# Otimização Natural - CPE 723

Nome: Patrícia de A. Kovaleski

Lista de Exercícios 1

(1) a) $\int_0^1 xe^{-x}\,dx = \left[-xe^{-x}\right]_0^1 - \int_0^1 -e^{-x}\,dx = (-e^{-1} - 0) - e^{-x}\Big|_0^1$

$= -e^{-1} - (e^{-1} - e^{0}) = 1 - 2e^{-1} \approx 0,2642$

b) $\bar{F} \approx \frac{1}{M}\sum_{j=1}^{M} F(x_j)$

$\begin{cases} a = \text{np.random.rand}(10) \\ b = [\,xe^{-x} \text{ for } x \text{ in } a\,] \\ c = \text{sum}(b)/10 \end{cases}$

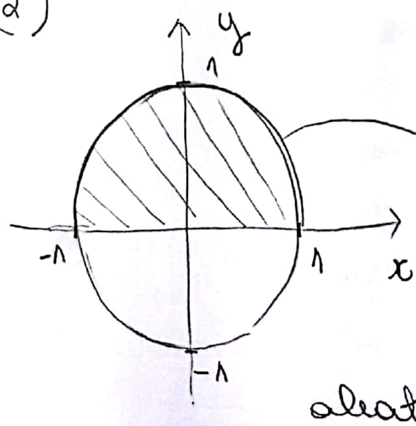
$c = 0,2612$ //

c)

$\begin{cases} a = \text{np.random.exponential}(\lambda = 1,\ \text{size} = 10) \\ b = [\,xe^{-x} \text{ for } x \text{ in } a\,] \\ c = \text{sum}(b)/10 \end{cases}$

$c = 0,2335$ //

(2)




$x^2 + y^2 = r^2 \rightsquigarrow y = \sqrt{r^2 - x^2}$

$\int_{-1}^{1} \sqrt{1-x^2}\,dx \rightsquigarrow \tilde{\pi} = 2\int_{-1}^{1} \sqrt{1-x^2}\,dx$

Podemos aproximar $\tilde{\pi}$, gerando M números aleatórios da forma:

↳ pelo método de Monte Carlo

$\tilde{\pi} \approx 2 \times \frac{1}{M} \times 2 \sum_{i=1}^{M} \sqrt{1 - x_i^2}$

$N = 20 \rightarrow \hat{\pi} \approx 3.2576$

$N = 1.000.000 \rightarrow \hat{\pi} \approx 3.1423$

Fica claro que quanto maior o número de exemplos (M) melhor ficará a aproximação desejada.

④ • Distribuição uniforme

| N | $\hat{x}$ | $\hat{J}$ | $\Delta J$ | $q > r$ | $x_{min}$ | $J_{min}$ |
|---|---|---|---|---|---|---|
| 0 | 0.1482 | -0.0345 | -2.5944 | — | 0.1482 | -0.0345 |
| 1 | 0.06177 | 0.9796 | 1.0140 | 1 | " | " |
| 2 | -0.0219 | 3.3487 | 2.3692 | 0 | " | " |
| 3 | -0.0426 | 4.2202 | 3.2406 | 0 | " | " |
| 4 | 0.3024 | -0.0427 | -1.0224 | — | 0.3024 | -0.0427 |
| 5 | 0.3915 | 0.2207 | 0.2634 | 1 | " | " |
| 6 | 0.3121 | -0.0128 | -0.2335 | — | " | " |
| 7 | 0.3733 | 0.1734 | 0.1863 | 0 | " | " |
| 8 | 0.1810 | -0.1673 | -0.1545 | — | 0.1810 | -0.1673 ← |
| 9 | 0.1267 | 0.1169 | 0.2842 | 0 | " | " |

• Distribuição gaussiana

| N | $\hat{x}$ | $\hat{J}$ | $\Delta J$ | $q > r$ | $x_{min}$ | $J_{min}$ |
|---|---|---|---|---|---|---|
| 0 | -0.0347 | 3.8733 | 1.3133 | 0 | 0 | 2.56 |
| 1 | -0.0584 | 4.9779 | 2.4179 | 0 | " | " |
| 2 | -0.0479 | 4.4652 | 1.9052 | 0 | " | " |
| 3 | 0.0595 | 1.0218 | -1.5382 | — | 0.0595 | 1.0218 |
| 4 | 0.0276 | 1.7447 | 0.7229 | 0 | " | " |
| 5 | 0.0821 | 0.6349 | -0.3868 | — | 0.0821 | 0.6349 |
| 6 | 0.0864 | 0.5701 | -0.0647 | — | 0.0864 | 0.5701 |
| 7 | 0.1539 | -0.0654 | -0.6356 | — | 0.1539 | -0.0654 ← |
| 8 | 0.1535 | -0.0632 | 0.0022 | 1 | " | " |
| 9 | 0.0724 | 0.7897 | 0.8529 | 0 | " | " |

