Gibbs sampler for progression model

Contents

1 Prelude: Simulate examples from the hierarchical model

We begin with some code to make example by simulating from the hierarchical model.

Recall that the hierarchical model is:

1. $\boldsymbol{\pi} \sim \text{Uniform}(\text{standard } J\text{-simplex in } \mathbb{R}^J)$.
2. $z_g | \boldsymbol{\pi} \sim \text{Categorical}(\boldsymbol{\pi})$ for all $g \in \{1, \ldots, G\}$, i.e., $z_g = j$ with probability π_j for $j \in \{1, \ldots, J\}$.
3. $\lambda_{j,\mathscr{T}} \sim \text{Gamma}(a_0, \nu)$ for all $j \in \{1, \ldots, J\}, \mathscr{T} \in \mathbb{T}_j$.
4. $x_{i,g,t} | \lambda_{z_g,\mathscr{T}} \sim \text{Gamma}(a, \lambda_{z_g,\mathscr{T}})$ for all $i \in \{1, \ldots, n\}, g \in \{1, \ldots, G\}, \mathscr{T} \in \mathbb{T}_{z_g}, t \in \mathscr{T}$.
5. $\mathbf{p}_\sigma \sim \text{Uniform}(\text{standard } T\text{-simplex in } \mathbb{R}^T)$ for $\sigma \in \{1, \ldots, \Sigma\}$.
6. $s_{i,g} = \sum_{t=1}^{T} p_{\sigma_i,t} x_{i,g,t}$ for all $i \in \{1, \ldots, n\}, g \in \{1, \ldots, G\}$.

So:

```
> # from http://tdunning.blogspot.com/2010/04/sampling-dirichlet-distributions_06.html
> rdirichlet = function(n, alpha) {
+   k = length(alpha)
+   r = matrix(0, nrow=n, ncol=k)
+   for (i in 1:k) {
+    r[,i] = rgamma(n, alpha[i], 1)
+   }
+   r = matrix(mapply(function(r, s) {return (r/s)}, r, rowSums(r)), ncol=k)
+   return (r)
+ }
> sample.overall <- function(n, Sigma, G, T, J, a, a0, nu, sigma, bbT) {
+   pi <- rdirichlet(1, array(1, J))
+   z <- sample(1:J, size=G, prob=pi, replace=TRUE)
+   lambda <- llply(1:J, function(j) {
+     llply(bbT[[j]], function(calT) {
+       rgamma(1, shape=a0, rate=nu)
+     })
+   })
+   x <- laply(1:n, function(i) {
+     laply(1:G, function(g) {
+       x_ig <- array(NaN,T)
+       for (calT.index in 1:length(bbT[[z[g]]])) {
+         calT <- bbT[[z[g]]][[calT.index]]
+         for (t in calT)
+           x_ig[t] <- rgamma(1, shape=a, rate=lambda[[z[g]]][[calT.index]])
+       }
+       x_ig
+     })
+   })
+   p <- laply(1:Sigma, function(sig) rdirichlet(1, array(1, T)))
+   s <- laply(1:n, function(i) {
+     laply(1:G, function(g) {
+       sum(laply(1:T, function(t) p[sigma[i],t]*x[i,g,t]))
+     })
+   })
+   list(pi=pi, z=z, lambda=lambda, x=x, p=p, s=s)
+ }
> make.example <- function(n=8, G=5, a=100.0, a0=50.0, nu=30.0) {
+   Sigma <- 4
+   T <- 3
+   J <- 5
+   bbT = list(
+     list(c(1,2,3)),
+     list(c(1,2),3),
+     list(1,c(2,3)),
+     list(c(1,3),2),
+     list(1,2,3)
+   )
+   reject <- TRUE
+   while (reject) {
+     sigma <- sample.int(Sigma, size=n, replace=TRUE)
+     reject <- FALSE
+     for (sig in 1:Sigma)
+       if (length(which(sigma == sig)) == 0)
+         reject <- TRUE
+   }
+   l1 <- list(n=n, Sigma=Sigma, G=G, T=T, J=J, alpha=list(a=a, a0=a0, nu=nu), sigma=sigma, bbT=bbT)
+   l2 <- sample.overall(n, Sigma, G, T, J, a, a0, nu, sigma, bbT)
+   c(l1, l2)
+ }
> #print(make.example())
```

2 Overview

We repeat forever:

1. Sample $\mathbf{x}, \mathbf{p} | \mathbf{z}, \boldsymbol{\pi}, \boldsymbol{\alpha}, \boldsymbol{\lambda}$.
2. Sample $\boldsymbol{\lambda} | \mathbf{x}, \mathbf{p}, \mathbf{z}, \boldsymbol{\pi}, \boldsymbol{\alpha}$.
3. Sample $\mathbf{z} | \mathbf{x}, \mathbf{p}, \boldsymbol{\pi}, \boldsymbol{\alpha}, \boldsymbol{\lambda}$.
4. Sample $\boldsymbol{\pi} | \mathbf{x}, \mathbf{p}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\lambda}$.
5. Sample $\boldsymbol{\alpha} | \mathbf{x}, \mathbf{p}, \mathbf{z}, \boldsymbol{\pi}, \boldsymbol{\lambda}$.

In code:

```
> gibbs <- function(num.iters, n, Sigma, G, T, J, initial.x, initial.p, initial.z, initial.pi, initial.alpha, ini
+   x <- list(initial.x)
+   p <- list(initial.p)
+   z <- list(initial.z)
+   pi <- list(initial.pi)
+   alpha <- list(initial.alpha)
+   lambda <- list(initial.lambda)
+   for (k in 1:num.iters) {
+     cat(sprintf("iteration %d:\n", k))
+     #XXX cat("sample.x.and.p\n")
+     o <- sample.x.and.p(n, Sigma, G, T, J, x[[k]], p[[k]], z[[k]], alpha[[k]], lambda[[k]], s, sigma, bbT)
+     x[[k+1]] <- o$x
+     p[[k+1]] <- o$p
+     #x[[k+1]] <- x[[k]]
+     #p[[k+1]] <- p[[k]]
+     #XXX cat("sample.lambda\n")
+     lambda[[k+1]] <- sample.lambda(n, J, x[[k+1]], z[[k]], alpha[[k]], lambda[[k]], bbT)
+     #lambda[[k+1]] <- lambda[[k]]
+     #XXX cat("sample.pi\n")
+     pi[[k+1]] <- sample.pi(J, z[[k]])
+     #pi[[k+1]] <- pi[[k]]
+     #XXX cat("sample.z\n")
+     z[[k+1]] <- sample.z(G, J, pi[[k+1]])
+     #z[[k+1]] <- z[[k]]
+     #XXX cat("sample.alpha\n")
+     alpha[[k+1]] <- sample.alpha(n, G, T, J, x[[k+1]], z[[k+1]], lambda[[k+1]], alpha[[k]], bbT)
+     # XXX begin tmp
+     #alpha[[k+1]] <- alpha[[k]]
+     # XXX end tmp
+     #XXX cat("save\n")
+     #save(x, p, z, pi, alpha, lambda, file="gibbs_intermed.Rda")
+   }
+   #XXX cat("\n")
+   list(x=x, p=p, z=z, pi=pi, alpha=alpha, lambda=lambda)
+ }
```

3 Sample $\mathbf{x},\mathbf{p}|\mathbf{z},\boldsymbol{\pi},\boldsymbol{\alpha}$

Note that we cannot just split up **x** and **p** and sample each from its full conditional distribution, because we cannot find a measure with respect to which $f(\mathbf{x},\mathbf{p}|\mathbf{z},\boldsymbol{\pi},\boldsymbol{\alpha})$ is nonzero almost everywhere for almost all choices of $\mathbf{x},\boldsymbol{\pi},\boldsymbol{\alpha}$.

Instead, we do the following. First, we consider each $\mathbf{p}_\sigma$ an element of $\mathbb{R}_+^{T-1}$ (rather than of $\mathbb{R}_+^T$), and we set $p_{\sigma,T} = p_{\sigma,T}(\mathbf{p}_\sigma) = 1-\sum_{t=1}^{T=1} p_{\sigma,T}$. Similarly, we also consider each $\mathbf{x}_{i,g}$ an element of $\mathbb{R}_+^{T-1}$ (rather than of $\mathbb{R}_+^T$), and we set $x_{i,g,T} = x_{i,g,T}(\mathbf{x}_{i,g},\mathbf{p}_{\sigma_i},s_{i,g}) = (s_{i,g}-\sum_{t=1}^{T-1} x_{i,g,t}p_{\sigma_i,t})/p_{\sigma_i,T}$.

From the hierarchical model, we see that the density of $\mathbf{x},\mathbf{p}|\mathbf{z},\boldsymbol{\pi},\boldsymbol{\alpha},\boldsymbol{\lambda}$ with respect to Lebesgue measure on $(\times_{i=1}^n \times_{g=1}^G \mathbb{R}_+^{T-1})\times(\times_{\sigma=1}^4 \mathbb{R}_+^{T-1})$ is

$$f(\mathbf{x},\mathbf{p}|\mathbf{z},\boldsymbol{\pi},\boldsymbol{\alpha},\boldsymbol{\lambda}) \propto 1_{\cap_{\sigma=1}^{\Sigma}\{\sum_{t=1}^{T-1} p_{\sigma,t}\le 1\}} \prod_{i=1}^n \prod_{g=1}^G 1_{\{(\mathbf{x}_{i,g},\mathbf{p}):\sum_{t=1}^{T-1} x_{i,g,t}p_{\sigma_i,t}\le s_{i,g}\}} \prod_{\mathscr{T}\in\mathbb{T}_{z_g}} f_{\Gamma(a_0,\nu)}(\lambda_{z_g,\mathscr{T}}) \prod_{t\in\mathscr{T}} f_{\Gamma(a,\lambda_{z_g,\mathscr{T}})}(x_{i,g,t})$$

where $f_{\Gamma(a,\lambda)}(x) = \lambda^a x^{a-1}e^{-\lambda x}/\Gamma(a)$ is the density of a Gamma distribution with shape a and scale λ.

Next, we substitute as follows. Define $\phi(\mathbf{y},\mathbf{p}) = ((\times_{i=1}^n \times_{g=1}^G y_{i,g,t}s_{i,g}/p_{\sigma_i,t}),\mathbf{p})$. Note that ϕ is bijective with inverse $(\mathbf{x},\mathbf{p})\mapsto((\times_{i=1}^n \times_{g=1}^G x_{i,g,t}p_{\sigma_i,t}/s_{i,g}),\mathbf{p})$. If $(\mathbf{x},\mathbf{p}')=\phi(\mathbf{y},\mathbf{p})$, then the partial derivatives

$$\begin{aligned}
\partial x_{i,g,t}/\partial y_{i,g,t} &= s_{i,g}/p_{\sigma_i,t}\\
\partial x_{i,g,t}/\partial p_{\sigma_i,t} &= y_{i,g,t}s_{i,g}(-1)p_{\sigma_i,t}^{-2}\\
\partial p'_{\sigma,t}/\partial p_{\sigma,t} &= 1
\end{aligned}$$

and all other partials are 0. From this, we see that the Jacobian matrix is upper triangular and has diagonal entries $s_{i,g}/p_{\sigma_i,t}$ for all i,g,t and 1 for all σ,t, so the Jacobian determinant J_ϕ of ϕ is

$$J_\phi(\mathbf{y},\mathbf{p}) = \prod_{i=1}^n \prod_{g=1}^G \prod_{t=1}^{T-1} s_{i,g}/p_{\sigma_i,t}$$

Thus, the transformed density is

$$f(\mathbf{y},\mathbf{p}|\mathbf{z},\boldsymbol{\pi},\boldsymbol{\alpha},\boldsymbol{\lambda}) \propto 1_{\cap_{\sigma=1}^{\Sigma}\{\sum_{t=1}^{T-1} p_{\sigma,t}\le 1\}} \prod_{i=1}^n \prod_{g=1}^G 1_{\{\mathbf{y}_{i,g}:\sum_{t=1}^{T-1} y_{i,g,t}\le 1\}} \Big[\prod_{\mathscr{T}\in\mathbb{T}_{z_g}} f_{\Gamma(a_0,\nu)}(\lambda_{z_g,\mathscr{T}}) \prod_{t\in\mathscr{T}} f_{\Gamma(a,\lambda_{z_g,\mathscr{T}})}(y_{i,g,t}s_{i,g}/p_{\sigma_i,t})\Big]\Big[\prod_{t=1}^{T-1} p_{\sigma_i,t}^{-1}\Big]$$

Now we can get nice full conditionals:

$$f(\mathbf{y}|\mathbf{p},\mathbf{z},\boldsymbol{\pi},\boldsymbol{\alpha},\boldsymbol{\lambda}) \propto \prod_{i=1}^n \prod_{g=1}^G 1_{\{\mathbf{y}_{i,g}:\sum_{t=1}^{T-1} y_{i,g,t}\le 1\}} \Big[\prod_{\mathscr{T}\in\mathbb{T}_{z_g}} \prod_{t\in\mathscr{T}} f_{\Gamma(a,\lambda_{z_g,\mathscr{T}})}(y_{i,g,t}s_{i,g}/p_{\sigma_i,t})\Big] \tag{ygp}$$

$$f(\mathbf{p}|\mathbf{y},\mathbf{z},\boldsymbol{\pi},\boldsymbol{\alpha},\boldsymbol{\lambda}) \propto 1_{\cap_{\sigma=1}^{\Sigma}\{\sum_{t=1}^{T-1} p_{\sigma,t}\le 1\}} \prod_{i=1}^n \prod_{g=1}^G \Big[\prod_{\mathscr{T}\in\mathbb{T}_{z_g}} \prod_{t\in\mathscr{T}} f_{\Gamma(a,\lambda_{z_g,\mathscr{T}})}(y_{i,g,t}s_{i,g}/p_{\sigma_i,t})\Big]\Big[\prod_{t=1}^{T-1} p_{\sigma_i,t}^{-1}\Big] \tag{pgy}$$

which are both nonzero almost everywhere on the relevant simplices for all values of the conditioning variables.

In our code we need here to just take care of the transformations back and forth between **x** and **y**:

```
> sample.x.and.p <- function(n, Sigma, G, T, J, x.prev, p.prev, z, alpha, lambda, s, sigma, bbT) {
+   y.prev <- array(0, c(n, G, T))
+   for (i in 1:n)
+     for (t in 1:T)
+       y.prev[i,,t] <- x.prev[i,,t] * p.prev[sigma[i],t] / s[i,]
+   y <- sample.y(n, G, T, J, Sigma, y.prev, p.prev, z, alpha, lambda, s, sigma)
+   p <- sample.p(n, Sigma, G, T, J, y, p.prev, z, alpha, lambda, s, sigma, bbT)
+   x <- array(0, c(n, G, T))
+   for (i in 1:n)
+     for (t in 1:T)
+       x[i,,t] <- y[i,,t] * s[i,] / p[sigma[i],t]
+   list(x=x, p=p)
+ }
```

The next task is to sample from each of these conditional distributions. We start with y.

3.1 Sample $\mathbf{y}|\mathbf{p},\mathbf{z},\boldsymbol{\pi},\boldsymbol{\alpha},\boldsymbol{\lambda}$

Note that

$$f(\mathbf{y}|\mathbf{p},\mathbf{z},\boldsymbol{\pi},\boldsymbol{\alpha},\boldsymbol{\lambda}) \propto \prod_{i=1}^{n}\prod_{g=1}^{G} 1_{\{\mathbf{y}_{i,g}:\sum_{t=1}^{T-1} y_{i,g,t}\leq 1\}} \Big[\prod_{\mathscr{T}\in\mathbb{T}_{z_g}}\prod_{t\in\mathscr{T}} f_{\Gamma(a,\lambda_{z_g,\mathscr{T}})}(y_{i,g,t}s_{i,g}/p_{\sigma_i,t})\Big] \tag{ygl}$$

To do this, we use a modification of the method of Kume and Walker, "Sampling from Compositional and Directional Distributions", as follows. (We have also implemented Fearnhead and Meligkotsidou (2004)'s forward-backward method and used this to sample $\mathbf{y}$, but it is too computationally intensive to be useful, and also has numerical issues [which could possibly be overcome].)

Recall that we want to sample from

$$\begin{aligned}
f(\mathbf{y}|\boldsymbol{\lambda},\mathbf{p},\mathbf{z},\boldsymbol{\pi},\boldsymbol{\alpha}) &\propto \prod_{i=1}^{n}\prod_{g=1}^{G} 1_{\{\mathbf{y}_{i,g}:\sum_{t=1}^{T-1} y_{i,g,t}\leq 1\}} \Big[\prod_{\mathscr{T}\in\mathbb{T}_{z_g}}\prod_{t\in\mathscr{T}} f_{\Gamma(a,\lambda_{z_g,\mathscr{T}})}(y_{i,g,t}s_{i,g}/p_{\sigma_i,t})\Big] \\
&\propto \prod_{i=1}^{n}\prod_{g=1}^{G} 1_{\{\mathbf{y}_{i,g}:\sum_{t=1}^{T-1} y_{i,g,t}\leq 1\}} \Big[\prod_{\mathscr{T}\in\mathbb{T}_{z_g}}\prod_{t\in\mathscr{T}} {y_{i,g,t}}^{a-1}\exp(-\lambda_{z_g,\mathscr{T}}y_{i,g,t}s_{i,g}/p_{\sigma_i,t})\Big] \\
&\propto \prod_{i=1}^{n}\prod_{g=1}^{G} 1_{\{\mathbf{y}_{i,g}:\sum_{t=1}^{T-1} y_{i,g,t}\leq 1\}} \exp\Big(-\sum_{t=1}^{T-1}\lambda_{z_g,\mathscr{T}(z_g,t)}s_{i,g}p_{\sigma_i,t}^{-1}y_{i,g,t} - \lambda_{z_g,\mathscr{T}(z_g,T)}s_{i,g}p_{\sigma_i,T}^{-1}(1-\sum_{t=1}^{T-1}y_{i,g,t})\Big) \\
&\quad \Big(\prod_{t=1}^{T-1} y_{i,g,t}^{a-1}\Big)\Big(1-\sum_{t=1}^{T-1}y_{i,g,t}\Big)^{a-1}
\end{aligned}$$

We see that it is sufficient to be able to sample according to

$$f(\mathbf{y}) = 1_{\{\mathbf{y}:\sum_{t=1}^{T-1}y_t\leq 1\}}\exp\Big(-\sum_{t=1}^{T-1}\beta_t y_t - \beta_T(1-\sum_{t=1}^{T-1}y_t)\Big)\Big(\prod_{t=1}^{T-1}y_t^{\alpha_t-1}\Big)\Big(1-\sum_{t=1}^{T-1}y_t\Big)^{\alpha_T-1}$$

(We cannot sample from this density using exactly Kume and Walker's method, because of the extra term in the exponential.) We introduce latent random variables:

$$f(\mathbf{y},v,w) = 1_{\{\mathbf{y}:\sum_{t=1}^{T-1}y_t\leq 1\}}1_{\{v<\exp(-\sum_{t=1}^{T-1}\beta_t y_t-\beta_T(1-\sum_{t=1}^{T-1}y_t))\}}\Big(\prod_{t=1}^{T-1}y_t^{\alpha_t-1}\Big)1_{\{w<(1-\sum_{t=1}^{T-1}y_t)^{\alpha_T-1}\}}$$

