DATA 604 HW 3

Dan Fanelli

9/27/2016

1 - Print Entire Cycle (11, 121, 99, 33)

```
x <- 1
for(i in c(1:20)){
  x <- 11 * x %% 16
  cat(i,':',x,'\n')
}
```

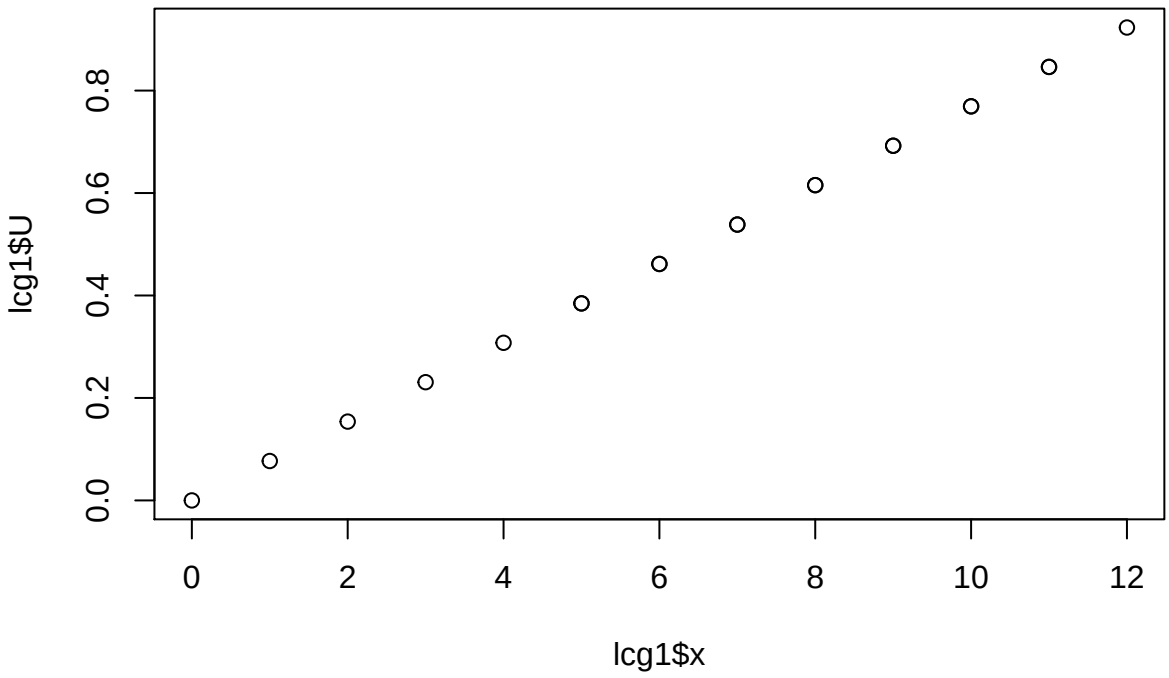
```
## 1 : 11
## 2 : 121
## 3 : 99
## 4 : 33
## 5 : 11
## 6 : 121
## 7 : 99
## 8 : 33
## 9 : 11
## 10 : 121
## 11 : 99
## 12 : 33
## 13 : 11
## 14 : 121
## 15 : 99
## 16 : 33
## 17 : 11
## 18 : 121
## 19 : 99
## 20 : 33
```

2 - Plot the Pairs and Discuss

The below graph's lattice structure shows [lcg1$U] on the y axis as a 0-to-1 scaled version of [lcg1$x] on the x-axis

```
# https://qualityandinnovation.com/2015/03/03/a-linear-congruential-generator-lcg-in-r/
lcg <- function(a,c,m,run.length,seed) {
    x <- rep(0,run.length)
    x[1] <- seed
    for (i in 1:(run.length-1)) {
       x[i+1] <- (a * x[i] + c) %% m
    }
    U <- x/m # scale all of the x's to
             # produce uniformly distributed
```

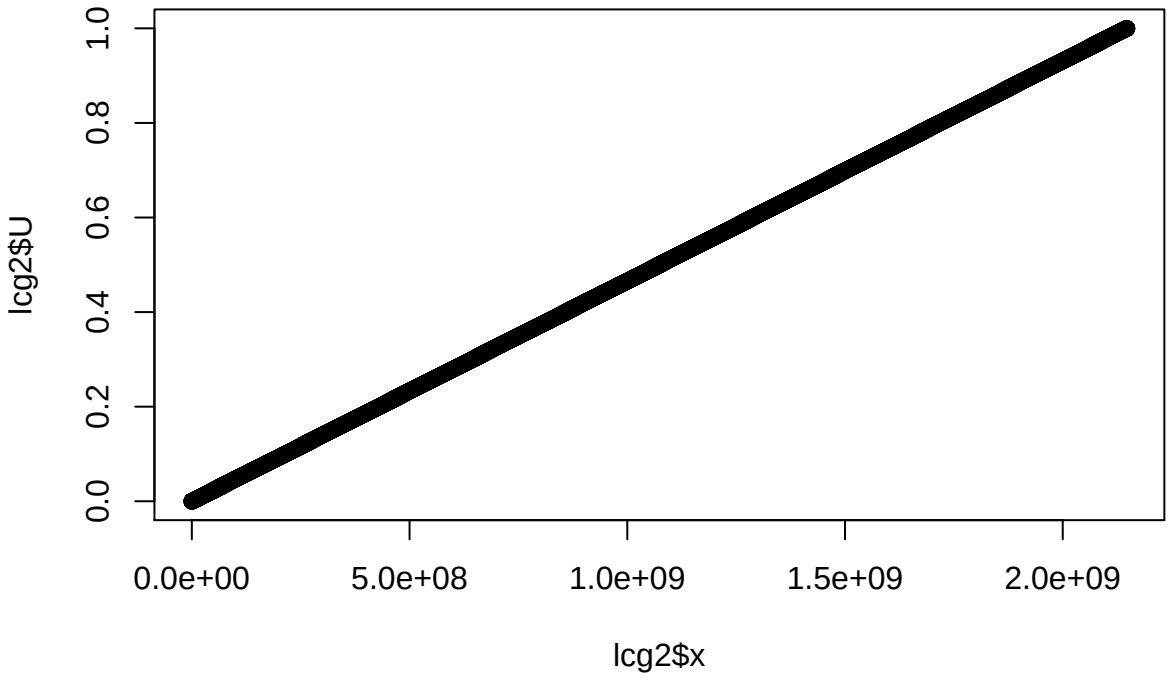
```
            # random numbers between [0,1)
    return(list(x=x,U=U))
}

lcg1 <- lcg(1,12,13,20,11)
plot(lcg1$x, lcg1$U)
```



