O_metodo_de_Wang-Landau_-_Guilherme_de_Abreu_Lima_Buitrago_Miranda_-_2018054788

July 8, 2021

1 Introdução à Física Estatística Computacional

1.1 O método de Wang-Landau

Aluno: Guilherme de Abreu Lima Buitrago Miranda

Matrícula: 2018054788

1.1.1 Imports

[1]:
```
import matplotlib.pyplot as plt
from numba import jit
import numpy as np

plt.style.use('seaborn-colorblind')
plt.ion()
```

1.1.2 Funções

As funções abaixo foram extraídas dos arquivos fornecidos no enunciado do exercício. No fim, há ainda outras funções criadas por mim.

[2]:
```
Niter=10000000  # Número máximo de passos de Monte Carlo (iterações do␣
 ↪algoritmo)
L=6            # Dimensão linear da rede
flatness = 0.8  # Condição para zerar o histograma e diminuir lnf quando
                # min(Histogram) > average(Histogram)*flattness

N=L*L           # Total de sítios da rede

print("Simulação do modelo de Ising 2D pelo método de Wang-Landau")
print("Rede quadrada",L,"x",L)

@jit(nopython=True)
def estado_ini(N):
    #Gera um estadon inicial aleatório para rede
    s = np.zeros(N,dtype=np.int8)
```

```
    for i in range(N):
        s[i] = np.sign(2*np.random.random()-1)
    return s

@jit(nopython=True)
def vizinhos(L,N):
    #Define a tabela de vizinhos
    viz = np.zeros((N,4),dtype=np.int16)
    for k in range(N):
        viz[k,0]=k+1
        if (k+1) % L == 0: viz[k,0] = k+1-L
        viz[k,1] = k+L
        if k > (N-L-1): viz[k,1] = k+L-N
        viz[k,2] = k-1
        if k % L == 0: viz[k,2] = k+L-1
        viz[k,3] = k-L
        if k < L: viz[k,3] = k+N-L
    return viz

@jit(nopython=True)
def energia(s,viz):
    #Calcula a energia da configuração s
    ener = 0
    for i in range(N):
        h = s[viz[i,0]]+s[viz[i,1]]
        ener -= s[i]*h
    ener = int((ener+2*N)/4)
    return ener

@jit(nopython=True)
def minh(H):
    #Calcula o menor valor de H excluindo as energias proibidas
    minh=H[0]
    for i in range(2,N-1):
        if H[i] < minh: minh=H[i]
    if H[-1] < minh: minh=H[-1]
    return minh

@jit(nopython=True)
def wang_landau(N,Niter,flatness,ener,s,viz):
    # Algoritmo de Wang-Landau
    lnge = np.zeros(N+1,dtype=np.float64)
    H = np.zeros(N+1,dtype=np.int16)
    Hc = np.zeros(N+1,dtype=np.int16)
    mmicro = np.zeros(N+1,dtype=np.float64)
    lnf = 1.0
    m = s.sum()
```

```
    for it in range(Niter):
        #Iterações do algoritmo
        for imc in range(N):
            #Passo de Monte Carlo - percorre toda a rede
            k = np.random.randint(0,N-1)
            h = s[viz[k,0]]+s[viz[k,1]]+s[viz[k,2]]+s[viz[k,3]] # soma dos␣
↪vizinhos
            ener2 = ener + int(s[k]*h*0.5)
            #print(lnge[ener]-lnge[ener2])
            if lnge[ener]>lnge[ener2]:
                s[k] = -s[k]
                ener = ener2
                m -= 2*s[k]
            else:
                p = np.exp(lnge[ener]-lnge[ener2])
                if np.random.random() < p:
                    s[k] = -s[k]
                    ener = ener2
                    m -= 2*s[k]
            H[ener] += 1
            lnge[ener] += lnf
            mmicro[ener] += abs(m)
        if it%1000 == 0:
            hmed = float(H.sum())/float((N-1))
            hmin = minh(H)
            if hmin > (flatness*hmed):
                Hc += H
                H = np.zeros(N+1,dtype=np.int16)
                lnf = 0.5*lnf
                print("Histograma flat!",lnf)
            if it%1000000 == 0: print("Iteração número",it)
        if lnf < 0.00000001: break

    mmicro = mmicro/Hc
    lnge = lnge - lnge[0]+np.log(2)
    return lnge,mmicro,ener,s
```

```
Simulação do modelo de Ising 2D pelo método de Wang-Landau
Rede quadrada 6 x 6
```

