Stan Is The Plan

Breck Baldwin

NYC PyData 2019

Goals

- Programmer focused on mechanics, not statistical theory
- Enough statistics to get you started understanding what is going on overall
- Get past high probability failure points

Outline Section 1

- Install CmdStanPy
- Contextualize Bayesian Inference
- Running Example: PuttBett
- Mechanics of authoring Stan model
- Explore execution environment
- Describe how inference is done
- Build simplest possible version of PuttBett

Resources

Tutorial repository

- https://github.com/breckbaldwin/StanIsThePlanDist

Install CmdStanPy

- https://cmdstanpy.readthedocs.io/en/latest/index.html

What do statistics/AI care about?

- Running Example
  - Physics of golf
  - Bayesians build a model to describe the data
- PuttBet--Predict future golf putts:
  - Deep Learning Inference
    - Use fixed model, inference optimized + data to predict
  - Bayesian Inference
    - Write own model, general inference + data to predict

PuttBet: Ideal Golf Betting App

- Will a putt go in?
  - Masters, Tiger Woods, Nov 6 2019, first putt?
- Best possible answer?
  - Return 0, putt will miss
  - Return 1, putt will go in
- What is the best possible backend?
  - Golf Data Weekly from the future
  - Simulation of parallel universe

Run the Final Code

>python run_stan.py mecha_puttbett.stan 4

-

Outline: Model Authoring for Beginners

- Pick the simplest parameters that make sense
  - Betting/Putting app: sink_or_miss(putt info) => [0,1]
  - 0 means miss, 1 means the put goes in (sink)
- Eliminate the impossible (prior knowledge)
  - One putt cannot miss twice or sink twice
  - A persons height can't be negative or > 10 ft
- Find a way to incorporate data into priors (likelihood) that turns into a posterior
- Have a way to decide if the posterior is useful

An Important Philosophical Point

- Can't model world for detail and scope reasons
  - Detail: Computationally too complex & don't have theory
  - Scope: Computationally too big & don't have data
- We can approximate however by averaging things out
  - We limit how much we look at
  - We ignore interactions hoping it won't matter
- This gets us uncertainty
  - Instead of 0 or 1, we say 0.01 or .5 or .99

Uncertainty

- Calibration: .99 probability means that 99 times out of 100 we get X, 1 time we don't....over time....
- Often abused, a system will claim .99 but it is not.
- Difficult to separate poor calibration from uncertainty.
  - We don't get HTHT on coin flips all the time
  - Over large amounts of data we should get .5 H
  - We should also get HHHHH with enough data

1st Revision to our model

- Is PuttBet modeling the entire universe?
  - No
- Does PuttBet have access to the future?
  - No
- The PuttBet app will now return chance in 5 to reflect our uncertainty.
  - 0 chance in 5 is 0% probability
  - 5 chance in 5 is 100%
  - 2.5 chance in 5 is 50%

Code model up in Stan

Go to:

https://github.com/breckbaldwin/StanIsThePlanDist/

Copy and paste from stan/no_putt.stan or just type into an editor

```
parameters {
      real <lower=0, upper=5> chance_in_5;
}

model {

}
```

Save as 'no_putt.stan'

```
>cd <path to cmdstan>
>make <path to no_putt.stan, but drop .stan>
><path to no_putt.stan> sample
```

Running Stan Program

```
>./no_putt sample
method = sample (Default)
  sample
    num_samples = 1000 (Default)
    num_warmup = 1000 (Default)
    …
Iteration: 2000 / 2000 [100%]  (Sampling)

 Elapsed Time: 0.013104 seconds (Warm-up)
               0.035397 seconds (Sampling)
               0.048501 seconds (Total)
```

