.9022324618  0.6402197341
[8131] -0.3194495349  1.6164147366  2.1603106946 -2.6063223047  1.4713181639
[8136]  1.2379292369 -0.5235376746 -0.5518500905 -3.3205557343  2.3637136271
[8141]  3.9660651575 -3.6358672809  1.2096163257 -1.7705145579 -0.2128213757
[8146]  1.3714993491 -3.3675791254  2.0745492007 -1.3716050241 -1.3197659386
[8151] -3.5258067902  0.4988152183  0.3602020114  2.2587402730 -0.5395528547
[8156]  0.7206110081  1.3351488535  1.0502955368  0.4736590560 -2.7636658262
[8161]  5.8113596143 -0.9145341453  1.9090619175  0.4215691071  0.8973444077
[8166] -2.5048787242  0.6256219106  1.5017468541  0.9130222048  3.0147145783
[8171] -0.1764384270  2.5650932688  1.2167141710  2.3129181788 -0.9331742250
[8176]  3.3352730683  0.1434218216  3.2084090748  0.2496562865 -0.0809508334
[8181] -0.5879957222  1.8199574685 -0.3185856250 -0.1781929423  0.0573093626
[8186] -0.1181009559  2.0073221950 -2.4140057767  0.4347491486  2.4320528348
[8191]  0.6804205649  0.0224229515 -1.5081865162  0.0128429239 -1.7126604977
[8196] -0.7417707552 -1.4279453153  1.5492273106 -4.7190691328 -1.1635394240
[8201] -3.1332944143 -0.7302550515  3.0414017194  2.3971033769  0.6680979302
[8206]  0.1651143385 -2.6323513751  2.0876469489 -2.0195043301 -1.0333904987
[8211]  1.4128005378  0.6226120271  5.3374560905  6.3467894807 -1.8505910012
[8216] -2.8695477741  2.1796853536 -0.4979534114 -0.7078722174 -1.4049523764
[8221] -1.6814346069  1.8617863786 -0.9717217914  0.0332089894  0.4329842815
[8226]  3.5075089608 -1.4580732955  1.3328866507  1.4550714399 -2.2494653820
[8231] -0.1410757934 -0.7599243184  2.8799447653 -1.1162845753  3.2266081366
[8236]  3.1781312525 -1.6813704960 -4.2722835151 -3.0293756800  1.9885013197
[8241]  0.3295973037 -0.8930513805  0.5586822052  1.5222694636 -0.2002289008
[8246]  1.9498820472  1.0713740741 -1.7304056090  1.5714452535  3.6197720726
[8251] -1.3997212978 -4.4765014047  0.4990494941 -0.1792267010  1.6791843653
[8256] -0.7918380222  0.5371632102  4.6698303790 -4.5357048655  0.8374121100
```

```
[8261]  2.6080613887  1.2764110205 -0.8434696001 -1.9278332740  2.1138984279
[8266]  0.7179855854  0.4238540234 -1.8340363444  0.7516330069  6.4480520521
[8271]  1.3958424564  4.4110602756 -1.8714692218 -2.0090367547 -0.5318599494
[8276] -2.4203609828  3.1901804237  0.3218716259  2.6385210057 -0.3625406721
[8281]  1.6123782818  3.3551355618  5.9057776061  4.2025701657 -2.0124889870
[8286]  0.7062342742 -4.3158373887 -1.6191514644  0.8156319224 -0.6169260069
[8291]  2.1283292789 -3.1322670579  4.6481967751 -2.3942425514  3.2844566675
[8296] -0.6101907806  0.1281754243 -0.2416076607 -3.2215060176  0.7590590992
[8301]  2.9132276074  1.1824978781  0.7077602386  1.0891632820  0.4612954688
[8306]  0.0354705491 -0.7529244825  2.7669874428  1.9732204195  2.9076263948
[8311]  1.1126872290 -1.7570482007  1.4570095784  3.2046505453  2.7131915128
[8316] -0.0174169947  2.0324060686  0.8867139148 -2.0509559577  0.9592193981
[8321] -0.9424965407  3.6423676382 -1.7298281158  0.7248385117  3.8740892278
[8326] -0.4098433262  1.6595373876  3.1837431980  1.5002955561  1.2118139913
[8331]  0.6788328018 -0.7158751966 -2.3140951122  2.4800006888 -2.0845534720
[8336] -0.1056156943 -1.0693355836  0.5794859561 -3.3748984344 -0.4201234229
[8341]  0.6956671693 -2.5286651230  1.6235551716  1.4045951184 -0.6905745880
[8346] -2.2310222541  0.1992373891  1.7966013882  0.1099789846  1.3494890805
[8351] -0.9375983092  0.8280811903 -2.9116626172 -2.1327310117 -2.1873509421
[8356] -2.3522733858  0.0611408561 -1.1975438840 -0.2376551618 -3.3869409023
[8361]  2.4482228877  1.6649785065  0.1530189145  0.2281306416  2.3493827165
[8366]  2.5242241433 -1.2527421274 -1.0344522802 -0.7307066172  1.7989754498
[8371]  1.0744880569 -2.8972400476 -2.8380371857 -1.3949955320 -0.1735064166
[8376] -1.7323589774  1.7547420752  4.3531172063 -0.6704903655  2.9134500633
[8381]  0.4937985745 -0.9264186582 -2.3096459118 -1.0644088889 -1.4708579330
[8386]  2.0338669977 -1.8937045614  0.6624021746  5.6122661575 -0.6796947015
[8391] -0.3530616639  2.5906256579 -3.0147013681  2.5385119002  1.7251748606
[8396]  1.8971652746  0.9473040985 -3.2755003353 -1.7625845031  2.9272022083
[8401] -3.1951849597  2.8562465425 -2.8947549091 -4.1777992946 -1.1882497658
[8406] -0.4614760392  5.3586557424  1.0067728943  3.1165430106  2.9593049629
[8411]  1.5293409416 -2.1385981994  0.5047154444  0.6117477924  0.9068370871
[8416]  0.1176006833 -0.7460538578 -0.6634479503 -1.1585145314 -1.7901928932
[8421]  0.9066230784 -0.2847181031 -1.3270749800  0.7124894142  0.5184596595
[8426] -1.1310350895 -4.3506215569  2.9329101587  1.4341228222 -0.0807962304
[8431]  3.2902414978 -0.0312869906 -1.1071711678 -5.4689601220  2.3526160348
[8436]  1.3884385241  1.7980660493  0.7934139888  0.7512481798 -2.0956400037
[8441] -1.5810623319  3.5181910775  0.1970359300 -0.8775268194 -0.7999075165
[8446] -2.5110813346 -1.7487378400  0.3921049030 -2.9794002528  0.0738520518
[8451] -2.7857338823  1.9828266296  0.4898205808 -1.2487520157 -0.4663046224
[8456] -0.1351533279 -1.1346561009  1.6655742944 -0.5168322378  1.1209290484
[8461]  0.7398423594  1.5514546760 -0.0607051012  0.0572141111  0.2587070100
[8466]  4.9862942817 -3.0693063655 -0.0893616289  1.0222907698  1.0472439149
[8471] -2.4317971754 -1.3365668783 -0.6944251851  2.8144033567  2.2035978551
```

