

# Experimentos usando Pi

[https://github.com/guilhermeivo/computing-projects/  
guilhermeivo](https://github.com/guilhermeivo/computing-projects/guilhermeivo)

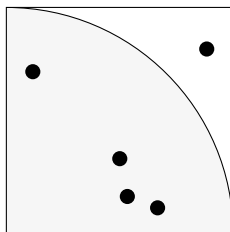
3 de fevereiro de 2025

Queremos implementar algoritmos para calcular o valor de  $\pi$  usando métodos computacionais.

## PI USANDO MONTE CARLO

O método de Monte Carlo é uma técnica de amostragem estatística, teve seu desencadeamento após o desenvolvimento do *ENIAC*<sup>1</sup>, primeiro computador eletrônico, desenvolvido durante a Segunda Guerra Mundial, para lidar com grandes cálculos. O método baseia-se em números “aleatórios” para resolver os seus problemas que é aplicada com sucesso em diversos problemas contemporâneos [2].

monteCarlo() O problema consiste em montar um quadrado de comprimento 1, e  $\frac{1}{4}$  de uma circunferência inscrita nesse quadrado e de forma aleatório disparar pontos dentro do espaço.

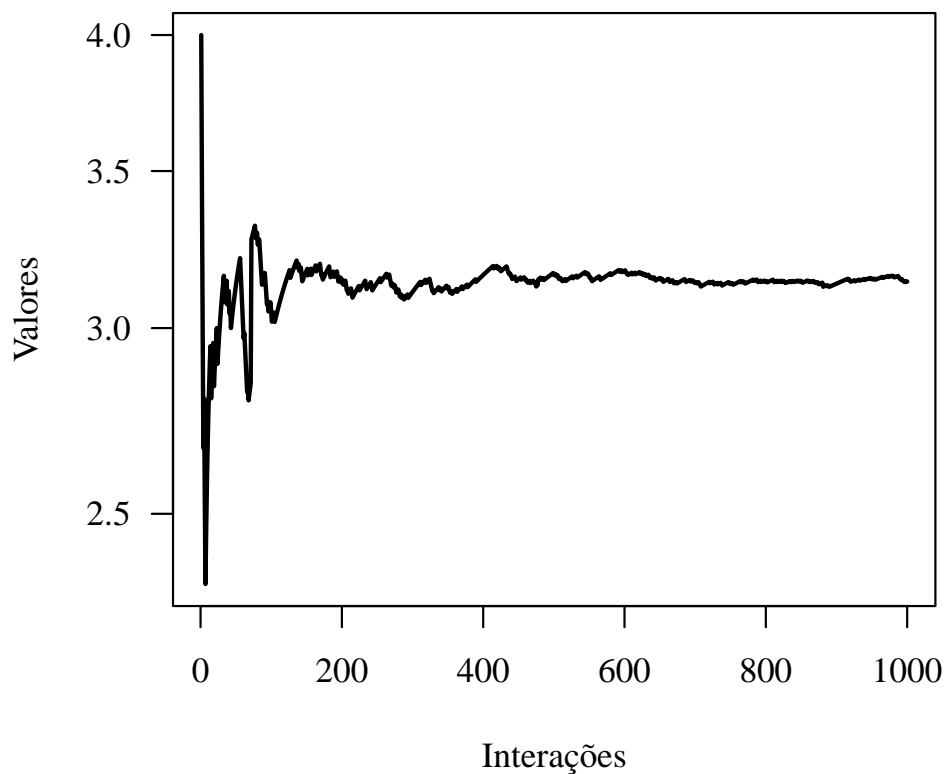


$$A = \pi \cdot R^2 \implies A = \pi$$
$$A_c = \frac{\pi}{4} \approx 4 \cdot \frac{\text{dentro}}{\text{total}} = \pi$$

```
long double monteCarlo(unsigned long long n) {  
    unsigned long long ins;  
    ins = 0;  
    long double x, y, z, pi;  
  
    srand(time(NULL));  
    unsigned long long i;  
    for (i = 0; i < n; i++) {  
        x = (long double) rand() / (long double) RAND_MAX;  
        y = (long double) rand() / (long double) RAND_MAX;
```