Inspect output.csv

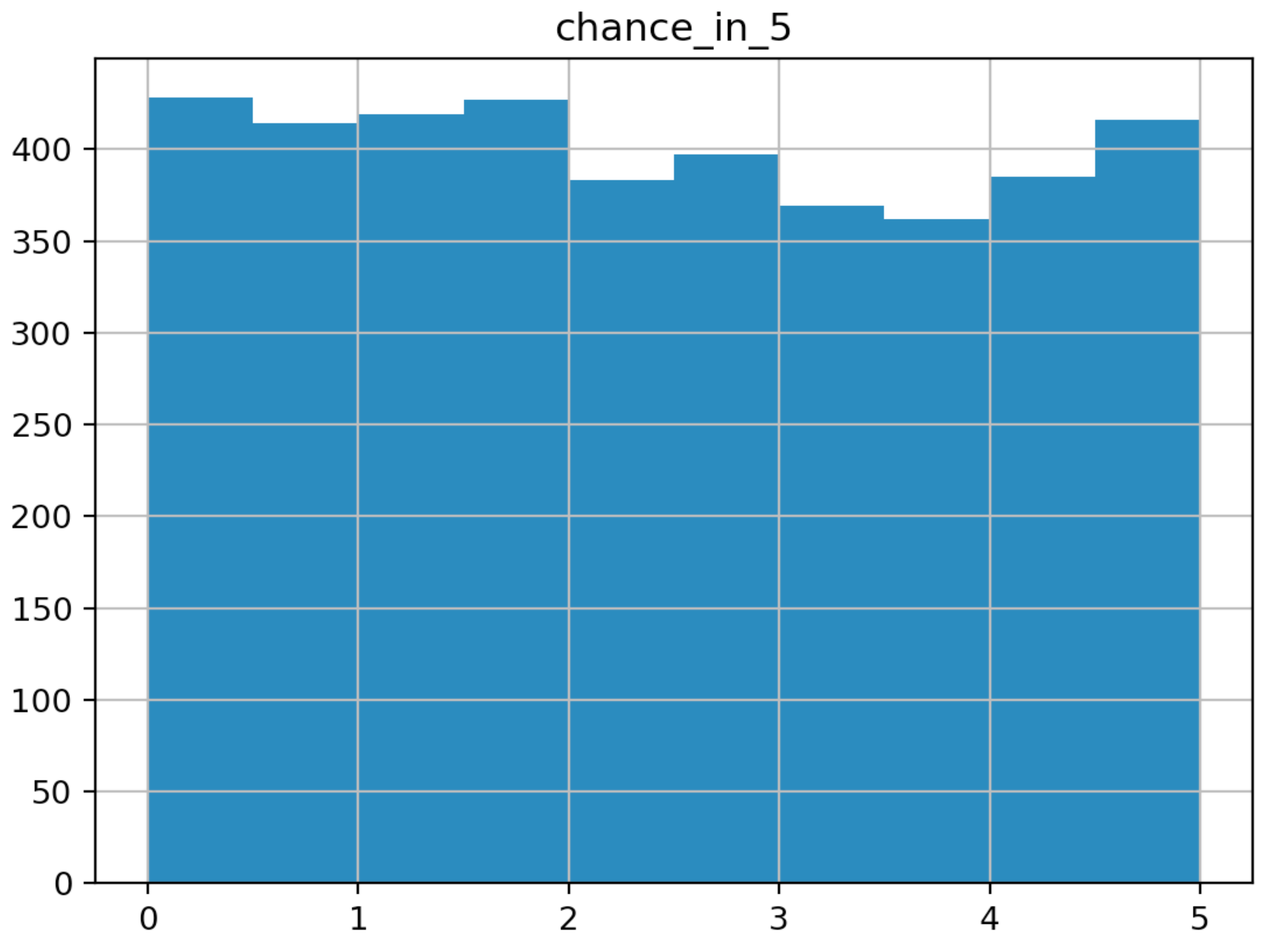
```
>less output.csv
lp__,accept_stat__,...,divergent__,energy__,  chance_in_5
-0.788662,0.834613,1.12129,1,1,0,0.799103,   4.49443
-0.34793,0.967041,1.12129,2,3,0,1.39449,     0.850982
-1.74739,0.802025,1.12129,2,3,0,1.78157,     4.81924
-1.74739,0.963619,1.12129,1,1,0,2.66079,     4.81924
…
0.171325,1,1.12129,1,1,0,-0.14552,           3.0618
0.0610676,0.958562,1.12129,1,1,0,-0.048304,  3.46703
-0.56155,0.829213,1.12129,1,3,0,0.878503,    0.656531
```

