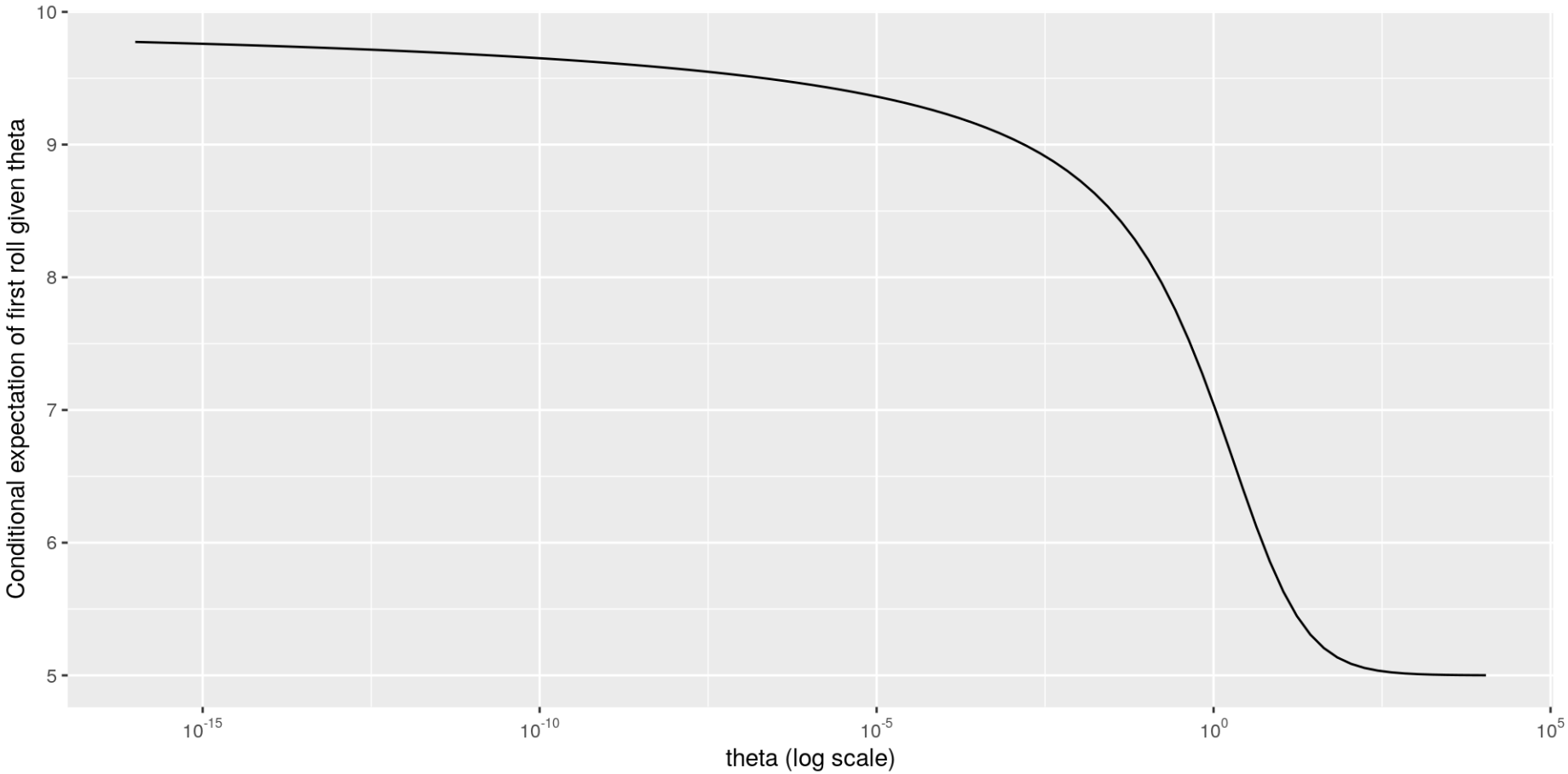
Markov Chain Monte Carlo for Bayesian Inference

Ben Goodrich

March 30, 2022

Plot from Last Week



Marginal Probability of the First Roll in Bowling

- The CONDITIONAL PMF of $X_1 \mid \theta$ is $\Pr(x_1 \mid n, \theta) = \frac{\log_{n+1+\theta}\left(1+\frac{1}{n+\theta-x_1}\right)}{1-\log_{n+1+\theta}(\theta)}$
- The BIVARIATE PDF of θ and X_1 is

  $f(\theta, x_1 \mid n) = \frac{b^a}{\Gamma(a)}\theta^{a-1}e^{-b\theta}\,\frac{\log_{n+1+\theta}\left(1+\frac{1}{n+\theta-x_1}\right)}{1-\log_{n+1+\theta}(\theta)}$
- The MARGINAL PMF of X_1 is $\Pr(x_1 \mid n, a, b) =$

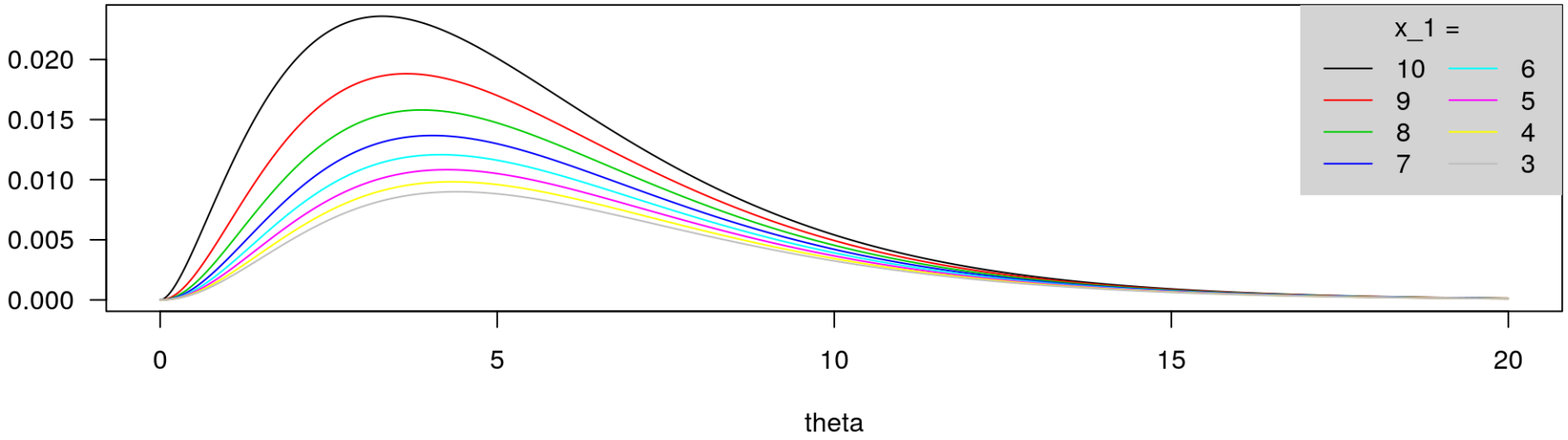
$$f\left(\cancel{\theta}, x_1 \mid n, a, b\right) = \int_0^\infty \frac{b^a}{\Gamma(a)}\theta^{a-1}e^{-b\theta}\,\frac{\log_{n+1+\theta}\left(1+\frac{1}{n+\theta-x_1}\right)}{1-\log_{n+1+\theta}(\theta)}\,d\theta$$

  but we can't obtain the antiderivative to evalute the area

- The Risch algorithm can tell you if a function has an elementary antiderivative

Marginalized Probability of the First Roll

```
a <- 3   # prior shape in gamma prior
b <- 0.5 # prior rate in gamma prior
joint <- function(theta, x_1)
  dgamma(theta, shape = a, rate = b) * sapply(theta, FUN = Pr, x = x_1, n = 10)
```




```
integrate(joint, lower = 0, upper = Inf, x_1 = 8)$value # little trapezoids
```

```
## [1] 0.1134608
```

Marginalized Probability of a Frame

```
marginal_Pr <- matrix(0, nrow = 11, ncol = 11, dimnames = list(Omega, Omega))
for (x_1 in Omega) {
  for(x_2 in 0:(10 - x_1)) {
    marginal_Pr[x_1 + 1, x_2 + 1] <- integrate(function(theta) {
      dgamma(theta, shape = a, rate = b) *
        sapply(theta, FUN = function(t) { # applies function to each cell of theta vector
          Pr(x_1, n = 10, theta = t) * Pr(x_2, n = 10 - x_1, theta = t)
        })
    }, lower = 0, upper = Inf)$value
  }
}
sum(marginal_Pr)
```

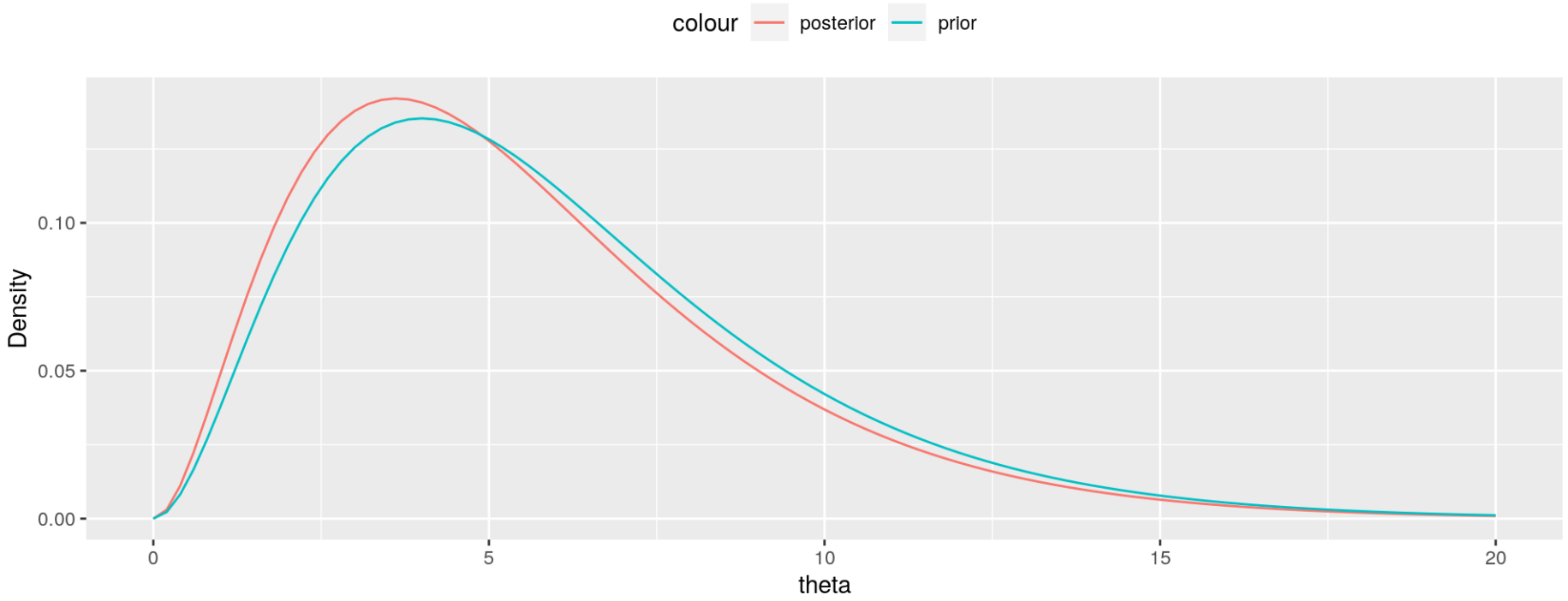
```
## [1] 1
```

Marginalized Probabily of a Frame

	0	1	2	3	4	5	6	7	8	9	10
0	0.003191	0.003394	0.003625	0.003892	0.004204	0.004575	0.005024	0.005581	0.0063	0.007277	0.008752
1	0.003833	0.004095	0.004396	0.004749	0.005168	0.005676	0.006307	0.007119	0.008225	0.009895	0
2	0.004682	0.005027	0.005431	0.005911	0.006492	0.007215	0.008146	0.009414	0.011329	0	0
3	0.005836	0.006305	0.006863	0.007539	0.00838	0.009464	0.01094	0.013172	0	0	0
4	0.00746	0.008121	0.008923	0.00992	0.011206	0.012959	0.015613	0	0	0	0
5	0.009848	0.010823	0.012036	0.013601	0.015736	0.018974	0	0	0	0	0
6	0.01357	0.015096	0.017067	0.01976	0.023851	0	0	0	0	0	0
7	0.019852	0.022459	0.026026	0.031462	0	0	0	0	0	0	0
8	0.031816	0.036917	0.044728	0	0	0	0	0	0	0	0
9	0.059767	0.072677	0	0	0	0	0	0	0	0	0
10	0.162309	0	0	0	0	0	0	0	0	0	0

Posterior Distribution Given $x_1 = 8$ and $x_2 = 2$

```
ggplot() + xlim(0, 20) + ylab("Density") + xlab("theta") + theme(legend.position = "top") +
  geom_function(fun = ~dgamma(.x, shape = a, rate = b) * sapply(.x, FUN = function(t) {
                    Pr(8, n = 10, t) * Pr(2, n = 10 - 8, t)
                }) / marginal_Pr["8", "2"], aes(color = "posterior")) +
  geom_function(fun = dgamma, args = list(shape = 3, rate = 0.5),  aes(color = "prior"))
```

