Introdução à Física Computacional II (4300318)

Prof. André Vieira
apvieira@if.usp.br
Sala 3120 – Edifício Principal

Aula 8

Geradores de números aleatórios uniformes
Probabilidades e moedas viciadas

Processos aleatórios

- Na física quântica a aleatoriedade é um ingrediente intrínseco a cada processo de medida, que envolve o “colapso da função de onda”.
- Na física clássica, não há processos verdadeiramente aleatórios, uma vez que as leis clássicas fundamentais são todas determinísticas. Mas há processos muito complicados, cujos detalhes não podemos descrever inteiramente. Buscamos então representar efetivamente esses detalhes introduzindo ingredientes aleatórios. Um exemplo é o **movimento browniano**.

http://physics.bu.edu/~duffy/HTML5/brownian_motion.html

Números (pseudo)aleatórios

- Podemos obter números genuinamente aleatórios acompanhando um experimento em que efeitos quânticos são fundamentais, como o decaimento de átomos radioativos. (Veja por exemplo aqui.)
- Há entretanto vários algoritmos para produzir números que parecem aleatórios (ao menos à primeira vista). Vamos descrever um dos tipos mais simples de algoritmo, mas para aplicações “sérias” vamos utilizar algoritmos mais sofisticados.

Geradores de números pseudoaleatórios

- A classe de *geradores lineares congruenciais* de números pseudoaleatórios é definida pela recursão

$$x_{j+1} = (a x_j + c) \bmod m,$$

com a, c e m números inteiros, e "mod" a operação de resto de uma divisão. Dado um número inteiro x_j, a recursão produz um novo número x_{j+1}, e a sequência dos números produzidos recursivamente pode lembrar números aleatórios.

Geradores de números pseudoaleatórios

$$x_{j+1} = (a x_j + c) \bmod m,$$

Vamos testar com

$$a = 4,\ c = 1,\ m = 9,\ x_0 = 3.$$

Temos

$$x_1 = (4 \times 3 + 1) \bmod 9 = 13 \bmod 9 = 4,$$
$$x_2 = (4 \times 4 + 1) \bmod 9 = 17 \bmod 9 = 8,$$
$$x_3 = (4 \times 8 + 1) \bmod 9 = 33 \bmod 9 = 6,$$
$$x_4 = 7,\ x_5 = 2,\ x_6 = 0,\ x_7 = 1,\ x_8 = 5$$
$$x_9 = (4 \times 5 + 1) \bmod 9 = 21 \bmod 9 = 3 = x_0$$

Geradores de números pseudoaleatórios

$$x_{j+1} = (a x_j + c) \bmod m,$$

Vamos testar com

$$a = 4,\ c = 1,\ m = 9,\ x_0 = 3.$$

Temos

$$x_1 = (4 \times 3 + 1) \bmod 9 = 13 \bmod 9 = 4,$$
$$x_2 = (4 \times 4 + 1) \bmod 9 = 17 \bmod 9 = 8,$$
$$x_3 = (4 \times 8 + 1) \bmod 9 = 33 \bmod 9 = 6,$$
$$x_4 = 7,\ x_5 = 2,\ x_6 = 0,\ x_7 = 1,\ x_8 = 5$$
$$x_9 = (4 \times 5 + 1) \bmod 9 = 21 \bmod 9 = 3 = x_0$$

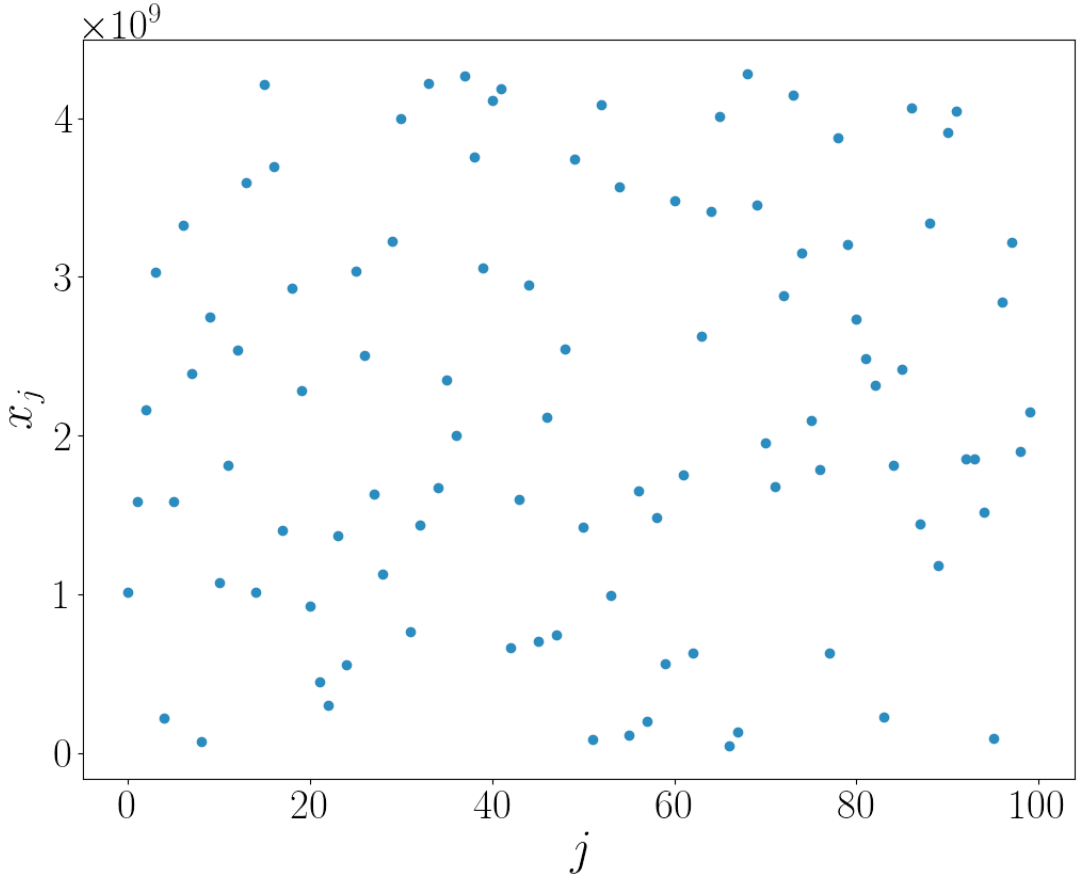
O período do gerador é no máximo igual a m.

