Package sads
Modeling Species Abundance Distributions
LOGBOOK

Paulo Inácio Prado and Cristiano Strieder

19 de abril de 2011

Warning

Though this document will hopefully become a comprehensive vignette, currently it is a draft logbook used to clarify some aspects of the theoretical background of this package to developers, and to guide the next steps.

Poisson Sample

Given that there is n individuals of a given species in a community and that a fraction a of these individuals were randomly sampled, the probability of getting y individuals in the sample is given by the Poisson distribution

$$f(y) = \frac{(a\,n)^y\, e^{-a\,n}}{y!} \tag{1}$$

Where the the single parameter of the Poisson λ correspond to the expected value $E[y]$ and thus is substituted by the product $a\,n$. Now suppose that the abundance n in the community is also a random variable that can be described by some probability distribution $g(n)$. Then the unconditional probability of getting y individuals in the sample is given by the integral

$$h(y) = \int^{\Omega} f(y)g(n)\,dn \tag{2}$$

Which is the sad model we are looking for. Although there are analytic solutions for many cases, our goal is to write a R function that solves numerically equation **??** in a generic way, that is, a single function whith an argument for $g(n)$. This function would then allow to compound any probability density function with the Poisson distribution. Taking this generalization further, we can substitute the Poisson distribution by another sampling model, e.g. the negative binomial to model aggregated sampling.

Our first step is to construct a function for a particular case for which the analytical solution is known, which allow us to check the results of the numerical integration.

Compounding with the exponential

Taking for $g(n)$ the exponential

$$g(n) = \lambda e^{-n\lambda} \tag{3}$$

The equation **??** solves to:

$$h(y) = \frac{\lambda a^y}{(a+\lambda)^{y+1}} \tag{4}$$

Using logarithms in the intermediary calculations to avoid overflow, we can define this function in R as

```
> dpoix <- function(x, frac, rate, log=FALSE) {
+           b <- x*log(frac)
+           m <- log(rate)
+           n <- (x+1)*log(rate+frac)
+           if(log)b+m-n else exp(b+m-n)
+         }
```

The equivalent function for numerical integration in R is[1]:

```
> dsad <- Vectorize(FUN=
+                   function(y,a=0.05,lambda=1/1000){
+                     poi <- function(y,n){
+                       w <- y*log(a*n)-lfactorial(y)-a*n
+                       exp(w)
+                     }
+                     f1 <- function(n){
+                       dexp(n,rate=lambda) * poi(y,n)
+                     }
+                     integrate(f1,0,Inf, abs.tol = .Machine$double.eps^0.25/1000000)$value
+                   },
+                   "y")
> integrate(dsad,0,Inf)
```

The functions seems to give the same results

```
> dsad <- Vectorize(FUN=
+                   function(y,a=0.05,lambda=1/1000){
+                     poi <- function(y,n){
+                       w <- y*log(a*n)-lfactorial(y)-a*n
+                       exp(w)
+                     }
+                     f1 <- function(n){
```

[1] This function must be passed to `Vectorize` to make `integrate` work with more than one value of `y`

```
+                       dexp(n,rate=lambda) * poi(y,n)
+                     }
+                     integrate(f1,0,Inf, abs.tol = .Machine$double.eps^0.25/1000000)$value
+                   },
+                   "y")
> integrate(dsad,0,Inf)

0.690484 with absolute error < 0.00011

> dpoix <- function(x, frac, rate, log=FALSE) {
+           b <- x*log(frac)
+           m <- log(rate)
+           n <- (x+1)*log(rate+frac)
+           if(log)b+m-n else exp(b+m-n)
+         }
> dsad(0:10,a=0.1,lambda=0.01)

 [1] 0.09090909 0.08264463 0.07513148 0.06830135 0.06209213 0.05644739
 [7] 0.05131581 0.04665074 0.04240976 0.03855433 0.03504939

> dpoix(0:10,frac=0.1,rate=0.01)

 [1] 0.09090909 0.08264463 0.07513148 0.06830135 0.06209213 0.05644739
 [7] 0.05131581 0.04665074 0.04240976 0.03855433 0.03504939
```

But as the value of y increases the numeric solution goes to zero. This effect is circumvented by the argument `abs.tol`, but it is not yet solved for small values of `frac` and `rate`, e.g.

```
> dsad(c(200,20000),0.1,0.00001)

[1] 6.319212e-64 0.000000e+00

> dpoix(c(200,20000),0.1,0.00001)

[1] 9.801016e-05 1.353353e-05
```

The figure **??** shows that this effect occurs even for moderate values of λ. The next step is to investigate this, and to correct the numerical integration.

