Methods to generate random sample from gamma distribution

zishan muzeeb[1]

[1]Indian Institute of Technology Kanpur (IITK)

1 Introduction

I was needed to provide different methods to generate samples from gamma distribution with different shape(α) parameters and scale(λ) parameter equal to one. There are few algorithms with the condition of $0 < \alpha < 1$. I needed to generate random variables from $\alpha > 1$, hence I have generated random variables for the integer part of α using inverse of exponential distribution and for the fractional part using different algorithms based on Acceptance-Rejection method. There are algorithms with condition of $\alpha > 1$, in those case I have directly implemented the algorithms to get random variables. I have then used the obtained variables to plot density graph as it gave better information about the distribution than histograms and compared it with actual graph obtained using standard library in R. I have observed and reported the running time of the algorithms and the amount of rejection in generating random variables using different algorithms for different α. I have presented the graphs that I found gave the most accurate fit with better efficiency.

2 Methods

2.1 Algorithm GT(*two-part method)*(Ahrens and Dieter, 1974a):

This method divides α into its integer(m) and fractional(f) part. The integer part is generated by adding up random variables generated from exponential distribution, m times and then adding up it with random variable generated for the fractional part(f) using ***Algorithm GS*** for $0 \leq \alpha \leq 1$. ***Algorithm GS*** uses acceptance-rejection method with the following majorization function.

$$t(x;\alpha) = \begin{cases} \frac{x^{\alpha-1}}{\Gamma(\alpha)} & \text{if } 0 < x < 1 \\ \\ \frac{e^{-x}}{\Gamma(\alpha)} & \text{if } x > 1 \end{cases}$$

which makes $c = \frac{e+\alpha}{e\Gamma(\alpha+1)}$.

Algo$(0 < \alpha \leq 1)$

1. Generate U. Set $b = \frac{(e+a)}{e}$ and $p = bu$. If $p > 1$ go to 3.
2. (Case $x \leq 1$). Set $x = p^{\frac{1}{\alpha}}$. Generate v. If $v > e^{-x}$ go to 1(rejection) or deliver x.
3. (Case $x > 1$). Set $x = -\ln\left(\frac{(b-p)}{a}\right)$. Generate v. If $v > x^{\alpha-1}$ go to 1(rejection), otherwise deliver x.

2.2 Algorithm RGS(Best, 1983)

Best modified Ahrens algorithm and proposed that the function should not change at unity but a better approximation of majorization function can be given by switching the function at d, where d should be a function of fractional part of shape parameter α i.e.

$$t(x;\alpha) = \begin{cases} \frac{x^{\alpha-1}}{\Gamma(\alpha)} & \text{if } 0 < x < d \\ \\ \frac{e^{-x}}{\Gamma(\alpha)} & \text{if } x > d \end{cases}$$

where d is chosen such that $c = \int_0^\infty t(x;\alpha)dx$ should be minimum, which boils down to the approximation $d = 0.07 + 0.75\sqrt{1-\alpha}$.

Algo$(0 < \alpha < 1)$

1. Initialize $z = 0.07 + 0.75\,(1-\alpha)^{\frac{1}{2}}$ and $b = 1 + \frac{e^{-z}\alpha}{z}$.
2. Generate U and set $p = bU$. If $p > 1$, go to 5.
3. Set $x = zp^{\frac{1}{\alpha}}$. Generate v. if $v \leq \frac{(2-x)}{2+x}$, deliver x.
4. If $v > e^{-x}$ go to 2, otherwise deliver x.
5. Set $x = -\ln\left(\frac{z(b-p)}{a}\right)$, $y = \frac{x}{z}$. Generate v. If $v\,(\alpha + y - \alpha y) < 1$, deliver x.
6. If $v > y^{\alpha-1}$ go to 2, otherwise deliver x.