```
[8476]  1.3537947105  4.5192788033  1.1681497077  1.7814358589 -3.2619341269
[8481] -0.9003951599  0.0733638647 -1.1143589124 -1.0949573053  2.5330223898
[8486] -0.0324743366 -3.0312116523  4.3170762613 -0.3423241859  2.0606433938
[8491]  0.0477099394 -0.2588933943 -2.4120042731 -4.8720773500  1.0685758335
[8496] -4.3169017256  0.2896636913 -2.9130062093  2.2471702600  4.2002162034
[8501] -0.3709518847 -0.3792975683  2.0498998636  1.4130644727 -2.9314730284
[8506]  1.0143684258 -1.2895087609 -2.6826986923 -2.4094476274 -2.0342979138
[8511]  2.0092754560 -2.4144254509 -1.5214386502  1.0774959540  2.1512642512
[8516] -0.3653261126  4.0903472987 -2.2625916742 -1.1286357874  0.0639250107
[8521]  0.4728834382 -0.4035447772  4.9477633067  4.3276476397  1.1910622574
[8526] -0.6455408572  0.1238018133 -0.0979516977 -0.0579417581 -0.3675070741
[8531]  0.4632848587 -0.9184176377  0.7995498059 -0.5387082857  0.3555516834
[8536] -2.5163907196 -0.7812131833  0.6785165453 -1.0266432528  0.3214652730
[8541] -0.7246223465  1.9956128469 -0.4254715956  0.1202072056 -1.1772982865
[8546]  1.2009102353 -0.6484050451 -1.6639194451 -1.1555250790 -3.0322904363
[8551]  3.7032739118  1.1741626986  0.2242717624 -0.6207638926 -1.1334081498
[8556]  0.0995559193 -1.6669011816  2.2912229819 -1.6268953415 -1.1014615750
[8561] -2.4699421815  3.1651685497 -1.0535741150  1.1648442451 -1.5386358241
[8566] -0.9880462075  2.5503728204 -0.2169725137 -1.4899807878  0.2095150432
[8571]  0.4756097252  0.5733069379 -0.8835524355  1.8891326350  0.4886598568
[8576] -1.3677223365 -3.5982189757 -2.6950260117  1.1055130028 -1.5025351113
[8581]  0.9745805427 -0.2020500487  3.5361377191 -1.6387791352 -1.1596062455
[8586] -2.1787995530  0.9726955919 -1.2608880969 -2.6190359718  0.7863996845
[8591] -0.9451474627  1.5965720970 -1.0545287449  0.6938870169  0.9367755257
[8596]  1.5661592444  1.4971883727  2.3044545134  2.2776699214 -0.6746239972
[8601]  2.1020598333  1.1552725955 -3.8164664251  1.5824569950 -0.2024838481
[8606]  1.4810548070 -2.0223619404  4.2724164943 -0.0392305392 -1.0898050283
[8611]  2.0474139933 -1.9628511370  2.1635492417 -1.4017696097  4.5709007887
[8616]  0.0922646462 -2.8413396969  0.8551405302 -0.0481850782  4.1403730681
[8621] -3.7880468919  0.4089557547 -0.5546713844 -1.1867602461  0.8434334166
[8626] -2.2174597972  0.9405157012  1.9581231843  1.7124685330  1.0493700293
[8631] -0.4728458176  1.5427697739 -2.0008313084 -2.0148336177  2.6126401703
[8636] -2.2409061944  1.7831091931 -0.4249024385  4.9549863724  1.2709950005
[8641]  5.6261435091  2.7857707077 -3.4704405304  1.6489088737  0.5780334851
[8646]  0.4855297615  3.5892170597 -2.5275845317 -1.4234180456 -1.7930432257
[8651]  0.6684479395 -0.5790026686 -1.6767705355  3.2126760243 -0.6091695592
[8656]  0.2588670524  0.2194064714  1.4784141184  0.4305237155  0.9024843800
[8661] -0.4211759905 -3.0739884241 -0.1011192049  1.8767625915  1.0966154887
[8666] -0.3025943217 -1.2721325178 -1.7442472535 -1.5436326488  2.3533291892
[8671]  0.8004099480  1.1235358258 -2.3523744778 -0.9865860478 -1.6598104380
[8676]  2.0864789767  2.9844721668  4.6874166697  1.3978665457  1.5058690057
[8681]  1.5378629165  4.5887623246 -0.6646852418  3.9034432700  0.0608829762
[8686] -2.7855595741  0.0179106329  3.7546679716 -1.1689531610 -0.2179140604
```

```
[8691]  6.4804150919 -0.0606637739  0.6289928793  1.3419667790  1.1742016203
[8696]  2.1164481221  0.8109142973  1.0362133571 -0.8820576085  0.0441965154
[8701]  3.4662102503  1.4371318274 -4.2958562586 -1.1288773025 -0.4133176938
[8706] -1.1022379412  0.4149182690 -0.9020886325 -0.4093185705 -1.9810543244
[8711] -1.7884220622 -0.6886135814  1.3668529459  1.0152345358  2.1123974024
[8716]  2.3982162835  0.5280547026  0.9547951727  1.1384619623  3.1227617897
[8721] -0.3654262788 -2.6902785384  2.4072893954 -0.2493666154  0.0254196880
[8726] -3.0848171384 -2.3067937174 -0.1974123499 -2.5395025735  4.5944071999
[8731]  3.2450109185  1.0863751370 -3.1717599469  0.3189190193 -2.2108936127
[8736]  0.3761443324  0.9121643904  0.0048678311 -0.8329870922 -0.1214961835
[8741] -2.0599800159 -0.0441910825 -4.1623434267  4.1011517348  0.9691619595
[8746] -1.2503135175  0.8134065415  3.1897556458  1.6801872199 -0.5522602217
[8751]  1.0769571629 -1.2122167138  0.3220115759 -2.8139102800 -2.5930534197
[8756] -0.6577909189  1.8789218130  1.0745679000 -2.3849028807  0.6326974913
[8761]  2.8256694441  1.7969113977  2.9979896629 -1.5741805435  1.0610680569
[8766] -3.1028906635 -1.7263139328  1.8790915105  3.8486059243 -0.1396385570
[8771] -0.2621870030 -0.9343572847  3.3223442421 -2.8737205124  3.9527478904
[8776] -1.0615011482 -2.4263517113  0.0070826671 -0.1354134337 -1.2880245319
[8781]  2.2141094160  1.9411023418  0.9486628440  0.6169022990  3.5032403229
[8786]  0.2703861153  4.7056501846 -1.3499088032 -1.4013492195  0.0811853325
[8791] -0.4037985478  2.5980474820  1.1867597822  0.9053059458  1.0876855025
[8796] -1.4651948461  1.6325592842 -1.4445467965 -3.3553115809  0.5036099113
[8801] -3.3033019528 -0.6451936842 -0.1575547750 -0.6421784195 -3.8701326958
[8806] -0.2368137165  2.2262744829 -1.4343579322  1.9292838915 -1.8154838468
[8811]  1.1919004159  0.7567413308  2.1442565603  3.0509497315 -0.7108634088
[8816]  2.9121530013  2.7138207869 -1.5375320904  1.2110623740  4.4915596576
[8821] -2.4944163718 -5.4183145501 -3.7699598256 -3.0609952209 -0.9255491411
[8826]  2.6998513799 -0.3562405762 -0.5919286913  0.1002873610 -2.3340022700
[8831]  3.8687829711 -1.4322249692  0.4270971544 -2.8275477950 -2.3892246308
[8836]  2.9188471189 -0.9853611305  2.3917744771 -2.1693464138 -2.6204620951
[8841] -1.3007832324  0.5734755060  2.3546975502  2.2240613933 -3.0255419143
[8846] -0.0918979423 -1.1843943812  2.3272819538 -1.0084640438 -3.8790700481
[8851] -2.2642840316 -1.6155758536 -0.4350072358  4.4787125973 -1.2745601940
[8856]  1.6338754055 -3.0096476415 -3.2267157063  0.3643405312 -2.7767396031
[8861]  0.5226254623  0.4185942244  1.8896872286 -0.3923907953  3.8521977435
[8866] -2.2313053516  1.4265234413 -0.5737522536 -2.1879638712 -0.8859085729
[8871]  3.2120266056  2.0929168142  1.7437391567 -0.6582295825 -2.3135298547
[8876]  3.6400208812  0.0949242409  2.4196682612 -0.0317217631 -0.3919138376
[8881]  1.3254819775 -3.5442310649 -3.2497275241  0.7308328357 -0.9045235837
[8886] -1.0887652113 -3.0318685473  0.6439704746 -0.8488357371 -0.7871099315
[8891]  1.2490018351 -2.9750540496 -0.8683073238 -2.2672659217  2.1177943859
[8896] -1.7802063906  0.6761030065  0.1099921431  0.6388259123 -0.2867725704
[8901]  0.2201107596  2.2597653533  0.0814441618  1.8361645388  1.7159570084
```