Numerical integration with the function `integrate`

The first step on investigating this integration is verify the details of the `integrate` function. From the R-help of the function, we find that it implements the adaptative quadrature method. Additionally to the expression being integrated and the limits of integration, `integrate` also allow to specify the absolute(`abs.tol`) and relative (`rel.tol`) accuracy of the function used to approximate the integrand.[2]

[2] http://www.amtp.cam.ac.uk/lab/people/sd/lectures/nummeth98/integration.htm

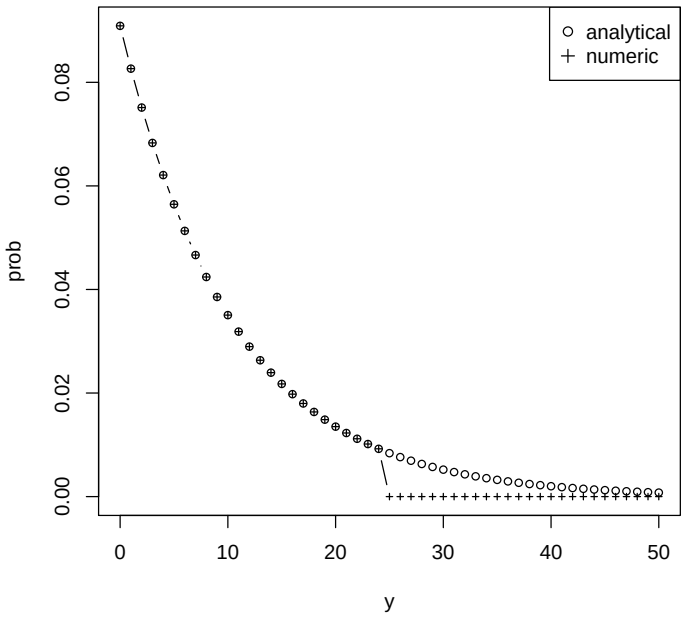


Figura 1: Probability value returned by the function `dpoix`, which calculates the Poisson-exponential analytically and the function `dsad`, which calculate the compound distribution by numerical integration.

As was said above, the numeric solution goes to zero as the value of `y` increases. And this effect is circumvented specifying a value for `abs.tol`. Although this approach conduct to a numerical result equivalent to the one otained analytically, it is easily found that for ranges(of the parameter `y` and the values of `frac` and `rate`) out of the considerated, then the explicit definition of `abs.tol` does not help at all. Alternativaly, a similar effect is observated when modifying the upper limit of integration and it is interesting to note that the effects are observated even for wide ranges of the parameters. May be the change on `abs.tol` induct a modification in the numerial estimation of `Inf` [3].

A first guess on replace `Inf`

We now propose another way to circumvent the problem, it consist in manipulate the upper limit of integration `Inf`. Such procedure is not recommended according to R-help but for our purposes it seems to produce good results.

[3] http://en.scientificcommons.org/43526693

With a few trials we found that substitutig `Inf` by particular values, this can make the numerical solution matches the anlytical. It is observated that increasing the upper limit, when `frac` and `rate` are getting lower, conduct to better aproximation of the numerical integration. Considering `frac` $= 10^{-1}$ and `rate` $= 10^{-5}$, with $200 \leq$ `y` ≥ 20000, a reasonably good choice is replace `Inf` by `k*y/(a+lambda)` with `k=1.9920941`. Figure **??** shows the analytical and numerical curves visually matching. Figure **??** is the result when integral is calculated with `abs.tol`.

