Monte Carlo Integration

Michael Rose

November 28, 2018

3.1 | Introduction

Two major classes of numerical problems in statistical inference are *optimization* and *integration.* These are in contrast to the analytical methods of computing estimators like maximum likelihood, bayes, method of moments, and others.

A general solution to the numerical solutions needed for analytically intractable integrals is to use simulations, of either the true or some substitute distributions, to calculate the quantities of interest.

Note that the possibility of producing an almost infinite number of random variables distributed according to a given distribution gives us access to the use of frequentist and asymptotic results much more easily than in the usual inferential settings, where the sample size is most often fixed.

Before this, it is worth noting the alternative to monte carlo integration: numerical methods like simpsons and the trapezoidal rule. R also provides functions for unidimensional integration, area() which channot handle infinite bounds, and integrate() which can, but is very fragile.

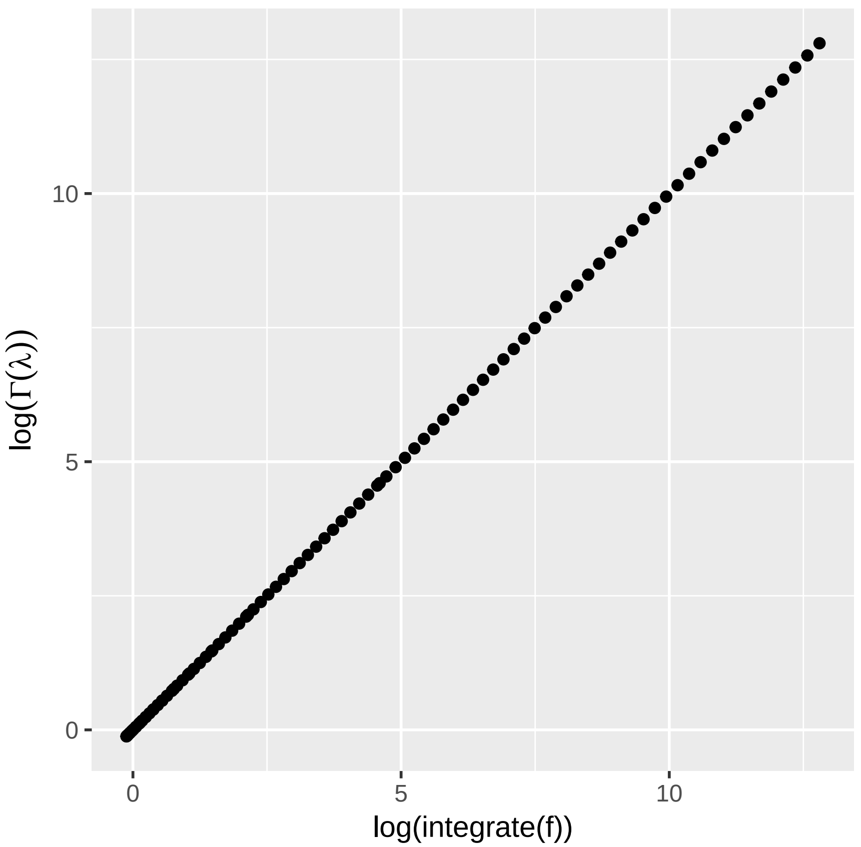
Example 3.1

As a test, lets compare the use of integrate() on the integral

$$\int_0^\infty x^{\lambda-1}\exp(-x)dx$$

with the computation of $\Gamma(\lambda)$ via the gamma() function.

```r
# create function to get integrate() values
int_gam <- function(lambda) {
    integrate(function(x) {
        x^(lambda - 1) * exp(-x)
    }, 0, Inf)$val
}

# make data frames
log_int_gam <- seq(0.01, 10, length.out = 100) %>% sapply(int_gam) %>%
    log() %>% tibble(output = .)
log_gam_fun <- seq(0.01, 10, length.out = 100) %>% sapply(lgamma) %>% tibble(output = .)

# plot
ggplot() + geom_point(aes(x = log_int_gam$output, y = log_gam_fun$output)) +
    coord_fixed() + xlab("log(integrate(f))") + ylab(expression(log(Gamma(lambda))))
```



The figure above shows that there is not a discrepancy, even for small values of lambda.

A main difficulty with numerical integration methods like integrate() is that they often fail to spot the region of importance for the function to be integrated. In contrast, simulation methods naturally tend to target this region by emploiting the information provided by the probability density associated with the integrals.