2.3 Algorithm GC(Ahrens and Dieter, 1974b)

Ahrens proposed another algorithm for $\alpha > 1$. He solved the problem of large computation time for large α. He discovered that the function $g\,(x)$ is proportional to a cauchy density whose integral is an arctan. Inverting the distribution function yeilded the following equation.

$$x = \alpha - 1 + (2\alpha - 1)^{1/2} tan(\pi(u - 0.5))$$

Algo$(\alpha > 1)$

1. Set $b = \alpha - 1$, $a = \alpha + b$ and $s = a^{\frac{1}{2}}$.
2. Generate U. Set $t = s\ \tan\left(\pi\,(U - 0.5)\right)$ and $x = b + t$.
3. If $x < 0$ go to 2.
4. Generate v. If $v > e^{\left(b\ \ln\left(\frac{x}{b}\right) - t + \ln\left(1 + t^{\frac{2}{a}}\right)\right)}$ go to 2(rejection), Otherwise deliver x.

2.4 Algorithm GN(Ahrens and Dieter, 1974a)

In order to reduce the number of rejection Dieter. He proposed a highly efficient algorithm combining a normal envelope $g\,(x)$ around the bulk of gamma distribution with an exponential cover $h\,(x)$ above the tail.

$$g(x) = e^{-\frac{(x-(\alpha-1))^2}{(2(\alpha+c\alpha^{\frac{1}{2}}))}}$$
$$h(x) = \frac{b}{\alpha-1}^{\alpha-1} e^{-(1-(\frac{a-1}{b})x)}$$

where $c = \left(\frac{8}{3}\right)^{\frac{1}{2}}$ and $b = \alpha - 1 + \left(\frac{3}{2}\right) c \left(\alpha + c\alpha^{\frac{1}{2}}\right)^{\frac{1}{2}}$.

In the following majorizing function $g\,(x)$ is in $(-\infty,\ b)$ and $h\,(x) + g\,(x)$ *in* $[b, \infty)$. However, all x from $g\,(x)$ are discarded if they are outside $[0, b]$.

Algo$(\alpha > 1)$

1. Set$\mu = \alpha - 1$, $\sigma = \left(\alpha + 1.63299316185545\alpha^{\frac{1}{2}}\right)^{\frac{1}{2}}$, $d = 2.44948974278318\sigma$ and $b = \mu + d$.(The constants are$(8/3)^{1/2}$ and $6^{1/2}$)
2. Generate U. If $U \leq 0.009572265238289$ go to 5.(The constant is β)
3. Take a sample s from the standard normal distribution and set $x = \mu + \sigma s$. If $x < 0$ *or* $x > b$ go to 2.
4. Generate v. If $\ln(u) > \mu\left(1 + \ln\left(\frac{x}{\mu}\right)\right) - x - \frac{s^2}{2}$ go to 2, otherwise deliver x.
5. Take samples from s from the standard exponential distribution and set $x = b\left(1 + \frac{s}{d}\right)$.
6. Generate v. If $\ln(u) > \mu\left(2 + \ln\left(\frac{x}{\mu}\right) - \frac{x}{b}\right) + 3.7203284924588 - b - \ln\left(\frac{\sigma d}{b}\right)$ go to 2, otherwise deliver x.(The constant is $-\ln\left(\frac{(2\pi)^{\frac{1}{2}}\beta}{1-\beta}\right)$)

2.5 Algorithm 1(Kundu and Gupta, 2007a)

This algorithm uses generalized exponential distribution($f_{GE}(x;\alpha,\lambda)$) to generate gamma random number($f_{GA}(x;\alpha)$). It proposes,

$$f_{GA}(x;a) \leq \frac{2^{\alpha}}{\Gamma(\alpha+1)} f_{GE}(x;\alpha;\tfrac{1}{2})$$

Algo$(0 < \alpha < 1)$

1. Generate U.
2. Compute $x = -2\ln\left(1 - U^{\frac{1}{\alpha}}\right)$
3. Generate V from uniform(0,1), independent of U.
4. If $V \leq \frac{x^{\alpha-1}e^{-\frac{x}{2}}}{2^{\alpha-1}\left(1-e^{-\frac{x}{2}}\right)^{\alpha-1}}$ accept x, otherwise go to 1.

2.6 Algorithm 2(Kundu and Gupta, 2007b)

The above algorithm was true for all $x > 0$ but for $1 < x < \infty$ it was not quite sharp. Hence this algorithm uses the following majorization function

$$t(x;\alpha) = \begin{cases} \frac{2^{\alpha}}{\Gamma(\alpha+1)} f_{GE}(x;\alpha,\frac{1}{2}) & \text{if } 0 < x < 1 \\ \\ \frac{e^{-x}}{\Gamma(\alpha+1)} & \text{if } x > 1 \end{cases}$$

and $c = \frac{1}{\Gamma(\alpha+1)}\left[2^{\alpha}\left(1 - e^{-\frac{1}{2}}\right)^{\alpha} + \alpha e^{-1}\right]$.

Algo$(0 < \alpha < 1)$

1. Set $a = \frac{\left(1-e^{-\frac{1}{2}}\right)^{\alpha}}{\left(1-e^{-\frac{1}{2}}\right)^{\alpha} + \frac{\alpha e^{-1}}{2^{\alpha}}}$ and $b = \left(1 - e^{-\frac{1}{2}}\right)^{\alpha} + \frac{\alpha e^{-1}}{2^{\alpha}}$.
2. Generate U from uniform(0,1).
3. If $U \leq a$, then $x = -2\ln\left[1 - (Ub)^{\frac{1}{\alpha}}\right]$, otherwise $x = -\ln\left[\frac{2^{\alpha}}{\alpha} b(1-U)\right]$.
4. Generate V from uniform(0,1). If $x \leq 1$, *and* $V \leq \frac{x^{\alpha-1}e^{-\frac{x}{2}}}{2^{\alpha-1}\left(1-e^{-\frac{x}{2}}\right)^{\alpha-1}}$ return x or go back to 2. If $x > 1$, and if $V \leq x^{\alpha-1}$ return x or go back to 2.

2.7 Algorithm 3(Kundu and Gupta, 2007b)

The above algorithm has two pieces of envelop with the change point being one, but as suggested by 2.2 the change might depend on α. Hence after using the idea of Best the majorization function becomes

$$t(x;\alpha) = \begin{cases} \frac{2^\alpha}{\Gamma(\alpha+1)} f_{GE}(x;\alpha,\frac{1}{2}) & \text{if } 0 < x < d_\alpha \\ \\ \frac{e^{-x}}{\Gamma(\alpha+1)} & \text{if } x > d_\alpha \end{cases}$$

and $c = \frac{1}{\Gamma(\alpha+1)}\left[2^\alpha\left(1-e^{-\frac{d_\alpha}{2}}\right)^\alpha + \alpha d_\alpha^{\alpha-1}e^{-d_\alpha}\right]$, where d_α was to be chosen s.t. c is minimized. d_α was approximated and was found to be

$$d = 1.0334 - 0.0766e^{2.2942\alpha}$$

Algo$(0 < \alpha < 1)$

1. Set $d = 1.0334 - 0.0766e^{2.2942\alpha}$, $a = 2^\alpha\left(1-e^{-\frac{d}{2}}\right)^\alpha$, $b = \alpha d^{\alpha-1}e^{-d}$ *and* $c = a + b$
2. Generate U from uniform(0,1).
3. If $U \leq \frac{a}{a+b}$, then $x = -2\ln\left[1-(cU)^{\frac{1}{\alpha}}\right]$, otherwise $x = -\ln\left[\frac{c(1-U)}{\alpha d^{\alpha-1}}\right]$.
4. Generate $V$from uniform(0,1). If $x \leq d$ *and* $V \leq \frac{x^{\alpha-1}e^{-\frac{x}{2}}}{2^{\alpha-1}\left(1-e^{-\frac{x}{2}}\right)^{\alpha-1}}$, then return x, otherwise go back to 2.
   2. If $x > d$ *and* $v \leq \left(\frac{d}{x}\right)^{1-\alpha}$, then retun x or go back to step 2.

3 Result

Refer to Table 1(3) for proportion of rejection and time elapsed for different algorithms.

By simulation, I observed that the efficient and best fit of the generated random variable is given by Algorithm 1(2.5). The shaded part is the density obtained from the random number generated and the curve in black covers original gamma distribution for that shape parameter. The vertical lines represent mean and median of the density obtained.

Algorithm	Shape parameter	Rejection	Time
2.1	1.5	0.239	0.11
	3.2	0.149	0.11
	100.7	0.265	0.11
2.2	1.5	0.23	0.18
	3.2	0.131	0.18
	100.7	0.206	0.19
2.3	1.5	0.44	0.08
	3.2	0.241	0.06
	100.7	0.044	0.06
2.4	1.5	0.654	0.13
	3.2	0.533	0.13
	100.7	0.421	0.14
2.5	1.5	0.359	0.08
	3.2	0.189	0.08
	100.7	0.398	0.08
2.6	1.5	0.386	0.31
	3.2	0.593	0.28
	100.7	0.163	0.41
2.7	1.5	0.348	0.14
	3.2	0.515	0.13
	100.7	0.417	0.12

Table 1: Rejection proportion and time to generate samples for different algorithms.

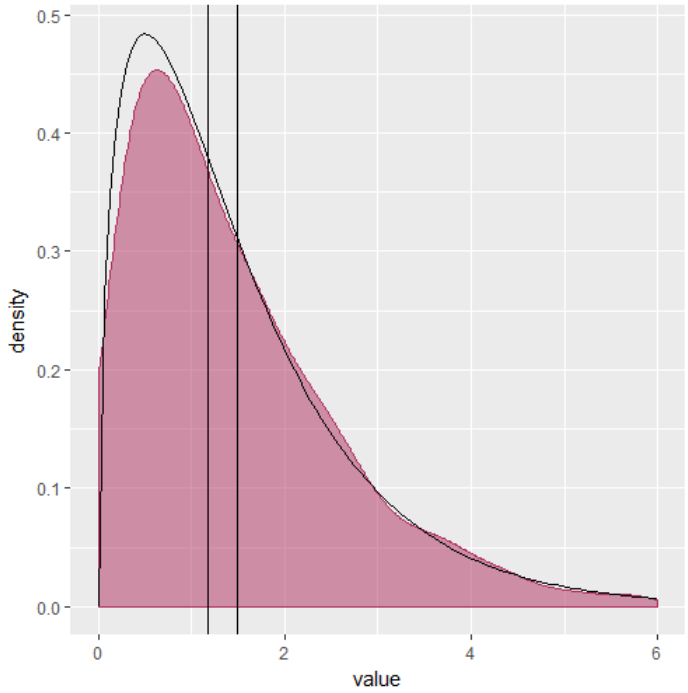

Figure 1: Density graph of generated random variable using 2.5 for $\alpha = 1.5$

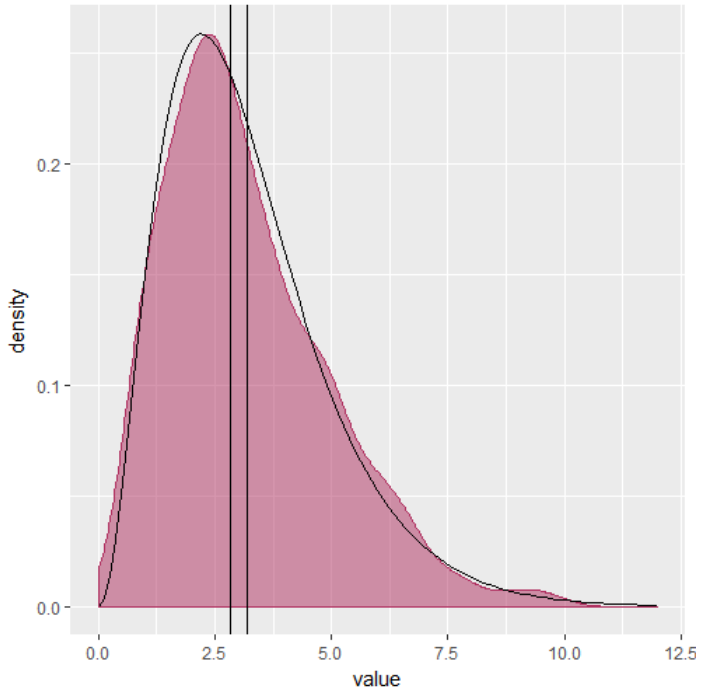

Figure 2: Density graph of generated random variable using 2.5 for $\alpha = 3.2$

