

# Template families

## Contents

Read in catalog from <i>Cochran et al.</i> [2018]	1
Function for my version of the interevent time distribution of <i>Saichev and Sornette</i> [2007]	1
Find the families	2
Make plots like their Fig. 3	2
First the time-consuming fitting of the <i>Saichev and Sornette</i> [2007] distributions . . . . .	2
Then extract the estimates (and the found max likelihood) . . . . .	3
Stan . . . . .	3
Finally make plots . . . . .	6
Some puzzling examples . . . . .	137
Widths of aftershock fraction CIs . . . . .	142
Comparison with <i>Hainzl et al.</i> [2006] . . . . .	142
Comparison to standard deviation of dt ala <i>Cochran et al.</i> [2018] . . . . .	143
Aftershock fraction estimation a la <i>Zaliapin and Ben-Zion</i> [2016] . . . . .	145
Todo	149
References	149

```
require(MASS)
require(rstan)
options(mc.cores = parallel::detectCores())
rstan_options(auto_write = TRUE)
source("readGrowclust.R")
```

## Read in catalog from *Cochran et al.* [2018]

```
s = readGrowclust("allCat_cc7_bs10_wTemplate.txt")
```

## Function for my version of the interevent time distribution of *Saichev and Sornette* [2007]

```
dSS2007.krd = function(t, p, loglambda.c, n, DEBUG=FALSE) {
  lambda.c = 10^loglambda.c
  if (DEBUG) print(c(p, lambda.c, n))
```

```

if (p <= 1 || p >= 3 || lambda.c <= 0 || n < 0 || n > 1) return(rep(0, length(t)))

d = (1-n) + n*(1 + t/(lambda.c))^{(-p+1)}
d = d*d
d = d - n/(lambda.c)*(1-p)*(1 + t/(lambda.c))^{(-p)}
d = d*exp( -(1-n)*t - (n*lambda.c/(2-p))*((1 + t/(lambda.c))^{(-p+2)} - 1))

d[t<0] = 0

return(d)
}
dSS2007.krd.ln = function(tau, ...) {dSS2007.krd(exp(tau), ...) * exp(tau)} # the pdf for the log of the
dtplot = 10^seq(-15, 5, length=1000)
dtauplot = log(dtplot)

```

## Find the families

```

fids = sort(unique(s$parent))

nev.in.fam = tapply(s$parent, s$parent, length)

```

## Make plots like their Fig. 3

### First the time-consuming fitting of the *Saichev and Sornette [2007]* distributions

First try using `fitdistr()`. Trying three different optimization methods. Note that sometimes when `fitdistr()` fails, the optimization worked but then failed when trying to calculate the Hessian for the standard error measures.

```

fitdistr.time = Sys.time()
fuse = which(nev.in.fam > 100)
fSS = fSS.SANN = fSS.BFGS = Cv = meandt = NULL

for (f in fuse) {
  i.f = match(f, fuse)
  ev.use = which(s$parent == fids[f])

  dt = diff(s$t0numeric[ev.use])
  meandt[i.f] = mean(dt)
  dt = dt/meandt[i.f]

  fSS[[i.f]] <- try(fitdistr(dt, dSS2007.krd, start=list(p=1.2, loglambda.c=log10(1/meandt[i.f])), n=0.5,
                        method="Nelder-Mead", DEBUG=FALSE), silent=TRUE)
  fSS.SANN[[i.f]] <- try(fitdistr(dt, dSS2007.krd, start=list(p=1.2, loglambda.c=log10(1/meandt[i.f])), n=0.5,
                        method="SANN", DEBUG=FALSE), silent=TRUE)
  fSS.BFGS[[i.f]] <- try(fitdistr(dt, dSS2007.krd, start=list(p=1.2, loglambda.c=log10(1/meandt[i.f])), n=0.5,
                        method="BFGS", DEBUG=FALSE), silent=TRUE)

  Cv[i.f] = sd(dt)/mean(dt)
}

```

```

}

fitdistr.time = Sys.time() - fitdistr.time

```

Then extract the estimates (and the found max likelihood)

```

p = sapply(fSS, function(f){ifelse(attr(f, 'class') == "try-error", NA, f$est["p"])})
p.SANN = sapply(fSS.SANN, function(f){ifelse(attr(f, 'class') == "try-error", NA, f$est["p"])})
p.BFGS = sapply(fSS.BFGS, function(f){ifelse(attr(f, 'class') == "try-error", NA, f$est["p"])})


n = sapply(fSS, function(f){ifelse(attr(f, 'class') == "try-error", NA, f$est["n"])})
n.SANN = sapply(fSS.SANN, function(f){ifelse(attr(f, 'class') == "try-error", NA, f$est["n"])})
n.BFGS = sapply(fSS.BFGS, function(f){ifelse(attr(f, 'class') == "try-error", NA, f$est["n"])})


loglambda.c = sapply(fSS, function(f){ifelse(attr(f, 'class') == "try-error", NA, f$est["loglambda.c"])}
loglambda.c.SANN = sapply(fSS.SANN, function(f){ifelse(attr(f, 'class') == "try-error", NA, f$est["loglambda.c"])}
loglambda.c.BFGS = sapply(fSS.BFGS, function(f){ifelse(attr(f, 'class') == "try-error", NA, f$est["loglambda.c"])}


l = sapply(fSS, function(f){ifelse(attr(f, 'class') == "try-error", NA, f$loglik)})
l.SANN = sapply(fSS.SANN, function(f){ifelse(attr(f, 'class') == "try-error", NA, f$loglik)})
l.BFGS = sapply(fSS.BFGS, function(f){ifelse(attr(f, 'class') == "try-error", NA, f$loglik)})

```

Nelder-Mead, SANN, and BFGS “worked” for 44, 116, 46 of the families, respectively (out of 130 total families). And took 44 secs seconds.

## Stan

Since `fitdist()` had trouble with many of the families (and estimated errors in the parameters are probably meaningless), let’s also try MCMC via Stan. The Stan program is:

```
writeLines(readLines("dSS2007.krd.stan"))
```

```

## functions {
## // log of pdf at a single interevent times t given the parameters p, loglambdac, and n
## // I can't get stan to work with a function that returns the sum of the log densities
## // for a vector of interevent times...
##   real dSS2007_krd_lpdf(real t, real p, real loglambdac, real n) {
##
##     real lambdac = 10^loglambdac;
##     real d;
##     real ld;
##
##     if (p <= 1) reject("p cannot be <= 1");
##     if (n < 0 || n > 1) reject("n must be between 0 and 1");
##     ##
##     d = (1-n) + n*(1 + t/(lambdac))^{-(p+1)};
##     d = d * d;
##     d = d - n/(lambdac)*(1-p)*(1 + t/(lambdac))^{-(p)};
##     ##
##     ld = log(d);
##     ##
##     ld = ld - (1-n)*t - (n*lambdac/(2-p))*((1 + t/(lambdac))^{-(p+2)} - 1);
##   }
}

```

```

##      return(1d);
##  }
##
## // unnormalized prior for p: flat from 1 to p1, then exponential decay with const. s0
## // I've mostly been using p1 = 1.5 and s0 = 0.3
## real p_prior_lpdf(real p, real p1, real s0) {
##
##     if (p <= 1) reject("p cannot be <= 1");
##
##     if (p <= p1) return(0);
##     else return(-(p - p1)/s0);
## }
##
## data {
##     int N;          // number of interevent times
##     real t[N];    // normalized interevent times
## }
##
## parameters {
##     real<lower=1, upper=3> p; // Omori p-value
##     real loglambdac; // log10(lambda * c), where lambda is mean event rate, and c is modified-Omori c
##     real<lower=0, upper=1> n; // branching ratio (and aftershock fraction)
## }
##
## model {
##     p ~ p_prior(1.5, 0.3);
##     loglambdac ~ normal(-3.4, 5); // should transform and put a prior directly on c
##     for (i in 1:N) t[i] ~ dSS2007_krd(p, loglambdac, n);
## }

```

Note the prior on  $p$  (as of this writing, flat from 1 to 1.5 then exponentially decaying at higher values). I also added a weak prior on  $\log_{10}(\lambda c)$  ( $\log_{10}(\bar{\lambda}c)$ ) to avoid some pathological behavior. I've tried using stronger priors on  $\log_{10}(\lambda c)$  to see if that would help with some of the convergence issues, but it didn't (and haven't thought about whether this would be justified in any case).

Just using the default number of chains, iterations, starting points, etc. And also, not checking for convergence, mixing, etc. yet.

```

stan.time = rep(NA, length(fuse))
f.stan = NULL

for (f in fuse) {
  i.f = match(f, fuse)
  stan.time[i.f] = Sys.time()
  ev.use = which(s$parent == fids[f])

  dt = diff(s$t0numeric[ev.use])
  meandt[i.f] = mean(dt)
  dt = dt/meandt[i.f]

  if (i.f == 1)
#    f.stan[[i.f]] = stan(file="dSS2007.krd.stan", data=list(N=length(dt), t=dt), open_progress=FALSE)

```

```

f.stan[[i.f]] = stan(file="dSS2007.krd.fixedloglambdac.stan",
                      data=list(N=length(dt), t=dt, lambda_bar=1/meandt[i.f]), open_progress=FALSE)
else
#   f.stan[[i.f]] = stan(fit=f.stan[[i.f-1]], data=list(N=length(dt), t=dt), open_progress=FALSE)
f.stan[[i.f]] = stan(fit=f.stan[[i.f-1]], data=list(N=length(dt), t=dt, lambda_bar=1/meandt[i.f]), open_progress=FALSE)

stan.time[i.f] = Sys.time() - stan.time[i.f]
}

```

Stan took 1125 seconds.

Extract the modes, means, and CIs (note that CIs are 68%!).

```

p.stan.median = loglambdac.stan.median = n.stan.median = rep(NA, length(fuse))
p.stan.mode = loglambdac.stan.mode = n.stan.mode = rep(NA, length(fuse))
p.stan.mean = loglambdac.stan.mean = n.stan.mean = rep(NA, length(fuse))
pCI.stan = loglambdacCI.stan = nCI.stan = matrix(NA, nrow=length(fuse), ncol=2)

for (f in fuse) {
  i.f = match(f, fuse)

  dt_sim = extract(f.stan[[i.f]], permute=TRUE)

  # mode
  p.stan.mode[i.f] = dt_sim$p[which.max(dt_sim$lp_)]
#  loglambdac.stan.mode[i.f] = dt_sim$loglambdac[which.max(dt_sim$lp_)]
  n.stan.mode[i.f] = dt_sim$n[which.max(dt_sim$lp_)]

  # mean
  p.stan.mean[i.f] = mean(dt_sim$p)
#  loglambdac.stan.mean[i.f] = mean(dt_sim$loglambdac)
  n.stan.mean[i.f] = mean(dt_sim$n)

  # median
  p.stan.median[i.f] = median(dt_sim$p)
#  loglambdac.stan.median[i.f] = median(dt_sim$loglambdac)
  n.stan.median[i.f] = median(dt_sim$n)

  # CIs
  alpha = 0.32 # 68% CIs
  pCI.stan[i.f,] = quantile(dt_sim$p, probs=c(alpha/2, 1-alpha/2))
#  loglambdacCI.stan[i.f,] = quantile(dt_sim$loglambdac, probs=c(alpha/2, 1-alpha/2))
  nCI.stan[i.f,] = quantile(dt_sim$n, probs=c(alpha/2, 1-alpha/2))

  c = 90 # seconds, needs to match in dSS2007.krd.fixedloglambdac.stan
  loglambdac.stan.mode[i.f] = loglambdac.stan.median[i.f] = loglambdac.stan.mean[i.f] =
    log10((1/meandt[i.f]) * c)
}

```

Extract some measures of convergence/mixing/performance/etc.

```

bfmi = sapply(f.stan, get_bfmi)
n_div = sapply(f.stan, get_num_divergent)
rhat = apply(t(sapply(f.stan, function(f){summary(f)$summary[, "Rhat"]})), 1, max)
n_eff = apply(t(sapply(f.stan, function(f){summary(f)$summary[, "n_eff"]})), 1, min)

```

Stan docs imply that `bfmi` < 0.3, `n_div` more than a few, `rhat` < 1.1, or `n_eff` < 1/10 total number of sampling interations (so 400 in this case - total number of sampling iterations is 4 chains x 1000/chain = 4000) would indicate problems.

- Fail `bfmi` test: 0/520
- Fail `n_div` test: 0/130
- Fail `rhat` test: 0/130
- Fail `n_eff` test: 4/130

## Finally make plots

Like their Fig. 3 with *Saichev and Sornette [2007]* fits (and estimated parameters) added in. And replaced upper, right location plot with Stan parameter space plot.

```

mkTimeMagCumNumberPlot = function(s, ev.use, fam) {
  plot(s$t0[ev.use], s$mag[ev.use], main=paste("Family", fam, "; nev =", length(ev.use)))
  lines(s$t0[ev.use], (1:length(ev.use)) * diff(range(s$mag[ev.use])) / length(ev.use) + min(s$mag[ev.us
    col="red")
}

mkLocationPlot = function(s, ev.use, ev.plot) {
  plot(s$utmx[ev.plot]/1e3, s$utmy[ev.plot]/1e3, col=gray(0.3), cex=0.3, asp=1)
  points(s$utmx[ev.use]/1e3, s$utmy[ev.use]/1e3, col="red3", cex=0.3)
}

mkLogHist = function(s, ev.use, fam, fSS,
                     p.stan, pCI.stan, loglambdac.stan, n.stan, nCI.stan) {
  dt = diff(s$t0numeric[ev.use])
  dt = dt/mean(dt)

  loghist(dt, nbr=15)
  if (attr(fSS, 'class') != "try-error") {
    lines(dtplot, dSS2007.krd(dtplot, p=fSS$est["p"], loglambdac.c=fSS$est["loglambdac.c"],
                               n=fSS$est["n"]), col="orange3", lwd=2)
    text(10^par("usr")[2], 10^par("usr")[4], adj=c(1.1,1.5),
         labels=paste(names(fSS$est), "=", format(fSS$est, digits=3), collapse="; "),
         cex=0.5, col="orange3")
  }
  lines(dtplot, dSS2007.krd(dtplot, p=p.stan, loglambdac.c=loglambdac.stan,
                             n=n.stan), col="red2", lwd=2, lty="dashed")
  text(10^par("usr")[2], 10^par("usr")[4], adj=c(1.1,3.5),
       labels=paste("p =", format(p.stan, digits=3), " [", format(pCI.stan[1], digits=3), ",",
                   format(pCI.stan[2], digits=3), "]; loglambdac =", format(loglambdac.stan, digits=2),
                   ";; n =", format(n.stan, digits=2), " [", format(nCI.stan[1], digits=2), ",",
                   format(nCI.stan[2], digits=2), "]; nCI.stan =", format(nCI.stan, digits=2)),
       cex=0.5, col="red2")
}

mkHistOfLn = function(s, ev.use, fam, fSS,
                     p.stan, pCI.stan, loglambdac.stan, n.stan, nCI.stan) {
  dt = diff(s$t0numeric[ev.use])
  dt = dt/mean(dt)

  h = hist(log(dt), breaks=25, main=paste("Cv =", format(sd(dt), digits=2)), probability=TRUE,
           xlab="ln(dt)", ylab="frequency", xaxt="bottom", xaxs="log", xlog=TRUE, plot=FALSE)
}
```

```

      xlab="ln(dt)")
if (attr(fSS, 'class') != "try-error") {
  lines(dtauplot, dSS2007.krd.ln(dtauplot, p=fSS$est["p"],
                                    loglambda.c=fSS$est["loglambda.c"], n=fSS$est["n"]),
        col="orange3", lwd=2)
#  text(par("usr")[2], par("usr")[4], adj=c(1.1,1.5),
#       labels=paste(names(fSS$est), "=", format(fSS$est, digits=2), collapse="; "), cex=0.5)
}
  lines(dtauplot, dSS2007.krd.ln(dtauplot, p=p.stan, loglambda.c=loglambdac.stan,
                                    n=n.stan), col="red2", lwd=2, lty="dashed")
}

mkMCMCParamSpacePlot = function(f.stan, bfmi, n_div, rhat, n_eff) {
  dt_sim = extract(f.stan, permute=TRUE)
  palette(jet(100))
  col = 1 + 99*(dt_sim$lp__ - min(dt_sim$lp__))/diff(range(dt_sim$lp__))
  plot(dt_sim$p, dt_sim$n, col=col, xlim=c(1, 2.5), ylim=c(0,1), xlab="p-value",
       ylab="Branching ratio / aftershock fraction",
       main=paste("BFMI =", format(bfmi, digits=2), "; n_div =", n_div, "\n",
                  "rhat =", format(rhat, digits=3), "; n_eff =", format(n_eff, digits=0)))
  mode = which.max(dt_sim$lp__)
  points(dt_sim$p[mode], dt_sim$n[mode], pch=4)
  points(median(dt_sim$p), median(dt_sim$n), pch=3)
  points(mean(dt_sim$p), mean(dt_sim$n), pch=2)
  legend("topright", c("Mode", "Median", "Mean"), pch=c(4,3,2))
}

mkFourPanelPlot = function(s, ev.use, ev.plot, fam, bfmi, n_div, rhat, n_eff,
                           fSS, f.stan, p.stan, pCI.stan,
                           loglambdac.stan, n.stan, nCI.stan) {
  par(mifrow=c(2,2))

  mkTimeMagCumNumberPlot(s, ev.use, fam)

#  mkLocationPlot(s, ev.use, ev.plot)

  mkMCMCParamSpacePlot(f.stan, bfmi, n_div, rhat, n_eff)

  mkLogHist(s, ev.use, fam, fSS, p.stan, pCI.stan,
             loglambdac.stan, n.stan, nCI.stan)

  mkHistOfLn(s, ev.use, fam, fSS, p.stan, pCI.stan,
             loglambdac.stan, n.stan, nCI.stan)
}

ev.plot = sample(1:length(s$t0), 1000) # plot just a sample of all seismicity

par(mifrow=c(2,2))
for (f in fuse) {
  i.f = match(f, fuse)
  ev.use = which(s$parent == fid[f])

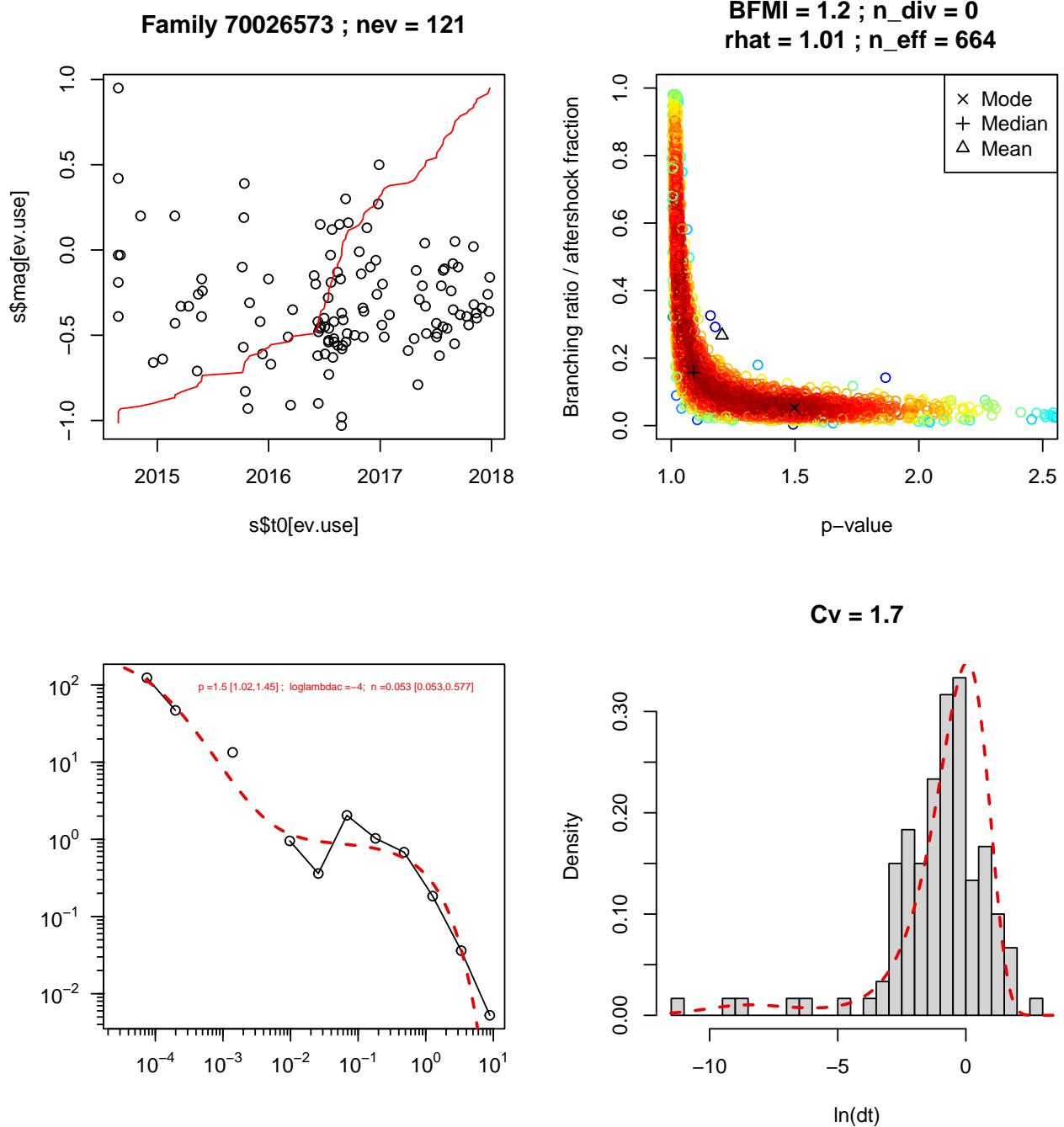
```

```

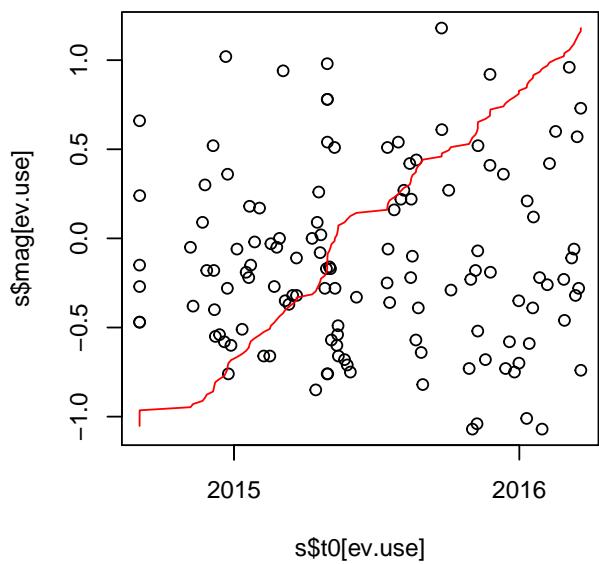
mkFourPanelPlot(s, ev.use, ev.plot, fids[f], bfmi[i.f], n_div[i.f], rhat[i.f], n_eff[i.f],
                fSS[[i.f]], f.stan[[i.f]],
                p.stan.mode[i.f], pCI.stan[i.f,], loglambdac.stan.mode[i.f],
                n.stan.mode[i.f], nCI.stan[i.f,])

cat("\r\n\r\n") # to try to get figures to not end up side-by-side
}

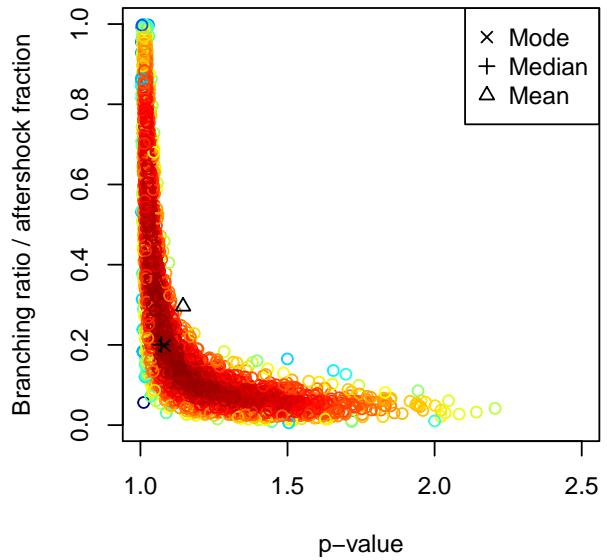
```



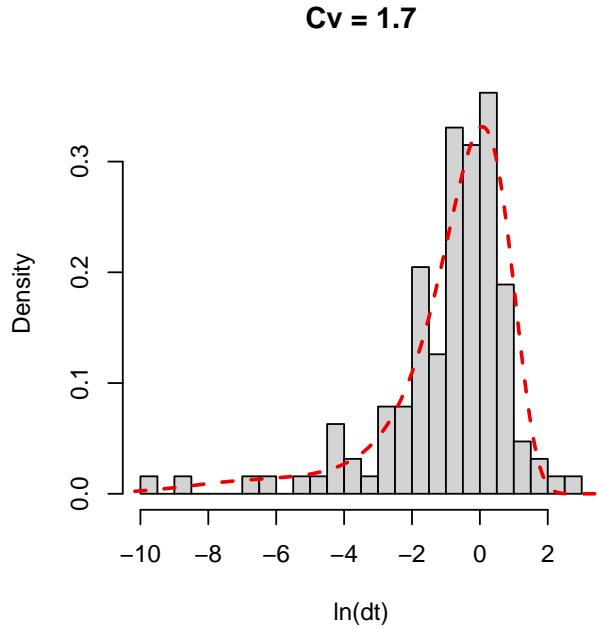
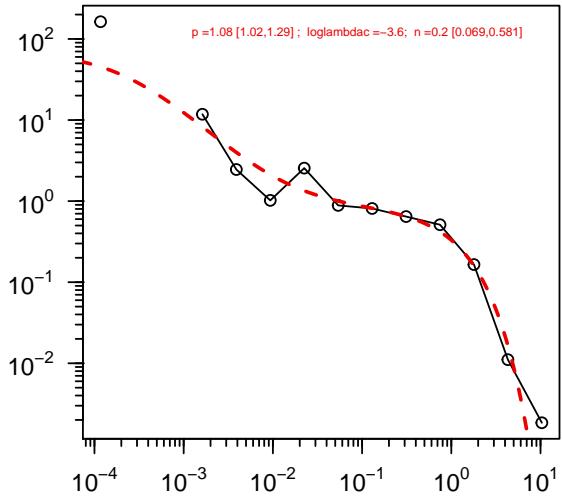
**Family 70028123 ; nev = 128**



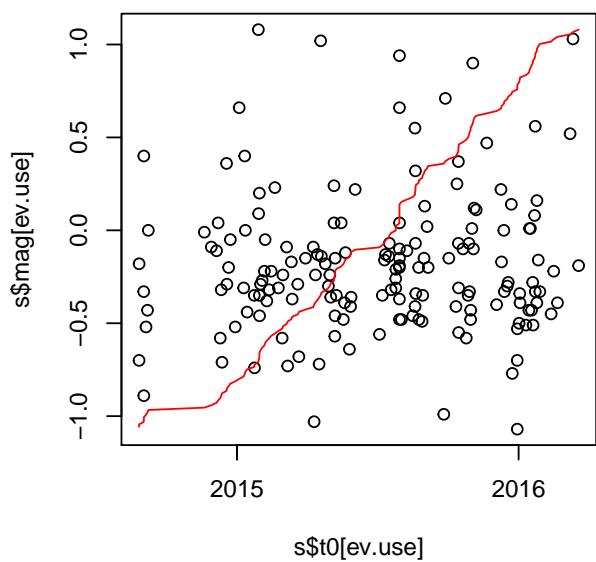
**BFMI = 1.1 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 680**



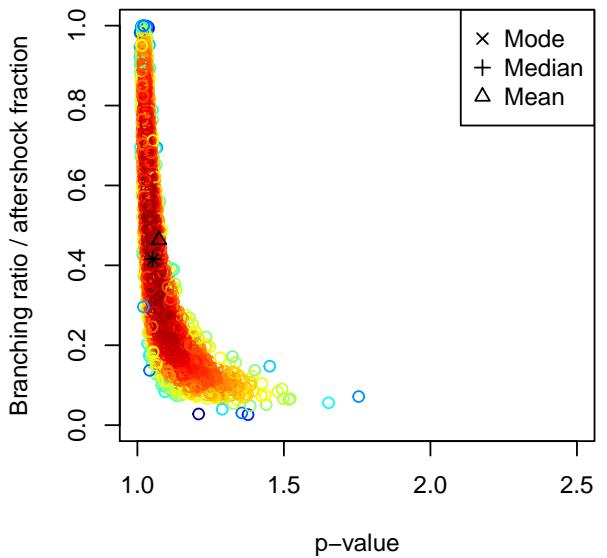
**Cv = 1.7**



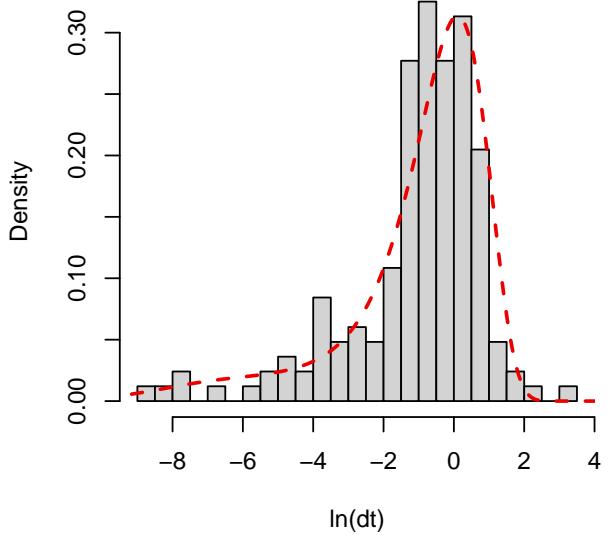
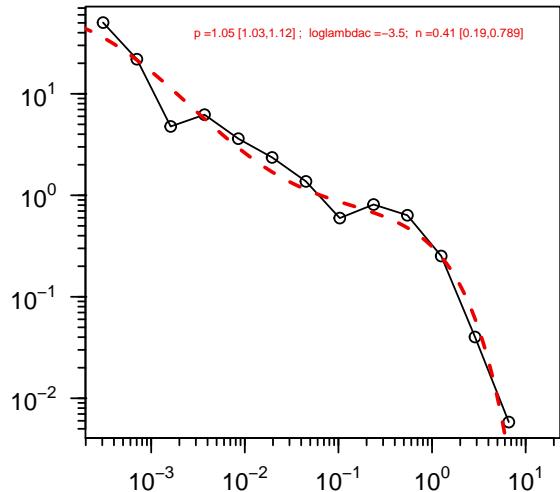
**Family 70028138 ; nev = 167**



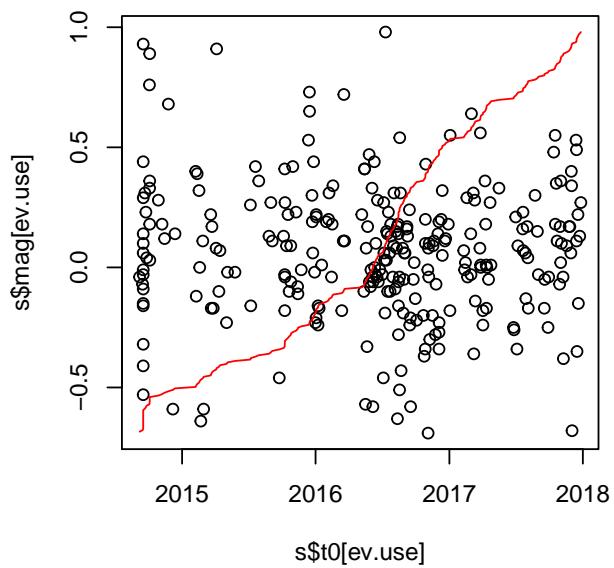
**BFMI = 1.1 ; n\_div = 0  
rhat = 1 ; n\_eff = 713**



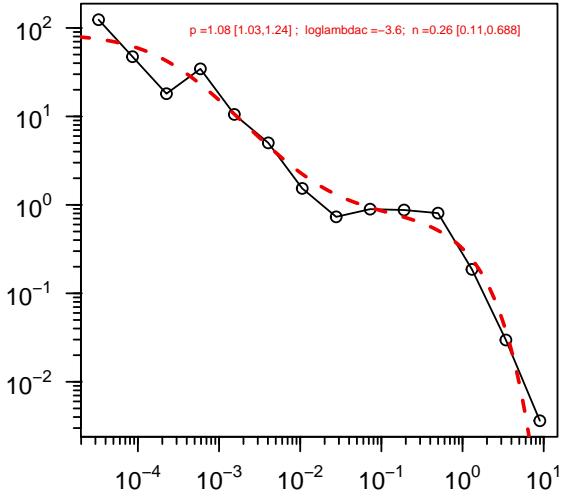
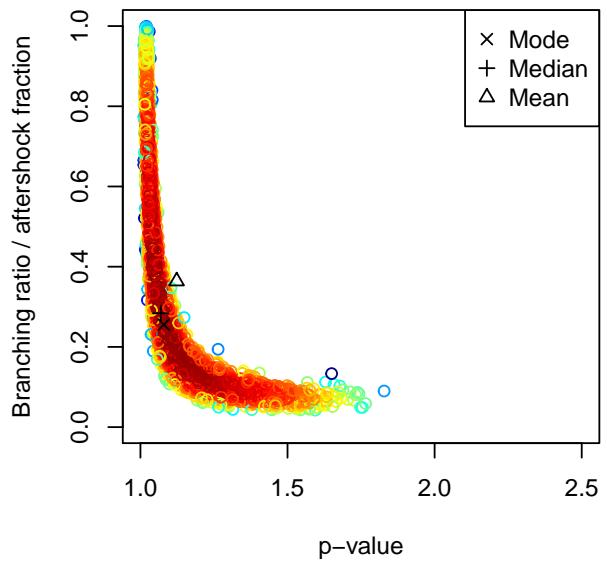
**Cv = 1.9**



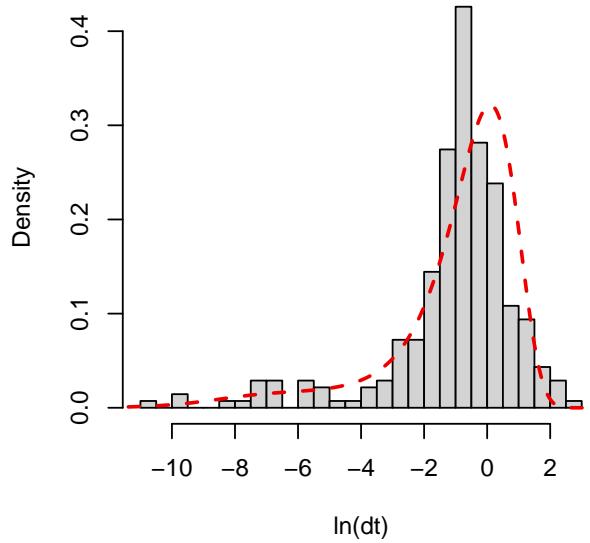
**Family 70033938 ; nev = 278**



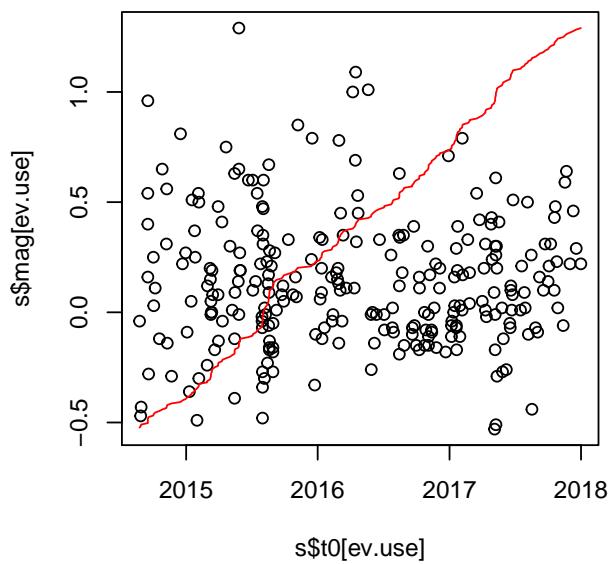
**BFMI = 1 ; n\_div = 0  
rhat = 1 ; n\_eff = 630**



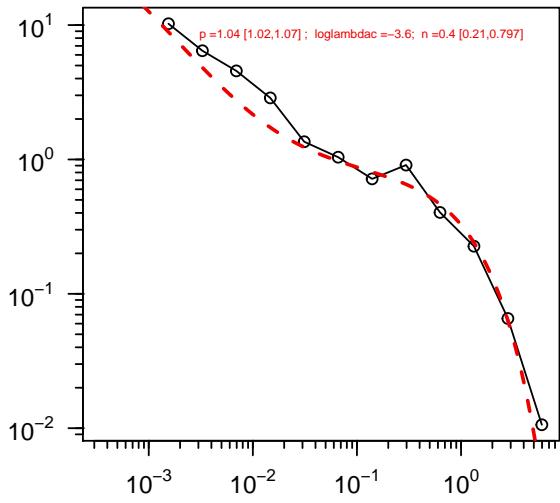
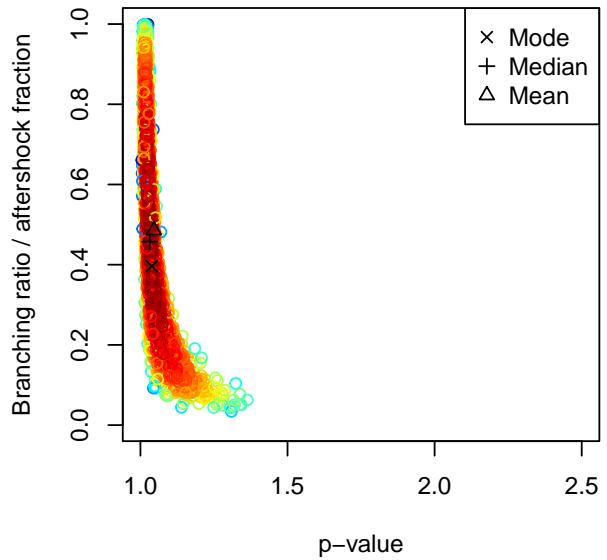
**Cv = 1.7**



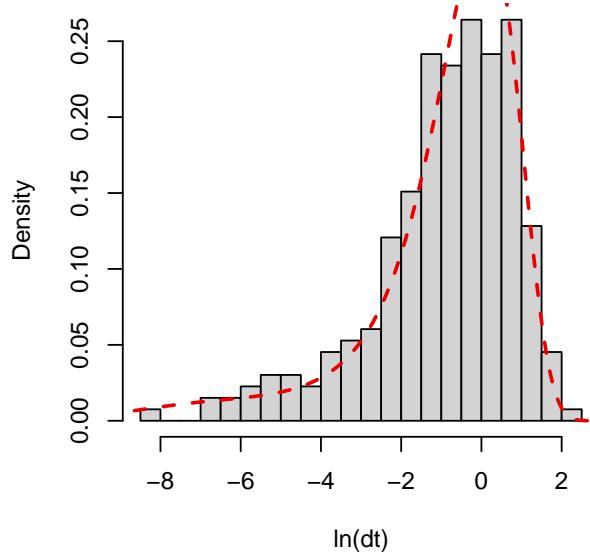
**Family 70033948 ; nev = 266**



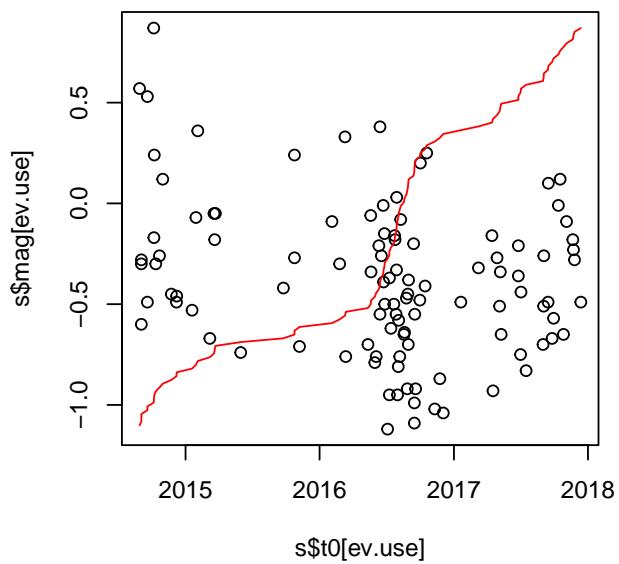
**BFMI = 1.1 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 610**



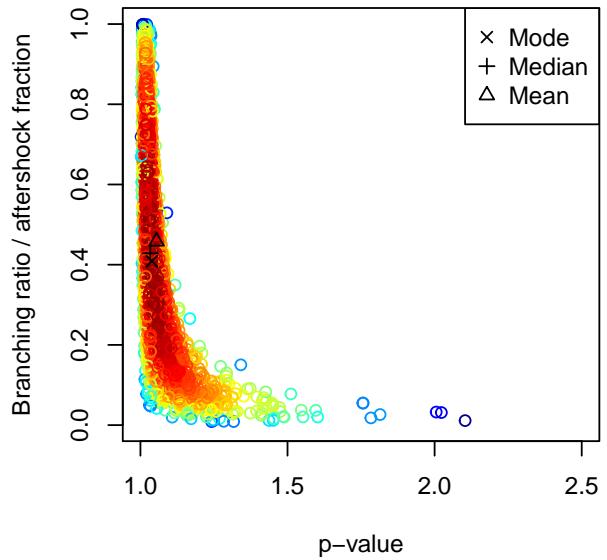
**Cv = 1.2**



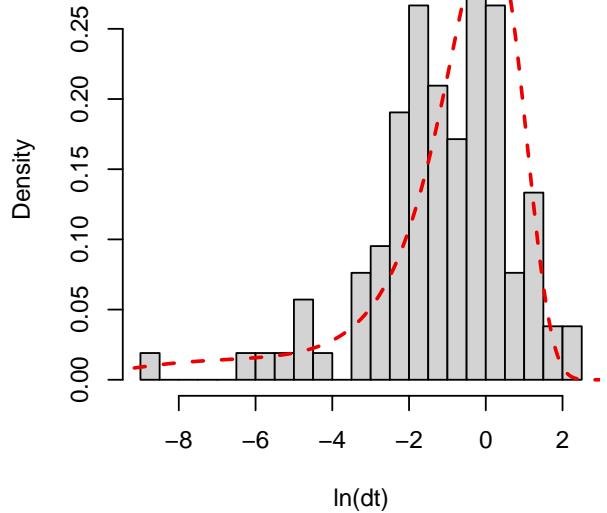
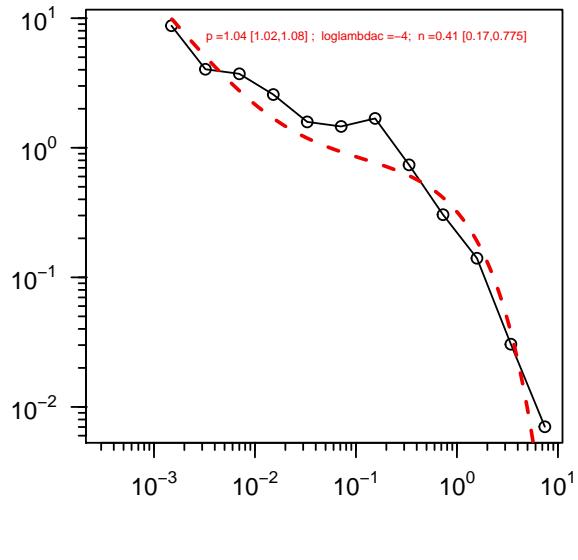
**Family 70034323 ; nev = 106**



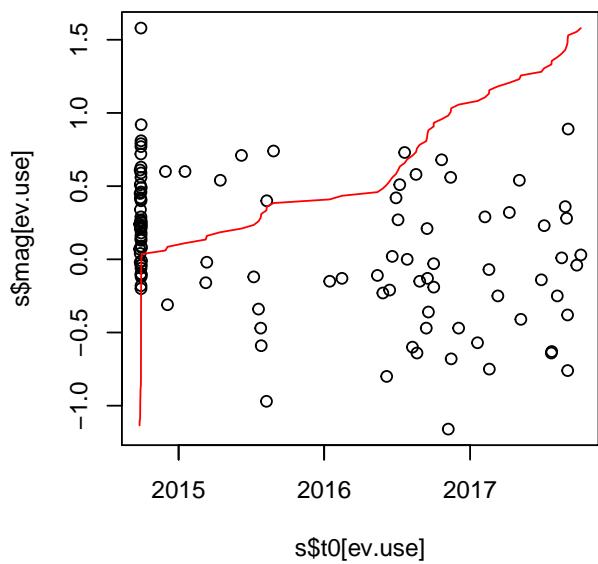
**BFMI = 1.1 ; n\_div = 0  
rhat = 1 ; n\_eff = 792**



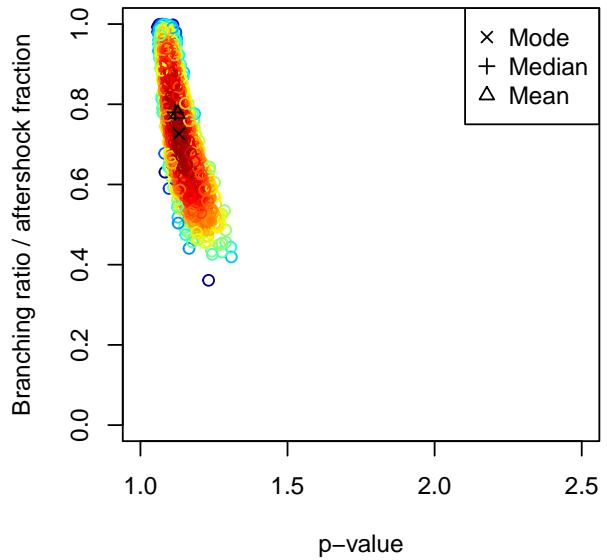
**Cv = 1.6**



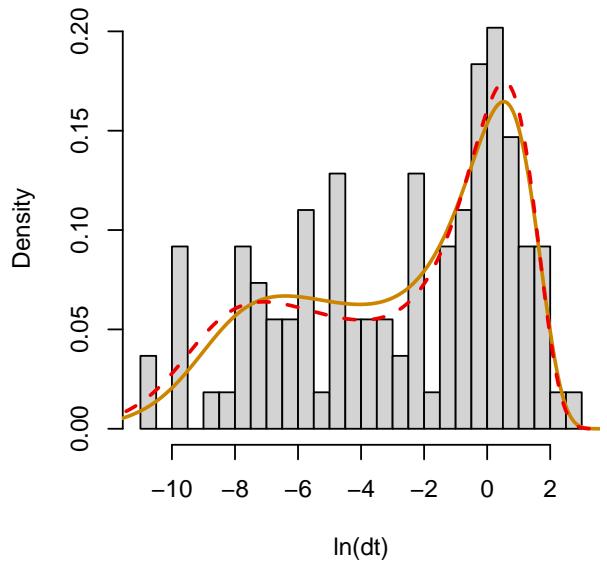
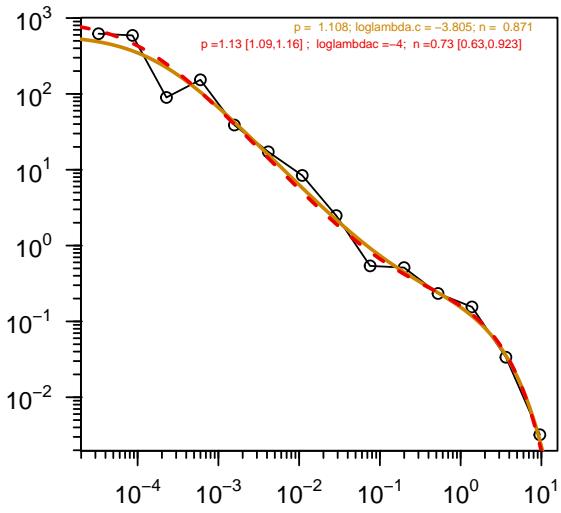
**Family 70036713 ; nev = 110**



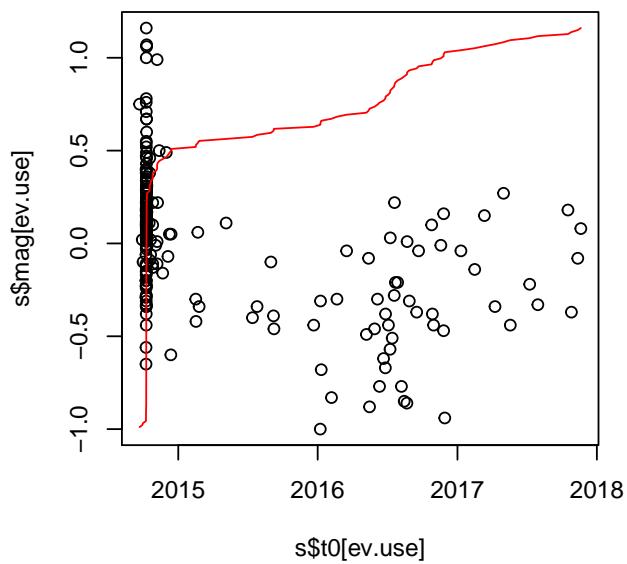
**BFMI = 0.77 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 661**



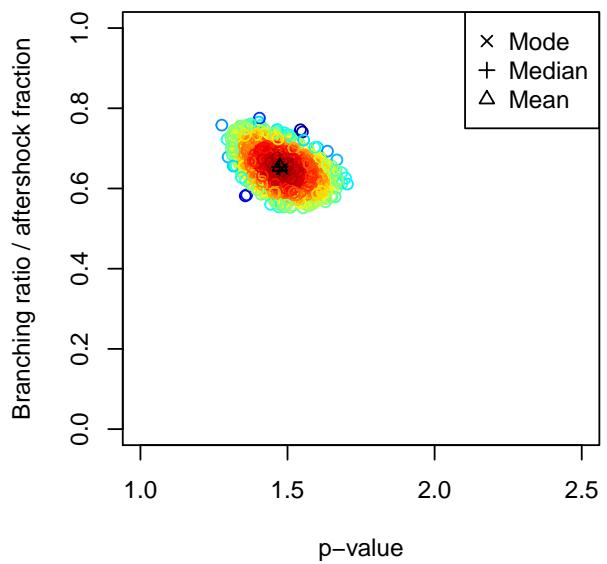
**Cv = 2**



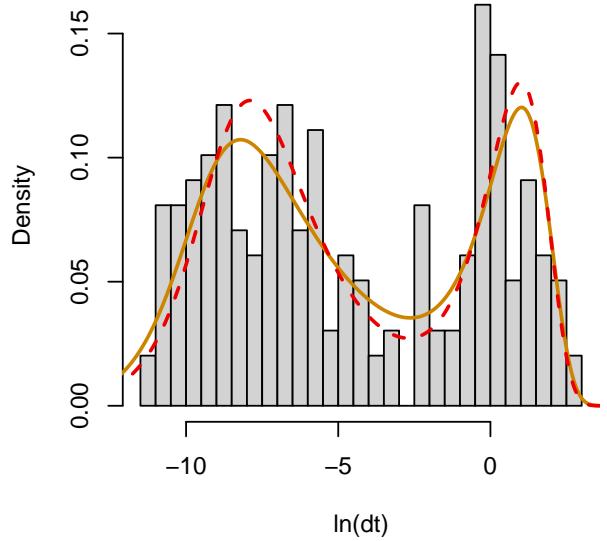
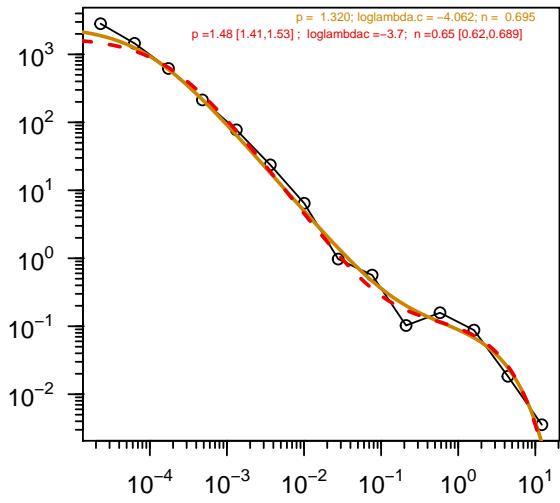
**Family 70038808 ; nev = 199**



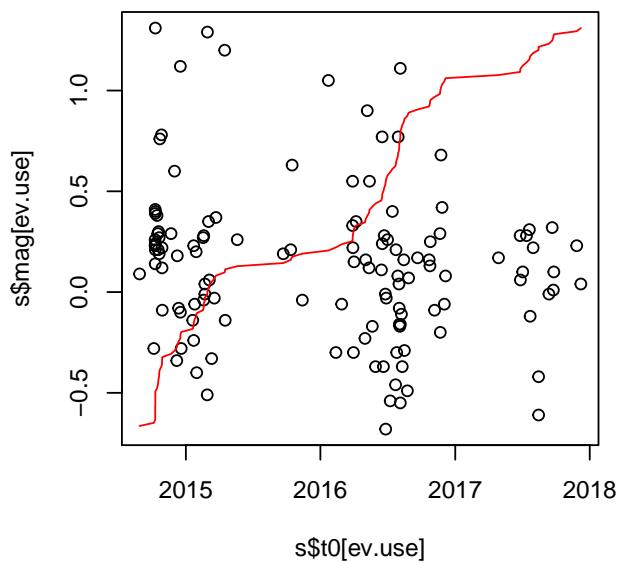
**BFMI = 1 ; n\_div = 0  
rhat = 1 ; n\_eff = 1914**



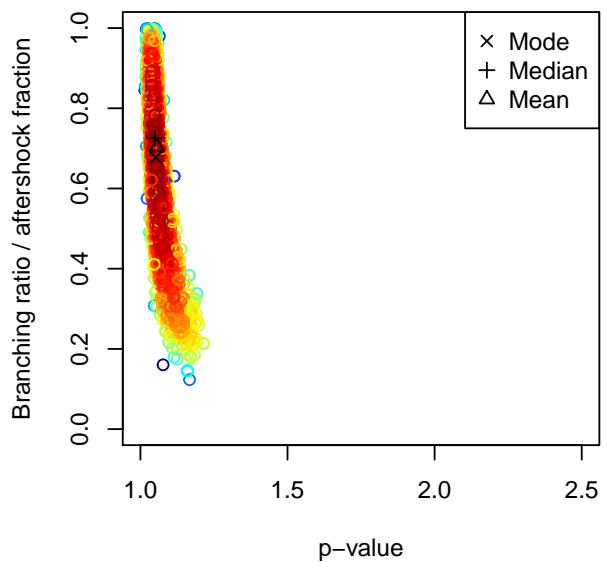
**Cv = 2.5**



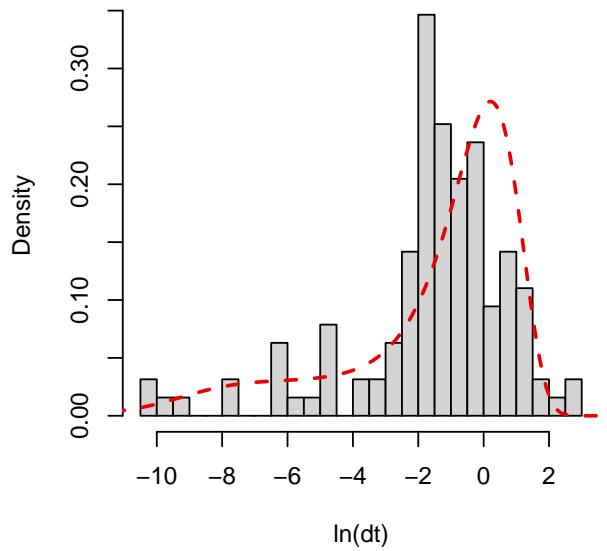
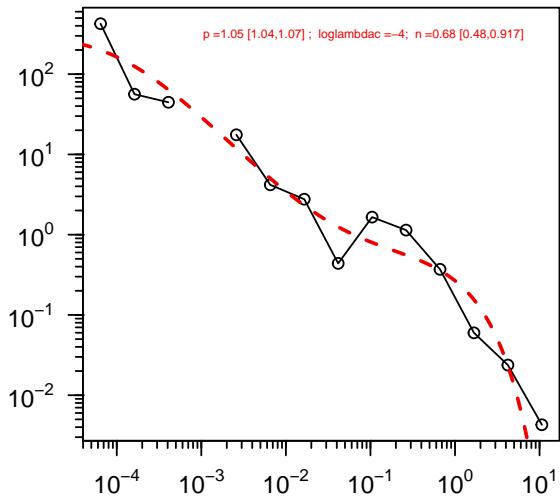
**Family 70038893 ; nev = 128**



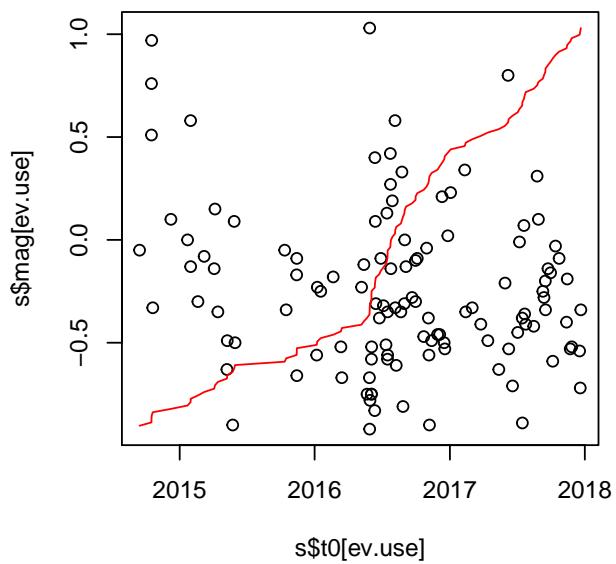
**BFMI = 1 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 487**



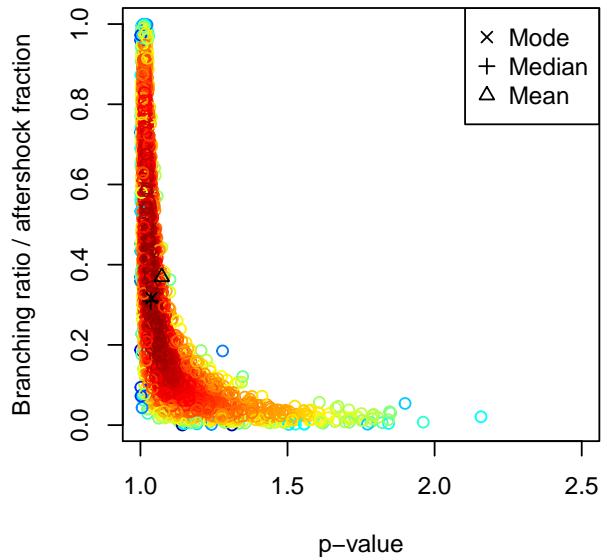
**Cv = 2.1**



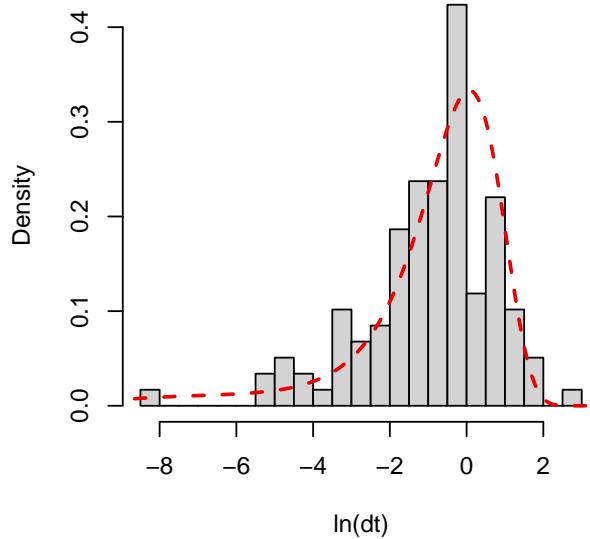
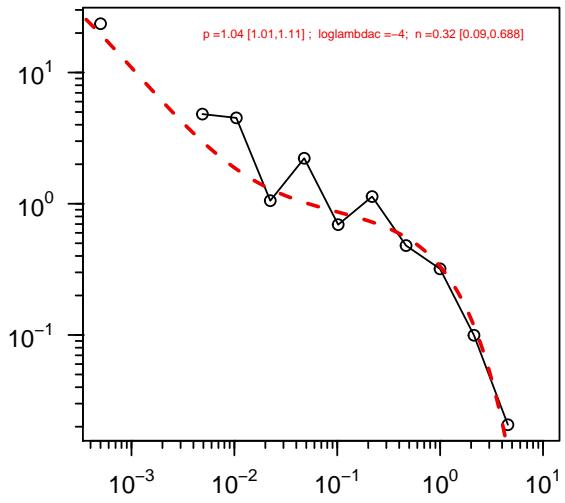
**Family 70039733 ; nev = 119**



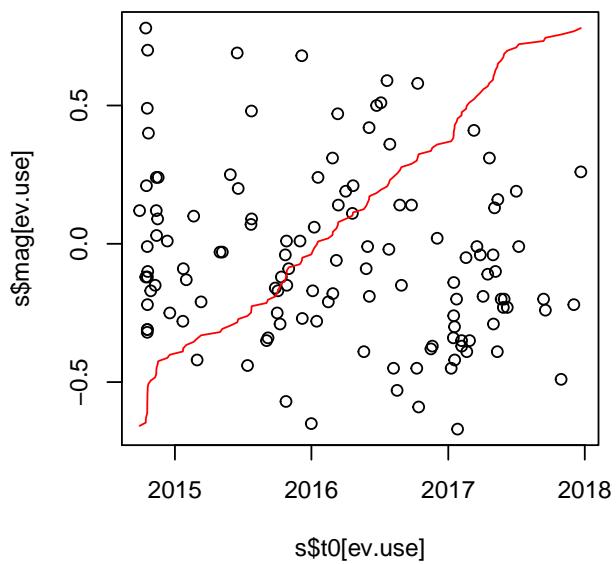
**BFMI = 1.1 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 672**



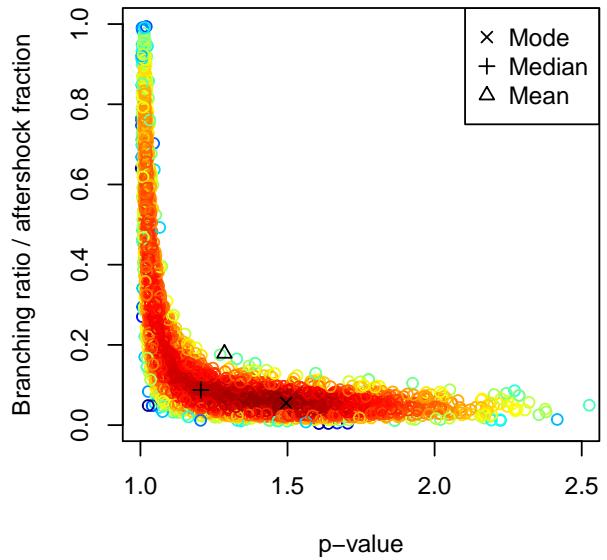
**Cv = 1.6**



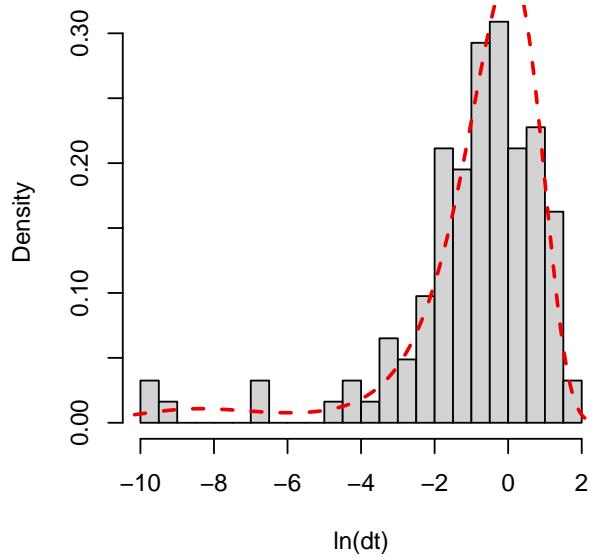
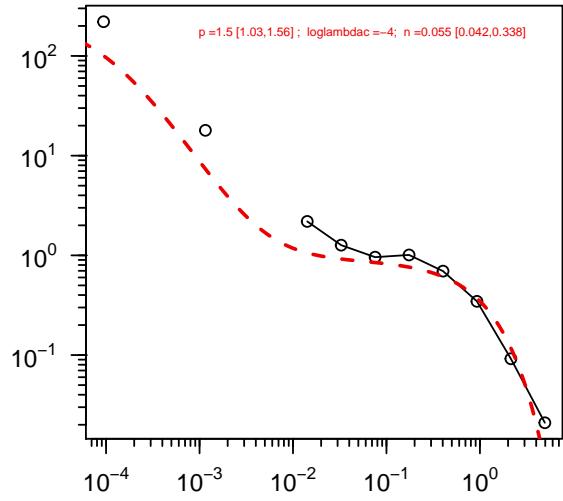
**Family 70040183 ; nev = 124**



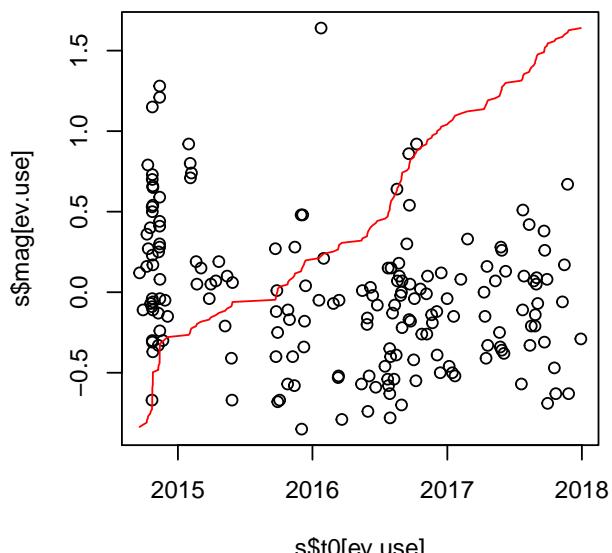
**BFMI = 0.86 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 545**



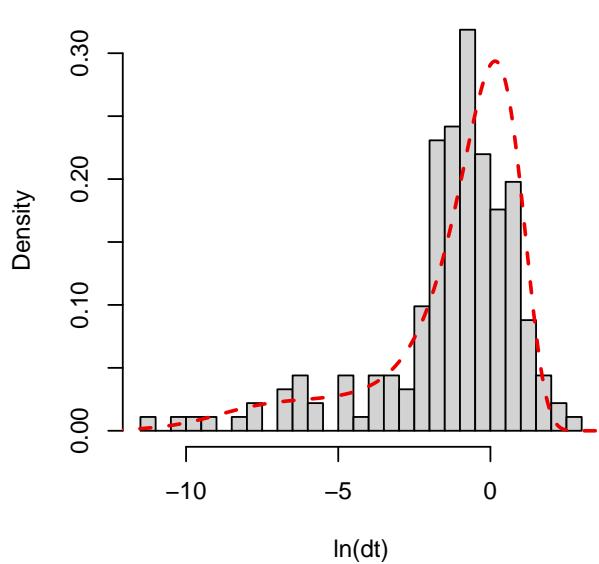
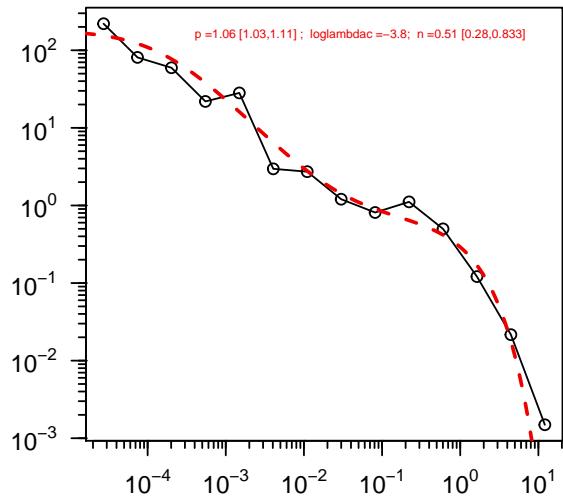
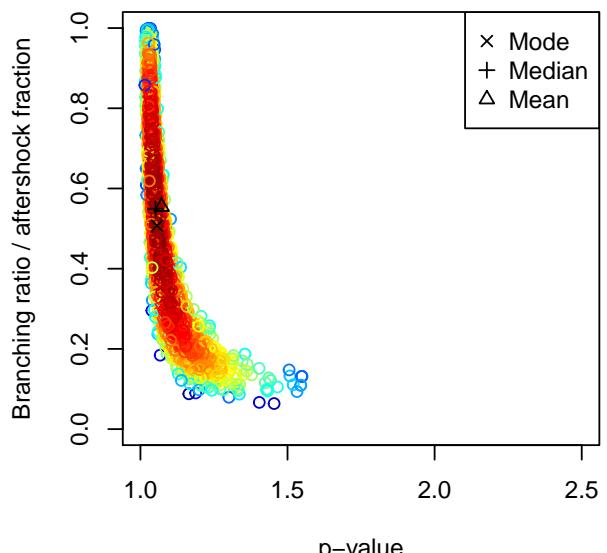
**Cv = 1.2**



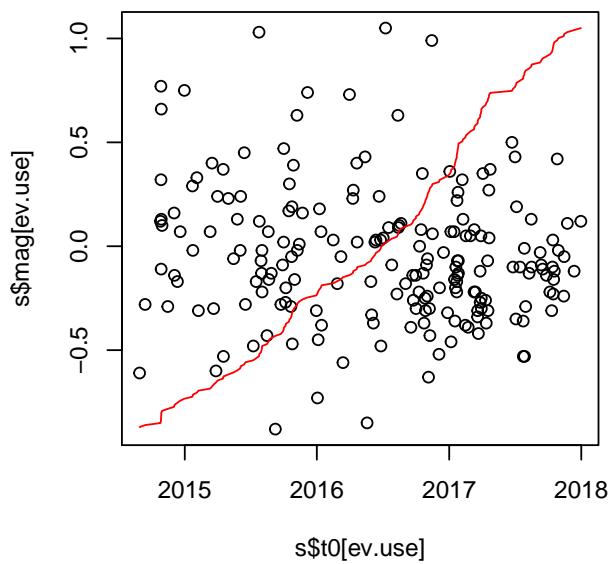
**Family 70041313 ; nev = 183**



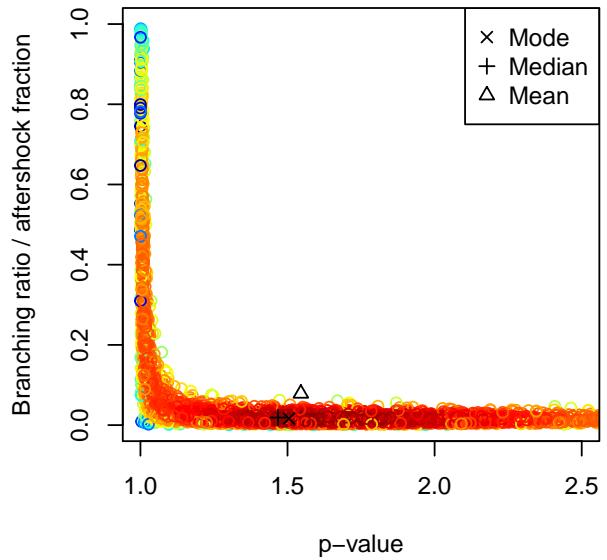
**BFMI = 1 ; n\_div = 0  
rhat = 1 ; n\_eff = 664**



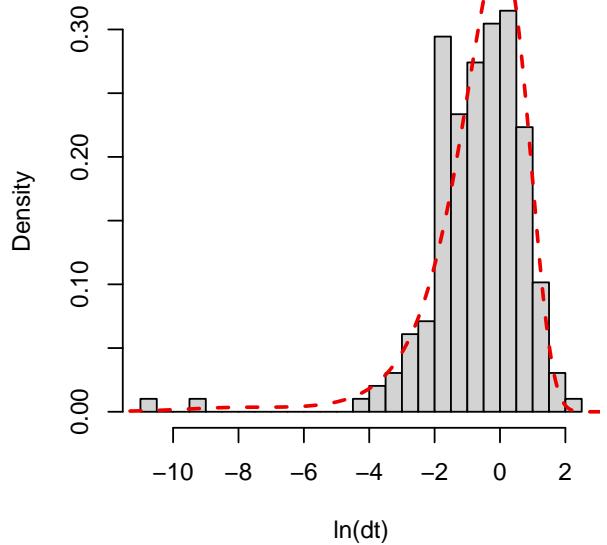
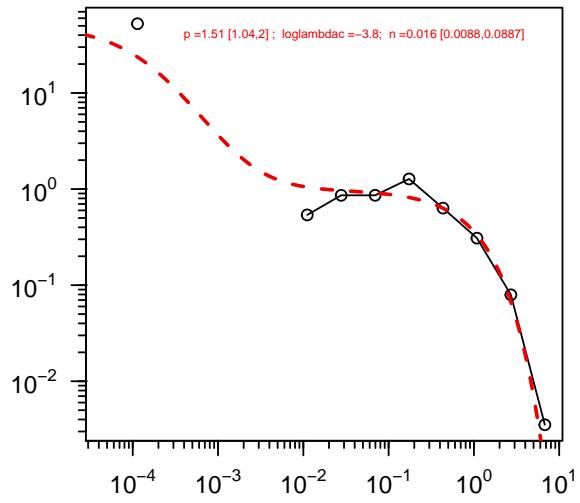
**Family 70042238 ; nev = 198**



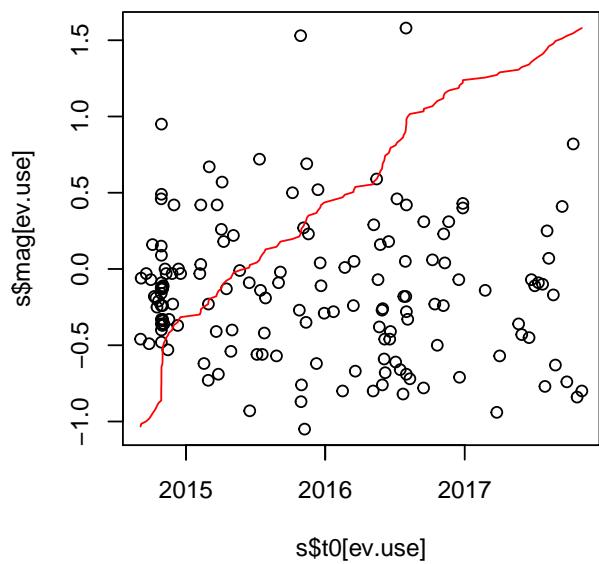
**BFMI = 1.1 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 375**



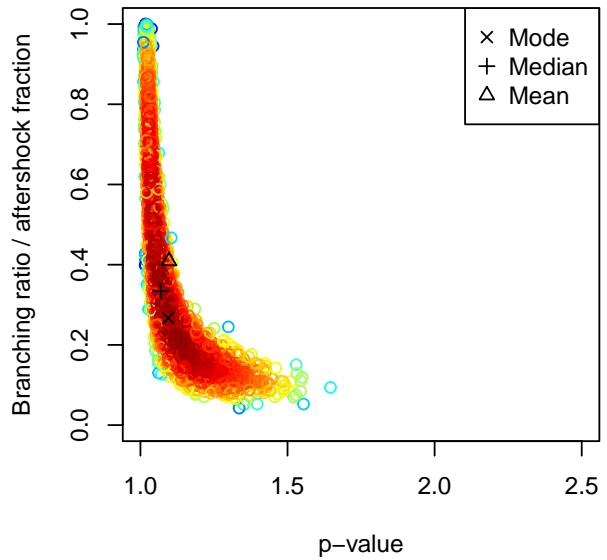
**Cv = 1.2**



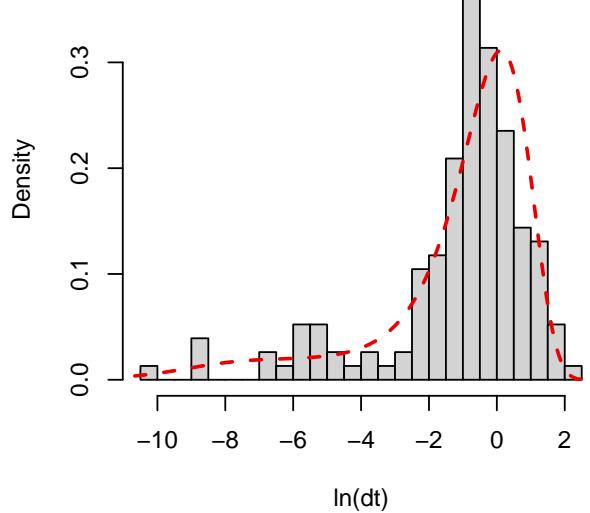
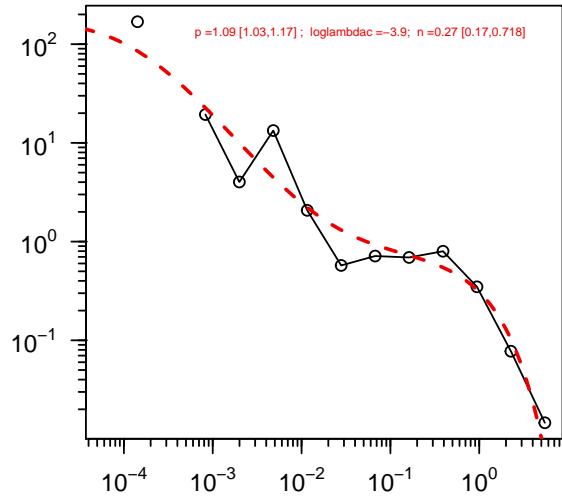
**Family 70042303 ; nev = 154**



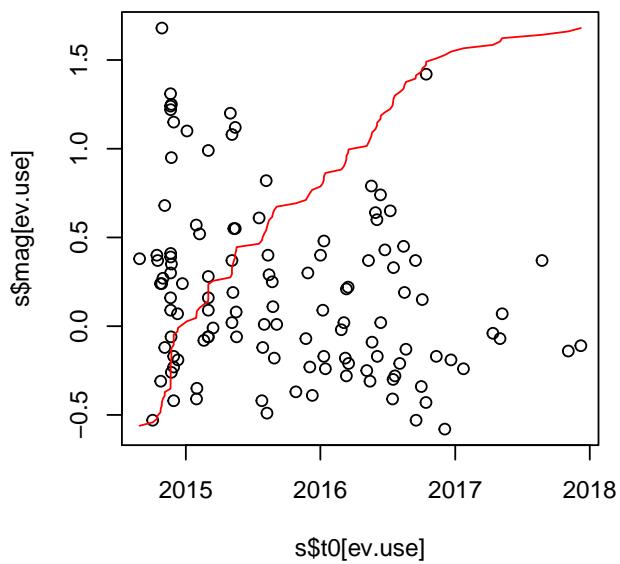
**BFMI = 1.1 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 764**



