ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΕΦΑΡΜΟΣΜΕΝΩΝ ΜΑΘΗΜΑΤΙΚΩΝ ΚΑΙ ΦΥΣΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΟΜΕΑΣ ΦΥΣΙΚΗΣ

Μελέτη του Προτύπου 2D-Potts σε Υπολογιστικό Περιβάλλον MATLAB

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΤΟΥ

Γιάννη Ασσιώτη

Επιβλέπων Καθηγητής

Κωνσταντίνος Ν. Αναγνωστόπουλος

Αθήνα

Ιούλιος 2011

Περίληψη

Η παρακάτω διπλωματική εργασία έγινε με σκοπό την ανάπτυξη μίας εφαρμογής που να υλοποιεί προσομοιώσεις Monte Carlo για μαγνητικό σύστημα στα πλαίσια του δισδιάστατου προτύπου Potts σε προγραμματιστικό περιβάλλον MATLAB. Η εργασία χωρίζεται σε τρία μέρη. Στην αρχή του πρώτου μέρους παρατίθεται η σχετική θεωρία όπου αναπτύσσονται συνοπτικά οι βασικές αρχές που διέπουν την στατιστική φυσική και τις προσομοιώσεις Monte Carlo, προς βοήθεια αναγνωστών που δεν είναι πλήρως εξοικειωμένοι με το θέμα. Στη συνέχεια γίνεται μία παρουσίαση των εμπλεκόμενων αλγορίθμων – Metropolis, Heat-bath και Wolff. Το δεύτερο μέρος περιέχει μία περιγραφή της διαδικασίας ανάπτυξης του κώδικα για την εφαρμογή στο MATLAB. Παράλληλα γίνεται σύγκριση της απόδοσης της με μία αντίστοιχη εφαρμογή γραμμένη σε κώδικα C, στηριγμένη στον κώδικα για το πρότυπο Ising των M. E. J. Newman και G. T. Barkema. Το μέρος τελειώνει με ένα σχολιασμό των αποτελεσμάτων της τρίτης και τελευταίας υλοποίησης της εφαρμογής – η εφαρμογή που γράφτηκε στο MATLAB βρέθηκε να υστερεί σε απόδοση από την αντίστοιχη της C –και την εξαγωγή των σχετικών συμπερασμάτων ως προς την καταλληλότητα του MATLAB για την εργασία που του ανατέθηκε. Στο τρίτο μέρος βρίσκονται τα παραρτήματα όπου γίνεται παράθεση όλου του κώδικα που γράφτηκε κατά την ανάπτυξη της εφαρμογής.

Abstract

The goal of the following thesis was the development of a MATLAB application capable of implementing a Monte Carlo simulation of a magnetic system in accordance with the two-dimensional Potts model. The project is divided upon three parts. The first part begins with a brief description of the basic principles that govern statistical physics and Monte Carlo simulations to help non-specialist to familiarize themselves with the subject. Following that, is an introduction to the algorithms involved – Metropolis, Heat-bath and Wolff. In the second part we find a report on the process of developing the application in MATLAB, together with a comparison of its performance with the performance of a similar C application, based on the Ising model code of M. E. J. Newman and G. T. Barkema. The part ends with some comments on the results given by the third and last implementation – the application written in MATLAB was found to perform worse than the one written in C – and a conclusion on the suitability of MATLAB for this kind of work. The third part is the appendices which contain the code written in the development process of the application.

Περιεχόμενα

Μέρος Α΄ Πίσω από την προσομοίωση

1 Στατιστική Μηχανική ... 3

1.1 Εισαγωγή ... 3

1.2 Ισορροπία ... 5

1.2.1 Διακυμάνσεις ... 7

1.2.2 Συνάρτηση συσχετισμού ... 8

1.2.3 Πρότυπο Potts ... 9

1.3 Αριθμητικές Μέθοδοι ... 11

1.3.1 Προσομοίωση Monte Carlo ... 12

2 Αρχές Προσομοίωση Monte Carlo ... 15

2.1 Δειγματοληψία ... 15

2.1.1 Διαδικασία Markov ... 16

2.1.2 Εργοδικότητα ... 17

2.1.3 Συνθήκη Λεπτομερούς Ισοζύγισης ... 17

2.2 Λόγος Αποδοχής ... 18

3 Αλγόριθμοι για το Πρότυπο Potts ... 21

3.1 Αλγόριθμος Metropolis ... 21

3.2 Αλγόριθμος Heat-bath ... 22

3.3 Αλγόριθμος Wolff ... 24

Μέρος Β΄ Υλοποίηση των αλγορίθμων

4 Δοκιμή 1 ... 29

4.1 Εκτέλεση – Αποτελέσματα ... 31

5 Δοκιμή 2 ... 33

5.1 Εκτέλεση – Αποτελέσματα ... 34

6 Τελικές βελτιώσεις ... 37

7 Συμπεράσματα 41

Παραρτήματα

A Κώδικας υλοποίησης 45

A.1 Δοκιμή 1 45

A.2 Δοκιμή 2 56

A.3 Δοκιμή 3 68

Βιβλιογραφία 72

Μέρος Α΄

Πίσω από την προσομοίωση

1

Στατιστική Μηχανική

1.1 Εισαγωγή

Η στατιστική μηχανική ασχολείται κυρίως με τον υπολογισμό των ιδιοτήτων συστημάτων συμπυκνωμένης ύλης. Λόγω του μεγάλου αριθμού των συνισταμένων μερών που τα αποτελούν (π.χ. άτομα, μόρια), προκύπτουν διάφορες δυσκολίες στον υπολογισμό αυτό. Τα συνιστάμενα αυτά μέρη είναι συνήθως άτομα ή μόρια, του ίδιου ή μερικών διαφορετικών τύπων, και συχνά περιγράφονται από απλές εξίσωσης κίνησης, επιτρέποντας έτσι στο όλο σύστημα να εκφραστεί αναλυτικά. Ο τεράστιος όμως αριθμός των σχετικών εξισώσεων καταστεί αδύνατο κάτι τέτοιο. Αυτό γίνεται καλύτερα εμφανές αν σκεφτούμε πως σε ένα σχετικά 'μικρό' σύστημα όπως ένα δοχείο ενός λίτρου που περιέχει οξυγόνο σε κανονική θερμοκρασία και πίεση, έχουμε 3×10^{22} μόρια του αερίου. Οι αριθμοί γίνονται δραματικά μεγαλύτεροι αν θεωρήσουμε ένα μεγαλύτερο σύστημα όπως ας πούμε η γήινη ατμόσφαιρα όπου βρίσκουμε της τάξης του 10^{44} μόρια. Παρόλο όμως που είναι αδύνατο να λυθεί το σύστημα των όλων αυτών των εξισώσεων λόγο του υπερβολικά μεγάλου αριθμού τους, παρατηρούμε ότι η μακροσκοπική συμπεριφορά ενός αερίου είναι σε μεγάλο βαθμό προβλέψιμη. Γίνεται έτσι προφανές η ιδιαιτερότητα αυτών των εξισώσεων, που στο 'μέσο όρο' μας δίνουν την συμπεριφορά ολόκληρου του συστήματος. Κλασικό παράδειγμα – η πίεση ενός αερίου ακολουθεί σχετικά απλούς νόμους παρόλο που στην πραγματικότητα είναι ένας μέσος όρος της πίεσης κάθε στοιχειώδους όγκου του αερίου. Η στατιστική μηχανική ασχολείται ακριβώς με αυτό – τον υπολογισμό των ιδιοτήτων μεγάλων συστημάτων με στοχαστικό τρόπο. Αντί να ψάχνουμε τις ακριβείς λύσεις των εξισώσεων του συστήματος, ασχολούμαστε με τις πιθανότητες το σύστημα να είναι σε μία κατάσταση ή σε μία άλλη. Και αυτό γιατί ουσιαστικά οι καταστάσεις στις οποίες μπορεί να βρεθεί ένα μεγάλο σύστημα ανήκουν σε ένα μικρό υποσύνολο όλων των δυνατών καταστάσεων, και έτσι μπορούμε με αρκετή σιγουριά να πούμε ότι το σύστημα θα κινηθεί πάνω σε αυτές τις καταστάσεις.

Για να δούμε πως η στατιστική μηχανική αντιμετωπίζει αυτά τα συστήματα ας θεωρήσουμε ένα τέτοιο σύστημα που διέπεται από μία Χαμιλτονιανή H η οποία μας δίνει την ολική ενέργεια του συστήματος στην κατάσταση που βρίσκεται. Και ας θεωρήσουμε ακόμη ότι το σύστημα μπορεί να βρεθεί σε καταστάσεις διαλεγμένες από ένα διακριτό σύνολο καταστάσεων (μιας και με τέτοια συστήματα θα ασχοληθώ παρακάτω), η κάθε μία με την δική της ενέργεια. Το σύστημα αυτό είναι σε θερμική επαφή με μία δεξαμενή θερμότητας με την οποία ανταλλάσσει διαρκώς θερμότητα, ωθώντας την θερμοκρασία του προς αυτήν της δεξαμενής. Αυτό εκδηλώνεται στην Χαμιλτονιανή σαν μία ασθενής διαταραχή και γι' αυτό μπορούμε να την αγνοήσουμε στον υπολογισμό της ενέργειας. Για να εισάγουμε την δράση της δεξαμενής στο σύστημα, ορίζουμε για αυτό μία δυναμική – ένα σύνολο κανόνων που ορίζουν τον τρόπο που το σύστημα πηγαίνει από μία κατάσταση σε μια άλλη. Πριν όμως ορίσουμε την δυναμική αυτή, ας εξετάσουμε κάποια γενικά χαρακτηριστικά της που δεν εξαρτώνται από την μορφή του συστήματος.

Έστω το σύστημα μας βρίσκεται στην κατάσταση μ. Ορίζουμε ως $R(\mu \to \nu)dt$ την πιθανότητα το σύστημα να βρίσκεται στην κατάσταση ν μετά από χρόνο dt. Το $R(\mu \to \nu)$ είναι ο ρυθμός μετάβασης από την μ στην ν και συνήθως τον θεωρούμε ανεξάρτητο του χρόνου. Μπορούμε να ορίσουμε τον ρυθμό μετάβασης για κάθε δυνατή κατάσταση ν στην οποία το σύστημα μπορεί να φτάσει. Αυτοί οι ρυθμοί μετάβασης είναι πολλές φορές το μόνο που γνωρίζουμε για την δυναμική του συστήματος, που σημαίνει ότι ακόμα και αν ξέρουμε σε ποια κατάσταση μ βρίσκεται το σύστημα, μετά από ένα μικρό χρονικό διάστημα θα μπορεί να βρίσκεται σε μία άλλη, επιλεγμένη από ένα πολύ μεγάλο σύνολο καταστάσεων. Και σε αυτό το σημείο είναι που υπεισέρχεται η στοχαστική αντιμετώπιση του προβλήματος. Ορίζουμε ένα σύνολο στατιστικών βαρών $w_\mu(t)$ που αντιπροσωπεύουν την πιθανότητα να βρούμε το σύστημα στην κατάσταση μ μετά από χρόνο t. Αυτά είναι και η μόνη πληροφορία που έχουμε για την κατάσταση του συστήματος. Η χρονική εξέλιξη αυτών των βαρών δίνεται από την **δεσπόζουσα εξίσωση** (master equation):

$$\frac{dw_\mu}{dt} = \sum_\nu \left[w_\nu(t) R(\nu \to \mu) - w_\mu(t) R(\mu \to \nu) \right]$$

Ο πρώτος όρος στην δεξιά μεριά της εξίσωσης αντιπροσωπεύει τον ρυθμό με το οποίο το σύστημα 'εισέρχεται' στην κατάσταση μ, ενώ ο δεύτερος τον ρυθμό με τον οποίο το σύστημα 'φεύγει' από την κατάσταση μ προς άλλες καταστάσεις. Επειδή τα βάρη αυτά είναι ουσιαστικά πιθανότητες και το σύστημα αναγκαστικά θα βρίσκεται σε κάποια κατάσταση, πρέπει να υπακούουν στην σχέση:

$$\sum_{\mu} w_{\mu}(t) = 1$$

για όλα τα t.

