Sampling Event Duration

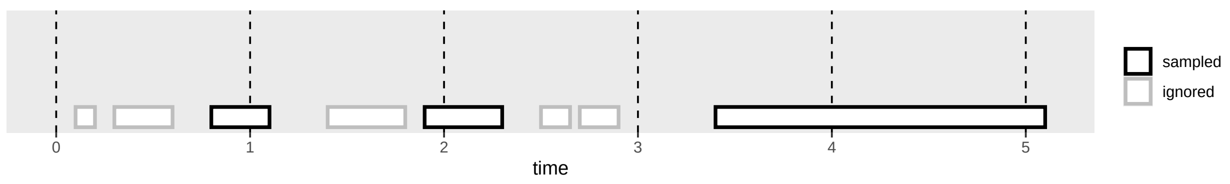
Alberto Dell'Era

A short summary is available at: https://github.com/alberto-dellera/sampling_event_duration/blob/main/README.md

Github readers: github on-line resolution is very low, and the pdf can be even truncated(!)
I'd suggest to download this pdf and open it locally

Abstract

We will study an event sampling process whose sampling period is uniform:



The process samples the duration of the events that are active ("in-flight") at sample time, and ignores all the others.

We will derive the probability distribution of the sampled durations given the original durations distribution in general, but focusing on the versatile Gamma distribution. We will build random generators to simulate the sampling process, and then design a Bayesian estimator to infer back the original distribution parameters, implemented in Stan.

Introduction

Consider performing a marketing survey to study the behaviour of customers of a huge physical library (or maybe an on-line retailer). To reduce the study cost, a plausible policy might be to visit the library every hour and interview every one inside, ignoring other customers that enter the library later on the same hour.

Imagine that one of the dimensions logged is "time spent inside the library". Since we will sample every one that spends more than one hour inside, but only half of people spending half an hour, one sixth of customers spending ten minutes, and so on, the sampled data will not be an accurate representation of the actual distribution of the inside-library duration. We would, for example, sadly overestimate the book-lovers proportion.

But if we know the theoretical distribution of the original data, we can in principle estimate its parameters from the samples, and obtain a cleanest picture.

A real-life example of such a sampling process can be found in IT. Specialists in software performance optimization make extensive use of *statistical profilers*[1]: tools that sample running programs execution at

[1]https://en.wikipedia.org/wiki/Profiling_(computer_programming)#Statistical_profilers

regular intervals and register what they were doing (e.g. waiting for disk, running on the CPU) for later analysis. These tools usually register only the occurrence on an event, not its duration; a notable exception is Oracle©® ASH[2], that samples the duration as well. Check (Beresniewicz 2020), especially slide 33, for an extensive discussion.

This paper is my personal journey into this fascinating inference problem, limited to simulated data for simplicity and lack of time, that might be a good starting point for someone interested in inferring from real data.

It's a Bayesian journey out of personal preference, but a Frequentist might easily adapt most of the paper for her toolbox; for example, an extension to MLE should be very simple, as hinted in the discussion.

Last but not least: everything is demonstrated using a reproducible R Markdown document, and I have strived to write the cleanest code possible and to carefully document it.

Probability Distribution of the samples

Letting $x \sim f(x)$ the event duration random variable and $y \sim g(y)$ the sampled one, we can easily derive the latter from the former by observing that the probability that x is sampled is proportional to x/T for $x < T$ (e.g. if x is half as long as T, its sampling probability is 0.5) and is 1 for $x >= T$. Hence we just need to multiply $f(x)$ by the hinge function

$$h(x) \triangleq \begin{cases} x/T & x < T \\ 1 & x >= T \end{cases}$$

and then renormalize, obtaining

$$g(y) = p(y = x) = \frac{h(x)f(x)}{\eta} \ ; \qquad \eta \triangleq \int_0^\infty h(\tau)f(\tau)d\tau$$

We can further decompose η as $\eta = \eta_A + \eta_B$, where

$$\eta_A \triangleq \int_0^T h(\tau)f(\tau)d\tau = \frac{1}{T}\int_0^T \tau f(\tau)d\tau$$

$$\eta_B \triangleq \int_T^\infty h(\tau)f(\tau)d\tau = \int_T^\infty f(\tau)d\tau = 1 - \int_0^T f(\tau)d\tau = 1 - F(T)$$

Note that η_A is essentially the average value of x over the interval $[0, T)$, and η_B is based on the CDF of the distribution; if $f(x)$ is a well-known distribution, both are usually readily available, at least numerically approximated.

Focus on the Gamma Distribution

Let's now focus on the case of the Gamma distribution, very important in the context of durations[3].

$$f(x) = Gamma(x|\alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}$$

where α is usually named the "shape" and β the "rate".