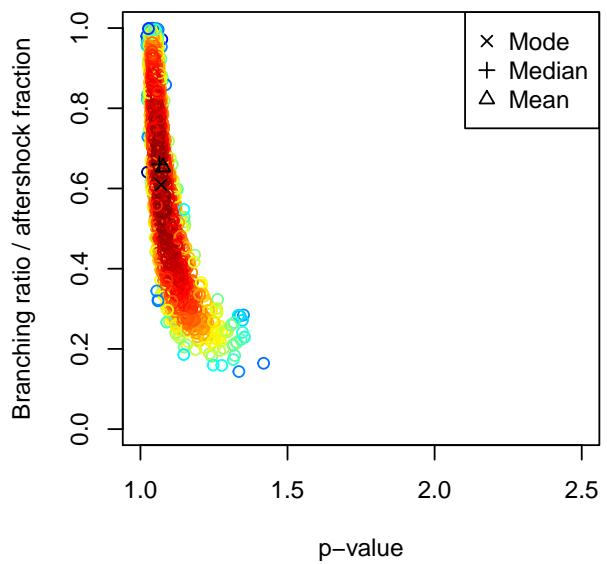
**Cv = 1.4**



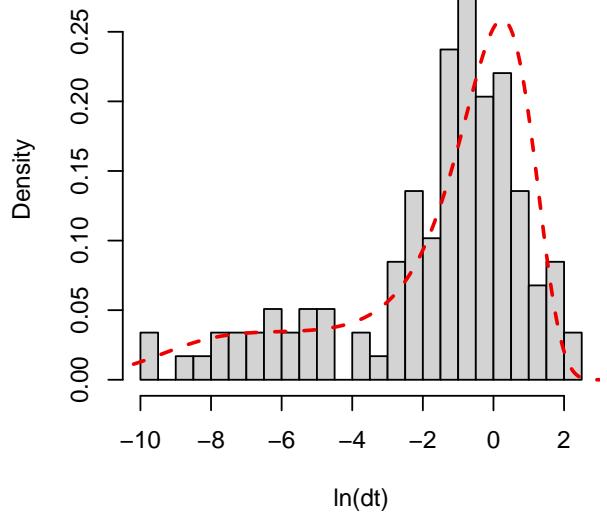
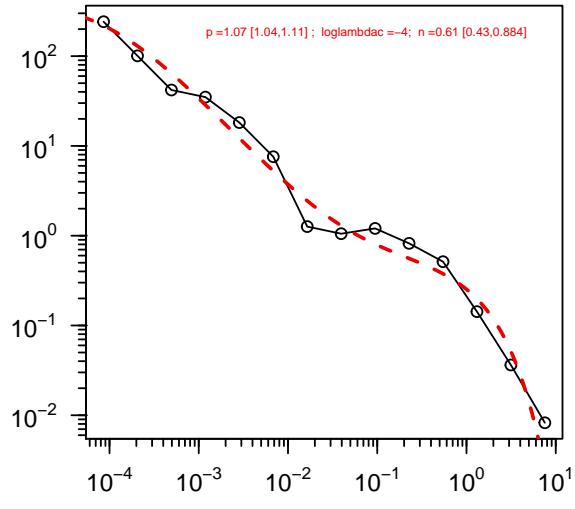
**Family 70042433 ; nev = 119**



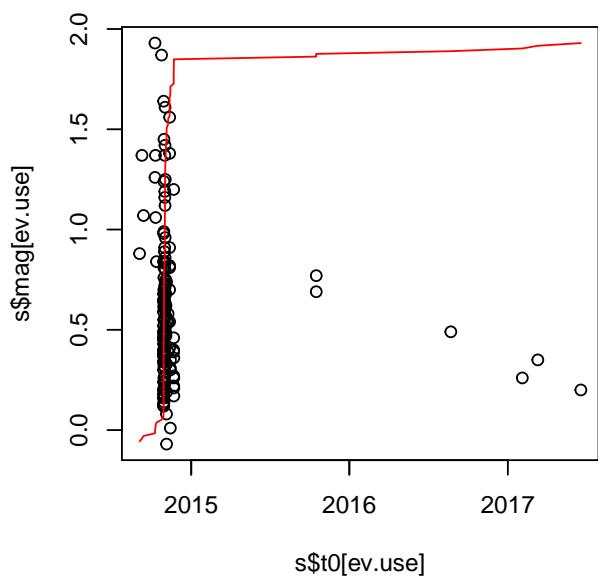
**BFMI = 0.97 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 324**



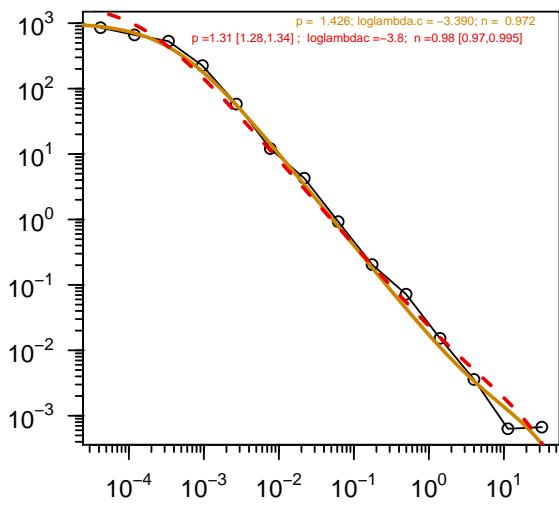
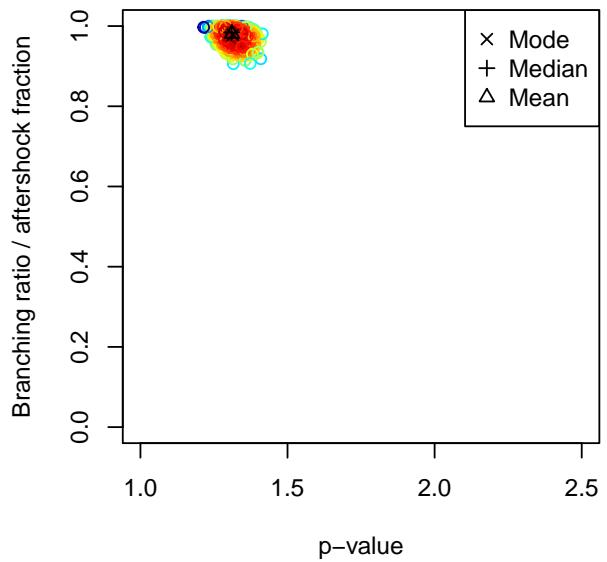
**Cv = 1.7**



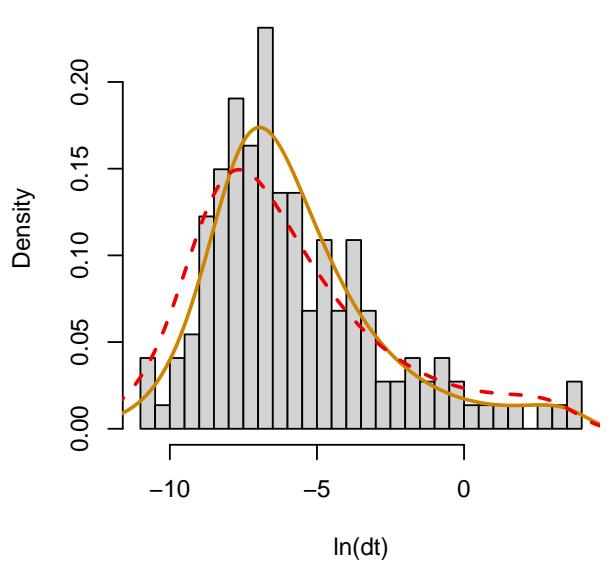
**Family 70042683 ; nev = 148**



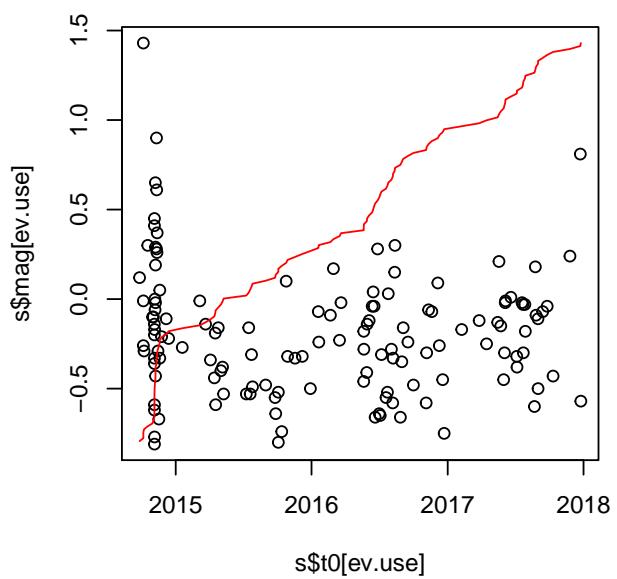
**BFMI = 1.2 ; n\_div = 0  
rhat = 1 ; n\_eff = 1133**



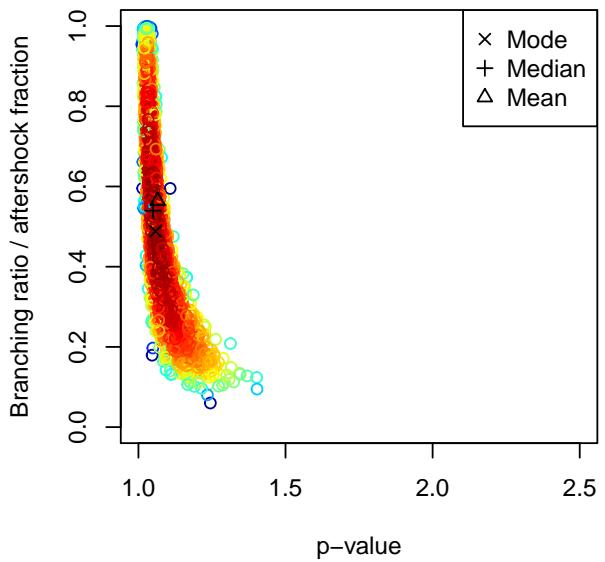
**Cv = 5.8**



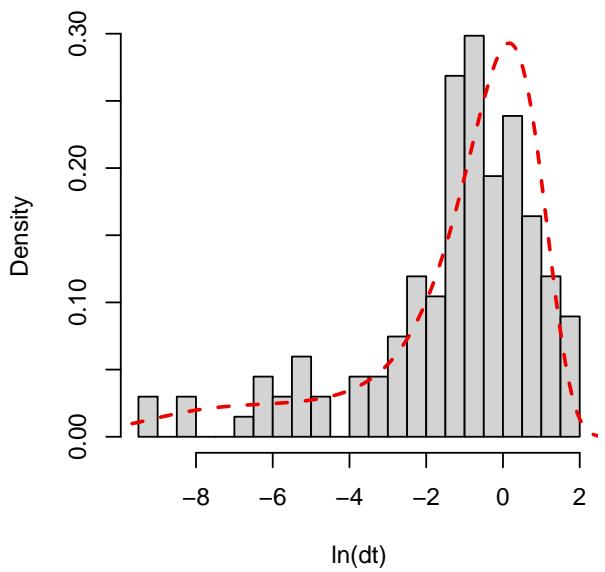
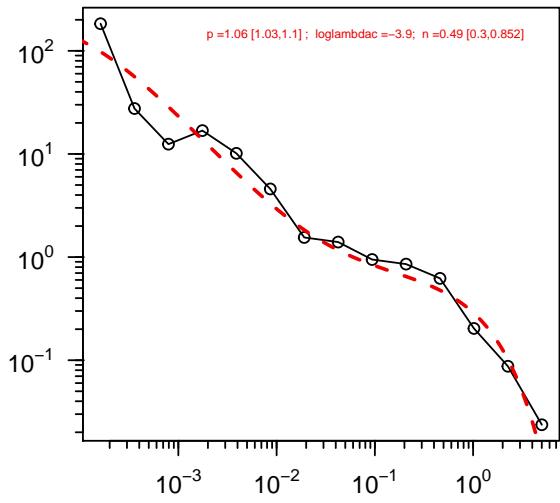
**Family 70042993 ; nev = 135**



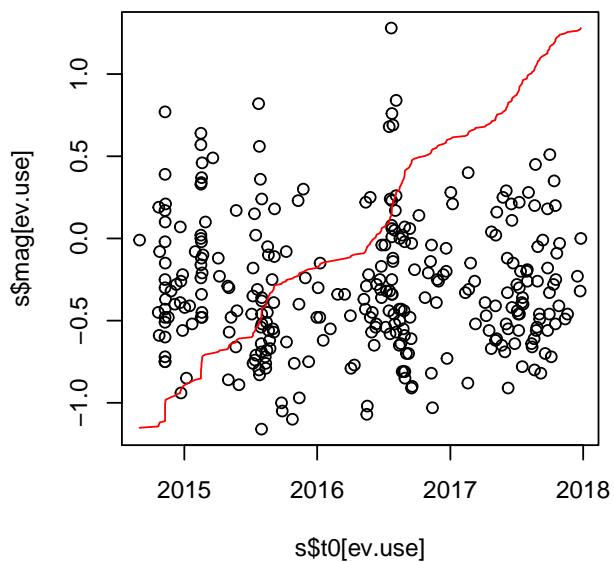
**BFMI = 0.95 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 664**



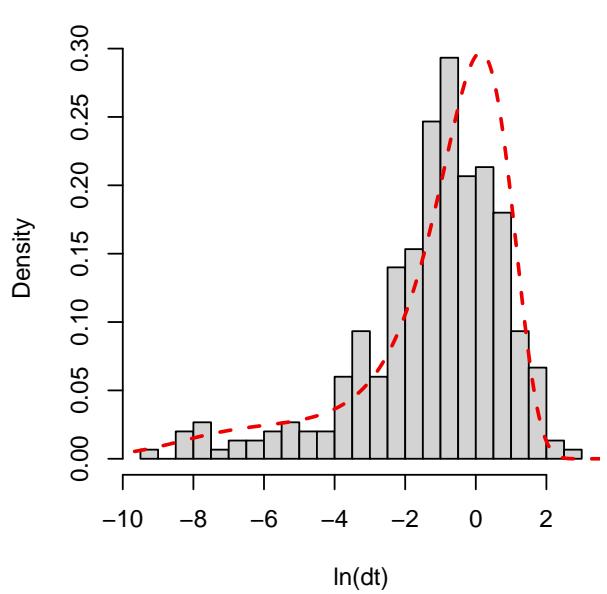
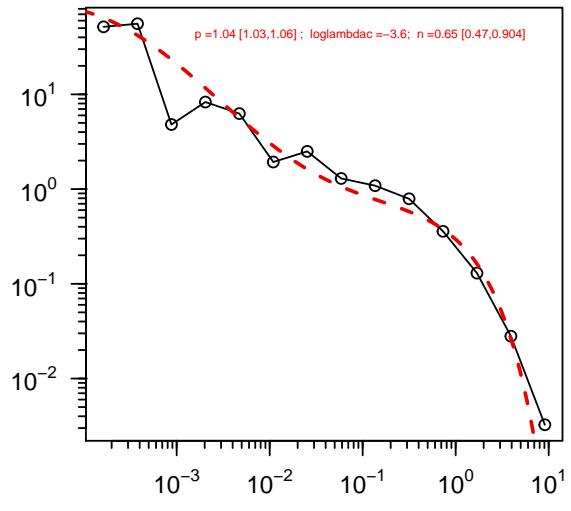
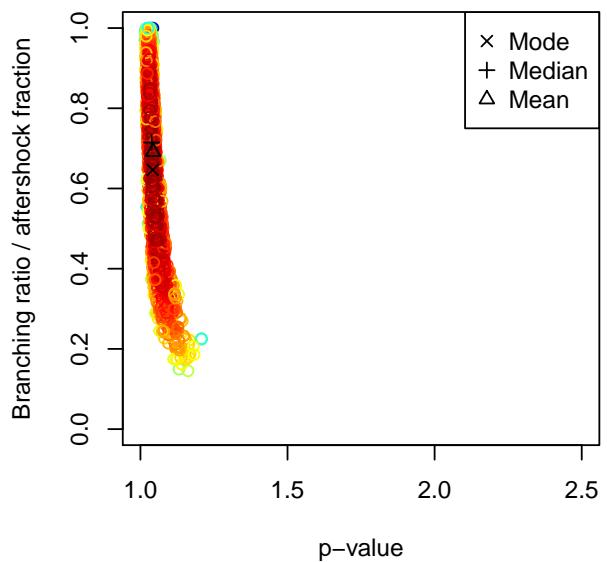
**Cv = 1.4**



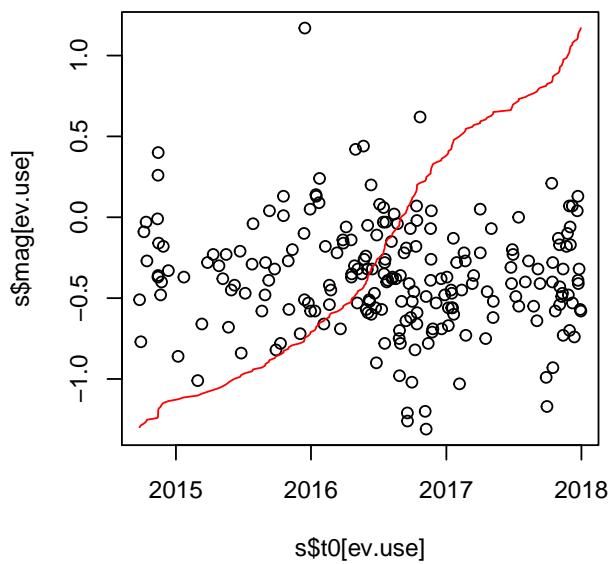
**Family 70043748 ; nev = 301**



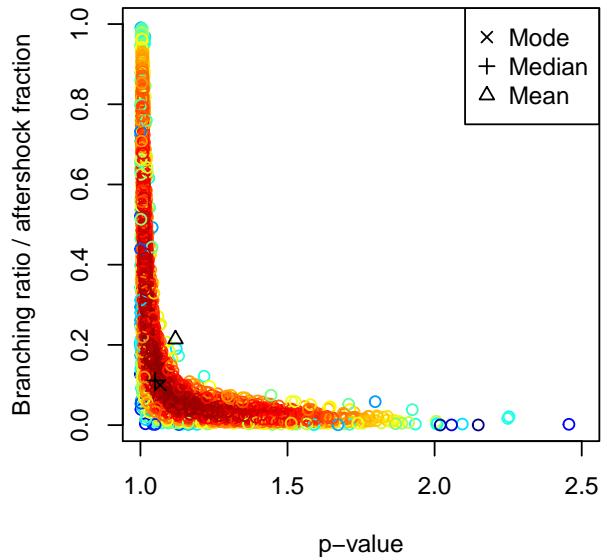
**BFMI = 1 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 512**



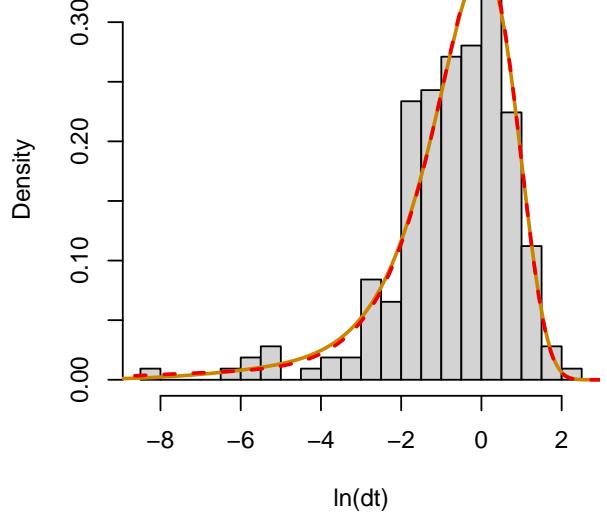
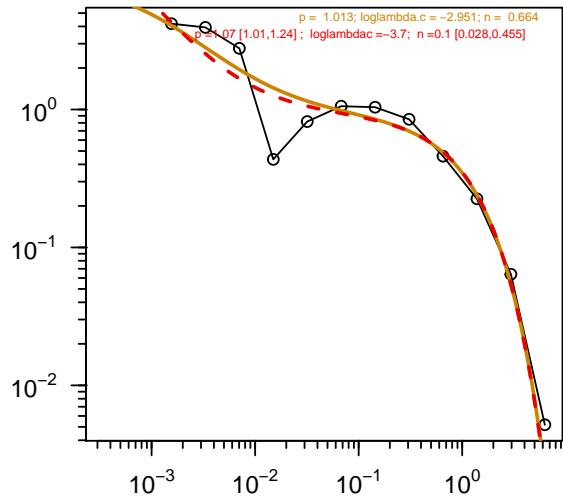
**Family 70047673 ; nev = 215**



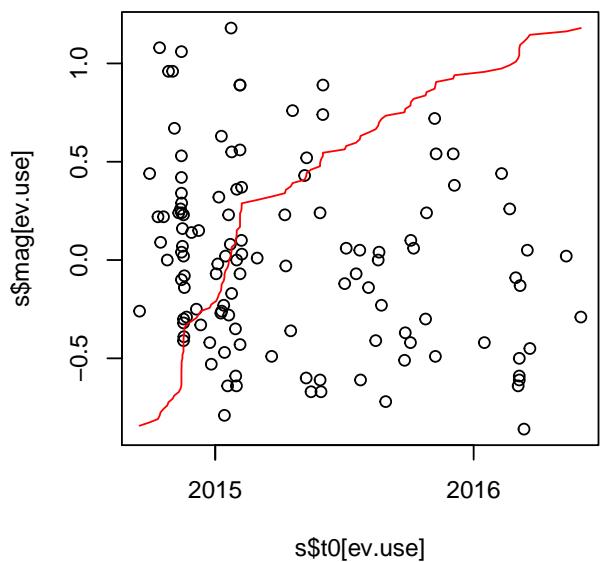
**BFMI = 0.98 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 687**



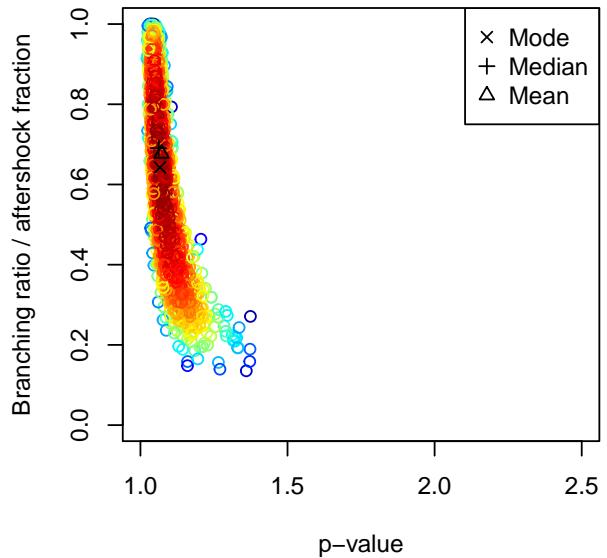
**Cv = 1.2**



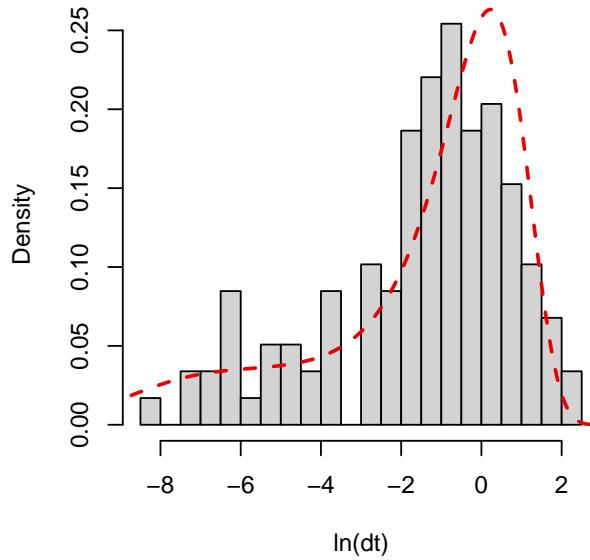
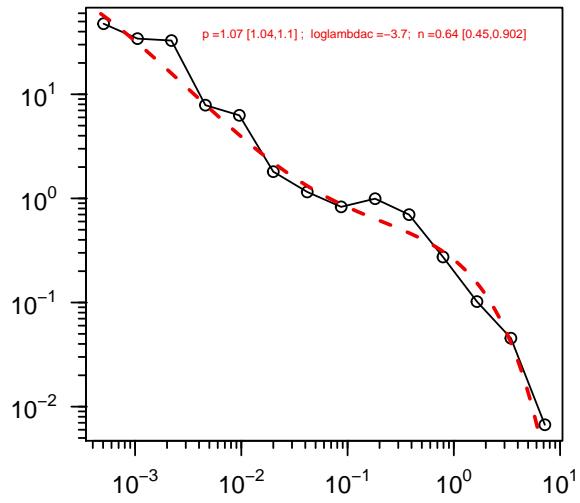
**Family 70048013 ; nev = 119**



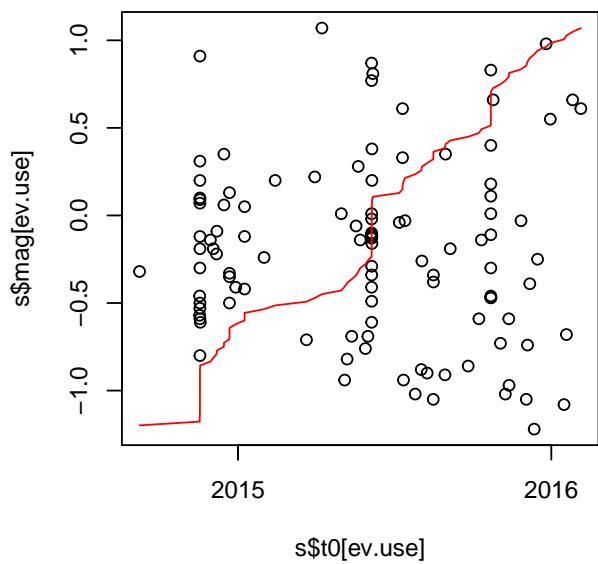
**BFMI = 0.96 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 450**



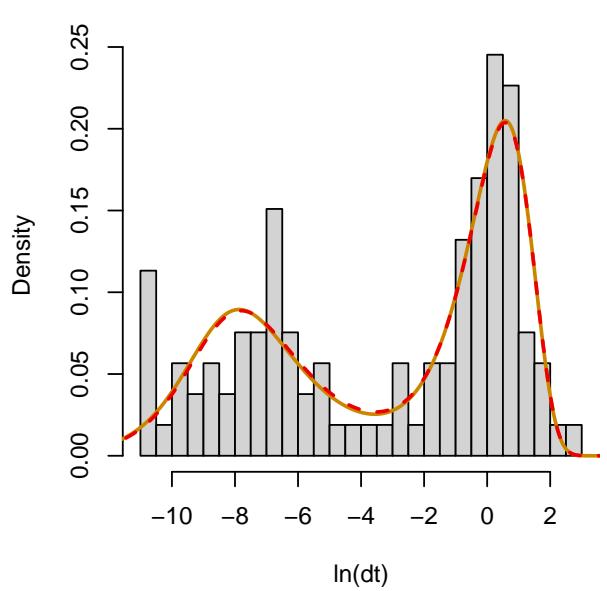
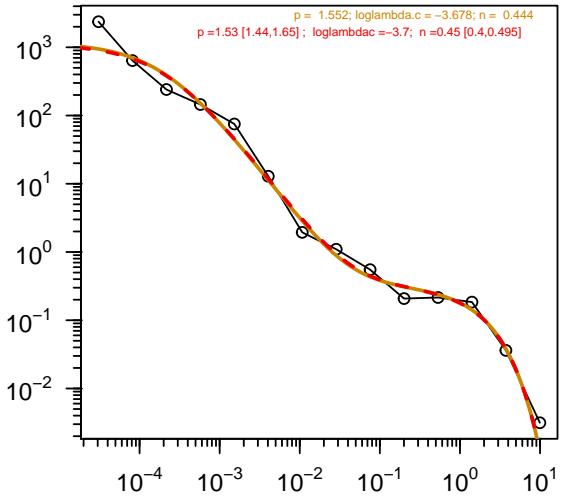
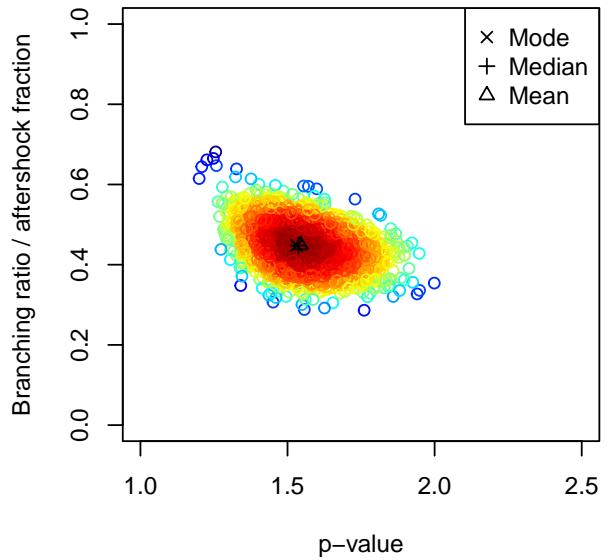
**Cv = 1.6**



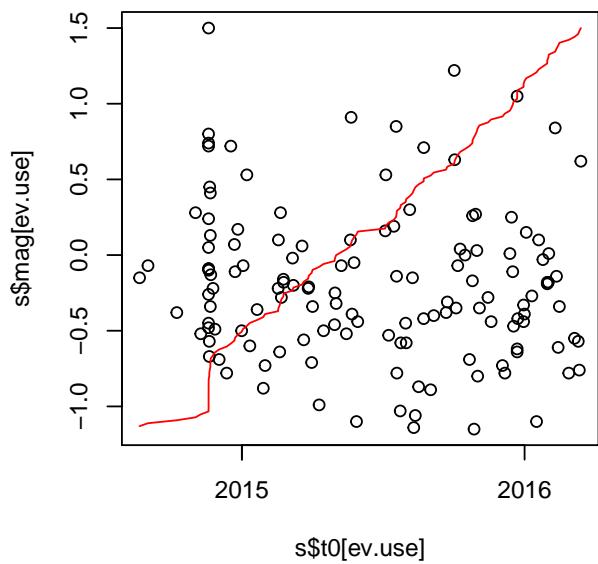
**Family 70048743 ; nev = 107**



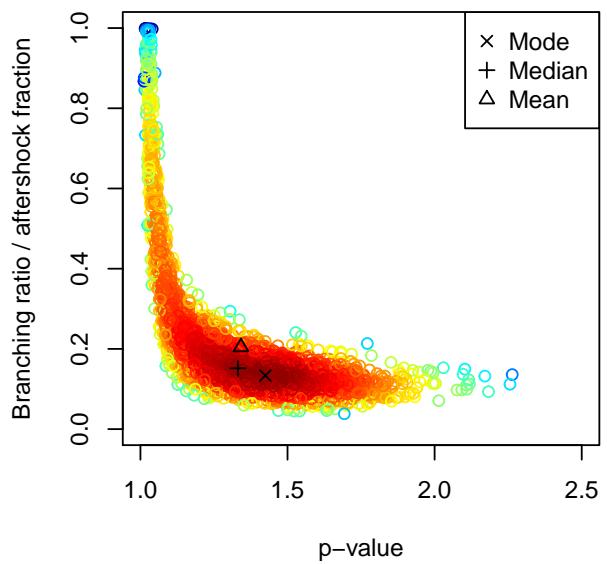
**BFMI = 0.97 ; n\_div = 0  
rhat = 1 ; n\_eff = 1724**



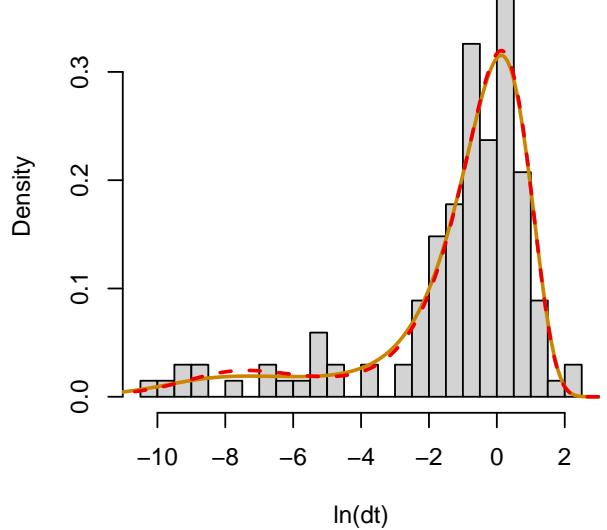
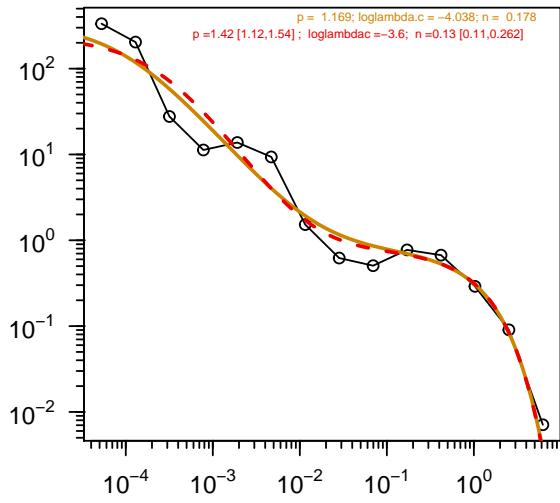
**Family 70049238 ; nev = 136**



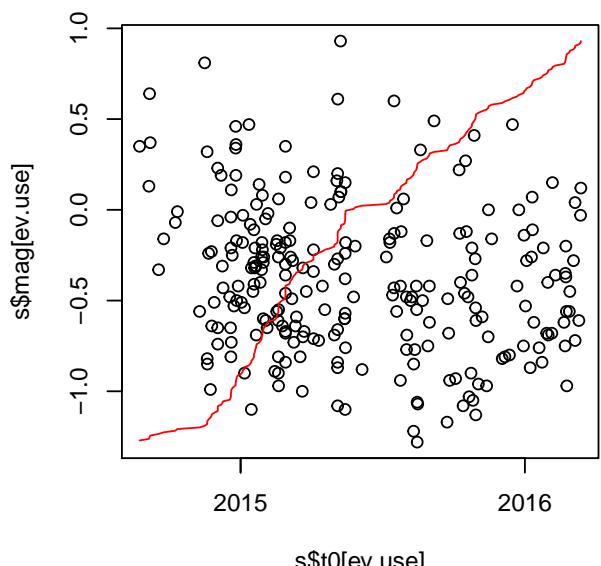
**BFMI = 1.1 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 355**



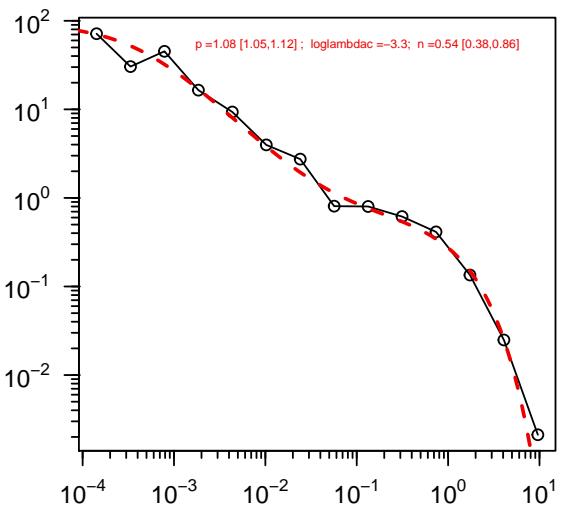
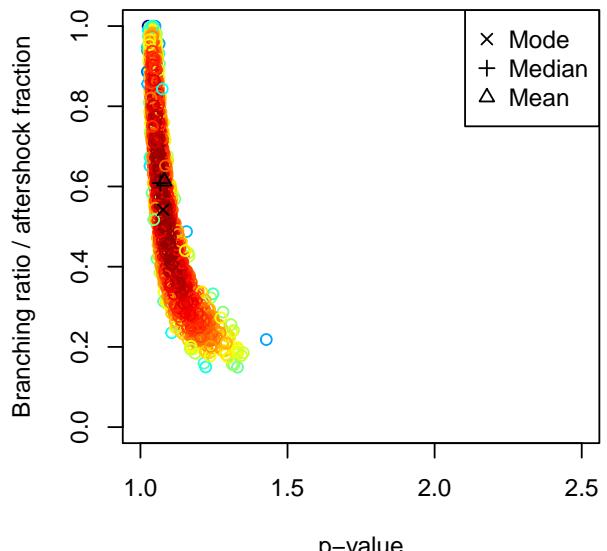
**Cv = 1.3**



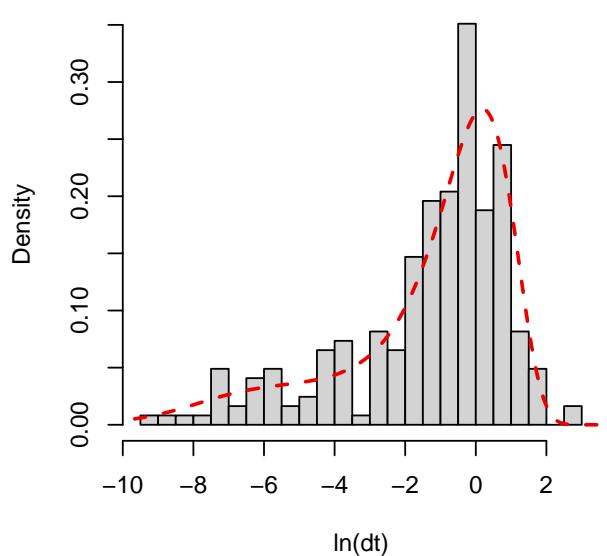
**Family 70049268 ; nev = 246**



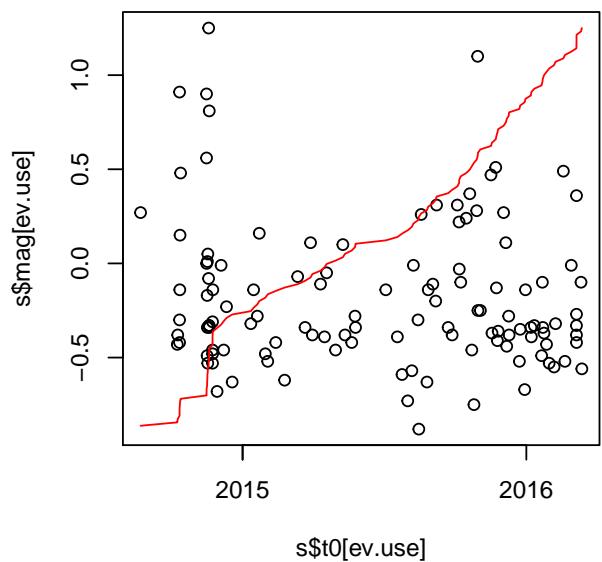
**BFMI = 1 ; n\_div = 0  
rhat = 1 ; n\_eff = 719**



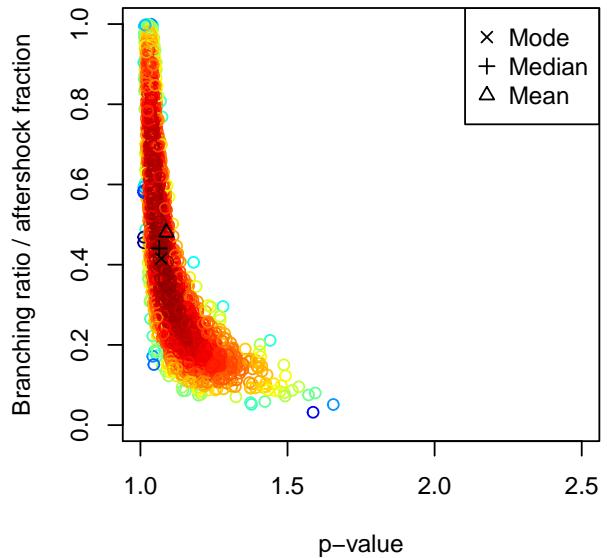
**Cv = 1.6**



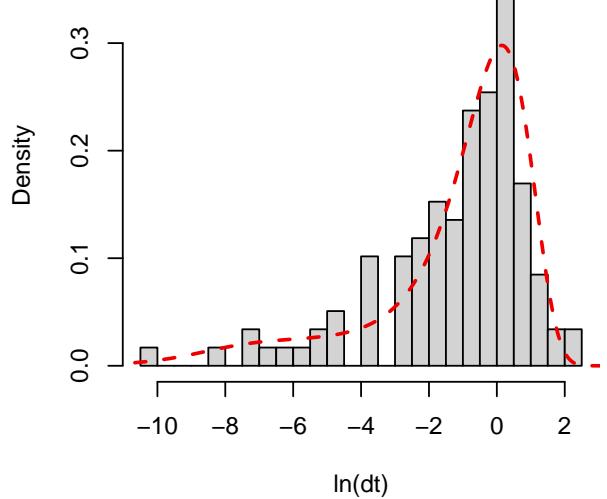
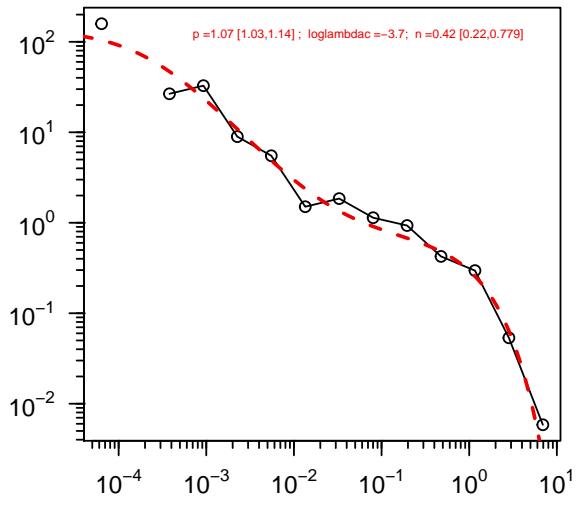
**Family 70049313 ; nev = 119**



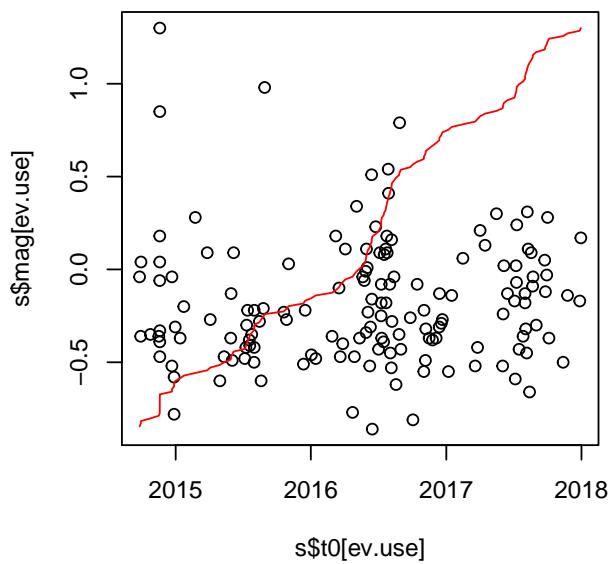
**BFMI = 0.97 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 816**



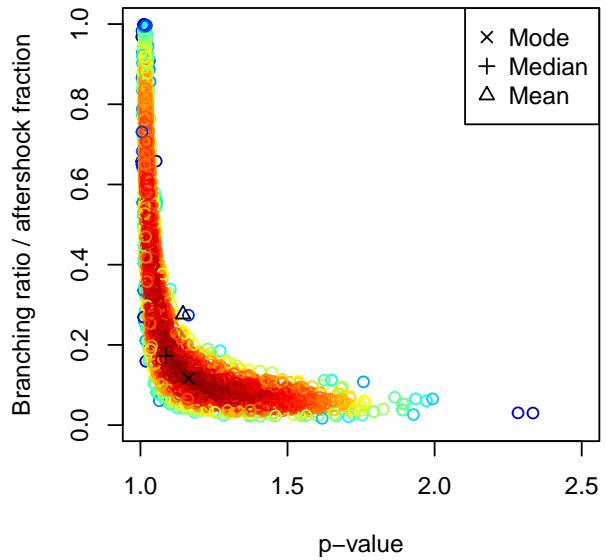
**Cv = 1.5**



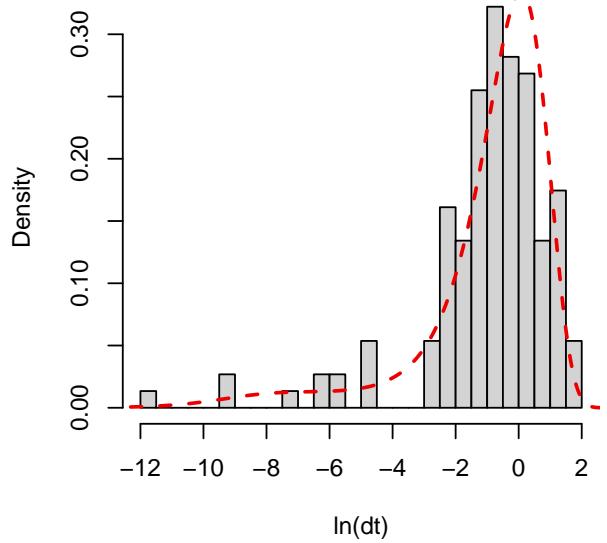
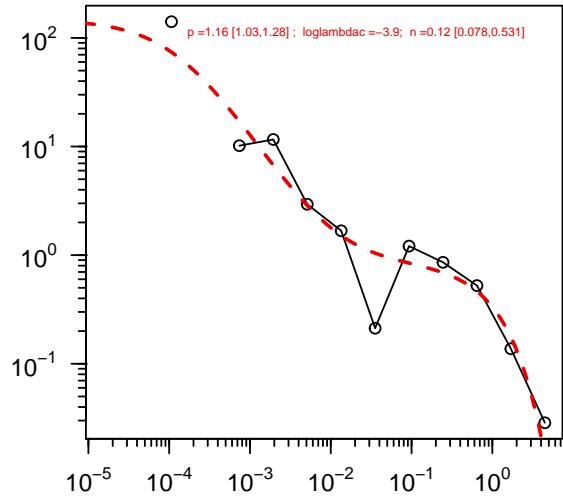
**Family 70049328 ; nev = 150**



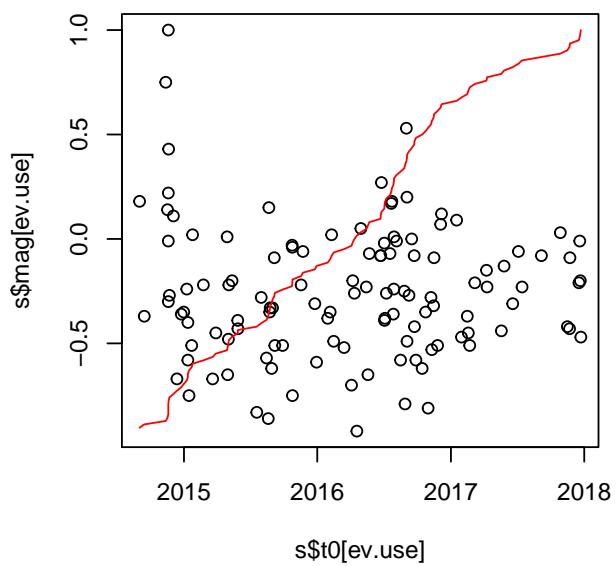
**BFMI = 0.93 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 695**



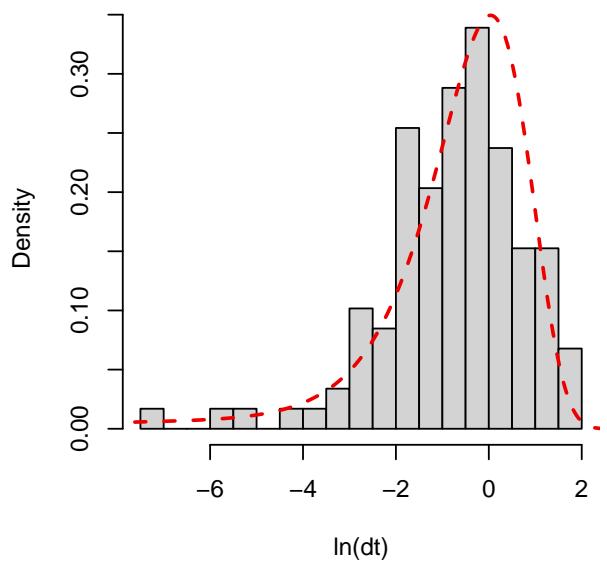
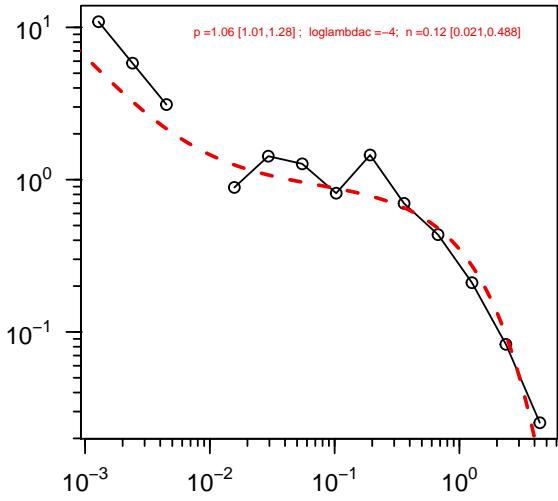
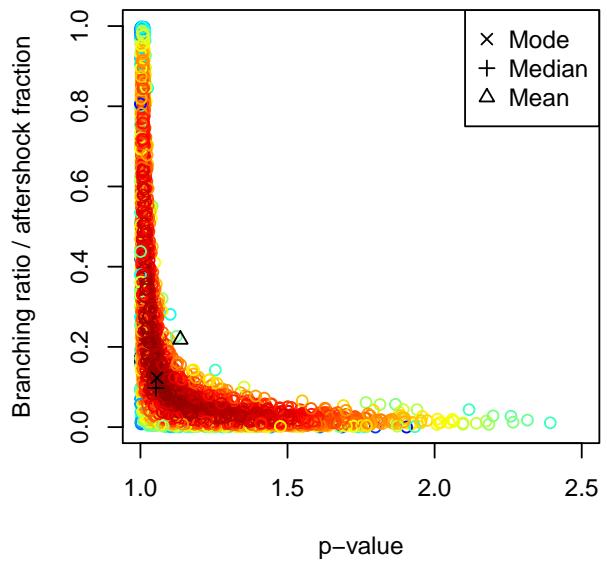
**Cv = 1.3**



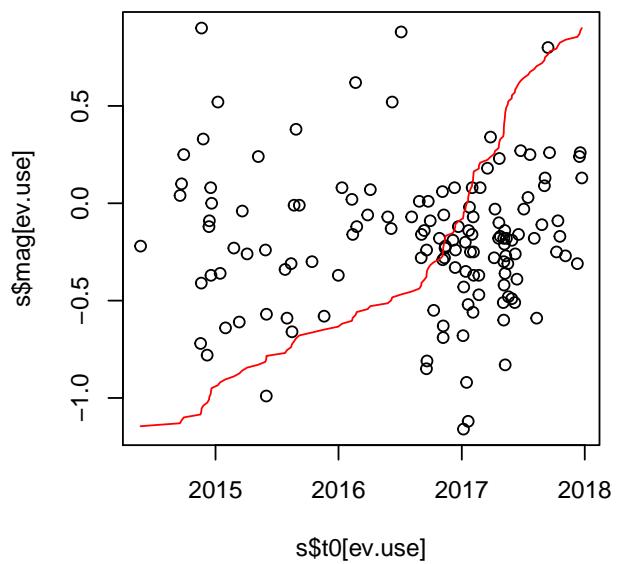
**Family 70049378 ; nev = 119**



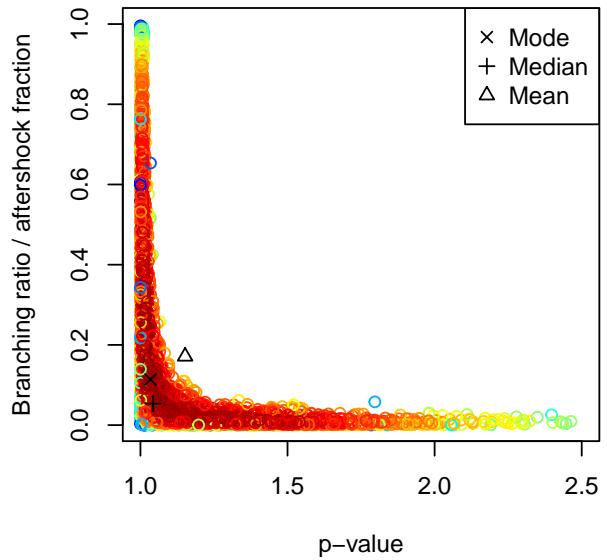
**BFMI = 1.1 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 754**



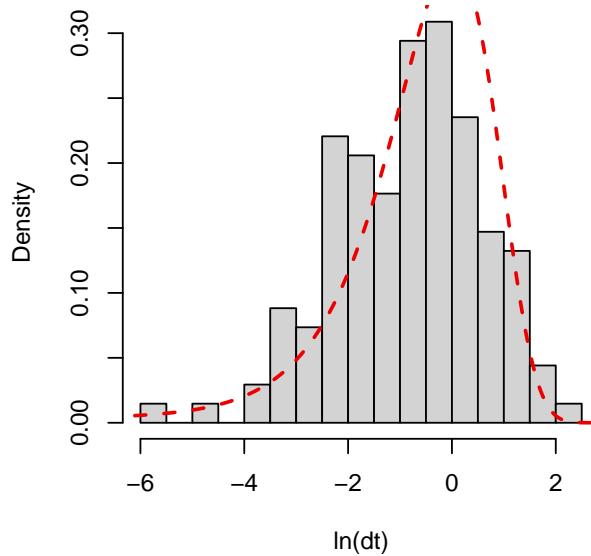
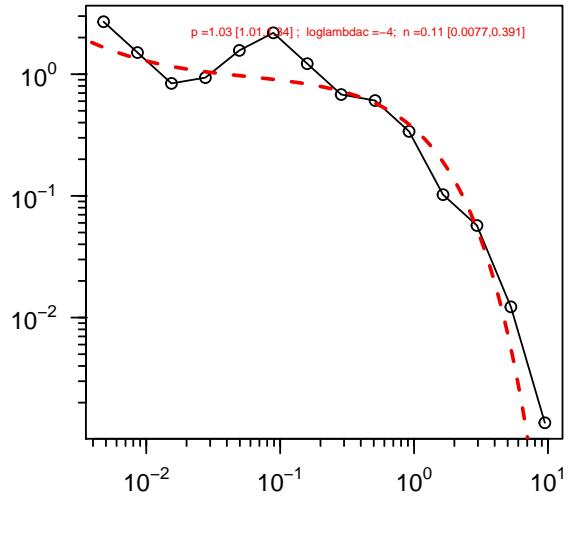
**Family 70049743 ; nev = 137**



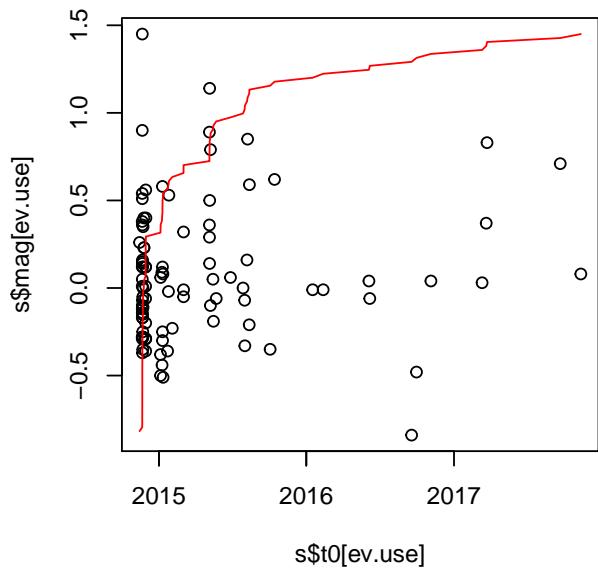
**BFMI = 1 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 789**



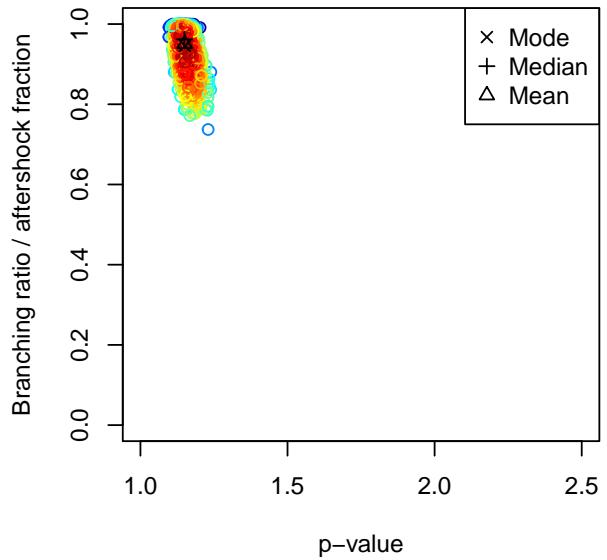
**Cv = 1.5**



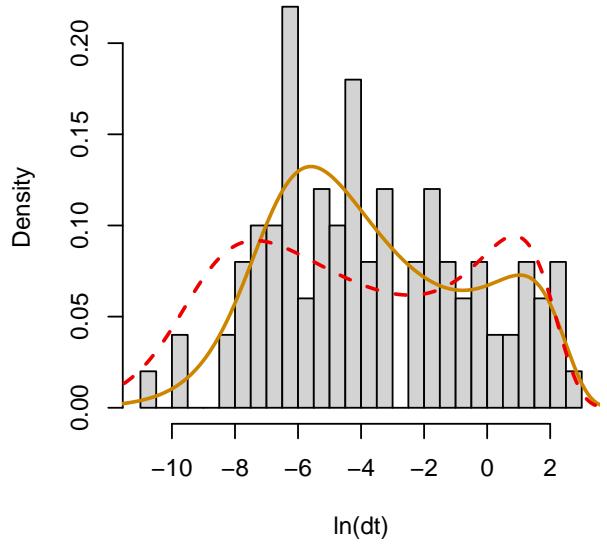
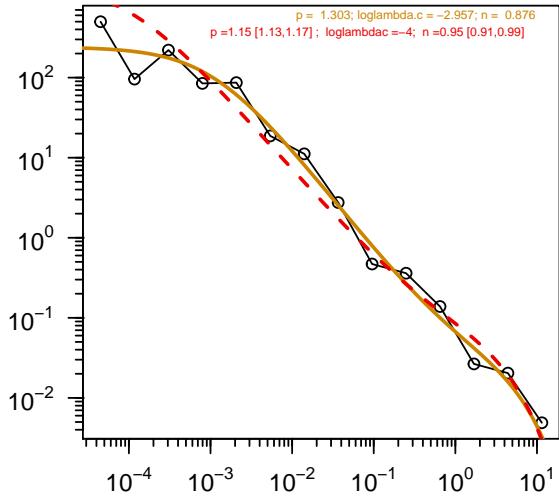
**Family 70049808 ; nev = 101**



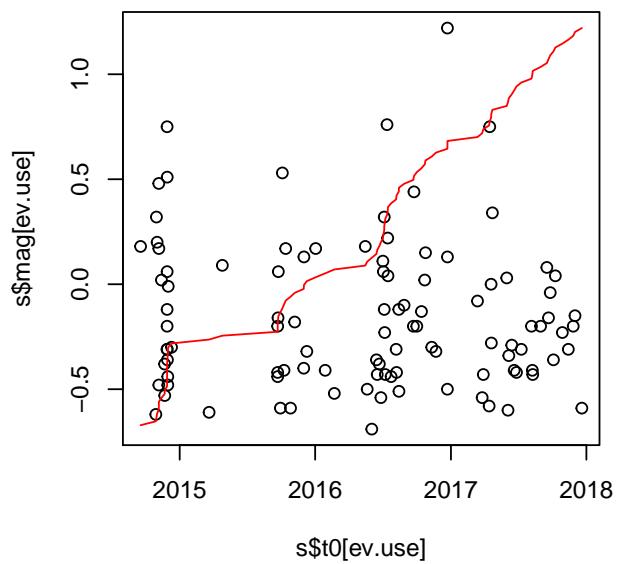
**BFMI = 1.2 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 1192**



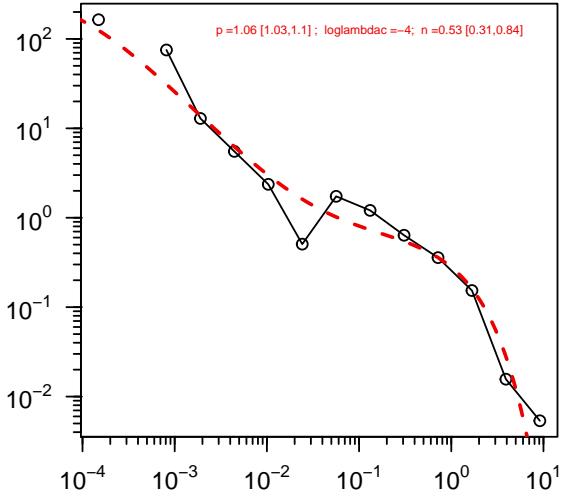
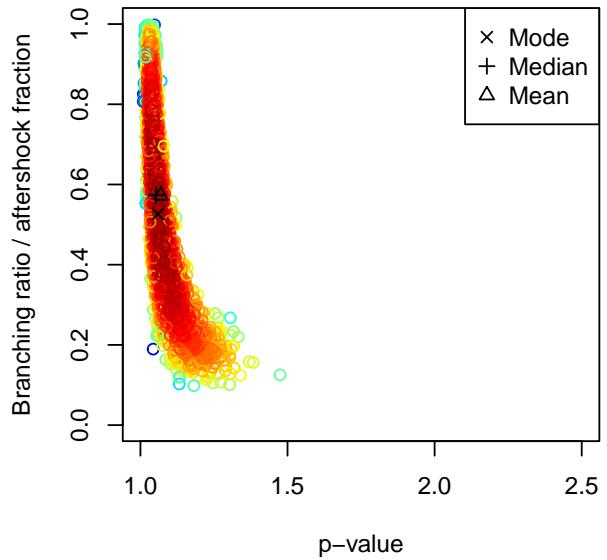
**Cv = 2.7**



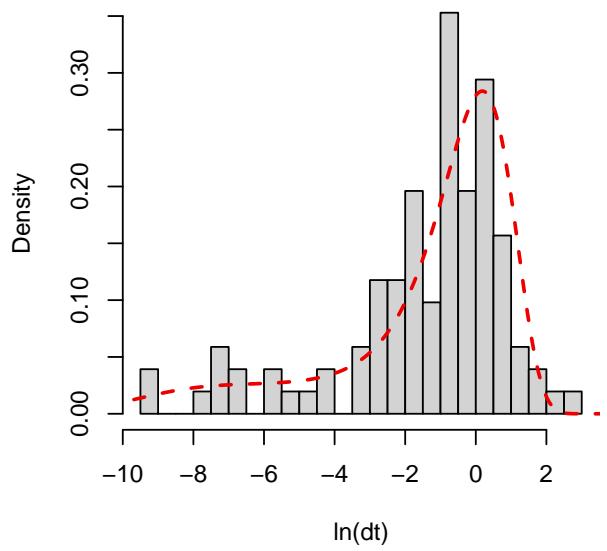
**Family 70053118 ; nev = 103**



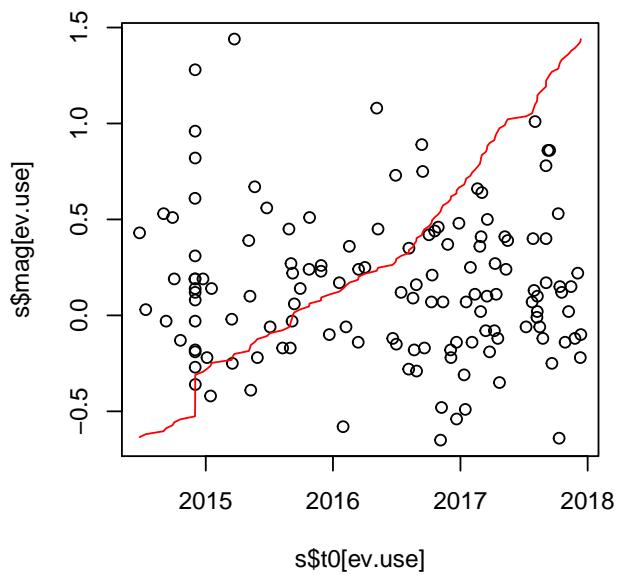
**BFMI = 1.1 ; n\_div = 0  
rhat = 1 ; n\_eff = 813**



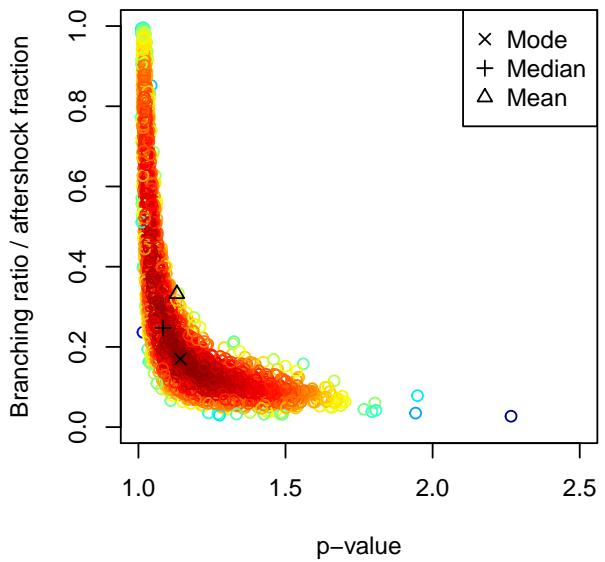
**Cv = 1.8**



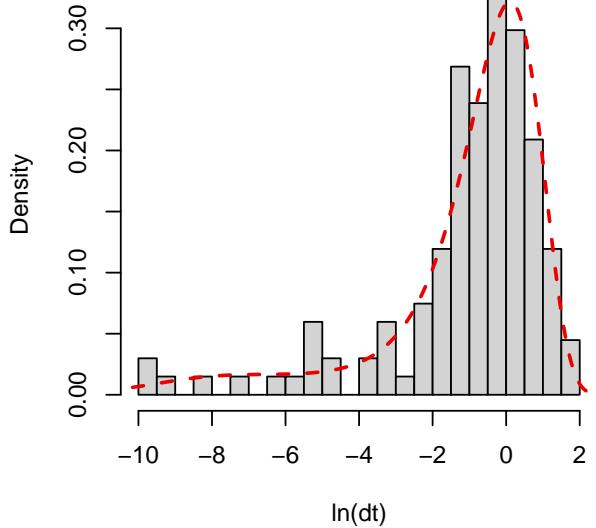
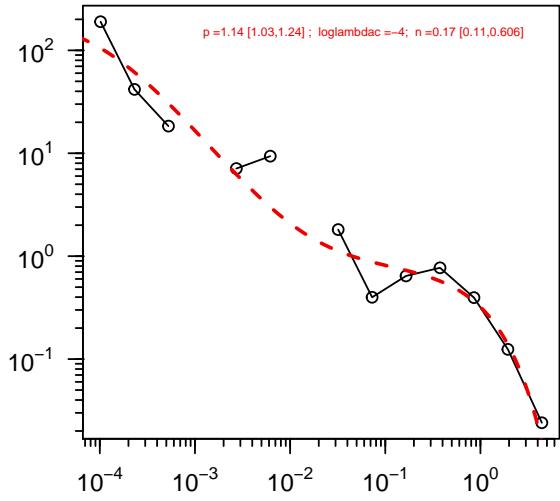
**Family 70054108 ; nev = 135**



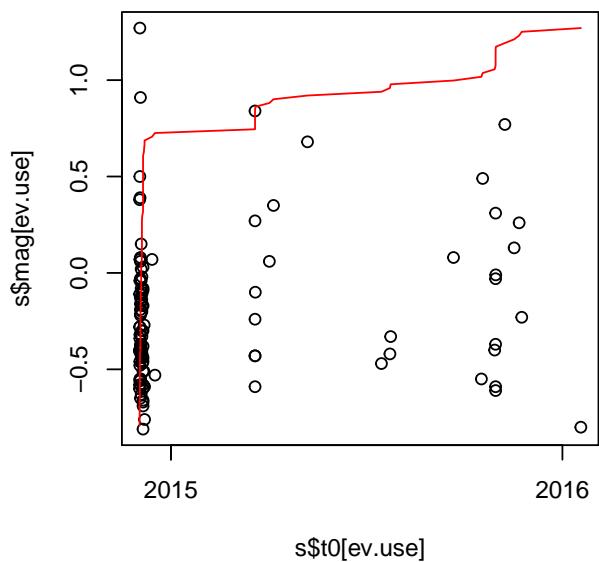
**BFMI = 1.2 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 687**



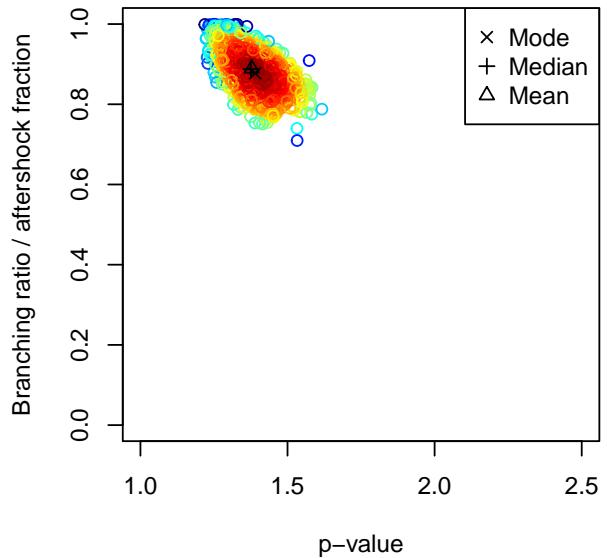
**Cv = 1.2**



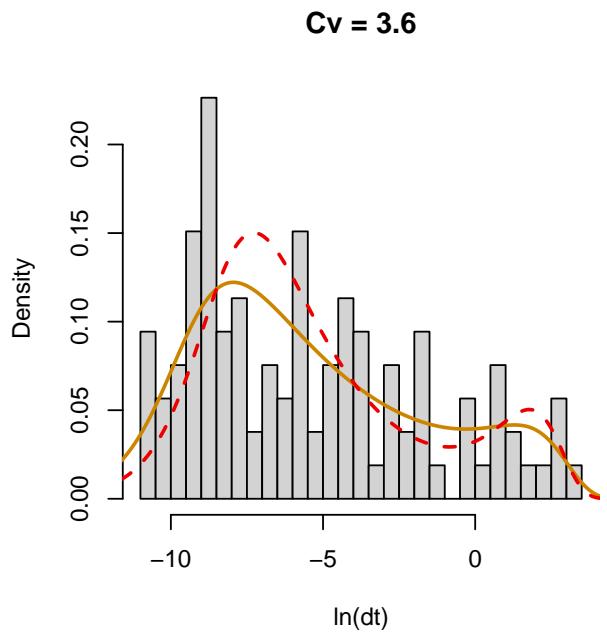
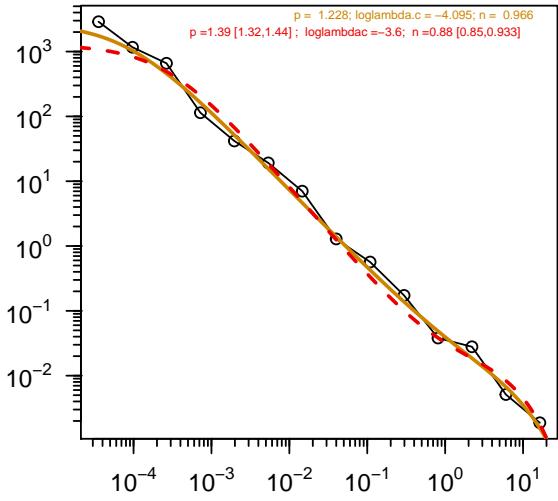
**Family 70054598 ; nev = 107**



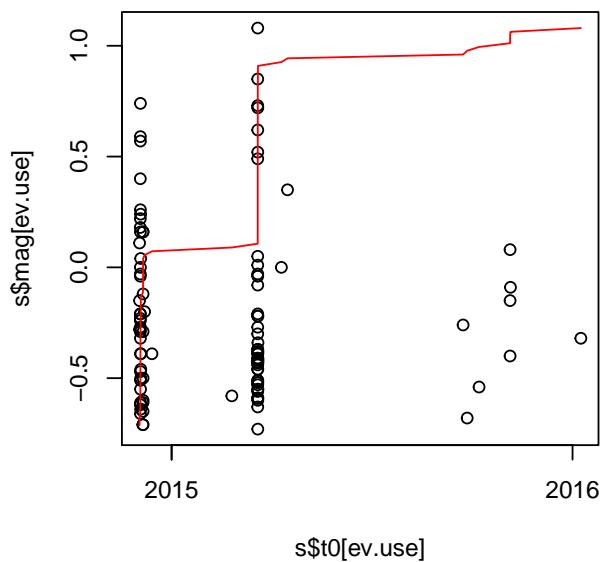
**BFMI = 1.3 ; n\_div = 0  
rhat = 1 ; n\_eff = 841**



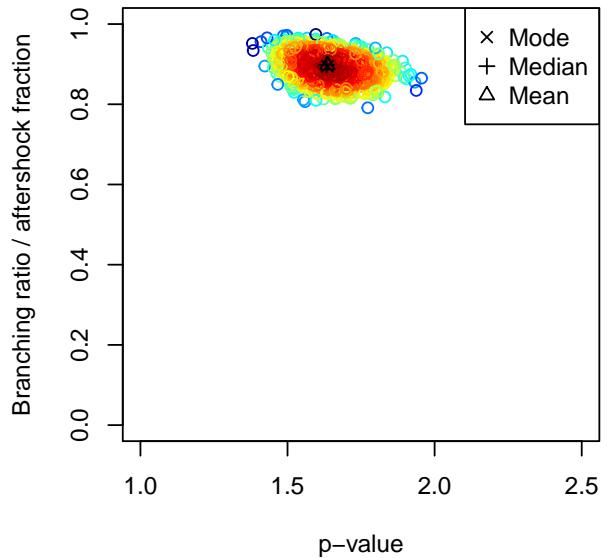
**Cv = 3.6**



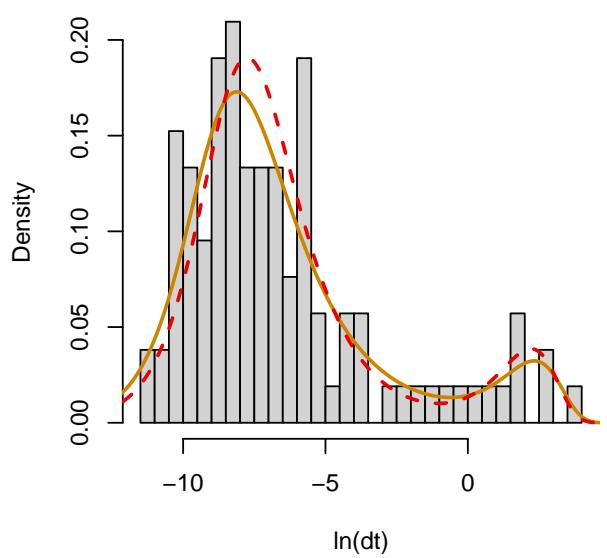
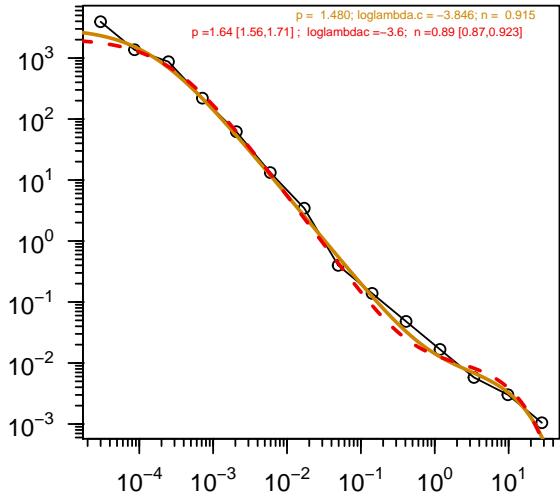
**Family 70054793 ; nev = 106**



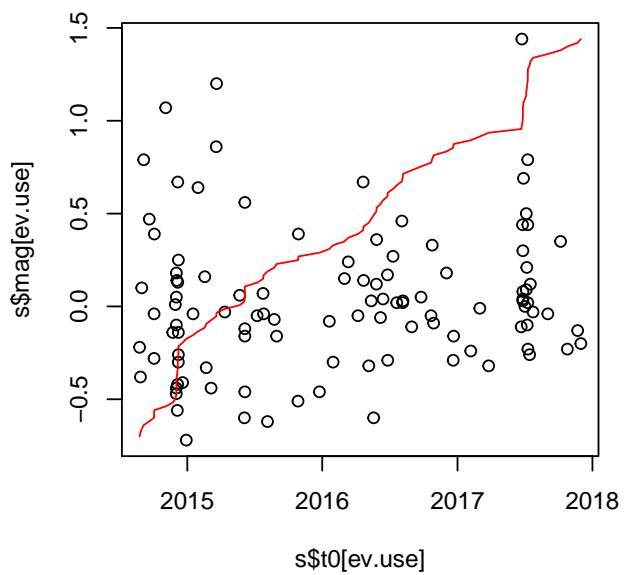
**BFMI = 1.1 ; n\_div = 0  
rhat = 1 ; n\_eff = 1682**