Έστω τώρα ότι ενδιαφερόμαστε για μία μακροσκοπική ιδιότητα του συστήματος Q που παίρνει την τιμή Q_{μ} στην μ κατάσταση του συστήματος. Ορίζουμε την αναμενόμενη τιμή του Q στη χρονική στιγμή t για το σύστημα μας ως:

$$\langle Q \rangle = \sum_{\mu} Q_{\mu} w_{\mu}(t) \qquad (1.1)$$

Αυτή η ποσότητα εμπεριέχει σημαντικές πληροφορίες για την πραγματική τιμή του Q που θα μετρούσαμε σε ένα πείραμα. Αν για παράδειγμα ξέρουμε ότι το σύστημα βρίσκεται σίγουρα στην κατάσταση ρ τότε το $\langle Q \rangle$ θα πάρει την τιμή Q_{ρ}. Ενώ αν το σύστημα έχει την ίδια πιθανότητα να βρίσκεται σε μία από ν πιθανές καταστάσεις, τότε το $\langle Q \rangle$ θα είναι ίσο με τον μέσο όρο αυτών των καταστάσεων. Σχετικά όμως με την ακριβή σχέση του $\langle Q \rangle$ με το Q, υπάρχουν δύο διαφορετικές θεωρήσεις. Η πρώτη που είναι και πιο αυστηρή μας λέει ότι το $\langle Q \rangle$ είναι ο μέσος όρος πάνω σε ένα δείγμα που παίρνουμε από μία ταυτόχρονη μέτρηση του Q σε ένα μεγάλο δείγμα πανομοιότυπων συστημάτων, το καθένα με τη δική του δεξαμενή θερμότητας. Αυτό όμως δεν είναι συνήθως εφικτό, και γι' αυτό ανατρέχουμε στην δεύτερη θεώρηση, όπου τώρα έχουμε ένα σύστημα από το οποίο παίρνουμε μετρήσεις σε διαφορετικούς χρόνους και βρίσκουμε το χρονικό μέσο όρο της ποσότητας Q. Η ακρίβεια εδώ εξαρτάτε ευθέως από αριθμό των μετρήσεων που θα κάνουμε.

1.2 Ισορροπία

Ας θεωρήσουμε ξανά την δεσπόζουσα εξίσωση. Αν το σύστημα μας τελικά φτάσει σε μία κατάσταση κατά την οποία οι δύο όροι του αθροίσματος αλληλοαναιρούνται για όλα τα μ, τότε τα dw_{μ}/dt μηδενίζονται και τα στατιστικά βάρη παίρνουν σταθερές τιμές για τον υπόλοιπο χρόνο. Έχουμε έτσι αυτό που ονομάζουμε κατάσταση ισορροπίας. Επειδή η δεσπόζουσα εξίσωση είναι πρώτης τάξης διαφορική εξίσωση με πραγματικούς όρους και ακόμη επειδή τα βάρη w_{μ} περιορίζονται στο διάστημα [0,1], διαπιστώνουμε ότι τα συστήματα που διέπονται από αυτές τις εξισώσεις θα πρέπει τελικά να φτάσουν σε μία κατάσταση.

Οι ρυθμοί μετάβασης $R(\mu \to \nu)$ που είδαμε πιο πάνω παίρνουν συγκεκριμένες τιμές που προκύπτουν από την φύση των αλληλεπιδράσεων μεταξύ του συστήματος και της δεξαμενής θερμότητας και είναι αναγκαίο να τους επιλέξουμε έτσι ώστε να μιμούνται σωστά την αλληλεπίδραση αυτή. Το σημείο κλειδί εδώ είναι ότι ξέρουμε εκ τον προτέρων τις τιμές για τα βάρη w_{μ}

στην κατάσταση ισορροπίας (στα οποία θα αναφερόμαστε ως p_μ):

$$p_\mu = \lim_{t\to\infty} w_\mu(t) \tag{1.2}$$

Μπορεί να αποδειχθεί (Gibbs – 1902)[1] ότι για ένα σύστημα με θερμοκρασία θερμικής δεξαμενής T, το πιο πάνω ισούται με την κατανομή Boltzmann:

$$p_\mu = \frac{1}{Z} e^{-E_\mu/kT} \tag{1.3}$$

όπου E_μ είναι η ενέργεια της κατάστασης μ, k η σταθερά του Boltzmann και Z μία σταθερά ολοκλήρωσης. Συνηθίζεται να συμβολίζουμε την ποσότητα $(kT)^{-1}$ με το γράμμα β. Το Z, που ονομάζεται και συνάρτηση επιμερισμού, ισούται με:

$$Z = \sum_\mu e^{-E_\mu/kT} = \sum_\mu e^{-\beta E_\mu} \tag{1.4}$$

και όπως προκύπτει έχει πολύ σημαντικότερο ρόλο από αυτόν μίας απλής σταθεράς ολοκλήρωσης. Η γνώση του Z μαζί με την θερμοκρασία ή/και οποιαδήποτε άλλη παράμετρο που επηρεάζει το σύστημα μας επιτρέπει να εξάγουμε σχεδόν όλες τις απαραίτητες πληροφορίες σχετικά με την μακροσκοπική συμπεριφορά του συστήματος.

Για την αναμενόμενη τιμή Q σε ένα σύστημα σε ισορροπία, χρησιμοποιώντας τις εξισώσεις (1.1), (1.2) και (1.3), παίρνουμε:

$$\langle Q\rangle = \sum_\mu Q_\mu p_\mu = \frac{1}{Z}\sum_\mu Q_\mu e^{-\beta E_\mu}$$

Για παράδειγμα, η αναμενόμενη τιμή της ενέργειας $\langle E\rangle$, η οποία είναι αυτό που στην θερμοδυναμική ονομάζουμε εσωτερική ενέργεια U, δίνεται από την σχέση:

$$U = \frac{1}{Z}\sum_\mu E_\mu e^{-\beta E_\mu}$$

Χρησιμοποιώντας την σχέση (1.4) μπορούμε να ξαναγράψουμε την πιο πάνω σχέση σαν συνάρτηση της παραγώγου της Z:

$$U = -\frac{1}{Z}\frac{\partial Z}{\partial \beta} = -\frac{\partial \log Z}{\partial \beta}$$

Η ειδική θερμότητα, που είναι η παράγωγος της εσωτερικής ενέργεια, είναι:

$$C = \frac{\partial U}{\partial T} = -k\beta^2\frac{\partial U}{\partial \beta} = k\beta^2\frac{\partial^2 \log Z}{\partial \beta^2}$$

Από τη θερμοδυναμική ξέρουμε ότι η ειδική θερμότητα σχετίζεται και με την εντροπία:

$$C = T\frac{\partial S}{\partial T} = -\beta\frac{\partial S}{\partial \beta}$$

Έτσι παίρνουμε:

$$S = -k\beta\frac{\partial \log Z}{\partial \beta} + k\log Z$$

Και αφού τώρα γνωρίζουμε την εσωτερική ενέργεια και την εντροπία, μπορούμε να υπολογίσουμε την ελεύθερη ενέργεια Helmholtz για το σύστημα μας:

$$F = U - TS = -kT \log Z$$

Έχουμε δηλαδή υπολογίσει τα U, F, C και S απευθείας από την συνάρτηση επιμερισμού Z . Μπορούμε να υπολογίσουμε και άλλα μεγέθη χρησιμοποιώντας τα πιο πάνω σε συνδυασμό με απλές σχέσεις από την θερμοδυναμική.

1.2.1 Διακυμάνσεις

Μέσω της στατιστικής μηχανικής, μπορούμε να πάρουμε πληροφορίες και για ιδιότητες του συστήματος πέραν από αυτές με τις οποίες ασχολείται η κλασική θερμοδυναμική και που αναφέραμε πιο πάνω. Μία τέτοια ιδιότητα που παρουσιάζει ιδιαίτερο ενδιαφέρον είναι οι διακυμάνσεις στης παρατηρήσιμες ποσότητες. Πολλές φορές είναι χρήσιμο να υπολογίσουμε, πέραν τη αναμενόμενης τιμής (θεωρώντας την ως τον χρονικό μέσο όρο), και την τυπική απόκλιση, δίνοντας μας έτσι ένα μέτρο για το πόσο η εξεταζόμενη ποσότητα αλλάζει με την πάροδο του χρόνου. Ας θεωρήσουμε για παράδειγμα την εσωτερική ενέργεια. Το τετράγωνο της τυπικής απόκλισης δίνεται από την σχέση:

$$(\Delta E)^2 = \langle (E - \langle E \rangle)^2 \rangle = \langle E^2 \rangle - \langle E \rangle^2$$

όπου υπολογίζουμε το $\langle E^2 \rangle$ ως εξής:

$$\langle E^2 \rangle = \frac{1}{Z} \sum_{\mu} E_{\mu}^2 e^{-\beta E_{\mu}} = \frac{1}{Z} \frac{\partial^2 Z}{\partial \beta^2}$$

Έχουμε δηλαδή:

$$(\Delta E)^2 = \frac{1}{Z} \frac{\partial^2 Z}{\partial \beta^2} - \left[\frac{1}{Z} \frac{\partial Z}{\partial \beta} \right]^2 = \frac{\partial^2 \log Z}{\partial \beta^2}$$

Χρησιμοποιώντας την σχέση για την ειδική θερμότητα που βρήκαμε πιο πάνω παίρνουμε:

$$(\Delta E)^2 = \frac{C}{k\beta^2}$$

Μπορούμε δηλαδή να υπολογίσουμε τις διακυμάνσεις από μεγέθη της κλασικής θερμοδυναμικής παρόλο εξαρτούνται από λεπτομέρειες στο μικροσκοπικό επίπεδο στης οποίες η θερμοδυναμική δεν έχει πρόσβαση. Αυτή η σχέση μεταξύ της ειδικής θερμότητας και των στατιστικών διακυμάνσεων είναι γνωστή ως το θεώρημα γραμμικής απόκρισης και ισχύει για οποιαδήποτε ποσότητα έχει γραμμική σύζευξη με το σύστημα. Μια τέτοια ποσότητα για ένα μαγνητικό σύστημα που βρίσκεται μέσα σε ένα μαγνητικό πεδίο B είναι και η μαγνήτιση M. Σε αυτή την περίπτωση η χαμιλτονιανή του συστήματος είναι:

$$H = E - BM$$

δεδομένου ότι το πεδίο και η μαγνήτιση είναι συγγραμμικά. Η συνάρτηση επιμερισμού γίνεται:

$$Z = \sum_{\mu} e^{-\beta E_\mu + \beta B M_\mu}$$

όπου E_μ και M_μ η ενέργεια και η μαγνήτιση του συστήματος στην κατάσταση μ. Έτσι, εξ' αιτίας της γραμμικής σύζευξης έχουμε για την αναμενόμενη τιμή της μαγνήτισης:

$$\langle M \rangle = \frac{1}{Z}\sum_{\mu} M_\mu e^{-\beta E_\mu + \beta B M_\mu} = \frac{1}{\beta Z}\frac{\partial Z}{\partial B} = -\frac{\partial F}{\partial B}$$

Και από την σχέση για την τυπική απόκλιση παίρνουμε:

$$(\Delta M)^2 = \langle (M - \langle M \rangle)^2 \rangle = \langle M^2 \rangle - \langle M \rangle^2$$

με:

$$\langle M^2 \rangle = \frac{1}{Z}\sum_{\mu} M_\mu^2 e^{-\beta E_\mu + \beta B M_\mu} = \frac{1}{\beta^2 Z}\frac{\partial^2 Z}{\partial B^2}$$

δηλαδή:

$$(\Delta M)^2 = \frac{1}{\beta^2}\left\{\frac{1}{Z}\frac{\partial^2 Z}{\partial B^2} - \frac{1}{Z^2}\left(\frac{\partial Z}{\partial B}\right)\right\} = \frac{1}{\beta^2}\frac{\partial^2 \ln Z}{\partial B^2} = \frac{1}{\beta}\frac{\partial \langle M \rangle}{\partial B}$$

Η μαγνητική επιδεκτικότητα χ_M ορίζεται ως:

$$\chi_M = \frac{1}{N}\frac{\partial \langle M \rangle}{\partial B} = \frac{\beta}{N}(\Delta M)^2$$

και είναι εμφανές ότι, κατ' ανάλογο τρόπο με την περίπτωση της ειδικής θερμότητας, σχετίζεται άμεσα με τις διακυμάνσεις της μαγνήτισης.