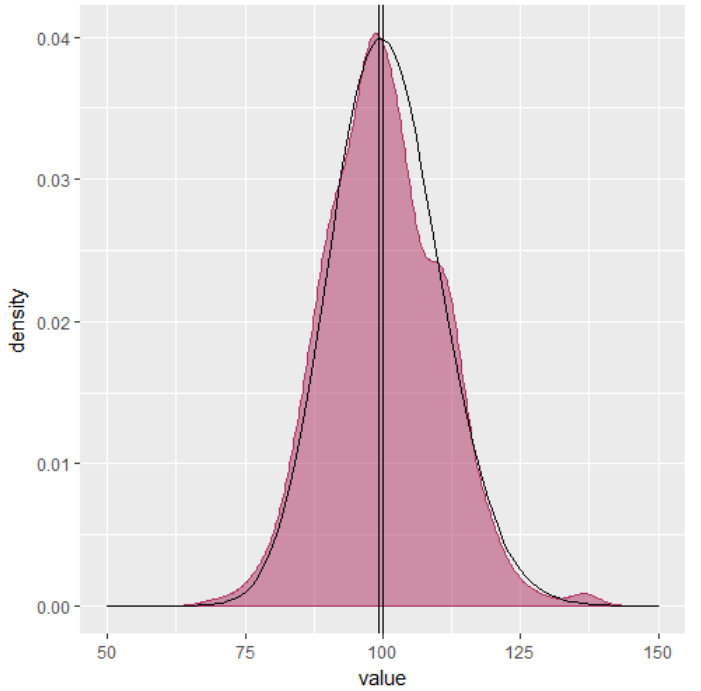

Figure 3: Density graph of generated random variable using 2.5 for $\alpha = 100.7$

4 Appendix

Codes for different algorithms can be found below.

2.1 Algorithm GT

```
time = proc.time()
out = vector()
alf = 0.7
for (val in 1:1000){
u = runif(1)
b = (exp(1)+alf)/exp(1)
p = b*u
x = p^(1/alf)
v = runif(1)
x = p^(1/alf)
if (x<=1){
  if (v > exp(-x)){
    next
  } else {
    p = runif(100)
    p = prod(p)
    q = -log(p)
    out = c(out,x+q)
  }
} else {
  x = -log((b-p)/alf)
  if (v>x^(alf-1)){
    next
  } else {
    p = runif(100)
    p = prod(p)
    q = -log(p)
    out = c(out,x+q)
  }
}
}
time2 = proc.time()-time
out
length(out)
h = melt(out)
ggplot(data= h,aes(x=value))+geom_density(color = "maroon", fill = "maroon", alpha = 0.5)+xlim(50,150)+
  geom_vline(xintercept = mean(out), color = "blue")+geom_vline(xintercept = median(out), color = "gree
  stat_function(fun=dgamma, args=list(shape=100.7, rate=1), color = "black")
```

2.2 Algorithm RGS

```
out = vector()
alf = 0.2
time = proc.time()
z = 0.07+0.75*(1-alf)^0.5
b = 1+exp(-z)*alf/z
for (i in 1:1000){
```

```
u = runif(1)
p = b*u
  if (p>1){
    x = -log(z*(b-p)/alf)
    y = x/z
    v = runif(1)
    if (v*(alf+y-alf*y)<1){
      p = runif(3)
      p = prod(p)
      q = -log(p)
      out = c(out,x+q)
    } else if(v > y^(alf-1)){
      next
    } else {
      p = runif(3)
      p = prod(p)
      q = -log(p)
      out = c(out,x+q)
    }
  } else {
    x = z*p^(1/alf)
    v = runif(1)
    if (v <= (2-x)/(2+x)){
      p = runif(3)
      p = prod(p)
      q = -log(p)
      out = c(out,x+q)
    } else if (v>exp(-1)){
      next
    } else {
      p = runif(3)
      p = prod(p)
      q = -log(p)
      out = c(out,x+q)
    }
  }
}
time2 = proc.time()-time
out
h = melt(out)
length(out)
h = melt(out)
ggplot(data= h,aes(x=value))+geom_density(color = "maroon", fill = "maroon", alpha = 0.5)+xlim(0,12)+
  geom_vline(xintercept = mean(out), color = "blue")+geom_vline(xintercept = median(out), color = "gree
  stat_function(fun=dgamma, args=list(shape=3.2, rate=1), color = "black")
```

2.3 Algorithm GC

```
out=vector()
alf = 1.5
time = proc.time()
for (i in 1:1000){
```

```
b = alf-1
A = alf+b
s = A^0.5
u = runif(1)
t = s*tan(3.14*(u-0.5))
x = b+t
if (x<0) {
  next
} else {
  v = runif(1)
  if (v > exp(b*log(x/b))-t+ log(1+(t^2)/A)){
    next
  } else {
    out = c(out,x)
  }
}
}
time2 = proc.time()-time
out
h = melt(out)
length(out)
ggplot(data= h,aes(x=value))+geom_density(color = "maroon", fill = "maroon", alpha = 0.5)+xlim(0,6)+
  geom_vline(xintercept = mean(out))+geom_vline(xintercept = median(out))+
  stat_function(fun=dgamma, args=list(shape=1.5, rate=1), color = "black")
```

2.4 Algorithm GN

```
out = vector()
alf = 1.5
time = proc.time()
for (i in 1:1000){
mu = alf-1
sig = (alf+1.63299316185545*(alf)^0.5)^0.5
d = 2.44948974278318*sig
b = mu+d
u = runif(1)
if (u<=0.009572265238289){
  s = rexp(1)
  x = b*(1+s/d)
  v = runif(1)
  v = log(v)
  z = mu*(2+log(x/mu)-(x/b))+3.7203284924588-b-log(sig*d/b)
  if (is.na(z)!=TRUE && v>z){
    next
  } else {
    out = c(out,x)
  }
} else {
  s = rnorm(1)
  x = mu+sig*s
  if (x<0 || x>b){
    next
```

```
  } else {
    v = runif(1)
    v = log(v)
    z = mu*(1+log(x/mu))-x-(s^2)/2
    if (is.na(z)!= TRUE && v>z){
      next
    } else {
      out = c(out,x)
    }
  }
}
}
time2 = proc.time()- time
time2
out
length(out)
h = melt(out)
ggplot(data= h,aes(x=value))+geom_density(color = "maroon", fill = "maroon", alpha = 0.5)+xlim(0,6)+
  geom_vline(xintercept = mean(out))+geom_vline(xintercept = median(out))+
  stat_function(fun=dgamma, args=list(shape=1.5, rate=1), color = "black")
```

2.5 Algorithm 1

```
time = proc.time()
o = vector()
for (val in 1:1000){
    x=runif(1)
    y = -2*log(1-(x)^(1/0.7))
    z = (((y)^(0.7-1))*exp(-y/2))/(((2)^(0.7-1))*((1-exp(-y/2))^(0.7-1)))
    p = runif(1)
    if (is.nan(z) == FALSE && p <=z)  {
        a = runif(100)
        a = prod(a)
        b = -log(a)
        o = c(o, y+b)
    } else {
        next
    }
}
time2 = proc.time()-time
time2
length(o)
h = melt(o)
ggplot(data= h,aes(x=value))+geom_density(color = "maroon", fill = "maroon", alpha = 0.5)+xlim(50,150)+
  geom_vline(xintercept = mean(o))+geom_vline(xintercept = median(o))+
  stat_function(fun=dgamma, args=list(shape=100.7, rate=1), color = "black")
```