Example 3.2

Consider a sample of ten Cauchy RVs x_i, $1 \leq i \leq 10$ with a location parameter $\theta = 350$. Then the pseudo-marginal distribution under a flat prior is

$$m(x) = \int_{-\infty}^{\infty} \prod_{i=1}^{10} \frac{1}{\pi} \frac{1}{1+(x_i-\theta)^2} d\theta.$$

However, integrate() gives the wrong value and fails to signal the difficulty since the error evaluation is absurdly small:

```
# define 10 cauchys
ten_cauchy <- rcauchy(10) + 350

# take the product of ten cauchys
likelihood <- function(theta) {
    u <- dcauchy(ten_cauchy[1] - theta)
    for (i in 2:10) {
        u <- u * dcauchy(ten_cauchy[i] - theta)
    }
    return(u)
}

# integrate product over +- inf
integrate(likelihood, -Inf, Inf)
```

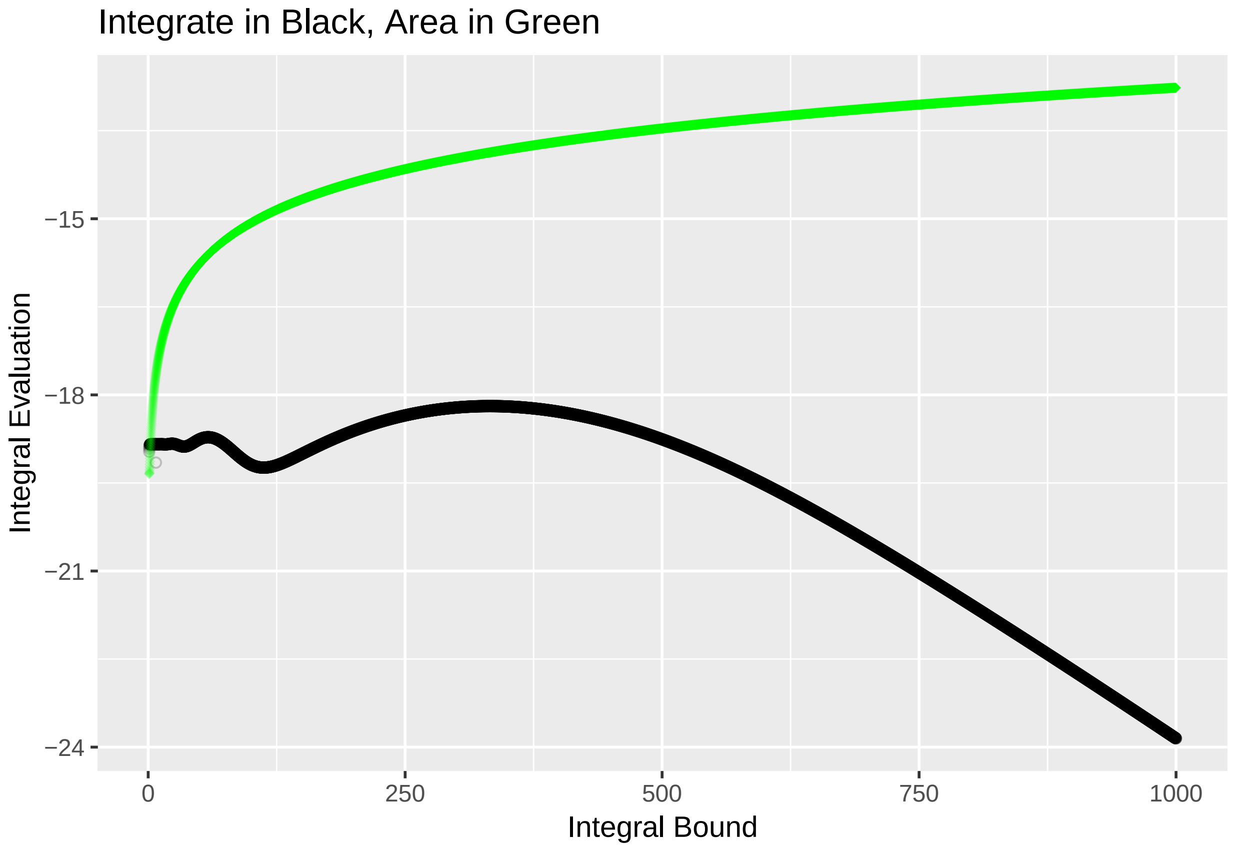
```
## 8.28808e-44 with absolute error < 1.6e-43
```

Furthermore, the result is not comparable to area:

```
# define 10 cauchys
ten_cauchy <- rcauchy(10)

# create integrate() function
cauc_int <- function(a) {
    integrate(likelihood, -a, a)$val
}

# create area() function
cauc_area <- function(a) {
    MASS::area(likelihood, -a, a)
}

# create vector of thetas
thetas <- seq(1, 10^3, length.out = 10^4)

# make dataframes
cauc_integrate <- thetas %>% sapply(cauc_int) %>% log() %>% tibble(output = .)
cauc_area <- thetas %>% sapply(cauc_area) %>% log() %>% tibble(output = .)

# find dataframe range
bounds <- cbind(cauc_area, cauc_integrate) %>% range()
bounds
```

```
## [1] -23.85240 -12.76762
```

```
ggplot() + geom_point(aes(x = thetas, y = cauc_integrate$output), shape = 1,
    alpha = 0.2) + geom_point(aes(x = thetas, y = cauc_area$output), shape = 18,
    color = "green", alpha = 0.2) + ylim(bounds) + xlab("Integral Bound") +
    ylab("Integral Evaluation") + ggtitle("Integrate in Black, Area in Green")
```



We can see that using area() in this case produces a more reliable evaluation, since it flattens out as a increases.