Exemplo 1

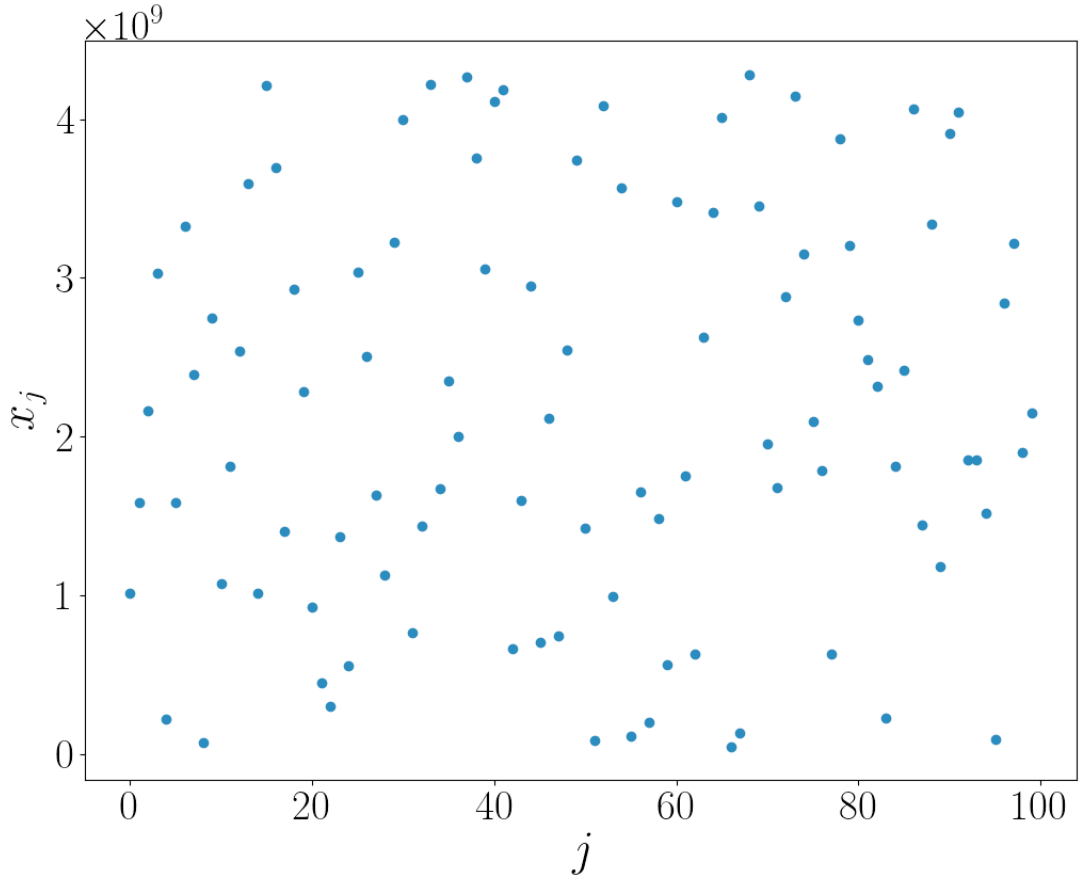
$$x_{j+1} = (a x_j + c) \bmod m,$$

Mas se utilizarmos $x_0 = 1$ com

$$a = 1\ 664\ 525,\quad c = 1\ 013\ 904\ 223,\quad m = 4\ 294\ 967\ 296$$



Exemplo 1



Embora a sequência de números pareça aleatória, ela é produzida de forma determinística, e o mesmo valor inicial de x (que é chamado de **semente**) produz sempre a mesma sequência de números.

Quantificando o grau de pseudoaleatoriedade

- Falamos até aqui de sequências que “parecem” aleatórias. Como podemos tornar essa ideia menos qualitativa?
- Uma forma é investigar as **correlações** entre os números da sequência. Números aleatórios independentes não devem exibir correlações, ou seja, o aparecimento de um número na sequência não deve afetar em que intervalo do conjunto dos inteiros os próximos números estarão.