[2]this is actually the tool that sparked my interest for this statistical problem. I've been using it for years, when dressing the hat of Oracle Performance Tuning specialist, one of my preferred outfits.

[3]mostly because of its versatility: it includes the important exponential as a special case, and can approximate the normal itself well for high values of *shape*

For $x < T$ we have of course

$$g(y) = p(y = x) = \frac{x}{T} * Gamma(x|\alpha, \beta)\frac{1}{\eta}$$

But, algebraically:

$$x * Gamma(x|\alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{(\alpha+1)-1} e^{-\beta x} = \beta^{-1}\frac{\Gamma(\alpha+1)}{\Gamma(\alpha)} Gamma(x|\alpha + 1, \beta) \quad (1)$$

that is

$$x \sim Gamma(\alpha, \beta) \implies y \sim (const) * Gamma(\alpha + 1, \beta); \quad for\ x < T$$

hence the *effect of sampling is simply to increase the shape by 1*, normalization constants apart. Amazing[4] !

A practical benefit of (1) is that we can easily write η, the most difficult part of the density expression, as a function of Gamma (FGamma being the CDF):

$$\eta_A = \frac{1}{T}\beta^{-1}\frac{\Gamma(\alpha+1)}{\Gamma(\alpha)} FGamma(T|\alpha + 1, \beta); \quad \eta_B = 1 - FGamma(T|\alpha, \beta)$$

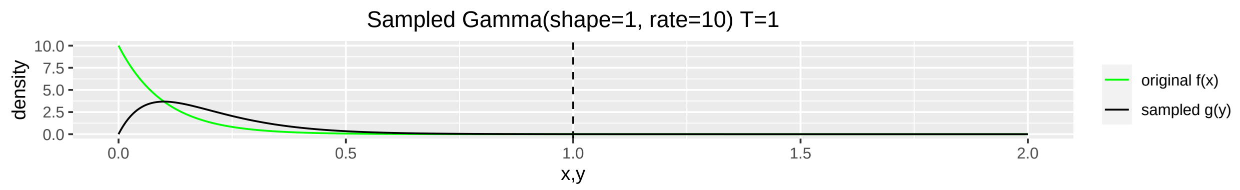
In R, that becomes:

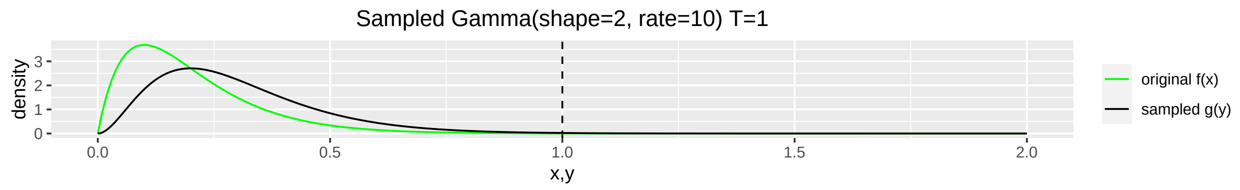
```r
#' Density of the Sampled Gamma Distribution
#'
#' @param x     Vector of random variable values
#' @param shape The original Gamma "shape" parameter
#' @param rate  The original Gamma "rate" parameter
#' @param T     The sampling period
#' @param log   If TRUE, returns the log density
#'
#' @return Vector of (log) densities
dsgamma <- function(x, shape, rate, T, log=FALSE) {
  T.log <- base::log(T)

  # calc eta
  A.log <- (-T.log) +
           (-base::log(rate)) +
           lgamma(shape+1) - lgamma(shape) + pgamma(T, shape=shape+1, rate=rate, log.p=TRUE)
  A <- exp(A.log)
  B <- 1.0 - pgamma(T, shape=shape, rate=rate)
  eta.log <- log(A + B)

  # calc density
  x  <- ifelse(x > .Machine$double.eps, x, NA)
  f.log <- dgamma(x, shape=shape, rate=rate, log=TRUE)
  h.log <- ifelse(x < T, base::log(x) - T.log, 0.0)
  g.log <- ifelse(is.na(x), -Inf, f.log + h.log - eta.log)
  if (log) {
    return (g.log)
  }
  exp(g.log)
}
```

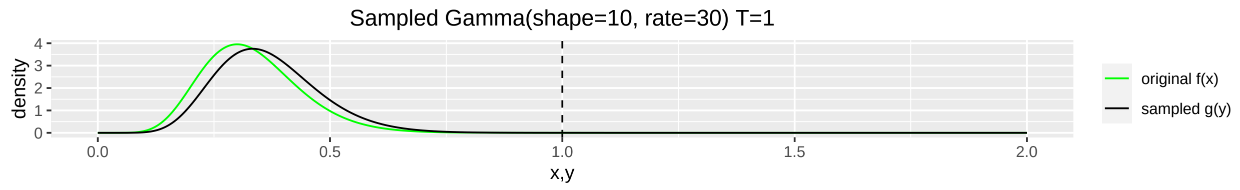
[4] if developed further, this result could probably give some insights about the physical meaning of "shape".

