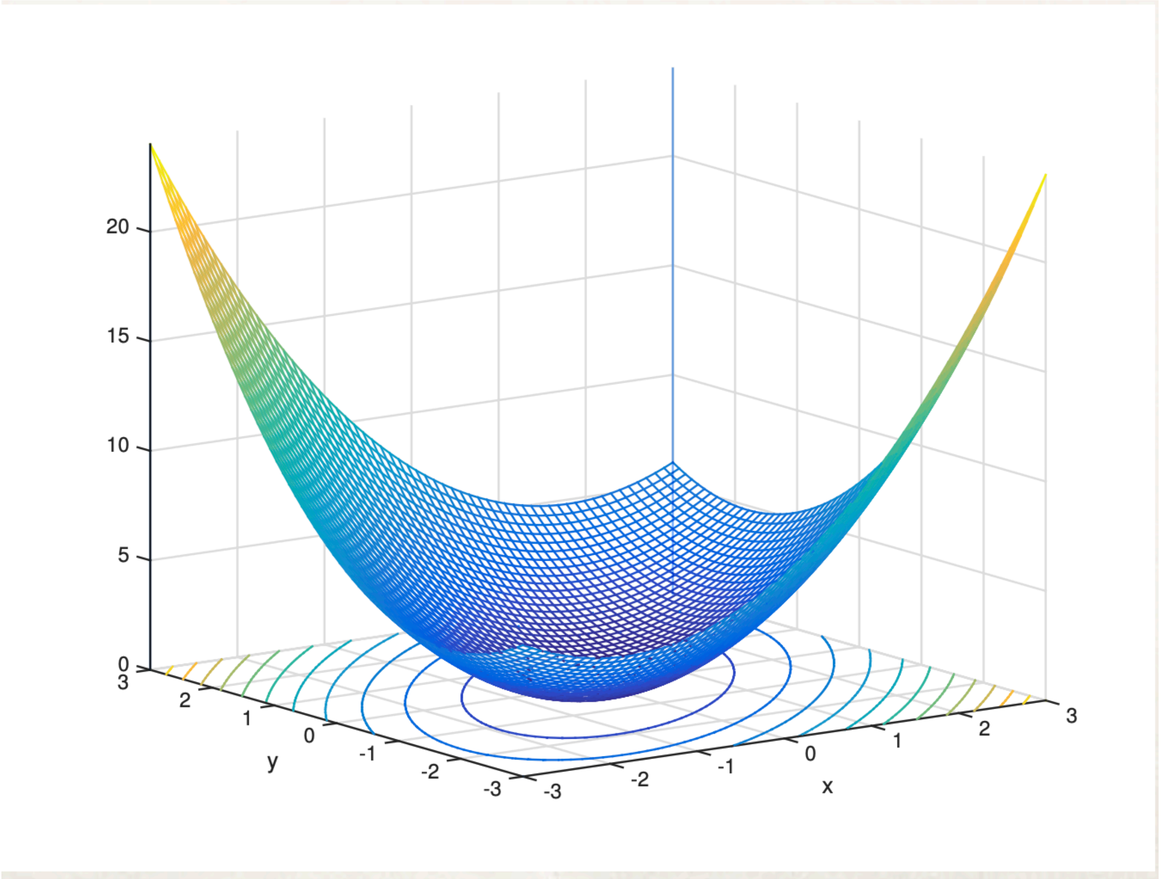
Intro to Stan

What is HMC?

- Hamiltonian Monte Carlo is a type of Metropolis-Hastings algorithm which uses physics equations to generate proposals
- It treats the posterior as a surface, and potential samples as particles traveling across the surface with kinetic and potential energy
- Uses the gradient of the posterior to do this
- Computationally intensive, but often successful at generating good proposals for challenging posteriors (high-dimensional or highly correlated)
- Usually can’t have discrete parameters (no gradients!)



Credit to Babak Shahbaba

What is Stan?

- Stan is a probabilistic programming language which implements the NUTS algorithm—a specialized version of HMC
    - Others include Pyro (Python) and Turing (Julia)
- A wide range of models can be implemented in Stan without doing any difficult math
    - You don’t have to write your own gradients
- It is also (relatively) well documented
- Models are written in a separate Stan file, which can be used in combination with other languages (such as R, Python and Julia) to produce posterior samples

Anatomy of a Stan Model

- Sections denoted by “**name** {}”
- **Data**
  - Everything the user inputs
  - Data and priors
- **Parameters**
  - Unobserved quantities
- **Model**
  - Priors and Likelihood

```
// The input data is a vector 'y' of length 'N'.
data {
  int<lower=0> N;
  vector[N] y;
}

// The parameters accepted by the model. Our model
// accepts two parameters 'mu' and 'sigma'.
parameters {
  real mu;
  real<lower=0> sigma;
}

// The model to be estimated. We model the output
// 'y' to be normally distributed with mean 'mu'
// and standard deviation 'sigma'.
model {
  y ~ normal(mu, sigma);
}
```

Beta-Binomial Hierarchical Model

$$\alpha \sim \text{Exp}(\lambda_\alpha)$$

$$\beta \sim \text{Exp}(\lambda_\beta)$$

$$\theta_i \sim \text{Beta}(\alpha, \beta)$$

$$x_i \sim \text{Binom}(n_i, \theta_i)$$

Data

- Quantities in the data need to have their types defined
- int<lower=0> **num_obs** means **num_obs** is an integer which must be greater than 0
- int x[num_obs] means x is a vector of integers of length **num_obs**
- real **lambda_alpha** is a parameter for the prior for lambda, it can be any real number
- Prior parameters go in the data block because the user specifies them

```
data {
  int<lower=0, upper=1> priors_only; //if true samples from the prior
  int<lower=0> num_obs; //number of observations
  int x[num_obs]; //number of successes
  int n[num_obs]; //number of trials
  real lambda_alpha; //lambda for alpha prior
  real lambda_beta; //lambda for beta prior
}
```

Parameters

- Theta is a vector of parameters of length num_obs, they are real numbers limited to be between 0 and 1
- alpha is a parameter for the prior for theta, it is a real number which must be greater than 0
- Same for beta
- Anything that gets a prior goes into parameters
  - (Not quite true, there's another block called transformed parameters)

```
parameters {
  real<lower=0, upper=1> theta[num_obs];
  real<lower=0> alpha;
  real<lower=0> beta;
}
```

Model

- Write the hyper-priors, priors and likelihood statements in this block
- Alpha ~ exponential(lambda_alpha) means $\alpha \sim \text{Exp}(\lambda_\alpha)$
- The for loop is one way to write
  - $\theta_i \sim \text{Beta}(\alpha, \beta)$
  - $x_i \sim \text{Binom}(n_i, \theta_i)$

```
model {
  //hyper priors
  alpha ~ exponential(lambda_alpha);
  beta ~ exponential(lambda_beta);

  //likelihood and prior theta
  for (i in 1:num_obs){
    theta[i] ~ beta(alpha, beta);

    if (!priors_only) {
      x[i] ~ binomial(n[i], theta[i]);
    }
  }

}
```

Other Blocks

- **Functions:** write your own functions which are used elsewhere in the model
  - For instance, an ODE solver
- **Transformed Data:** create functions of your data
- **Transformed Parameters:** create functions of your parameters
  - For instance, ODE solutions
  - Stan will produce draws of anything defined in this block
- **Generated Quantities:** produce draws from the posterior predictive distribution