```
[8906]  3.0585875386  1.5914061245  2.2698469504 -3.3688580690 -4.6050797538
[8911]  0.3026093754 -1.3840040817  1.8749392644  5.4671680567 -1.5763125360
[8916]  0.6118000033 -0.7942777078  4.5048321657 -1.6359758911  2.0889506084
[8921]  0.3662886105  1.1513733863  0.1757181158  4.0063644794 -0.6670632091
[8926]  1.6173676001 -1.4550798189  1.0966277196  1.1101912962  4.1398023811
[8931] -2.7875404172 -0.1041427612 -0.3782445934 -0.3052597217 -0.7543209201
[8936] -0.4203149486 -0.3440685984  2.0732802600 -1.2154312673  1.7835556835
[8941] -1.7585303052 -1.5878467391 -0.3800744805  1.0325933150  1.2925063595
[8946]  0.8526351773 -0.5930463036  2.0661182969  1.9490132612 -2.1914202809
[8951]  3.5453605307  1.4177517403 -0.2907213381  2.7575482228  1.7940062144
[8956]  1.2677587384  0.4745850906 -1.4982609249  1.5266097043 -2.5371818980
[8961]  0.0875072871 -0.2154043047  4.5888327401 -2.7443118358  3.0727220217
[8966] -5.5872608871 -0.2463605659  1.2123549844  1.8442524458  1.1742436021
[8971] -0.0273932109 -1.1163065676 -0.2935697981  2.1004711294 -0.0324911377
[8976] -0.2816471724 -1.7252728849 -1.6062830841 -0.0848357859 -0.6547405055
[8981] -3.0658632872 -4.4704103258  1.7804103009  4.8945234539  3.6145410022
[8986] -1.0232220717  0.0241537166 -0.4279799863 -0.8444943523 -0.1528834074
[8991] -0.8451120130  1.6684538072  2.9597787391  0.2165725747  0.0575249564
[8996] -3.0989135030  3.2110954537  1.2590797989 -0.0347530570  2.9464670197
[9001]  0.1531655754  0.6581138441  0.6805663462  2.7079544172 -0.4163131229
[9006] -0.4429680660  2.6642919288 -2.0852710879  2.0309544303  1.5400536022
[9011]  1.4148383676 -0.9429141407 -0.3985744051  4.5997885534  1.2051023586
[9016]  2.0656668124 -1.4771108954  1.8290843937  1.6051588105 -1.3978350756
[9021] -2.1021295332 -3.1910660897 -3.7444151939  1.1605815407 -0.7445727374
[9026]  0.7022932023 -1.4697496650  1.0839923403  2.6101405208  0.0860558054
[9031] -0.7241728860 -2.2591491733 -0.7378369454  1.3120031555  3.1981524905
[9036]  2.9736761701  2.3165865679 -1.5899741573 -1.2824141206 -1.9775802501
[9041]  1.5692579198  0.3927752524  6.0976250508 -1.3953803607  1.6301254255
[9046]  2.0991932273  4.1162654419  0.5999379084  0.4683659114  1.0548539577
[9051]  4.5506916552 -1.5732471154 -0.2388403479  3.3938744773  1.6638631745
[9056]  1.4103435074 -1.1789856117  1.0373719476 -1.6869744840 -0.5066794665
[9061]  0.5940827769 -2.9276323976 -0.0061139215  0.3067513240 -0.6767136491
[9066]  2.4995464098 -1.1744389688  0.1572175829 -1.7462522667 -3.0230488194
[9071] -1.4507255139 -3.3553342967 -0.0627350100  2.2672562585  3.8485521230
[9076] -2.1276010498  3.1744361594  0.8817512830  2.4940683931  3.4243152438
[9081]  0.3031824397  0.9692121538  2.6931964103 -0.9780089546  2.0574109622
[9086] -1.3950048601  0.1433646880 -3.4082482338 -0.6371483323  1.7079734632
[9091]  0.2850918679  1.7678212169  0.9242216296  0.1575116211 -0.4131672389
[9096]  1.8857352653  0.1349658283 -0.4529215188  0.4888934786 -1.2232014089
[9101] -1.4073892046  5.7829858011 -0.0583646364  0.5453445786 -0.8416604273
[9106] -2.0784470386 -3.3343043609 -1.6501283446 -2.7747476202  0.3094769874
[9111] -1.0944599460 -4.3237159433 -1.6449274602  0.6903423069  2.1810560423
[9116]  0.8786450004 -2.3295896336 -2.2745872811  2.2608507620  1.3678923026
```