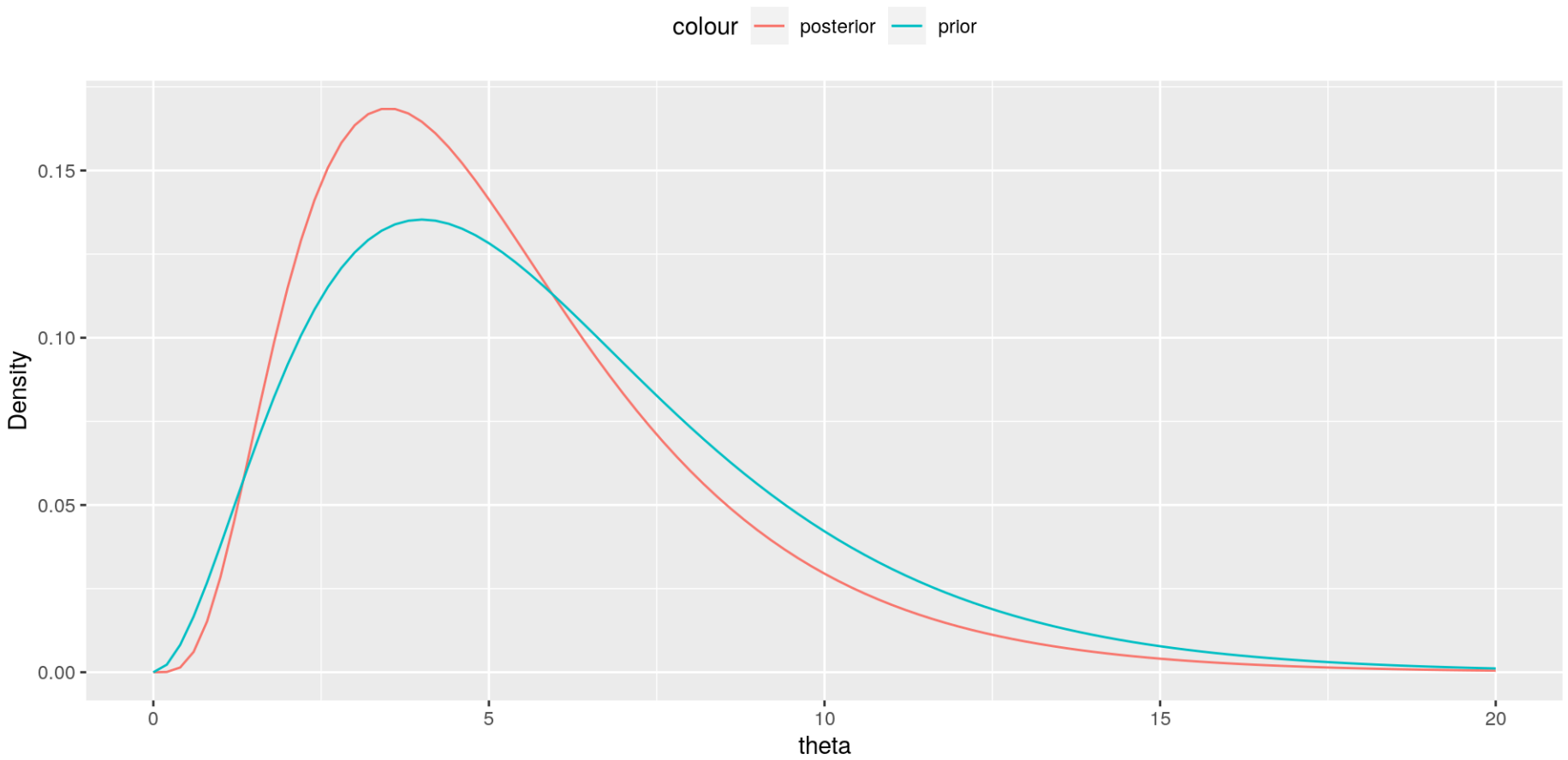


Posterior Distribution Conditional on One Game

```
game <- rbind(
  `1st` = c(7, 2), `2nd` = c(7, 1), `3rd` = c(10, 0), `4th` = c(5, 3),  `5th` = c(9, 1),
  `6th` = c(6, 1), `7th` = c(8, 2), `8th` = c(4,  5), `9th` = c(7, 3), `10th` = c(8, 1) )
log_likelihood <- function(theta) { # not a function --- in the mathematical sense --- of data
  sapply(theta, FUN = function(t) {
  sum(log(Pr(x = game[ , 1], n = 10, theta = t)), # use (natural) logs for numerical reasons
      log(Pr(x = game[ , 2], n = 10 - game[ , 1], theta = t)))
  })
}
numerator <- function(theta) {
  exp(dgamma(theta, shape = a, rate = b, log = TRUE) + log_likelihood(theta))
}
denominator <- integrate(numerator, lower = 0, upper = Inf)$value


ggplot() + xlim(0, 20) + ylab("Density") + xlab("theta") +
  geom_function(fun = ~numerator(.x) / denominator, aes(color = "posterior")) +
  geom_function(fun = dgamma, args = list(shape = a, rate = b),  aes(color = "prior")) +
  theme(legend.position = "top")
```

Plot from Previous Slide



A Very, Very Bayesian Example

- Taking limits, we can express Bayes' Rule for continuous random variables with Probability Density Functions (PDFs)

$$f(B \mid A) = \frac{f(B)\, f(A \mid B)}{f(A)}$$

- The PDF of the Gamma distribution (shape-rate parameterization) is

$$f(\mu \mid a, b) = \frac{b^a}{\Gamma(a)} \mu^{a-1} e^{-b\mu}$$

- Poisson PMF for N observations is again $f(y_1, \ldots, y_n \mid \mu) = \frac{\mu^S e^{-N\mu}}{S!}$
- Bayes' Rule is $f(\mu \mid a, b, y_1, \ldots, y_n) = \frac{\mu^{a-1} e^{-b\mu} \mu^S e^{-N\mu}}{?} = \frac{\mu^{a+S-1} e^{-(b+N)\mu}}{?}$
- ? must be $\frac{\Gamma(a^*)}{(b^*)^{a^*}}$ where $a^* = a + S$ and $b^* = b + N$ so posterior is Gamma

Ex Ante Probability (Density) of *Ex Post* Data

A likelihood function is the same expression as a P{D,M}F with 3 distinctions:

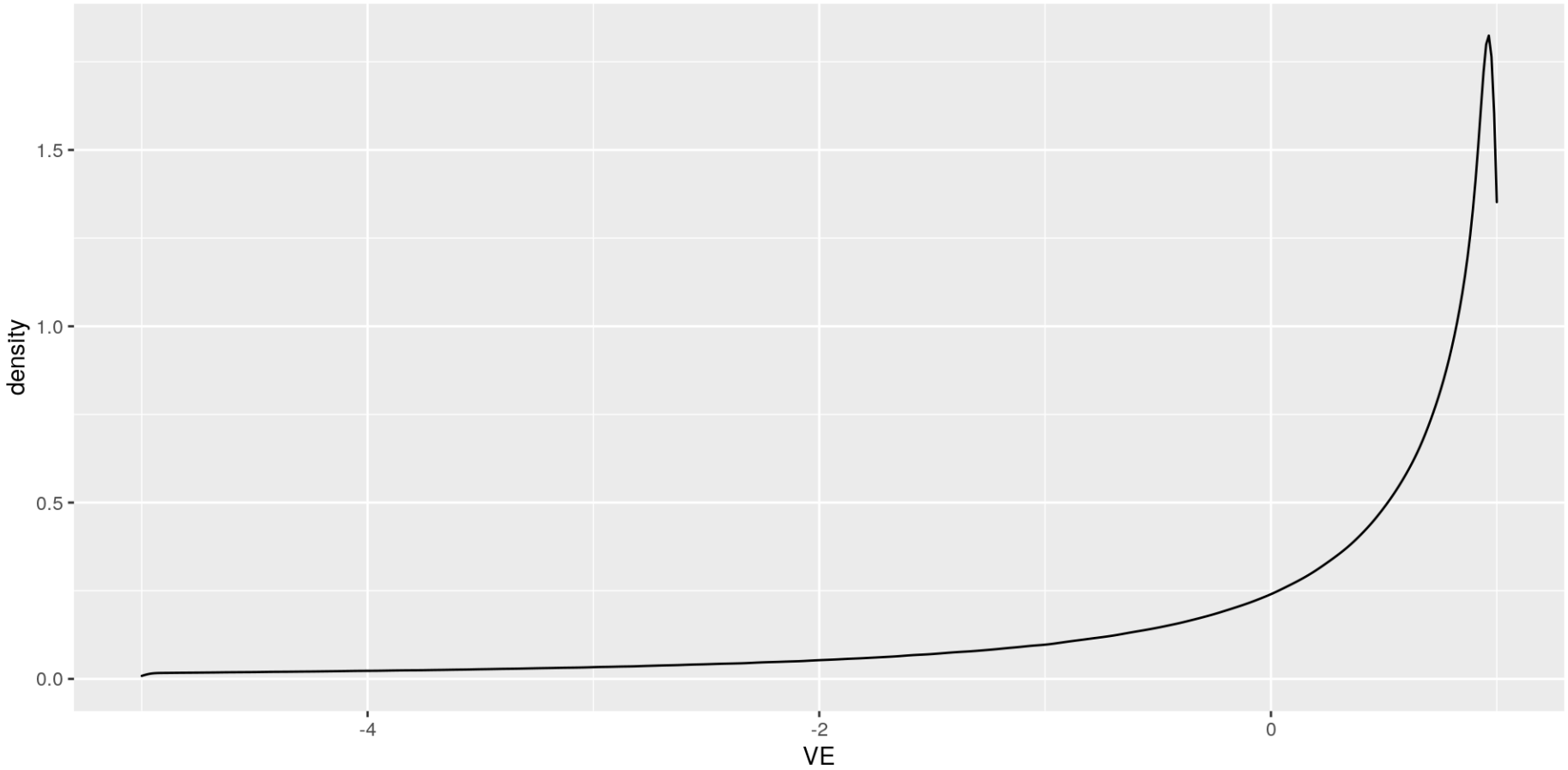
1. For the PDF or PMF, $f\left(x\middle|\boldsymbol{\theta}\right)$, we think of X as a random variable and $\boldsymbol{\theta}$ as given, whereas we conceive of the likelihood function, $\mathcal{L}\left(\boldsymbol{\theta};x\right)$, to be a function of $\boldsymbol{\theta}$ (in the mathematical sense) evaluted at the OBSERVED data, x
   - As a consequence, $\int_{-\infty}^{\infty} f\left(x\middle|\boldsymbol{\theta}\right)dx = 1$ or $\sum_{x\in\Omega} f\left(x\middle|\boldsymbol{\theta}\right) = 1$ while $\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty}\mathcal{L}\left(\boldsymbol{\theta};x\right)d\theta_1 d\theta_2\ldots d\theta_K$ may not exist and is never 1
2. We often think of "the likelihood function" for N conditionally independent observations, so $\mathcal{L}\left(\boldsymbol{\theta};\mathbf{x}\right) = \prod_{n=1}^{N}\mathcal{L}\left(\boldsymbol{\theta};x_n\right)$
3. By "the likelihood function", we often really mean the natural logrithm thereof, a.k.a. the log-likelihood function $\ell\left(\boldsymbol{\theta};\mathbf{x}\right) = \ln\mathcal{L}\left(\boldsymbol{\theta},\mathbf{x}\right) = \sum_{n=1}^{N}\ln\mathcal{L}\left(\boldsymbol{\theta};x_n\right)$

Biontech / Pfizer Analysis of First Covid Vaccine

- Let π_v be the probability of getting covid given that someone is vaccinated (in the Fall of 2020), π_c be the probability of getting covid given that someone is not vaccinated, $\theta = \frac{\pi_v}{\pi_v + \pi_c}$, and the "Vaccine Effect" is $\mathrm{VE}(\theta) = \frac{1-2\theta}{1-\theta} \leq 1$
- Beta distribution has PDF $f(\theta \mid a, b) = \frac{\theta^{a-1}(1-\theta)^{b-1}}{\int_0^1 t^{a-1}(1-t)^{b-1} dt} = \frac{\theta^{a-1}(1-\theta)^{b-1}}{B(a,b)}$
- Prior for θ was Beta with $a = 0.700102$ and $b = 1$, which was chosen (poorly) so that the VE at $\mathbb{E}\theta = \frac{a}{a+b}$ was about 0.3 (but it mattered little)

```
a <- 0.700102
b <- 1
ggplot(tibble(theta = rbeta(n = 10^6, shape1 = a, shape2 = b),
              VE = (1 - 2 * theta) / (1 - theta))) +
  geom_density(aes(x = VE)) + xlim(-5, 1) # see next slide
```

Implied Prior Distribution of $\mathrm{VE}\left(\theta\right)$



Deriving a Posterior Distribution of θ Analytically

- $\Pr(y \mid \theta, n) = \binom{n}{y}\theta^y(1-\theta)^{n-y}$ is binomial given θ, where "success" is getting covid when vaccinated and "failure" is getting covid when unvaccinated
- $y = 8$ vaccinated people and $n - y = 86$ non-vaccinated people got covid
- What are their beliefs about θ? ($\propto$ means "proportional to", i.e. the kernel)

$$f(\theta \mid a, b, n, y) = \frac{f(\theta \mid a, b)\, L(\theta; n, y)}{\int_0^1 f(\theta \mid a, b)\, L(\theta; n, y)\, d\theta} \propto$$

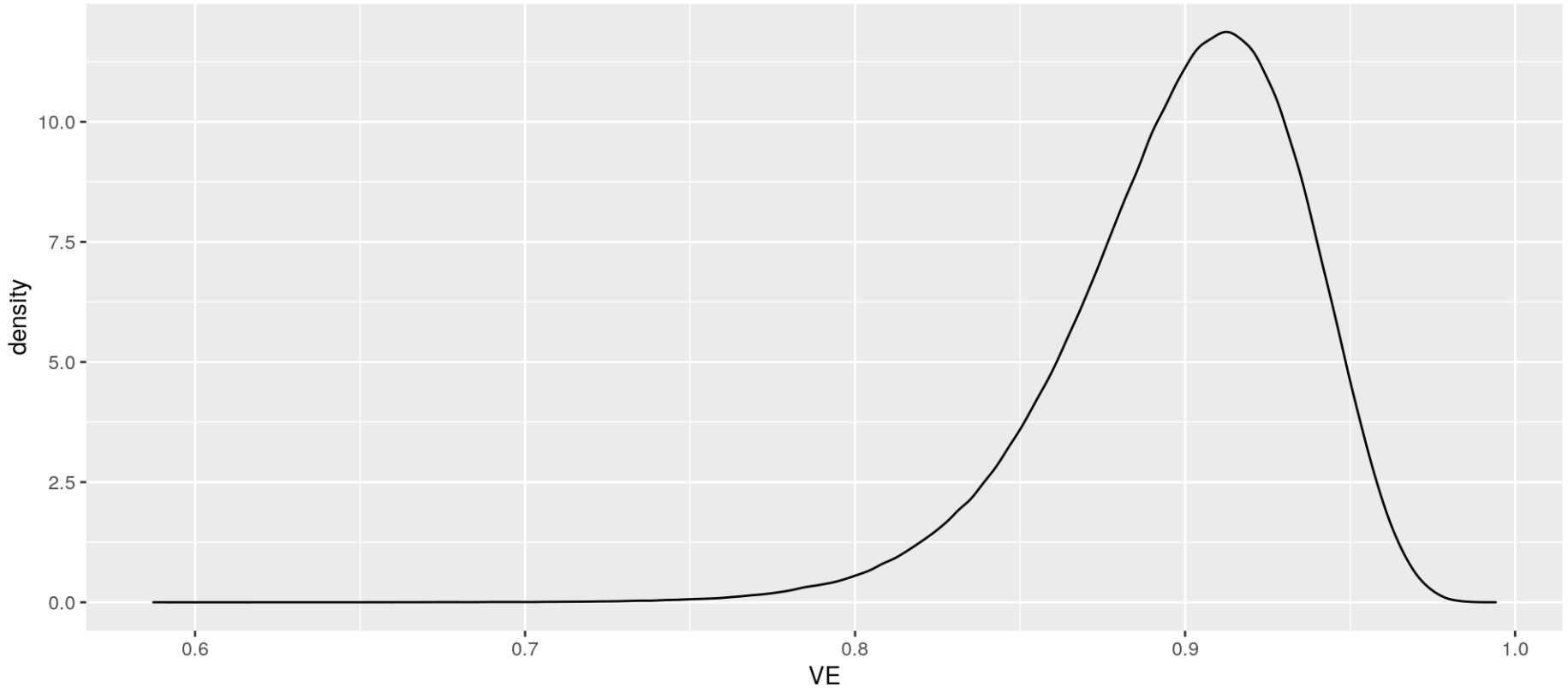
$$\theta^{a-1}(1-\theta)^{b-1}\theta^y(1-\theta)^{n-y} = \theta^{a+y-1}(1-\theta)^{b+n-y-1} = \theta^{a^*-1}(1-\theta)^{b^*-1}$$

where $a^* = a + y = 8.700102$ and $b^* = b + n - y = 87$

- $f(\theta \mid a^*, b^*)$ has the kernel of a Beta PDF and therefore its normalizing constant must be the reciprocal of $B(a^*, b^*) = \int_0^1 t^{a^*-1}(1-t)^{b^*-1}dt$

