# Section 6.
# Fast Stan

**Vincent Dorie**

New York University

# Part I
# Vectorization

# Matrices vs Arrays

```
+------------+                                  +---------------+
|real a[3,2];|                                  |matrix[3,2] b;|
+------------+     +-------------+              +---------------+
| real a[2]; |----->|a[1,1],a[1,2]|              |    b[1,1]     |
| real a[2]; |----+ +-------------+              |    b[2,1]     |
| real a[2]; |-+   |                             |    b[3,1]     |
+------------+ |   | +-------------+              |    b[1,2]     |
               |   +>|a[2,1],a[2,2]|              |    b[2,2]     |
               |     +-------------+              |    b[2,3]     |
               |                                  +---------------+
               |     +-------------+
               +--->|a[3,1],a[3,2]|
                     +-------------+
```

# Vectorizing Models

Stan can calculate parts of the model more efficiently if contributions to the log likelihood are vectorized

- Multilevel model

```
for (n in 1:N)
  y[n] ~ bernoulli_logit(x[n,] * beta + alpha[g[n]]);
```

vs

```
{
  vector[N] eta;
  for (n in 1:N)
    eta[n] = x[n,] * beta + alpha[g[n]];
  y ~ bernoulli_logit(eta);
}
```

vs

```
y ~ bernoulli_logit(x * beta + alpha[g]);
```

# Part II
# Integration

# Dimension Reduction

- If a parameter can be integrated from the posterior, doing so can increase performance
- Often makes model code messier
- Parameter can be recoved by simulating from its conditional distribution in generated quantities

# Marginalization Example

Varying intercept, linear model

$$y_i \mid \alpha \sim \mathsf{Normal}(x_i^\top \beta + \alpha_{g[i]}, \sigma_y^2),$$
$$\alpha_j \sim \mathsf{Normal}(0, \sigma_\alpha^2).$$

Integrate out $\alpha$ to obtain:

$$y_i \sim \mathsf{Normal}(x_i^\top \beta, \sigma_y^2 + \sigma_\alpha^2),$$
$$\mathrm{Cov}(y_i, y_j) = \begin{cases} \sigma_\alpha^2 & \text{if } g[i] = g[j] \\ & 0 \text{ otherwise} \end{cases} \quad \text{for } i \neq j$$

Stan jumps on $K + 2$ parameters vs $K + J + 2$

# Marginalization Example pt. 2

```
data {
  int<lower = 0> N;
  int<lower = 0> K;
  int<lower = 0> N_group;

  vector[N] y;
  int<lower = 1, upper = J> group[N];
  matrix[N,K] x;
}
parameters {
  real<lower = 0> sigma_y;
  real<lower = 0> sigma_alpha;

  vector[K] beta;
}
```

# Marginalization Example pt. 3

```
transformed parmeters {
  cov_matrix[N] Sigma_y;

  Sigma_y = diag_matrix(rep_vector(sigma_y, N));
  for (row in 1:N) {
    for (col in 1:N) {
      if (group[row] == group[col]) Sigma_y[row,col] =
        Sigma_y[row,col] + sigma_alpha;
    }
  }
}
model {
  y ~ multi_normal(x * beta, Sigma_y);
  beta ~ cauchy(0, 10);
  sigma_y ~ cauchy(0, 2.5);
  sigma_alpha ~ cauchy(0, 2.5);
}
```

# Marginalization Example pt. 4

Add a generated quantities block to simulate:

$$\alpha_j \mid y \sim \text{Normal}\left(\frac{n_j/\sigma_y^2}{n_j/\sigma_y^2 + 1/\sigma_\alpha^2}\bar{y}_j, \frac{1}{n_j/\sigma_y^2 + 1/\sigma_\alpha^2}\right),$$

where $\bar{y}_j = \frac{1}{n_j}\sum_{i \in g_j} y_i$

# When to Marginalize

- When a model is slow, examine math and see if a parameter can be integrated
- Required for finite mixture models (cluster)
- Required for latent discrete parameters (change point models, noisy categorical measurements)
- Censored values
- (Not recommended in MLM)

# Exploiting Independence

```
data {
  int<lower = 0> N_in_group[N_group];
}
model {
  for (ng in 1:N_group) {
    cov_matrix[N_in_group[ng]] Sigma_y_ng;

    Sigma_y_ng =
      rep_matrix(sigma_gamma, N_in_group[ng], N_in_group[ng]);
    for (i in 1:N_in_group[ng])
      Sigma_y_ng[i,i] = Sigma_y_ng[i,i] + sigma_y;

    y[group[ng]] ~
      multi_normal(x[group[ng],] * beta, Sigma_y_ng);
  }
}
```

# Part III
# Decomposition

# Matrix Math Can Be Costly

```
data {
  int<lower = 0> N;
  int<lower = 0> K;

  vector[N] y;
  matrix[N,K] x;

  cov_matrix[N] Sigma_y;
}
parameters {
  vector[K] beta;
}
model {
  y ~ multi_normal(x * beta, Sigma_y);
}
```

# Matrix Math Can Be Costly pt. 2

- Each sample requires computing Multi-Normal$(x\beta, \Sigma_y)$

$$p(y) = (2\pi)^{-n/2}|\Sigma_y|^{-1/2} \exp\left\{-\frac{1}{2}(y - x\beta)^\top \Sigma_y^{-1}(y - x\beta)\right\}.$$

- Matrix determinants and inverses are *slow* and potentially *unstable*

# Matrix Math Can Be Costly pt. 3

- Internally, Multi-Normal$(x\beta, \Sigma_y)$ is computed by decomposing $\Sigma_y$
- Symmetric positive definite matrices behave like positive numbers ($a^\top A a > 0$), which have square roots
- Cholesky decomposition $\Sigma_y = L_y L_y^\top$, $L_y$ is lower-triangular

$$\begin{bmatrix} 4 & 12 \\ 12 & 37 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 6 & 1 \end{bmatrix} \begin{bmatrix} 2 & 6 \\ 0 & 1 \end{bmatrix}.$$

# Matrix Math Can Be Costly pt. 3

- Determinant of $L$ is product of diagonal, inverse straight-forward to compute

$$
\begin{aligned}
y &\sim \text{Multi-Normal}(x\beta, \Sigma_y), \\
y_0 &= y - x\beta, \\
y_0 &\sim \text{Multi-Normal}(0, \Sigma_y), \\
z &= L^{-1}(y - x\beta), \\
z &\sim \text{Multi-Normal}(0, \mathrm{I}_n).
\end{aligned}
$$

```
model {
  vector[N] z;
  z = L.inv * (y - x * beta);
  z ~ normal(0, 1);
}
```

# Matrix Math Can Be Costly pt. 4

```
data {
  ...
  cov_matrix[N] Sigma_y;
}
transformed data {
  cholesky_factor_cov[N] L_y;

  L_y = cholesky_decompose(Sigma_y);
}
model {
  y ~ multi_normal_cholesky(x * beta, L_y);
}
```

# Matrix Math Can Be Costly pt. 5

- Avoid for loops if possible, use vectorized constructions
- Reuse decompositions if possible
- If changing on every iteration, gain nothing by storing decomp
- Further work on efficient matrix construction - email to ask/request

  https://groups.google.com/forum/#!forum/stan-users