Exemplo 2

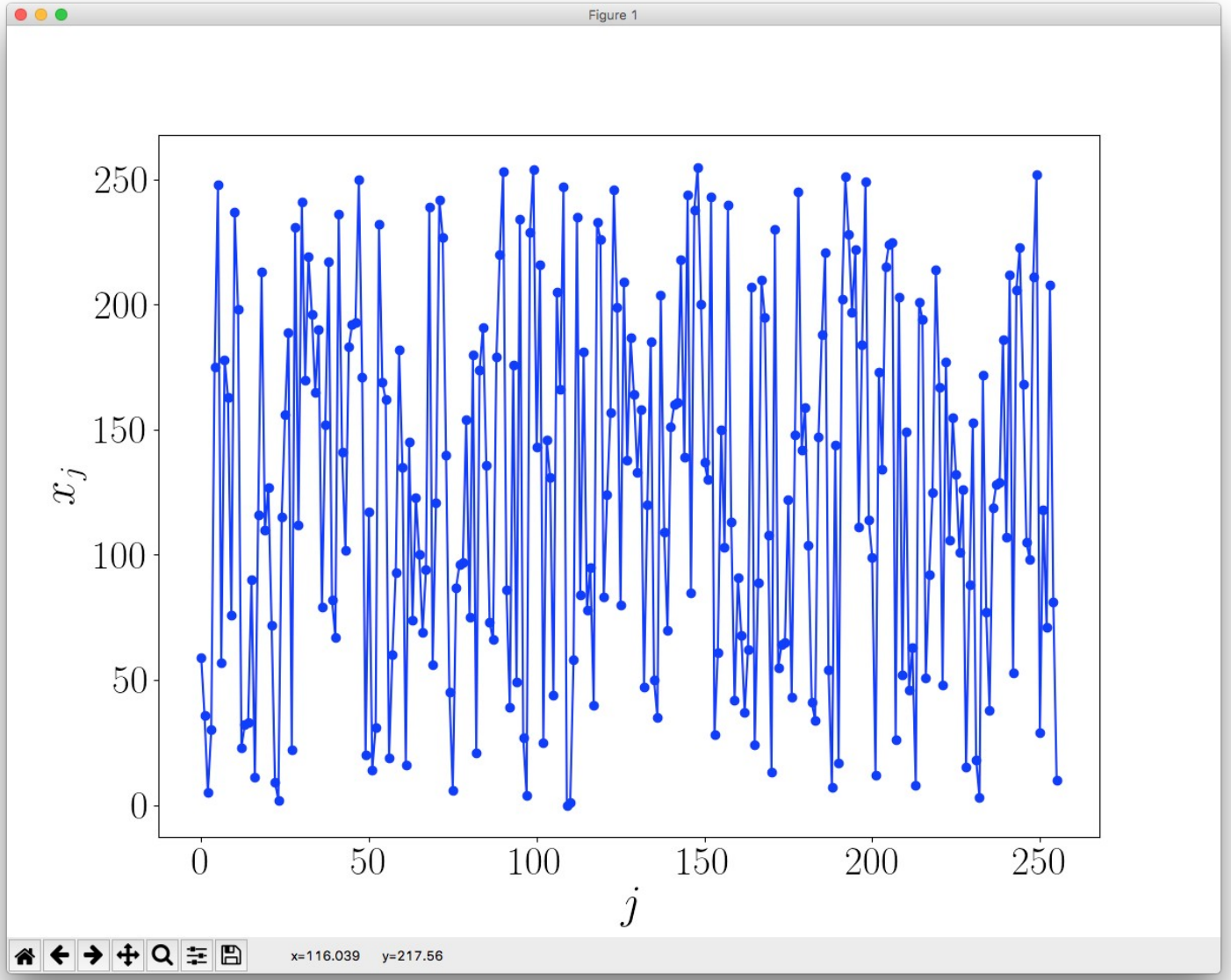
$$x_{j+1} = (a x_j + c) \bmod m,$$

$a = 57$

$c = 1$

$m = 256$

$x_0 = 10$



À primeira vista, a sequência parece bastante aleatória.

