Wykorzystanie algorytmu Proppa-Wilsona w modelu Isinga

Jan Wojtas, Oskar Werner

Motywacja

Motywacją do powstania modelu Isinga było przeprowadzenie następującego doświadczenia:
Wyobraźmy sobie, że dysponujemy kulką wykonaną z materiału wykazującego własności ferromagnetyczne (np. z żelaza). Kulkę zawieszamy na sznurku, a w pewnej odległości od niej ustawiamy magnes. Pod magnesem zaś znajduje się świeca. Podczas przeprowadzenia doświadczenia zaobserwowano ciekawą własność. Po przyciągnięciu przez magnes owej kulki okazywało się, że w wyniku ogrzania ciepłem pochodzącym od świecy kulka wracała do swojego pierwotnego położenia - traciła właściwości magnetyczne. Formalnie, pod wpływem ciepła kulka stawała się paramagnetykiem.
W celu analizowania tego zjawiska powstał model Isinga.

Doświadczenie

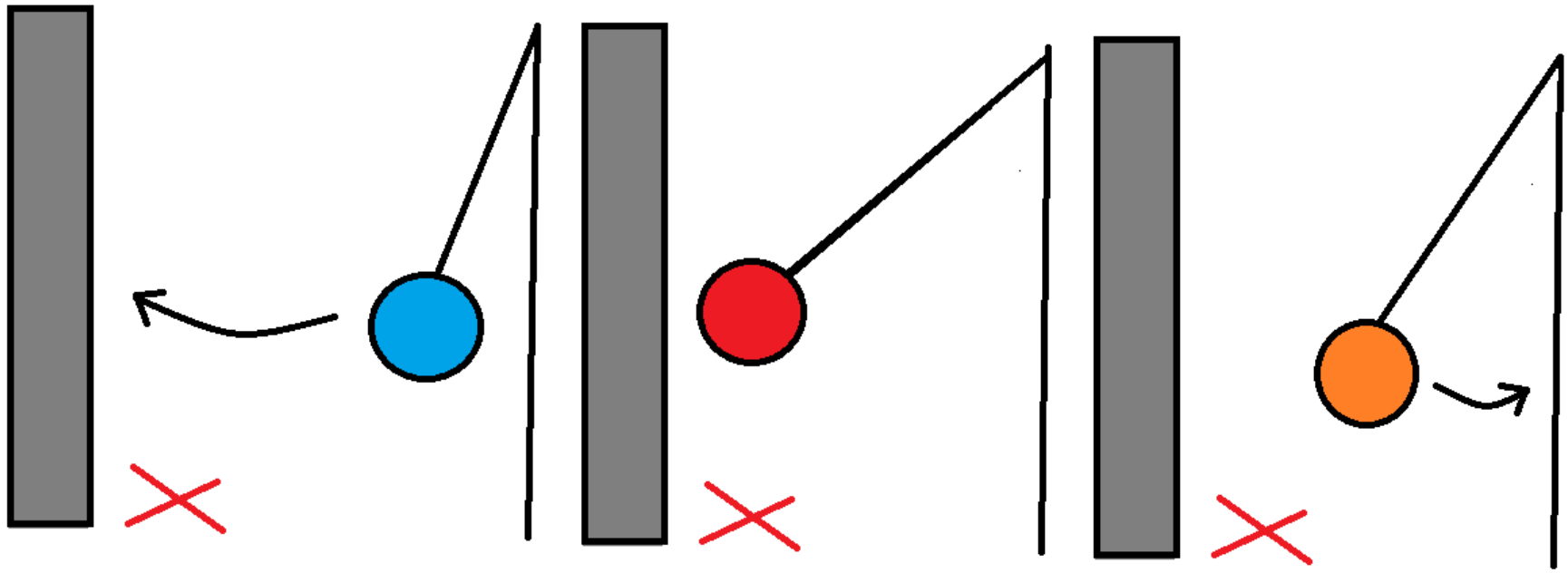


Figure 1: Motywacja dla modelu Isinga

Model Matematyczny

Z fizycznego punktu widzenia zakładamy, że dysponujemy ferromagnetykiem, umieszczonym w izolowanym rezerwuarze termicznym, o stałej temperaturze $T = const$. Dla uproszczenia przyjmujemy również, że ferromagnetyk zbudowany jest wyłącznie z atomów oraz węzłów pomiędzy atomami.
Patrząc na problem od strony matematycznej, rozpatrujemy graf $G = (V, E)$, gdzie V jest zbiorem wierzchołków reprezentujących atomy, zaś E zbiorem krawędzi - węzłów. Ponadto, każdemu z wierzchołków przypisujemy pewną wartość $\sigma_v \in \{-1, 1\}$, tzw. spin. Wprowadzamy oznaczenie $\xi \in \{-1, 1\}^N$ jednoznacznie określające konkretny układ spinów w grafie.

Energia wybranego układu ξ zadana jest przez hamiltonian, opisany następującym wzorem:

$$H(\xi) = -\sum_{(i,j)\in E} J_{ij}\sigma_i\sigma_j - \sum_{i\in V} h_i\sigma_i,$$

gdzie J_{ij} jest wartością oddziaływania i z j, zaś h_i opisuje wartość zewnętrznego pola magnetycznego wierzchołka i. W niniejszej pracy rozpatrujemy model uproszczony, gdzie $J_{ij} = J = 1$ dla każdej pary $(i,j) \in E$ oraz $h_i = 0$ dla każdego $i \in V$. Wówczas wzór redukuje się do postaci:

$$H(\xi) = -\sum_{(i,j)\in E} \sigma_i\sigma_j$$

Energi wpływa bezpośrednio na prawdopodobieństwo pojawienia się układu w określonej temperaturze T. Ściślej, prawdopodobieństwo wystąpienia konkretnego układu ξ jest zadane przez rozkład Boltzmanna:

$$\pi_{G,\beta}(\xi) = \frac{1}{Z_{G,\beta}} \exp\{-\beta H(\xi)\}, \tag{1}$$

gdzie $Z_{G,\beta} = \sum_{\eta} \exp\{-\beta H(\eta)\}$ jest stałą normującą, zaś $\beta = \frac{1}{k_B T}$ jest współczynnikiem odwrotnie proporcjonalnym do temperatury otoczenia (k_B to stała Boltzmanna).

Cel projektu

Celem projektu jest analiza zachowania się układu spinów ferromagnetyka w ustalonym rezerwuarze termicznym. Sprowadza się to do opracowania algorytmu umożliwiającego próbkowanie z zadanego rozkładu $\pi_{G,\beta}$ dla ustalonego grafu (np. krata, graf pełny) i temperatury otoczenia.

Algorytmiczne rozwiązanie

Bezpośrednie próbkowanie z rozkładu (1) jest zadaniem trudnym. Liczba możliwych układów wynosi 2^N, więc w szczególności dla modelu o dużej liczbie wierzchołków zadanie może być wręcz niewykonalne. Stąd w praktyce, w celu próbkowania z takiego rozkładu korzysta się z metod Monte Carlo - metod opartych na skonstruowaniu nieprzywiedlnego i nieokresowego łańcucha Markowa o rozkładzie stacjonarnym $\pi_{G,\beta}$, który zapewnia asymptotyczną zbieżność do zadanego rozkładu. W niniejszej pracy wykorzystany został algorytm Proppa-Wilsona, z którego implementację przeanalizujemy w dalszej części pracy.

Zamiana liczby w macierz

```
1  def num_to_matrix(num,N):
2      n2 = N**2
3      x = bin(num)[2:]
4
5      if len(x)<n2:
6          y = (n2-len(x))*'0' + x
7      else:
8          y = x
9
10     matrix = [int(char) for char in y]
11     matrix = np.array(matrix).reshape(N,N)
12
13     return(matrix)
```

Funkcja dokonuje zamianę liczby z systemu dziesiętnego na dwójkowy, a następnie konstruuje odpowiednią macierz (reprezentująca stan modelu Isinga).

Zamiana systemu dwójkowego na dziesiętny

```
1 def binary_to_decimal(lista_binarna):
2     num = 0
3     for value in lista_binarna:
4         num = 2 * num + value
5     return num
```

Funkcja zamienia liczbę w systemie dwójkowym na liczbę w systemie dziesiętnym.

Hamiltonian dla kraty

```
def get_hamiltonian(matrix):
    kern = np.array([
        [0,1,0],
        [1,0,1],
        [0,1,0]
    ])

    return((matrix * convolve(matrix, kern, ↩
   mode = 'constant', cval = 0)).sum()/2)
```

Funkcja oblicza hamiltonian konkretnego stanu modelu Isinga, reprezentowanego przez macierz (dla kraty).

Hamiltonian dla grafu pełnego

```
1 def get_hamiltonian2(matrix):
2     spin_up_count = matrix[matrix == 1].sum()
3     n2 =  (matrix.shape[0])**2
4     if spin_up_count == 0 or spin_up_count == ↩
    n2:
5         energy = (n2)*(n2-1)/2
6     else:
7         energy = spin_up_count*(spin_up_count↩
    -1)/2 + (n2 - spin_up_count)*(n2 - ↩
    spin_up_count-1)/2 - (n2 - spin_up_count)*↩
    spin_up_count
8     return energy
```

Funkcja oblicza hamiltonian konkretnego stanu modelu Isinga, reprezentowanego przez macierz (dla grafu pełnego).

Znalezienie teoretycznego rozkładu stacjonarnego

```
1  def get_equilibrium_distribution(N,beta,graph_type):
2      if graph_type == 'krata':
3          g = get_hamiltonian
4      elif graph_type == 'pelny':
5          g = get_hamiltonian2
6      n = 2**(N*N)
7      pi_theoretical = np.zeros(n)
8      Z = 0
9      for i in range(n):
10         matrix = num_to_matrix(i,N)
11         matrix[matrix == 0] = -1
12         pi_theoretical[i] = np.exp(beta * g(matrix))
13         Z += pi_theoretical[i]
14     pi_theoretical = 1/Z * np.array(pi_theoretical)
15     return pi_theoretical
```

Funkcja wyznacza teoretyczny rozkład stacjonarny modelu Isinga na podstawie wzoru

$$\pi_{G,\beta}(\xi) = \frac{1}{Z_{G,\beta}} \exp\{-\beta H(\xi)\}, \tag{2}$$

gdzie $Z_{G,\beta} = \sum_{\eta} \exp\{-\beta H(\eta)\}$ jest stałą normującą, zaś $\beta = \frac{1}{k_B T}$ jest współczynnikiem odwrotnie proporcjonalnym do temperatury otoczenia (k_B to stała Boltzmanna).