<sup>1</sup>O ENIAC produziu 2,037 dígitos de  $\pi$  em 70 horas em 1949 usando a *Fórmula de Machin*[1, p. 592 e 627]

```
    z = x * x + y * y;  
    if (z <= 1) {  
        ins++;  
    }  
}  
  
pi = (long double) 4 * ins;  
pi = (long double) pi / (long double) n;  
  
return pi;  
}
```

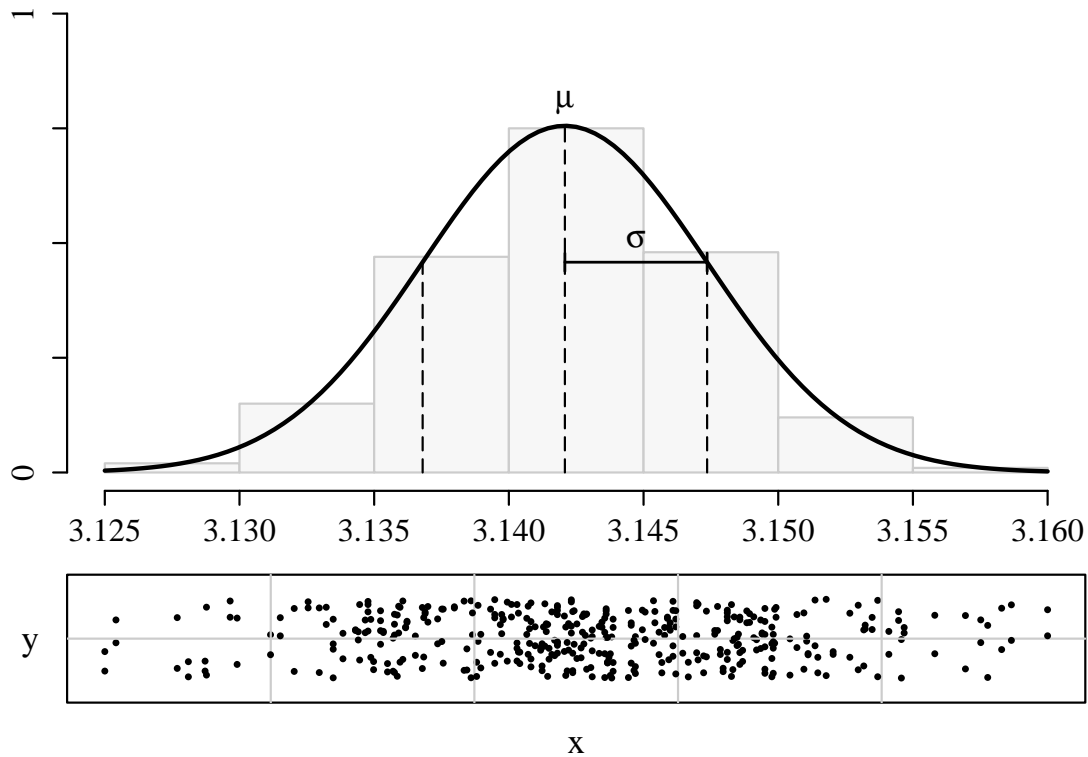
Figura 1: Convergência para o método de *Monte Carlo*

Fonte: Subseção B.1

Observando que os valores coletados na Figura 1 convergem com flutuações nos valores, pois os valores são obtidos de forma aleatória. Então, podemos interpretar estatisticamente este resultado realizando uma hipótese de normalidade na amostra.

Temos, um histograma:

Figura 2: Histograma dos valores obtidos no método de *Monte Carlo*, com curva de densidade, aplicado à coleta de  $\pi$



Fonte: Subseção B.2

Devido às propriedades do gráfico, temos:

- O ponto  $x = \mu$  é um ponto de máximo absoluto, sendo o único ponto crítico do gráfico, sendo representado pela linha pontilhada central da Figura 2
- Possui dois pontos de inflexão  $x = \mu - \sigma$  e  $x = \mu + \sigma$ , sendo representado pelas linhas pontilhadas nas extremidades do gráfico.
- A área entre  $\mu \pm \sigma$  representa 68% da área total do gráfico, ou seja, 68% dos valores de  $\pi$  estão presentes nesse intervalo.