Implied Posterior Distribution of $\mathrm{VE}\left(\theta\right)$

```
y <- 8; n <- 94; a_star <- a + y; b_star <- b + n - y
ggplot(tibble(theta = rbeta(n = 10^6, shape1 = a_star, shape2 = b_star),
              VE = (1 - 2 * theta) / (1 - theta))) + geom_density(aes(x = VE))
```



Inverse CDF Sampling of Continuous RVs

- In principle, we have an algorithm to draw from ANY univariate continuous distribution
- If U is distributed standard uniform, what is the PDF of $X = F^{-1}(U)$?
- $\Pr(U \leq u) = u = \Pr(U \leq u(x))$
- $u(x) = F(x \mid \boldsymbol{\theta})$ with derivative $f(x \mid \boldsymbol{\theta})$
- So the PDF of X is $1 \times f(x \mid \boldsymbol{\theta})$
- `rnorm(1, mu, sigma)` is implemented by `qnorm(runif(1), mu, sigma)`

Generalized Lambda Distribution (GLD)

GLD lacks an explicit PDF & CDF so it is defined by its inverse CDF from p to Ω:

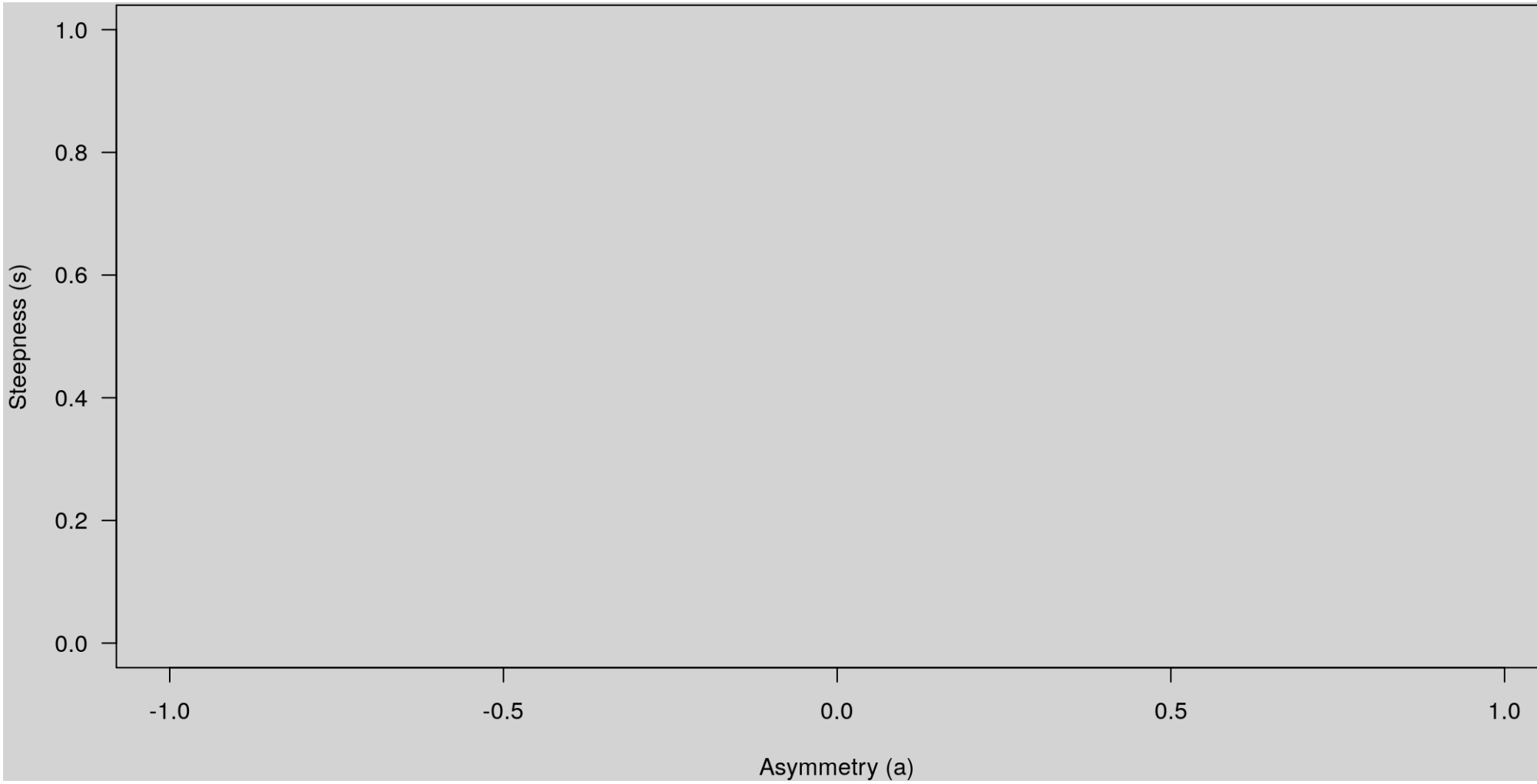
$$F^{-1}\left(p \mid m, r, a, s\right) = m + r \times F^{-1}\left(p \mid m = 0, r = 1, a, s\right)$$

$$F^{-1}\left(p \mid m = 0, r = 1, a, s\right) = \frac{S\left(p; a, s\right) - S\left(0.5; a, s\right)}{S\left(0.75; a, s\right) - S\left(0.25; a, s\right)}$$

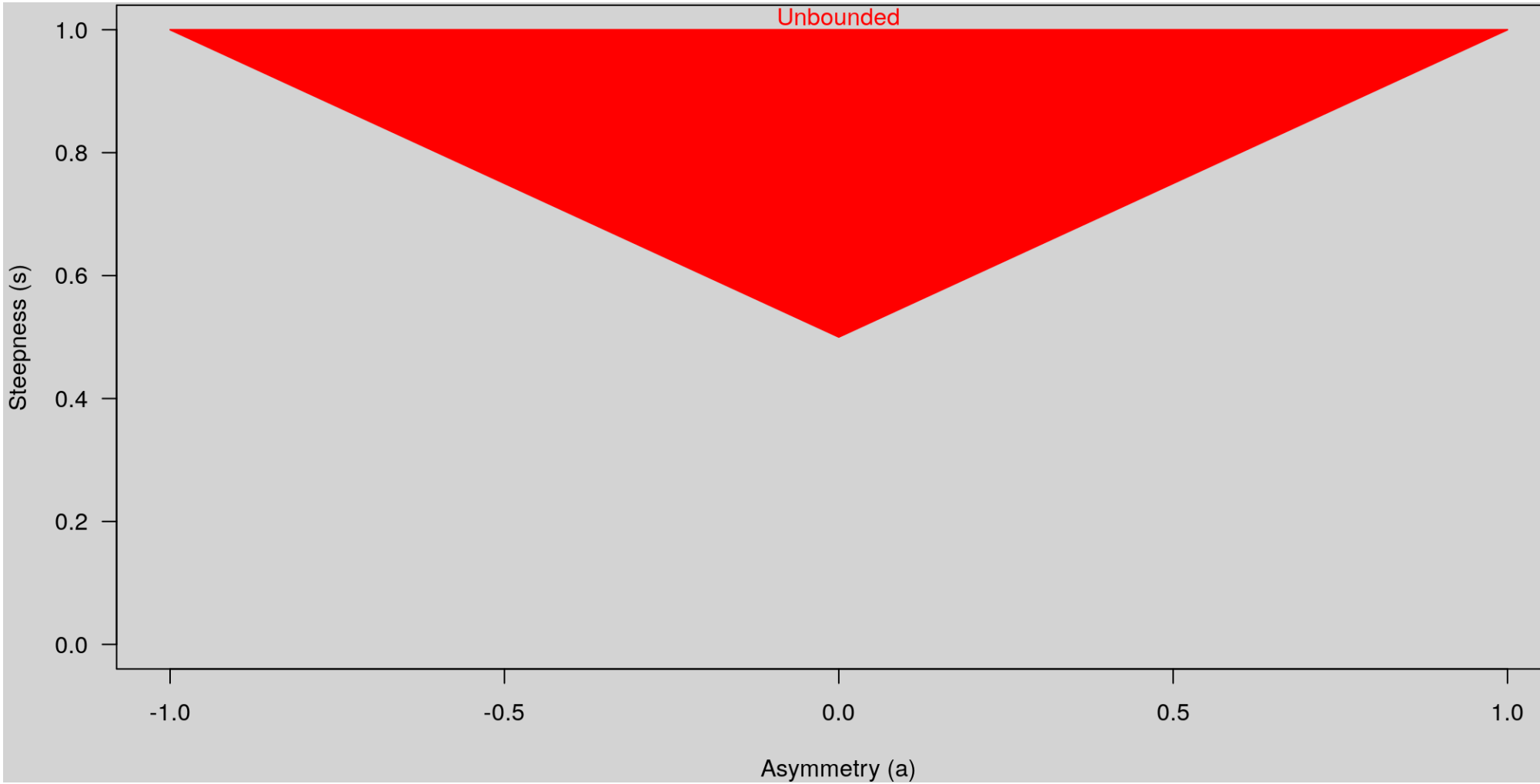
$$S\left(p; a, s\right) = \frac{p^{\alpha+\beta} - 1}{\alpha + \beta} - \frac{\left(1 - p\right)^{\alpha-\beta} - 1}{\alpha - \beta}, \alpha = \frac{0.5 - s}{2\sqrt{s\left(1 - s\right)}}, \beta = \frac{a}{2\sqrt{1 - a^2}}$$

- m is the median
- $r > 0$ is the interquartile range, i.e. the difference between the quartiles
- $a \in \left(-1, 1\right)$ controls the asymmetry (if symmetric, then $a = 0$)
- $s \in \left(0, 1\right)$ controls the steepness (i.e. the heaviness) of its tails
- Limits are needed to evaluate $S\left(p; a, s\right)$ as $2s \to 1 \pm a$

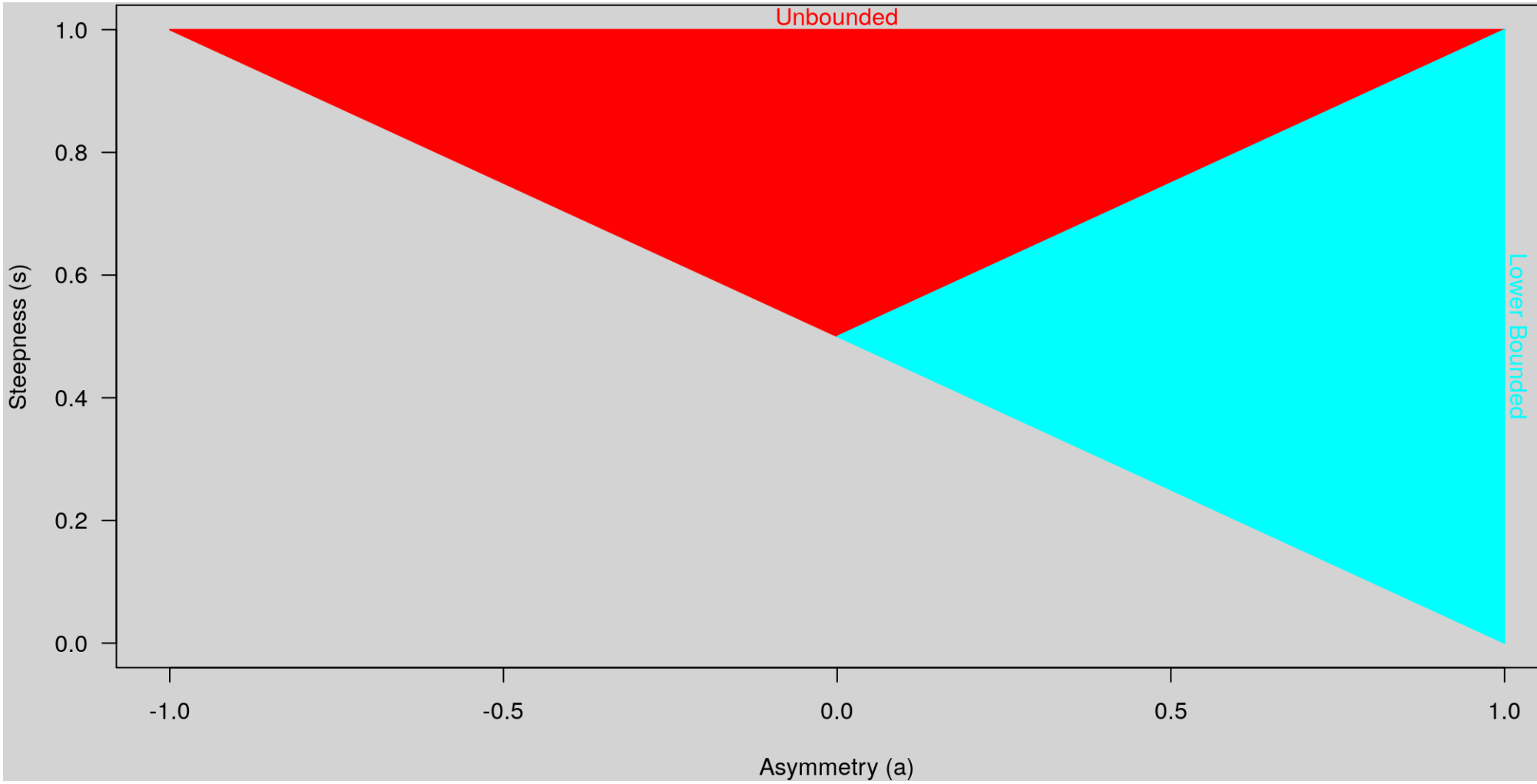
Special Cases of the GLD (for some m and r)