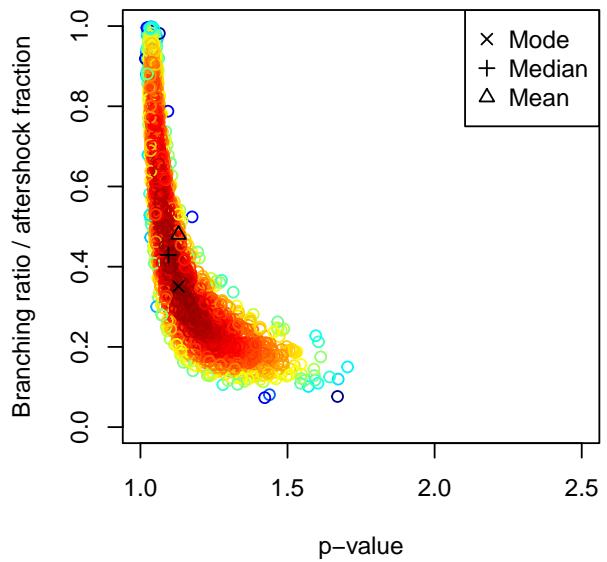
**Cv = 4.8**



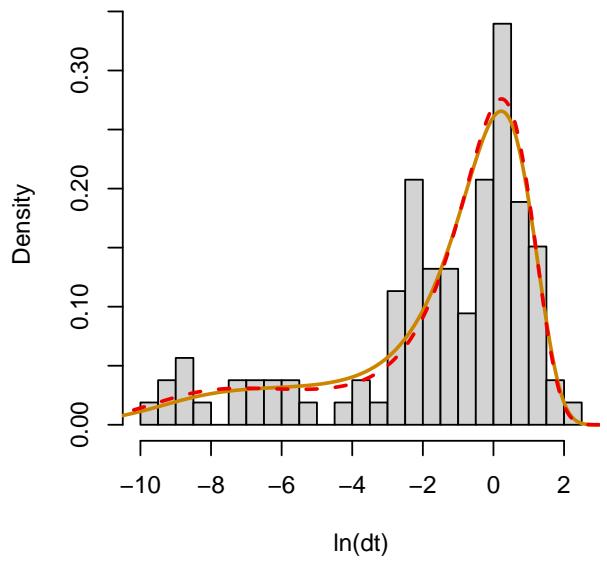
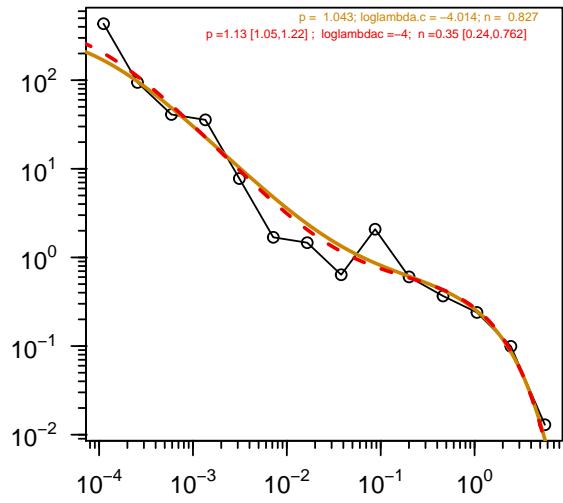
**Family 70055573 ; nev = 107**



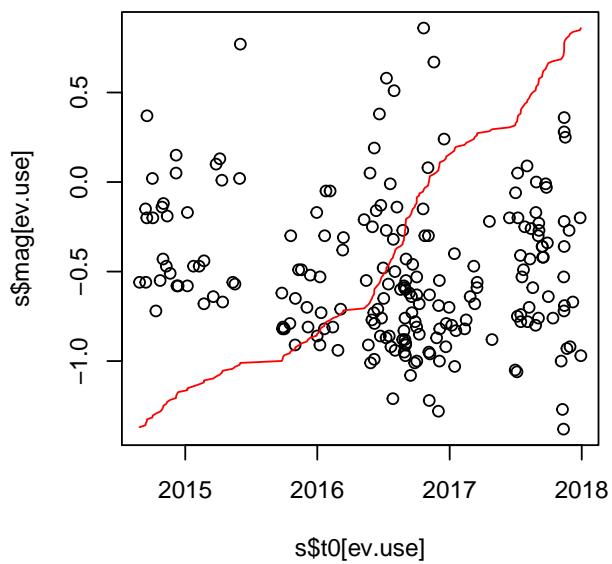
**BFMI = 1 ; n\_div = 0  
rhat = 1 ; n\_eff = 744**



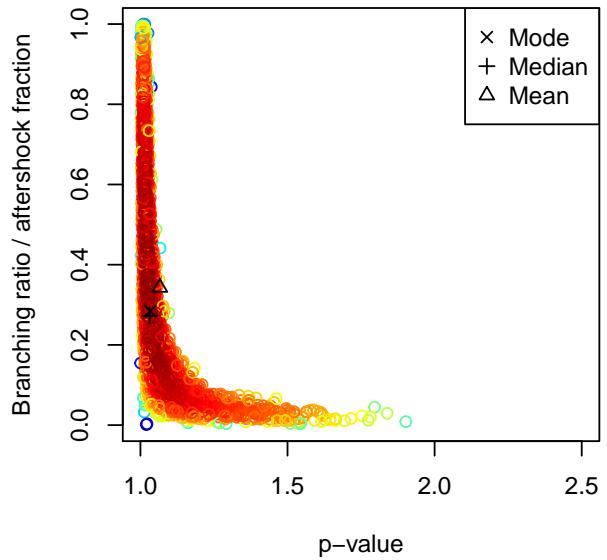
**Cv = 1.3**



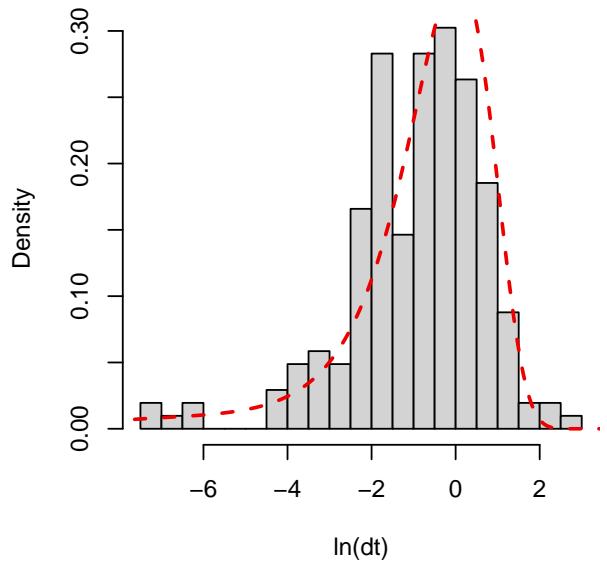
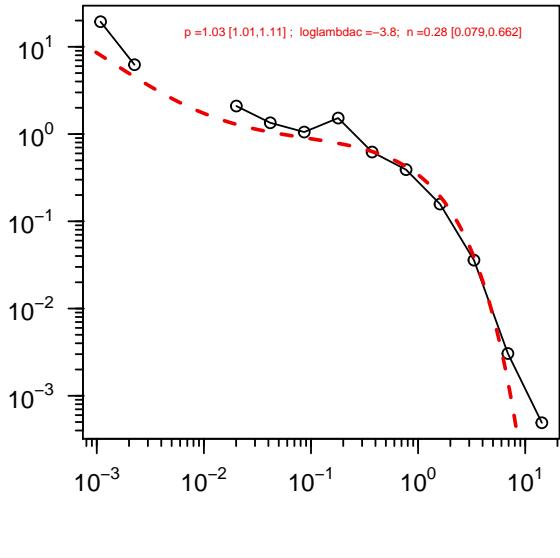
**Family 70055778 ; nev = 206**



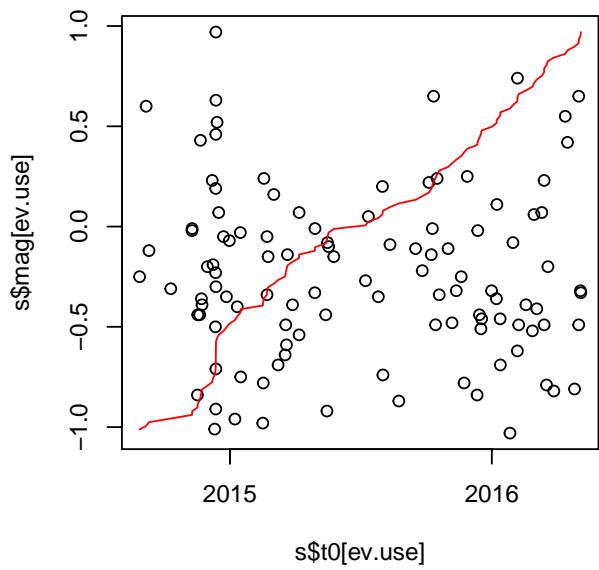
**BFMI = 1.1 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 721**



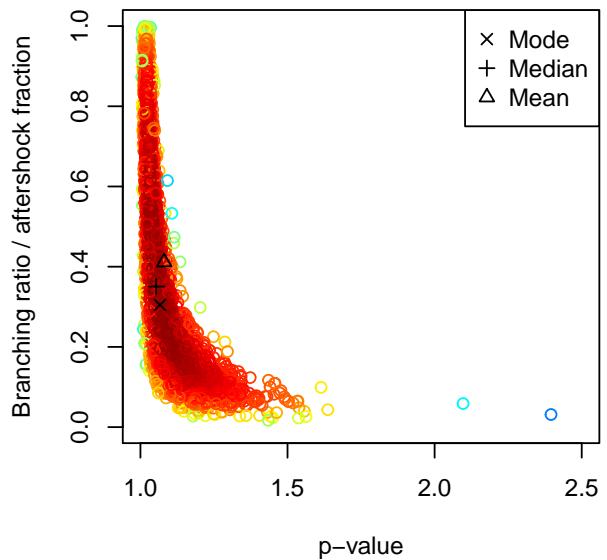
**Cv = 1.8**



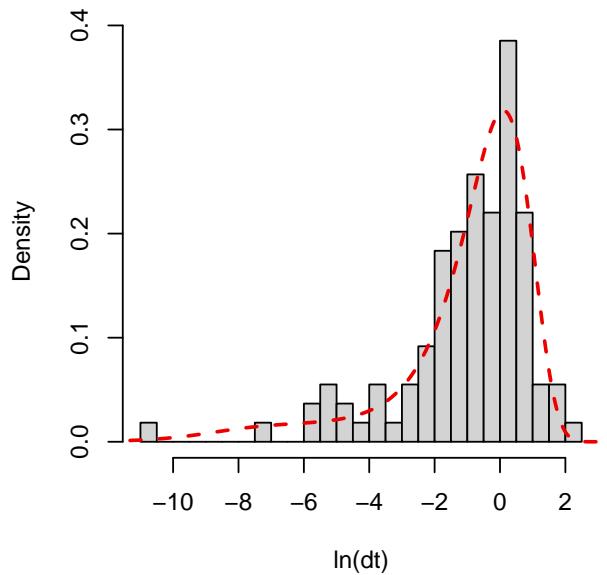
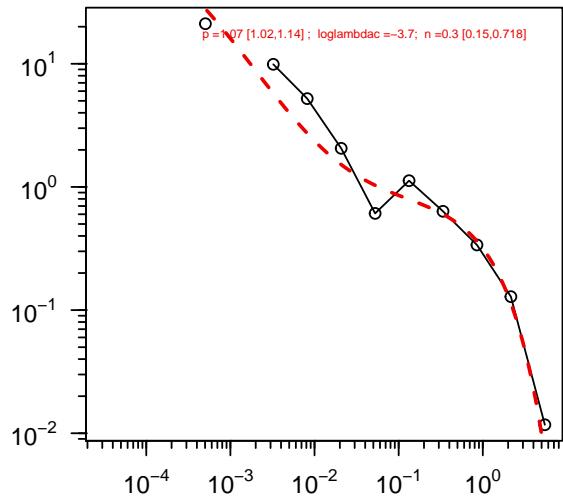
**Family 70057133 ; nev = 110**



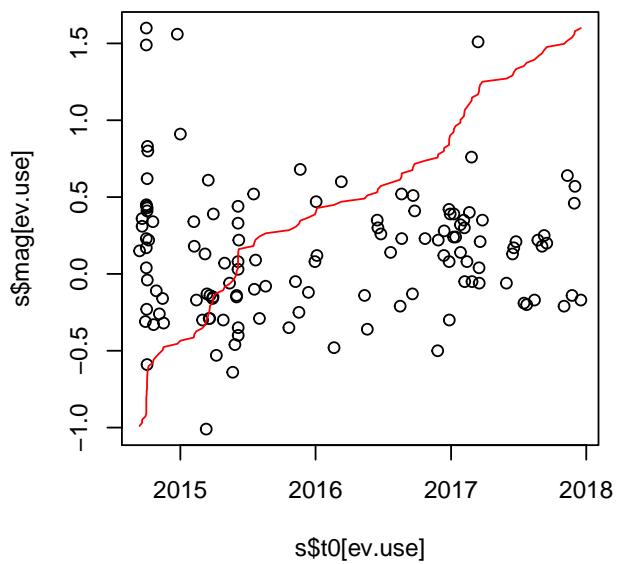
**BFMI = 1.2 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 620**



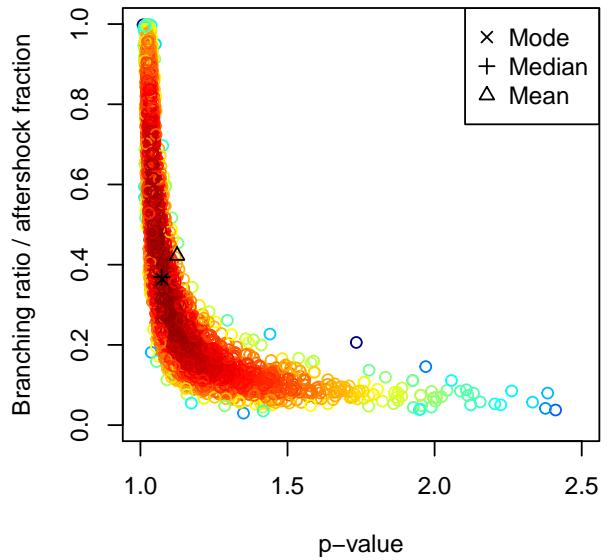
**Cv = 1.3**



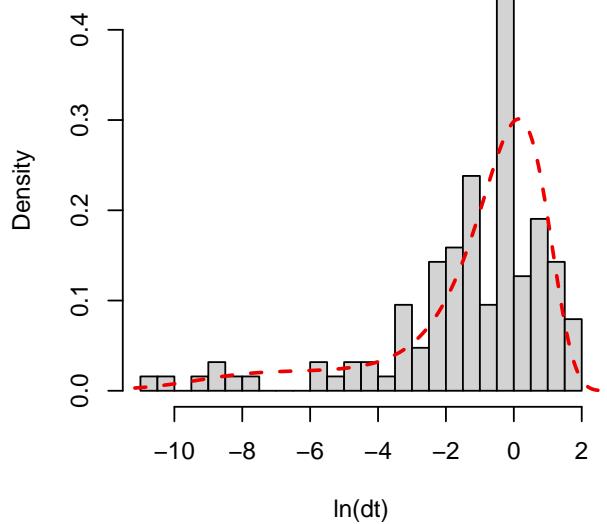
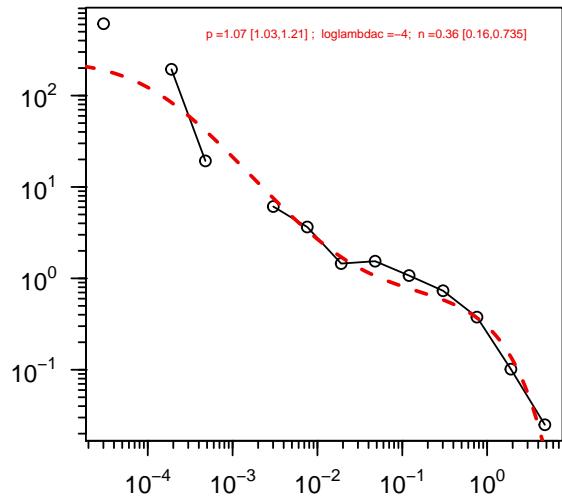
**Family 70059633 ; nev = 127**



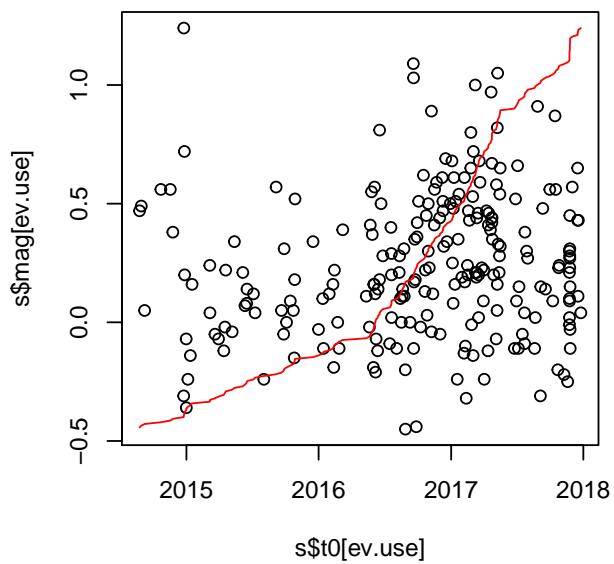
**BFMI = 1.1 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 612**



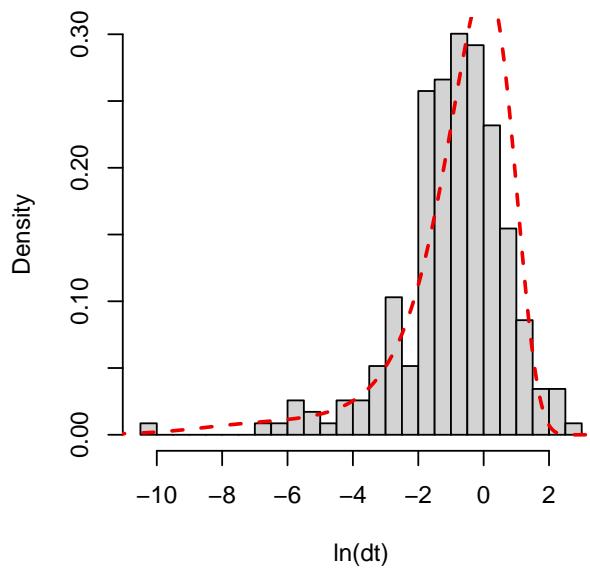
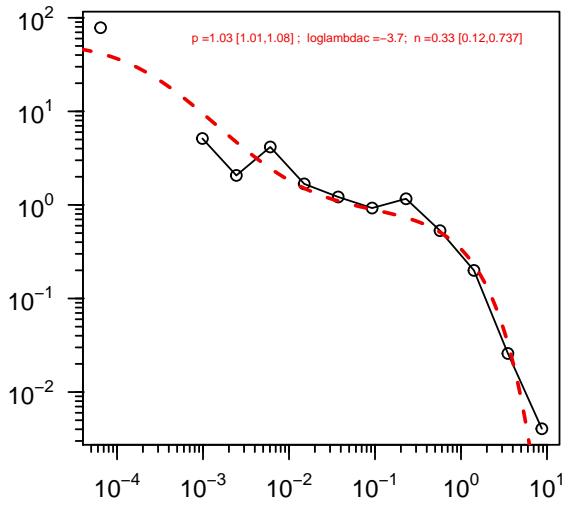
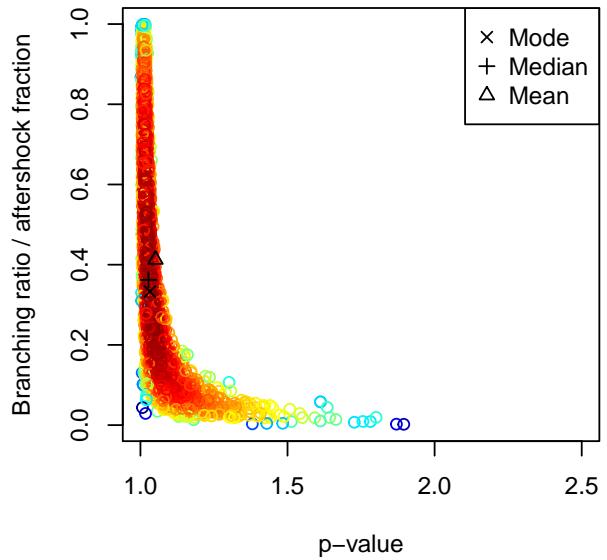
**Cv = 1.4**



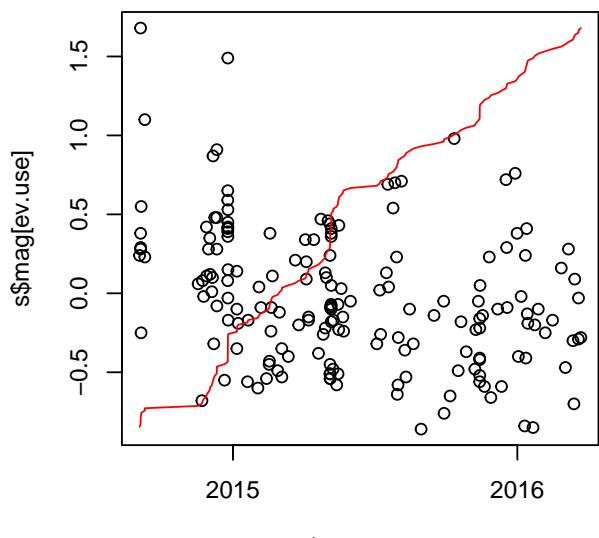
**Family 70060103 ; nev = 234**



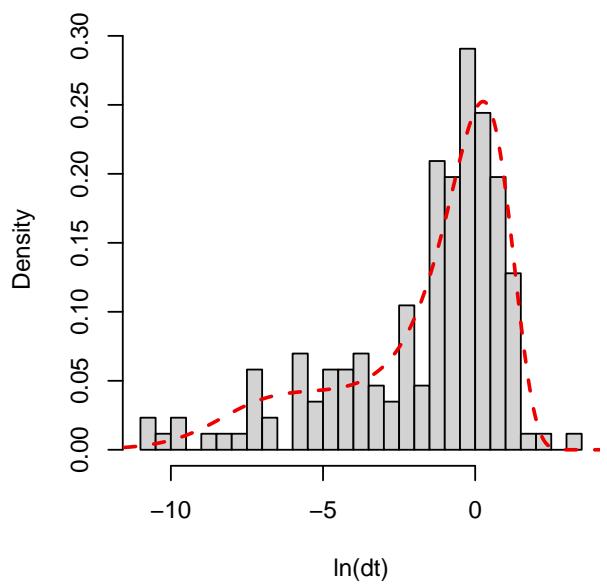
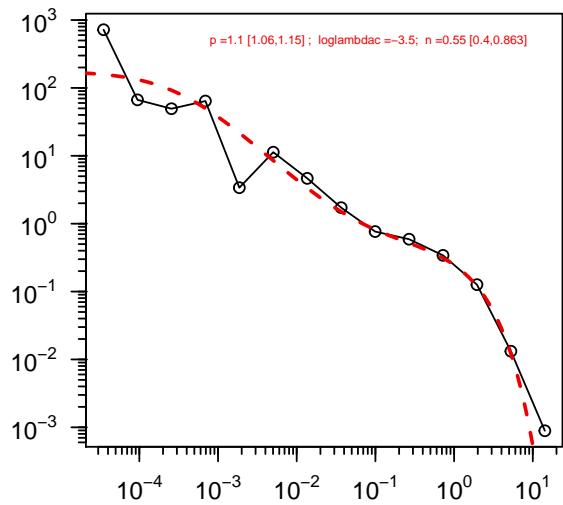
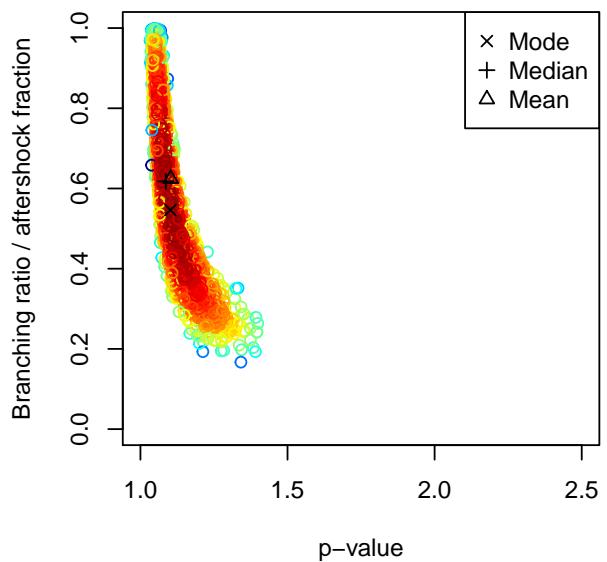
**BFMI = 0.87 ; n\_div = 0  
rhat = 1 ; n\_eff = 765**



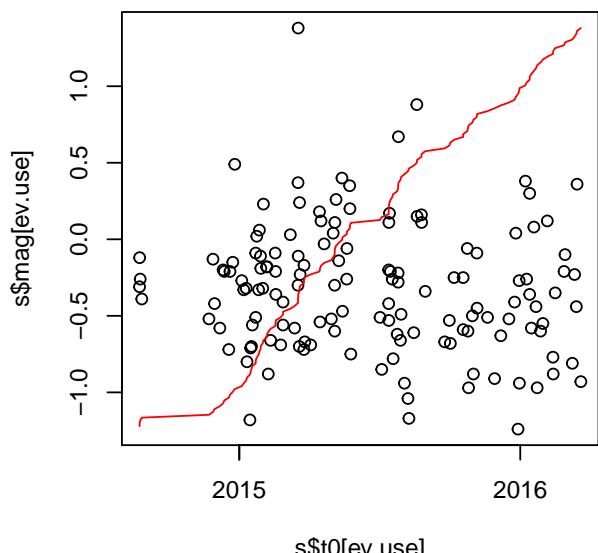
**Family 70060173 ; nev = 173**



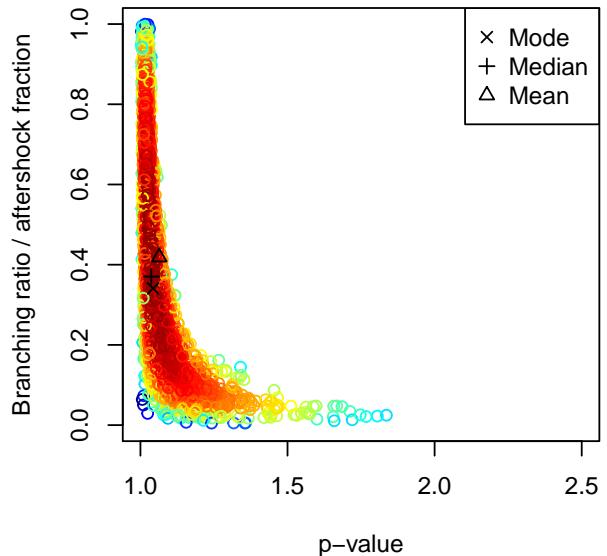
**BFMI = 0.84 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 737**



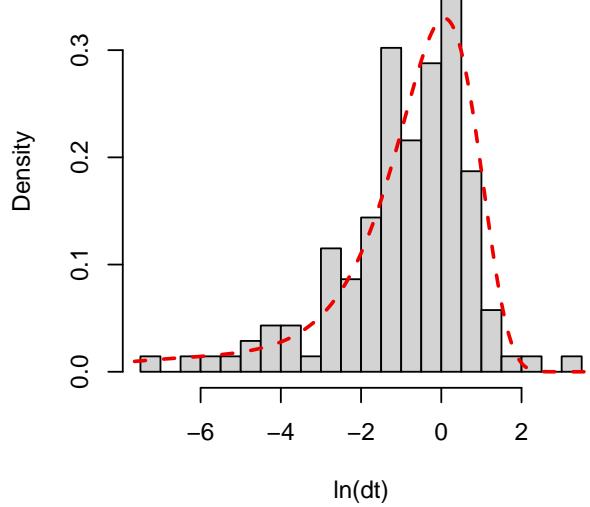
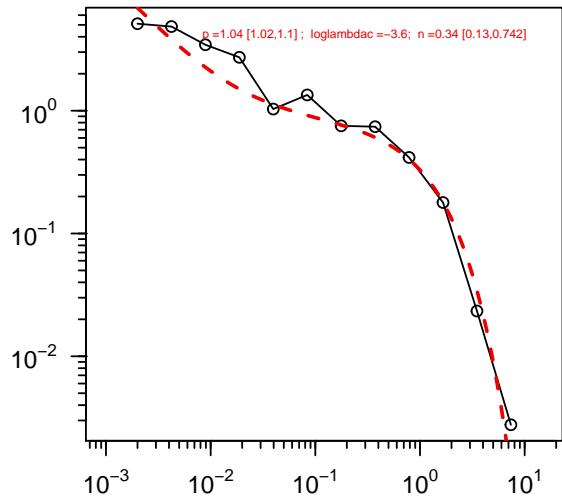
**Family 70060258 ; nev = 140**



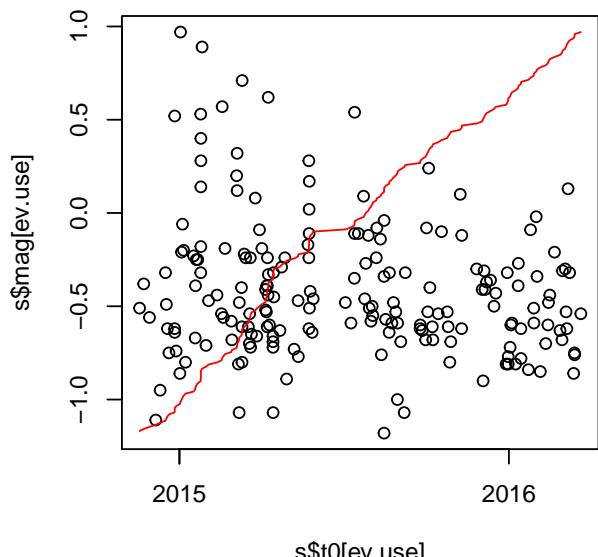
**BFMI = 0.87 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 801**



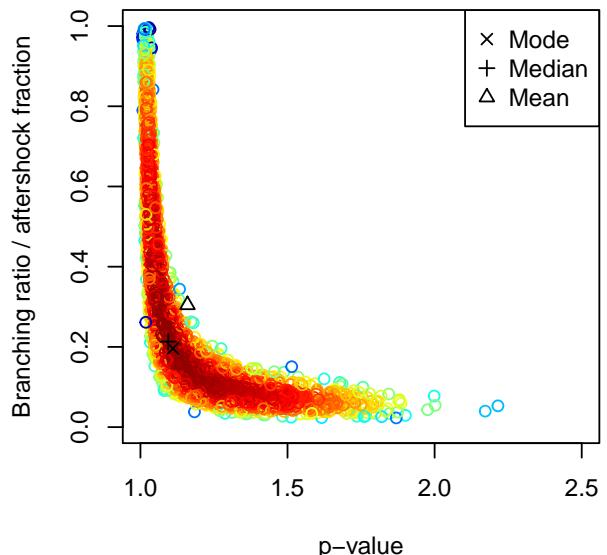
**Cv = 2.1**



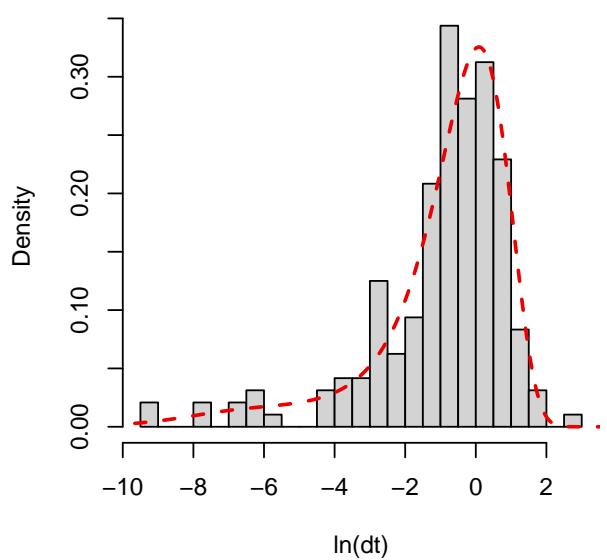
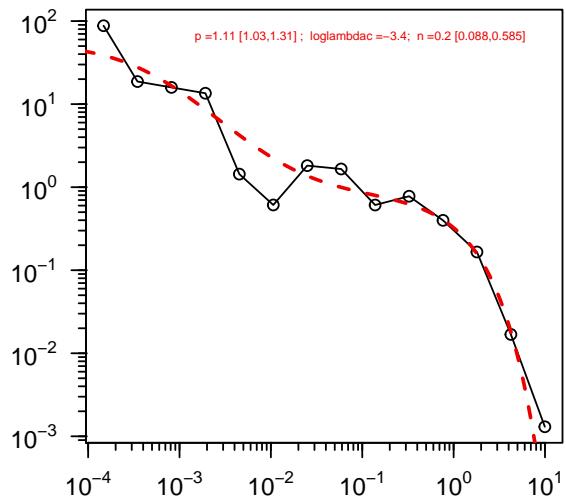
**Family 70060358 ; nev = 193**



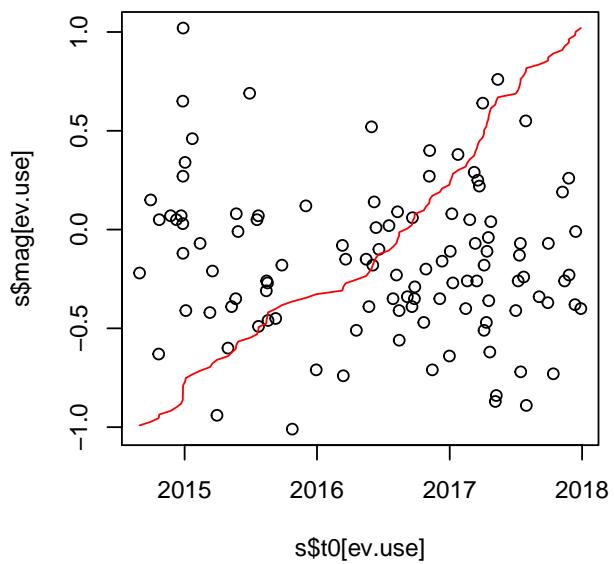
**BFMI = 0.89 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 652**



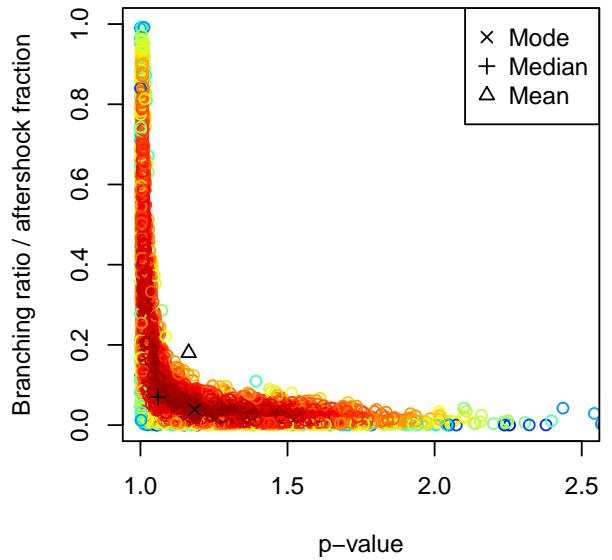
**Cv = 1.4**



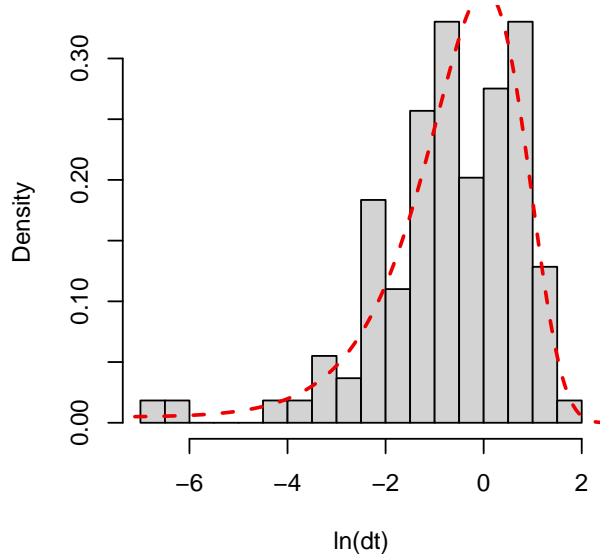
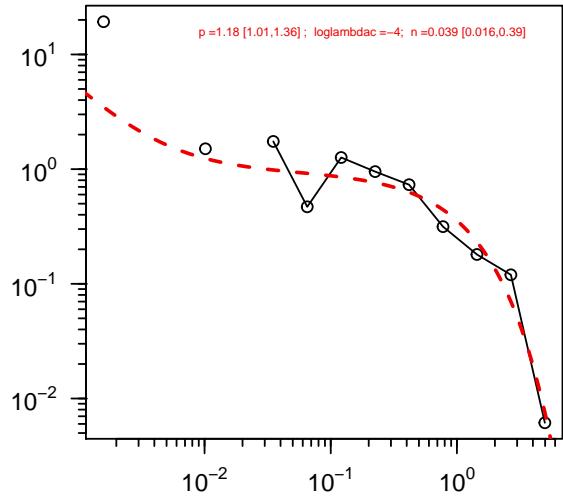
**Family 70060468 ; nev = 110**



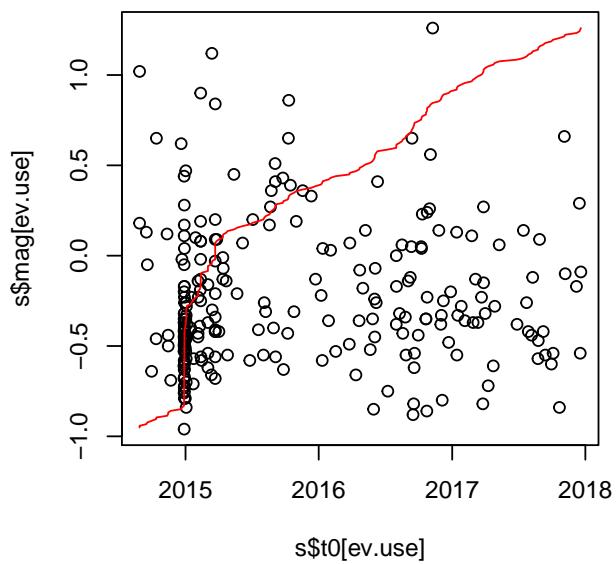
**BFMI = 1 ; n\_div = 0  
rhat = 1 ; n\_eff = 706**



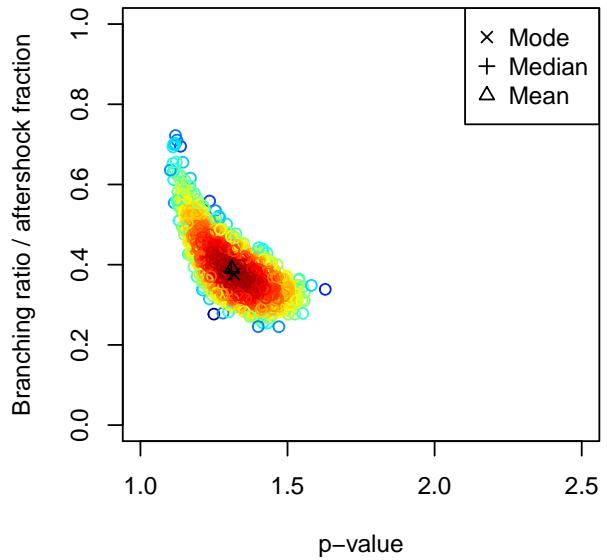
**Cv = 1.1**



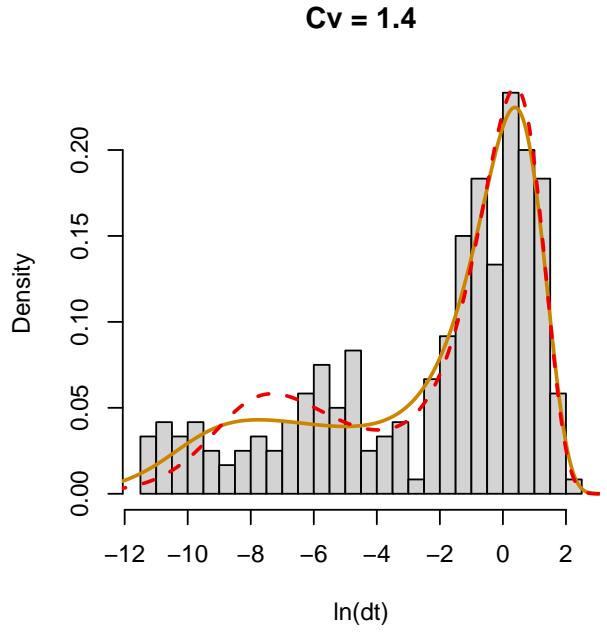
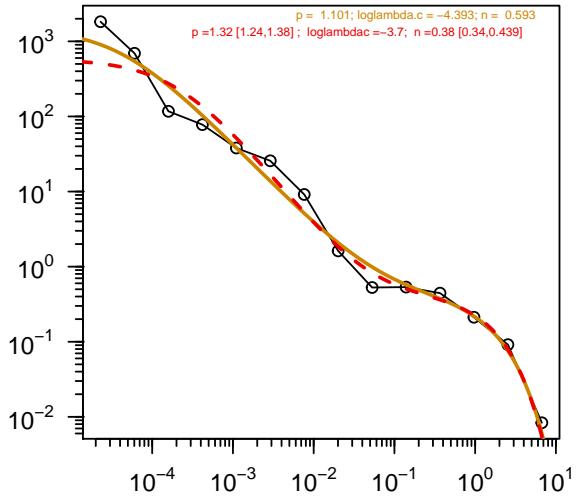
**Family 70060568 ; nev = 241**



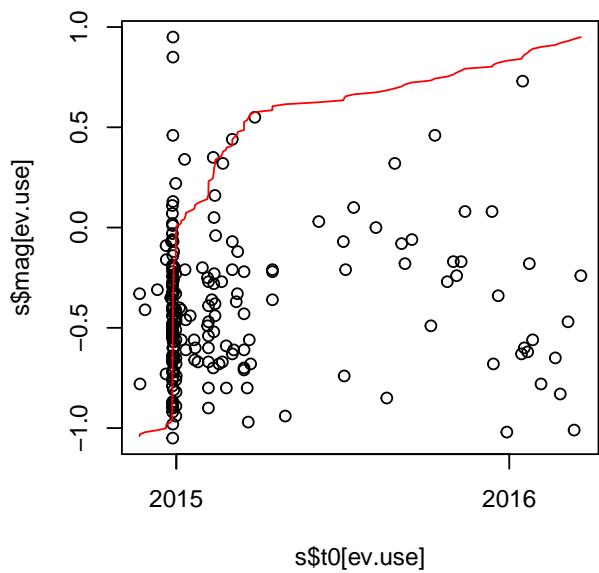
**BFMI = 1 ; n\_div = 0  
rhat = 1 ; n\_eff = 801**



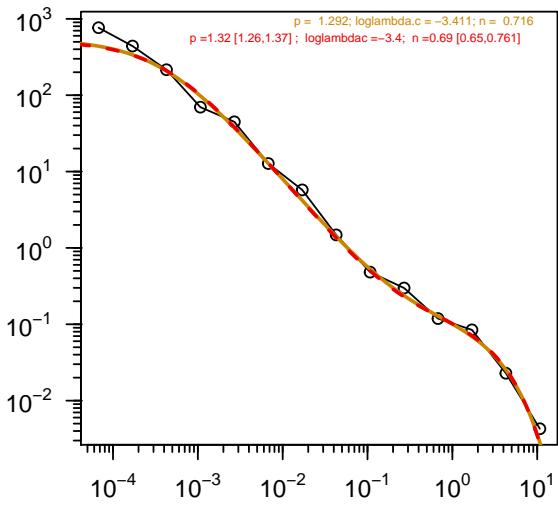
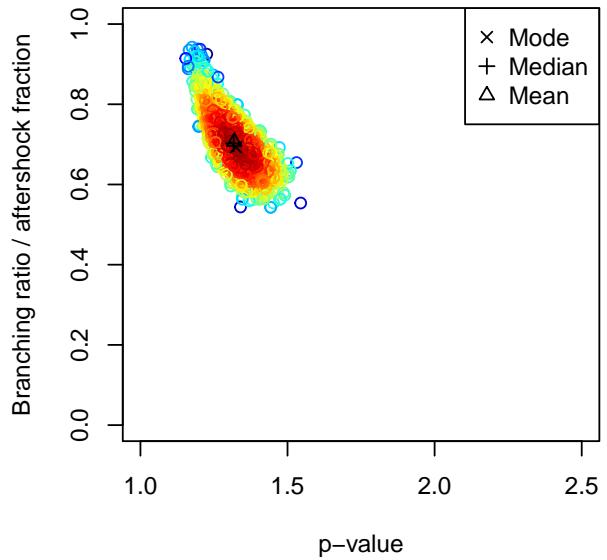
**Cv = 1.4**



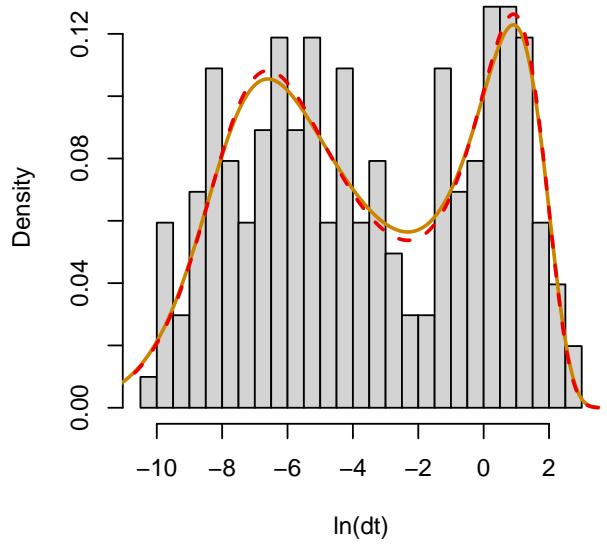
**Family 70060593 ; nev = 203**



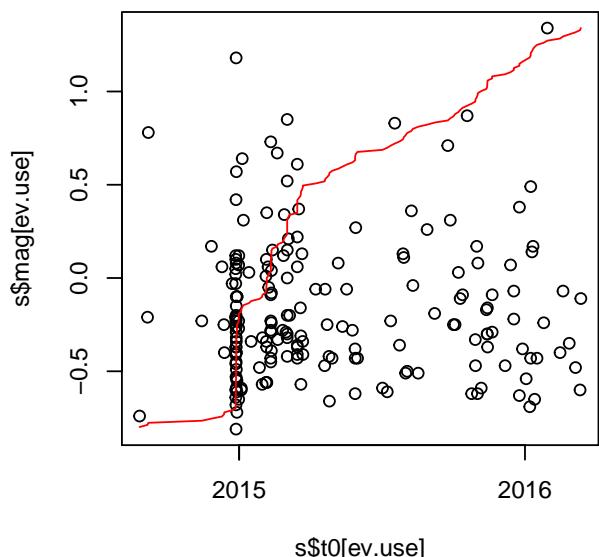
**BFMI = 1.1 ; n\_div = 0  
rhat = 1 ; n\_eff = 1361**



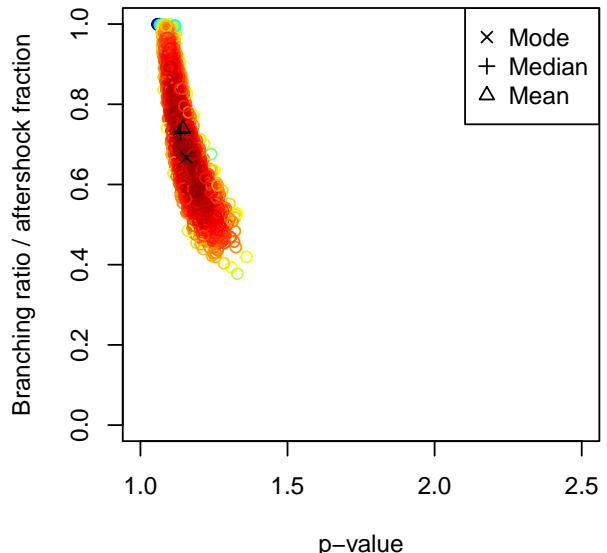
**Cv = 2.2**



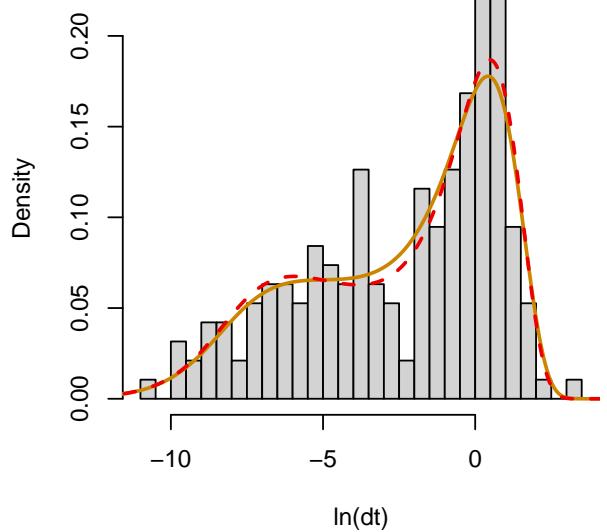
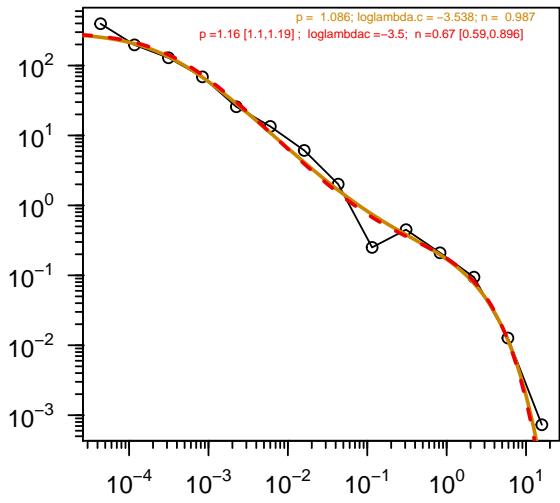
**Family 70060673 ; nev = 191**



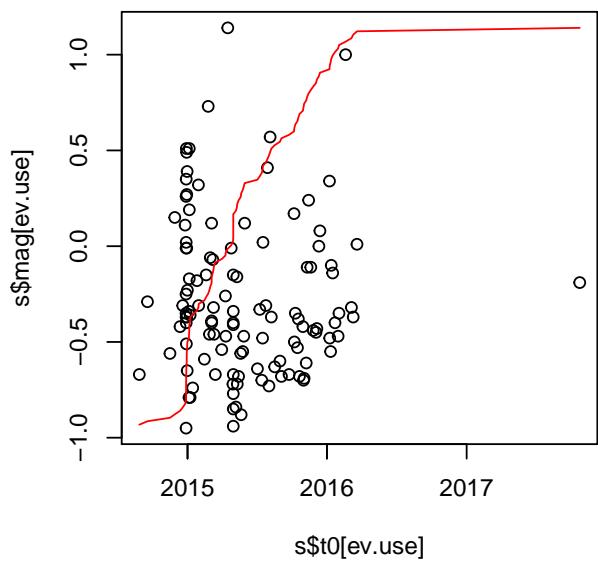
**BFMI = 0.99 ; n\_div = 0  
rhat = 1 ; n\_eff = 701**



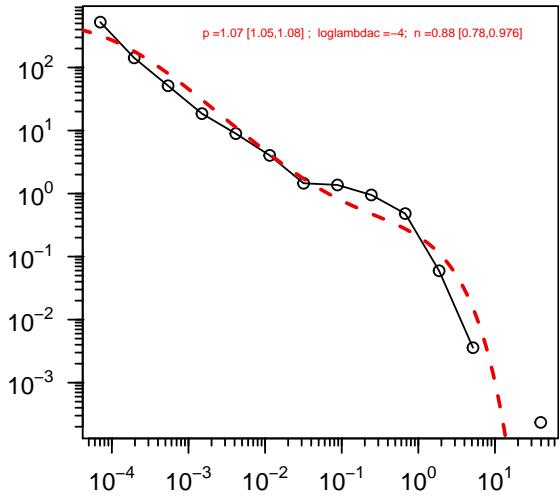
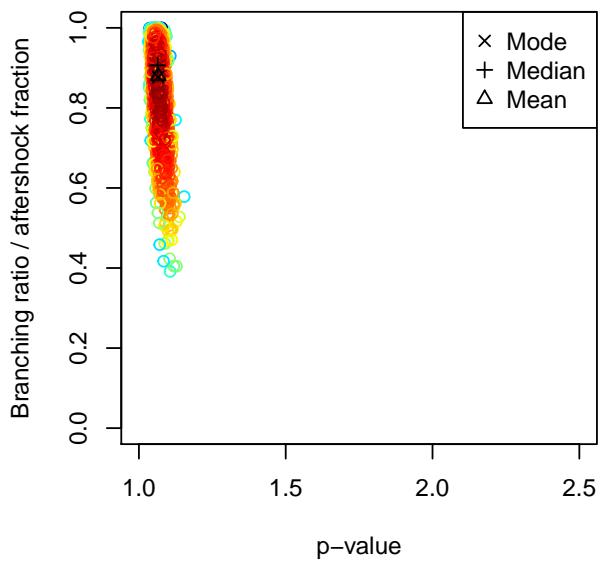
**Cv = 2.1**



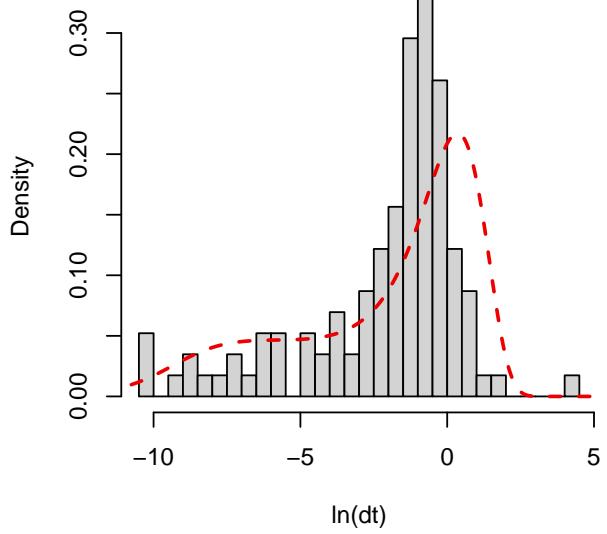
**Family 70061463 ; nev = 116**



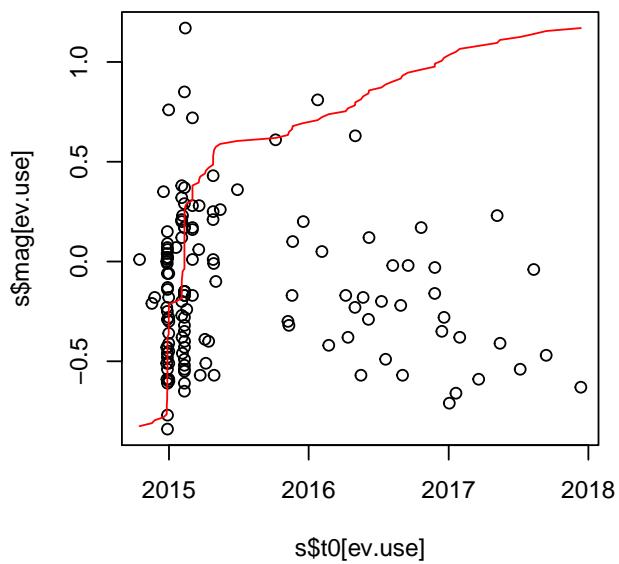
**BFMI = 0.91 ; n\_div = 0  
rhat = 1 ; n\_eff = 966**



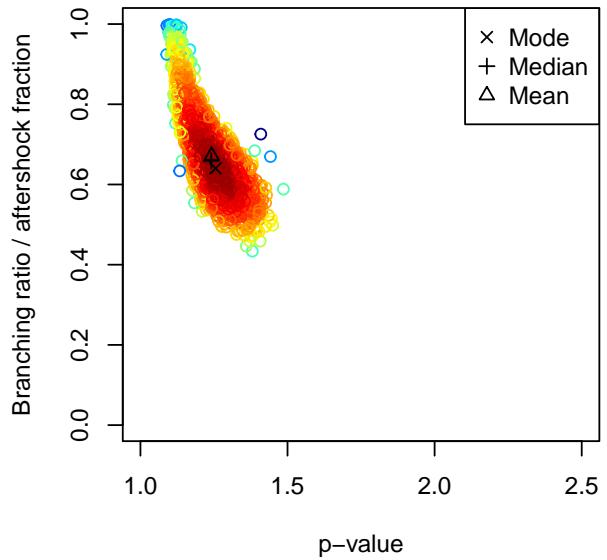
**Cv = 5.4**



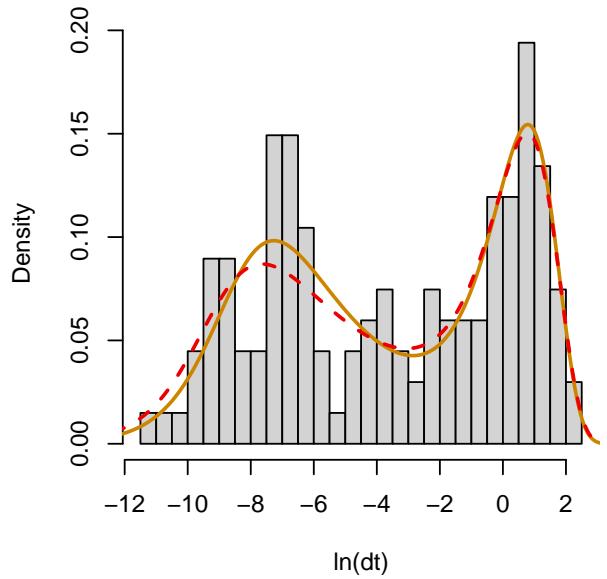
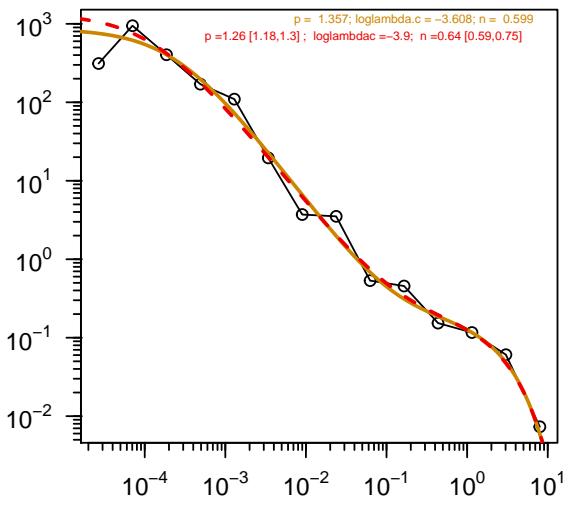
**Family 70061938 ; nev = 135**



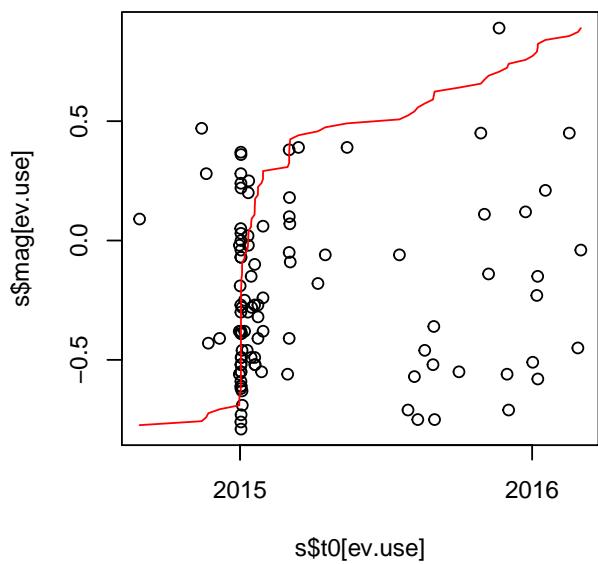
**BFMI = 1.1 ; n\_div = 0  
rhat = 1 ; n\_eff = 587**



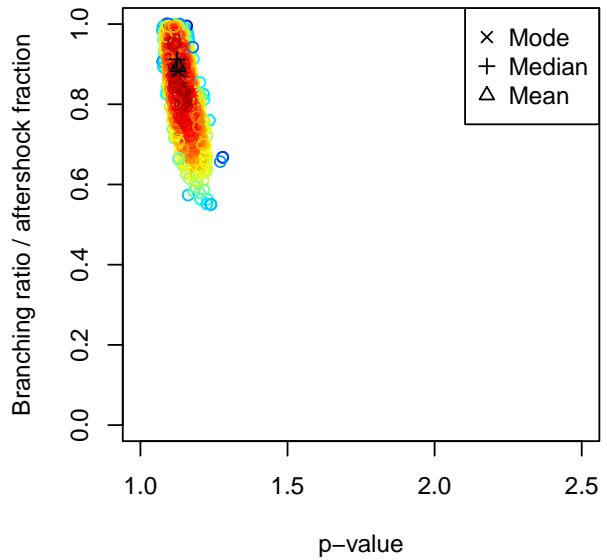
**Cv = 1.9**



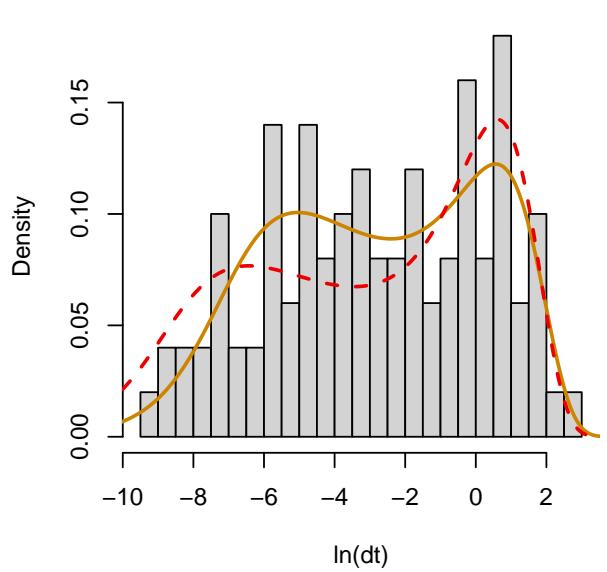
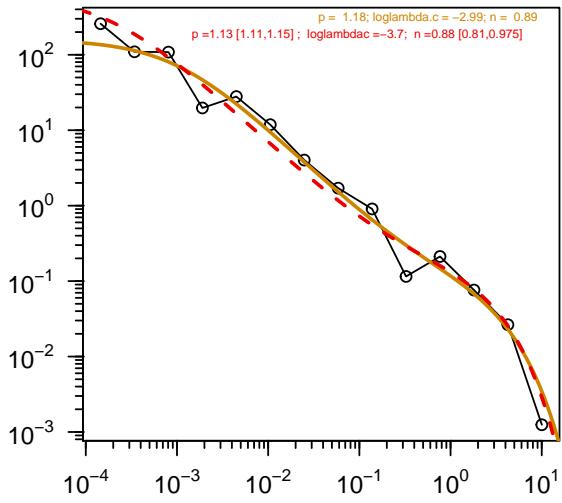
**Family 70062403 ; nev = 101**



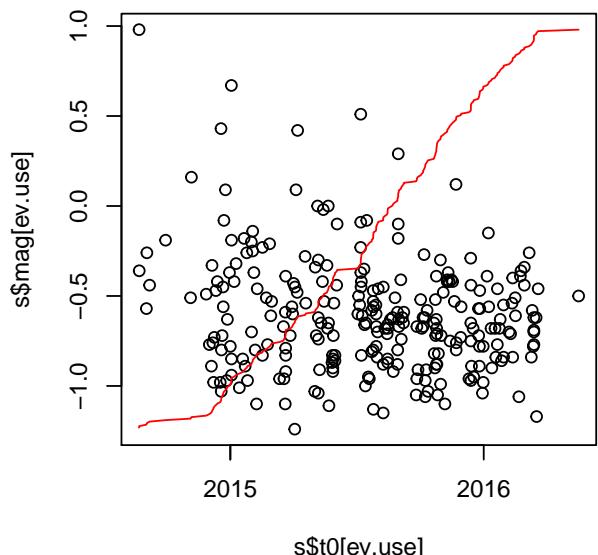
**BFMI = 0.96 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 652**



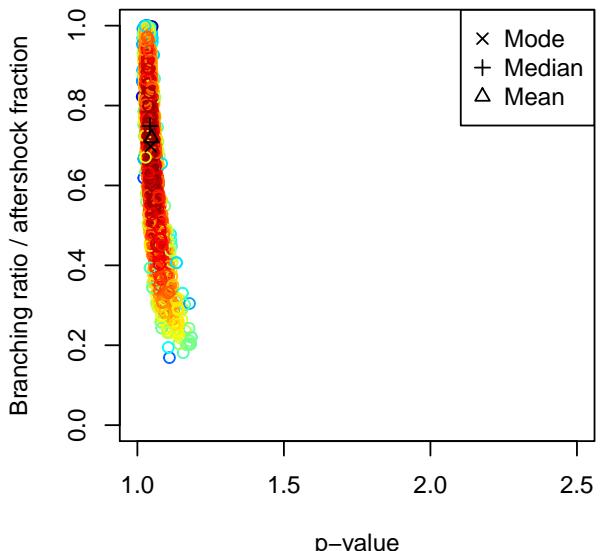
**Cv = 2.2**



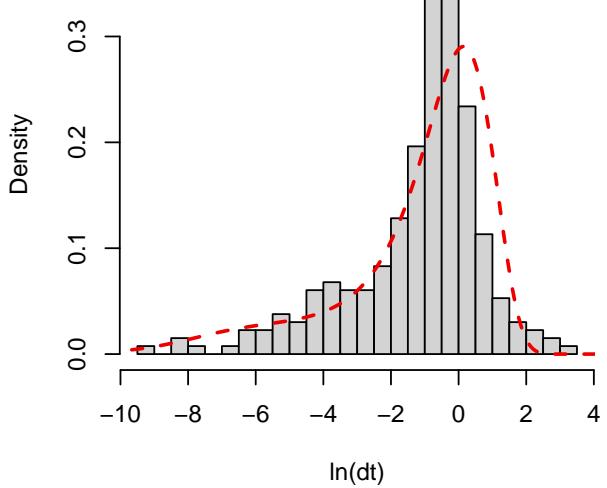
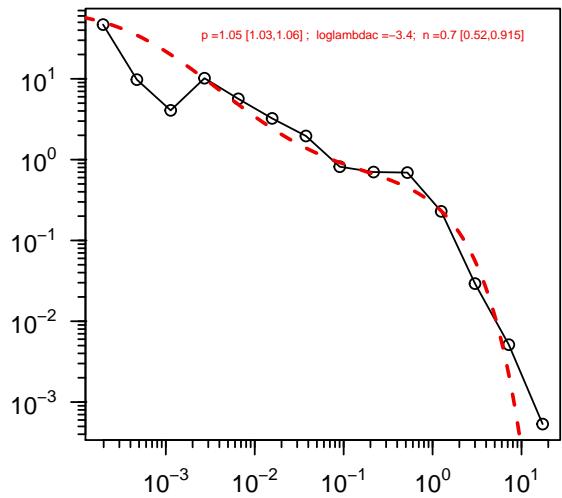
**Family 70062518 ; nev = 266**



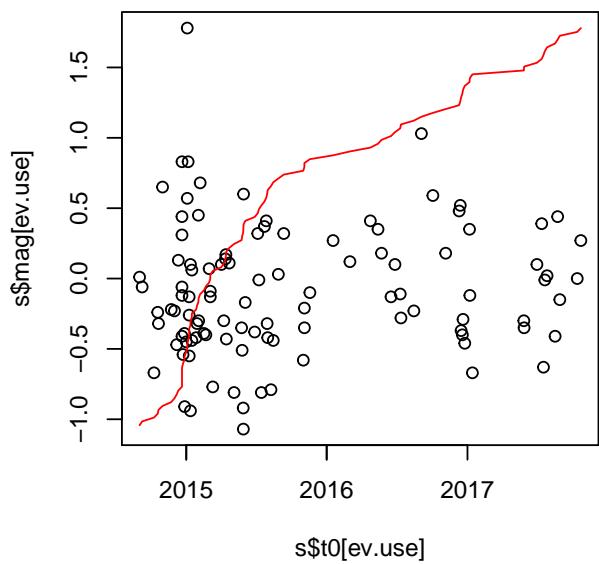
**BFMI = 1 ; n\_div = 0  
rhat = 1 ; n\_eff = 645**



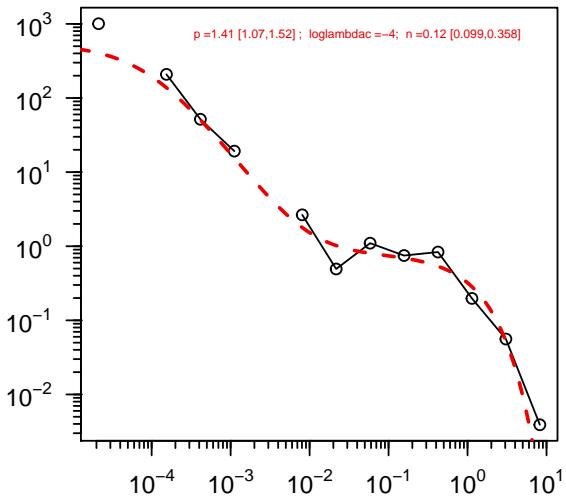
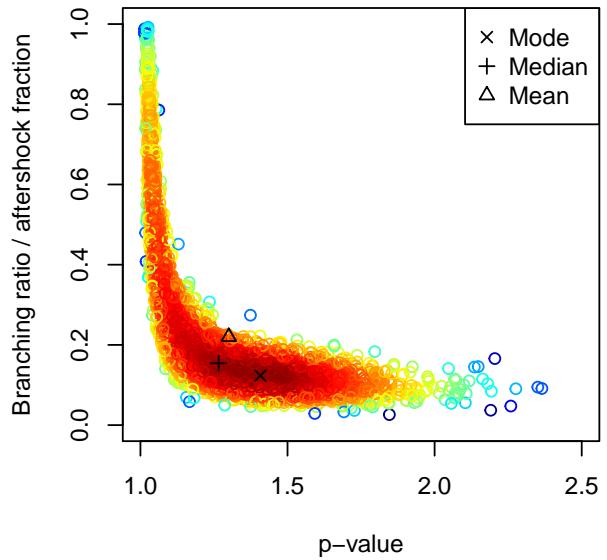
**Cv = 2.2**



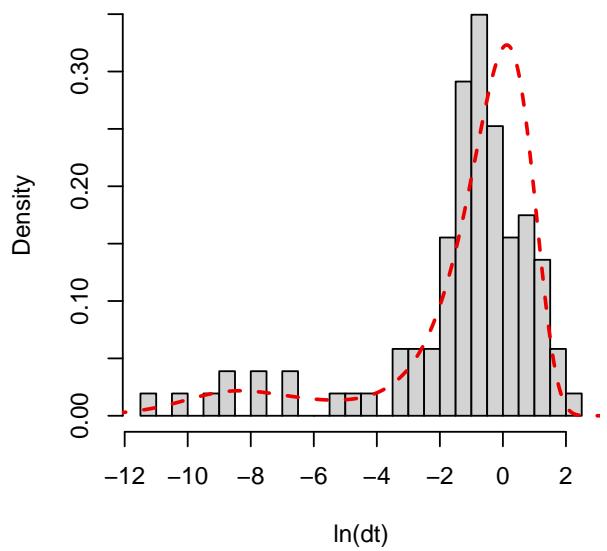
**Family 70062978 ; nev = 104**



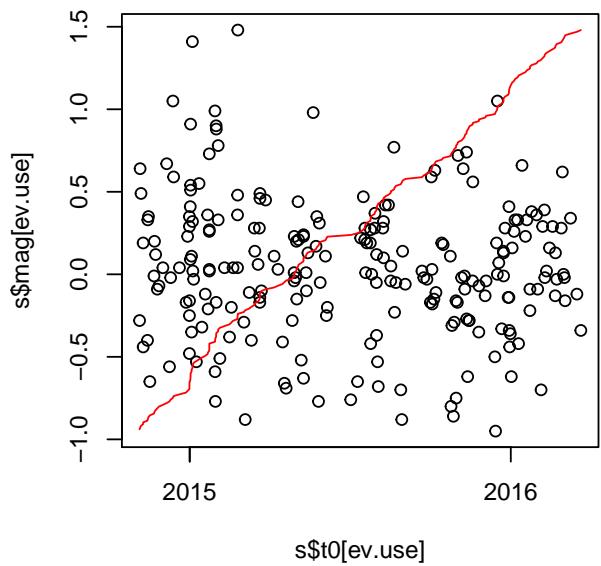
**BFMI = 0.86 ; n\_div = 0  
rhat = 1 ; n\_eff = 558**



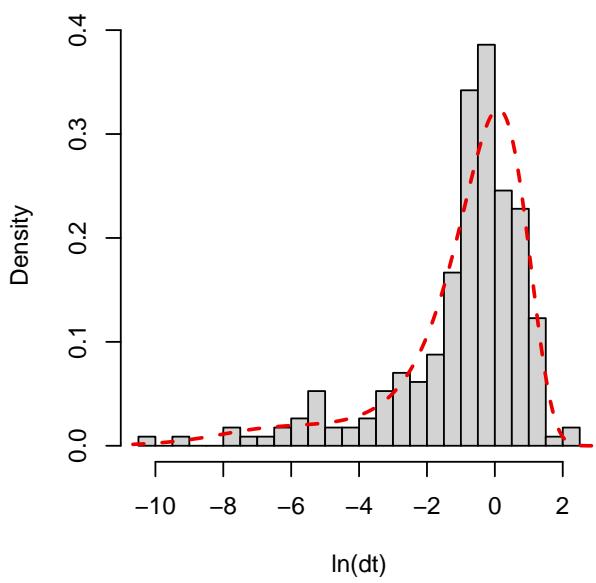
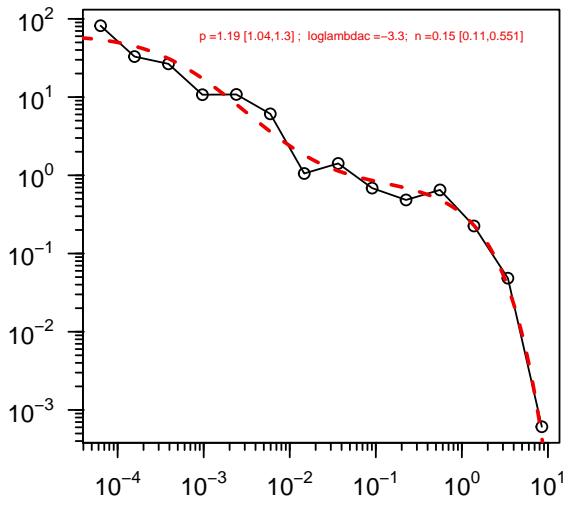
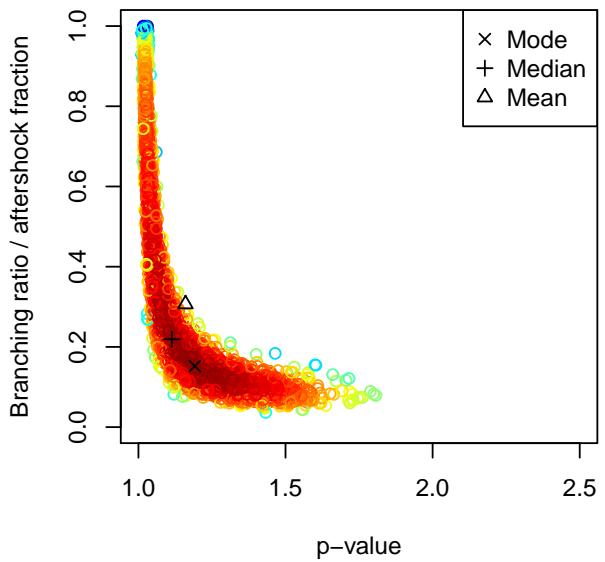
**Cv = 1.6**



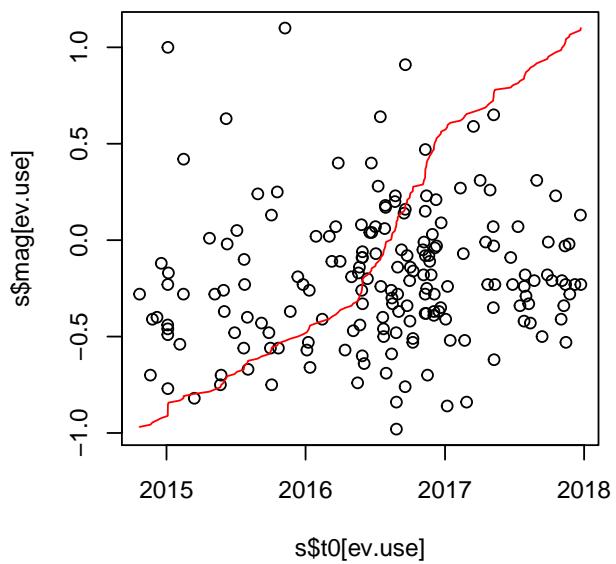
**Family 70063033 ; nev = 229**



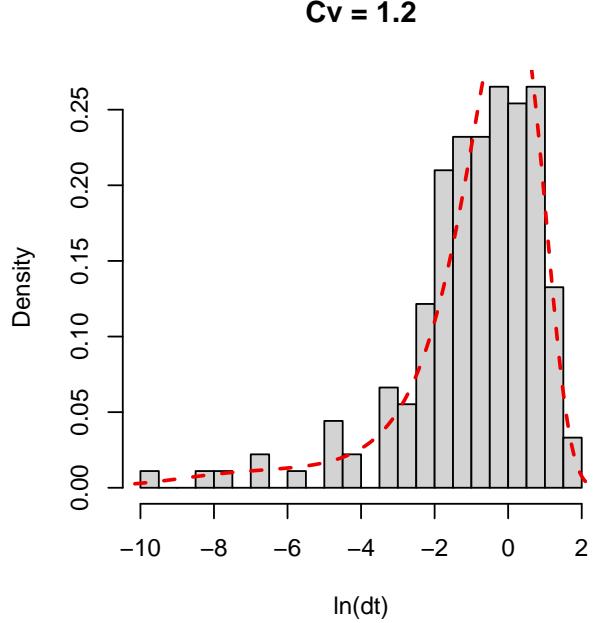
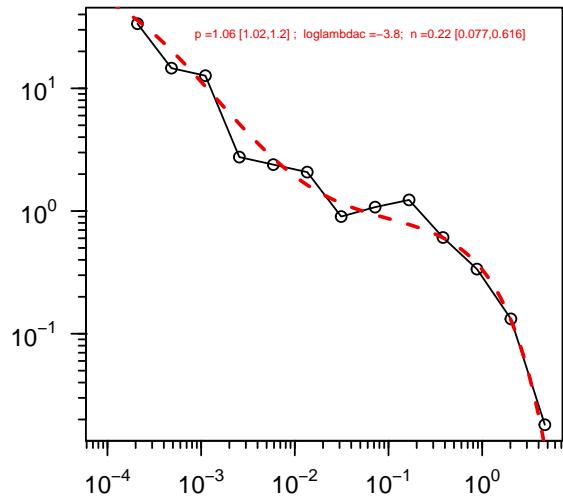
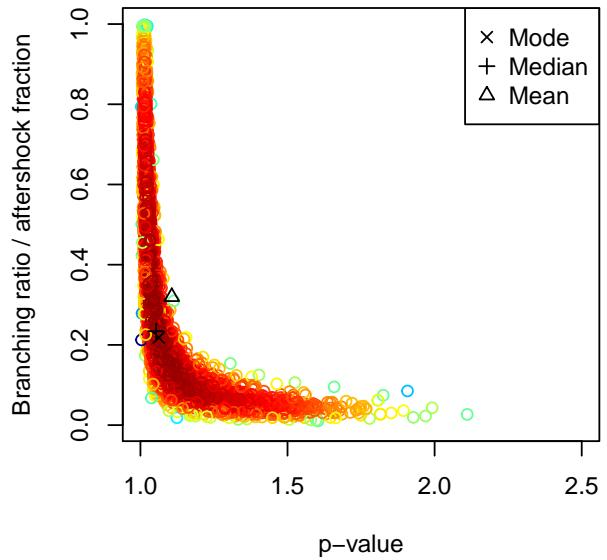
**BFMI = 0.74 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 635**



