Prior Models For Positive Spaces

Michael Betancourt

August 2025

Table of contents

In most applications engineering a productive containment prior is relatively straightforward. Given a lower threshold l and upper threshold u we just need to find a unimodal prior model that allocates sufficient prior probability to the interval $[l, u]$. Usually we don't have to be too precious about the precise shape of this prior model, including the behavior of the tails outside of the thresholds.

That said in some applications the precise tail behavior can be important. For example a prior model with excessively heavy tails can generate samples at extreme behaviors that, while rare, cause implementation issues.

When working on a real line we have plenty of well-behaved families of probability density functions that awkward tails are seldom if ever a concern. Developing well-behaved containment prior models on constrained spaces like the positive real line, however, can be more challenging. The challenge is especially severe if either of the thresholds are close to the boundaries of the constrained space.

In this short note we'll consider a variety of prior model choices over a positive real line, comparing and contrasting both their overall shapes and their lower and upper tail behaviors. Note that this is not the first time that I've explored this topic.

1 Setup

First we'll need to setup our local `R` environment. In particular we'll need to load `RStan` so that we can use Stan's `algebra_solver` function to tune more sophisticated prior models to specific containment criteria.

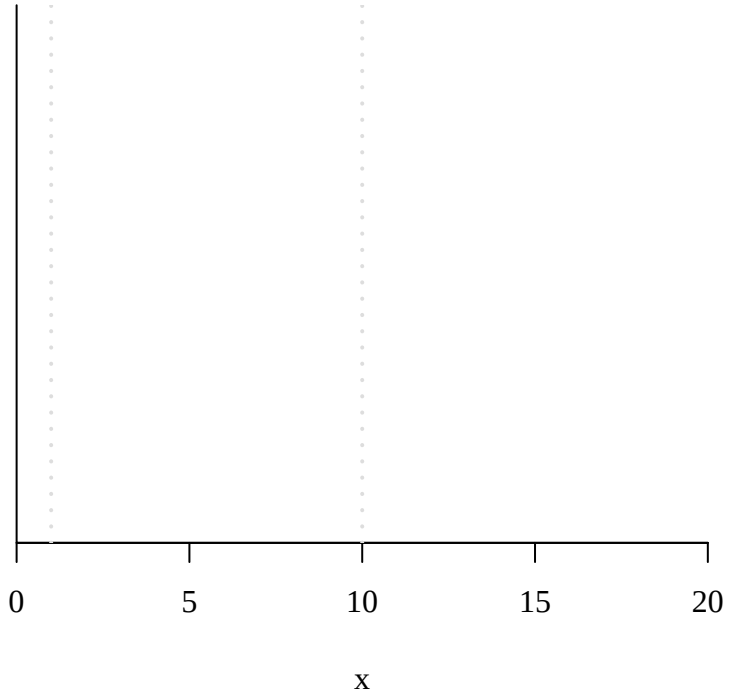
```
par(family="serif", las=1, bty="l",
    cex.axis=1, cex.lab=1, cex.main=1,
    xaxs="i", yaxs="i", mar = c(5, 5, 3, 1))

library(colormap)

nom_colors <- c("#DCBCBC", "#C79999", "#B97C7C",
                "#A25050", "#8F2727", "#7C0000")
lcs <- rev(colormap(colormap=nom_colors, nshades=6))

library(rstan)
rstan_options(auto_write = TRUE)            # Cache compiled Stan programs
options(mc.cores = parallel::detectCores()) # Parallelize chains
parallel:::setDefaultClusterOptions(setup_strategy = "sequential")
```

2 Containment Thresholds

For this brief study we'll consider a lower threshold l that is relatively close to zero. The short distance between the lower threshold and the left boundary of the positive real line will make engineering containment more burdensome.

```
l <- 1
u <- 10
```

```
par(mfrow=c(1, 1), mar=c(5, 1, 1, 1))

plot(0, type='n',
     xlab='x', xlim=c(0, 20),
     yaxt='n', ylab="", ylim=c(0, 2))

abline(v=l, lwd=2, lty=3, col="#DDDDDD")
abline(v=u, lwd=2, lty=3, col="#DDDDDD")
```



3 Standard Models

Let's start by considering three families of probability density functions over a positive real line that are available in most statistical software. For more details of the log normal, gamma, and inverse gamma family of probability density functions see this chapter.

3.1 Log Normal

The log normal model is given by pushing forward a normal family of probability density functions through an exponential function

$$x = \exp(y).$$

One nice feature of this pushforward construction is that, because the exponential function is monotonic, we can engineer containment on the unconstrained input space or the positive output space. Specifically if we can engineer a normal model that allocates probability p to the interval

$$l \le x \le u$$

then the corresponding log normal model will allocate probability p to the interval

$$\log(l) \le y = \log(x) \le \log(u).$$

Conveniently engineering a normal model with desired interval containment is relatively straightforward: we just set the location parameter to the center of the containment thresholds and then set the scale to ensure the desired containment.

```
ll <- log(l)
lu <- log(u)
```

For this note we'll consider 98% containment probability, with 1% extending below the lower threshold and 1% extending above the upper threshold.

```
mu <- 0.5 * (lu + ll)
sigma <- 0.5 * (lu - ll) / 2.32
```

```
xs <- seq(0.01, 20, 0.01)
N <- length(xs)

ln_ds <- dnorm(log(xs), mu, sigma) / xs
```

3.2 Gamma

Satisfying a given containment criterion is more difficult for the gamma family of probability density functions. Here we'll take advantage of Stan's algebraic solver to tune an appropriate gamma model numerically.

```
tune <- stan(file='stan_programs/gamma_tune.stan',
             data=list("l" = l, "u" = u, "delta"=1),
             iter=1, warmup=0, chains=1,
             seed=4838282, algorithm="Fixed_param")
```

```
alpha = 4.62909
beta = 1.10337

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                0 seconds (Sampling)
Chain 1:                0 seconds (Total)
Chain 1:
```

```
alpha <- extract(tune)$alpha
beta <- extract(tune)$beta
```

```
gm_ds <- sapply(xs, function(x) dgamma(x, alpha, beta))
```

3.3 Inverse Gamma

Numerical tuning is also the most productive approach for tuning an inverse gamma prior model.

```
tune <- stan(file='stan_programs/inv_gamma_tune.stan',
             data=list("l" = l, "u" = u, "delta"=1),
             iter=1, warmup=0, chains=1,
             seed=4838282, algorithm="Fixed_param")
```

```
alpha = 4.62909
beta = 11.0337

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                0 seconds (Sampling)
Chain 1:                0 seconds (Total)
Chain 1:
```

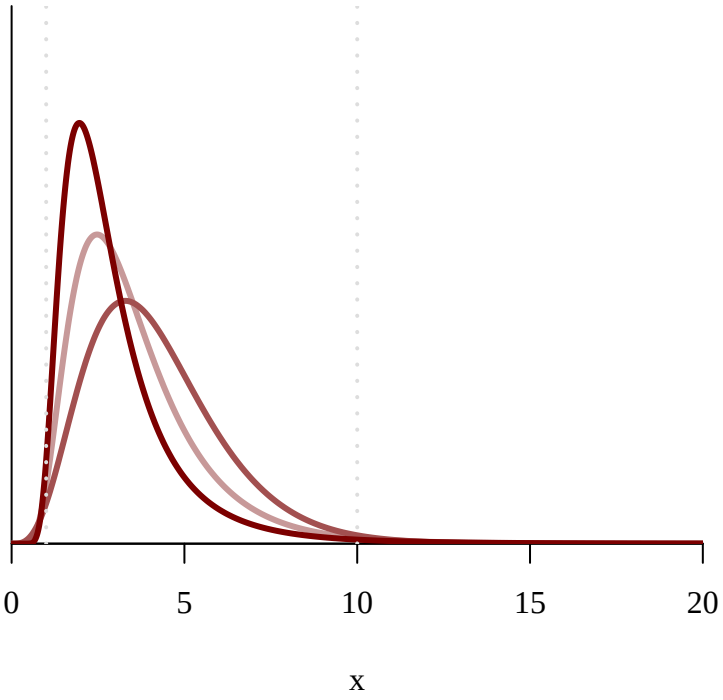
```
alpha <- extract(tune)$alpha
beta <- extract(tune)$beta
```

```
igm_ds <- sapply(xs,
                 function(x) dgamma(1 / x, alpha, beta) * (1 / x**2))
```

3.4 Comparison

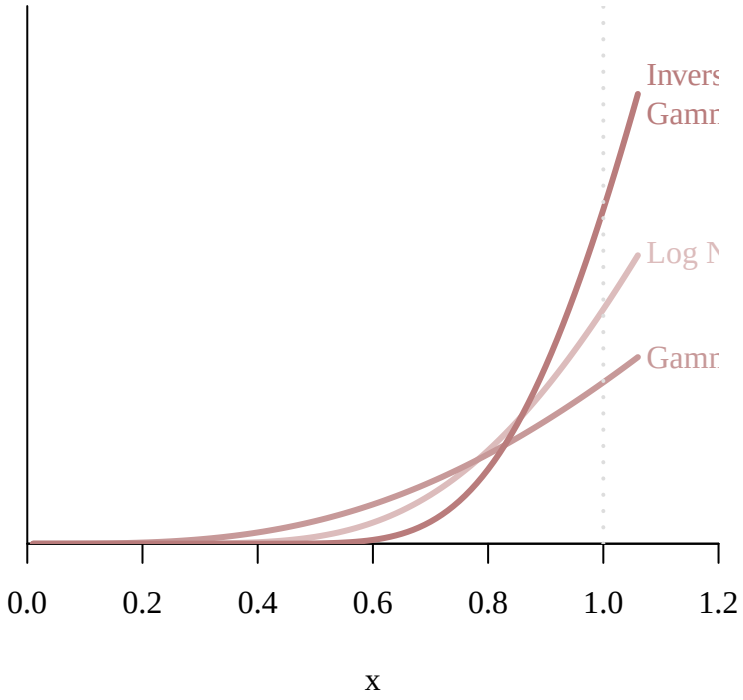
Interestingly these three models fall into a natural organization. The gamma model is the least skewed, the inverse gamma is the most skewed, and the log normal model falls somewhere in between.

```
par(mfrow=c(1, 1), mar=c(5, 1, 1, 1))

plot(0, type='n',
     xlab='x', xlim=c(0, 20),
     yaxt='n', ylab="", ylim=c(0, 0.5))

lines(xs, ln_ds, lwd=3, col=lcs[2])
lines(xs, gm_ds, lwd=3, col=lcs[4])
lines(xs, igm_ds, lwd=3, col=lcs[6])

abline(v=l, lwd=2, lty=3, col="#DDDDDD")
abline(v=u, lwd=2, lty=3, col="#DDDDDD")
```