```
[9121] -2.3496719356 -0.9121523006  0.4574451483  4.6040969402  2.6195009115
[9126] -0.9010334275  1.5335333319  0.1808653518  2.9091757313 -0.8328386469
[9131] -0.8455907581  3.1786795126 -2.1249270470  2.5308623019 -1.5522921107
[9136]  0.8550397092  1.1668558296 -1.6971062728  0.7435027771 -2.6858672432
[9141]  1.6071954724  1.0318680451 -1.3156678832  0.7709623753 -1.1445022295
[9146]  2.6794730437 -0.1364790834 -3.1542345567 -0.3056318795  0.2839405598
[9151] -2.2465140158 -2.8459786930  2.3365040910  0.3264109250 -2.4803074432
[9156]  3.3477332222 -0.3914716374 -0.6217705958  0.7295349727  2.3126452284
[9161] -1.3958474786  1.8300178344  2.0553185642  0.5638434496 -1.4953339370
[9166]  1.4382395460  3.1327909553 -0.4179112226  2.2056385812  1.2214715916
[9171]  0.6037977644  4.1570754419  1.2681371932  2.9870258860 -1.6122289600
[9176]  3.6351415786 -0.3975163973 -1.8474577800  2.7692516408 -2.1885611081
[9181] -0.0413386704 -0.6193607438 -1.1853536787 -3.0373441598 -1.3065509029
[9186] -0.8382362111  1.3650604469 -1.5005218504  0.7259899086 -1.7257995505
[9191]  0.7112856010 -0.3394950885  0.0613482370  0.3022545299  2.5304624036
[9196]  3.3603279611  2.7567839265  1.7966113043  1.6679964528  2.3575719106
[9201] -1.5389338166 -3.7542429897  3.5929592777 -2.8211560123 -1.9505176763
[9206] -3.2113484583 -0.0083844595 -0.1838412209 -1.9626000948 -0.7129823255
[9211] -1.0944957952  4.5110287649  0.0636188507  0.6651468226 -1.8555749893
[9216] -1.6687622964 -1.3196868612 -1.8863856224  1.7950199343 -2.0486634504
[9221]  3.7341641234  1.9688993742  2.0105368447 -2.8146747729 -2.5809958403
[9226] -2.3303704209 -2.0562767590 -5.0339615239 -1.6910680694 -0.0202142067
[9231] -0.9087803320  0.9278700302  1.4185105925 -0.0595718343  0.1025875327
[9236]  0.7241089240 -0.4031143713  2.2548532146  1.7418751810  1.2732880511
[9241] -1.4916233071  0.2403680753 -2.7554465100  2.8035649281  1.8999640941
[9246]  2.4471745439 -2.7461011837 -2.3369960910  2.8336567959  1.0958840840
[9251]  2.7239954427  0.1704434851 -1.2913111172 -4.6103306104  1.1470686801
[9256] -0.1130296461  1.8816620708  2.6160208678 -1.9283933673  1.3909085028
[9261] -4.2697607151  0.9864747899  2.6480956900 -1.3043342735 -2.6757849309
[9266] -2.4485544951  1.2664570144  0.8352160380  2.3840456783  1.4161233516
[9271]  1.4393363393  1.4025425196 -0.4300528038 -2.8820671213 -0.2254640629
[9276] -1.4690288787 -2.5906518401  2.6342477955  1.9701536133 -0.8260146382
[9281] -3.1535157663  0.3070910708  1.2361701716  1.5622068843 -1.2524531064
[9286] -2.9382396296  1.1478012013  2.6252765602  0.3182990461  1.9726010859
[9291]  0.1290574450  0.0310695468  0.1707565640 -1.9984201138  2.2250065756
[9296]  1.9585146511 -1.2625426278 -1.1755204527 -1.8529975959  1.0908109137
[9301] -1.1625357266 -3.6968608706 -2.9578961130 -0.2583557479  1.7634432869
[9306]  2.8560310302 -0.9992466198 -1.7114475164  2.0566639914  3.1113666099
[9311] -0.3612406501 -2.9056256990  0.0735242997  0.3520656501 -2.3011769021
[9316]  3.8560163250 -2.4042590801  2.4927950864  0.2486500001 -1.7800860581
[9321]  2.1502326232 -3.7558053728  0.4164038744  0.1864713024  1.5720204194
[9326]  0.1734213597 -1.2675940773 -1.9326901718 -1.1762693862  1.4676672010
[9331]  2.0963767142  1.1344889435  2.9811151188 -1.8854064580  1.0755831463
```