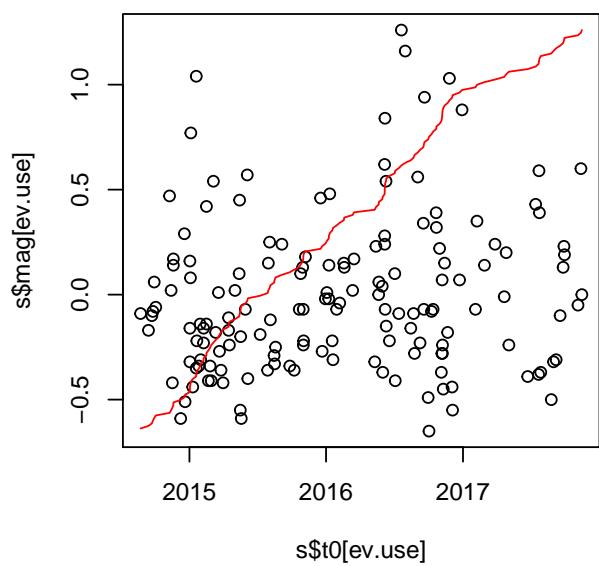
**Family 70063188 ; nev = 182**



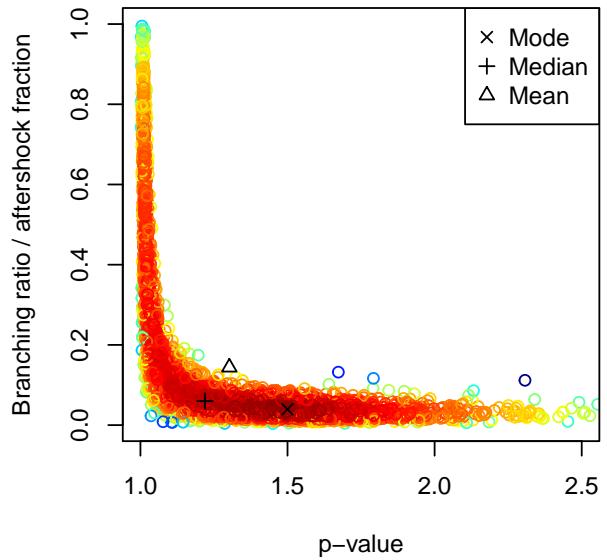
**BFMI = 0.84 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 826**



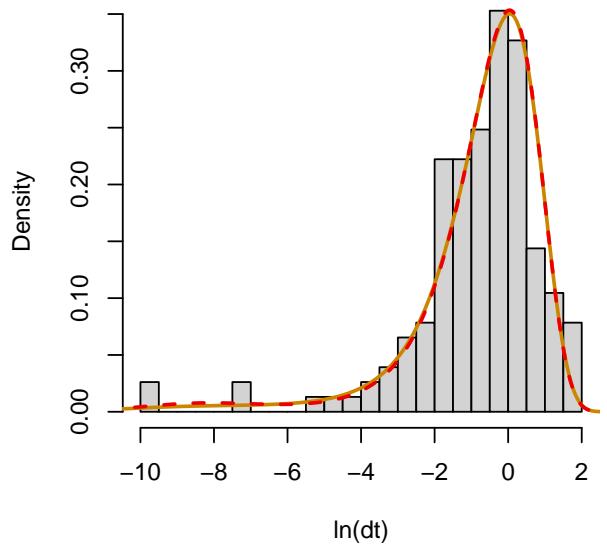
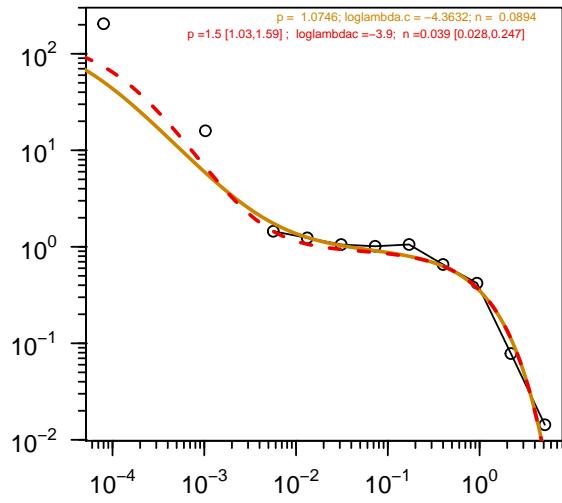
**Family 70063198 ; nev = 154**



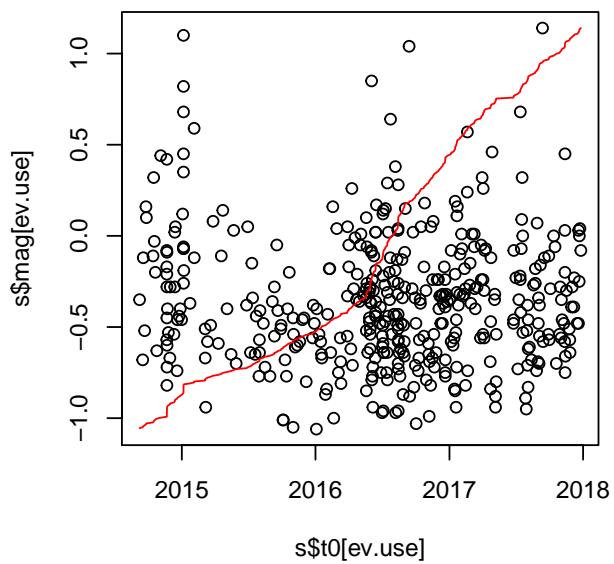
**BFMI = 0.55 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 495**



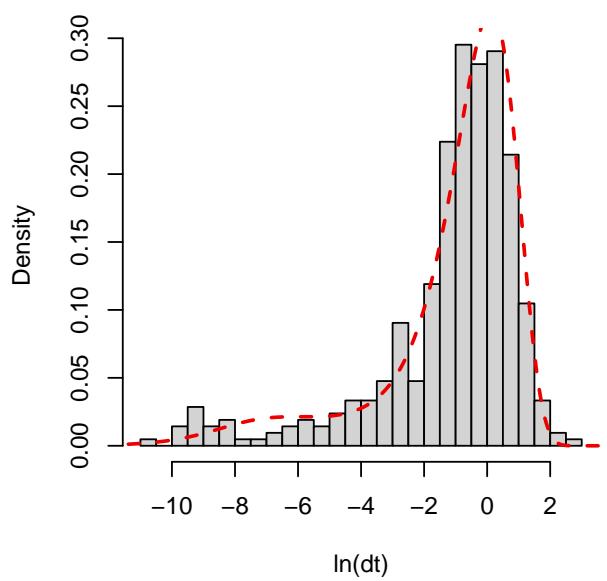
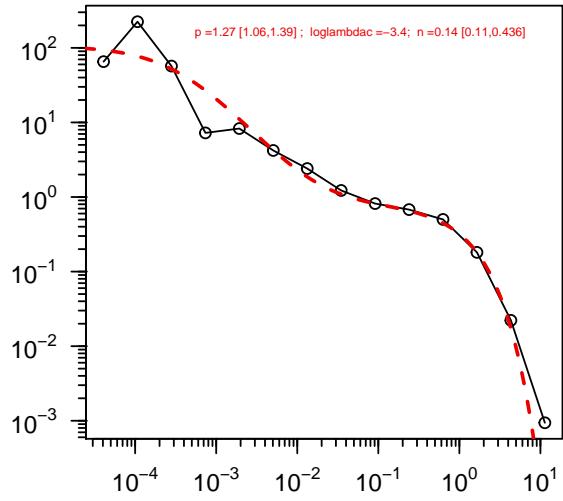
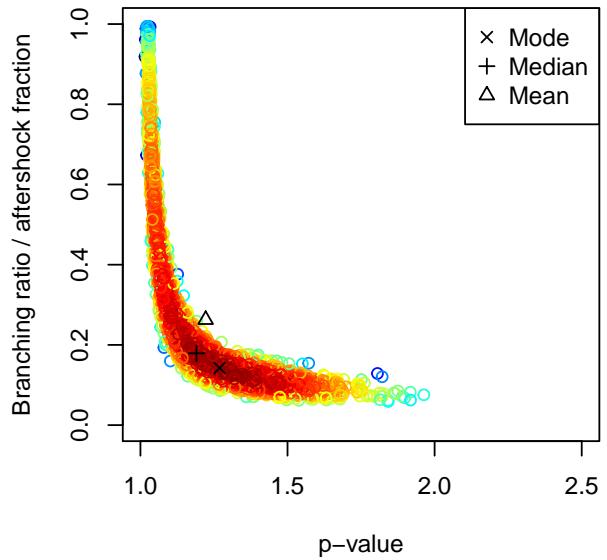
**Cv = 1.2**



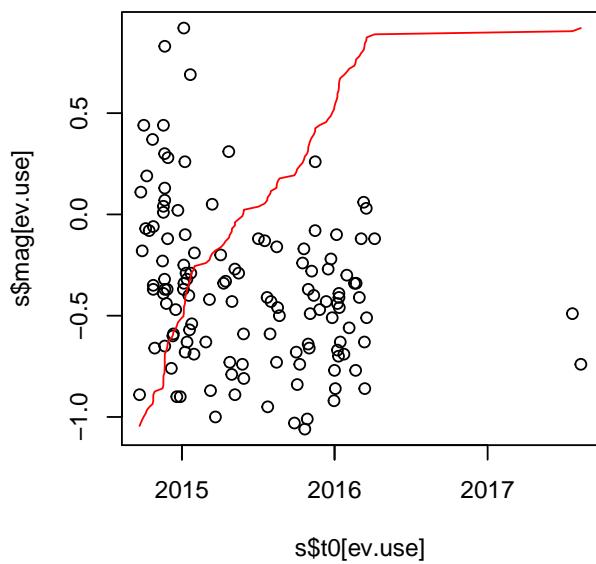
**Family 70063308 ; nev = 421**



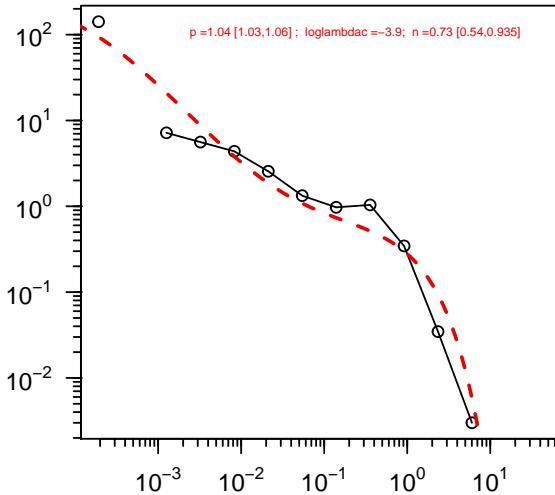
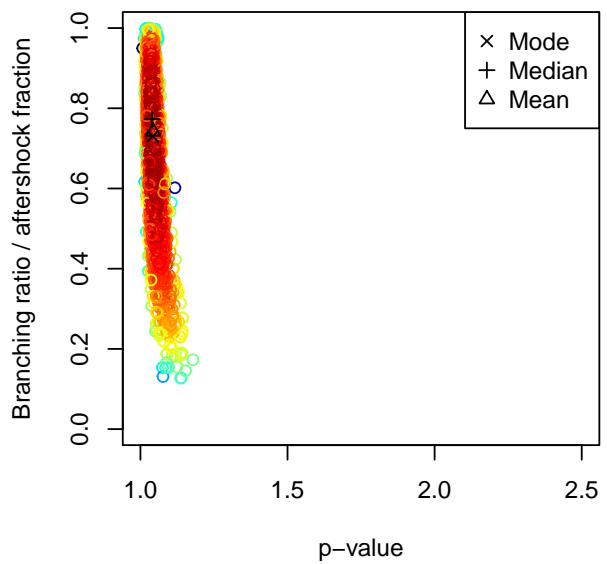
**BFMI = 0.87 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 487**



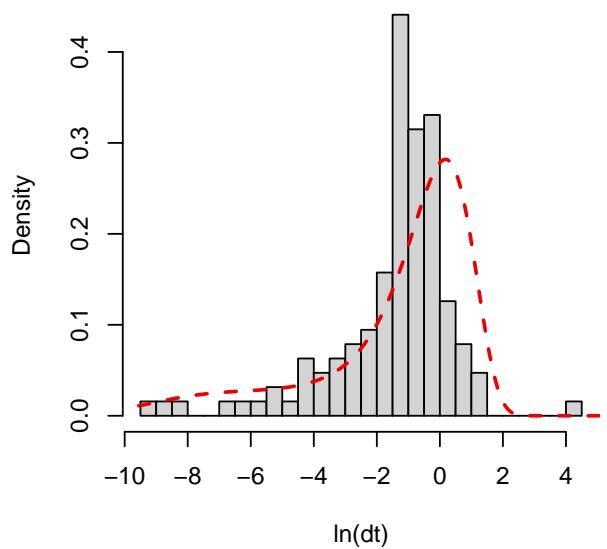
**Family 70063318 ; nev = 128**



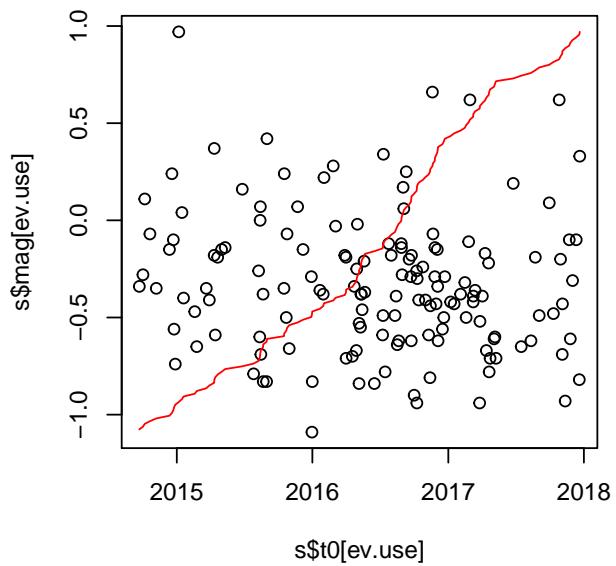
**BFMI = 1 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 844**



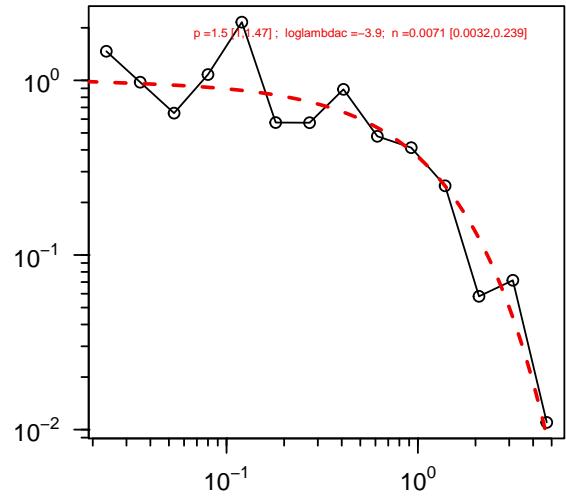
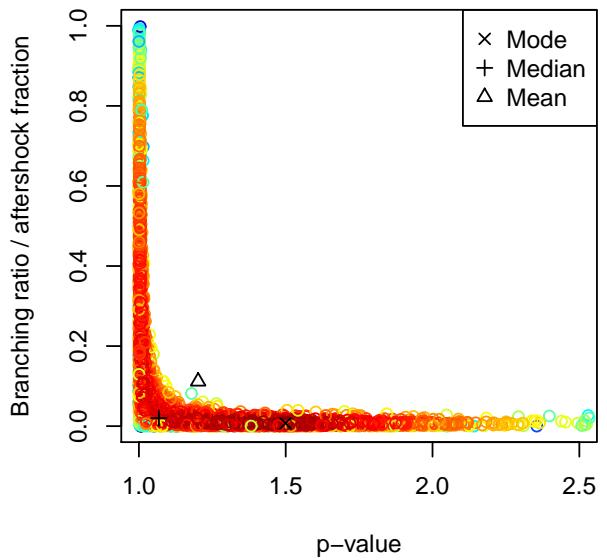
**Cv = 5.1**



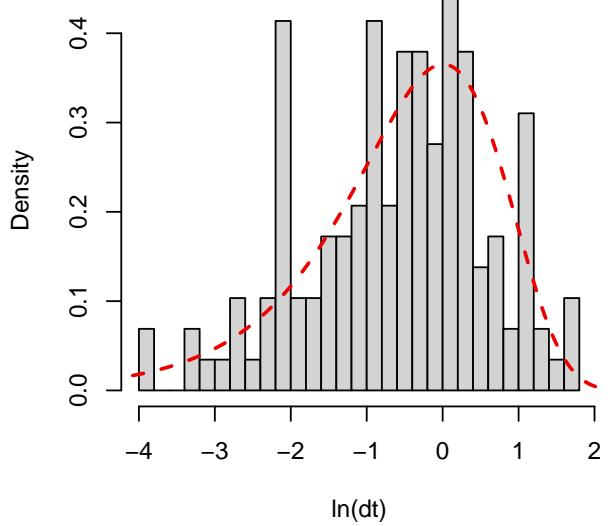
**Family 70063728 ; nev = 146**



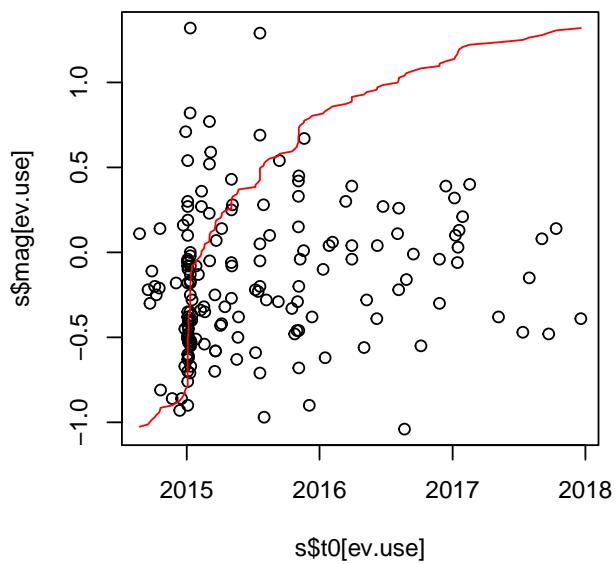
**BFMI = 1.1 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 715**



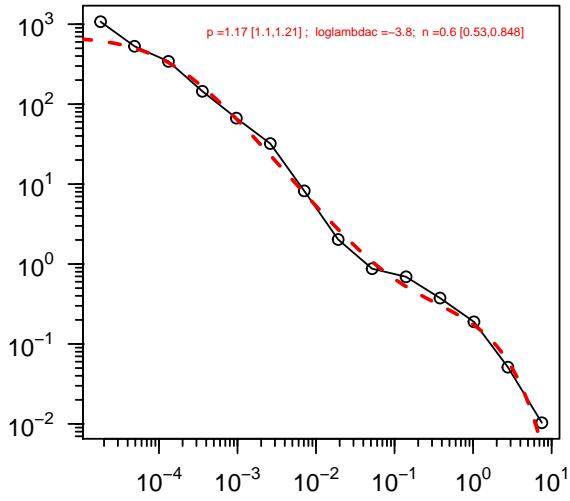
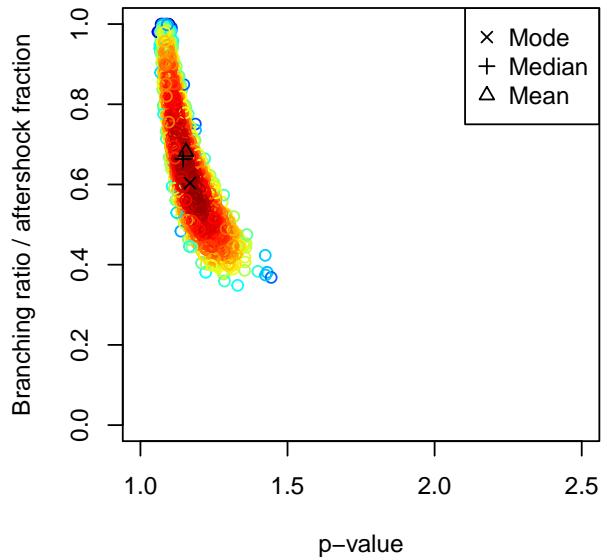
**Cv = 1.1**



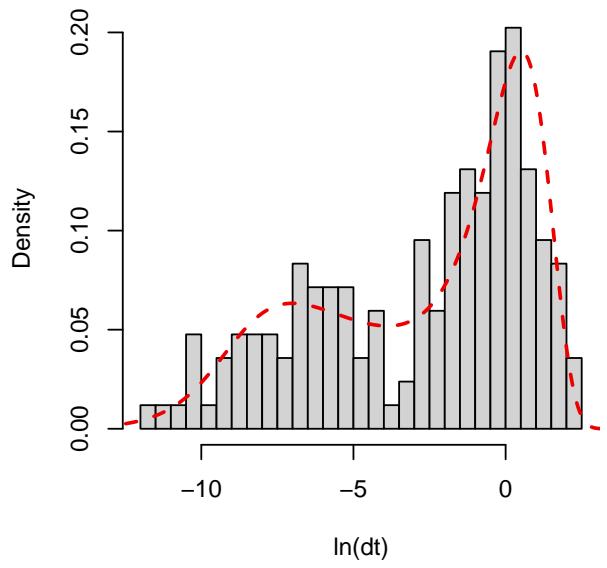
**Family 70065443 ; nev = 169**



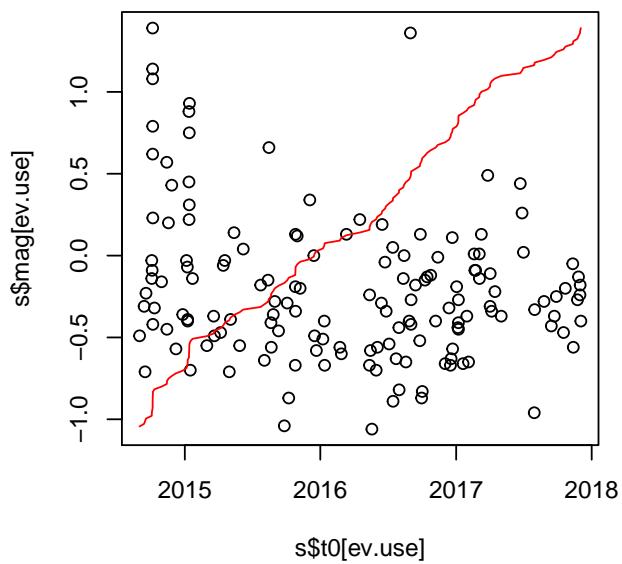
**BFMI = 1.1 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 661**



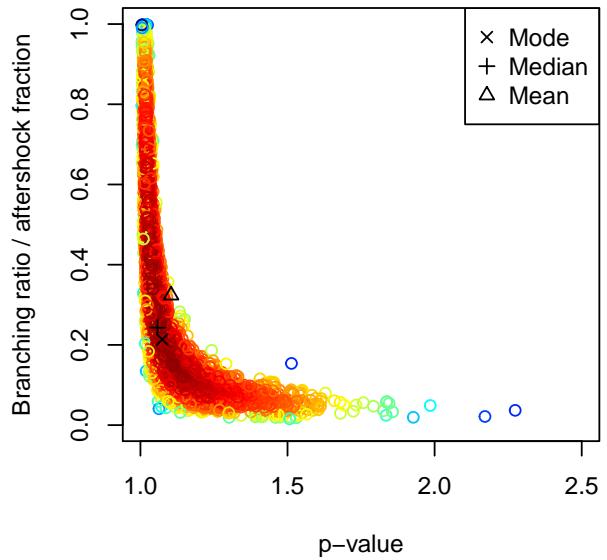
**Cv = 1.8**



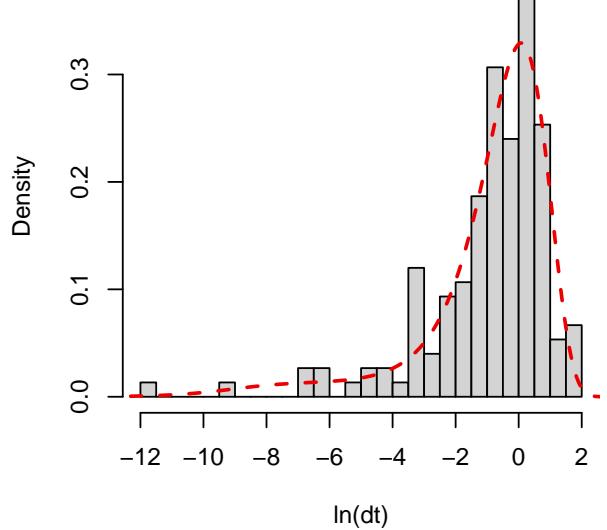
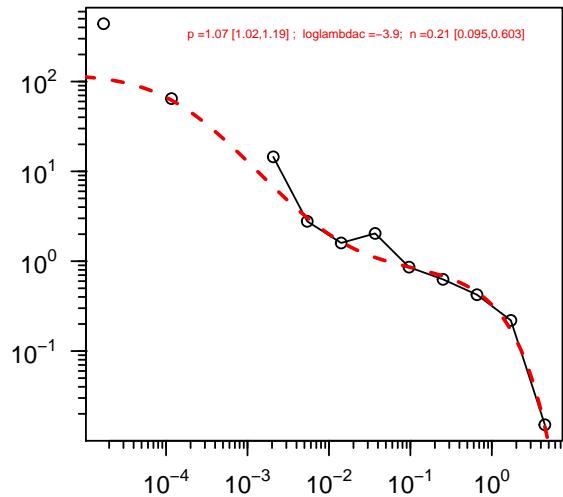
**Family 70066173 ; nev = 151**



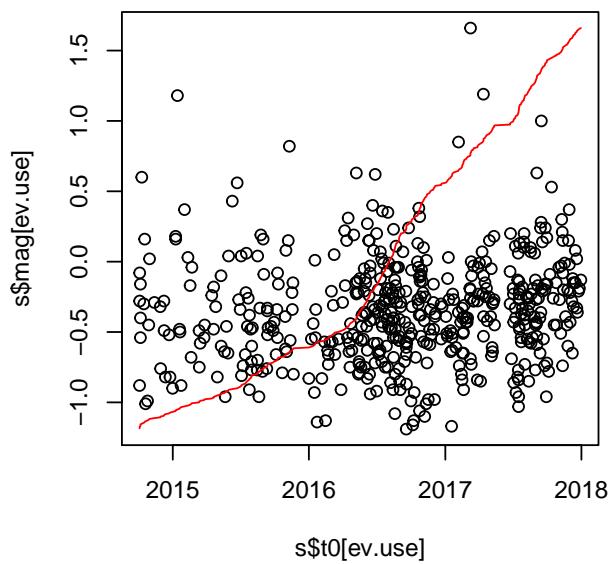
**BFMI = 1.1 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 875**



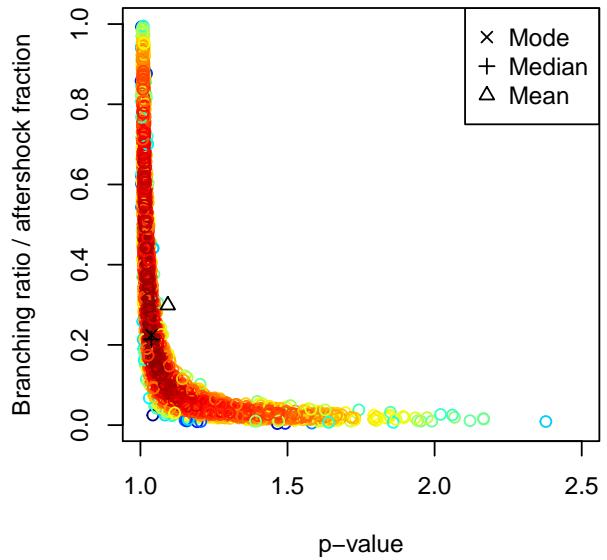
**Cv = 1.2**



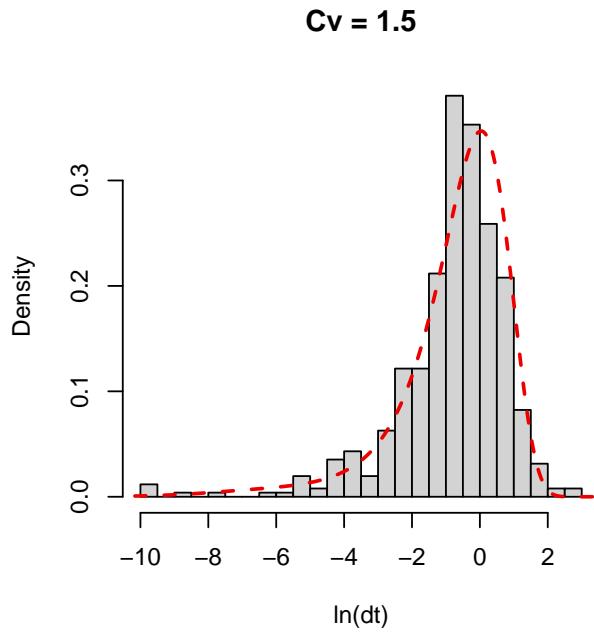
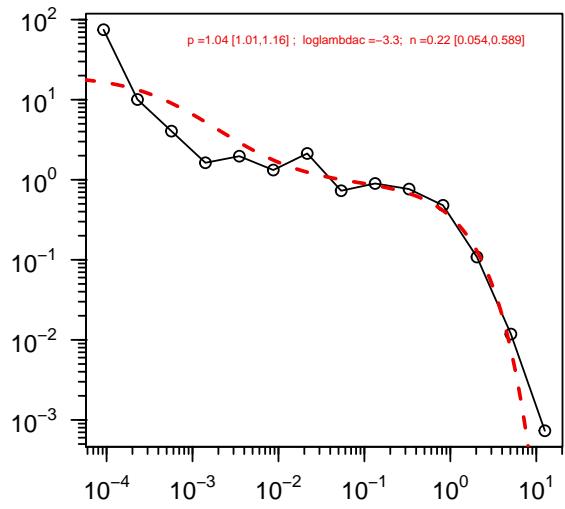
**Family 70066428 ; nev = 511**



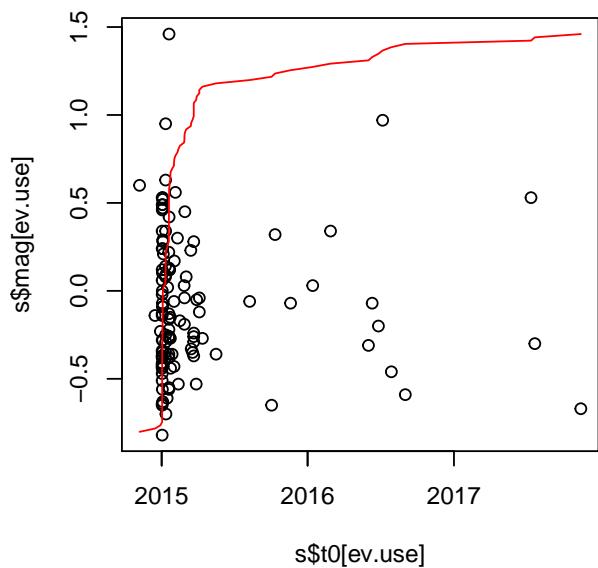
**BFMI = 0.97 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 638**



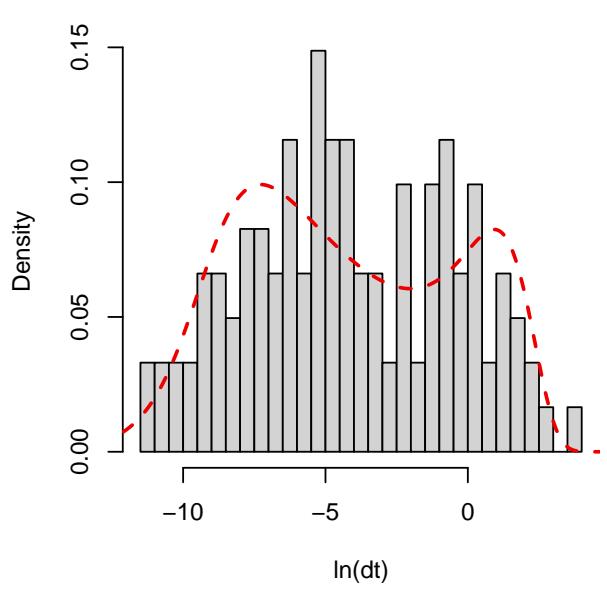
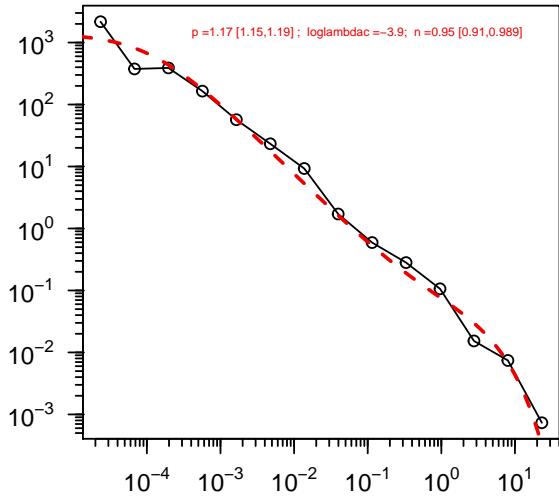
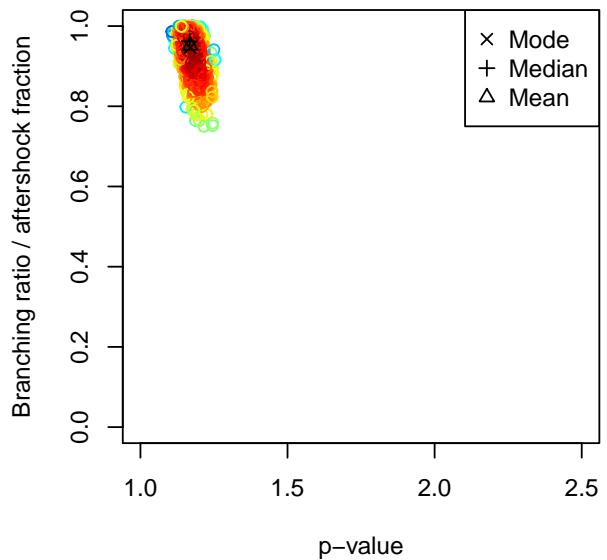
**Cv = 1.5**



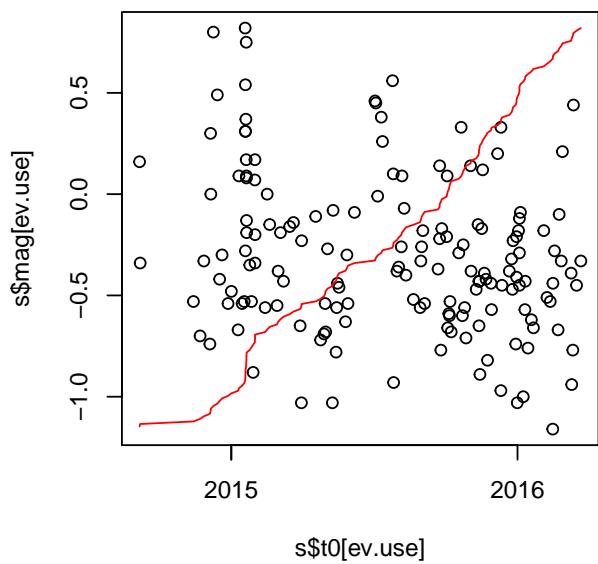
**Family 70067933 ; nev = 122**



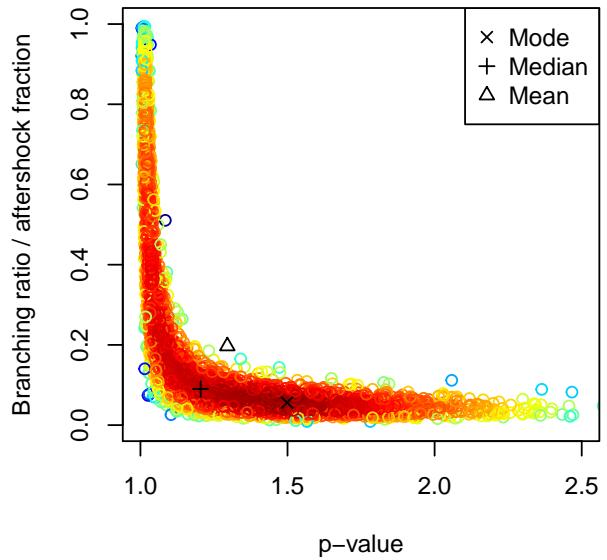
**BFMI = 1 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 955**



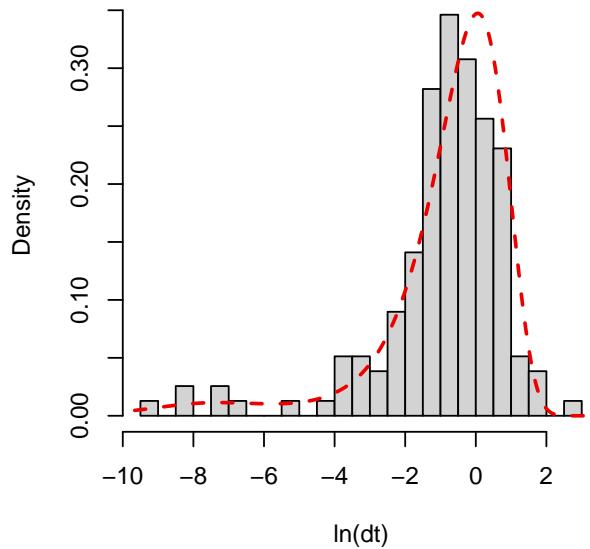
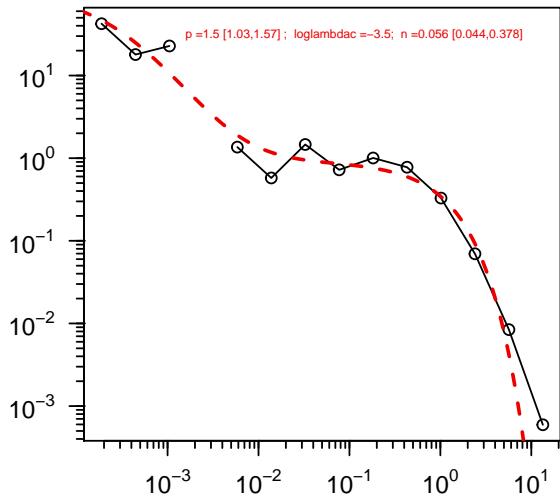
**Family 70067948 ; nev = 157**



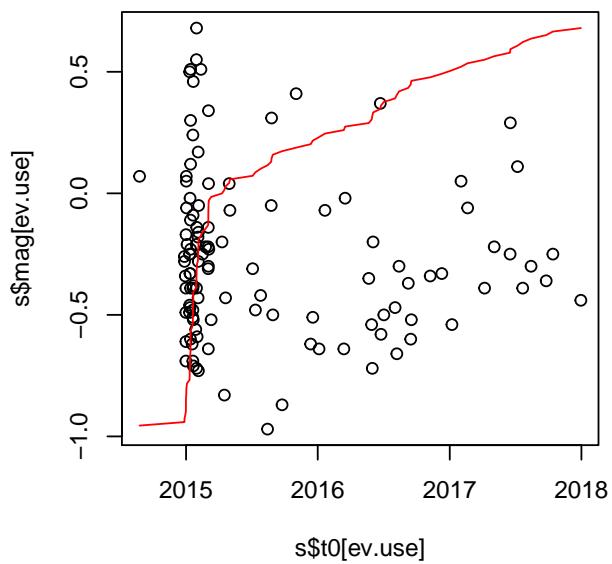
**BFMI = 1.2 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 569**



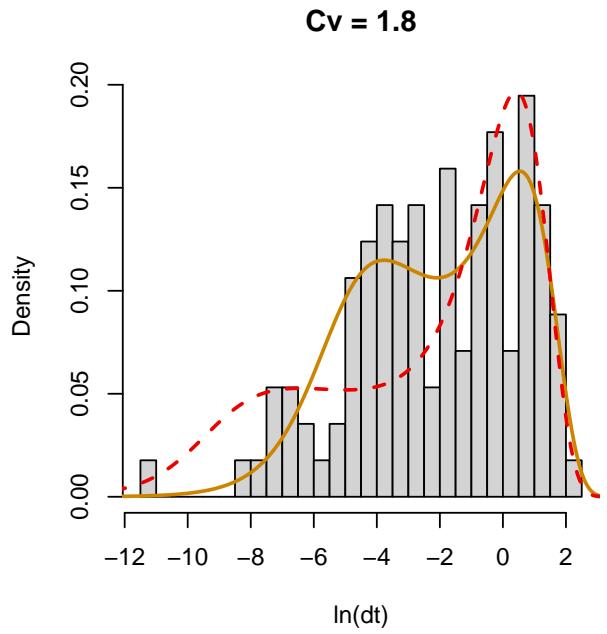
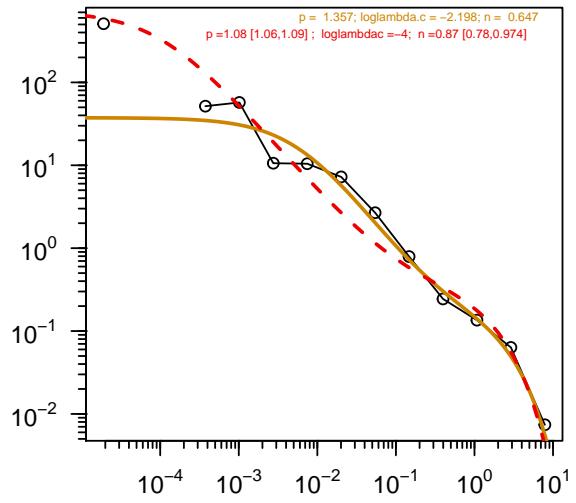
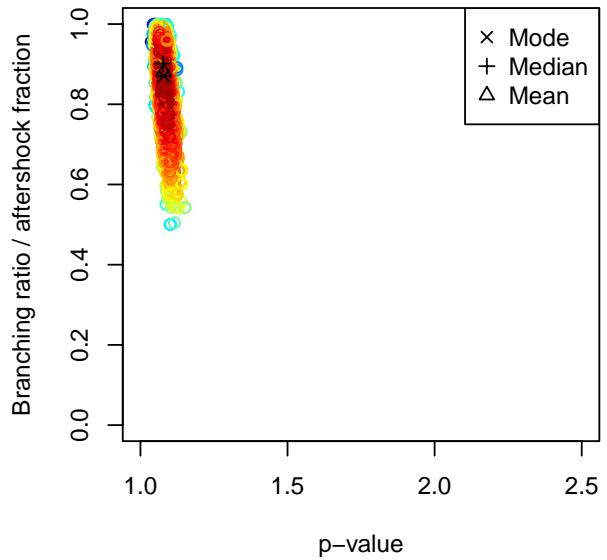
**Cv = 1.8**



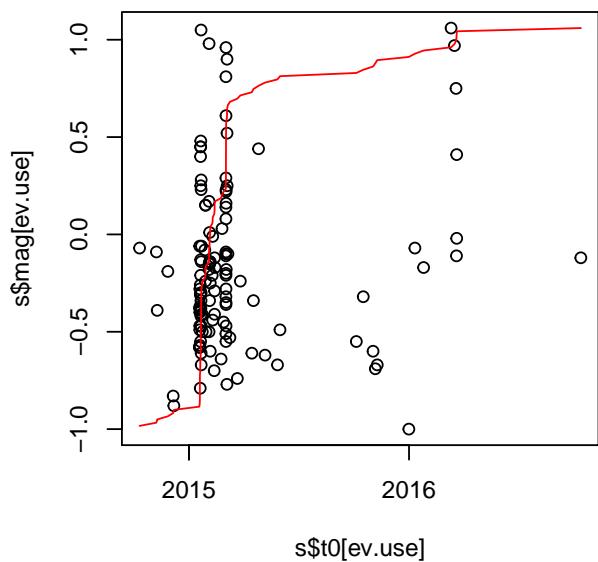
**Family 70068268 ; nev = 114**



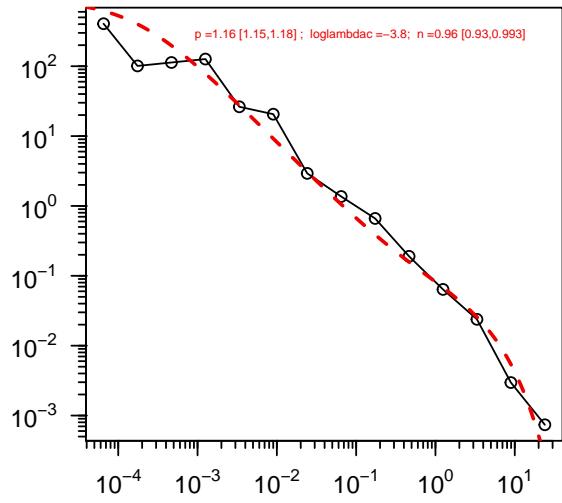
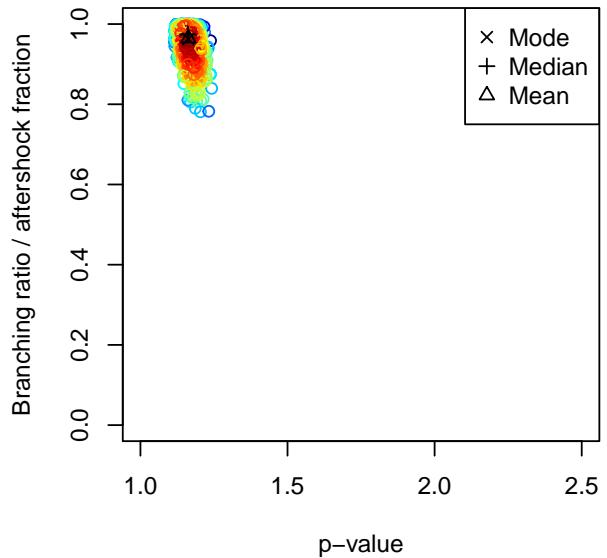
**BFMI = 0.98 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 825**



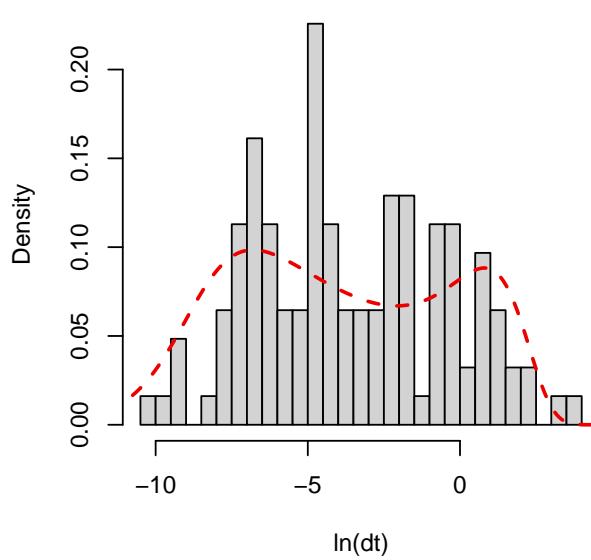
**Family 70068463 ; nev = 125**



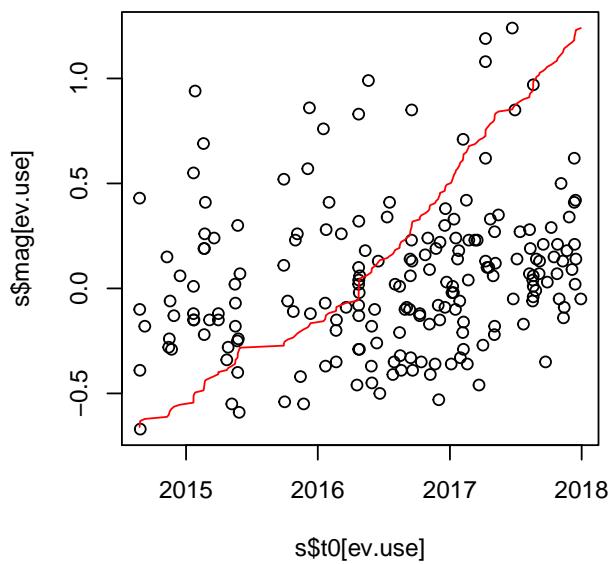
**BFMI = 1 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 1282**



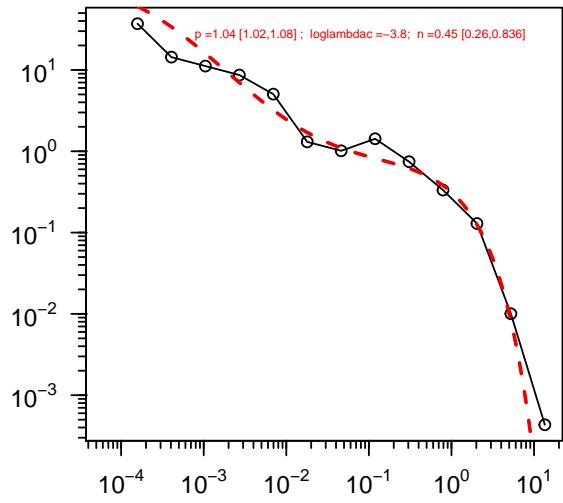
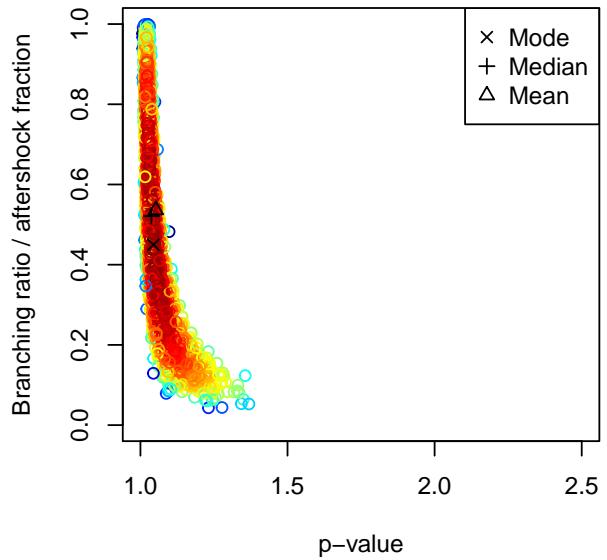
**Cv = 3.9**



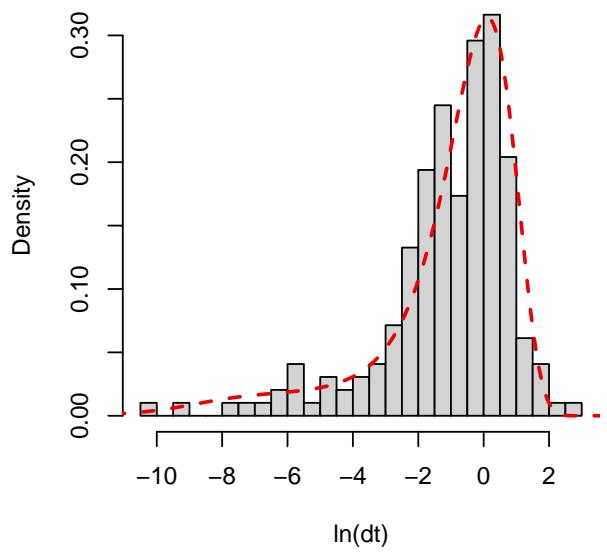
**Family 70068563 ; nev = 197**



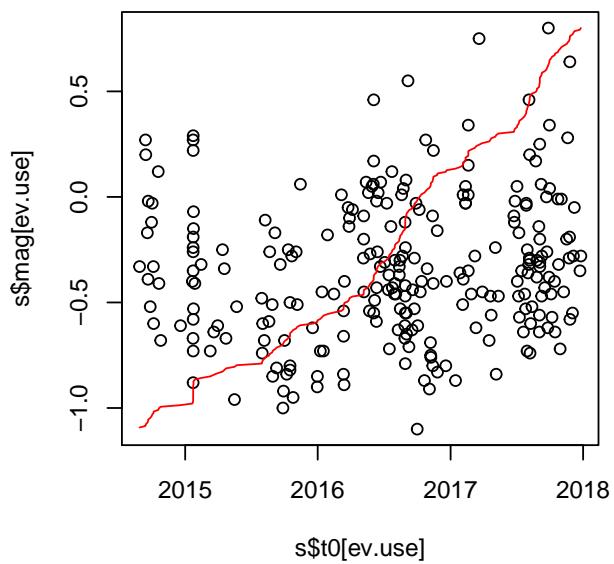
**BFMI = 1.1 ; n\_div = 0  
rhat = 1 ; n\_eff = 771**



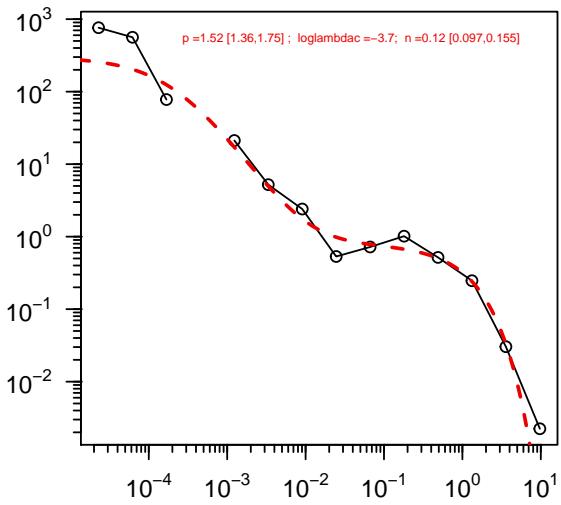
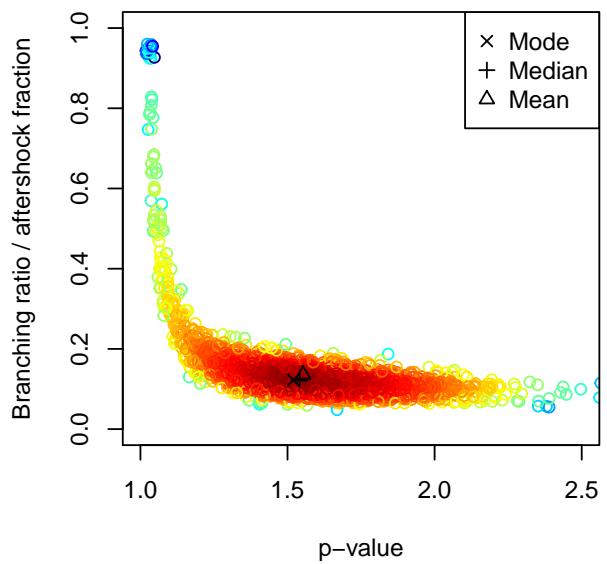
**Cv = 1.8**



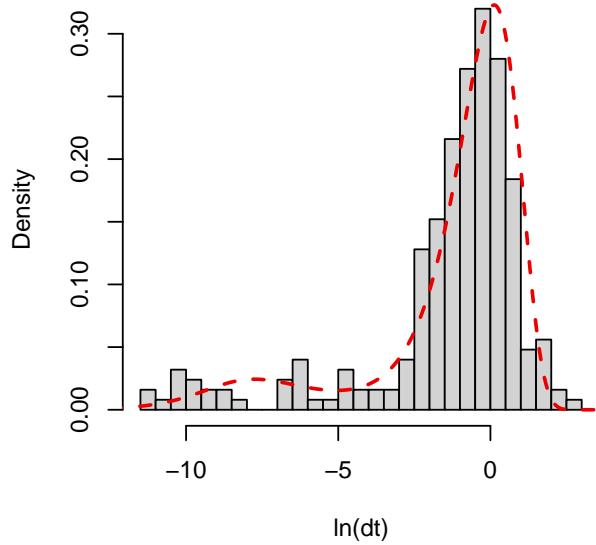
**Family 70068908 ; nev = 251**



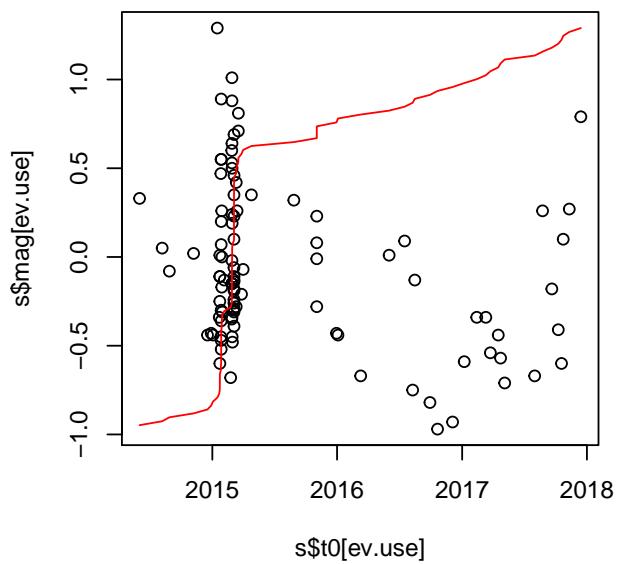
**BFMI = 0.96 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 219**



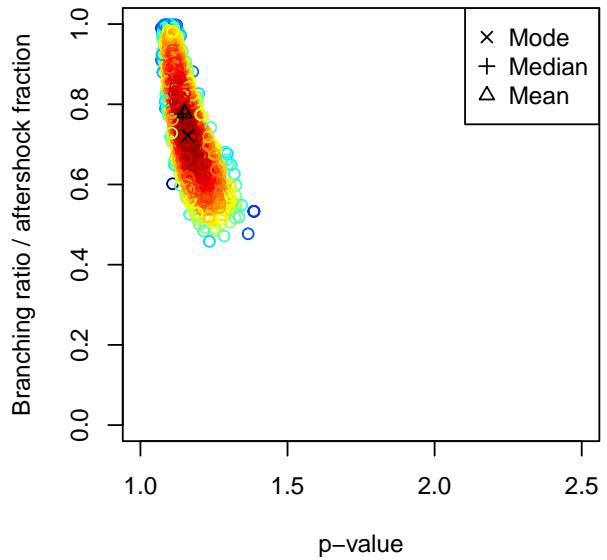
**Cv = 1.6**



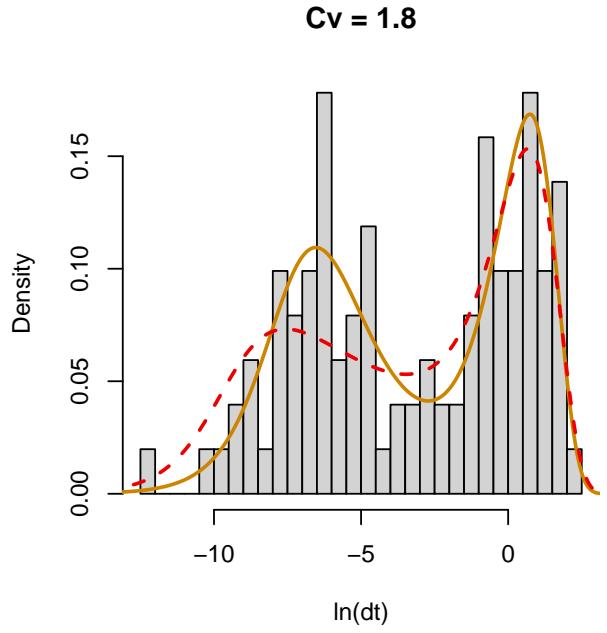
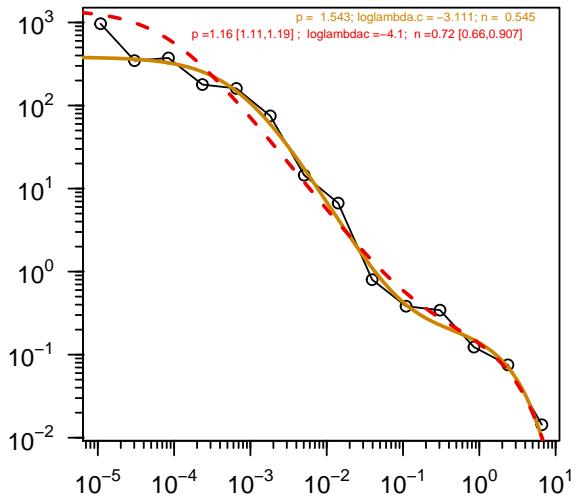
**Family 70070093 ; nev = 102**



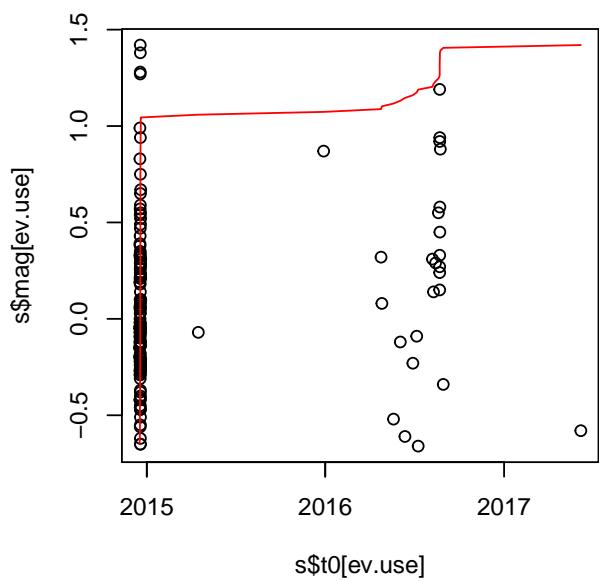
**BFMI = 0.92 ; n\_div = 0  
rhat = 1 ; n\_eff = 741**



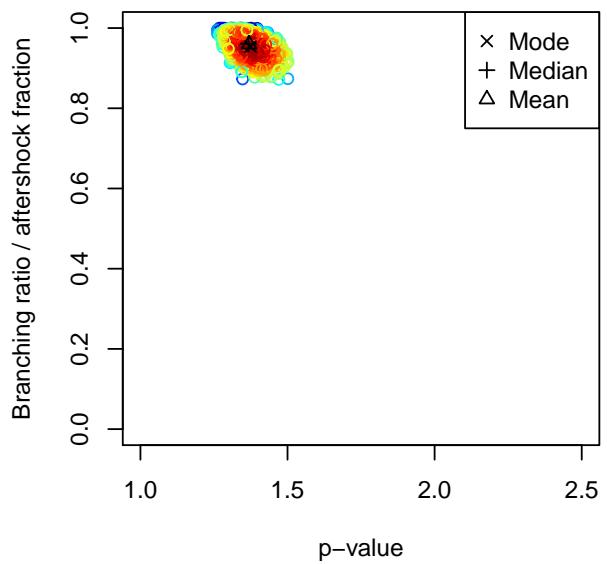
**Cv = 1.8**



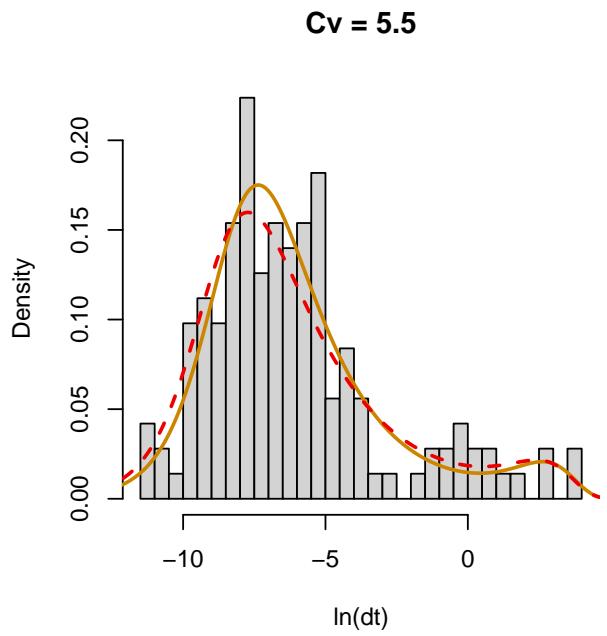
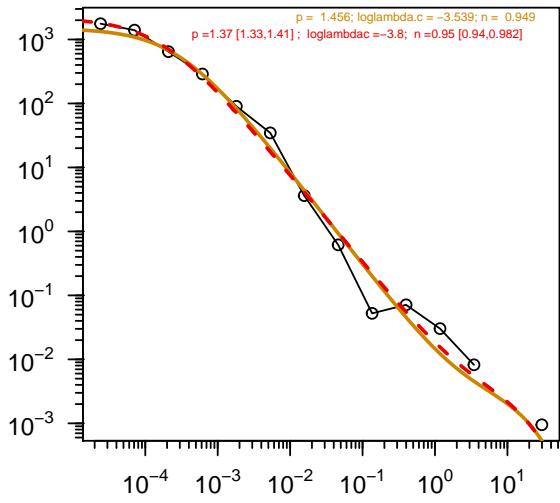
**Family 70072358 ; nev = 144**



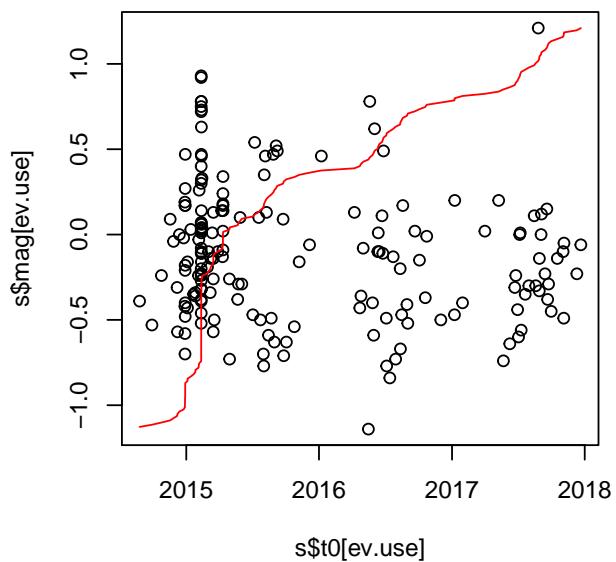
**BFMI = 0.98 ; n\_div = 0  
rhat = 1 ; n\_eff = 899**



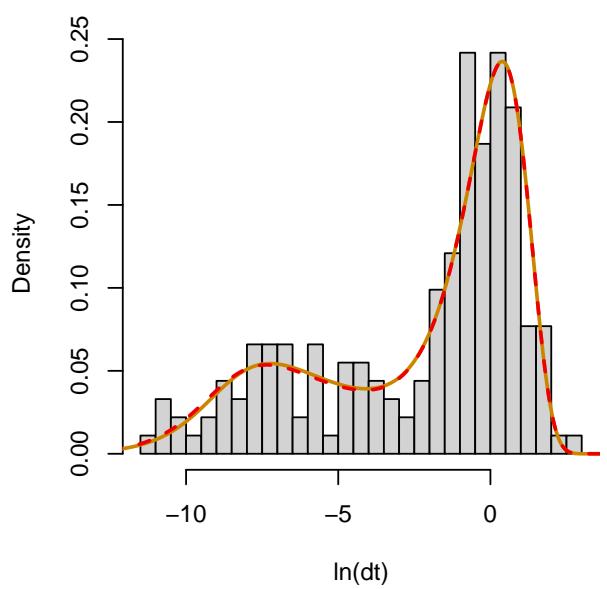
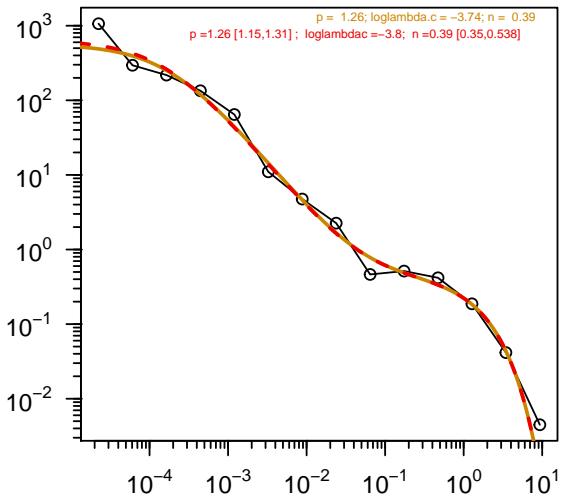
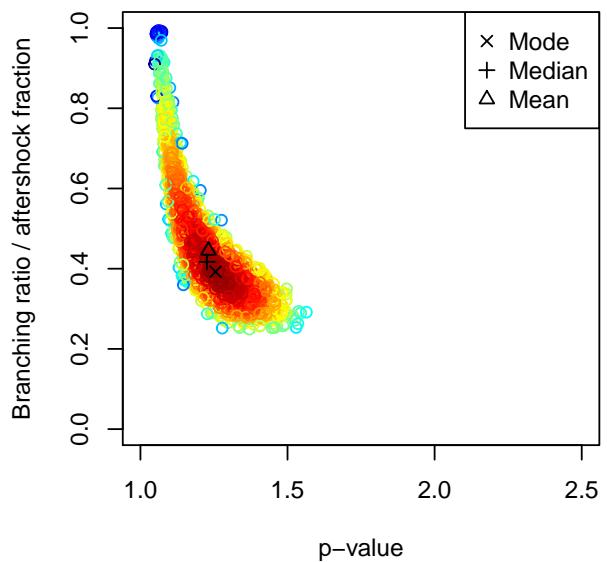
**Cv = 5.5**



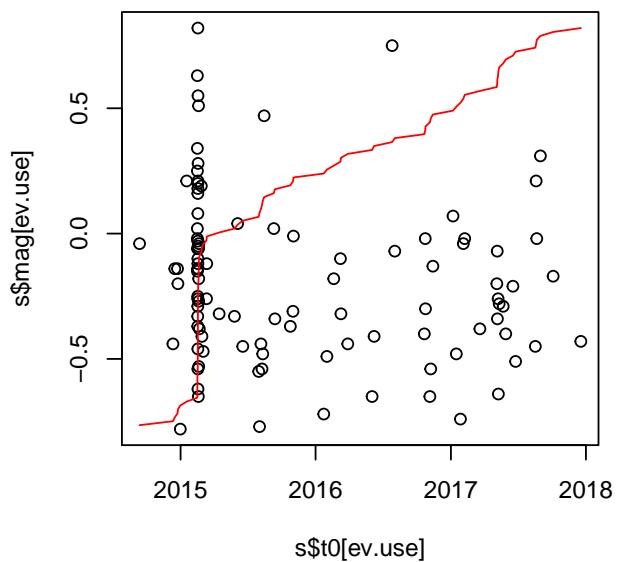
**Family 70074968 ; nev = 183**



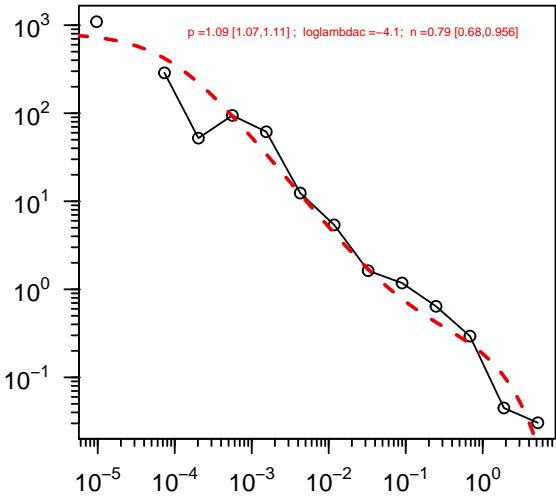
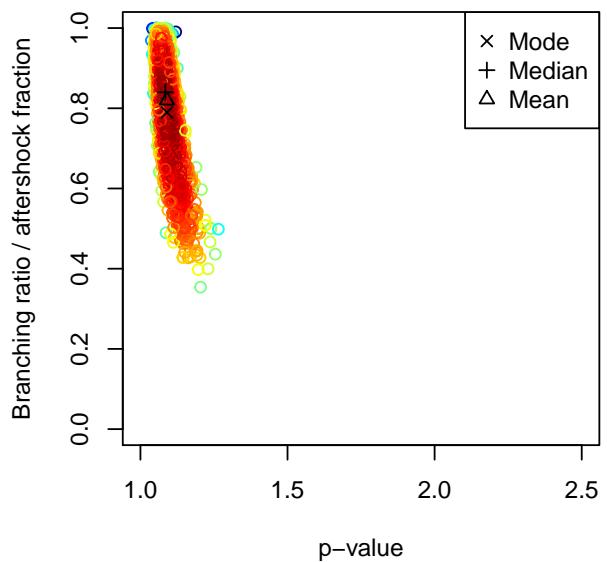
**BFMI = 1.1 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 460**



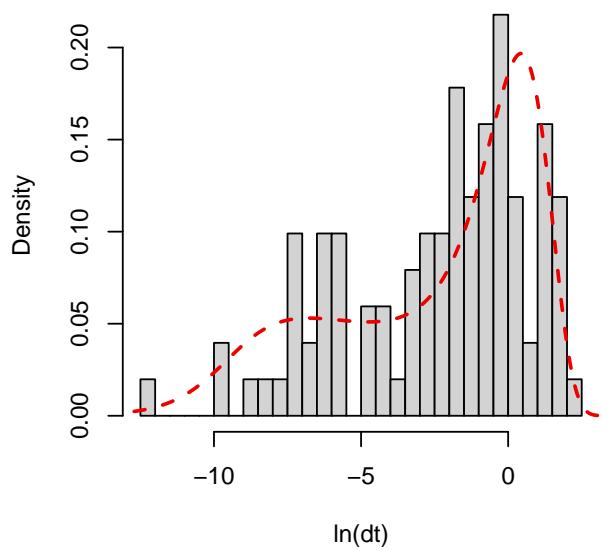
**Family 70076378 ; nev = 102**



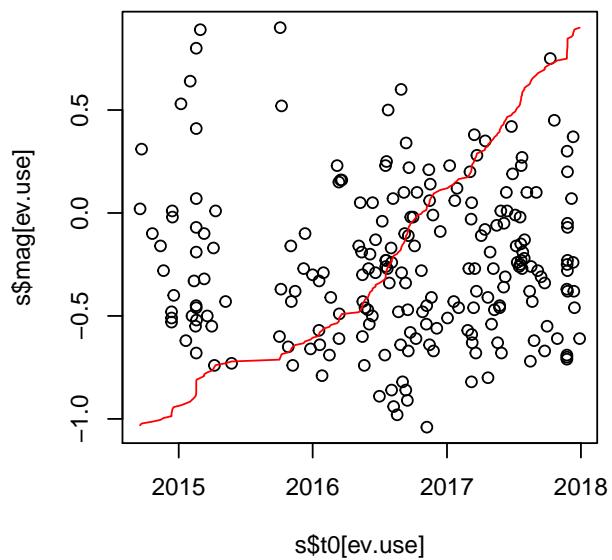
**BFMI = 1 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 849**



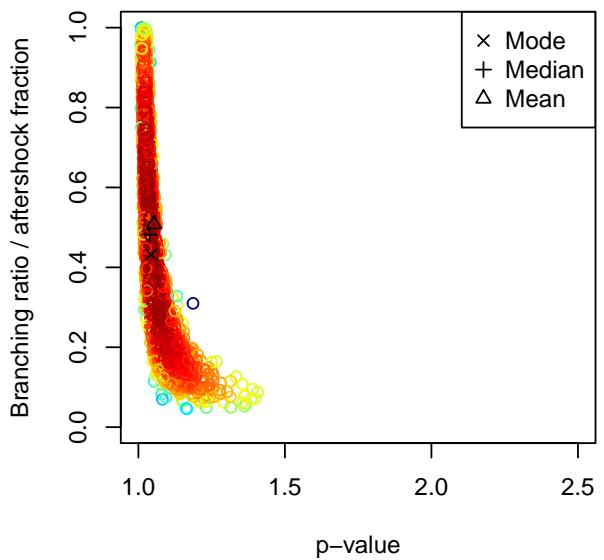
**Cv = 1.7**



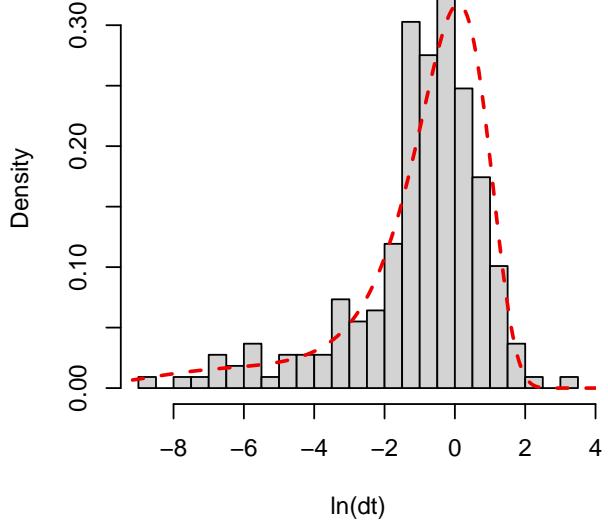
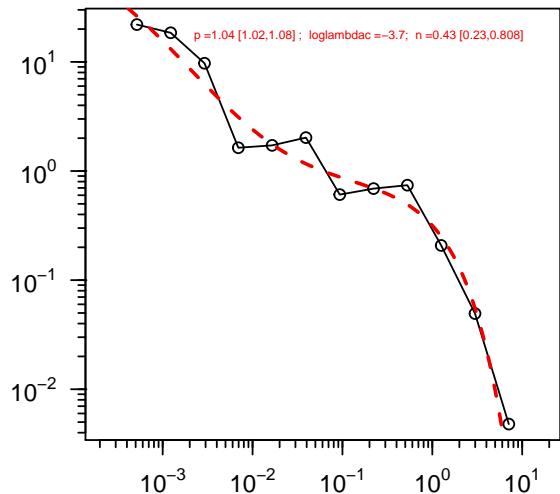
**Family 70076588 ; nev = 219**



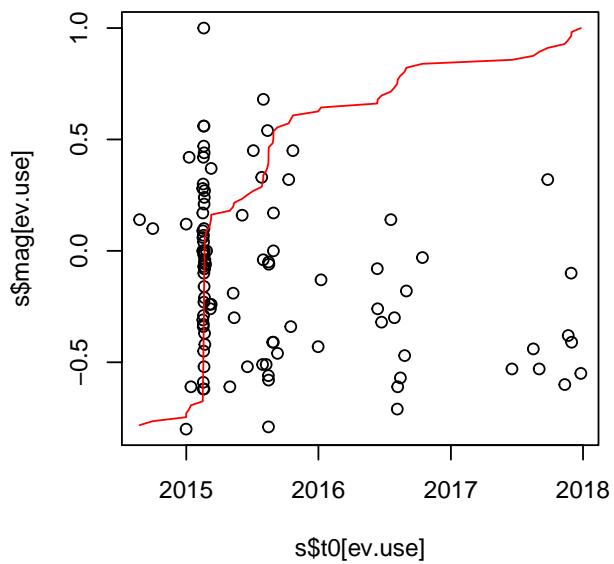
**BFMI = 1 ; n\_div = 0  
rhat = 1 ; n\_eff = 792**