Με ανάλογο τρόπο φτάνουμε και στον γενικό ορισμό της επιδεκτικότητας χ που ορίζεται από την σχέση:

$$\chi \equiv \frac{\partial \langle X \rangle}{\partial Y}$$

όπου Y είναι ένα 'πεδίο' που την τιμή του οποίου κρατούμε σταθερή, και X η συζυγής σε αυτό ποσότητα.

1.2.2 Συνάρτηση συσχετισμού

Επεκτείνοντας τώρα την ιδέα της επιδεκτικότητας, λαμβάνοντας υπόψη και τις μικροσκοπικές ιδιότητες του συστήματος, μπορούμε να θεωρήσουμε τι συμβαίνει όταν αλλάξουμε την τιμή μίας παραμέτρου ή ενός πεδίου σε μία συγκεκριμένη θέση του συστήματος, στη συζυγή της/του παράμετρο σε κάποιο άλλο σημείο. Για λόγους ευκολίας (και επειδή τέτοιου είδος συστήματα θα μας απασχολήσουν παρακάτω) ας θεωρήσουμε ότι το σύστημα μας εκτίνεται πάνω σε ένα πλέγμα με διακριτές θέσεις. Ας υποθέσουμε ότι έχουμε ένα πεδίο που μεταβάλλεται από θέση σε θέση και παίρνει την τιμή Y_i στη θέση i του πλέγματος. Αν συμβολίσουμε με x_i την συζυγή σε αυτό το πεδίο μεταβλητή, η χαμιλτονιανή γίνεται:

$$H = E - \sum_i x_i Y_i$$

Και τώρα με τρόπο ανάλογο με πριν, βρίσκουμε την αναμενόμενη τιμή του x_i:

$$\langle x_i \rangle = \frac{1}{Z} \sum_\mu x_i^\mu e^{-\beta E_\mu} = \frac{1}{\beta} \frac{\partial \log Z}{\partial Y_i}$$

όπου x_i^μ είναι η τιμή του x_i στην κατάσταση μ. Τώρα μπορούμε να ορίσουμε την γενικευμένη επιδεκτικότητα χ_{ij} που μας δίνει ένα μέτρο για την απόκριση του $\langle x_i \rangle$ σε αλλαγές στο πεδίο Y_j σε διαφορετικό κόμβο του πλέγματος:

$$\chi_{ij} = \frac{\partial \langle x_i \rangle}{\partial Y_j} = \frac{1}{\beta} \frac{\partial^2 \log Z}{\partial Y_i \partial Y_j}$$

Με αντικατάσταση του $Z = \sum_\mu e^{-\beta E_\mu}$, και έπειτα από μερικές πράξεις παίρνουμε:

$$\chi_{ij} = \beta\big(\langle x_i x_j \rangle - \langle x_i \rangle \langle x_j \rangle\big) = \beta G_c^{(2)}(i,j)$$

Η ποσότητα $G_c^{(2)}(i,j)$ ονομάζεται **συνάρτηση συσχετισμού** δύο σημείων για το x μεταξύ των κόμβων i και j. Ο εκθέτης (2) είναι για να την ξεχωρίζουμε από συναρτήσεις συσχετισμού ψηλότερων τάξεων. Η συνάρτηση αυτή είναι ένα μέτρο του συσχετισμού μεταξύ των τιμών της μεταβλητής x σε δύο κομβους – παίρνει θετική τιμή όταν οι τιμές του x διακυμαίνονται στην ίδια κατεύθυνση, αρνητική τιμή όταν διακυμαίνονται σε αντίθετες κατευθύνσεις και μηδέν όταν είναι τελείως ασυσχέτιστες. Η γενική συμπεριφορά της συνάρτησης συσχετισμού έχει ως εξής

$$G_c^{(2)}(i,j) \sim e^{-\frac{|x_{ij}|}{\xi}}$$

όπου το $|x_{ij}|$ είναι η απόσταση των σημείων i και j. Ο ρυθμιστικός παράγοντας ξ που προκύπτει ονομάζεται μήκος συσχετισμού και όπως είναι φυσικό έχει μονάδες μήκους. Είναι ένα μέγεθος που εξαρτάται από της δυναμικές ιδιότητες του συστήματος και μας δίνει ένα μέτρο της χωρικής έκτασης του συσχετισμού των τιμών της μαγνήτισης σε δύο πλεγματικές θέσεις. Η τιμή γενικώς που παίρνει το μήκος συσχετισμού είναι της τάξης μεγέθους της πλεγματικής σταθεράς α (η απόσταση μεταξύ δύο γειτνιαζόντων κόμβων του πλέγματος).

1.2.3 Πρότυπο Potts

Για να δούμε πως εφαρμόζονται στην πράξη τα παραπάνω θα τα εφαρμόσω πάνω στο πρότυπο που θα μας απασχολήσει παρακάτω – το πρότυπο Potts. Στα πλαίσια του προτύπου αυτού θεωρούμε ένα πλέγμα που σε κάθε κόμβο έχουμε ένα ατομικό σπιν. Η συνδυασμένη μαγνητική διπολική ροπή από όλους τους κόμβους δίνει τις μακροσκοπικές μαγνητικές ιδιότητες του υλικού. Κάθε ένα

από αυτά τα σπιν παίρνει τιμές από ένα σύνολο q διαφορετικών καταστάσεων σπιν τις οποίες θα αντιστοιχίσω σε q φυσικούς αριθμούς $\{1,2,\dots,q\}$. Θεωρούμε επίσης ότι κάθε σπιν αλληλεπιδρά μόνο με τα σπιν που βρίσκονται στην άμεση γειτονιά του με ένταση J. Σε περίπτωση που έχουμε και ένα εξωτερικό πεδίο B, πρεπει να λάβουμε υπόψη και την σύζευξη των σπιν με το πεδίο αυτό. Έτσι η χαμιλτονιανή του συστήματος παίρνει την μορφή:

$$H = -(J + B)\sum_{\langle ij\rangle} \delta_{s_i s_j}$$

όπου η σήμανση $\langle ij\rangle$ υποδηλώνει ότι οι κόμβοι i και j είναι πλησιέστεροι γήτονες (nearest neighbors). Οι καταστάσεις σε αυτό το πρότυπο ορίζονται ως τα διάφορα σύνολα τιμών που μπορεί να πάρει το σπιν για τις θέσεις του πλέγματος. Επειδή η τιμή του σπιν σε κάθε πλεγματική θέση μπορεί να πάρει q τιμές, σε ένα πλέγμα με N κόμβους έχουμε q^N πιθανές καταστάσεις για το σύστημα. Έτσι τώρα μπορούμε να γράψουμε την συνάρτηση επιμερισμού:

$$Z = \sum_{\{s_i\}} e^{-\beta H}$$

όπου στη θέση του H βάζουμε την χαμιλτονιανή που βρήκαμε πιο πάνω. Η άθροιση γίνεται πάνω στις q δυνατές τιμές του σπιν για κάθε πλεγματική θέση.

Χρησιμοποιώντας τώρα την συνάρτηση επιμερισμού, έχουμε τώρα την δυνατότητα να υπολογίσουμε τα διάφορα μακροσκοπικά μεγέθη που σχετίζονται με το σύστημα μας:

$$\langle M\rangle = \frac{1}{\beta Z}\frac{\partial Z}{\partial B}$$

$$(\Delta M)^2 = \frac{1}{\beta^2}\frac{\partial^2 \ln Z}{\partial B^2}$$

$$\chi_M = \frac{\beta}{N}(\Delta M)^2$$

κ.ο.κ.

Αξίζει να αναφέρουμε μία ιδιαίτερη ιδιότητα που παρουσιάζει το δυσδιάστατο πρότυπο Potts για $B = 0$, όταν το β πάρει την τιμή:

$$\beta_c = \ln\left(1 + \sqrt{q}\right)$$

που είναι γνωστή ως κρίσιμη θερμοκρασία. Σε αυτή την περίπτωση το σύστημα παρουσιάζει μετάβαση φάσης από την διατεταγμένη φάση (σιδηρομαγνητική) όπου το σύστημα είναι μαγνητισμένο, στην άτακτη φάση (παραμαγνητική) όπου η μαγνήτιση μηδενίζεται. Οι μόνοι παράγοντες που επηρεάζουν το σύστημα στο πρότυπο που ορίσαμε είναι η θερμοκρασία και το μαγνητικό πεδίο. Έτσι αυτοί είναι οι καθοριστικοί παράγοντες που ορίζουν την μεταβολή του συστήματος από τη μία φάση στην άλλη. Το σε ποια φάση βρίσκεται το σύστημα υποδηλώνεται από την τιμή της παραμέτρου τάξης, που

στην προκειμένη περίπτωση είναι η μαγνήτιση ανά πλεγματική θέση m. Στην παραμαγνητική φάση το m παίρνει την τιμή 0, ενώ στην σιδηρομαγνητική φάση την τιμή 1.

Όταν έχουμε μετάβαση φάσης το μήκος συσχετισμού παύει να είναι πεπερασμένο και τείνει προς το άπειρο (πρακτικά $\xi \to L$). Συγκεκριμένα συμπεριφέρεται ασυμπτωτικά σαν

$$\xi(\beta) \equiv \xi(t) \sim |\xi|^{-\nu}$$

όπου $t = (\beta_c - \beta)/\beta_c$. Σε αυτή την περίπτωση η συνάρτηση συσχετισμού συμπεριφέρεται σύμφωνα με την εξίσωση

$$G_c^{(2)}(i,j) \sim \frac{1}{|x_{ij}|^{\eta}}$$

Ο εκθέτης η που εμφανίζεται στην παραπάνω σχέση είναι ένας από τους λεγόμενους κρίσιμους εκθέτες που έχουν το χαρακτηριστικό να μην εξαρτώνται από τις λεπτομέρειες του πλέγματος και της αλληλεπίδρασης.

1.3 Αριθμητικές Μέθοδοι

Παρόλο που η θεμελίωση της στατιστικής μηχανικής είναι από πολλές απόψεις ιδιαιτέρα κομψή, ο πραγματικός υπολογισμός των ιδιοτήτων ενός μοντέλου παρουσιάζει αρκετά προβλήματα. Αν προσπαθήσουμε να εξάγουμε τα αποτελέσματα μας υπολογίζοντας την συνάρτηση επιμερισμού Z, θα παρατηρήσουμε ότι προκύπτουν αθροίσματα πάνω σε πολύ μεγάλο αριθμό καταστάσεων. Ο ακριβής υπολογισμός αυτών των αθροισμάτων έχει επιτευχθεί μόνο για απλά πρότυπα με διακριτές ενεργειακές στάθμες, με πιο γνωστό το πρότυπο Ising σε δύο διαστάσεις. Κάτι τέτοιο όμως δεν έγινε κατορθωτό για πρότυπο Potts, ούτε και για τα περισσότερα πρότυπα που απασχολούν την επιστημονική κοινότητα σήμερα. Αυτό έδωσε ώθηση στην ανάπτυξη προσεγγιστικών υπολογιστικών μεθόδων που μας δίνουν τα ζητούμενα αποτελέσματα.