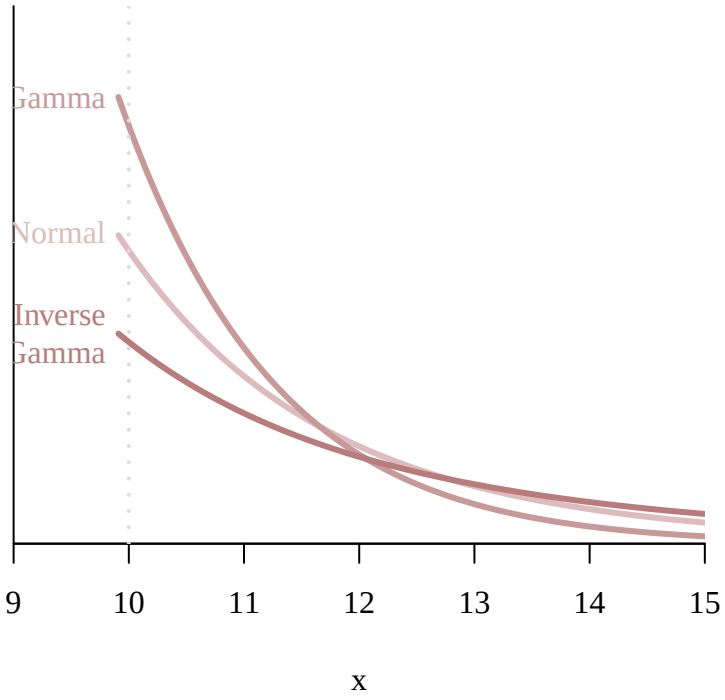
The more extreme the skew the model the more quickly the left tail decays. Consequently the gamma model features the heaviest left tail, and hence the weakest containment in that direction. The inverse gamma model exhibits the lightest left tail, and the log normal model falls in between.

```
par(mfrow=c(1, 1), mar=c(5, 1, 1, 1))

plot(0, type='n',
     xlab='x', xlim=c(0, 1.2),
     yaxt='n', ylab="", ylim=c(0, 0.125))

N_lt <- which(xs > 1.05)[1]

lines(xs[1:N_lt], ln_ds[1:N_lt],  lwd=3, col=lcs[1])
lines(xs[1:N_lt], gm_ds[1:N_lt],  lwd=3, col=lcs[2])
lines(xs[1:N_lt], igm_ds[1:N_lt], lwd=3, col=lcs[3])

abline(v=l, lwd=2, lty=3, col="#DDDDDD")

text(1.075, ln_ds[N_lt],  adj=0, labels='Log Normal', col=lcs[1])
text(1.075, gm_ds[N_lt],  adj=0, labels='Gamma', col=lcs[2])
text(1.075, igm_ds[N_lt], adj=0, labels='Inverse\nGamma', col=lcs[3])
```



On the other hand a larger skew also increases the weight of the right tail. The gamma model features the lightest right tail, and hence the strongest containment in that direction. On the other hand the inverse gamma model exhibits the heaviest right tail and the log normal model once again falls in between.

```
par(mfrow=c(1, 1), mar=c(5, 1, 1, 1))

plot(0, type='n',
     xlab='x', xlim=c(9, 15),
     yaxt='n', ylab="", ylim=c(0, 0.01))

N_rt <- which(xs > 9.9)[1]

lines(xs[N_rt:N], ln_ds[N_rt:N],  lwd=3, col=lcs[1])
lines(xs[N_rt:N], gm_ds[N_rt:N],  lwd=3, col=lcs[2])
lines(xs[N_rt:N], igm_ds[N_rt:N], lwd=3, col=lcs[3])

abline(v=u, lwd=2, lty=3, col="#DDDDDD")

text(9.8, ln_ds[N_rt],  adj=1, labels='Log Normal', col=lcs[1])
text(9.8, gm_ds[N_rt],  adj=1, labels='Gamma', col=lcs[2])
text(9.8, igm_ds[N_rt], adj=1, labels='Inverse\nGamma', col=lcs[3])
```



Overall the log normal model features the most balanced tails while the gamma and inverse gamma models tip that balance in different directions. In particular the gamma model is more useful when suppressing larger values is more important than suppressing smaller values, and the inverse gamma model is more useful in the opposite circumstances.

4 Platykurtic Log Models

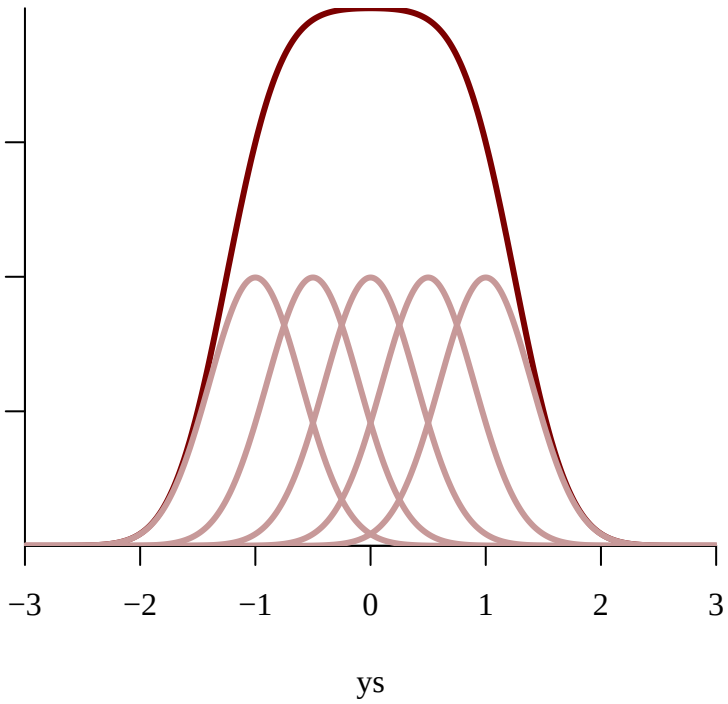
If the containment of the log normal, gamma, and inverse gamma models is not sufficient then we need to engineer new models with even lighter tails. One way of approaching this problem is to follow the construction of the log normal model, only considering lighter-tailed, or more **platykurtic**, models over a real line that are then pushed forward along an exponential function.

4.1 Log Mixture of Normals

One way to engineer platykurtic behavior on a real line is by mixing together multiple normal models with common scales but varying locations. With care in the width that common scale relative to the spacing of the component locations this mixture will result in a smooth probability density function with light tails in both directions.

```
mu <- 0
sigma <- 1
K <- 5

nus <- mu + (2 * (1:K) - 1 - K) / (K - 1) * sigma
tau <- 2 * sigma / K

ys <- seq(-3, 3, 0.01)
mn_ds <- lapply(nus, function(nu) (1 / K) * dnorm(ys, nu, tau))
```

```
par(mfrow=c(1, 1), mar=c(5, 1, 1, 1))

plot(ys, Reduce(f ="+", x=mn_ds),
     type="l", lwd=3, col=lcs[6])

for (d in mn_ds) lines(ys, d, lwd=3, col=lcs[2])
```



Now we can pushforward this probabilistic model along an exponential function to derive a candidate prior model over a real line.

```
dmn <- function(x, mu, sigma, K) {
  nus <- mu + sigma * (2 * (1:K) - 1 - K) / (K - 1)
  tau <- 2 * sigma / K
  sum(sapply(nus, function(nu) (1 / K) * dnorm(x, nu, tau)))
}

# dlmn(x) = dmn(log(x), mu, sigma, K) / x)
```

When tuning this model to achieve a given containment criterion we can work on either the input real line or the output positive real line. In particular, as in the log normal case, we can map containment thresholds on a positive real line to log containment thresholds on a full real line and then tune an appropriate mixture of normals model.

Because this specific mixture of normals is always symmetric, its location configuration μ will always be in between the two containment thresholds.

```
mu <- 0.5 * (lu + ll)
```

This symmetry also implies that the lower and upper thresholds will be the same distance from μ so we can determine the scale σ from either lower or upper threshold.