Values for chance_in_5

```
  [1] 4.983672319 0.504675345 0.956908493 2.875748848 0.997364239 0.417071450 4.501827071 0.936049387 0.863498351
 [10] 4.153279253 2.060605266 1.139810114 3.779605672 1.658594148 1.067616110 4.116676326 2.330651455 3.967579063
 [19] 1.842490519 0.455000172 1.157479467 2.752047595 0.915102101 4.982286286 4.391265367 4.992451716 2.447138984
 [28] 2.205051616 4.231899958 0.812689175 3.989032044 0.796250061 0.859436737 1.028724611 1.924055572 3.208221532
 [37] 2.210415693 2.136084733 1.248423340 3.315607704 2.254323347 4.784930724 4.885145022 1.982817767 4.984226181
 [46] 0.048460657 0.282379449 3.722942450 0.999613762 2.502718692 3.243868759 0.822543222 0.211249991 1.147694845
 [55] 4.896531799 4.199396308 0.665606386 3.141129394 2.607873759 4.690558455 4.696524164 4.031962569 2.755539678
 [64] 1.283222482 2.248947510 0.319166050 0.636175197 1.570415334 2.837057149 2.539923313 3.181580786 1.233043905
 [73] 3.061787637 4.764964321 0.918676119 0.480524564 1.491754639 3.321344687 0.492543512 2.089836199 4.983904438
 [82] 1.547706472 2.414447829 0.282179524 1.666381844 0.367431461 4.361380324 3.373313419 4.397065423 1.689771086
 [91] 1.141912694 2.459350776 4.159030935 3.116608167 3.076992472 0.823561946 2.969654653 1.165449756 1.658783029
[100] 0.351305357 1.694081739 1.259235890 1.387418442 2.528962612 4.852489185 1.374381734 2.941864848 1.157997846
[109] 0.168440287 4.373520019 3.886488608 1.093028201 4.987565650 4.812540735 4.595358682 0.655243261 2.834453881
[118] 2.728336762 1.977742924 2.179817460 0.320485894 3.711874986 0.189774115 2.125163616 2.366089869 2.752047595
[127] 2.270779435 2.060605266 0.497764905 1.841243712 0.039907438 3.040357791 2.607759999 1.721114910 2.466011311
...
```

Histogram of 'chance_in_5'

```
>python run_visualize.py stan/no_putt.stan chance_in_5
```



What do Histograms Provide?

- End goal is to figure out probability for values of 'chance_in_5'.
- We bin 10 ways, 0-0.499, .5-.999, ...4.5-5
  - Any exact value is unlikely to be found in output.csv
  - Human interpretable
- P(chance_in_5 < 2.5) = 50%
  - count(values < 2.5) / count(all values)
  - Look at values in output.csv
  - Look at fit object returned in python

Exercise

- Given the uniform distribution above:
  - What are some ranges of 'chance_in_5' that have 50% probability?
  - Is any value of the uniform distribution more likely than any other?
  - Name some phenomenon and the relative parameter scale that is uniformly distributed.
  - Name some phenomenon that are not uniformly distributed.

Playing with parameter scales

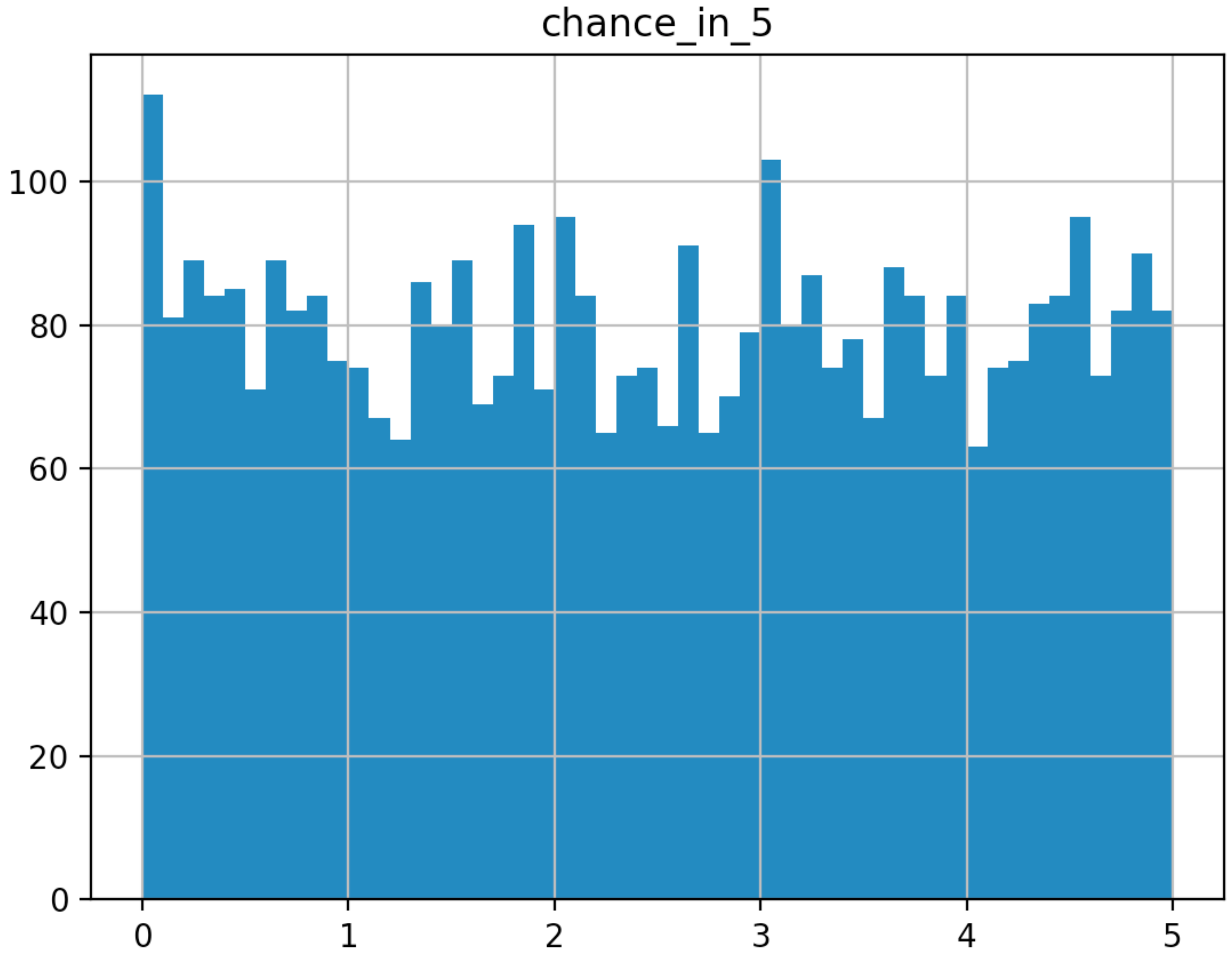
- Instead of 0 to 5 we use A-E
  - A=0-.999
  - B=1-1.999
  - C=2-2.999
  - D=3-3.999
  - E=4-5
- What is probability of E?