```
[9336]  2.9016383325  0.2730011217 -2.8839496881  2.1695572618  0.9822160585
[9341] -3.0106854678  1.1964487969  0.8291843441  2.0590769188 -1.6214900587
[9346]  1.0345126596  0.1795782104 -3.6772259891  0.2357302431  2.1189246369
[9351]  1.7789281640  0.4942917876 -2.7610924142 -1.1117397334  1.0515886723
[9356]  2.2658031707 -1.3810580830 -0.3671718191 -2.2024594006 -2.3744381456
[9361]  4.3053589845  1.7143816268 -2.1651829121  0.3762590204 -6.3822648050
[9366] -2.1291932935  2.1892362262  1.4365251353 -2.8424205767 -0.3016513347
[9371] -0.8582970910  0.5393123094  0.9331965990  1.2018161918 -0.8921196727
[9376]  0.3939198412 -1.9735506678 -1.6518697098 -1.9254174553  3.4155041712
[9381]  2.5089494116 -3.0449214905 -3.0985460189  0.0005607364  2.5349054002
[9386] -0.1631994185  2.8668113315 -0.7828786802  4.5969802433  2.5264840860
[9391]  0.2061810215  1.6271032885 -2.7871794007 -0.5296929202  1.9362114909
[9396]  0.8518352703 -0.9606426373  3.7565392530 -3.3403818130  0.7276103305
[9401]  1.1881135106 -1.2705116278 -2.3896178994  2.1058695222  0.8374550080
[9406] -3.3589504273 -0.5674580875 -3.1339563131 -1.9266290960  0.8334379017
[9411] -0.8409069421  1.5205408962 -1.3894022326  0.6749201510 -0.3484328178
[9416]  2.7514063846  0.4993118196  1.3901840620 -0.9395375057 -3.5518577228
[9421] -1.8147802940 -1.0963462257  0.2756630978 -2.0602394705 -2.7010310747
[9426]  4.1016151398  2.2362067390 -0.8634224578  0.3471858468  0.5937591445
[9431]  2.0664022423 -0.7511864139  0.9533785810 -0.5382482477  1.7764082148
[9436] -0.0135585238 -0.2261034848  1.2744876316  0.1227646816  1.3207752372
[9441] -1.3425287643 -0.4598365277  3.5717693220 -0.7618568325  2.1228021877
[9446]  0.6647127887 -4.0607319670 -2.3404206800  0.8340893882  0.8925015840
[9451] -3.4847167322  3.0060228358 -0.0691117034 -0.0387179468 -0.0265044407
[9456]  0.7819230660  0.6160491430  2.4099931157 -1.4965123786 -1.4302746092
[9461]  3.2613933697 -1.2836303758  3.3532047220  1.6377950267  0.7667145109
[9466] -3.1198475584  3.6194951818  2.7947779886 -1.1941062618  0.1519425733
[9471]  0.2948086154 -1.5217940924 -0.6072226319 -2.9176425384 -1.8153679433
[9476] -1.9590151224 -0.9310957435 -1.2429073314 -2.6507131720  0.3456888970
[9481] -1.1733576639  3.3444181020 -1.3809110667 -0.4234333149  3.4512905541
[9486] -1.4028554787 -0.0148563691 -0.6979020178 -2.8433816647 -3.0641830278
[9491] -2.8806143824 -1.4773129574 -3.1317264096  0.6911501397  1.6331904854
[9496] -2.2115103233 -3.8619587240  1.6335454044  1.5492313827  0.4275641441
[9501]  2.0150401576  0.2020929599 -1.6969333627  0.0307067157  0.6886133621
[9506] -0.2927796607 -0.7472904286 -0.1711201076  0.8861716823  1.5468192846
[9511]  0.4904475875  0.3538165661  0.1401697560  0.9846937934 -0.9052103188
[9516]  1.6253526534 -2.5270810430  1.7282255313  1.3711359947  2.8506507340
[9521] -1.2976630063 -2.2276769149 -0.8603432749  2.5829841312  4.5530474025
[9526]  2.8750210560 -0.2148655864  0.7746303598  0.5062192192 -2.7190743625
[9531] -0.9650625074  0.5182797187 -0.7822224711 -2.3276069966  0.6263683645
[9536]  1.7090943166  2.5376002201  0.3254893191  2.0629498152  0.6892650016
[9541]  1.9666300079 -0.9900208494 -2.5810443860  0.2674967337 -0.8506916074
[9546]  1.3138117564  0.3370715002 -0.3673553787 -0.0535911741  4.1849838227
```

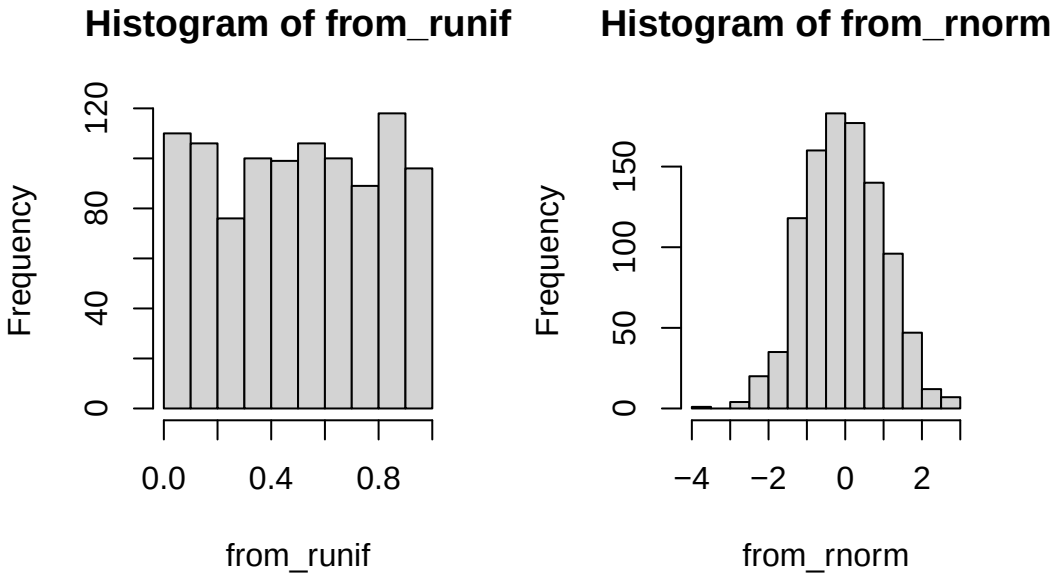
```
[9551]  0.0200492343  2.2176943523 -0.1168059317 -4.4419119078  1.0270741075
[9556]  0.3527060265 -2.8595046499 -1.6509230517 -1.5246597120  2.8118871424
[9561] -1.2625826019 -1.6196437787 -1.9322479464  1.9680973460  0.1634681284
[9566]  2.8677507110 -1.1181510707 -2.0070204605 -3.9341346889  4.3555921913
[9571] -1.2732921829  0.4339457845 -0.5619844711 -0.7921717602 -0.8386673020
[9576] -4.1811197691  0.0706243954  0.1279327198  4.9982702895 -1.3420281239
[9581] -1.4470953253 -1.7184308625  1.3480464963 -1.5846106764 -1.5632012426
[9586]  2.9410506298 -3.2300760126  0.9310891670  2.0256109741 -2.4038840687
[9591] -1.1836747203  0.2343478781 -3.0679037751  0.0992845477  3.0427685018
[9596] -4.1191021637 -0.4343754345  1.9289489124  0.8714140556 -2.8864890813
[9601]  4.6806948182 -0.2750522760  0.3833968570 -1.1268468170 -4.2372196918
[9606] -0.7624237898  0.7883542828 -1.1733734337 -2.2504226959  2.2094839589
[9611]  0.5380034546  2.1751927152  1.5129563050 -2.0134485562  2.5005177521
[9616]  0.2407742796  0.3209537837  1.4235061065  0.0561232342 -3.7405988920
[9621]  1.9086394958  0.6272867358 -2.2129853583 -0.6746082546 -3.2653848378
[9626]  2.2383091326  2.5184158837 -3.0409103172  0.2527596448 -2.1860750002
[9631] -1.9344581239  0.7198334302 -3.4889205896  3.6997815540  0.5747290886
[9636]  0.6109016432 -1.6280885817 -3.7081520885 -0.9333876952  2.1432820558
[9641] -0.0802739141 -0.2289820215 -3.5500208864 -0.5075300311 -1.9015965978
[9646]  0.9150370820 -0.6676972054 -2.5128701685  0.7561772568  3.0178268867
[9651]  0.3359437001  1.3015599344  2.5401348183  1.0686872199 -2.6756812699
[9656]  1.3895768891 -0.7656496910 -1.6524888562  1.2282765785  1.7201884005
[9661] -3.6092905988 -1.5425525496 -2.2536596331 -0.6600898532 -2.7019611191
[9666] -1.6716830528 -3.1733494627  3.7518218911  0.1283885461 -3.1292385198
[9671]  4.3450737974 -2.9230023548  0.6580675531 -0.0603316163  0.0658323634
[9676] -0.7320235736 -3.0728671225 -0.0261383966 -0.1078505468 -0.7569664967
[9681]  3.0100439371 -3.8225905803 -2.5003299917  0.2422665235  0.0151549692
[9686]  1.4526977586 -0.9624347029  2.0139167360  0.1740966411 -0.3348848467
[9691]  0.8292715086 -1.2700243797  3.1544062743 -0.7241707565 -2.3556763557
[9696] -2.2765032834 -3.3759569062 -0.4881433069 -0.7390982624  1.1930003593
[9701]  3.7446617013  3.0161332775 -1.9107181798  0.1767920094 -2.0140052210
[9706]  0.7689206932  0.0877350535  1.4066755634  1.2199721446  3.2501312537
[9711] -1.0803209238  0.3180586841 -0.7806908246 -0.1227329256  4.2734878165
[9716]  1.6735729927  3.6961374422  0.4391619895 -0.6669714429 -0.2787107502
[9721]  0.3569035632 -4.8753834457  0.3621407562  2.2171531897 -1.6356289574
[9726]  3.3060409673  0.3729418258 -3.6397486334 -0.2097202252 -1.2674708788
[9731] -3.1954659942  3.2355040034 -0.8963251151 -2.4934266610 -1.6292181504
[9736]  2.3336514076 -3.5318287959  2.4285514264 -1.1329901518  0.2844185688
[9741] -2.1898750904 -5.7003705340  2.7762171498 -3.6711377726  0.1311126731
[9746] -1.1039638953 -2.6573217962 -2.8898647360  3.0635786682 -0.8283877163
[9751] -3.4294120674  0.2555695802 -2.1014693177  0.4403325583 -1.1381419640
[9756]  3.3689361089  1.7492195798  2.8084252383  1.4060734981 -0.1039442280
[9761] -2.4054242574  0.5334999675  0.5120929575  0.1799755443 -0.8860895309
```