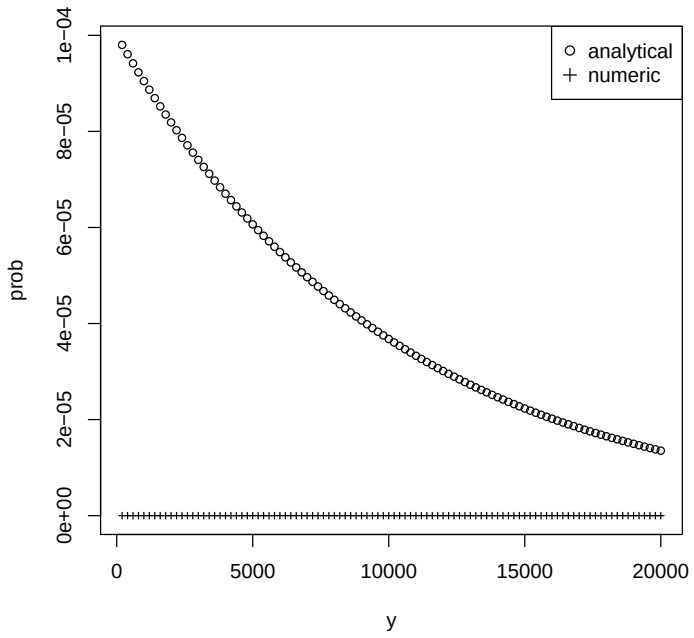


Figura 2: Probability value returned by the function `dpoix` and the function `dsad`. Considering `frac` $= 10^{-1}$ and `rate` $= 10^{-5}$, with $200 \leq$ `y` ≥ 20000
.

y1 <- dpoix(x,frac,rate)

In the above case the value of `k` was manually tunned, hopefully giving exact results for:

```
> dsad(c(200,20000),0.1,0.00001)

[1] 6.319212e-64 0.000000e+00

> dpoix(c(200,20000),0.1,0.00001)
```

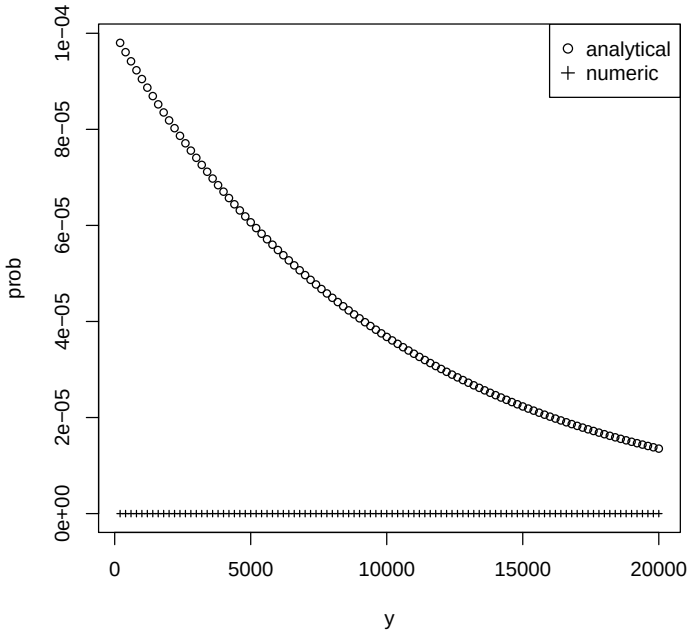


Figura 3: Probability value returned by the function `dpoix` and the function `dsad`. Considering `frac` $= 10^{-1}$ and `rate` $= 10^{-5}$, with $200 \leq$ `y` ≥ 20000

```
[1] 9.801016e-05 1.353353e-05
```

For ranges out of the cited above, then the value of `k`, or even the entire solution, may be defined more consistently. This result can be improved, for wide ranges of `frac`, `rate` and `y`, if we specify the upper limit of integration in an another alternative manner. This time the idea is to split the total range of any parameter, `y` for example, in two or more parts and then specify the numerical integration for each part, as is shown below.

```
> dsad <- Vectorize(FUN=
+                   function(y,a,lambda){
+                     poi <- function(y,n){
+                       w <- y*log(a*n)-lfactorial(y)-a*n
+                       exp(w)
+                     }
+                     f1 <- function(n){
+                       dexp(n,rate=lambda) * poi(y,n)
+                     }
```

```
+                                       if (y<10){
+                                               integrate(f1,0,100/max(a,lambda))$value
+                                       }else if (y<50){
+                                               k <- 6
+                                               integrate(f1,0,k*y/(a+lambda))$value
+                                       }else{
+                                               k <- 1.9920941
+                                               integrate(f1,0,k*y/(a+lambda))$value
+                                       }
+                   },
+                   "y")
```

The above expression is an attempt to correct the result from the numerical integration provided by the standart `integrate` R function. The limits for each `y` interval were selected to exemplify.

In fact, this is not a final solution yet, but results are improved as shows Figure **??**. Unfortunately both curves do not match for all ranges. Figure **??** show an example when it does not work, for `rate` $= 10^{-6}$ and `frac` $= 10^{-1}$. Another methods/functions for numerical integration must be tested.