Messing with Distributions

```
>python rv.py stan/uniform_uniform.stan
chance_in_5

parameters {

      real <lower=0, upper = 5> chance_in_5;

}

model {

      chance_in_5 ~ uniform(0,5);

}
```

- Note that the lower/upper are the same.

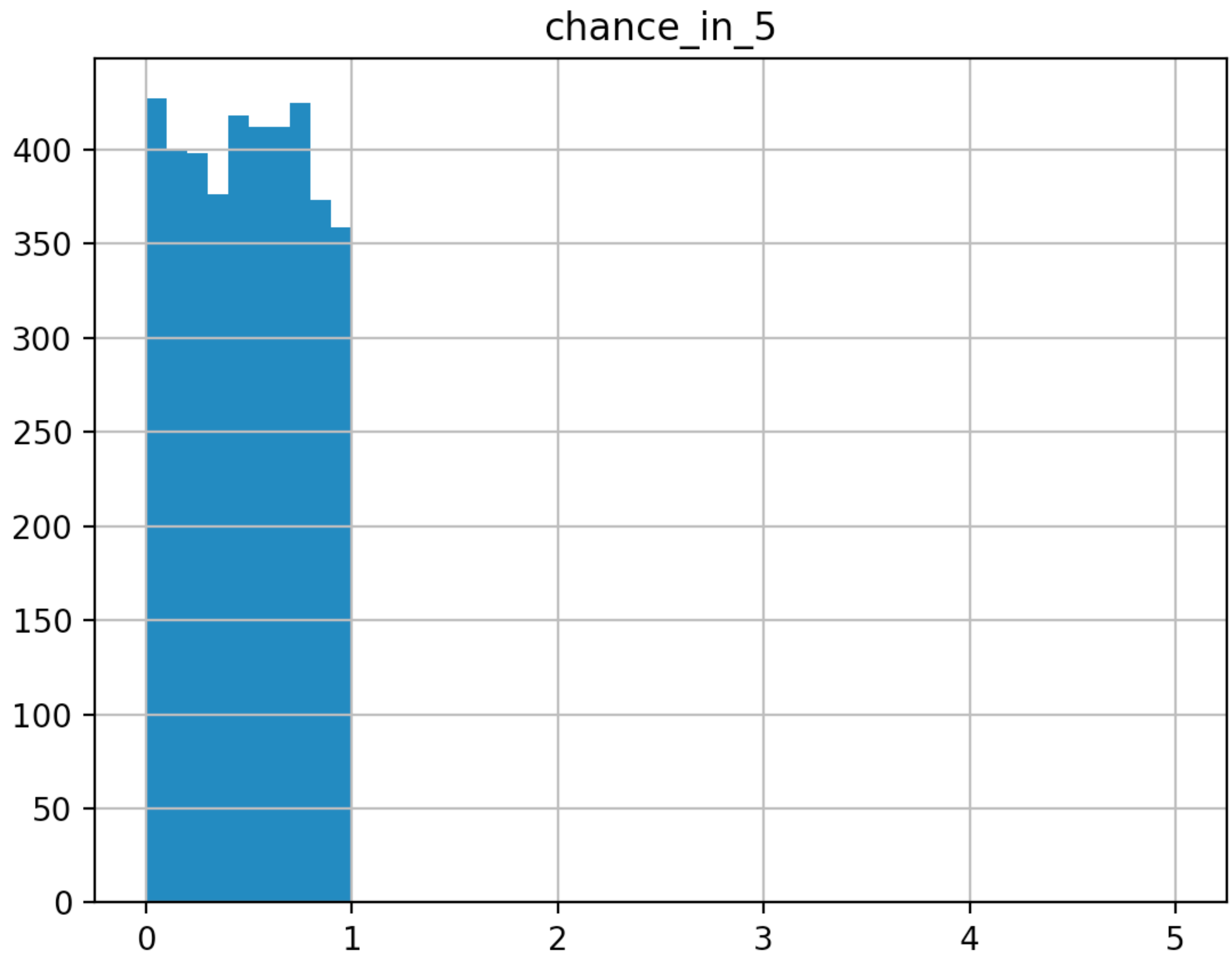
Uniform x Uniform



Uniform Haircut

```
parameters {
   real <lower=0, upper = 5> chance_in_5;
}

model {
  chance_in_5 ~ uniform(0,1);
  //target += uniform_lpdf(chance_in_5|0,1);
}
> python run_visualize.py
stan/targ_unif_unif.stan chance_in_5
```

uniform(0,5) x uniform(0,1)



Animation

Same model

```
parameters {
      real <lower=0, upper = 5> chance_in_5;
}
model {
  //chance_in_5 ~ uniform(0,1);
  target += uniform_lpdf(chance_in_5|0,1);
}
python rv.py stan/targ_unif_unif.stan chance_in_5
```

Same model

```
parameters {
      real <lower=0, upper = 5> chance_in_5;
}
model {
  //chance_in_5 ~ uniform(0,1);
  target += uniform_lpdf(chance_in_5|0,1);
}
python rv.py stan/targ_unif_unif.stan chance_in_5
```

The Mechanics of the Haircut

```
parameters {
   real chance_in_5;
}

model {
  //chance_in_5 ~ uniform(0,1);
  real prob_of_param;
  print("chance_in_5=",chance_in_5);
  print("start: target()=",target())
  print("   exp(target())=",exp(target()));

  prob_of_param = uniform_lpdf(chance_in_5|0,1);
  print("uniform_lpdf(chance_in_5|0,1)=",prob_of_param);
  print("exp(uniform_lpdf(chance_in_5|0,1))=",exp(prob_of_param));