```
[9766] -0.0600649847 -1.8320299103 -2.5168589848  0.5441996776 -0.1725766036
[9771] -1.7698262526  1.1860358981  1.4012966477  0.6827057152 -1.7597373569
[9776]  0.7496175413 -0.1228859316 -1.6158507806 -3.2205635894 -0.6285661257
[9781]  1.3364980861 -3.8997149464  1.8181893546  2.0215442298  0.9727127652
[9786] -1.9685595725  1.3701133217  1.8864190698 -1.4089173851  1.7490915432
[9791] -1.4349162762  0.0084539942 -0.0118417459  0.4523784586 -3.1933892090
[9796] -2.1268053857 -0.2860892995 -0.4102124810  3.9369641482  1.4256626117
[9801] -0.9651127194  0.1074959423  0.1219493421  3.4605598135  0.4239574066
[9806] -0.7769512704  0.2544709980  1.5108102048 -0.2066802135  2.6144200487
[9811] -0.6094386488 -0.2305848894  1.3998329973 -1.7387285113  0.1208598822
[9816]  0.6474559724 -0.0616777966 -1.3642389774 -2.7458505378 -1.4066575184
[9821] -0.6386243951  0.2723651923  3.6798723285  1.1455221598 -1.2610723105
[9826]  1.0903549812 -3.5111837520  0.0499453692 -2.3370710170  0.2947059765
[9831]  1.3890240883 -1.5692591824  0.3276833748  2.4487598264 -3.6208336353
[9836]  3.3919194039 -0.1708022074  0.1851125983 -1.3220560202  2.0296757131
[9841]  0.9650051755  4.5809069294  0.0712696990  1.2215220002  3.1575720242
[9846] -3.1893044388 -1.2710576912 -0.8039814334 -2.1290026586  0.5960778376
[9851]  2.5359733618 -3.4731061816  2.3180157021 -1.3177352594 -0.8242956581
[9856] -0.2368637429 -0.9295765620 -2.3389930603 -0.7397708025 -1.2638055781
[9861]  0.6820430586 -2.5476435034  5.1183828855  2.4820616077  0.4434960555
[9866]  1.6059127856 -0.7498049864 -0.6064579802  1.4778868976 -0.4991613536
[9871] -0.8134329512  4.6058339939  5.4910882432  2.2215969306  4.0408408501
[9876]  1.7384183818 -0.9363494664 -3.4470009089 -0.2420744235  3.4480782542
[9881]  0.2495936474  1.1793462139 -2.4967696031 -1.9052385192  0.0384041352
[9886] -0.9425122969 -1.7200194737 -1.7787881119 -1.8931118259 -2.2401666980
[9891]  2.4660391222 -1.9065061121  3.5908032888  0.0529290482 -2.8002865376
[9896] -2.8515422256 -1.3309143458 -1.9876821170  1.3227900118  1.8184460849
[9901] -1.2455597333  1.7325330852 -3.2058738926 -0.0634480714  0.0899417484
[9906] -1.6754982845  2.9654451313 -3.5757053715 -0.9681189367  4.7370647659
[9911]  0.1556071881 -2.2574122802  2.0760298982 -0.9925795657  0.6226720995
[9916] -1.4379996044  0.4608168618 -1.8178159039  0.1583872686 -1.6906851952
[9921] -2.2957063629 -0.5182102717  1.2702465729 -1.8940956139 -0.6595781684
[9926]  0.4379193371 -1.3664923353  1.5003718878 -1.1449250791 -0.5240957229
[9931]  0.6819618755 -2.3131986478 -4.4619202908 -3.7468157958  0.6021416247
[9936]  1.1291723773 -4.5901591687  0.6151788796 -0.5286955663  0.6790986901
[9941]  0.9672530511 -1.2548263398  0.5788773567 -1.4807897507  1.3103315751
[9946] -0.3616132077  4.6064394178 -0.1706106904 -4.6958830610 -2.0469361770
[9951] -0.6889262899 -0.2699656348  0.3923204088  0.0225440174  1.3181250003
[9956] -2.0240946603  0.7314287923 -2.5547907976  0.1304057146  0.4558111570
[9961] -1.5945506112  0.4447535376  2.4144325143 -1.0297743536  0.1708156053
[9966] -2.2511744805 -1.6573794864 -6.3399518698  4.3199203319  0.0269319184
[9971] -0.9875159077  2.3683814848 -0.5678027748  2.3646113851  0.8439232062
[9976] -0.7061029016  2.5685878454 -1.5246792417 -0.5056854466  3.9578937230
```

```
[9981]  2.9211122603 -1.0494571298  0.4165317412 -0.7522120408  3.5884623698
[9986]  0.0244445237  0.2523059882  2.2020794937 -2.3750373973 -3.3759064110
[9991]  1.0797368839  0.2777465538  0.8085019970 -0.5334991766 -1.5120096362
[9996] -1.7536537854 -2.3697719315 -2.7304196527 -2.7638246860  1.1894638147
```

To better visualize the difference between the output of `runif` and `rnorm`, let's generate lots of each and plot a histogram.

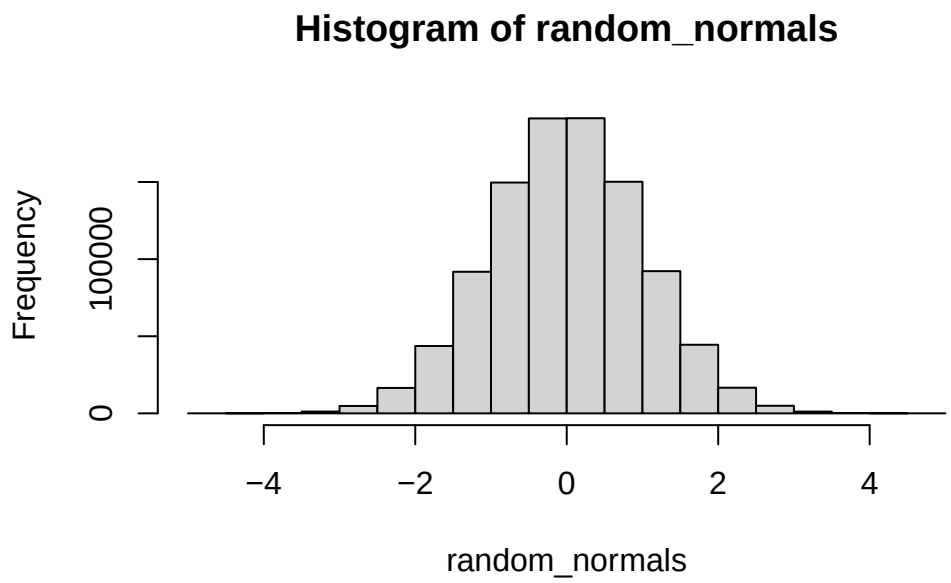
```
from_runif <- runif(n = 1000)
from_rnorm <- rnorm(n = 1000)

