Histogram

1. V Pythonu vytvořte histogram z naměřených hodnot uložených v souboru `data.dat`
   Nalezněte optimální šířku binu.

Histogram

1. V Pythonu vytvořte histogram z naměřených hodnot uložených v souboru `data.dat` Nalezněte optimální šířku binu.

Práce s histogramy v Pythonu:

vytvoření histogramu

```
hist,bin_edges=np.histogram(data,bins=nbins,density='True')
```

pole obsahující hranice binů

pole obsahující počty hodnot v jednotlivých binech

data, z kterých se má udělat histogram

počet binů

normalizace histogramu

Histogram

histogram.py

```
import numpy as np
import matplotlib.pyplot as plt

nbins=100 #pocet binu
data=np.loadtxt('data.dat') #nacteni dat ze souboru data.dat
hist,bin_edges=np.histogram(data,bins=nbins) #vytvoreni histogramu

plt.step(bin_edges[0:nbins],hist) #nakresleni histogramu
```

Histogram

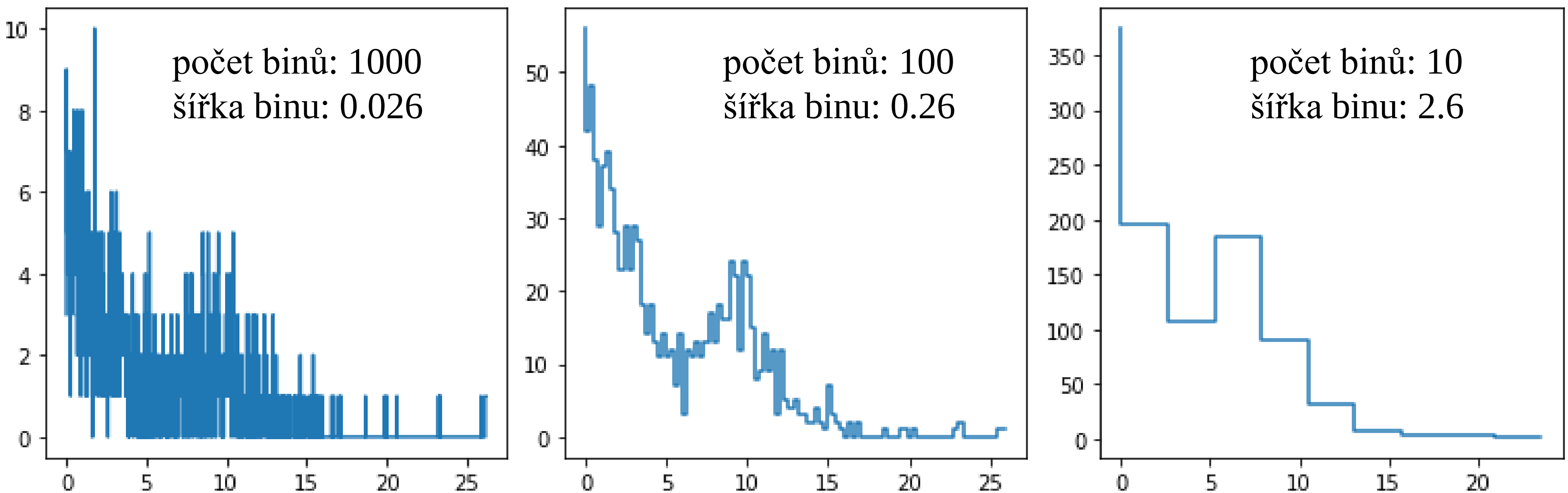
1. V Pythonu vytvořte histogram z naměřených hodnot uložených v souboru `data.dat`
   Nalezněte optimální šířku binu.

počet dat: 1000
minimum: 0.004
maximum: 26.198

`histogram.py`

Excel $m = \left\lceil \sqrt{N} \right\rceil = 32$

Sturges $m = \left\lceil \frac{\log N}{\log 2} + 1 \right\rceil = 11$



Histogram

Algoritmus pro nalezení optimální šířky binu

Shimazaki and Shinomoto. Neural Comput, 2007, 19(6), 1503-1527

1. zvol počet binů *m*, vypočítej šířku binu $\Delta = (x_{max} - x_{min})/m$ a vyrob histogram