A general criteria to set the upper integration limit

An abundance value that is larger than the abundances of all or most of the species in the community is a natural upper limit for the integration. This value can be set with the quantile functions of the probability distibutions. For example, for a community sad that follow a gamma distribution with parameters `rate=0.000075` and `shape=0.75` [4], 99.99% of the species will have an abundance below:

```
> qgamma(0.9999,rate=0.000075, shape=0.75)

[1] 112629.0
```

This provides an abundance high enough to be used as upper integration limit, and that can be used with any community sad distribution, since all density functions has the [q] quantile function. Initial tests suggest that this may provide a general solution (see below in the section on generalization of the function), but systematic evaluation with a wide range of values is still to be done.

Numerical Integration with function `GLIntegrate`

Package `distr` provide an object oriented implementation of distributions. Additionaly, package `distrEx` provide extensions of package distr. That extensions include two functions for integration: `GLIntegrate` and `distrExIntegrate`.

[4] These are the values used by Green & Plotkin fig. 1

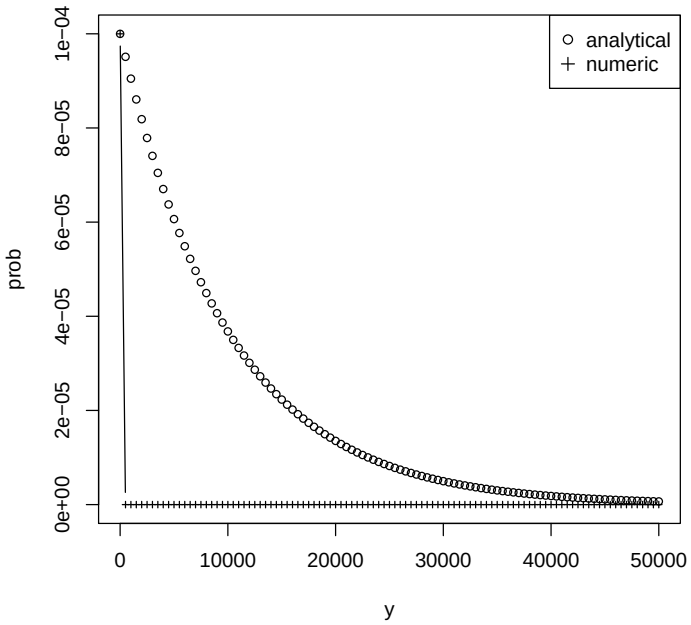


Figura 4: Probability value returned by the function `dpoix` and the function `dsad`. Considering `frac` $= 10^{-1}$ and `rate` $= 10^{-5}$, with $0 \leq$ `y` ≥ 50000

The R-help description of the `GLIntegrate` function says it use Gauss-Legendre quadrature over a finite interval, in contrast with our problem that has an infinite upper limit.

The `distrExIntegrate` function does not have the same limitation relative to the interval of integration. But, as the description of the function explain, it does a numerical integration via `integrate`, in case `integrate` fails a Gauss-Legendre quadrature is performed.

Figures **??** and **??** shows the result from using `GLIntegrate` and `distrExIntegrate` respectively. As expected, from the description of the function, `distrExIntegrate` exibits a shape like the one got from `integrate`.

Numerical Integration with function `adaptIntegrate`

The function `adaptIntegrate` come with the Cubature package. Cubature is a package for mutidimensional integration over hypercubes.

