Structured Correlation Matrices in Stan

StanCon 2024

Sean Pinkney

sean.pinkney@gmail.com

Managing Director at Omnicom Media Group

Stan Developer

What is a correlation matrix?

- Symmetric
- 1s along the diagonal
- Off diagonal elements between [-1, 1]
- Positive semi-definite

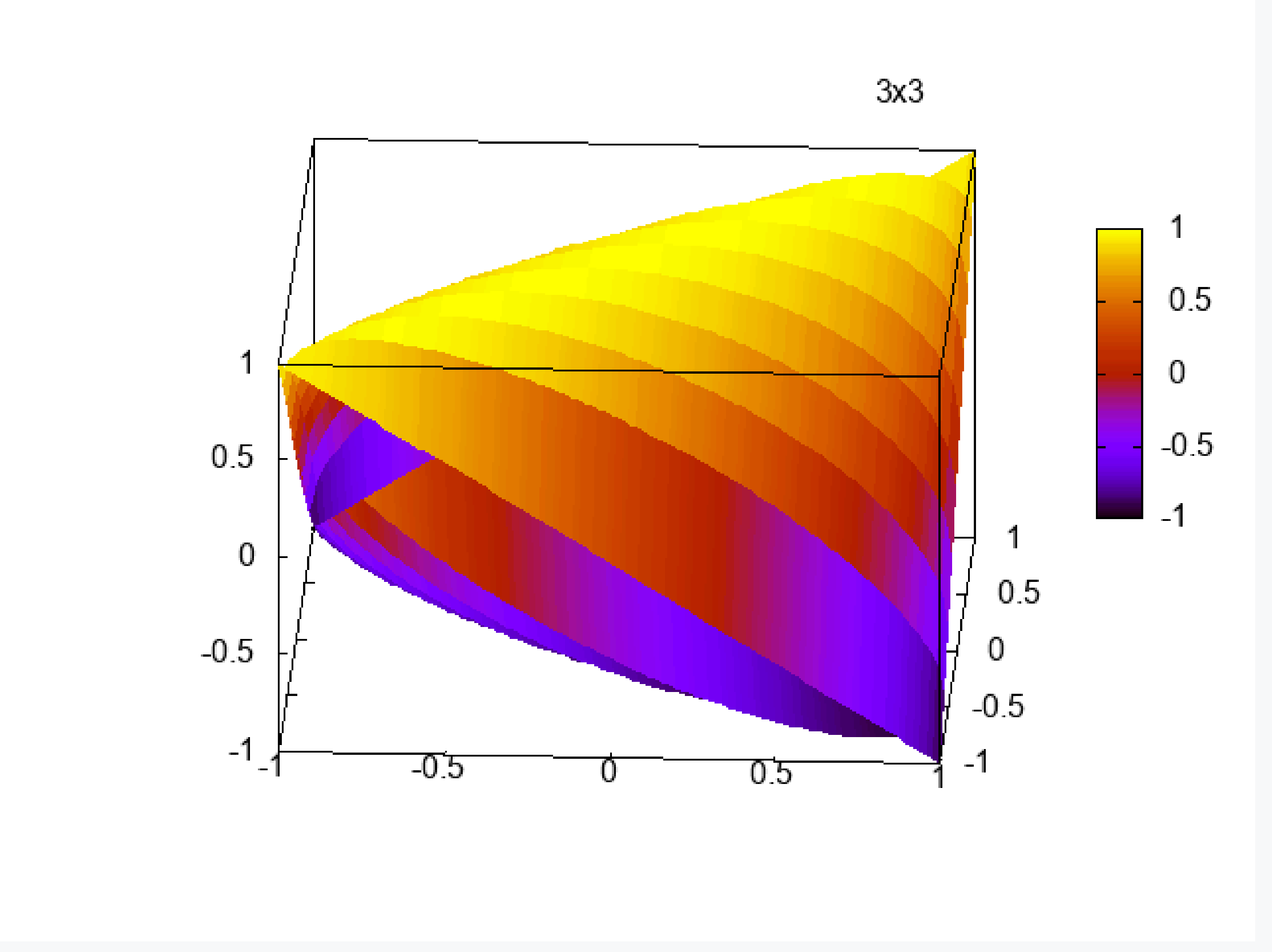
That last requirement is the tough one.

An $n \times n$ matrix Σ is p.s.d iff

$$\mathbf{x}^\top \Sigma \mathbf{x} \geq 0 \quad \forall \mathbf{x} \in \mathbb{R}^n$$

Why is positive semi-definite tough?