Próbnik Gibbsa dla Kraty

```
1  def update(beta, x, y, krata):
2      delta = 0
3      if x>0:
4          delta += krata[x-1,y]
5      if x<N-1:
6          delta += krata[x+1,y]
7      if y>0:
8          delta += krata[x,y-1]
9      if y<N-1:
10         delta += krata[x,y+1]
11
12     return(np.exp(2*beta*(delta))/(np.exp(2*↩
       beta*(delta))+1))
```

Funkcja wyznacza wartość funkcji akceptacji próbnika Gibbsa, wykorzystywanego w algorytmie Proppa-Wilsona dla kraty.

Funkcja zwraca taką wartość, ponieważ z [1] wiemy, że

$$\pi_{G,\beta}\left(X(x) = +1 | X(V \setminus \{x\}) = \xi\right) = \frac{exp(2\beta(k_+(x,\xi) - k_-(x,\xi)))}{exp(2\beta(k_+(x,\xi) - k_-(x,\xi))) + 1},$$

gdzie $\delta = k_+(x,\xi) - k_-(x,\xi)$, zaś $k_+(x,\xi)$ i $k_-(x,\xi)$ to odpowiednio liczba sąsiadów o spinie $+1$ i liczba sąsiadów o spinie -1 naszego wylosowanego punktu x. V to są wierzchołki z modelu Isinga, $X \in \{-1,+1\}^V$ jest losowym stanem modelu Isinga, a $\xi \in \{-1,+1\}^{V \setminus x}$ to X z wyłączeniem punktu x.

Próbnik Gibbsa dla grafu pełnego

```
1 def update2(beta, x, y, graf, energia_grafu):
2     delta = energia_grafu - graf[x,y]
3 
4     return(np.exp(2*beta*(delta))/(np.exp(2*↩
    beta*(delta))+1))
```

Funkcja wyznacza wartość funkcji akceptacji próbnika Gibbsa, wykorzystywanego w algorytmie Proppa-Wilsona dla grafu pełnego. Funkcja działa na tej samej zasadzie jak funkcja **update**, jedyna różnica to sposób zliczania spinów sąsiadów wierzchołka (x,y).

Algorytm Proppa-Wilsona - krata

```
1  N = 2100
2  beta = 0.04
3  KrataP = np.ones((N,N))
4  KrataN = -1*np.ones((N,N))
5  rzuty = [random()]
6  while (KrataN == KrataP).all() == False:
7      for i in range(len(rzuty)):
8          x, y = randint(0,N,2)
9          KrataP[x,y] = 1 if rzuty[-(i+1)] < ↩
       update(beta, x, y, KrataP) else -1
10         KrataN[x,y] = 1 if rzuty[-(i+1)] < ↩
       update(beta, x, y, KrataN) else -1
11     rzuty +=[random() for i in range(len(rzuty)↩
       )]
```

Powyżej prezentujemy implementację algorytmu Proppa-Wilsona dla kraty. Stosujemy tutaj metodę sandwichingu, która sprawia, że algorytm Proppa-Wilsona wymaga puszczenia tylko dwóch łańcuchów markowa - startujących ze stanów o wszystkich spinach równych 1 i wszystkich spinach równych -1 odpowiednio - a nie 2^k łańcuchów, gdzie k to liczba konfiguracji w modelu Isinga. Z tego względu sandwiching umożliwia nam realizację algorytmu dla dużych k.

Przykładowo, rozpatrzmy model Isinga o macierzy 2100×2100 i $\beta = 0.04$. Losujemy zmienną losową U_{-1} z rozkładu jednostajnego na $[0, 1]$, a następnie punkt, który chcemy potencjalnie zamienić. Zamiany dokonujemy na podstawie odpowiedniej funkcji akceptacji dla macierzy samych $+1$ i macierzy samych -1 oddzielnie. Jeśli w chwili 0 łańcuchy markowa startujące z tych macierzy nie znajdą się w tym samym stanie, podwajamy liczbę zmienną losowych U z rozkładu jednostajnego, jedocześnie pamiętając stare U i puszczamy algorytm ponownie. Pełny opis algorytmu znajdziemy w [1], rozdział **The Propp-Wilson Algorithm**. W powyższym przykładzie dla macierzy 2100 x 2100 kod liczył się 1 godzinę i trzeba było przechować 134217728 U w pamięci.

Algorytm Proppa Wilsona - graf pełny

```
N = 40
beta = 0.05
KrataP = np.ones((N,N))
KrataN = -1*np.ones((N,N))
energiaP = N**2
energiaN = -N**2

rzuty = [random()]
```