[3]:
```
def media_termodinamica(lnge, t, n, mag):
    Z = 0
    ener = 0
    ener2 = 0
    magmed = 0
    mag2 = 0
    e_range = ((np.arange(n + 1)) * 4) - (2 * n)
```

```
    e_range = np.delete(e_range, 1)
    e_range = np.delete(e_range, -2)

    lnge = np.delete(lnge, 1)
    lnge = np.delete(lnge, -2)

    var_lambda = max(lnge - e_range / t)

    for i in range(len(e_range)):
        ege = np.exp(lnge[i] - e_range[i] / t - var_lambda)
        Z += ege
        ener += e_range[i] * ege
        ener2 += e_range[i] * e_range[i] * ege
        magmed += mag[i] * ege
        mag2 += mag[i] * mag[i] * ege


    ener /= Z
    ener2 /= Z
    magmed /= Z
    mag2 /= Z

    cv = ((1/t) * (1/t)) * (ener2 - (ener * ener)) / n

    return Z, ener, cv, e_range
```

[4]:
```
def execute_all(l, n):
    s = estado_ini(n)
    viz = vizinhos(l, n)
    ener = energia(s, viz)
    lnge, mmicro, ener, s = wang_landau(n, Niter, flatness, ener, s, viz)

    x = np.linspace(1, 5, 50)
    cvs = []
    Zs = []
    es = []
    es_ = []
    for x_ in x:
        z, e_, cv, E = media_termodinamica(lnge, x_, n, mmicro)
        cvs.append(cv)
        Zs.append(z)
        es.append(e_)
        es_.append(E)

    return cvs, Zs, es, x, np.exp(lnge), mmicro, es_
```

[5]:
```
def plot(y_list, x):
    for y in y_list:
        plt.plot(x, y)
```

1.1.3 Execução

As células abaixo invocam os métodos que realizam os cálculos e, em seguida, geram os seguintes gráficos:

i) Energia por spin, $\langle e \rangle$, como função da temperatura

ii) Calor específico, c_v, como função da temperatura

iii) Energia livre por spin, $f(T) = \frac{-1}{N\beta} lnZ$

iv) Entropia por spin, $s(T) = \frac{\langle e \rangle - f(T)}{T}$

v) Magnetização média por spin, $\langle |m| \rangle$ / N.

[6]:
```
ls = [6, 12, 18, 24]
ns = [l * l for l in ls]

cvs = []
Zs = []
es = []
xs = []
gs = []
mmicro_list = []
es_ = []

for l in ls:
    cv, Z, e, x, g, mmicro, E = execute_all(l, (l * l))
    cvs.append(cv)
    Zs.append(Z)
    es.append(e)
    xs.append(x)
    gs.append(g)
    mmicro_list.append(mmicro)
    es_.append(E)
```

