Relazione di Laboratorio Computazionale

Alessio Marchetti

Le parti in grassetto sono commenti generici o placeholder perchè QUESTA È UNA BOZZA!

Abstract

In questa relazione prenderemo in considerazione il problema di estrarre campioni di valori casuali data una certa distribuzione di probabilita‘ discreta. Supporremo di sapere generare variabili uniformi sull'intervallo $[0, 1]$, che verranno implementate operativamente come le variabili generate dalla libreria `numpy`.

Una prima soluzione del problema è quella di dividere $[0, 1]$ in intervalli di lunghezza pari a ciascuna componente del vettore di probabilità, e scegliere il risultato in funzione dell'intervallo a cui appartiene una variabile uniforme. Questo presenta diversi inconvenienti: il metodo infatti richiede un numero di somme proporzionale al numero di componenti del vettore di probabilità. Questo potrebbe essere intrattabile quando molto grande. Inoltre spesso il vettore di probabilità è noto solo a meno di un coefficiente di normalizzazione, il cui calcolo richiederebbe di nuovo $O(n)$ somme.

Uno dei metodi più utilizzati per ovviare a questi problemi è il metodo di Monte Carlo (abbreviato spesso con MCMC, Markov Chain Monte Carlo), che consiste nella simulazione di una camminata su una catena di Markov con distribuzione invariante la distribuzione data. Dopo un numero sufficiente di step, la frequenza di visita di un nodo sarà arbitrariamente vicina a quella voluta. In questo caso però il numero di passi necessari ad una determinata distribuzione non è noto a priori ed è di difficile calcolo.

Si andrà dunque a presentare l'algoritmo di Propp-Wilson, una modifica del MCMC che ha il vantaggio di ottenere la distribuzione esatta e di terminare una volta raggiunta questa. Applicheremo tale algoritmo al modello di Ising, una modellizzazione del comportamento magnetico della materia.

1 Il modello di Ising

Sia $G = (V, E)$ un grafo non orientato. Considereremo dunque l'arco $(x, y$ uguale all'arco (y, x). I vertici andranno a rappresentare i singoli atomi di un materiale, e gli archi indicano quali atomi interagiscono fra loro. Ad ogni atomo viene quindi associato uno spin che può essere $+1$ o -1. Una configurazione è quindi una funzione $f\colon V \to \{+1, -1\}$. A ciascuno di questi modelli si associa l'energia

$$H(f) = -\sum_{(x,y)\in E} f(x)f(y).$$

Inoltre viene dato un parametro reale del sistema $\beta \geq 0$ detta temperatura inversa. Il modello di ising associa ad ogni configurazione $f \in \{+1, -1\}^V$ la probabilità

$$\pi(f) = \frac{1}{Z}\exp(-\beta H(f))$$

Dove Z è il coefficiente di normalizzazione pari a

$$Z = \sum_{f\in\{+1,-1\}^V} \exp(-\beta H(f))$$

L'obiettivo che ci prefiggiamo è quello di estrarre un campione da $\{+1, -1\}^V$ con probabilità π.

2 Metodo di Monte Carlo

Come prima cosa, data una distribuzione π su un insieme Ω, cerchiamo una matrice di transizione P per una catena di Markov omogenea che abbia π come misura invariante. Per farlo chiediamo una condizione più forte su P, cioè la reversibilità, ovvero si vuole che $\pi(i)P_{i,j} = \pi(j)P_{j,i}$. Se la matrice P è irriducibile e aperiodica, data una distribuzione iniziale π_0, la distribuzione di probabilità dopo n passi $\pi_n = \pi_0 P^n$ converge a π.

Cerchiamo allora le matrici del tipo $P_{i,j} = A_{i,j}Q_{i,j}$ per $i \neq j$, dove la matrice Q è irriducibile e viene detta matrice generatrice dei candidati, e A è da terminare in modo tale che P abbia le proprietà richieste e $0 \leq A \leq 1$. Le componenti $P_{i,i}$ sono determinate in maniera tale che P sia stocastica. A livello di interpretazione si può pensare che a ogni step si sceglie vertice candidato a cui passare con probabilità dettata da Q e poi si esegue il passagio con probabilità dettata da A, altrimenti si rimane nel vertice di partenza. Per questo motivo A è la matrice delle probabilità di accettazione.

Generalmente si sceglie A della forma

$$A_{i,j} = \frac{S_{i,j}}{1 + T_{i,j}}$$

con

$$T_{i,j} = \frac{\pi(i)Q_{i,j}}{\pi(j)Q_{j,i}}$$

e S una matrice simmetrica. L'algoritmo di Metropoli-Hastings sceglie

$$A_{i,j} = \min\left(1, \frac{\pi(i)Q_{j,i}}{\pi(j)Q_{i,j}}\right).$$

In questo modo si determina una P con tutte le proprietà richieste.