  target += prob_of_param;
  print("end: target()=",target());
  print(" exp(target())=",exp(target()));
}

> rv.py stan/print_targ_unif_unif.stan
```

Print Output

```
chance_in_5=0.0162785
start: target()=0
   exp(target())=1
uniform_lpdf(chance_in_5|0,1)=0
exp(uniform_lpdf(chance_in_5|0,1))=1
end: target()=0
 exp(target())=1

chance_in_5=-0.0613036
start: target()=0
   exp(target())=1
uniform_lpdf(chance_in_5|0,1)=-inf
exp(uniform_lpdf(chance_in_5|0,1))=0
end: target()=-inf
 exp(target())=0
```

Contents of output-1.csv

aka target()

```
lp__,accept_stat__,stepsize__,treedepth__,n_leapfrog__,divergent__,energy__,chance_in_5
# Adaptation terminated
# Step size = 0.158814
# Diagonal elements of inverse mass matrix:
# 0.079692
0,0.909091,0.158814,3,11,1,0.140977,0.849007
0,0.974359,0.158814,5,39,1,0.00612349,0.903583
0,0.5,0.158814,1,2,1,1.28248,0.975385
0,0.984615,0.158814,6,65,1,0.0200276,0.481883
0,0.994565,0.158814,7,184,1,0.00198173,0.18552
```

aka target()

Contents of output-1.csv

```
lp__,accept_stat__,stepsize__,treedepth__,n_leapfrog__,divergent__,energy__,chance_in_5
# Adaptation terminated
# Step size = 0.158814
# Diagonal elements of inverse mass matrix:
# 0.079692
0,0.909091,0.158814,3,11,1,0.140977,0.849007
0,0.974359,0.158814,5,39,1,0.00612349,0.903583
0,0.5,0.158814,1,2,1,1.28248,0.975385
0,0.984615,0.158814,6,65,1,0.0200276,0.481883
0,0.994565,0.158814,7,184,1,0.00198173,0.18552
```