Algorytm Proppa Wilsona - graf pełny

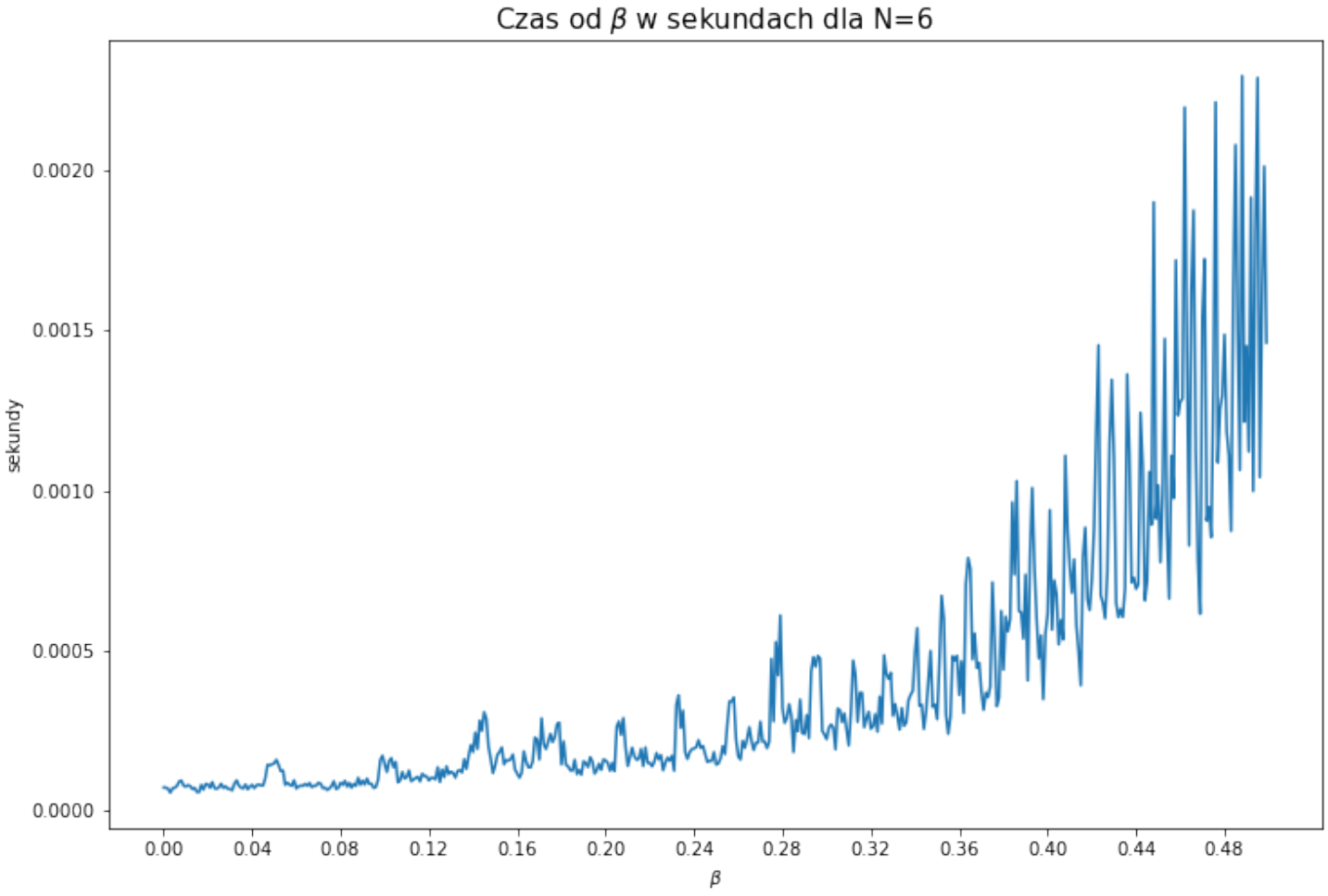
```
while (KrataN == KrataP).all() == False:
    for i in range(len(rzuty)):
        x, y = randint(0,N,2)
        KrataP_xy = KrataP[x,y]
        KrataN_xy = KrataN[x,y]
        KrataP[x,y] = 1 if rzuty[-(i+1)] < update2(beta, x, y, KrataP, energiaP) else -1
        KrataN[x,y] = 1 if rzuty[-(i+1)] < update2(beta, x, y, KrataN, energiaN) else -1

        if KrataP_xy != KrataP[x,y]:
            energiaP += 2*KrataP[x,y]
        if KrataN_xy != KrataN[x,y]:
            energiaN += 2*KrataN[x,y]

    rzuty +=[random() for i in range(len(rzuty))]
```

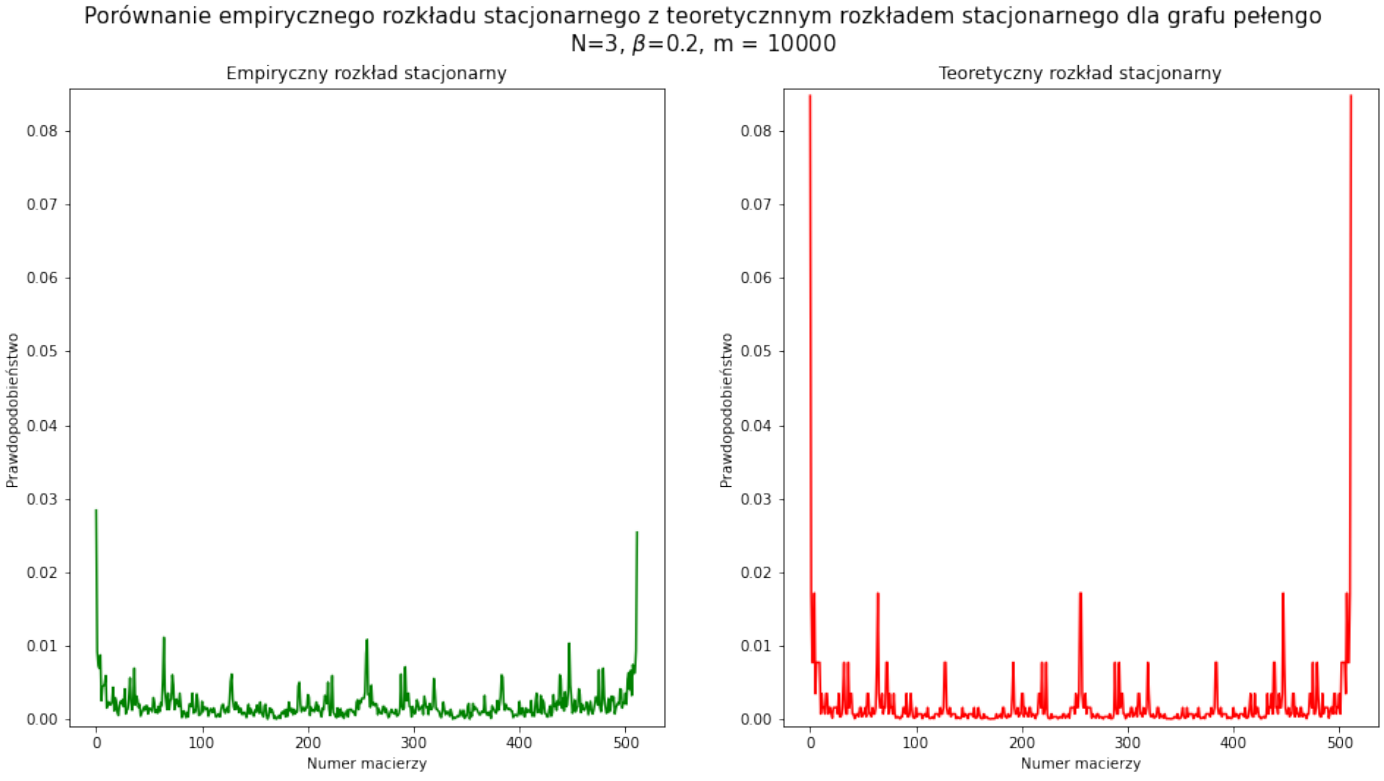
Powyżej przedstawiamy kod algorytmu Proppa-Wilsona dla grafu pełnego. Działa on na tej samej zasadzie co poprzedni, jedyną różnicą jest przechowywanie dodatkowych zmiennych reprezentujących energię odpowiednich układów, która jest potrzebna do wyznaczenia sumy spinów sąsiadów w kolejnych iteracjach.

Zależność czasu kompilacji od β

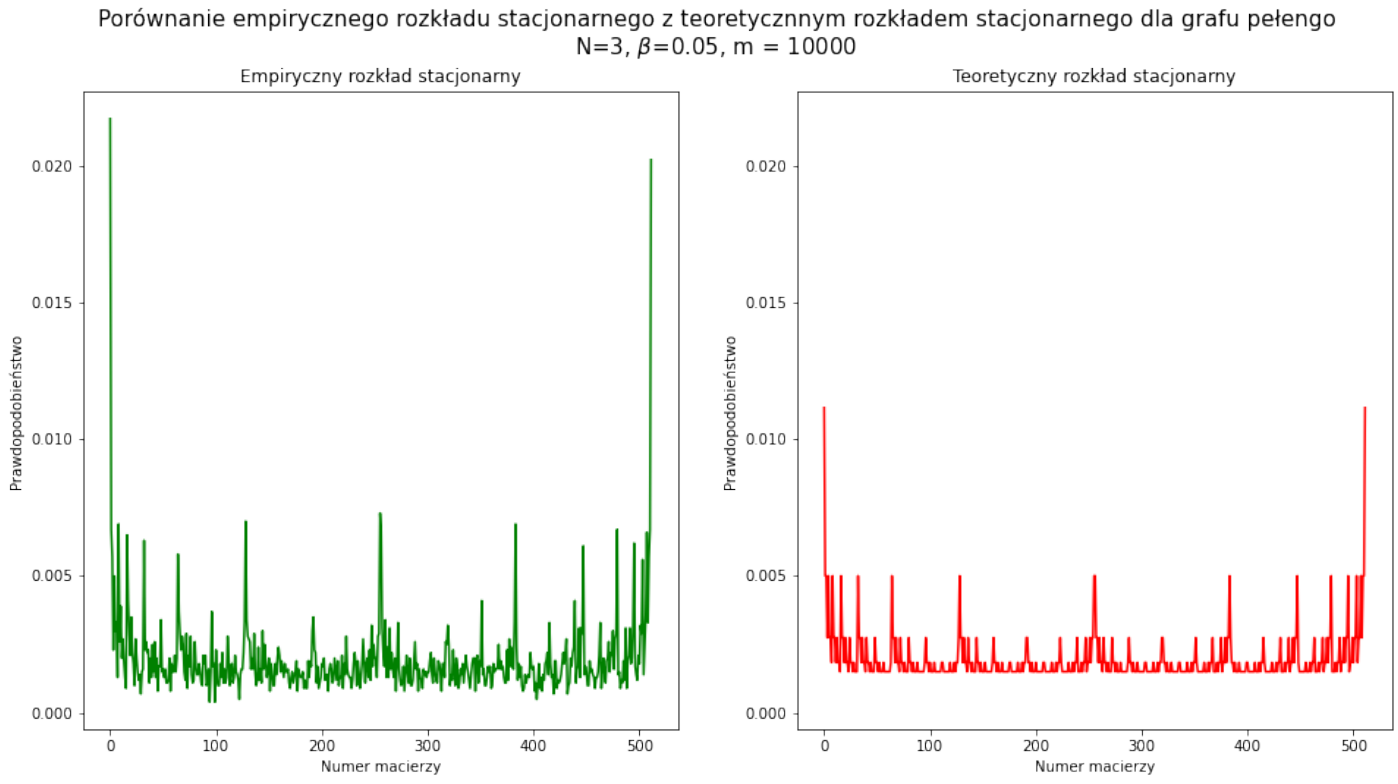


Z powyższego wykresu od razu widać, że wraz ze wzrostem β, czas potrzebny do symulacji łańcucha rośnie wykładniczo. Z teorii wiemy, że od $\beta = 0.441$ czas na wykonanie algorytmu powinien drastycznie wzrosnąć i to też widzimy na wykresie.

Rozkład stacjonarny dla kraty



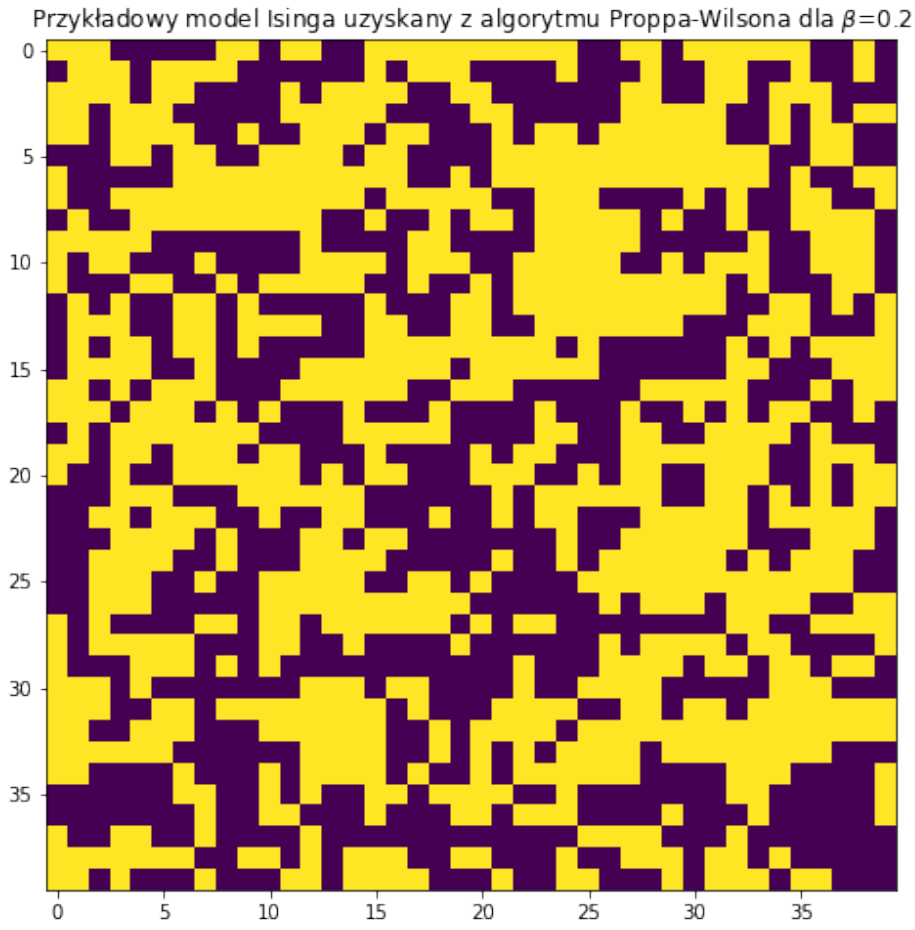
Rozkład stacjonarny dla grafu pełnego



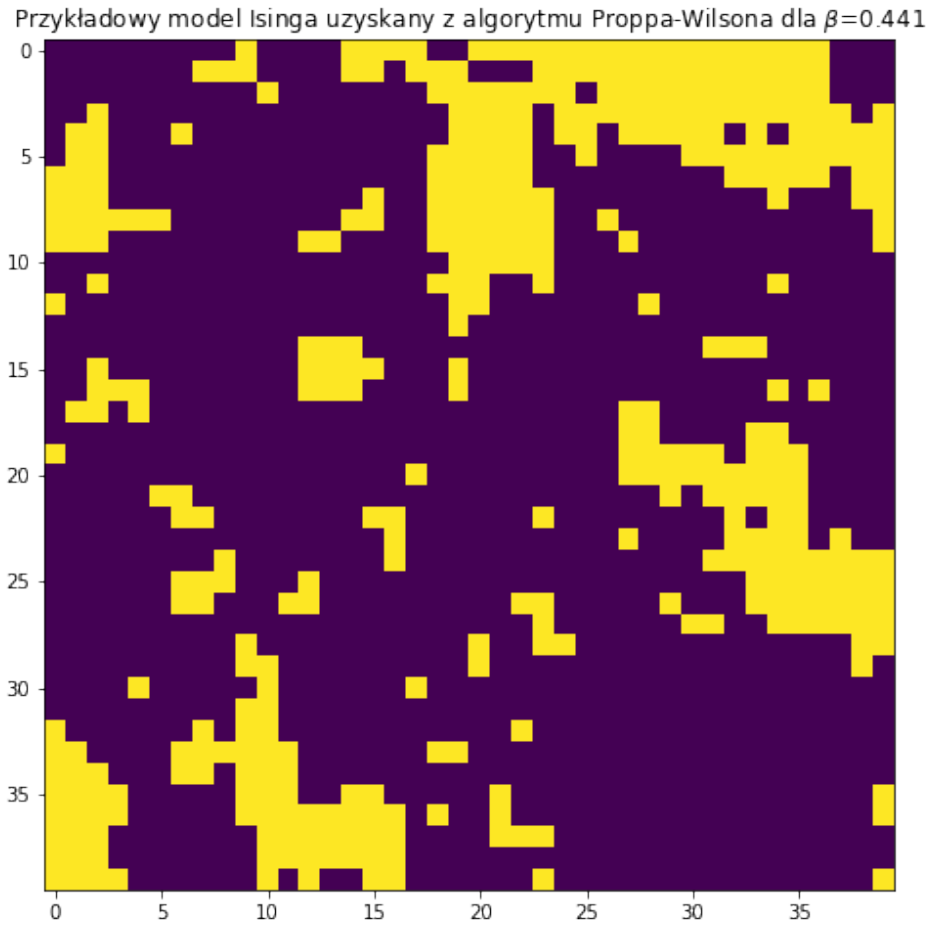
Co prawda celem projektu nie jest znajdywanie dokładnego rozkładu stacjonarnego, tylko możliwość próbkowania z niego, ale chcieliśmy sprawdzić, czy nasz kod rzeczywiście działa. Dla macierzy 3×3 istnieje 512 stanów i wyznaczyliśmy prawdopodobieństwo każdego z nich za pomocą wzoru, wykres po prawej przedstawia te prawdopodobieństwa. Puściliśmy nasz kod 10000 razy i znormalizowaliśmy wyniki występowania łańcucha na końcu algorytmu. Jak widać, rozkłady prawdopodobieństwa mają podobny kształt, więc dobrze zaimplementowaliśmy algorytm.

Próbka modelu Isinga dla $\beta = 0.05$

Próbka modelu Isinga dla $\beta = 0.2$



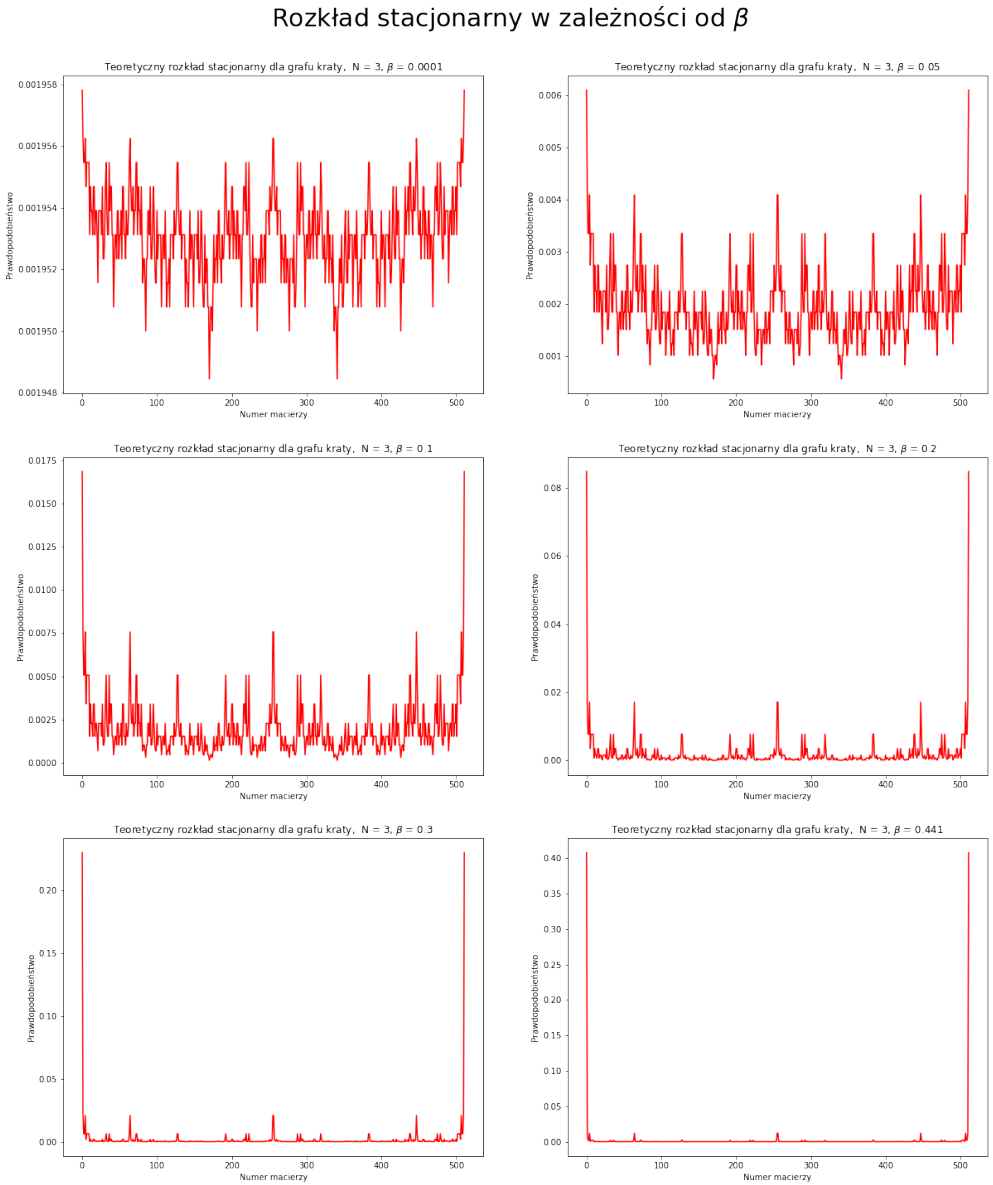
Próbka modelu Isinga dla $\beta = 0.441$



Opis

Powyżej przedstawiliśmy przykładowe macierze 20×20, które powstały na końcu algorytmu, dla poszczególnych β. Z fizyki wynika, że dla większych β powinny być większe skupiska jednego spinu, co dobrze obrazują te przykładowe wykresy.

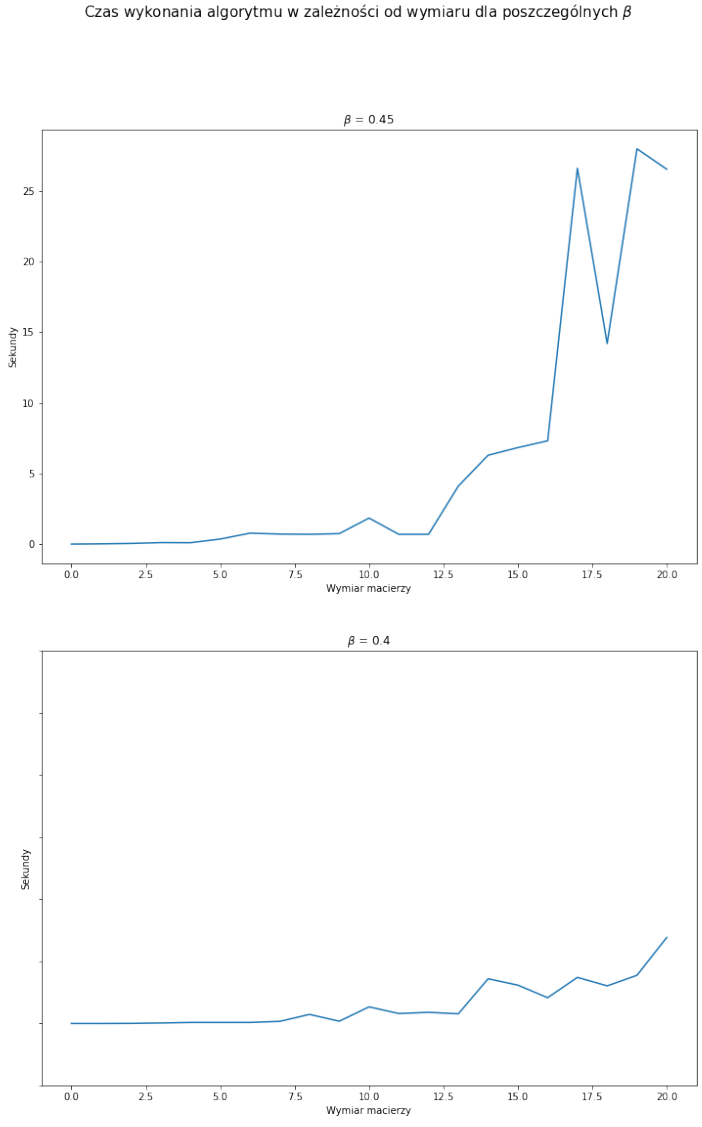
Zależność rozkładu stacjonarnego od β



Opis

Chcieliśmy też zobaczyć, jaki wpływ ma β na rozkład stacjonarny i sprawdziliśmy to dla macierzy 3×3. Jak widać, wraz ze wzrostem β preferowane są stany ze spinami o takim samym znaku, ponieważ mają mniej energii.

Zależność czasu wykonania algorytmu od wymiaru macierzy dla konkretnych β



Opis

Z teorii wiemy, że dla $\beta > 0.441$ powinny być problemy związane z kompilacją. Dla $\beta = 0.45$ oraz $\beta = 0.4$ zbadaliśmy, jakie macierze będą dłużej się liczyły niż 4 minuty. Dla $\beta = 0.45$, od $N = 16$ czas wykonania rośnie wykładniczo z każdym powiększeniem wymiaru, w przeciwieństwie do przypadku $\beta = 0.4$, gdzie w miarę liniowo zwiększa się czas wykonania. Z tego wynika, że temperatura krytyczna równa $\beta = 0.441$ istnieje, dla $\beta > 0.441$ czas wykonania może być strasznie długi dla większy macierzy, lub algorytm może wcale się nie wykonać.

Bibliografia

1. Olle Häggström "Finite Markov Chains and Algorithmic Applications"
2. https://www.youtube.com/watch?v=ULMokGuoHzI