Sendo:  $mean(var)$ , média aritmética dos valores de  $\pi$  obtidos.

$$var = \{x_1, x_2, \dots, x_n\}$$

$$\mu = \bar{var} = \frac{1}{n} \sum_{i=1}^n x_i = \frac{x_1 + x_2 + \dots + x_n}{n}$$

$sd(var)$ , desvio padrão dos valores obtidos, ou seja, medida de dispersão com as mesmas unidades que a amostra.

$$\sigma = sd = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{var})^2}$$

$sd(var)/sqrt(m)$ , desvio padrão da média ( $\bar{v}ar$ ), no limite para  $m$  grande. O desvio padrão da média ( $\xi$ ) caracteriza as flutuações dos valores  $\bar{v}ar_i$  em torno da média população  $\mu$ .

$$\xi = \frac{\sigma}{\sqrt{m}}$$

E realizando testes estatísticos nos dados para determinar se os dados afastam ou não da distribuição normal.

```
shapiro.test(var)
```

```
[1] Shapiro-Wilk normality test
```

```
data: var
```

```
W = 0.9987, p-value = 0.6886
```

Observando que o *valor – p* retornado pelo teste é maior que 0,05, fortalecendo a hipótese que os dados são normalmente distribuídos.

Além disso, tem-se:

```
library(nortest)
```

```
ad.test(var)
```

```
[1] Anderson-Darling normality test
```

```
data: var
```

```
A = 0.2726, p-value = 0.6684
```

```
ks.test(var, "pnorm", mean(var), sd(var))
```

```
[1] Asymptotic one-sample Kolmogorov-Smirnov test
```

```
data: var
```

```
D = 0.015002, p-value = 0.978
```

```
alternative hypothesis: two-sided
```

O *valor – p* em todos os testes anteriores foram superiores que 0,05, fortalecendo a nossa hipótese inicial de que a normalidade não é rejeitada, como demonstrado na Figura 2. Então, apresentando erros de medição aleatórios.

## DESENVOLVENDO UM CASO GERAL

**P**ara facilitar o desenvolvimento dos algoritmos para calcular arctg vamos realizar um caso geral, onde iremos adotar o valor para  $x$  sendo igual a  $1/a$ , pois em todas as fórmulas utilizadas é usado arctg do tipo arctg  $1/a$ . Então, podemos desenvolver a série da função arctg da seguinte maneira:

$$\begin{aligned}
 A \cdot \arctg x &= A \cdot \sum_{k=0}^n \left( (-1)^k \frac{x^{2k+1}}{2k+1} \right) + A \cdot R_n(x) \Rightarrow \\
 \Rightarrow A \cdot \arctg \left( \frac{1}{a} \right) &= A \cdot \sum_{k=0}^n \left( (-1)^k \frac{\left( \frac{1}{a} \right)^{2k+1}}{2k+1} \right) + A \cdot R_n \left( \frac{1}{a} \right) \\
 &= A \cdot \underbrace{\sum_{k=0}^n \left( (-1)^k \frac{1}{a^{2k+1}} \cdot \frac{1}{2k+1} \right)}_{P_n \left( \frac{1}{a} \right)} + A \cdot R_n \left( \frac{1}{a} \right)
 \end{aligned}$$

$$\begin{aligned}
 P_n \left( \frac{1}{a} \right) &= \frac{1}{a} - \frac{1}{3a^3} + \frac{1}{5a^5} - \frac{1}{7a^7} + \frac{1}{9a^9} - \frac{1}{11a^{11}} + \dots + \\
 &\quad + \frac{1}{(2n+1) \cdot a^{2n+1}} - \frac{1}{(2n+3) \cdot a^{2n+3}}
 \end{aligned}$$