Η πιο ευθύς αριθμητική μέθοδος για την επίλυση ενός προβλήματος της στατιστικής μηχανικής είναι να θεωρήσουμε το πρότυπο που μας ενδιαφέρει πάνω σε ένα πλέγμα πεπερασμένου μεγέθους. Έτσι η συνάρτηση επιμερισμού γίνεται ένα άθροισμα πεπερασμένων όρων το οποίο μπορούμε να υπολογίσουμε αριθμητικά με την χρήση Η/Υ.

Ας δούμε τώρα τι γίνεται αν θεωρήσουμε το πρότυπο Potts με 8 διαφορετικές καταστάσεις για το σπιν σε ένα πλέγμα 5×5 θέσεων. Συνηθίζεται να υιοθετούμε περιοδικές συνοριακές συνθήκες έτσι ώστε να έχουμε αλληλεπιδράσεις μεταξύ των σπιν που βρίσκονται στην μία πλευρά του πλέγματος με τα σπιν στην απέναντι πλευρά. Ο αριθμός των καταστάσεων στις οποίες μπορεί να βρεθεί το σύστημα είναι

$8^{25} = 37\,778\,931\,862\,957\,161\,709\,568$. Ο τρόπος με τον οποίο εξάγουμε τις ιδιότητες του συστήματος (ενέργεια – μαγνήτιση) μας επιτρέπει να εκμεταλλευτούμε τη συμμετρία του συστήματος ως προς την μετάθεση των καταστάσεων του σπιν. Με 8 διαφορετικές καταστάσεις για το σπιν έχουμε $8! = 40320$, που μας δίνει 936 977 476 759 850 241 διαφορετικές καταστάσεις. Όπως είναι φανερό, αυτός είναι ένας πολύ μεγάλος αριθμός που αυξάνεται εκθετικά με την αύξηση του μεγέθους του πλέγματος. Το γεγονός αυτό καθιστά τη διαδικασία του υπολογισμού της συνάρτησης επιμερισμού ιδιαίτερα δύσκολη. Για να ξεπεράσουμε την δυσκολία αυτή χρησιμοποιούμε την τεχνική που είναι γνωστή ως προσομοίωση **Monte Carlo**.

1.3.1 Προσομοίωση Monte Carlo

Για να υπολογίσουμε αριθμητικά την συνάρτηση επιμερισμού για μεγάλα πλέγματα υπάρχει ουσιαστικά μόνο ένας τρόπος – η προσομοίωση Monte Carlo. Αυτό που κάνει αυτή η μέθοδος είναι να εξομοιώνει τις τυχαίες θερμικές διακυμάνσεις ενός συστήματος, πηγαίνοντας έτσι από μια κατάσταση σε μια άλλη. Δηλαδή δημιουργούμε ένα πρότυπο σύστημα στον υπολογιστή μας, και το βάζουμε να περάσει από μία πληθώρα καταστάσεων με τέτοιο τρόπο που η πιθανότητα του να βρίσκεται σε μία κατάσταση μ μετά από χρόνο t να είναι ίση με το στατιστικό βάρος $w_\mu(t)$ που αυτή η κατάσταση θα είχε σε ένα πραγματικό σύστημα. Για να το πετύχουμε αυτό, πρέπει να διαλέξουμε τα κατάλληλα δυναμικά χαρακτηριστικά για την προσομοίωση μας – ένα κανόνα δηλαδή που να διέπει τις αλλαγές από μία κατάσταση του συστήματος σε μία άλλη – έτσι ώστε η κάθε κατάσταση να εμφανίζεται πιθανότητα που ικανοποιεί την δεσπόζουσα εξίσωση. Διαλέγουμε δηλαδή ένα σύνολο ρυθμών μετάβασης $R(\mu \to \nu)$ με τέτοιο τρόπο που η λύση για την δεσπόζουσα εξίσωση σε συνθήκες θερμικής ισορροπίας να είναι ακριβώς η κατανομή Boltzmann. Στη συνέχεια χρησιμοποιούμε αυτούς τους ρυθμούς για να διαλέξουμε από ποιες καταστάσεις θα περάσει το σύστημα που εξομοιώνουμε, και από αυτές της καταστάσεις κάνουμε τις εκτιμήσεις για τις παρατηρήσιμες ποσότητες που μας ενδιαφέρουν.

Τα παραπάνω έχουν σαν αποτέλεσμα την δραματική μείωση των καταστάσεων που πρέπει να λάβουμε υπόψη μας στον υπολογισμό της συνάρτηση επιμερισμού. Αυτό λειτουργεί όμως σαν δίκοπο μαχαίρι γιατί, παρόλο που τώρα η συνάρτηση επιμερισμού είναι πλέων υπολογίσιμη, υπεισέρχονται στατιστικά σφάλματα στον υπολογισμό της. Τα στατιστικά αυτά σφάλματα δημιουργούν δυσκολίες στον υπολογισμό των παρατηρήσιμων μεγεθών που εξάγουμε με χρησιμοποιώντας τις σχέσεις που αναφέραμε πιο πάνω, καθώς η παραγώγιση παρουσία στατιστικού θορύβου δεν συνιστάται.

Προς αντιμετώπιση αυτού του εμποδίου, κάνουμε απευθείας υπολογισμούς των μεγεθών από το πλέγμα.

2

Αρχές Προσομοίωση Monte Carlo

Σε αυτή την ενότητα θα δούμε τις βασικές αρχές που διέπουν την προσομοίωση με την μέθοδο Monte Carlo όπως η δειγματοληψία με κριτήριο σημαντικότητας (importance sampling), η συνθήκη λεπτομερούς ισοζύγισης (detailed balance condition) και οι λόγοι αποδοχής (acceptance ratios).

2.1 Δειγματοληψία

Ο στόχος μίας προσομοίωσης Monte Carlo ενός θερμικού συστήματος είναι συνήθως ο υπολογισμός της αναμενόμενης τιμής $\langle Q \rangle$ για κάποιο παρατηρήσιμο Q όπως η εσωτερική ενέργεια ή η μαγνήτιση. Η διαδικασία που ακολουθούμε προς αυτό τον στόχο είναι να εφαρμόσουμε την γνωστή σχέση:

$$\langle Q \rangle = \frac{\sum_\mu Q_\mu e_\mu^{-\beta E_\mu}}{\sum_\mu e_\mu^{-\beta E_\mu}}$$

αθροίζοντας όμως πάνω σε ένα πεπερασμένο σύνολο καταστάσεων $\{\mu_1, \mu_2, \dots, \mu_M\}$ οι οποίες επιλέγονται σύμφωνα με την κατανομή πιθανότητας p_μ. Έτσι ορίζουμε τον **εκτιμητή** (estimator) Q_M του $\langle Q \rangle$:

$$Q_M = \frac{\sum_{i=1}^{M} Q_{\mu_i} p_{\mu_i}^{-1} e^{-\beta E_{\mu_i}}}{\sum_{j=1}^{M} p_{\mu_j}^{-1} e^{-\beta E_{\mu_j}}}$$

Αυξάνοντας τον αριθμό M των καταστάσεων του δείγματος μας παίρνουμε όλο και καλύτερη εκτίμηση για το $\langle Q \rangle$, καταλήγοντας στο $Q_M = \langle Q \rangle$ όταν $M \to \infty$.

Πώς όμως διαλέγουμε ποιες καταστάσεις θα συμπεριλάβουμε στο δείγμα μας; Με άλλα λόγια, ποια είναι η κατανομή πιθανότητας p_μ που πρέπει να επιλέξουμε; Αν πάρουμε το $p_\mu = \sigma\tau\alpha\theta$ – αν διαλέγουμε δηλαδή την κάθε κατάσταση με ίση πιθανότητα – καταλήγουμε στην σχέση

$$Q_M = \frac{\sum_{i=1}^{M} Q_{\mu_i} e^{-\beta E_{\mu_i}}}{\sum_{j=1}^{M} e^{-\beta E_{\mu_j}}}$$

για τον εκτιμητή μας. Αυτή όμως η επιλογή είναι προβληματική. Για να το δούμε αυτό ας θεωρήσουμε ένα μικρό τρισδιάστατο σύστημα $10 \times 10 \times 10$ στο

πρότυπο Potts με $q = 8$ διαφορετικές πιθανές καταστάσεις σπιν για πλεγματική θέση. Αυτό μας δίνει $8^{1000} \approx 10^{900}$ διαφορετικές καταστάσεις. Στις εξομοιώσεις που θα κάνω παρακάτω, το σύστημα περνάει από $\sim 10^8$ διαφορετικές καταστάσεις, πράγμα που σημαίνει ότι επιλέγεται μία κατάσταση ανά 10^{892} πιθανές καταστάσεις. Και επειδή η συμπεριφορά των αθροισμάτων στον εκτιμητή συνήθως κυριαρχείται από ένα μικρό αριθμό καταστάσεων –ιδικά στις περιπτώσεις που έχουμε χαμηλή θερμοκρασία – είναι ιδιαίτερα απίθανο να επιλέξουμε τις καταστάσεις που έχουν σημαντική συνεισφορά με αυτό τον τρόπο.

Αν όμως επιλέγαμε το δείγμα των καταστάσεων με κριτήριο το βάρος τους κατά Boltzmann

$$p_\mu = \frac{1}{Z} e^{-\beta E_\mu}$$

ο εκτιμητής γίνεται:

$$Q_M = \frac{\sum_{i=1}^{M} Q_{\mu_i} \left(e^{-\beta E_{\mu_i}}\right)^{-1} e^{-\beta E_{\mu_i}}}{\sum_{j=1}^{M} \left(e^{-\beta E_{\mu_j}}\right)^{-1} e^{-\beta E_{\mu_j}}} = \frac{1}{M} \sum_{i=1}^{M} Q_{\mu_i}$$

που δίνει σίγουρα μία καλύτερη εκτίμηση για το ζητούμενο μέγεθος. Ο παραπάνω τρόπος δειγματοληψίας ονομάζεται **δειγματοληψία με κριτήριο σημαντικότητας** (importance sampling). Το μόνο που μένει τώρα είναι να ορίσουμε τον τρόπο με τον οποίο θα διαλέγουμε τις καταστάσεις μας ώστε η κάθε μία να εμφανίζεται με την σωστή κατά Boltzmann πιθανότητα. Για να το κάνουμε αυτό πρέπει πρώτα να ορίσουμε την διαδικασία Markov.

2.1.1 Διαδικασία Markov

Η **διαδικασία Markov** είναι ένας μηχανισμός που δίνοντας του για είσοδο μία κατάσταση μ, συνθέτει μία νέα κατάσταση ν. Και αυτό γίνεται με στοχαστικό τρόπο – δηλαδή εισάγοντας την κατάσταση μ δεν παίρνουμε πάντα την ίδια κατάσταση ν σαν έξοδο. Έτσι ορίζεται η **πιθανότητα μετάβασης** (transition probability) $P(\mu \to \nu)$ ως η πιθανότητα να πάρουμε την κατάσταση ν εισάγοντας την κατάσταση μ στην διαδικασία Markov. Οι πιθανότητες μετάβασης πρέπει να υπακούνε σε δύο σημαντικούς κανόνες: (1) πρέπει να είναι ανεξάρτητες του χρόνου, και (2) πρέπει να εξαρτώνται μόνο από τις τρέχοντες καταστάσεις μ και ν, και όχι από οποιαδήποτε άλλη προηγούμενη. Σαν πιθανότητες εννοείται ότι πρέπει να υπακούνε και στην σχέση

$$\sum_{\nu} P(\mu \to \nu) = 1$$

καθώς η διαδικασία Markov θα δώσει αναγκαστικά κάποια κατάσταση. Αξίζει να σημειώσουμε ότι η πιθανότητα μετάβασης $P(\mu \to \mu)$ δεν είναι αναγκαστικά ίση με μηδέν.