3 Il Gibbs sampler

Vogliamo specializzare il MCMC ai casi in cui l'insime degli stati sia un insieme di funzioni (con dominio e codominio finiti). Questa capita nel modello di Ising che studieremo in seguito, per esempio. Abbiamo dunque un processo stocastico a valori in $E = \Lambda^V$ per certi insiemi Λ e V.

Con il Gibbs sampler il processo passa da uno stato $X_n = f$ a $X_{n+1} = g$ con le seguenti regole: sia v un elemento scelto in modo casuale uniforme da V. Allora f e g devono coincidere su $V \setminus \{v\}$. Il punto $g(v)$ assume il valore λ con probabilità $\mathbb{P}\left[X(v) = \lambda \mid X(V \setminus \{v\}) = f(V \setminus \{v\})\right]$. Si verifica che la distribuzione cercata soddisfa la condizione di reversibilità e che la catena risultante è irriducibile e aperiodica. Dunque si ha la converganza del metodo di Monte Carlo.

In particolare il Gibbs sampler è un caso particolare del MCMC con Metropoli-Hastings. In effetti scegliamo la matrice dei candidati Q con le probabilià sopra definite. A queto punto si verifica che tutte le entrate di A sono pari a 1, e dunque $P = Q$.

4 Coupling from the past

In questa sezione svilupperemo un metodo per estrarre un campione con una probabilità esatta π, data P una matrice di transizione irriducibile e aperiodica Su un insieme finito di stati $S = \{s_1, \ldots, s_n\}$ e con probabilità invariante π. Possiamo associare alla catena di Markov definita da P una funzione di transizione

$$f \colon S \times [0,1] \longrightarrow S$$

tale che se U è una variabile uniforme sull'intervallo $[0,1]$, allora

$$\mathbb{P}\left[f(s_i, U) = s_j\right] = P_{i,j} \qquad \forall\ s_i, s_j \in S.$$

Siano U_n variabili uniformi su $[0,1]$ per ogni $n \in \mathbb{Z}$. Costruiamo allora delle sequenze $X_m^r(i)$ con $i = 1, \ldots, n$, $r \leq 0$ e $m \geq r$ interi nel modo seguente:

$$X_r^r(i) = s_i$$

$$X^r_{m+1}(i) = f(X^r_m(i), U_m).$$

Sia ora

$$\tau^- = \max(\ r\ \$\ X^r_0(1) = \cdots = X^r_0(n))$$

e definiamo $Y = X_0^{\tau^-}(1)$. La condizione con cui è stato definito τ^- è detta coalescenza e τ^- è detto tempo di accoppiamento all'indietro.

Si verifica che Y ha distribuzione pari a π.

Questo metodo è noto come algoritmo di Propp-Wilson.

Esempio. Proviamo a descrivere una semplificazione dell'algoritmo di Propp-Wilson e descriviamo un esempio per cui essa non funziona. Come sopra prendiamo le U_m variabili indipendenti uniformi sull'intervallo unitario con m che varia sui numeri naturali. Consideriamo le n catene $X(i)$ in modo simile a prima, partendo però da 0, ovvero

$$X_0(i) = s_i$$

e

$$X_{m+1}(i) = f(X_m(i), U_m).$$

Consideriamo il tempo di accoppiamento nel futuro

$$\tau^+ = \min(\ m\ \$\ X_m(1) = \cdots = x_m(n)\ )$$

e il rispettivo stato $Y = X_{\tau^+}(1)$. In questo caso però Y non ha la distribuzione π richiesta. Per farlo vedere prendiamo l'insieme degli stati composto da due elementi $S = \{s_1, s_2\}$ e la seguente matrice di transizione:

$$P = \begin{pmatrix} 0.5 & 0.5 \\ 1 & 0 \end{pmatrix}.$$

La catena ha probabilità invariante $\pi = (\frac{2}{3}, \frac{1}{3})$. Per definizione, le due catene al tempo τ^+ si trovano nello stesso stato, e al tempo $\tau^+ - 1$ si trovano una in s_1 e l'altra in s_2. La seconda allora al passo τ^+ deve andare in s_1 con probabilità 1. Quindi con questo algoritmo, $Y = s_1$ quasi certamente.

Pensando all'algoritmo, a priori dovremmo simulare la catena X^{-1}, vedere se si ha la coalescenza, e in caso negativo simulare X^{-2} e procedere in maniera analoga. Questo richiederebbe in effetti il calcolo di un numero di step della catena di Markov pari a $1 + \cdots + \tau^- = O((\tau^-)^2)$. Notiamo però che se $r' > \tau^-$ si ha che $X_0^{r'}(i) = Y$. Questo rende possibile calcolare le catene X^{-2^k} ottenendo lo stesso risultato. Il numero di step richiesto in questo caso è $2^0 + 2^1 + \cdots + 2^c$ dove c è il più piccolo intero tale che $2^c \geq \tau^-$, in particolare si ha $2^c < 2\tau^-$. La somma viene dunque maggiorata da $4\tau^-$, un costo decisamente vantaggioso rispetto al precedente.