Let's investigate the behavior of the appropriately tuned log mixture of normals models for a sequence of components, K.

```
K <- 4

tune <- stan(file='stan_programs/platy_mixture_tune.stan',
             data=list("u" = lu, "mu" = mu, "K" = K),
             iter=1, warmup=0, chains=1,
             seed=4838282, algorithm="Fixed_param")
```

```
sigma = 0.612023

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                0 seconds (Sampling)
Chain 1:                0 seconds (Total)
Chain 1:
```

```
sigma4 <- extract(tune)$sigma
lmn4_ds <- sapply(xs, function(x) dmn(log(x), mu, sigma4, K) / x)
```

```
K <- 6

tune <- stan(file='stan_programs/platy_mixture_tune.stan',
             data=list("u" = lu, "mu" = mu, "K" = K),
             iter=1, warmup=0, chains=1,
             seed=4838282, algorithm="Fixed_param")
```

```
sigma = 0.754381

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                0 seconds (Sampling)
Chain 1:                0 seconds (Total)
Chain 1:
```

```
sigma6 <- extract(tune)$sigma
lmn6_ds <- sapply(xs, function(x) dmn(log(x), mu, sigma6, K) / x)
```

```
K <- 8

tune <- stan(file='stan_programs/platy_mixture_tune.stan',
             data=list("u" = lu, "mu" = mu, "K" = K),
             iter=1, warmup=0, chains=1,
             seed=4838282, algorithm="Fixed_param")
```

```
sigma = 0.846607

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                0 seconds (Sampling)
Chain 1:                0 seconds (Total)
Chain 1:
```

```
sigma8 <- extract(tune)$sigma
lmn8_ds <- sapply(xs, function(x) dmn(log(x), mu, sigma8, K) / x)
```

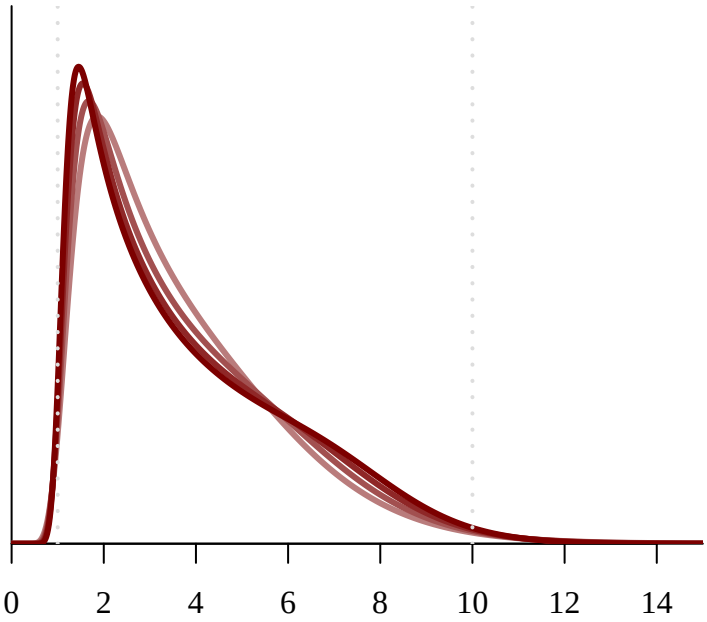
```
K <- 10

tune <- stan(file='stan_programs/platy_mixture_tune.stan',
             data=list("u" = lu, "mu" = mu, "K" = K),
             iter=1, warmup=0, chains=1,
             seed=4838282, algorithm="Fixed_param")
```

```
sigma = 0.909955

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                0 seconds (Sampling)
Chain 1:                0 seconds (Total)
Chain 1:
```

```
sigma10 <- extract(tune)$sigma
lmn10_ds <- sapply(xs, function(x) dmn(log(x), mu, sigma10, K) / x)
```

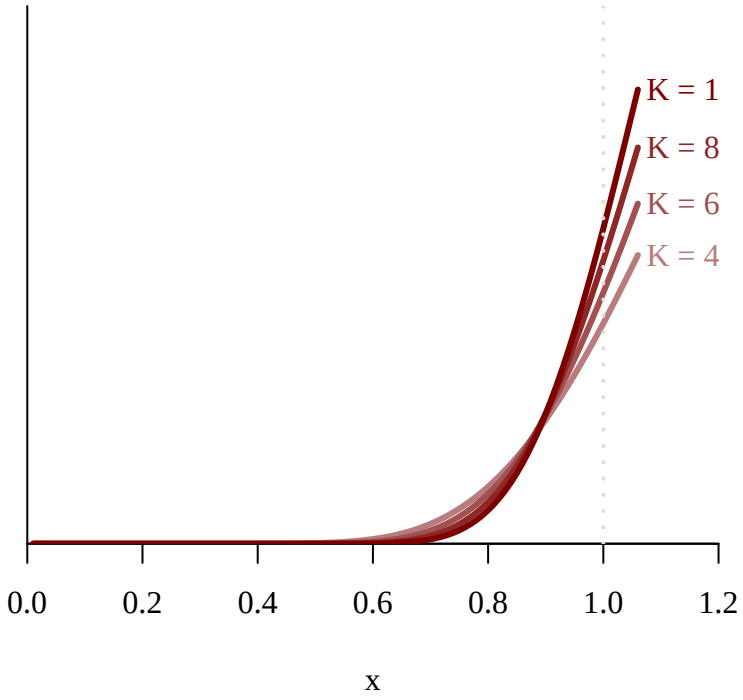
As the number of components in the latent mixture increases the induced probability density function becomes more and more skewed towards smaller values, mimicking the trend we saw above. In this case, however, the skew becomes so large that we also see a "shoulder" arising as we approach the upper containment threshold.

```
par(mfrow=c(1, 1), mar=c(5, 1, 1, 1))

plot(0, type='n',
     xlab='x', xlim=c(0, 15),
     yaxt='n', ylab="", ylim=c(0, 0.35))

lines(xs, lmn4_ds,  lwd=3, col=lcs[3])
lines(xs, lmn6_ds,  lwd=3, col=lcs[4])
lines(xs, lmn8_ds,  lwd=3, col=lcs[5])
lines(xs, lmn10_ds, lwd=3, col=lcs[6])

abline(v=l, lwd=2, lty=3, col="#DDDDDD")
abline(v=u, lwd=2, lty=3, col="#DDDDDD")
```



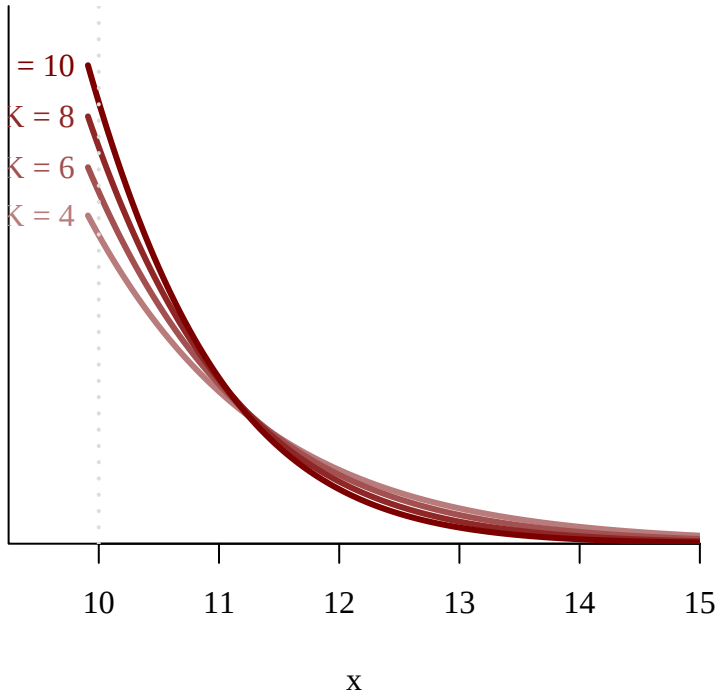
If we zoom into the left containment threshold then we can see that the left tails become lighter and lighter as we increase the number of components and the latent mixture of normals becomes more and more platykurtic.

```
par(mfrow=c(1, 1), mar=c(5, 1, 1, 1))

plot(0, type='n',
     xlab='x', xlim=c(0, 1.2),
     yaxt='n', ylab="", ylim=c(0, 0.175))

lines(xs[1:N_lt], lmn4_ds[1:N_lt],  lwd=3, col=lcs[3])
lines(xs[1:N_lt], lmn6_ds[1:N_lt],  lwd=3, col=lcs[4])
lines(xs[1:N_lt], lmn8_ds[1:N_lt],  lwd=3, col=lcs[5])
lines(xs[1:N_lt], lmn10_ds[1:N_lt], lwd=3, col=lcs[6])

abline(v=l, lwd=2, lty=3, col="#DDDDDD")

text(1.075, lmn4_ds[N_lt],  adj=0, labels='K = 4',  col=lcs[3])
text(1.075, lmn6_ds[N_lt],  adj=0, labels='K = 6',  col=lcs[4])
text(1.075, lmn8_ds[N_lt],  adj=0, labels='K = 8',  col=lcs[5])
text(1.075, lmn10_ds[N_lt], adj=0, labels='K = 10', col=lcs[6])
```



In this case, however, the lighter left tails don't come at the expense of heavier right tails. Instead the right tails also become lighter as we add more components to the latent model.

```
par(mfrow=c(1, 1), mar=c(5, 1, 1, 1))

plot(0, type='n',
     xlab='x', xlim=c(9.25, 15),
     yaxt='n', ylab="", ylim=c(0, 0.0125))

lines(xs[N_rt:N], lmn4_ds[N_rt:N],  lwd=3, col=lcs[3])
lines(xs[N_rt:N], lmn6_ds[N_rt:N],  lwd=3, col=lcs[4])
lines(xs[N_rt:N], lmn8_ds[N_rt:N],  lwd=3, col=lcs[5])
lines(xs[N_rt:N], lmn10_ds[N_rt:N], lwd=3, col=lcs[6])

abline(v=u, lwd=2, lty=3, col="#DDDDDD")

text(9.8, lmn4_ds[N_rt],  adj=1, labels='K = 4',  col=lcs[3])
text(9.8, lmn6_ds[N_rt],  adj=1, labels='K = 6',  col=lcs[4])
text(9.8, lmn8_ds[N_rt],  adj=1, labels='K = 8',  col=lcs[5])
text(9.8, lmn10_ds[N_rt], adj=1, labels='K = 10', col=lcs[6])
```



4.2 Log Generalized Normal

There are many ways to derive more platykurtic behavior from a normal model. The symmetric generalized normal model expands beyond a normal model by modifying the power in the exponent, and updating the normalization according.

```
dgnorm <- function(x, mu, alpha, beta) {
  C <- alpha**(1 / beta)
  (beta * C) / (2 * gamma(1 / beta)) * exp( -alpha * abs(x - mu)**beta )
}
```