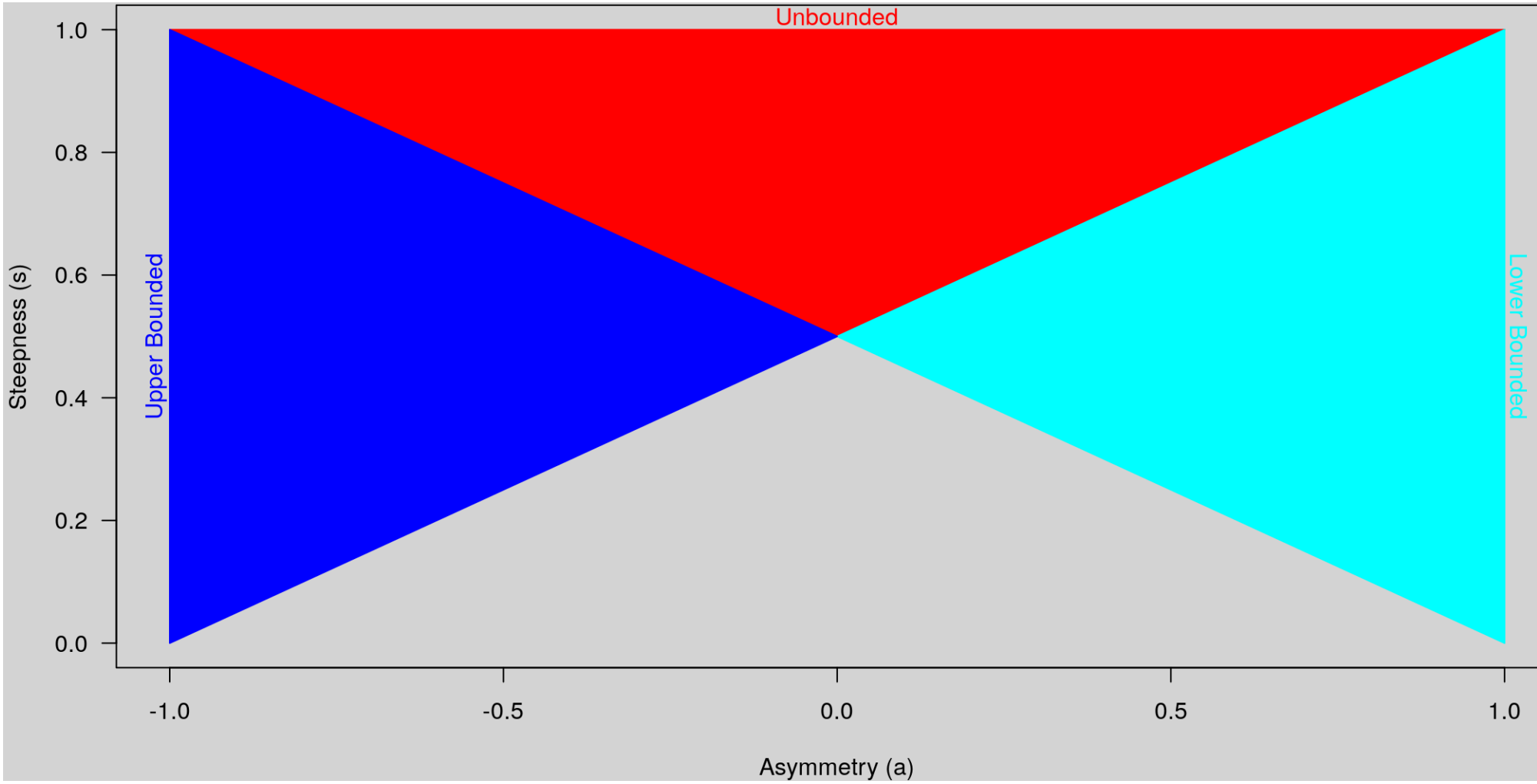
Special Cases of the GLD (for some m and r)



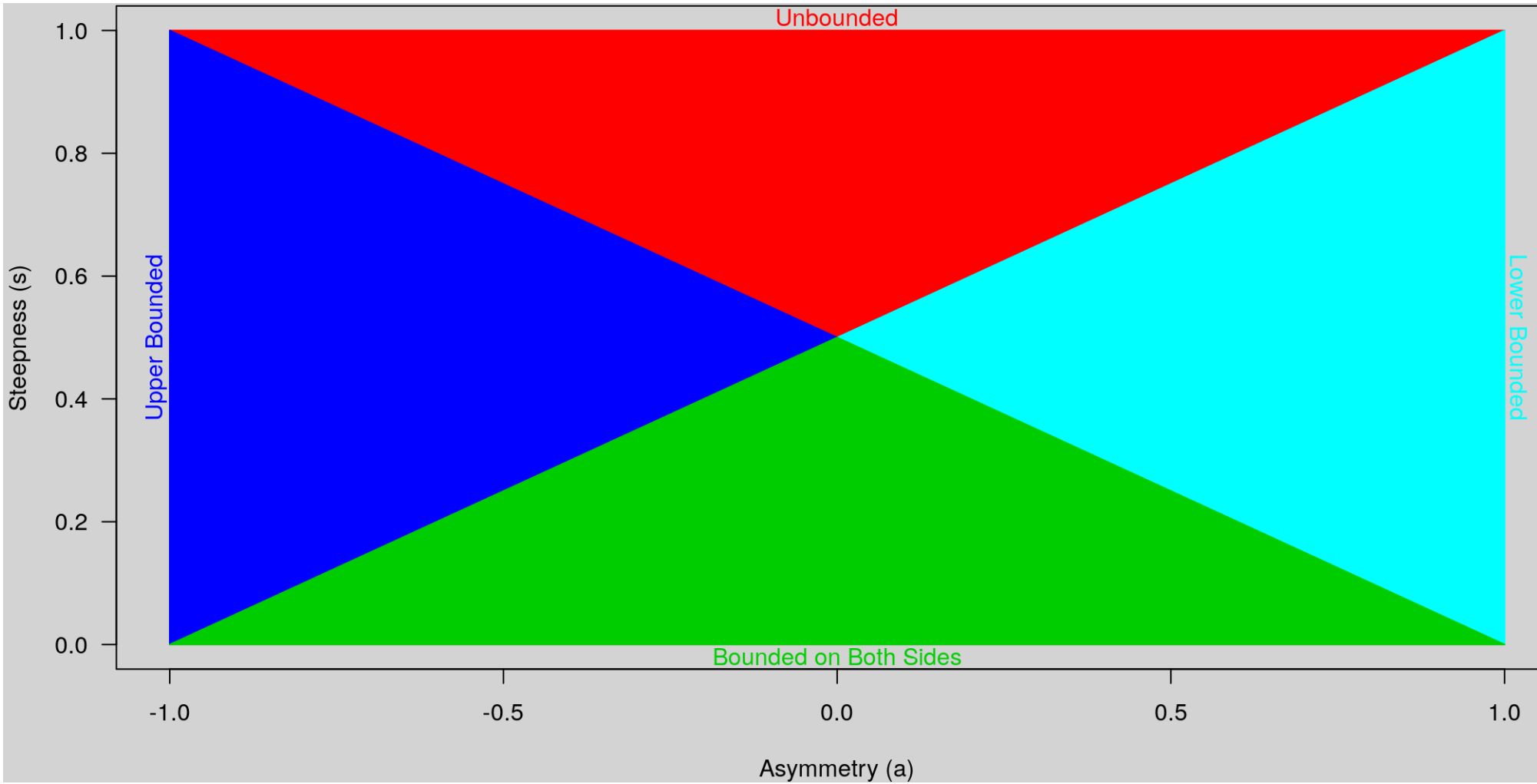
Special Cases of the GLD (for some m and r)



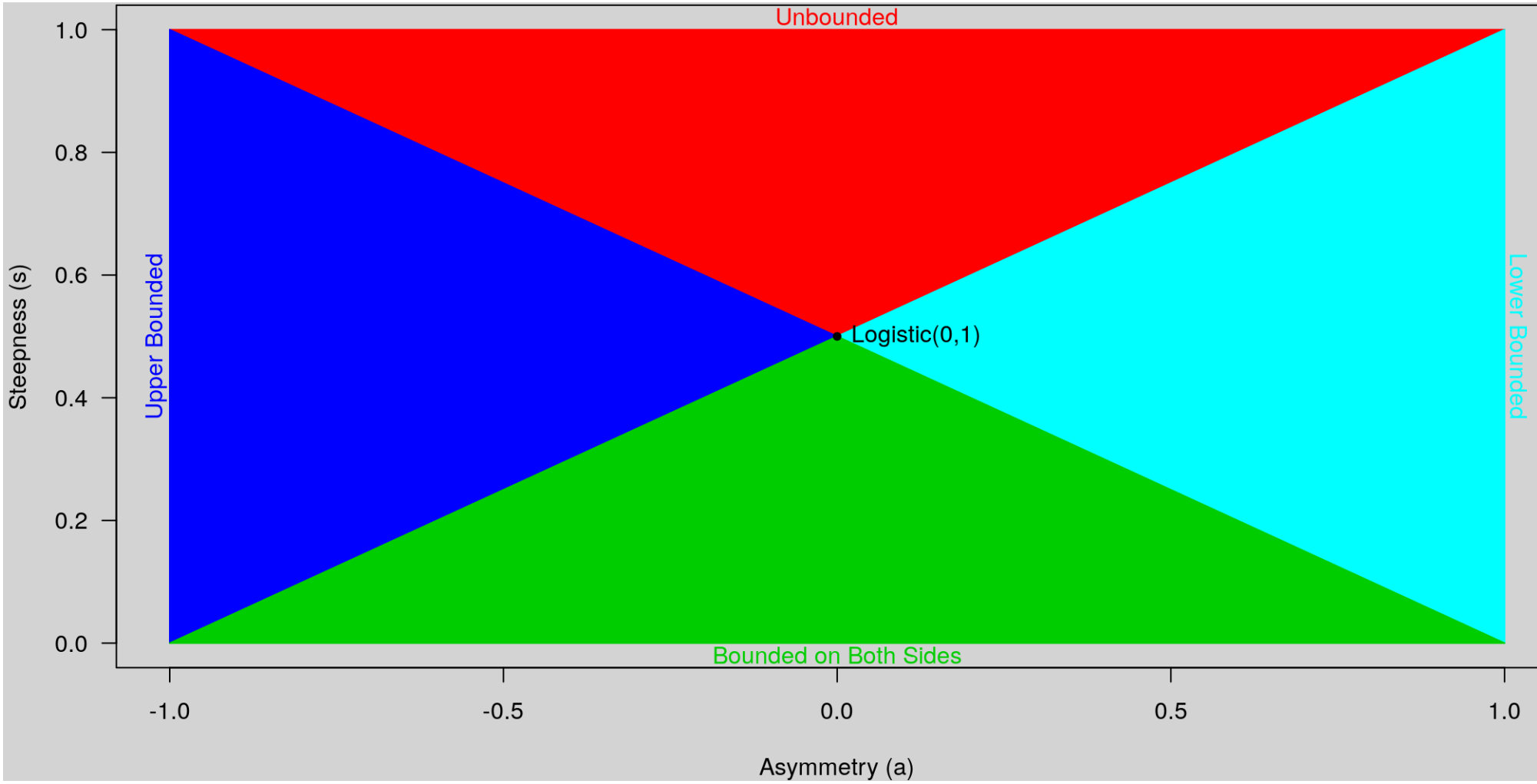
Special Cases of the GLD (for some m and r)



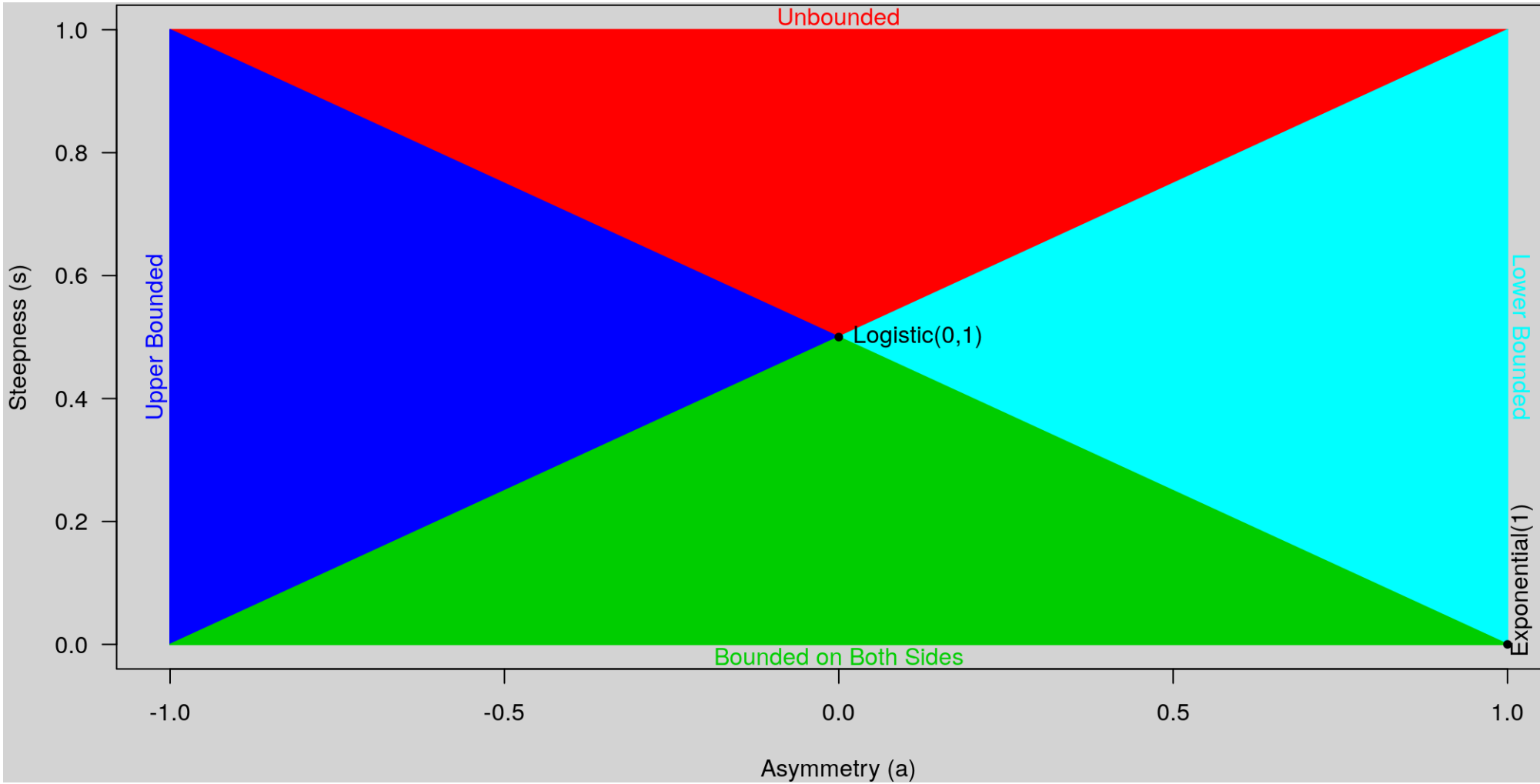
Special Cases of the GLD (for some m and r)