5 Sandwiching

Il metodo presentato sopra è funzionante e risolve gli scopi che ci eravamo prefissi, tuttavia simulare tutte le n catene di Markov è praticamente impossibile per n grandi. In questa sezione ci occuperemo di migliorare il metodo per renderlo computazionalmente più leggero.

In questo caso supporremo di avere un ordine parziale tra gli stati in S che denoteremo con il simbolo $\preceq$. Richiediamo che la funzione di transizione rispetti la condizione di monotonia

$$f(s_i, u) \leq f(s_j, u) \qquad \forall\ s_i \preceq s_j,\ u \in [0,1]$$

e che esistano in S un massimo e un minimo, che senza perdita di generalità chiameremo s_1 e s_n. Si verifica facilmente che la coalescenza di X_0^r si verifica se e solo se $X_0^r(1) = X_0^r(n)$. Inoltre il tempo di coalescenza τ^- è quasi certamente finito e la variabile $Y = X_0^{\tau^-}(1)$ ha distribuzione π.

Dimostrazione. Notiamo come prima cosa che il tempo τ^+ (definito come nell'esempio sopra) è dominato dal primo m positivo per cui $X_m(n) = s_1$. Questo m è finito perché la catena di Markov in considerazione è ergodica.

Ora mostriamo che per ogni k naturale si ha che $\mathbb{P}[\tau^- \leq k] = \mathbb{P}[\tau^- \leq k]$. Consideriamo la successione di valori casuali per valutare gli step U_m, e trasliamola indietro di k posizioni, cioè prendiamo $U'_m = U_{m-k}$. Sia τ' il tempo di accopiamento all'indietro realtivo alla successione U'. Ovviamente si ha che τ' e τ^- sono variabili identicamente distribuite. Supponiamo che valga $\tau^+ \leq k$. Allora nel modello degli U' si ha la coalescenza al tempo $\tau^+ - k \leq 0$, e quindi abbiamo trovato che $\tau' \leq k$. Dunque

$$\mathbb{P}[\tau^+ \leq k] \leq \mathbb{P}[\tau' \leq k] = \mathbb{P}[\tau^- \leq k].$$

Viceversa prendiamo $\tau' \leq k$. Ragionando in modo analogo a prima, otteniamo che la catena originaria ha coalescnza in avanti prima del tempo k, e si ha la disuguaglianza opposta.

Mettendo assieme i due pezzetti si trova che τ^- è quasi certamente finito.

Verifichiamo ora la distribuzione di Y. Per $r \geq \tau^-$ vale

$$\begin{aligned}
\mathbb{P}[Y = j] &= \mathbb{P}[Y = j,\ \tau^- > n] + \mathbb{P}[Y = j,\ \tau^- \leq n] \\
&= \mathbb{P}[Y = j,\ \tau^- > n] + \mathbb{P}[X_0^r(i) = j,\ \tau^- \leq n] \\
&= \mathbb{P}[Y = j,\ \tau^- > n] - \mathbb{P}[X_0^r(i) = j,\ \tau^- > n] + \mathbb{P}[X_0^r(i) = j] \\
&= A_n - B_n + P_{i,j}^n.
\end{aligned}$$

Gli A_n e B_n sono limitati dall'alto da $\mathbb{P}[\tau^- > n]$, che è una quantità che tende a zero per n grande perchè il tempo di accoppiamento è quasi certamente finito. L'ultimo termine tende invece alla probabilità invariante e questo conclude la dimostrazione.

6 Implementazione del modello di Ising

Andiamo ad utilizzare le tecnice viste al modello di Ising. Innanzi tutto ci ridurremo al caso in cui il grafo è una griglia quadrata di lato l. Sui possibili stati di un grafo, consideriamo la relazione d'ordine definita da: $f \preceq g$ se e solo se per ogni vertice v si ha che $f(v) \leq g(v)$. Con questa realzione si hanno due stati particolari, le due funzioni costanti, che sono il massimo e il minimo. Lo scopo è quello di simulare le catene di Markov che partono da questi stati.

Sia ξ una possibile configurazione sul grafo (V, E). Sia inoltre v un vertice e consideriamo le configurazioni ξ_p e ξ_n che sono uguali a ξ in tutti i punti tranne v, dove valgono rispettivamente $+1$ e -1. Andiamo allora a calcolare il rapporto

$$\begin{aligned}
\frac{\pi(\xi_p)}{\pi(\xi_n)} =& \frac{\exp(\beta \sum_{(y,z)\in E} \xi_p(y)\xi_p(z))}{\exp(\beta \sum_{(y,z)\in E} \xi_n(y)\xi_n(z))} \\
=& \exp\left[\beta \sum_{(y,z)\in E} (\xi_p(y)\xi_p(z) - \xi_n(y)\xi_n(z))\right] \\
=& \exp\left[\beta \sum_{(v,y)\in E} (\xi_p(v)\xi_p(y) - \xi_n(v)\xi_n(y))\right] \\
=& \exp\left[\beta \sum_{(v,y)\in E} (\xi_p(v) - \xi_n(v))\,\xi(y)\right] \\
=& \exp\left[2\beta \sum_{(v,y)\in E} \xi(y)\right] = \exp\left(2\beta\kappa(\xi, v)\right)
\end{aligned}$$

dove $\kappa(f, x)$ indica la somma di tutti i valori di f nei vertici adiacenti a x.