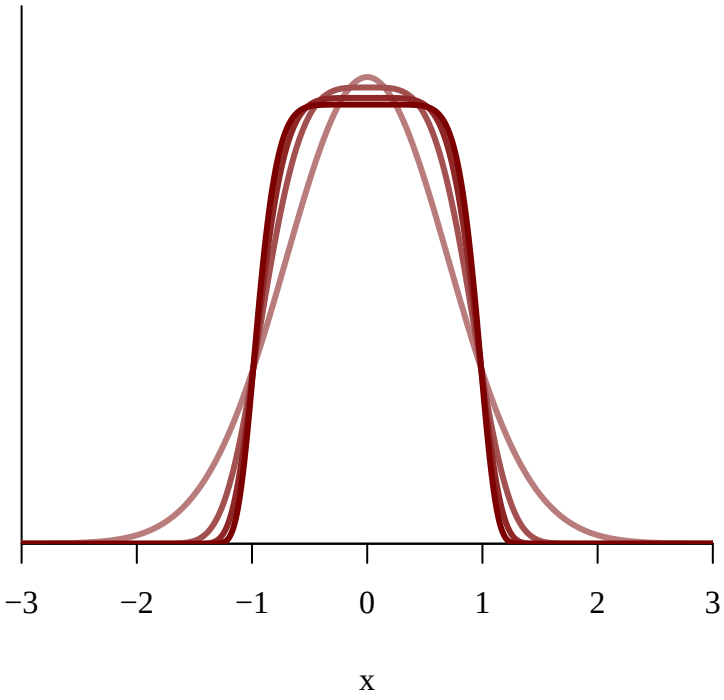
This generalized model includes the normal family as a special case when the exponent β equals two and the the Laplace model when $\beta = 1$. As β increases above two the probability density functions become more and more platykurtic.

```
mu <- 0
sigma <- 1
```

```
ys <- seq(-3, 3, 0.01)

dgn2_ds <- lapply(ys, function(y) dgnorm(y, mu, sigma, 2))
```

```
dgn4_ds <- lapply(ys, function(y) dgnorm(y, mu, sigma, 4))
dgn6_ds <- lapply(ys, function(y) dgnorm(y, mu, sigma, 6))
dgn8_ds <- lapply(ys, function(y) dgnorm(y, mu, sigma, 8))
```

```
par(mfrow=c(1, 1), mar=c(5, 1, 1, 1))

plot(0, type='n',
     xlab='x', xlim=c(-3, 3),
     yaxt='n', ylab="", ylim=c(0, 0.65))

lines(ys, dgn2_ds,  lwd=3, col=lcs[3])
lines(ys, dgn4_ds,  lwd=3, col=lcs[4])
lines(ys, dgn6_ds,  lwd=3, col=lcs[5])
lines(ys, dgn8_ds, lwd=3, col=lcs[6])
```



We can then push the symmetric generalized normal model forward along an exponential function to derive yet another possible model over a positive real line.

As with the mixture of normals model the symmetric generalized normal model is, well, symmetric. This allows us to immediately determine an appropriate location from the log containment thresholds.

```
mu <- 0.5 * (lu + ll)
```

Similarly the scale σ is completely determined by the distance between the μ and either the lower and upper log thresholds. Here we'll use the upper log threshold.

```
beta <- 2

tune <- stan(file='stan_programs/gen_norm_tune.stan',
             data=list("u" = lu, "mu" = mu, "beta" = beta),
             iter=1, warmup=0, chains=1,
             seed=4838282, algorithm="Fixed_param")
```

```
alpha = 2.04149

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                0 seconds (Sampling)
Chain 1:                0 seconds (Total)
Chain 1:
```

```
alpha2 <- extract(tune)$alpha
lgn2_ds <- sapply(xs, function(x) dgnorm(log(x), mu, alpha2, beta) / x)
```

```
beta <- 4

tune <- stan(file='stan_programs/gen_norm_tune.stan',
             data=list("u" = lu, "mu" = mu, "beta" = beta),
             iter=1, warmup=0, chains=1,
             seed=4838282, algorithm="Fixed_param")
```

```
alpha = 1.07407

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                0 seconds (Sampling)
Chain 1:                0 seconds (Total)
Chain 1:
```

```
alpha4 <- extract(tune)$alpha
lgn4_ds <- sapply(xs, function(x) dgnorm(log(x), mu, alpha4, beta) / x)
```

```
beta <- 6

tune <- stan(file='stan_programs/gen_norm_tune.stan',
             data=list("u" = lu, "mu" = mu, "beta" = beta),
             iter=1, warmup=0, chains=1,
             seed=4838282, algorithm="Fixed_param")
```

```
alpha = 0.650303

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                0 seconds (Sampling)
Chain 1:                0 seconds (Total)
Chain 1:
```

```
alpha6 <- extract(tune)$alpha
lgn6_ds <- sapply(xs, function(x) dgnorm(log(x), mu, alpha6, beta) / x)
```

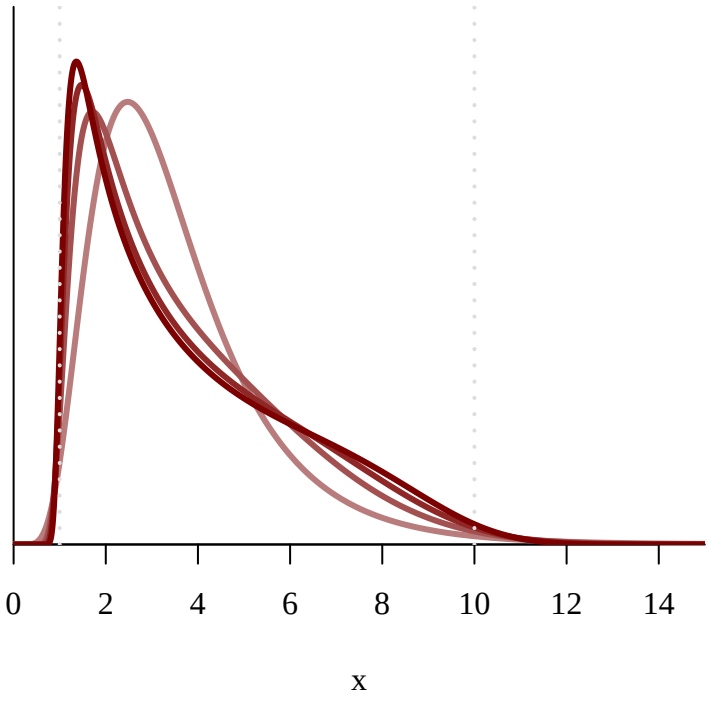
```
beta <- 8

tune <- stan(file='stan_programs/gen_norm_tune.stan',
             data=list("u" = lu, "mu" = mu, "beta" = beta),
             iter=1, warmup=0, chains=1,
             seed=4838282, algorithm="Fixed_param")
```

```
alpha = 0.415655

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                0 seconds (Sampling)
Chain 1:                0 seconds (Total)
Chain 1:
```

```
alpha8 <- extract(tune)$alpha
lgn8_ds <- sapply(xs, function(x) dgnorm(log(x), mu, alpha8, beta) / x)
```

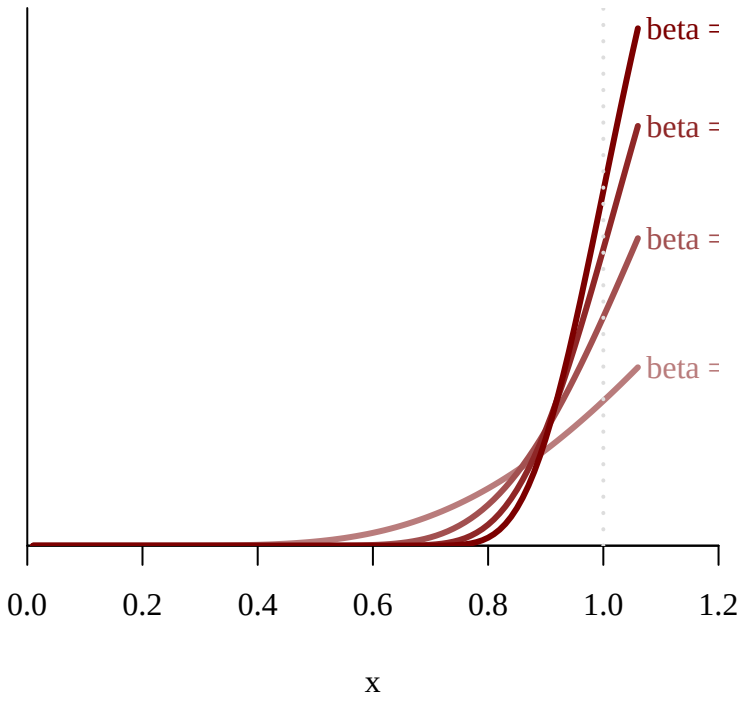
The induced behaviors as we increase β parallel what we saw in the log mixture of normal model as we increased the number of components K. As the latent model becomes more platykurtic the induced model becomes more skewed towards smaller values with a shoulder peaking up closer to the upper containment threshold.

```
par(mfrow=c(1, 1), mar=c(5, 1, 1, 1))

plot(0, type='n',
     xlab='x', xlim=c(0, 15),
     yaxt='n', ylab="", ylim=c(0, 0.35))

lines(xs, lgn2_ds,  lwd=3, col=lcs[3])
lines(xs, lgn4_ds,  lwd=3, col=lcs[4])
lines(xs, lgn6_ds,  lwd=3, col=lcs[5])
lines(xs, lgn8_ds, lwd=3, col=lcs[6])

abline(v=l, lwd=2, lty=3, col="#DDDDDD")
abline(v=u, lwd=2, lty=3, col="#DDDDDD")
```