Exemplo 2

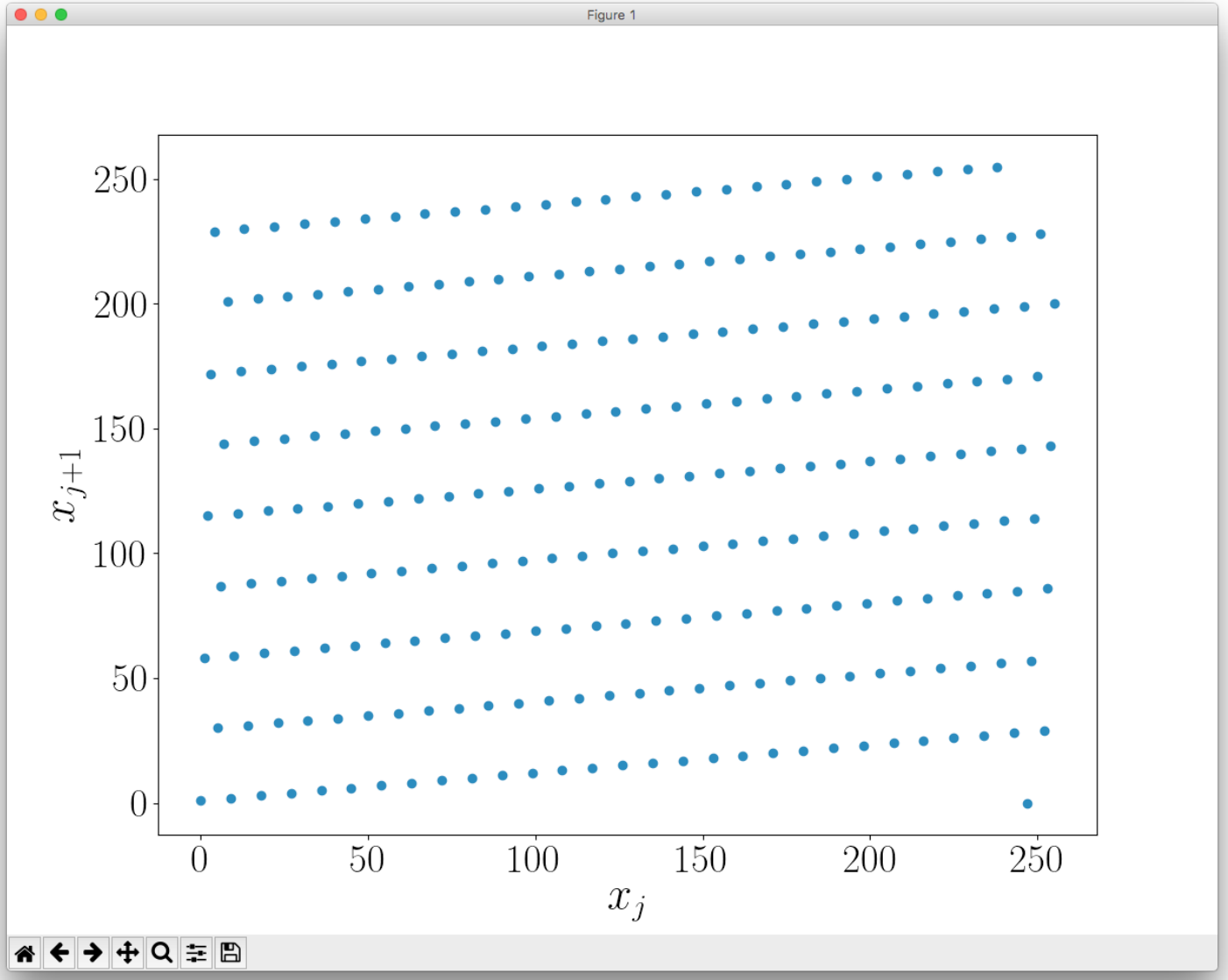
$$x_{j+1} = (a x_j + c) \bmod m,$$

$a = 57$

$c = 1$

$m = 256$

$x_0 = 10$



Mas traçando x_{j+1} contra x_j vemos que há claras correlações.

Exemplo 3

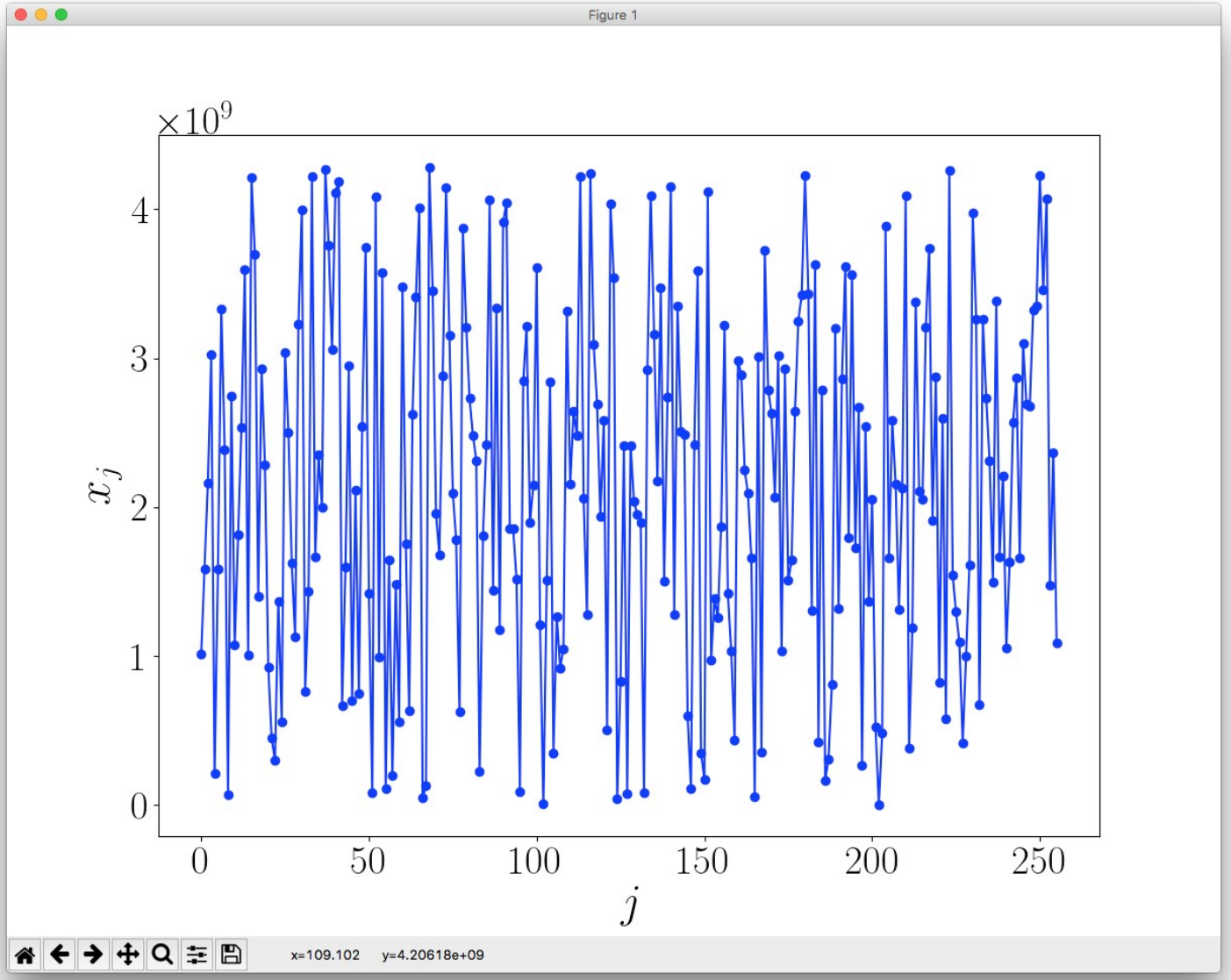
$$x_{j+1} = (a x_j + c) \bmod m,$$

$a = 1664525$

$c = 1013904223$

$m = 4294967296$

$x_0 = 1$



À primeira vista, a sequência parece bastante aleatória.