Sia X una variabile aleatoria con la distribuzione del modello di Ising. Andiamo allora a calcolare la probabilità condizionale del Gibbs sampler.

$$\begin{aligned}
\eta(\xi, v) =& \mathbb{P}\left[X(v) = +1 \mid X(V \setminus \{v\} = \xi\right] \\
=& \frac{\pi(\xi_p)}{\pi(\xi_p) + \pi(\xi_n)} = \frac{\exp(2\beta\kappa(\xi, v))}{\exp(2\beta\kappa(\xi, v)) + 1}
\end{aligned}$$

Dunque per compiere uno step nella catena di Markov partendo da uno stato X_m, si sceglie uniformemente un vertice v e lo si pone a $+1$ con probabilità $\eta(X_n, v)$. Cioè si prende una variabile uniforme sull'intervallo unitario U_m e si pone v a $+1$ se $U_m < \eta(X_m, v)$. Definiamo la funzione di aggiornamento

$$\Phi(\xi, u)(v) = \begin{cases} +1 & \text{se } u < \eta(\xi, v) \\ -1 & \text{altrimenti} \end{cases}$$

e $\Phi(\xi, u)(w) = \xi(w)$ per tutti i vertici w diversi da v.

Al fine di avvantaggiarci con la tecnica del sandwiching, bisogna ancora verificare che queste operazioni conservano l'ordine parziale. Prendiamo quindi $\xi \preceq \xi'$ due possibili stati, un vertice v e un reale $u \in [0, 1]$. Notiamo intanto che

$$\kappa(\xi, v) = \sum_{(v,y)\in E} \xi'(y) \leq \sum_{(v,y)\in E} \xi(y) = \kappa(\xi', v).$$

Ne segue che

$$\eta(\xi, v) = \frac{\exp(2\beta\kappa(\xi, v))}{\exp(2\beta\kappa(\xi, v)) + 1} \leq \frac{\exp(2\beta\kappa(\xi', v))}{\exp(2\beta\kappa(\xi', v)) + 1} = \eta(\xi', v)$$

e dunque si ha anche la monotonia di Φ nel primo argomento.

Si fornisce in seguito l'implementazione dell'algoritmo descritto in Python. In particolare la funzione `get_ising` accetta un intero `N` e il valore `beta` e restituisce una matrice con un campione del modello di Ising sul grafo fatto come una griglia quadrata di lato `N`.

```
import numpy as np
import numpy.random as rnd

class Random_gen:
    def __init__(self, N):
        self.N = N
        self.depth = 0
        self.rarr = np.zeros((0,), dtype=float)
        self.narr = np.zeros((0,2), dtype=int)

    def advance_depth(self, nd):
        self.rarr.resize((nd,))
        self.narr.resize((nd,2))
        diff = nd - self.depth
        self.rarr[self.depth:] = rnd.random((diff,))
        self.narr[self.depth:] = rnd.randint(0, self.N,
            size=(diff,2))
        self.depth = nd

def multiple_step(M, beta, rarr, narr):
    n = rarr.size
    for i in range(n):
        index = n - i - 1
        r = rarr[index]
        x = narr[index,0]
        y = narr[index,1]
        acc = 0
        if x > 0:
                acc += M[x-1,y]
        if y > 0:
                acc += M[x,y-1]
        if y < N-1:
                acc += M[x,y+1]
        if x < N-1:
                acc += M[x+1,y]
        a = np.exp( 2 * beta * acc )
        u = a / (a+1)
        if r < u:
            M[x, y] = 1
```

```
39          else:
40              M[x, y] = -1
41
42  def check_coalescence(M1, M2):
43      for i in range(N):
44          for j in range(N):
45              if M1[i,j] != M2[i,j]:
46                  return False
47      return True
48
49  def get_ising(N, beta):
50      depth = 1
51      ran = Random_gen(N)
52      ran.advance_depth(depth)
53      while True:
54          up = np.ones((N,N), dtype=int)
55          down = np.full((N,N), -1, dtype=int)
56          multiple_step(up, beta, ran.rarr, ran.narr)
57          multiple_step(down, beta, ran.rarr, ran.narr)
58          if check_coalescence(up, down):
59              break
60          else:
61              depth = max(depth+1, depth*2)
62              ran.advance_depth(depth)
63      return up
```

Provando a fare girare il programma a diversi valori di β e della dimensione della griglia, notiamo, come mostrato in 1 che i tempi di attesa diventano molto lunghi per la temperatura inversa che si avvicina a 0.4. In particolare notiamo che la crescita del numero di step necessari alla coalescenza è circa esponenziale in β. Per ovviare a questo problema, che a livello pratico impedisce alla computazione di terminare per molti valori dei parametri, nella prossima sezione si presenterà un miglioramento del metodo.