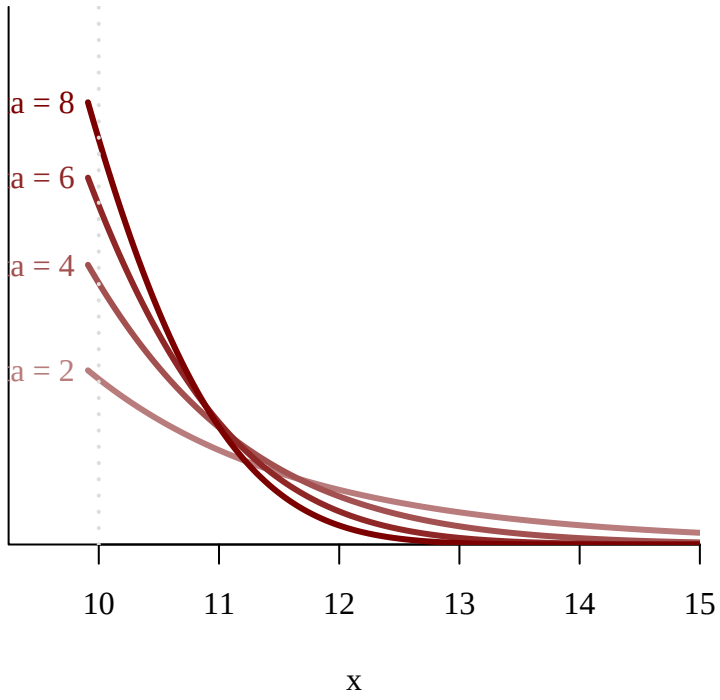


Once again both the left and right tails systematically lighten. This is not in spite of the increasingly awkward skew but rather because of it!

```
par(mfrow=c(1, 1), mar=c(5, 1, 1, 1))

plot(0, type='n',
     xlab='x', xlim=c(0, 1.2),
     yaxt='n', ylab="", ylim=c(0, 0.2))

lines(xs[1:N_lt], lgn2_ds[1:N_lt], lwd=3, col=lcs[3])
lines(xs[1:N_lt], lgn4_ds[1:N_lt], lwd=3, col=lcs[4])
lines(xs[1:N_lt], lgn6_ds[1:N_lt], lwd=3, col=lcs[5])
lines(xs[1:N_lt], lgn8_ds[1:N_lt], lwd=3, col=lcs[6])

abline(v=1, lwd=2, lty=3, col="#DDDDDD")

text(1.075, lgn2_ds[N_lt], adj=0, labels='beta = 2', col=lcs[3])
text(1.075, lgn4_ds[N_lt], adj=0, labels='beta = 4', col=lcs[4])
text(1.075, lgn6_ds[N_lt], adj=0, labels='beta = 6', col=lcs[5])
text(1.075, lgn8_ds[N_lt], adj=0, labels='beta = 8', col=lcs[6])
```



```
par(mfrow=c(1, 1), mar=c(5, 1, 1, 1))

plot(0, type='n',
     xlab='x', xlim=c(9.25, 15),
     yaxt='n', ylab="", ylim=c(0, 0.0175))

lines(xs[N_rt:N], lgn2_ds[N_rt:N], lwd=3, col=lcs[3])
lines(xs[N_rt:N], lgn4_ds[N_rt:N], lwd=3, col=lcs[4])
lines(xs[N_rt:N], lgn6_ds[N_rt:N], lwd=3, col=lcs[5])
lines(xs[N_rt:N], lgn8_ds[N_rt:N], lwd=3, col=lcs[6])

abline(v=u, lwd=2, lty=3, col="#DDDDDD")

text(9.8, lgn2_ds[N_rt], adj=1, labels='beta = 2', col=lcs[3])
text(9.8, lgn4_ds[N_rt], adj=1, labels='beta = 4', col=lcs[4])
text(9.8, lgn6_ds[N_rt], adj=1, labels='beta = 6', col=lcs[5])
text(9.8, lgn8_ds[N_rt], adj=1, labels='beta = 8', col=lcs[6])
```

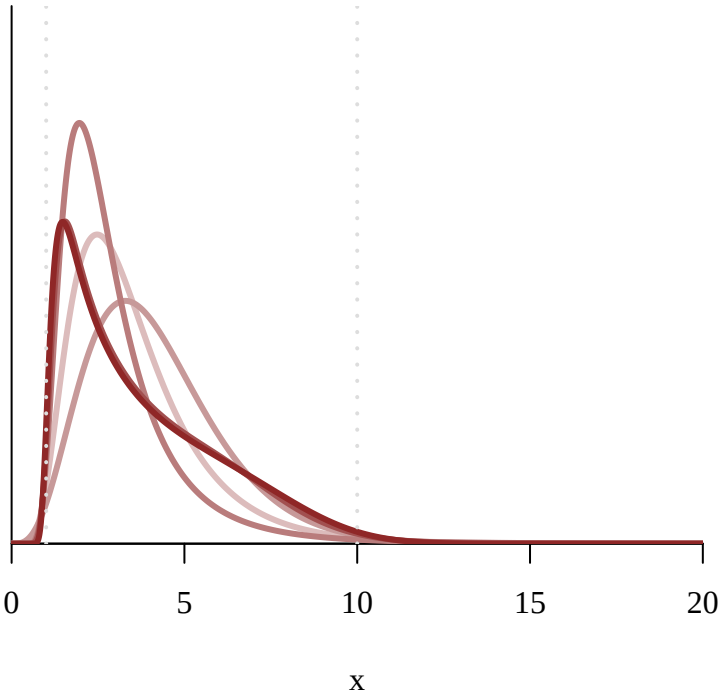


4.3 Comparison

Now let's put these new log platykurtic models into context by comparing them to our initial triptych of the log normal, gamma, and inverse gamma models. For simplicity we'll look at log mixture of normals and log symmetric generalized normal models with particularly light tails.

The log platykurtic models enjoy the symmetric tail behaviors of the log normal model, the light right tails of the gamma model, and the light left tails of the inverse gamma model. That said this all comes at the expense of increasingly extreme skews.

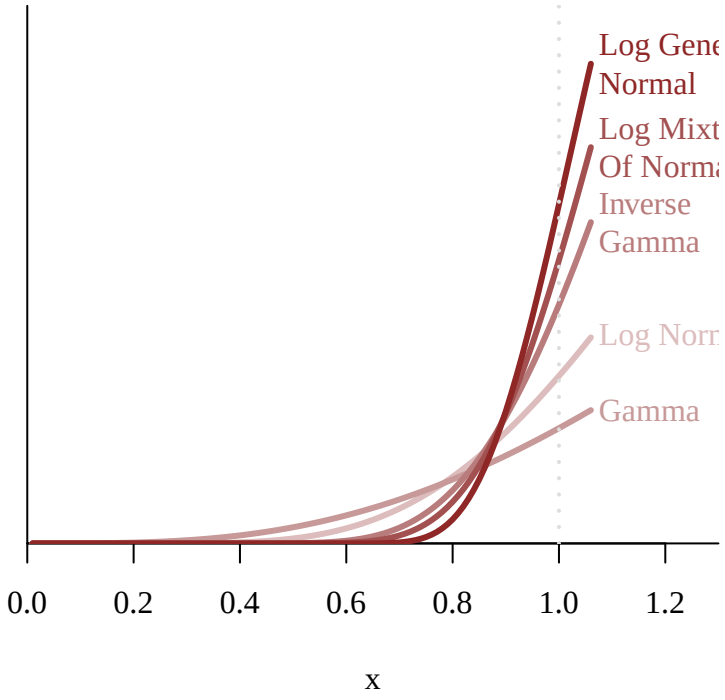
```
par(mfrow=c(1, 1), mar=c(5, 1, 1, 1))

plot(0, type='n',
     xlab='x', xlim=c(0, 20),
     yaxt='n', ylab="", ylim=c(0, 0.5))

lines(xs, ln_ds,   lwd=3, col=lcs[1])
lines(xs, gm_ds,   lwd=3, col=lcs[2])
lines(xs, igm_ds,  lwd=3, col=lcs[3])
lines(xs, lmn8_ds, lwd=3, col=lcs[4])
lines(xs, lgn6_ds, lwd=3, col=lcs[5])

abline(v=l, lwd=2, lty=3, col="#DDDDDD")
abline(v=u, lwd=2, lty=3, col="#DDDDDD")
```



The tail behaviors are easier to see if we zoom in to the containment thresholds. Again the log platykurtic models exhibit the lightest tails in both directions.