$$\begin{aligned}
 &\frac{1}{(2n+1) \cdot a^{2n+1}} - \frac{1}{(2n+3) \cdot a^{2n+3}} = \\
 &= \frac{(2n+3) \cdot a^{2n+3} - (2n+1) \cdot a^{2n+1}}{(2n+1) \cdot a^{2n+1} \cdot (2n+3) \cdot a^{2n+3}} = \\
 &= \frac{(2n+3) \cdot a^2 - (2n+1)}{(2n+1) \cdot (2n+3) \cdot a^{2n+3}}, \text{ usando que } n = 2k, \text{ temos,} \\
 &= \frac{(4k+3) \cdot a^2 - (4k+1)}{(4k+1) \cdot (4k+3) \cdot a^{4k+3}}
 \end{aligned}$$

$$\begin{aligned}
 P_n \left( \frac{1}{a} \right) &= \sum_{k=0}^n \left( \frac{(4k+3) \cdot a^2 - (4k+1)}{(4k+1) \cdot (4k+3) \cdot a^{4k+3}} \right) \\
 &= \sum_{k=0}^n \left( \frac{(i+3) \cdot a^2 - (i+1)}{(i+1) \cdot (i+3) \cdot a^{i+3}} \right), i+=4
 \end{aligned}$$

```

generalArctg() {
1  long double generalArctg(unsigned long int n, unsigned long int a) {
2      unsigned long int pow, aux_pow, numerator, quotient;
3      long double res;

```

```

4
5     pow = a * a * a;
6     aux_pow = a * a * a * a;
7     res = 0;
8
9     int i;
10    for (i = 0; i < n; i+=4) {
11        numerator = (unsigned long int) (i + 3) * (a * a) - (i + 1);
12        quotient = (unsigned long int) (i + 1) * (i + 3) * pow;
13        res += (long double) numerator / quotient;
14
15        pow *= aux_pow;
16    }
17
18    return res;
19 }
20

```

$$\left| R_n \left( \frac{1}{a} \right) \right| \leq \frac{\left( \frac{1}{a} \right)^{2n+3}}{2n+3}$$

$$A \cdot \frac{1}{a^{2n+3}} \cdot \frac{1}{2n+3} < A \cdot \frac{1}{a^{2n+3}} \cdot \frac{1}{2 \cdot (n+1)} \stackrel{?}{<} \epsilon$$

$$\frac{1}{\epsilon} \cdot \frac{A}{2a} < a^{2 \cdot (n+1)} \cdot (n+1)$$

Como  $\epsilon = 10^{-d}$ , tem-se:  $10^d \cdot \frac{A}{2a} < a^{2 \cdot (n+1)} \cdot (n+1)$ , sendo  $n$  então a quantidade de iterações necessárias.

```

generalErrorHandle()
1  unsigned long int generalErrorHandle(unsigned long int d, unsigned
2      long int A, unsigned long int a) {
3      unsigned long int n, err, small, pow, aux_pow;
4
5      n = 0;
6      err = 1;
7
8      aux_pow = a * a;
9      pow = aux_pow;
10
11     int i;
12     for (i = 0; i < d; i++) {
13         err *= 10;
14     }
15

```

```

14
15     small = err * A / (2 * a);
16
17     while (small >= (unsigned long int) pow * (n + 1)) {
18         pow *= aux_pow;
19         n++;
20     }
21
22     return n;
23 }
24

```

Porém utilizando essas funções para  $\pi/4 = \text{arctg}(1)$  o maior erro que conseguimos alcançar, devido às limitações dos tipos utilizados (`unsigned long int` e `long double`) nas funções, é  $\epsilon = 10^{-9}$ . Então é necessário utilizar funções para cálculos de precisão arbitrária, sendo, nesse caso, utilizado a biblioteca *GNU MPFR*, uma biblioteca *C* para cálculos de ponto flutuante de precisão múltipla com arredondamento correto, é baseado na biblioteca *GNU Multiple Precision Arithmetic (GMP)*.