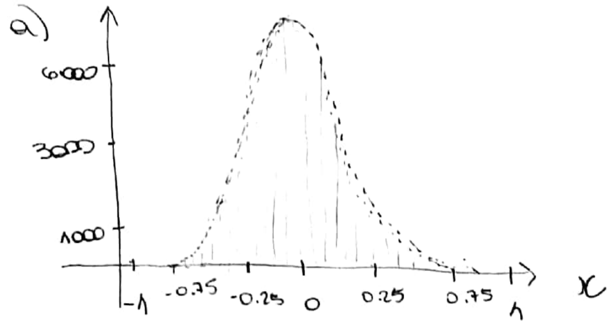
$k=1$ ; $x=0$ ; $j = J(x)$ ; $x_{mín} = x$ ; $j_{mín} = j$ ; $fim = False$ ;

```
while( ! fim )
    T = 0.5 / log_2(1+k)

    for i in range(N):
        x̂ = x + 0.1 * random()
        ĵ = J(x̂)
        ΔJ = ĵ - j
        q = e^(-ΔJ/T)
        r = U(0,1)
        a = 0 if r>q else 1
        x = (1-a)x + a x̂
        j = J(x)

        if j < jmin :
            xmín = x
            jmín = j

        k += 1
        if k > KMÁX:
            fim = True
```

- Executando para $N = 10.000$ e $K_{MAX} = 5$:
  - random() : uniforme
    $x_{mín}$: 0.8130 $j_{mín}$: -0.8066
  - random() : gaussiana
    $x_{mín}$: 0.8129 $j_{mín}$: -0.8066

③

a)




Os valores encontrados tenderam a ficar próximos de zero; como esperado para a distribuição de Boltzmann.

b)

$x(0) = 0.2796$

| N | $\hat{x}$ | $\hat{J}$ | $\Delta J$ | $q > r$ |
|---|---|---|---|---|
| 1 | 0.3674 | 0.1350 | 0.0569 | 1 |
| 2 | 0.3098 | 0.0959 | -0.0390 | — |
| 3 | 0.3212 | 0.1032 | 0.0072 | 1 |
| 4 | 0.2235 | 0.0499 | -0.0532 | — |
| 5 | 0.3023 | 0.0914 | 0.0414 | 1 |
| 6 | 0.3088 | 0.0954 | 0.0039 | 1 |
| 7 | 0.3834 | 0.1471 | 0.0516 | 0 |
| 8 | 0.3123 | 0.0975 | 0.0021 | 1 |
| 9 | 0.2195 | 0.0482 | -0.0493 | — |
| 10 | 0.2116 | 0.0448 | -0.0034 | — |

⑤ A função $f(x,y) = x^2 + 10y^2$ tem seu ponto mínimo em $x = 0$ e $y = 0$. Executando o algoritmo para $N = 100.000$ e $K_{MAX} = 5$ temos $x = -0.6462 \times 10^{-3}$, $y = -1,1882 \times 10^{-3}$ e $f_{min} = 1,4537 \times 10^{-5}$.

```
k=1; x= np.random.rand(2) ; j= J(*x) ; xmin = x ; jmin = j
while(not fim):
    T= 0.5/log2(k+1)
    for i in range(N):
        x̂ = x + 0.1 (np.random.rand()*2 - 1)
        ĵ = J(*x̂)
        ΔJ = ĵ - j
```

e →

```
def J(x,y):
    return pow(x,2) - 10 * pow(y,2)
```

```
q = pow(np.e, -ΔJ/T)
r = np.random.rand()
a = 0 if r>q else 1

x = (1-a)x + a.x̂
j = J(*x)

if j < jmin:
    jmin = j
    xmin = x

k += 1
if k > kMÁX:
    fim = True
```