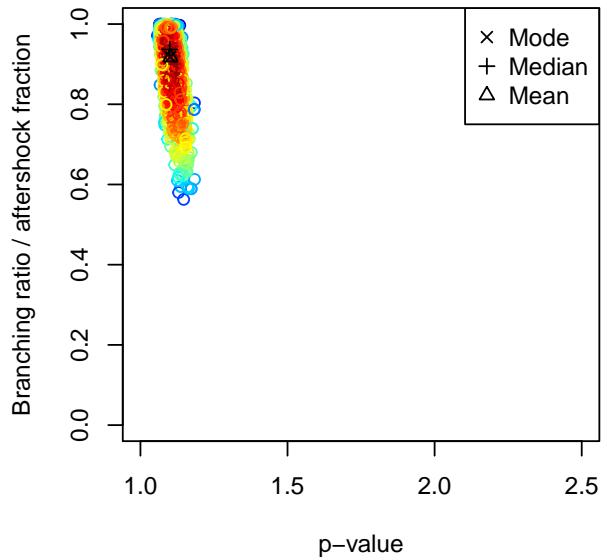
**Cv = 1.9**



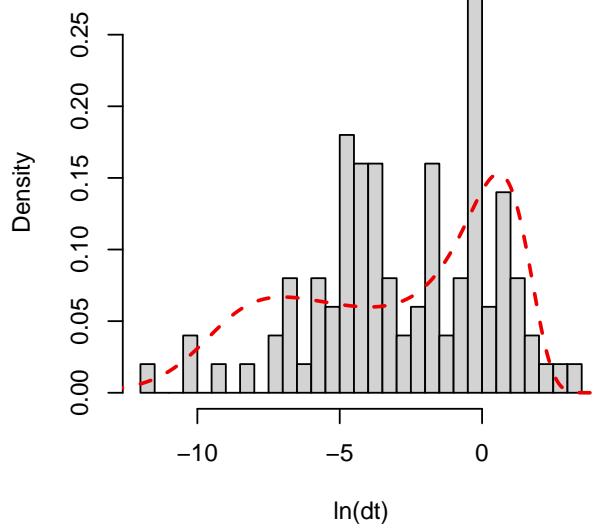
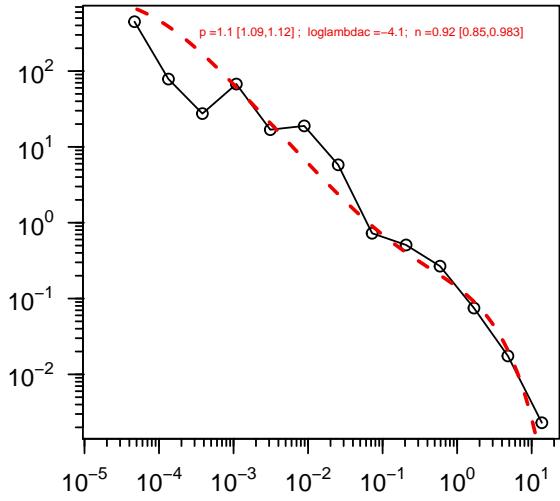
**Family 70076658 ; nev = 101**



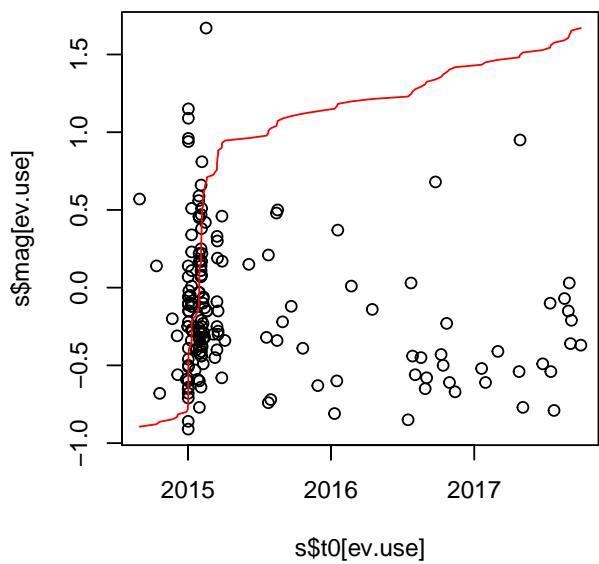
**BFMI = 1.1 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 663**



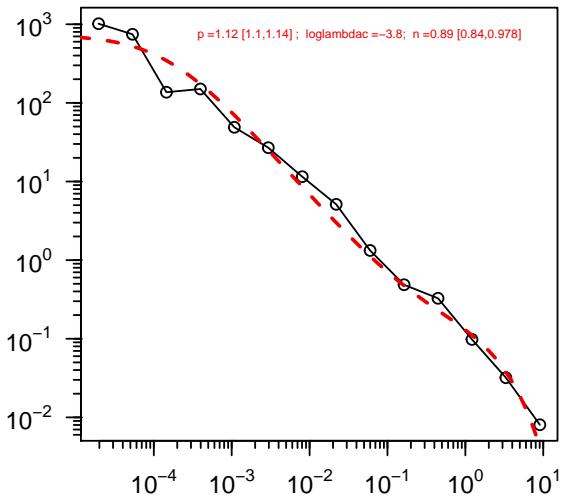
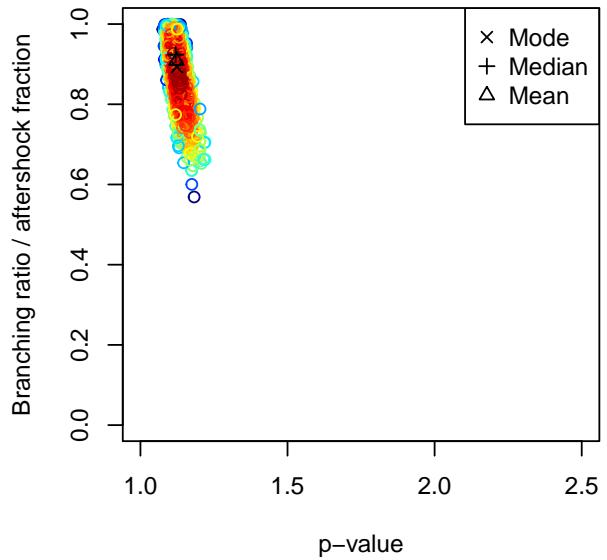
**Cv = 2.6**



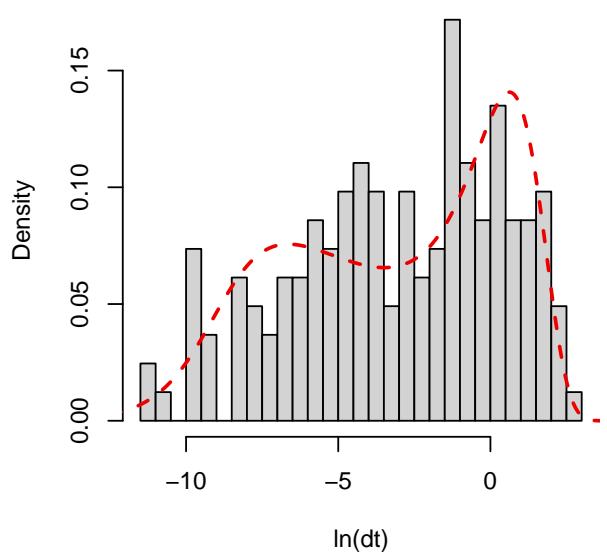
**Family 70076778 ; nev = 164**



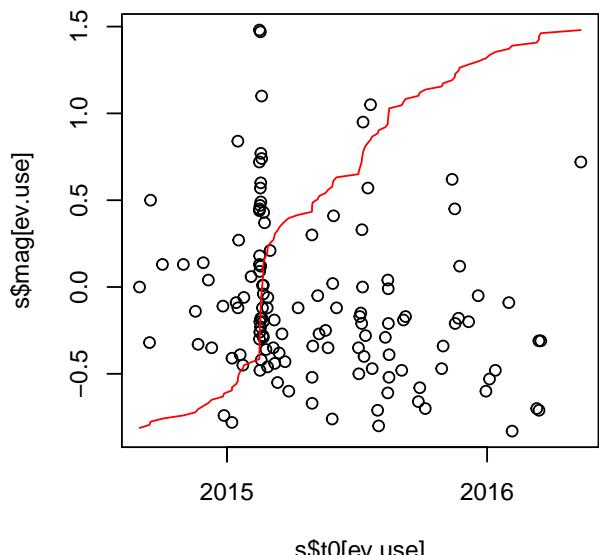
**BFMI = 1.1 ; n\_div = 0  
rhat = 1 ; n\_eff = 742**



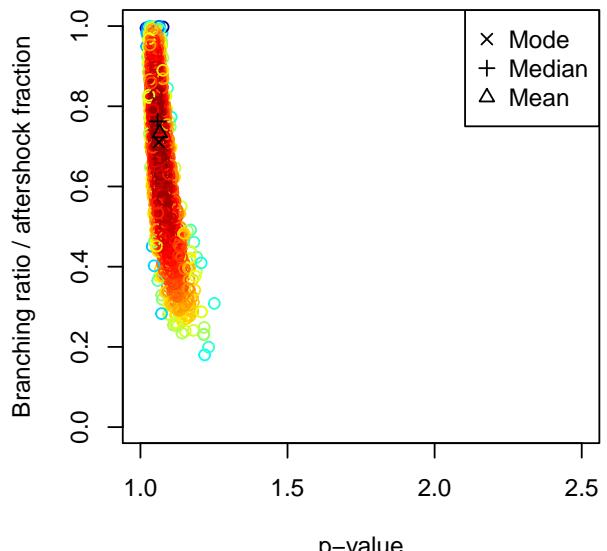
**Cv = 2.1**



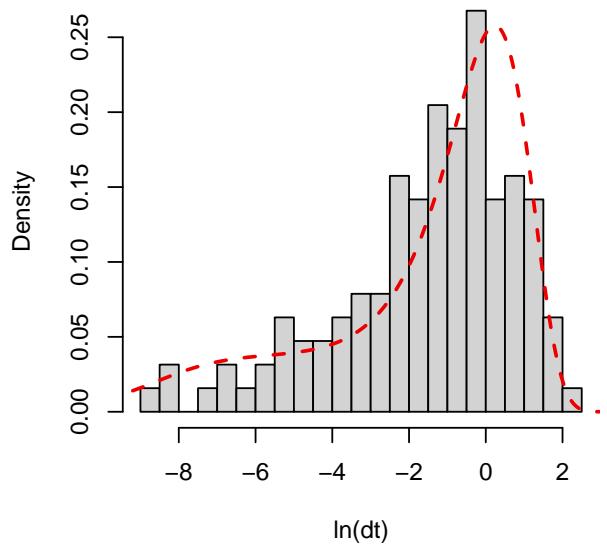
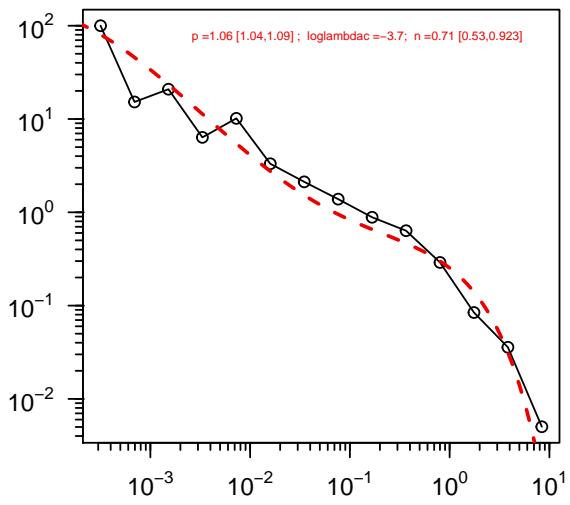
**Family 70076973 ; nev = 128**



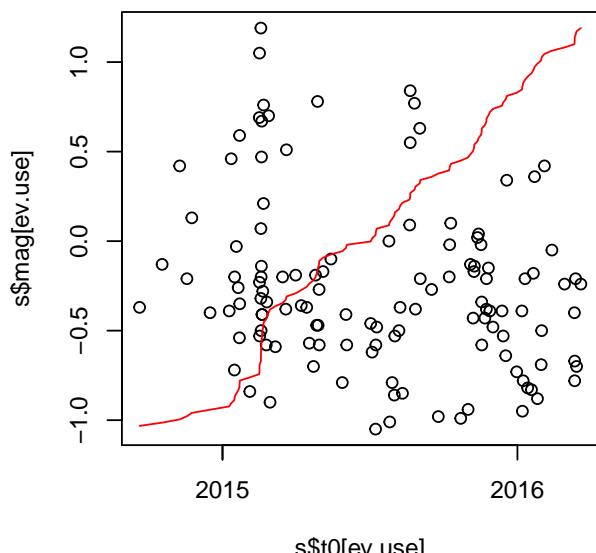
**BFMI = 0.96 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 607**



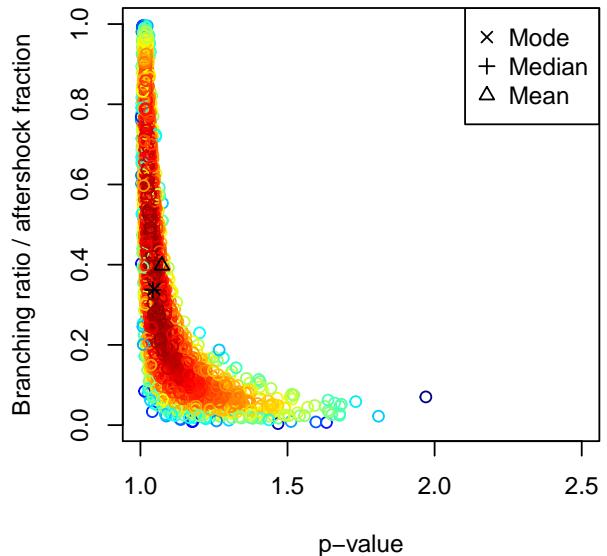
**Cv = 1.7**



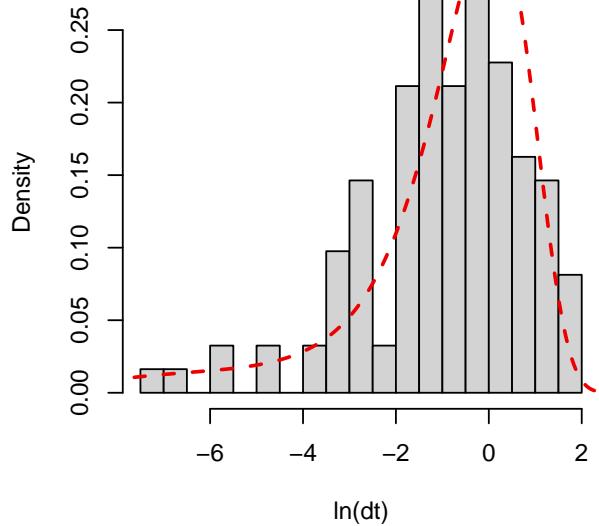
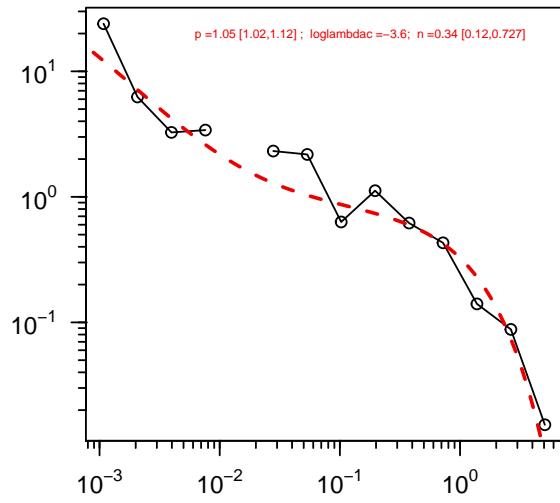
**Family 70077163 ; nev = 124**



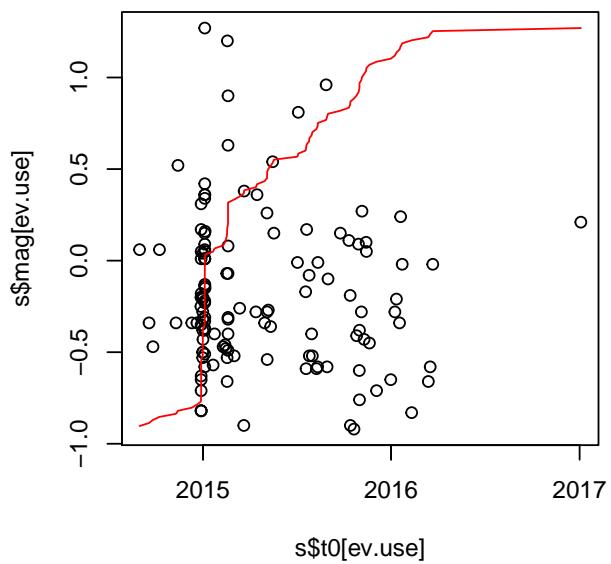
**BFMI = 0.97 ; n\_div = 0  
rhat = 1 ; n\_eff = 749**



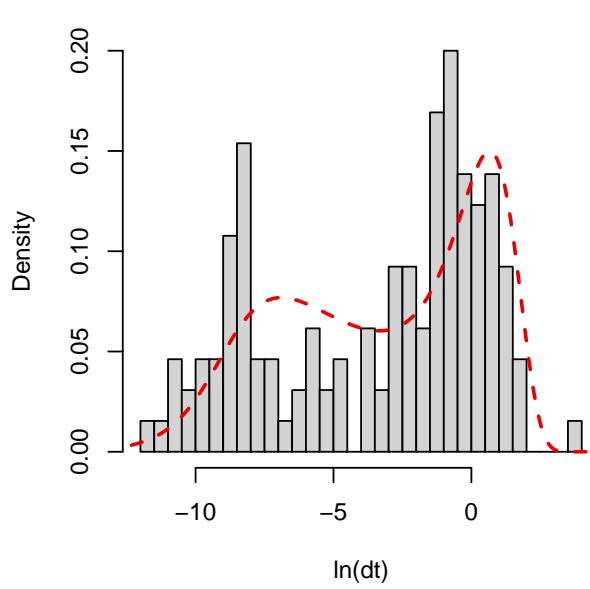
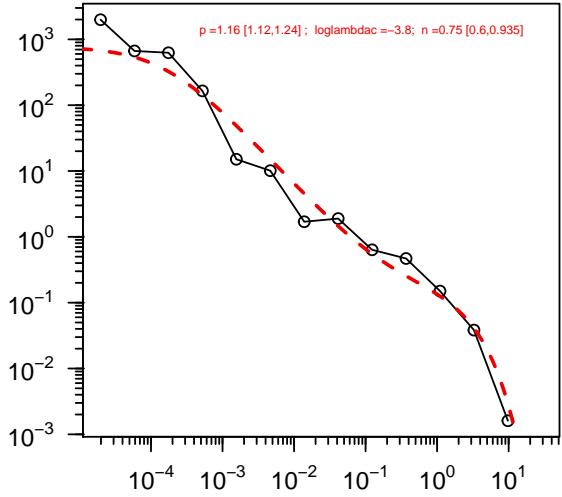
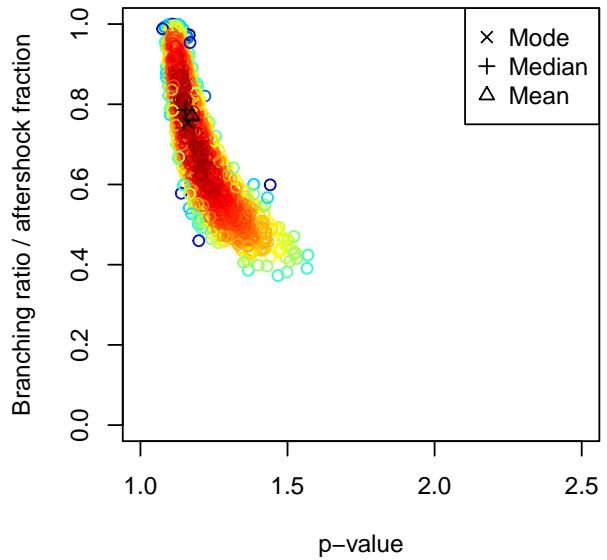
**Cv = 1.3**



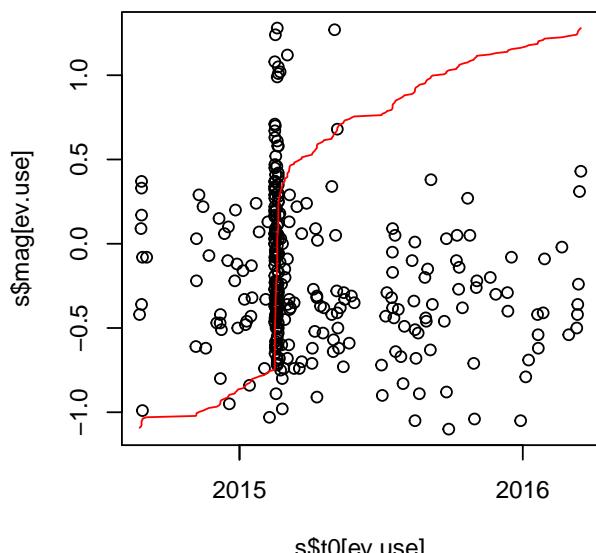
**Family 70077193 ; nev = 131**



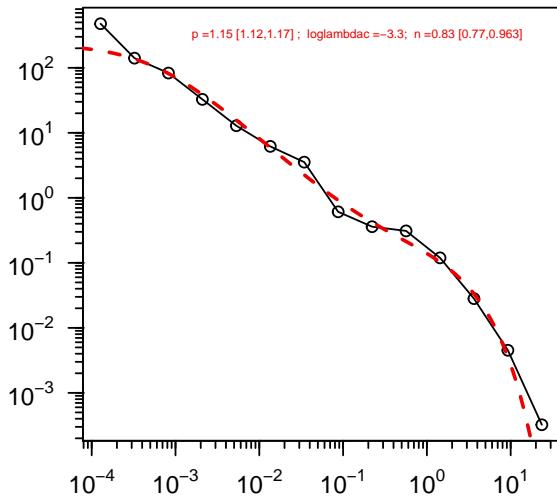
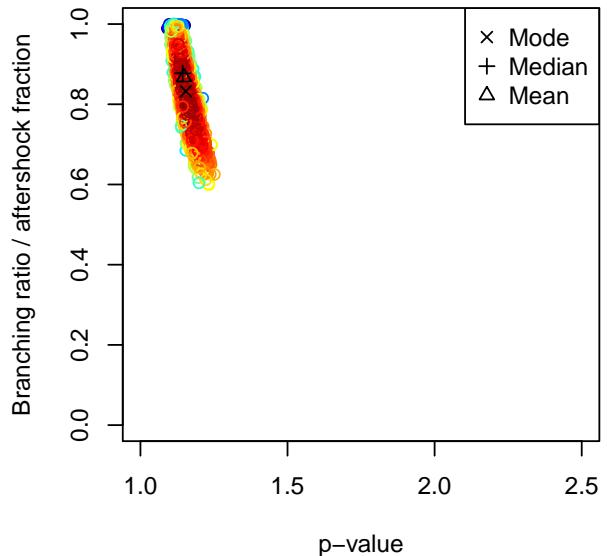
**BFMI = 0.89 ; n\_div = 0**  
**rhat = 1 ; n\_eff = 607**



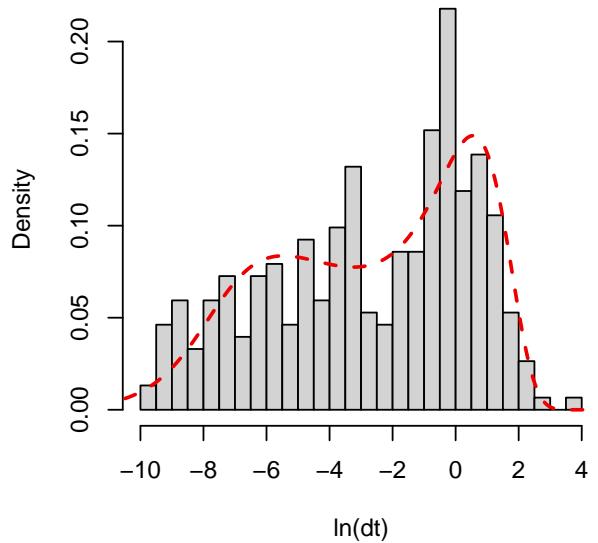
**Family 70077198 ; nev = 304**



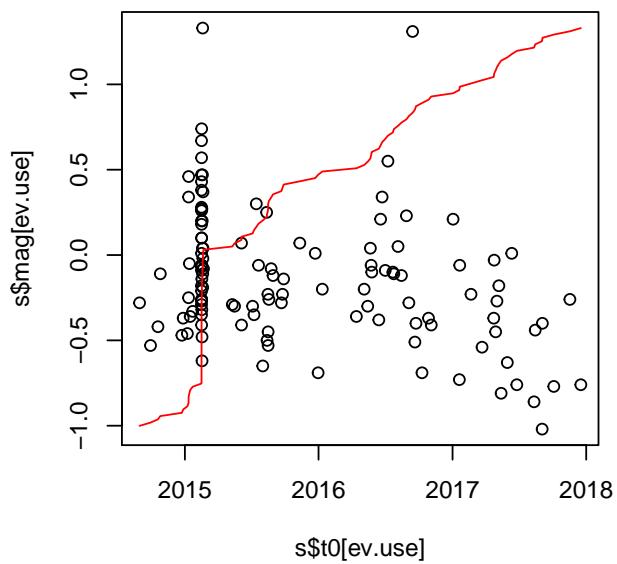
**BFMI = 1 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 735**



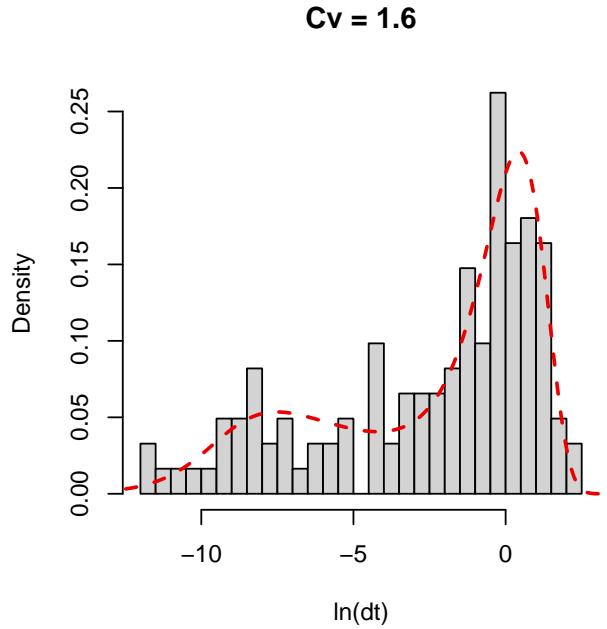
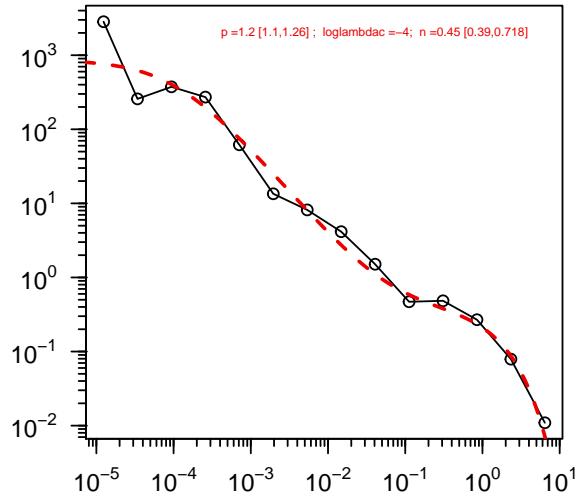
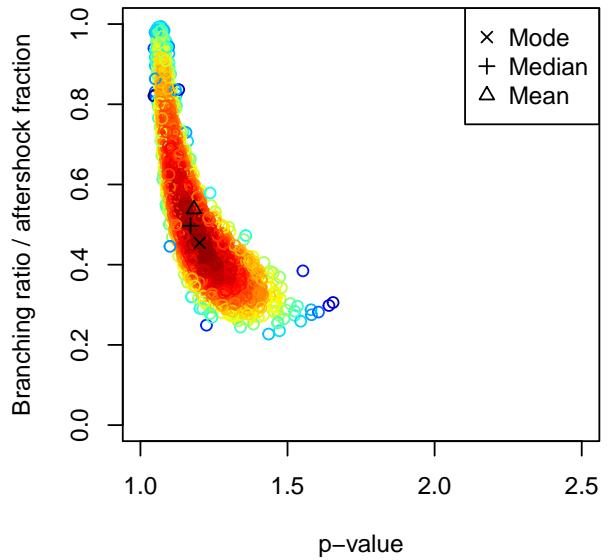
**Cv = 2.7**



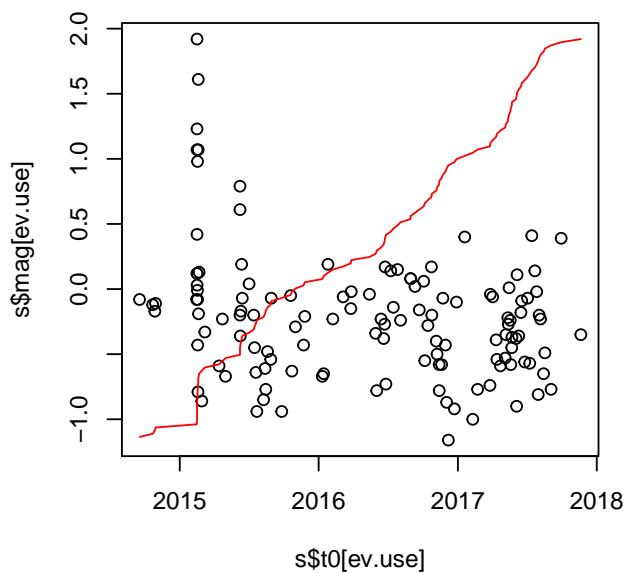
**Family 70077208 ; nev = 123**



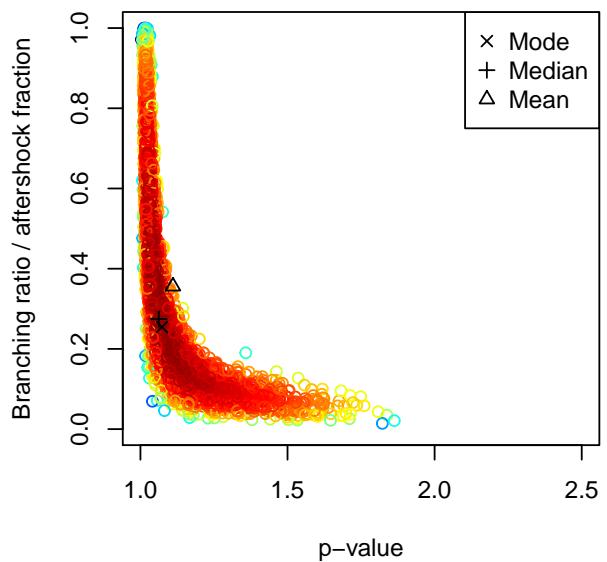
**BFMI = 0.95 ; n\_div = 0  
rhat = 1 ; n\_eff = 650**



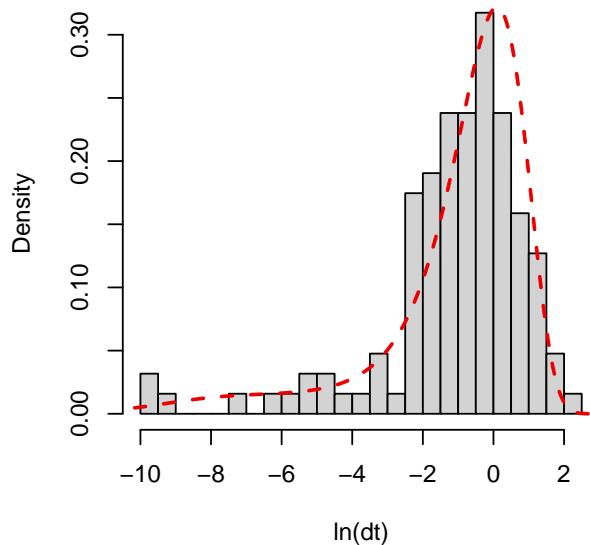
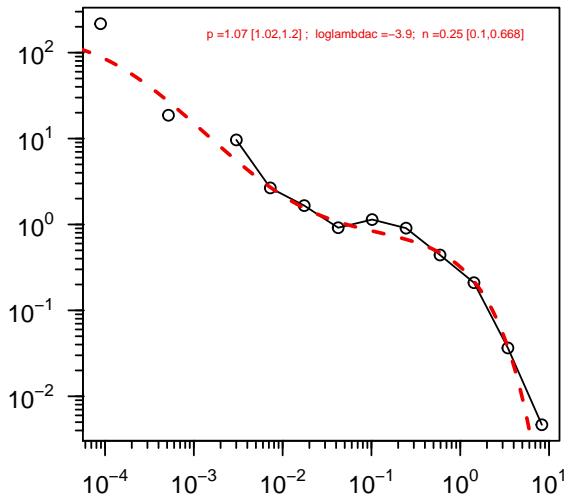
**Family 70077238 ; nev = 127**



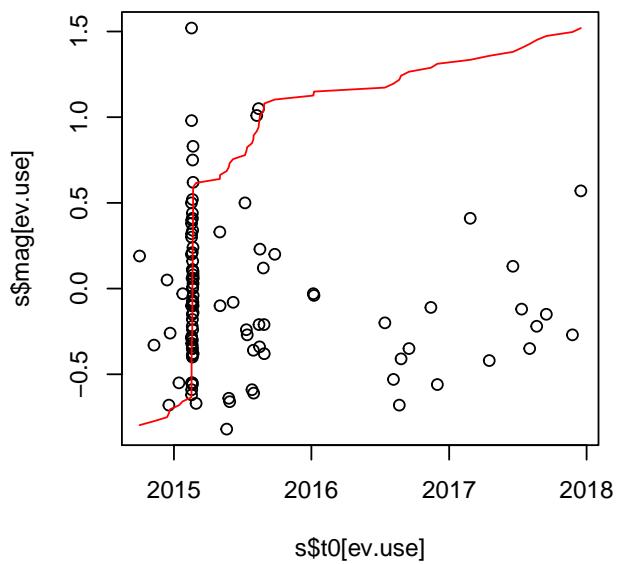
**BFMI = 1 ; n\_div = 0  
rhat = 1 ; n\_eff = 632**



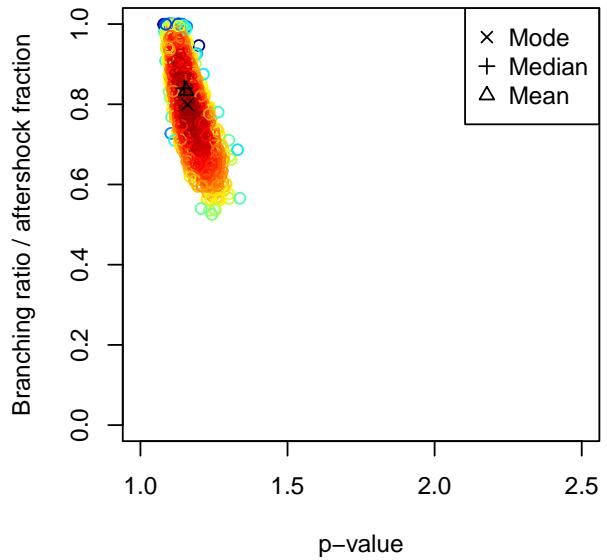
**Cv = 1.5**



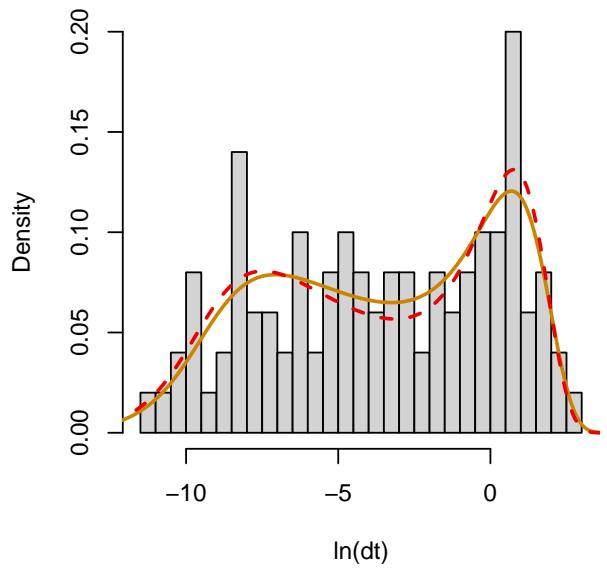
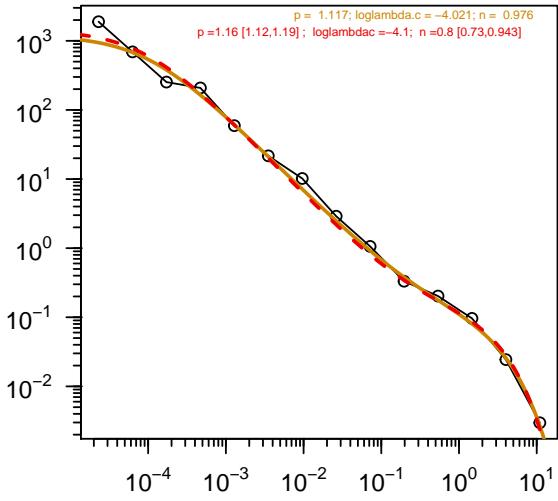
**Family 70077433 ; nev = 101**



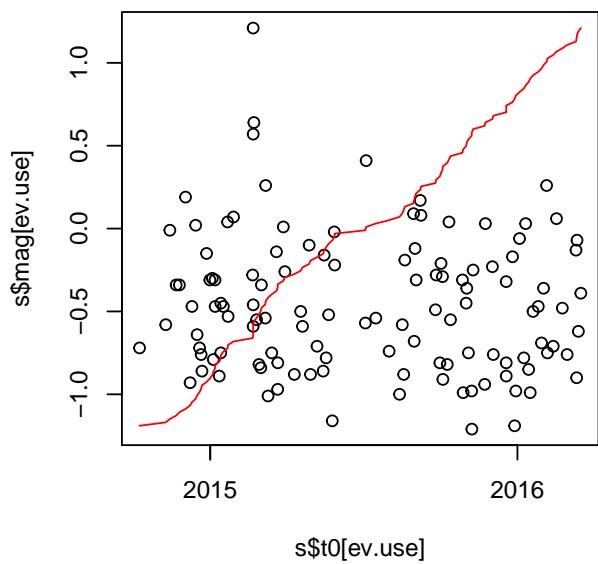
**BFMI = 1.1 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 537**



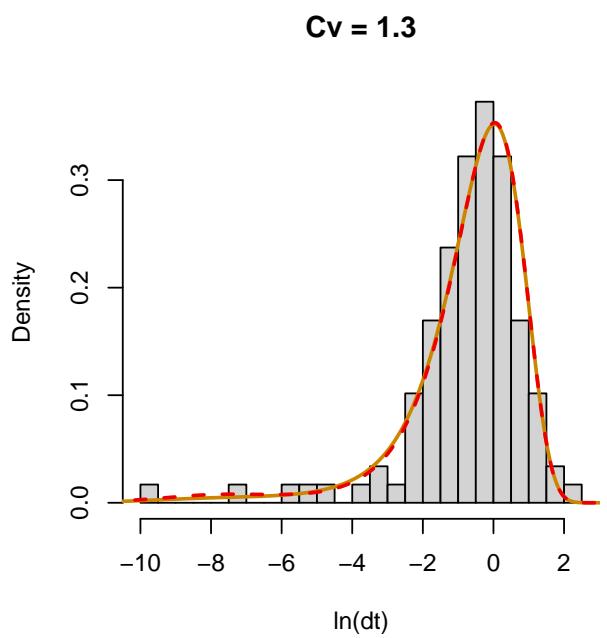
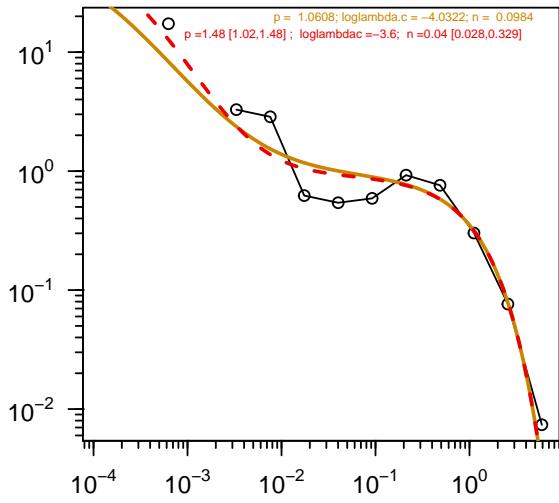
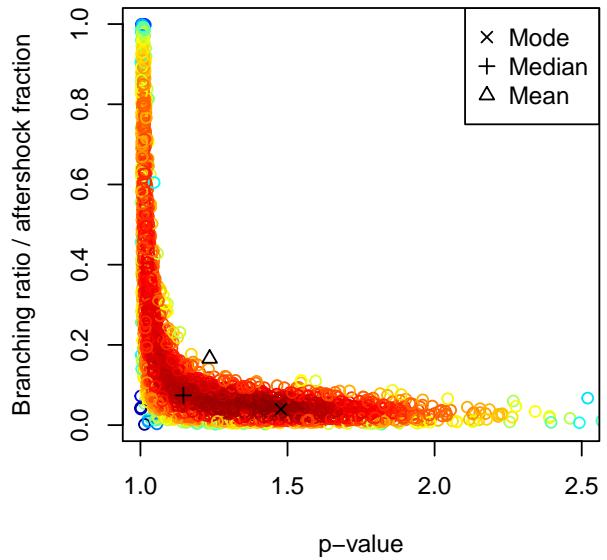
**Cv = 2.3**



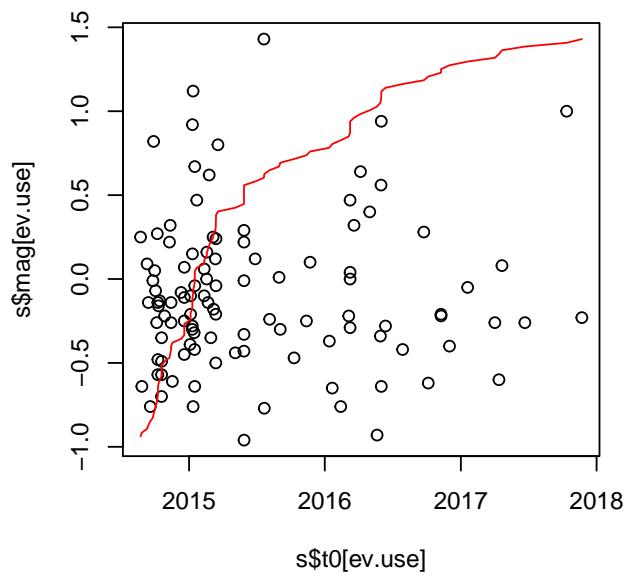
**Family 70078193 ; nev = 119**



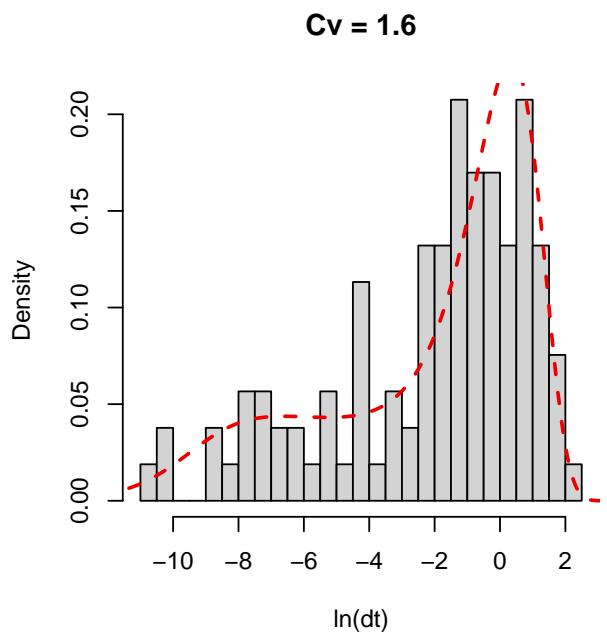
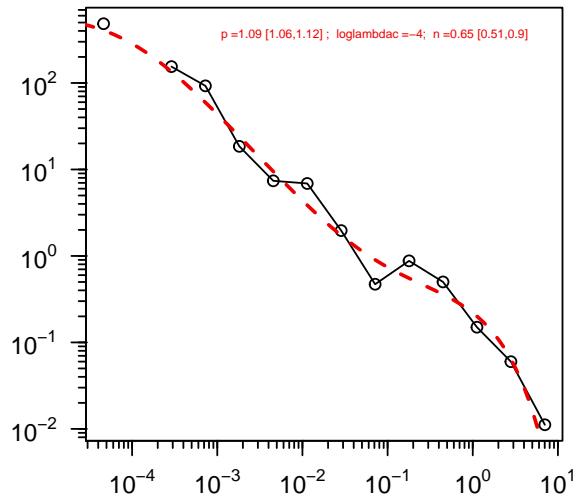
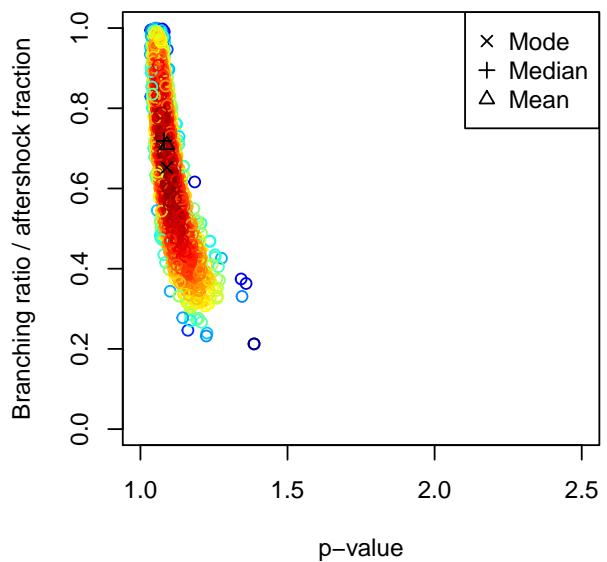
**BFMI = 0.96 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 451**



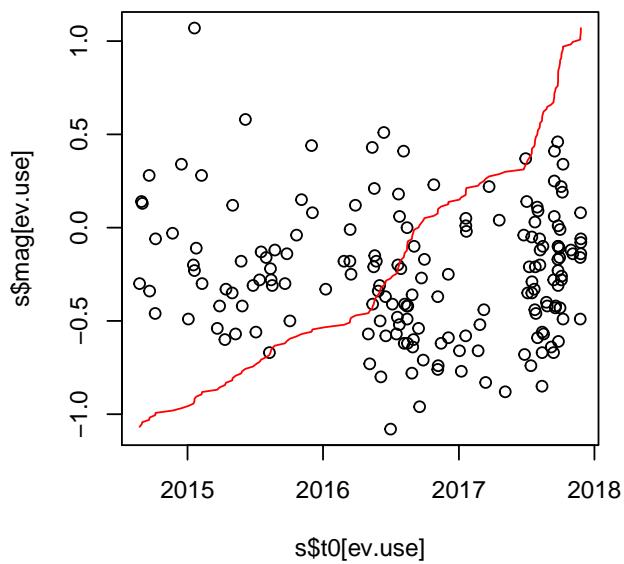
**Family 70078333 ; nev = 107**



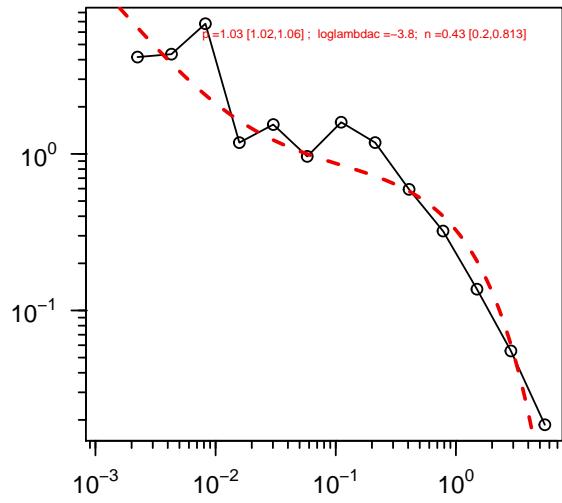
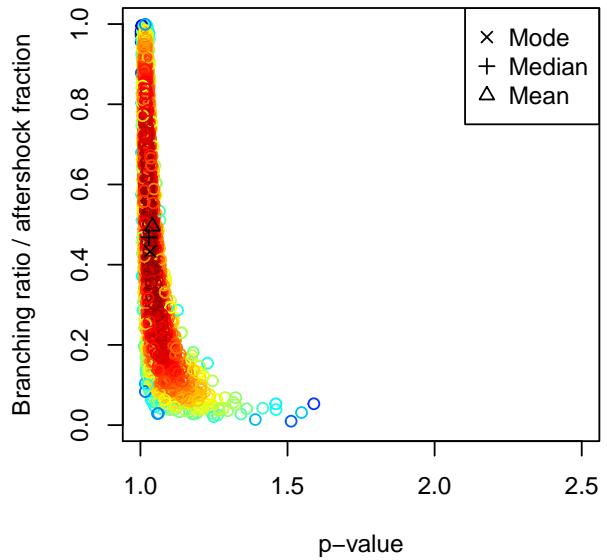
**BFMI = 1 ; n\_div = 0  
rhat = 1 ; n\_eff = 584**



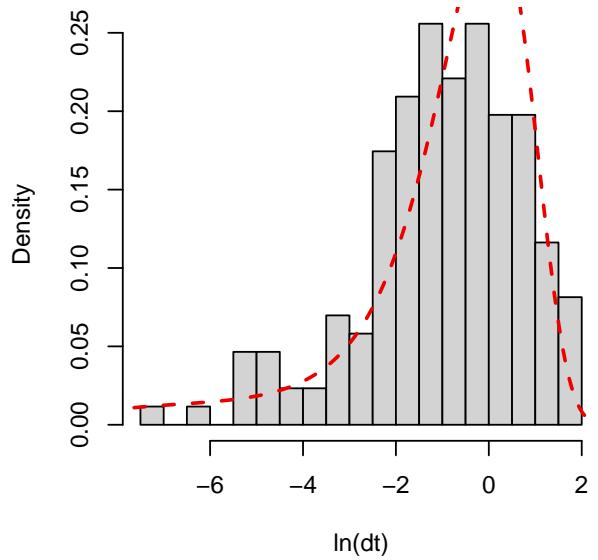
**Family 70078773 ; nev = 173**



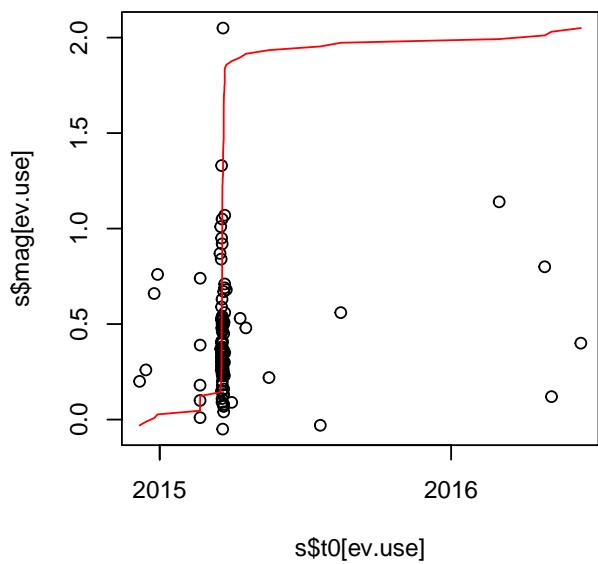
**BFMI = 1.1 ; n\_div = 0  
rhat = 1 ; n\_eff = 783**



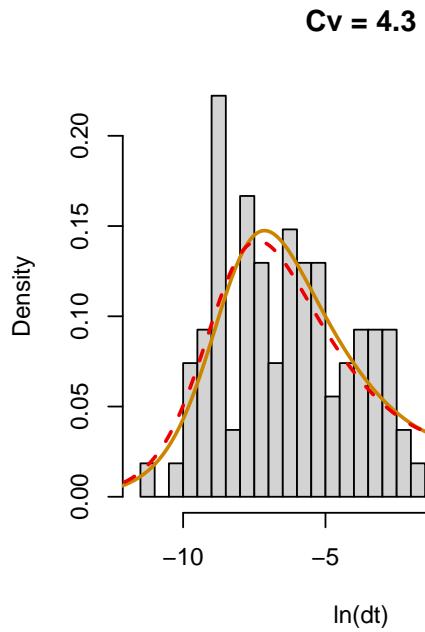
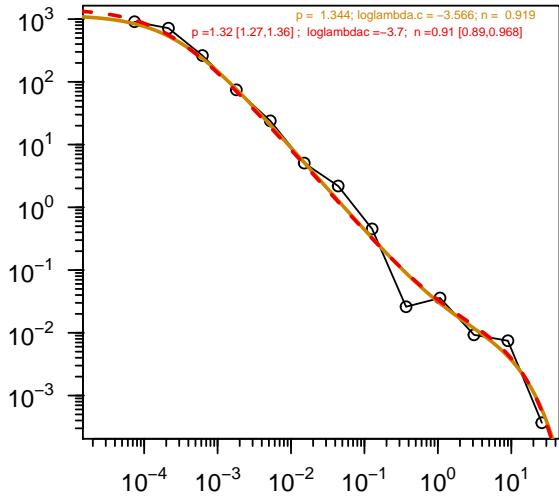
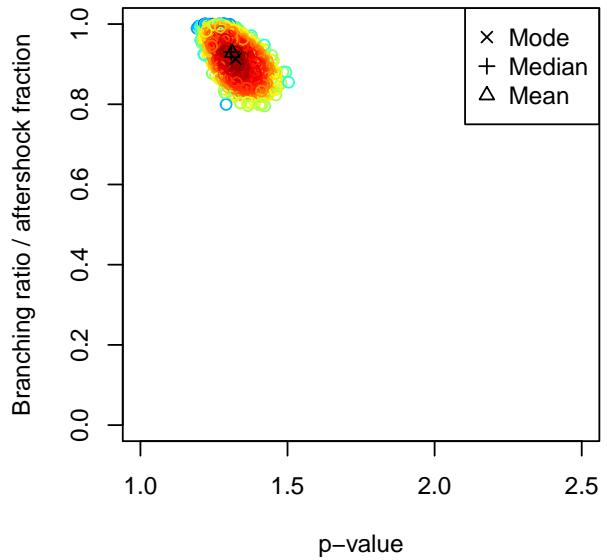
**Cv = 1.4**



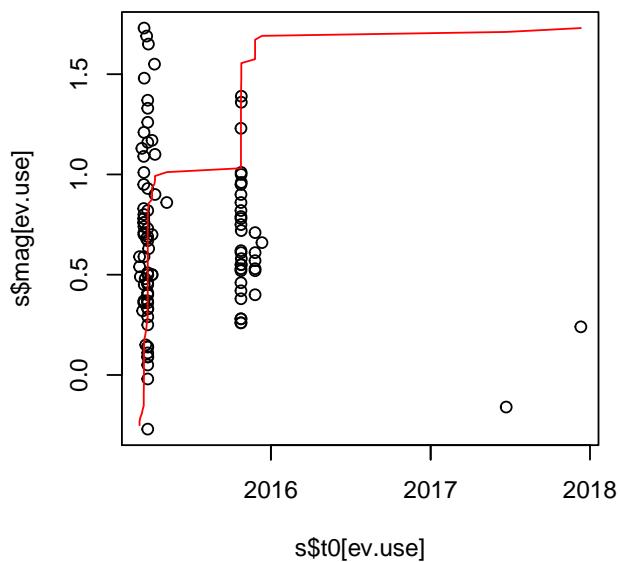
**Family 70085543 ; nev = 109**



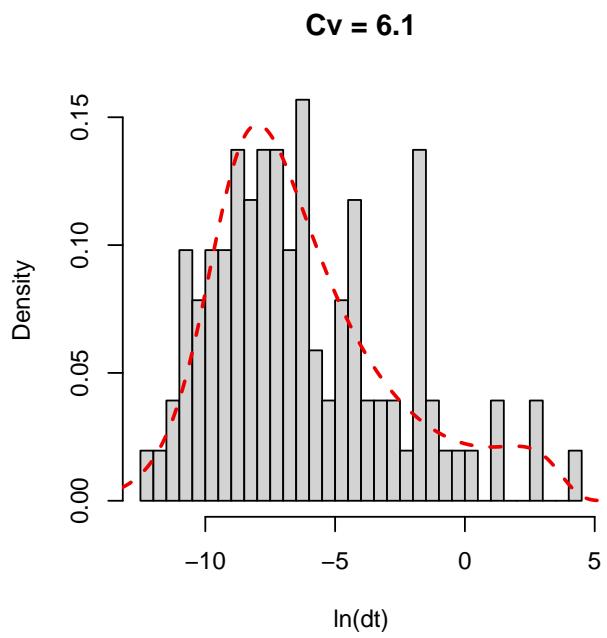
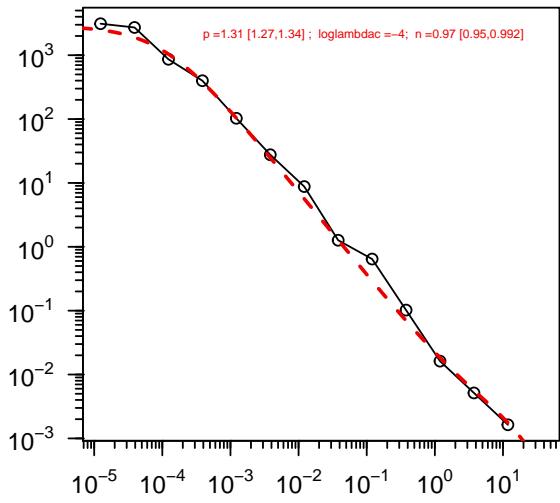
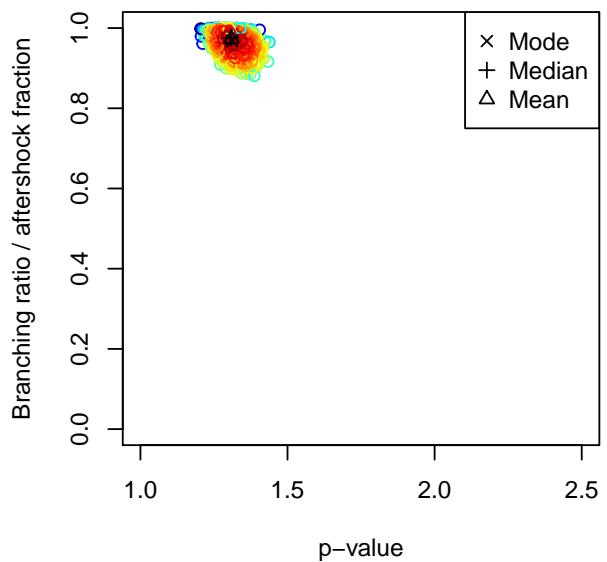
**BFMI = 1.1 ; n\_div = 0  
rhat = 1 ; n\_eff = 870**



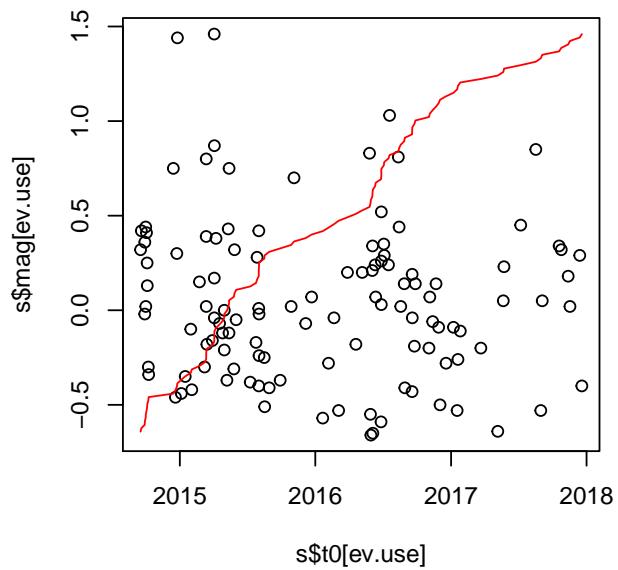
**Family 70087383 ; nev = 103**



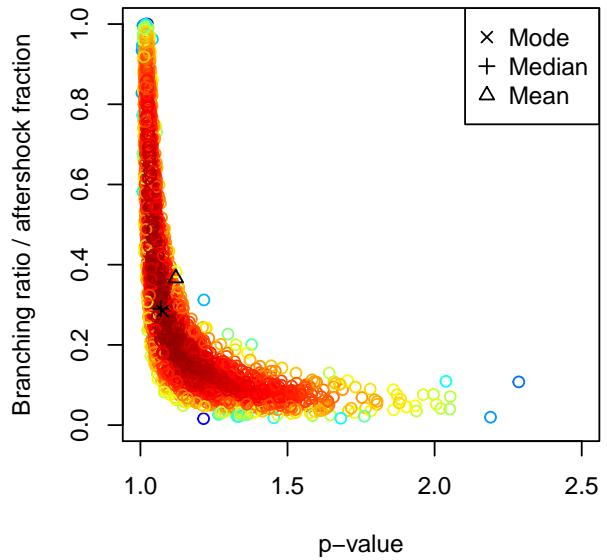
**BFMI = 1.1 ; n\_div = 0  
rhat = 1 ; n\_eff = 1203**



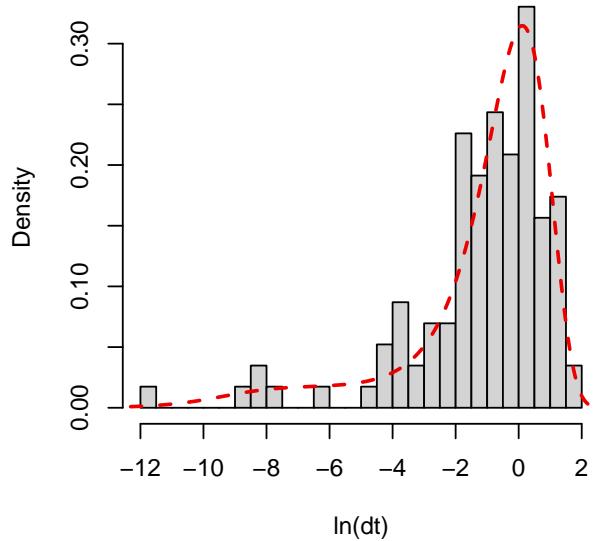
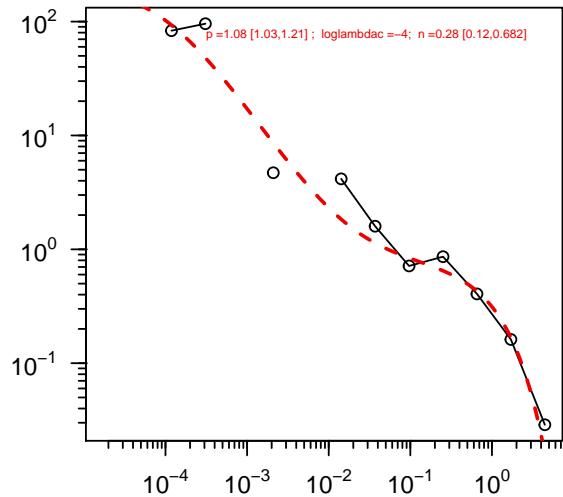
**Family 70089778 ; nev = 116**



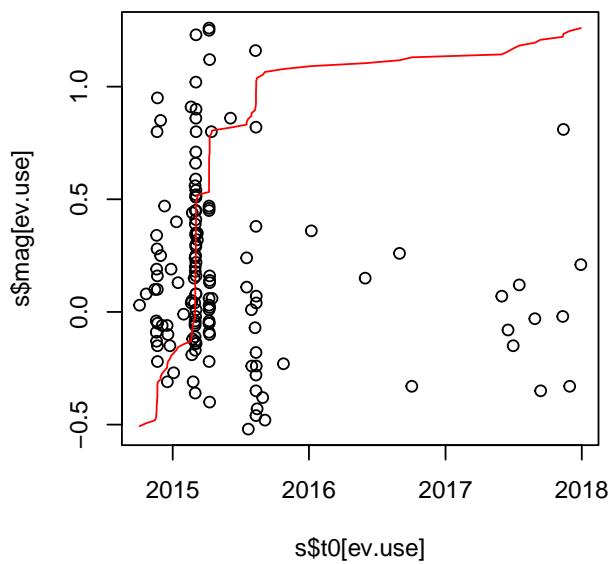
**BFMI = 1.1 ; n\_div = 0  
rhat = 1 ; n\_eff = 755**



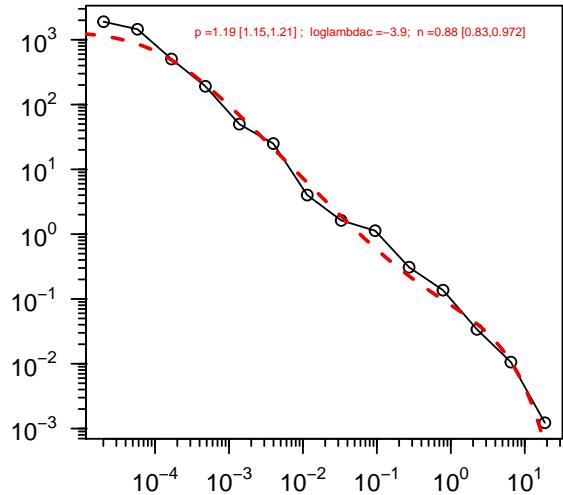
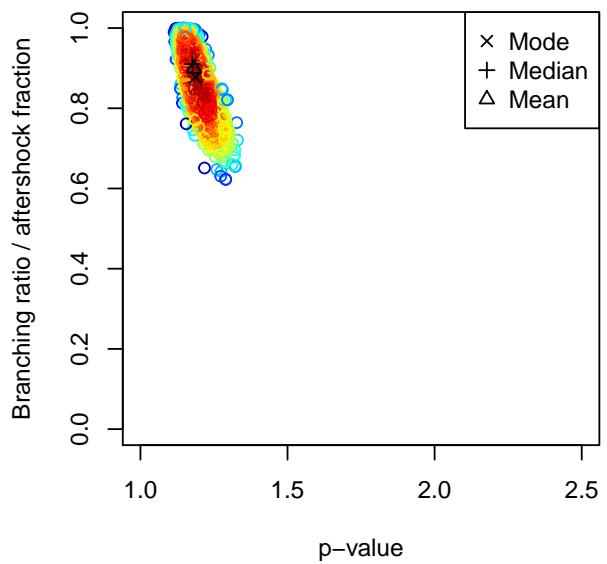
**Cv = 1.2**



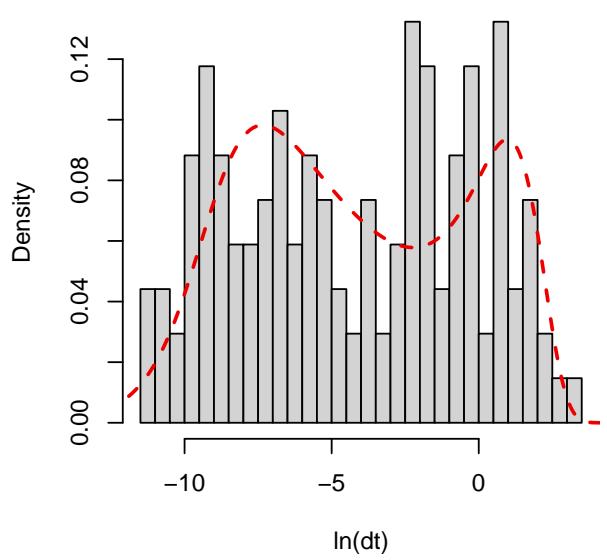
**Family 70091408 ; nev = 137**



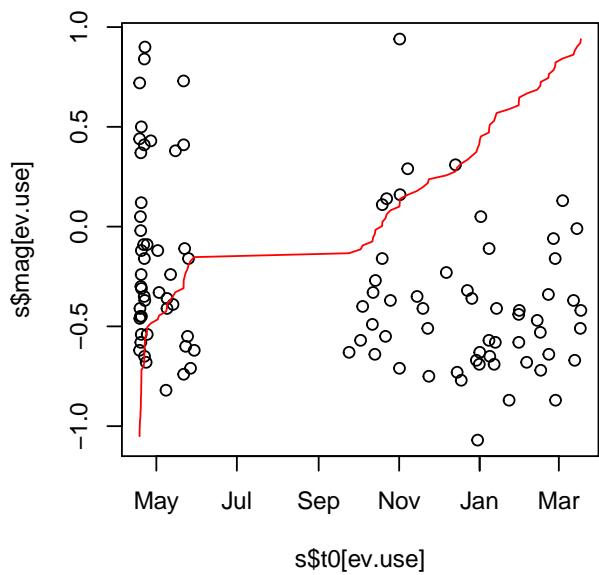
**BFMI = 0.79 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 587**



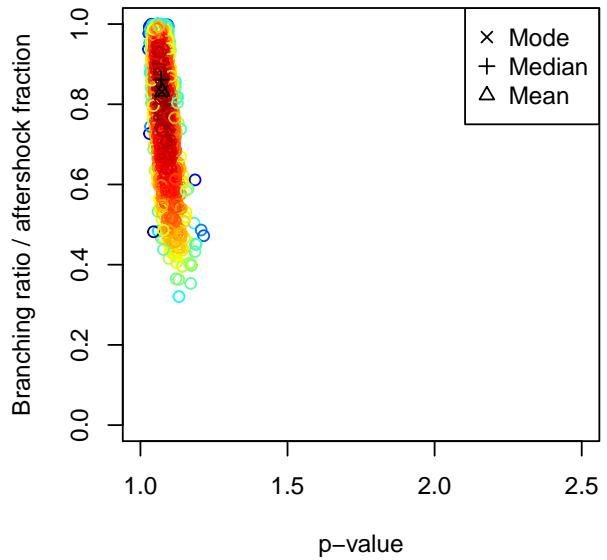
**Cv = 3.2**



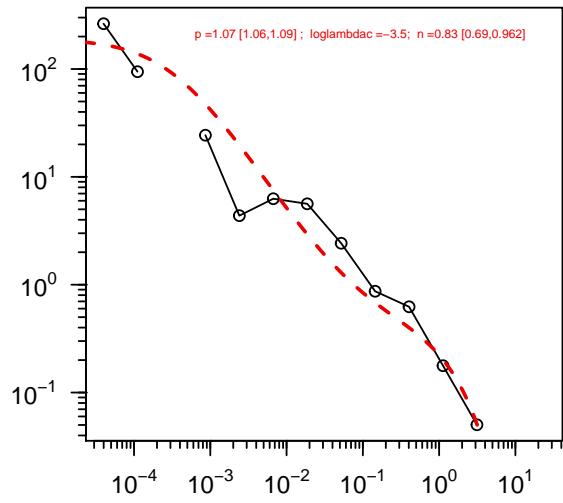
**Family 70095378 ; nev = 103**



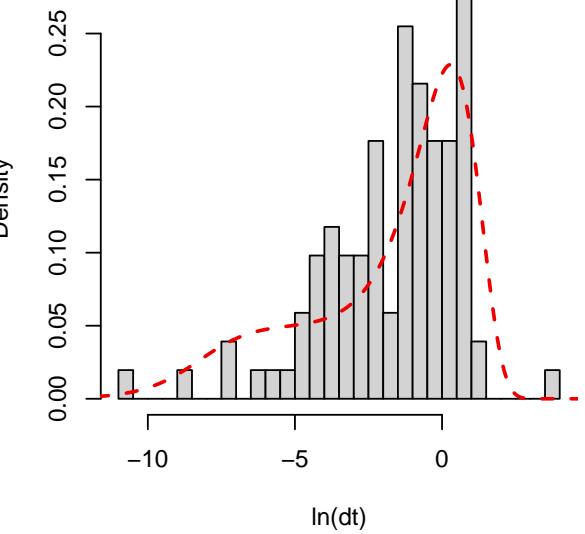
**BFMI = 0.99 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 823**



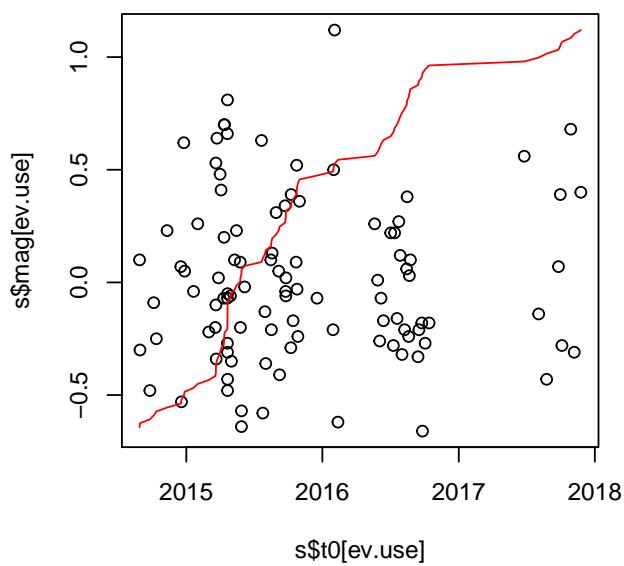
**Cv = 3.6**



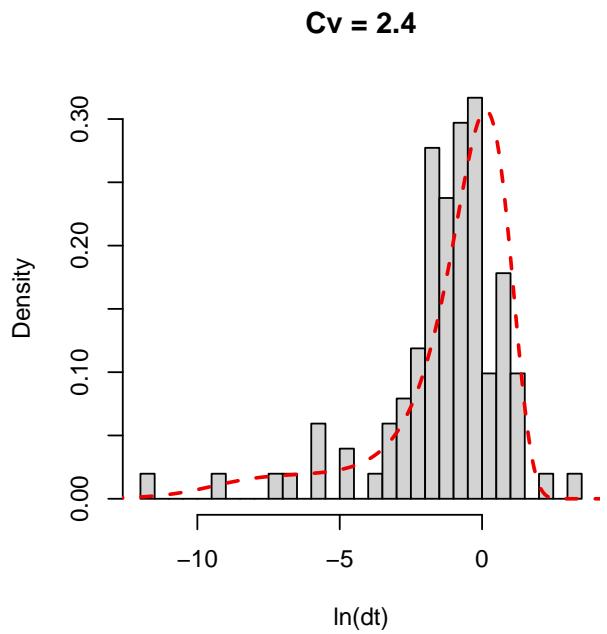
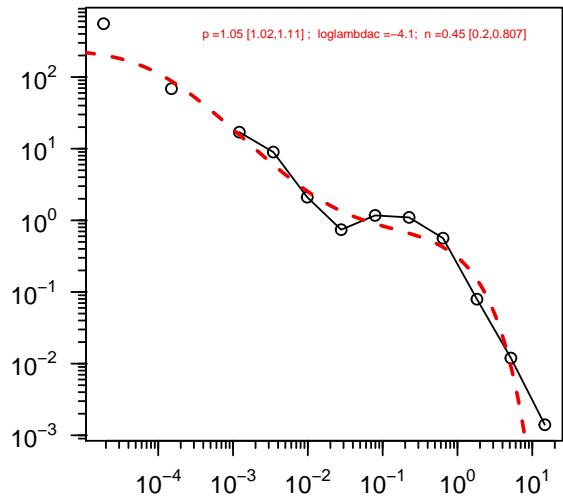
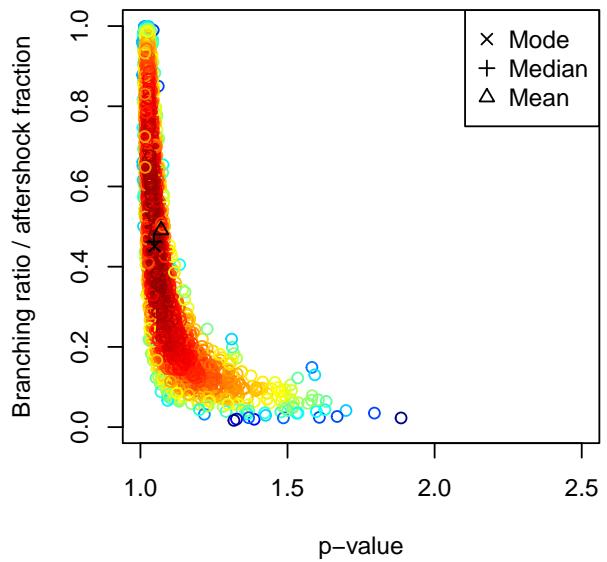
**Density**



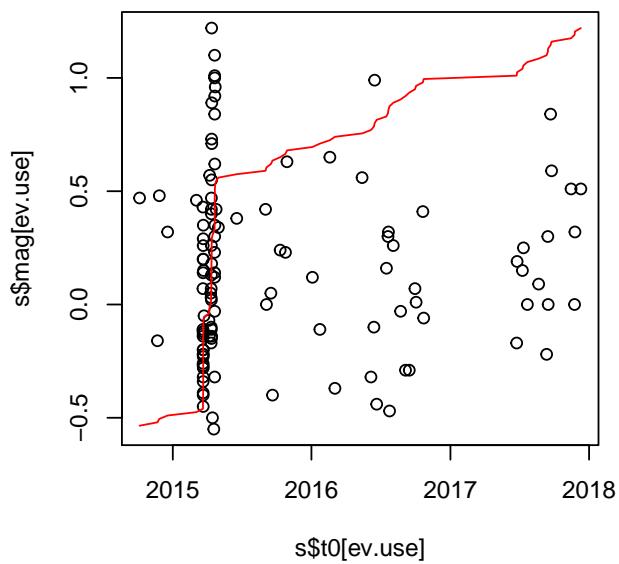
**Family 70096263 ; nev = 102**



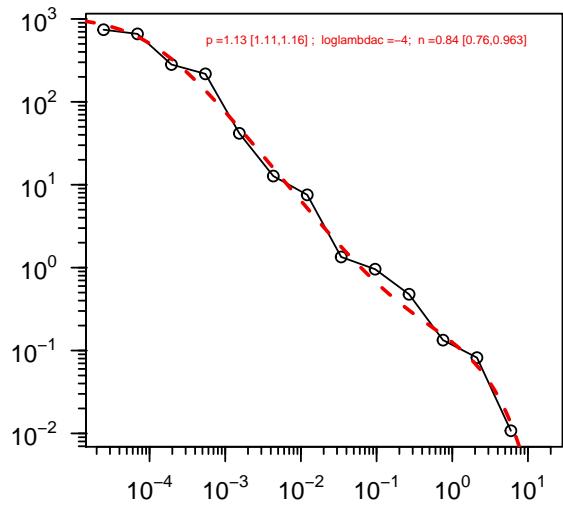
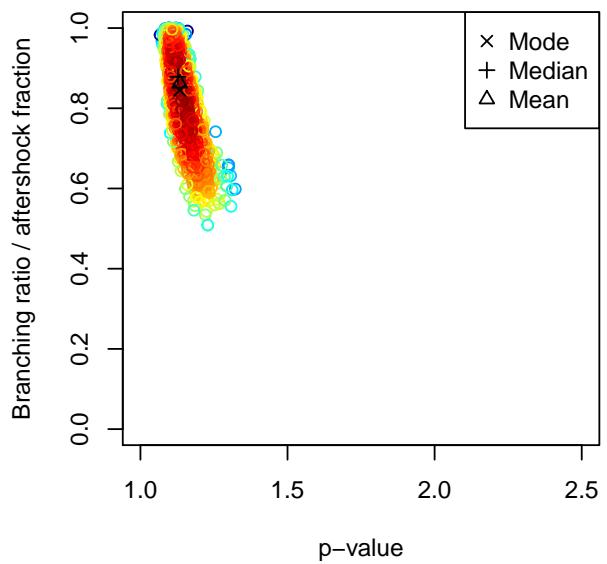
**BFMI = 0.69 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 716**



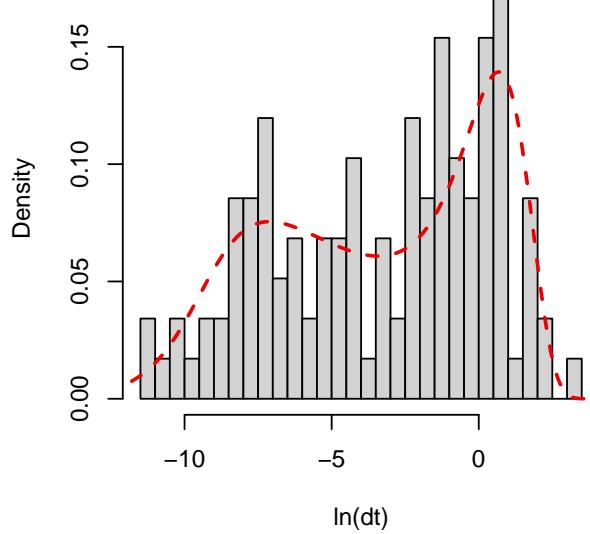
**Family 70096388 ; nev = 118**



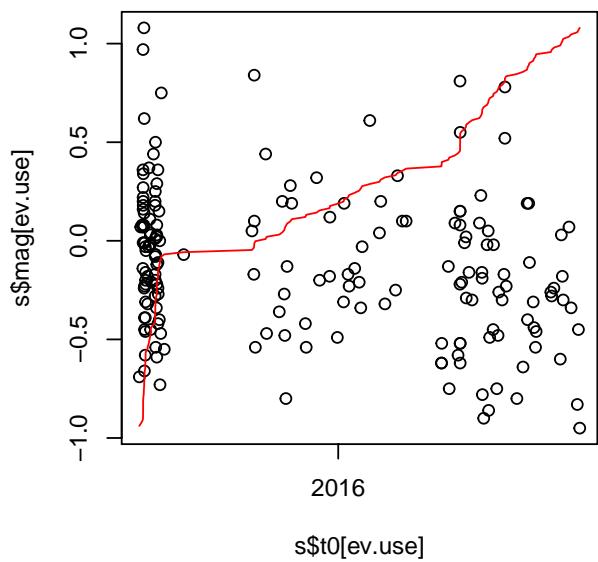
**BFMI = 0.84 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 427**



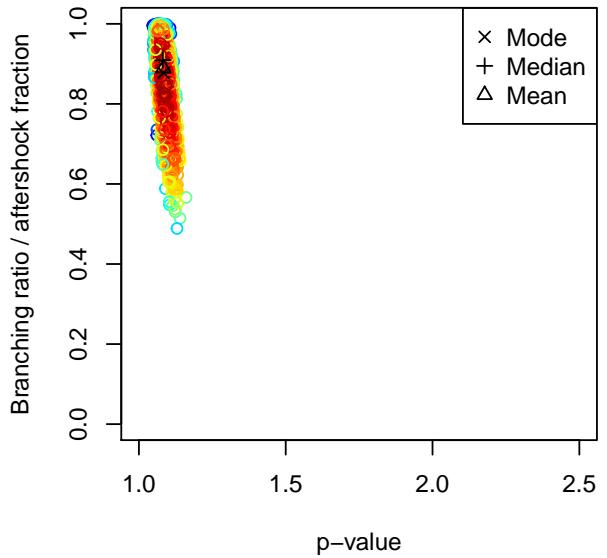
**Cv = 2.7**



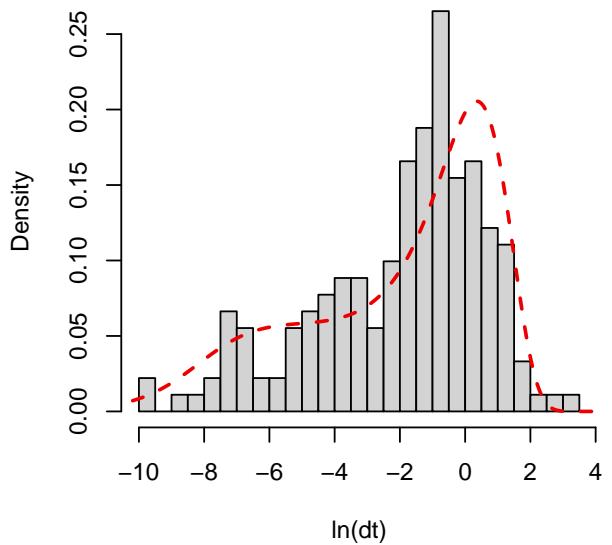
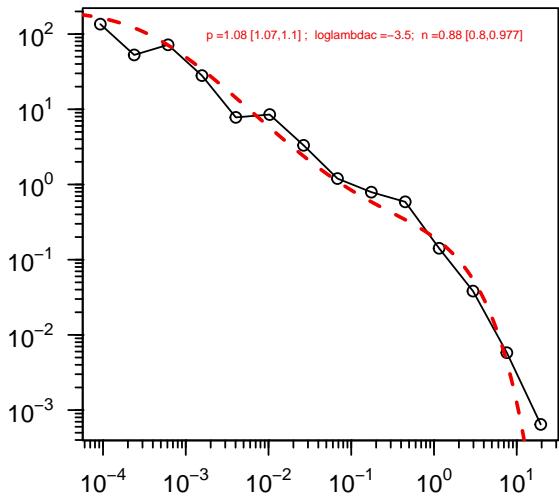
**Family 70101568 ; nev = 182**



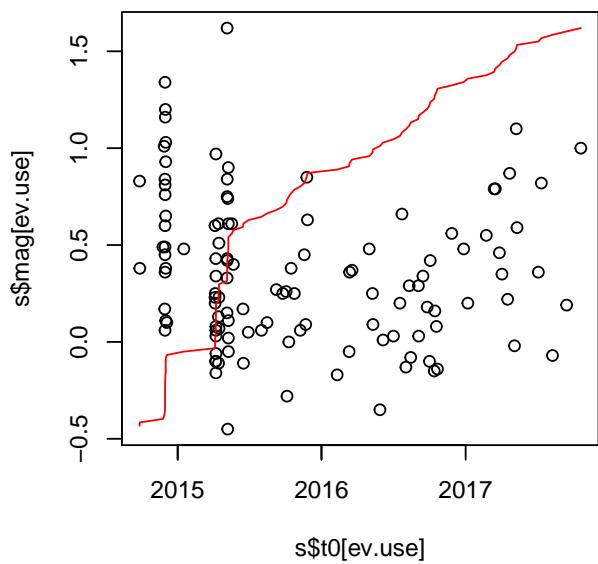
**BFMI = 0.9 ; n\_div = 0  
rhat = 1 ; n\_eff = 926**



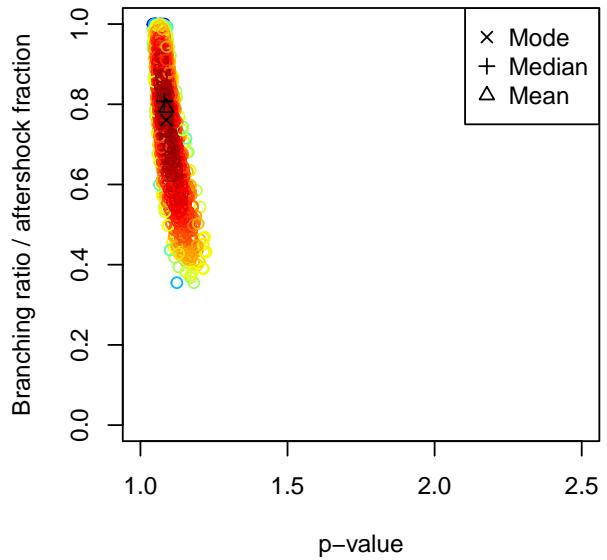
**Cv = 2.6**



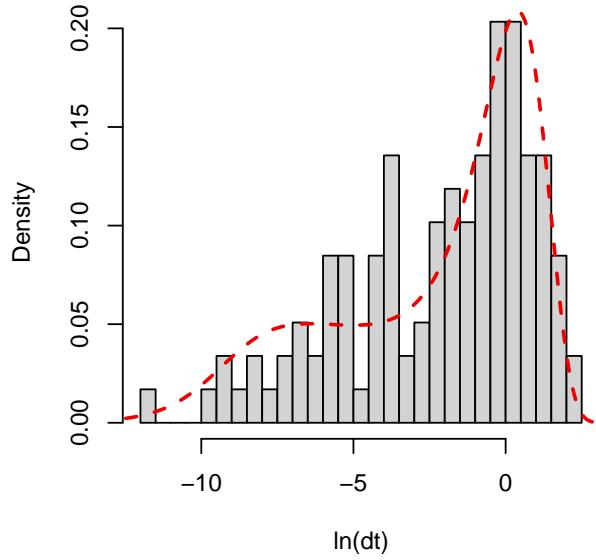
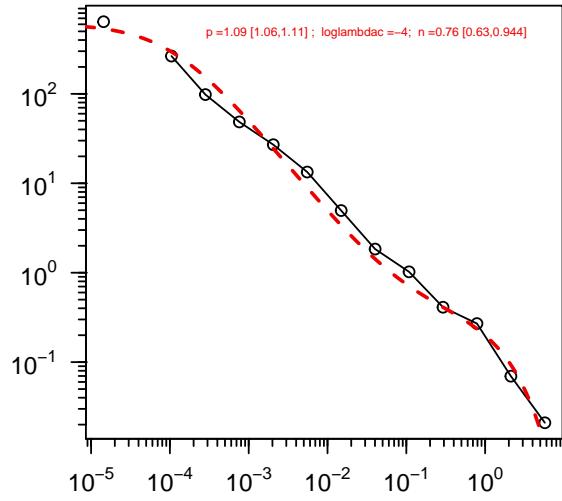
**Family 70101678 ; nev = 119**



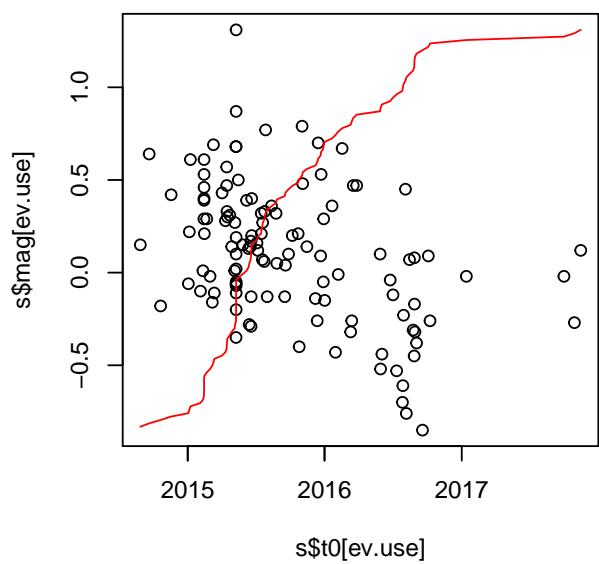
**BFMI = 1.2 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 679**



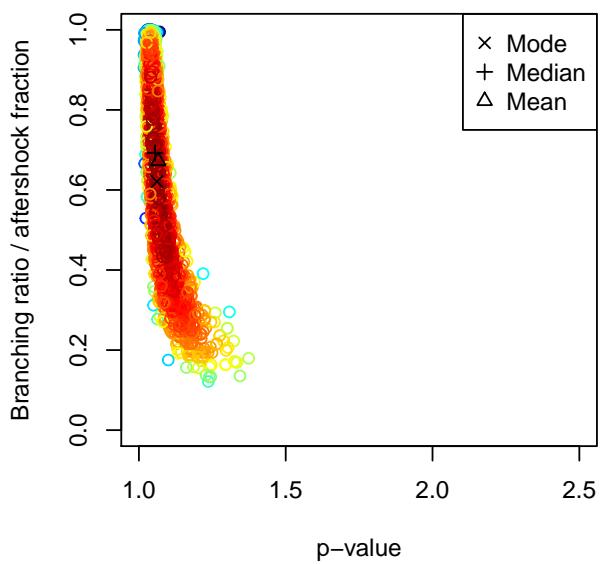
**Cv = 1.6**



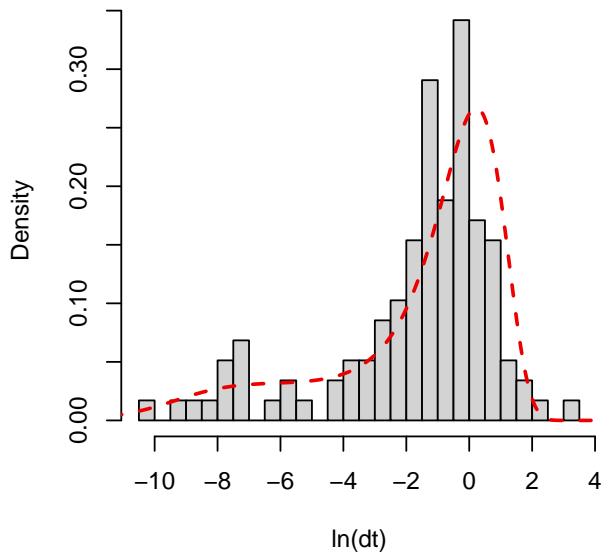
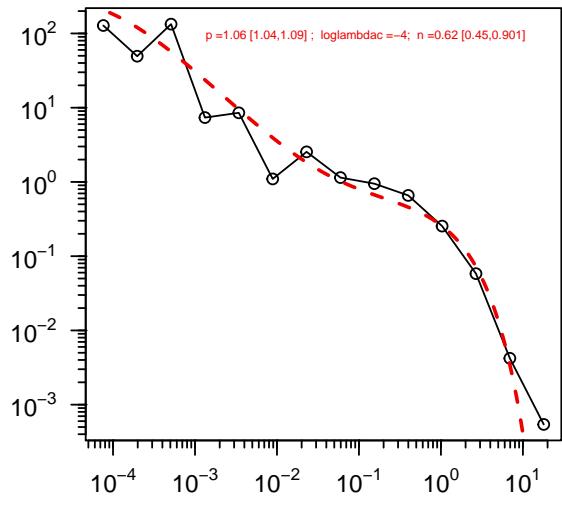
**Family 70102778 ; nev = 118**



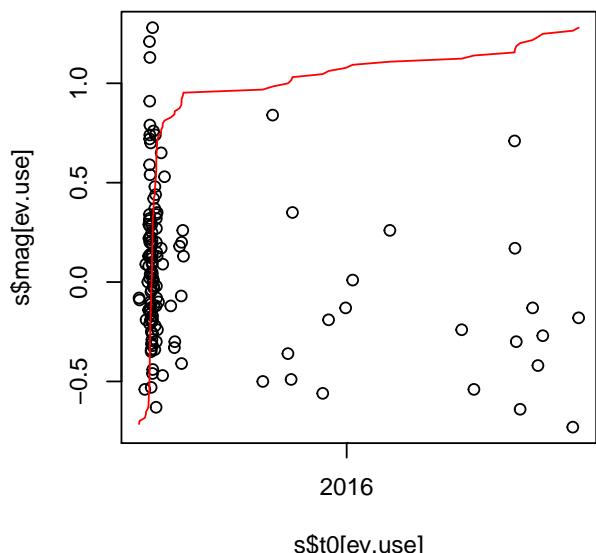
**BFMI = 1.1 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 795**



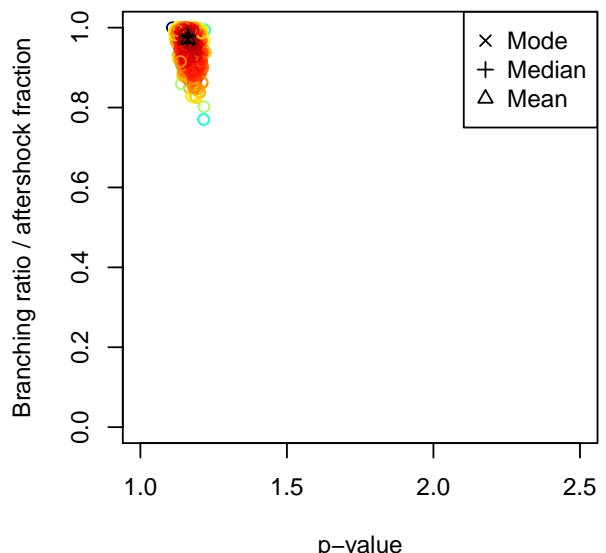
**Cv = 2.6**



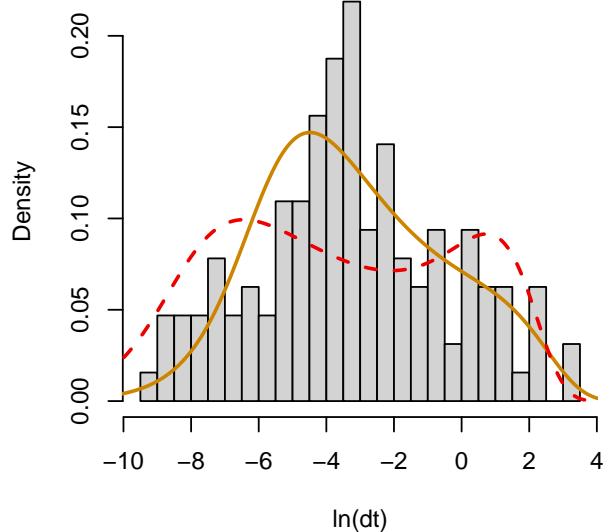
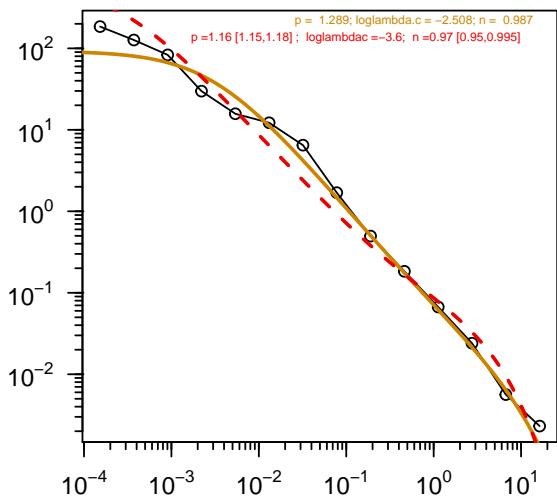
**Family 70103613 ; nev = 129**



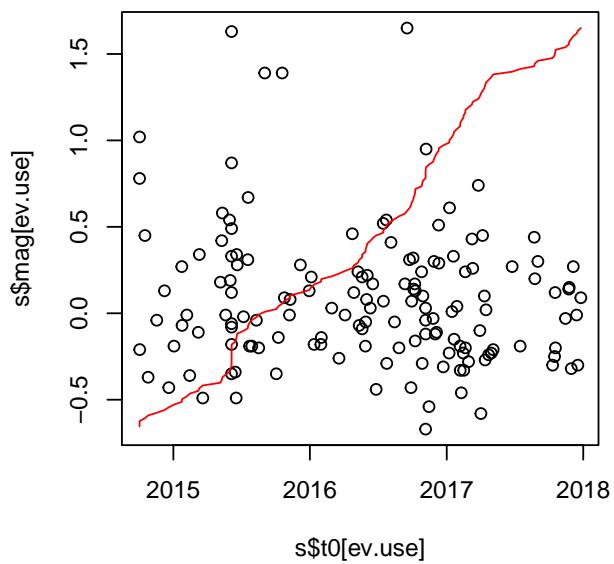
**BFMI = 1.1 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 1046**



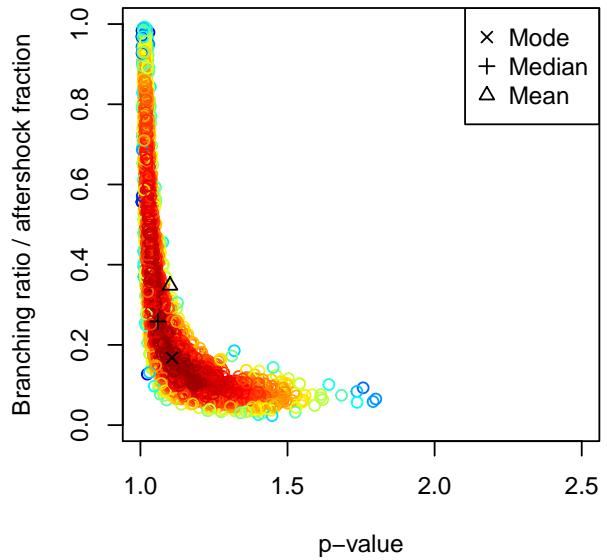
**Cv = 3.3**



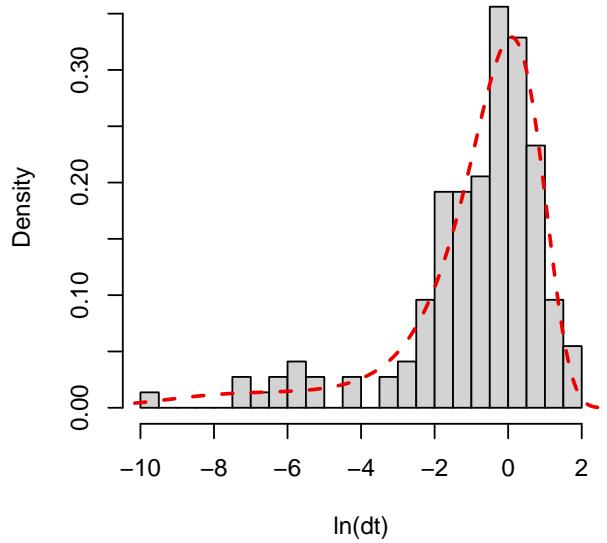
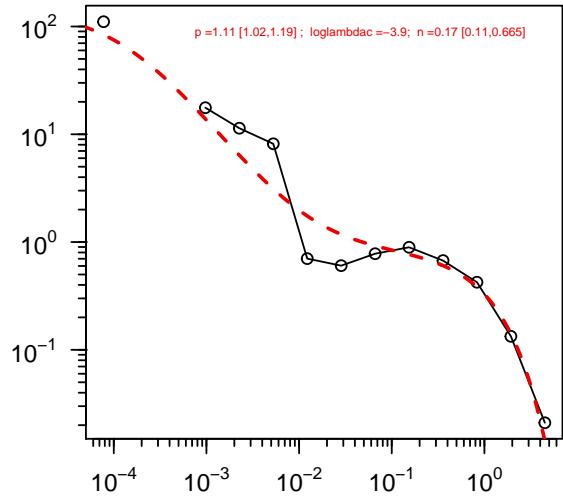
**Family 70110303 ; nev = 147**



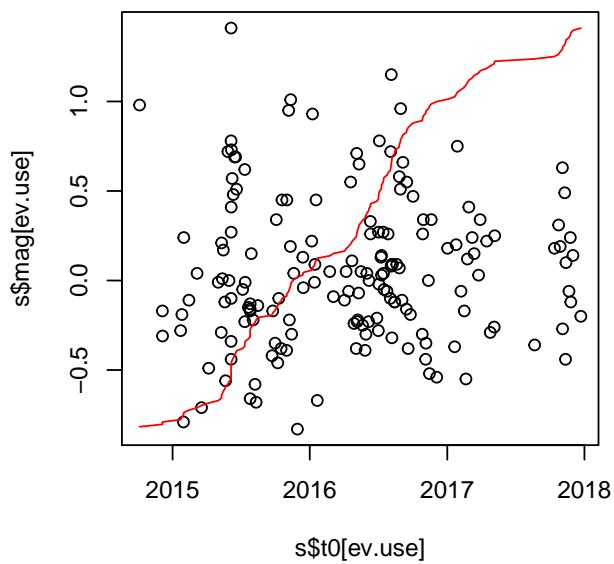
**BFMI = 0.95 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 785**



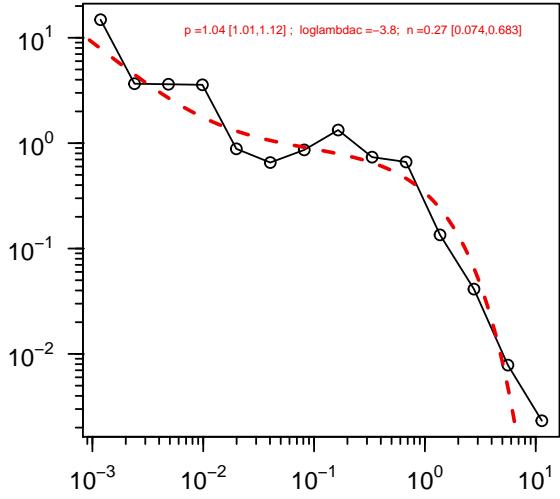
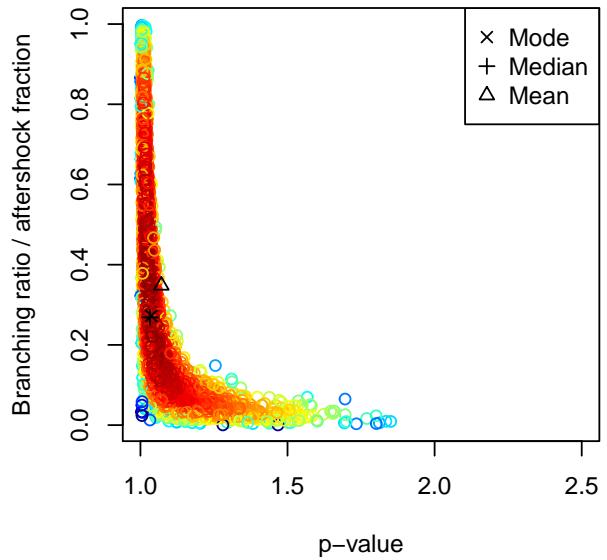
**Cv = 1.1**



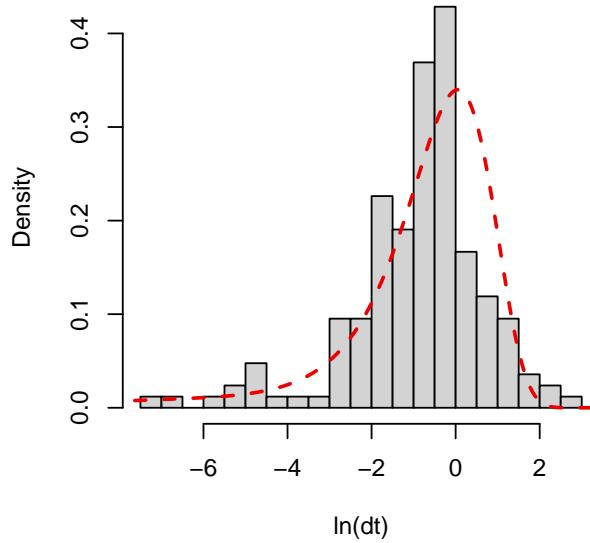
**Family 70110393 ; nev = 169**



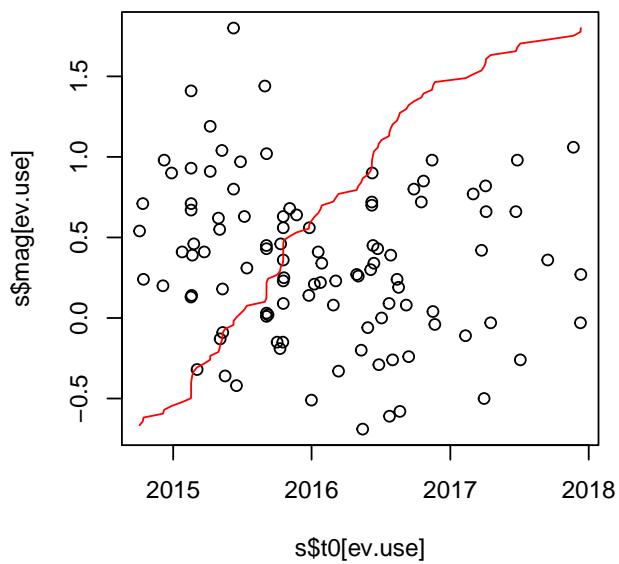
**BFMI = 0.96 ; n\_div = 0  
rhat = 1 ; n\_eff = 732**



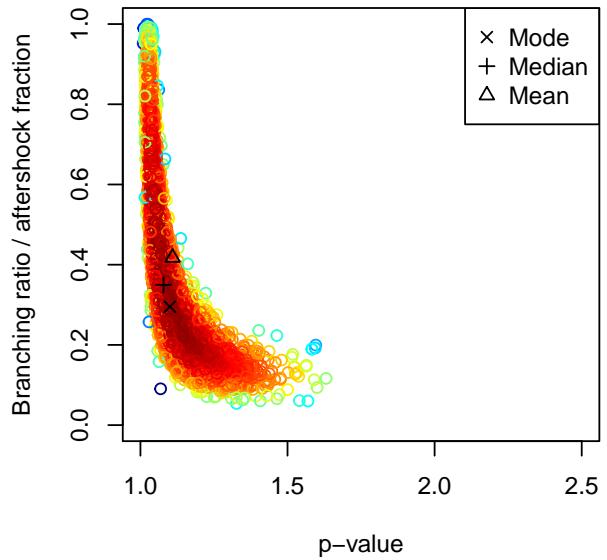
**Cv = 1.7**



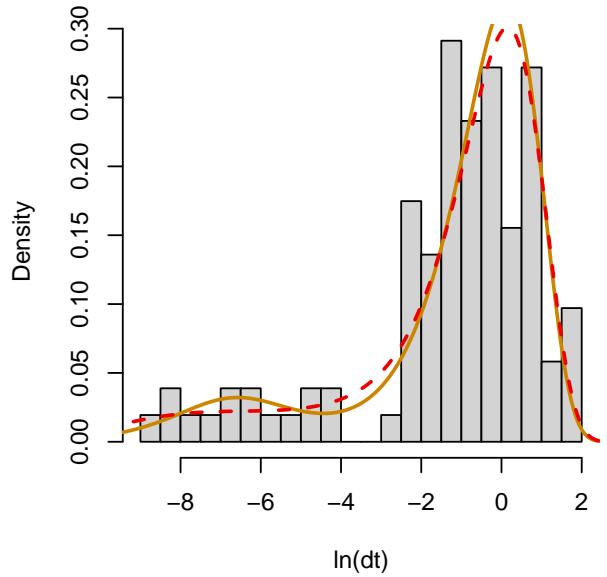
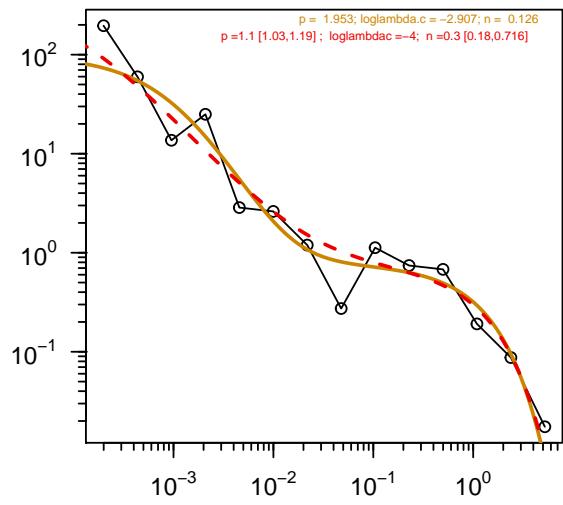
**Family 70111533 ; nev = 104**



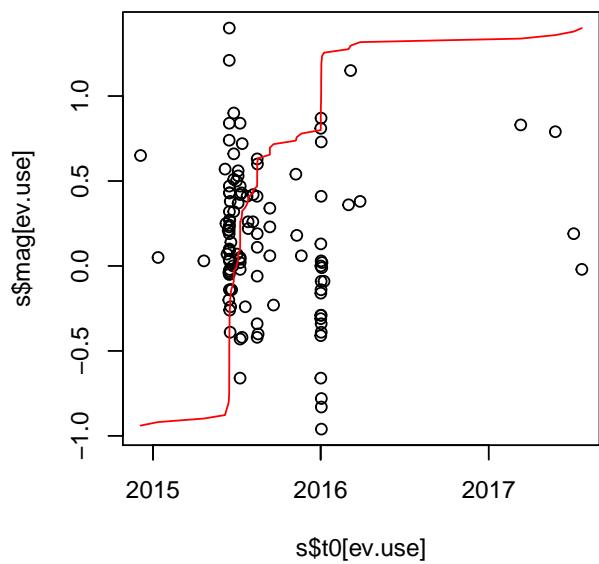
**BFMI = 0.83 ; n\_div = 0  
rhat = 1.02 ; n\_eff = 762**



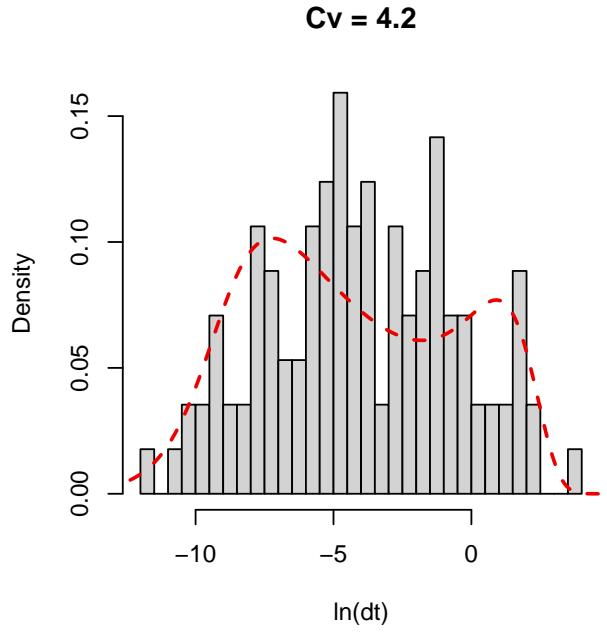
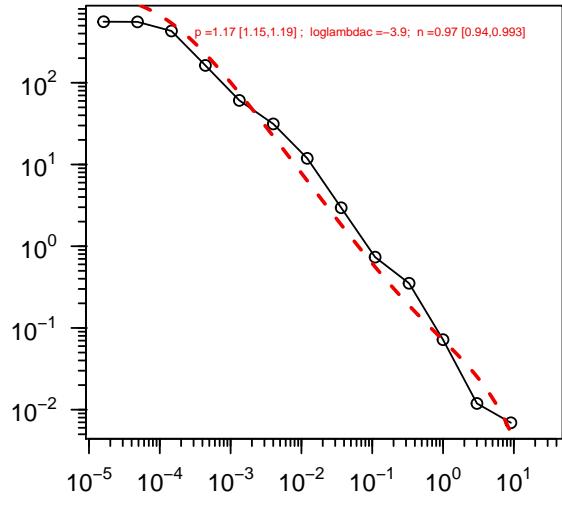
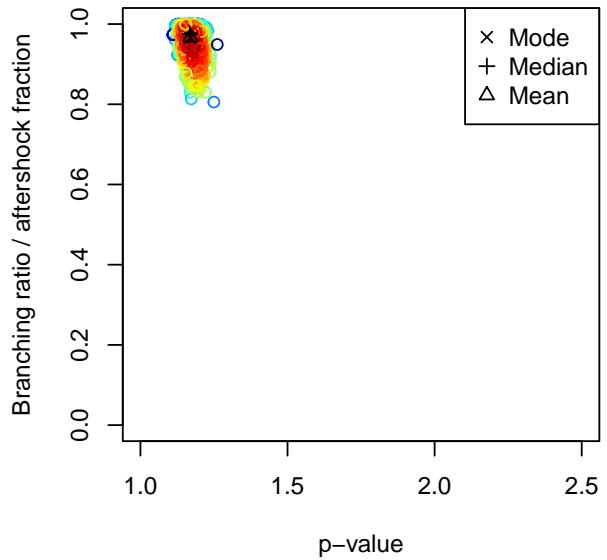
**Cv = 1.4**



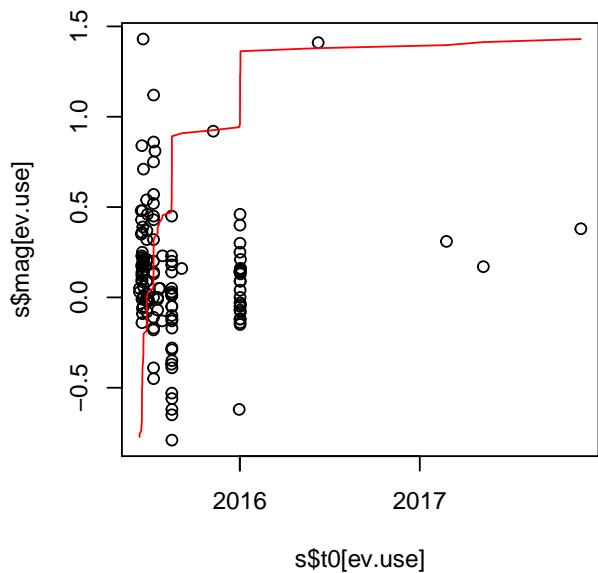
**Family 70113458 ; nev = 114**



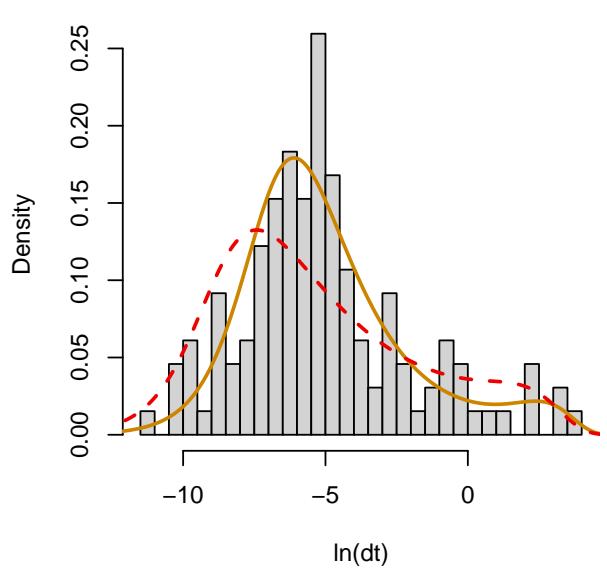
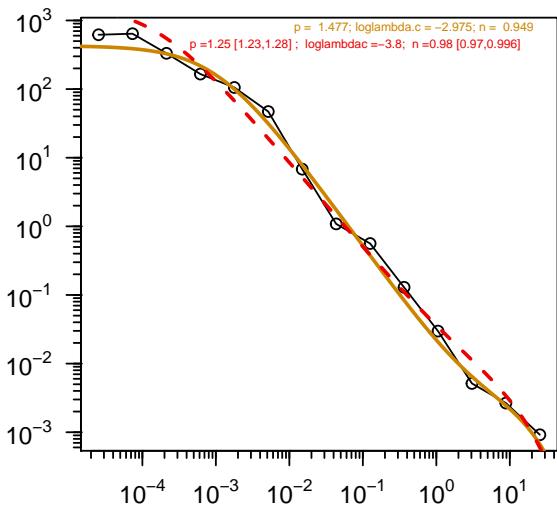
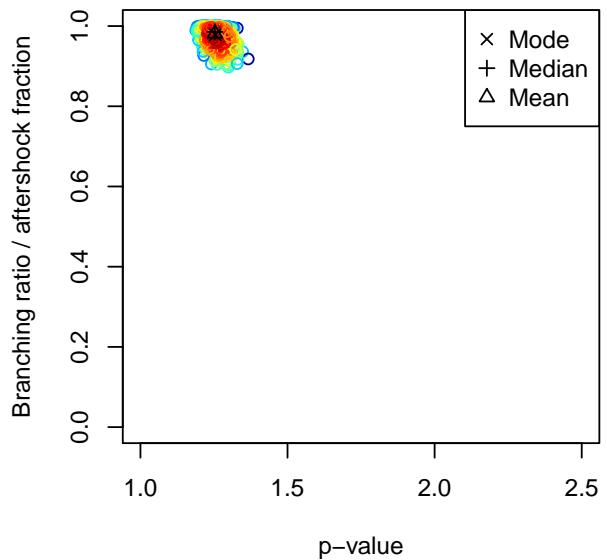
**BFMI = 1.1 ; n\_div = 0  
rhat = 1 ; n\_eff = 1401**



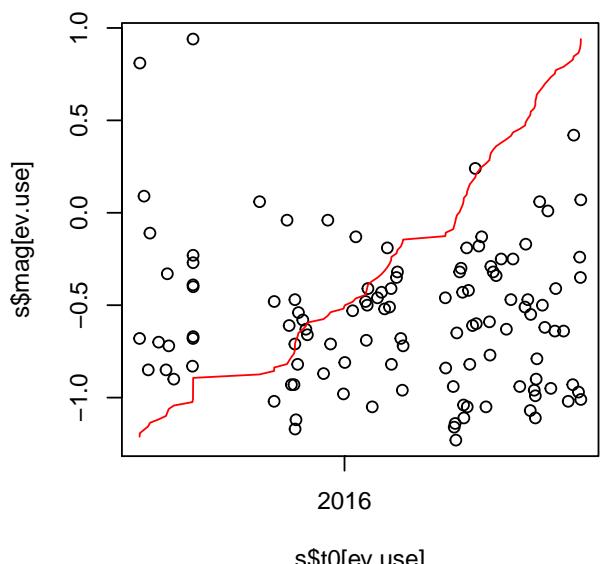
**Family 70114143 ; nev = 132**



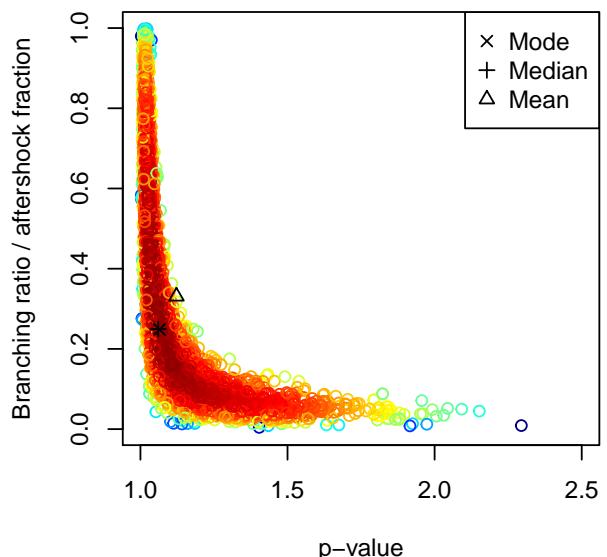
**BFMI = 1.2 ; n\_div = 0  
rhat = 1 ; n\_eff = 818**



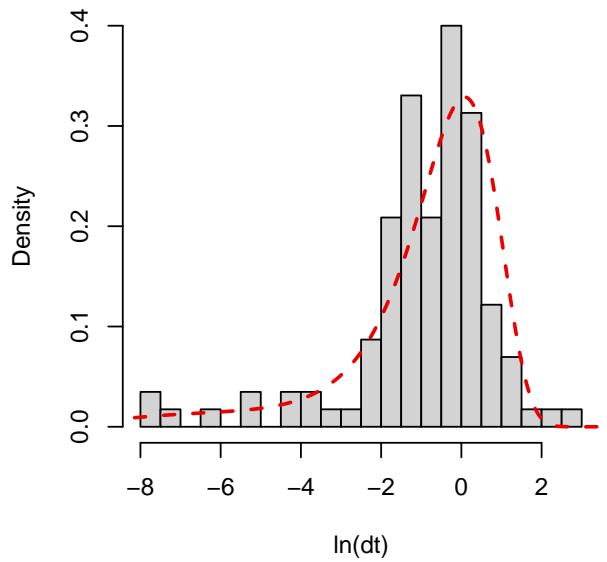
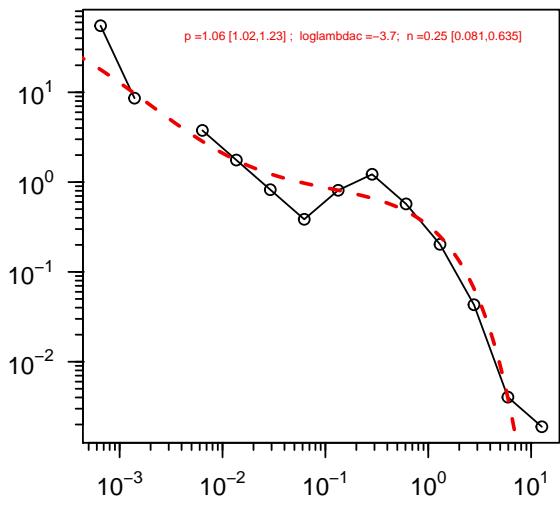
**Family 70115813 ; nev = 116**



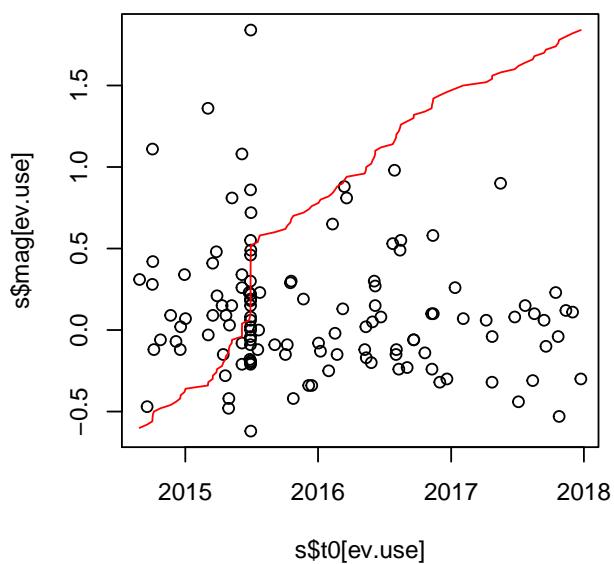
**BFMI = 1.1 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 812**



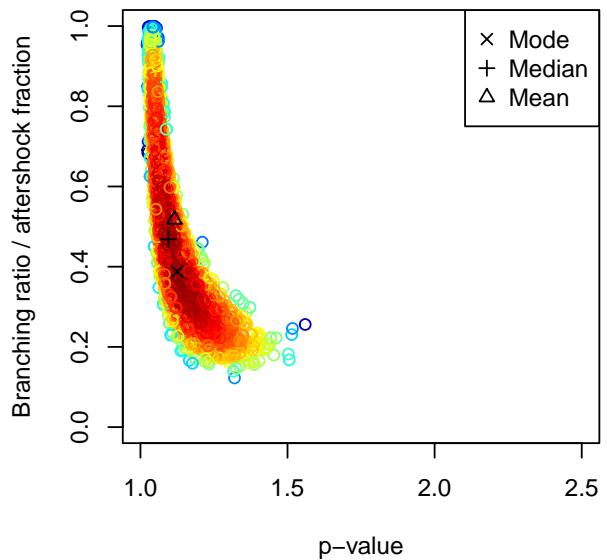
**Cv = 2**



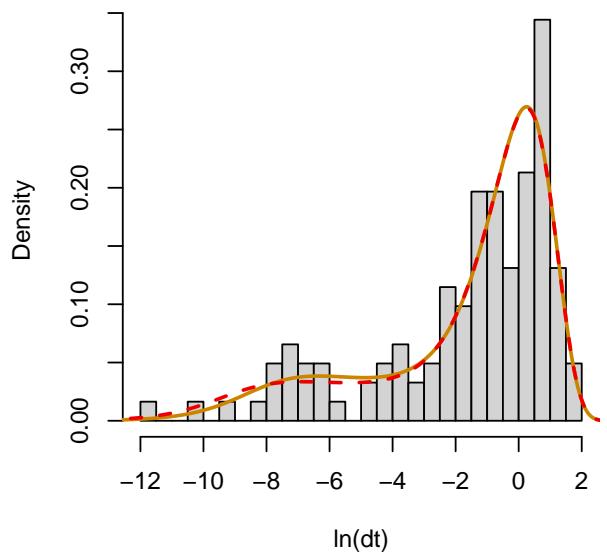
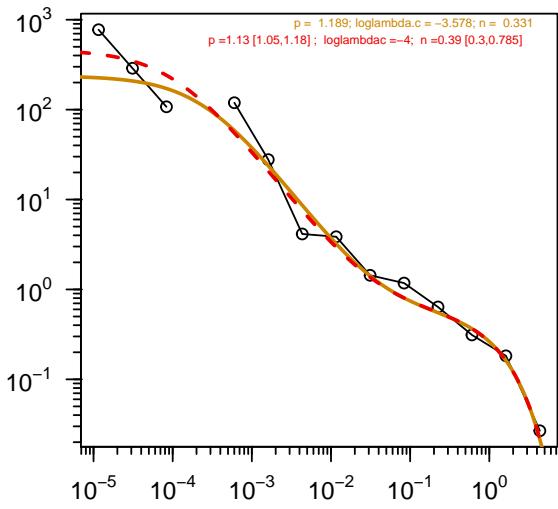
**Family 70116973 ; nev = 123**



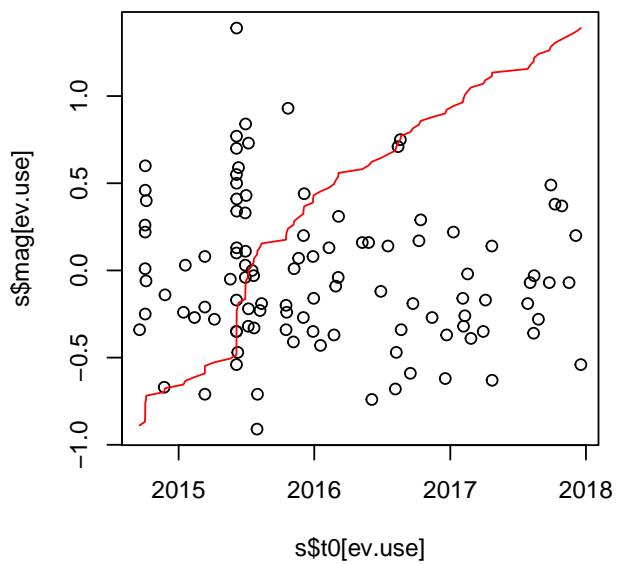
**BFMI = 1 ; n\_div = 0  
rhat = 1 ; n\_eff = 672**



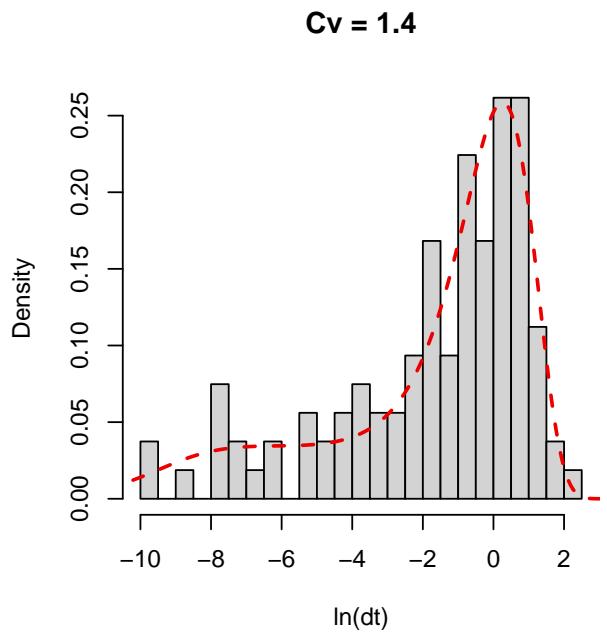
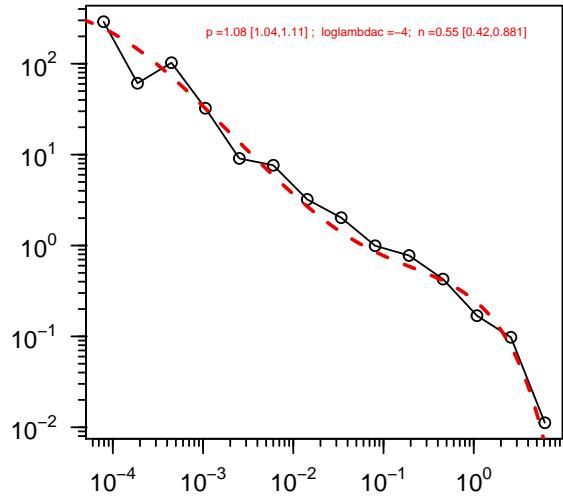
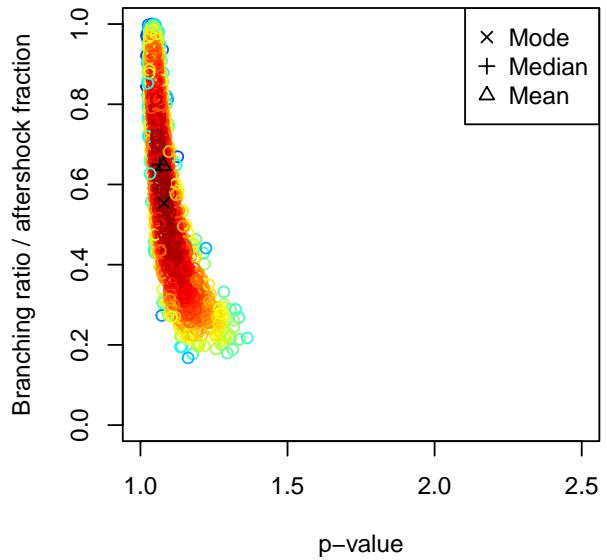
**Cv = 1.3**



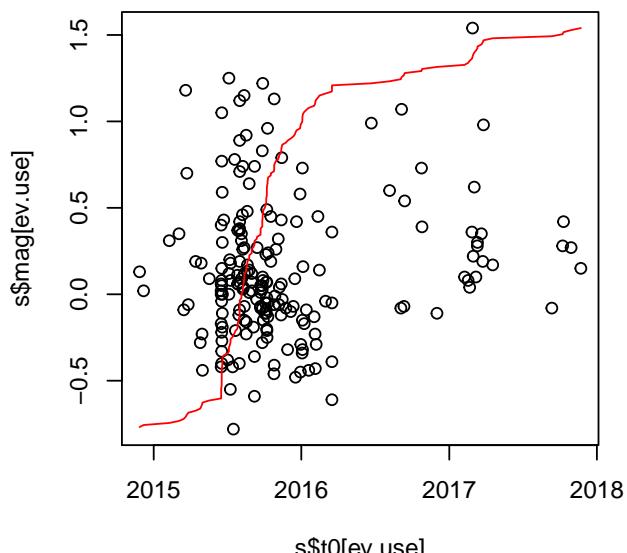
**Family 70117223 ; nev = 108**



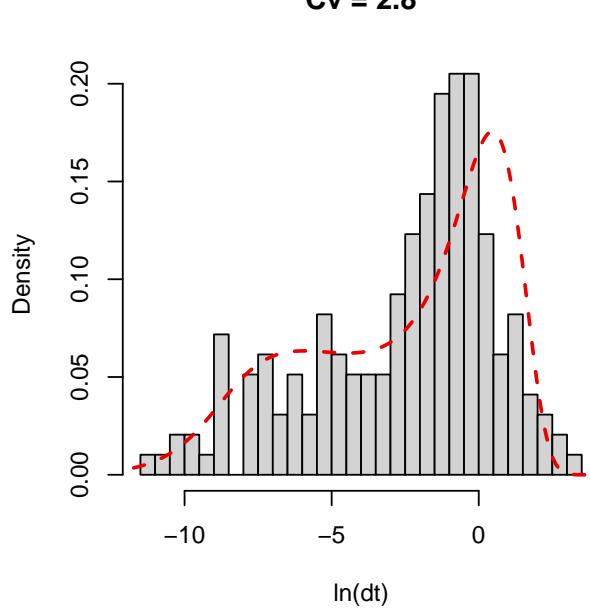
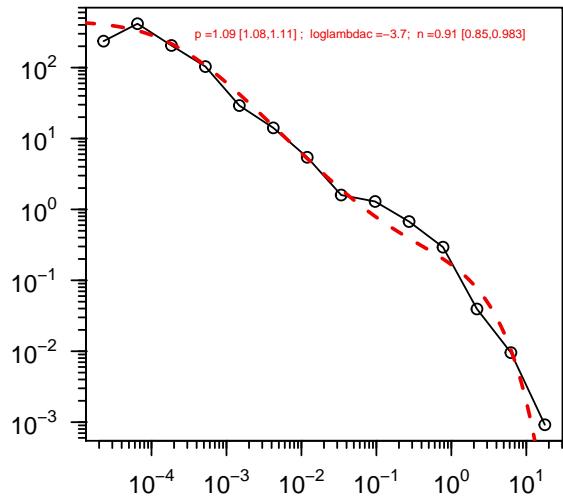
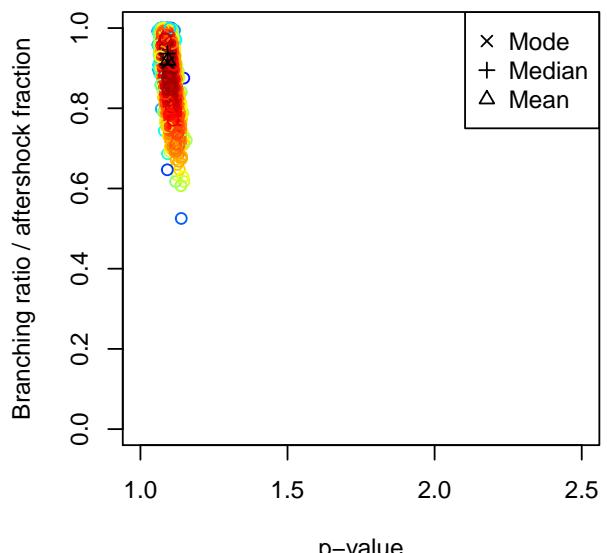
**BFMI = 0.89 ; n\_div = 0  
rhat = 1 ; n\_eff = 695**



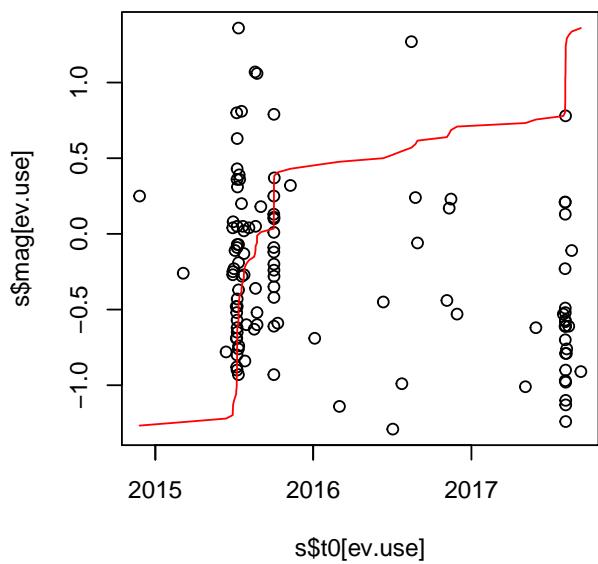
**Family 70118748 ; nev = 196**



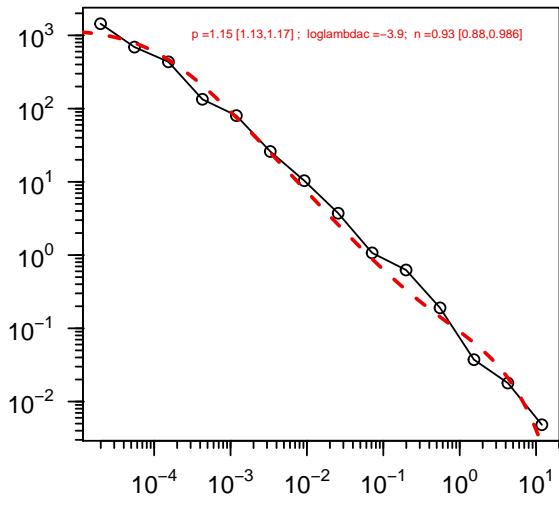
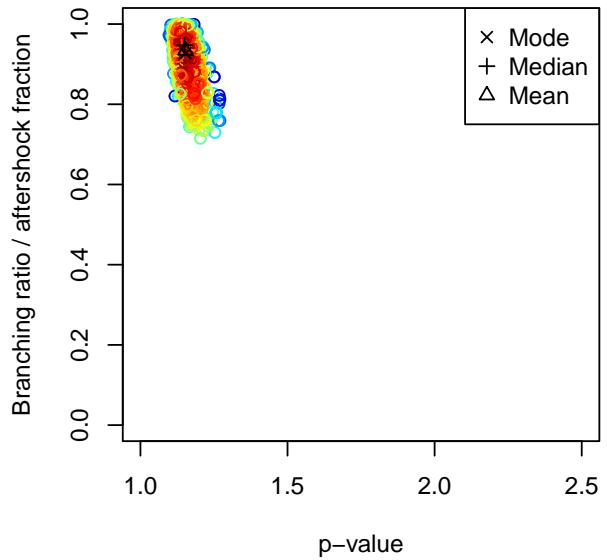
**BFMI = 0.88 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 804**



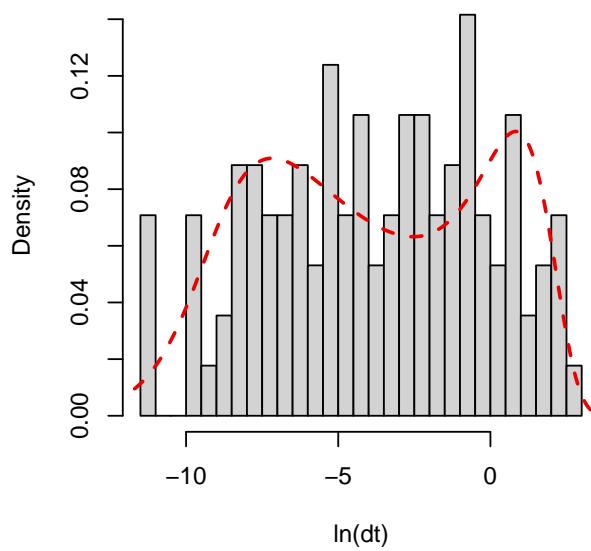
**Family 70119283 ; nev = 114**



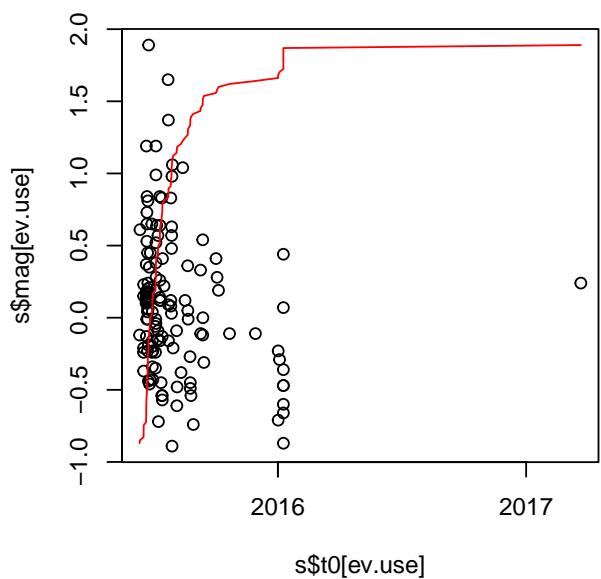
**BFMI = 0.97 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 590**



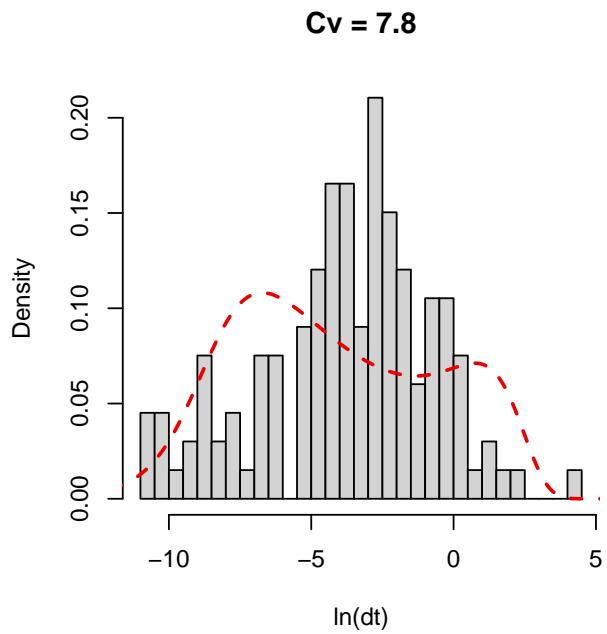
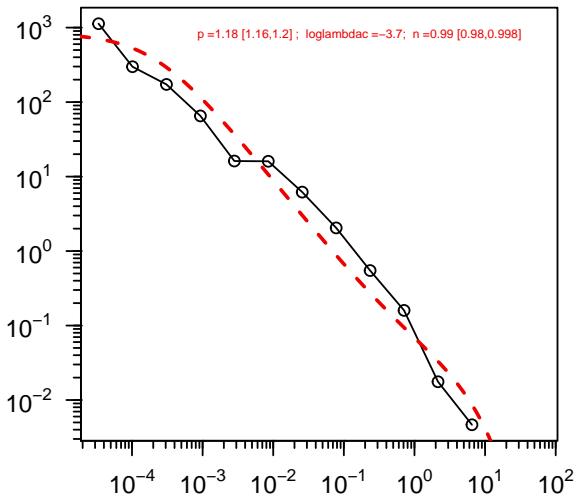
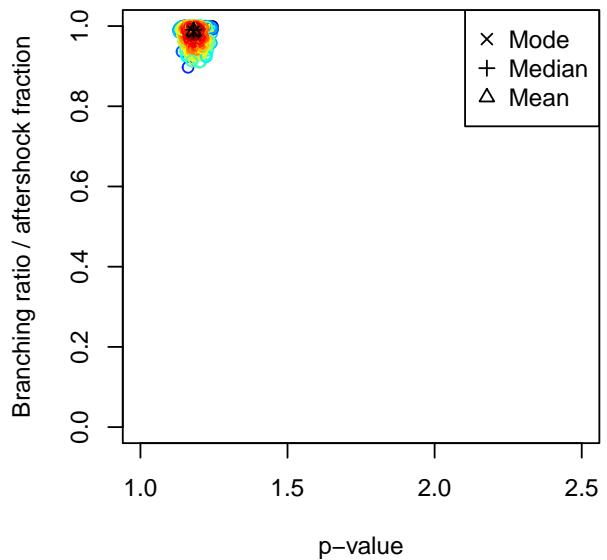
**Cv = 2.7**



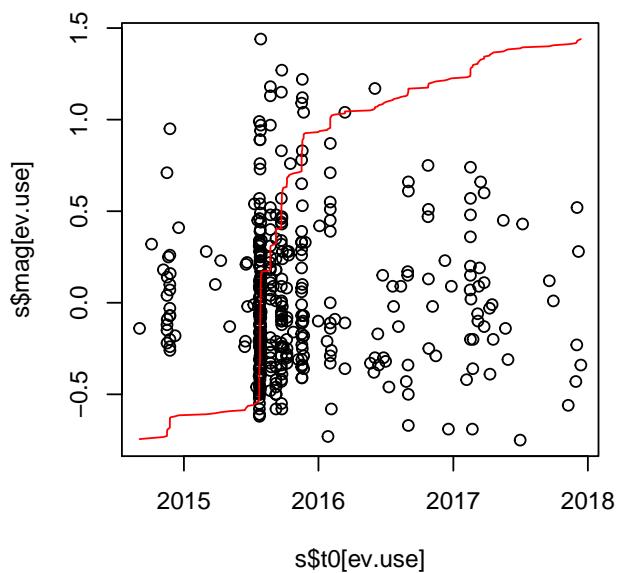
**Family 70123208 ; nev = 134**



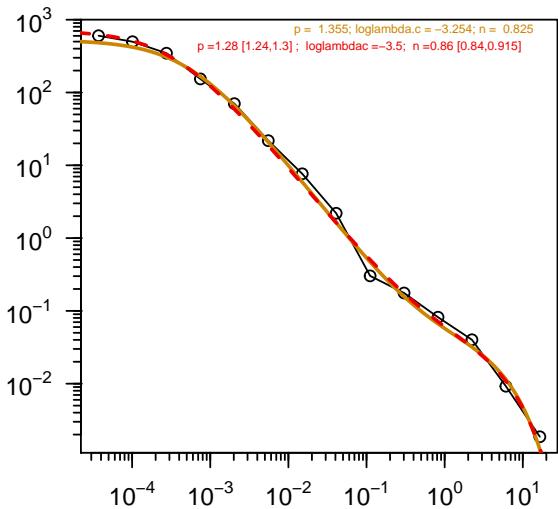
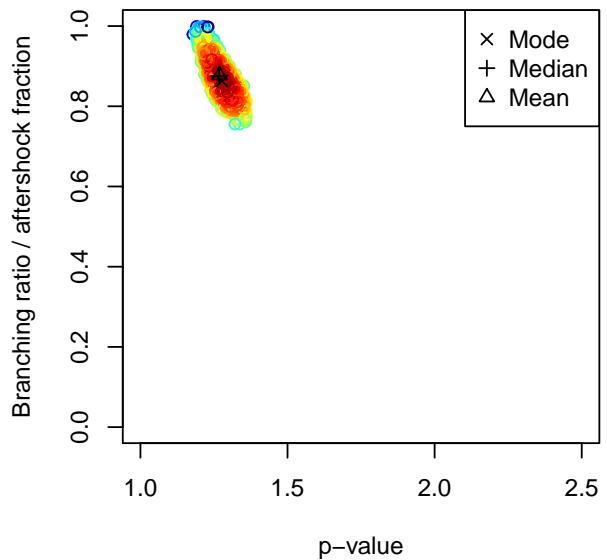
**BFMI = 0.83 ; n\_div = 0  
rhat = 1 ; n\_eff = 1549**



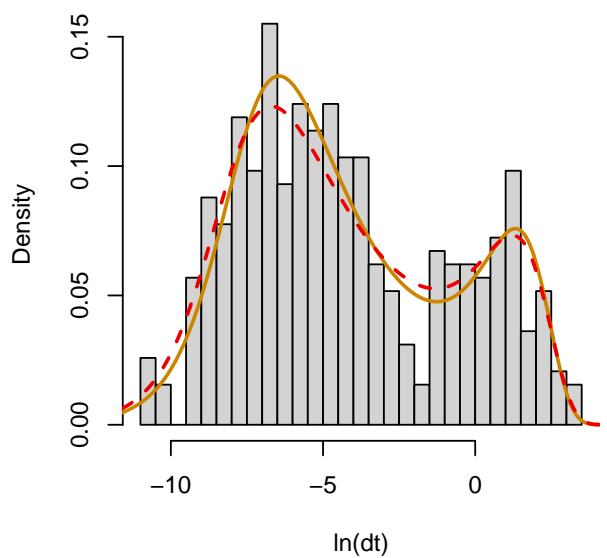
**Family 70124853 ; nev = 388**



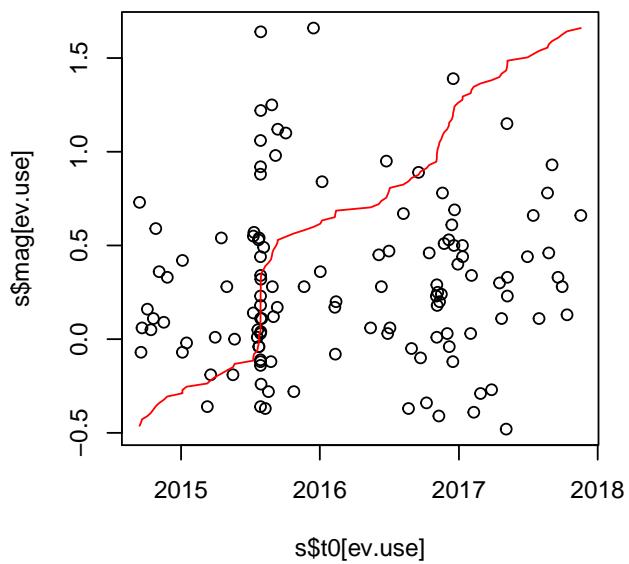
**BFMI = 0.89 ; n\_div = 0  
rhat = 1 ; n\_eff = 1013**



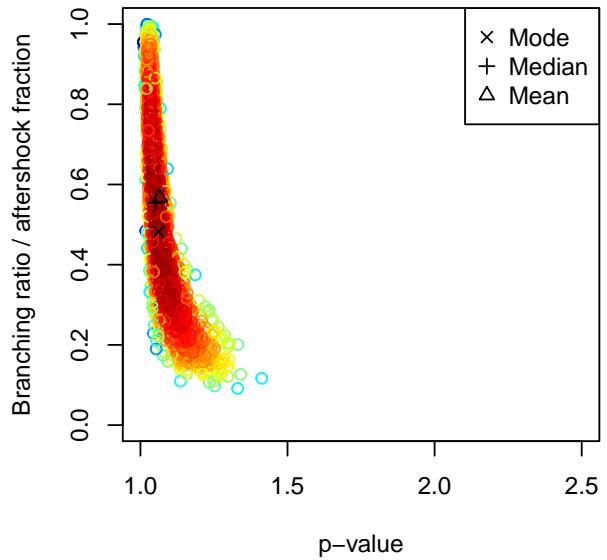
**Cv = 3**



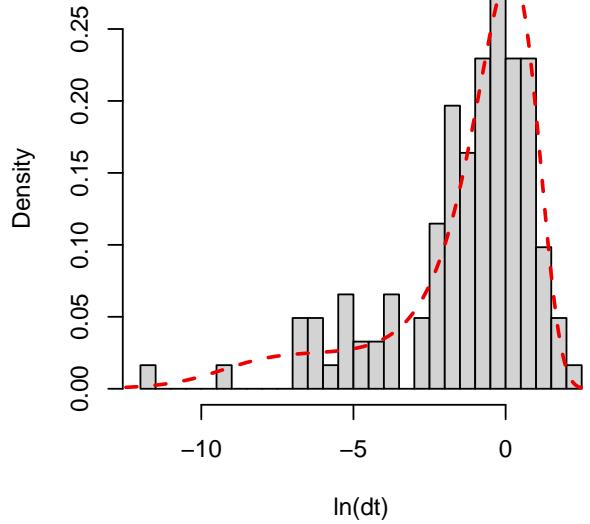
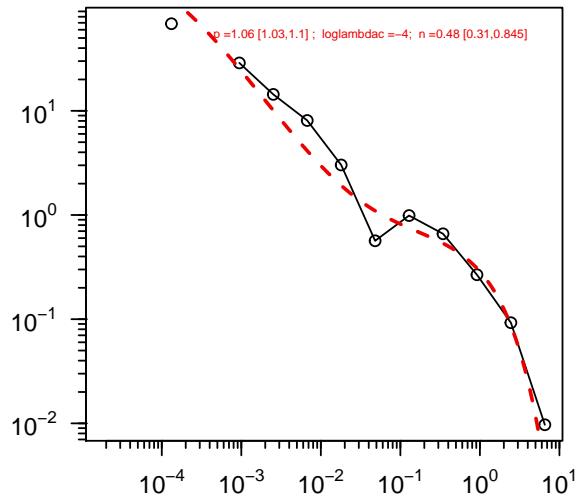
**Family 70124913 ; nev = 123**



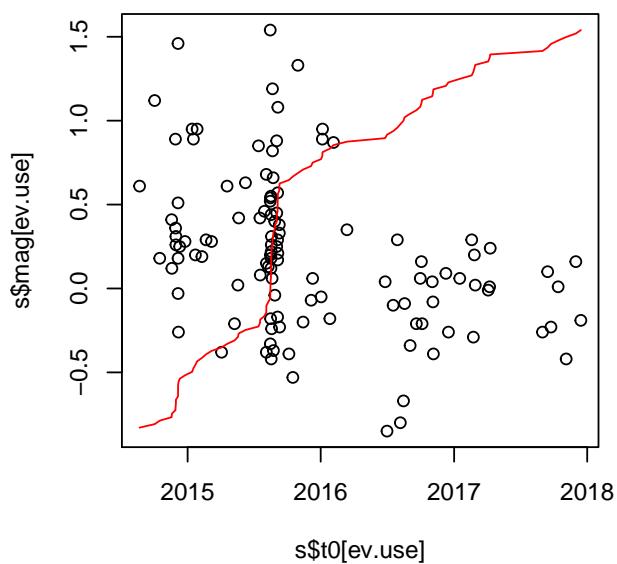
**BFMI = 0.89 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 670**



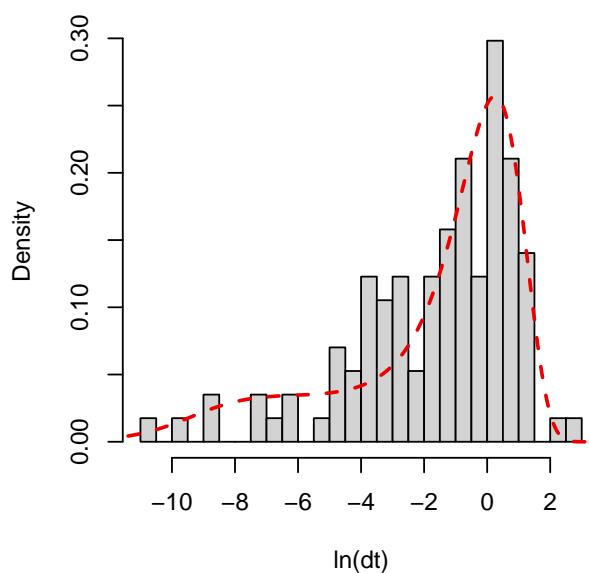
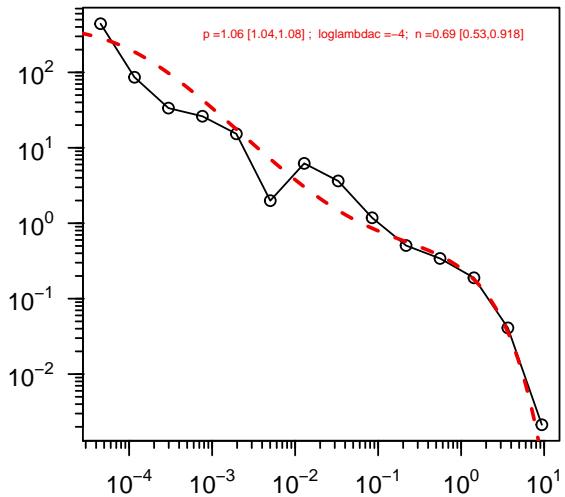
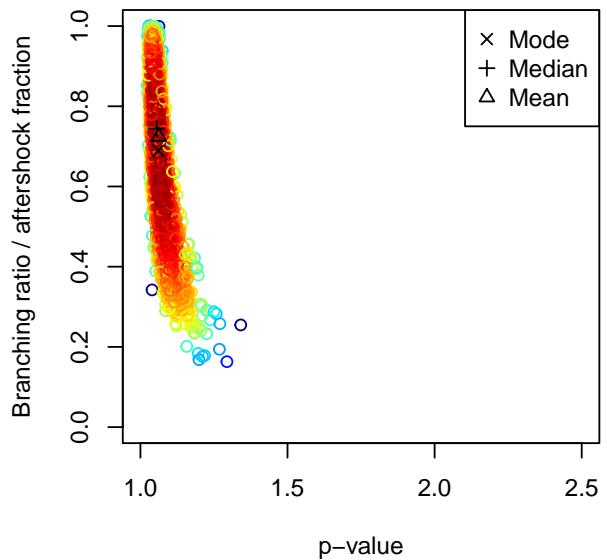
**Cv = 1.4**