Installation can be performed with:

```
> library("cubature")
```

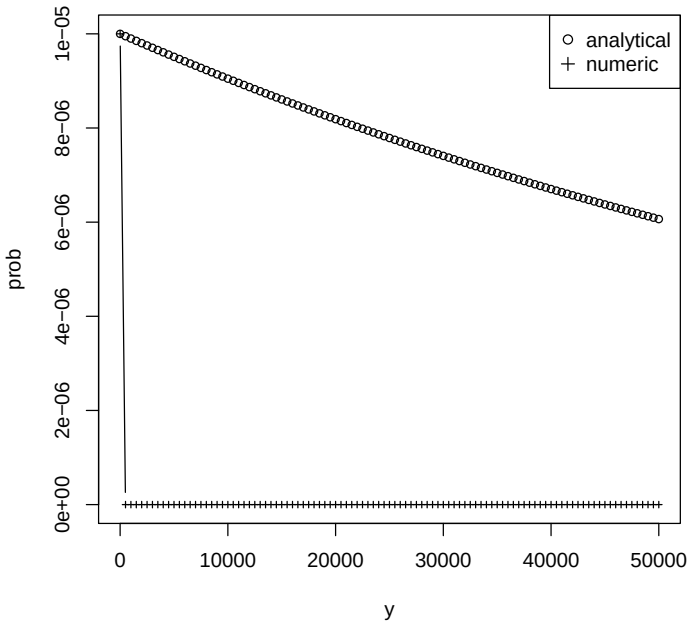


Figura 5: Probability value returned by the function `dpoix` and the function `dsad`. Considering `frac` $= 10^{-1}$ and `rate` $= 10^{-6}$, with $0 \leq$ `y` ≥ 50000

```
> dsad <- Vectorize(FUN=
+                     function(y,a,lambda){
+                       poi <- function(y,n){
+                         w <- y*log(a*n)-lfactorial(y)-a*n
+                         exp(w)
+                       }
+                       f1 <- function(n){
+                         dexp(n,rate=lambda) * poi(y,n)
+                       }
+                                     adaptIntegrate(f1,0,10^6)$integral
+                     },
+                     "y")
```

Some aspects of the numerical integration

One important aspect to verify when dealing with Probability Density Functions - PDFs, is that the sum of probabilities over the entire length is 1. This is tested

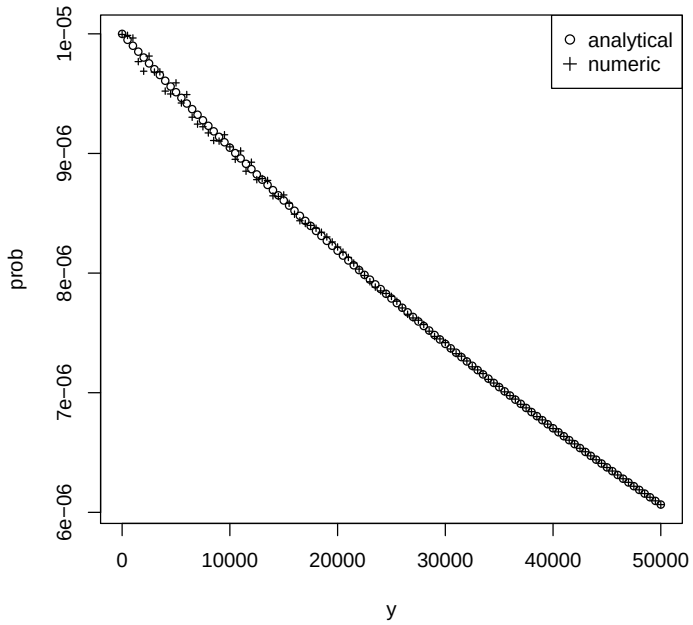


Figura 6: Probability value returned by the function `dpoix` and the function `dsad`. Considering `frac` $= 10^{-1}$ and `rate` $= 10^{-6}$, with $0 \leq$ `y` ≥ 50000. For the upper limit of integration it was used 10^6.
For the

for `dpoix`:

```
> dpoix <- function(y, frac=0.05, rate=1/1000, log=FALSE) {
+     b <- y*log(frac)
+           m <- log(rate)
+           n <- (y+1)*log(rate+frac)
+     if(log)b+m-n else exp(b+m-n)
+ }
> integrate(dpoix,0,Inf)

0.9901637 with absolute error < 6.6e-06
```