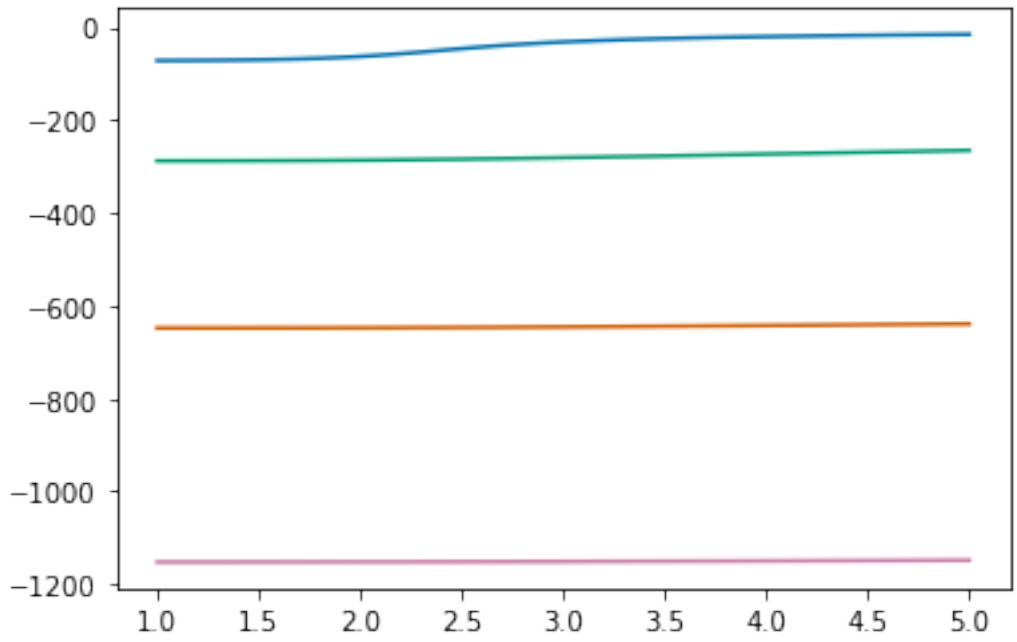
```
Iteração número 0
Histograma flat! 0.5
Histograma flat! 0.25
Histograma flat! 0.125
Histograma flat! 0.0625
Histograma flat! 0.03125
Histograma flat! 0.015625
Histograma flat! 0.0078125
Histograma flat! 0.00390625
Histograma flat! 0.001953125
```

```
Histograma flat! 0.0009765625
Histograma flat! 0.00048828125
Histograma flat! 0.000244140625
Histograma flat! 0.0001220703125
Histograma flat! 6.103515625e-05
Histograma flat! 3.0517578125e-05
Histograma flat! 1.52587890625e-05
Histograma flat! 7.62939453125e-06
Histograma flat! 3.814697265625e-06
Histograma flat! 1.9073486328125e-06
Histograma flat! 9.5367431640625e-07
Histograma flat! 4.76837158203125e-07
Histograma flat! 2.384185791015625e-07
Histograma flat! 1.1920928955078125e-07
Histograma flat! 5.960464477539063e-08
Histograma flat! 2.9802322387695312e-08
Histograma flat! 1.4901161193847656e-08
Histograma flat! 7.450580596923828e-09
Iteração número 0
Histograma flat! 0.5
Histograma flat! 0.25
Histograma flat! 0.125
Histograma flat! 0.0625
Histograma flat! 0.03125
Histograma flat! 0.015625
Histograma flat! 0.0078125
Histograma flat! 0.00390625
Histograma flat! 0.001953125
Histograma flat! 0.0009765625
Histograma flat! 0.00048828125
Histograma flat! 0.000244140625
Histograma flat! 0.0001220703125
Histograma flat! 6.103515625e-05
Histograma flat! 3.0517578125e-05
Histograma flat! 1.52587890625e-05
Histograma flat! 7.62939453125e-06
Histograma flat! 3.814697265625e-06
Histograma flat! 1.9073486328125e-06
Histograma flat! 9.5367431640625e-07
Histograma flat! 4.76837158203125e-07
Histograma flat! 2.384185791015625e-07
Histograma flat! 1.1920928955078125e-07
Histograma flat! 5.960464477539063e-08
Histograma flat! 2.9802322387695312e-08
Histograma flat! 1.4901161193847656e-08
Histograma flat! 7.450580596923828e-09
Iteração número 0
Histograma flat! 0.5
```