3.2 | Classical Monte Carlo Integration

Suppose we wish to evaluate the integral

$$\mathbb{E}_f[h(X)] = \int_{\chi} h(x)f(x)dx$$

where χ denotes the set where the random variable X takes its values, which is usually equal to the support of the density f.

The principle of the Monte Carlo method is to generate a sample $(X_1, ..., X_n)$ from the density f and propose as an approximation the empirical average

$$\bar{h_n} = \frac{1}{n}\sum_{j=1}^{n} h(x_j)$$

computed by mean(h(x)) in R, since $\bar{h_n}$ converges almost surely (i.e. for almost every generated sequence) to $\mathbb{E}_f[h(X)]$ by the Strong Law of Large Numbers.

When $h^2(X)$ has a finite expectation under f, the speed of convergence of $\bar{h_n}$ can be assessed since the convergence takes place at a rate of $O(\sqrt{n}$ and the asymptotic variance of the approximation is

$$\text{var}(\bar{h_n}) = \frac{1}{n}\int_{\chi}(h(x) - \mathbb{E}_f[h(X)])^2 f(x)dx$$

which can also be estimated from the sample $(X_1, ..., X_n)$ through

$$v_n = \frac{1}{n^2}\sum_{j=1}^{n}[h(x_j) - \bar{h_n}]^2$$

More specifically, due to the **Central Limit Theorem**, for large n,

$$\frac{\bar{h_n} - \mathbb{E}_f[h(X)]}{\sqrt{v_n}} \sim \mathcal{N}(0,1)$$

which leads to the construction of a convergence test and confidence bounds on the approximation of $\mathbb{E}_f[h(X)]$.