7 Random Cluster

I risultati originali qui presentati sono stati trovati da Fortuin e Kasteleyn e sono stati usati da Propp e Wilson nel loro articolo. Purtoppo non sono riuscito a trovare l'originale. In Edwards-Sokal (1988) c'è una bozza di dimostrazione che ho sviluppato e riportato qui.

Consideriamo un grafo non orientato $G = (V, E)$, sia poi Σ lo spazio delle possibili configurazioni di un modello di Ising su G, ossia $\{0, 1\}^V$. Siano inoltre Ω l'insieme dei sottografi di G, che identificheremo anche con l'insieme delle parti di E, e $\bar{\omega}\colon \Sigma \longrightarrow \Omega$ la funzione tale che

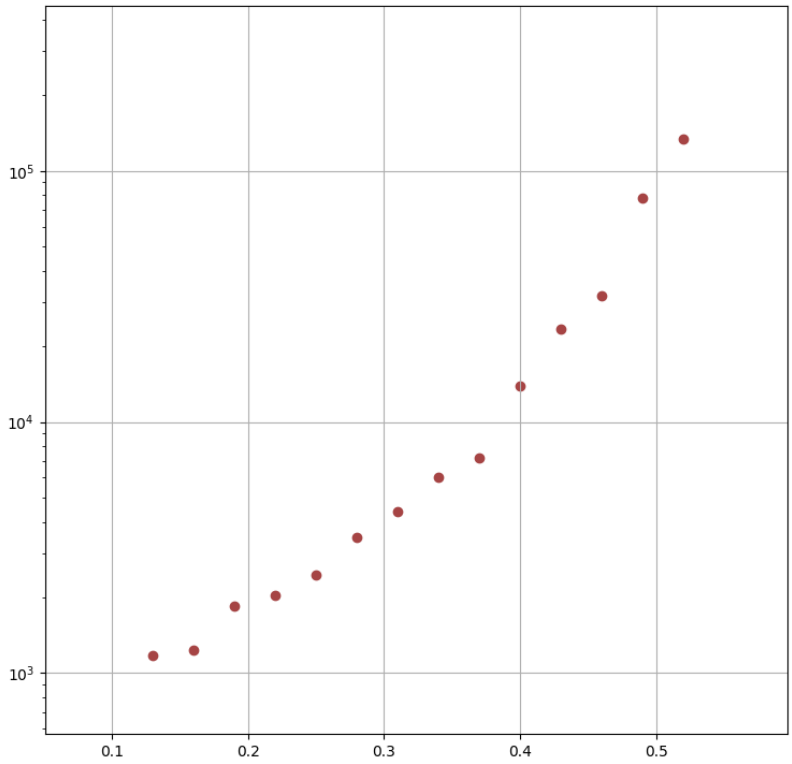
$$\bar{\omega}(\sigma) = \{(i, j) \in E \ \$ \ \sigma(i) = \sigma(j)\}$$

e che associa ad ogni configurazione σ il più grande sottografo di G tale per cui σ è costante sulle sue componenti connesse.

Consideriamo la distribuzione di probabilità definita su $\Sigma \times \Omega$ come

$$\mu_p(\sigma, \omega) = \frac{1}{Z_{FK}} p^{|\omega|}(1-p)^{|E|-|\omega|}\mathbb{1}(\omega \subseteq \bar{\omega})$$

Figura 1: Numero di step richiesti per la coalescenza per una griglia di lato 10.



dove Z_{FK} è un opportuno coefficiente di normalizzazione e $\mathbb{1}$ vale 1 se e solo se il suo argomento è vero e 0 altrimenti.

Calcoliamo le distribuzioni marginali su Ω e Σ.

$$\begin{aligned}\mu_p^1(\sigma) =& \sum_{\omega\in\Omega} \mu_p(\sigma,\omega) = Z_{FK}^{-1} \sum_{\omega\subseteq\bar{\omega}(\sigma)} p^{|\omega|}(1-p)^{|E|-|\omega|} \\ =& Z_{FK}^{-1} \sum_{k=0}^{|\bar{\omega}(\sigma)|} \binom{|\bar{\omega}(\sigma)|}{k} p^k (1-p)^{|E|-k} \\ =& Z_{FK}^{-1}(1-p)^{|E|-|\bar{\omega}(\sigma)|} \sum_{k=0}^{|\bar{\omega}(\sigma)|} p^k \binom{|\bar{\omega}(\sigma)|}{k} (1-p)^{|\bar{\omega}(\sigma)|-k} \\ =& Z_{FK}^{-1}(1-p)^{|E|-|\bar{\omega}(\sigma)|}\end{aligned}$$

Notiamo che vale

$$|\bar{\omega}(\sigma)| = \sum_{(x,y)\in E} \frac{\sigma(x)\sigma(y)+1}{2} = \frac{|E|}{2} - \frac{1}{2}H(\sigma)$$

che ci permette di riscrivere

$$\mu_p^1(\sigma) = Z^{-1}\exp(-\beta H(\sigma))$$

per $p = 1 - e^{-2\beta}$ e per Z che assorbe tutti i termini rimanenti (che non dipendono da σ), ritrovando così la distribuzione del modello di Ising.

L'altra distribuzione marginale vale

$$\mu_p^2(\omega) = \nu(\omega) = \sum_{\omega\in\Omega} \mu_p(\sigma,\omega) = Z_{FK}^{-1} p^{|\omega|}(1-p)^{|E|-|\omega|} |\{\sigma \ \$ \ \omega \subseteq \bar{\omega}(\sigma)\}|$$

Il numero di elementi di $\{\sigma \ \$ \ \omega \subseteq \bar{\omega}(\sigma)\}$ è pari al numero di σ costanti sulle componenti connesse di ω, cioè è il numero di possibili assegnazioni di $+1$ o -1 a esse. Se indichiamo con $C(\omega)$ il numero di componenti connesse di ω,

$$\mu_p^2(\omega) = Z_{FK}^{-1} p^{|\omega|}(1-p)^{|E|-|\omega|} 2^{C(\omega)}.$$

Come ulteriore passo, ci serve ancora conoscere la probabilità condizionata $\mu_p[(\sigma,\omega) \mid (\cdot,\omega)]$, che vale

$$\mu_p[(\sigma,\omega) \mid (\cdot,\omega)] = \frac{\mathbb{1}(\omega \subseteq \bar{\omega})}{2^{C(\omega)}}.$$

Per cui, conoscendo ω, la configurazione σ è scelta uniformemente fra tutte le configurazioni costanti sulle componenti connesse di ω.

A questo punto abbiamo una strategia per trovare una configurazione del modello di Ising con la distribuzione desiderata: si costruisce un sottografo ω con distribuzione ν con l'algoritmo di Propp-Wilson, poi ad ogni componente connessa di esso si associa il valore $+1$ o -1 con probabilità $1/2$. A prima vista questa può sembrare una macchinosa complicazione, ma in realtà l'implentazione con i cluster richiede molti meno step per la coalescenza.

8 Implementazione con Random Cluster

Identifichiamo gli elementi di Ω (sottoinsiemi di E) con le funzioni da E in $\{0,1\}$. Dati allora $\omega \in \Omega$ e $e \in E$, costruiamo ω_p, $\omega_n \in \Omega$ tali che essi concordano con ω su tutto $E \setminus \{e\}$, e inoltre $\omega_p(e) = 1$ mentre $\omega_n(e) = 0$. Poichè ω_p e ω_n differiscono per un solo arco, o hanno lo stesso numero di componenti connesse, o ω_p ha esattamente una componente connessa in meno di ω_n. Questo permette di calcolare agevolmente la probabilità condizionale del Gibbs sampler. Se X è una variabile aleatoria a valori in Ω e distribuzione ν,

$$\mathbb{P}\left[X = \omega_p \mid X(e') = \omega(e')\ \forall\, e' \neq e\right] = \frac{\nu(\omega_p)}{\nu(\omega_p) + \nu(\omega_n)} = \begin{cases} p & \text{se } C(\omega_p) \neq C(\omega_n) \\ \frac{p}{2-p} & \text{altrimenti.} \end{cases}$$

Notiamo che per $0 < p < 1$ vale che $\frac{p}{2-p} < p$, che, scelto uniformemente un arco e, consente di costruire la funzione per lo step:

$$\Psi(\omega, u)(e) = \begin{cases} \omega \cup \{e\} & \text{se } u < \frac{p}{2-p} \\ \omega \cup \{e\} & \text{se } u < p \text{ e } C(\omega_p) = C(\omega_n) \\ \omega \setminus \{e\} & \text{altrimenti.} \end{cases}$$

Su tutti gli altri archi $\Psi(\omega)$ concorda con ω.

Ordiniamo Ω per inclusione (questo ordine corrisponde a quello definito su Σ se visto Ω come insieme di funzioni) e verifichiamo la monotonia di Ψ nel suo primo argomento.

Vale intanto che se $\omega \subseteq \omega'$ e $C(\omega_p) = C(\omega_n)$ allora $C(\omega'_p) = C(\omega'_n)$. Si distinguono allora i casi:

- Se $u < \frac{p}{2-p}$ allora ω e ω' si aggiornano nello stesso modo.
- Se $u \geq p$ vale lo stesso ragionamento.
- Se $\frac{p}{2-p} \leq u < p$ e $\Psi(\omega') = \omega'_n$ allora $C(\omega'_p) \neq C(\omega'_n)$ e dunque $C(\omega_p) \neq C(\omega_n)$, concludendo che $\Psi(\omega) = \omega_n$. Cioè gli archi si aggiornano nello stesso modo.
- Se $\frac{p}{2-p} \leq u < p$ e $\Psi(\omega) = \omega_p$ allora si ragiona come nel caso precedente e si ottiene che gli archi si aggiornano nello stesso modo.
- In tutti i rimanenti casi vale che $\Psi(\omega') = \omega'_p$ e $\Psi(\omega) = \omega_n$.