Things get really, really constrained as the dimension of n increases



What is a structured correlation matrix?

Anything with structure.

> “So any help you could give us would be most…helpful.” - Monthy Python

Types of structure

- Known values
- Block structure
- Bounds on correlation values

Cholesky factors

A positive definite matrix, $\mathbf{\Sigma}$, can be factored into a lower triangular matrix, $\mathbf{L}$ such that

$$\mathbf{\Sigma} = \mathbf{L}\mathbf{L}^\top$$

Bonus:

A positive definite matrix, $\mathbf{\Sigma}$, can also be factored into a strictly lower triangular matrix, $\mathbf{L}_*$ and diagonal matrix, $\mathbf{D}$ such that

$$\mathbf{\Sigma} = \mathbf{L}_*\mathbf{D}\mathbf{L}_*{}^\top$$

Stan's current Cholesky factor of correlation matrix transform

```
data {
  int<lower=0> K;
}
transformed data {
 int K_choose_2 = choose(K, 2);
}
parameters {
  // y is a vector K-choose-2 unconstrained parameters
  vector[K_choose_2] y;
}
transformed parameters {
  // L is a Cholesky factor of a K x K correlation matrix
  cholesky_factor_corr[K] L = identity_matrix(K);
  real log_det_jacobian = 0;
  {
    int counter = 1;
    real sum_sqs;
    vector[K_choose_2] z = tanh(y);
    log_det_jacobian += sum(log1m(square(z)));

    for (i in 2 : K) {
      L[i, 1] = z[counter];
      counter += 1;
      sum_sqs = square(L[i, 1]);
      for (j in 2 : (i - 1)) {
        log_det_jacobian += 0.5 * log1m(sum_sqs);
        L[i, j] = z[counter] * sqrt(1 - sum_sqs);
```

Motivating Example

Constrain the `y` "raw" parameters to be positive. This should make the correlation matrix all positive.

```
...
  vector[K_choose_2] z = inv_logit(y); // z is between 0 and 1
  log_det_jacobian += sum(log_inv_logit(y) + log1m_inv_logit(y));
...
```

Let's first see how rejection sampling handles this

Sample a bunch and if **all** are positive then keep

```julia
using ArviZ,
      Distributions,
      PrettyTables

xs = Matrix[];

while length(xs) < 10000
    X = rand(LKJ(5, 4))
    if all(≥(0), X)
        push!(xs, X)
    end
end
```