2. vypočítej

   - odhad střední hodnoty výšky sloupečků histogramu: $\hat{\mu} = \frac{1}{N}\sum_{i=1}^{m} N_i$

   - odhad rozptylu výšek sloupečků histogramu: $\hat{\sigma}^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{\mu})^2$

3. vypočítej "ztrátovou funkci" $C(\Delta) = \frac{2\hat{\mu} - \hat{\sigma}^2}{\Delta^2}$

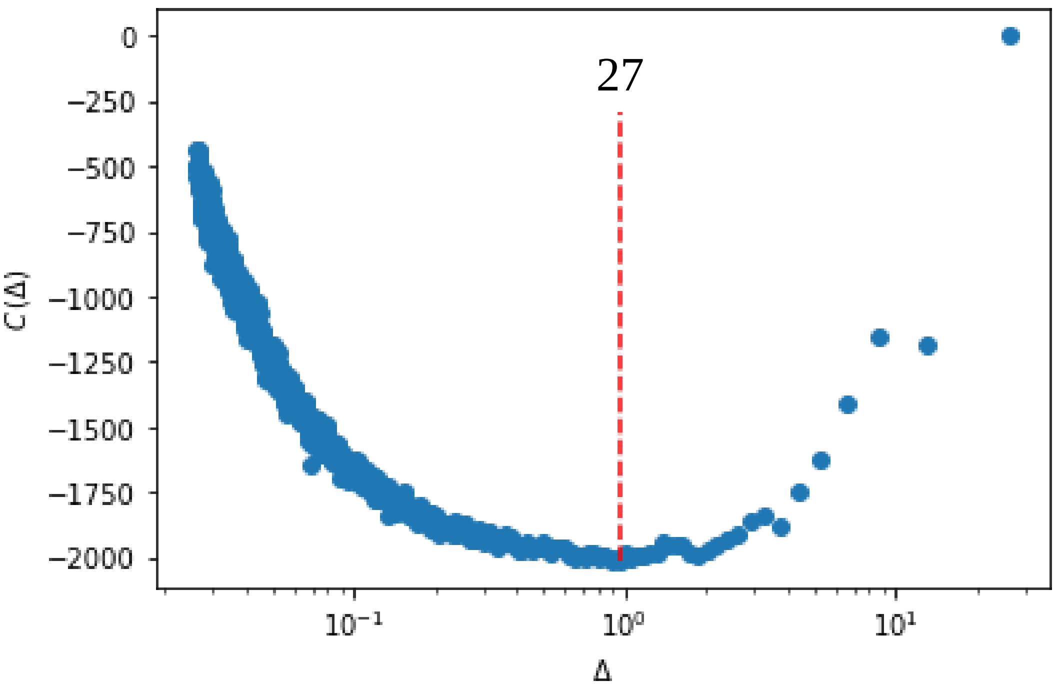
opakuj pro různé počty binů *m* (tj. různé Δ)

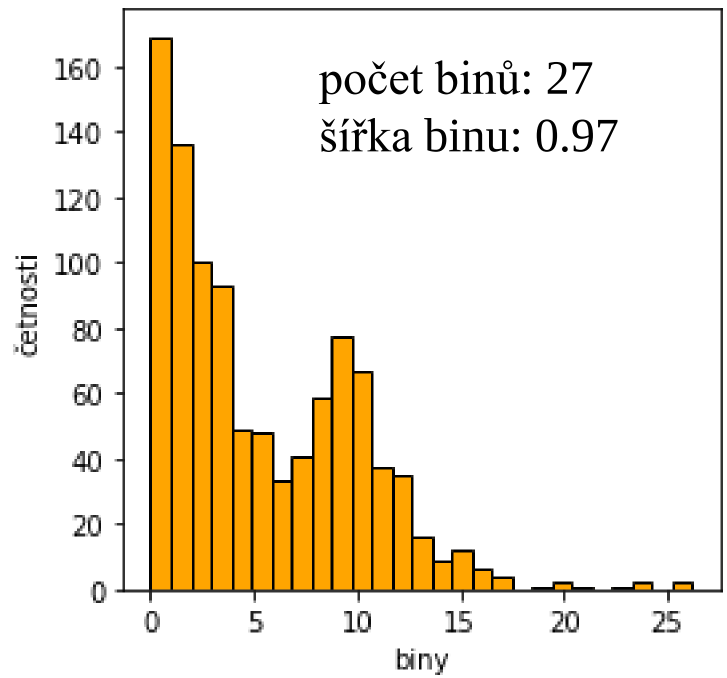
4. **Vyber takové Δ, pro které je *C* minimální**