3 - Pseudo Random Number Generator

These runs are do not contradict the null hypothesis, with p-value well above 5% or even 10%

```
#num_iterations_to_do <- 100
num_iterations_to_do <- 100000
lcg2 <- lcg(16807,0,((2^31)-1),num_iterations_to_do,1234567)
plot(lcg2$x, lcg2$U)
```




```
chisq.test(lcg2$x)
```

```
##
##  Chi-squared test for given probabilities
##
## data:  lcg2$x
## X-squared = 3.5879e+13, df = 99999, p-value < 2.2e-16
```

```
chisq.test(lcg2$U)
```

```
## Warning in chisq.test(lcg2$U): Chi-squared approximation may be incorrect
```

```
##
##  Chi-squared test for given probabilities
##
## data:  lcg2$U
## X-squared = 16707, df = 99999, p-value = 1
```

```
library("randtests")
runs.test(lcg2$x)$p.value
```

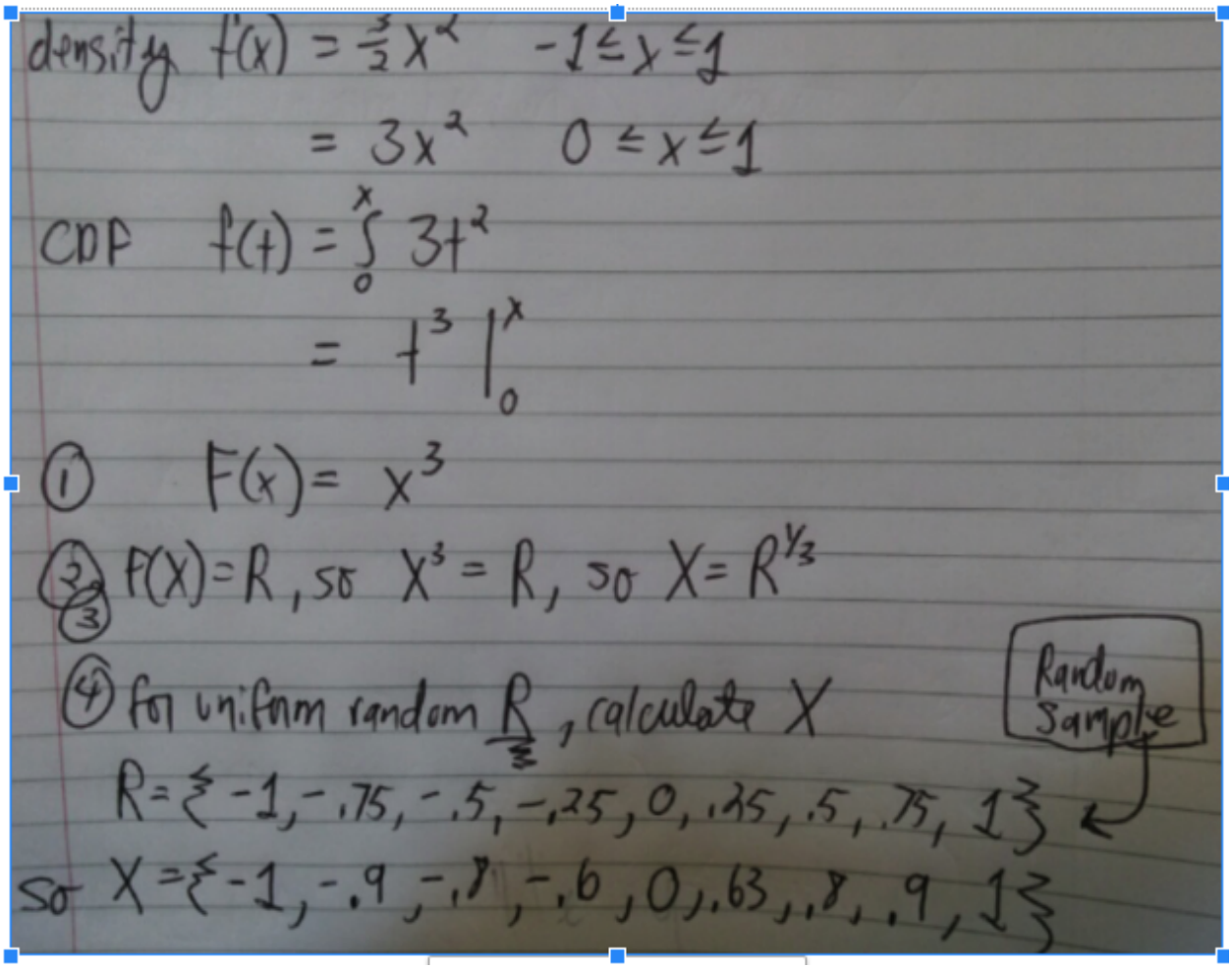
```
## [1] 0.7422475
```

```
runs.test(lcg2$U)$p.value
```

```
## [1] 0.7422475
```