In tutte le situazioni l'ordine viene mantenuto.

Per chiarezza si presenta lo pseudocodice per lo step della catena di Markov:

```
def step(omega):
    e = random_edge() # Selezione casuale di un arco
    u = random() # Selezione casuale di un reale in [0,1]
    remove(omega, e) # Viene rimosso (se presente) l'arco
    if u < p/(2-p):
        insert(omega, e) # Si inserisce l'arco

    # are_in_the_same_component ritorna vero se e solo se
    # gli estremi di e si trovano nella stessa componente
    # connessa di omega
    elif u < p and are_in_the_same_component(omega, e):
        insert(omega, e)
```

Nel codice che segue, `are_in_the_same_component` è implementato attreverso una visita BFS della componente connessa di uno dei suoi vertici. Il suo tempo è quindi dominato dalla dimensione dei cluster e quindi del numero totale di vertici.

Inoltre le configurazioni su Ω sono rappresentate come matrici $N \times N \times 2$, con N lato della griglia quadrata. L'elemento in posizione (i, j, d) rappresenta l'arco che parte dal vertice in (i, j) e va alla sua destra se $d = 0$, immdiatamente verso il basso se $d = 1$.

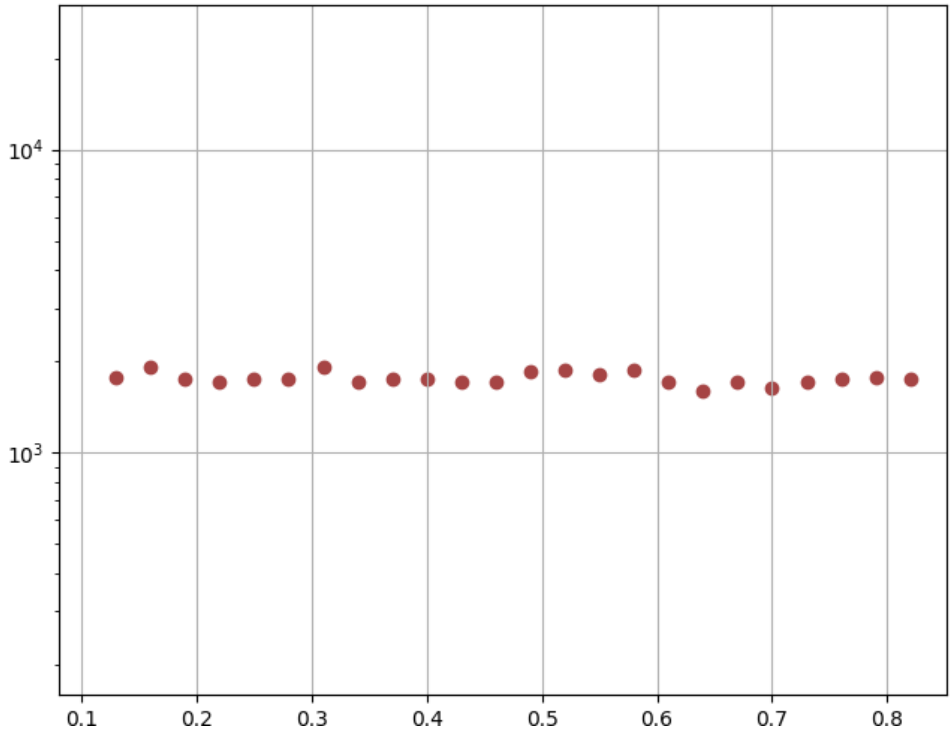
```
import numpy as np
import numpy.random as rnd
from collections import deque


class Random_gen:
    def __init__(self, N):
        self.nmax = N * N * 2
        self.depth = 0
        self.rarr = np.zeros((0,), dtype=float)
        self.narr = np.zeros((0), dtype=int)

    def advance_depth(self, nd):
        self.rarr.resize((nd,), refcheck=False)
        self.narr.resize((nd,), refcheck=False)
        self.rarr[self.depth:] = rnd.random((nd-self.depth,))
        self.narr[self.depth:] = rnd.randint(0, self.nmax,
            size=(nd-self.depth,))
        self.depth = nd

def are_in_same_cc(V, x1, x2):
    que = deque()
    x = x1
    N = V.shape[0]
    adj = [(1,0,0),(0,1,1),(-1,0,0),(0,-1,1)]
    visited = np.zeros((N,N), dtype=bool)
    visited[x] = True
    while len(que) > 0:
        x = que.popleft()
        if x == x2:
            return True
        visited[x] = True
        (i,j) = x
        if i+1 < N and V[i,j,0] == 1 and visited[i+1, j] == 0:
            que.append((i+1, j))
        if j+1 < N and V[i,j,1] == 1 and visited[i, j+1] == 0:
            que.append((i, j+1))
        if i > 0 and V[(i-1,j,0] == 1 and visited[i-1, j] == 0:
            que.append((i-1, j))
```

```
        if j > 0 and V[i,j-1,1] == 1 and visited[i, j-1] == 0:
            que.append((i, j-1))
    return False

def multiple_step(p, V, narr, rarr):
    depth = rarr.size
    N = V.shape[0]
    for i in range(depth):
        index = depth - i - 1
        u = rarr[index]
        x = narr[index]
        i = x % N
        x = x // N
        j = x % N
        x = x // N
        d = x % 2
        e = (i,j,d)
        th1 = p / (2-p)
        if d == 0:
            y1 = 1
            y2 = 0
        else:
            y1 = 0
            y2 = 1
        x2 = ((i+y1)%N, (j+y2)%N)
        x1 = (i,j)
        V[i,j,d] = 0
        if u < th1:
            V[i,j,d] = 1
        elif u < p and are_in_same_cc(V, x1, x2):
            V[i,j,d] = 1

def get_clustered(N, p):
    depth = 1
    ran = Random_gen(N)
    ran.advance_depth(depth)
    while True:
        up = np.ones((N,N,2), dtype=bool)
        down = np.zeros((N,N,2), dtype=bool)
        multiple_step(p, up, ran.narr, ran.rarr)
        multiple_step(p, down, ran.narr, ran.rarr)
        if np.array_equal(up, down):
            break
        else:
            depth = max(depth+1, depth*2)
            ran.advance_depth(depth)
    return up

def get_ising(N, beta):
    p = (1 - np.exp(-beta*2))
    V = get_clustered(N, p)
    M = np.zeros((N,N), dtype=int)
    for i in range(N):
        for j in range(N):
            if M[i,j] != 0:
                continue
            sigma = rnd.randint(0,2) * 2 - 1
            que = deque()
            que.append((i,j))
            while len(que) > 0:
                i,j = que.popleft()
                M[i,j] = sigma
                if i+1 < N and V[i,j,0] == 1 and M[i+1, j] ==
                    0:
                    que.append((i+1, j))
```

```
                if j+1 < N and V[i,j,1] == 1 and M[i, j+1] ==
                    0:
                    que.append((i, j+1))
                if i > 0 and V[(i-1,j,0] == 1 and M[i-1, j] ==
                    0:
                    que.append((i-1, j))
                if j > 0 and V[i,j-1,1] == 1 and M[i, j-1] ==
                    0:
                    que.append((i, j-1))
    return M
```