Contents of output-1.csv

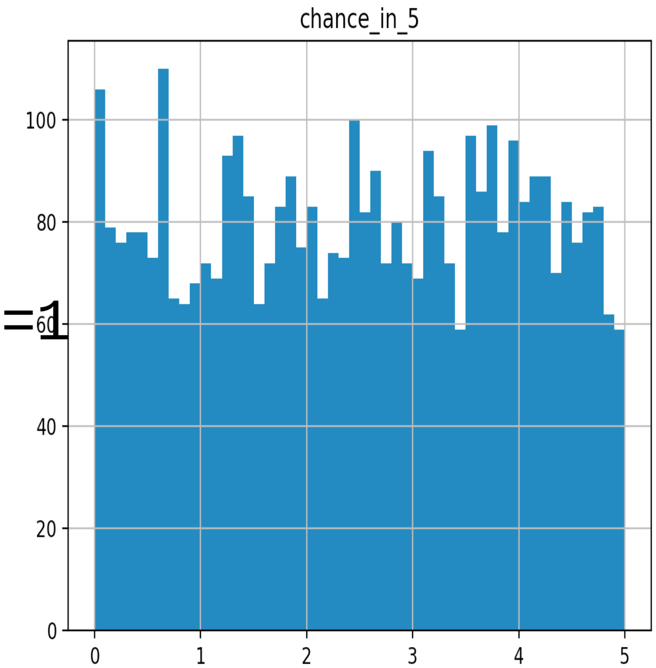
aka target()

```
lp__,accept_stat__,stepsize__,treedepth__,n_leapfrog__,divergent__,energy__,chance_in_5
# Adaptation terminated
# Step size = 0.158814
# Diagonal elements of inverse mass matrix:
# 0.079692
0,0.909091,0.158814,3,11,1,0.140977,0.849007
0,0.974359,0.158814,5,39,1,0.00612349,0.903583
0,0.5,0.158814,1,2,1,1.28248,0.975385
0,0.984615,0.158814,6,65,1,0.0200276,0.481883
0,0.994565,0.158814,7,184,1,0.00198173,0.18552
```

Rescale from 0-1 to 0-5

```
prob_of_param = uniform_lpdf(chance_in_5|0,1);
1)prob_of_param = uniform_lpdf(chance_in_5/5|0,1);
2)prob_of_param = uniform_lpdf(chance_in_5|0,5);
```

```
chance_in_5=4.6081
start: target()=0
   exp(target())=1
uniform_lpdf(chance_in_5|0,1)=0
exp(uniform_lpdf(chance_in_5|0,1))=1
end: target()=0
 exp(target())=1
```



Summarizing

- Set up a parameter to estimate:chance_in_5
- We sampled from it with no model
- All values between 0 and 5 equi-probable
- Probabilities are determined by histogram
- We added a uniform prior—had scaling issues
- Big part: target() aka lp__

Summarizing

- Set up a parameter to estimate:chance_in_5
- We sampled from it with no model
- All values between 0 and 5 equi-probable
- Probabilities are determined by histogram
- We added a uniform prior—had scaling issues
- Big part: target() aka lp__