Compounding with the gamma

Taking for $g(n)$ the gamma

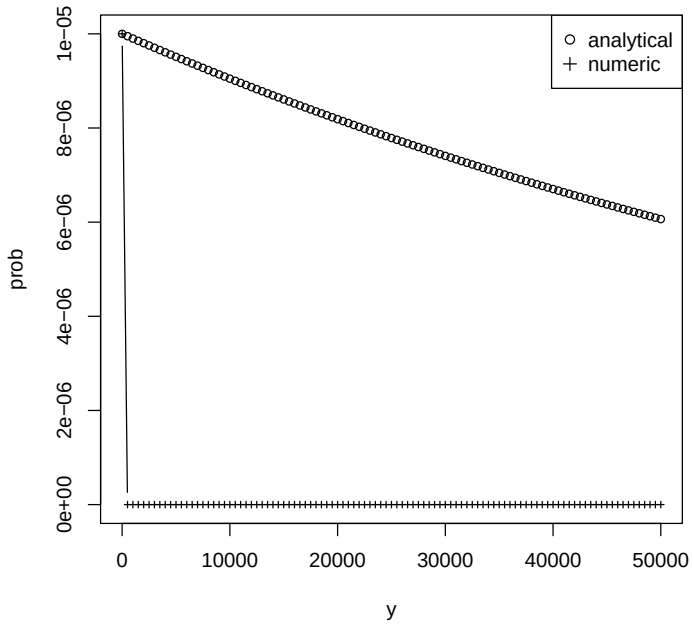


Figura 7: Probability value returned by the function `dpoix` and the function `dsad`. Considering `frac` $= 10^{-1}$ and `rate` $= 10^{-6}$, with $0 \leq$ `y` ≥ 50000. For the upper limit of integration it was used `Inf`.

$$g(n) = \frac{\lambda^{\beta} n^{\beta-1} e^{-\lambda n}}{\Gamma(\beta)} \tag{5}$$

The equation **??** solves to:

$$h(y) = \frac{a^{y} \lambda^{\beta} \Gamma(y+\beta)}{y! \Gamma(\beta)(a+\lambda)^{y+\beta}} \tag{6}$$

We can define this function in R as

```
> dpoig <- function(x, frac, rate, shape) {
+   b <- (frac^x)*(rate^shape)*gamma(x+shape)
+   c <- factorial(x)*gamma(shape)*(frac+rate)^(x+shape)
+   b/c
+ }
```

The equivalent function for numerical integration in R is:

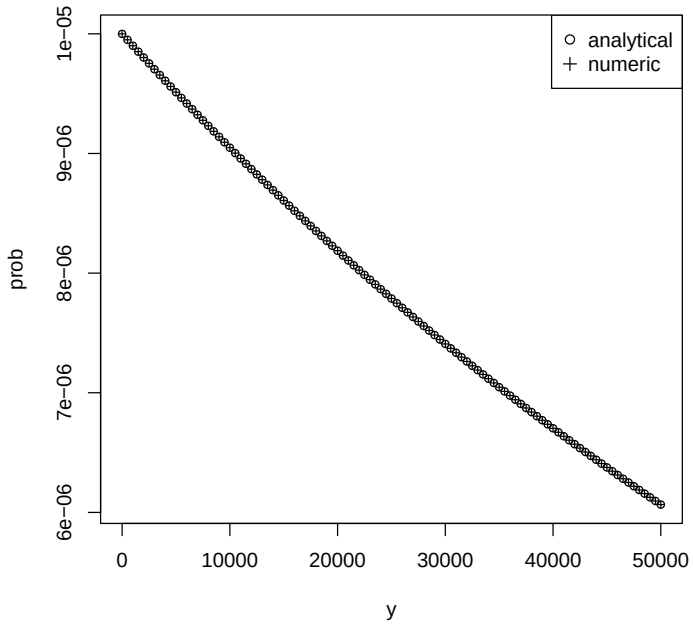


Figura 8: Cubature. Probability value returned by the function `dpoix` and the function `dsad`. Considering `frac` $= 10^{-1}$ and `rate` $= 10^{-6}$, with $0 \leq$ `y` ≥ 50000

```
> dsad_gamma <- Vectorize(FUN=
+ function(y,a,lambda,shape=0.75,fn=2){
+   poi <- function(y,n){
+     w <- y*log(a*n)-lfactorial(y)-a*n
+     exp(w)
+   }
+   if(fn==1){
+     f1 <- function(n){
+       dexp(n,rate=lambda) * poi(y,n)
+     }
+   }else{
+     f1 <- function(n){
+       dgamma(n, shape, rate = lambda, scale = 1/rate) * poi(y,n)
+     }
+   }
+   if (y<10){
+           integrate(f1,0,100/max(a,lambda))$value
```

```
+   }else if (y<50){
+           k <- 6
+           integrate(f1,0,k*y/(a+lambda))$value
+   }else{
+           k <- 1.9920941
+           integrate(f1,0,k*y/(a+lambda))$value
+   }
+ },
+ "y")
```

The functions seems to give the same results

```
> frac <- 10^-1
> rate <- 10^-1
> x <- c(1, 10, 50, 100)
> shape <- 0.75
> dsad_gamma(x,frac,rate,shape,fn=2)

[1] 2.229763e-01 2.640021e-04 1.617665e-16 1.209309e-31

> dpoig(x,frac,rate,shape)

[1] 2.229763e-01 2.640021e-04 1.617665e-16 1.209309e-31
```