Mean should be about 0.28 and std 0.18

parameter String	mean Float64	std Float64	hdi_2.5% Float64	hdi_97.5% Float64
x[1,1]	1.0	0.0	1.0	1.0
x[2,1]	0.282	0.181	0.0	0.611
x[3,1]	0.2806	0.1783	0.0001	0.6035
x[4,1]	0.2809	0.1815	0.0	0.6098
x[5,1]	0.2801	0.1808	0.0001	0.6115
x[1,2]	0.282	0.181	0.0	0.611
x[2,2]	1.0	0.0	1.0	1.0
x[3,2]	0.2819	0.1816	0.0	0.6119
x[4,2]	0.2787	0.1792	0.0	0.6091
x[5,2]	0.2827	0.1808	0.0	0.6109
x[1,3]	0.2806	0.1783	0.0001	0.6035
x[2,3]	0.2819	0.1816	0.0	0.6119
x[3,3]	1.0	0.0	1.0	1.0
x[4,3]	0.277	0.1799	0.0001	0.6068
x[5,3]	0.2792	0.1801	0.0004	0.6065
x[1,4]	0.2809	0.1815	0.0	0.6098
x[2,4]	0.2787	0.1792	0.0	0.6091
x[3,4]	0.277	0.1799	0.0001	0.6068
x[4,4]	1.0	0.0	1.0	1.0
x[5,4]	0.2778	0.1791	0.0003	0.6003
x[1,5]	0.2801	0.1808	0.0001	0.6115
x[2,5]	0.2827	0.1808	0.0	0.6109
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Output in Stan

```
# A tibble: 25 × 10
   variable    mean median    sd   mad     q5   q95  rhat ess_bulk ess_tail
   <chr>      <dbl>  <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl>    <dbl>    <dbl>
 1 Omega[1,1] 1      1     0     0     1      1     NA         NA       NA
 2 Omega[2,1] 0.235  0.206 0.167 0.175 0.0195 0.549  1.00   44411.   20215.
 3 Omega[3,1] 0.235  0.205 0.168 0.177 0.0195 0.555  1.00   51831.   22326.
 4 Omega[4,1] 0.235  0.206 0.167 0.177 0.0200 0.555  1.00   46224.   21451.
 5 Omega[5,1] 0.235  0.206 0.166 0.175 0.0198 0.551  1.00   50078.   21105.
 6 Omega[1,2] 0.235  0.206 0.167 0.175 0.0195 0.549  1.00   44411.   20215.
 7 Omega[2,2] 1      1     0     0     1      1     NA         NA       NA
 8 Omega[3,2] 0.280  0.257 0.165 0.176 0.0510 0.587  1.00   53901.   27702.
 9 Omega[4,2] 0.280  0.258 0.167 0.178 0.0497 0.588  1.00   45097.   25476.
10 Omega[5,2] 0.280  0.258 0.166 0.175 0.0498 0.588  1.00   47192.   24449.
11 Omega[1,3] 0.235  0.205 0.168 0.177 0.0195 0.555  1.00   51831.   22326.
12 Omega[2,3] 0.280  0.257 0.165 0.176 0.0510 0.587  1.00   53901.   27702.
13 Omega[3,3] 1      1     0     0     1      1     NA         NA       NA
14 Omega[4,3] 0.325  0.307 0.165 0.176 0.0870 0.624  1.00   51438.   29513.
15 Omega[5,3] 0.324  0.307 0.164 0.175 0.0870 0.621  1.00   48371.   28163.
16 Omega[1,4] 0.235  0.206 0.167 0.177 0.0200 0.555  1.00   46224.   21451.
17 Omega[2,4] 0.280  0.258 0.167 0.178 0.0497 0.588  1.00   45097.   25476.
18 Omega[3,4] 0.325  0.307 0.165 0.176 0.0870 0.624  1.00   51438.   29513.
19 Omega[4,4] 1      1     0     0     1      1     NA         NA       NA
20 Omega[5,4] 0.368  0.355 0.163 0.175 0.122  0.657  1.00   50304.   30427.
21 Omega[1,5] 0.235  0.206 0.166 0.175 0.0198 0.551  1.00   50078.   21105.
22 Omega[2,5] 0.280  0.258 0.166 0.175 0.0498 0.588  1.00   47192.   24449.
23 Omega[3,5] 0.324  0.307 0.164 0.175 0.0870 0.621  1.00   48371.   28163.
24 Omega[4,5] 0.368  0.355 0.163 0.175 0.122  0.657  1.00   50304.   30427.
```

Wtf?

???

A foray into Cholesky factors

Andre Cholesky 

Cholesky factors

The value $C_{i,j} \in \mathbf{C}$ is a correlation value that is between $-1, 1$

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ C_{2,1} & \sqrt{1 - L_{2,1}^2} & 0 & \cdots & 0 \\ C_{3,1} & \left(C_{3,2} - L_{3,1}L_{2,1}\right)/L_{2,2} & \sqrt{1 - L_{3,1}^2 - L_{3,2}^2} & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ C_{n,1} & \left(C_{n,2} - L_{n,1}L_{2,1}\right)/L_{2,2} & \left(C_{n,3} - L_{n,1}L_{2,1} - L_{n,2}L_{3,2}\right)/L_{3,3} & \cdots & \sqrt{1 - \sum_{k=1}^{n-1} L_{n,k}^2} \end{pmatrix}$$

Let's try something

Let's condense it into an equation

$$L_{j,j} = \sqrt{1 - \sum_{k=1}^{j-1} L_{j,k}^2}$$

$$L_{i,j} = \frac{1}{L_{j,j}} \left(C_{i,j} - \sum_{k=1}^{j-1} L_{i,k} L_{j,k} \right) \quad \text{for } i > j$$

Re-arrange and add in user specified **lower** and **upper** bounds

$$-L_{j,j} + \sum_{k=1}^{j-1} L_{i,k} L_{j,k} \leq \boldsymbol{a}_{\boldsymbol{i},\boldsymbol{j}} < C_{i,j} < \boldsymbol{b}_{\boldsymbol{i},\boldsymbol{j}} \leq L_{j,j} + \sum_{k=1}^{j-1} L_{i,k} L_{j,k}$$

Continuing

$$-\sqrt{1-\sum_{k=1}^{j-1} L_{i,k}^2} \le \frac{\boldsymbol{a}_{\boldsymbol{i,j}} - \sum_{k=1}^{j-1} L_{i,k}L_{j,k}}{L_{j,j}} < L_{i,j} < \frac{\boldsymbol{b}_{\boldsymbol{i,j}} - \sum_{k=1}^{j-1} L_{i,k}L_{j,k}}{L_{j,j}} \le \sqrt{1-\sum_{k=1}^{j-1} L_{i,k}^2}$$

$$\max\left\{-\sqrt{1-\sum_{k=1}^{j-1} L_{j,k}^2},\ \frac{\boldsymbol{a}_{\boldsymbol{i,j}} - \sum_{k=1}^{j-1} L_{i,k}L_{j,k}}{L_{j,j}}\right\} < L_{i,j} < \min\left\{\sqrt{1-\sum_{k=1}^{j-1} L_{j,k}^2},\ \frac{\boldsymbol{b}_{\boldsymbol{i,j}} - \sum_{k=1}^{j-1} L_{i,k}L_{j,k}}{L_{j,j}}\right\}$$

This gives us bounds but we need to be really careful

The Jacobian is "easy"

Extra Care

Assume we have 3×3 matrix that we wish to constrain all correlation values to be negative.

Let us choose $C_{2,1} = C_{3,1} = \frac{-1}{\sqrt{2}}$ and solve for the bounds of $C_{3,2}$:

$$
\begin{gathered}
\max\left\{-1, -\sqrt{1 - C_{3,1}^2}\right\} < \frac{B - C_{2,1}C_{3,1}}{L_{2,2}} < \min\left\{0, \sqrt{1 - C_{3,1}^2}\right\} \\
-\sqrt{0.5} < \frac{B - 0.5}{\sqrt{0.5}} < 0 \\
\implies 0 < B < 0.5
\end{gathered}
$$

Note

This is all in Pinkney (2024)

The Stan code

```
1 matrix cholesky_corr_constrain_lp (vector col_one_raw, vector off_raw,
2                                     real lb, real ub) {
3     int K = num_elements(col_one_raw) + 1;
4     matrix[K, K] L = rep_matrix(0, K, K);
5     L[1, 1] = 1;
6     L[2, 1] = lb_ub_lp(col_one_raw[1], lb, ub);
7     L[2, 2] = sqrt(1 - L[2, 1]^2);
8 
9     int cnt = 1;
10 
11     for (i in 3:K) {
12        L[i, 1] = lb_ub_lp(col_one_raw[2:K - 1], lb,  ub);
13        real l_ij_old = log1m(L[i, 1]^2);
14       for (j in 2:i - 1) {
15          real b1 = dot_product(L[j, 1:(j - 1)], L[i, 1:(j - 1)]);
16          real stick_length = exp(0.5 * l_ij_old);
17          real low = max({-stick_length, (lb - b1) / L[j, j]});
18          real up = min({stick_length, (ub - b1) /  L[j, j] });
19          L[i, j] = lb_ub_lp(off_raw[cnt], low, up);
20          l_ij_old = log_diff_exp(l_ij_old, 2 * log(abs(L[i, j])));
21          cnt += 1;
22         }
23          L[i, i] = exp(0.5 * l_ij_old);
24       }
25     return L;
26   }
```

This is a bit more stable than what's in the paper but also more opaque.

The exp()'s here are just calculating stuff on the log-scale.

Line 13 I take the log and try to stick with this as much as possible.

Back to the motivating example

The new method that gives the right answer!

```
# A tibble: 25 × 10
  variable     mean median    sd   mad     q5   q95  rhat ess_bulk ess_tail
  <chr>       <dbl>  <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl>    <dbl>    <dbl>
1 Omega[1,1] 1      1      0     0     1      1     NA         NA       NA
2 Omega[2,1] 0.278  0.256 0.179 0.198 0.0281 0.607  1.00   29806.   17895.
3 Omega[3,1] 0.279  0.258 0.180 0.200 0.0273 0.609  1.00   34574.   18758.
4 Omega[4,1] 0.279  0.258 0.180 0.199 0.0276 0.607  1.00   29344.   16380.
5 Omega[5,1] 0.281  0.259 0.181 0.201 0.0291 0.610  1.00   36064.   18496.
6 Omega[1,2] 0.278  0.256 0.179 0.198 0.0281 0.607  1.00   29806.   17895.
7 Omega[2,2] 1      1      0     0     1      1     NA         NA       NA
8 Omega[3,2] 0.279  0.258 0.179 0.200 0.0283 0.605  1.00   37118.   21150.
...
```

What else can this do?