Algoritmus pro nalezení optimální šířky binu

Shimazaki and Shinomoto. Neural Comput, 2007, 19(6), 1503-1527

“ztrátová funkce“ $$C(\Delta) = \frac{2\hat{\mu} - \hat{\sigma}^2}{\Delta^2}$$

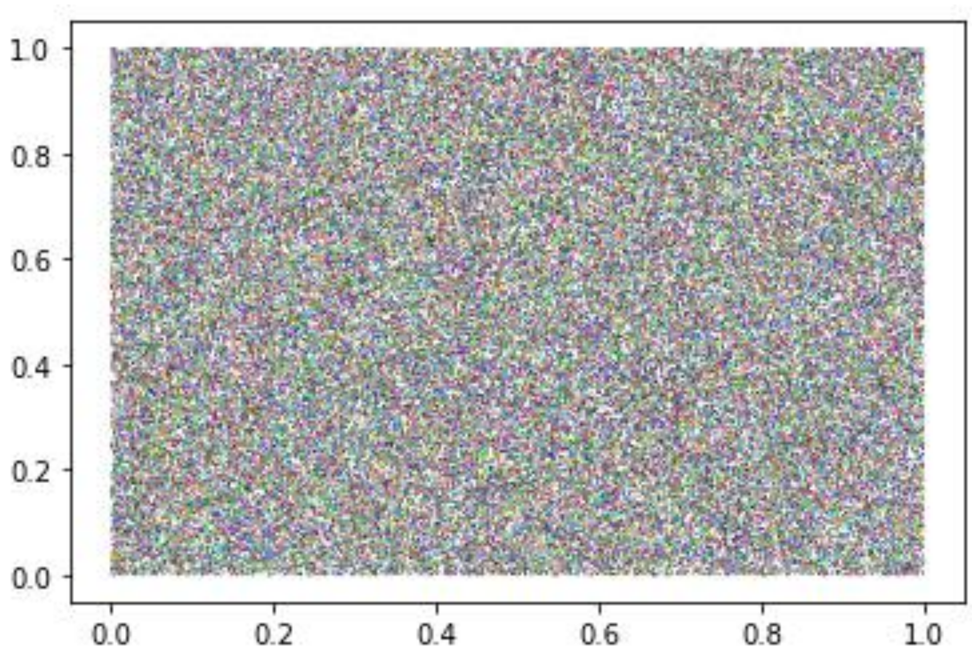




Barevný test v Pythonu

color-test.py

```
import numpy as np
import matplotlib.pyplot as plt
n=100000 #pocet dat, ktera b udeme simulovat
point=np.array([n,5]) #5 nahodnych cisel x-souradnice, y-souradnice, barva R,G,B
point=np.random.random_sample([n,5]) #vygenerovani nahodnych cisel
plt.scatter(point[0:n,0],point[0:n,1],s=2,c=point[0:n,2:5],edgecolor="none") #nakresleni grafu
```



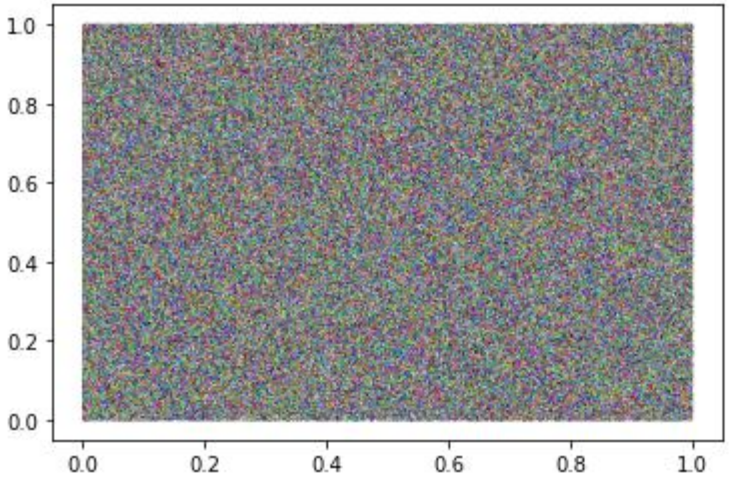
Lineární kongruentní generátor   kongruentni-generator.py