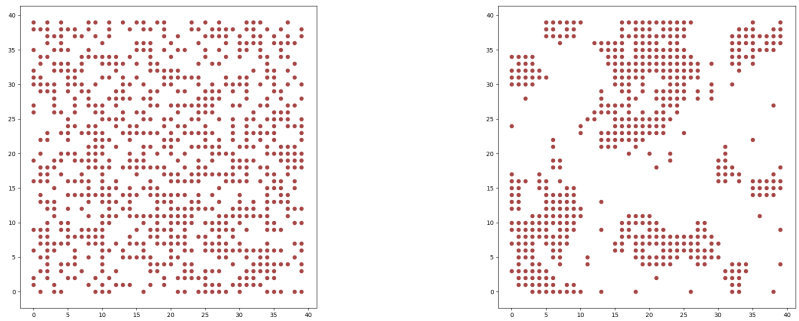
Notiamo che con i random cluster, i tempi necessari alla coalescenza rimangono stabili al veriare del parametro β, come mostrato in 2.

Figura 2: Numero di step richiesti per la coalescenza per una griglia di lato 10 con i random cluster.



9 Transizione di fase

In 3 si mostrano diverse configurazioni tipo a diverse temperature. Il punto colorato indica lo spin +1. Si nota che man mano che la temperatura diminuisce, cioè man mano che β aumenta, si tendono a formare regioni più grandi di vertici orientati nello stesso modo, e equivalentemente i cluster diventano più grandi.



(a) β=0 (b) β=0.5

Figura 3: Configurazioni con $N = 40$

Per mostrare bene questo comportamento definiamo la magnetizzazione di una configurazione σ come il valore assoluto della somma di tutti gli spin normalizzata sul numero di vertici. Andando a vedere come la magnetizzazione cambia al varirare di β in **??**, si nota che si ha una brusca salità tra i valori 0.4 e 0.5. Tale cambio di comportamento è detta transizione di fase.

Figura 4: Magnetizzazione in funzione di β per $N = 25$. Per ogni valore di β sono stati fatti 40 campionamenti, rappresentati dai punti chiari. I punti scuri sono le medie.