Simulating Inverse Gaussian Distribution

Amin Rezaei

Computer Science BSc - Amirkabir University of Technology

January 29, 2021

Abstract

Inverse Gaussian Distribution (also known as Wald Distribution) is a two-parameter continous probability distributions with support $(0, \infty)$. In this project we will cover generating samples from this distribution with 3 different approaches.

1 Distribution properties

This distribution has two parameters and has support $(0, \infty)$, also its pdf is defined as follows :

$$f(x; \mu, \lambda) = \sqrt{\frac{\lambda}{2\pi x^3}} exp(-\frac{\lambda(x-\mu)^2}{2\mu^2 x})$$

for $x > 0$, $\mu > 0$ is mean and $\lambda > 0$ is shape parameter.

To indicate that random variable X is Inverse-Gaussian distributed we write: $X \sim IG(\mu, \lambda)$.

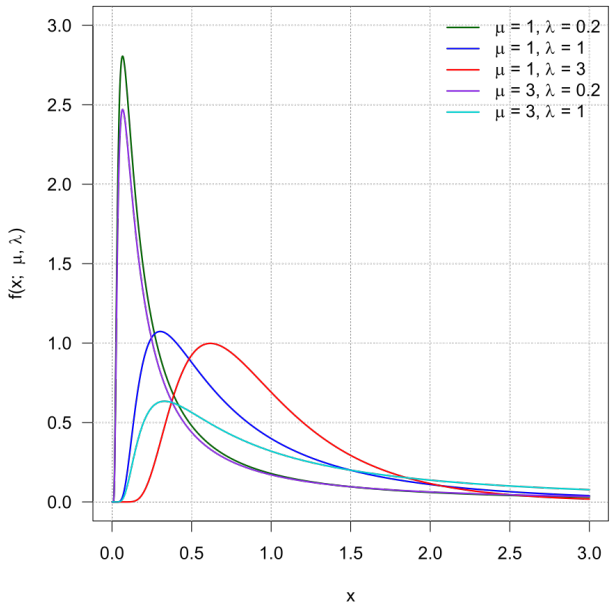

Figure 1: Graph of Probability Density Function for different parameters

2 Acceptance-Rejection Method

First, We analyze the general algorithm then discuss using it for generating from mentioned distribution.

Algorithm

1. First we find random variable Y such that it's density is $g(x)$ and exists c which the inequality $\frac{f(x)}{g(x)} \leq c$ holds for all x that $f(x) > 0$. It's recommended that generating random variates from density $g(x)$ be easy.

2. To generate each random variate:

(a) Sample y from density g.

(b) Sample u from $U(0,1)$.

(c) If $u < \frac{f(y)}{cg(y)}$ holds, y is accepted as a random observation from g; otherwise we repeat from step (a).

Generating observation from inverse-gaussian with acceptance-rejection With paying some attention to pdf graph of the distribution, we notice that it has a global maximum and it's value tends to zero by increasing x. Consider k as the function maximum and d the point that function no more changes significantly (It has tended to zero). Y will be distributed as $U(0,d)$. We have:

$$\frac{f(x)}{g(x)} = \frac{f(x)}{\frac{1}{d}} \leq d * max(f(x)) = kd$$

The value $c = kd$ is one that we need for acceptance-rejection algorithm:

$$u < \frac{f(y)}{cg(y)} \Rightarrow u < \frac{f(y)}{kd\frac{1}{d}} \Rightarrow u < \frac{f(y)}{k}$$

Thus, it's enough to check that the condition is met then y can be accepted as an observation from target distribution.