General Function

The purpose is to generalize the function to allow the combination of Poisson or Binomial Negative sampling with any distribution in the community. The current version of this generalized function includes the upper limit set by the quantile criteria, and some tweaks to avoid some of the biases related above:

```
> dsad <- function(y,frac,sad,samp="Poisson",log=FALSE,upper=0.9999,...){
+   qcom <- paste("q",deparse(substitute(sad)),sep="")
+   dcom <- paste("d",deparse(substitute(sad)),sep="")
+   dots <- c(as.name("n"),list(...))
+   uplim <- do.call(qcom,c(upper,list(...)))
+   f1 <- function(z){
+     f2 <- function(n){
+       t1 <- do.call(dcom,dots)
+       t2 <- dpois(z,frac*n)
+       ifelse(t1==0|t2==0,0,t1*t2*1e12)
+     }
+   integrate(f2,0,uplim,rel.tol=sqrt(.Machine$double.eps),subdivisions=500)$value
+   }
+   res <- sapply(y,f1)/(1e12*upper)
+   if(log)log(res) else res
+ }
```

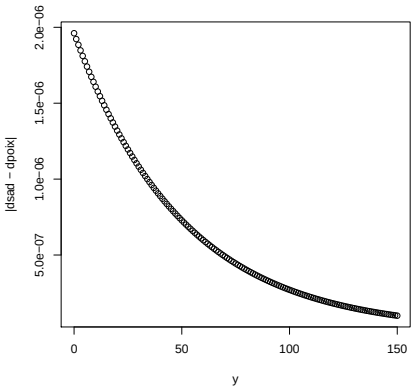
Some observations:

- First working version substituted the old version of `dsad`
- A first version of the help page is already done, see for details
- The function no uses `dpois` for the Poisson distribution
- `rel.tol` and `subdivisions` increased following `https://stat.ethz.ch/pipermail/r-help/attachments/20100921/56db7df0/attachment.pl`
- `log` argument added; necessary to model fitting by maximum likelihood
- arguments to be passed to the community distribution choosen by `sad` vary, and are all under `....` They should be named.
- Old functions removed from the directory

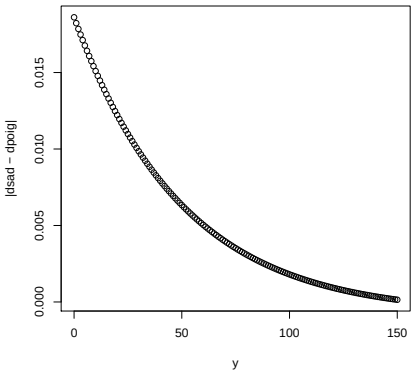
To Do List

1. Functions with analytical solution for Poisson-truncated hyperbolic
2. To use `dnbinom` to check `dpoig`: a compound Poisson-gamma is a negative binomial for integers, so they must give the same results.
3. To set the parameters of `dpoig` to match the parameters of `dnbinom`
4. To standardize arguments and their names across the density functions.
5. To check the thoroughly compound solutions of the generic function `dsads` against their analytical counterparts (`dpoix`, `dpoig` ...)
6. After all checks with Poisson sampling were done and OK, proceed to set the functions for Binomial Sampling (Table 2 of Green & Plotkin).

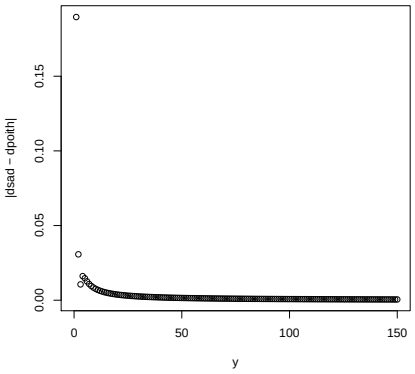
To accomplish the item 5 above, it was measured the difference between the curves, Figure **??**:



(a) `frac` $= 0.05$, `rate` $= 1/1000$



(b) `frac` $= 0.05$, `rate` $= 1/1000$, `prob` $= 0.5$, `shape` $= 1$



(c) `frac` $= 0.05$, `Theta=c(2,2,2,2)`

Figura 9: Difference between curves.