Exemplo 3

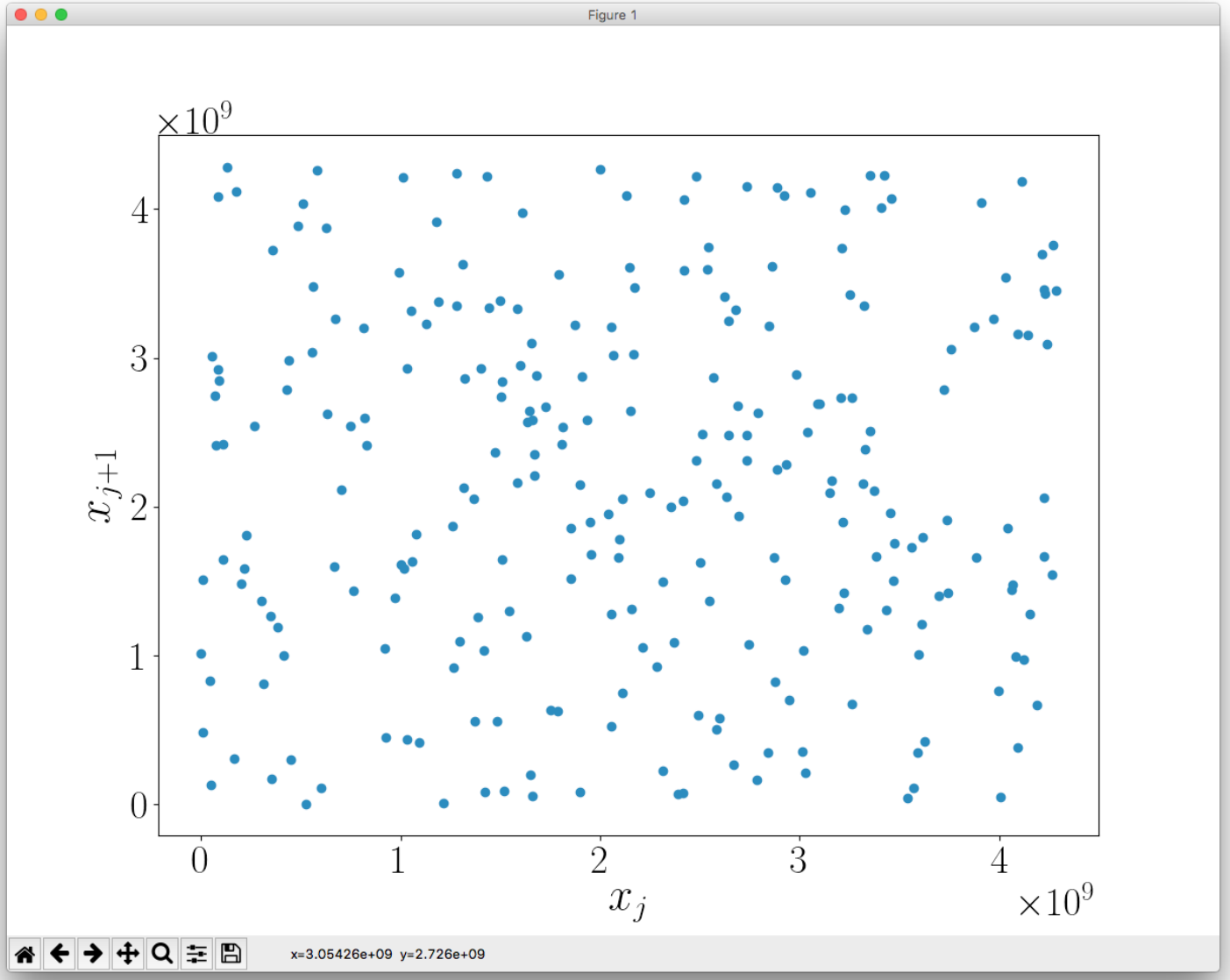
$$x_{j+1} = (a x_j + c) \bmod m,$$

$a = 1664525$

$c = 1013904223$

$m = 4294967296$

$x_0 = 1$



E traçando x_{j+1} contra x_j também não vemos claras correlações.

Distribuições uniformes

- Até aqui trabalhamos com números inteiros. Se quisermos utilizar nosso gerador para produzir números pseudoaleatórios no intervalo $[0,1)$, basta dividir cada número da sequência por m.
- Se utilizarmos bons parâmetros, devemos obter para os valores produzidos uma distribuição que emula uma densidade de probabilidades uniforme:

$$p(x) = \begin{cases} 0 \text{ se } x<0 \text{ ou } x \geq 1, \\ 1 \text{ se } 0 \leq x < 1. \end{cases}$$

$$P(x_1 \leq x \leq x_2) = \int_{x_1}^{x_2} p(x)\,dx = x_2 - x_1, \quad \text{se} \quad x_1, x_2 \in [0,1)$$

Números pseudoaleatórios em Python

- O gerador que discutimos pode ser razoável para aplicações simples, mas o pacote `random` do Python oferece um gerador bem melhor, o "entortador" de Mersenne, que tem período $2^{19937}-1$ e é o mais utilizado hoje em aplicações científicas.
- Funções úteis disponíveis no pacote `random`:
  - `random()`, que fornece um número real uniformemente distribuído entre 0 e 1;
  - `randrange(n)`, que fornece um inteiro aleatório entre 0 e $n-1$;
  - `randrange(m,n,k)`, que fornece um inteiro aleatório entre m e $n-1$, com intervalo k.