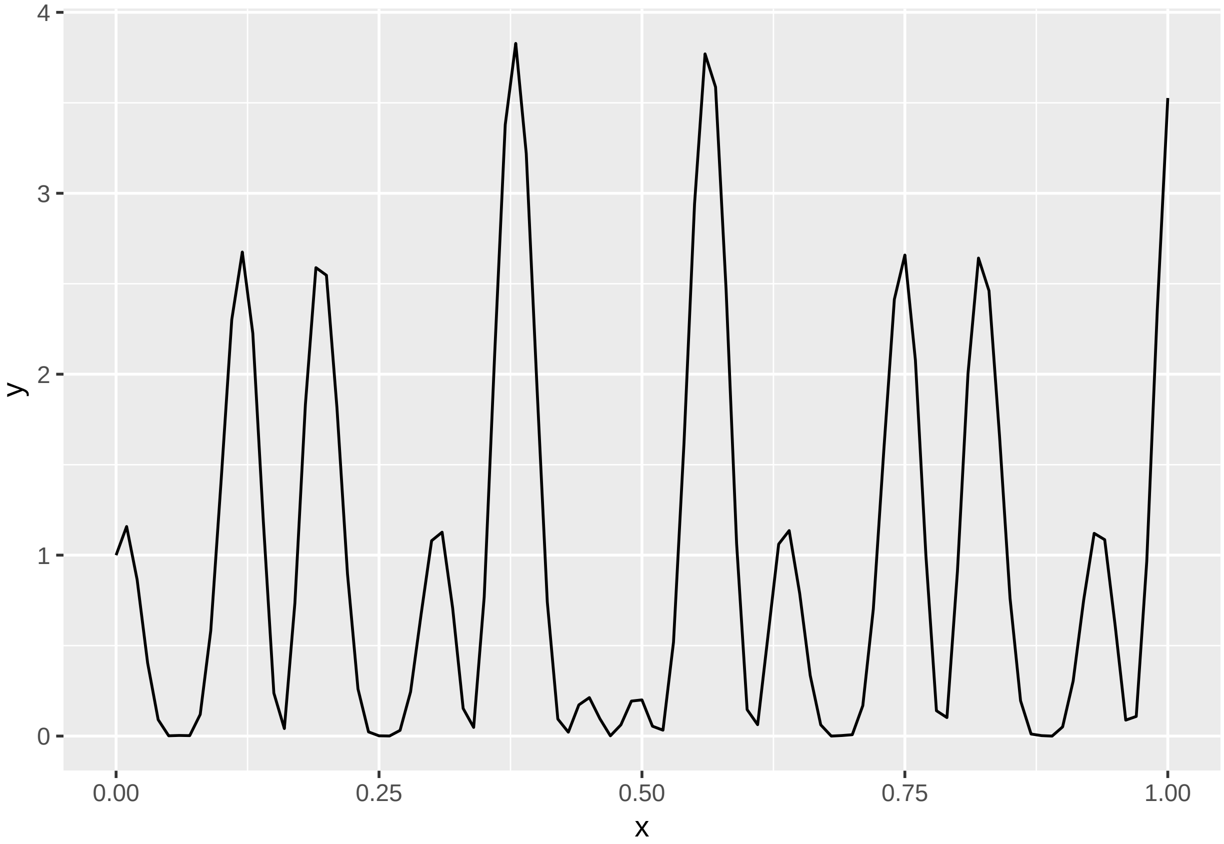
Example 3.3

Given $h(x) = [\cos(50x) + \sin(20x)]^2$, consider evaluating its integral over $[0,1]$. It can be seen as a uniform expectation, so we can generate $U_1, U_2, ..., U_n$ iid $\mathcal{U}(0,1)$ random variables and approximate $\int h(x)dx$ with $\sum \frac{h(U_i)}{n}$.

```
# define function
sin_cos_fn <- function(x) {
    return((cos(50 * x) + sin(20 * x))^2)
}

# get integrate() area
integrate(sin_cos_fn, 0, 1)
```

```
## 0.9652009 with absolute error < 1.9e-10
```

```
# plot function
ggplot(tibble(x = c(0, 0.2, 0.4, 0.6, 0.8, 1)), aes(x)) + stat_function(fun = sin_cos_fn)
```

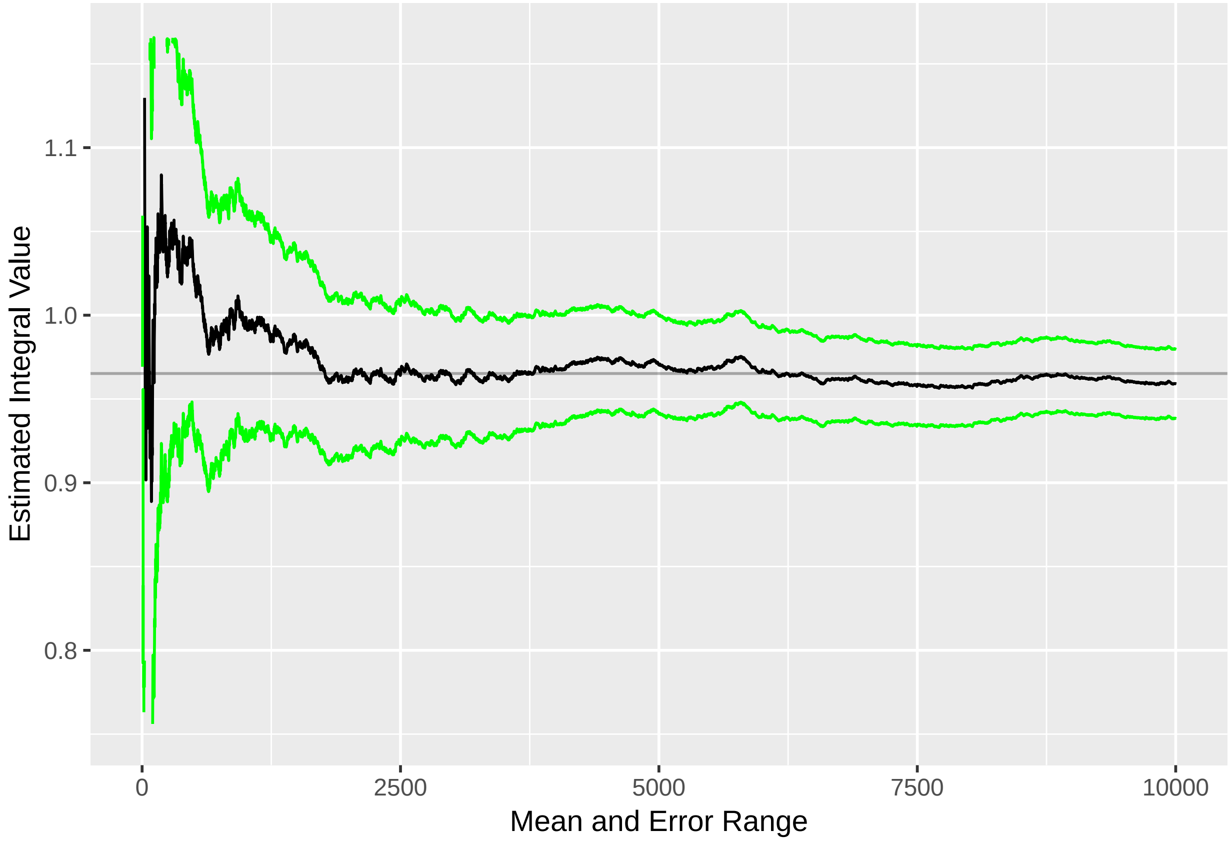


Now with Monte Carlo Integration

```r
# run 10^4 uniform samples through our function
unif_rvs <- sin_cos_fn(runif(10^4))

# get estimate of integral area
estimated_integral <- cumsum(unif_rvs)/(1:10^4)

# get estimate of error
estimation_error <- sqrt(cumsum((unif_rvs - estimated_integral)^2))/(1:10^4)

# add to tibble
estimated <- tibble(Integral = estimated_integral, Error = estimation_error) %>%
    mutate(Index = row_number()) %>% select("Index", "Integral", "Error")

# value of integral
max(estimated_integral[9750:10^4])
```

```
## [1] 0.9605904
```

```r
# plot
ggplot(estimated, aes(x = estimated$Index, y = estimated$Integral)) + geom_line() +
    xlab("Mean and Error Range") + ylab("Estimated Integral Value") + ylim(mean(unif_rvs) +
    20 * c(-estimated$Error[10^4], estimated$Error[10^4])) + geom_line(aes(y = estimated$Integral +
    2 * estimated$Error), color = "green") + geom_line(aes(y = estimated$Integral -
    2 * estimated$Error), color = "green") + geom_hline(yintercept = 0.9652009,
    alpha = 0.3)
```



While the bonus brought by the simultaneous evaluation of the error of the monte carlo estimate can not be disputed, we must be aware that it is only trustworthy as far as v_n is a proper estimate of the variance of $\bar{h}_n$. In critical situations where v_n does not converge at all or does not even converge fast enough for a CLT to apply, this estimate and the confidence region associated with it can not be trusted.

Example 3.4

Given a normal $\mathcal{N}(0,1)$ sample of size $n, (x_1, ..., x_n)$, the approximation of

$$\Phi(t) = \int_{-\infty}^{t} \frac{1}{\sqrt{2\pi}} e^{-\frac{y^2}{2}} dy$$

by the Monte Carlo method is

$$\hat{\Phi}(t) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}_{x_i \le t}$$

with exact variance $\frac{\Phi(t)[1-\Phi(t)]}{n}$, since the variables $\mathbb{I}_{x_i \le t}$ are independent Bernoulli random variables with success probability $\Phi(t)$.

3.3 | Importance Sampling

We will now look at importance sampling, which relies on *importance functions*, which are instrumental distributions in lieu of the original distributions. In fact, an evaluation of

$$\mathbb{E}_f[h(X)] = \int_{\mathcal{X}} h(x) f(x) dx$$

based on simulations from f is almost never optimal in the sense that using alternative distributions can improve the variance of the resulting estimator.

3.3.1 | An arbitrary change of reference measure

Given an arbitrary density g that is strictly positive when $h \times f$ is different from zero, we can rewrite

$$\mathbb{E}_f[h(X)] = \int_{\mathcal{X}} h(x) \frac{f(x)}{g(x)} g(x) dx = \mathbb{E}_g[\frac{h(X)f(X)}{g(X)}]$$

that is, as an expectation under the density g. This *importance sampling fundamental identity* justifies the use of the estimator

$$\frac{1}{n} \sum_{j=1}^{n} \frac{f(X_j)}{g(X_j)} h(X_j) \to \mathbb{E}_f[h(X)]$$

based on a sample $X_1, ..., X_n$ generated from g, not f. Since $\mathbb{E}_f[h(X)]$ can be written as the expectation under g, $\mathbb{E}_g[h(X)]$, our estimator does converge to $\mathbb{E}_f[h(X)]$ for the same reason the regular Monte Carlo estimator h_n converges whatever the choice of the distribution g (as long as $\sup g \supset \sup p(h \times f)$).

This property relates to the fact that $\mathbb{E}_f[h(X)]$ can be represented in an infinite number of ways by pairs (h, f) and thus a given integral is not intrinsically associated with a given distribution. There is almost absolute freedom in its representation as an expectation.

The constraint on the support of g, $\sup g \supset \sup p(h \times f)$ is absolute in that using a smaller support truncates the integral over g and, as a result, produces a biased result. This means that when considering nonparametric solutions for g, the support of the kernel must be unrestricted.

Example 3.5

Mentioned previously, approximating tail probabilities using Monte Carlo sums breaks down once we go far enough into the tails.

Suppose we are interested in the probability of a very rare event. Then naive simulation from f will require a huge number of simulations to get a stable answer. Thanks to importance sampling, we can greatly improve our accuracy and bring down the number of simulations by several orders of magnitude.

Consider a distribution with a restricted support on $(4.5, \infty)$. A natural choice is to take g as the density of the exponential distribution $\mathcal{E}\S_{\surd}(1)$ truncated at 4.5,

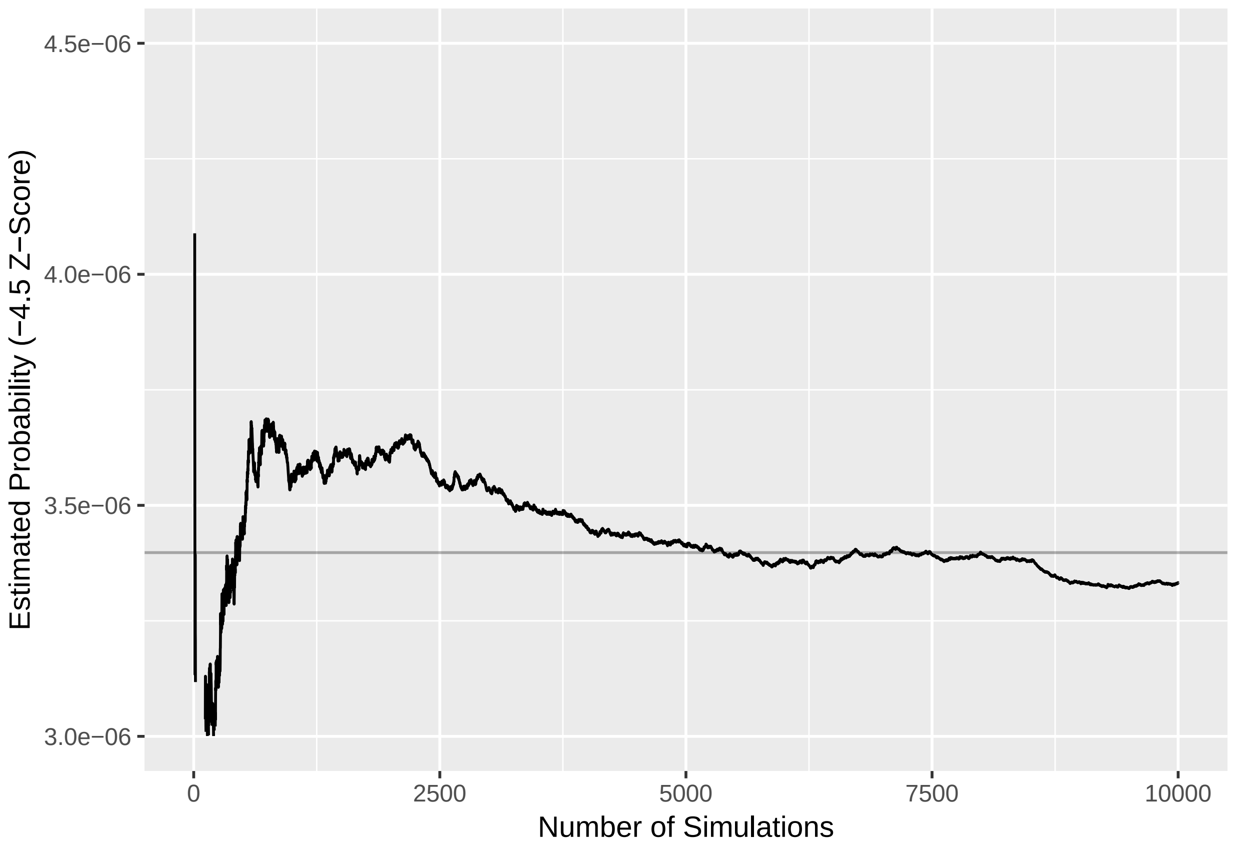
$$g(y) = \frac{e^{-y}}{\int_{4.5}^{\infty} e^{-x} dx} = e^{-(y-4.5)}$$

and the corresponding importance sampling estimator of the tail probability is

$$\frac{1}{n}\sum_{i=1}^{n} \frac{f(Y^{(i)})}{g(Y^{(i)})} = \frac{1}{n}\sum_{i=1}^{n} \frac{e^{-Y_i^2/2+Y_i-4.5}}{\sqrt{2\pi}}$$

where the Y_i's are iid generations from g.

```r
# parameters
numsims <- 10^4

# g
exp_dists <- rexp(numsims) + 4.5

# ratio
norm_exp_ratio <- dnorm(exp_dists)/dexp(exp_dists - 4.5)

# importance sampling estimator
imp_samp_est <- cumsum(norm_exp_ratio)/1:numsims

# place in tibble
norm_exp <- tibble(Importance_Sampling_Estimator = imp_samp_est, Normal_Exponential_Ratio = norm_exp_ra
    mutate(Index = row_number()) %>% select("Index", "Importance_Sampling_Estimator",
    "Normal_Exponential_Ratio")

# plot
ggplot(norm_exp, aes(x = norm_exp$Index, y = norm_exp$Importance_Sampling_Estimator)) +
    geom_line() + xlab("Number of Simulations") + ylab("Estimated Probability (-4.5 Z-Score)") +
    ylim(c(3e-06, 4.5e-06)) + geom_hline(yintercept = pnorm(-4.5), alpha = 0.3)
```



Example 3.6

When considering an observation $x \sim \mathcal{B}(\alpha, \beta)$

$$x \sim \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}\mathbb{I}_{[0,1]}(x)$$

there exists a family of conjugate priors on (α, β) of the form

$$\pi(\alpha, \beta) \propto \{\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\}^{\lambda} x_0^{\alpha} y_0^{\beta}$$

where λ, x_0, y_0 are hyperparameters. Then the posterior is equal to

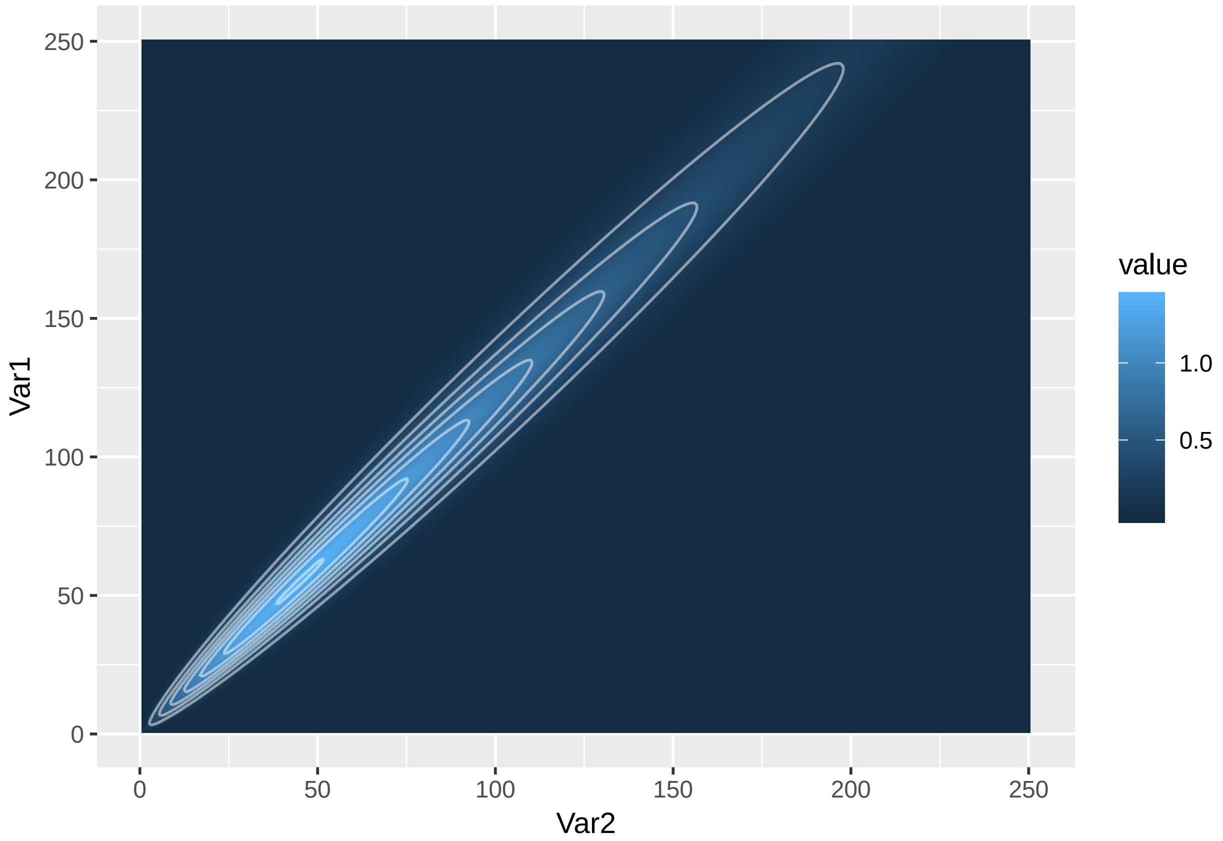
$$\pi(\alpha, \beta|x) \propto \{\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\}^{\lambda+1}[x x_0]^{\alpha}[(1-x)y_0]^{\beta}$$

This family of distributions is intractable if only because of the difficulty of dealing with gamma functions. Simulating directly from $\pi(\alpha, \beta|x)$ is impossible. We must use a substitute distribution $g(\alpha, \beta)$. We can get an idea of what may fit by getting a preliminary look at an image representation of $\pi(\alpha, \beta|x)$.

Let $\lambda = 1, x_0 = y_0 = 0.5, x = 0.6$

```
# define beta gamma conjugate function
beta_gamma <- function(a, b) {
    exp(2 * (lgamma(a + b) - lgamma(a) - lgamma(b)) + a * log(0.3) + b *
        log(0.2))
}

# parameters for grid
alpha_a <- 1:250
```

```r
beta_b <- 1:250

# outer creates a matrix of dimension dim(a), dim(b) whose A[i, j]
# element is f(a[i], b[j])
points <- outer(alpha_a, beta_b, beta_gamma)

# melt takes wide-format data and melts it into long form data
points <- reshape2::melt(points)

# plot
ggplot(points, aes(x = Var2, y = Var1)) + geom_raster(aes(fill = value)) +
    geom_contour(aes(z = value), colour = "white", alpha = 0.5)
```

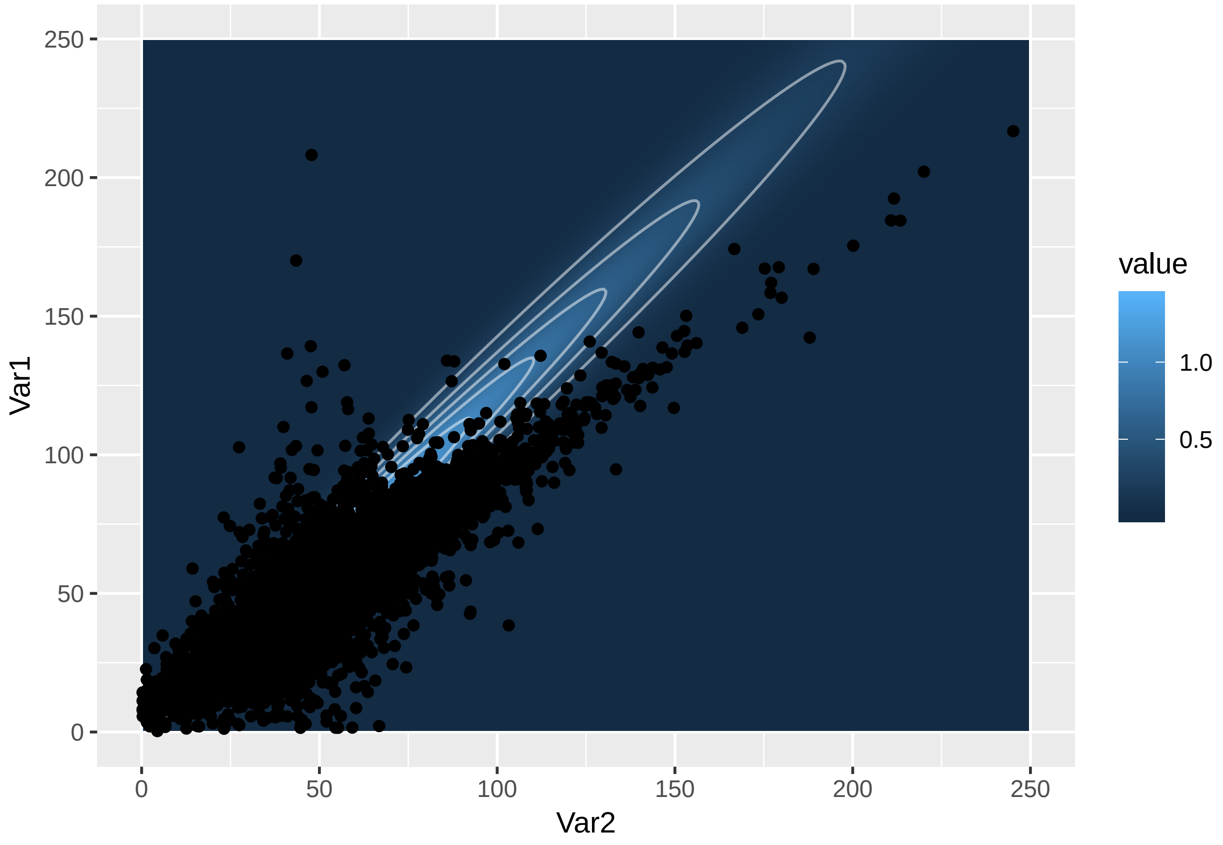


The examination of the figure above shows that a normal or Student's t distribution on the pair (α, β) could be appropriate.

Choosing a Student's $\mathcal{T}(3, \mu, \Sigma)$ distribution with $\mu = (50, 45)$ and $\Sigma = \begin{pmatrix} 220 & 190 \\ 190 & 180 \end{pmatrix}$ produces a reasonable fit. The covariance matrix was chosen by trial and error (looking closely at the plot).

```r
# create matrix of student t samples
student_t_samples <- matrix(rt(2 * 10^4, 3), ncol = 2)

# scale matrix
scale_sigma <- matrix(c(220, 190, 190, 210), ncol = 2)

# generate student t points
```

```
student_t_points <- t(t(chol(scale_sigma)) %*% t(student_t_samples) + c(50,
    50))
student_t_points <- student_t_points %>% as_tibble()

# plot
ggplot(points, aes(x = Var2, y = Var1)) + geom_raster(aes(fill = value)) +
    geom_contour(aes(z = value), colour = "white", alpha = 0.5) + geom_point(data = student_t_points,
    aes(x = V1, y = V2)) + xlim(c(0, 250)) + ylim(c(0, 250))
```


Var1
Var2
value