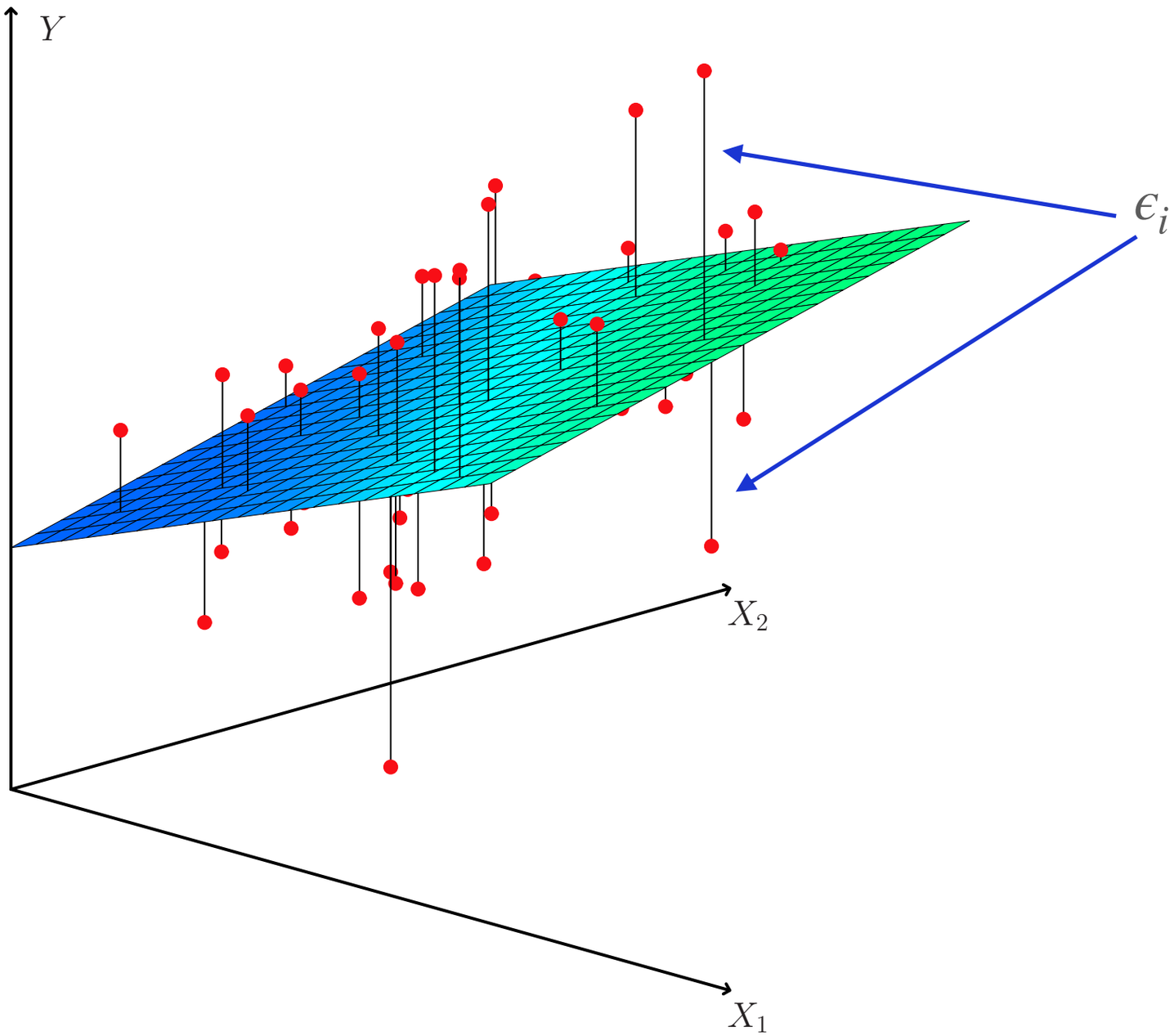
WITH LOW POWER COMES GREAT RESPONSIBILITY.

Stan for ecological modelling.

Instructors (in order of appearance):

Daniel Simpson, University of Toronto
Andrew MacDonald, University of Montreal
Vianey Leos Barajas, NC State / University of Toronto

REGRESSION IN A PICTURE



Some figures taken from "An Introduction to Statistical Learning, with applications in R" (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani

FREQUENTIST INFERENCE

SO WHAT ARE WE DOING?