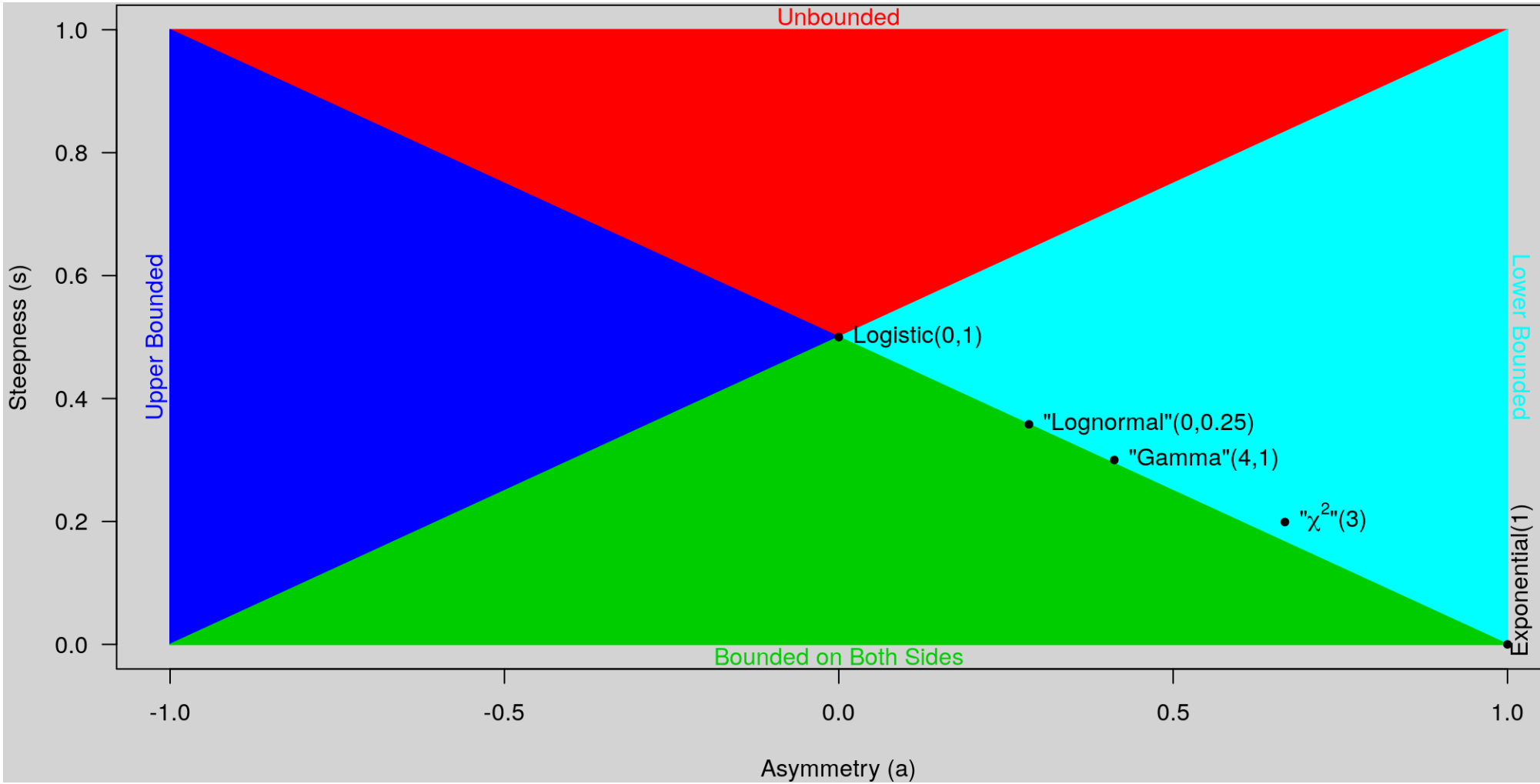
Special Cases of the GLD (for some m and r)



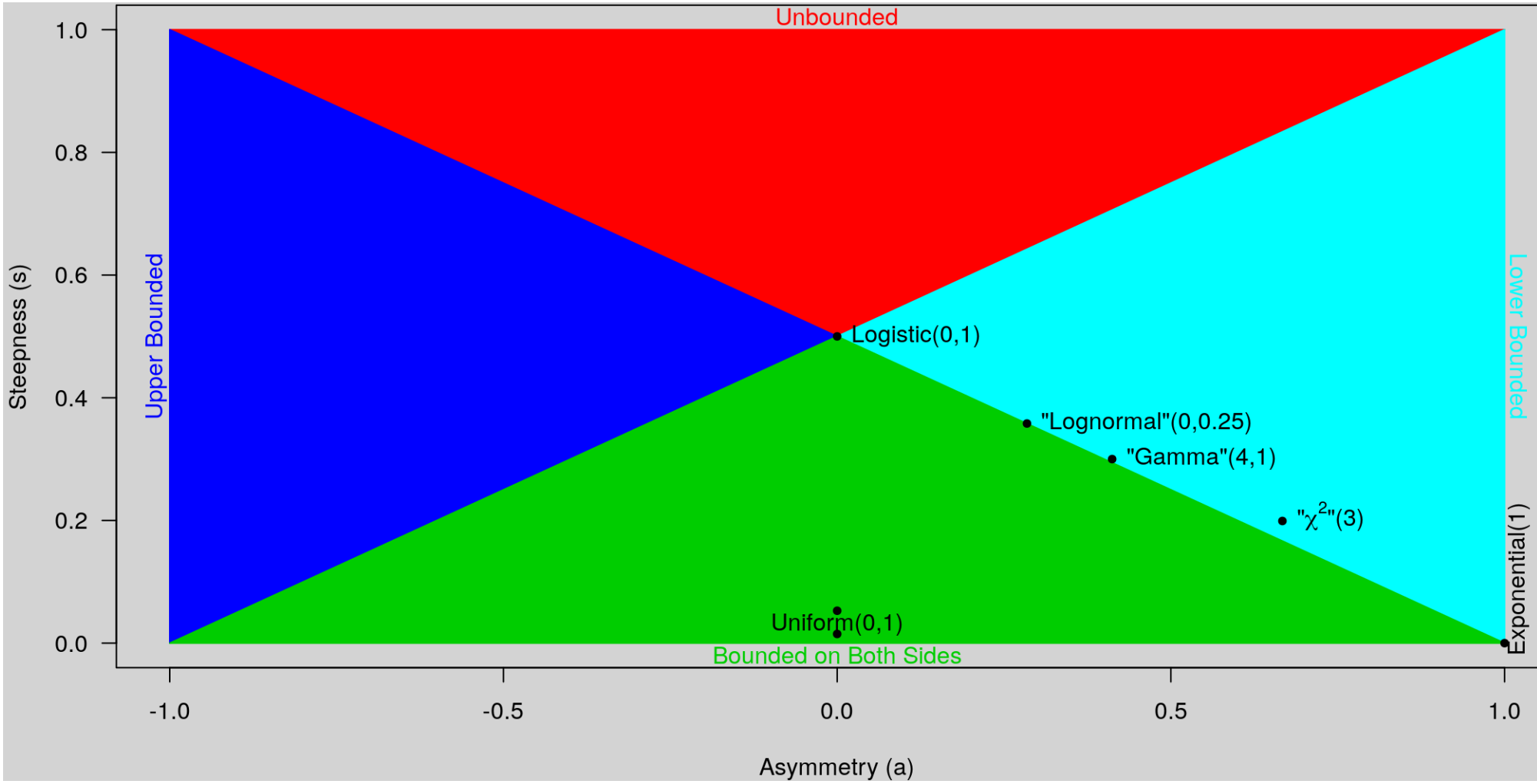
Special Cases of the GLD (for some m and r)



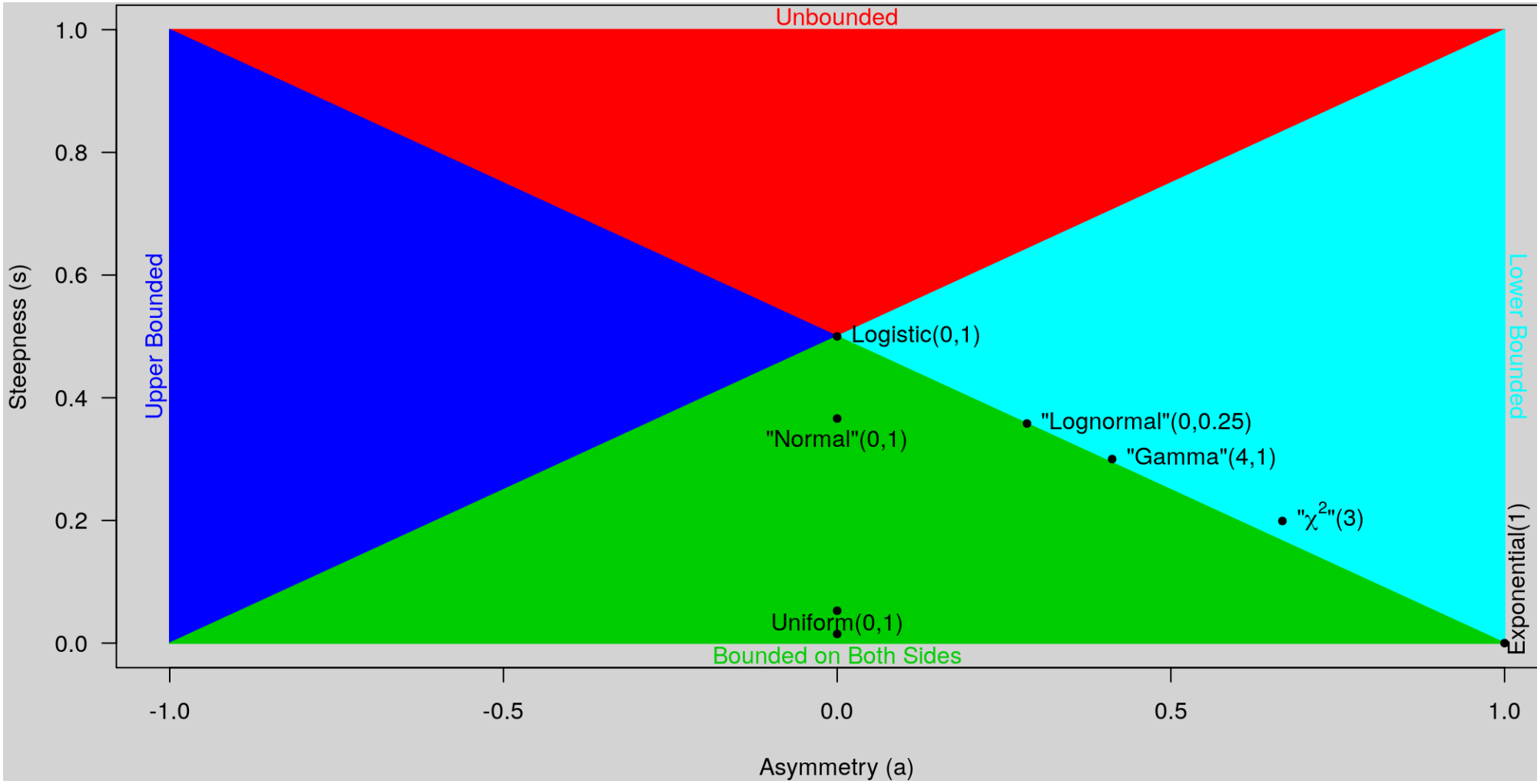
Special Cases of the GLD (for some m and r)



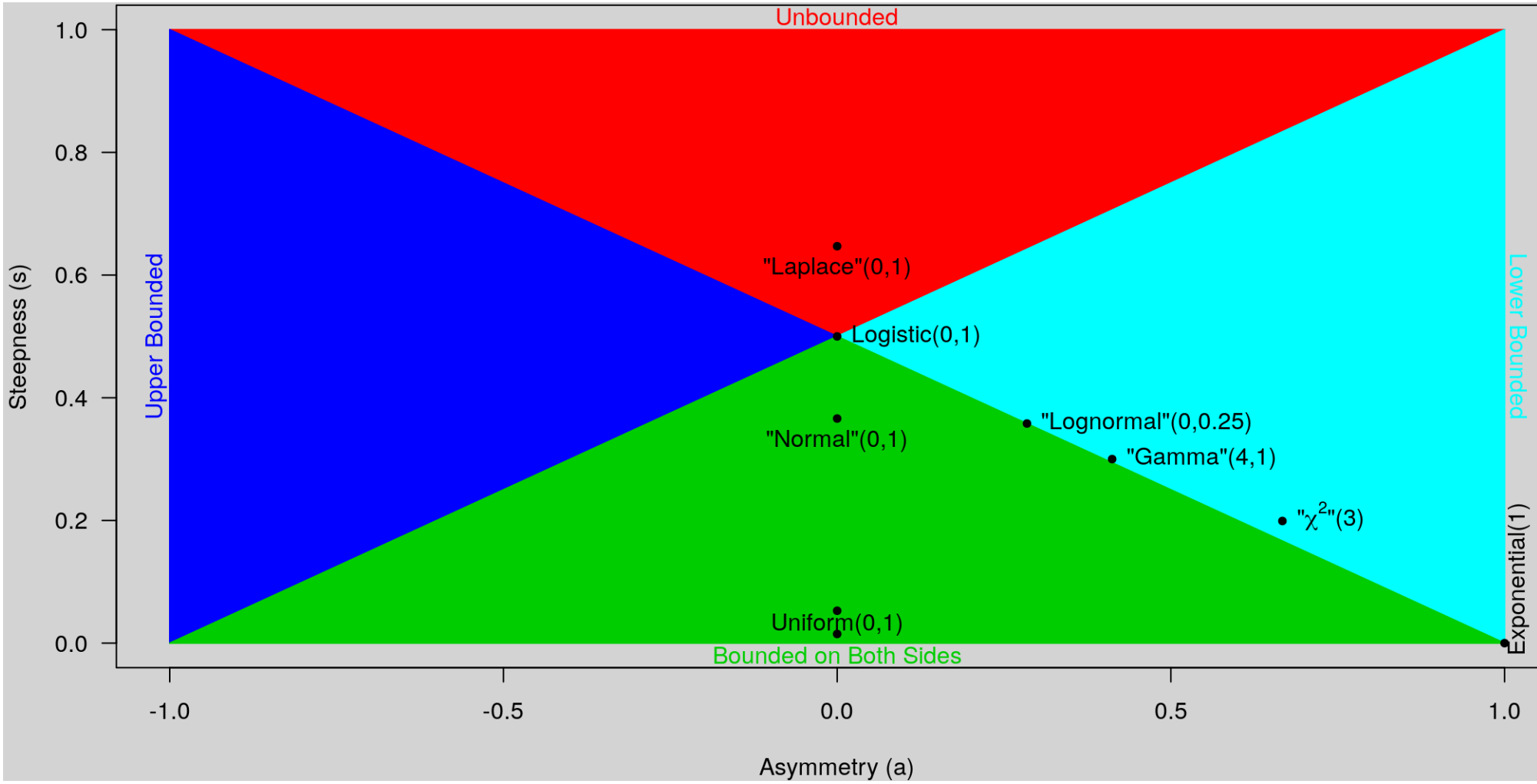
Special Cases of the GLD (for some m and r)