The full conditionals are:

1. $f(v|\mathbf{y},w) \propto 1_{\{v<\exp(-\sum_{t=1}^{T-1}\beta_t y_t - \beta_T(1-\sum_{t=1}^{T-1} y_t))\}}$. To avoid numerical issues, we sample $\tilde{v} := \log v$ instead of v. We do this by:
   (a) Sample $r \sim \text{Uniform}(0,1)$.
   (b) Let $\upsilon = \sum_{t=1}^{T-1}\beta_t y_t + \beta_T(1-\sum_{t=1}^{T-1} y_t)$.
   (c) Let $\tilde{v} = (\log r) - \upsilon$.
   (For justification of this procedure, see the section on sampling **p**.)
2. $f(w|\mathbf{y},v) \propto 1_{\{w<(1-\sum_{t=1}^{T-1} y_t)^{\alpha_T - 1}\}}$.
3. $f(y_{t_0}|\mathbf{y}_{-t_0},v,w) \propto 1_{\{\mathbf{y}:\sum_{t=1}^{T-1} y_t \le 1\}} 1_{\{v<\exp(-\sum_{t=1}^{T-1}\beta_t y_t - \beta_T(1-\sum_{t=1}^{T-1} y_t))\}} \left(\prod_{t=1}^{T-1} y_t^{\alpha_t - 1}\right) 1_{\{w<(1-\sum_{t=1}^{T-1} y_t)^{\alpha_T-1}\}} \propto y_{t_0}^{\alpha_{t_0}-1} 1_{\{y_{t_0}\in(c,d)\}}$, where (c,d) is determined by the indicator functions:
   (a) $\sum_{t=1}^{T-1} y_t \le 1$,
   i.e., $y_{t_0} \le 1 - \sum_{t\ne t_0} y_t$.
   (b) (Recall that we require $\alpha_t > 1$, so $\alpha_t - 1 > 0$.)
   $w < (1-\sum_{t=1}^{T-1} y_t)^{\alpha_T-1}$,
   i.e., $w^{1/(\alpha_T-1)} < 1 - \sum_{t=1}^{T-1} y_t$,
   i.e., $y_{t_0} < 1 - \sum_{t\ne t_0} y_t - w^{1/(\alpha_T-1)}$.
   (c) (We assume that $\beta_T \le \beta_t$ for all other t, so that $\beta_{t_0} - \beta_T \ge 0$.)
   $v < \exp(-\sum_{t=1}^{T-1}\beta_t y_t - \beta_T(1-\sum_{t=1}^{T-1} y_t))$,
   i.e., $\log v < -\sum_{t=1}^{T-1}\beta_t y_t - \beta_T(1-\sum_{t=1}^{T-1} y_t) = -\sum_{t\ne t_0}\beta_t y_t - \beta_T(1-\sum_{t\ne t_0} y_t) - \beta_{t_0}y_{t_0} + \beta_T y_{t_0}$,
   i.e., $(\beta_{t_0} - \beta_T)y_{t_0} < -\sum_{t\ne t_0}\beta_t y_t - \beta_T(1-\sum_{t\ne t_0} y_t) - \log v$,
   i.e., $y_{t_0} < [-\sum_{t\ne t_0}\beta_t y_t - \beta_T(1-\sum_{t\ne t_0} y_t) - \log v]/[\beta_{t_0} - \beta_T]$.

   Note that condition (a) is always true if condition (b) is true, since $w \ge 0$. So $c = 0$ and

$$d = \min\left(1 - \sum_{t\ne t_0} y_t - w^{1/(\alpha_T-1)}, \left[-\sum_{t\ne t_0}\beta_t y_t - \beta_T(1-\sum_{t\ne t_0} y_t) - \log v\right]/[\beta_{t_0} - \beta_T]\right)$$

   We can sample y_{t_0} using the inverse CDF method:
   (a) Sample $s \sim \text{Uniform}(0,1)$.
   (b) Set $y_{t_0} = F^{-1}(s)$ where

$$F(x) = \frac{\int_0^{\min(x,d)} y^{\alpha_{t_0}-1}\,dy}{\int_0^d y^{\alpha_{t_0}-1}\,dy} = \min((x/d)^{\alpha_{t_0}}, 1)$$

   so, given $s \in (0,1)$, solving $s = F(x) = (x/d)^{\alpha_{t_0}}$ gives

$$F^{-1}(s) = ds^{1/\alpha_{t_0}}$$

In code:

```
> # make sure that beta[t] >= beta[T] for all T, then call pseudo.km.for.y
> sample.pseudo.km.for.y <- function(T, old.y, alpha, beta, num.iters=1) {
+   # find the index t0 of the minimum beta
+   t0 <- which(beta == min(beta))[1]
+   map <- c((1:T)[-t0], t0)
+   if (t0 == 1)
+     inv <- c(T, 1:(T-1))
+   else if (t0 == T)
+     inv <- 1:T
+   else
+     inv <- c(1:(t0-1), T, t0:(T-1))
+
+   # map all variables so that t0 is last
+   n <- dim(old.y)[1]
+   G <- dim(old.y)[2]
+   old.y <- old.y[map]
+   alpha <- alpha[map]
+   beta <- beta[map]
+
+   # call pseudo.km.for.y
+   y <- pseudo.km.for.y(T, old.y, alpha, beta, num.iters)
+
+   # map back to the original ordering
+   y <- y[inv]
+   y
+ }
> pseudo.km.for.y <- function(T, old.y, alpha, beta, num.iters=1) {
+   stopifnot(all(alpha > 1))
+   stopifnot(all(beta >= beta[T]))
+   stopifnot(all(beta > 0))
+
+   y <- old.y
+   for (iter in 1:num.iters) {
+     # sample log v
+     upsilon.1 <- beta[1:(T-1)] %*% y[1:(T-1)]
+     upsilon.2 <- beta[T]*(1-sum(y[1:(T-1)]))
+     upsilon <- upsilon.1 + upsilon.2
+     r <- runif(1, 0, 1)
+     log.v <- log(r) - upsilon
+
+     # sample w
+     w.ub <- (1 - sum(y[1:(T-1)]))^(alpha[T]-1)
+     if (abs(w.ub - 0) < 1e-3) # XXX is the correct?
+       w <- 0
+     else
+       w <- runif(1, 0, w.ub)
+
+     # sample p
+     for (t0 in 1:(T-1)) {
+       # determine d
+       not.t0 <- (1:(T-1))[-t0]
+       sum.y <- sum(y[not.t0])
+       d1 <- 1 - sum.y - w^(1/(alpha[T]-1))
+       d2 <- ((-(beta[not.t0] %*% y[not.t0])
+             - beta[T]*(1 - sum.y) - log.v)
+            /(beta[t0] - beta[T]))
+       d <- min(d1, d2)
+       stopifnot(d > 0);
+
+       # sample using inverse CDF
+       s <- runif(1, 0, 1)
+       y[t0] <- d*s^(1/alpha[t0])
+     }
+     y[T] <- 1 - sum(y[1:(T-1)])
+   }
+   y
+ }
```

We test the above as follows.

1. Choose some $A = \times_{t=1}^{T-1}(a_t, b_t)$.
2. Sample some $\mathbf{y}_1, \ldots, \mathbf{y}_N$ using our code.
3. Compute $P(\mathbf{y} \in A) = [\int_A f(\mathbf{y})\,d\mathbf{y}]/[\int f(\mathbf{y})\,d\mathbf{y}]$ directly, using nested quadrature, where

$$f(\mathbf{y}) = 1_{\{\mathbf{y}:\sum_{t=1}^{T-1} y_t \le 1\}} \exp\Big(-\sum_{t=1}^{T-1} \beta_t y_t - \beta_T(1-\sum_{t=1}^{T-1} y_t)\Big)\Big(\prod_{t=1}^{T-1} y_t^{\alpha_t - 1}\Big)\Big(1-\sum_{t=1}^{T-1} y_t\Big)^{\alpha_T - 1}$$

4. Check that $|\{i : \mathbf{y}_i \in A\}|/N \approx P(\mathbf{y} \in A)$.

Since we made the assumption that $\beta_t \geq \beta_T$ for all t, we also have a wrapper procedure that ensures this condition is met. In order to test the wrapper, we run the above T times, with the smallest β_t originally in each position $1, \ldots, T$.

```
> integrate.2d <- function(f, lower, upper) {
+   integrate(function(xs) {
+     laply(xs, function(x) {
+       integrate(function(ys) {
+         laply(ys, function(y) {
+           f(c(x,y))
+         })
+       }, lower[2], upper[2])$value
+     })
+   }, lower[1], upper[1])$value
+ }
> test.pseudo.km.for.y <- function(T, alpha, beta, a, b) {
+   stopifnot(T == 3)
+
+   is.in.A <- function(y)
+     (all(a < y[1:(T-1)]) && all(y[1:(T-1)] < b))
+
+   f <- function(y) {
+     if (sum(y[1:(T-1)]) >= 1)
+       0
+     else {
+       f1 <- -beta[1:(T-1)] %*% y[1:(T-1)]
+       f2 <- -beta[T]*(1 - sum(y[1:(T-1)]))
+       f3 <- (alpha[1:(T-1)]-1) %*% log(y[1:(T-1)])
+       f4 <- (alpha[T]-1)*log(1 - sum(y[1:(T-1)]))
+       as.numeric(exp(f1+f2+f3+f4))
+     }
+   }
+
+   numer <- integrate.2d(function(y)
+     ifelse(is.in.A(y), f(y), 0), c(0,0), c(1,1))
+   denom <- integrate.2d(f, c(0,0), c(1,1))
+   true.prob <- numer/denom
+
+   # run this T times, once with the smallest beta in each place
+   t0 <- which(beta == min(beta))[1]
+   empirical.probs <- laply(1:T, function(t00) {
+     # interchange t0 and t00
+     map <- 1:T
+     map[t00] <- t0
+     map[t0] <- t00
+     beta0 <- beta[map]
+     alpha0 <- alpha[map]
+
+     # do the actual sampling
+     N <- 10000
+     num.good <- 0
+     #y <- c(1/3, 1/3, 1/3)
+     y <- runif(T, 0, 1)
+     y <- y/sum(y)
+     y <- sample.pseudo.km.for.y(T, y, alpha0, beta0, num.iters=1000)
+     for (i in 1:N) {
+       y <- sample.pseudo.km.for.y(T, y, alpha0, beta0, num.iters=1)
+       if (is.in.A(y))
+         num.good <- num.good + 1
+     }
+     empirical.prob <- num.good/N
+   })
+
+   list(true.prob=true.prob, empirical.probs=empirical.probs)
+ }
> if (run.slow.code) {
+   (function() {
+     T <- 3
+     #alpha <- c(5, 6, 7)
+     #beta <- c(3, 2, 1)
+     #a <- c(0.2, 0.3)
+     #b <- c(0.3, 0.4)
+     alpha <- c(1.01, 1.01, 1.01)
+     beta <- c(0.01, 0.002, 0.001)
+     a <- c(0, 0)
+     b <- c(0.2, 0.2)
+     results <- test.pseudo.km.for.y(T, alpha, beta, a, b)
+     print(results)
+   })()
+ }
```

The true probability and empirical probability are usually more or less close, and for some α, β, and A they are quite close. I went through the code and derivation closely and everything seems to be correct. Possibly the convergence is happening slowly in some cases?

Finally, we use this to write our function to sample $\mathbf{y}$:

```
> sample.y.unbelievably.slow <- function(n, G, T, old.y, p, z, alpha, lambda, s, sigma, bbT) {
+   laply(1:n, function(i) {
+     laply(1:G, function(g) {
+       # should this be commented out?:
+       #fm.y <- y[i,g,t]
+       fm.alpha <- array(alpha$a, T)
+       fm.beta <- array(NaN, T)
+       for (calT.index in 1:length(bbT[[z[g]]])) {
+         calT <- bbT[[z[g]]][[calT.index]]
+         for (t in calT)
+           fm.beta[t] <- lambda[[z[g]]][[calT.index]] * s[i,g] / p[sigma[i],t]
+       }
+       sample.pseudo.km.for.y(T, old.y[i,g,], fm.alpha, fm.beta, num.iters=1)
+     })
+   })
+ }
```

Rewrite in C

The above is quite slow, so we rewrite it in C as follows. First, we write the main part of the code in `sample_y.c`:

```
#include "sample_y.h"

// changes y
void sample_kw_for_y(size_t T, gsl_vector *y, gsl_vector *alpha, gsl_vector *beta, gsl_rng *rng)
{
  // check that alpha_t > 1 for all t
  #ifdef DEBUG
  for (size_t t = 0; t < T; t++)
    assert(alpha[t] > 1);
  #endif

  // find index of the minimum beta
  size_t t_min = gsl_vector_min_index(beta);

  // map all variables so t_min is last
  gsl_vector_swap_elements(y, t_min, T-1);
  gsl_vector_swap_elements(alpha, t_min, T-1);
  gsl_vector_swap_elements(beta, t_min, T-1);

  // sample log v
  double log_v;
  double sum_y = 0;
  {
    double upsilon_1 = 0;
    for (size_t t = 0; t < T-1; t++) {
      upsilon_1 += get1(beta, t) * get1(y, t);
      sum_y += get1(y, t);
    }
    double upsilon_2 = get1(beta, T-1) * (1 - sum_y);
    double upsilon = upsilon_1 + upsilon_2;
    double r = gsl_rng_uniform(rng);
    log_v = log(r) - upsilon;
  }

  // sample w
  double w;
  {
    double w_ub = pow(1 - sum_y, get1(alpha, T-1) - 1);
    if (fabs(w_ub - 0) < 1e-3)
      w = 0;
    else
      w = gsl_ran_flat(rng, 0, w_ub);
  }

  // sample p
  for (size_t t0 = 0; t0 < T-1; t0++) {
    // preliminaries
    double beta_dot_y = 0;
    double sum_y = 0;
    for (size_t t = 0; t < T-1; t++) {
      if (t != t0) {
        beta_dot_y += get1(beta, t) * get1(y, t);
        sum_y += get1(y, t);
      }
    }

    // determine d
    double d1 = 1 - sum_y - pow(w, 1/(get1(alpha, T-1) - 1));
    double d2 = ((-beta_dot_y
                  - get1(beta, T-1)*(1 - sum_y) - log_v)
                /(get1(beta, t0) - get1(beta, T-1)));
    double d = GSL_MIN(d1, d2);
    assert(d > 0);

    // sample using inverse CDF
    double s = gsl_rng_uniform(rng);
    set1(y, t0, d * pow(s, 1/get1(alpha, t0)));
  }

  // set the last element
  sum_y = 0;
  for (size_t t = 0; t < T-1; t++)
    sum_y += get1(y, t);
  set1(y, T-1, 1 - sum_y);

  // map back to original ordering
  gsl_vector_swap_elements(y, t_min, T-1);
  gsl_vector_swap_elements(alpha, t_min, T-1);
  gsl_vector_swap_elements(beta, t_min, T-1);
}

// changes y
```

```
void sample_y(size_t n, size_t G, size_t T, cube_t *y, gsl_matrix *p, gsl_vector_int *z, double a, double a0, dou
{
  gsl_vector *kw_alpha = gsl_vector_alloc(T);
  gsl_vector_set_all(kw_alpha, a);
  for (size_t i = 0; i < n; i++) {
    for (size_t g = 0; g < G; g++) {
      gsl_vector *kw_beta = gsl_vector_alloc(T);
      for (size_t calT_index = 0; calT_index < bbTj_size(bbT, get1i(z, g)); calT_index++) {
        double lamb = get_lambda(lambda, get1i(z, g), calT_index);
        for (size_t t_index = 0; t_index < calT_size(bbT, get1i(z, g), calT_index); t_index++) {
          size_t t = get_t(bbT, get1i(z, g), calT_index, t_index);
          set1(kw_beta, t, lamb * get2(s, i, g) / get2(p, get1i(sigma, i), t));
        }
      }
      gsl_vector_view y_ig = cube_row(y, i, g);
      sample_kw_for_y(T, &y_ig.vector, kw_alpha, kw_beta, rng);
    }
  }
}

#ifdef SAMPLE_Y_TEST
/*
   gcc -g -DSAMPLE_Y_TEST -I/s/include -L/s/lib --std=c99 sample_y.c cube.c bbT.c lambda.c -lgsl -lgslcblas -lm
   LD_LIBRARY_PATH=/s/lib valgrind --leak-check=full ./a.out
*/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <assert.h>
#include <string.h>
#include <gsl/gsl_rng.h>
int main(int argc, char **argv)
{
  size_t T = 3;

  gsl_vector *y = gsl_vector_calloc(T);
  set1(y, 0, 0.2); set1(y, 1, 0.4); set1(y, 2, 0.4);

  gsl_vector *alpha = gsl_vector_calloc(T);
  set1(alpha, 0, 1.1); set1(alpha, 1, 1.7); set1(alpha, 2, 2.1);

  gsl_vector *beta = gsl_vector_calloc(T);
  //set1(beta, 0, 10.8); set1(beta, 1, 13.6); set1(beta, 2, 7.8);
  set1(beta, 0, 510.8); set1(beta, 1, 513.6); set1(beta, 2, 507.8);

  const gsl_rng_type *rT;
  gsl_rng *rng;
  gsl_rng_env_setup();
  rT = gsl_rng_default;
  rng = gsl_rng_alloc(rT);

  unsigned long int seed = time(0);
  gsl_rng_set(rng, seed);

  sample_kw_for_y(T, y, alpha, beta, rng);

  gsl_vector_fprintf(stdout, y, "%f");

  gsl_rng_free(rng);
  gsl_vector_free(y);
  gsl_vector_free(alpha);
  gsl_vector_free(beta);
}
#endif
```

We will call this using the `inline` package. We make a convenience wrapper first:

```
> require("inline")
> nf.cfunction <- function(sig, code, convention=".C", language="C", includes=NA, cppargs=NA, libargs=NA, verbose
+   d <- function(str) gsub("\\$D", getwd(), str)
+   if (is.na(includes)[1])
+     includes <- c("#include \"cube.h\"",
+                   "#include \"bbT.h\"",
+                   "#include \"lambda.h\"",
+                   "#include \"shortcuts.h\"",
+                   "#include \"sample_y.h\"",
+                   "#include \"gsl/gsl_rng.h\"",
+                   "#include \"gsl/gsl_matrix.h\"",
+                   "#include \"gsl/gsl_vector.h\"",
+                   "#include \"math.h\"",
+                   "#include \"time.h\"",
+                   "#include \"assert.h\"",
+                   "#include \"stdlib.h\"",
+                   "#include \"stdio.h\"")
+   if (is.na(cppargs)[1])
+     cppargs <- d("-I/s/include -I$D")
+   if (is.na(libargs)[1])
+     libargs <- d("$D/sample_y.o $D/cube.o $D/bbT.o $D/lambda.o -L/s/lib -lgsl -lgslcblas -lm")
+   cfunction(sig, code, convention=convention, language=language, includes=includes, cppargs=cppargs, libargs=li
+ }
```

We also have an auxiliary function to make a linear array representation of $\boldsymbol{\lambda}$:

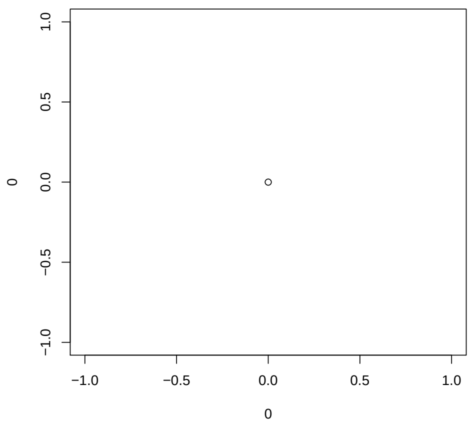
```
> flatten.lambda <- function(lambda) {
+   K <- sum(laply(lambda, function(a) sum(laply(a, function(b) length(b)))))
+   f <- array(0, K)
+   k <- 1
+   for (a in lambda)
+     for (b in a)
+       for (c in b) {
+         f[k] <- c
+         k <- k + 1
+       }
+   f
+ }
```

Next we wrap the C function as follows:

```
> sample.y.raw <- nf.cfunction(
+   signature(n="integer",
+             G="integer",
+             T="integer",
+             J="integer",
+             Sigma="integer",
+             y="numeric",
+             p="numeric",
+             z="integer",
+             a="numeric",
+             a0="numeric",
+             nu="numeric",
+             flattened_lambda="numeric",
+             s="numeric",
+             sigma="integer"),
+   "
+     cube_view_t          y_view = cube_view_array(y, *n, *G, *T);
+     gsl_matrix_view      p_view = gsl_matrix_view_array(p, *Sigma, *T);
+     gsl_vector_int      *z_vec = gsl_vector_int_alloc(*G);
+     for (size_t g = 0; g < *G; g++)
+       gsl_vector_int_set(z_vec, g, z[g]);
+     bbT_t               *bbT = bbT_alloc(*T, *J);
+     lambda_t            *lambda = lambda_alloc(bbT);
+     lambda_init(lambda, flattened_lambda);
+     gsl_matrix_view      s_view = gsl_matrix_view_array(s, *n, *G);
+     gsl_vector_int      *sigma_vec = gsl_vector_int_alloc(*n);
+     for (size_t i = 0; i < *n; i++)
+       gsl_vector_int_set(sigma_vec, i, sigma[i]);
+
+     static gsl_rng *rng = 0;
+     if (!rng) {
+       printf(\"initialized the gsl rng\\n\");
+       const gsl_rng_type *T;
+       gsl_rng_env_setup();
+       T = gsl_rng_default;
+       rng = gsl_rng_alloc(T);
+       unsigned long int seed = time(0);
+       gsl_rng_set(rng, seed);
+     }
+
+     sample_y(*n, *G, *T,
+              &y_view.cube,
+              &p_view.matrix,
+              z_vec,
+              *a, *a0, *nu,
+              lambda,
+              &s_view.matrix,
+              sigma_vec,
+              bbT,
+              rng);
+
+     lambda_free(lambda);
+     bbT_free(bbT);
+   ")
> sample.y <- function(n, G, T, J, Sigma, old_y, p, z, alpha, lambda, s, sigma) {
+   y <- aperm(old_y, c(3,2,1))
+   p <- t(p)
+   z <- t(z)-1
+   flattened_lambda <- flatten.lambda(lambda)
+   s <- t(s)
+   sigma <- sigma-1
+   res <- sample.y.raw(n, G, T, J, Sigma, y, p, z, alpha$a, alpha$a0, alpha$nu, flattened_lambda, s, sigma)
+   y <- array(res$y, c(T, G, n))
+   aperm(y, c(3,2,1))
+ }
```

We test this against the R version as follows:

```
> (function() {
+  if (run.slow.code) {
+     ex <- make.example()
+     y.prev <- array(0, c(ex$n, ex$G, ex$T))
+     for (i in 1:ex$n)
+       for (t in 1:ex$T)
+         y.prev[i,,t] <- ex$x[i,,t] * ex$p[ex$sigma[i],t] / ex$s[i,]
+
+     N <- 500
+     slow.y <- laply(1:N, function(k) sample.y.unbelievably.slow(ex$n, ex$G, ex$T, y.prev, ex$p, ex$z, ex$alpha,
+     fast.y <- laply(1:N, function(k) sample.y(ex$n, ex$G, ex$T, ex$J, ex$Sigma, y.prev, ex$p, ex$z, ex$alpha, e
+     save(slow.y,fast.y,file="tmp.Rda")
+
+     slow.df <- a2d(slow.y, c("N", "i", "g", "t"), "y")
+     slow.df$label = "slow"
+     fast.df <- a2d(fast.y, c("N", "i", "g", "t"), "y")
+     fast.df$label = "fast"
+     df <- rbind(slow.df, fast.df)
+     print(ggplot(df, aes(x=y, group=label)) + geom_density(aes(color=label)) + facet_grid(t ~ g))
+   } else
+     plot(0,0)
+ })()
```



3.2 Sample $\mathbf{p}|\mathbf{y},\mathbf{z},\boldsymbol{\pi},\boldsymbol{\alpha},\boldsymbol{\lambda}$

Next, we need to sample **p** given the others. Recall (Eq. (pgy))

$$\begin{aligned}
f(\mathbf{p}|\mathbf{y},\mathbf{z},\boldsymbol{\pi},\boldsymbol{\alpha},\boldsymbol{\lambda}) &\propto 1_{\cap_{\sigma=1}^{\Sigma}\{\sum_{t=1}^{T-1} p_{\sigma,t}\le 1\}} \prod_{i=1}^{n}\prod_{g=1}^{G}\Big(\Big[\prod_{\mathscr{T}\in\mathbb{T}_{z_g}}\prod_{t\in\mathscr{T}} f_{\Gamma(a,\lambda_{z_g,\mathscr{T}})}(y_{i,g,t}s_{i,g}/p_{\sigma_i,t})\Big]\Big[\prod_{t=1}^{T-1} p_{\sigma_i,t}^{-1}\Big]\Big) \\
&\propto \prod_{\sigma=1}^{\Sigma} 1_{\{\mathbf{p}_\sigma:\sum_{t=1}^{T-1} p_{\sigma,t}\le 1\}} \prod_{i:\sigma_i=\sigma}\prod_{g=1}^{G}\Big(\Big[\prod_{\mathscr{T}\in\mathbb{T}_{z_g}}\prod_{t\in\mathscr{T}} (1/p_{\sigma_i,t})^{a-1}\exp(-\lambda_{z_g,\mathscr{T}}y_{i,g,t}s_{i,g}/p_{\sigma_i,t})\Big]\Big[\prod_{t=1}^{T-1} p_{\sigma_i,t}^{-1}\Big]\Big) \\
&\propto \prod_{\sigma=1}^{\Sigma} 1_{\{\mathbf{p}_\sigma:\sum_{t=1}^{T-1} p_{\sigma,t}\le 1\}} \prod_{t=1}^{T} p_{\sigma,t}^{-c_{1,\sigma,t}}\exp(-c_{2,\sigma,t}p_{\sigma,t}^{-1}) \qquad \text{(pgy2)}
\end{aligned}$$

where

$$\begin{aligned}
c_{1,\sigma,t} &= |\{i:\sigma_i=\sigma\}|G\tilde{a}_t \\
c_{2,\sigma,t} &= \sum_{i:\sigma_i=\sigma}\sum_{g=1}^{G}\lambda_{z_g,\mathscr{T}(z_g,t)}y_{i,g,t}s_{i,g} \\
\tilde{a}_t &= \begin{cases} a & \text{if } t<T \\ a-1 & \text{if } t=T \end{cases} \\
\mathscr{T}(j,t) &= \mathscr{T}\in\mathbb{T}_j \text{ s.t. } t\in\mathscr{T}
\end{aligned}$$

We sample this using a modified (even more than for **y**) version of Kume and Walker's approach. (We have also investigated using a Metropolis-Hasting step, with a Dirichlet$((-c_{1,\sigma,t}-\frac{c_{2,\sigma,t}}{p_{\sigma,t}\log p_{\sigma,t}})t=1^T)$ proposal.)

To do so, we think of $p_{\sigma,T} = 1 - \sum_{t=1}^{T-1} p_{\sigma,t}$, so the above can be rewritten as

$$f(\mathbf{p}|\mathbf{y},\mathbf{z},\boldsymbol{\pi},\boldsymbol{\alpha},\boldsymbol{\lambda}) \propto \prod_{\sigma=1}^{\Sigma} 1_{\{\mathbf{p}_\sigma : \sum_{t=1}^{T-1} p_{\sigma,t} \le 1\}} \exp\Big(-\sum_{t=1}^{T-1} c_{2,\sigma,t} p_{\sigma,t}^{-1} - c_{2,\sigma,T}\Big(1 - \sum_{t=1}^{T-1} p_{\sigma,t}\Big)^{-1}\Big) \Big(\prod_{t=1}^{T-1} p_{\sigma,t}^{-c_{1,\sigma,t}}\Big)\Big(1 - \sum_{t=1}^{T-1} p_{\sigma,t}\Big)^{-c_{1,\sigma,T}}$$

Since the $\mathbf{p}_\sigma$ are independent, we simplify notation for fixed σ by substituting $\mathbf{p} = \mathbf{p}_\sigma$, $\alpha_t = c_{1,\sigma,t}$, and $\beta_t = c_{2,\sigma,t}$. Using this notation, we want to sample according to

$$f(\mathbf{p}) \propto 1_{\{\mathbf{p} : \sum_{t=1}^{T-1} p_t \le 1\}} \exp\Big(-\sum_{t=1}^{T-1} \beta_t p_t^{-1} - \beta_T \big(1 - \sum_{t=1}^{T-1} p_t\big)^{-1}\Big)\Big(\prod_{t=1}^{T-1} p_t^{-\alpha_t}\Big)\Big(1 - \sum_{t=1}^{T-1} p_t\Big)^{-\alpha_T}$$

Auxiliary random variables

We introduce auxiliary random variables v and w to get

$$f(\mathbf{p},v,w) \propto 1_{\{\mathbf{p} : \sum_{t=1}^{T-1} p_t \le 1\}} 1_{\{v < \exp(-\sum_{t=1}^{T-1} \beta_t p_t^{-1} - \beta_T (1-\sum_{t=1}^{T-1} p_t)^{-1})\}} \Big(\prod_{t=1}^{T-1} p_t^{-\alpha_t}\Big) 1_{\{w < (1-\sum_{t=1}^{T-1} p_t)^{-\alpha_T}\}}$$

Substitutions

For numerical reasons, we substitute $\tilde{v} = \log v$ and $\tilde{w} = w^{-1/\alpha_T}$. Let $\phi(\mathbf{p},\tilde{v},\tilde{w}) = (\phi_{1,1}(p_1),\ldots,\phi_{1,T-1}(p_{T-1}),\phi_2(\tilde{v}),\phi_3(\tilde{w}))$, where

1. $\phi_{1,t}(p_t) = p_t$,
2. $\phi_2(\tilde{v}) = \exp(\tilde{v})$,
3. $\phi_3(\tilde{w}) = \tilde{w}^{-\alpha_T}$.

Note that

$$f(\mathbf{p},\tilde{v},\tilde{w}) \propto f(\phi(\mathbf{p},\tilde{v},\tilde{w}))|\det J_\phi(\mathbf{p},\tilde{v},\tilde{w})|$$

where $|J_\phi|$ is the Jacobian determinant. To find the Jacobian determinant, note that

1. $\frac{\partial \phi_{1,t}}{\partial p_t} = 1, \quad \frac{\partial \phi_{1,t}}{\partial p_{t'}} = 0 \quad (t \ne t'), \quad \frac{\partial \phi_{1,t}}{\partial \tilde{v}} = 0, \quad \frac{\partial \phi_{1,t}}{\partial \tilde{w}} = 0,$
2. $\frac{\partial \phi_2}{\partial p_t} = 0, \quad \frac{\partial \phi_2}{\partial \tilde{v}} = \exp(\tilde{v}), \quad \frac{\partial \phi_2}{\partial \tilde{w}} = 0,$
3. $\frac{\partial \phi_3}{\partial p_t} = 0, \quad \frac{\partial \phi_3}{\partial \tilde{v}} = 0, \quad \frac{\partial \phi_3}{\partial \tilde{w}} = -\alpha_T \tilde{w}^{-\alpha_T - 1}.$

So the Jacobian matrix J is diagonal, with determinant $\det J_\phi(\mathbf{p},\tilde{v},\tilde{w}) = 1 \cdots 1 \cdot \exp(\tilde{v}) \cdot (-\alpha_T \tilde{w}^{-\alpha_T-1})$ so $|\det J_\phi| = \alpha_T \tilde{w}^{-\alpha_T-1}\exp(\tilde{v})$ and

$$\begin{aligned} f(\mathbf{p},\tilde{v},\tilde{w}) &\propto f(\phi(\mathbf{p},\tilde{v},\tilde{w}))\tilde{w}^{-\alpha_T-1}\exp(\tilde{v}) \\ &= 1_{\{\mathbf{p} : \sum_{t=1}^{T-1} p_t \le 1\}} 1_{\{\exp(\tilde{v}) < \exp(-\sum_{t=1}^{T-1} \beta_t p_t^{-1} - \beta_T (1-\sum_{t=1}^{T-1} p_t)^{-1})\}} \Big(\prod_{t=1}^{T-1} p_t^{-\alpha_t}\Big) 1_{\{\tilde{w}^{-\alpha_T} < (1-\sum_{t=1}^{T-1} p_t)^{-\alpha_T}\}} \tilde{w}^{-\alpha_T-1}\exp(\tilde{v}) \\ &= 1_{\{\mathbf{p} : \sum_{t=1}^{T-1} p_t \le 1\}} 1_{\{\tilde{v} < -\sum_{t=1}^{T-1} \beta_t p_t^{-1} - \beta_T (1-\sum_{t=1}^{T-1} p_t)^{-1}\}} 1_{\{\tilde{w} > 1 - \sum_{t=1}^{T-1} p_t\}} \tilde{w}^{-\alpha_T-1}\exp(\tilde{v})\Big(\prod_{t=1}^{T-1} p_t^{-\alpha_t}\Big) \end{aligned}$$

Full conditional for $\tilde{v}|\mathbf{p},\tilde{w}$

The density of $\tilde{v}|\mathbf{p},\tilde{w}$ is

$$\begin{aligned} f(\tilde{v}|\mathbf{p},\tilde{w}) &\propto 1_{\{\tilde{v} < -\sum_{t=1}^{T-1} \beta_t p_t^{-1} - \beta_T (1-\sum_{t=1}^{T-1} p_t)^{-1}\}} \exp(\tilde{v}) \\ &= 1_{(-\infty,-\upsilon)}(\tilde{v})\exp(\tilde{v}) \end{aligned}$$

where $\upsilon = \sum_{t=1}^{T-1} \beta_t p_t^{-1} + \beta_T (1 - \sum_{t=1}^{T-1} p_t)^{-1}$. We sample according to this density as follows.

First, note that $\int_{\mathbb{R}} f(\tilde{v}|\mathbf{p}, \tilde{w})\, d\tilde{v} = \exp(-\upsilon)$ so

$$f(\tilde{v}|\mathbf{p}, \tilde{w}) = 1_{(-\infty,-\upsilon)}(\tilde{v}) \exp(\tilde{v} + \upsilon)$$

and the cdf

$$F(\tilde{v}) = 1_{(-\infty,-\upsilon)}(\tilde{v}) \exp(\tilde{v} + \upsilon)$$

also. Note that F is bijective on $(-\infty, -\upsilon)$ with inverse

$$F^{-1}(r) = (\log r) - \upsilon$$

since $r = \exp(\tilde{v} + \upsilon)$ iff $\log r = \tilde{v} + \upsilon$ iff $\tilde{v} = (\log r) - \upsilon$.

Thus we can sample $\tilde{v}|\mathbf{p}, \tilde{w}$ using the inverse cdf method:

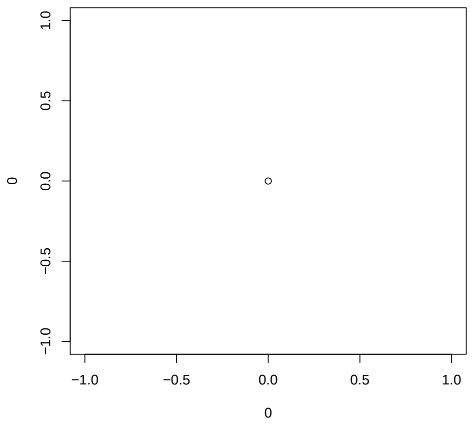
1. Sample $r \sim \text{Uniform}(0, 1)$.
2. Let $\tilde{v} = (\log r) - \upsilon$.

We implement this as follows:

```
> sample.tilde.v <- function(T, p, beta) {
+   upsilon <- (beta[1:(T-1)] %*% (1/p[1:(T-1)])
+             + beta[T] / (1 - sum(p[1:(T-1)])))
+   r <- runif(1, 0, 1)
+   log(r) - upsilon
+ }
```

We test this by sampling a bunch of $\tilde{v}$ using the above method, and a bunch of v according to $\text{Uniform}(0, \exp(-\upsilon))$, then taking the log, and comparing the results:

```
> (function() {
+   if (run.slow.code) {
+     T <- 3
+     p <- runif(T, 0, 1)
+     p <- p/sum(p)
+     beta <- runif(T, 0, 10)
+     upsilon <- (beta[1:(T-1)] %*% (1/p[1:(T-1)])
+               + beta[T] / (1 - sum(p[1:(T-1)])))
+     tilde.v <- laply(1:1000, function(i) sample.tilde.v(T, p, beta))
+     log.v <- log(runif(1000, 0, exp(-upsilon)))
+     df <- rbind(data.frame(tilde.v=tilde.v, type="our method"),
+                 data.frame(tilde.v=log.v, type="log of uniform"))
+     print(ggplot(df, aes(x = tilde.v)) + geom_histogram() + facet_grid(. ~ type))
+   } else {
+     plot(0,0)
+   }
+ })()
```



Full conditional for $\tilde{w}|\mathbf{p},\tilde{v}$

The density of $\tilde{w}|\mathbf{p},\tilde{v}$ is

$$f(\tilde{w}|\mathbf{p},\tilde{v}) \propto 1_{\{\tilde{w}>1-\sum_{t=1}^{T-1} p_t\}}\tilde{w}^{-\alpha_T-1} = 1_{\{\tilde{w}>p_T\}}\tilde{w}^{-\alpha_T-1}$$

where $p_T = 1-\sum_{t=1}^{T-1} p_t$. Note that $\int_{\mathbb{R}} f(\tilde{w}|\mathbf{p},\tilde{v})\,d\tilde{w} = \alpha_T^{-1} p_T^{-\alpha_T}$ so

$$f(\tilde{w}|\mathbf{p},\tilde{v}) = 1_{\{\tilde{w}>1-\sum_{t=1}^{T-1} p_t\}}\alpha_T p_T^{\alpha_T}\tilde{w}^{-\alpha_T-1}$$

and the cdf

$$F(\tilde{w}) = \begin{cases} 0 & \tilde{w} \le p_T \\ 1-(\tilde{w}/p_T)^{-\alpha_T} & \tilde{w} > p_T \end{cases}$$

The cdf is bijective on (p_T,∞) with inverse

$$F^{-1}(r) = p_T(1-r)^{-1/\alpha_T}$$

since $1-(\tilde{w}/p_T)^{-\alpha_T} = r$ iff $\tilde{w} = p_T(1-r)^{-1/\alpha_T}$.