čistě mutiplikativní generátor

$$I_{j+1} = a\, I_{j+1} \pmod{m}$$

$$a = 16807$$

$$m = 2^{31} - 1 = 2147483647$$



počet dat $N = 10^6$

```
import numpy as np
import matplotlib.pyplot as plt
#ciste multiplikativni generator
a=16807
m=2147483647
#IBM RANDU
#a=65539
#m=2147483648
i_seed=1234
n=1000000
x = np.empty(n, dtype=float) #deklarace pole x-souradnic
y = np.empty(n, dtype=float) #deklarace pole y-souradnic
color=np.empty([n,3],dtype=float) #deklarace pole barva RGB
#ciste multiplikativni generator
i_old=i_seed
for i in range(0,n):
   i_next=(a*i_old) % m
   i_old=i_next
   x[i]=i_next/m
   i_next=(a*i_old) % m
   i_old=i_next
   y[i]=i_next/m
   i_next=(a*i_old) % m
   i_old=i_next
   color[i,0]=i_next/m
   i_next=(a*i_old) % m
   i_old=i_next
   color[i,1]=i_next/m
   i_next=(a*i_old) % m
   i_old=i_next
   color[i,2]=i_next/m
plt.scatter(x,y,s=1,c=color,edgecolors="none") #nakresli graf
```