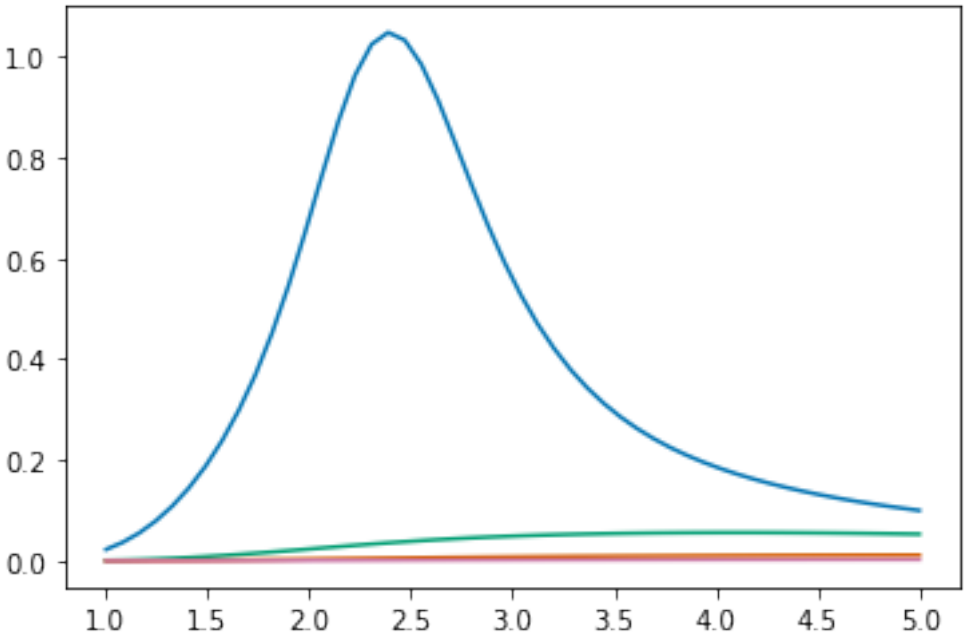
```
Histograma flat! 0.25
Histograma flat! 0.125
Histograma flat! 0.0625
Histograma flat! 0.03125
Histograma flat! 0.015625
Histograma flat! 0.0078125
Histograma flat! 0.00390625
Histograma flat! 0.001953125
Histograma flat! 0.0009765625
Histograma flat! 0.00048828125
Histograma flat! 0.000244140625
Histograma flat! 0.0001220703125
Histograma flat! 6.103515625e-05
Histograma flat! 3.0517578125e-05
Histograma flat! 1.52587890625e-05
Histograma flat! 7.62939453125e-06
Histograma flat! 3.814697265625e-06
Histograma flat! 1.9073486328125e-06
Histograma flat! 9.5367431640625e-07
Histograma flat! 4.76837158203125e-07
Histograma flat! 2.384185791015625e-07
Histograma flat! 1.1920928955078125e-07
Histograma flat! 5.960464477539063e-08
Histograma flat! 2.9802322387695312e-08
Histograma flat! 1.4901161193847656e-08
Histograma flat! 7.450580596923828e-09
Iteração número 0
Histograma flat! 0.5
Histograma flat! 0.25
Histograma flat! 0.125
Histograma flat! 0.0625
Histograma flat! 0.03125
Histograma flat! 0.015625
Histograma flat! 0.0078125
Histograma flat! 0.00390625
Histograma flat! 0.001953125
Histograma flat! 0.0009765625
Histograma flat! 0.00048828125
Histograma flat! 0.000244140625
Histograma flat! 0.0001220703125
Histograma flat! 6.103515625e-05
Histograma flat! 3.0517578125e-05
Histograma flat! 1.52587890625e-05
Histograma flat! 7.62939453125e-06
Histograma flat! 3.814697265625e-06
Histograma flat! 1.9073486328125e-06
Histograma flat! 9.5367431640625e-07
Histograma flat! 4.76837158203125e-07
```

```
Histograma flat! 2.384185791015625e-07
Histograma flat! 1.1920928955078125e-07
Histograma flat! 5.960464477539063e-08
Histograma flat! 2.9802322387695312e-08
Histograma flat! 1.4901161193847656e-08
Histograma flat! 7.450580596923828e-09
```