2.6 Algorithm 2

```
time=proc.time()
out=vector()
alp=0.2
a=(((1-exp(-1/2))^alp)/(((1-exp(-1/2))^alp)+(alp*exp(-1))/2^alp))
```

```
b=((1-exp(-1/2))^alp+(alp*exp(-1))/2^alp)
for (variable in 1:1000) {
  u=runif(1)
  if(u<=a){
    x=-2*log(1-(u*b)^1/alp)
  }
  else{
    x=-log((2^alp)*b*(1-u)/alp)
  }
  z=(((x)^(alp-1))*exp(-x/2))/(((2)^(alp-1))*((1-exp(-x/2))^(alp-1)))
  v=runif(1)
  if(is.na(x) != TRUE && x<=1){
    if(v<=z){
      p=runif(3)
      p= prod(p)
      q = -log(p)
     out=c(out,q+x)
    }
    else{
      next
    }
  }
   else if(is.na(x)!= TRUE && x>1){
     if(v<=x^(alp-1)){
       p=runif(3)
       p= prod(p)
       q = -log(p)
       out=c(out,q+x)
     }
    else{
      next
    }
   }
}
out
time2=proc.time()-time
length(out)
h = melt(out)
ggplot(data= h,aes(x=value))+geom_density(color = "maroon", fill = "maroon", alpha = 0.5)+xlim(0,12)+
  geom_vline(xintercept = mean(o))+geom_vline(xintercept = median(o))+
  stat_function(fun=dgamma, args=list(shape=3.2, rate=1), color = "black")
```

2.7 Algorithm 3

```
time = proc.time()
o = vector()
alf = 0.2
d = 1.0334-0.0766*exp(2.2942*alf)
a = (2^alf)*((1-exp(-d/2))^alf)
b = alf*(d^(alf-1))*exp(-d)
c = a+b
for (i in 1:1000){
```

```
  u = runif(1)
  if (u<=a/(a+b)){
    x = -2*log(1-((c*u)^(1/alf))/2)
  }
  else {
    x = -2*log(c*(1-u)*(1/(alf*d^(alf-1))))
  }
  v = runif(1)
  if (x<=d) {
    z = ((x^(alf-1))*exp(-x/2))/((2^(alf-1))*(1-exp(-x/2))^(alf-1))
    if (is.nan(z) == FALSE && v <= z){
      a = runif(3)
      a = prod(a)
      b = -log(a)
      o = c(o, b+x)
    }
  } else if (v<= (d/x)^(1-alf)){
    a = runif(3)
    a = prod(a)
    b = -log(a)
    o = c(o, b+x)
  }
}
h = melt(o)
time2 = proc.time()-time
length(o)
ggplot(data= h,aes(x=value))+geom_density(color = "maroon", fill = "maroon", alpha = 0.5)+xlim(0,12)+
  geom_vline(xintercept = mean(o))+geom_vline(xintercept = median(o))+
  stat_function(fun=dgamma, args=list(shape=3.2, rate=1), color = "black")
```

References

Joachim H Ahrens and Ulrich Dieter. Computer methods for sampling from gamma, beta, poisson and bionomial distributions. *Computing*, 12(3):223–246, 1974a.

Joachim H Ahrens and Ulrich Dieter. Computer methods for sampling from gamma, beta, poisson and bionomial distributions. *Computing*, 12(3):223–246, 1974b.

DJ Best. A note on gamma variate generators with shape parameter less than unity. *Computing*, 30(2): 185–188, 1983.

Debasis Kundu and Rameshwar D. Gupta. A convenient way of generating gamma random variables using generalized exponential distribution. *Computational Statistics & Data Analysis*, 51(6):2796 – 2802, 2007a. ISSN 0167-9473. doi: https://doi.org/10.1016/j.csda.2006.09.037. URL http://www.sciencedirect.com/science/article/pii/S0167947306003616.

Debasis Kundu and Rameshwar D. Gupta. A convenient way of generating gamma random variables using generalized exponential distribution. *Computational Statistics & Data Analysis*, 51(6):2796 – 2802, 2007b. ISSN 0167-9473. doi: https://doi.org/10.1016/j.csda.2006.09.037. URL http://www.sciencedirect.com/science/article/pii/S0167947306003616.