Στις εξομοιώσεις Monte Carlo κάνουμε επαναλαμβανόμενη χρήση τις διαδικασίας Markov, δημιουργώντας έτσι τις λεγόμενες αλυσίδες Markov. Ξεκινώντας δηλαδή από μία κατάσταση μ, διμιουργούμε μία νέα κατάσταση ν την οποία εισάγουμε με την σειρά της στην διαδικασία Markov, δημιουργώντας έτσι την νέα κατάσταση λ κ.ο.κ. Βασικό χαρακτηριστικό της διαδικασίας Markov είναι το ότι ξεκινώντας από οποιαδήποτε κατάσταση μ και μετά από κάποιο πεπερασμένο αριθμό διαδοχικών παραγωγών θα φτάσει στο σημείο που οι καταστάσεις που θα συνθέτει θα εμφανίζονται με τις πιθανότητες που ορίζει η κατανομή Boltzmann. Όταν αυτό συμβεί λέμε ότι το σύστημα έφτασε την **κατάσταση θερμικής ισορροπίας**. Απαραίτητη προϋπόθεση για να συμβεί αυτό όμως είναι να τηρούνται τα **κριτήριο εργοδικότητας** (ergodicity) και η **συνθήκη λεπτομερούς ισοζύγισης** (detailed balance).

2.1.2 Εργοδικότητα

Το κριτήριο της εργοδικότητας απαιτεί από την διαδικασία Markov να έχει τη δυνατότητα, ξεκινώντας από οποιαδήποτε κατάσταση στου συστήματος, να μπορεί να φτάσει σε οποιαδήποτε άλλη, σε πεπερασμένο αριθμό βημάτων. Κάθε κατάσταση ν εμφανίζεται με μη-μηδενική πιθανότητα p_ν την κατανομή Boltzmann. Γι αυτό το λόγο πρέπει να υπάρχει μία αλληλουχία καταστάσεων που να οδηγεί σε αυτή από όποια κατάσταση και να ξεκινήσουμε. Στις περισσότερες εφαρμογές η πλειοψηφία των πιθανοτήτων μετάβασης θέτονται ίσες με μηδέν και χρειάζεται ιδιαίτερη προσοχή ώστε να μην παραβιαστεί το κριτήριο της εργοδικότητας.

2.1.3 Συνθήκη Λεπτομερούς Ισοζύγισης

Η άλλη συνθήκη που επιβάλαμε στην διαδικασία Markov είναι η συνθήκη λεπτομερούς ισοζύγισης. Αυτή η συνθήκη μας εγγυάται ότι η κατανομή πιθανότητας που παίρνουμε όταν το σύστημα φτάσει στην κατάσταση ισορροπίας είναι ακριβώς η κατανομή Boltzmann και όχι κάποια άλλη κατανομή.

Έστω έχουμε ένα σύστημα που βρίσκεται σε κατάσταση ισορροπίας. Ουσιαστικά αυτό σημαίνει ότι οι ρυθμοί με τους οποίους το σύστημα κάνει μεταβάσεις από και προς κάθε κατάσταση μ είναι ίσοι μεταξύ τους. Αυτό μαθηματικά εκφράζεται ως:

$$\sum_{\nu} p_\mu P(\mu \to \nu) = \sum_{\nu} p_\nu P(\nu \to \mu)$$

που κάνοντας χρήση του $\sum_\nu P(\mu \to \nu) = 1$, μπορεί να ξαναγραφτεί ως:

$$p_\mu = \sum_{\nu} p_\nu P(\nu \to \mu)$$

Η συνθήκη αυτή είναι αναγκαία αλλά δεν μας εγγυάται το ζητούμενο. Από την άλλη η συνθήκη

$$p_\mu P(\mu \to \nu) = p_\nu P(\nu \to \mu)$$

παρόλο που δεν είναι αναγκαία, μας εξασφαλίζει ότι το σύστημα θα φτάσει τελικώς σε κατάσταση θερμικής ισορροπίας και τότε οι καταστάσεις θα επιλέγονται βάσει της κατά Boltzmann πιθανότητας τους.

Αν ξαναγράψουμε τη σχέση αυτή αντικαθιστώντας τις πιθανότητες Boltzmann

$$\frac{P(\mu \to \nu)}{P(\nu \to \mu)} = \frac{p_\nu}{p_\mu} = e^{-\beta(E_\nu - E_\mu)}$$

παίρνουμε την σχέση που πρέπει να ικανοποιούν οι πιθανότητες μετάβασης.

Έχοντας τώρα υπόψη τα πιο πάνω κριτήρια και συνθήκες, προχωράμε στον σχεδιασμό μίας εφαρμογής σε Η/Υ εκτελεί την διαδικασία Markov με τις κατάλληλες πιθανότητες μετάβασης, δημιουργώντας έτσι μία αλληλουχία καταστάσεων για το σύστημα. Αφού περιμένουμε αρκετό χρόνο ώστε η κατανομή πιθανότητας των καταστάσεων να πλησιάσει αρκετά την κατανομή Boltzmann, χρησιμοποιούμε τον εκτιμητή μας Q_M για τις επόμενες M καταστάσεις και υπολογίζουμε τα διάφορα μετρήσιμα μεγέθη που μας αφορούν.

2.2 Λόγος Αποδοχής

Η τελευταία σχέση μας δίνει αρκετή ελευθερία στην επιλογή της γενικής μορφής που θα έχουν οι πιθανότητες μετάβασης, ορίζοντας έτσι μία πληθώρα διαδικασιών Markov. Αλλά παρόλο που είναι εύκολο να ορίσουμε διάφορες υποψήφιες διαδικασίες Markov για το πρόβλημα που αντιμετωπίζουμε, η εύρεση αυτής που μας δίνει ακριβώς το κατάλληλο σύνολο πιθανοτήτων μετάβασης μπορεί να εμπεριέχει δυσκολίες. Αυτό όμως δεν αποτελεί πρόβλημα καθώς μπορούμε να διαλέξουμε όποιο αλγόριθμο παραγωγής νέων καταστάσεων θέλουμε κρατώντας παράλληλα το σύνολο πιθανοτήτων μετάβασης της επιλογής μας εισάγοντας αυτό που ονομάζουμε **λόγο αποδοχής** (acceptance ratio)

Όπως αναφέραμε και πιο πριν, μπορούμε να θέσουμε την πιθανότητα μετάβασης $P(\mu \to \mu)$ διάφορη του μηδενός. Και επειδή, όποια τιμή και να δώσουμε στην $P(\mu \to \mu)$, η συνθήκη λεπτομερούς ισοζύγισης πάντα ικανοποιείται για $\nu = \mu$, έχουμε κάποια ευελιξία στην επιλογή των υπολοίπων πιθανοτήτων μετάβασης για $\nu \neq \mu$. Πιο συγκεκριμένα μπορούμε να ρυθμίσουμε την τιμή των $P(\mu \to \nu)$ κάνοντας μία ίση αλλά αντιστροφή τροποποίηση στη $P(\mu \to \mu)$, χωρίς να σταματήσει να ισχύει η $\sum_\nu P(\mu \to \nu) = 1$. Πρέπει όμως να προσέξουμε η $P(\mu \to \mu)$ να μην πάρει τιμή έξω από το

επιτρεπτό διάστημα (0,1). Αν τροποποιήσουμε κατάλληλα και την τιμή της $P(\nu \to \mu)$ διατηρούμε και την ισχύ της σχέσης.

Ας δούμε τώρα πως κάνουμε αυτές τις ρυθμίσεις. Αρχίζουμε χωρίζοντας τις πιθανότητες μετάβασης σε δύο μέρη:

$$P(\mu \to \nu) = g(\mu \to \nu)A(\mu \to \nu)$$

Η ποσότητα $g(\mu \to \nu)$ ονομάζεται **πιθανότητα επιλογής** (selection probability) και είναι η πιθανότητα ο αλγόριθμος μας να μας δώσει την κατάσταση ν έχοντας την κατάσταση μ σαν είσοδο. Η ποσότητα $A(\mu \to \nu)$ από την άλλη, ονομάζεται λόγος αποδοχής και είναι το ποσοστό των μεταβάσεων $\mu \to \nu$ που θα αποδεχόμαστε, δεδομένου φυσικά ότι προέκυψαν από τον αλγόριθμο που χρησιμοποιούμε. Τις υπόλοιπες φορές το σύστημα θα παραμένει στην κατάσταση μ. Η τιμή του λόγου αποδοχής μπορεί να πάρει όποια τιμή θέλουμε από το διάστημα [0,1].

Ο περιορισμός που έχουμε εξαιτίας την συνθήκης λεπτομερούς ισοζύγισης εκφράζεται υπό την μορφή λόγου. Έτσι έχουμε:

$$\frac{P(\mu \to \nu)}{P(\nu \to \mu)} = \frac{g(\mu \to \nu)A(\mu \to \nu)}{g(\nu \to \mu)A(\nu \to \mu)}$$

Ο λόγος $A(\mu \to \nu)/A(\nu \to \mu)$ μπορεί να πάρει όποια τιμή διαλέξουμε από το διάστημα $(0, \infty)$, δινοντας έτσι την δυνατότητα στα $g(\mu \to \nu)$ και $g(\nu \to \mu)$ να πάρουν όποια τιμή εμείς θέλουμε.

Αυτό που θα πρέπει να επιδιώκουμε είναι να βρούμε ένα αλγόριθμο με πιθανότητες επιλογής τέτοιες ώστε να δίνουν τους μέγιστους δυνατούς λόγου αποδοχής για καταστάσεις ν οσο το δυνατό πιο ασυσχέτιστες με την αρχική κατάσταση μ.

3

Αλγόριθμοι για το Πρότυπο Potts

Σε αυτή την ενότητα θα δούμε μερικούς από τους αλγόριθμους που υλοποιούν την διαδικασία Markov για το πρότυπο Potts. Όπως θα δούμε ο καθένας έχει τις δικές του δυνάμεις και αδυναμίες.

3.1 Αλγόριθμος Metropolis

Ο πρώτος αλγόριθμος που θα δούμε είναι ίσως ο πιο ευρέος χρησιμοποιημένος και δημοφιλής αλγόριθμος από όλους. Πήρε το όνομα του από τον δημιουργό του, Νικόλα Μητρόπουλο. Έχει, όπως θα περιμέναμε ένα τρόπο να επιλέγει μία νέα κατάσταση ν δοθέντος μίας αρχικής μ, και ένα λόγο αποδοχής για κάθε μετάβασης $A(\mu \to \nu)$ έτσι ώστε να ικανοποιείται η συνθήκη λεπτομερούς ισοζύγισης.

Έχοντας κατά νου ότι ο μηχανισμός που θα δημιουργεί τις νέες καταστάσεις πρέπει να υπακούει στο κριτήριο εργοδικότητας, θα πρέπει οι πιθανότητες επιλογής $g(\mu \to \nu)$ να επιτρέπουν στο σύστημα να φτάσει από οποιαδήποτε αρχική κατάσταση σε οποιαδήποτε άλλη τελική κατάσταση. Ξέρουμε όμως ότι για ένα σύστημα που βρίσκεται σε κατάσταση θερμικής ισορροπίας η διακύμανση της ενέργειας του είναι πολύ μικρότερη από την ολική ενέργεια του συστήματος. Αυτό στην ουσία σημαίνει ότι τον περισσότερο καιρό το σύστημα παραμένει περιορισμένο σε ένα υποσύνολο του καταστατικού χώρου που τα μέλη του δεν διαφέρουν πολύ ενεργειακά. Εκμεταλλευόμενοι αυτό το γεγονός ορίζουμε της πιθανότητες επιλογής $g(\mu \to \nu)$ να είναι όλες ίσες μεταξύ τους για τις περιπτώσεις που η αρχική κατάσταση διαφέρει από την τελική κατά ένα μόνο σπιν, και μηδέν για όλες τις άλλες περιπτώσεις. Και επειδή αλλάζοντας διαδοχικά ένα-ένα σπιν μπορούμε να φτιάξουμε όποια κατάσταση θέλουμε σε πεπερασμένο αριθμό βημάτων, το κριτήριο της εργοδικότητας πληρείται. Αυτός ο αλγόριθμος λέμε ότι ανήκει στην ομάδα των single-flip αλγορίθμων. Στη συνέχεια προχωράμε στην επιλογή του λόγου αποδοχής έτσι ώστε να ικανοποιείτε η συνθήκη λεπτομερούς ισοζύγισης. Για να το κάνουμε αυτό ας θεωρήσουμε πρώτα ένα σύστημα με N σπιν τα οποία μπορούν να πάρουν q διαφορετικές τιμές.