4 - Acceptance-rejection

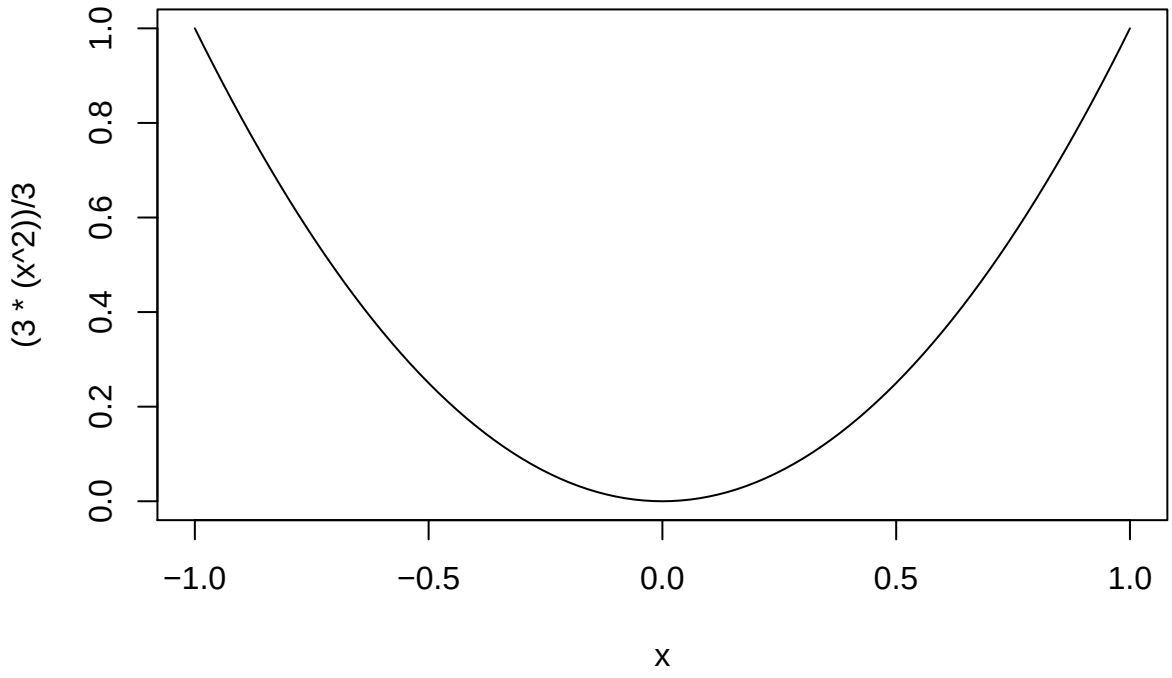
Inverse-transform: (for -1 <= x <= 1, 0 otherwise):



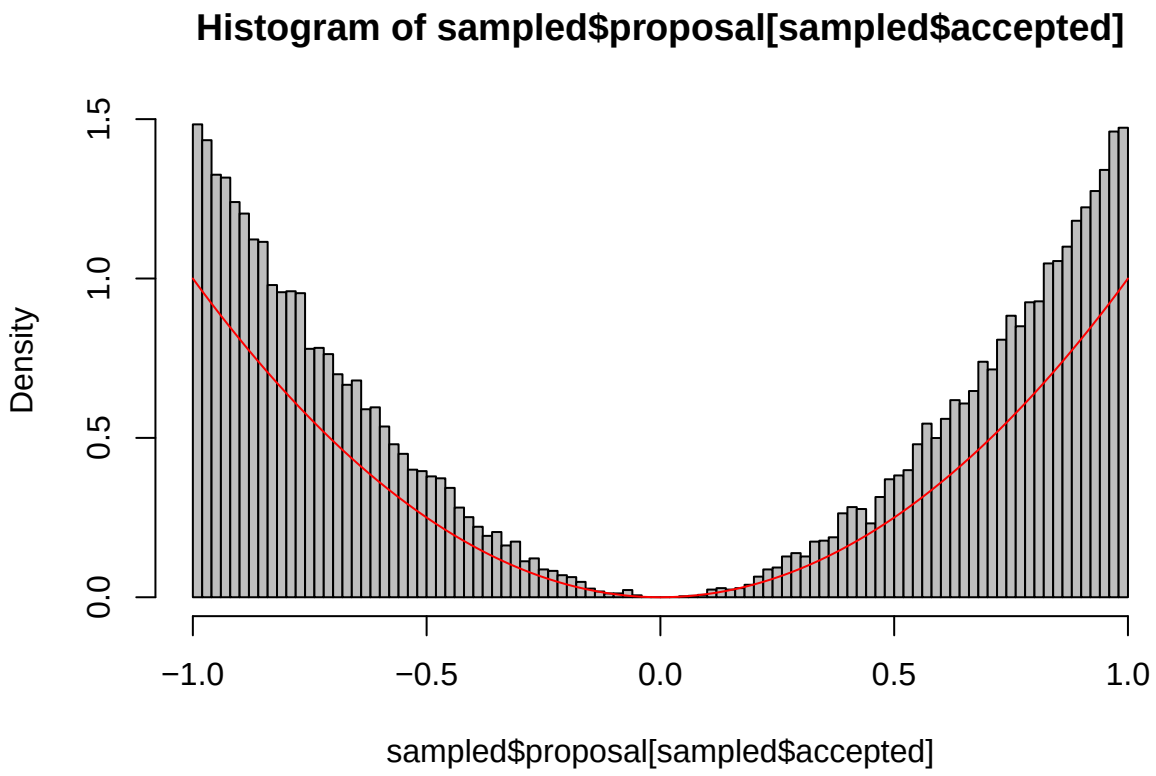
Composition: I think that line 2 above, cutting the domain in half and doubling the output was an example of compostion. (http://web.ics.purdue.edu/~hwan/IE680/Lectures/Chap08Slides.pdf)

Acceptance-rejection algorithm:

```
# (3x^2)/2 from -1 to 1
curve((3*(x^2))/3, -1, 1)
```




```
sampled <- data.frame(proposal = runif(100000,-1,1))
sampled$targetDensity <- (3*(sampled$proposal^2))/3
maxDens = max(sampled$targetDensity, na.rm = T)
sampled$accepted = ifelse(runif(100000,0,1) < sampled$targetDensity / maxDens, TRUE, FALSE)
hist(sampled$proposal[sampled$accepted], freq = F, col = "grey", breaks = 100)
curve((3*(x^2))/3, -1, 1, add =T, col = "red")
```