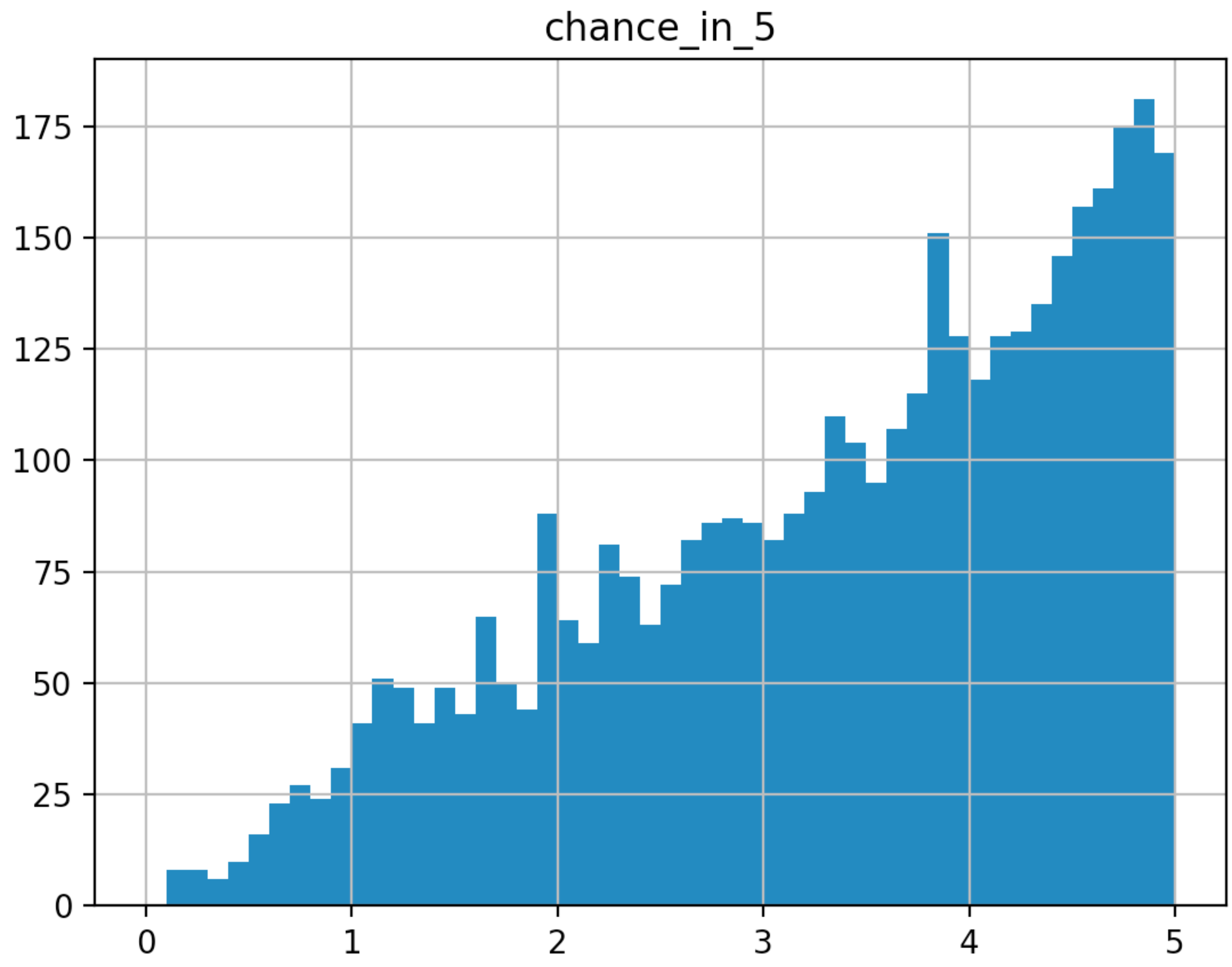
Adding some data to our model

- The PuttBet app returns chance in 5
- Teach it about putts with historical data
- Putt went in = 1, putt did not = 0

```
> python pv.py stan/one_putt.stan chance_in_5:

parameters {
   real<lower=0, upper=5> chance_in_5;
}

model {
  real chance_in_1 = chance_in_5/5;

  //chance_in_1 ~ uniform(0,1);
  //  1 ~ bernoulli(chance_in_1);

  target += uniform_lpdf(chance_in_1|0,1);
  target += bernoulli_lpmf(1|chance_in_1);
}
```

Running the model

```
> python rv.py stan/one_putt.stan chance_in_5
```



What happened?

- Understand the Bernoulli distribution
- Expose the implicit loop around blocks
- Give the intuition around what the target() does

What is this Bernoulli thing?

- Putt goes in: 1
  - exp(bernoulli_lpmf(1|.99)) = .99
  - exp(bernoulli_lpmf(1|.1)) = .1
  - exp(bernoulli_lpmf(1|.5)) = .5
- Putt does not go in: 0
  - exp(bernoulli_lpmf(0|.99)) = .01
  - exp(bernoulli_lpmf(0|.1)) = .9
  - exp(bernoulli_lpmf(0|.5)) = .5

breck_noulli function

```
> python rv.py stan/breck_noulli.stan theta

functions {

  real breck_noulli_lpmf(int zero_or_one,
                         real param_to_return_prob_of) {
    if (zero_or_one == 1) {
      return log(param_to_return_prob_of);
    }
    return log(1.0-param_to_return_prob_of);
  }
}

parameters {
  real<lower=0,upper=1> theta;
}

model {
  1 ~ breck_noulli(theta);
}
```