- This type of classical inference (sometimes known as **frequentist** inference) has an underlying probabilistic framework:
  - The data y is random
  - The estimator $\hat{\beta}(y)$ is a **deterministic** function of the data
  - We can then make probability statements about how often the **true value is within some interval around the estimator.**

INTERPRETING LEAST SQUARES

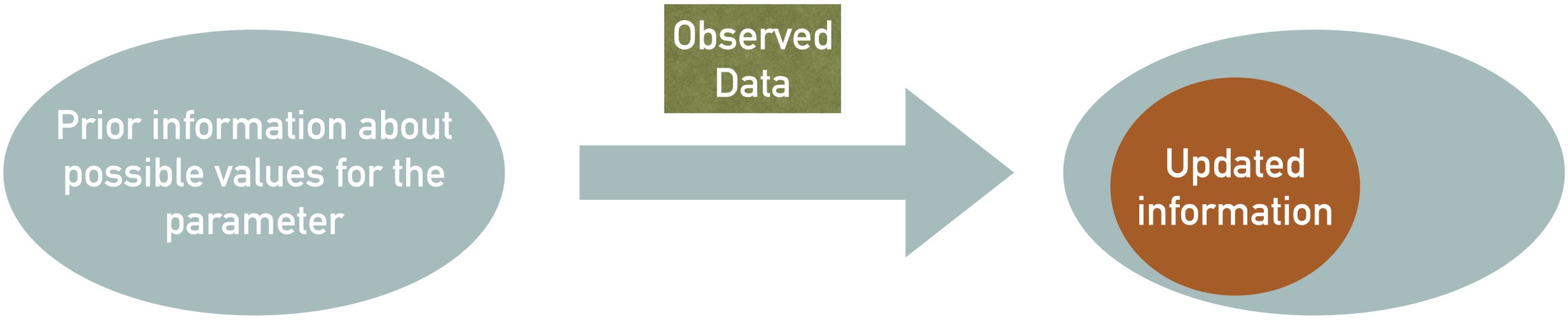
- This means that we actually construct a **box** of values for β and say that with some high probability the **true value** is in that box.
- So we are always making probabilistic statements about the true value of the regression line and how uncertain we are as a function of data

BUT THERE'S ANOTHER WAY

A DIFFERENT QUESTION

- What if, instead of asking questions about the true value that we would know perfectly if we had infinite data, we asked a slightly different question?
- ***Which values of β are consistent with the data we have observed?***
- This is a different question, but in a lot of cases it can give similar answers
- It does not rely on getting infinite amounts of data, but instead focuses on the data in hand.

HOW DO WE DO THIS?



$$\Pr(\beta) \qquad \Pr(y \mid \beta) \qquad \Pr(\beta \mid y) = \frac{\Pr(y, \beta)}{\Pr(y)}$$

Prior

Likelihood

Posterior

WHAT DO WE WANT TO COMPUTE?

- First, we need to ask what we actually want to do with this posterior distributions.

- The obvious things are computing means, medians, and standard deviations, and intervals.

- We also want to plot marginal (eg $p(\beta_j \mid y)$) and joint distributions

- **Key insight:** We can get all of this if we have **samples** from the posterior.

SAMPLES!!!!!!

- Let's say we have a bag of J samples θ_j from a posterior distribution $p(\theta \mid y)$.

- We can use these samples to approximate posterior expectations by the sample mean

$$\mathbb{E}(h(\theta) \mid y) \approx \frac{1}{J} \sum_{j=1}^{J} h(\theta_j)$$

- We can approximate the median (or any quantile) by the sample quantile.

SAMPLES!!!

- So a key question in applied Bayesian statistics turns out to be **"How can I sample from an essentially arbitrary distribution?"**