Έχουμε δηλαδή $N \times (q-1)$ διαφορετικές καταστάσεις που είναι προσβάσιμες με ένα βήμα του single-flip αλγορίθμου μας. Και επειδή όπως είπαμε οι πιθανότητες επιλογής $g(\mu \to \nu)$ για αυτές τις καταστάσεις είναι όλες ίσες μεταξύ τους, έχουμε:

$$g(\mu \to \nu) = \frac{1}{N} \times \frac{1}{q-1}$$

Αντικαθιστώντας στην σχέση για την συνθήκη λεπτομερούς ισοζύγισης παίρνουμε:

$$\frac{P(\mu \to \nu)}{P(\nu \to \mu)} = \frac{g(\mu \to \nu)A(\mu \to \nu)}{g(\nu \to \mu)A(\nu \to \mu)} = \frac{A(\mu \to \nu)}{A(\nu \to \mu)} = e^{-\beta(E_\nu - E_\mu)}$$

Ο αλγόριθμος Metropolis ικανοποιεί αυτή την απαίτηση θέτοντας το λόγο αποδοχής ίσο με:

$$A(\mu \to \nu) = \begin{cases} e^{-\beta(E_\nu - E_\mu)} & \text{αν } E_\nu - E_\mu > 0 \\ 1 & \text{αλλιώς} \end{cases}$$

Με άλλα λόγια αν επιλέξουμε μία νέα κατάσταση με χαμηλότερη ή ίση ενέργεια την αποδεχόμαστε, ενώ αν είναι κατάσταση ψηλότερης ενέργειας την αποδεχόμαστε με την πιθανότητα που ορίσαμε πιο πάνω.

Σε μία εφαρμογή που θα υλοποιεί τον αλγόριθμο Metropolis θα πρέπει πρώτα να ορίζουμε την τοπογραφία του συστήματος. Καθορίζουμε δηλαδή το μέγεθος N του πλέγματος, τον αριθμό z των πλησιέστερων γειτόνων που έχει η κάθε πλεγματική θέση και το είδος των συνοριακών συνθηκών που υιοθετούμε. Πρέπει επίσης να ορίσουμε τον αριθμό q των διαφορετικών καταστάσεων του σπιν, την θερμοκρασία β της δεξαμενή θερμότητας και την αρχική κατάσταση του συστήματος μας. Κάθε βήμα του αλγορίθμου έχει ως εξής:

1. Επιλέγουμε τυχαία ένα σπιν από το πλέγμα
2. Επιλέγουμε τυχαία μία νέα τιμή για το σπιν της θέσης αυτής
3. Αποδεχόμαστε την νέα κατάσταση με πιθανότητα $e^{-\beta(E_\nu - E_\mu)}$ αν είναι κατάσταση υψηλότερης ενέργειας από την αρχική, και με πιθανότητα 1 αν είναι χαμηλότερης

Επειδή ο Metropolis είναι single-flip αλγόριθμος, η διαφορά ενέργειας μεταξύ της αρχικής και τελικής κατάστασης εξαρτάτε μόνο από το σπιν που έχουμε επιλέξει και τους πλησιέστερους γείτονες του. Γι αυτό το λόγο είναι αρκετό να υπολογίσουμε μόνο την συνεισφορά που έχει το εμπλεκόμενο στη δημιουργία της νέας κατάσταση σπιν.

3.2 Αλγόριθμος Heat-bath

Για σχετικά μικρό αριθμό πιθανών τιμών σπιν q, ο **αλγόριθμος Metropolis** λειτουργεί πολύ καλά. Ειδικά στην περίπτωση που έχουμε $q = 2$ (≡ Ising)

έχει αποδειχτεί ότι είναι ο πιο αποδοτικός από όλους. Σε συστήματα όμως με μεγάλο q παύει πλέων να είναι ο καταλληλότερος. Για να το δούμε αυτό ας θεωρήσουμε την ακραία περίπτωση που έχουμε $q = 100$. Και ειδικότερα όταν η θερμοκρασία της δεξαμενής θερμότητας είναι χαμηλή, οπότε θα υπάρχει και μία τάση ευθυγράμμισης γειτνιαζόντων σπιν, σχηματίζοντας έτσι σιδηρομαγνητικές ομάδες. Επιπλέον ας θεωρήσουμε την περίπτωση που οι πλησιέστεροι γείτονες έχουν όλοι διαφορετικές τιμές μεταξύ τους και από το σπιν που εξετάζουμε. Παρατηρούμε ότι οι τέσσερις καταστάσεις σπιν των γειτόνων έχουν μεγαλύτερο βάρος Boltzmann από τις υπόλοιπες 96 πιθανές καταστάσεις σπιν του συστήματος, καθώς θα δίνουν καταστάσεις με χαμηλότερη ενέργεια. Έτσι όταν κάνουμε την επιλογή της νέας τιμής του σπιν με τυχαίο τρόπο έχουμε πιθανότητα μόνο $1/25$ να επιλέξουμε μία που ωθεί το σύστημα προς χαμηλότερα επίπεδα ενέργειας. Αυτό σημαίνει ότι το σύστημα θα παραμένει για αρκετό χρόνο σε καταστάσεις που δεν διαφέρουν ενεργειακά, κάνοντας έτσι την πορεία προς την ισορροπία πιο αργή. Το πρόβλημα αυτό έρχεται να λύσει ο **αλγόριθμος Heat-bath**.

Ο αλγόριθμος Heat-bath ανήκει και αυτός στην οικογένεια των single-flip αλγορίθμων αλλά έχει την ιδιότητα να είναι πιο αποδοτικός στις περιπτώσεις που έχουμε μεγάλο q. Όπως και στην περίπτωση του αλγόριθμου Metropolis, ξεκινάμε επιλέγοντας με τυχαίο τρόπο ένα σπιν k από το πλέγμα. Ανεξάρτητα από την τιμή που έχει, επιλέγουμε την νέα τιμή s_k για το σπιν στηριζόμενοι στο βάρος Boltzmann που έχει η κάθε μία από τις πιθανές τιμές. Έχουμε δηλαδή

$$p_\nu = \frac{e^{-\beta E_\nu}}{\sum_{\mu=1}^{q} e^{-\beta E_\mu}}$$

όπου E_ν είναι η ενέργεια του συστήματος όταν $s_k = \nu$. Το κριτήριο της εργοδικότητας τηρείται και εδώ για τον ίδιο λόγο που τηρείται και στον αλγόριθμο Metropolis. Μετά από πεπερασμένο αριθμό βημάτων κάθε πιθανή κατάσταση είναι προσβάσιμη. Επίσης αν αντικαταστήσουμε τις τιμές του p_ν στην συνθήκη λεπτομερούς ισοζύγισης

$$\frac{p(\nu \to \nu')}{p(\nu' \to \nu)} = \frac{p_{\nu'}}{p_\nu} = \frac{e^{-\beta E_{\nu'}}}{\sum_{\mu=1}^{q} e^{-\beta E_\mu}} \times \frac{\sum_{\mu=1}^{q} e^{-\beta E_\mu}}{e^{-\beta E_\nu}} = e^{-\beta(E_{\nu'} - E_\nu)}$$

διαπιστώνουμε ότι και αυτή ικανοποιείται.

Ο αλγόριθμος Heat-bath φτάνει το σύστημα πολύ πιο γρήγορα στην κατάστασης ισορροπίας από τον αλγόριθμο Metropolis στην περίπτωση που έχουμε μεγάλο αριθμό διαφορετικών καταστάσεων του σπιν q, παρόλο που είναι υπολογιστικά πιο δαπανηρός από αυτόν.

3.3 Αλγόριθμος Wolff

Οι δύο αλγόριθμοι που αναπτύξαμε πιο πάνω έχουν πολύ καλή συμπεριφορά για συστήματα όπου η θερμοκρασία β δεν βρίσκεται στην περιοχή της κρίσιμης θερμοκρασίας $\beta_c = \ln(1+\sqrt{q})$. Στην περίπτωση όμως που συμβαίνει το αντίθετο, προκύπτουν διάφορα προβλήματα που καθιστούν του δύο αυτούς αλγόριθμους (καθώς και κάθε άλλο single-flip αλγόριθμο) υπολογιστικά ασύμφορους.

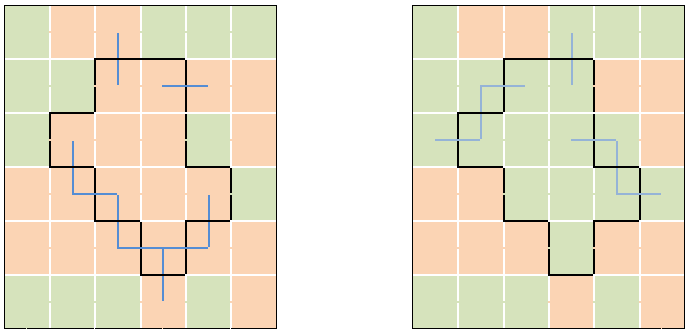
Για να δούμε πως αυτό συμβαίνει ας θεωρήσουμε ότι έχουμε ένα σύστημα που πλησιάζει την κρίσιμη θερμοκρασία από την φάση αταξίας. Αυτό που συμβαίνει είναι ότι, καθώς το σύστημα μας 'κρυώνει', αρχίζουν να δημιουργούνται μεγάλες περιοχές στο πλέγμα όπου τα σπιν έχουν τον ίδιο προσανατολισμό. Στο εσωτερικό αυτών των περιοχών, ο κάθε single-flip αλγόριθμος δυσκολεύεται να κάνει αλλαγές στα σπιν επειδή έρχεται αντιμέτωπος με πολύ χαμηλούς λόγους αποδοχής, και ουσιαστικά περιορίζεται στο αλλάζει των προσανατολισμό των σπιν που βρίσκονται στο σύνορο των περιοχών. Για να δημιουργηθεί ή να καταστραφεί μια τέτοια περιοχή απαιτούνται πολλά βήματα του αλγορίθμου, πράγμα που στοιχίζει σε υπολογιστικό χρόνο. Το φαινόμενο αυτό ονομάζεται **κρίσιμη επιβράδυνση**.

Λύση σε αυτό το πρόβλημα επιτυγχάνεται χρησιμοποιώντας του λεγόμενους **cluster αλγορίθμους**. Αυτό που κάνουν είναι να αλλάζουν με ένα βήμα το σπιν περιοχών συγκρίσιμες σε μέγεθος με τις μεγάλες περιοχές όμοιων σπιν που αναφέραμε πιο πάνω. Ένας τέτοιος αλγόριθμος είναι και ο **αλγόριθμος Wolff**.