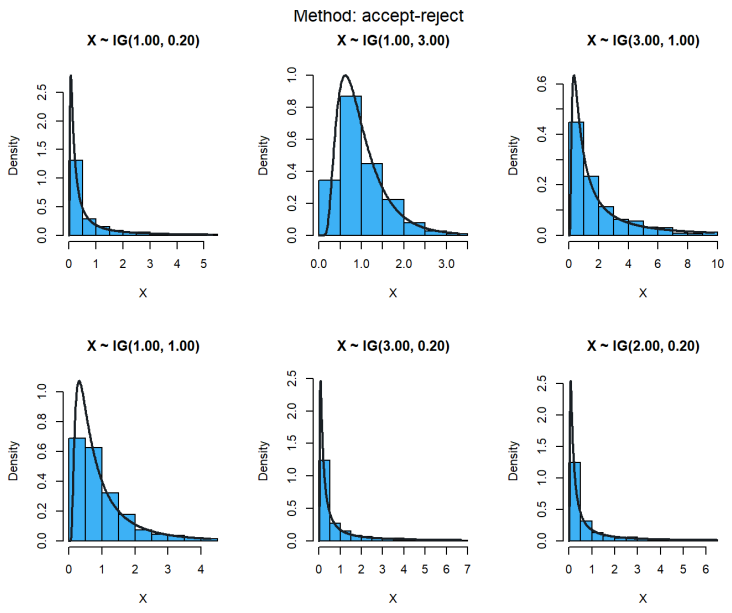

Figure 2: Comparison of pdf function with generated samples density for different pair of parameters

Test Method	$IG(1, 0.2)$	$IG(1, 1)$	$IG(1, 3)$	$IG(3, 0.2)$	$IG(3, 1)$	$IG(2, 0.2)$
Chi-Squared	0.7746	0.4648	0.5087	0.6202	0.0505	0.5942
Kolmogorov–Smirnov	0.0673	0.4879	0.7332	0.0000	0.0000	0.0002

Table 1: Goodness of Fit Tests results

Code We implement the discussed algorithm in R. Following section contains utility functions about distribution itself:

```
dInvGauss <- function(x, lambda, mu) {
  sqrt(lambda / (2 * pi * (x * x * x))) * exp(-1 *
                                                lambda *
                                                (x - mu) *
                                                (x - mu) *
                                                (1 / (2 * mu * mu * x)))
}
dInvGaussGen <- function(lambda, mu) {
  return(function(x) {
      dInvGauss(x, lambda, mu)
    })
}
pInvGauss <- function(q, lambda, mu) {
  t1 <- sqrt(lambda / q)
  t2 <- lambda / mu
  t3 <- q / mu
  return(pnorm(t1 * (t3 - 1)) + exp(2 * t2) * pnorm(-1 * t1 * (t3 + 1)))
}
```

Function findXNearZero is implemented to find value d (Point that function tends zero) . To find k (Maximum) we use optimize function that is builtin in R to find the maximum numerically. The rest is clear, at last function rInvGaussAR is implemeneted so that it generates variates

```r
findXNearZero <- function(targetFunc, threshold = 0.05) {
  seenHigher <- FALSE
  i <- 0.001
  while (TRUE) {
    lastV <- targetFunc(i)
    if (lastV > threshold && !seenHigher) {
      seenHigher <- TRUE
    }
    if (lastV < threshold && seenHigher) {
      break
    }
    i <- i + 0.04
  }
  return(i)
}

rInvGaussAR <- function(n, lambda, mu) {
  d <- findXNearZero(dInvGaussGen(lambda, mu), threshold = 0.01)
  k <- optimize(f=dInvGaussGen(lambda, mu), interval = c(0,d), maximum=T)
  k <- k * 1.4
  needed <- n
  X <- c()
  while (needed > 0){
    m <- as.integer(k * needed) + 1
    y <- runif(m, 0, d)
    u <- runif(m, 0, 1)
    accepted <- y[(dInvGauss(y, lambda, mu) / k) ≥ u]
    X <- c(X, accepted)
    needed <- needed - length(accepted)
  }
  return(X[1:n])
}
```

3 Independent Metropolis-Hastings Algorithm

Algorithm If we have distribution with density g independent from $X^{(t)}$ and go ahead to run normal Metropolis-Hastings Algorithm with it, it's called Independent Metropolis-Hastings Algorithm:
Consider we have $x^{(t)}$