5 - Implement, Test, and Compare methods to generate from a N(0,1) distribution

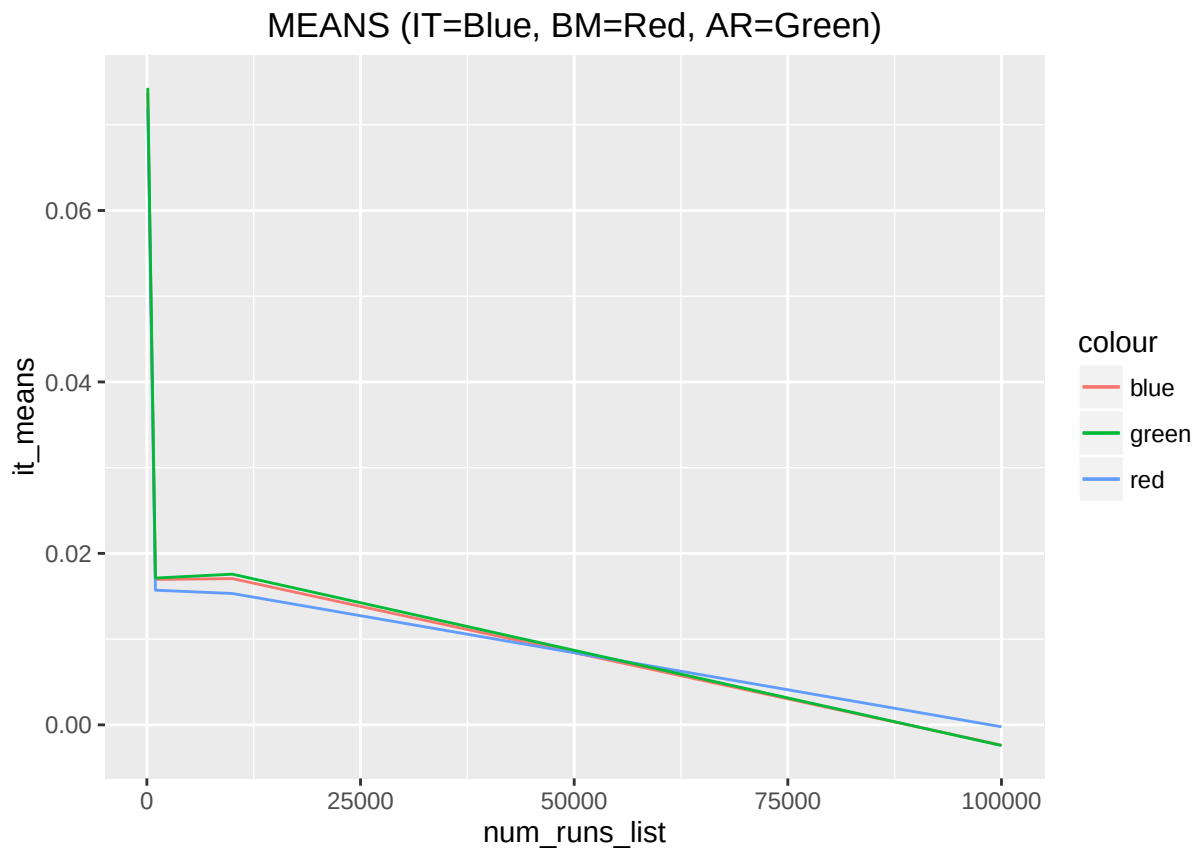
```
# A: it = inverse transform
normrandit <- function(){
  uniform_random_number <- runif(1,0,1)
  return (qnorm(uniform_random_number))
}

itstats <- function(N){
  samples <- c()
  for(i in 1:N){
    samples <- c(samples, normrandit())
  }
  return (c(mean(samples), sd(samples)))
}

# B: bm = box mueller transform
normrandbm <- function(){
  uniform_1 <- runif(1,0,1)
  uniform_2 <- runif(1,0,1)
  X <- sqrt((-2 * log(uniform_1)))*(cos(2 * pi * uniform_2))
  Y <- sqrt((-2 * log(uniform_1)))*(sin(2 * pi * uniform_2))
```

```
  return (c(X,Y))
}

bmstats <- function(N){
  samples <- c()
  for(i in 1:N){
    samples <- c(samples, normrandbm())
  }
  return (c(mean(samples), sd(samples)))
}

# C: ar = accept reject
normrandar <- function(){
  accepted <- FALSE
  X <- 0
  while(!accepted){
    uniform_1 <- runif(1,0,1)
    uniform_2 <- runif(1,0,1)
    X <- (-1 * log(uniform_1))
    Y <- (-1 * log(uniform_2))

    if(Y >= ((X-1)^2)/2){
      accepted <- TRUE
      pos_neg <- runif(1,0,1)
      if(pos_neg < 0.5){
        X <- (-1) * X
      }
    }
  }
  return (X)
}

arstats <- function(N){
  samples <- c()
  for(i in 1:N){
    samples <- c(samples, normrandar())
  }
  return (c(mean(samples), sd(samples)))
}

# D
for(x in c(1:10)){
  it_means <- c();
  it_sd <- c();
  it_tm <- c();
  bm_means <- c();
  bm_sd <- c();
  bm_tm <- c();
  ar_means <- c();
  ar_sd <- c();
  ar_tm <- c();

  num_runs_list <- c(100,1000,10000,100000)
```