par(mfrow = c(1, 2)) ## base-R parameter for two plots at once
hist(from_runif)
hist(from_rnorm)
```



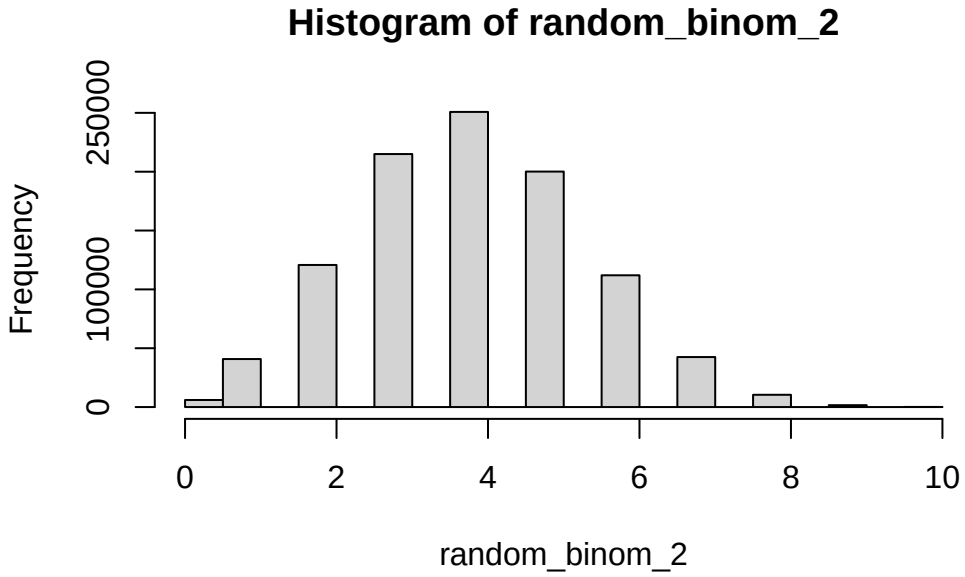
15.4 r, p, and d

Each distribution can do more than generate random numbers (the prefix `r`). We can compute the cumulative probability by the function `pbinom()`, `punif()`, and `pnorm()`. Also the density – the value of the PDF – by `dbinom()`, `dunif()` and `dnorm()`.

```
random_normals <- rnorm(n=1000000, mean=0, sd=1)
hist(random_normals)
```




```
random_binom_2 <- qbinom(runif(n=1000000), size=10, prob=.4)
hist(random_binom_2)
```



15.5 `set.seed()`

R doesn't have the ability to generate truly random numbers! Random numbers are actually very hard to generate. (Think: flipping a coin –> can be perfectly predicted if I know wind

speed, the angle the coin is flipped, etc.). Some people use random noise in the atmosphere or random behavior in quantum systems to generate "truly" (?) random numbers. Conversely, R uses deterministic algorithms which take as an input a "seed" and which then perform a series of operations to generate a sequence of random-seeming numbers (that is, numbers whose sequence is sufficiently hard to predict).

Let's think about this another way. Sampling is a stochastic process, so every time you run `sample()` or `runif()` you are bound to get a different output (because different random seeds are used). This is intentional in some cases but you might want to avoid it in others. For example, you might want to diagnose a coding discrepancy by setting the random number generator to give the same number each time. To do this, use the function `set.seed()`.

In the function goes any number. When you run a sample function in the same command as a preceding `set.seed()`, the sampling function will always give you the same sequence of numbers. In a sense, the sampler is no longer random (in the sense of unpredictable to use; remember: it never was "truly" random in the first place)

```
set.seed(02138)
runif(n = 10)
```

```
[1] 0.51236144 0.61530551 0.37451441 0.43541258 0.21166530 0.17812129
[7] 0.04420775 0.45567854 0.88718264 0.06970056
```

The random number generator should give you the exact same sequence of numbers if you precede the function by the same seed,

```
set.seed(02138)
runif(n = 10)
```

```
[1] 0.51236144 0.61530551 0.37451441 0.43541258 0.21166530 0.17812129
[7] 0.04420775 0.45567854 0.88718264 0.06970056
```

15.6 Calculating an expectation using Monte Carlo

We can use repeated, independent draws from a random process in order to approximate its mean (or variance, or higher-order moments). The intuition comes from the law of large numbers. As we obtain repeated independent and identically distributed realizations from a random variable and take their average, this average should get closer and closer to the true expected value. The longer we run the simulation, the better the approximation.

Crucially, we can do this for essentially any random process that we can repeatedly obtain independent draws from. All we need to do is be able to implement the process in code. This can often be easier than trying to calculate an expectation analytically.

Consider the problem of calculating the expectation of the maximum of two independent 20-sided die rolls. We could construct a function that took the maximum of two independent draws from a 1-20 vector. Doing this a large number of times and taking the average allows us to approximate the true expectation of this new random variable without having to do any math!

```
radvantage <- function(sides=20, k=2){
  return(max(sample.int(sides, k, replace = T)))
}

# Take the mean of 100,000 rolls with "advantage"
set.seed(60639)
mean(map_vec(1:1e5, radvantage))
```

```
[1] 33379
```

Next, consider the "birthday problem" or the probability that among k randomly chosen people, at least two people will have the same birthday. Recall that the expected value of an indicator random variable is equal to the probability that the indicator is equal to 1 (so we can calculate probabilities using Monte Carlo by getting draws from an indicator variable for the event of interest).

We could construct a function that simulates one hypothetical set of k people. Assume no leap years and uniformly distributed birthdays.

```
birthday <- function(k){
  # For each person, sample a birthday
  bdays <- map_vec(1:k, function(x) sample.int(365, 1))
  # If any birthdays are duplicated, the length of unique(x) will be shorter than the leng
  return(as.numeric(length(bdays) != length(unique(bdays))))
}

set.seed(60639)
# For a room of 23 people
mean(map_vec(1:1e4, function(x) birthday(23)))
```

```
[1] 0.5156
```

```
# For a room of 90 people
mean(map_vec(1:1e4, function(x) birthday(60)))
```

```
[1] 0.9953
```

Notably, it is still important that the expectation of the random variable **exists**. Consider the *Cauchy* distribution which has no finite mean. Suppose I take a large number of independent draws from this distribution and average them. I get wildly different results across different runs!