1. Sample $y_t \sim g(y)$
2. Define: $\rho(x^{(t)}, y_t) = min(1, \frac{f(y_t)g(x^{(t)})}{f(x^{(t)})g(y_t)})$
3. Assign:

$$X^{(t+1)} = \begin{cases} y_t & with\ probability\ \rho \\ x^{(t)} & with\ probability\ 1-\rho \end{cases}$$

For discussed distribution we will select density g Gamma Distribution Density with parameters that mean and the variance of distribution be as same as ours. Each time we will sample from $U(0,1)$ and check it with ρ and select the corresponding expression in the step 3 assignment. A chain with desired distribution can be accomplished by repeating the process over and over again.

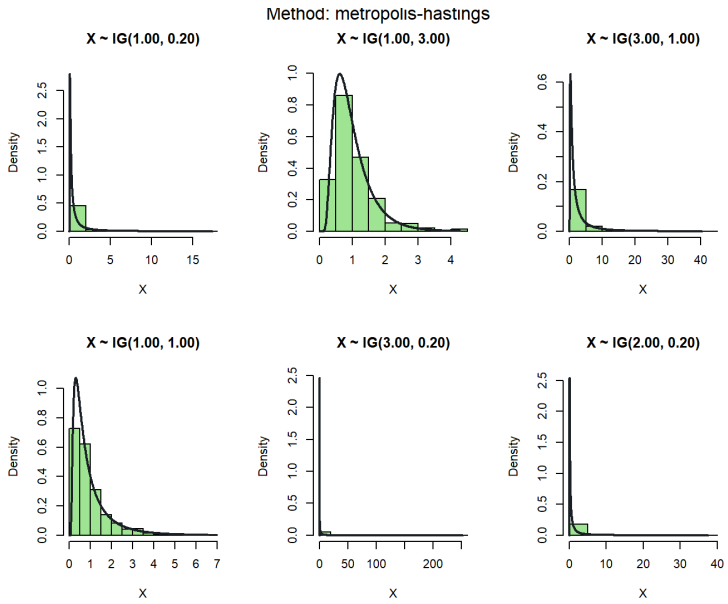

Figure 3: Comparison of pdf function with generated samples density for different pair of parameters

Test Method	$IG(1, 0.2)$	$IG(1, 1)$	$IG(1, 3)$	$IG(3, 0.2)$	$IG(3, 1)$	$IG(2, 0.2)$
Chi-Squared	0.1124	0.4053	0.0005	0.0005	0.8856	0.0005
Kolmogorov–Smirnov	–	–	–	–	–	–

Table 2: Goodness of Fit Tests results

Code We will implement function rInvGaussMH to sample from desired distribution with Independent Metropolis-Hasting Method.

```
rInvGaussMH <- function(n, lambda, mu) {
  alpha <- lambda / mu
  beta <- lambda / (mu * mu)
  scale <- 1/beta
  X <- numeric(n)
  X[1] <- rgamma(1, shape=alpha, scale=scale) # initialize the chain
  dGauss <- function (x) dInvGauss(x, lambda, mu)
  dGamma <- function (x) dgamma(x, shape=alpha, scale=scale)
  for (i in 2:n) {
    Y <- rgamma(1, shape=alpha, scale=scale)
    rho <- (dGauss(Y) * dGamma(X[i - 1])) / (dGauss(X[i - 1]) * dGamma(Y)
    if (rho > 1) {
      rho <- 1
    }
    X[i] <- X[i - 1] + (Y - X[i - 1]) * (runif(1) < rho)
  }
  return(X)
}
```

4 Transform with multiple roots

In this section, We will discuss and use the proposed algorithm by [4]. Consider having a transform $v = g(x)$ that could replace some expression in main density, if sampling from v be accessible by solving the transform for x we could end with result that is distributed as our density. The main problem is such transform may have multiple roots, How can we know whom to select? The mentioned paper proposes a way to obtain selection probability for each of roots that solves our problem.