Το κάθε βήμα του αλγορίθμου ξεκινά επιλέγοντας ένα τυχαίο σπιν-γεννήτορα, γύρο από το οποίο κατασκευάζουμε το cluster με στοχαστικό τρόπο. Αν ένα σπιν στη γειτονία του γεννήτορα είναι το ίδιο με αυτόν, το προσθέτουμε στο cluster με πιθανότητα P_{add} που εξαρτάται από την θερμοκρασία β . Η πιθανότητα P_{add} πρέπει να είναι τέτοια ώστε να ικανοποιείται η συνθήκη λεπτομερούς ισοζύγισης:

$$\frac{P(\mu \to \nu)}{P(\nu \to \mu)} = e^{-\beta(E_\nu - E_\mu)}$$

Η διαφορά ενέργειας που εμφανίζεται στο εκθετικό της πιο πάνω σχέσης προκύπτει ουσιαστικά από την καταστροφή και δημιουργία δεσμών στο σύνορο του cluster, όπως φαίνεται στο παρακάτω σχήμα όπου παρουσιάζονται με μπλε γραμμές:



Έχοντας αυτό κατά νου, συνεχίζουμε επιλέγοντας την κατάλληλη πιθανότητα επιλογής $g(\mu \to \nu)$ που ουσιαστικά ταυτίζεται με την πιθανότητα σχηματισμού του συγκεκριμένου cluster. Από τον τρόπο που δημιουργείται το cluster προκύπτει ότι μπορούμε να χωρίσουμε την πιθανότητα επιλογής σε γινόμενο τριών ανεξάρτητων όρων:

$$g(\mu \to \nu) = p_{seed} \times p_{yes}^{int} \times p_{no}^{border}$$

Ο πρώτος όρος προκύπτει από την τυχαία επιλογή του σπιν-γεννήτορα και ισούται με

$$p_{seed} = \frac{1}{N}$$

Ο δεύτερος όρος είναι η πιθανότητα, ξεκινώντας από το σπιν-γεννήτορα, να ενσωματώσουμε όλα τα σπιν του cluster στο cluster. Επειδή όμως είναι η ίδια και για την αντίστροφη διαδικασία $q(\nu \to \mu)$ δεν θα προχωρήσουμε στον υπολογισμό της. Έχουμε δηλαδή

$$p_{yes}^{int}(\mu \to \nu) = p_{yes}^{int}(\nu \to \mu) = C_{\mu\nu}$$

Και φτάνουμε έτσι στον τρίτο και πιο σημαντικό όρο στην διαδικασία ορισμού της σωστής P_{add}. Η επέκταση του cluster σταματάει όταν πούμε 'όχι' στην ενσωμάτωση των γειτονικών ομοίων σπιν. Αυτό γίνεται κάθε φορά με πιθανότητα $1 - P_{add}$. Έστω ότι στην κατάσταση μ αυτό έγινε m φορές ενώ στην κατάσταση ν έγινε n φορές. Τότε παίρνουμε για τις p_{no}^{border}

$$p_{no}^{border}(\mu \to \nu) = (1 - P_{add})^m$$

και

$$p_{no}^{border}(\nu \to \mu) = (1 - P_{add})^n$$

Αντικαθιστώντας όλα τα παραπάνω στην σχέση της συνθήκης λεπτομερούς ισοζύγισης παίρνουμε

$$\frac{P(\mu \to \nu)}{P(\nu \to \mu)} = \frac{\frac{1}{N} \times C_{\mu\nu} \times (1 - P_{add})^m A(\mu \to \nu)}{\frac{1}{N} \times C_{\mu\nu} \times (1 - P_{add})^n A(\nu \to \mu)} = e^{-\beta(E_\nu - E_\mu)}$$

Τώρα επειδή κάθε δεσμός που δημιουργείσαι μειώνει την ενέργεια κατά ένα ενώ κάθε δεσμός που σπάει την αυξάνει κατά ένα (βλ. χαμιλτονιανή του προτύπου Potts) έχουμε

$$E_\nu - E_\mu = (-n) - (-m) = m - n$$

Εισάγοντας και αυτή την πληροφορία στην σχέση παίρνουμε

$$(1 - P_{add})^{m-n} \frac{A(\mu \to \nu)}{A(\nu \to \mu)} = e^{-\beta(m-n)}$$

Όπως έχουμε αναφέρει και σε προηγούμενη ενότητα, στόχος μας είναι να έχουμε το δυνατό μεγαλύτερους λόγους αποδοχής. Από την πιο πάνω σχέση προκύπτει ότι μπορούμε να θέσουμε και τους δύο λόγους αποδοχής ίσους με την μονάδα

$$A(\mu \to \nu) = A(\nu \to \mu) = 1$$

και παράλληλα να ικανοποιήσουμε την συνθήκη λεπτομερούς ισοζύγισης, αν ορίσουμε την P_{add} να ισούται με

$$P_{add} = 1 - e^{-\beta}$$

Είναι ξεκάθαρο ότι το κριτήριο της εργοδικότητας τηρείται από τον αλγόριθμο του Wolff, καθώς ο στοχαστικός τρόπος με τον οποίο δημιουργούνται τα clusters επιτρέπει την δημιουργία τέτοιων που να περιέχουν μόνο ένα στοιχείο, καθιστώντας έτσι όλες τις πιθανές καταστάσεις προσβάσεως σε πεπερασμένο αριθμό βημάτων του αλγορίθμου.

Συνοψίζοντας τα πιο πάνω, το κάθε βήμα του αλγόριθμου του Wolff έχει ως εξής:

1. Επιλέγουμε τυχαία το σπιν-γεννήτορα από το πλέγμα μας που θα είναι το πρώτο μέλος του cluster
2. Επαγωγικά, για κάθε νέο μέλος του cluster εξετάζουμε τους πλησιέστερους γείτονες του που δεν ανήκουν στο cluster. Αν έχουν την ίδια τιμή σπιν με το cluster, τα προσθέτουμε σε αυτό με πιθανότητα $P_{add} = 1 - e^{-\beta}$.
3. Όταν δεν έχουμε άλλα νέα μέλη, η διαδικασία σχηματισμού του cluster σταματάει
4. Δίνουμε στα σπιν του cluster μία νέα (διάφορη της αρχικής) τιμή επιλεγμένη τυχαία από το σύνολο των $q - 1$ πιθανών τιμών για το σπιν.

Μέρος Β΄

Υλοποίηση των αλγορίθμων

4

Δοκιμή 1

Και στις τρεις περιπτώσεις που αναφέραμε πιο πάνω, ξεκινάμε ορίζοντας ένα array που αντιπροσωπεύει το πλέγμα μας και το θέτουμε στην αρχική του κατάσταση. Πάνω σε αυτό ορίζουμε και τις συνοριακές συνθήκες που είναι κατάλληλες για την κάθε περίπτωση. Στην πρώτη δοκιμαστική υλοποίηση χρησιμοποίησα ελικοειδής συνοριακές συνθήκες, τέτοιες που να ορίζουν ένα δισδιάστατο τετραγωνικό πλέγμα.

Στους αλγόριθμους Metropolis και Heat-bath, όπου απαιτείται υπολογισμός των ενεργειών του πλέγματος, μπορούμε να απλοποιήσουμε την διεργασία αν λάβουμε υπόψη το ότι έχουμε να κάνουμε με single-flip αλγόριθμους. Συγκεκριμένα στην περίπτωση του αλγόριθμου Metropolis έχουμε να κάνουμε με διαφορά ενέργειας μεταξύ της αρχικής και της υπό εξέταση τελικής κατάστασης. Επειδή όμως αυτή η ενεργειακή διαφορά είναι αποτέλεσμα μίας και μόνο αλλαγής σε τιμή του σπιν, μπορούμε να θεωρήσουμε την ενεργειακή συνεισφορά μόνο του σπιν που βρίσκεται στην πλεγματική θέση όπου θα γίνει η αλλαγή. Η ενεργειακή συνεισφορά των υπολοίπων σπιν είναι η ίδια και για τις δύο καταστάσεις και έτσι αλληλοαναιρείται στην διαφορά. Δεν χρειάζεται δηλαδή να υπολογίζουμε σε κάθε βήμα του αλγορίθμου την ενέργεια ολοκλήρου του συστήματος, μειώνοντας έτσι δραματικά τον υπολογιστικό χρόνο που απαιτείται για την προσομοίωση μας. Αν ακόμη λάβουμε υπόψη και το ότι κάθε πλεγματική θέση έχει τέσσερις πλησιέστερους γείτονες (που υπεισέρχονται στον υπολογισμό της ενεργειακής συνεισφοράς του εξεταζόμενου σπιν) βρίσκουμε ότι η διαφορά ενέργειας μπορεί να πάρει μόνο τις ακέραιες τιμές στο διάστημα $(-4,4)$. Έτσι μπορούμε να περιορίζουμε περεταίρω τον υπολογιστικό χρόνο αν υπολογίσουμε στην αρχή της προσομοίωσης τις τιμές των εκθετικών που εμφανίζονται στους υπολογισμούς μας και τις αποθηκεύσουμε σε ένα array στο οποίο θα ανατρέχουμε όταν προκύπτει η ανάγκη.

Στην υλοποίηση του αλγόριθμου Heat-bath τα πράγματα είναι λίγο πιο πολύπλοκα. Όπως έχουμε δει και πιο πάνω η κάθε νέα κατάσταση επιλέγεται βάση του στατιστικού βάρους της

$$p_n = \frac{e^{-\beta E_n}}{\sum_{m=1}^{q} e^{-\beta E_m}}$$

Επειδή όμως είναι ασύμφορο να υπολογίζουμε κάθε φορά την ολική ενέργεια του συστήματος για κάθε μία από τις q πιθανές καταστάσεις, εφαρμόζουμε ένα τέχνασμα που μας απαλλάσσει από αυτή την ανάγκη. Πολλαπλασιάζουμε την σχέση με τον όρο $e^{\beta E_\nu}/e^{\beta E_\nu} = 1$, όπου ν είναι μία κατάσταση που το σπιν της εξεταζόμενης πλεγματικής θέσης δεν συνεισφέρει στην ολική ενέργεια του συστήματος (δεν είναι συγγραμμικό με κανένα από τα σπιν των πλησιέστερων γειτόνων). Τότε η σχέση μας γίνεται:

$$\begin{aligned} p_n &= \frac{e^{-\beta E_n}}{\sum_{m=1}^{q} e^{-\beta E_m}} \frac{e^{\beta E_\nu}}{e^{\beta E_\nu}} \\ &= \frac{e^{-\beta (E_n - E_\nu)}}{\sum_{m=1}^{q} e^{-\beta (E_m - E_\nu)}} \\ &= \frac{e^{-\beta \Delta E_n}}{\sum_{m=1}^{q} e^{-\beta \Delta E_m}} \end{aligned}$$

Αυτό ουσιαστικά σημαίνει ότι ο υπολογισμός των στατιστικών βαρών γίνεται τοπικός. Τα στατιστικά βάρη εξαρτώνται μόνο από την ενεργειακή συνεισφορά του υπό εξέταση σπιν. Όπως και στην περίπτωση του αλγόριθμου Metropolis τα εμφανιζόμενα εκθετικά υπολογίζονται στην αρχή της προσομοίωσης και αποθηκεύονται σε ένα array στο οποίο θα ανατρέχουμε όταν προκύπτει η ανάγκη.

Από την άλλη πλευρά, η υλοποίηση του αλγόριθμου Wolff είναι πιο απλή. Το κάθε βήμα ξεκινά επιλέγοντας τυχαία ένα σπιν-γεννήτορα που γύρω από αυτό θα αναπτυχθεί το cluster. Έπειτα δίνεται σε αυτό το σπιν η νέα τιμή που επιλέχτηκε. Στη συνέχεια κατασκευάζεται ένα stack που θα κρατήσει τα σπιν του cluster. Με επαναληπτικό τρόπο προστίθενται στο stack τα γειτονικά σπιν με την πιθανότητα που οριστικέ πιο πάνω με την προϋπόθεση ότι έχουν την ίδια τιμή με το σπιν-γεννήτορα. Ταυτόχρονα τους δίνεται η νέα τιμή του σπιν που έχει επιλεχτεί. Όταν το stack έχει πια αδειάσει, το βήμα του αλγόριθμου έχει ολοκληρωθεί.