Lineární kongruentní generátor

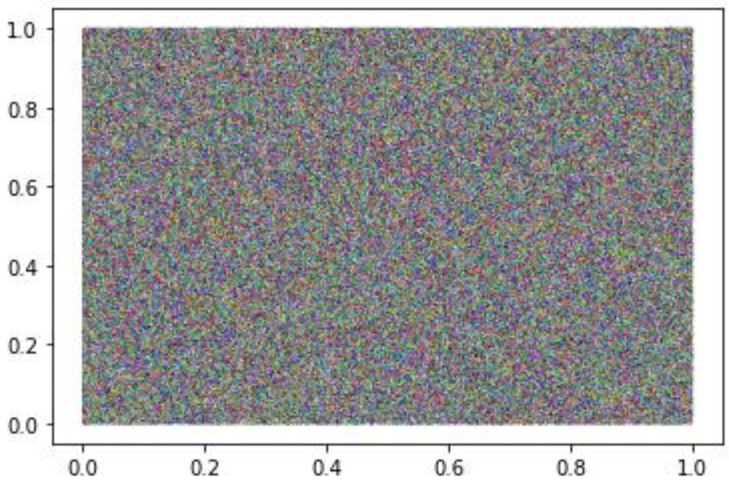
`kongruentni-generator.py`

čistě mutiplikativní generátor

$I_{j+1} = a\, I_{j+1} \pmod{m}$

$a = 16807$

$m = 2^{31} - 1 = 2147483647$



počet dat $N = 10^6$

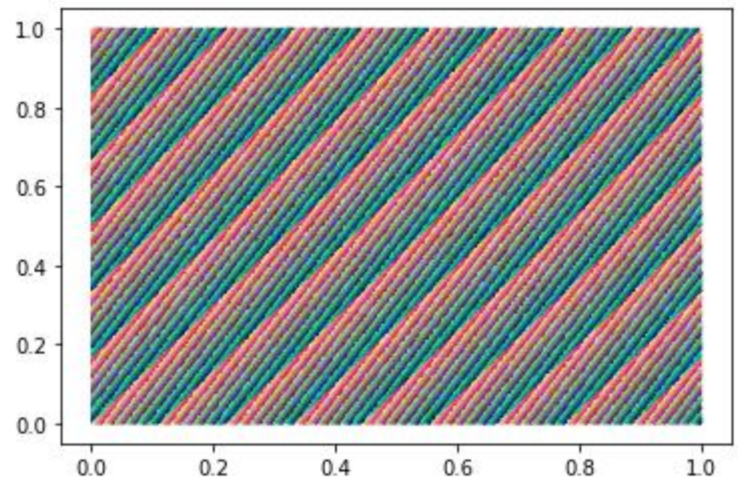
IBM RANDU

$I_{j+1} = a\, I_{j+1} \pmod{m}$

$a = 65539$

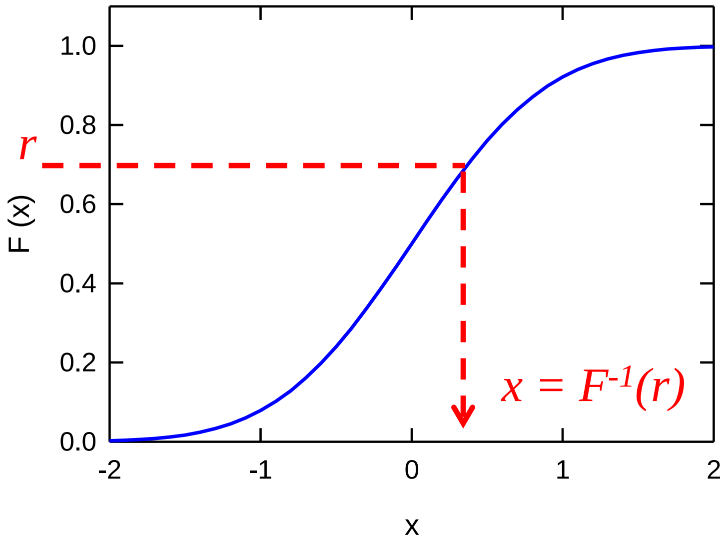
$m = 2^{31} = 2147483648$

“ We guarantee that each number is random individually, but we don’t guarantee that more than one of them is random.”



počet dat $N = 10^6$

Monte Carlo simulace – metoda inverzní funkce



Metoda inverzní funkce

1. Vygeneruj náhodnou proměnnou r z U(0,1)

2. Vypočítej $x = F^{-1}(r)$, kde F^{-1} je inverzní funkce k distribuční funkci $F(x)$ požadovaného rozdělení

Nechť x je náhodná proměnná s rozdělením popsaným hustotou pravděpodobnosti $f(x)$ a distribuční funkcí $F(x)$ potom náhodná proměnná $r = F(x)$ má rovnoměrné rozdělní U(0,1)

Monte Carlo simulace

2. Doba života vybuzeného stavu elektronu je 100 μs. Při rozpadu se emituje foton. Proveďte v Pythonu simulaci měření fotoluminiscence (200 hodnot). Nakreslete histogram naměřených hodnot.

- hustota pravděpodobnosti: $f(t) = \frac{1}{\tau} e^{-\frac{t}{\tau}}$
- distribuční funkce: $F(t) = \int_0^t \frac{1}{\tau} e^{-\frac{z}{\tau}} \, dz \longrightarrow F(t) = 1 - e^{-\frac{t}{\tau}}$
- náhodná proměnná s rovnoměrným rozdělením: $r \in U(0,1)$
- inverzní funkce k distribuční funkci: $F(r)^{-1} = -\tau \ln(1-r)$
- náhodnou proměnnou s exponenciálním rozdělením získáme takto: $t = -\tau \ln(1-r)$
- ekvivalentní je $t = -\tau \ln(r)$

Monte Carlo simulace

exp-rozpad.py

2. Doba života vybuzeného stavu elektronu je 100 µs. Při rozpadu se emituje foton. Proveďte v Pythonu simulaci měření fotoluminiscence (10000 hodnot). Nakreslete histogram naměřených hodnot.

```
import numpy as np
import matplotlib.pyplot as plt

N=10000
tau=100
r=np.random.random_sample(N)
x=-tau*np.log(r)
xp=np.linspace(0,10*tau,100)
yp=1/tau*np.exp(-xp/tau)
fig,ax=plt.subplots()
plt.hist(x,bins=100,density='True')
plt.plot(xp,yp,c='red')
ax.set_xlabel('t ($\mu$s)')
ax.set_ylabel('pdf')
```