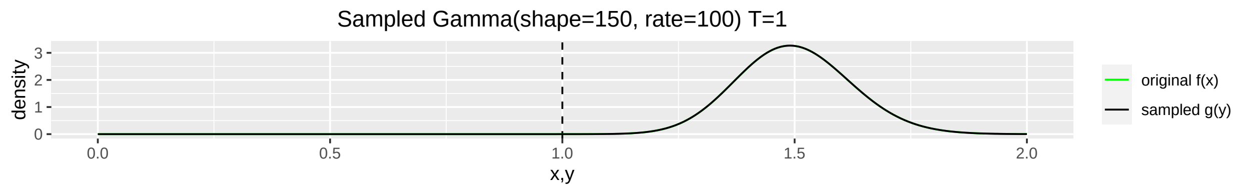
discussion

Let's plot the densities for a few interesting cases:









The first plot is the case of the ubiquitous exponential distribution. The sampling distortion is quite dramatic: the exponential silhouette is gone; small durations, that used to concentrate most of the mass, have almost disappeared, and the mode itself has shifted from zero to about 0.1.

In the second case, the distortion is still quite noticeable; only in the third case, when the shape has got to 10, the distortion is relatively minimal.

The fourth plot shows the case of a distribution whose mass is well above the sampling period. As expected, the sampled distribution is virtually the same as the original.

Random Generators and Simulation

We now discuss two numerical methods for drawing random numbers from the sampled distribution $g(x)$ given a drawing function for $f(x)$, a crucial task for any serious application.

by physical process simulation

The most straightforward way is to directly implement the physical sampling process:

```
#' Draw samples y distributed as g(y) from the original distribution f(x),
#'   by simulating the physical process
#'
```

```r
#' @param n      The number of draws desired from g(y) (i.e. length of y)
#' @param f      A function that draws n numbers from the original durations distribution
#' @param T      The sampling period
#' @param b      A function that draws n numbers from the distribution of in-between time
#' @param fill.x Whether to return the original x draws
#'
#' @return a list of two vectors: "x" with all the draws, and "y" for the sampled subset
rsampled.phy <- function(n, f, T, b=NULL, fill.x=FALSE) {
  y <- rep(-1, n)
  x <- numeric(0)
  xi <- 0; yi <- 0
  sta <- 0; sto <- NA

  repeat {
    # draw x from f(x)
    dur <- f(1)
    if (dur < 0) stop(paste0("negative duration from f: ", dur))
    sto <- sta + dur

    # store x if so desired
    if (fill.x) {
      xi <- xi + 1
      if (xi > length(x)) {
        x <- append(x, rep(NA, 10*n))
      }
      x[xi] <- dur
    }

    # sample x as y if across sampling instant
    if (floor(sta/T) < floor(sto/T)) {
      yi <- yi + 1
      y[yi] <- dur
      if (yi == n) break
    }

    # draw from the in-between distribution
    if (!is.null(b)) {
      dur_break <- b(1)
      if (dur_break < 0) stop(paste0("negative duration from b: ", dur_break))
      sto <- sto + dur_break
    }

    sta <- sto
  }
  list(x=x[1:xi], y=y)
}
```

This trivial function simply draws a duration from the provided $f(x)$, sets accordingly the start and stop extremata of the event, and if the interval crosses the sampling instant it returns the duration as y. It can optionally store all the original draws from x (which the y vector is a subset of).

Side note: to accurately simulate the physical process, an optional $b(x)$ distribution can be provided to model the time in-between events. This distribution does not alter[5] the density of the returned y random variate,

[5]this is because the *stopping intention* of this function is to stop when n samples have been provided. If the stopping

but it could change the average size of x (for example, if $f(x) \equiv b(x)$, the expected size of x would be twice as long). $b(x)$ will never be used in this paper, but can be useful for other purposes.