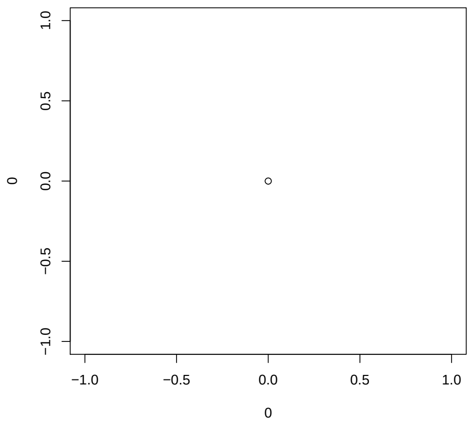
Thus we can sample $\tilde{w}|\mathbf{p},\tilde{v}$ using the inverse CDF method:

```
> sample.tilde.w <- function(T, p, alpha) {
+   p.T <- 1 - sum(p[1:(T-1)])
+   r <- runif(1, 0, 1)
+   exp(log(p.T) - (1/alpha[T])*log(1-r))
+ }
```

We doublecheck this as follows:

1. Sample a bunch of $\tilde{w}_i$ using the above.
2. Sample a bunch of w_i using the original formula.
3. Fix an interval (a,b).
4. Check that $|\{i : \tilde{w}_i \in (a,b)\}| \approx |\{i : w_i^{-1/\alpha_T} \in (a,b)\}|$.

```
> (function() {
+   if (run.slow.code) {
+     sample.w <- function(num.samples, T, p, alpha)
+       runif(num.samples, 0, (1-sum(p[1:(T-1)]))^(-alpha[T]))
+
+     T <- 3
+     alpha <- runif(T, 0, 10)
+     p <- runif(T, 0, 1)
+     p <- p/sum(p)
+     N <- 10000
+     w <- sample.w(N, T, p, alpha)
+     mapped.w <- w^(-1/alpha[T])
+     tilde.w <- laply(1:N, function(i) sample.tilde.w(T, p, alpha))
+
+     df <- rbind(data.frame(tilde.w=tilde.w, type="our method"),
+                 data.frame(tilde.w=mapped.w, type="transformed uniform"))
+     print(ggplot(df, aes(x = tilde.w)) + geom_histogram() + facet_grid(. ~ type))
+   } else {
+     plot(0,0)
+   }
+ })()
```



Full conditional for $p_t|\mathbf{p}_{-t},\tilde{v},\tilde{w}$

Note that the density of $p_{t_0}|\mathbf{p}_{-t_0},\tilde{v},\tilde{w}$ is

$$\begin{aligned} f(p_{t_0}|\mathbf{p}_{-t_0},\tilde{v},\tilde{w}) &\propto 1_{\{\mathbf{p}:\sum_{t=1}^{T-1} p_t \le 1\}} 1_{\{\tilde{v}<-\sum_{t=1}^{T-1}\beta_t p_t^{-1}-\beta_T(1-\sum_{t=1}^{T-1}p_t)^{-1}\}} 1_{\{\tilde{w}>1-\sum_{t=1}^{T-1}p_t\}} p_{t_0}^{-\alpha_{t_0}} \\ &= 1_A(p_{t_0}) p_{t_0}^{-\alpha_{t_0}} \end{aligned}$$

where A is the collection of $p_{t_0} \in (0,1)$ such that (letting $\bar{p} = \sum_{t\neq t_0} p_t$ and $d = \sum_{t\neq t_0} \beta_t/p_t$)

1. $\sum_{t=1}^{T-1} p_t \le 1$, i.e., $p_{t_0} \in (0, 1-\bar{p})$
2. $\tilde{w} > 1-\sum_{t=1}^{T-1} p_t$, i.e., $p_{t_0} \in (1-\bar{p}-\tilde{w}, 1)$.
3. $\tilde{v} < -\sum_{t=1}^{T-1}\beta_t p_t^{-1} - \beta_T(1-\sum_{t=1}^{T-1}p_t)^{-1}$,
   i.e., $0 < ap_{t_0} + b + cp_{t_0}^{-1}$ where
   (a) $a = \tilde{v} + d$
   (b) $b = \beta_{t_0} - \beta_T - (1-\bar{p})(\tilde{v}+d)$
   (c) $c = -(1-\bar{p})\beta_{t_0}$

i.e., $0 < ap_{t_0}^2 + bp_{t_0} + c$. Let $g(p_{t_0}) = ap_{t_0}^2 + bp_{t_0} + c$. There are now several cases:

(a) If the discriminant $b^2 - 4ac < 0$, then g has no roots. In this case, g is positive everywhere if convex (i.e., if $a > 0$), and negative everywhere otherwise.

(b) If the discriminant $b^2 - 4ac = 0$, then g has one root. In this case, g is positive almost everywhere if convex (i.e., if $a > 0$), and negative almost everywhere otherwise.

(c) If the discriminant $b^2 - 4ac > 0$, then g has two roots $p_{*,1} < p_{*,2}$. In this case:
   i. If g is convex (i.e., $a > 0$), then g is positive in $(0, p_{*,1}) \cup (p_{*,2}, 1)$.
   ii. If g is concave (i.e., $a < 0$), then g is positive in $(p_{*,1}, p_{*,2})$.

We doublecheck the claims above as follows:

```
> (function () {
+   if (run.slow.code) {
+     N <- 10000
+     T <- 3
+     for (iter in 1:N) {
+       t0 <- sample.int(T-1, 1)
+       p <- runif(T, 0, 1)
+       p <- p/sum(p)
+       beta <- runif(T, 0, 10)
+       tilde.v <- runif(1, -100, -10)
+
+       not.t0 <- (1:(T-1))[-t0]
+       barp <- sum(p[not.t0])
+       d <- beta[not.t0] %*% (1/p[not.t0])
+       a <- tilde.v + d
+       b <- beta[t0] -beta[T] - (1-barp)*(tilde.v + d)
+       c <- -(1-barp)*beta[t0]
+
+       lhs.1 <- tilde.v
+       rhs.1 <- -beta[1:(T-1)] %*% (1/p[1:(T-1)]) - beta[T]/(1-sum(p[1:(T-1)]))
+       bool.1 <- (lhs.1 < rhs.1)
+
+       lhs.2 <- 0
+       rhs.2 <- a*p[t0]^2 + b*p[t0] + c
+       bool.2 <- (lhs.2 < rhs.2)
+
+       discriminant <- b^2-4*a*c
+       # no roots -> positive everywhere if convex, negative everywhere o.w.
+       if (discriminant < 0)
+         bool.3 <- a > 0
+       # one root -> positive a.e if convex, negative a.e o.w.
+       else if (discriminant == 0)
+         bool.3 <- a > 0
+       # two roots
+       else {
+         roots <- sort(c((-b+sqrt(b^2-4*a*c))/(2*a),
+                         (-b-sqrt(b^2-4*a*c))/(2*a)))
+         if (a > 0)
+           bool.3 <- p[t0] < roots[1] || p[t0] > roots[2]
+         else
+           bool.3 <- p[t0] > roots[1] && p[t0] < roots[2]
+       }
+
+       stopifnot(bool.1 == bool.2)
+       stopifnot(bool.1 == bool.3)
+     }
+     cat(sprintf("passed %d tests\n", N))
+   }
+ })()
```

Altogether, $A = (c_1, d_1) \cup (c_2, d_2)$, where

1. If $b^2 - 4ac < 0$ and $a > 0$, then

    (a) $c_1 = \max(1-\bar{p}-\tilde{w}, 0)$,
    (b) $d_1 = 1-\bar{p}$,
    (c) $c_2 = d_2 = 1$, i.e., $(c_2, d_2) = \varnothing$.
2. If $b^2-4ac < 0$ and $a < 0$, then
    (a) $c_1 = d_1 = 1$, i.e., $(c_1, d_1) = \varnothing$,
    (b) $c_2 = d_2 = 1$, i.e., $(c_2, d_2) = \varnothing$.
3. If $b^2-4ac > 0$ and $a > 0$, then
    (a) $c_1 = \max(1-\bar{p}-\tilde{w}, 0)$,
    (b) $d_1 = \min(1-\bar{p}, p_{*,1})$,
    (c) $c_2 = \max(1-\bar{p}-\tilde{w}, p_{*,2})$,
    (d) $d_2 = 1-\bar{p}$.
4. If $b^2-4ac > 0$ and $a < 0$, then
    (a) $c_1 = \max(1-\bar{p}-\tilde{w}, p_{*,1}, 0)$,
    (b) $d_1 = \min(1-\bar{p}, p_{*,2})$,
    (c) $c_2 = d_2 = 1$, i.e., $(c_2, d_2) = \varnothing$.

Above, we ignore the edge cases $b^2-4ac = 0$ or $a = 0$ for now, since they hardly ever (almost surely never) occur. The second case above, when both intervals are empty, should only happen in case of an error. We implement this as follows:

```
> intervals.for.p.t0 <- function(T, t0, p, tilde.v, tilde.w, beta) {
+   not.t0 <- (1:(T-1))[-t0]
+   barp <- sum(p[not.t0])
+   d <- beta[not.t0] %*% (1/p[not.t0])
+   a <- tilde.v + d
+   b <- beta[t0] -beta[T] - (1-barp)*(tilde.v + d)
+   c <- -(1-barp)*beta[t0]
+   discriminant <- b^2-4*a*c
+   if (discriminant < 0) {
+     if (a > 0) {
+       list(num.intervals = 1,
+            c1 = max(1-barp-tilde.w, 0),
+            d1 =     1-barp)
+     } else if (a < 0) {
+       list(num.intervals = 0)
+     } else {
+       stop("in intervals.for.p.t0, a == 0 is not implemented yet")
+     }
+   } else if (discriminant > 0) {
+     roots <- sort(c((-b+sqrt(b^2-4*a*c))/(2*a),
+                     (-b-sqrt(b^2-4*a*c))/(2*a)))
+     if (a > 0) {
+       list(num.intervals = 2,
+            c1 = max(1-barp-tilde.w, 0),
+            d1 = min(1-barp,         roots[1]),
+            c2 = max(1-barp-tilde.w, roots[2]),
+            d2 =     1-barp)
+     } else if (a < 0) {
+       list(num.intervals = 1,
+            c1 = max(1-barp-tilde.w, roots[1], 0),
+            d1 = min(1-barp,         roots[2]))
+     } else {
+       stop("in intervals.for.p.t0, a == 0 is not implemented yet")
+     }
+   } else {
+     stop("in intervals.for.p.t0, discriminant == 0 is not implemented yet")
+   }
+ }
> is.in.interval.for.p.t0 <- function(p.t0, intervals) {
+   if (intervals$num.intervals == 0)
+     FALSE
+   else if (intervals$num.intervals == 1)
+     p.t0 > intervals$c1 && p.t0 < intervals$d1
+   else {
+     bool.2a <- (p.t0 > intervals$c1 && p.t0 < intervals$d1)
+     bool.2b <- (p.t0 > intervals$c2 && p.t0 < intervals$d2)
+     bool.2a || bool.2b
+   }
+ }
```

We doublecheck this as follows:

```
> (function () {
+   if (run.slow.code) {
+     N <- 10000
+     T <- 3
+     num.both.true <- 0
+     for (iter in 1:N) {
+       # sample an initial state
+       t0 <- sample.int(T-1, 1)
+       p <- runif(T, 0, 1)
+       p <- p/sum(p)
+       beta <- runif(T, 0, 10)
+       tilde.v <- runif(1, -100, -10)
+       tilde.w <- runif(1, 0, 1)
+
+       # sample a new p_{t_0}
+       p[t0] <- runif(1, 0, 1)
+
+       # check if each separate condition holds
+       bool.1a <- (sum(p[1:(T-1)]) <= 1)
+       bool.1b <- (tilde.v < -beta[1:(T-1)] %*% (1/p[1:(T-1)]) - beta[T]/(1-sum(p[1:(T-1)])))
+       bool.1c <- (tilde.w > 1-sum(p[1:(T-1)]))
+       bool.1 <- (bool.1a && bool.1b && bool.1c)
+
+       # check if p_{t_0} is in the interval we compute
+       intervals <- intervals.for.p.t0(T, t0, p, tilde.v, tilde.w, beta)
+       bool.2 <- is.in.interval.for.p.t0(p[t0], intervals)
+
+       stopifnot(bool.1 == bool.2)
+       if (bool.1)
+         num.both.true <- num.both.true + 1
+     }
+     cat(sprintf("passed %d tests (in %d cases, both were true)\n", N, num.both.true))
+   }
+ })()
```

The cdf of $p_{t_0}|\mathbf{p}_{-t_0}, \tilde{v}, \tilde{w}$ is as follows. First, note (recalling that always $p_{t_0} \in [0,1]$) that

$$F(x) = \frac{1}{Z}\int_0^x [1_{(c_1,d_1)}(y) + 1_{(c_2,d_2)}(y)]y^{-\alpha_{t_0}}\,dy \quad \text{where } Z = \int_0^1 [1_{(c_1,d_1)}(y) + 1_{(c_2,d_2)}(y)]y^{-\alpha_{t_0}}\,dy$$

Note that

$$\int_0^x 1_{(c,d)}(y)y^{-\alpha_{t_0}}\,dy = \frac{c^{-(\alpha_{t_0}-1)} - \min(x,d)^{-(\alpha_{t_0}-1)}}{\alpha_{t_0}-1} = (1/h)[c^{-h} - \min(x,d)^{-h}]$$

where $h = \alpha_{t_0} - 1$. So

$$Z = \frac{c_1^{-h} - d_1^{-h}}{h} + \frac{c_2^{-h} - d_2^{-h}}{h} = Z'/h$$

where $Z' = c_1^{-h} - d_1^{-h} + c_2^{-h} - d_2^{-h}$. Thus

$$F(x) = \begin{cases} 0 & x < c_1 \\ (1/Z')[c_1^{-h} - x^{-h}] & x \in (c_1, d_1) \\ (1/Z')[c_1^{-h} - d_1^{-h}] & x \in (d_1, c_2) \\ (1/Z')[c_1^{-h} - d_1^{-h} + c_2^{-h} - x^{-h}] & x \in (c_2, d_2) \\ 1 & x > d_2 \end{cases}$$

In code:

```
> cdf.for.p.t0 <- function(x, intervals, alpha.t0) {
+   h <- alpha.t0 - 1
+   if (intervals$num.intervals == 0)
+     stop("cdf.for.p.t0: no intervals")
+   else if (intervals$num.intervals == 1) {
+     c1 <- intervals$c1
+     d1 <- intervals$d1
+     Z.prime <- c1^(-h) - d1^(-h)
+     if      (x < c1)            0
+     else if (x >= c1 && x < d1) (1/Z.prime) * (c1^(-h) - x^(-h))
+     else if (x >= d1)           1
+     else                        stop("cdf.for.p.t0: unexpected value")
+   } else if (intervals$num.intervals == 2) {
+     c1 <- intervals$c1
+     d1 <- intervals$d1
+     c2 <- intervals$c2
+     d2 <- intervals$d2
+     Z.prime <- c1^(-h) - d1^(-h) + c2^(-h) - d2^(-h)
+     if      (x < c1)            0
+     else if (x >= c1 && x < d1) (1/Z.prime) * (c1^(-h) - x^(-h))
+     else if (x >= d1 && x < c2) (1/Z.prime) * (c1^(-h) - d1^(-h))
+     else if (x >= c2 && x < d2) (1/Z.prime) * (c1^(-h) - d1^(-h)
+                                              + c2^(-h) - x^(-h))
+     else if (x >= d2)           1
+     else                        stop("cdf.for.p.t0: unexpected value")
+   } else
+     stop("cdf.for.p.t0: unexpected value")
+ }
```

We doublecheck this by comparing our formula for the cdf to an empirical cdf. We make the empirical cdf by sampling a bunch of points x according to the distribution of $p_{t_0} | \mathbf{p}_{-\mathbf{t}_0}, \tilde{v}, \tilde{w}$ using brute force rejection:

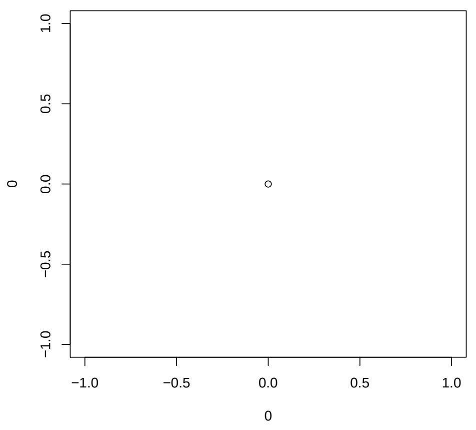
1. Let $y_* = \max_{x \in A} x^{-\alpha_{t_0}}$, computed using a grid.
2. Draw $x \sim \text{Uniform}(0, 1)$.
3. Draw $y \sim \text{Uniform}(0, y_*)$.
4. If $y > x^{-\alpha_{t_0}}$, reject and try again, else continue.
5. If $x \notin A$, reject and try again, else accept.