## USANDO MPFR E GMP

\*

`arctg_mpfr()` Substituindo os tipos tradicionais pelos utilizados nas bibliotecas, temos:

```

1 void generalArctg_mpfr(const mpz_t n, unsigned long int a, fp_t res)
  {
2     fp_set_ui(res, 0);
3
4     mpz_t i, i_1, i_3, aux_pow_4, aux_pow_2, pow, a_gmp, numerator,
        quotient;
5     mpz_inits(i, i_1, i_3, aux_pow_4, aux_pow_2, pow, a_gmp,
        numerator, quotient, NULL);
6
7     mpz_set_ui(a_gmp, a);
8     mpz_pow_ui(aux_pow_4, a_gmp, 4); // aux = a^4
9     mpz_pow_ui(aux_pow_2, a_gmp, 2); // aux = a^2
10    mpz_pow_ui(pow, a_gmp, 3); // pow = a^3
11
12    fp_t frac, numerator_mpfr, quotient_mpfr;
13    fp_inits(frac, numerator_mpfr, quotient_mpfr, NULL);
14
15    for (mpz_set_ui(i, 0); mpz_cmp(i, n) < 0; mpz_add_ui(i, i, 4)) {
        // for (int i = 0; i < n; i+=4) {...}
16        mpz_add_ui(i_1, i, 1); // i + 1
    }

```

```

17     mpz_add_ui(i_3, i, 3); // i + 3
18
19     // numerator
20     mpz_mul(numerator, i_3, aux_pow_2); // (i + 3) * a^2
21     mpz_sub(numerator, numerator, i_1); // (i + 3) * a^2 - (i +
        1)
22
23     // quotient
24     mpz_mul(quotient, i_1, i_3); // (i + 1) * (i + 3)
25     mpz_mul(quotient, quotient, pow); // (i + 1) * (i + 3) * a^{i
        +3}
26
27     fp_set_z(numerator_mpfr, numerator);
28     fp_set_z(quotient_mpfr, quotient);
29
30     fp_div(frac, numerator_mpfr, quotient_mpfr); // frac =
        numerator / quotient
31     fp_add(res, res, frac); // res += frac
32
33     mpz_mul(pow, pow, aux_pow_4); // pow *= a^4
34 }
35 mpz_clears(i, i_1, i_3, aux_pow_4, aux_pow_2, pow, a_gmp, NULL);
36
37 fp_clears(frac, numerator_mpfr, quotient_mpfr, NULL);
38 }
39

```

rrorHandle\_gmp()

```

1 void generalErrorHandle_gmp(unsigned long int d, unsigned long int A,
    unsigned long int a, mpz_t n) {
2     mpz_set_ui(n, 0);
3
4     mpz_t err, aux_pow, pow, aux_cmp;
5     mpz_inits(err, aux_pow, pow, aux_cmp, NULL);
6     mpz_set_ui(err, 1);
7
8     for (int i = 0; i < d; i++) {
9         mpz_mul_ui(err, err, 10);
10    }
11
12    mpz_mul_ui(err, err, A);
13    mpz_div_ui(err, err, 2 * a);
14
15    mpz_set_ui(aux_pow, a);
16    mpz_set(pow, aux_pow);
17
18    mpz_set(aux_cmp, pow);

```



```

19     mpz_mul(aux_cmp, aux_cmp, n);
20     while (mpz_cmp(err, aux_cmp) > 0) { // err >= aux_cmp
21         mpz_mul(pow, pow, aux_pow);
22         mpz_set(aux_cmp, pow);
23         mpz_add_ui(n, n, 1);
24         mpz_mul(aux_cmp, aux_cmp, n);
25     }
26
27     mpz_clears(err, aux_pow, pow, aux_cmp, NULL);
28 }
29