A call example is the specialization for the Gamma distribution that we will use from now on:

```
rsgamma.phy <- function(n, shape, rate, T, b=NULL, fill.x=FALSE) {
  f <- function(n) {
    rgamma(n, shape=shape, rate=rate)
  }
  rsampled.phy(n, f, T, b, fill.x)
}

set.seed(5)
rsgamma.phy(2, shape=1, rate=1, T=1.0, fill.x=TRUE)
```

```
## $x
## [1] 0.08218487 0.17824958 0.55082843 2.44242891 0.16455419 0.61080940
##
## $y
## [1] 2.4424289 0.6108094
```

by rejection sampling

Another possibility is to draw a duration, and then accept it as a sample with probability $h(x)$:

```
#' Draw samples y distributed as g(y) from the original distribution f(x),
#'   by acceptance/rejection sampling accordingly to h(x)
#'
#' @param n      The number of draws desired from g(y) (i.e. length of y)
#' @param f      A function that draws n numbers from the original durations distribution
#' @param T      The sampling period
#'
#' @return a vector containing the sampled y
rsampled.rej <- function(n, f, T) {
  y <- rep(-1, n)
  yi <- 0
  repeat {
    dur <- f(1)
    if (dur < 0) stop(paste0("negative duration from f: ", dur))
    h = ifelse(dur < T, dur/T, 1)
    p <- runif(1)
    if (h > p) {
      # accept
      yi <- yi + 1
      y[yi] <- dur
      if (yi == n) break
    } else {
      #reject
```

intention were, say, to stop after a certain time has elapsed, the number of samples would be a random variable, not a given number, influenced by $b(x)$ as well as $f(x)$. This is actually the most common real-life scenario but it is considerably more complex to handle and hence out-of-scope for this paper, that focuses on the fundamental theory; for the intricacies of this sampling intention see (Kruschke 2015, 308, "With intention to fix duration"), in the context of p-values. Anecdotally, the stopping intention should not change the results that much for practical applications.

```
    }
  }
  y
}
```

This is a great example of the classic technique of acceptance-rejection sampling. Actually, it is a marvelous example of an acceptance-rejection sampling that *happens physically*, not as a numeric artifact to efficiently draw random numbers from distributions.

The drawback of rsampled.rej (and rsampled.phy actually) is that it is slow for distributions whose mass is concentrated near very small values; for example, for durations around 1e-3 it takes 1000 draws for each y on average. Speed is anyway adequate for our purposes[6].

Call example, and specialization for the Gamma distribution:

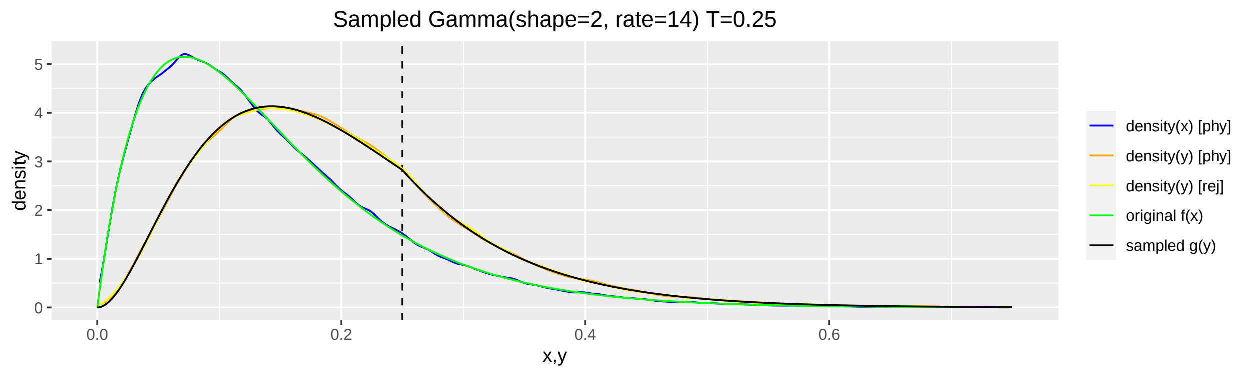
```
rsgamma.rej <- function(n, shape, rate, T) {
  f <- function(n) {
    rgamma(n, shape=shape, rate=rate)
  }
  rsampled.rej(n, f, T)
}

set.seed(5)
rsgamma.rej(2, shape=1, rate=1, T=1.0)
```

```
## [1] 0.6847818 0.5611688
```

Consistency checks

Let's now check[7] that both our random generators are consistent with the theory we developed, to cross-check all of them.

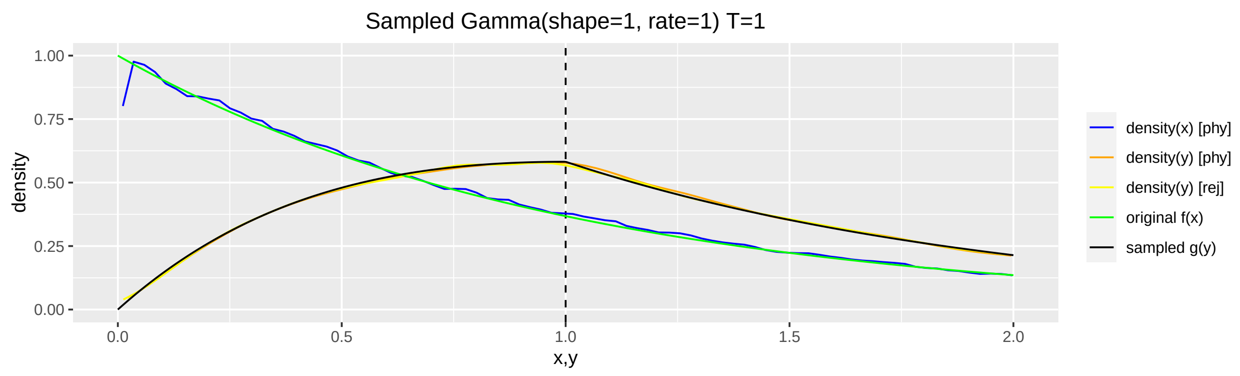
First, consider a generic Gamma:



The two generators empirical densities for y lay perfectly over the sampled theoretical density (shown in black), and the same goes for the original x duration draws (the original theoretical density is shown in green).

Next, consider the Exponential case:

[6] I was experimenting with faster alternatives when time ran out; I might pursue that road again in the future.
[7] the checking code is not shown but of course available in the original Rmarkdown from Github (see link at the bottom).



Note the same perfect juxtaposition around T, the critical value around which the density changes[8] its silhouette.

Reparameterization (and a look at the Likelihood)

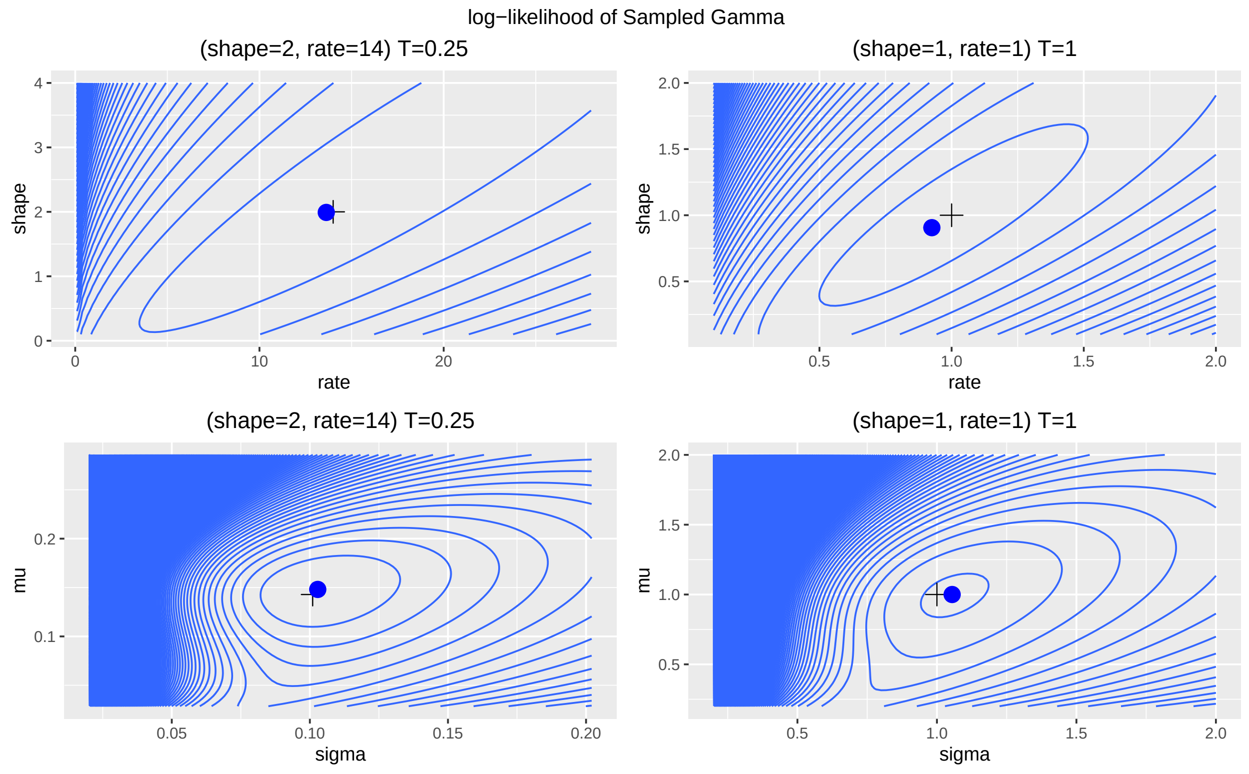
The following well-know formulae allows us to reparameterize the Gamma in term of the expected mean and standard deviation:

$$x \sim Gamma(\alpha, \beta) \implies \begin{cases} \mu \triangleq E[x] = \alpha/\beta \\ \sigma \triangleq sd[x] = \sqrt{\alpha/\beta^2} \end{cases} \implies \begin{cases} \alpha = \mu^2/\sigma^2 \\ \beta = \mu/\sigma^2 \end{cases}$$

The main motivation for this change of parameters is for Bayesian analysis, since prior knowledge is usually expressed in term of μ and σ rather than α and β; for example, we may probably easily know that the a-priori wait time for a storage disk is 5msec $\pm$ 1 msec from previous measurements or from the disk technical specification, but it is less likely (in general) that we know plausible intervals for α and β.

Let's now take a look at the resulting likelihoods shapes, by taking some draws from a Sampled Gamma and then calculating the log-likelihood over a grid around the parameter values. Let's consider again the two densities we used in the "consistency check" section:

[8] also note that the density is still continuous at T, but the first derivative is not



We can see that the maximum value (the big blue dot) is very close to the true value (the black cross). Also, the curve looks continuous (probably concave in general), and that bodes well for using an algorithm like gradient-descent to build an MLE estimator.

Bayesian Inference by Stan

The sampling process, as we have seen, might sample only a small portion of the original durations (e.g. one out of 1,000 for durations around 1msec if T=1sec), and that might leave us with very few data to build our inference on. Mathematical models are preferable in general with small datasets, and Bayesian models in particular, since they can incorporate prior (or "domain") knowledge to further improve the inference.

We will explore the simple case of uninformative priors, both for simplicity and because the results should be nearly the same we might obtain using MLE or similar Frequentist techniques. Adding other priors in the Stan model is definitely trivial anyway.

We will parameterize the model by μ and σ mostly because, as already discussed, prior knowledge is probably more readily expressed on these parameters rather then on *shape* and *rate*.

Side note: another reason is that the chains seem to converge quicker using μ and σ, but I have not made an exhaustive comparison, so take my word with a grain of salt; yet, the likelihood plots previously shown seem to indicate a smaller correlation between μ and σ than between *shape* and *rate*.

I have of course checked the chains convergence using the usual techniques (visual inspection of the chains, Gelman-Rubin statistics, etc). I will not plot them for the sake of brevity; interested readers can inspect them by activating the plots in the original Rmarkdown source[9].

Here's the Stan model:

[9]just knit with stan.inference.print.conv set to TRUE

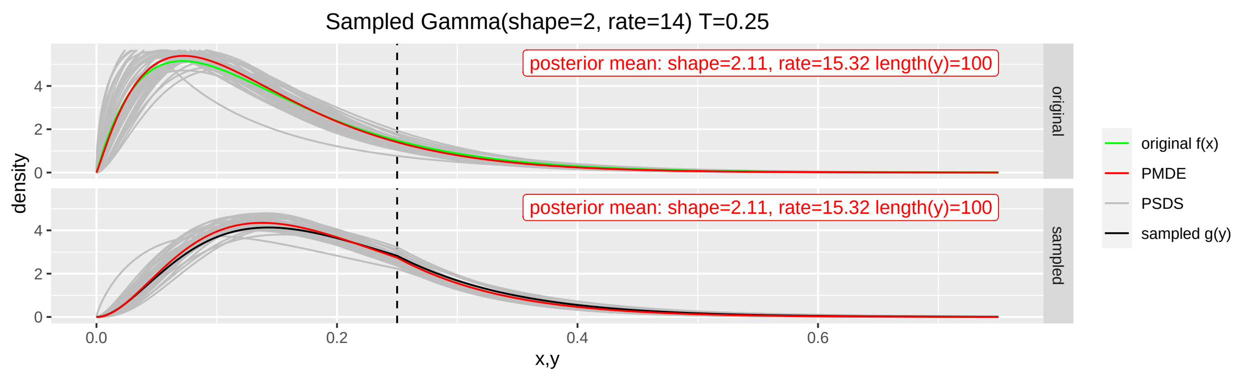
```
stan_model_code <- "
functions {
  real sampled_gamma_lpdf(data real[] x, real shape, real rate, data real T) {
    real T_log = log(T) ;

    // calc eta
    real A_log = (-T_log) +
                 (-log(rate)) +
                 lgamma(shape+1.0) - lgamma(shape) + gamma_lcdf ( T | (shape+1.0), rate );
    real A = exp(A_log) ;
    real B = 1.0 - gamma_cdf ( T, shape, rate );
    real eta_log = log(A + B);

    // calc density
    real acc = 0.0;
    for (i in 1:num_elements(x)) {
      real xx = x[i];
      real f_log = gamma_lpdf(xx | shape, rate);
      real h_log = xx < T ? log(xx) - T_log : 0.0;
      real d_log = f_log + h_log - eta_log;
      acc += d_log;
    }
    return acc;
  }

  real sampled_gamma_mu_sigma_lpdf(data real[] x, real mu, real sigma, data real T) {
    real shape = mu^2 / sigma^2;
    real rate  = mu   / sigma^2;
    return sampled_gamma_lpdf(x | shape, rate, T);
  }
}
data {
  int<lower=0> N;
  real y[N];
  real T;
}
parameters {
  real<lower=0> mu;
  real<lower=0> sigma;
}
model {
  mu    ~ uniform(0.0000001, 100000); // uninformative prior
  sigma ~ uniform(0.0000001, 100000); // uninformative prior
  y ~ sampled_gamma_mu_sigma_lpdf(mu, sigma, T);
}
generated quantities {
  real shape = mu^2 / sigma^2;
  real rate  = mu   / sigma^2;
}

"
```

Note that the core of the model is a simple translation (almost a copy-and-paste) of dsgamma from R to the

Stan language[10]. Also note the uninformative priors, and that in the posterior MCMC samples we derive *shape* and *rate* as well for convenience.

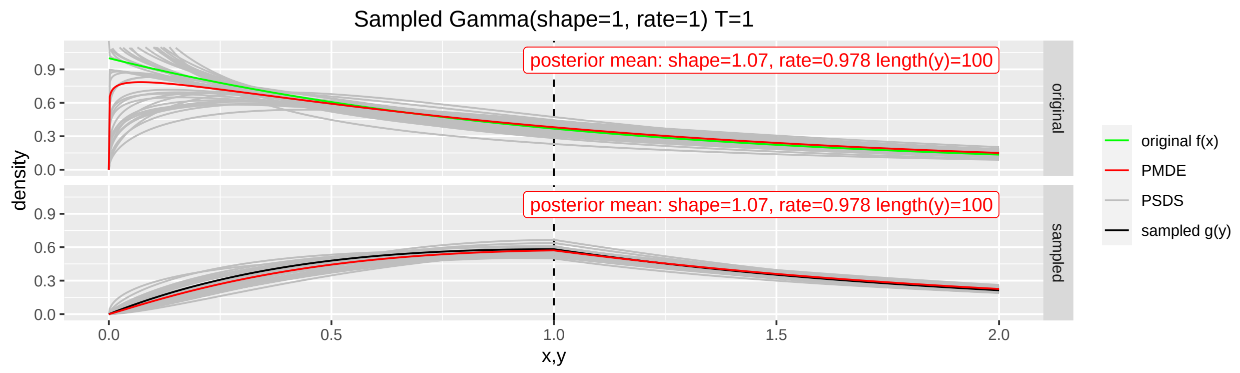
Let's now investigate the inference, by generating 100 duration samples, feeding them to Stan to get back the posterior MCMC chains, and by plotting both PMDE (Posterior Mean Density Estimate), the density curve whose parameters are the mean of the posterior chains, and the PSDS (Posterior Samples Density Set), a smattering of densities whose parameters are chosen at random from the posterior chains.

Here are the final inferences for our usual pair of densities. First, consider the generic Gamma:



We can see that the PMDE closely fits the true density curve, and that the PSDS are packed around it, albeit not very tightly, as expected given the low number of samples.

Next, consider the exponential:



Both the PMDE and the PSDS seem to be qualitatively near to the true density for $x >> 0$, and far from it for x near zero. But, this is actually an intrinsic issue for the exponential distribution when considered as a particular case of the Gamma: you can never get accurate estimates for x near to zero. This is because

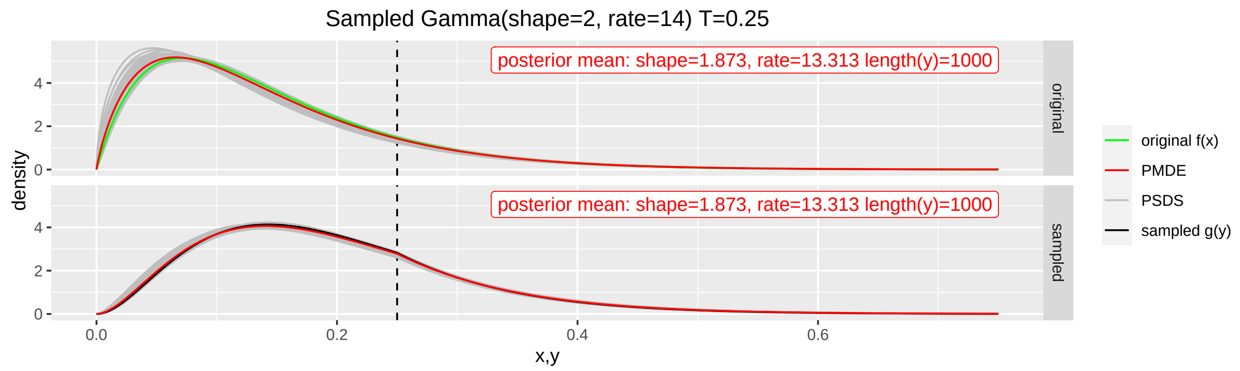
$$\lim_{x\to 0} Gamma(x|\alpha,\beta) = \begin{cases} +\infty & \alpha < 1 \\ \beta & \alpha = 1 \\ 0 & \alpha > 1 \end{cases}$$