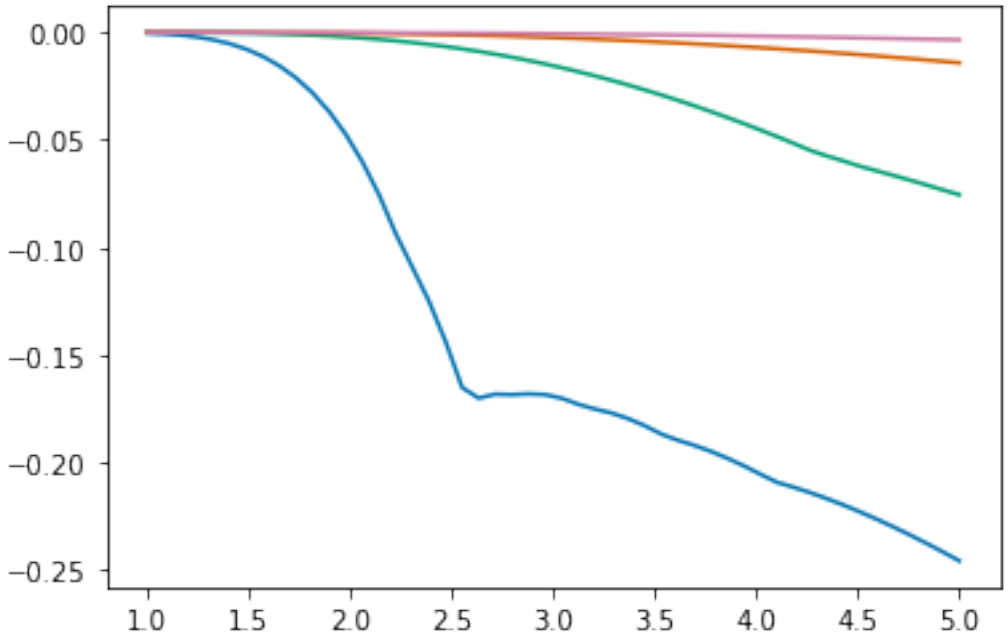
[7]:
```
plot(es, x)
```



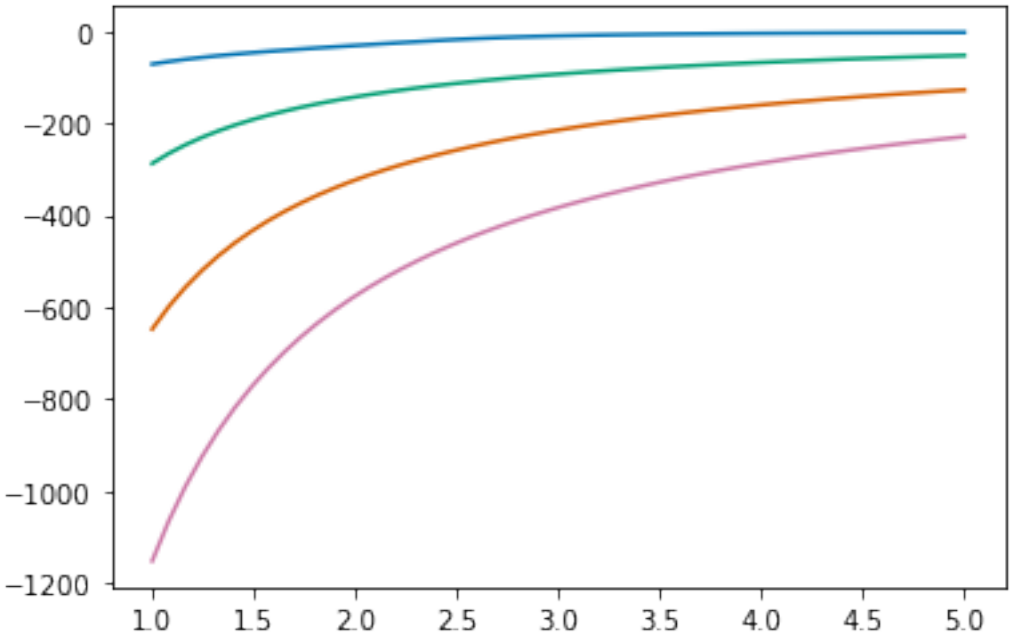
[8]:
```
plot(cvs, x)
```



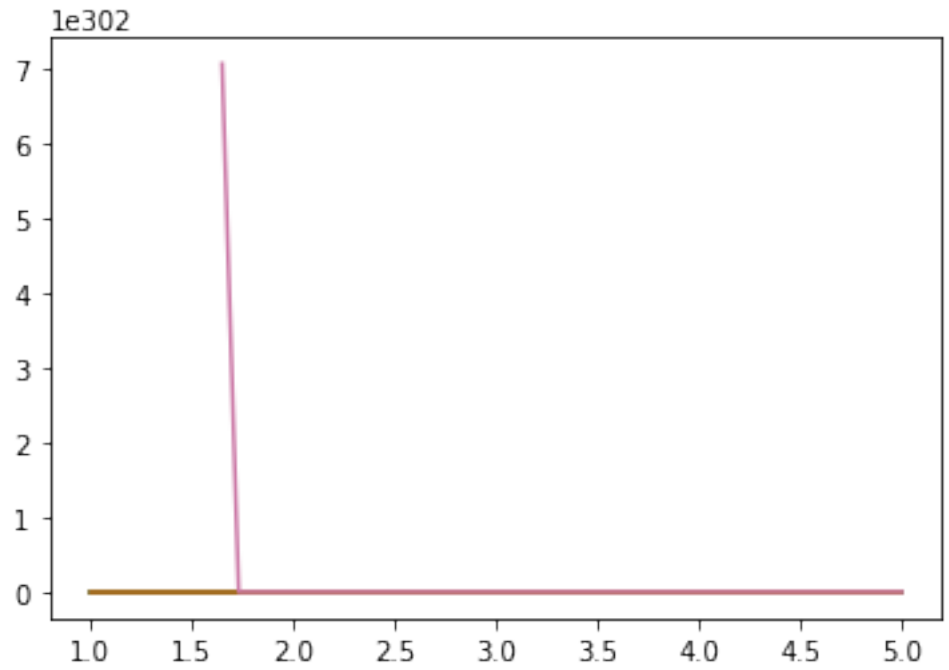
[9]:
```python
def calc_f(T, Z, n):
    beta = 1/T
    y = (-1/(n * beta)) * np.log(Z)
    return y

fs = []
fs.append(calc_f(x, Zs[0], ns[0]))
fs.append(calc_f(x, Zs[1], ns[1]))
fs.append(calc_f(x, Zs[2], ns[2]))
fs.append(calc_f(x, Zs[3], ns[3]))

plot(fs, x)
```



[10]:
```
def calc_s(T, e_, f):
    return ((e_ - f)/T)

ss = []
ss.append(calc_s(x, es[0], fs[0]))
ss.append(calc_s(x, es[1], fs[1]))
ss.append(calc_s(x, es[2], fs[2]))
ss.append(calc_s(x, es[3], fs[3]))

plot(ss, x)
```



[11]:
```
def calc_m(m, g, Z, E, T):
    m = np.nan_to_num(m)
    result = np.float128(0)
    beta = 1/T
    for i, e in enumerate(E):
        result += np.float128(np.mean(m[i])*g[i]*np.exp(-beta*e, dtype=np.
 ↪float128))

    result /= Z

    return result

mm = []

mm.append(calc_m(abs(mmicro_list[0]), gs[0], Zs[0], es_[0][0], xs[0]) / ns[0])
mm.append(calc_m(abs(mmicro_list[1]), gs[1], Zs[1], es_[1][0], xs[1]) / ns[1])
mm.append(calc_m(abs(mmicro_list[2]), gs[2], Zs[2], es_[2][0], xs[2]) / ns[2])
mm.append(calc_m(abs(mmicro_list[3]), gs[3], Zs[3], es_[3][0], xs[3]) / ns[3])

plot(mm, x)
```



1.1.4 Análise dos Resultados Obtidos

Como pode ser observado a partir do primeiro gráfico apresentado acima, a energia por spin é praticamente constante em função da temperatura, mas varia de acordo com a instância do problema executada, sendo mais negativa conforme aumenta o tamanho do problema.