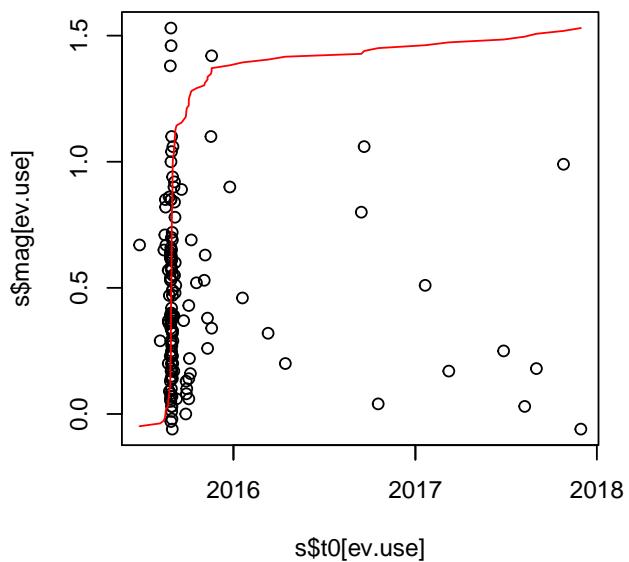
**Family 70128478 ; nev = 115**



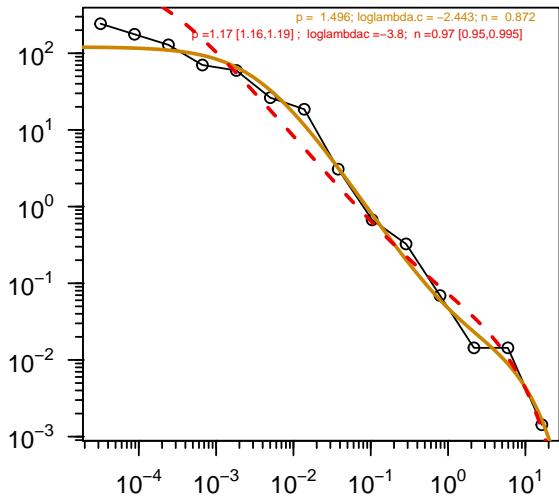
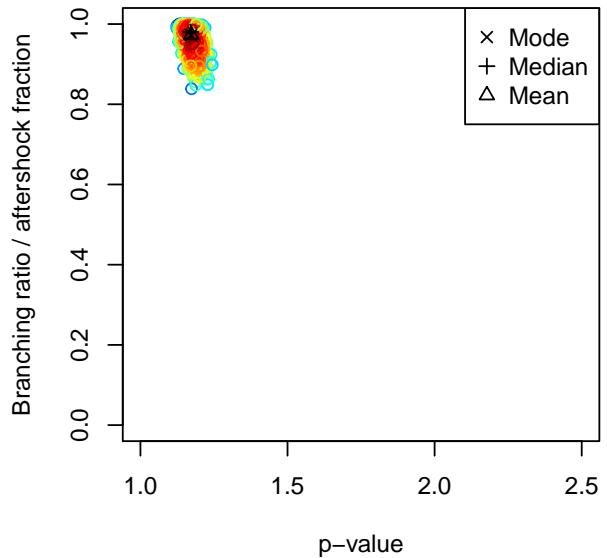
**BFMI = 0.87 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 501**



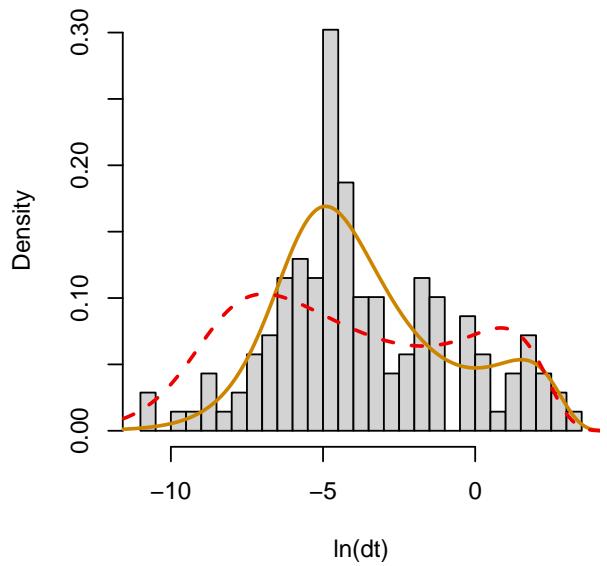
**Family 70131548 ; nev = 140**



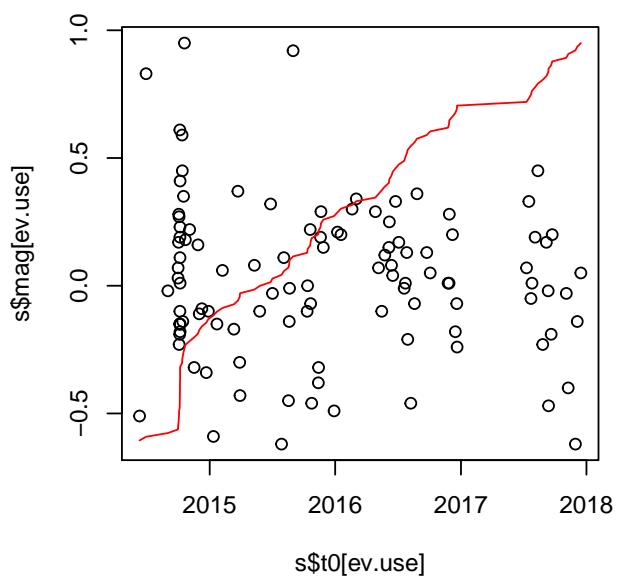
**BFMI = 1.2 ; n\_div = 0  
rhat = 1 ; n\_eff = 1075**



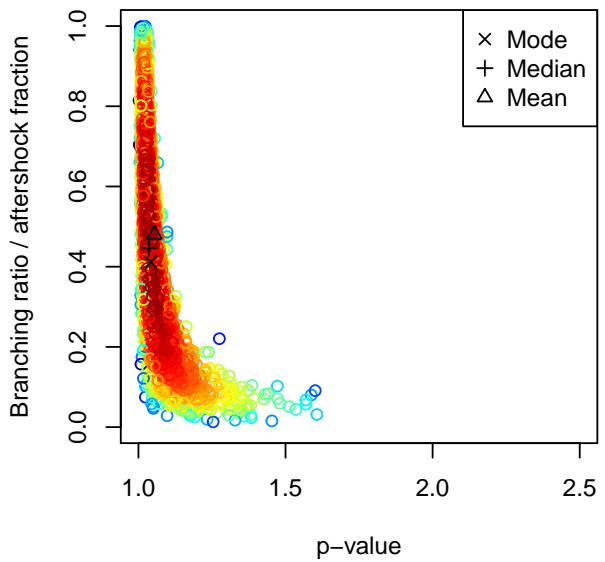
**Cv = 3.2**



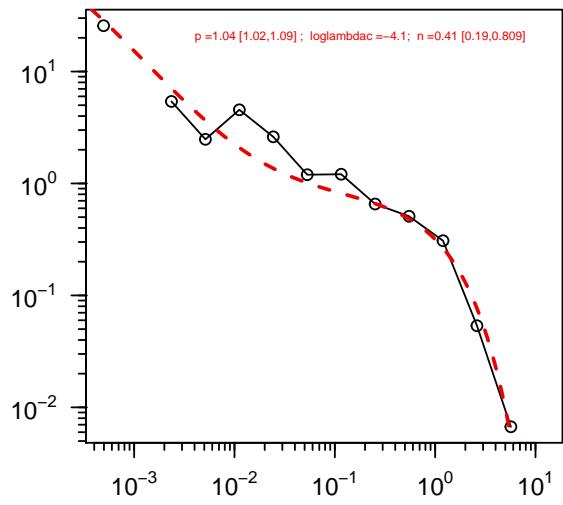
**Family 70132693 ; nev = 109**



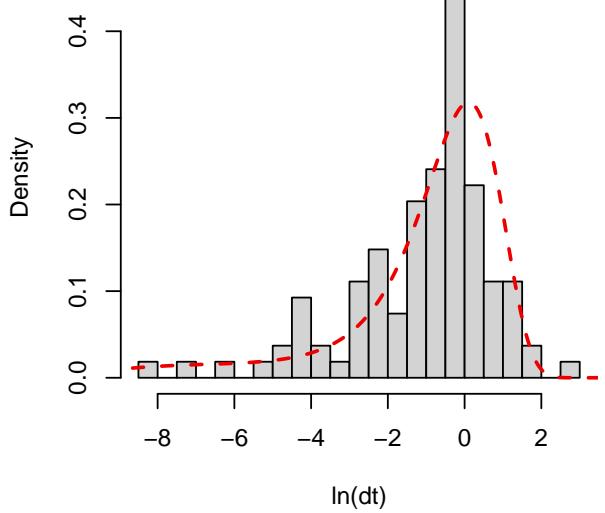
**BFMI = 0.93 ; n\_div = 0  
rhat = 1 ; n\_eff = 779**



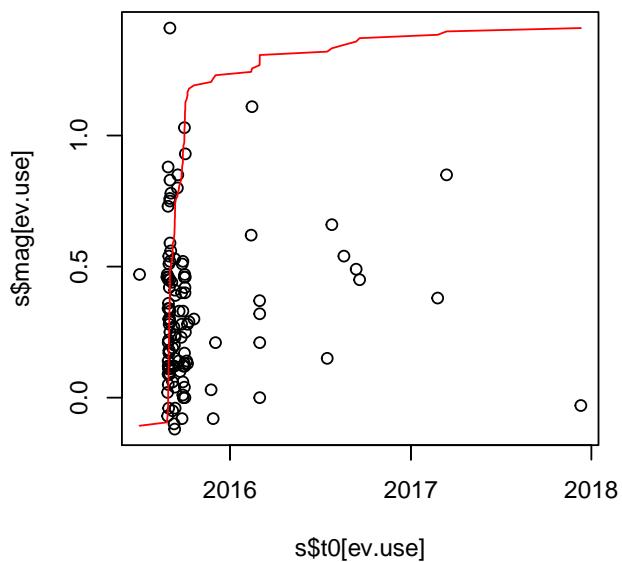
**Cv = 1.9**



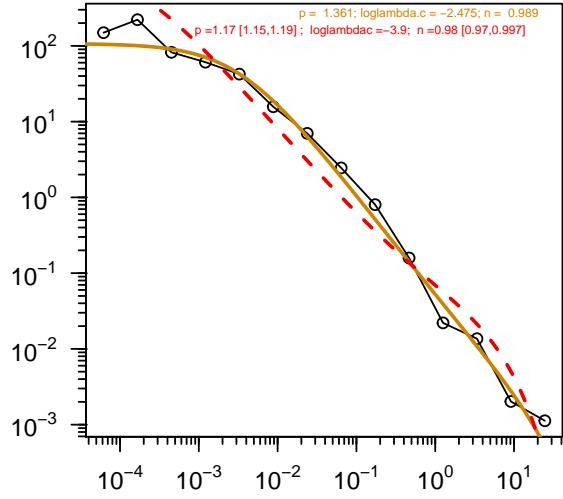
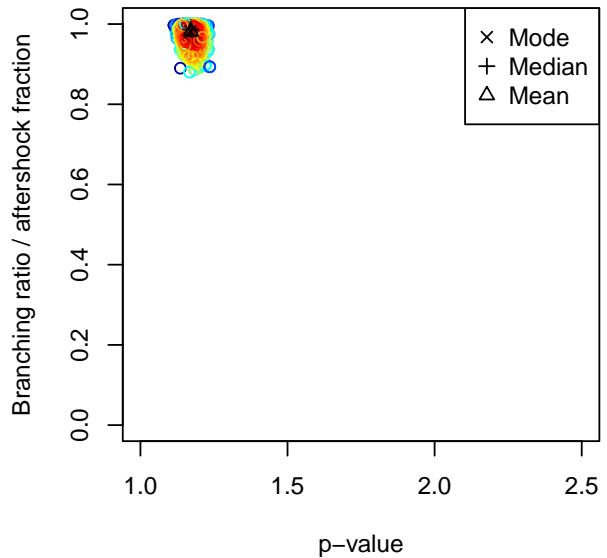
**Density**



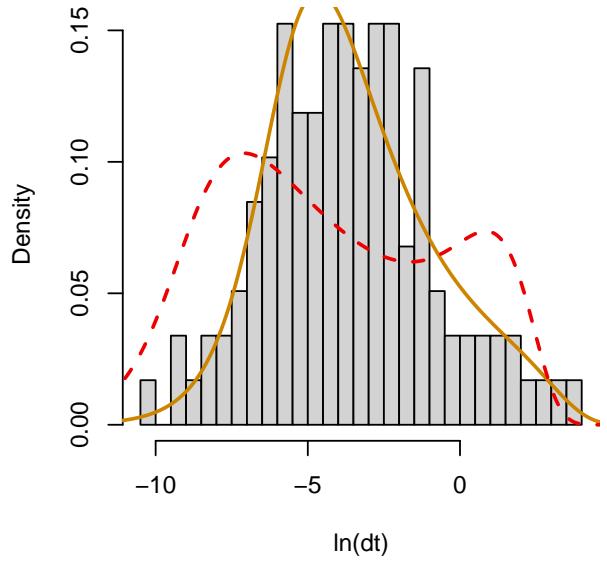
**Family 70132968 ; nev = 119**



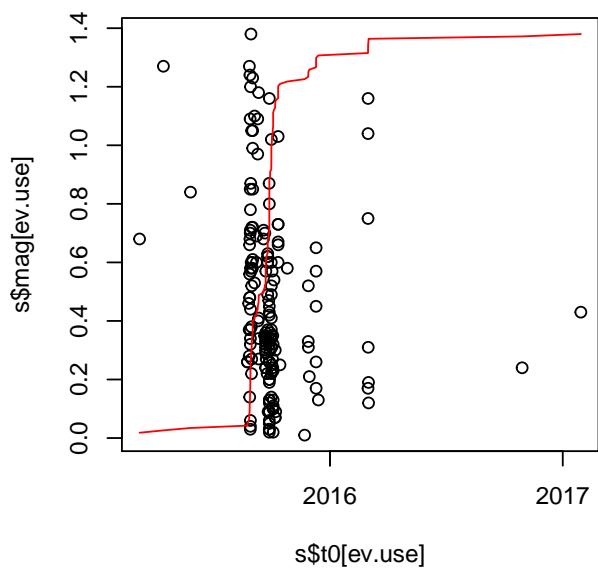
**BFMI = 0.94 ; n\_div = 0  
rhat = 1 ; n\_eff = 1419**



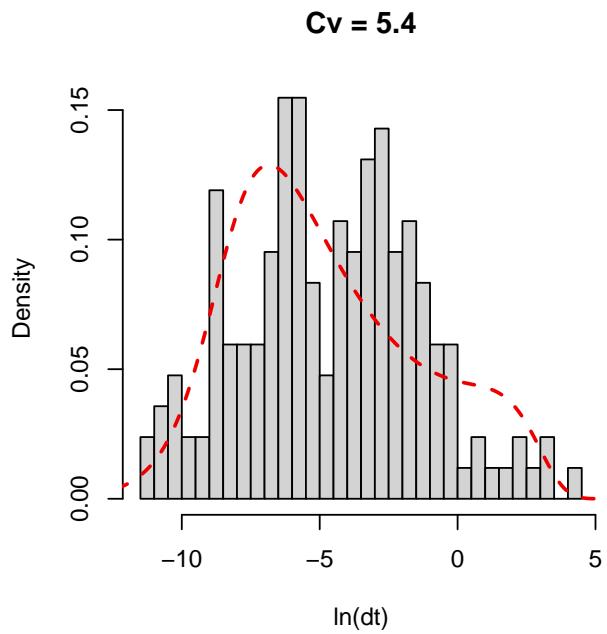
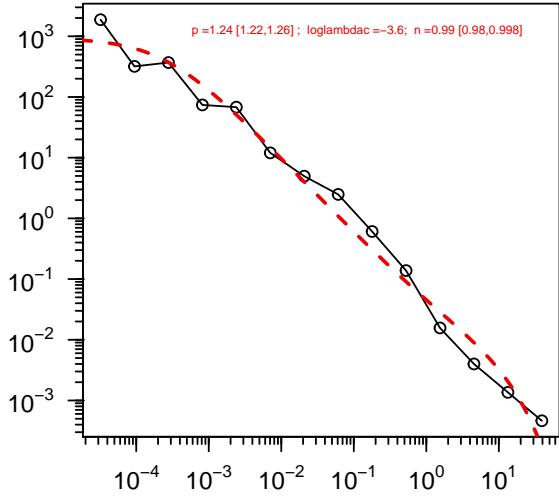
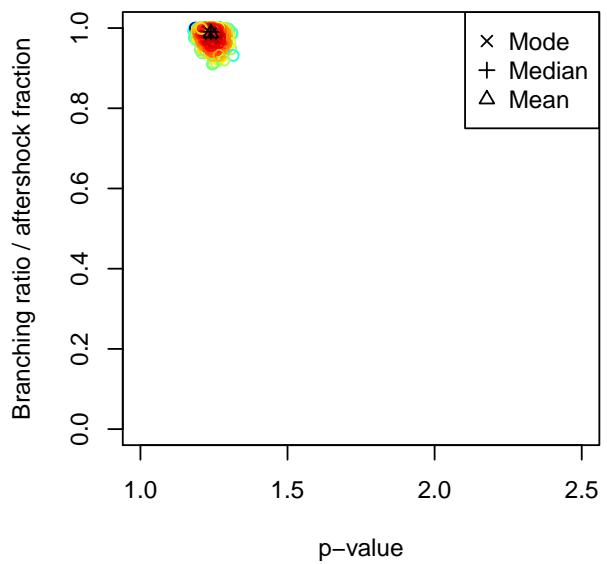
**Cv = 4.3**



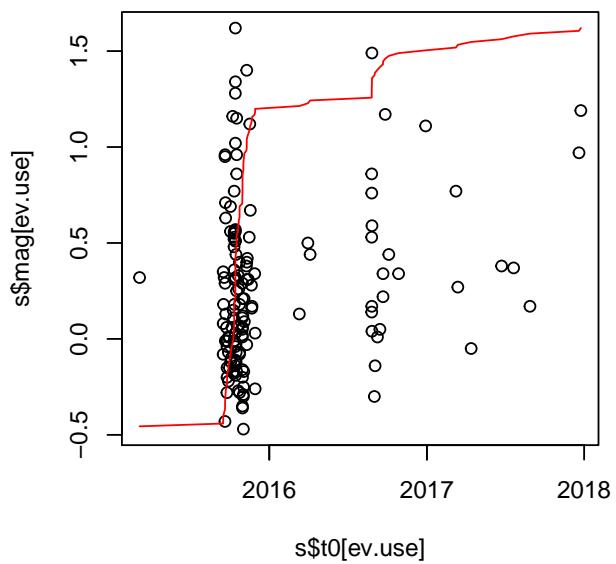
**Family 70139288 ; nev = 169**



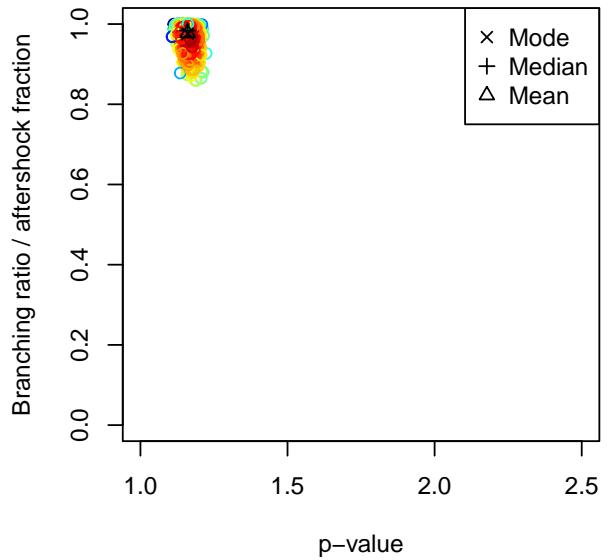
**BFMI = 1 ; n\_div = 0  
rhat = 1 ; n\_eff = 1362**



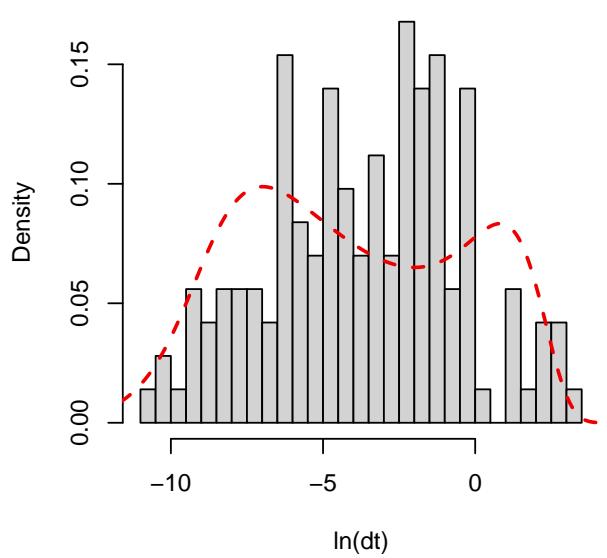
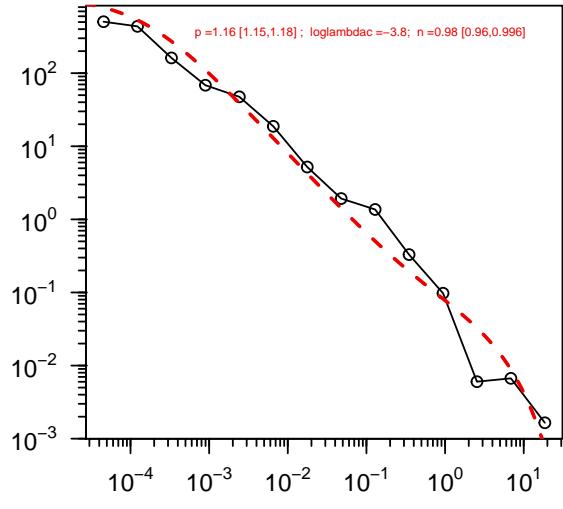
**Family 70144233 ; nev = 144**



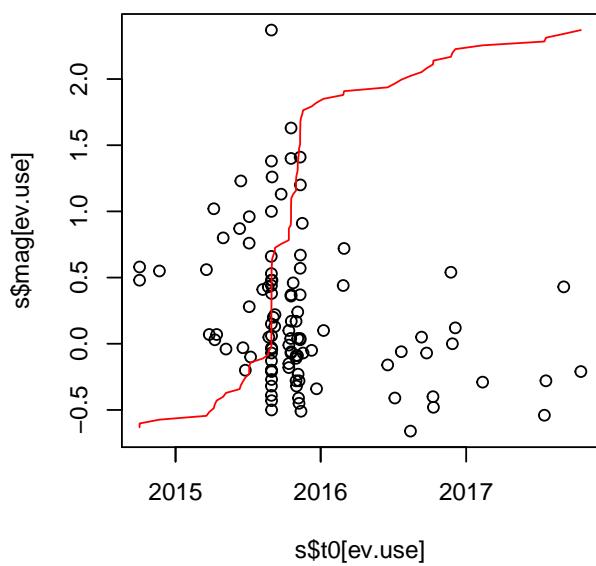
**BFMI = 0.95 ; n\_div = 0  
rhat = 1 ; n\_eff = 1245**



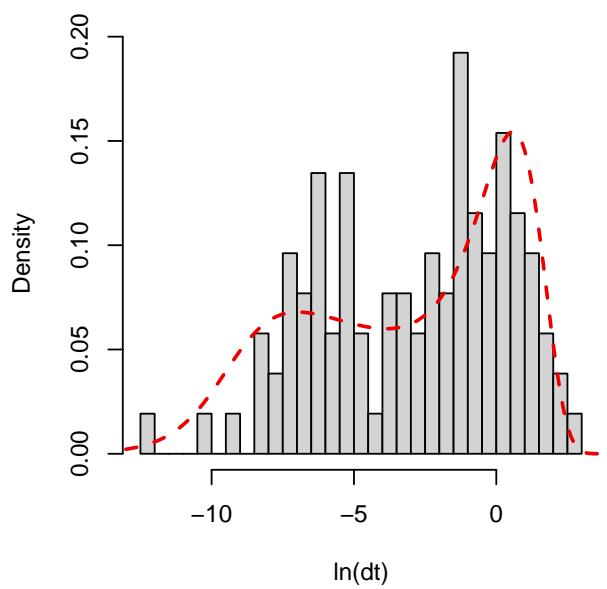
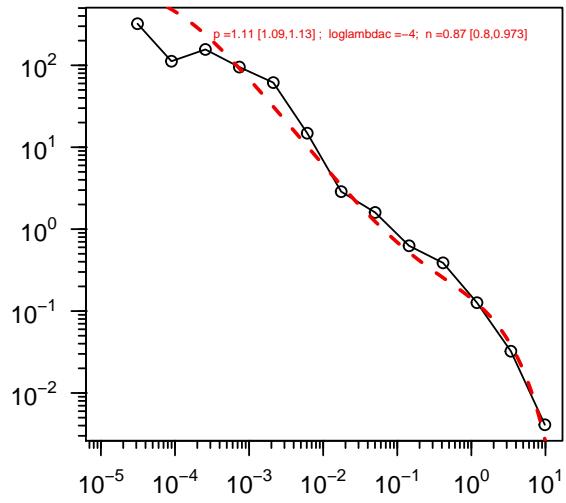
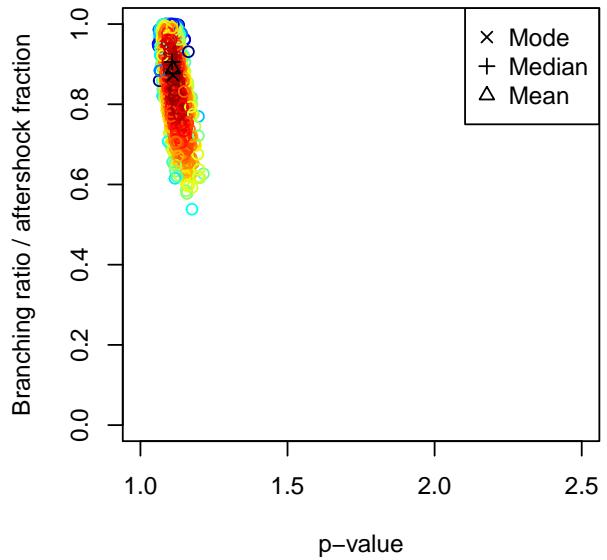
**Cv = 3.6**



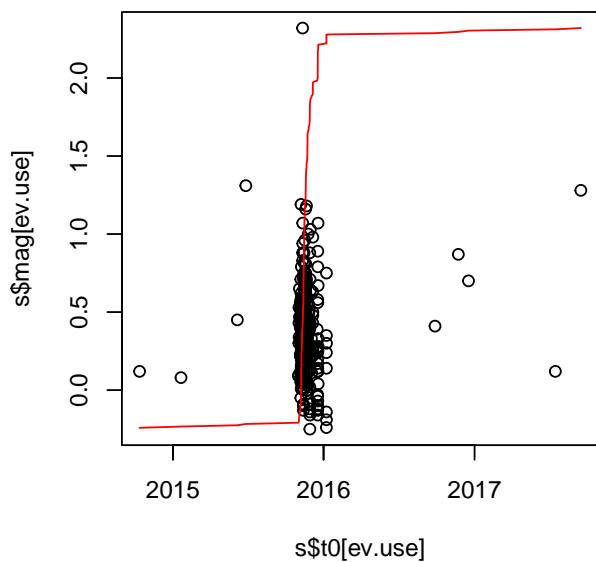
**Family 70145308 ; nev = 105**



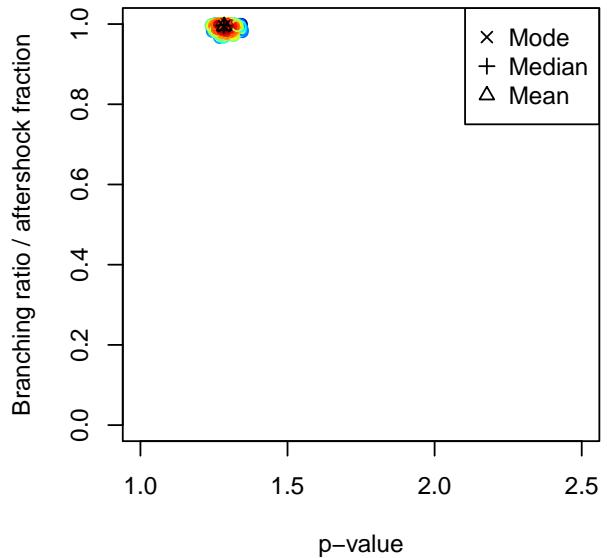
**BFMI = 1.1 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 1033**



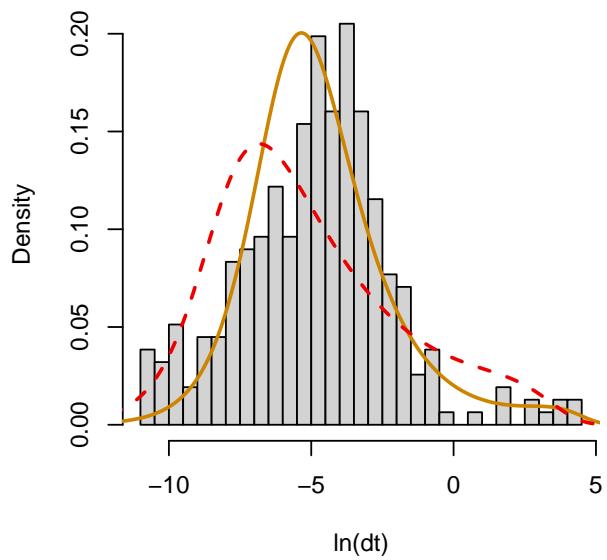
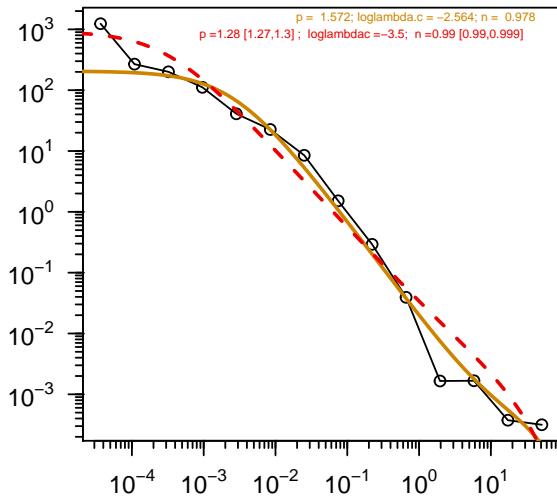
**Family 70152973 ; nev = 313**



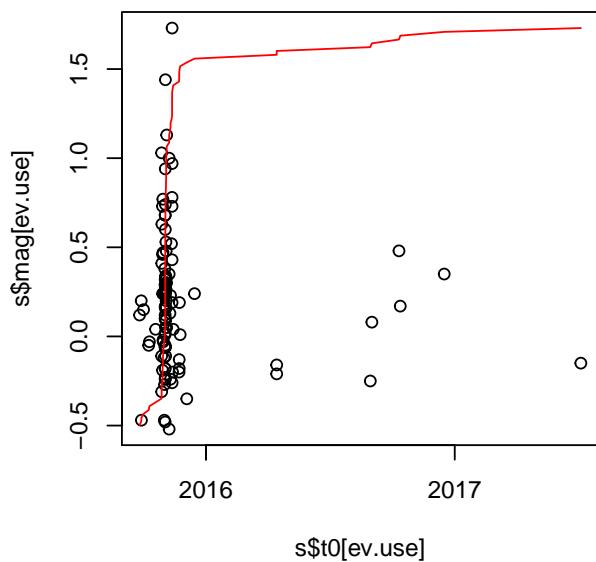
**BFMI = 0.95 ; n\_div = 0  
rhat = 1 ; n\_eff = 1047**



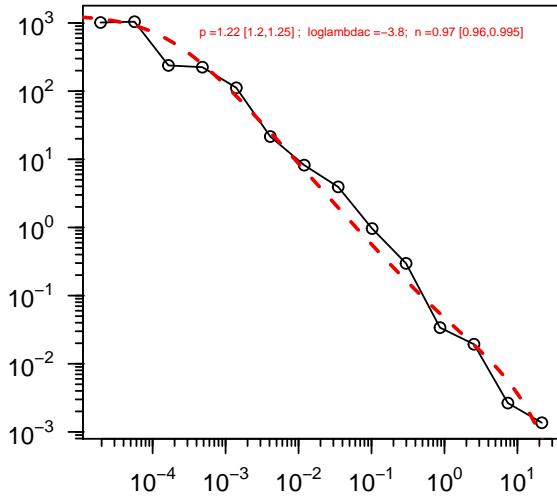
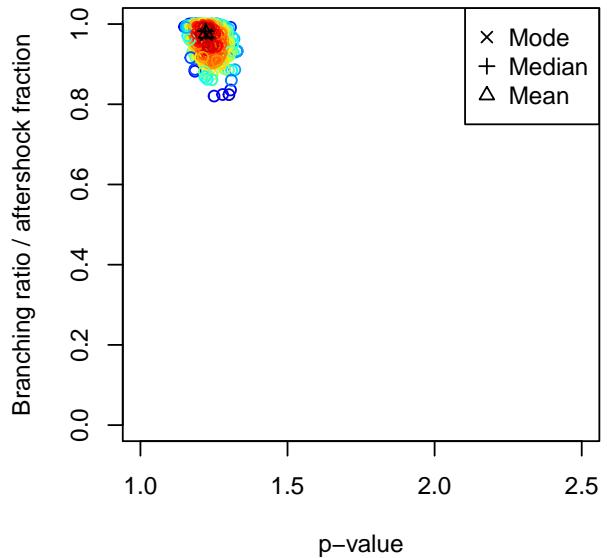
**Cv = 6.7**



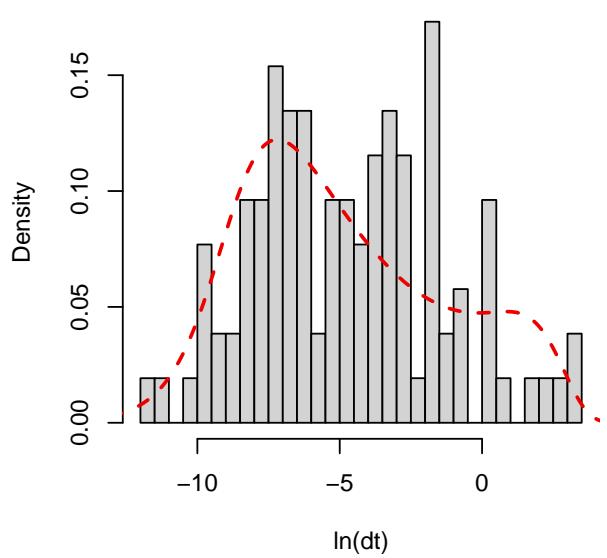
**Family 70153298 ; nev = 105**



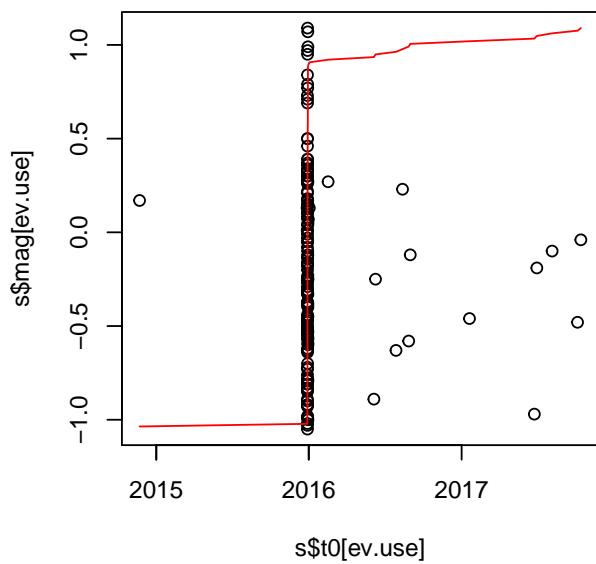
**BFMI = 1.1 ; n\_div = 0  
rhat = 1 ; n\_eff = 1226**



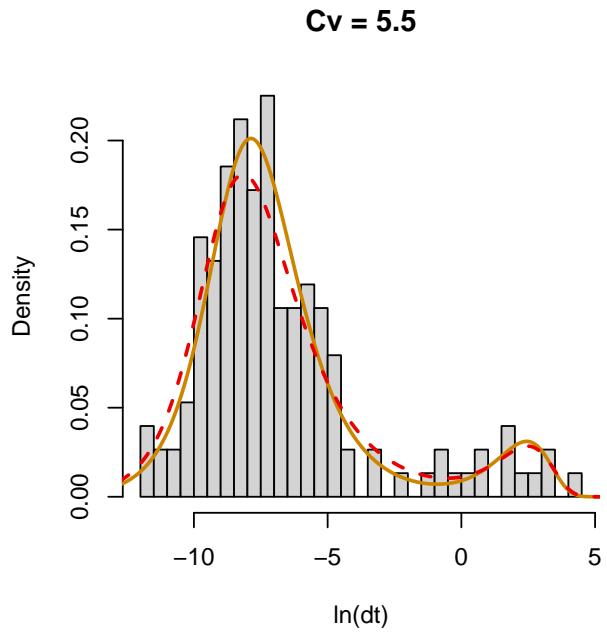
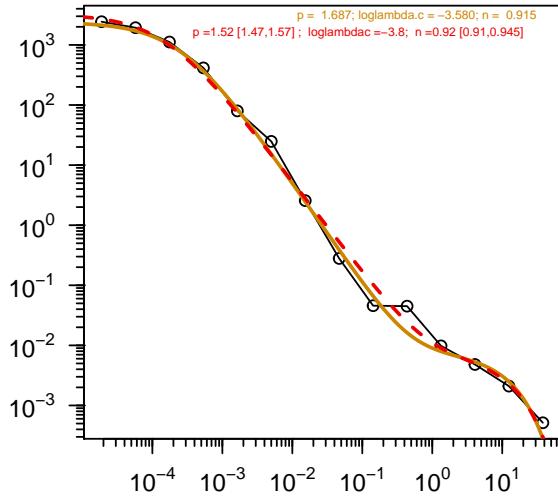
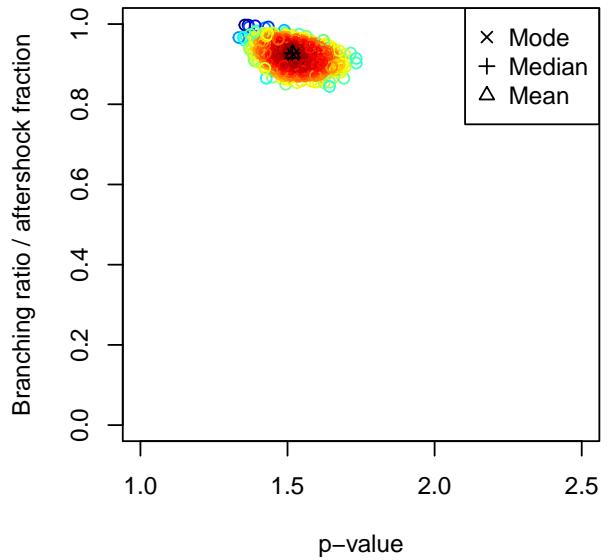
**Cv = 4.4**



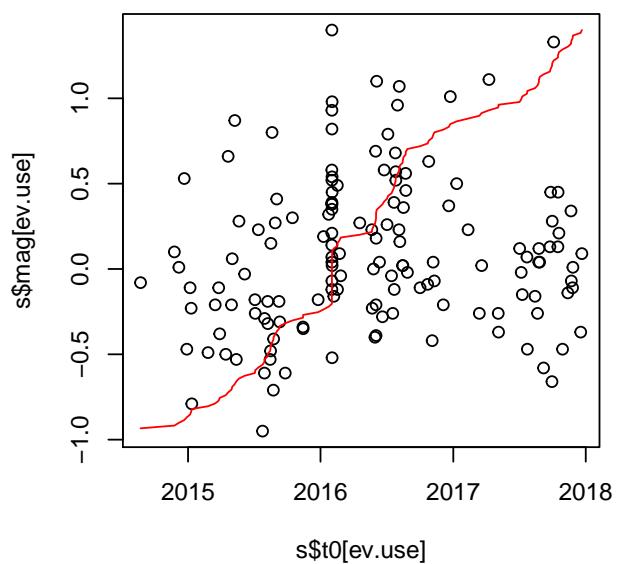
**Family 70164238 ; nev = 152**



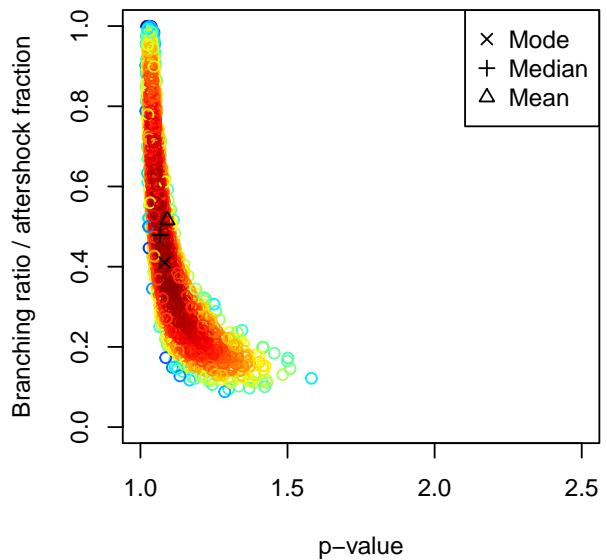
**BFMI = 1.1 ; n\_div = 0  
rhat = 1 ; n\_eff = 1230**



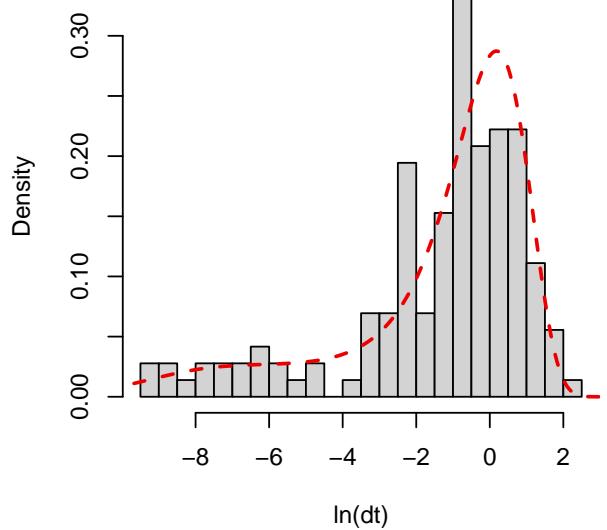
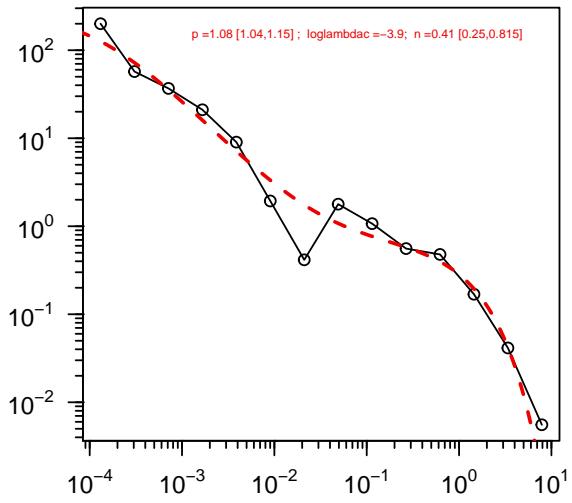
**Family 70170133 ; nev = 145**



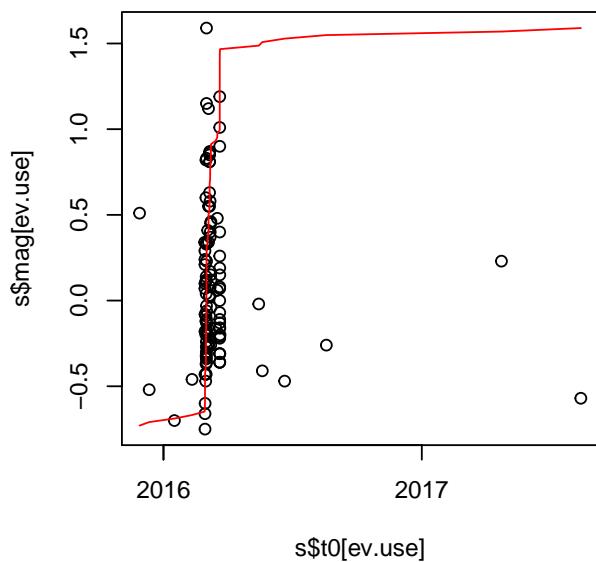
**BFMI = 1 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 789**



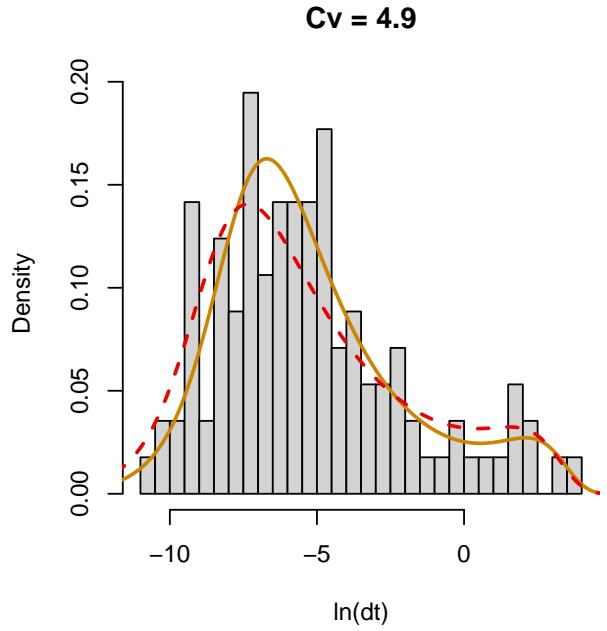
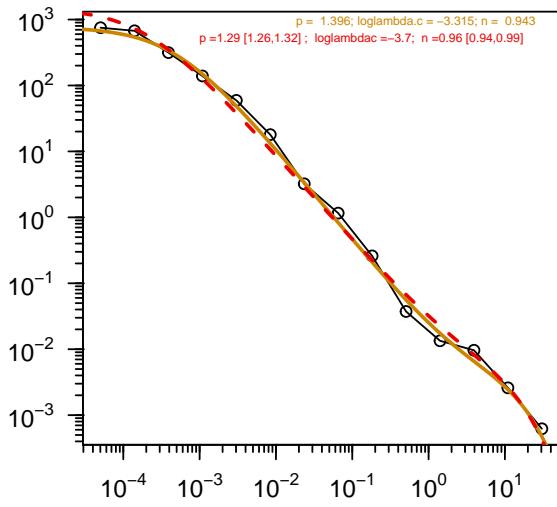
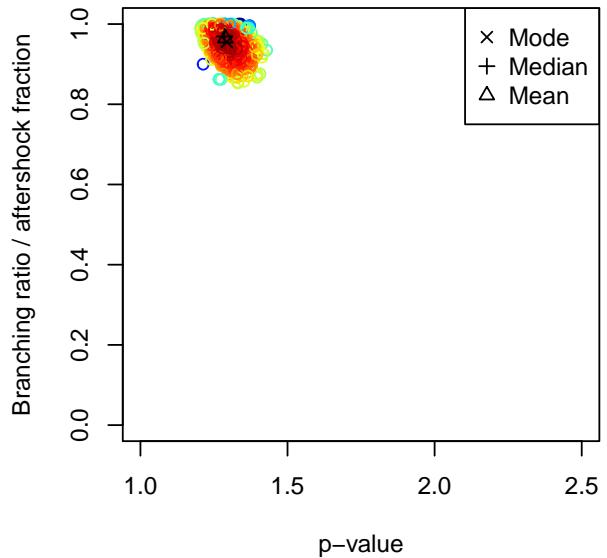
**Cv = 1.5**



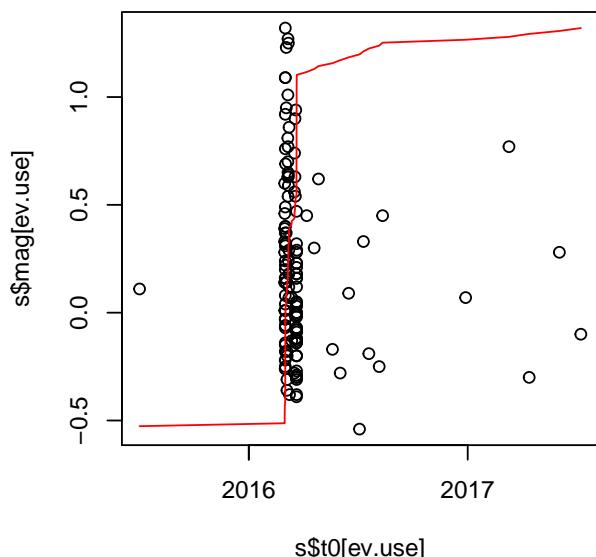
**Family 70174828 ; nev = 114**



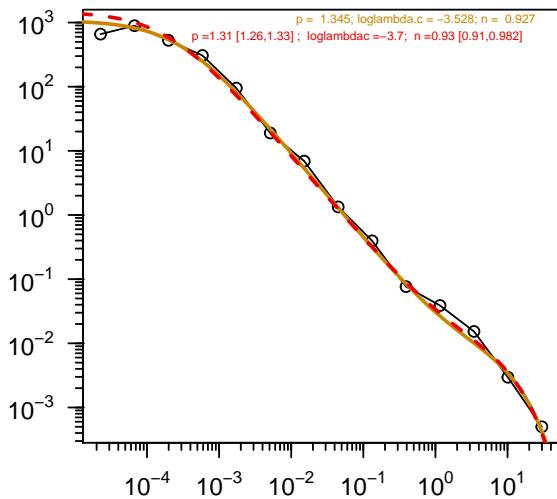
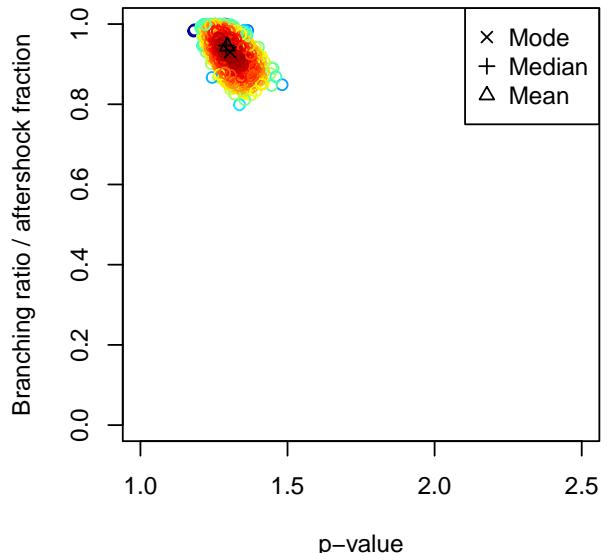
**BFMI = 1.1 ; n\_div = 0  
rhat = 1 ; n\_eff = 1118**



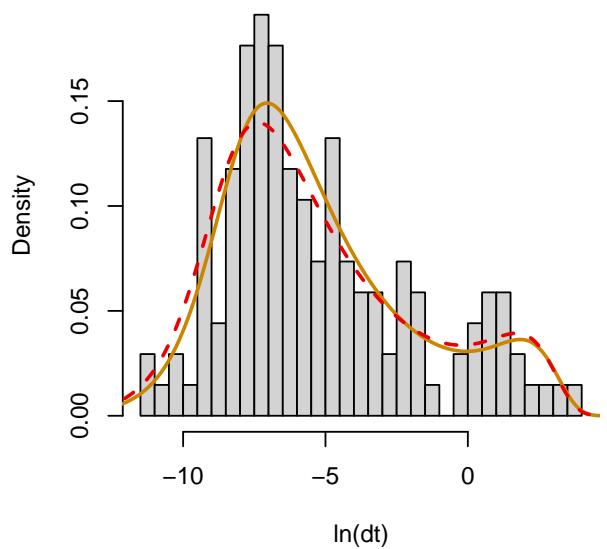
**Family 70174863 ; nev = 137**



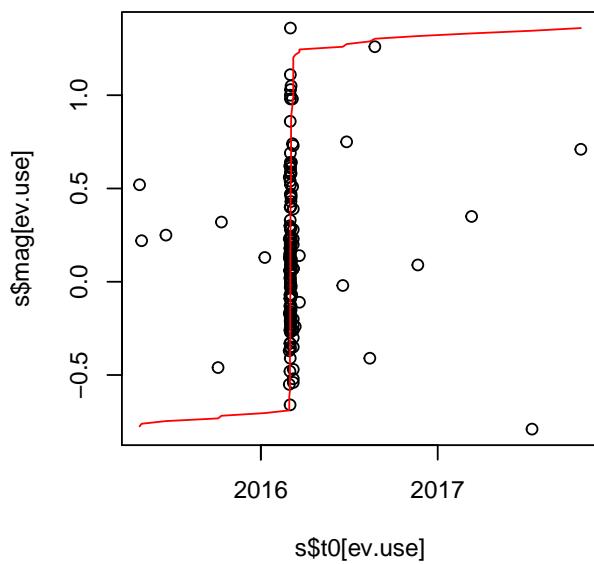
**BFMI = 0.97 ; n\_div = 0  
rhat = 1 ; n\_eff = 935**



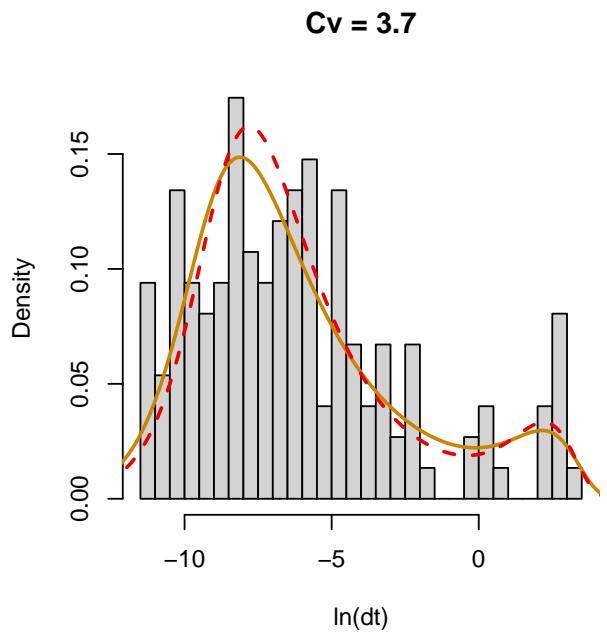
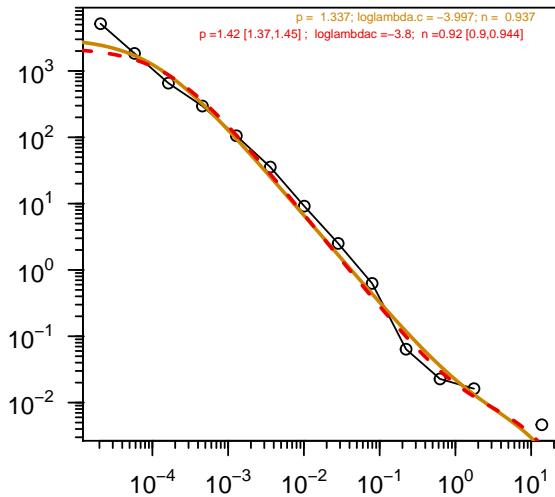
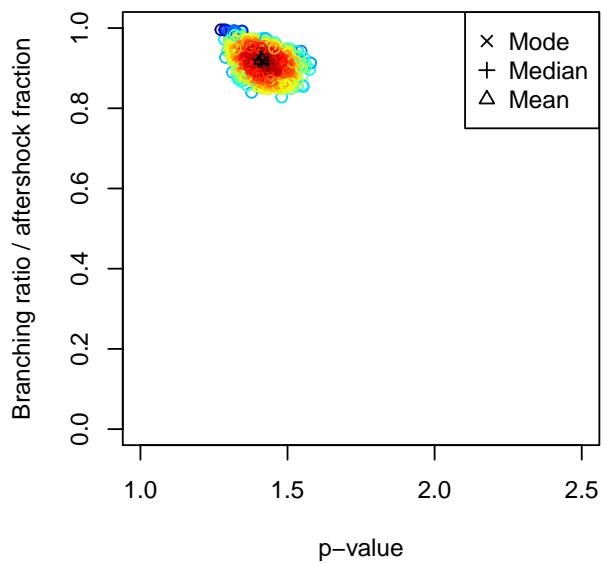
**Cv = 4.7**



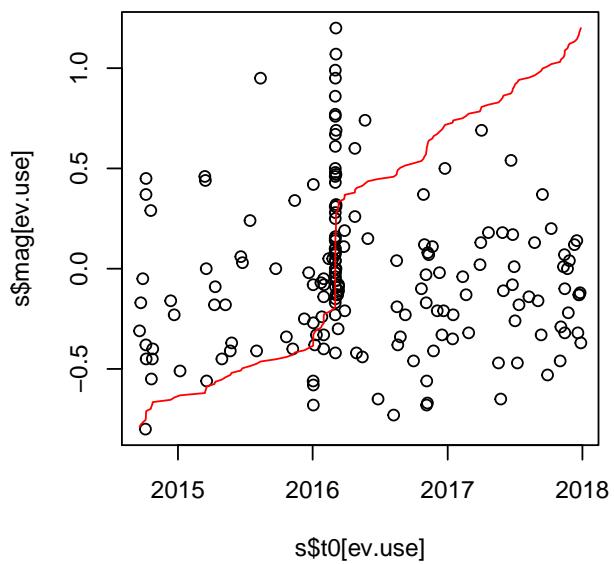
**Family 70174893 ; nev = 150**



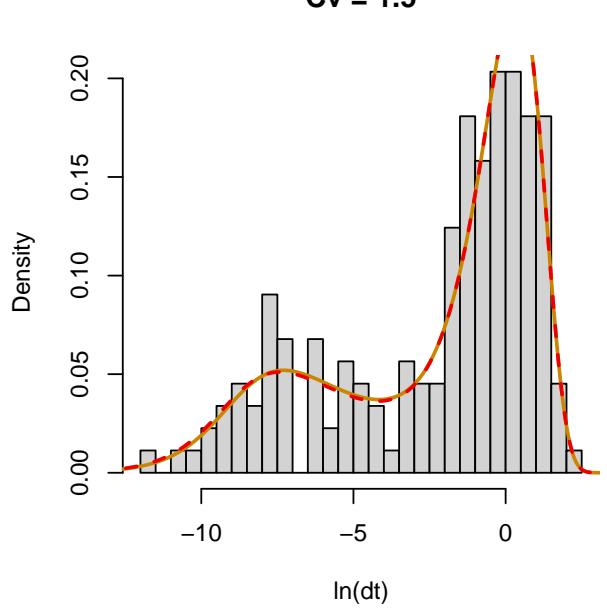
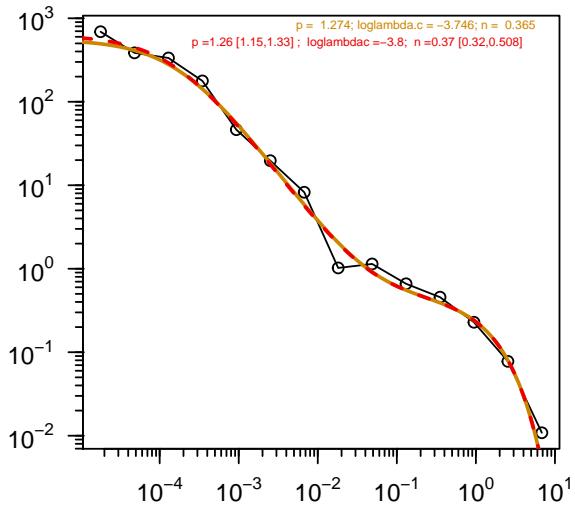
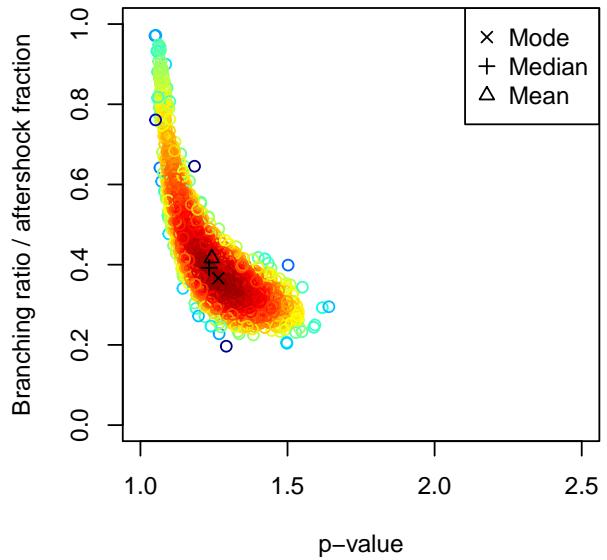
**BFMI = 1 ; n\_div = 0  
rhat = 1 ; n\_eff = 1071**



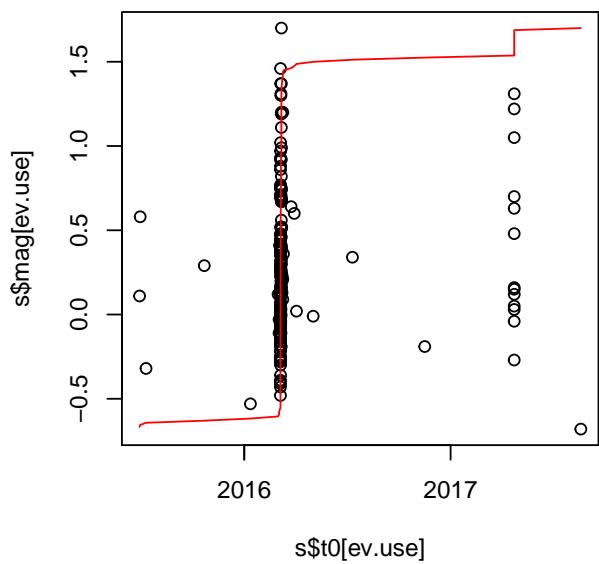
**Family 70174958 ; nev = 178**



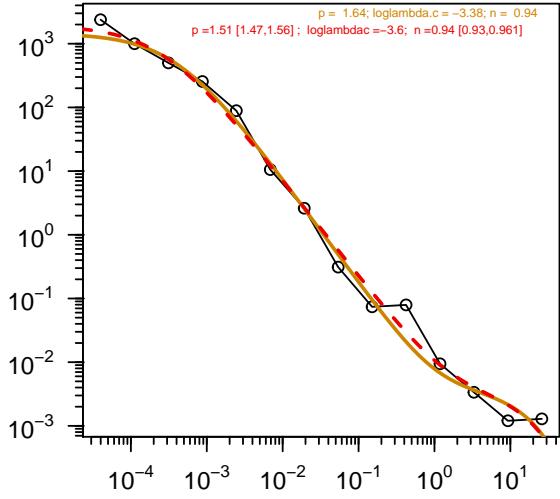
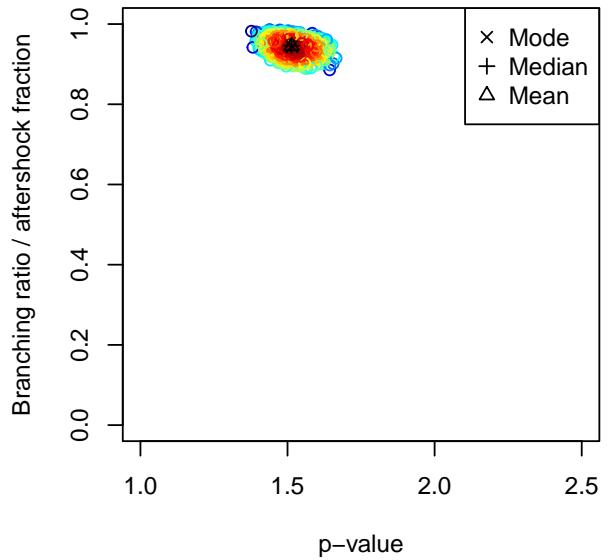
**BFMI = 1.1 ; n\_div = 0  
rhat = 1 ; n\_eff = 764**



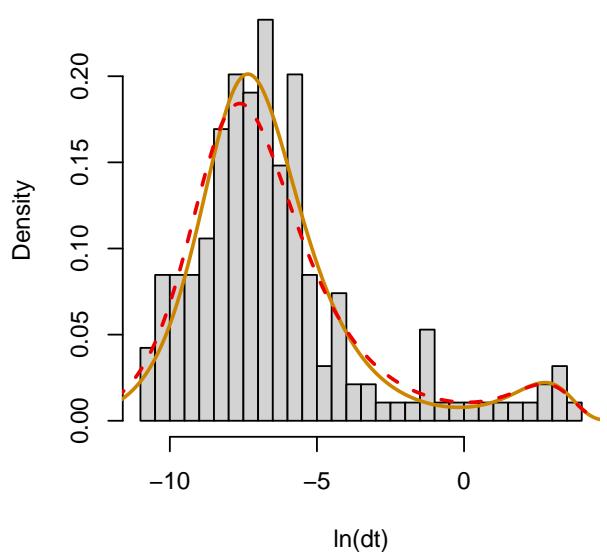
**Family 70175408 ; nev = 190**



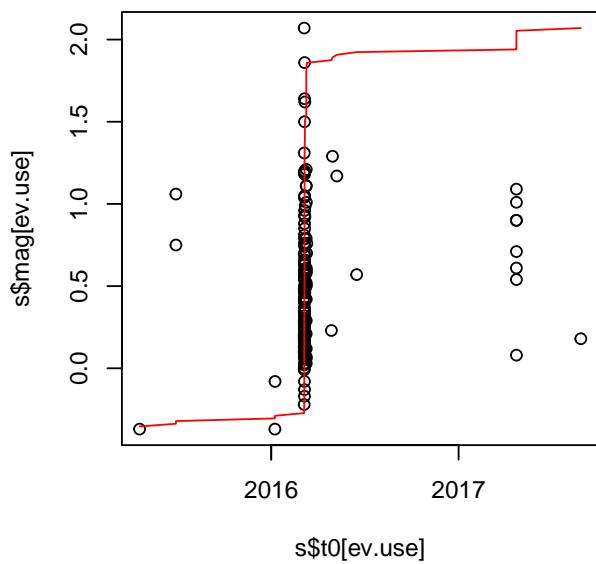
**BFMI = 0.86 ; n\_div = 0  
rhat = 1 ; n\_eff = 1649**



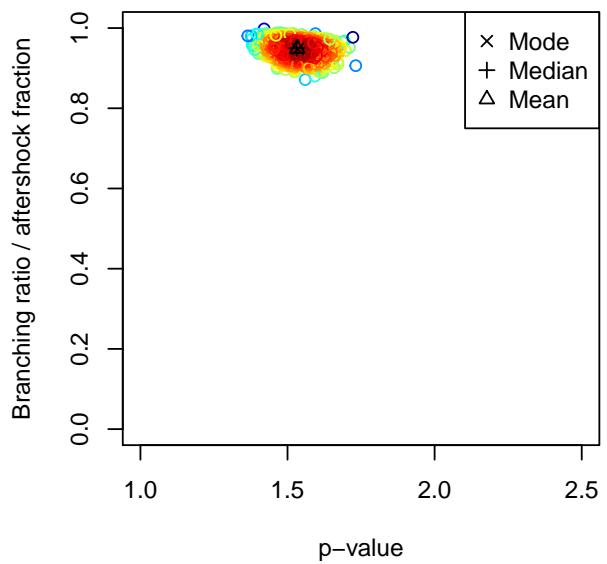
**Cv = 4.9**



**Family 70175558 ; nev = 150**



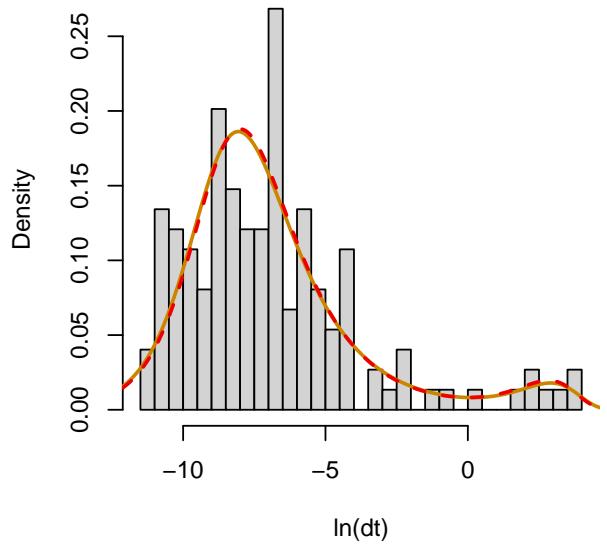
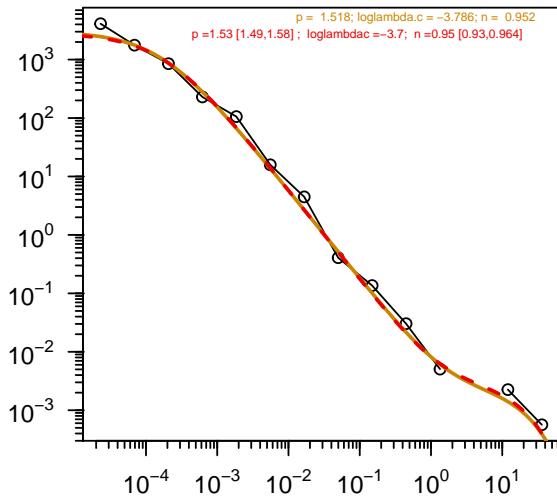
**BFMI = 1.1 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 1439**



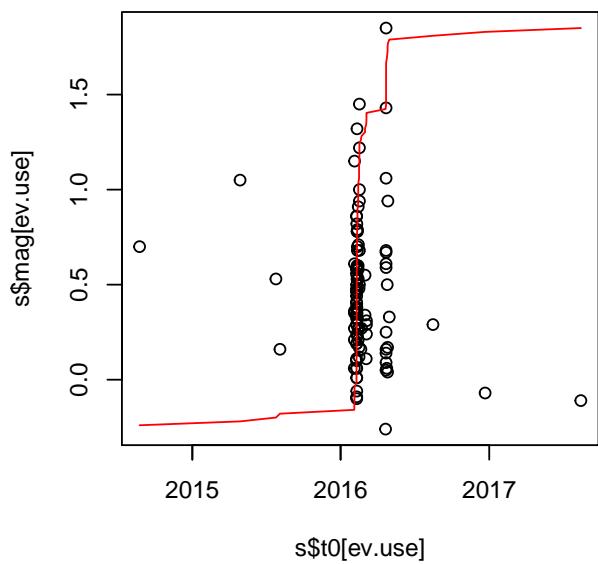
\$\\$t0[ev.use]

p-value

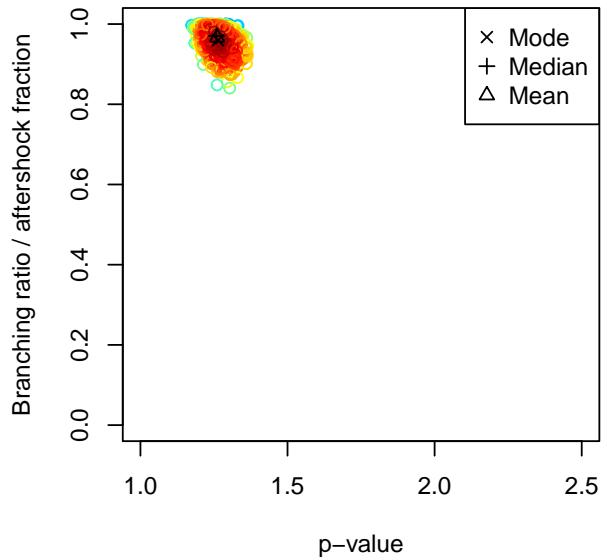
**Cv = 5.6**



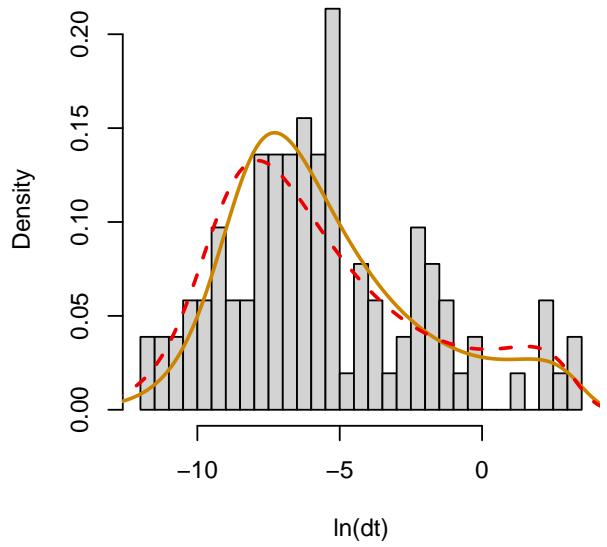
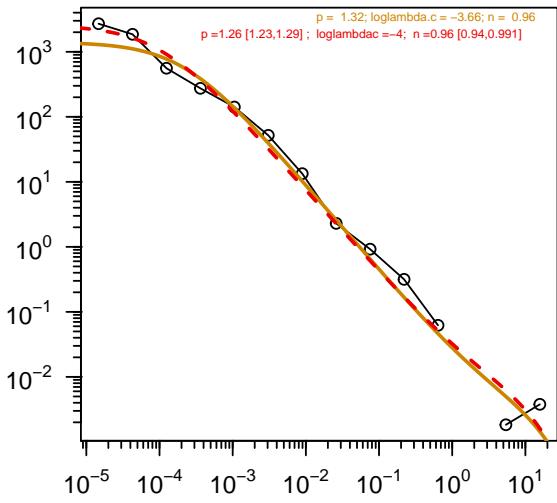
**Family 70182583 ; nev = 104**



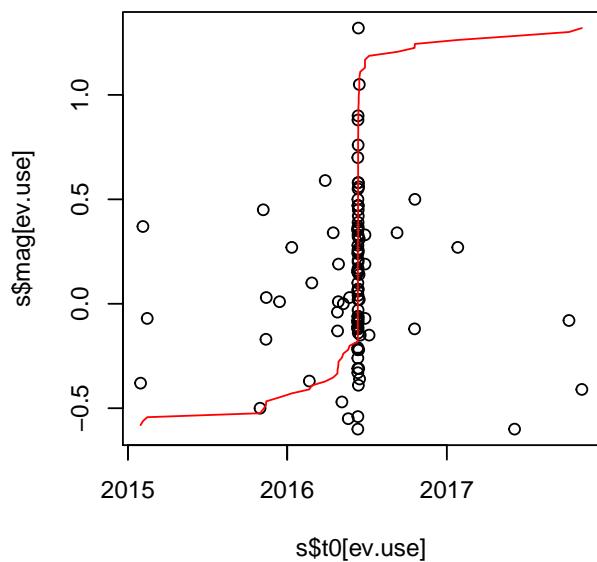
**BFMI = 0.85 ; n\_div = 0  
rhat = 1 ; n\_eff = 858**



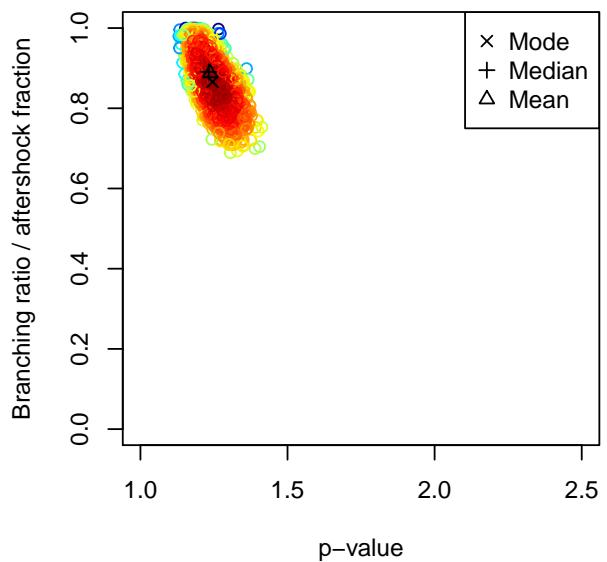
**Cv = 4**



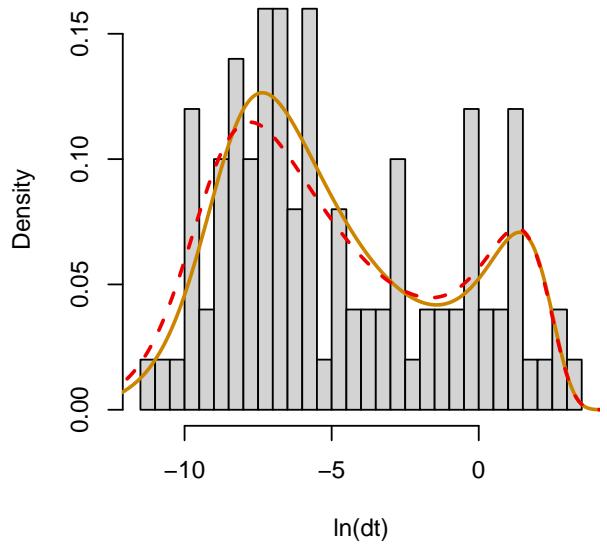
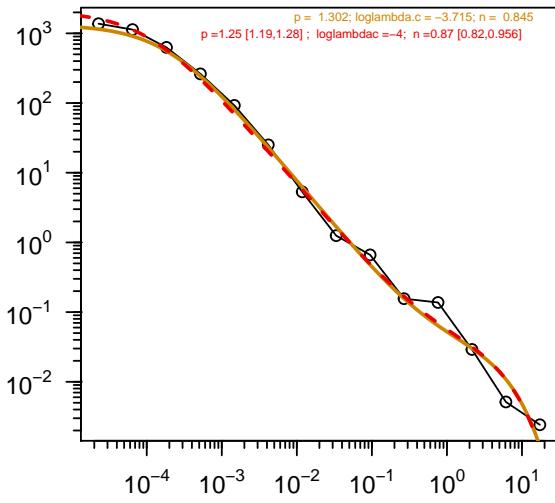
**Family 70191283 ; nev = 101**