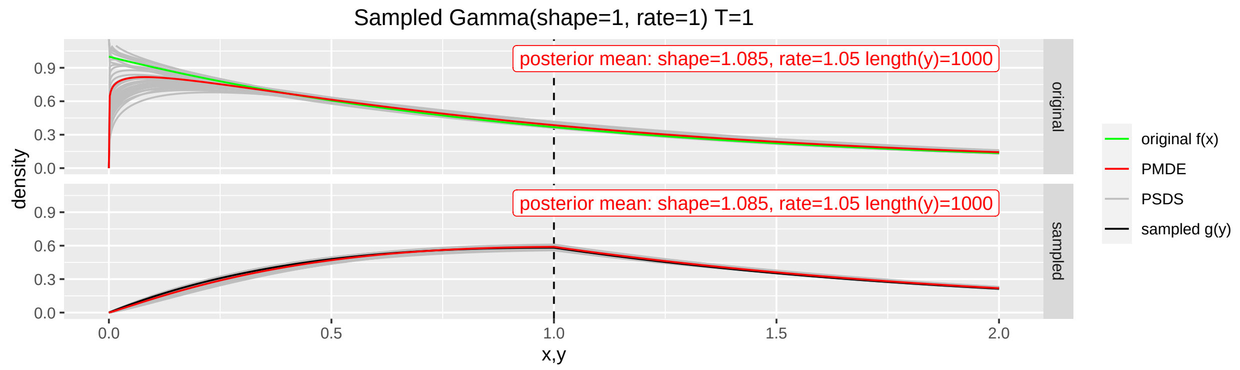
Hence any error of the *shape* estimate will make you drift away from *beta*, towards zero or $+\infty$ depending on the error sign, whatever small the error modulus is, as patently obvious from the PSDS plot above.

[10]it is also re-parameterized using a simple wrapper function, and by skipping the wrapper one can get back the parameterization in term of *shape* and *rate*.

In other words: the behaviour for x near zero is structural[11], nothing is wrong with the Bayesian estimate.

Of course the estimates neatly improve if more data is available. Here are the results with a dataset of 1000 duration samples:





Conclusion, and next steps

We have explored a Bayesian way to infer the parameters of a duration distribution given some samples over an uniform interval, focusing on the versatile Gamma distribution. We have simulated the data and the sampling process using R and the Stan MCMC tool, exploring the fitness of the estimates to the true data using conventional Bayesian techniques.

The next obvious step is to experiment with real data. The most challenging step is to choose a good distribution for the physical process that we want to study; for software program statistical profilers (including ASH) we should probably use a distribution arising from Queueing Theory, since “everything is a Queue” inside a computer, after all.

Using Bayesian Statistics would also allows us to factor in expert knowledge, that would dictate which Prior to use (I have used only uninformative priors in the paper, for simplicity). For example, an expert might tell us that the average disk response time is between 0.5ms and 3msec, and that would certainly help both the precision of the estimates and the MCMC convergence.

And the “expert” might be an automated tool that mined historical data, producing a general, broad Prior to be used as input.

[11] if there are strong (theoretical?) reasons to believe that the true density is actually exponential, the best approach is probably to use an exponential model instead of a Gamma one. One could simply plug the exponential as $f(x)$ in the first formulae we derived, or could possibly use our Stan model by forcing $shape = 1$.

Document version: 2021/01/09

References

Gihub repository: (https://github.com/alberto-dellera/sampling_event_duration), with the Rmarkdown source

Beresniewicz, John. 2020. *DB Time, Average Active Sessions, and Ash Math - Oracle Performance Fundamentals*. RMOUG. https://t.co/wjz3blBIkv?amp=1.

Kruschke, John K. 2015. *Doing Bayesian Data Analysis: A Tutorial with R, Jags, and Stan.* 2nd ed. Academic Press / Elsevier. https://sites.google.com/site/doingbayesiandataanalysis.