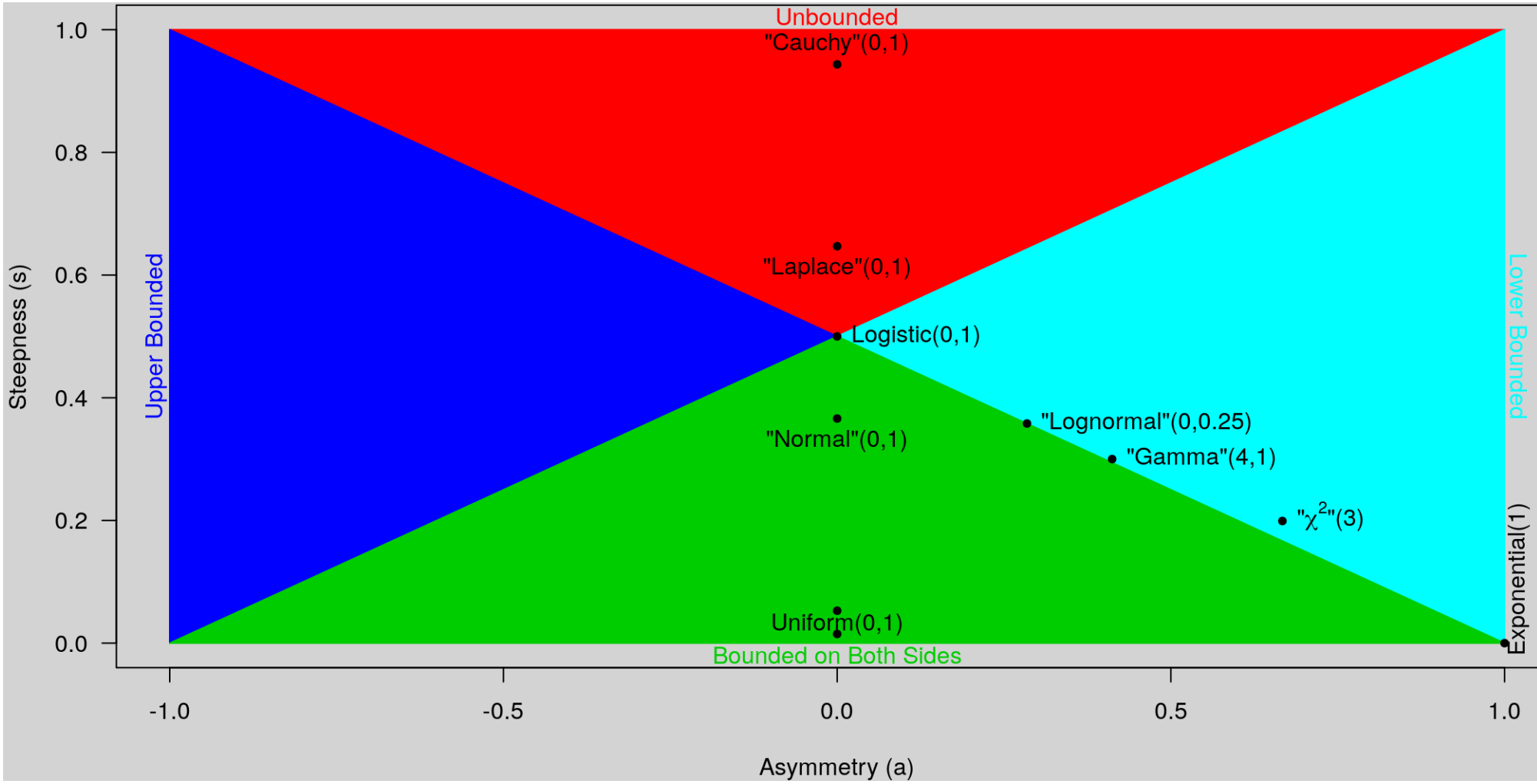
Special Cases of the GLD (for some m and r)



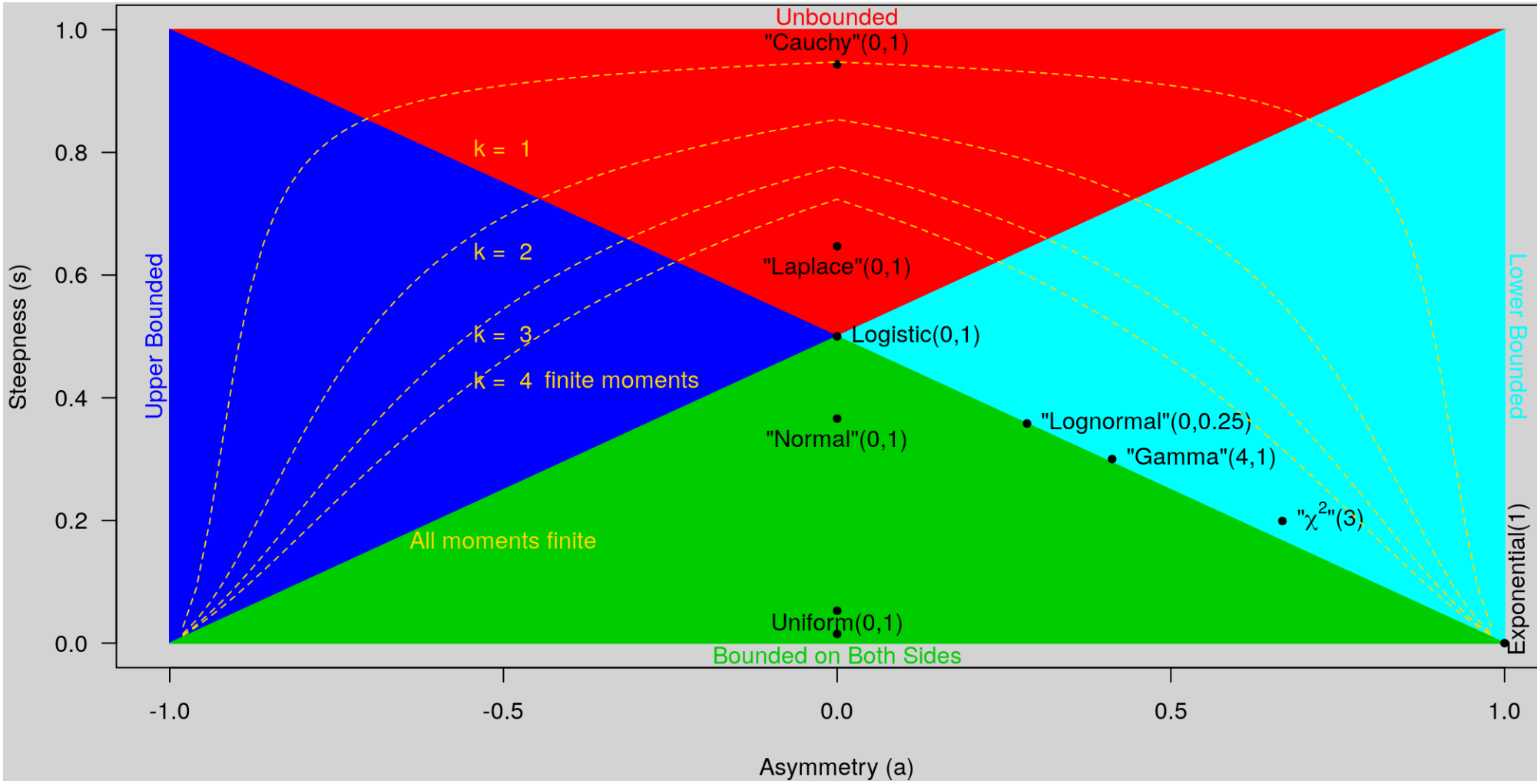
Special Cases of the GLD (for some m and r)



Special Cases of the GLD (for some m and r)



Special Cases of the GLD (for some m and r)



Using a GLD Prior for Vaccine Effectiveness

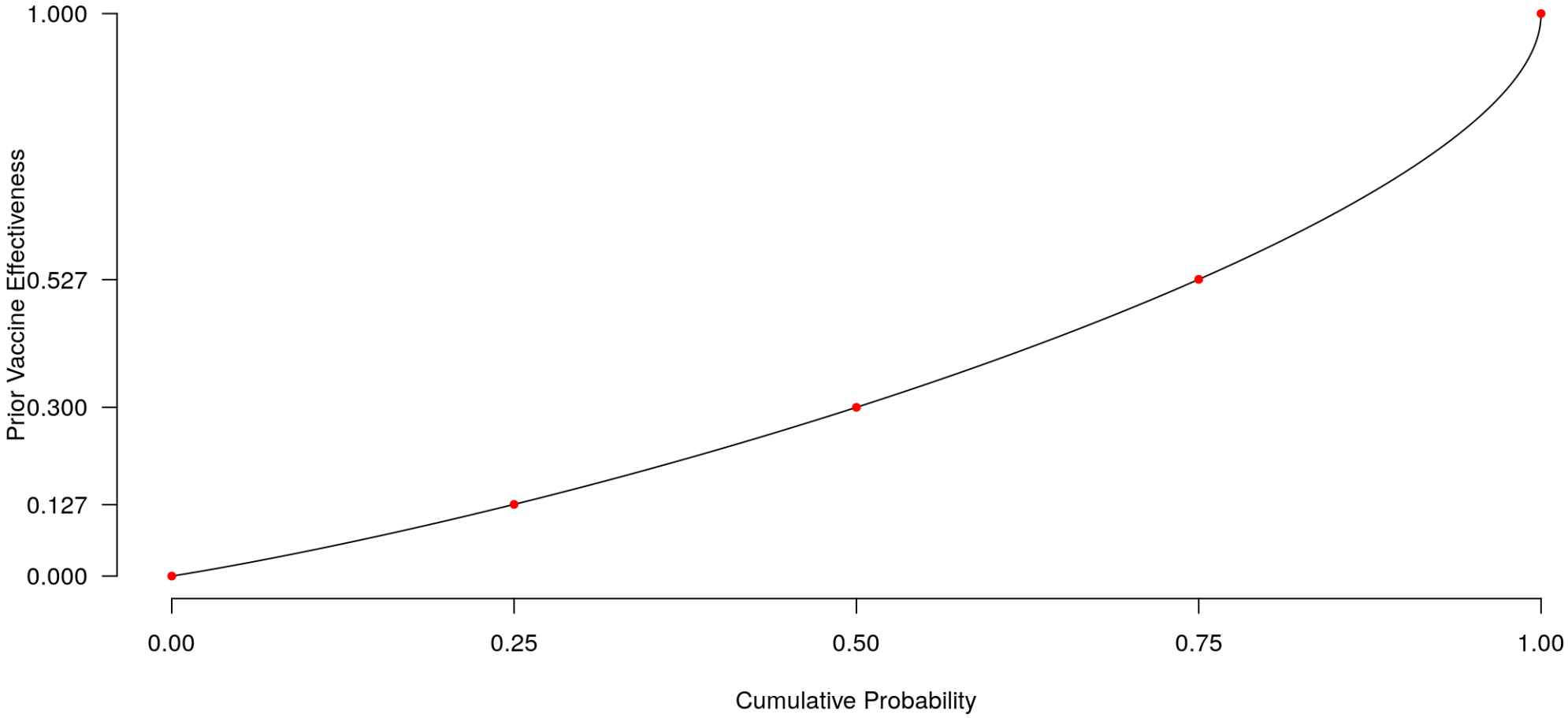
```
source("GLD_helpers.R") # defines GLD_solver_bounded() and related functions
(a_s <- GLD_solver_bounded(bounds = 0:1, median = 0.3, IQR = 0.4)) # note warning
```

```
##  asymmetry  steepness
## 0.73085563 0.05062537
```

```
p <- c(0, 0.25, 0.5, 0.75, 1)
VE <- qgld(p, median = 0.3, IQR = 0.4, asymmetry = a_s[1], steepness = a_s[2])
names(VE) <- p
VE
```

```
##             0          0.25           0.5          0.75             1
## 0.00002022707 0.12745854621 0.30000000000 0.52745854621 0.99999764199
```

Plot of Previous Prior Quantile Function



Prior Predictive Distribution

- The prior predictive distribution, which is the marginal distribution of future data integrated over the parameters, is formed by

  1. Draw $\tilde{\theta}$ from its prior distribution
  2. Draw $\tilde{y}$ from its conditional distribution given the realization of $\tilde{\theta}$
  3. Store the realization of $\tilde{y}$

```
theta <- qgld(runif(4000), median = 0.3, IQR = 0.1, a_s[1], a_s[2])
y <- rbinom(n = length(theta), size = n, prob = theta)
summary(y)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    8.00   24.00   29.00   29.29   34.00   54.00
```

- If you prior on θ is plausible, prior predictive distribution should be plausible

Prior Predictive Distribution Matching

- When the outcome is a small-ish count, a good algorithm to draw S times from the posterior distribution is to keep the realization of $\tilde{\theta}$ iff $\tilde{y} = y$

```
S <- 4000; VE <- rep(NA, S); s <- 1; tries <- 0
while (s <= S) {
  VE_ <- qgld(runif(1), median = 0.3, IQR = 0.4, asymmetry = a_s[1], steepness = a_s[2])
  theta_ <- (1 - VE_) / (2 - VE_) # theta_ is just an intermediate; VE is primitive
  y_ <- rbinom(1, size = n, prob = theta_) # draw outcome conditional on theta_
  if (y_ == y) { # check condition implied by observed outcome
    VE[s] <- VE_
    s <- s + 1
  } # otherwise do nothing
  tries <- tries + 1
}
summary(VE) # posterior summary of VE
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.5998  0.8581  0.8873  0.8817  0.9102  0.9693
```

Unbounded GLD Priors (in "GLD_helpers.R")

- $\mathrm{VE} = \frac{1-2\theta}{1-\theta}$ is negative if $\theta > \frac{1}{2}$, (i.e. the vaccine gives you covid). We could handle that possibility with an additional VE quantile, such as

```
(a_s <- GLD_solver_LBFGS(lower_quartile = 0.15, median = 0.3, upper_quartile = 0.55,
                         other_quantile = -0.5, alpha = 0.01)) # 1% chance VE < -0.5
```

```
## asymmetry steepness
## 0.6446718 0.8954998
```

- α can also be 0 or 1, making `other_quantile` a lower or upper bound

```
GLD_solver(lower_quartile = 0.15, median = 0.37, upper_quartile = 0.55,
           other_quantile = 1, alpha = 1) # GLD_solver_BFGS doesn't work well here
```

```
##  asymmetry  steepness
## -0.4050088  0.4063638
```

Four Ways to Execute Bayes Rule

1. Analytically integrate the kernel of Bayes Rule over the parameter(s)
    - Makes incremental Bayesian learning obvious but is only possible in simple models when the distribution of the outcome is in the exponential family
2. Numerically integrate the kernel of Bayes Rule over the parameter(s)
    - Most similar to what we did in the discrete case but is only feasible when there are few parameters and can be inaccurate even with only one
3. Draw from the prior predictive distribution and keep realizations of the parameters iff the realization of the outcome matches the observed data
    - Very intuitive what is happening but is only possible with discrete outcomes and only feasible with few observations and parameters
4. Perform MCMC (via Stan) to sample from the posterior distribution
    - Works for any posterior PDF that is differentiable w.r.t. the parameters

Bivariate Normal Distribution

The PDF of the bivariate normal distribution over $\Omega = \mathbb{R}^2$ is

$$f\left(x, y \middle| \mu_X, \mu_Y, \sigma_X, \sigma_Y, \rho\right) =$$

$$\frac{1}{2\pi\sigma_X\sigma_Y\sqrt{1-\rho^2}} e^{-\frac{1}{2\left(1-\rho^2\right)}\left(\left(\frac{x-\mu_X}{\sigma_X}\right)^2+\left(\frac{y-\mu_Y}{\sigma_Y}\right)^2-2\rho\frac{x-\mu_X}{\sigma_X}\frac{y-\mu_Y}{\sigma_Y}\right)} =$$

$$\frac{1}{\sigma_X\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu_X}{\sigma_X}\right)^2} \times \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{y-\left(\mu_y+\beta\left(x-\mu_X\right)\right)}{\sigma}\right)^2},$$

where X is MARGINALLY normal and $Y \middle| X$ is CONDITIONALLY normal with expectation $\mu_Y + \beta\left(x - \mu_X\right)$ and standard deviation $\sigma = \sigma_Y\sqrt{1-\rho^2}$, where $\beta = \rho\frac{\sigma_Y}{\sigma_X}$ is the OLS coefficient when Y is regressed on X and σ is the error standard deviation. We can thus draw $\tilde{x}$ and then condition on it to draw $\tilde{y}$.