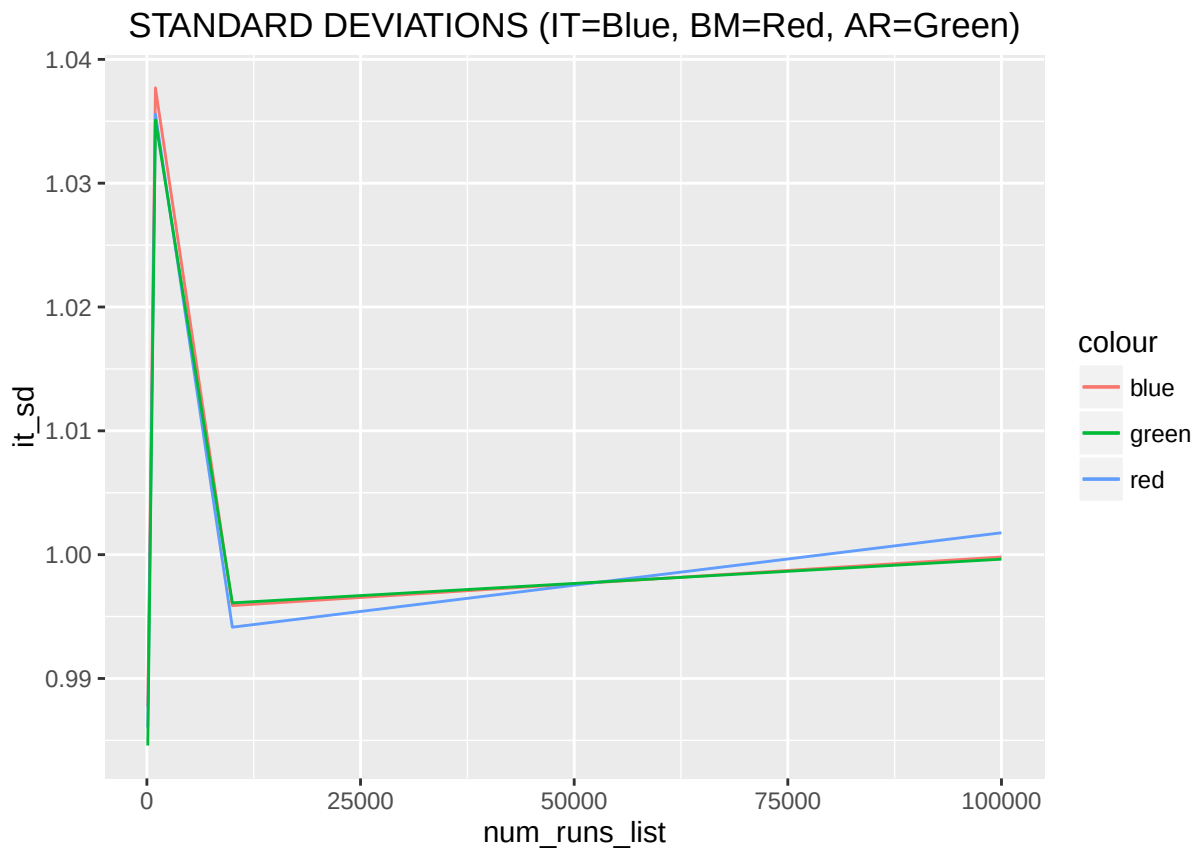
```
  #num_runs_list <- c(100,1000,10000)

  for(runs in num_runs_list){
    start.time <- Sys.time()
    itst <- itstats(runs)
    end.time <- Sys.time()
    it_means <- c(it_means, itst[1])
    it_sd <- c(it_sd, itst[2])
    it_tm <- c(it_tm,(end.time - start.time))

    start.time <- Sys.time()
    bmst <- bmstats(runs)
    end.time <- Sys.time()
    bm_means <- c(bm_means, itst[1])
    bm_sd <- c(bm_sd, itst[2])
    bm_tm <- c(bm_tm,(end.time - start.time))

    start.time <- Sys.time()
    arst <- arstats(runs)
    end.time <- Sys.time()
    ar_means <- c(ar_means, itst[1])
    ar_sd <- c(ar_sd, itst[2])
    ar_tm <- c(ar_tm,(end.time - start.time))
  }
}
# E
library(knitr)
kable(df_e <- data.frame(num_runs_list,it_means,bm_means,ar_means,it_sd,bm_sd,ar_sd,it_tm,bm_tm,ar_tm))
```

num_runs_list	it_means	bm_means	ar_means	it_sd	bm_sd	ar_sd	it_tm	bm_tm
1e+02	0.0723668	0.0723668	0.0723668	0.9857343	0.9857343	0.9857343	0.0000000	0.0156229
1e+03	0.0176848	0.0176848	0.0176848	1.0358556	1.0358556	1.0358556	0.0000000	0.0156250
1e+04	0.0162985	0.0162985	0.0162985	0.9941211	0.9941211	0.9941211	0.2449131	0.4779489
1e+05	-0.0022002	-0.0022002	-0.0022002	1.0015223	1.0015223	1.0015223	20.9029062	52.3721938

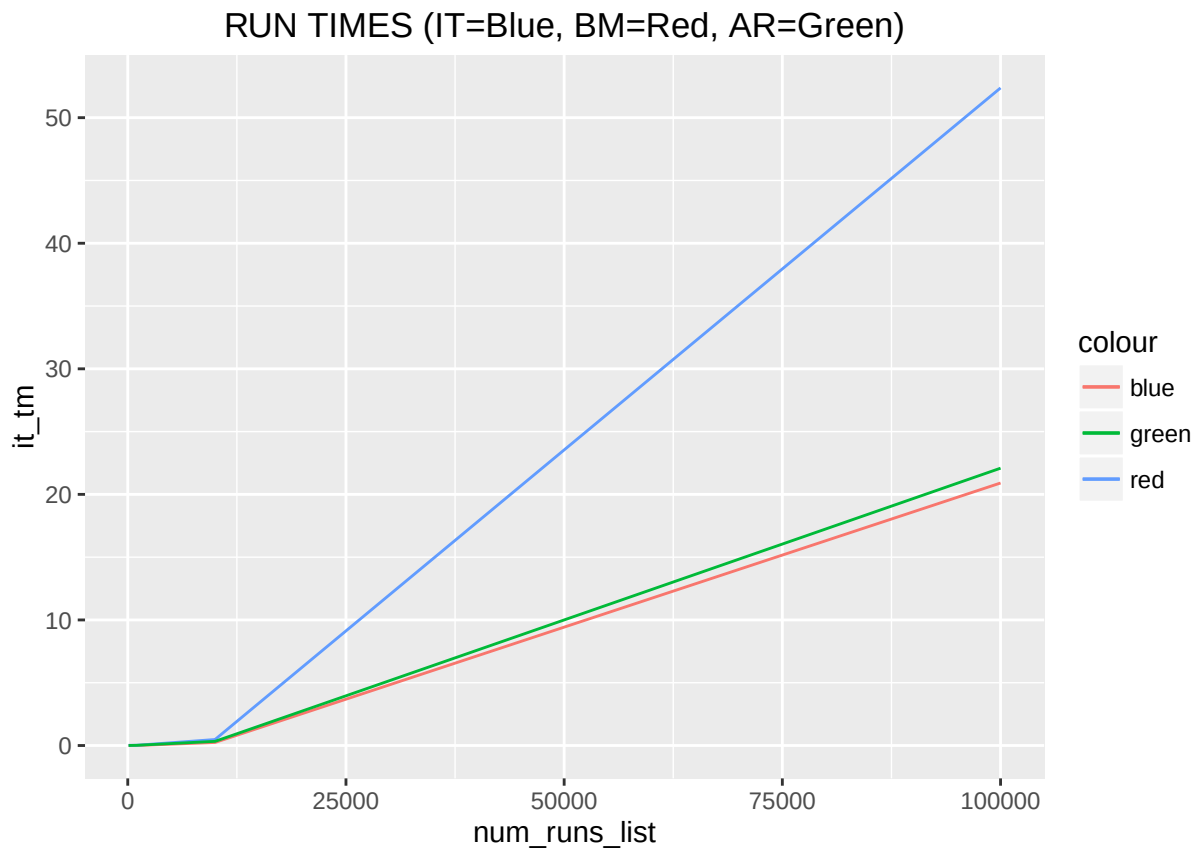
```
library(ggplot2)
ggplot(df_e, aes(x=num_runs_list)) +
  geom_line(aes(y = it_means, colour = "blue"), position=position_jitter(w=0.005, h=0.005)) +
  geom_line(aes(y = bm_means, colour = "red"), position=position_jitter(w=0.005, h=0.005)) +
  geom_line(aes(y = ar_means, colour = "green"), position=position_jitter(w=0.005, h=0.005)) +
  ggtitle("MEANS (IT=Blue, BM=Red, AR=Green)")
```



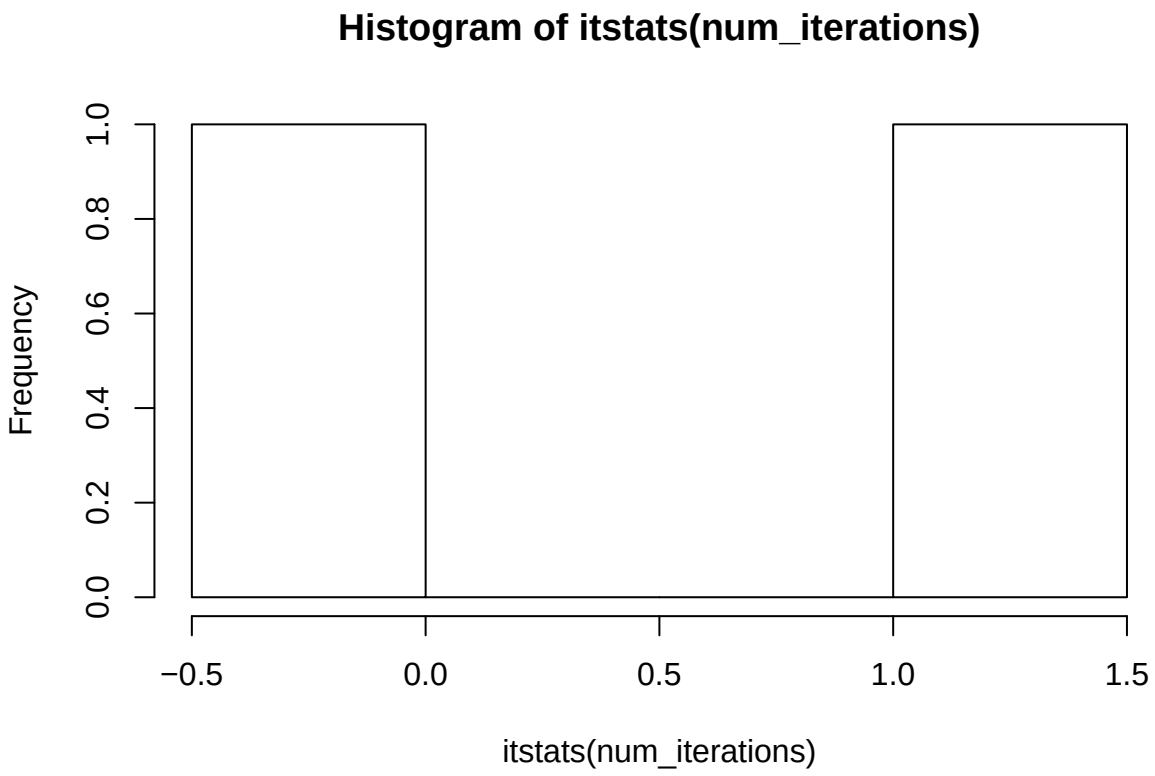

```
ggplot(df_e, aes(x=num_runs_list)) +
  geom_line(aes(y = it_sd, colour = "blue"), position=position_jitter(w=0.005, h=0.005)) +
  geom_line(aes(y = bm_sd, colour = "red"), position=position_jitter(w=0.005, h=0.005)) +
  geom_line(aes(y = ar_sd, colour = "green"), position=position_jitter(w=0.005, h=0.005)) +
  ggtitle("STANDARD DEVIATIONS (IT=Blue, BM=Red, AR=Green)")
```



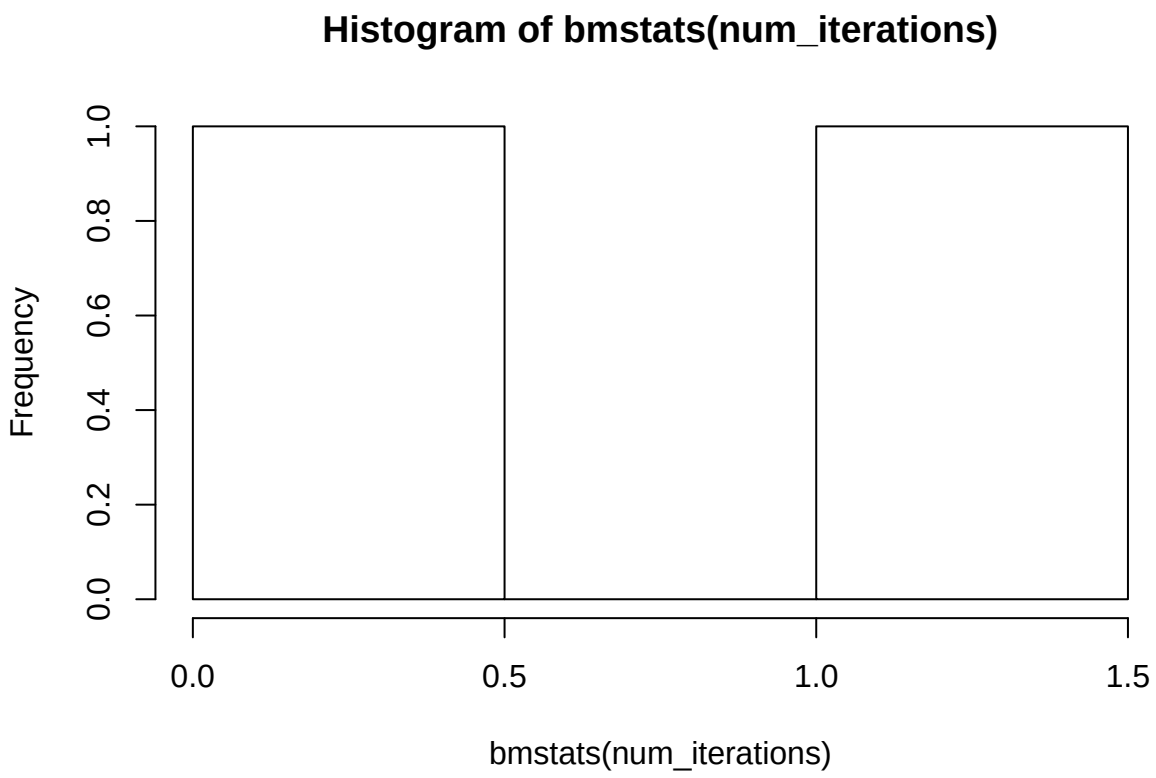

```
ggplot(df_e, aes(x=num_runs_list)) +
  geom_line(aes(y = it_tm, colour = "blue"), position=position_jitter(w=0.005, h=0.005)) +
  geom_line(aes(y = bm_tm, colour = "red"), position=position_jitter(w=0.005, h=0.005)) +
  geom_line(aes(y = ar_tm, colour = "green"), position=position_jitter(w=0.005, h=0.005)) +
  ggtitle("RUN TIMES (IT=Blue, BM=Red, AR=Green)")
```



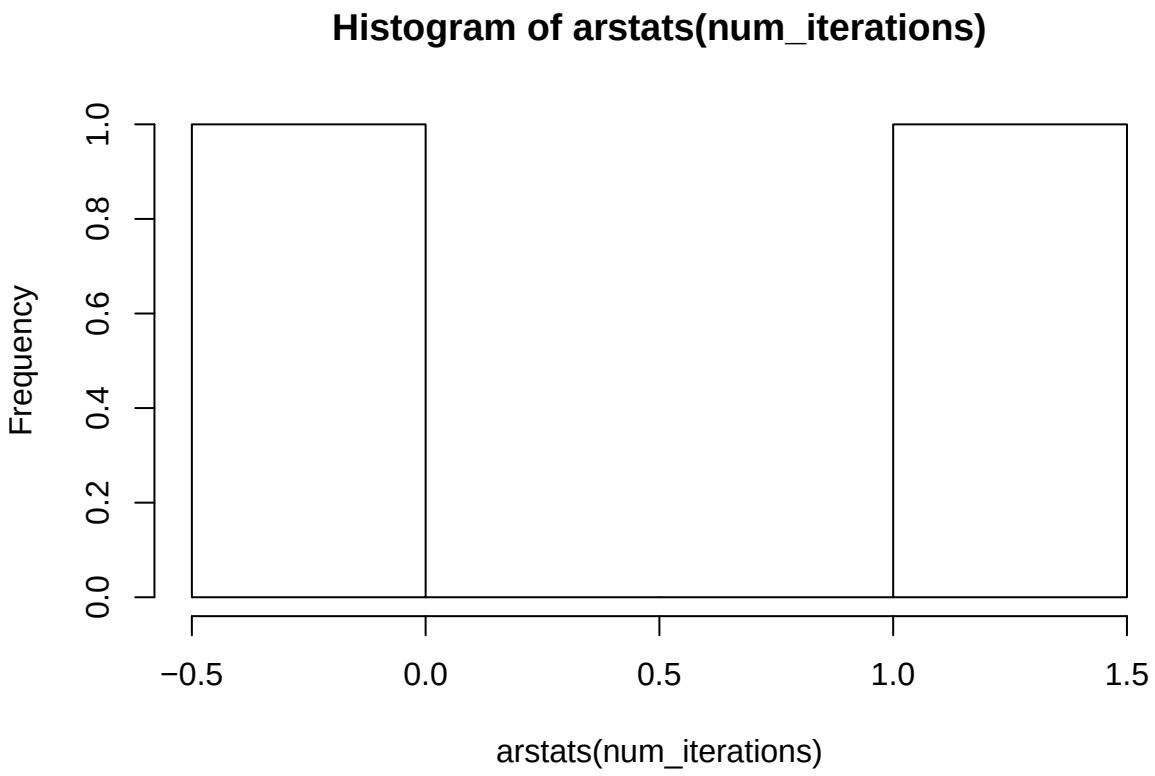

```
num_iterations <- 100#0000
hist(itstats(num_iterations))
```




```
hist(bmstats(num_iterations))
```




```
hist(arstats(num_iterations))
```



6 - Monte Carlo estimation of Pi

```
inside_circle <- function(rand_x, rand_y){
  radius_length <- sqrt(rand_x * rand_x + rand_y * rand_y)
  if(radius_length <= 1){
    return (1.0)
  }else{
    return (0.0)
  }
}

estimatepi <- function(num_pairs){
  rand_x <- runif(num_pairs, 0, 1)
  rand_y <- runif(num_pairs, 0, 1)
  sum_below_circle <- 0
  for(i in 1:num_pairs){
    sum_below_circle <- sum_below_circle + inside_circle(rand_x[i], rand_y[i])
  }
  return (4*sum_below_circle/num_pairs)
}

estimatepi(1)
```

```
## [1] 0
```

```
estimatepi(10)
```

```
## [1] 3.6
```

```
estimatepi(100)
```

```
## [1] 3.12
```

```
estimatepi(1000)
```

```
## [1] 3.212
```

```
estimatepi(10000)
```

```
## [1] 3.1556
```

All Code

```
x <- 1
for(i in c(1:20)){
  x <- 11 * x %% 16
  cat(i,':',x,'\n')
}
# https://qualityandinnovation.com/2015/03/03/a-linear-congruential-generator-lcg-in-r/
lcg <- function(a,c,m,run.length,seed) {
    x <- rep(0,run.length)
    x[1] <- seed
    for (i in 1:(run.length-1)) {
       x[i+1] <- (a * x[i] + c) %% m
    }
    U <- x/m # scale all of the x's to
             # produce uniformly distributed
             # random numbers between [0,1)
    return(list(x=x,U=U))
}

lcg1 <- lcg(1,12,13,20,11)
plot(lcg1$x, lcg1$U)
#num_iterations_to_do <- 100
num_iterations_to_do <- 100000
lcg2 <- lcg(16807,0,((2^31)-1),num_iterations_to_do,1234567)
plot(lcg2$x, lcg2$U)
chisq.test(lcg2$x)
chisq.test(lcg2$U)
library("randtests")
runs.test(lcg2$x)$p.value
```

```r
runs.test(lcg2$U)$p.value
# (3x^2)/2 from -1 to 1
curve((3*(x^2))/3, -1, 1)
sampled <- data.frame(proposal = runif(100000,-1,1))
sampled$targetDensity <- (3*(sampled$proposal^2))/3
maxDens = max(sampled$targetDensity, na.rm = T)
sampled$accepted = ifelse(runif(100000,0,1) < sampled$targetDensity / maxDens, TRUE, FALSE)
hist(sampled$proposal[sampled$accepted], freq = F, col = "grey", breaks = 100)
curve((3*(x^2))/3, -1, 1, add =T, col = "red")
# A: it = inverse transform
normrandit <- function(){
  uniform_random_number <- runif(1,0,1)
  return (qnorm(uniform_random_number))
}

itstats <- function(N){
  samples <- c()
  for(i in 1:N){
    samples <- c(samples, normrandit())
  }
  return (c(mean(samples), sd(samples)))
}

# B: bm = box mueller transform
normrandbm <- function(){
  uniform_1 <- runif(1,0,1)
  uniform_2 <- runif(1,0,1)
  X <- sqrt((-2 * log(uniform_1)))*(cos(2 * pi * uniform_2))
  Y <- sqrt((-2 * log(uniform_1)))*(sin(2 * pi * uniform_2))

  return (c(X,Y))
}

bmstats <- function(N){
  samples <- c()
  for(i in 1:N){
    samples <- c(samples, normrandbm())
  }
  return (c(mean(samples), sd(samples)))
}

# C: ar = accept reject
normrandar <- function(){
  accepted <- FALSE
  X <- 0
  while(!accepted){
    uniform_1 <- runif(1,0,1)
    uniform_2 <- runif(1,0,1)
    X <- (-1 * log(uniform_1))
    Y <- (-1 * log(uniform_2))

    if(Y >= ((X-1)^2)/2){
      accepted <- TRUE
```

```
      pos_neg <- runif(1,0,1)
      if(pos_neg < 0.5){
        X <- (-1) * X
      }
    }
  }
  return (X)
}

arstats <- function(N){
  samples <- c()
  for(i in 1:N){
    samples <- c(samples, normrandar())
  }
  return (c(mean(samples), sd(samples)))
}

# D
for(x in c(1:10)){
  it_means <- c();
  it_sd <- c();
  it_tm <- c();
  bm_means <- c();
  bm_sd <- c();
  bm_tm <- c();
  ar_means <- c();
  ar_sd <- c();
  ar_tm <- c();

  num_runs_list <- c(100,1000,10000,100000)
  #num_runs_list <- c(100,1000,10000)

  for(runs in num_runs_list){
    start.time <- Sys.time()
    itst <- itstats(runs)
    end.time <- Sys.time()
    it_means <- c(it_means, itst[1])
    it_sd <- c(it_sd, itst[2])
    it_tm <- c(it_tm,(end.time - start.time))

    start.time <- Sys.time()
    bmst <- bmstats(runs)
    end.time <- Sys.time()
    bm_means <- c(bm_means, itst[1])
    bm_sd <- c(bm_sd, itst[2])
    bm_tm <- c(bm_tm,(end.time - start.time))

    start.time <- Sys.time()
    arst <- arstats(runs)
    end.time <- Sys.time()
    ar_means <- c(ar_means, itst[1])
    ar_sd <- c(ar_sd, itst[2])
    ar_tm <- c(ar_tm,(end.time - start.time))
```

```
  }
}
# E
library(knitr)
kable(df_e <- data.frame(num_runs_list,it_means,bm_means,ar_means,it_sd,bm_sd,ar_sd,it_tm,bm_tm,ar_tm))

library(ggplot2)
ggplot(df_e, aes(x=num_runs_list)) +
  geom_line(aes(y = it_means, colour = "blue"), position=position_jitter(w=0.005, h=0.005)) +
  geom_line(aes(y = bm_means, colour = "red"), position=position_jitter(w=0.005, h=0.005)) +
  geom_line(aes(y = ar_means, colour = "green"), position=position_jitter(w=0.005, h=0.005)) +
  ggtitle("MEANS (IT=Blue, BM=Red, AR=Green)")

ggplot(df_e, aes(x=num_runs_list)) +
  geom_line(aes(y = it_sd, colour = "blue"), position=position_jitter(w=0.005, h=0.005)) +
  geom_line(aes(y = bm_sd, colour = "red"), position=position_jitter(w=0.005, h=0.005)) +
  geom_line(aes(y = ar_sd, colour = "green"), position=position_jitter(w=0.005, h=0.005)) +
  ggtitle("STANDARD DEVIATIONS (IT=Blue, BM=Red, AR=Green)")

ggplot(df_e, aes(x=num_runs_list)) +
  geom_line(aes(y = it_tm, colour = "blue"), position=position_jitter(w=0.005, h=0.005)) +
  geom_line(aes(y = bm_tm, colour = "red"), position=position_jitter(w=0.005, h=0.005)) +
  geom_line(aes(y = ar_tm, colour = "green"), position=position_jitter(w=0.005, h=0.005)) +
  ggtitle("RUN TIMES (IT=Blue, BM=Red, AR=Green)")

num_iterations <- 100#0000
hist(itstats(num_iterations))
hist(bmstats(num_iterations))
hist(arstats(num_iterations))

inside_circle <- function(rand_x, rand_y){
  radius_length <- sqrt(rand_x * rand_x + rand_y * rand_y)
  if(radius_length <= 1){
    return (1.0)
  }else{
    return (0.0)
  }
}

estimatepi <- function(num_pairs){
  rand_x <- runif(num_pairs, 0, 1)
  rand_y <- runif(num_pairs, 0, 1)
  sum_below_circle <- 0
  for(i in 1:num_pairs){
    sum_below_circle <- sum_below_circle + inside_circle(rand_x[i], rand_y[i])
  }
  return (4*sum_below_circle/num_pairs)
}

estimatepi(1)
estimatepi(10)
estimatepi(100)
estimatepi(1000)
```

```
estimatepi(10000)
##
```