- Known values
- Specific bounds on certain values
- Block structure

Let's do a really motivating example

An even more motivating example

We want a 6 x 6 correlation matrix Σ where

$$\sigma_{2,1} = 0, \, \sigma_{4,3} = 0$$

$$\sigma_{4,1} \in (-0.8, 0), \, \sigma_{5,3} \in (0.3, 0.7)$$

```
--
 0.       --
NA        NA   --
(-0.8, 0) NA    0          --
NA        NA   (0.3, 0.7) NA   --
NA        NA   NA         NA   NA   --
```

Output

variable	mean	median	sd	mad	q5	q95	rhat	ess_bulk	ess_tail
<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
Omega[1,1]	1.0000	1.0000	0.0000	0.0000	1.0000	1.0000	NA	NA	NA
Omega[2,1]	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	NA	NA	NA
Omega[3,1]	-0.0056	-0.0101	0.2484	0.2559	-0.4211	0.4068	0.9999	4410.0862	3220.8831
Omega[4,1]	**-0.2124**	**-0.1881**	**0.1512**	**0.1579**	**-0.5044**	**-0.0173**	**1.0003**	**4729.1236**	**2149.4316**
Omega[5,1]	-0.0057	-0.0065	0.2621	0.2772	-0.4435	0.4217	1.0000	4657.2520	3017.0624
Omega[6,1]	-0.0018	-0.0002	0.2553	0.2599	-0.4284	0.4143	1.0013	4674.8017	2903.7293
Omega[2,2]	1.0000	1.0000	0.0000	0.0000	1.0000	1.0000	NA	NA	NA
Omega[3,2]	-0.0043	-0.0078	0.2499	0.2636	-0.4178	0.4093	1.0001	4023.8517	2705.3045
Omega[4,2]	0.0043	0.0027	0.2594	0.2767	-0.4159	0.4328	1.0007	5443.1496	3098.8828
Omega[5,2]	0.0014	0.0093	0.2575	0.2682	-0.4298	0.4201	1.0017	4128.4046	2987.4422
Omega[6,2]	0.0058	0.0064	0.2538	0.2693	-0.4197	0.4224	1.0006	4736.2346	2668.6795
Omega[3,3]	1.0000	1.0000	0.0000	0.0000	1.0000	1.0000	NA	NA	NA
Omega[4,3]	**-0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**1.0014**	**4280.0269**	NA
Omega[5,3]	**0.4128**	**0.3934**	**0.0882**	**0.0884**	**0.3072**	**0.5888**	**1.0012**	**4809.9923**	**2377.6445**
Omega[6,3]	-0.0032	-0.0064	0.2624	0.2747	-0.4394	0.4265	1.0022	4673.5773	3037.9778
Omega[4,4]	1.0000	1.0000	0.0000	0.0000	1.0000	1.0000	NA	NA	NA
Omega[5,4]	0.0024	0.0020	0.2371	0.2475	-0.3873	0.3970	1.0002	3895.9938	2933.7159
Omega[6,4]	-0.0004	-0.0033	0.2576	0.2659	-0.4183	0.4327	1.0019	4303.5343	2949.4916
Omega[5,5]	1.0000	1.0000	0.0000	0.0000	1.0000	1.0000	NA	NA	NA
Omega[6,5]	0.0009	-0.0018	0.2533	0.2675	-0.4113	0.4213	1.0002	4139.6808	2995.0615

Do you want to see the Stan code for this?

```
matrix cholesky_corr_constrain_lp(int K, vector raw, int N_blocks,
                                  array[,] int res_index,
                                  array[] int res_id,
                                  array[,] int known_index,
                                  vector known_vals,
                                  vector lb, vector ub) {
  matrix[K, K] L = rep_matrix(0, K, K);
  int cnt = 1;
  int N_res = num_elements(res_id);
  int N_known = size(known_vals);
  vector[N_blocks] x_cache;
  array[N_blocks] int res_id_cnt = ones_int_array(N_blocks);
  int res_row = 1;
  int known_row = 1;

  L[1, 1] = 1;
  if (known_index[known_row, 1] == 2) {
    L[2, 1] = known_vals[1];
    known_row += 1;
  } else {
    L[2, 1] = lb_ub_lp(raw[cnt], lb[cnt], ub[cnt]);
    if (res_index[res_row, 1] == 2) {
      x_cache[res_id[res_row]] = L[2, 1];
      res_id_cnt[res_id[res_row]] += 1;
      res_row += 1;
    }
    cnt += 1;
```

The Future

LDL parameterization is done. It's even more stable.

Look-ahead given bound constraints and known values to ensure the impossible bounds don't hit

Explore new priors for correlation matrices that exploit structure

References:

Pinkney, Sean. 2024. "A Short Note on a Flexible Cholesky Parameterization of Correlation Matrices." https://arxiv.org/abs/2405.07286.