Drawing from the Bivariate Normal Distribution

```
rbinorm <- function(n, mu_X, sigma_X, mu_Y, sigma_Y, rho) {
  x <- rnorm(n, mean = mu_X, sd = sigma_X)
  y <- rnorm(n, mean = mu_Y + rho * sigma_Y / sigma_X * (x - mu_X),
             sd = sigma_Y * sqrt((1 + rho) * (1 - rho)))
  return(cbind(x, y))
}
mu_X <- 0; mu_Y <- 0; sigma_X <- 1; sigma_Y <- 1; rho <- 0.75
indep <- replicate(26, colMeans(rbinorm(100, mu_X, sigma_X, mu_Y, sigma_Y, rho)))
rownames(indep) <- c("x", "y"); colnames(indep) <- letters
round(indep, digits = 3)
```

```
##        a      b      c     d      e      f     g     h     i      j     k     l     m
## x -0.030 -0.012 -0.075 0.094 -0.112 -0.048 0.006 0.117 0.078 -0.064 0.104 0.110 0.127
## y -0.043  0.043 -0.094 0.104 -0.124  0.052 0.091 0.118 0.000 -0.109 0.126 0.135 0.008
##        n      o      p     q     r      s      t      u     v      w     x     y     z
## x -0.050 -0.004 -0.111 0.056 0.035 -0.101 -0.065 -0.163 0.050 -0.051 0.035 0.108 0.194
## y  0.105  0.013 -0.052 0.052 0.034 -0.008 -0.119 -0.194 0.087 -0.016 0.001 0.193 0.214
```

Autoregressive Markov Processes

- A Markov process is a sequence of random variables with a particular dependence structure where the future is conditionally independent of the past given the present, but nothing is marginally independent of anything else
- An AR1 model is a linear Markov process: $x_t = \mu\left(1-\rho\right) + \rho x_{t-1} + \epsilon_t$
- As $T \uparrow \infty$, the T-th realization is distributed normal with expectation μ and standard deviation $\frac{\sigma}{\sqrt{1-\rho^2}}$, where σ is the standard deviation of ϵ_t, as in

```
T <- 500; S <- 1000; x_T <- replicate(S, {
  x <- rpois(n = 1, 1)
  for (t in 1:T) x <- mu_X * (1 - rho) + rho * x + rnorm(n = 1, mean = 0, sd = sigma_X)
  return(x)
})
c(mean_diff = mean(x_T) - mu_X, sd_diff = sd(x_T) - sigma_X / sqrt(1 - rho^2))
```

```
##  mean_diff    sd_diff
## 0.02381946 0.04613637
```

General Markov Processes

- Let X_t have conditional PDF $f_t\left(X_t \middle| X_{t-1}\right)$. Their joint PDF is

$$f\left(X_0, X_1, \ldots, X_{T-1}, X_T\right) = f_0\left(X_0\right) \prod_{t=1}^{T} f_t\left(X_t \middle| X_{t-1}\right),$$

  but we usually consider a special case where $f_t\left(X_t \middle| X_{t-1}\right) = f\left(X_t \middle| X_{t-1}\right)$
- Can we construct a (homogeneous) Markov process such that the marginal distribution of X_T has a sought after distribution as $T \uparrow \infty$?
- Yes, although generally with a nonlinear, homogeneous Markov process
- If so, they you can get a random draw — or a set of S dependent draws — from the target distribution by letting that Markov process run for a long time
- Basic idea is that you can marginalize by going through a lot of conditionals
- Metropolis, Gibbs, Stan, and others all satisfy this as $T \uparrow \infty$

Metropolis-Hastings Markov Chain Monte Carlo

- Suppose you want to draw from some distribution whose PDF is $f\left(\boldsymbol{\theta}\middle|\ldots\right)$ but do not have a customized algorithm to do so
- Initialize $\boldsymbol{\theta}$ to some value in Θ and then repeat S times:
  1. Draw a proposal for $\boldsymbol{\theta}$, say $\boldsymbol{\theta}'$, from a distribution whose PDF is $q\left(\boldsymbol{\theta}'\middle|\ldots\right)$
  2. Let $\alpha^* = \min\{1, \frac{f(\boldsymbol{\theta}'|\ldots)}{f(\boldsymbol{\theta}|\ldots)}\frac{q(\boldsymbol{\theta}|\ldots)}{q(\boldsymbol{\theta}'|\ldots)}\}$. N.B.: Constants cancel so not needed!
  3. If α^* is greater than a standard uniform variate, set $\boldsymbol{\theta} = \boldsymbol{\theta}'$
  4. Store $\boldsymbol{\theta}$ as the s-th draw
- The S draws of $\boldsymbol{\theta}$ have PDF $f\left(\boldsymbol{\theta}\middle|\ldots\right)$ but are NOT independent
- If $\frac{q(\boldsymbol{\theta}|\ldots)}{q(\boldsymbol{\theta}'|\ldots)} = 1$, called Metropolis MCMC such as $q\left(\theta \mid a, b\right) = \frac{1}{b-a}$

Efficiency in Estimating $\mathbb{E}X$ & $\mathbb{E}Y$ w/ Metropolis

```
means <- replicate(26, colMeans(Metropolis(S, 2.75, mu_X, sigma_X, mu_Y, sigma_Y, rho)))
rownames(means) <- c("x", "y"); colnames(means) <- LETTERS; round(means, digits = 3)
```

```
##        A      B     C      D     E     F      G      H      I     J     K     L     M
## x -0.139 -0.062 0.087 -0.050 0.043 0.058 -0.078  0.004 -0.110 0.157 0.019 0.051 -0.12
## y -0.162  0.028 0.184 -0.102 0.064 0.025 -0.036 -0.051 -0.055 0.033 0.001 0.098 -0.03
##       N    O      P      Q      R     S     T      U      V      W      X     Y      Z
## x 0.325 0.02  0.045  0.097 -0.022 0.087 0.069 -0.038 -0.072 -0.185 -0.150 0.067 -0.175
## y 0.320 0.09 -0.040 -0.016 -0.080 0.186 0.091  0.008  0.005 -0.162 -0.031 0.091 -0.048
```

```
round(indep, digits = 3) # note S was 100 before, rather than 1000
```

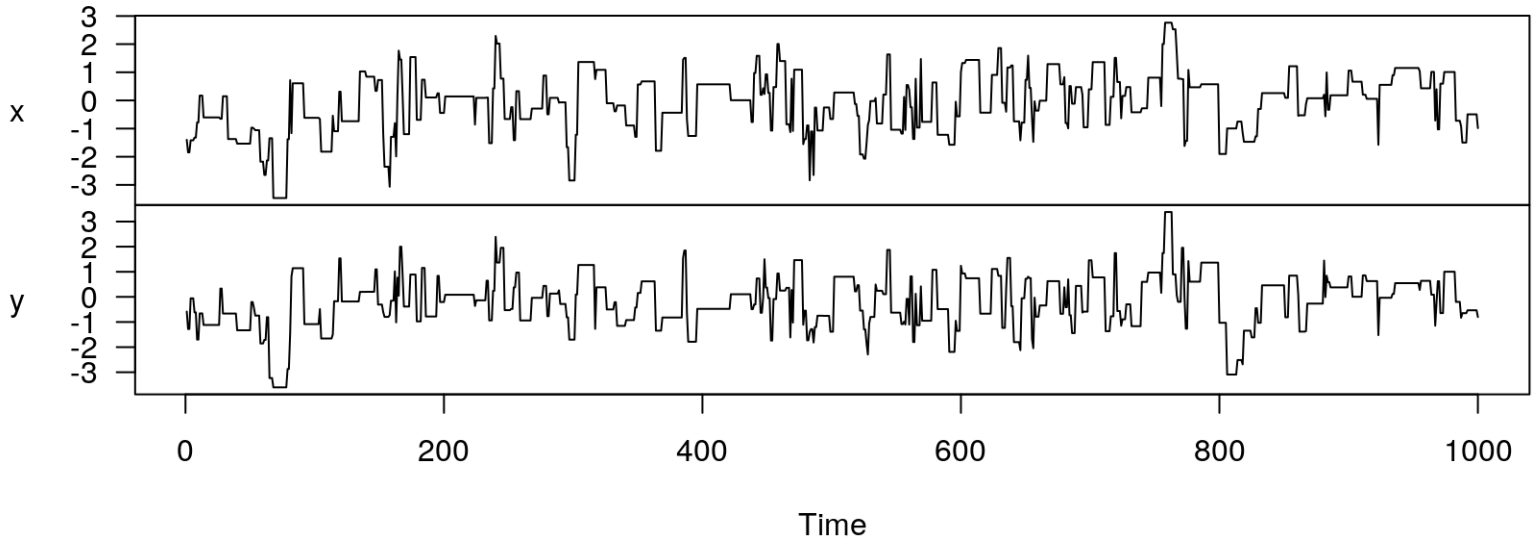
```
##        a      b      c     d      e      f     g     h     i      j     k     l     m
## x -0.030 -0.012 -0.075 0.094 -0.112 -0.048 0.006 0.117 0.078 -0.064 0.104 0.110 0.127
## y -0.043  0.043 -0.094 0.104 -0.124  0.052 0.091 0.118 0.000 -0.109 0.126 0.135 0.008
##        n      o      p     q     r      s      t      u     v      w     x     y     z
## x -0.050 -0.004 -0.111 0.056 0.035 -0.101 -0.065 -0.163 0.050 -0.051 0.035 0.108 0.194
## y  0.105  0.013 -0.052 0.052 0.034 -0.008 -0.119 -0.194 0.087 -0.016 0.001 0.193 0.214
```

Autocorrelation of Metropolis MCMC

```
xy <- Metropolis(S, 2.75, mu_X, sigma_X, mu_Y, sigma_Y, rho); nrow(unique(xy))
```

```
## [1] 245
```

```
colnames(xy) <- c("x", "y"); plot(as.ts(xy), main = "")
```



Effective Sample Size of Markov Chain Output

- If a Markov Chain mixes fast enough for the MCMC CLT to hold, then
  - The Effective Sample Size is $n_{eff} = \frac{S}{1+2\sum_{n=1}^{\infty}\rho_n}$, where ρ_n is the ex ante autocorrelation between two draws that are n iterations apart
  - The MCMC Standard Error of the mean of the S draws is $\frac{\sigma}{\sqrt{n_{eff}}}$ where σ is the true posterior standard deviation
- If $\rho_n = 0 \forall n$, then $n_{eff} = S$ and the MCMC-SE is $\frac{\sigma}{\sqrt{S}}$, so the Effective Sample Size is the number of INDEPENDENT draws that would be expected to estimate the posterior mean of some function with the same accuracy as the S DEPENDENT draws that you have from the posterior distribution
- Both have to be estimated and unfortunately, the estimator is not that reliable when the true Effective Sample Size is low (~5% of S)
- For the Metropolis example, n_{eff} is estimated to be ≈ 100 for both margins

Gibbs Samplers

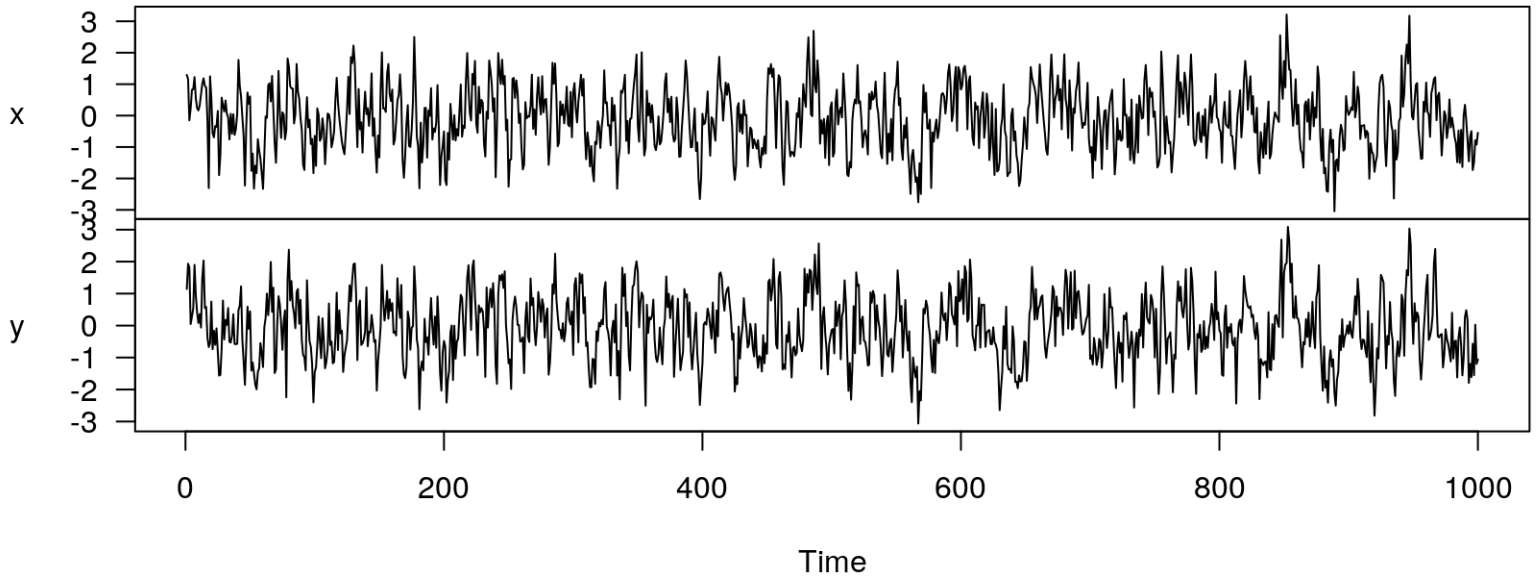
- Metropolis-Hastings where $q\left(\theta_k' \middle| \ldots\right) = f\left(\theta_k' \middle| \boldsymbol{\theta}_{-k} \ldots\right)$ and $\boldsymbol{\theta}_{-k}$ consists of all elements of $\boldsymbol{\theta}$ except the k-th
- $\alpha^* = \min\{1, \frac{f(\boldsymbol{\theta}'|\ldots)}{f(\boldsymbol{\theta}|\ldots)} \frac{f(\theta_k|\boldsymbol{\theta}_{-k}\ldots)}{f(\theta_k'|\boldsymbol{\theta}_{-k}\ldots)}\} = \min\{1, \frac{f(\theta_k'|\boldsymbol{\theta}_{-k}\ldots)f(\boldsymbol{\theta}_{-k}|\ldots)}{f(\theta_k|\boldsymbol{\theta}_{-k}\ldots)f(\boldsymbol{\theta}_{-k}|\ldots)} \frac{f(\theta_k|\boldsymbol{\theta}_{-k}\ldots)}{f(\theta_k'|\boldsymbol{\theta}_{-k}\ldots)}\} = 1$ so θ_k' is ALWAYS accepted by construction. But θ_k' may be very close to θ_k when the variance of the "full-conditional" distribution of θ_k' given $\boldsymbol{\theta}_{-k}$ is small
- Can loop over k to draw sequentially from each full-conditional distribution
- Presumes that there is an algorithm to draw from the full-conditional distribution for each k. Most times have to fall back to something else.