Exemplo 4

- Ao invocar essas funções, um novo número pseudoaleatório é gerado a cada vez.

```
from random import randrange

print("Vamos gerar um número aleatório.")
z = randrange(10)
print("Um segundo número aleatório é",randrange(10))
print("O primeiro número aleatório foi",z)
```

```
Vamos gerar um número aleatório.
Um segundo número aleatório é 8
O primeiro número aleatório foi 1

Vamos gerar um número aleatório.
Um segundo número aleatório é 0
O primeiro número aleatório foi 7
```

- Note que <u>não</u> é necessário informar uma semente para produzir sequências distintas de números aleatórios. Se quisermos forçar uma mesma sequência (para testar um programa), devemos utilizar a função `seed(semente)`.

Exemplo 5

```
from random import randrange,seed

seed(181) # Diferentes argumentos geram diferentes sequências
print("Vamos gerar um número aleatório.")
z = randrange(10)
print("Um segundo número aleatório é",randrange(10))
print("O primeiro número aleatório foi",z)
```

```
 RESTART: /Users/apvieira/Dropbox/disciplinas/4300318/2020/aulas/aula08/exemplo5.py
Vamos gerar um número aleatório.
Um segundo número aleatório é 1
O primeiro número aleatório foi 9
>>>
 RESTART: /Users/apvieira/Dropbox/disciplinas/4300318/2020/aulas/aula08/exemplo5.py
Vamos gerar um número aleatório.
Um segundo número aleatório é 1
O primeiro número aleatório foi 9
```

Probabilidades e “moedas viciadas”

- Em muitas aplicações, queremos descrever fenômenos que acontecem com uma certa probabilidade p.
- Por exemplo, podemos impor que uma partícula mova-se com probabilidade 0.6 ou fique em repouso com probabilidade 0.4. Isso seria equivalente a lançar uma moeda viciada que produzisse “cara” com probabilidade 0.6 ou “coroa” com probabilidade 0.4.

```
from random import random
if random() < 0.6:
   print("Cara")
else:
   print("Coroa")
```

Probabilidades e “moedas viciadas”

```
from random import random
if random() < 0.6:
   print("Cara")
else:
   print("Coroa")
```

- Note que a função `random()` produz um número pseudoaleatório uniformemente distribuído entre 0 e 1, logo a probabilidade de que ela retorne um número menor que 0.6 é justamente 0.6, e a probabilidade de que ela retorne um número maior que 0.6 é 0.4. Em média, o programa irá imprimir `"Cara"` 60% das vezes e `"Coroa"` 40% das vezes, como pretendíamos.

Exemplo 6

- Um processo de decaimento radioativo envolve radioisótopos de tálio (Tl) 208, com uma meia-vida de $3{,}053$ minutos, que decaem para átomos estáveis de chumbo (Pb) 208.
- Pela definição de meia-vida, o número de átomos de tálio nesse processo varia segundo

$$N(t) = N(0)\,2^{-t/\tau},$$

em que τ é a meia-vida e t é o tempo transcorrido desde que o número de átomos de tálio era $N(0)$.

Exemplo 6

$$N(t) = N(0)\,2^{-t/\tau}$$

- Entre os instantes t e $t+\Delta t$, a fração restante de átomos diminui de

$$\frac{N(t) - N(t+\Delta t)}{N(t)} = 1 - 2^{-\Delta t/\tau}.$$

- Portanto, a probabilidade de que um átomo decaia durante um tempo Δt é

$$p = 1 - 2^{-\Delta t/\tau}.$$

O exemplo a seguir simula esse processo.

Exemplo 6

$$N(0) = 1000$$
$$\tau = 3.053 \times 60\,\mathrm{s}$$
$$\Delta t = 1\,\mathrm{s}$$

$$p = 1 - 2^{-\Delta t/\tau}$$
$$\simeq 3.78 \times 10^{-3}$$

```
from random import random
from numpy import arange
import matplotlib.pyplot as plt

# Constantes
NTl = 1000          # Número inicial de átomos de tálio
NPb = 0             # Número inicial de átomos de chumbo
tau = 3.053*60      # Meia-vida do tálio em segundos
h = 1.0             # Tamanho do passo de tempo em segundos
p = 1 - 2**(-h/tau) # Probabilidade de decaimento em um passo
tmax = 1000         # Tempo total da simulação

# Criando as listas para os gráficos
t_lista = arange(0.0,tmax,h)
Tl_lista, Pb_lista = [], []

# Laço principal
for t in t_lista:
    Tl_lista.append(NTl)
    Pb_lista.append(NPb)

    # Dando a cada átomo a chance de decair
    decaimento = 0
    for i in range(NTl):
        if random() < p:
            decaimento += 1
    NTl -= decaimento
    NPb += decaimento
```