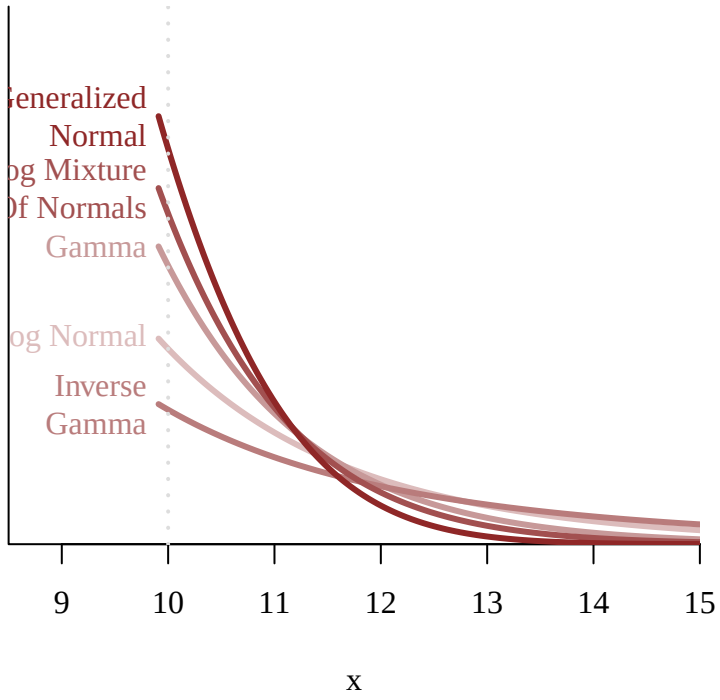
```
par(mfrow=c(1, 1), mar=c(5, 1, 1, 1))

plot(0, type='n',
     xlab='x', xlim=c(0, 1.3),
     yaxt='n', ylab="", ylim=c(0, 0.175))

lines(xs[1:N_lt], ln_ds[1:N_lt],   lwd=3, col=lcs[1])
lines(xs[1:N_lt], gm_ds[1:N_lt],   lwd=3, col=lcs[2])
lines(xs[1:N_lt], igm_ds[1:N_lt],  lwd=3, col=lcs[3])
lines(xs[1:N_lt], lmn8_ds[1:N_lt], lwd=3, col=lcs[4])
lines(xs[1:N_lt], lgn6_ds[1:N_lt], lwd=3, col=lcs[5])

abline(v=1, lwd=2, lty=3, col="#DDDDDD")

text(1.075, ln_ds[N_lt], adj=0,
     labels='Log Normal', col=lcs[1])
text(1.075, gm_ds[N_lt], adj=0,
     labels='Gamma', col=lcs[2])
text(1.075, igm_ds[N_lt], adj=0,
     labels='Inverse\nGamma', col=lcs[3])
```

```
text(1.075, lmn8_ds[N_lt], adj=0,
     labels='Log Mixture\nOf Normals', col=lcs[4])
text(1.075, lgn6_ds[N_lt], adj=0,
     labels='Log Generalized\nNormal', col=lcs[5])
```




```
par(mfrow=c(1, 1), mar=c(5, 1, 1, 1))

plot(0, type='n',
     xlab='x', xlim=c(8.5, 15),
     yaxt='n', ylab="", ylim=c(0, 0.015))

lines(xs[N_rt:N], ln_ds[N_rt:N],   lwd=3, col=lcs[1])
lines(xs[N_rt:N], gm_ds[N_rt:N],   lwd=3, col=lcs[2])
lines(xs[N_rt:N], igm_ds[N_rt:N],  lwd=3, col=lcs[3])
lines(xs[N_rt:N], lmn8_ds[N_rt:N], lwd=3, col=lcs[4])
lines(xs[N_rt:N], lgn6_ds[N_rt:N], lwd=3, col=lcs[5])

abline(v=u, lwd=2, lty=3, col="#DDDDDD")

text(9.8, ln_ds[N_rt], adj=1,
     labels='Log Normal', col=lcs[1])
text(9.8, gm_ds[N_rt], adj=1,
```

```
     labels='Gamma', col=lcs[2])
text(9.8, igm_ds[N_rt], adj=1,
     labels='Inverse\nGamma', col=lcs[3])
text(9.8, lmn8_ds[N_rt], adj=1,
     labels='Log Mixture\nOf Normals', col=lcs[4])
text(9.8, lgn6_ds[N_rt], adj=1,
     labels='Log Generalized\nNormal', col=lcs[5])
```



5 Log Skew Normal Model

Both of the log platykurtic models that we considered suffered from the same compromise: the only way to achieve lighter tails was with the introduction of stronger skews towards smaller values and an awkward shoulder at larger values.

In a practical Bayesian application these features will often be overwhelmed by the contribution from the likelihood function, but that is not always guaranteed. What do we do if we want to engineer an even more robust prior model that is more symmetric between the containment thresholds?

The main problem with our initial log platykurtic models is that the latent models were symmetric. On the other hand the exponential function that we use to push this model

forward to a model over positive real numbers is asymmetric. It squeezes negative inputs into the narrow output interval $(0, 1)$ but allows positive inputs to stretch out across the interval $(1, +\infty)$. Pushing forward a symmetric model along an asymmetric function will necessarily result in an asymmetric output model.

Consequently if we want a symmetric output model then we need to compensate for the asymmetry of the exponential function with an asymmetric input model. In other words we need to preemptively skew the latent model to avoid skew in the output model.

Here let's demonstrate this strategy with a log skew normal model.

```
library(sn)
ys <- seq(-1, 3, 0.01)
```

```
alpha <- -2

tune <- stan(file='stan_programs/skew_normal_tune.stan',
             data=list("l" = ll, "u" = lu, alpha=alpha),
             iter=1, warmup=0, chains=1,
             seed=4838282, algorithm="Fixed_param")
```

```
mu = 1.81332
sigma = 0.703974

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                0 seconds (Sampling)
Chain 1:                0 seconds (Total)
Chain 1:
```

```
mu <- extract(tune)$mu
sigma <- extract(tune)$sigma

sn2_ds <- sapply(ys, function(y) dsn(y, mu, sigma, alpha))
lsn2_ds <- sapply(xs, function(x) dsn(log(x), mu, sigma, alpha) / x)
```

```
alpha <- -3

tune <- stan(file='stan_programs/skew_normal_tune.stan',
```

```
             data=list("l" = ll, "u" = lu, alpha=alpha),
             iter=1, warmup=0, chains=1,
             seed=4838282, algorithm="Fixed_param")
```

```
mu = 1.96873
sigma = 0.764309

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                0 seconds (Sampling)
Chain 1:                0 seconds (Total)
Chain 1:
```

```
mu <- extract(tune)$mu
sigma <- extract(tune)$sigma

sn3_ds <- sapply(ys, function(y) dsn(y, mu, sigma, alpha))
lsn3_ds <- sapply(xs, function(x) dsn(log(x), mu, sigma, alpha) / x)
```

```
alpha <- -4

tune <- stan(file='stan_programs/skew_normal_tune.stan',
             data=list("l" = ll, "u" = lu, alpha=alpha),
             iter=1, warmup=0, chains=1,
             seed=4838282, algorithm="Fixed_param")
```

```
mu = 2.06018
sigma = 0.799811

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                0 seconds (Sampling)
Chain 1:                0 seconds (Total)
Chain 1:
```

```
mu <- extract(tune)$mu
sigma <- extract(tune)$sigma

sn4_ds <- sapply(ys, function(y) dsn(y, mu, sigma, alpha))
lsn4_ds <- sapply(xs, function(x) dsn(log(x), mu, sigma, alpha) / x)
```

```
alpha <- -5

tune <- stan(file='stan_programs/skew_normal_tune.stan',
             data=list("l" = ll, "u" = lu, alpha=alpha),
             iter=1, warmup=0, chains=1,
             seed=4838282, algorithm="Fixed_param")
```

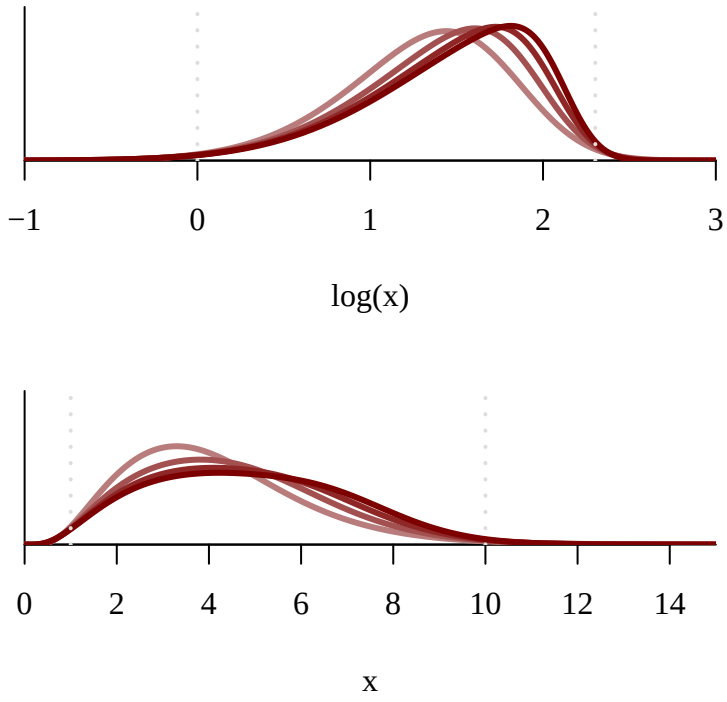
```
mu = 2.11836
sigma = 0.822401

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                0 seconds (Sampling)
Chain 1:                0 seconds (Total)
Chain 1:
```

```
mu <- extract(tune)$mu
sigma <- extract(tune)$sigma

sn5_ds <- sapply(ys, function(y) dsn(y, mu, sigma, alpha))
lsn5_ds <- sapply(xs, function(x) dsn(log(x), mu, sigma, alpha) / x)
```

The more skewed towards larger values the latent skew normal models are the more symmetric the derived models become.

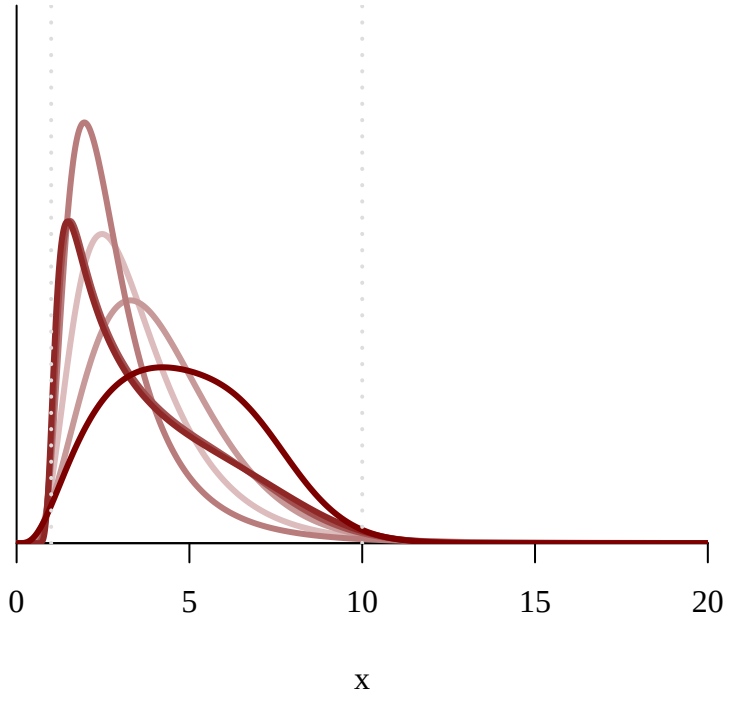
```
par(mfrow=c(2, 1), mar=c(5, 1, 1, 1))

plot(0, type='n',
     xlab='log(x)', xlim=c(-1, 3),
     yaxt='n', ylab="", ylim=c(0, 1))

lines(ys, sn2_ds, lwd=3, col=lcs[3])
```

```
lines(ys, sn3_ds, lwd=3, col=lcs[4])
lines(ys, sn4_ds, lwd=3, col=lcs[5])
lines(ys, sn5_ds, lwd=3, col=lcs[6])

abline(v=ll, lwd=2, lty=3, col="#DDDDDD")
abline(v=lu, lwd=2, lty=3, col="#DDDDDD")

plot(0, type='n',
     xlab='x', xlim=c(0, 15),
     yaxt='n', ylab="", ylim=c(0, 0.35))

lines(xs, lsn2_ds, lwd=3, col=lcs[3])
lines(xs, lsn3_ds, lwd=3, col=lcs[4])
lines(xs, lsn4_ds, lwd=3, col=lcs[5])
lines(xs, lsn5_ds, lwd=3, col=lcs[6])

abline(v=l, lwd=2, lty=3, col="#DDDDDD")
abline(v=u, lwd=2, lty=3, col="#DDDDDD")
```