Note that after the first rejection, x follows $x^{-\alpha_{t_0}}$, and after the second rejection, it follows the distribution of $p_{t_0} | \mathbf{p}_{-\mathbf{t}_0}, \tilde{v}, \tilde{w}$.

```
> brute.force.empirical.cdf.for.p.t0 <- function(intervals, alpha.t0) {
+   # empirical and true cdfs
+   N <- 1000
+   if (intervals$num.intervals == 1)
+     grid <- seq(intervals$c1, intervals$d1, len=100)
+   else if (intervals$num.intervals == 2)
+     grid <- c(seq(intervals$c1, intervals$d1, len=100),
+               seq(intervals$c2, intervals$d2, len=100))
+   y.star <- max(grid^(-alpha.t0))
+   x <- laply(1:N, function(i) {
+     while (1) {
+       xi <- runif(1, 0, 1)
+       yi <- runif(1, 0, y.star)
+       if (yi < xi^(-alpha.t0)
+           && is.in.interval.for.p.t0(xi, intervals))
+         break
+     }
+     xi
+   })
+   emp.cdf <- function(y) length(which(x < y))/N
+   emp.cdf
+ }
> doublecheck.of.empirical.cdf.for.p.t0 <- function(num.intervals) {
+   # sample alpha_{t_0} and intervals
+   alpha.t0 <- 1.5
+   if (num.intervals == 1) {
+     #cd <- sort(runif(2, 0, 1))
+     cd <- c(0.2, 0.5)
+     intervals <- list(num.intervals = 1,
+                       c1 = cd[1],
+                       d1 = cd[2])
+   } else {
+     cd <- sort(runif(4, 0, 1))
+     cd <- c(0.2, 0.5, 0.6, 0.8)
+     intervals <- list(num.intervals = 2,
+                       c1 = cd[1],
+                       d1 = cd[2],
+                       c2 = cd[3],
+                       d2 = cd[4])
+   }
+
+   emp.cdf <- brute.force.empirical.cdf.for.p.t0(intervals, alpha.t0)
+   true.cdf <- function(y) cdf.for.p.t0(y, intervals, alpha.t0)
+
+   # plot
+   grid <- (1:100)/100
+   df <- rbind(data.frame(x=grid, cdf=laply(grid, emp.cdf), type="empirical cdf"),
+               data.frame(x=grid, cdf=laply(grid, true.cdf), type="our cdf"))
+   print(ggplot(df, aes(x, cdf, group=type)) + geom_line(aes(color=type)))
+ }
```

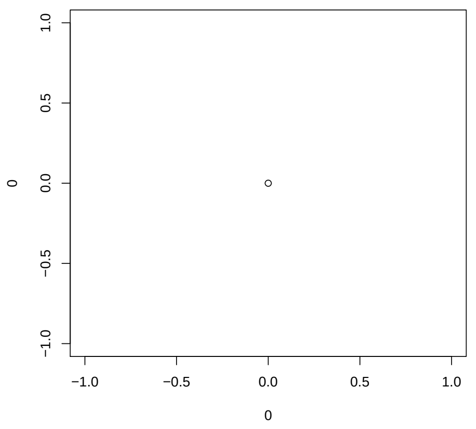
With one interval:

```
> (function() {
+   if (run.slow.code)
+     doublecheck.of.empirical.cdf.for.p.t0(1)
+   else
+     plot(0,0)
+ })()
```



With two intervals:

```
> (function() {
+   if (run.slow.code)
+     doublecheck.of.empirical.cdf.for.p.t0(2)
+   else
+     plot(0,0)
+ })()
```



The CDF is bijective on A, and its inverse is as follows. Let $g = (1/Z')[c_1^{-h} - d_1^{-h}]$.

1. If $s < g$ or $(c_2, d_2) = \varnothing$, then $F^{-1}(s)$ is the solution of $(1/Z')[c_1^{-h} - x^{-h}] = s$, namely $x = (c_1^{-h} - sZ')^{-1/h}$.
2. If $s > g$ and $(c_2, d_2) \neq \varnothing$, then $F^{-1}(s)$ is the solution of $(1/Z')[c_1^{-h} - d_1^{-h} + c_2^{-h} - x^{-h}] = s$, namely $x = (c_1^{-h} - d_1^{-h} + c_2^{-h} - sZ')^{-1/h}$.

Thus

$$F^{-1}(s) = \begin{cases} (c_1^{-h} - sZ')^{-1/h} & (c_2, d_2) = \varnothing \text{ or } s < g \\ (c_1^{-h} - d_1^{-h} + c_2^{-h} - sZ')^{-1/h} & (c_2, d_2) \neq \varnothing \text{ and } s > g \end{cases}$$

For numerical reasons, we note that if $(c_2,d_2)=\varnothing$ or $s<g$, then we may rewrite the inverse CDF as follows:

$$\begin{aligned}
F^{-1}(s) &= (c_1^{-h} - sZ')^{-1/h} \\
&= (c_1^{-h} - s(c_1^{-h} - d_1^{-h}))^{-1/h} \\
&= ((1-s)c_1^{-h} + sd_1^{-h})^{-1/h} \\
&= \exp(-1/h + \log[\exp(\log(1-s) - h\log c_1) + \exp(\log s - h\log d_1)]) \\
&= \exp(-1/h + \text{logsumexp}(\log(1-s) - h\log c_1, \log s - h\log d_1))
\end{aligned}$$

where $\text{logsumexp}(a,b) = \log(\exp(a)+\exp(b))$. Similarly, if $(c_2,d_2)\neq\varnothing$ and $s>g$, then

$$\begin{aligned}
F^{-1}(s) &= (c_1^{-h} - d_1^{-h} + c_2^{-h} - sZ')^{-1/h} \\
&= (c_1^{-h} - d_1^{-h} + c_2^{-h} - s(c_1^{-h} - d_1^{-h} + c_2^{-h} - d_2^{-h}))^{-1/h} \\
&= ((1-s)c_1^{-h} - (1-s)d_1^{-h} + (1-s)c_2^{-h} - sd_2^{-h})^{-1/h} \\
&= \exp(-1/h + \log[\exp(\log(1-s) - h\log c_1) - \exp(\log(1-s) - h\log d_1) + \exp(\log(1-s) - h\log c_2) + \exp(\log s - h\log d_2)]) \\
&= \exp(-1/h + \text{logsumexp}(\log(1-s) - h\log c_1, \log(1-s) - h\log d_1, \log(1-s) - h\log c_2, \log s - h\log d_2; +,-,+,+))
\end{aligned}$$

where $\text{logsumexp}(a,b,c,d;+,-,+,+) = \log(\exp(a)-\exp(b)+\exp(c)+\exp(d))$. Also note that g has log

$$\begin{aligned}
\log g &= \log((1/Z')[c_1^{-h} - d_1^{-h}]) \\
&= \log[c_1^{-h} - d_1^{-h}] - \log[c_1^{-h} - d_1^{-h} + c_2^{-h} - d_2^{-h}] \\
&= \log[\exp(-h\log c_1) - \exp(-h\log d_1)] - \log[\exp(-h\log c_1) - \exp(-h\log d_1) + \exp(-h\log c_2) - \exp(-h\log d_2)] \\
&= \text{logsumexp}(-h\log c_1, -h\log d_1; +,-) - \text{logsumexp}(-h\log c_1, -h\log d_1, -h\log c_2, -h\log d_2; +,-,+,-)
\end{aligned}$$

We implement this as follows:

```
> log.sum.exp <- function(x, signs=NA) {
+   # from http://r.789695.n4.nabble.com/logsumexp-function-in-R-td3310119.html
+   # with a modification based on http://jblevins.org/notes/log-sum-exp
+   # and another modification by me
+   if (length(signs) == 1 && is.na(signs))
+     signs <- 0*x+1
+   xmax <- which.max(x)
+   xmin <- which.min(x)
+   if (abs(x[xmin]) > abs(x[xmax]))
+     xmax <- xmin
+   log1p(sum(signs[-xmax]*exp(x[-xmax]-x[xmax])))+x[xmax]
+ }
> inverse.cdf.for.p.t0 <- function(s, intervals, alpha.t0) {
+   h <- alpha.t0 - 1
+   if (intervals$num.intervals == 0)
+     stop("inverse.cdf.for.p.t0: no intervals")
+   else if (intervals$num.intervals == 1) {
+     c1 <- intervals$c1
+     d1 <- intervals$d1
+     log.a <- log(1-s) - h*log(c1)
+     log.b <- log(s) - h*log(d1)
+     exp(-(1/h) * log.sum.exp(c(log.a, log.b)))
+   } else if (intervals$num.intervals == 2) { # XXX this branch is not
+                                              # thoroughly tested; see sidenote
+                                              # below
+     c1 <- intervals$c1
+     d1 <- intervals$d1
+     c2 <- intervals$c2
+     d2 <- intervals$d2
+     log.g <- log.sum.exp(-h*log(c(c1, d1, c2, d2)), c(1, -1, 1, -1))
+     if (log(s) < log.g) {
+       log.a <- log(1-s) - h*log(c1)
+       log.b <- log(s) - h*log(d1)
+       exp(-(1/h) * log.sum.exp(c(log.a, log.b)))
+     } else {
+       log.a <- log(1-s) - h*log(c1)
+       log.b <- log(1-s) - h*log(d1)
+       log.c <- log(1-s) - h*log(c2)
+       log.d <- log(s) - h*log(d2)
+       exp(-(1/h) * log.sum.exp(c(log.a, log.b, log.c, log.d), c(1, -1, 1, 1)))
+     }
+   } else
+     stop("inverse.cdf.for.p.t0: unexpected value")
+ }
```

We doublecheck the above by repeatedly sampling $p \in (0,1)$ and checking that $F^{-1}(F(p)) \approx p$:

```
> (function() {
+   if (run.slow.code) {
+     N <- 10000
+     for (i in 1:N) {
+       # sample alpha_{t_0} and intervals
+       alpha.t0 <- runif(1, 1, 10)
+       #num.intervals <- round(runif(1, 1, 2))
+       num.intervals <- 1 # XXX skip 2 for now, see sidenote below
+       if (num.intervals == 1) {
+         #cd <- sort(runif(2, 0, 1))
+         cd <- c(0.2, 0.5)
+         intervals <- list(num.intervals = 1,
+                           c1 = cd[1],
+                           d1 = cd[2])
+       } else {
+         cd <- sort(runif(4, 0, 1))
+         cd <- c(0.2, 0.5, 0.6, 0.8)
+         intervals <- list(num.intervals = 2,
+                           c1 = cd[1],
+                           d1 = cd[2],
+                           c2 = cd[3],
+                           d2 = cd[4])
+       }
+
+       # sample p and compute p2
+       while (1) {
+         p <- runif(1, 0, 1)
+         if (is.in.interval.for.p.t0(p, intervals))
+           break
+       }
+       F <- cdf.for.p.t0(p, intervals, alpha.t0)
+       p2 <- inverse.cdf.for.p.t0(F, intervals, alpha.t0)
+       #cat(sprintf("num.intervals=%d, p=%f, F=%f, p2=%f\n", num.intervals, p, F, p2))
+       stopifnot(abs(p - p2) < 1e-3)
+     }
+     cat(sprintf("passed %d tests", N))
+   }
+ })()
```

[Sidenote, not yet incorporated above: We claim that only 1 interval is possible, i.e., that neither $b^2 - 4ac < 0$ and $a < 0$ (resulting in 0 intervals), nor $b^2 - 4ac > 0$ and $a > 0$ (resulting in 2 intervals), is possible. We do not have a proof yet, but the following simulation supports our claim.

```
> (function() {
```

```
+   if (run.slow.code) {
+     for (T in c(3, 7, 11)) {
+       N <- 1000
+       df <- ldply(1:N, function(i) {
+         n <- 128
+         G <- 40000
+         alpha <- array(n*G*T, T)
+         alpha[T] <- n*G*(T-1)
+         beta <- runif(T, n*G*200, n*G*300)
+         p <- runif(T, 0, 1)
+         p <- p/sum(p)
+         tilde.v <- sample.tilde.v(T, p, beta)
+         tilde.w <- sample.tilde.w(T, p, alpha)
+         ldply(1:(T-1), function(t0) {
+           intervals <- intervals.for.p.t0(T, t0, p, tilde.v, tilde.w, beta)
+           data.frame(intervals)
+         })
+       })
+       cat(sprintf("for T=%d\n", T))
+       print(ddply(df, .(num.intervals), "nrow"))
+     }
+   }
+ })()
```

End sidenote.]

Finally, we sample $p_{t_0}|\mathbf{p}_{-t_0}, \tilde{v}, \tilde{w}$ using the inverse CDF method:

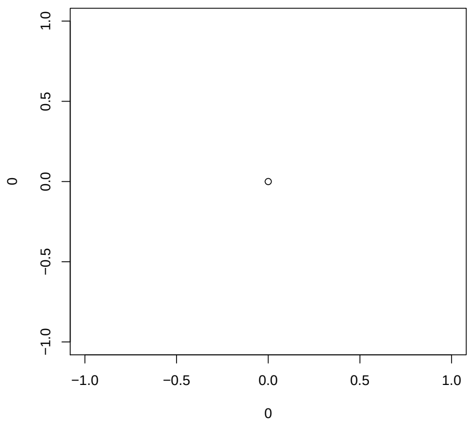
1. Sample $s \sim \text{Uniform}(0,1)$.
2. Set $p_{t_0} = F^{-1}(s)$.

We implement this as follows:

```
> sample.p.t0 <- function(num.samples, T, t0, p, tilde.v, tilde.w, alpha, beta) {
+   s <- runif(num.samples, 0, 1)
+   intervals <- intervals.for.p.t0(T, t0, p, tilde.v, tilde.w, beta)
+   laply(s, function(s0) inverse.cdf.for.p.t0(s0, intervals, alpha[t0]))
+ }
```

We doublecheck this as follows, by comparing the empirical cdf based on samples using our procedure to the empirical cdf based on samples using a brute-force rejection procedure, which was described above.

```
> (function() {
+   if (run.slow.code) {
+     N <- 1000
+     T <- 3
+     t0 <- sample.int(T-1, 1)
+     p <- runif(T, 0, 1)
+     p <- p/sum(p)
+     #alpha <- runif(T, 1, 10)
+     alpha <- array(1.01, T) # close to 1 so the brute force method runs faster
+     beta <- runif(T, 0, 3)
+     tilde.v <- sample.tilde.v(T, p, beta)
+     tilde.w <- sample.tilde.w(T, p, alpha)
+
+     ps <- sample.p.t0(N, T, t0, p, tilde.v, tilde.w, alpha, beta)
+     emp.cdf.1 <- function(y) length(which(ps < y))/N
+
+     intervals <- intervals.for.p.t0(T, t0, p, tilde.v, tilde.w, beta)
+     emp.cdf.2 <- brute.force.empirical.cdf.for.p.t0(intervals, alpha[t0])
+
+     # plot
+     grid <- (1:100)/100
+     df <- rbind(data.frame(x=grid, cdf=laply(grid, emp.cdf.1), type="ours"),
+                 data.frame(x=grid, cdf=laply(grid, emp.cdf.2), type="brute force"))
+     print(ggplot(df, aes(x, cdf, group=type)) + geom_line(aes(color=type)))
+   } else
+     plot(0,0)
+ })()
```



Use this to sample $\mathbf{p}|\mathbf{y},\mathbf{z},\boldsymbol{\pi},\boldsymbol{\alpha},\boldsymbol{\lambda}$

Finally, we use the preceeding to sample $\mathbf{p}|\mathbf{y},\mathbf{z},\boldsymbol{\pi},\boldsymbol{\alpha},\boldsymbol{\lambda}$. (Refer to the beginning of this section for details about the transformations below.)

```
> sample.p.slow <- function(Sigma, G, T, J, y, old.p, z, alpha, lambda, s, sigma, bbT) {
+   laply(1:Sigma, function(sig) {
+     # compute "alpha"
+     c1.sig <- laply(1:T, function(t) {
+       tilde.a <- if (t < T) alpha$a else (alpha$a-1)
+       length(which(sigma == sig)) * G * tilde.a
+     })
+     # compute "beta"
+     c2.sig <- array(0, T)
+     for (i in which(sigma == sig)) {
+       for (g in 1:G) {
+         for (calT.index in 1:length(bbT[[z[g]]])) {
+           calT <- bbT[[z[g]]][[calT.index]]
+           for (t in calT)
+             c2.sig[t] <- c2.sig[t] + lambda[[z[g]]][[calT.index]] * y[i,g,t] * s[i,g]
+         }
+       }
+     }
+     # sample tilde.v and tilde.w
+     tilde.v <- sample.tilde.v(T, old.p[sig,], c2.sig)
+     tilde.w <- sample.tilde.w(T, old.p[sig,], c1.sig)
+     # sample p
+     p.sig <- old.p[sig,]
+     for (t in 1:(T-1))
+       p.sig[t] <- sample.p.t0(1, T, t, p.sig, tilde.v, tilde.w, c1.sig, c2.sig)
+     p.sig[T] <- 1 - sum(p.sig[1:(T-1)])
+     p.sig
+   })
+ }
```

This is a unnecessarily slow, so we do the following instead:

```
> sample.p <- function(n, Sigma, G, T, J, y, old.p, z, alpha, lambda, s, sigma, bbT) {
+   # make a faster datastructure for lambdas
+   l = array(0, c(J, T))
+   for (j in 1:J)
+     for (calT.index in 1:length(bbT[[j]]))
+       for (t in bbT[[j]][[calT.index]])
+         l[j,t] <- lambda[[j]][[calT.index]]
+
+   # compute "alpha"
+   c1 <- array(0, c(Sigma, T))
+   for (sig in 1:Sigma) {
+     for (t in 1:T) {
+       tilde.a <- if (t < T) alpha$a else (alpha$a-1)
+       c1[sig, t] <- length(which(sigma == sig)) * G * tilde.a
+     }
+   }
+
+   # compute "beta"
+   c2 <- array(0, c(Sigma, T))
+   for (i in 1:n) {
+     sig <- sigma[i]
+     for (t in 1:T)
+       c2[sig, t] <- sum(l[z,t] * y[i,,t] * s[i,])
+   }
+
+   p <- array(0, c(Sigma, T))
+   for (sig in 1:Sigma) {
+     # sample tilde.v and tilde.w
+     tilde.v <- sample.tilde.v(T, old.p[sig,], c2[sig,])
+     tilde.w <- sample.tilde.w(T, old.p[sig,], c1[sig,])
+     # sample p
+     p.sig <- old.p[sig,]
+     for (t in 1:(T-1))
+       p.sig[t] <- sample.p.t0(1, T, t, p.sig, tilde.v, tilde.w, c1[sig,], c2[sig,])
+     p.sig[T] <- 1 - sum(p.sig[1:(T-1)])
+     p[sig,] <- p.sig
+   }
+
+   p
+ }
```

We test this against the slower version as follows:

```
> (function() {
+  if (run.slow.code) {
+     ex <- make.example()
+     y <- array(0, c(ex$n, ex$G, ex$T))
+     for (i in 1:ex$n)
+       for (t in 1:ex$T)
+         y[i,,t] <- ex$x[i,,t] * ex$p[ex$sigma[i],t] / ex$s[i,]
+
+     N <- 500
+     slow.p <- laply(1:N, function(k) sample.p.slow(ex$Sigma, ex$G, ex$T, ex$J, y, ex$p, ex$z, ex$alpha, ex$lamb
+     fast.p <- laply(1:N, function(k) sample.p(ex$n, ex$Sigma, ex$G, ex$T, ex$J, y, ex$p, ex$z, ex$alpha, ex$lam
+
+     cat(sprintf("the mean norm between p matrices is: %f\n", mean(laply(1:500, function(k) norm((fast.p-slow.p)
+
+     # slow.df <- a2d(slow.p, c("N", "sig", "t"), "p")
+     # slow.df$label = "slow"
+     # fast.df <- a2d(fast.p, c("N", "sig", "t"), "p")
+     # fast.df$label = "fast"
+     # df <- rbind(slow.df, fast.df)
+     # save(slow.p, fast.p, df, file="tmp.Rda")
+     # print(ggplot(df, aes(x=p, group=label)) + geom_density(aes(color=label)) + facet_grid(sig ~ t))
+   } else {
+     # plot(0,0)
+   }
+ })()
```

```
NULL
```

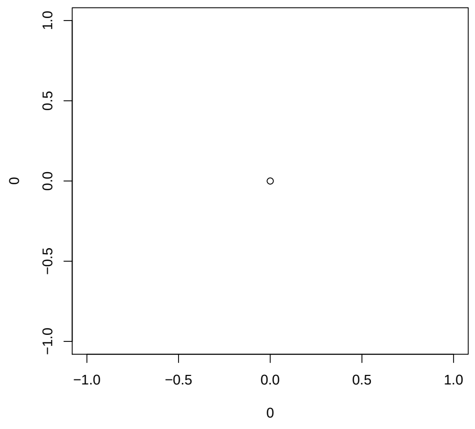
3.3 Sanity check of the overall procedure to sample $\mathbf{x},\mathbf{p}|\mathbf{z},\boldsymbol{\pi},\boldsymbol{\alpha},\boldsymbol{\lambda}$

We repeatedly sample $\mathbf{x}|\mathbf{p}$,others then $\mathbf{p}|\mathbf{x}$,others.

```
> if (run.slow.code || 0) {
+   example.for.sample.xp <- (function() {
+     ex <- make.example()
+     N <- 1000
+     xps <- list()
+     xps[[1]] <- list(x=ex$x, p=ex$p)
+     for (k in 2:N) {
+       xps[[k]] <- sample.x.and.p(ex$n, ex$Sigma, ex$G, ex$T, ex$J,
+         xps[[k-1]]$x, xps[[k-1]]$p, ex$z, ex$alpha, ex$lambda,
+         ex$s, ex$sigma, ex$bbT)
+       save(xps, file="tmp.Rda")
+     }
+     save(ex, N, xps, file="example_for_sample_xp.Rda")
+     list(N=N, xps=xps)
+   })()
+ } else {
+   example.for.sample.xp <- (function() {
+     load("example_for_sample_xp_14454.Rda")
+     list(N=N, xps=xps)
+   })()
+ }
```

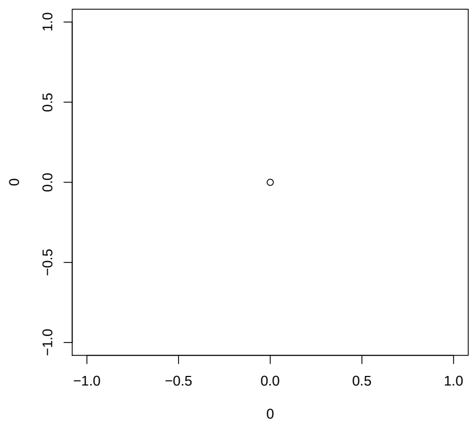
We plot the proportions $\mathbf{p}_\sigma$ for $\sigma = 1$:

```
> (function() {
+   if (run.slow.code) {
+     xps <- example.for.sample.xp$xps
+     sig <- 1
+     df <- ldply(1:3, function(t)
+       data.frame(type=sprintf("sig=%d,t=%d", sig, t),
+                  iter=1:length(xps),
+                  p=laply(xps, function(xp) xp$p[sig,t])))
+     print(ggplot(df, aes(x=iter, y=p)) + geom_point() + facet_wrap(~ type))
+   } else {
+     plot(0,0)
+   }
+ })()
```



We plot the coordinates $\mathbf{x}_{i,g}$ for $i = g = 1$:

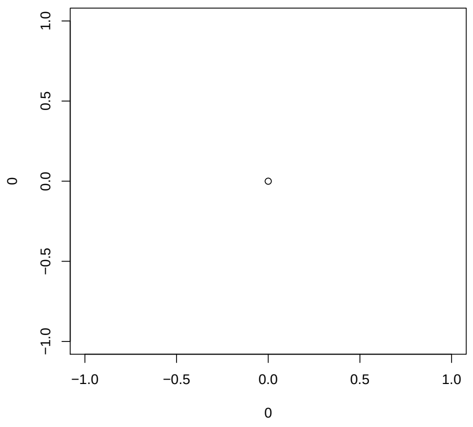
```
> (function() {
+   if (run.slow.code) {
+     xps <- example.for.sample.xp$xps
+     i <- 1
+     g <- 1
+     df <- ldply(1:3, function(t)
+       data.frame(type=sprintf("i=%d,g=%d,t=%d", i, g, t),
+                  iter=1:length(xps),
+                  x=laply(xps, function(xp) xp$x[i,g,t])))
+     print(ggplot(df, aes(x=iter, y=x)) + geom_point() + facet_wrap(~ type))
+   } else {
+     plot(0,0)
+   }
+ })()
```



We plot the matrix norm of $\mathbf{x}_{\cdot,\cdot,t}$.

```
> (function() {
+   if (run.slow.code) {
```

```
+      xps <- example.for.sample.xp$xps
+      df <- ldply(1:3, function(t)
+        data.frame(type=sprintf("t=%d", t),
+                   iter=1:length(xps),
+                   norm=laply(xps, function(xp) norm(xp$x[,,t]))))
+      print(ggplot(df, aes(x=iter, y=norm)) + geom_point() + facet_wrap(~ type))
+    } else {
+      plot(0,0)
+    }
+ })()
```



4 Sample $\lambda|\mathbf{x},\mathbf{p},\mathbf{z},\boldsymbol{\pi},\boldsymbol{\alpha}$

Recall that there are distinct $\lambda_{j,\mathscr{T}}$ for each pattern of differential expression $j = 1,\dots,J$ and each element $\mathscr{T}$ of the partition $\mathbb{T}_j$.

Note that

$$\begin{aligned}
f(\boldsymbol{\lambda}|\mathbf{x},\mathbf{p},\mathbf{z},\boldsymbol{\pi},\boldsymbol{\alpha}) &\propto \prod_{i=1}^{n}\prod_{g=1}^{G}\prod_{\mathscr{T}\in\mathbb{T}_{z_g}} f_{\Gamma(a_0,\nu)}(\lambda_{z_g,\mathscr{T}})\prod_{t\in\mathscr{T}} f_{\Gamma(a,\lambda_{z_g,\mathscr{T}})}(x_{i,g,t})\\
&\propto \prod_{j=1}^{J}\prod_{\mathscr{T}\in\mathbb{T}_j}\prod_{i=1}^{n}\prod_{g:z_g=j}\lambda_{j,\mathscr{T}}^{a_0-1}\exp(-\nu\lambda_{j,\mathscr{T}})\prod_{t\in\mathscr{T}}\lambda_{j,\mathscr{T}}^{a}\exp(-\lambda_{j,\mathscr{T}}x_{i,g,t})\\
&\propto \prod_{j=1}^{J}\prod_{\mathscr{T}\in\mathbb{T}_j} f_{\Gamma(c_{1,j,\mathscr{T}},c_{2,j,\mathscr{T}})}(\lambda_{j,\mathscr{T}})
\end{aligned}$$

where

$$\begin{aligned}
c_{1,j,\mathscr{T}} &= nm_j((a_0-1)+|\mathscr{T}|a)+1\\
c_{2,j,\mathscr{T}} &= nm_j\nu+\sum_{i=1}^{n}\sum_{g:z_g=j}\sum_{t\in\mathscr{T}} x_{i,g,t}\\
m_j &= |\{g : z_g = j\}|
\end{aligned}$$

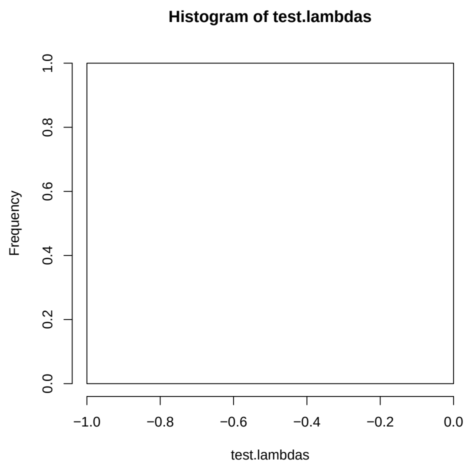
In code:

```
> sample.lambda.gibbs <- function(n, J, x, z, alpha, old.lambda, bbT) {
+   a <- alpha$a
+   a0 <- alpha$a0
+   nu <- alpha$nu
+   lambda <- llply(1:J, function(j) {
+     mj <- length(which(z == j))
+     #if (mj == 0)
+     #  stop(sprintf("no genes follow expression pattern j=%d", j))
+     if (mj == 0) {
+       if (verbose)
+         cat(sprintf("sample.lambda: warning: no genes follow expression pattern j=%d; using old value of lambda
+       old.lambda[[j]]
+     }
+     else {
+       llply(bbT[[j]], function(calT) {
+         c1 <- n*mj*((a0-1) + length(calT)*a) + 1
+         c2 <- n*mj*nu + sum(x[, which(z == j), calT])
+
+         # c1.check <- 0
+         # for (i in 1:n) {
+         #   for (g in which(z == j)) {
+         #     c1.check <- c1.check + a0-1
+         #     for (t in calT)
+         #       c1.check <- c1.check + a
+         #   }
+         # }
+         # c1.check <- c1.check + 1
+         #
+         # c2.check <- 0
+         # for (i in 1:n) {
+         #   for (g in which(z == j)) {
+         #     c2.check <- c2.check + nu
+         #     for (t in calT)
+         #       c2.check <- c2.check + x[i,g,t]
+         #   }
+         # }
+         #
+         # cat(sprintf("c1=%f, c1.check=%f\n", c1, c1.check))
+         # cat(sprintf("c2=%f, c2.check=%f\n", c2, c2.check))
+
+         rgamma(1, shape=c1, rate=c2)
+       })
+     }
+   })
+   lambda
+ }
> sample.lambda.marg <- function(n, J, x, z, alpha, old.lambda, bbT) {
+   a <- alpha$a
+   a0 <- alpha$a0
+   nu <- alpha$nu
+   lambda <- llply(1:J, function(j) {
+     llply(bbT[[j]], function(calT) {
+       rgamma(1, shape=a0, rate=nu)
+     })
+   })
+ }
> sample.lambda <- sample.lambda.marg
```

Test:

```
> test.lambdas <- (function() {
+  if (run.slow.code) {
+     # make an example that is as small as possible -- big hack!
+     redo <- TRUE
+     while (redo) {
+       redo <- FALSE
+       ex <- make.example(n=4,G=50)
+       for (j in 1:ex$J) {
+         matches <- which(ex$z == j)
+         if (length(matches) == 0)
+           redo <- TRUE
+         if (length(matches) > 2) {
+           bad.matches <- matches[3:length(matches)]
+           ex$z <- ex$z[-bad.matches]
+           ex$x <- ex$x[,-bad.matches,]
+           ex$s <- ex$s[,-bad.matches]
+           ex$G <- ex$G - length(matches) + 2
+         }
+       }
+     }
+
+     # simulate
+     N <- 1000
+     lambda <- llply(1:N, function(i)
+       sample.lambda(ex$n, ex$J, ex$x, ex$z, ex$alpha, NaN, ex$bbT))
+
+     # pick a specific lambda_{j,calT} to look at
+     j <- 3
+     calT.index <- 2
+     calT <- ex$bbT[[j]][[calT.index]]
+     l <- laply(1:N, function(i) lambda[[i]][[j]][[calT.index]])
+
+     # pick two intervals A_1 and A_2
+     r <- max(l) - min(l)
+     a.1 <- min(l)
+     b.1 <- min(l) + 0.3*r
+     a.2 <- min(l) + 0.5*r
+     b.2 <- min(l) + 0.6*r
+
+     # compute the empirical probabilities
+     M.1 <- length(intersect(which(l > a.1), which(l < b.1)))
+     M.2 <- length(intersect(which(l > a.2), which(l < b.2)))
+     emp.prob.1 <- M.1/N
+     emp.prob.2 <- M.2/N
+
+     # the density of lambda_{j,calT}
+     f <- function(inputs) {
+       laply(inputs, function(lambda.j.calT) {
+         exp(sum(laply(1:ex$n, function(i) {
+           laply(1:ex$G, function(g) {
+             if (ex$z[g] == j) {
+               (dgamma(lambda.j.calT, shape=ex$alpha$a0,
+                 rate=ex$alpha$nu, log=TRUE)
+               + sum(laply(calT, function(t)
+                   dgamma(ex$x[i,g,t], shape=ex$alpha$a,
+                     rate=lambda.j.calT, log=TRUE))))
+             } else {
+               0
+             }
+           })
+         })))
+       })
+     }
+
+     # compute the true probabilities
+     numer.1 <- integrate(f, a.1, b.1)$value
+     numer.2 <- integrate(f, a.2, b.2)$value
+     denom <- integrate(f, 0, Inf)$value
+     true.prob.1 <- numer.1/denom
+     true.prob.2 <- numer.2/denom
+
+     df <- data.frame(prob.A.1=c(numer.1, true.prob.1, emp.prob.1),
+                      prob.A.2=c(numer.2, true.prob.2, emp.prob.2),
+                      ratio=c(numer.1/numer.2,
+                              true.prob.1/true.prob.2,
+                              emp.prob.1/emp.prob.2))
+     rownames(df) <- c("true numer", "true", "empirical")
+     print(df)
+
+     1
+   } else 0
+ })()
```

```
> hist(test.lambdas)
```



5 Sample $\boldsymbol{\alpha}|\mathbf{x},\mathbf{p},\mathbf{z},\boldsymbol{\pi},\boldsymbol{\lambda}$

We sample each element a, a_0, ν of $\boldsymbol{\alpha}$ separately.

```
> sample.alpha <- function(n, G, T, J, x, z, lambda, alpha.prev, bbT) {
+   a <- sample.a(1, n, G, T, J, x, z, alpha.prev$a, lambda, bbT)
+   a0 <- sample.a0(1, n, G, T, J, z, alpha.prev$a0, lambda, bbT, alpha.prev$nu)
+   nu <- sample.nu(1, n, G, J, z, lambda, a0, bbT)
+   list(a=a, a0=a0, nu=nu)
+ }
```

5.1 Prelude: how to sample from $f(x) \propto k^x/(\Gamma(x))^n$

We will want to sample according to $f(x) \propto k^x/(\Gamma(x))^n$. Here we assume that $x \geq 1$ and $k \geq \exp(2)$. The user whuber at stats.stackexchange.com[1] shows how to sample from $k^x/\Gamma(x)$. We adapt his method to our slightly different problem.

The procedure is as follows:

1. Let $a = 3/2, M = 2$.
2. Let $\log g(u) = \log(u^{a-1} f(u^a)) = u^a \log(k) + (a-1)\log(u) - n \log\Gamma(u^a)$.
3. Find the maximizer $u_* = \arg\max_u \log g(u)$ numerically, starting from $u = k^{1/(na)}$. Note that $g'(u) = (1/u)[au^a(\log(k) - n\psi(u^a)) + a - 1]$ where ψ is the digamma function.

[1] http://stats.stackexchange.com/questions/13630/how-to-sample-from-ca-da-1-gammaa

4. Let $\ell = -(1/2)u_*^{-2}((a-1-na/2)(a-1)-(a-1)-n(a^2u_*^a - a/2))$.
5. Let $h(u) = M\exp(-\ell(u-u_*)^2)$, which is proportional to the density of a normal random variable with mean u_* and standard deviation $\sigma = \sqrt{1/(2\ell)}$.
6. Let $\log\tilde{g}(u) = \log g(u) - \log g(u_*)$.
7. Sample u according to h.
8. Sample v according to $\text{Uniform}(0,1)$.
9. If $\log v < \log\tilde{g}(u) - \log h(u)$, accept u, else repeat the previous two steps.
10. Let $x = u^a$.

The rejection constant M could be made smaller, but $M = 2$ appears very safe, and since this procedure is called only twice per iteration of our overall MCMC algorithm, the loss of efficiency is not important.

Actually, since sometimes we have exceedingly large k, and above $\log k$ is mainly used, we substitute $\tilde{k} = \log k$:

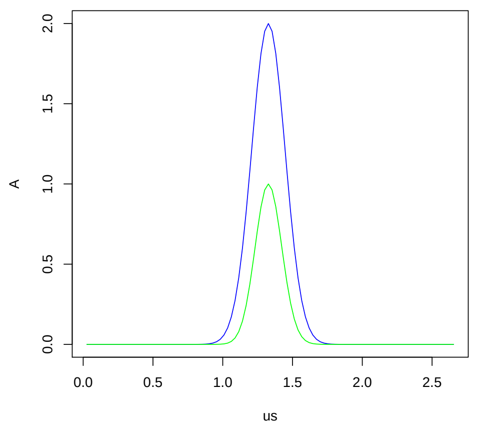
1. Let $a = 3/2$, $M = 2$, $\tilde{k} = \log k$.
2. Let $\log g(u) = \log(u^{a-1}f(u^a)) = u^a\tilde{k} + (a-1)\log(u) - n\log\Gamma(u^a)$.
3. Find the maximizer $u_* = \arg\max_u \log g(u)$ numerically, starting from $u = \exp((\tilde{k})/(na))$. Note that $g'(u) = (1/u)[au^a(\tilde{k} - n\psi(u^a)) + a - 1]$ where ψ is the digamma function.
4. Let $\ell = -(1/2)u_*^{-2}((a-1-na/2)(a-1)-(a-1)-n(a^2u_*^a - a/2))$.
5. Let $h(u) = M\exp(-\ell(u-u_*)^2)$, which is proportional to the density of a normal random variable with mean u_* and standard deviation $\sigma = \sqrt{1/(2\ell)}$.
6. Let $\log\tilde{g}(u) = \log g(u) - \log g(u_*)$.
7. Sample u according to h.
8. Sample v according to $\text{Uniform}(0,1)$.
9. If $\log v < \log\tilde{g}(u) - \log h(u)$, accept u, else repeat the previous two steps.
10. Let $x = u^a$.

We implement the above as follows:

```
> wh.preliminaries <- function(log.k, n) {
+   a <- 3/2
+   M <- 2
+   log.g <- function(u) u^a*log.k +(a-1)*log(u) - n*lgamma(u^a)
+   log.g.deriv <- function(u) (1/u)*(a*u^a*(log.k - n*digamma(u^a)) + a-1)
+   ustar <- nlm(function(u) {
+     res <- -log.g(u)
+     attr(res, "gradient") <- -log.g.deriv(u)
+     res
+   }, exp(log.k/(n*a)))$estimate
+   ell <- -(1/2)*ustar^(-2)*( (a-1-n*a/2)*(a-1) - (a-1) - n*(a^2 * ustar^a - a/2) )
+   h <- function(u) M*exp(-ell*(u-ustar)^2)
+   sigma <- sqrt(1/(2*ell))
+   log.tilde.g <- function(u) log.g(u) - log.g(ustar)
+   list(a=a, M=M, ustar=ustar, sigma=sigma, log.tilde.g=log.tilde.g, h=h)
+ }
> sample.wh <- function(num.samples, log.k, n) {
+   if (log.k < log(5)) {
+     cat(sprintf("sample.wh: warning: received log.k=%f\n", log.k))
+   }
+   p <- wh.preliminaries(log.k, n)
+   laply(1:num.samples, function(iter) {
+     while (1) {
+       u <- rnorm(1, mean=p$ustar, sd=p$sigma)
+       if (u < 1)
+         next
+       v <- runif(1, min=0, max=1)
+       if (log(v) < p$log.tilde.g(u) - log(p$h(u)))
+         break
+       else if (verbose)
+         cat(sprintf("sample.wh: rejected, since %f = log(v) is not < p$log.tilde.g(u) - log(p$h(u)) = %f\n", lo
+     }
+     u^(p$a)
+   })
+ }
> sample.normal.trunc.geq1 <- function(mean, sd) { # XXX stupid way to do this
+   while (1) {
+     x <- rnorm(1, mean=mean, sd=sd)
+     if (x > 1)
+       break
+   }
+   x
+ }
> log.density.normal.trunc.geq1 <- function(x, mean, sd) {
+   dnorm(x, mean, sd, log=TRUE) - pnorm(1, mean, sd, log=TRUE, lower.tail=FALSE)
+ }
> sample.wh.or.mh <- function(num.samples, log.k, n, old.x) {
+   if (log.k < log(5)) {
+     laply(1:num.samples, function(iter) {
+       sd <- 3*5
+       proposed.x <- sample.normal.trunc.geq1(old.x, sd)
+       wh.log.f <- function(x) x*log.k - n*lgamma(x)
+       log.a1 <- wh.log.f(proposed.x) - wh.log.f(old.x)
+       log.a2 <- log.density.normal.trunc.geq1(old.x, proposed.x, sd) - log.density.normal.trunc.geq1(proposed.x
+       a <- exp(log.a1 + log.a2)
+       if (a >= 1)
+         proposed.x
+       else {
+         r = runif(1, 0, 1)
+         if (r < a)
+           proposed.x
+         else
+           old.x
+       }
+     })
+   } else {
+     sample.wh(num.samples, log.k, n)
+   }
+ }
> sample.wh.or.mh.just.mh <- function(num.samples, log.k, n, old.x) {
+     # actually just mh!
+     laply(1:num.samples, function(iter) {
+       sd <- 3*5
+       proposed.x <- sample.normal.trunc.geq1(old.x, sd)
+       wh.log.f <- function(x) x*log.k - n*lgamma(x)
+       log.a1 <- wh.log.f(proposed.x) - wh.log.f(old.x)
+       log.a2 <- log.density.normal.trunc.geq1(old.x, proposed.x, sd) - log.density.normal.trunc.geq1(proposed.x
+       cat(sprintf("log.k=%f, n=%f\n", log.k, n))
+       cat(sprintf("log.a1=%f,log.a2=%f\n", log.a1, log.a2))
+       a <- exp(log.a1 + log.a2)
+       if (a >= 1)
+         proposed.x
+       else {
+         r = runif(1, 0, 1)
+         if (r < a) {
+           cat(sprintf("accepted, because %f<%f\n", r, a));
+           proposed.x
+         } else {
+           cat(sprintf("rejected, because %f>=%f\n", r, a));
+           old.x
+           }
+         }
+       })
+ }
```

I have not typed up the justification for this procedure yet. However, we can get an intuition by looking at a plot:

```
> wh.plot.example <- function(log.k, n) {
+   p <- wh.preliminaries(log.k, n)
+   us <- ((1:100)/100)*2*p$ustar
+   A <- laply(us, p$h)
+   B <- exp(laply(us, p$log.tilde.g))
+   plot(us, A, type="l", col="blue")
+   lines(us, B, col="green")
+ }
> wh.plot.example(log(10), 40)
```



This works for a wide range of k and n.

We test this as follows:

1. Pick $A = (a, b)$.
2. Let $P(X \in A) = [\int_a^b f(x)\,dx]/[\int_0^\infty f(x)\,dx]$, computed numerically (truncating at $2u_*$).
3. Sample $x_1, \ldots, x_N$.
4. Check that $|\{i : x_i \in A\}|/N \approx P(X \in A)$.

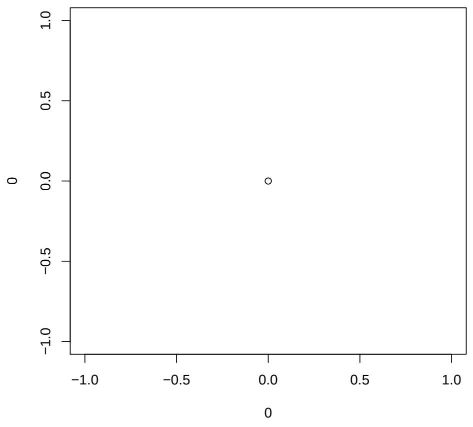
```
> (function() {
+   if (run.slow.code) {
+     #log.k <- log(2)
+     log.k <- log(10)
+     n <- 40
+     a <- 1.2
+     b <- 1.3
+
+     p <- wh.preliminaries(log.k, n)
+     f <- function(x) exp(x*log.k - n*lgamma(x))
+     numer <- integrate(f, a, b)$value
+     denom <- integrate(f, 0, 2*p$ustar)$value
+     true.prob <- numer/denom
+
+     N <- 1000
+     x <- sample.wh(N, log.k, n)
+     M <- length(intersect(which(x > a), which(x < b)))
+     emp.prob <- M/N
+
+     N <- 1000
+     x1 <- x[1]
+     x <- array(0, 1000)
+     x[1] <- x1
+     for (i in 2:N)
+       x[i] <- sample.wh.or.mh(1, log.k, n, x[i-1])
+     M <- length(intersect(which(x > a), which(x < b)))
+     emp.prob.mh <- M/N
+
+     data.frame(true.prob, emp.prob, emp.prob.mh)
+   }
+ })()
```

Different approach to this test:

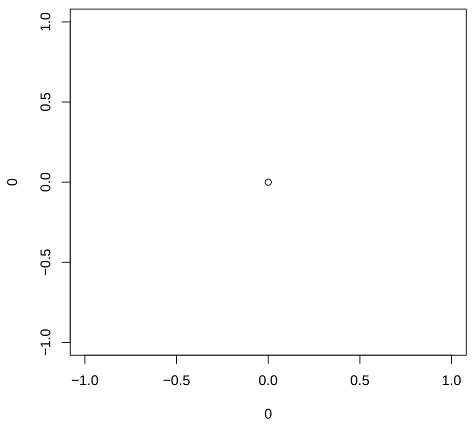
```
> (function() {
+   if (run.slow.code) {
+     log.k <- log(10)
+     n <- 40
+     a <- 1.2
+     b <- 1.3
+     c <- 1.5
+     d <- 1.7
+
+     f <- function(x) exp(x*log.k - n*lgamma(x))
+     true.1 <- integrate(f, a, b)$value
+     true.2 <- integrate(f, c, d)$value
+     true.ratio <- true.1/true.2
+
+     N <- 1000
+     x <- sample.wh(N, log.k, n)
+     M.1 <- length(intersect(which(x > a), which(x < b)))
+     M.2 <- length(intersect(which(x > c), which(x < d)))
+     emp.1 <- M.1/N
+     emp.2 <- M.2/N
+     emp.ratio <- emp.1/emp.2
+
+     df <- data.frame(one=c(true.1, emp.1), two=c(true.2, emp.2), ratio=c(true.ratio, emp.ratio))
+     rownames(df) <- c("true", "emp")
+     df
+   }
+ })()
```

We also test the behavior of this procedure for various values of *k* and *n*.

```
> (function() {
+   if (run.slow.code) {
+     df <- ldply(2:40, function(log.k) data.frame(log.k, n=1:40));
+     df <- adply(df, 1, function(row) {
+       x <- sample.wh(1, row$log.k, row$n);
+       data.frame(row, x=x)
+     })
+     print(ggplot(df, aes(log.k, x, group=n, color=n)) + geom_line() + scale_y_log10())
+   } else
+     plot(0,0)
+ })()
```



```
> (function() {
+   if (run.slow.code) {
+     n <- 128
+     G <- 40000
+     T <- 3
+     df <- ldply((1:40)*10, function(log.k) data.frame(log.k, n=n*G*T));
+     df <- adply(df, 1, function(row) {
+       x <- sample.wh(1, row$log.k, row$n);
+       data.frame(row, x=x)
+     })
+     print(ggplot(df, aes(log.k, x, group=n, color=n)) + geom_line())
+   } else
+     plot(0,0)
+ })()
```



5.2 Sample $a|\mathbf{x},\mathbf{p},\mathbf{z},\boldsymbol{\pi},\boldsymbol{\lambda},a_0,\nu$

Note that

$$\begin{aligned}
f(a|\mathbf{x},\mathbf{p},\mathbf{z},\boldsymbol{\pi},\boldsymbol{\lambda},a_0,\nu) &\propto \prod_{i=1}^{n}\prod_{g=1}^{G}\prod_{\mathscr{T}\in\mathbb{T}_{z_g}}\prod_{t\in\mathscr{T}} f(x_{i,g,t}|a,\lambda_{z_g,\mathscr{T}}) \\
&\propto \prod_{i=1}^{n}\prod_{g=1}^{G}\prod_{\mathscr{T}\in\mathbb{T}_{z_g}}\prod_{t\in\mathscr{T}} \lambda_{z_g,\mathscr{T}}^{a} x_{i,g,t}^{a-1}/\Gamma(a) \\
&\propto c_1^a/(\Gamma(a))^{c_2}
\end{aligned}$$

where

$$\begin{aligned}
c_1 &= \prod_{i=1}^{n}\prod_{g=1}^{G}\prod_{\mathscr{T}\in\mathbb{T}_{z_g}}\prod_{t\in\mathscr{T}} \lambda_{z_g,\mathscr{T}} x_{i,g,t} \\
c_2 &= nGT
\end{aligned}$$

Note that

$$\begin{aligned}
\log c_1 &= \sum_{i=1}^{n}\sum_{g=1}^{G}\sum_{\mathscr{T}\in\mathbb{T}_{z_g}}\sum_{t\in\mathscr{T}} (\log\lambda_{z_g,\mathscr{T}} + \log x_{i,g,t}) \\
&= \Big(\sum_{i=1}^{n}\sum_{g=1}^{G}\sum_{t=1}^{T}\log x_{i,g,t}\Big) + \Big(\sum_{i=1}^{n}\sum_{j=1}^{J}\sum_{g=1}^{G} 1_{\{z_g=j\}} \sum_{\mathscr{T}\in\mathbb{T}_j}\sum_{t\in\mathscr{T}}\log\lambda_{j,\mathscr{T}}\Big) \\
&= \Big(\sum_{i=1}^{n}\sum_{g=1}^{G}\sum_{t=1}^{T}\log x_{i,g,t}\Big) + \Big(n\sum_{j=1}^{J}|\{g:z_g=j\}|\sum_{\mathscr{T}\in\mathbb{T}_j}|\mathscr{T}|\log\lambda_{j,\mathscr{T}}\Big) \\
&= \Big(\sum_{i=1}^{n}\sum_{g=1}^{G}\sum_{t=1}^{T}\log x_{i,g,t}\Big) + \Big(\sum_{j=1}^{J}\sum_{\mathscr{T}\in\mathbb{T}_j} c_{3,j,\mathscr{T}}\log\lambda_{j,\mathscr{T}}\Big)
\end{aligned}$$

where

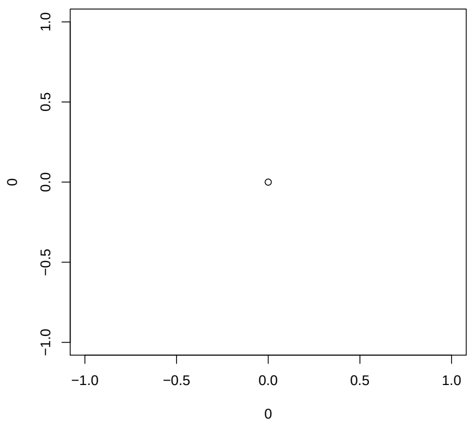
$$c_{3,j,\mathscr{T}} = n|\{g:z_g=j\}||\mathscr{T}|$$

In code:

```
> sample.a <- function(num.samples, n, G, T, J, x, z, old.a, lambda, bbT) {
+   log.c1 <- sum(log(x))
+   for (j in 1:J) {
+     for (calT.index in 1:length(bbT[[j]])) {
+       calT <- bbT[[j]][[calT.index]]
+       c3 <- n * length(which(z == j)) * length(calT)
+       log.c1 <- log.c1 + c3 * log(lambda[[j]][[calT.index]])
+     }
+   }
+   c2 <- n*G*T
+   sample.wh.or.mh(num.samples, log.c1, c2, old.a)
+ }
```

Test:

```
> (function() {
+   if (run.slow.code) {
+     ex <- make.example(n=4, G=2, a=5.5, a0=3, nu=4)
+
+     a.1 <- 5.3
+     b.1 <- 5.4
+     a.2 <- 5.5
+     b.2 <- 5.6
+
+     log.gamma.unnormalized <- function(x, shape, rate) { # considered as a function of shape
+       if (x > 0)
+         shape*log(rate) + (shape - 1)*log(x) - lgamma(shape)
+       else
+         NA
+     }
+
+     # figure out a good normalizing constant
+     trial.as <- c(runif(10, a.1, b.1), runif(10, a.2, b.2))
+     unnormalizeds <- laply(trial.as, function(a) {
+       sum(laply(1:ex$n, function(i) {
+         laply(1:ex$G, function(g) {
+           sum(laply(1:length(ex$bbT[[ex$z[g]]]), function(calT.index) {
+             calT <- ex$bbT[[ex$z[g]]][[calT.index]]
+             sum(laply(calT, function(t) {
+               log.gamma.unnormalized(ex$x[i,g,t], shape=a,
+                 rate=ex$lambda[[ex$z[g]]][[calT.index]])
+             }))
+           }))
+         })
+       }))
+     })
+     normalization <- mean(unnormalizeds)
+
+     f <- function(as) {
+       val.. <- laply(as, function(a) {
+         exp(sum(laply(1:ex$n, function(i) {
+           laply(1:ex$G, function(g) {
+             sum(laply(1:length(ex$bbT[[ex$z[g]]]), function(calT.index) {
+               calT <- ex$bbT[[ex$z[g]]][[calT.index]]
+               sum(laply(calT, function(t) {
+                 log.gamma.unnormalized(ex$x[i,g,t], shape=a,
+                   rate=ex$lambda[[ex$z[g]]][[calT.index]]) - normalization
+               }))
+             }))
+           })
+         })))
+       })
+       val..
+     }
+     numer.1 <- integrate(f, a.1, b.1)$value
+     numer.2 <- integrate(f, a.2, b.2)$value
+     denom <- integrate(f, 1, Inf)$value # note: start at 1 b/c we require a>1
+     true.prob.1 <- numer.1/denom
+     true.prob.2 <- numer.2/denom
+
+     N <- 1000
+     a <- sample.a(N, ex$n, ex$G, ex$T, ex$J, ex$x, ex$z, ex$alpha$a, ex$lambda, ex$bbT)
+     M.1 <- length(intersect(which(a > a.1), which(a < b.1)))
+     M.2 <- length(intersect(which(a > a.2), which(a < b.2)))
+     emp.prob.1 <- M.1/N
+     emp.prob.2 <- M.2/N
+
+     df <- data.frame(prob.A.1=c(true.prob.1, emp.prob.1),
+                      prob.A.2=c(true.prob.2, emp.prob.2),
+                      ratio=c(true.prob.1/true.prob.2,
+                               emp.prob.1/emp.prob.2))
+     rownames(df) <- c("true", "empirical")
+     print(df)
+
+     hist(a)
+   } else
+     plot(0,0)
+ })()
```



The above is close but not perfect. I have more faith in the sampling code than the above evaluation code, though, because of numerical trouble computing the integrals.

5.3 Sample $a_0|\mathbf{x},\mathbf{p},\mathbf{z},\boldsymbol{\pi},\boldsymbol{\lambda},a,\nu$

Note that

$$\begin{aligned}
f(a_0|\mathbf{x},\mathbf{p},\mathbf{z},\boldsymbol{\pi},\boldsymbol{\lambda},a,\nu) &\propto \prod_{i=1}^{n}\prod_{g=1}^{G}\prod_{\mathscr{T}\in\mathbb{T}_{zg}} f(\lambda_{z_g,\mathscr{T}}|a_0,\nu) \\
&\propto \prod_{i=1}^{n}\prod_{g=1}^{G}\prod_{\mathscr{T}\in\mathbb{T}_{zg}} \nu^{a_0}\lambda_{z_g,\mathscr{T}}^{a_0-1}/\Gamma(a_0) \\
&\propto c_1^{a_0}/(\Gamma(a_0))^{c_2}
\end{aligned}$$

where

$$\begin{aligned}
c_1 &= \nu^{c_2}\prod_{i=1}^{n}\prod_{g=1}^{G}\prod_{\mathscr{T}\in\mathbb{T}_{zg}} \lambda_{z_g,\mathscr{T}} \\
c_2 &= n\sum_{g=1}^{G}|\mathbb{T}_{z_g}|
\end{aligned}$$

Note

$$\begin{aligned}\log c_1 &= c_2 \log(\nu) + \sum_{i=1}^{n} \sum_{g=1}^{G} \sum_{\mathscr{T} \in \mathbb{T}_{z_g}} \log \lambda_{z_g,\mathscr{T}} \\ &= c_2 \log(\nu) + \sum_{i=1}^{n} \sum_{j=1}^{J} \sum_{g=1}^{G} 1_{\{z_g = j\}} \sum_{\mathscr{T} \in \mathbb{T}_j} \log \lambda_{j,\mathscr{T}} \\ &= c_2 \log(\nu) + n \sum_{j=1}^{J} |\{z_g = j\}| \sum_{\mathscr{T} \in \mathbb{T}_j} \log \lambda_{j,\mathscr{T}} \\ &= c_2 \log(\nu) + \sum_{j=1}^{J} \sum_{\mathscr{T} \in \mathbb{T}_j} c_{3,j} \log \lambda_{j,\mathscr{T}}\end{aligned}$$

where

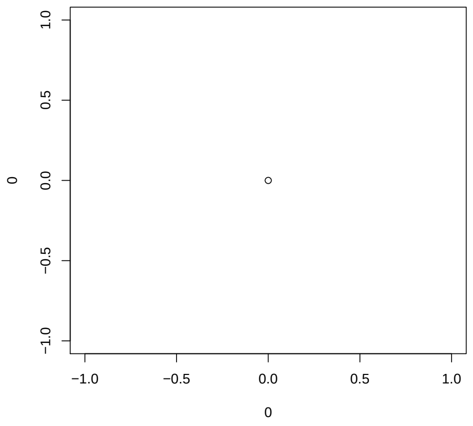
$$c_{3,j} = n|\{g : z_g = j\}|$$

In code:

```
> sample.a0 <- function(num.samples, n, G, T, J, z, old.a0, lambda, bbT, nu) {
+   c2 <- n*sum(laply(1:G, function(g) length(bbT[[z[g]]])))
+   log.c1 <- c2*log(nu)
+   for (j in 1:J) {
+     for (calT.index in 1:length(bbT[[j]])) {
+       c3 <- n * length(which(z == j))
+       log.c1 <- log.c1 + c3 * log(lambda[[j]][[calT.index]])
+     }
+   }
+   sample.wh.or.mh(num.samples, log.c1, c2, old.a0)
+ }
```