Exemplo 6

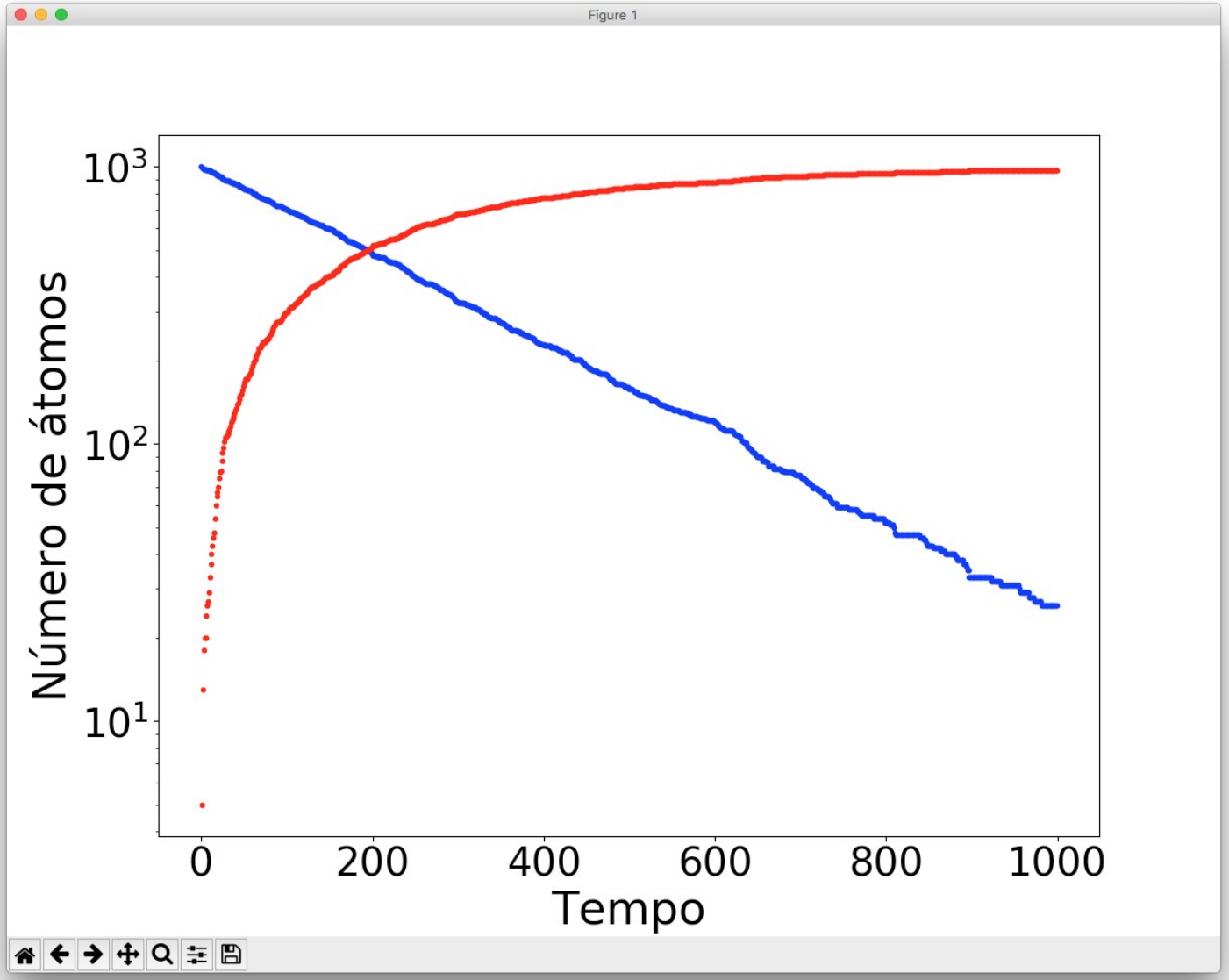
$N(0) = 1000$

$\tau = 3.053 \times 60\,\mathrm{s}$

$\Delta t = 1\,\mathrm{s}$

$p = 1 - 2^{-\Delta t/\tau}$

$\simeq 3.78 \times 10^{-3}$



Número de átomos de tálio em **azul**, número de átomos de **chumbo** em vermelho. Note a escala logarítmica das ordenadas.

Exemplo 7

- Um modelo simplificado para o movimento browniano é uma **caminhada aleatória** de uma partícula que vive em um reticulado quadrado, com um parâmetro de rede unitário e contendo L células em cada direção. A cada passo de tempo, a partícula move-se aleatoriamente para cima, para baixo, para a esquerda ou para a direita.
- O exemplo a seguir simula esse processo e permite observar as trajetórias aleatórias da partícula.