Algorithm

1. Consider transform V as written below, In [5] it has been proved that V is distributed as $\chi^2_{(1)}$. Sampling from mentioned distribution is handy.

$$V = g(X) = \frac{\lambda(X-\mu)^2}{\lambda^2 X} \sim \chi^2_{(1)}$$

2. With solving the equation for x, we reach:

$$x_1 = \mu + \frac{\mu^2 v_0}{2\lambda} - \frac{\mu}{2\lambda}\sqrt{4\mu\lambda v_0 + \mu^2 v_0^2};\ x_2 = \mu^2/x_1$$

3. Probability of x_1 being answer can be calculated by method proposed in [4] and is:

$$p_1(v_0) = (1 + |\frac{g'(x_1)}{g'(x_2)}|\frac{f(x_2)}{f(x_1)})^{-1} = \frac{\mu}{\mu + x_1}$$

4. So just with sampling from $\chi^2_{(1)}$ and obtaining the probability from later step, We can select desired root with sampling from $U(0,1)$ and checking the probability and at last we reach our needed variates from main distribution.

Test Method	$IG(1, 0.2)$	$IG(1,1)$	$IG(1,3)$	$IG(3, 0.2)$	$IG(3,1)$	$IG(2, 0.2)$
Chi-Squared	0.5752	0.0415	0.4218	0.1264	0.7191	0.1654
Kolmogorov–Smirnov	0.5645	0.7781	0.8274	0.6049	0.9716	0.2497

Table 3: Goodness of Fit Tests results

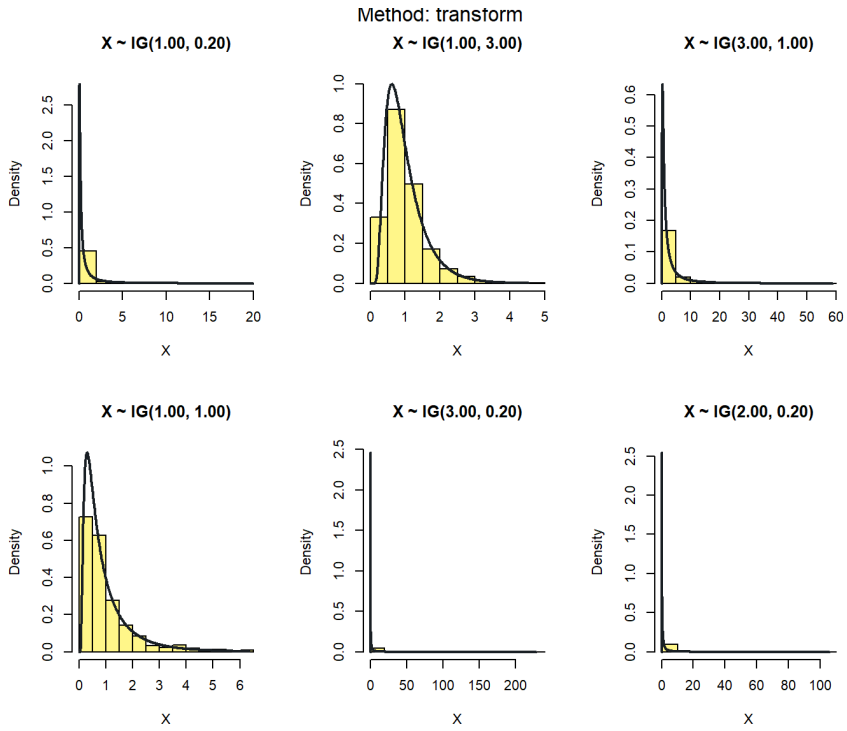

Figure 4: Comparison of pdf function with generated samples density for different pair of parameters

Code We will implement function rInvGaussT to sample from desired distribution with Multiple root transform method as discussed.

```
rInvGaussT ← function(n, lambda, mu) {
  v ← rnorm(n)
  v ← v * v
  w ← v * mu
  c ← mu / (2. * lambda)
  x ← mu + c * (w - sqrt(w * (4 * lambda + w)))
  z ← runif(n)
  truth ← as.integer(z ≤ (mu / (x + mu)))
  R ← truth * x + (1 - truth) * (mu * mu / x)
  return(R)
}
```