Já para o segundo gráfico, é notável que o calor específico cresce muito lentamente para os tamanhos 12, 18 e 24; a instância 6x6 do problema tem um comportamento particular, em que há um rápido crescimento até 2.5, seguido por um decaimento e achatamento da curva até o 5.

O gráfico de número 3, representando energia livre por spin, mostra uma tendência de queda quanto maior a temperatura. Essa queda é acelerada quanto menor a instância do problema. Assim, o 24x24 parece ser quase constante, o 18x18 cai devegar, o 12x12 cai mais rapidamente a partir do 3.5 e, por fim, o 6x6 cai muito rapidamente já a partir do 1.5, desacelerando a queda posteriormente.

O quarto gráfico, de entropia por spin, representa bem o contrário do gráfico anterior. Quanto maior a temperatura, maior a entropia por spin; além disso, quanto maior a instância do problema, mais rapidamente a entropia por spin cresce, estabilizando seu crescimento posteriormente.

Por fim, o último gráfico, de magnetização média por spin, é pouco representativo. Para gerá-lo, foi necessário usar o tipo flutuante de quádrupla precisão e, em razão da grande disparidade dos valores, a visualização encontrada não diz muito sobre a grandeza observada. Ainda assim, aparentemente há uma diminuição do valor a medida que a variável N aumenta, ou seja, são inversamente proporcionais.

[ ]: