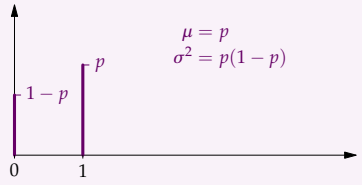
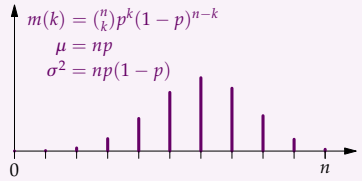
Probability: Common Distributions

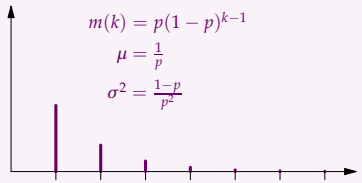
1 **Bernoulli** (Ber(p)): A weighted coin flip.



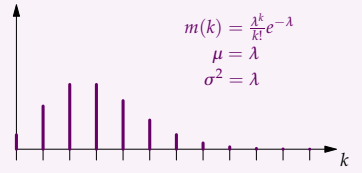
2 **Binomial** (Bin(n, p)): A sum of n independent Ber(p)'s.



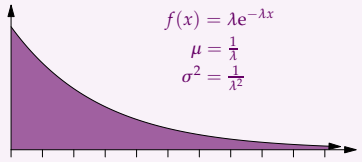
3 **Geometric** (Geom(p)): Time to first success (1) in a sequence of independent Ber(p)'s.



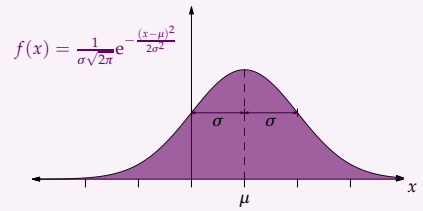
4 **Poisson distribution** (Poiss(λ)): Limit as $n \to \infty$ of Binomial($n, \frac{\lambda}{n}$).



5 **Exponential distribution** (Exp(λ)): Limit as $n \to \infty$ of distribution of $1/n$ times a Geometric(λ/n).



6 **Normal distribution** ($\mathcal{N}(\mu, \sigma^2)$): Limit as $n \to \infty$ of the distribution of $\frac{X_1+X_2+\cdots+X_n}{\sqrt{n}}$, for any independent sequence $X_1, \ldots, X_n$ of identically distributed random variables (i.i.d.) with $\mathbb{E}[X_1] = \mu$ and $\mathrm{Var}(X_1) = \sigma^2 < \infty$ (see Central Limit Theorem).

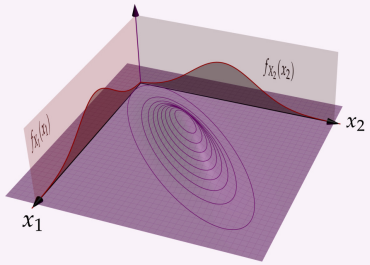


7 **Multivariate normal distribution** ($\mathcal{N}(\mathbf{0}, \Sigma)$): if $\boldsymbol{Z} = (Z_1, Z_2, \ldots, Z_n)$ is a vector of independent $\mathcal{N}(0,1)$'s, A is an $m \times n$ matrix of constants, and $\mu \in \mathbb{R}^m$, then the vector

$$\boldsymbol{X} = A\boldsymbol{Z} + \mu$$

is **multivariate normal**. The covariance matrix of $\mathbf{X}$ is $\Sigma = AA'$.

$$f(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^n \det \Sigma}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})'\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})}$$



Programming in Julia

1 A **value** is a fundamental entity that may be manipulated by a program. Values have **types**; for example, `5` is an `Int` and `"Hello world!"` is a `String`.

2 A **variable** is a name used to refer to a value. We can **assign** a value `5` to a variable `x` using `x = 5`.

3 A **function** performs a particular task. You prompt a function to perform its task by **calling** it. Values supplied to a function are called **arguments**. For example, in the function call `print(1,2)`, `1` and `2` are arguments.

4 An **operator** is a function that can be called in a special way. For example, `*` is an operator since we can call the multiplication function with the syntax `3 * 5`.

5 A **statement** is an instruction to be executed (like `x = -3`). An **expression** is a combination of values, variables, operators, and function calls that a language interprets and **evaluates** to a value.

6 A numerical value can be either an **integer** or a **float**. The basic operations are `+`, `-`, `*`, `/`, `^`, and expressions are evaluated according to the order of operations.

7 Numbers can be compared using `<`, `>`, `==`, `≤` or `≥`.

8 Textual data is represented using **strings**. `length(s)` returns the number of characters in `s`. The `*` operator concatenates strings.

9 A **boolean** is a value which is either `true` or `false`. Booleans can be combined with the operators `&&` (and), `||` (or), `!` (not).

10 Code blocks can be executed conditionally:

```
if x > 0
    "x is positive"
elseif x == 0
    "x is zero"
else
    "x is negative"
end
```

11 Functions may be defined using the familiar math notation: `f(x,y) = 3x + 2y` or using a `function` block (`shift` is a **keyword argument**):

```
function f(x,y; shift=0)
    3x + 2y + shift
end
```

12 The **scope** of a variable is the region in the program where it is accessible. Variables defined in the body of a function are not accessible outside the body of the function.

13 `Array` is a compound data type for storing lists of objects. Entries of an array may be accessed with square bracket syntax using an index or using a **range** object `a:b`: `A = [-5,3,2,1]; A[2]; A[3:end]`.

14 An **array comprehension** can be used to generate new arrays: `[k^2 for k=1:10 if mod(k,2) == 0]`

15 A **dictionary** encodes a discrete function by storing input-output pairs and looking up input values when indexed. This expression returns `[0,0,1.0]`:

```
Dict("blue"=>[0,0,1.0],"red"=>[1.0,0,0])["blue"]
```

16 A `while` loop takes a conditional expression and a body and evaluates them alternatingly until the conditional expression returns false. A `for` loop evaluates its body once for each entry in a given *iterator* (for example, a range, array, or dictionary). Each value in the iterator is assigned to a loop variable which can be referenced in the body of the loop.

```
while x > 0
    x -= 1
end
```

```
for i=1:10
    print(i)
end
```

Learning Standards

- [ ] **1** JULIA
- [ ] **2** LINALG
- [ ] **3** MATALG
- [ ] **4** EIGEN
- [ ] **5** OPT
- [ ] **6** MATDIFF
- [ ] **7** MACHARITH
- [ ] **8** NUMERROR
- [ ] **9** PRNG
- [ ] **10** NUMOPT
- [ ] **11** PROBSPACE
- [ ] **12** CONDPROB
- [ ] **13** BAYES
- [ ] **14** IND
- [ ] **15** EXP
- [ ] **16** COV
- [ ] **17** CONDEXP
- [ ] **18** COMDISTD
- [ ] **19** COMDISTC
- [ ] **20** CLT
- [ ] **21** POINTEST
- [ ] **22** BOOT
- [ ] **23** HYPTEST
- [ ] **24** MLE