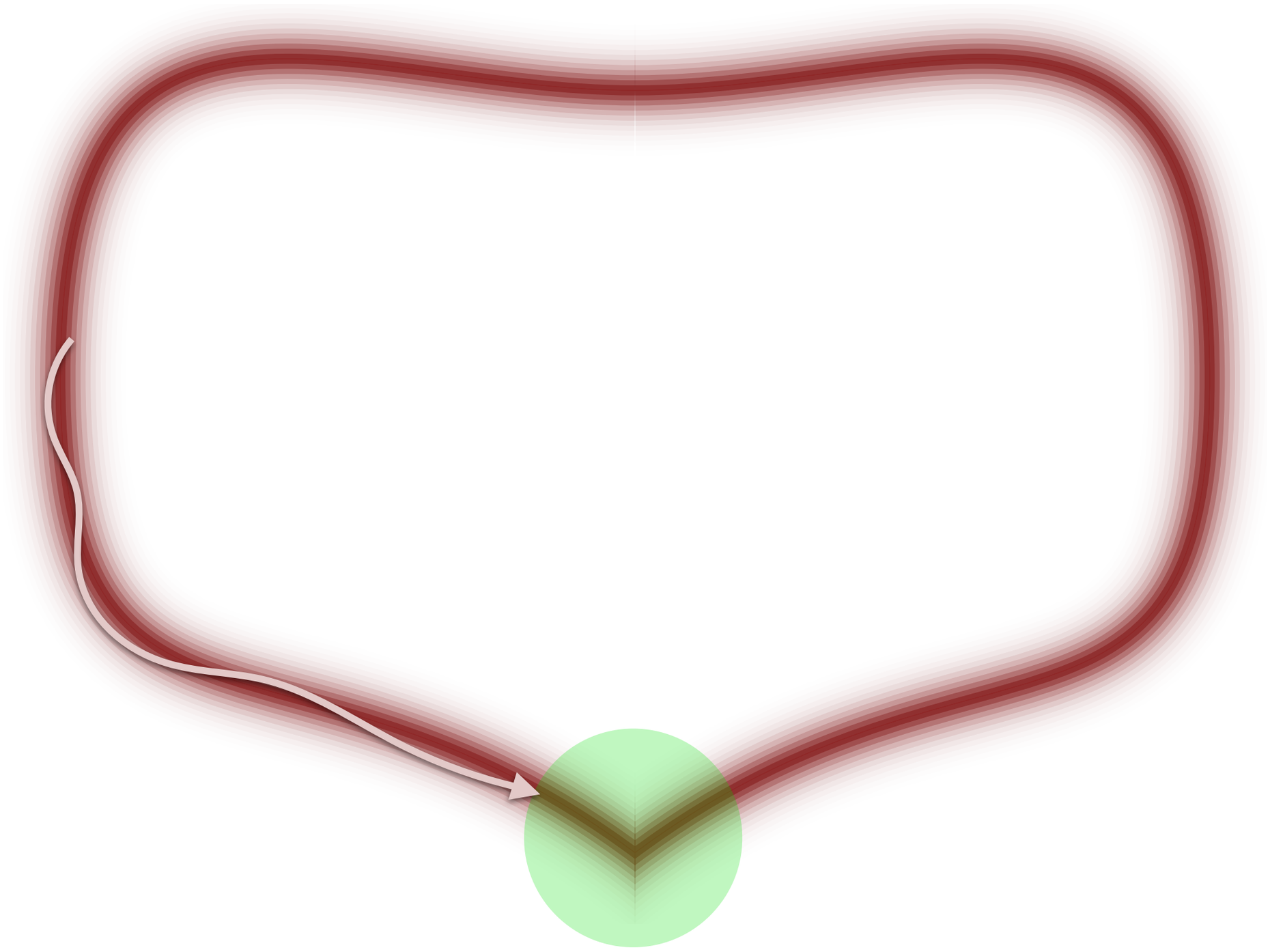
- Somewhat surprisingly, there is a fairly simple algorithm for doing this, known as the **Metropolis-Hastings algorithm.**

- It is part of a general class of algorithms called **Markov Chain Monte Carlo** or MCMC algorithms

HOW DO WE ACTUALLY DO THIS?

- In the bad old days, to do Bayesian statistics you (or someone near and dear to you) had to program up your own MCMC method.
- For extremely complex problems, you still have to.
- But for the sort of problems we hit in real life, we have probabilistic programming languages (like Stan) that do the hard work for you.
- They also provide high-quality diagnostic tools so you know when you can trust the results.

THIS IS WHERE STAN COMES IN



THIS IS WHERE STAN COMES IN

HOW DOES STAN DO IT?

- HMC (Hamiltonian Monte Carlo) is a fancy algorithm for sampling from probability distributions that are known up to a normalizing constant.
- It uses first derivative information about the log-target density to carefully move around the posterior in a geometrically motivated way.
- Stan contains a **dynamic** implementation of HMC that extends the No-U-Turn sampler of Hoffman and Gelman (2014)
- It is stable, efficient, and the implementation is robust (when it fails it fails noisily)

INTRODUCING STAN

WHAT IS IT?