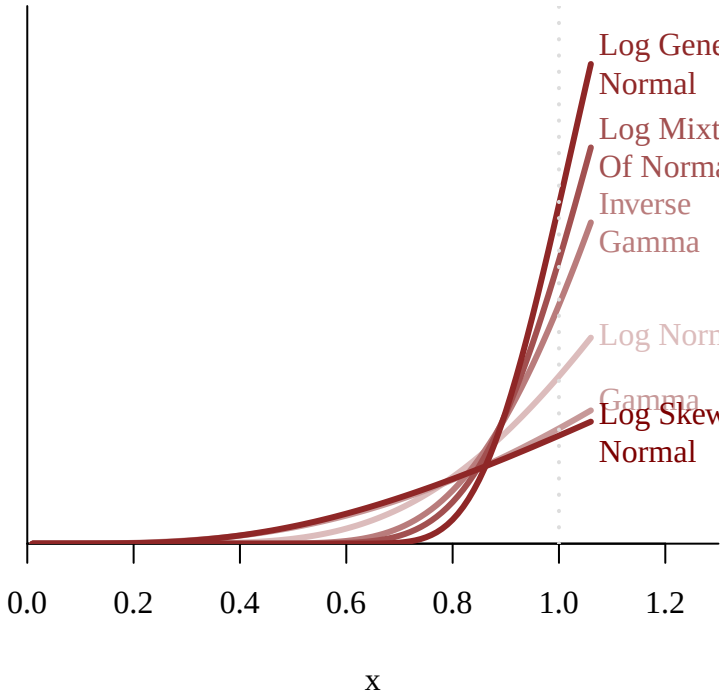
With enough latent asymmetric a log skew normal model can achieve the desired containment with far more symmetry than any of the models that we have previously considered.

```
par(mfrow=c(1, 1), mar=c(5, 1, 1, 1))

plot(0, type='n',
     xlab='x', xlim=c(0, 20),
     yaxt='n', ylab="", ylim=c(0, 0.5))

lines(xs, ln_ds,   lwd=3, col=lcs[1])
lines(xs, gm_ds,   lwd=3, col=lcs[2])
lines(xs, igm_ds,  lwd=3, col=lcs[3])
lines(xs, lmn8_ds, lwd=3, col=lcs[4])
lines(xs, lgn6_ds, lwd=3, col=lcs[5])
lines(xs, lsn5_ds, lwd=3, col=lcs[6])

abline(v=l, lwd=2, lty=3, col="#DDDDDD")
abline(v=u, lwd=2, lty=3, col="#DDDDDD")
```



That said the tails of the log skew normal model are not as well-behaved as the log platykurtic model. The strong latent skew results in a relatively heavy left tail on the input space which propagates to a heavier left tail on the output space. In particular the left tail of this skew normal model is just as heavy as the left tail of the gamma model.

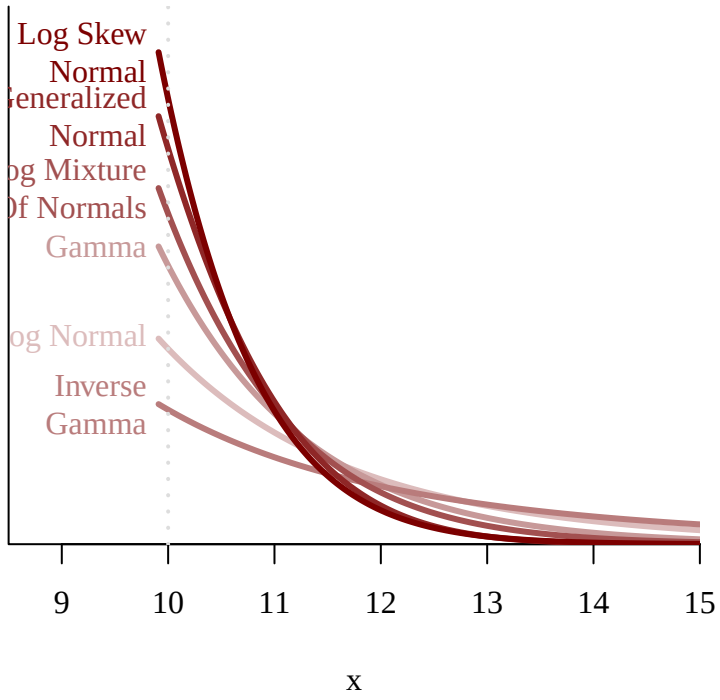
```
par(mfrow=c(1, 1), mar=c(5, 1, 1, 1))

plot(0, type='n',
     xlab='x', xlim=c(0, 1.3),
     yaxt='n', ylab="", ylim=c(0, 0.175))

lines(xs[1:N_lt], ln_ds[1:N_lt],   lwd=3, col=lcs[1])
lines(xs[1:N_lt], gm_ds[1:N_lt],   lwd=3, col=lcs[2])
lines(xs[1:N_lt], igm_ds[1:N_lt],  lwd=3, col=lcs[3])
lines(xs[1:N_lt], lmn8_ds[1:N_lt], lwd=3, col=lcs[4])
lines(xs[1:N_lt], lgn6_ds[1:N_lt], lwd=3, col=lcs[5])
lines(xs[1:N_lt], lsn5_ds[1:N_lt], lwd=3, col=lcs[5])

abline(v=l, lwd=2, lty=3, col="#DDDDDD")

text(1.075, ln_ds[N_lt], adj=0,
     labels='Log Normal', col=lcs[1])
text(1.075, gm_ds[N_lt] + 0.0035, adj=0,
     labels='Gamma', col=lcs[2])
text(1.075, igm_ds[N_lt], adj=0,
     labels='Inverse\nGamma', col=lcs[3])
text(1.075, lmn8_ds[N_lt], adj=0,
     labels='Log Mixture\nOf Normals', col=lcs[4])
text(1.075, lgn6_ds[N_lt], adj=0,
     labels='Log Generalized\nNormal', col=lcs[5])
text(1.075, lsn5_ds[N_lt] - 0.0035, adj=0,
     labels='Log Skew\nNormal', col=lcs[6])
```



The lightness of the right tail of the log skew normal model, however, competes well against all of the other models.

```
par(mfrow=c(1, 1), mar=c(5, 1, 1, 1))

plot(0, type='n',
     xlab='x', xlim=c(8.5, 15),
     yaxt='n', ylab="", ylim=c(0, 0.015))

lines(xs[N_rt:N], ln_ds[N_rt:N],   lwd=3, col=lcs[1])
lines(xs[N_rt:N], gm_ds[N_rt:N],   lwd=3, col=lcs[2])
lines(xs[N_rt:N], igm_ds[N_rt:N],  lwd=3, col=lcs[3])
lines(xs[N_rt:N], lmn8_ds[N_rt:N], lwd=3, col=lcs[4])
lines(xs[N_rt:N], lgn6_ds[N_rt:N], lwd=3, col=lcs[5])
lines(xs[N_rt:N], lsn5_ds[N_rt:N], lwd=3, col=lcs[6])

abline(v=u, lwd=2, lty=3, col="#DDDDDD")

text(9.8, ln_ds[N_rt],  adj=1,
     labels='Log Normal', col=lcs[1])
text(9.8, gm_ds[N_rt],  adj=1,
     labels='Gamma', col=lcs[2])
text(9.8, igm_ds[N_rt], adj=1,
```

```
     labels='Inverse\nGamma', col=lcs[3])
text(9.8, lmn8_ds[N_rt], adj=1,
     labels='Log Mixture\nOf Normals', col=lcs[4])
text(9.8, lgn6_ds[N_rt], adj=1,
     labels='Log Generalized\nNormal', col=lcs[5])
text(9.8, lsn5_ds[N_rt], adj=1,
     labels='Log Skew\nNormal', col=lcs[6])
```



Overall the log skew normal model can be configured to allow for relatively symmetric behavior with a particularly light right tail. The left tail, however, is necessarily on the heavier side. Consequently care would be needed if there is especially pathological behavior just to the left of the lower threshold.

6 Conclusion

In this note I've investigated a few different strategies for engineering more flexible tail behaviors beyond what we can achieve with just a log normal, gamma, or inverse gamma model. Overall symmetric light tails, and tight probabilistic containment, requires asymmetric behavior in between the containment thresholds. This asymmetry is particularly apparent when the lower containment threshold is close to zero.

Ultimately these kinds of compromises are not pathological but rather inherit to working on constrained spaces. The closer a probabilistic model concentrates towards the boundaries of a constrained space the more their will have to contort to achieve particular behaviors, such as containment, without violating the constraints.

Theoretically we can reduce this contortion but considering more and more flexible models, but that also rapidly increases the implementation and tuning burdens. I did play around with a few more candidate models, such as log mixtures of skew normals and Box-Cox power exponential models, but they were awkward to tune and often introduced new, undesired behaviors.

To avoid overwhelming ourselves I strongly recommend starting as simply as possible in practice and increasing sophistication only as needed. For example when developing an appropriate containment model over a positive real line we might first reach for a log normal model and all of its convenient properties. At that point if we encounter problems because the containment of the log normal model is too tight, such as excessively extreme simulations or posterior distributions that stretch too far past the containment thresholds into numerically ill-behaved regions, then we can consider a more elaborate model.

That said we still want to keep the iterative changes as simple as possible. If the problematic behaviors are arising on only one side of the containment interval then we can reach for a gamma or inverse gamma model. Only if we are encountering problems on both sides should we consider a more sophisticated model such as the ones we investigated in this note. Even then we don't want to be too precious about features like symmetry until we actually encounter substantial consequences tied to those features.

Remember our prior models are not meant to be a perfect encapsulation of all of our domain expertise. That would require infinite time and energy. Our goal in practical prior modeling is to incorporate the aspects of our domain expertise that result in the best inferences. We can always iteratively improve an initial prior model to include more detailed domain expertise as needed.

License

The code in this case study is copyrighted by Michael Betancourt and licensed under the new BSD (3-clause) license:

https://opensource.org/licenses/BSD-3-Clause

The text and figures in this chapter are copyrighted by Michael Betancourt and licensed under the CC BY-NC 4.0 license:

https://creativecommons.org/licenses/by-nc/4.0/

Original Computing Environment

```
writeLines(readLines(file.path(Sys.getenv("HOME"), ".R/Makevars")))
```

```
CC=clang

CXXFLAGS=-O3 -mtune=native -march=native -Wno-unused-variable -Wno-unused-function -Wno-macr
CXX=clang++ -arch x86_64 -ftemplate-depth-256

CXX14FLAGS=-O3 -mtune=native -march=native -Wno-unused-variable -Wno-unused-function -Wno-ma
CXX14=clang++ -arch x86_64 -ftemplate-depth-256
```

```
sessionInfo()
```

```
R version 4.3.2 (2023-10-31)
Platform: x86_64-apple-darwin20 (64-bit)
Running under: macOS 15.6.1

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/lib/libRlapack.dylib;

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

time zone: America/New_York
tzcode source: internal

attached base packages:
[1] stats4    stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
[1] sn_2.1.1           rstan_2.32.6       StanHeaders_2.32.7 colormap_0.1.4

loaded via a namespace (and not attached):
 [1] gtable_0.3.4        jsonlite_1.8.8      compiler_4.3.2
 [4] Rcpp_1.0.11         stringr_1.5.1       parallel_4.3.2
 [7] gridExtra_2.3       scales_1.3.0        yaml_2.3.8
[10] fastmap_1.1.1       ggplot2_3.4.4       R6_2.5.1
```

```
[13] curl_5.2.0          knitr_1.45          tibble_3.2.1
[16] munsell_0.5.0       pillar_1.9.0        rlang_1.1.2
[19] utf8_1.2.4          V8_4.4.1            inline_0.3.19
[22] stringi_1.8.3       xfun_0.41           RcppParallel_5.1.7
[25] cli_3.6.2           magrittr_2.0.3      digest_0.6.33
[28] grid_4.3.2          lifecycle_1.0.4     vctrs_0.6.5
[31] mnormt_2.1.1        evaluate_0.23       glue_1.6.2
[34] numDeriv_2016.8-1.1 QuickJSR_1.0.8      codetools_0.2-19
[37] pkgbuild_1.4.3      fansi_1.0.6         colorspace_2.1-0
[40] rmarkdown_2.25      matrixStats_1.2.0   loo_2.6.0
[43] pkgconfig_2.0.3     tools_4.3.2         htmltools_0.5.7
```

Stan
Program 1 gamma_tune.stan

```
functions {
  vector tail_delta(vector y, vector theta,
                    real[] x_r, int[] x_i) {
    vector[2] deltas;

    if (!is_nan(y[1]) && !is_nan(y[2])) {
      deltas[1] = gamma_cdf(theta[1], exp(y[1]), exp(y[2])) - 0.01;
      deltas[2] = gamma_cdf(theta[2], exp(y[1]), exp(y[2])) - 0.99;
    } else {
      deltas[1] = 0;
      deltas[2] = 0;
    }
    return deltas;
  }
}

data {
  real<lower=0> l;
  real<lower=l> u;
  real delta;
}

transformed data {
  real alpha_guess = square(delta * (u + l) / (u - l));
  real beta_guess =  2 * square(delta) * (u + l) / square(u - l);
  vector[2] y_guess = [log(alpha_guess), log(beta_guess)]';

  vector[2] theta = [l, u]';
  vector[2] y;
  real x_r[0];
  int x_i[0];

  y = algebra_solver(tail_delta, y_guess, theta, x_r, x_i);
  print("alpha = ", exp(y[1]));
  print("beta = ", exp(y[2]));
}

generated quantities {
  real alpha = exp(y[1]);
  real beta = exp(y[2]);
}
```

Stan
Program 2 `inv\_gamma\_tune.stan`

```
functions {
  vector tail_delta(vector y, vector theta,
                    real[] x_r, int[] x_i) {
    vector[2] deltas;

    if (!is_nan(y[1]) && !is_nan(y[2])) {
      deltas[1] = inv_gamma_cdf(theta[1], exp(y[1]), exp(y[2])) - 0.01;
      deltas[2] = inv_gamma_cdf(theta[2], exp(y[1]), exp(y[2])) - 0.99;
    } else {
      deltas[1] = 0;
      deltas[2] = 0;
    }
    return deltas;
  }
}

data {
  real<lower=0> l;
  real<lower=l> u;
  real delta;
}

transformed data {
  real alpha_guess
    = square(delta * (u + l) / (u - l)) + 2;
  real beta_guess
    = ((u + l) / 2) * ( square(delta * (u + l) / (u - l)) + 1);
  vector[2] y_guess = [log(alpha_guess), log(beta_guess)]';

  vector[2] theta = [l, u]';
  vector[2] y;
  real x_r[0];
  int x_i[0];

  y = algebra_solver(tail_delta, y_guess, theta, x_r, x_i);
  print("alpha = ", exp(y[1]));
  print("beta = ", exp(y[2]));
}

generated quantities {
  real alpha = exp(y[1]);
  real beta = exp(y[2]);
}
```

Stan
Program 3 `platy\_mixture\_tune.stan`

```
functions {
  real platy_mixture_lpdf(real x, real mu, real sigma, int K) {
    real n1 = 1.0 / K;
    real n2 = 1.0 / (K - 1);

    vector[K] nus
      =   mu
        + sigma * (2 * linspaced_vector(K, 1, K) - 1 - K) * n2;
    real tau = 2 * sigma * n1;

    vector[K] comp_lpds;
    for (k in 1:K) {
      comp_lpds[k] = normal_lpdf(x | nus[k], tau);
    }

    return log_sum_exp(comp_lpds) - log(K);
  }

  real platy_mixture_cdf(real x, real mu, real sigma, int K) {
    real n1 = 1.0 / K;
    real n2 = 1.0 / (K - 1);

    vector[K] nus
      =   mu
        + sigma * (2 * linspaced_vector(K, 1, K) - 1 - K) * n2;
    real tau = 2 * sigma * n1;

    vector[K] comp_ps;
    for (k in 1:K) {
      comp_ps[k] = normal_cdf(x | nus[k], tau);
    }

    return sum(comp_ps) * n1;
  }

  vector tail_delta(vector y, vector theta,
                    real[] x_r, int[] x_i) {
    vector[1] deltas
      = [   platy_mixture_cdf(theta[1], theta[2], exp(y[1]), x_i[1])
          - 0.99 ]';
    return deltas;
  }
}

data {
  real u;
  real mu;
  int<lower=1> K;
}
```

Stan
Program 4 gen_norm_tune.stan

```
functions {
  real gen_norm_lpdf(real x, real mu, real alpha, real beta) {
    return - alpha * pow(abs(x - mu), beta)
           + log(beta) - (1 / beta) * log(alpha)
           - log2() - lgamma(1 / beta);
  }

  real gen_norm_cdf(real x, real mu, real alpha, real beta) {
    real p = gamma_cdf( alpha * pow(abs(x - mu), beta), 1 / beta, 1);
    if ( (x - mu) > 0)
      return 0.5 * (1 + p);
    else
      return 0.5 * (1 - p);
  }

  vector tail_delta(vector y, vector theta,
                    real[] x_r, int[] x_i) {
    vector[1] deltas
      = [   gen_norm_cdf(theta[1], theta[2], exp(y[1]), theta[3])
          - 0.99 ]';
    return deltas;
  }
}

data {
  real u;
  real mu;
  real<lower=0> beta;
}

transformed data {
  vector[1] y_guess = [ pow(u - mu, 1 / beta) ]';

  vector[3] theta = [u, mu, beta]';
  vector[1] y;
  real x_r[0];
  int x_i[0];

  y = algebra_solver(tail_delta, y_guess, theta, x_r, x_i);
  print("alpha = ", exp(y[1]));
}

generated quantities {
  real alpha = exp(y[1]);
}
```

Stan
Program 5 `skew\_normal\_tune.stan`

```
functions {
  vector tail_delta(vector y, vector theta,
                    array[] real x_r, array[] int x_i) {
    vector[2] deltas = [
      skew_normal_cdf(theta[1] | y[1], exp(y[2]), theta[3] ) - 0.01,
      1 - skew_normal_cdf(theta[2] | y[1], exp(y[2]), theta[3] ) - 0.01 ]';
    return deltas;
  }
}

data {
  real<lower=0> l;
  real<lower=l> u;
  real alpha;
}

transformed data {
  vector[2] y_guess = [0.5 * (u + l), log(0.5 * (u - l)) ]';

  vector[3] theta = [l, u, alpha]';
  vector[2] y;
  array[0] real x_r;
  array[0] int x_i;

  y = algebra_solver(tail_delta, y_guess, theta, x_r, x_i);

  print("mu = ", y[1]);
  print("sigma = ", exp(y[2]));
}

generated quantities {
  real mu = y[1];
  real sigma = exp(y[2]);
}
```