```
set.seed(60639)
mean(rcauchy(1e5)) # Average of 100,000 draws from a Cauchy(0, 1)
```

```
[1] 7.696414
```

```
mean(rcauchy(1e5)) # Another average of 100,000 draws from a Cauchy(0, 1) -- wildly differ
```

```
[1] 0.8749143
```

Exercise

Baccarat

Monte Carlo simulations are frequently used to analyze expected outcomes in casino games and calculate the "house edge" or the percent of a player's winnings that they can expect to lose on a particular bet. One casino game that is known to have a very **low** house edge is Baccarat. In this exercise, you will implement a function that simulates a game of Baccarat. Using this function, you will calculate the expected value for each of the three types of bets in the game. You will then simulate how these expected values might change if the casino set different payouts for some of the bets.

First, a brief primer on baccarat. Baccarat is a table game in which two hands of playing cards – the "player" hand and the "banker" hand – are drawn and then compared against one another. The hand containing the highest score wins the round. The most common version played in modern casinos is called "punto banco" baccarat and essentially plays itself according to a fixed set of card drawing rules. The names "player" and "banker" are simply labels for each of the two hands.

Before each round of play, bettors can place one of three bets: that the "player" hand will win, that the "banker" hand will win, and that the hands will "tie" and have the same score.

Then, the round begins. The game is played by drawing cards from a "shoe" of cards containing multiple decks of standard playing cards (typically six or eight). The shoe is shuffled to randomize the order. **Two** cards are dealt to the *player hand* and **two** cards are dealt to the *banker hand* We will ignore common casino practice of "burning" some cards from the top of the deck to discourage card counting as this does not impact the simulation.

A third card may be then dealt to the player hand depending on the value of that hand. A third card may then also be dealt to the banker hand depending also on the value of the banker hand and whether the player was given a third card. The rules for determining whether either hand receives a third card are below. After all cards are dealt, the value of the hands is calculated.

Face cards and 10s are worth zero points, aces are worth one point, and all other cards are worth their value (2-9). The value of a hand is the ones digit of the sum of the point values of the cards in that hand. So a hand consisting of a 2 and a 5 would be worth 7 points, while a hand consisting of a 4, 5 and 3 would be worth 2 points ($4 + 5 + 3 = 12, 12 \bmod 10 = 2$). In other words, hands are valued at their point sum modulo 10.

The hand with the highest value is declared the winner. If the hands are equally valued, the result is a "tie"

The rules for determining whether the player or banker hand receives a third card in a given round are somewhat complex:

First, if **either** the player or the banker has a hand valued at 8 or 9, then no third cards are drawn, the round ends, and a winner is declared based on the value of the initial two-card hands.

Second, the game decides whether to give the "player hand" a third card. If the player's initial hand value is 5 or less, then they are given a third card. If the player's initial hand value is 6 or 7, they "stand"

Third, the game decides whether to give the "banker hand" a third card. If the player did not receive a third card, the banker acts by the same rule as the player (draw if 5 or less, stand if 6 or 7). If the player did receive a third card, then the decision to draw depends on both the value of the banker's current hand and the value of the **third card drawn** by the player.

- If the banker's hand is valued at 2 or less, they always draw a third card irrespective of what the player was dealt.
- If the banker's hand is valued at 3, they draw a third card unless the player's third card is an 8.
- If the banker's hand is valued at 4, they draw a third card if the player's third card is between 2 and 7 (inclusive)

- If the banker's hand is valued at 5, they draw a third card if the player's third card is between 4 and 7 (inclusive)
- If the banker's hand is valued at 6, they draw a third card only if the player's third card is a 6 or 7.
- If the banker's hand is valued at 7, they do not draw a card

The Wikipedia page for Baccarat has a nice table that summarizes the "hit/stand" decision for the banker.

After the cards are dealt and the winner determined for the round, bets are paid.

- If the bettor bet "player" and the "player" hand wins, they keep the amount wagered and receive an equal amount from the casino (pays 1-to-1)
- If the bettor bet "banker" and the "banker" hand wins, they keep the amount wagered and receive 95% of their wager from the casino (pays 19-to-20).
- If the bettor bet either "player" or "banker" and the result is a tie, they keep the amount wagered but receive no additional money from the casino (push).
- If the bettor bet "tie" and the result is a tie, they keep the amount wagered and receive 8 times their wagered amount from the casino (pays 8-to-1)
- If the bettor bet "player" and the "banker" wins or if they bet "banker" and the "player" wins, they lose the amount wagered.
- If the bettor bet "tie" and either "banker" or "player" wins, they lose the amount wagered.

15.6.1 Part 1 (-)

Implement, in code, a function that simulates a single round of play (ignore the betting process for now). Assume the casino is playing with a shoe of six decks of standard playing cards and that this shoe is refreshed after every round.

Below is an outline for the sorts of functions you should implement to break this problem down into smaller component parts along with some hints.

```
# This function should take as input some number of decks and generate a shoe of cards in
# HINT: You don't need to store the actual cards, just their values
# HINT 2: Read the documentation for the 'rep' function
gen_deck <- function(){

}
```

```
# This function should take as input some vector that represents a hand of cards and retur
# HINT: In R, the modulo operator is %%
value_hand <- function(){
```

```
}
```

```
# This function should take as input the banker's and the player's hands and return the ou
determine_winner <- function(){

}
```

```
# This is your main function. It should return (at a minimum) the outcome of the round of
# HINT: Write out each phase of the round in plain language in the comments. Then try to i
# HINT 2: You'll likely need to use a lot of conditional statements to implement the drawi
play_round <- function(){

}
```

15.6.2 Part 2 (-)

Now implement a function that takes as input a bettor's chosen outcome and their wager. The function should then play a round of baccarat and return the amount that the bettor receives from the casino. There are a few equally valid ways to implement "winnings." For the purposes of this problem, you should have the function return 0 if the player loses their entire wager.

```
# This function should take as input a choice and a wager. You can choose to have it wrap
standard_payoff <- function(){

}
```

15.6.3 Part 3 (-)

Using a Monte Carlo simulation, calculate the expected return of a wager of 100 dollars on the "player" bet. In other words, if a bettor pays 100 to bet on "player", what is the amount that they expect to win. Set your seed once to 60639 and run it for 300,000 iterations (try running for fewer iterations as you're testing, but you'll need a decent number of iterations to get the desired numerical precision)

Calculate the "house edge" of the "player" bet. The house edge is the difference between a bettor's amount wagered and their expected return, divided by the amount wagered. In other words, it's the amount that a casino expects to keep of a player's bet.

$$\text{HouseEdge} = \frac{\text{AmountWagered} - \text{ExpectedReturn}}{\text{AmountWagered}}$$

```
set.seed(60639) # Set the seed
# Simulation code here
```

15.6.4 Part 4 (-)

Use a Monte Carlo simulation to calculate the expected return of a wager of 100 dollars on the "banker" bet. Calculate the "house edge" of the "banker" bet. Set the seed to 60640 and run for 300,000 iterations.

```
set.seed(60640) # Set the seed
# Simulation code here
```

Which bet has the lower house edge?

15.6.5 Part 5 (-)

One odd feature of Baccarat is that the "banker" bets pay 19-to-20 rather than the 1-to-1 for player bets. Using a Monte Carlo simulation, show why casinos don't pay 1-to-1 on the banker (calculate the house edge). You'll need to write a new payoff function to handle the alternative payout structure. Set the seed to 60641 and run for 300,000 iterations.

```
set.seed(60641) # Set the seed
# Simulation code here
```

15.6.6 Challenge problem (-)

A new version of Baccarat that has become popular in some areas called "EZ-Baccarat" purports to solve the problem you identified in Part 5 while still paying "banker" bets at 1-to-1. It does so by having any round where the banker wins with a hand of 3 cards that totals 7 points be a "push" (bettors keep their wagers but don't win any additional money) rather than a "win" for the banker hand.

Show, using a Monte Carlo simulation, how this payout structure retains the house edge on "banker" bets. You may have to modify your simulation functions to return more information about the outcome of the round in order to implement a new payoff function.

See the state regulatory documents for more info about this version of the game.

```
set.seed(60642) # Set the seed
```