Final Function Function rInvGauss is implemented as final function which samples random variates with given length and parameters with desired method:

```
rInvGauss <- function(n, lambda, mu, method = "transform") {
  if (method == "metropolis-hastings") {
    return(rInvGaussMH(n, lambda, mu))
  }else if (method == "accept-reject") {
    return(rInvGaussAR(n, lambda, mu))
  }else if (method == "transform") {
    return(rInvGaussT(n, lambda, mu))
  }
}
```

5 Conclusion

With comparing the graphs and test results, It's obvious that the last method (Transform with multiple roots) is the most stable way to generate variates. Acceptance-Rejection method is also performs well and all the tests suit. One of disadvantages of Metropolis-Hastings method is that the variates are not independent and duplicates may be also generated which prevents us from using Kolmogorov–Smirnov test. Also, a few tweaks may be needed for selecting the other distribution to increase accuracy against different parameters. As you can see, the tests show that the used distribution with discussed parameters was not the ideal one and lead to bad results for some pair of parameters.

References

[1] A. MohammadPour. Simulation lecture and slides.

[2] M.L. Rizzo. *Statistical Computing with R.* Chapman & Hall/CRC The R Series. Taylor & Francis, 2007.

[3] Wikipedia. Inverse Gaussian distribution — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Inverse%20Gaussian%20distribution&oldid=990830775, 2021. [Online; accessed 23-January-2021].

[4] John R. Michael, William R. Schucany, and Roy W. Haas. Generating random variates using transformations with multiple roots. *The American Statistician*, 30(2):88–90, 1976.

[5] Shuster J. On the inverse gaussian distribution function. *J. Amer. Statist. Assoc.*, 63:1514, 1968.

A Appendix 1

All the project source code (including Plots, Tests, ...) is provided in this section. Usage of zoo library is for the rollapply function needed for Chi-Squared Test. Also, final revision is present at the following github repository: https://github.com/AminRezaei0x443/InverseGaussianDist

```
library(zoo)

dInvGauss <- function(x, lambda, mu) {
  sqrt(lambda / (2 * pi * (x * x * x))) * exp(-1 *
                                                lambda *
                                                (x - mu) *
                                                (x - mu) *
                                                (1 / (2 * mu * mu * x)))
}

dInvGaussGen <- function(lambda, mu) {
  return(function(x) {
    dInvGauss(x, lambda, mu)
```

```
  })
}

pInvGauss <- function(q, lambda, mu) {
  t1 <- sqrt(lambda / q)
  t2 <- lambda / mu
  t3 <- q / mu
  return(pnorm(t1 * (t3 - 1)) + exp(2 * t2) * pnorm(-1 * t1 * (t3 + 1)))
}

rInvGaussMH <- function(n, lambda, mu) {
  alpha <- lambda / mu
  beta <- lambda / (mu * mu)
  scale <- 1/beta
  X <- numeric(n)
  X[1] <- rgamma(1, shape=alpha, scale=scale) # initialize the chain
  dGauss <- function (x) dInvGauss(x, lambda, mu)
  dGamma <- function (x) dgamma(x, shape=alpha, scale=scale)
  for (i in 2:n) {
    Y <- rgamma(1, shape=alpha, scale=scale)
    rho <- (dGauss(Y) * dGamma(X[i - 1])) /
           (dGauss(X[i - 1]) * dGamma(Y))
    if (rho > 1) {
      rho <- 1
    }
    X[i] <- X[i - 1] + (Y - X[i - 1]) * (runif(1) < rho)
  }
  return(X)
}


findXNearZero <- function(targetFunc, threshold = 0.05) {
  seenHigher <- FALSE
  i <- 0.001
  while (TRUE) {
```

```
    lastV <- targetFunc(i)
    if (lastV > threshold && !seenHigher) {
      seenHigher <- TRUE
    }
    if (lastV < threshold && seenHigher) {
      break
    }
    i <- i + 0.04
  }
  return(i)
}

rInvGaussAR <- function(n, lambda, mu) {
  d <- findXNearZero(dInvGaussGen(lambda, mu), threshold = 0.01)
  k <- optimize(f=dInvGaussGen(lambda, mu), interval = c(0,d),
                maximum=T)$objective
  #for increasing confidency and lesser loops
  k <- k * 1.4
  needed <- n
  X <- c()
  while (needed > 0){
    m <- as.integer(k * needed) + 1
    y <- runif(m, 0, d)
    u <- runif(m, 0, 1)
    accepted <- y[(dInvGauss(y, lambda, mu) / k) ≥ u]
    X <- c(X, accepted)
    needed <- needed - length(accepted)
  }
  return(X[1:n])
}

rInvGaussT <- function(n, lambda, mu) {
  v <- rnorm(n)
  v <- v * v
  w <- v * mu
```

```
  c <- mu / (2. * lambda)
  x <- mu + c * (w - sqrt(w * (4 * lambda + w)))
  z <- runif(n)
  truth <- as.integer(z <= (mu / (x + mu)))
  R <- truth * x + (1 - truth) * (mu * mu / x)
  return(R)
}

rInvGauss <- function(n, lambda, mu, method = "transform") {
  if (method == "metropolis-hastings") {
    return(rInvGaussMH(n, lambda, mu))
  }else if (method == "accept-reject") {
    return(rInvGaussAR(n, lambda, mu))
  }else if (method == "transform") {
    return(rInvGaussT(n, lambda, mu))
  }
}

invGaussResults <- function(n) {
  par(mfcol = c(2, 3))
  muC <- c(1, 1, 1, 3, 3, 2)
  lambdaC <- c(0.2, 1, 3, 0.2, 1, 0.2)
  methods <- c("accept-reject", "metropolis-hastings", "transform")
  mColor <- c("#3DB1F5", "#9EE493", "#FFF689")
  lColor <- c("#191F24", "#191F24", "#191F24")
  mIdx <- 1
  for (m in methods) {
    for (i in seq_along(muC)) {
      mu <- muC[i]
      lambda <- lambdaC[i]
      X <- rInvGauss(n, lambda, mu, m)
      cH <- hist(X, plot = FALSE)
      dMax <- max(cH$density)
      i <- seq(0.001, max(X), .01)
      dY <- dInvGauss(i, lambda, mu)
```

```
      hist(X, prob = TRUE, col = mColor[mIdx],
       ylim = c(0, max(dMax, dY)), main=sprintf("X ~ IG(%.2f, %.2f)",
        mu, lambda))
      lines(i, dY, type = "l", lw = 2, col =lColor[mIdx])
      t <- -1
      if(length(unique(X)) == length(X)){
        t <- ks.test(X, pInvGauss, lambda=lambda, mu=mu)$p.value
      }

      breaks_cdf <- pInvGauss(cH$breaks, lambda=lambda, mu=mu)
      null.probs <- rollapply(breaks_cdf, 2, function(x) x[2]-x[1])
      a <- chisq.test(cH$counts, p=null.probs, rescale.p=TRUE,
       simulate.p.value=TRUE)
      if(t != -1){
        print(
          sprintf("K-S p-value - IG(%.2f,%.2f) %s method: %.4f",
           mu, lambda, m, t))
      }else{
        print
        (sprintf("KS p-value - IG(%.2f,%.2f) %s method: NaN (has ties)",
         mu, lambda, m))
      }
      print(
        sprintf("ChiSq p-value - IG(%.2f,%.2f) %s method: %.4f",
         mu, lambda, m, a$p.value))
    }
    mIdx <- mIdx + 1
    mtext(sprintf("Method: %s", m), outer = TRUE, cex = 1, line = -1)
  }

}
set.seed(9712017)
invGaussResults(1000)
```