```

## FÓRMULA DE *MACHIN* ORIGINAL

\*

`machin()` Função `machin` resolvendo a *Fórmula de Machin Original*:

$$\frac{\pi}{4} = 4 \cdot \operatorname{arctg} \frac{1}{5} - \operatorname{arctg} \frac{1}{239}$$

```

1 void machin(unsigned long long err, fp_t pi) {
2     fp_t arctg_5;
3     generalArctgWithError_mpfr(arctg_5, err, 4, 5);
4
5     fp_t arctg_239;
6     generalArctgWithError_mpfr(arctg_239, err, 1, 239);
7
8     fp_sub(pi, arctg_5, arctg_239);
9     fp_mul_ui(pi, pi, 4);
10
11     fp_clears(arctg_5, arctg_239, NULL);
12 }
13

```

## FÓRMULA DE *KIKUO TAKANO*

\*

`takano()` Função `takano` resolvendo a *Fórmula de Kikuo Takano*:

$$\frac{\pi}{4} = 12 \cdot \operatorname{arctg} \frac{1}{49} + 32 \cdot \frac{1}{57} - 5 \cdot \frac{1}{239} + 12 \cdot \operatorname{arctg} \frac{1}{110.443} \quad (\text{K. Takano, 1982})$$

```

1 void takano(unsigned long long err, fp_t pi) {
2     fp_t arctg_49;
3     generalArctgWithError_mpfr(arctg_49, err, 12, 49);
4
5     fp_t arctg_57;
6     generalArctgWithError_mpfr(arctg_57, err, 32, 57);
7
8     fp_t arctg_239;
9     generalArctgWithError_mpfr(arctg_239, err, 5, 239);
10
11    fp_t arctg_110443;
12    generalArctgWithError_mpfr(arctg_110443, err, 12, 110443);
13
14    fp_add(pi, arctg_49, arctg_57);
15    fp_sub(pi, pi, arctg_239);
16    fp_add(pi, pi, arctg_110443);
17    fp_mul_ui(pi, pi, 4);
18
19    fp_clears(arctg_49, arctg_57, arctg_239, arctg_110443, NULL);
20 }
21

```

## FÓRMULA DE *FREDRIK CARL MÜLERTZ STØRMER*

\*

stormer() Função stormer resolvendo a *Fórmula de Fredrik Carl Mülertz Størmer*:

$$\frac{\pi}{4} = 44 \cdot \arctg \frac{1}{57} + 7 \cdot \frac{1}{239} - 12 \cdot \frac{1}{682} + 24 \cdot \arctg \frac{1}{12.943} \quad (\text{F. C. M. Størmer, 1896})$$

```

1 void stormer(unsigned long long err, fp_t pi) {
2     fp_t arctg_57;
3     generalArctgWithError_mpfr(arctg_57, err, 44, 57);
4
5     fp_t arctg_239;
6     generalArctgWithError_mpfr(arctg_239, err, 7, 239);
7
8     fp_t arctg_682;
9     generalArctgWithError_mpfr(arctg_682, err, 12, 682);
10
11    fp_t arctg_12943;
12    generalArctgWithError_mpfr(arctg_12943, err, 24, 12943);
13
14    fp_add(pi, arctg_57, arctg_239);

```

```
15     fp_sub(pi, pi, arctg_682);
16     fp_add(pi, pi, arctg_12943);
17     fp_mul_ui(pi, pi, 4);
18
19     fp_clears(arctg_57, arctg_239, arctg_682, arctg_12943, NULL);
20 }
21
```

## REFERÊNCIAS

- [1] BERGGREN, L.; BORWEIN, J.; BORWEIN, P. **Pi**: a source book. Nova Iorque: Springer Science, 1997. ISBN 978-1-4757-2738-8.
- [2] ULAM, S. **Special Issue**. 15, [S. l.]: Los Alamos Science, 1987.

## **APÊNDICES**

## A VALOR DE PI

Sendo,  $\pi_n$ ,  $n$  a quantidade de casas decimais, temos:

$\pi_{100} = 3.14159\ 26535\ 89793\ 23846\ 26433\ 83279\ 50288\ 41971\ 69399\ 37510\ 58209\ 74944$   
 $59230\ 78164\ 06286\ 20899\ 86280\ 34825\ 34211\ 70679$

## B CÓDIGO FONTE E DADOS

### B.1 CONVERGÊNCIA PARA O MÉTODO DE *Monte Carlo*

Código 1: Código fonte para a convergência para o método de *Monte Carlo*

```
1 par(family = "serif")
2
3 pdf("generated/plot_monteCarlo.pdf", width = 5.6, height = 4)
4 dat <- read.table("generated/pi_monteCarlo_convergence.dat")
5 par(mar=c(5, 5, 1, 5))
6 plot(dat$V1, dat$V2,
7       type = "l",
8       xlab = "Interações",
9       ylab = "Valores",
10      las = 1,
11      lwd = 2,
12      log = "y",
13      family = "serif")
14 dev.off()
```

## B.2 HISTOGRAMA DOS VALORES OBTIDOS NO MÉTODO DE *MONTÉ CARLO*

Código 2: Código fonte para o histograma dos valores obtidos no método de Monte Carlo

```
1 par(family = "serif")
2
3 dat <- read.table("generated/pi_monteCarlo_histogram.dat")
4 var <- dat$V2
5
6 m <- mean(var)
7 s <- sd(var)
8 #e <- sd(var) / sqrt(m)
9
10 pdf("generated/histogram_monteCarlo.pdf", height = 4, width = 5.6)
11
12 layout(matrix(c(1, 1, 1,
13                 1, 1, 1,
14                 1, 1, 1,
15                 2, 2, 2), nrow=4, byrow=TRUE))
16
17 fun <- dnorm(var, mean = m, sd = s)
18
19 par(mar=c(3, 3, 2, 3))
20 histogram <- hist(var,
21   prob = TRUE,
22   ylim = c(0, 100),
23   col = c("gray97"),
24   border = c("gray80"),
25   axes = FALSE,
26   ylab = "", xlab = "", main=NULL, family = "serif")
27
28 xfit <- seq(min(var), max(var), by = (max(var) - min(var)) / 3)
29
30 axis(side = 1, family = "serif", cex.axis = 1.35)
31 axis(side = 2, at = c(0,25,50,75,100), labels = c(0,"",""," ",1),
32   family = "serif", cex.axis = 1.35)
33
34 curve(dnorm(x, mean = m, sd = s),
35   col = "black",
36   lwd = 2,
37   yaxt = "n",
38   add = TRUE)
39
40 segments(m, 0, m, dnorm(m, m, s), col = "black", lty = "longdash",
41   lwd = 1)
```



```

41 text(m, dnorm(m, m, s) + 5, expression(mu), family = "serif", cex =
    1.35)
42 segments(m + s, 0, m + s, dnorm(m + s, m, s), col = "black", lty = "
    longdash", lwd = 1)
43 segments(m - s, 0, m - s, dnorm(m - s, m, s), col = "black", lty = "
    longdash", lwd = 1)
44
45 segments(m, dnorm(m + s, m, s), m + s, dnorm(m + s, m, s), col = "
    black", lty = "solid", lwd = 1.25)
46 segments(m, dnorm(m + s, m, s) - 2, m, dnorm(m + s, m, s) + 2, col =
    "black", lty = "solid", lwd = 1.25)
47 segments(m + s, dnorm(m + s, m, s) - 2, m + s, dnorm(m + s, m, s) +
    2, col = "black", lty = "solid", lwd = 1.25)
48 text((m + m + s) / 2, dnorm(m + s, m, s) + 5, expression(sigma),
    family = "serif", cex = 1.35)
49
50 strip_chart <- function(add = FALSE) {
51     chart <- stripchart(var, method = "jitter",
52         ylab = "", xlab = "", xaxt="n", ylim = c(0.85, 1.15), pch =
            16,
53         family = "serif", add = add)
54 }
55
56 par(mar=c(3, 3, 0, 3))
57 strip_chart()
58 grid(nx = 5, ny = 2, lty = "solid", lwd = 1, col = "gray80")
59 mtext("x", side = 1, line = 1, las = 1, family = "serif", cex = 1)
60 mtext("y", side = 2, line = 1, las = 1, family = "serif", cex = 1)
61 strip_chart(add = TRUE)
62
63 dev.off()

```