Gibbs Sampling from the Bivariate Normal

```
Gibbs <- function(S, mu_X, sigma_X, mu_Y, sigma_Y, rho) {
  draws <- matrix(NA_real_, nrow = S, ncol = 2)
  x <- rpois(n = 1, 1) # arbitrary starting value
  beta   <- rho * sigma_Y / sigma_X
  lambda <- rho * sigma_X / sigma_Y
  sqrt1mrho2 <- sqrt( (1 + rho) * (1 - rho) )
  sigma_XY <- sigma_X * sqrt1mrho2 # these are both smaller than the
  sigma_YX <- sigma_Y * sqrt1mrho2 # marginal standard deviations!!!
  for (s in 1:S) { # draw from each CONDITIONAL distribution
    y <- rnorm(n = 1, mean = mu_Y + beta *   (x - mu_X), sd = sigma_YX)
    x <- rnorm(n = 1, mean = mu_X + lambda * (y - mu_Y), sd = sigma_XY)
    draws[s, ] <- c(x, y)
  }
  return(draws)
}
```

Autocorrelation of Gibbs Sampling: $n_{eff} \approx 300$

```
xy <- Gibbs(S, mu_X, sigma_X, mu_Y, sigma_Y, rho)
colnames(xy) <- c("x", "y")
plot(as.ts(xy), main = "")
```

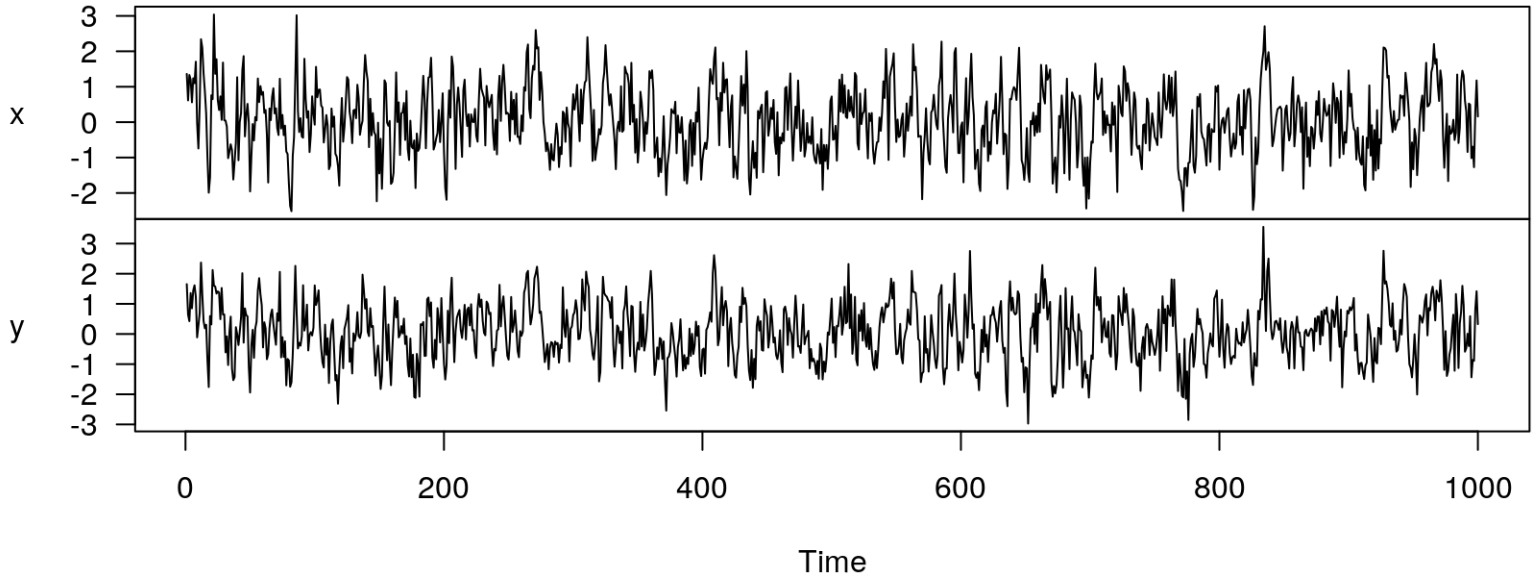


What the BUGS Software Family Essentially Does

```
library(Runuran) # defines ur() which draws from the approximate ICDF via pinv.new()
BUGSish <- function(log_kernel, # function of theta outputting posterior log-kernel
                    theta,      # starting values for all the parameters
                    ...,        # additional arguments passed to log_kernel()
                    LB = rep(-Inf, J), UB = rep(Inf, J), # optional bounds on theta
                    S = 1000) { # number of posterior draws to obtain
  J <- length(theta); draws <- matrix(NA_real_, nrow = S, ncol = J)
  for(s in 1:S) { # these loops are slow, as is approximating the ICDF of theta[-k]
    for (j in 1:J) {
      full_conditional <- function(theta_j)
        return(log_kernel(c(head(theta, j - 1), theta_j, tail(theta, J - j)), ...))
      theta[j] <- ur(pinv.new(full_conditional, lb = LB[j], ub = UB[j], islog = TRUE,
                              uresolution = 1e-8, smooth = TRUE, center = theta[j]))
    }
    draws[s, ] <- theta
  }
  return(draws)
}
```

Gibbs Sampling a la BUGS: $n_{eff} \approx 200$

```
xy <- BUGSish(log_kernel = dbinorm, theta = c(x = -1, y = 1), mu_X, sigma_X,
              mu_Y, sigma_Y, rho, log = TRUE)
colnames(xy) <- c("x", "y")
plot(as.ts(xy), main = "")
```



Comparing Stan to Archaic MCMC Samplers

- Stan only requires user to specify kernel of Bayes Rule
- Unlike Gibbs sampling, proposals are joint
- Like Gibbs sampling, proposals always accepted
- Like Gibbs sampling, tuning of proposals is (often) not required
- Unlike Gibbs sampling, the effective sample size is typically 25% to 125% of the nominal number of draws from the posterior distribution because ρ_1 can be negative in $n_{eff} = \frac{S}{1+2\sum_{n=1}^{\infty}\rho_n}$
- Unlike Gibbs sampling, Stan produces warning messages when things are not going swimmingly. Do not ignore these (although we did on HW3)!
- Unlike BUGS, Stan does not permit discrete unknowns but even BUGS has difficulty drawing discrete unknowns with a sufficient amount of efficiency

Differentiating the Log Posterior Kernel

- Stan always works with log-PDFs or really log-kernels (in $\boldsymbol{\theta}$)

$$\ln f\left(\boldsymbol{\theta} \mid \mathbf{y}, \ldots\right) = \ln f\left(\boldsymbol{\theta} \mid \ldots\right) + \ln L\left(\boldsymbol{\theta}; \mathbf{y}\right) - \ln f\left(\mathbf{y} \mid \ldots\right)$$

- The gradient of the log posterior PDF is the gradient of the log-kernel

$$\boldsymbol{\nabla} \ln f\left(\boldsymbol{\theta} \mid \mathbf{y}, \ldots\right) = \boldsymbol{\nabla} \ln f\left(\boldsymbol{\theta} \mid \ldots\right) + \boldsymbol{\nabla} \ln L\left(\boldsymbol{\theta}; \mathbf{y}\right) + \mathbf{0}$$

- This gradient is basically exact, and the chain rule can be executed by a C++ compiler without the user having to compute any derivatives

Hamiltonian Monte Carlo

- Stan pairs the J "position" variables $\boldsymbol{\theta}$ with J "momentum" variables $\boldsymbol{\phi}$ and draws from the joint posterior distribution of $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$
- Since the likelihood is NOT a function of ϕ_j, the posterior distribution of ϕ_j is the same as its prior, which is normal with a "tuned" standard deviation. So, at the s-th MCMC iteration, we just draw each $\widetilde{\phi}_j$ from its normal distribution.
- Using physics, the realizations of each $\widetilde{\phi}_j$ at iteration s "push" $\boldsymbol{\theta}$ from iteration $s-1$ for a random amount of time through the parameter space whose topology is defined by the (negated) log-kernel of the posterior distribution
- Although the ODEs must be solved numerically, the integral in "time" is one-dimensional and there are very good customized numerical integrators

Hamiltonian Monte Carlo

- Instead of simply drawing from the posterior distribution whose PDF is $f\left(\boldsymbol{\theta}|\,\mathbf{y}\ldots\right) \propto f\left(\boldsymbol{\theta}\right) L\left(\boldsymbol{\theta};\mathbf{y}\right)$ Stan augments the "position" variables $\boldsymbol{\theta}$ with an equivalent number of "momentum" variables $\boldsymbol{\phi}$ and draws from

$$f\left(\boldsymbol{\theta}|\,\mathbf{y}\ldots\right) \propto \int_{-\infty}^{\infty} \ldots \int_{-\infty}^{\infty} \prod_{k=1}^{K} \frac{1}{\sigma_k\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\phi_k}{\sigma_k}\right)^2} f\left(\boldsymbol{\theta}\right) L\left(\boldsymbol{\theta};\mathbf{y}\right) d\phi_1 \ldots d\phi_K$$

- Since the likelihood is NOT a function of ϕ_k, the posterior distribution of ϕ_k is the same as its prior, which is normal with a "tuned" standard deviation. So, at the s-th MCMC iteration, we just draw each $\widetilde{\phi}_k$ from its normal distribution.
- Using physics, the realizations of each $\widetilde{\phi}_k$ at iteration s "push" $\boldsymbol{\theta}$ from iteration $s-1$ through the parameter space whose topology is defined by the negated log-kernel of the posterior distribution: $-\ln f\left(\boldsymbol{\theta}\right) - \ln L\left(\boldsymbol{\theta};\mathbf{y}\right)$
- See HMC.R demo and next slide

Demo of Hamiltonian Monte Carlo

No U-Turn Sampling (NUTS)

- The location of $\boldsymbol{\theta}$ moving according to Hamiltonian physics at any instant would be a valid draw from the posterior distribution
- But (in the absence of friction) $\boldsymbol{\theta}$ moves indefinitely so when do you stop?
- Hoffman and Gelman (2014) proposed stopping when there is a "U-turn" in the sense the footprints turn around and start to head in the direction they just came from. Hence, the name No U-Turn Sampling.
- After the U-Turn, one footprint is selected with probability proportional to the posterior kernel to be the realization of $\boldsymbol{\theta}$ on iteration s and the process repeates itself
- NUTS discretizes a continuous-time Hamiltonian process in order to solve a system of Ordinary Differential Equations (ODEs), which requires a stepsize that is also tuned during the warmup phase
- Video and R code

Using Stan via R

1. Write the program in a (text) .stan file w/ R-like syntax that ultimately defines a posterior log-kernel. Stan's parser, `rstan::stanc`, does two things:
   - checks that program is syntactically valid and tells you if not
   - writes a conceptually equivalent C++ source file to disk
2. C++ compiler creates a binary file from the C++ source
3. Execute the binary from R (can be concurrent with 2)
4. Analyze the resulting samples from the posterior
   - Posterior predictive checks
   - Model comparison
   - Decision

A Better Model for Vaccine Effectiveness

```
#include quantile_functions.stan
data { /* these are known and passed as a named list from R */
  int<lower = 0> n;                           // number of tests
  int<lower = 0, upper = n> y;                // number of positives
  real m;
  real<lower = 0> IQR;
  real<lower = -1, upper = 1> asymmetry;
  real<lower =  0, upper = 1> steepness;
}
parameters { /* these are unknowns whose posterior distribution is sought */
  real<lower = 0, upper = 1> p;             // CDF of positive test probability
}
transformed parameters { /* deterministic unknowns that get stored in RAM */
  real VE = GLD_qf(p, m, IQR, asymmetry, steepness); // positive probability
} // this function ^^^ is defined in the quantile_functions.stan file
model { /* log-kernel of Bayes' Rule that essentially returns "target" */
  target += binomial_lpmf(y | n, (VE - 1) / (VE - 2)); // log-likelihood
} // implicit: p ~ uniform(0, 1) <=> theta ~ GLD(m, IQR, asymmetry, steepness)
```

Drawing from a Posterior Distribution with NUTS

```
library(rstan)
post <- stan("coronavirus.stan",
             data = list(n = 94, y = 8, m = 0.3, IQR = 0.1,
                         asymmetry = a_s[1], steepness = a_s[2]))


post


## Inference for Stan model: coronavirus.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##       mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## p     0.96    0.00 0.00  0.95  0.95  0.96  0.96  0.96   967    1
## VE    0.89    0.00 0.04  0.81  0.87  0.89  0.92  0.95  1012    1
## lp__ -5.59    0.02 0.68 -7.49 -5.76 -5.33 -5.15 -5.10  1054    1
##
## Samples were drawn using NUTS(diag_e) at Wed Mar 30 13:21:54 2022.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

Warnings You Should Be Aware Of

1. Divergent Transitions: This means the tuned stepsize ended up too big relative to the curvature of the log-kernel. Increase `adapt_delta` above its default value (usually 0.8) and / or use more informative priors
2. Hitting the maximum treedepth: This means the tuned stepsize ended up so small that it could not get all the way around the parameter space in one iteration. Increase `max_treedepth` beyond its default value of 10 but each increment will double the wall time, so only do so if you hit the max a lot
3. Bulk / Tail Effective Sample Size too low: This means the tuned stepsize ended up so small that adjacent draws have too much dependence. Increase the number of iterations or chains
4. $\widehat{R} > 1.01$: This means the chains have not converged. You could try running the chains longer, but there is probably a deeper problem.
5. Low Bayesian Fraction of Information: This means that you posterior distribution has really extreme tails. You could try running the chains longer, but there is probably a deeper problem.