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.

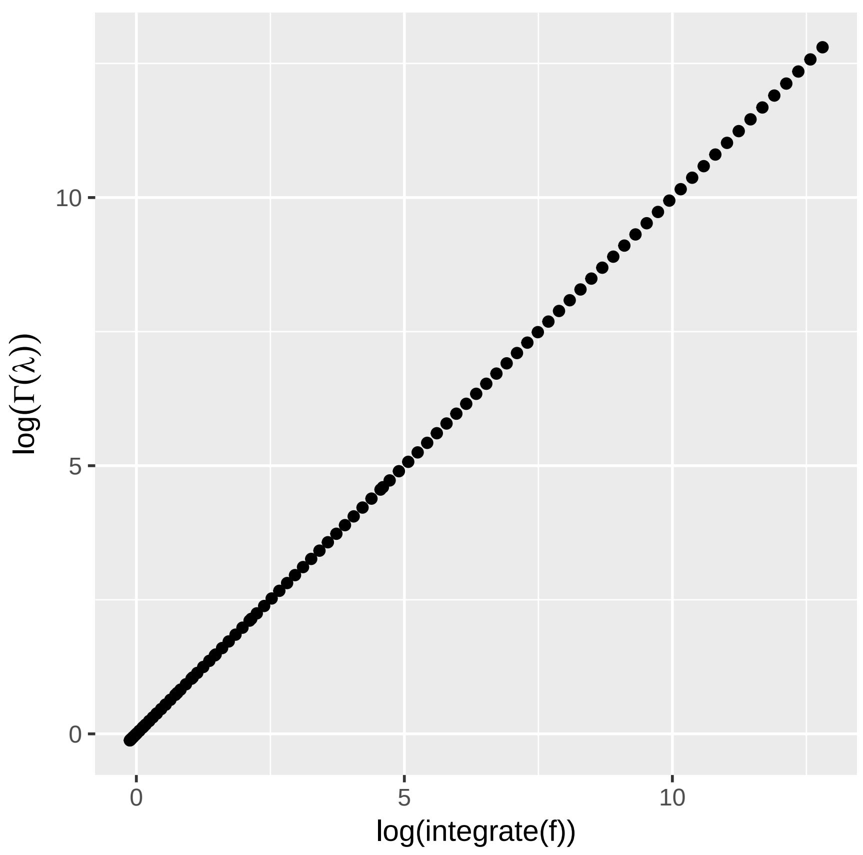
Example 3.1

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)
```

```r
}

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

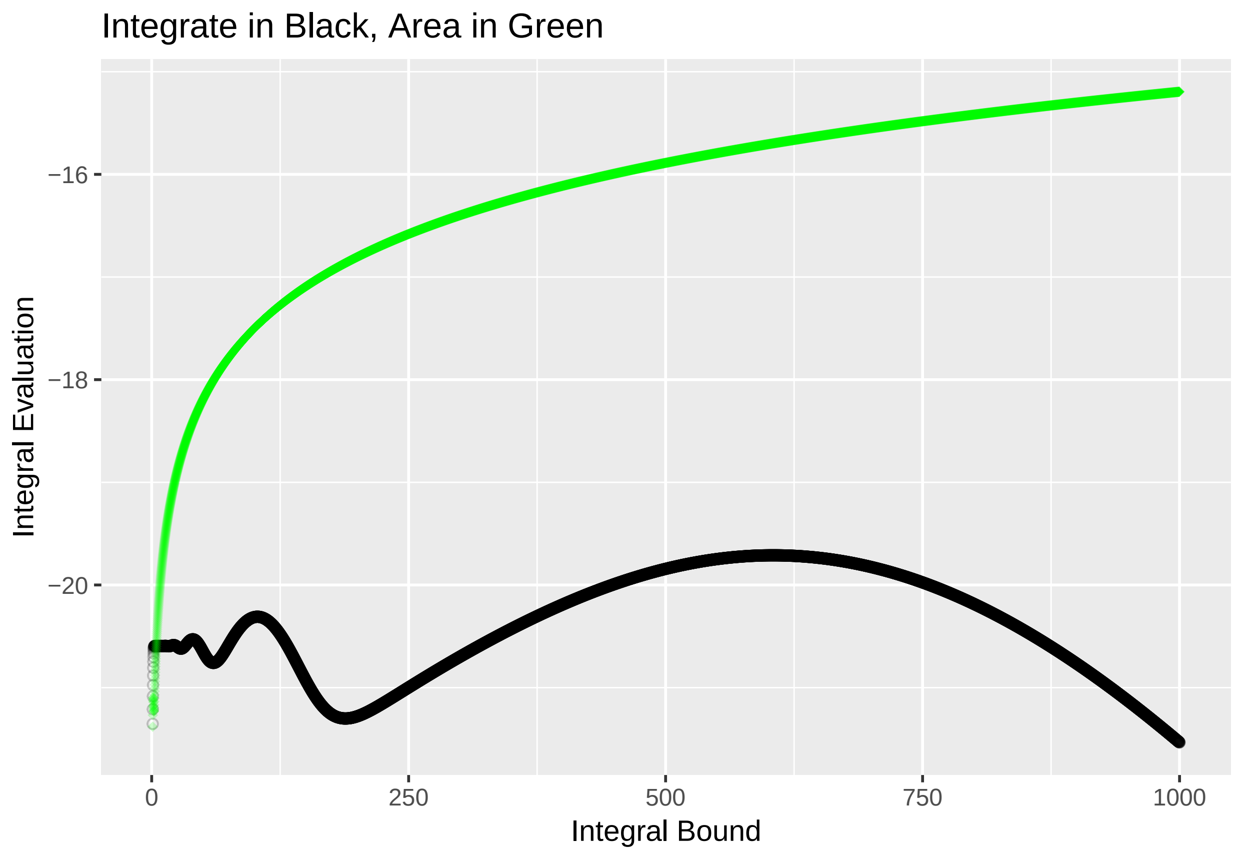
```
## 8.827559e-44 with absolute error < 1.7e-43
```

Furthermore, the result is not comparable to area:

```r
# 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] -21.53424 -15.19497
```

```r
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