Test:

```
> (function() {
+   if (run.slow.code) {
+     ex <- make.example(n=4, G=2, a=5.5, a0=3, nu=4)
+
+     a.1 <- 2.5
+     b.1 <- 2.9
+     a.2 <- 3.0
+     b.2 <- 3.1
+
+     f <- function(a0s) {
+       laply(a0s, function(a0) {
+         exp(sum(laply(1:ex$n, function(i) {
+           laply(1:ex$G, function(g) {
+             sum(laply(1:length(ex$bbT[[ex$z[g]]]), function(calT.index) {
+               dgamma(ex$lambda[[ex$z[g]]][[calT.index]],
+                 shape=a0, rate=ex$alpha$nu, log=TRUE)
+             }))
+           })
+         })))
+       })
+     }
+     numer.1 <- integrate(f, a.1, b.1)$value
+     numer.2 <- integrate(f, a.2, b.2)$value
+     denom <- integrate(f, 1, Inf)$value # note: start at 1 b/c we require a0>1
+     true.prob.1 <- numer.1/denom
+     true.prob.2 <- numer.2/denom
+
+     N <- 1000
+     a0 <- sample.a0(N, ex$n, ex$G, ex$T, ex$J, ex$z, ex$alpha$a0, ex$lambda, ex$bbT, ex$alpha$nu)
+     M.1 <- length(intersect(which(a0 > a.1), which(a0 < b.1)))
+     M.2 <- length(intersect(which(a0 > a.2), which(a0 < b.2)))
+     emp.prob.1 <- M.1/N
+     emp.prob.2 <- M.2/N
+
+     df <- data.frame(prob.A.1=c(true.prob.1, emp.prob.1),
+                      prob.A.2=c(true.prob.2, emp.prob.2),
+                      ratio=c(true.prob.1/true.prob.2,
+                              emp.prob.1/emp.prob.2))
+     rownames(df) <- c("true", "empirical")
+     print(df)
+
+     hist(a0)
+   } else
+     plot(0,0)
+ })()
```



5.4 Sample $\nu|\mathbf{x},\mathbf{p},\mathbf{z},\boldsymbol{\pi},\boldsymbol{\lambda},a,a_0$

Note that

$$\begin{aligned} f(\nu|\mathbf{x},\mathbf{p},\mathbf{z},\boldsymbol{\pi},\boldsymbol{\lambda},a,a_0) &\propto \prod_{i=1}^{n}\prod_{g=1}^{G}\prod_{\mathscr{T}\in\mathbb{T}_{zg}} f(\lambda_{z_g,\mathscr{T}}|a_0,\nu) \\ &\propto \prod_{i=1}^{n}\prod_{g=1}^{G}\prod_{\mathscr{T}\in\mathbb{T}_{zg}} \nu^{a_0}\exp(-\nu\lambda_{z_g,\mathscr{T}}) \\ &\propto \nu^{c_1-1}\exp(-\nu c_2) \end{aligned}$$

where

$$c_1 = n \sum_{g=1}^{G} |\mathbb{T}_{z_g}| a_0 + 1$$

$$c_2 = n \sum_{g=1}^{G} \sum_{\mathscr{T} \in \mathbb{T}_{z_g}} \lambda_{z_g, \mathscr{T}}$$

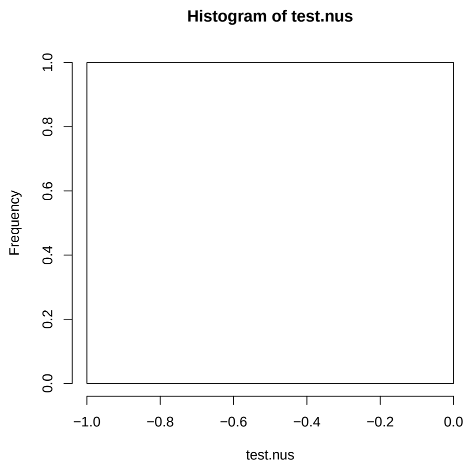
In code:

```
> sample.nu.slow <- function(num.samples, n, G, z, lambda, a0, bbT) {
+   c1 <- n * sum(laply(1:G, function(g) length(bbT[[z[g]]]))) * a0 + 1
+   c2 <- n * sum(laply(1:G, function(g) {
+     sum(laply(1:length(bbT[[z[g]]]), function(calT.index)
+       lambda[[z[g]]][[calT.index]]))
+   }))
+   rgamma(num.samples, shape=c1, rate=c2)
+   # XXX why does this seem to work!?! and not the above
+   #rgamma(num.samples, shape=c1, rate=c1/c2)
+ }
```

Test:

```
> test.nus <- (function() {
+   if (run.slow.code) {
+     ex <- make.example(n=4, G=2, a=5.5, a0=3, nu=4)
+
+     N <- 1000
+     nu <- sample.nu.slow(N, ex$n, ex$G, ex$z, ex$lambda, ex$alpha$a0, ex$bbT)
+
+     r <- max(nu) - min(nu)
+     a.1 <- min(nu)
+     b.1 <- min(nu) + 0.3*r
+     a.2 <- min(nu) + 0.5*r
+     b.2 <- min(nu) + 0.6*r
+
+     M.1 <- length(intersect(which(nu > a.1), which(nu < b.1)))
+     M.2 <- length(intersect(which(nu > a.2), which(nu < b.2)))
+     emp.prob.1 <- M.1/N
+     emp.prob.2 <- M.2/N
+
+     f <- function(nus) {
+       laply(nus, function(nu) {
+         exp(sum(laply(1:ex$n, function(i) {
+           laply(1:ex$G, function(g) {
+             sum(laply(1:length(ex$bbT[[ex$z[g]]]), function(calT.index) {
+               dgamma(ex$lambda[[ex$z[g]]][[calT.index]],
+                 shape=ex$alpha$a0, rate=nu, log=TRUE)
+             }))
+           })
+         })))
+       })
+     }
+     numer.1 <- integrate(f, a.1, b.1)$value
+     numer.2 <- integrate(f, a.2, b.2)$value
+     denom <- integrate(f, 1, Inf)$value
+     true.prob.1 <- numer.1/denom
+     true.prob.2 <- numer.2/denom
+
+     df <- data.frame(prob.A.1=c(true.prob.1, emp.prob.1),
+                       prob.A.2=c(true.prob.2, emp.prob.2),
+                       ratio=c(true.prob.1/true.prob.2,
+                               emp.prob.1/emp.prob.2))
+     rownames(df) <- c("true", "empirical")
+     print(df)
+
+     nu
+   } else 0
+ })()
```

```
> hist(test.nus)
```

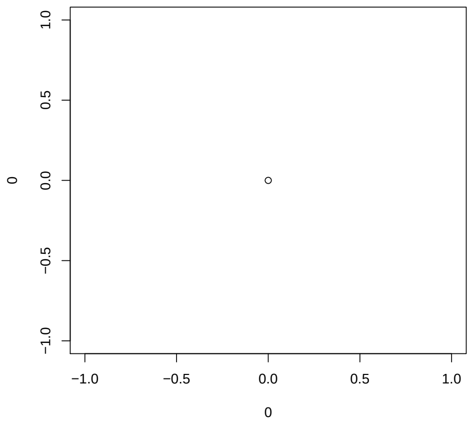


This is very slow, so we rewrite it:

```
> sample.nu <- function(num.samples, n, G, J, z, lambda, a0, bbT) {
+   l1 <- array(0, J)
+   for (j in 1:J)
+     l1[j] <- length(bbT[[j]])
+   c1 <- n * sum(l1[z]) * a0 + 1
+
+   l2 <- array(0, J)
+   for (j in 1:J) {
+     for (calT.index in 1:(l1[j]))
+       l2[j] <- l2[j] + lambda[[j]][[calT.index]]
+   }
+   c2 <- n * sum(l2[z])
+   rgamma(num.samples, shape=c1, rate=c2)
+ }
```

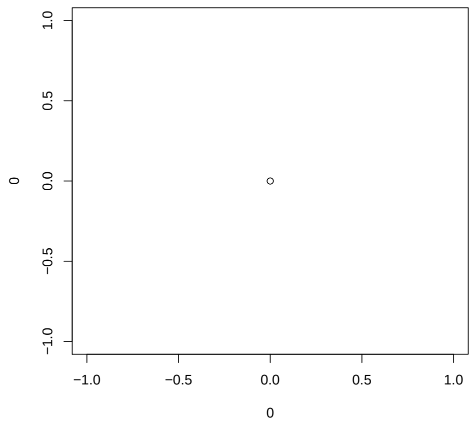
We test this against the slow version as follows:

```
> (function() {
+   if (run.slow.code) {
+     ex <- make.example(n=4, G=2, a=5.5, a0=3, nu=4)
+     N <- 1000
+     slow.nu <- sample.nu.slow(N, ex$n, ex$G, ex$z, ex$lambda, ex$alpha$a0, ex$bbT)
+     fast.nu <- sample.nu(N, ex$n, ex$G, ex$J, ex$z, ex$lambda, ex$alpha$a0, ex$bbT)
+     df <- data.frame(nu=c(slow.nu, fast.nu), label=c(rep("slow", N), rep("fast", N)))
+     print(ggplot(df, aes(x=nu, group=label)) + geom_density(aes(color=label)))
+   } else
+     plot(0,0)
+ })()
```



5.5 Sanity check of $a, a_0, \nu|$others

```
> (function() {
+   if (run.slow.code) {
+     ex <- make.example(n=4, G=2, a=5.5, a0=10, nu=10)
+     N <- 1000
+     lambda <- list(ex$lambda)
+     a  <- list(ex$alpha$a)
+     a0 <- list(ex$alpha$a0)
+     nu <- list(ex$alpha$nu)
+     for (iter in 2:N) {
+       lambda[[iter]] <- sample.lambda(ex$n, ex$J, ex$x, ex$z, list(a=a[[iter-1]], a0=a0[[iter-1]], nu=nu[[iter-
+       a[[iter]] <- sample.a(1, ex$n, ex$G, ex$T, ex$J, ex$x, ex$z, a[[iter-1]], lambda[[iter]], ex$bbT)
+       a0[[iter]] <- sample.a0(1, ex$n, ex$G, ex$T, ex$J, ex$z, a0[[iter-1]], lambda[[iter]], ex$bbT, nu[[iter-1
+       nu[[iter]] <- sample.nu(1, ex$n, ex$G, ex$J, ex$z, lambda[[iter]], a0[[iter]], ex$bbT)
+     }
+     df <- data.frame(
+             iter=rep(1:N, 3),
+             value=c(as.numeric(a), as.numeric(a0), as.numeric(nu)),
+             variable=c(rep("a", N), rep("a0", N), rep("nu", N)))
+     print(ggplot(df, aes(iter, value, group=variable)) + geom_point(aes(color=variable)) + facet_grid(variable
+   } else
+     plot(0,0)
+ })()
```



6 Sample $\mathbf{z}|\mathbf{x}, \mathbf{p}, \boldsymbol{\lambda}, \boldsymbol{\pi}, \boldsymbol{\alpha}$

From the hierarchical model's structure, we see that

$$f(\mathbf{z}|\mathbf{x}, \mathbf{p}, \boldsymbol{\lambda}, \boldsymbol{\pi}, \boldsymbol{\alpha}) \propto f(\mathbf{z}|\boldsymbol{\pi}) \propto \prod_{g=1}^{G} \pi_{z_g}$$

So we do:

```
> sample.z <- function(G, J, pi) {
+   sample(1:J, size=G, prob=pi, replace=TRUE)
+ }
```

7 Sample $\boldsymbol{\pi}|\mathbf{x}, \mathbf{z}, \mathbf{p}, \boldsymbol{\lambda}, \boldsymbol{\pi}, \boldsymbol{\alpha}$

From the hierarchical model's structure, we see that

$$\begin{aligned} f(\boldsymbol{\pi}|\mathbf{x}, \mathbf{p}, \boldsymbol{\lambda}, \boldsymbol{\pi}, \boldsymbol{\alpha}) &\propto 1_{\{\boldsymbol{\pi}:\sum_{j=1}^{J} \pi_j = 1\}} f(\mathbf{z}|\boldsymbol{\pi}) \\ &\propto 1_{\{\boldsymbol{\pi}:\sum_{j=1}^{J} \pi_j = 1\}} \prod_{g=1}^{G} \pi_{z_g} \\ &= 1_{\{\boldsymbol{\pi}:\sum_{j=1}^{J} \pi_j = 1\}} \prod_{j=1}^{J} \pi_j^{m_j}, \qquad m_j = |\{g : z_g = j\}| \end{aligned}$$

the density of a Dirichlet$(m_1 + 1, \ldots, m_J + 1)$ random variable.

However, this will lead to problems (density not being nonzero almost everywhere) if we ever end up with no genes following a particular expression pattern. So we add a pseudocount:

$$f(\boldsymbol{\pi}|\mathbf{x}, \mathbf{p}, \boldsymbol{\lambda}, \boldsymbol{\pi}, \boldsymbol{\alpha}) = 1_{\{\boldsymbol{\pi}:\sum_{j=1}^{J} \pi_j = 1\}} \prod_{j=1}^{J} \pi_j^{m_j}, \qquad m_j = |\{g : z_g = j\}| + 1$$

the density of a Dirichlet$(m_1 + 2, \ldots, m_J + 2)$ random variable.

So our code is:

```
> sample.pi <- function(J, z) {
+   m.plus.2 <- laply(1:J, function(j) length(which(z == j))+2)
+   rdirichlet(1, m.plus.2)
+ }
```

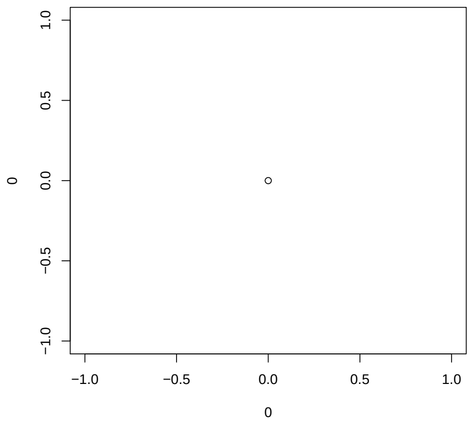
8 Test of the overall Gibbs sampling procedure

Needless to say, the test in this section is only preliminary. A much more thorough investigation remains to be done. (It seems that something is wrong with the a, a_0, ν parameters?)

```
> overall.test.results.of.gibbs <- (function() {
+   if (0) {
+     num.iters <- 500
+     ex <- make.example(n=50, G=5, a=10, a0=2, nu=0.5)
+ #make.example <- function(n=8, G=5, a=100.0, a0=50.0, nu=30.0) {
+     results <- gibbs(num.iters, ex$n, ex$Sigma, ex$G, ex$T, ex$J, ex$x, ex$p, ex$z, ex$pi, ex$alpha, ex$lambda,
+     save(ex, results, file="gibbs_results.Rda")
+     list(ex=ex, results=results)
+   }
+ })()
```

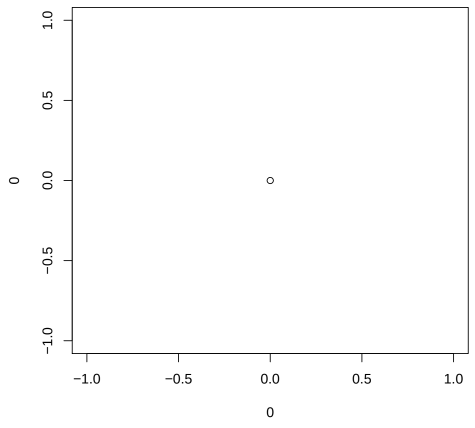
First we look at a.

```
> (function() {
+   if (run.slow.code) {
+     load("gibbs_results.Rda")
+     cat(sprintf("Truth: a=%f\n", ex$alpha$a))
+     print(qplot(1:length(results$alpha), laply(results$alpha, function(alph) alph$a)))
+   } else
+     plot(0,0)
+ })()
```



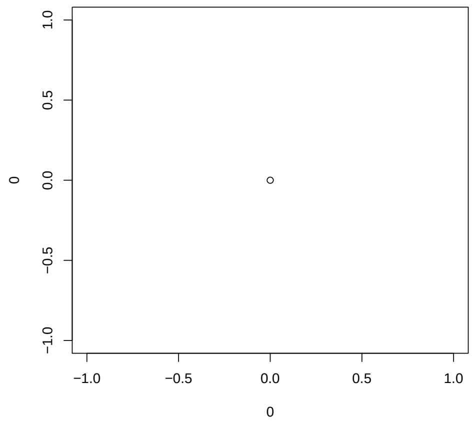
Next we look at a_0.

```
> (function() {
+   if (run.slow.code) {
+     load("gibbs_results.Rda")
+     cat(sprintf("Truth: a0=%f\n", ex$alpha$a0))
+     print(qplot(1:length(results$alpha), laply(results$alpha, function(alph) alph$a0)))
+   } else
+     plot(0,0)
+ })()
```



Next we look at ν.

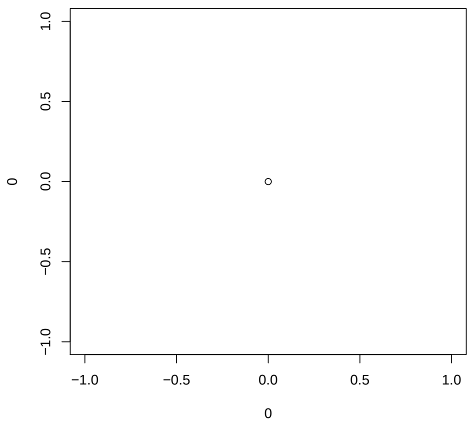
```
> (function() {
+   if (run.slow.code) {
+     load("gibbs_results.Rda")
+     cat(sprintf("Truth: nu=%f\n", ex$alpha$nu))
+     print(qplot(1:length(results$alpha), laply(results$alpha, function(alph) alph$nu)))
+   } else
+     plot(0,0)
+ })()
```



Next we look at p.

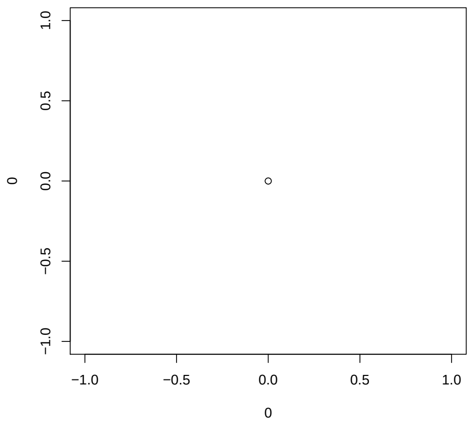
```
> (function() {
+   if (run.slow.code) {
```

```
+     load("gibbs_results.Rda")
+     cat(sprintf("Truth: p=")); print(ex$p)
+     df <- ldply(1:4, function(sig)
+       ldply(1:3, function(t)
+         data.frame(sig=sig,
+                    t=t,
+                    iter=1:length(results$p),
+                    p=laply(results$p, function(p) p[sig,t]))))
+     print(ggplot(df, aes(x=iter, y=p, shape=t)) + geom_point(aes(color=t)) + facet_grid(sig~.))
+   } else
+     plot(0,0)
+ })()
```



Next we look at π.

```
> (function() {
+   if (run.slow.code) {
+     load("gibbs_results.Rda")
+     cat(sprintf("Truth: pi=")); print(ex$pi)
+     df <- ldply(1:ex$J, function(j)
+       data.frame(j=j,
+                  iter=1:length(results$pi),
+                  pi=laply(results$pi, function(pi) pi[j])))
+     print(ggplot(df, aes(x=iter, y=pi)) + geom_point() + facet_grid(j~.))
+   } else
+     plot(0,0)
+ })()
```


1.0
0.5
0.0
−0.5
−1.0
0
−1.0
−0.5
0.0
0.5
1.0
0