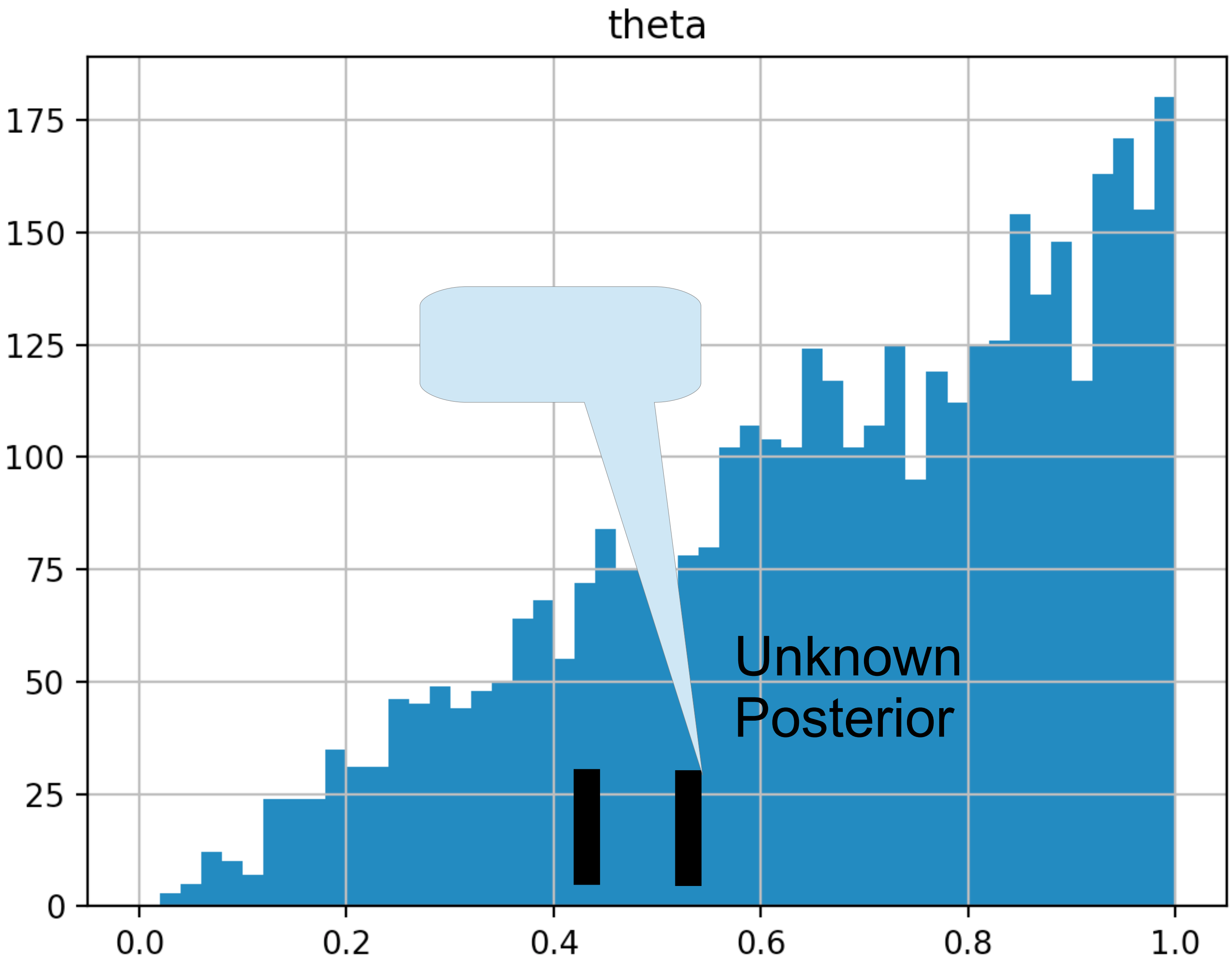
Simplified (Wrong) Evaluation

1. Once:
   1. Read `data{}` from outside only (R data, JSON)
   2. Execute `transformed data{}`, can assert data here, grab variables from `data{}`.
2. For each sample:
   1. Make a guess at all `parameters{}`: the proposal
   2. Start `target()=0`, `log(target())=1`
   3. Multiply the `target()` with priors/likelihood.
   4. If target() > last target(), accept proposal.
   5. If target() < last target(), accept at target()/last target() ratio, else keep last target().

How did Bernoulli change the posterior (Metropolis Hastings)?

- last_exp(target())= .5, values of params recorded.
- `exp(target()) = 1`.
- `chance_in_1 = .4`
- exp(target()) * exp(uniform_lpdf(.4,1,0)) = 1
- exp(target()) * bernoulli_lpmf(1|.4) = .4
- .4/.5 < 1 so we accept 40% of the time
- 40% of .4 param survive, they reduce in histogram count.

Different Version



Distributions in Action

- Animation uniformXbernoulli

Summary

- We have a basic idea how information flows
- Posteriors are interpreted as histograms
- We can convert '~' notation to a more programmer understandable version
- We know that several scales are in play
  - log()/exp()
  - Rescale parameter fit standard distribution
  - All are magnitude preserving log(a) > log(b), a> b

Mess up the Scale

- Look at console output, lots of warnings
- Run
  - `~/cmdstan-2.21.0/bin/stansummary output-1.csv`

```
divergent__ 0.52  5.5e+03  1.0e+00
```

```
52% of samples were problematic
```

Mess up the Scale

- Look at console output, lots of warnings
- Run
  - `~/cmdstan-2.21.0/bin/stansummary output-1.csv`

```
divergent__ 0.52  5.5e+03  1.0e+00
```

```
52% of samples were problematic
```