Exemplo 7a

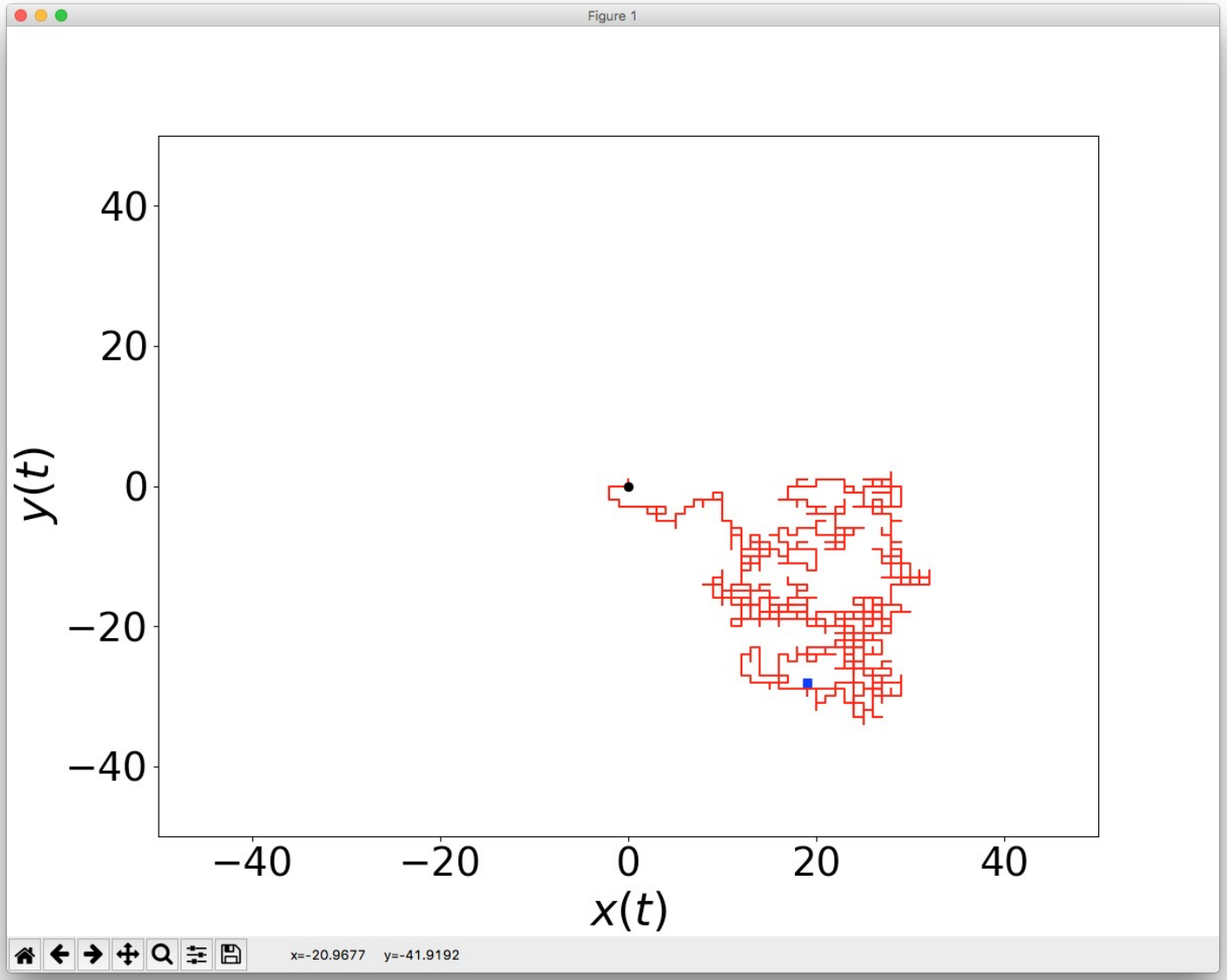
```
from random import randrange
from numpy import arange
import matplotlib.pyplot as plt

# Constantes
L = 100     # L+1, com L par, é o lado da rede quadrada
N = 10**3   # Número de passos da simulação

# Criando listas para armazenar a trajetória
x_lista, y_lista = [], []

# A partícula sai da origem
x, y = 0, 0
n = 0
while n<N:
    x_lista.append(x)
    y_lista.append(y)
    z = randrange(4)# Sorteamos uma de quatro direções
    if z == 0:
        dr = [1,0]  # Passo para a direita
    if z == 1:
        dr = [0,1]  # Passo para cima
    if z == 2:
        dr = [-1,0] # Passo para a esquerda
    if z == 3:
        dr = [0,-1] # Passo para baixo
    # Não permitimos que a partícula saia da rede
    if (abs(x+dr[0])<L/2) and (abs(y+dr[1])<L/2):
        x += dr[0]
        y += dr[1]
        n += 1
```

Exemplo 7a



A partícula sai do ponto circular **preto** e após 1000 passos está no ponto quadrado **azul**.

Exemplo 7b: animação

```
from random import randrange,seed
from numpy import arange,empty
import matplotlib.pyplot as plt
import matplotlib.animation as animation

# Constantes
L = 100     # L+1, com L par, é o lado da rede quadrada
N = 10**6   # Número de passos da simulação

# Criando listas para armazenar a trajetória
data = empty((2,N),dtype=int)

# A partícula sai da origem
x, y = 0, 0
n = 0
seed(52)
while n<N:
    data[0,n]=x
    data[1,n]=y
    z = randrange(4)# Sorteamos uma de quatro direções
    if z == 0:
        dr = [1,0]  # Passo para a direita
    if z == 1:
        dr = [0,1]  # Passo para cima
    if z == 2:
        dr = [-1,0] # Passo para a esquerda
    if z == 3:
        dr = [0,-1] # Passo para baixo
    # Não permitimos que a partícula saia da rede
    if (abs(x+dr[0])<L/2) and (abs(y+dr[1])<L/2):
        x += dr[0]
        y += dr[1]
        n += 1
```

```
# Parâmetros da exibição dos gráficos
plt.rcParams['xtick.labelsize'] = 28
plt.rcParams['ytick.labelsize'] = 28
plt.rcParams['axes.labelsize'] = 32

# Atualização do quadro da animação, que contém todos
# os passos, representados como linhas, até o passo n.
def atualiza_linha(n, data, line):
    line.set_data(data[..., :n])
    return line,

# Traçando o gráfico
fig = plt.figure(figsize=(12,9))
l, = plt.plot([], [], 'r-')
plt.xlim(-L/2,L/2)
plt.ylim(-L/2,L/2)
plt.xlabel("$x(t)$")
plt.ylabel("$y(t)$")
linha_ani = animation.FuncAnimation(fig, \
                                    atualiza_linha, \
                                    fargs=(data, l), \
                                    interval=10,\
                                    blit=True)
plt.show()
```

A construção `data[…, :n]` pega ambas as linhas da matriz `data` até a coluna `n-1`, sendo `n` o passo da simulação.

Exercícios no moodle

- Há dois exercícios, explorando o conteúdo da aula de hoje, que podem ser feitos com base nos programas dos exemplos, disponíveis no moodle.

A data para entrega é o dia **13 de maio**.