- Stan (it's a name, not an acronym)

- It's named after Stanislaw Ulam, who pioneered the Monte Carlo method and thermonuclear bombs.

- (Like Julia, Python, R, Rust, etc) it is very hard to Google.

HOW TO INTERACT WITH STAN

- **The interfaces:**
  - The main ones (very flexible):
    - CmdStan (BSD), CmdStanPy/CmdStanR (BSD–in alpha)
    - PyStan (GPL3)
    - Rstan (GPL3)
- The nice ones! (limited but a lower barrier to entry):
  - rstanarm, R package, good for multilevel models!
  - brms, R package, good for flexible regression models!

WHY MAKE A NEW LANGUAGE?

- Also it allows people who work natively in R work with people who work in Python work with command line fiends
- Gives a **language** for communicating statistical models
- Arguable downsides include:
  - Barrier to entry
  - It's not "pythonic" or "R-ish" or "Stata-like"
  - It's **strongly typed** (ie you have to specify for each parameter what type of thing it is, eg vector, real, integer, etc)

Posterior densities are specified in a comprehensive, but *user-oriented* probabilistic programming language.

```
data {
  int<lower=1> N;
  real x[N];
  real y[N];
}

parameters {
  real beta;
  real alpha;
  real<lower=0> sigma;
}

model {
  beta ~ normal(0, 1);
  // target += normal_lpdf(beta |0,1);

  alpha ~ normal(0, 1);
  sigma ~ normal(0, 2.5);

  y ~ normal(beta * x + alpha, sigma);
}
```

THE NICER INTERFACES ARE JUST NICER

- rstanarm has very similar syntax to lme4

```
fit_partialpool <-
  stan_glmer(cbind(Hits, AB - Hits) ~ (1 | Player), data = bball,
             family = binomial("logit"),
             prior_intercept = wi_prior, seed = SEED)
```

- brms has very similar syntax to mgcv

```
m2 <- brm(bf(accel ~ s(times)),
          data = mcycle, family = gaussian(), cores = 4, seed = 17,
          iter = 4000, warmup = 1000, thin = 10, refresh = 0,
          control = list(adapt_delta = 0.99))
```

- The differences come because you've got to set some priors!

SOME THINGS THAT STAN DEFINITELY CANNOT DO

- Stan is excellent but it is not magic.

- It has **no support** for discrete parameters.

- Stan will is not guaranteed to properly explore strongly multimodal distributions (but nothing else is either).

- **Stan will not fix problems with your model.**

OUTLINE FOR THE REST OF THE COURSE

- Andrew will take us through how prior distributions work and how to make the associated modelling choices.
- Vianey will take us through our first models in the Stan language and touch on how to select variables
- (We will all have a nice night's sleep)
- Dan will continue on some modelling techniques with Stan
- Andrew will take us through model checking and model choice after you've computed the posterior.
- Vianey will give a broad picture of where to go from here.

BONUS REFERENCE SLIDES!

SO HOW DOES THE STAN LANGUAGE WORK?

- Organize the program into **blocks**
- The blocks respect the natural structure of a Bayesian model
  - (Functions)
  - Data
  - (Transformed Data)
  - Parameters
  - (Transformed Parameters)
  - Model
  - Generated quantities

	data	transformed data	parameters	transformed parameters	model	generated quantities
Execution	Per chain	Per chain	NA	Per gradient	Per gradient	Per iteration
Variable Declarations	Yes	Yes	Yes	Yes	Yes	Yes
Variable Scope	Global	Global	Global	Global	Local	Local
Variables Saved?	No	No	Yes	Yes	No	Yes
Add to *target*?	No	No	No	No	Yes	No
RNGs?	No	Yes	No	No	No	Yes

Static typing facilitates program transforms, automated diagnostics, and robust specification in general.

Primitive: int, real

Matrix: matrix[M,N], vector[M], row_ vector[N]

Bounded: <lower=L>, <upper=U>, <lower=L, upper=U>

Constrained Vectors: simplex[K], ordered[N],
positive_ordered[N]

Constrained Matrices: cov_matrix[K], corr_matrix[K],
cholesky_factor_cov[M,N],
cholesky_factor_corr[K]