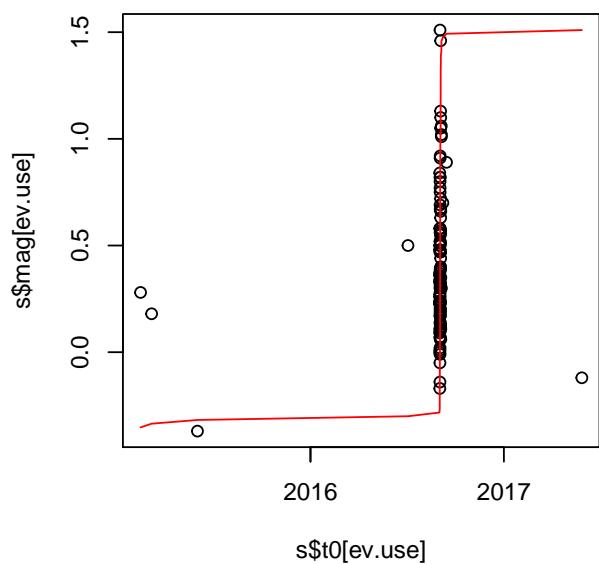
**BFMI = 0.95 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 884**



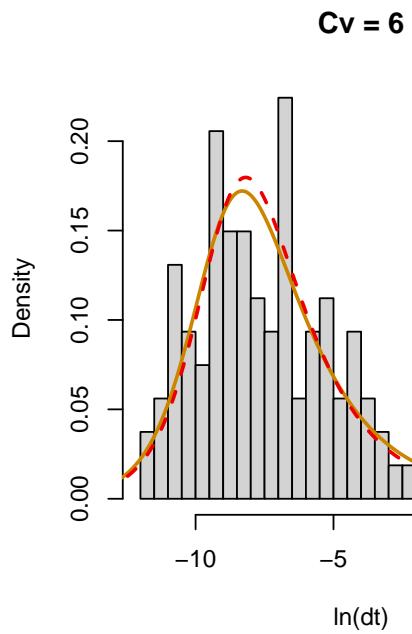
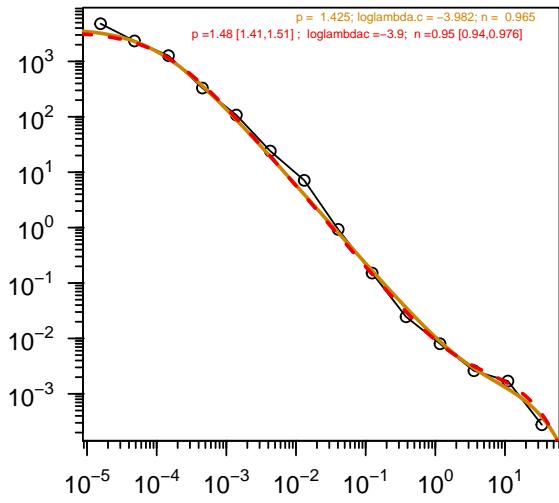
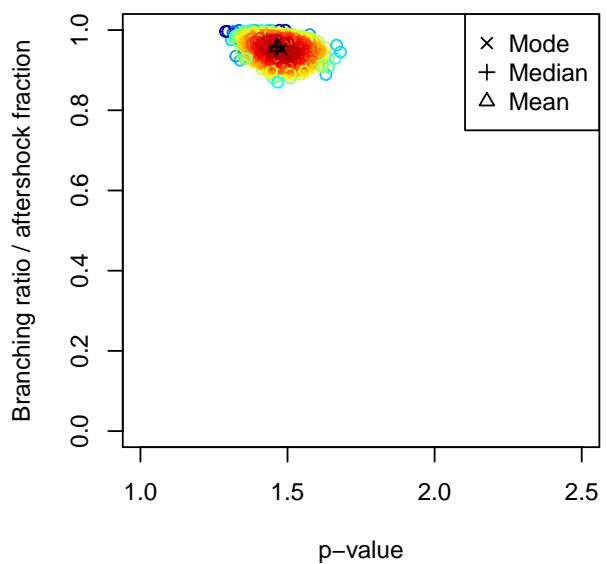
**Cv = 3.3**



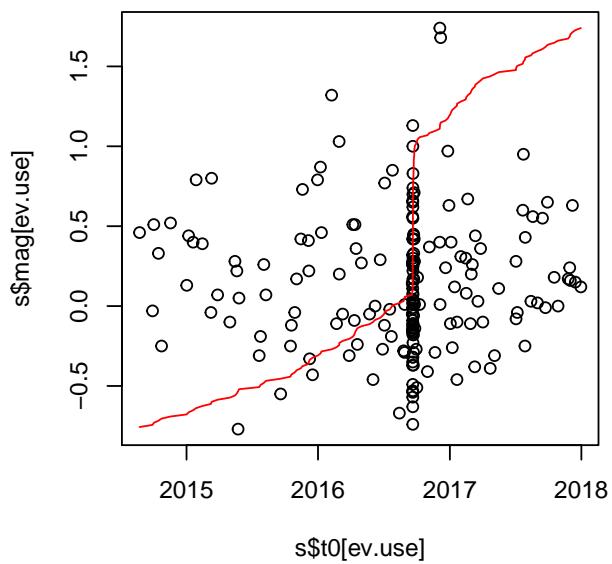
**Family 70205393 ; nev = 108**



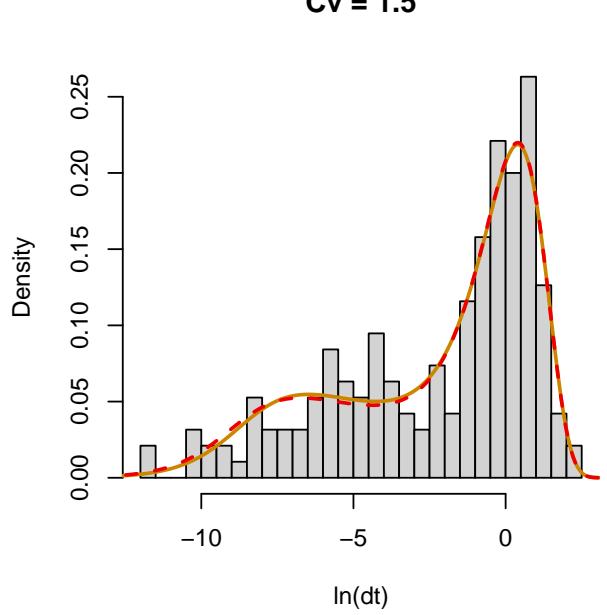
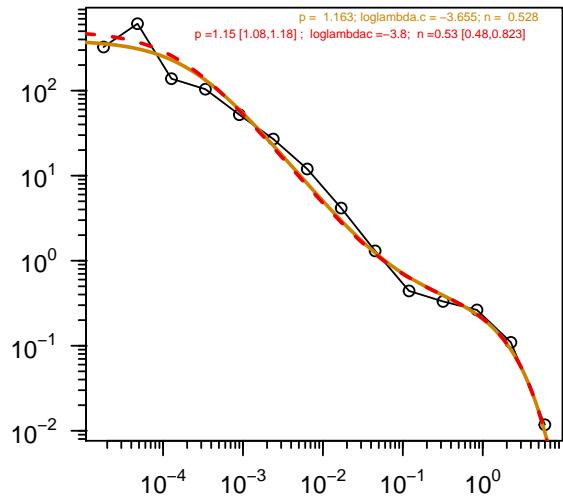
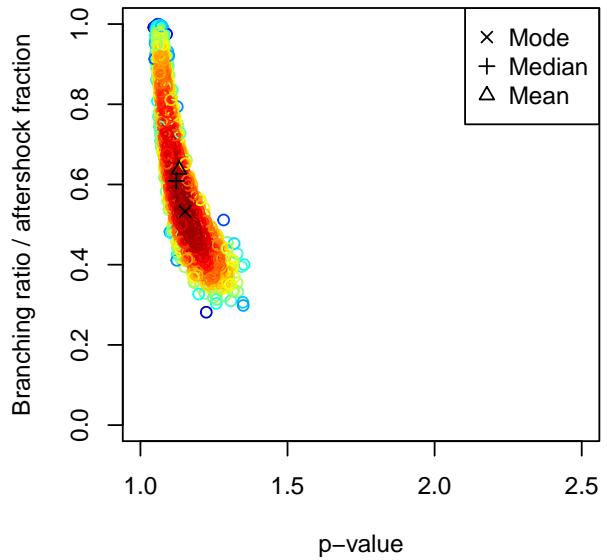
**BFMI = 1 ; n\_div = 0  
rhat = 1 ; n\_eff = 1003**



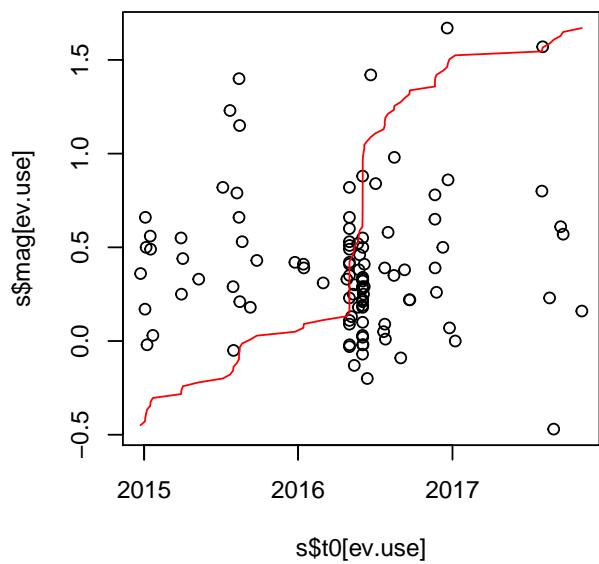
**Family 70207983 ; nev = 191**



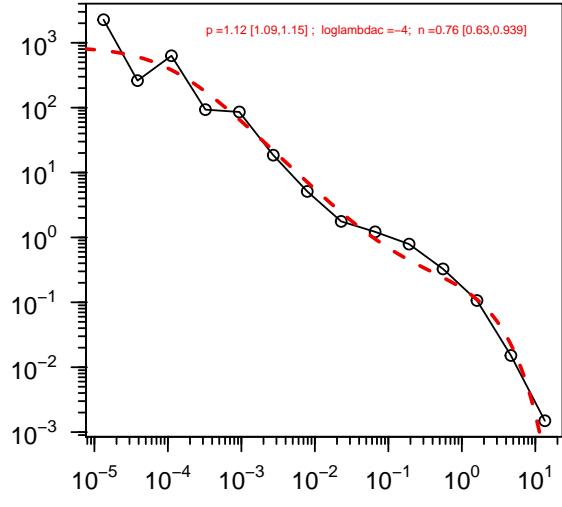
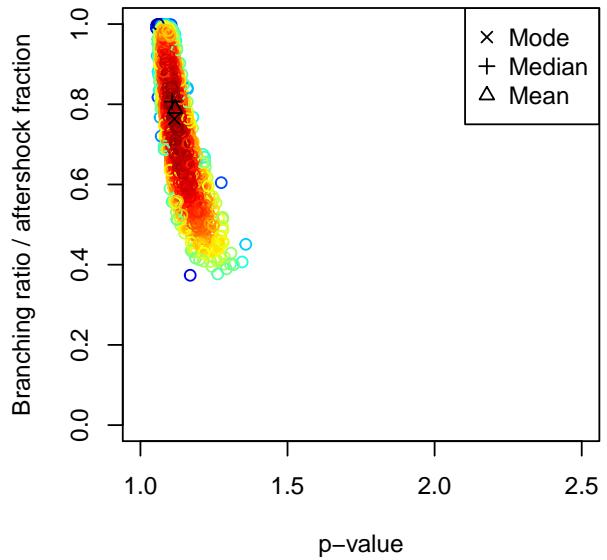
**BFMI = 1.1 ; n\_div = 0  
rhat = 1 ; n\_eff = 567**



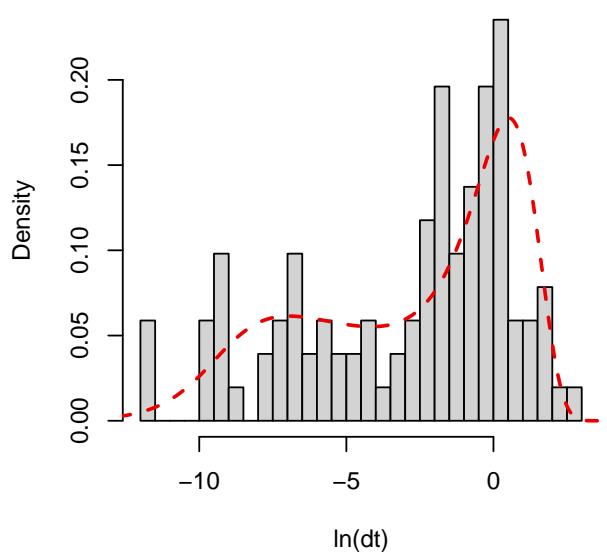
**Family 70215743 ; nev = 103**



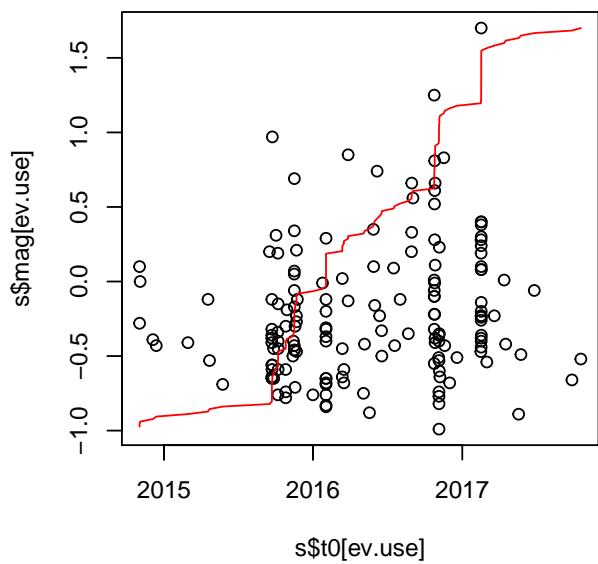
**BFMI = 1.1 ; n\_div = 0  
rhat = 1 ; n\_eff = 654**



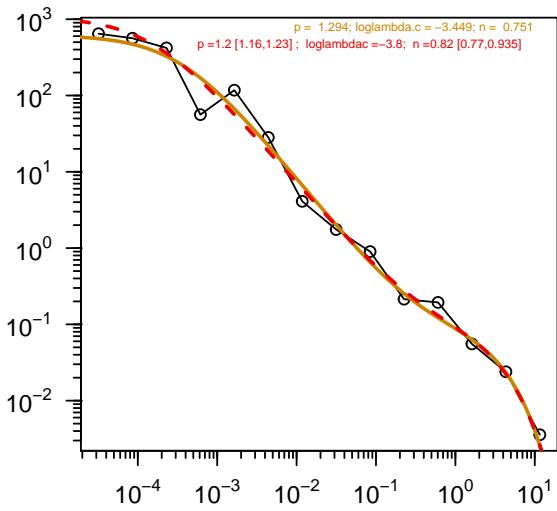
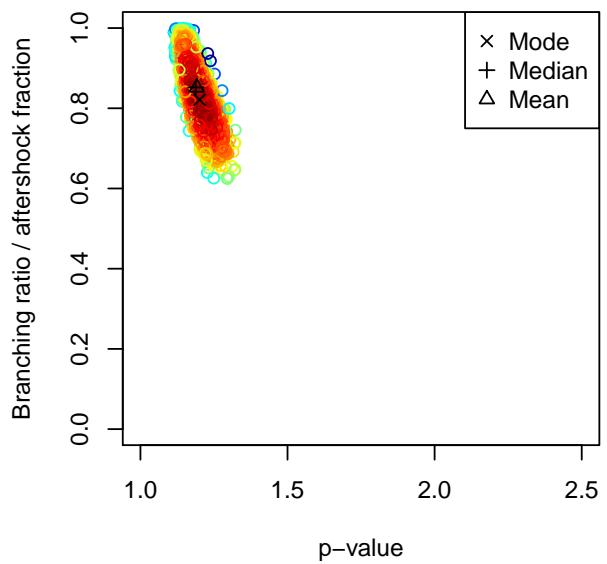
**Cv = 2.5**



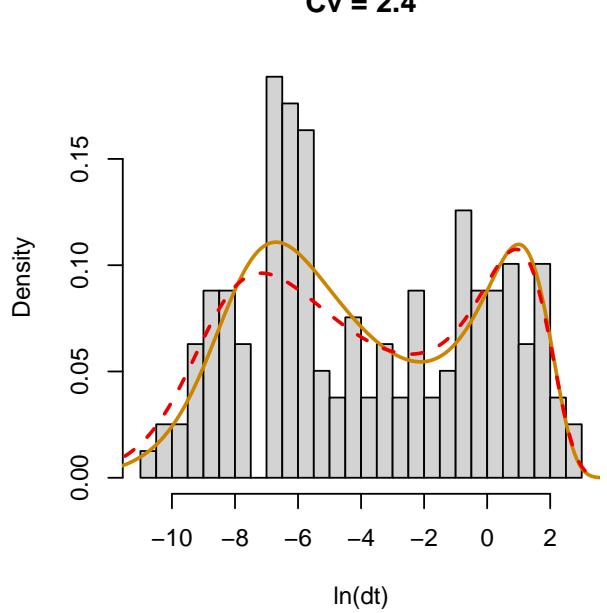
**Family 70220983 ; nev = 160**



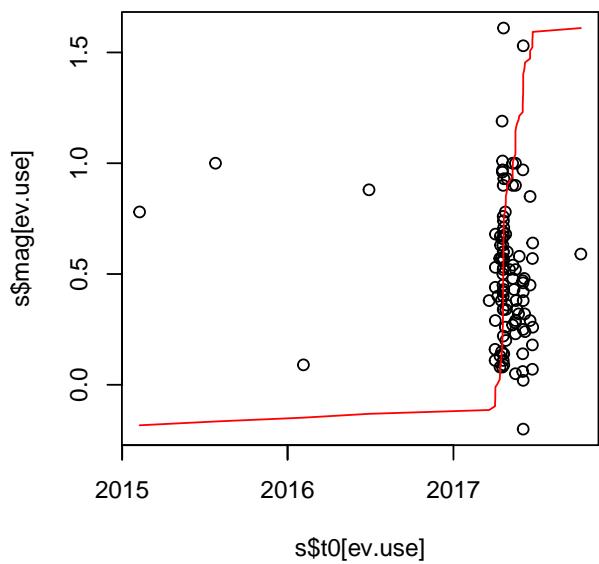
**BFMI = 0.87 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 757**



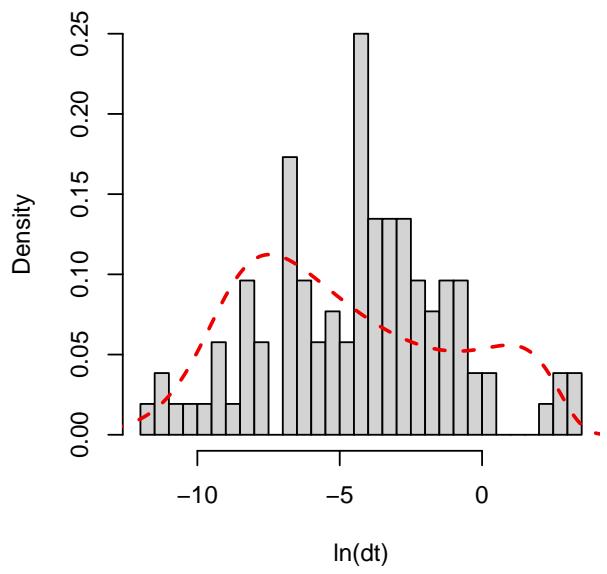
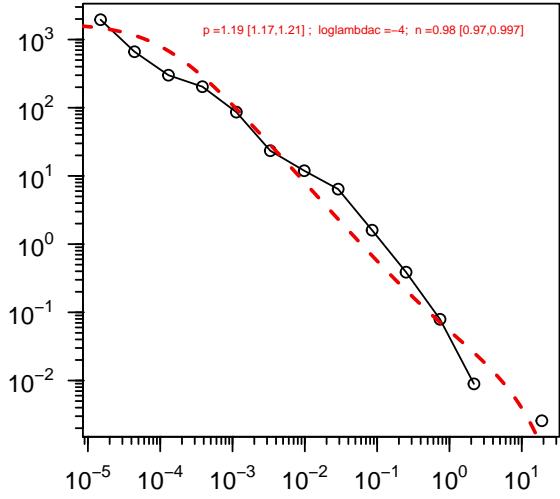
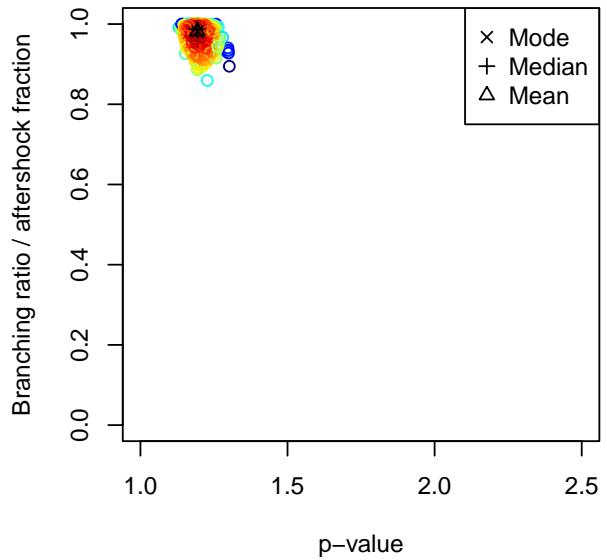
**Cv = 2.4**



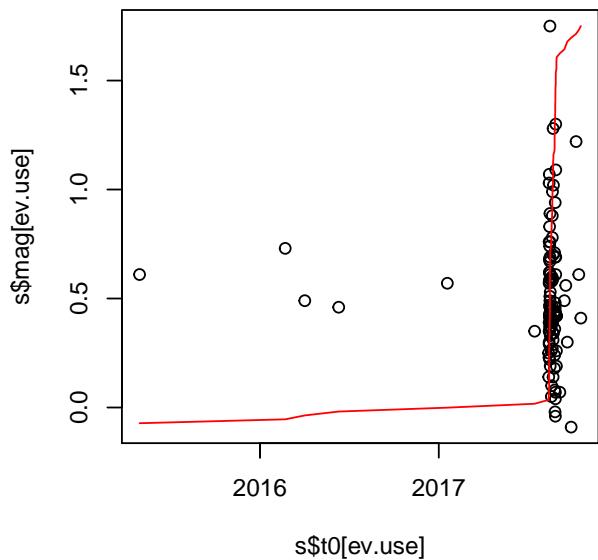
**Family 70225248 ; nev = 105**



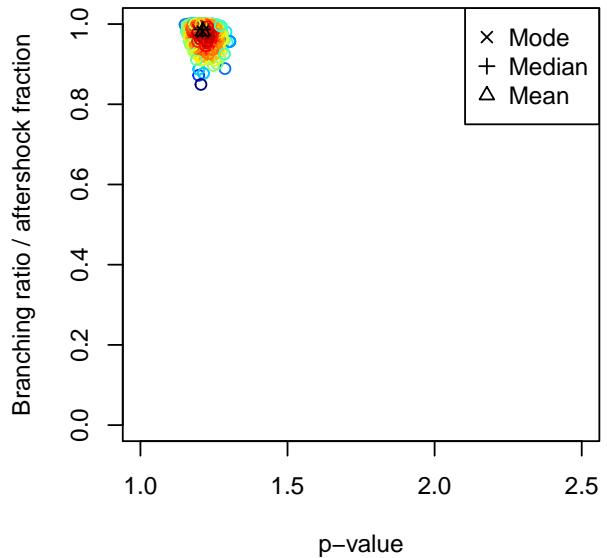
**BFMI = 1.1 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 1175**



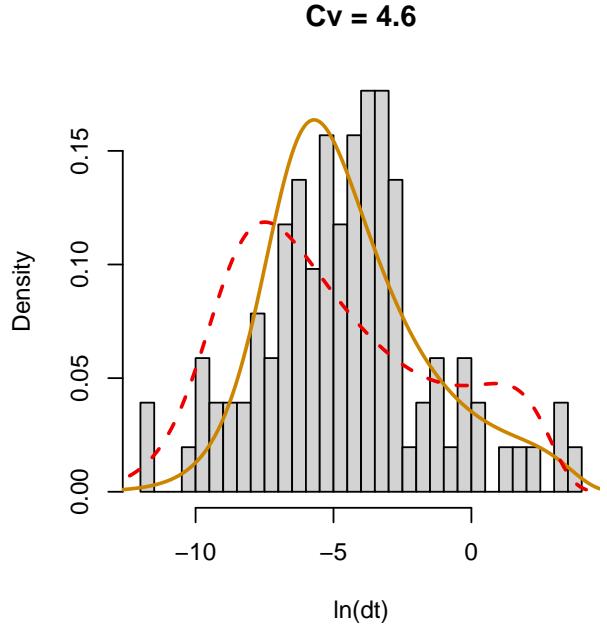
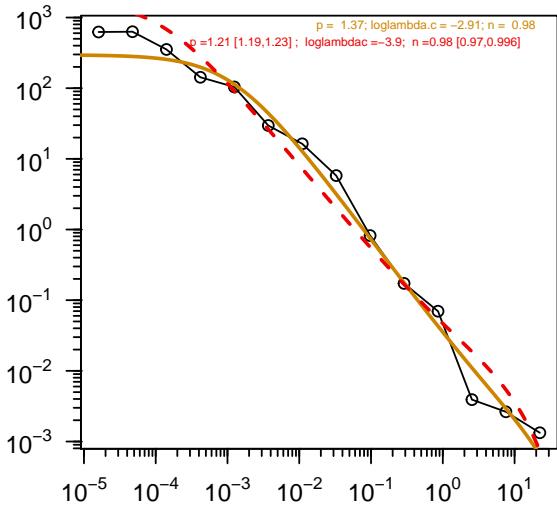
**Family 70233158 ; nev = 103**



**BFMI = 0.96 ; n\_div = 0  
rhat = 1.01 ; n\_eff = 1087**



**Cv = 4.6**



## Some puzzling examples

### Mode outside 68% CIs

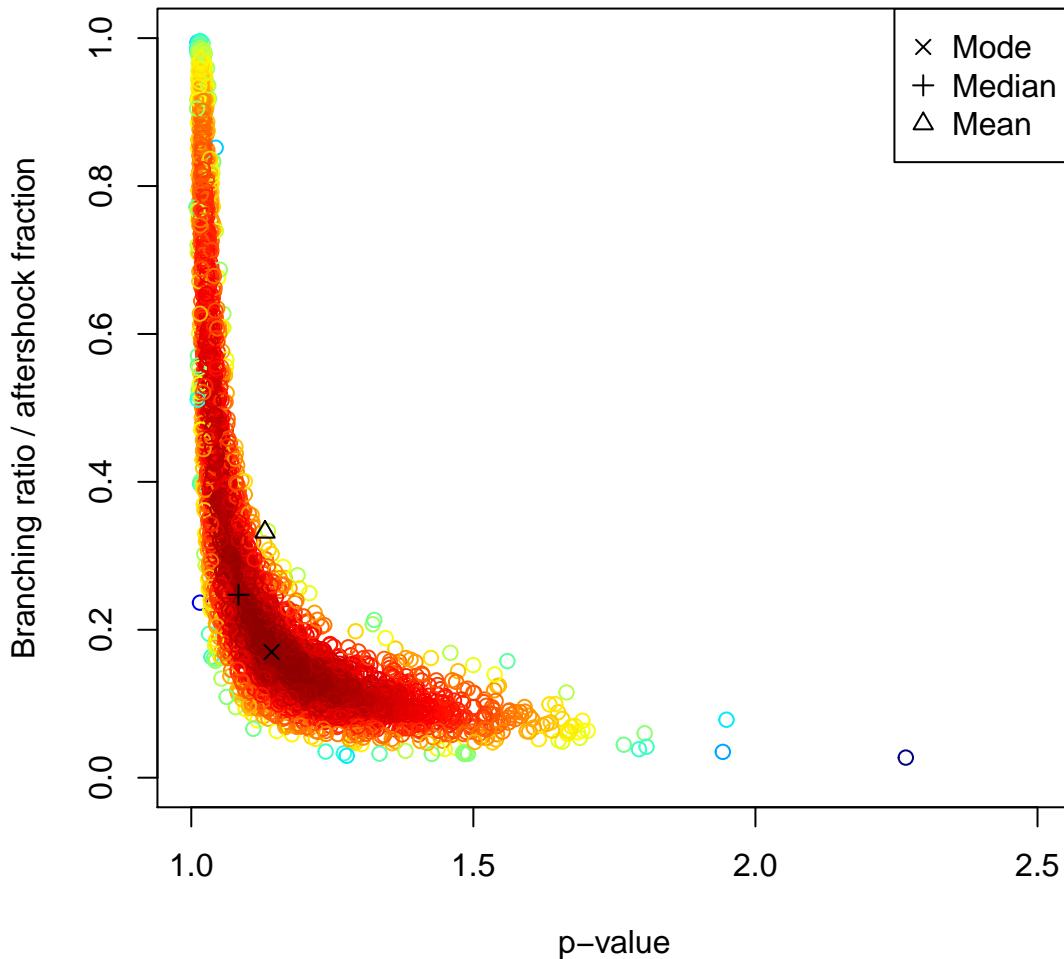
I was originally using the mode (more precisely, the point in parameter space in the MCMC sample that had the highest value of the posterior pdf) as the central estimates of the parameters. There are cases where this mode falls outside the confidence intervals. That is, even though this is only a 3-dimensional parameter space, we can still have cases where the volume around the mode is small and far from central part of the “typical set” where most of the total probability lies.

Below is a plot of the MCMC samples to show what is going on. Note that the samples are colored by the value of the likelihood at those values (with “hotter” color indicating higher likelihoods).

```
fam = 70054108
f = match(fam, fids)
i.f = match(f, fuse)

mkMCMCPParamSpacePlot(f.stan[[i.f]], bfmi[i.f], n_div[i.f], rhat[i.f], n_eff[i.f])
```

**BFMI = 1.2 ; n\_div = 0**  
**rhat = 1.01 ; n\_eff = 687**



From this example, it might seem like using, *e.g.* the medians of the marginal pdfs of the MCMC sampled parameters as the central values, but depending on the shape of the typical set, the means/medians are sometimes not in the typical set at all:

```
fam = 70067948
f = match(fam, fids)
i.f = match(f, fuse)

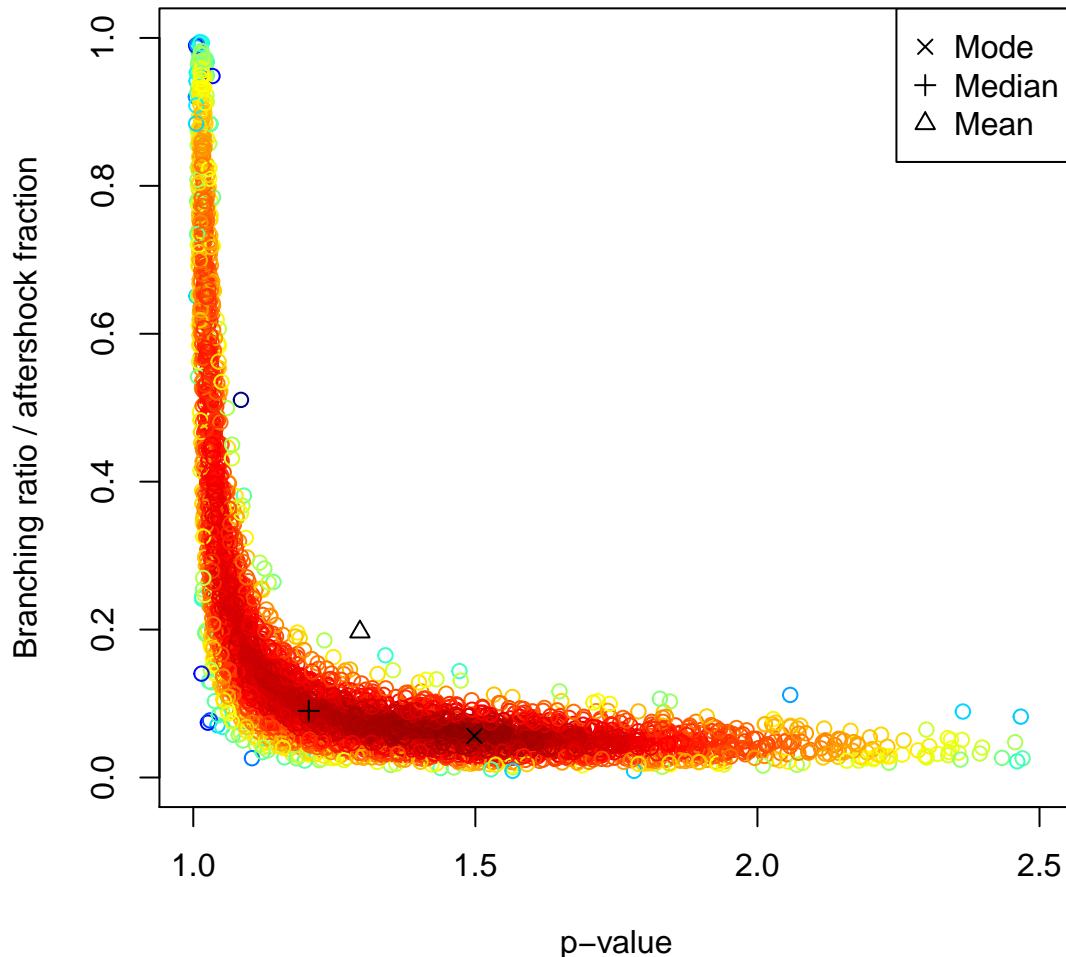
dt_sim = extract(f.stan[[i.f]], permute=TRUE)
palette(jet(100))
col = 1 + 99*(dt_sim$lp__ - min(dt_sim$lp__))/diff(range(dt_sim$lp__))
plot(dt_sim$p, dt_sim$n, col=col, xlim=c(1, 2.5), ylim=c(0,1), xlab="p-value",
```

```

ylab="Branching ratio / aftershock fraction",
main=paste("Family", fam))
points(p.stan.mode[i.f], n.stan.mode[i.f], pch=4)
points(p.stan.median[i.f], n.stan.median[i.f], pch=3)
points(p.stan.mean[i.f], n.stan.mean[i.f], pch=2)
legend("topright", c("Mode", "Median", "Mean"), pch=c(4,3,2))

```

## Family 70067948



In this case, using the median/mean parameters won't even generate a reasonable fit to the distribution. So I will stick with using the mode for the central value, but still quoting the CIs of the marginal distributions.

This (mode/max-likelihood being far away from the median/mean) does not always happen; here is an example where all three are close together.

```

fam = 70070093
f = match(fam, fids)
i.f = match(f, fuse)

dt_sim = extract(f.stan[[i.f]], permute=TRUE)
palette(jet(100))
col = 1 + 99*(dt_sim$lp__ - min(dt_sim$lp__))/diff(range(dt_sim$lp__))
plot(dt_sim$p, dt_sim$n, col=col, xlim=c(1, 2.5), ylim=c(0,1), xlab="p-value",

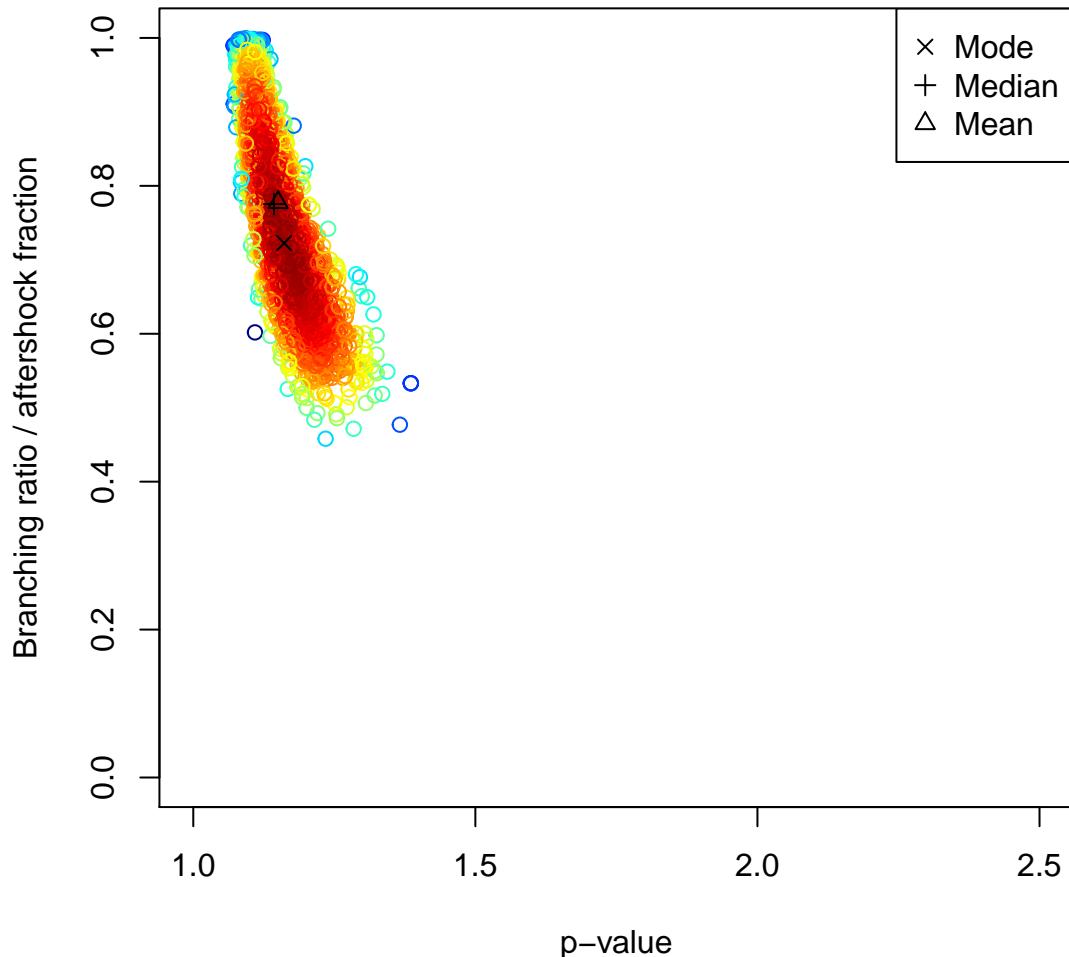
```

```

ylab="Branching ratio / aftershock fraction",
main=paste("Family", fam))
points(p.stan.mode[i.f], n.stan.mode[i.f], pch=4)
points(p.stan.median[i.f], n.stan.median[i.f], pch=3)
points(p.stan.mean[i.f], n.stan.mean[i.f], pch=2)
legend("topright", c("Mode", "Median", "Mean"), pch=c(4,3,2))

```

### Family 70070093



```

#fam = 70139288
fam = 70067948
f = match(fam, fids)
i.f = match(f, fuse)

dt_sim = extract(f.stan[[i.f]], permute=TRUE)
palette(jet(100))
col = 1 + 99*(dt_sim$lp__ - min(dt_sim$lp__))/diff(range(dt_sim$lp__))
plot(dt_sim$p, dt_sim$n, col=col, xlim=c(1, 2.5), ylim=c(0,1), xlab="p-value",
     ylab="Branching ratio / aftershock fraction",
     main=paste("Family", fam))
points(p.stan.mode[i.f], n.stan.mode[i.f], pch=4)
points(p.stan.median[i.f], n.stan.median[i.f], pch=3)

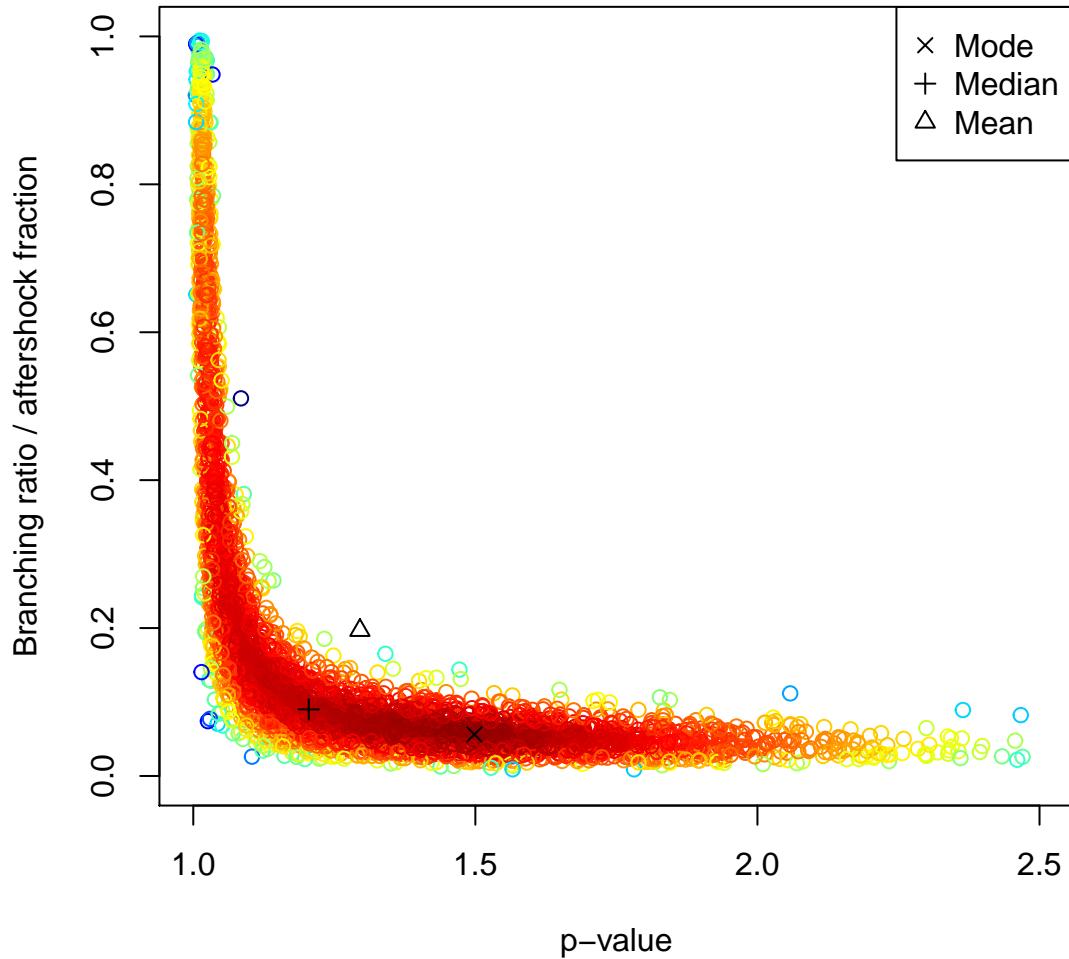
```

```

points(p.stan.mean[i.f], n.stan.mean[i.f], pch=2)
legend("topright", c("Mode", "Median", "Mean"), pch=c(4,3,2))

```

## Family 70067948



```

for (f in fuse) {
  i.f = match(f, fuse)

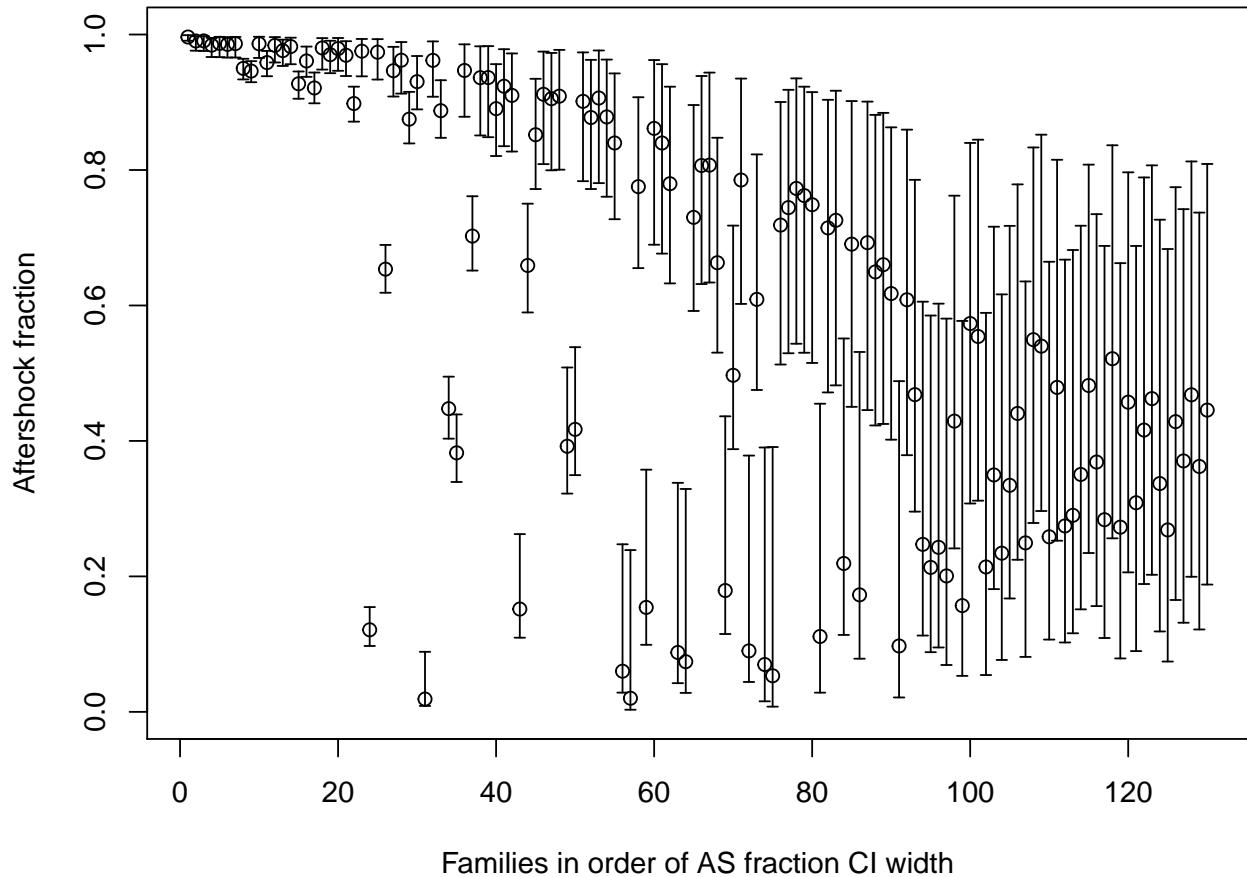
  dt_sim = extract(f.stan[[i.f]], permute=TRUE)
  palette(jet(100))
  col = 1 + 99*(dt_sim$lp__ - min(dt_sim$lp__))/diff(range(dt_sim$lp__))
  plot(dt_sim$p, dt_sim$n, col=col, xlim=c(1, 2.5), ylim=c(0,1), xlab="p-value",
       ylab="Branching ratio / aftershock fraction",
       main=paste("Family", fam))
  points(p.stan.mode[i.f], n.stan.mode[i.f], pch=4)
  points(p.stan.median[i.f], n.stan.median[i.f], pch=3)
  points(p.stan.mean[i.f], n.stan.mean[i.f], pch=2)
  legend("topright", c("Mode", "Median", "Mean"), pch=c(4,3,2))

  cat("\r\n\r\n\r\n")
}

```

## Widths of aftershock fraction CIs

```
dn = nCI.stan[,2] - nCI.stan[,1]
ord = order(dn)
plot(1:length(fuse), n.stan.median[ord], ylim=c(0,1),
     xlab="Families in order of AS fraction CI width",
     ylab="Aftershock fraction")
arrows(1:length(fuse), nCI.stan[ord,1], 1:length(fuse), nCI.stan[ord,2], angle=90, ylim=c(0,1), length=)
```



## Comparison with *Hainzl et al. [2006]*

Note that I haven't applied their bias correction here, this is just their basic procedure of taking the reciprocal of the variance of the (normalized) interevent times as the background fraction (and so the aftershock fraction is that subtracted from 1).

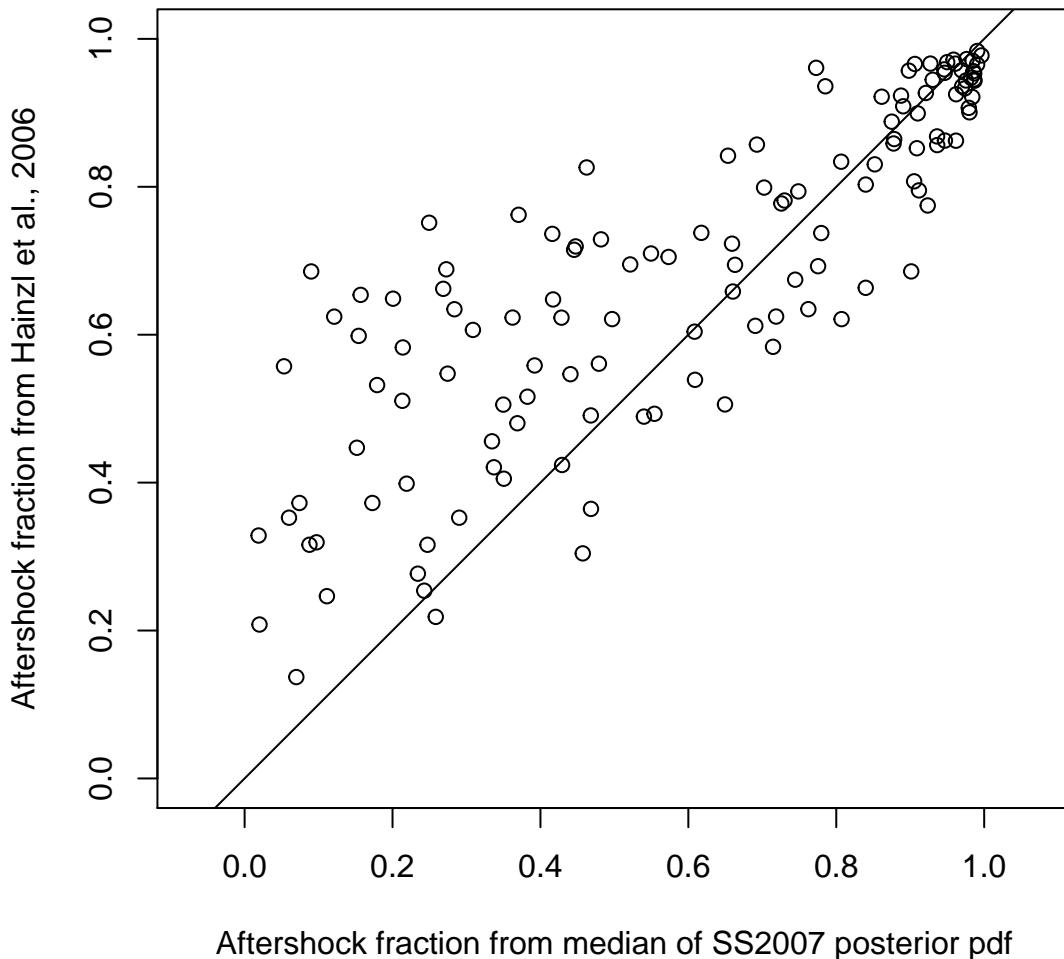
```
n.hainzl = rep(NA, length(fuse))
for (f in fuse) {
  i.f = match(f, fuse)
  ev.use = which(s$parent == fids[f])
  dt = diff(s$t0numeric[ev.use])
  dt = dt/mean(dt)
  n.hainzl[i.f] = 1 - 1/var(dt)
}

plot(n.stan.median, n.hainzl, asp=1, xlim=c(0,1), ylim=c(0,1),
```

```

xlab="Aftershock fraction from median of SS2007 posterior pdf",
ylab="Aftershock fraction from Hainzl et al., 2006")
abline(0,1)

```



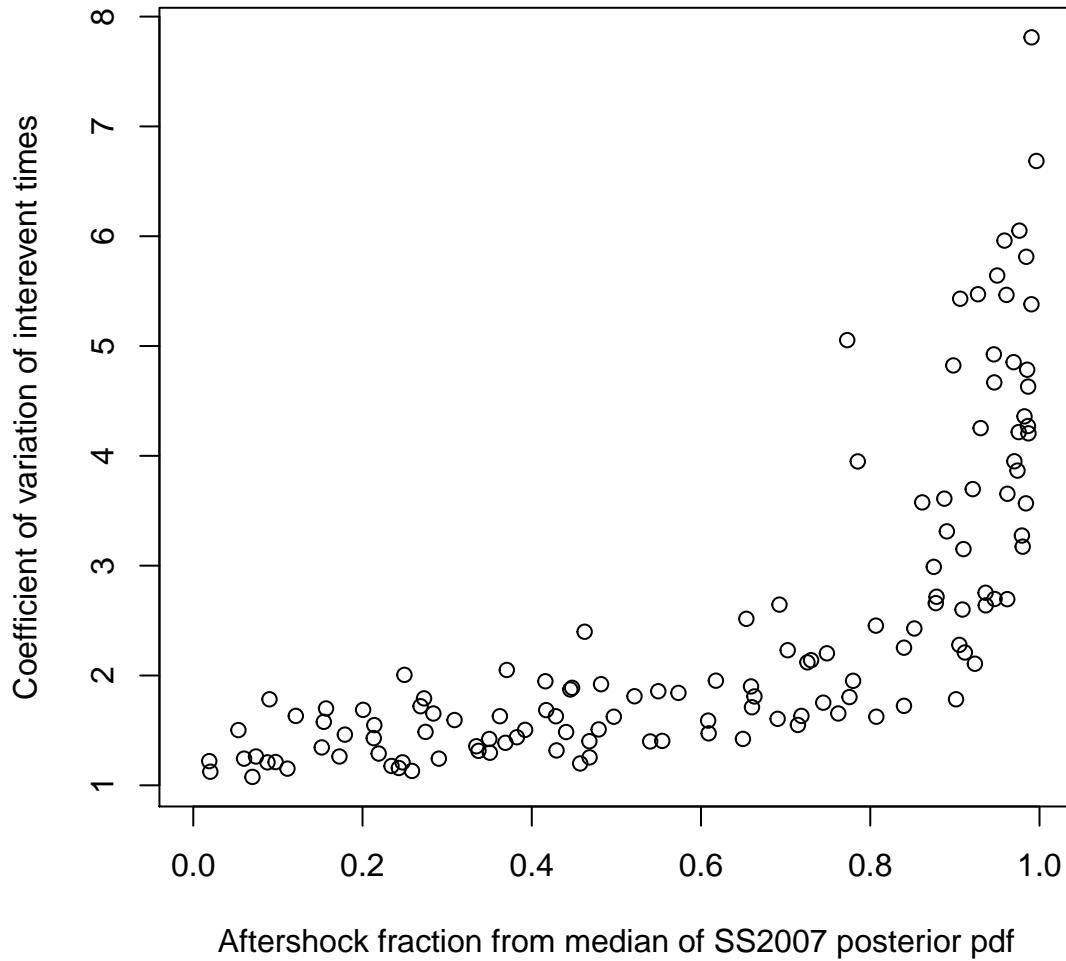
Comparison to standard deviation of dt ala *Cochran et al.* [2018]

```

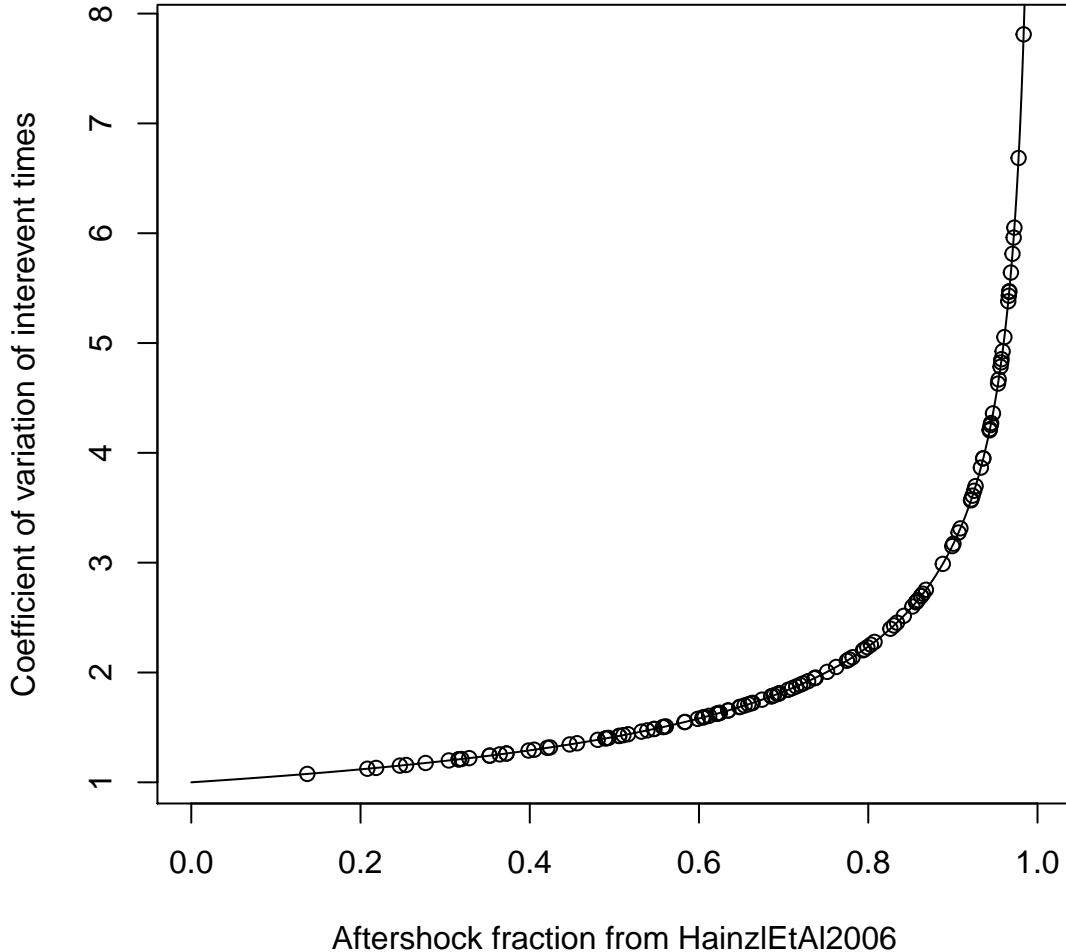
sd.cochran = rep(NA, length(fuse))
for (f in fuse) {
  i.f = match(f, fuse)
  ev.use = which(s$parent == fids[f])
  dt = diff(s$t0numeric[ev.use])
  dt = dt/mean(dt)
  sd.cochran[i.f] = sd(dt)
}

plot(n.stan.median, sd.cochran, xlim=c(0,1),
      xlab="Aftershock fraction from median of SS2007 posterior pdf",
      ylab="Coefficient of variation of interevent times")

```



```
plot(n.hainzl, sd.cochran, xlim=c(0,1),
      xlab="Aftershock fraction from HainzlEtAl2006",
      ylab="Coefficient of variation of interevent times")
n.plot = seq(0, 1, length=1000)
lines(n.plot, (1-n.plot)^(-1/2))
```



*Cochran et al.* [2018]'s use of the coefficient of variation of the interevent times as a measure of the relative fractions of background vs. aftershock events is essentially the same as that of *Hainzl et al.* [2006]. The curve on the above plot is given by  $C_v = (1 - f_a)^{-1/2}$ . From the paragraph at the beginning of pg. 315 of *Hainzl et al.* [2006], their estimate of the background fraction,  $f_b$ , is just  $1/\beta$  where  $\beta$  is a parameter of the gamma distribution (their eqn. (1)), and equal to the variance of the normalized interevent time distribution,  $\beta = \sigma_\tau^2$ . The coefficient of variation of *Cochran et al.* [2018] ( $C_v = \sigma_T/\bar{T}$ , where  $T$  are the unnormalized interevent times) is just equal to *Hainzl et al.* [2006]'s  $\sigma_\tau$ : since  $\tau = T/\bar{T}$ ,  $\sigma_\tau = \sigma_T/\bar{T}$ , and thus  $C_v = \sigma_\tau$ . So  $f_b = 1/C_v^2$ , and since  $f_a = 1 - f_b$ ,  $f_a = 1 - 1/C_v^2$  or  $C_v = (1 - f_a)^{-1/2}$ .

### Aftershock fraction estimation a la Zaliapin and Ben-Zion [2016]

Still in progress: so far results don't look right - nearly always less than that from SS2007 - need to look more carefully at some individual cases. I think problem may be that many events have exactly the same location?

```
n.zaliapin = rep(NA, length(fuse))
for (f in fuse) {
  i.f = match(f, fuse)
  ev.use = which(s$parent == fids[f])
  z = zaliapin(list(t0yr = s$t0numeric[ev.use]/(365.25*86400), M=s$mag[ev.use],
                    x=s$utmx[ev.use], y=s$utmy[ev.use], t0=s$t0numeric[ev.use]),
               sderrh=100)
  n.zaliapin[i.f] = sum(log10(z$eta) < z$log10eta.c, na.rm=TRUE)/sum(is.finite(z$eta))
}
```

```

## [1] 0.8669339 0.1330661 -3.1880364 -4.3468965 0.5838018 1.6700145
## [1] 0.90738383 0.09261617 -3.92224832 -5.92038131 0.64912581 1.43207187
## [1] 0.8470818 0.1529182 -3.8807582 -5.3758216 0.5582242 1.1446130
## [1] 0.7854556 0.2145444 -3.9467592 -5.4289747 0.5191455 1.1774859
## [1] 0.08800939 0.91199061 -5.28861078 -4.04067613 1.09406440 0.62269270
## [1] 0.1301972 0.8698028 -3.9171842 -3.1317312 1.6107708 0.5203818
## [1] 0.5155931 0.4844069 -3.3674581 -6.1913543 0.4907685 1.1750499
## [1] 1.000000e+00 8.247095e-11 -5.600448e+00 -9.128839e+00 1.875316e+00
## [6] 0.000000e+00
## [1] 0.7570960 0.2429040 -3.9893610 -5.4729297 0.5874894 1.3341868
## [1] 0.5812441 0.4187559 -3.1796502 -4.2632315 0.4711669 0.7645282
## [1] 0.8773689 0.1226311 -3.4521243 -5.0180808 0.6131161 1.1341270
## [1] 0.7911020 0.2088980 -3.7914767 -5.9190238 0.5168468 1.6281961
## [1] 0.97667086 0.02332914 -3.72696412 -5.31491845 0.58164500 2.73719081
## [1] 0.6762177 0.3237823 -3.5119714 -4.9618546 0.4826929 1.0683933
## [1] 0.77776966 0.2223034 -3.8948604 -6.1833525 0.6164390 0.7834945
## [1] 0.8335614 0.1664386 -6.4524786 -7.3326189 1.0524636 0.2766537
## [1] 0.6159662 0.3840338 -3.2539772 -4.9219478 0.4501739 1.2232803
## [1] 0.7289613 0.2710387 -3.6818069 -4.9596289 0.5149176 1.0321262
## [1] 0.7300121 0.2699879 -3.3297038 -4.2890240 0.5081062 0.9504798
## [1] 0.3745489 0.6254511 -3.8246341 -4.6187746 0.2864871 1.1769464
## [1] 0.532044 0.467956 -3.672249 -6.594998 0.490704 1.054996
## [1] 0.8486442 0.1513558 -3.8773755 -6.4622714 0.6294963 0.9001226
## [1] 0.5590940 0.4409060 -3.7590609 -4.8735625 0.4360487 1.0834405
## [1] 0.7539702 0.2460298 -3.7343243 -5.1991799 0.5457887 1.1983753
## [1] 0.8304969 0.1695031 -3.4515612 -4.9942686 0.4946922 1.3198164
## [1] 0.7292154 0.2707846 -3.1328080 -4.0203646 0.4233849 0.8291517
## [1] 0.97160096 0.02839904 -3.47874081 -5.34972029 0.59644326 0.04395247
## [1] 0.8952594 0.1047406 -4.8767840 -6.0678290 1.2650039 0.2012075
## [1] 0.7062441 0.2937559 -3.3318746 -4.8593257 0.5215481 1.3964765
## [1] 0.8589125 0.1410875 -3.7539745 -6.0283272 0.6048435 1.0613089
## [1] 0.1837753 0.8162247 -3.5281410 -6.4577300 0.6121423 1.0348472
## [1] 0.1143787 0.8856213 -3.5001689 -6.7355149 0.6501575 0.7812125
## [1] 0.4758589 0.5241411 -3.2553867 -4.7834670 0.3535793 1.3342235
## [1] 0.9618785 0.0381215 -3.4169824 -5.5760892 0.6158647 0.8224735
## [1] 0.8236527 0.1763473 -3.4665077 -5.2807841 0.5407003 1.0898973
## [1] 0.92040094 0.07959906 -3.89505657 -7.22820876 0.70479416 0.86455312
## [1] 0.8409488 0.1590512 -3.9976041 -5.2825268 0.6057111 1.0761867
## [1] 0.6658167 0.3341833 -4.0170774 -5.8539270 0.4755648 1.2542683
## [1] 0.6281086 0.3718914 -3.5645440 -4.3615365 0.3776269 0.9173761
## [1] 0.7363962 0.2636038 -3.6693787 -4.8402379 0.4959088 1.2167330
## [1] 0.7441931 0.2558069 -3.2065723 -4.1763869 0.5169905 0.8961020
## [1] 0.5584895 0.4415105 -3.5558443 -5.6907716 0.4708024 1.2459487
## [1] 0.90073467 0.09926533 -5.00617243 -7.05662650 1.34531678 0.38487321
## [1] 0.4577316 0.5422684 -3.8974073 -5.6416577 0.5394354 1.2643920
## [1] 0.4449790 0.5550210 -3.4468942 -4.4794601 0.4083087 1.3158606
## [1] 0.4472632 0.5527368 -3.3691828 -6.1431643 0.6680665 0.9823959
## [1] 0.8548916 0.1451084 -4.1604557 -6.4762848 1.0343566 0.3068899
## [1] 0.7770586 0.2229414 -3.5539075 -5.0141367 0.5254482 0.8279790
## [1] 0.8192701 0.1807299 -3.5503533 -5.8423975 0.5654645 1.1691548
## [1] 0.7783143 0.2216857 -4.2002712 -5.5455911 0.4686974 1.1579232
## [1] 0.8493928 0.1506072 -3.5080283 -4.9377894 0.5169356 0.9673475
## [1] 0.8566080 0.1433920 -3.7051815 -4.6298573 0.5012318 1.4256912
## [1] 0.8574051 0.1425949 -3.7712527 -5.8949213 0.5932176 1.1229378

```

```

## [1] 0.7709859 0.2290141 -3.4858730 -4.7529671 0.5470107 1.0450658
## [1] 0.5035984 0.4964016 -2.9828977 -3.7551286 0.3345131 0.4640188
## [1] 0.4530050 0.5469950 -3.4097610 -5.5867970 0.5560161 1.3440058
## [1] 0.8422907 0.1577093 -3.4201795 -5.2471019 0.6097236 1.7292839
## [1] 0.6890375 0.3109625 -3.7728348 -4.5719945 0.4798221 0.8716528
## [1] 0.8831593 0.1168407 -4.9058909 -6.9911856 1.2305199 0.8290902
## [1] 0.96186997 0.03813003 -3.81604360 -6.86355295 0.61962406 0.38053249
## [1] 0.95353712 0.04646288 -3.95821779 -5.75614864 1.03671886 0.06132440
## [1] 0.91141262 0.08858738 -4.92400256 -6.14906240 1.29273201 1.01337842
## [1] 0.8214997 0.1785003 -3.8755497 -4.8699268 0.5717315 1.2456111
## [1] 0.7515274 0.2484726 -3.4728571 -5.2131510 0.4933993 1.6347454
## [1] 0.992586346 0.007413654 -6.346797975 -8.347014991 1.282812481
## [6] 0.165176892
## [1] 0.5721366 0.4278634 -3.4190700 -5.8766706 0.5035860 1.3356827
## [1] 0.4901732 0.5098268 -3.1013185 -5.2638157 0.5104665 1.0974319
## [1] 0.7292603 0.2707397 -3.5437952 -4.5689140 0.5005775 1.0593389
## [1] 0.9115018 0.0884982 -4.3005019 -4.7399132 1.3775392 0.0900997
## [1] 0.1491564 0.8508436 -2.9444717 -5.1696507 0.3021409 1.3532918
## [1] 0.4111939 0.5888061 -3.6895413 -4.9496843 0.4510306 1.1327186
## [1] 0.5115371 0.4884629 -3.5711608 -4.4179869 0.5526076 1.0651707
## [1] 0.6234423 0.3765577 -3.6230196 -6.7734700 0.6681391 0.8703395
## [1] 0.3233445 0.6766555 -3.7527611 -5.8782949 0.4723273 1.2095541
## [1] 0.6559624 0.3440376 -3.3091376 -6.2913767 0.6151772 0.9715708
## [1] 0.7755121 0.2244879 -3.5296709 -5.4826587 0.5797890 1.4378640
## [1] 0.2276729 0.7723271 -2.6015854 -5.3629454 0.4177085 1.4258060
## [1] 0.7368220 0.2631780 -3.3503989 -4.6050194 0.4146172 1.5307197
## [1] 0.6172384 0.3827616 -3.5293589 -5.1571900 0.5114209 1.2899537
## [1] 0.2511801 0.7488199 -3.2160523 -3.8544214 0.1555464 0.8600999
## [1] 0.4778982 0.5221018 -5.6302704 -7.1068107 1.5458935 0.5877080
## [1] 0.96833081 0.03166919 -6.53872085 -8.34532752 1.32850719 0.07630969
## [1] 0.91792855 0.08207145 -3.79109669 -6.59394765 0.60140520 0.95157469
## [1] 0.4616931 0.5383069 -3.8732760 -6.5060142 0.7715390 1.0894063
## [1] 0.5629601 0.4370399 -3.6006120 -4.9946575 0.4716617 0.9541453
## [1] 0.8622699 0.1377301 -3.6319330 -5.4257347 0.6083754 1.3846622
## [1] 0.4123862 0.5876138 -3.4970085 -5.7912781 0.5480840 1.2811189
## [1] 0.3844025 0.6155975 -3.7273240 -5.1308354 0.5035295 0.9862059
## [1] 0.3777856 0.6222144 -3.6302114 -5.3523761 0.4386957 1.4682487
## [1] 0.8732830 0.1267170 -3.8630479 -6.9196236 0.7491357 0.6790985
## [1] 0.8008887 0.1991113 -5.0998322 -6.1712368 1.2613857 0.6478935
## [1] 0.8147039 0.1852961 -3.8060599 -5.4558723 0.4905956 1.1942229
## [1] 0.7080166 0.2919834 -3.7394745 -4.7340156 0.4864577 0.8528665
## [1] 0.5672970 0.4327030 -3.6914179 -4.9668105 0.5359772 1.0101194
## [1] 0.8301526 0.1698474 -5.1455664 -6.8175454 1.2117696 1.0346007
## [1] 0.7244460 0.2755540 -5.7153847 -5.8464358 1.2842678 0.2389391
## [1] 0.8730703 0.1269297 -3.2752193 -5.1368644 0.4493648 1.1609440
## [1] 0.6060701 0.3939299 -3.5304723 -5.4588231 0.4526075 1.2623908
## [1] 0.7886315 0.2113685 -3.5585458 -5.9578418 0.6778481 0.6635083
## [1] 0.5794132 0.4205868 -4.2438051 -5.7692681 0.7725604 1.2968943
## [1] 0.8283209 0.1716791 -4.4968842 -6.9507515 1.1524893 0.5157259
## [1] 0.4639311 0.5360689 -5.2293967 -5.7400531 0.5323676 1.3362272
## [1] 0.2664684 0.7335316 -3.9277530 -6.4554493 0.6900809 0.9324080
## [1] 0.6800498 0.3199502 -3.8196225 -5.3987293 0.5245639 1.2178198
## [1] 0.4446453 0.5553547 -3.5135816 -4.9270874 0.3656760 1.1116447
## [1] 0.97351692 0.02648308 -5.51715232 -7.18736203 1.14083049 0.08857108

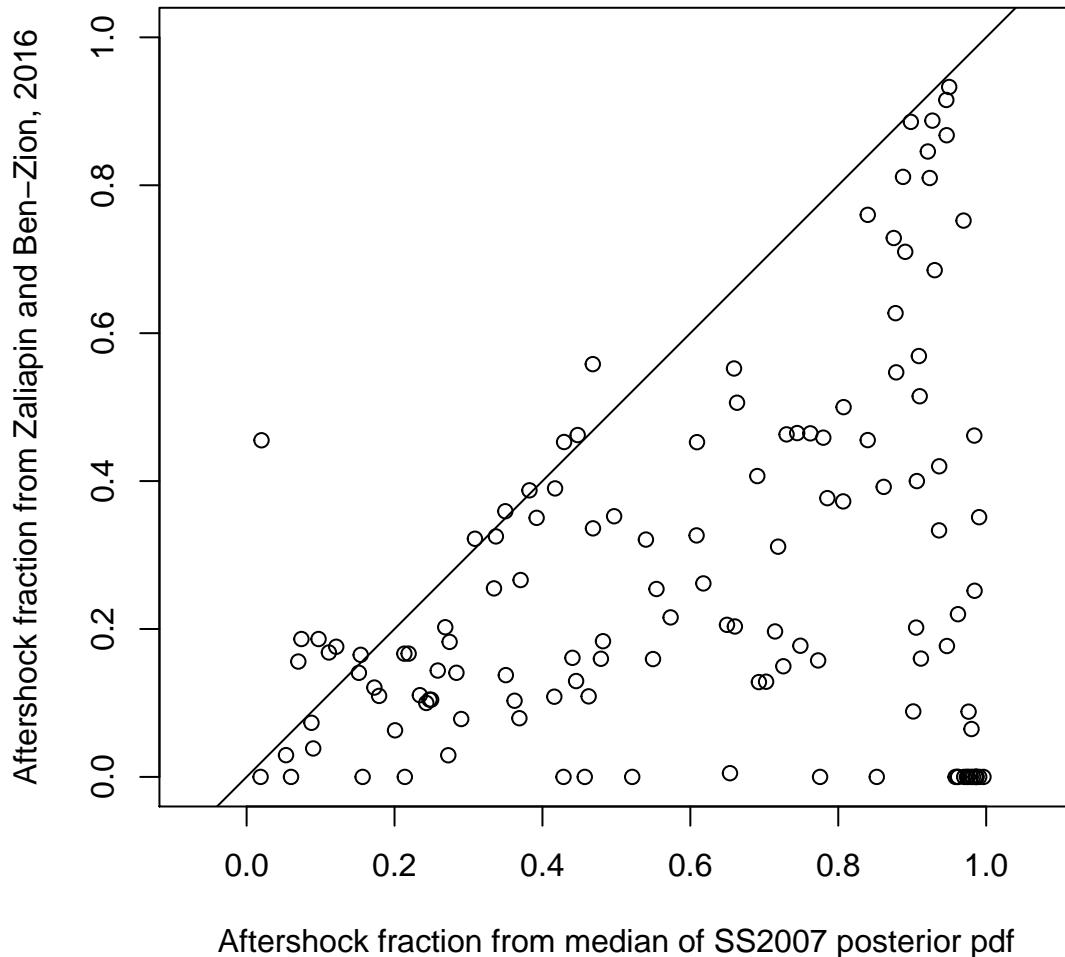
```

```

## [1] 0.8042979 0.1957021 -3.4846245 -4.8793107 0.6139817 1.0913067
## [1] 0.3151007 0.6848993 -5.2159706 -5.2907376 0.3739572 1.3396138
## [1] 0.5760082 0.4239918 -5.5801865 -6.7601902 1.1019933 1.1495721
## [1] 0.5190139 0.4809861 -4.4621053 -6.0026322 0.9555924 1.0345516
## [1] 0.8083298 0.1916702 -4.7868527 -7.2369153 0.9685981 0.5576418
## [1] 0.6304448 0.3695552 -6.0318014 -6.2912606 0.6467685 1.6089324
## [1] 0.7456042 0.2543958 -5.3541616 -6.4802106 1.1841868 1.1056256
## [1] 0.1221290 0.8778710 -3.4979652 -6.7836722 0.9439829 0.7464738
## [1] 0.8271523 0.1728477 -3.7423808 -6.5247251 0.6007023 1.1501753
## [1] 0.3454889 0.6545111 -4.9689665 -6.4688806 1.2633449 0.8361493
## [1] 0.1294972 0.8705028 -3.8175607 -6.5111799 0.6641770 0.9006096
## [1] 0.1864524 0.8135476 -4.2753149 -6.8988905 1.2547515 0.9123740
## [1] 0.6445054 0.3554946 -3.4762150 -6.1980454 0.5564428 1.0184043
## [1] 0.08510732 0.91489268 -3.67829916 -7.06466302 0.99797990 0.79561673
## [1] 0.06946016 0.93053984 -3.89285940 -7.41088251 0.98579495 0.73175528
## [1] 0.5818033 0.4181967 -5.9583393 -6.2910945 1.7906300 0.7850538
## [1] 0.2892697 0.7107303 -3.2752241 -6.3260259 0.6768774 0.8460248
## [1] 0.1718024 0.8281976 -5.5547436 -7.0122196 2.4146118 0.7633404
## [1] 0.4805150 0.5194850 -3.7976999 -5.7464364 0.5852767 1.3128840
## [1] 0.5523409 0.4476591 -3.9575219 -5.9581623 0.5728639 1.3189322
## [1] 0.91453531 0.08546469 -4.78310851 -7.28236928 1.35745725 1.19101085
## [1] 0.8032118 0.1967882 -5.2455250 -6.9111087 1.0322594 1.0983560
## [1] 0.4815551 0.5184449 -5.5781386 -5.9785101 1.7121300 0.7586046

plot(n.stan.median, n.zaliapin, asp=1, xlim=c(0,1), ylim=c(0,1),
      xlab="Aftershock fraction from median of SS2007 posterior pdf",
      ylab="Aftershock fraction from Zaliapin and Ben-Zion, 2016")
abline(0,1)

```



## Todo

- wrap making plots into a function
- check converge, etc. of Stan fits
- fit full ETAS model?
- add comparisons to Hainzl et al. [2006]
- plots/discussions of widths of estimates
  - `plot(n.stan.median, ylim=c(0,1))`
  - `arrows(1:130, nCI.stan[,1], 1:130, nCI.stan[,2], angle=90, ylim=c(0,1), length=0.05, code=3)`

## References

Cochran, E. S., Z. E. Ross, R. M. Harrington, S. L. Dougherty, and J. L. Rubinstein (2018), Induced earthquake families reveal distinctive evolutionary patterns near disposal wells, *Journal of Geophysical Research: Solid Earth*, 123(9), 8045–8055.

Hainzl, S., F. Scherbaum, and C. Beauval (2006), Estimating background activity based on interevent-time distribution, *Bulletin of the Seismological Society of America*, 96(1), 313–320.

Saichev, A., and D. Sornette (2007), Theory of earthquake recurrence times, *Journal of Geophysical Research: Solid Earth*, 112(B4).

Zaliapin, I., and Y. Ben-Zion (2016), Discriminating characteristics of tectonic and human-induced seismicity, *Bulletin of the Seismological Society of America*, 106(3), 846–859.