Για να έχει νόημα η όποια προσομοίωση πρέπει να μπορούμε να εξάγουμε κάποια μετρήσιμη ποσότητα από αυτή. Η παρούσα εφαρμογή παίρνει μετρήσεις για την ολική ενέργεια του συστήματος καθώς και της μαγνήτισης ανά πλεγματική θέση. Για ένα σύστημα N πλεγματικών θέσεων, αυτό γίνεται μία φορά ανά N βήματα (1 σάρωση) του αλγορίθμου. Μετά το πέρας της προσομοίωσης γίνεται επεξεργασία αυτών των μετρήσεων χρησιμοποιώντας την μέθοδο ανάλυσης Jackknife που δίνει την μέση τιμή και τις διακυμάνσεις αυτών των μεγεθών στην κατάσταση ισορροπίας.

Έχω επίσης προσθέσει μία μέθοδο που παράγει μία οπτική αναπαράσταση του πλέγματος η οποία καλείται μετά από κάθε σάρωση. Οι εικόνες αυτές

μπορούν στη συνέχεια να χρησιμοποιηθούν για την κατασκευή βίντεο που δείχνει πως το σύστημα φτάνει στην κατάσταση ισορροπίας και πως διακυμαίνεται έπειτα.

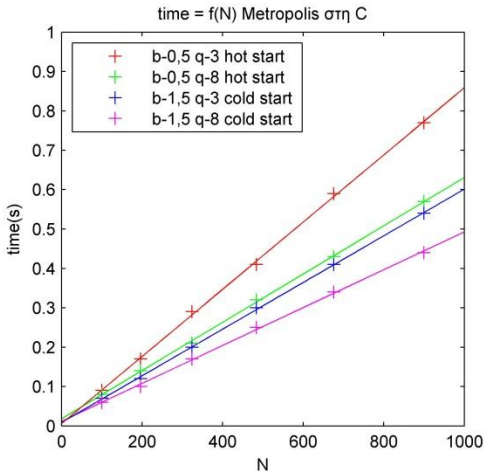
Για μία πιο εκτενή περιγραφή των υλοποιήσεων παραπέμπω στο παράρτημα 1 όπου είναι συγκεντρωμένος ο σχετικός κώδικας μαζί με λεπτομερή σχόλια.

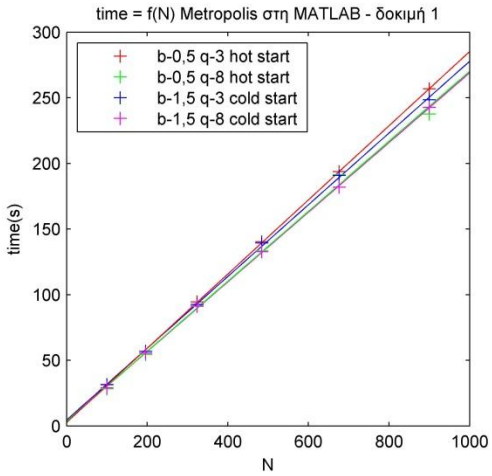
4.1 Εκτέλεση – Αποτελέσματα

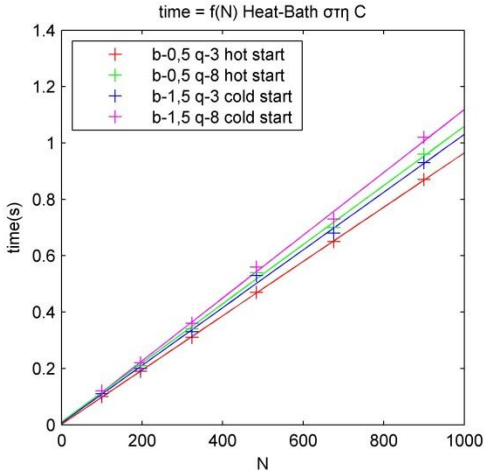
Όλες οι προσομοιώσεις που ακολουθούν έγιναν με $q = 3\ 8$, και $b = 0{,}5\ 1{,}5$ (παραμαγνητική και σιδηρομαγνητική περιοχή αντίστοιχα).

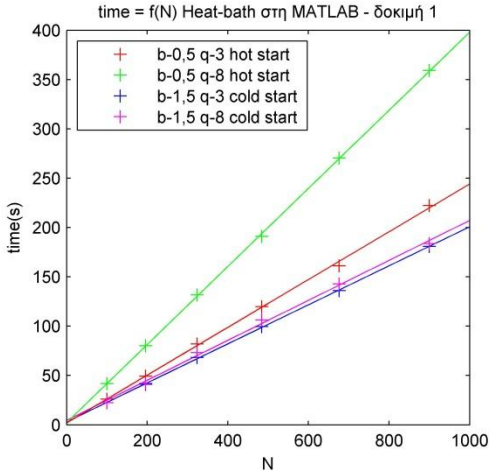
Αυτό που γίνεται αμέσως φανερό με την εκτέλεση της εφαρμογής είναι ο μεγάλος χρόνος που παίρνει να εκτελεστεί, συγκριτικά με την αντίστοιχη υλοποίηση σε C στην οποία είναι στηριγμένος ο κώδικας (ο κώδικας της C που χρησιμοποίησα είναι μία τροποποιημένη μορφή του κώδικα των M. E. J. Newman και G. T. Barkema για το πρότυπο Ising[2]). Πιο συγκεκριμένα ο χρόνος εκτέλεσης είναι μεγαλύτερος κατά ένα παράγοντα 450 για τον αλγόριθμο Metropolis, 250 για τον αλγόριθμο Heat-bath και 8 για τον αλγόριθμο Wolff, όπως υποδηλώνουν τα παρακάτω διαγράμματα.. Αυτό οφείλεται στον τρόπο λειτουργίας του περιβάλλοντος MATLAB όπου η διαδικασία της μεταγλώττισης του κώδικα γίνεται σε πραγματικό χρόνο κατά το τρέξιμο του. Αυτό καθιστά τα loops ιδιαίτερα ακριβά σε υπολογιστικό χρόνο. Παρόλα ταύτα, όπως φαίνεται, διατηρείται η γραμμική εξάρτηση του χρόνου εκτέλεσης από το μέγεθος N του συστήματος

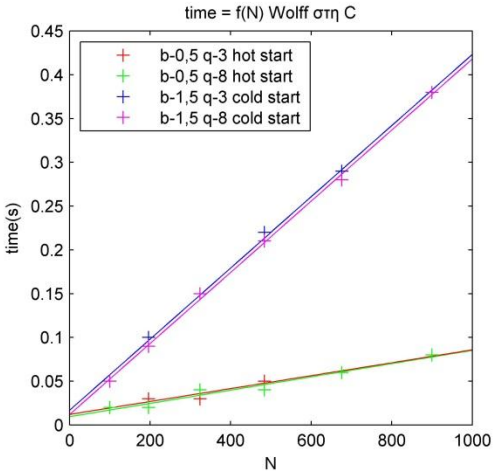
Για να περιοριστεί αυτό το πρόβλημα, ο κώδικας πρέπει να ξαναγραφτεί με τέτοιο τρόπο που να αποφεύγεται η χρήση των loops. Αυτό γίνεται εφικτό με την χρήση πινάκων. Εξάλλου το MATLAB (που το όνομα του είναι συντομογραφία για MATrix LABoratory) εξειδικεύεται στον χειρισμό δεδομένων σε μορφή πινάκων, δίνοντας την δυνατότητα στις περισσότερες συναρτήσεις του να παίρνουν σαν όρισμα arrays ή πίνακες

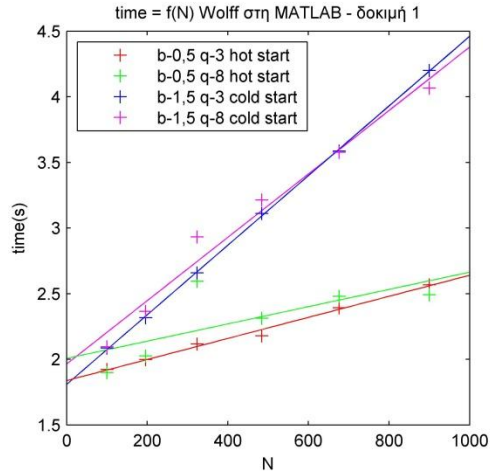

time = f(N) Metropolis στη C
b-0,5 q-3 hot start
b-0,5 q-8 hot start
b-1,5 q-3 cold start
b-1,5 q-8 cold start
time(s)
N


time = f(N) Metropolis στη MATLAB - δοκιμή 1
b-0,5 q-3 hot start
b-0,5 q-8 hot start
b-1,5 q-3 cold start
b-1,5 q-8 cold start
time(s)
N


time = f(N) Heat-Bath στη C
b-0,5 q-3 hot start
b-0,5 q-8 hot start
b-1,5 q-3 cold start
b-1,5 q-8 cold start
time(s)
N


time = f(N) Heat-bath στη MATLAB - δοκιμή 1
b-0,5 q-3 hot start
b-0,5 q-8 hot start
b-1,5 q-3 cold start
b-1,5 q-8 cold start
time(s)
N


time = f(N) Wolff στη C
b-0,5 q-3 hot start
b-0,5 q-8 hot start
b-1,5 q-3 cold start
b-1,5 q-8 cold start
time(s)
N


time = f(N) Wolff στη MATLAB - δοκιμή 1
b-0,5 q-3 hot start
b-0,5 q-8 hot start
b-1,5 q-3 cold start
b-1,5 q-8 cold start
time(s)
N

.

5

Δοκιμή 2

Η δεύτερη υλοποίηση των αλγορίθμων λαμβάνει υπόψη τον τρόπο λειτουργίας του περιβάλλοντος MATLAB. Καθώς όμως οι αλγόριθμοι με την μορφή που είχαν δεν επέτρεπαν το γράψιμο τους σε μορφή πινάκων χρειάστηκε να κάνω κάποιες τροποποιήσεις σε αυτούς.

Στην περίπτωση των αλγορίθμων Metropolis και Heat-bath στηρίχτηκα σε μία κρίσιμη παρατήρηση – για πλέγματα με ζυγό μήκος πλευράς μπορούμε να τα χωρίσουμε με απλό τρόπο σε δύο ομάδες έτσι ώστε οι πλησιέστεροι γείτονες κάθε πλεγματικής θέση που ανήκει στη μία ομάδα, να βρίσκονται όλοι στην άλλη. Προϋπόθεση για αυτό είναι η μετάβαση από ελικοειδής σε απλές περιοδικές συνοριακές συνθήκες. Ο τρόπος χωρισμού του πλέγματος μπορεί να γίνει καλύτερα κατανοητός μέσω του παρακάτω σχήματος ενός 8×8 τετραγωνικού πλέγματος. Η πρώτη ομάδα παρουσιάζεται με μαύρο χρώμα ενώ η δεύτερη με άσπρο.

	57	58	59	60	61	62	63	64	
8	1	2	3	4	5	6	7	8	1
16	9	10	11	12	13	14	15	16	9
24	17	18	19	20	21	22	23	24	17
32	25	26	27	28	29	30	31	32	25
40	33	34	35	36	37	38	39	40	33
48	41	42	43	44	45	46	47	48	41
56	49	50	51	52	53	54	55	56	49
64	57	58	59	60	61	62	63	64	57
	1	2	3	4	5	6	7	8	

Σε αυτό το σημείο είναι που υπεισέρχεται η διαφοροποίηση από τους αλγορίθμους. Αντί κάθε σάρωση του αλγορίθμου να επιλέγει με τυχαίο τρόπο N θέσεις και να τους αλλάζει το σπιν βάση των διάφορων κανόνων που ορίζει ο κάθε αλγόριθμος, η νέα υλοποίηση χειρίζεται την κάθε μία από τις δύο ομάδες σαν ένα πίνακα και δρα πάνω σε όλα τα στοιχεία της ταυτόχρονα. Αυτό

είναι επιτρεπτό επειδή πρακτικά η αλλαγή ή η μη αλλαγή του σπιν σε κάθε μεμονωμένη πλεγματική θέση εξαρτάται μόνο από τις τιμές του σπιν που έχουν οι πλησιέστεροι γείτονες αυτής.

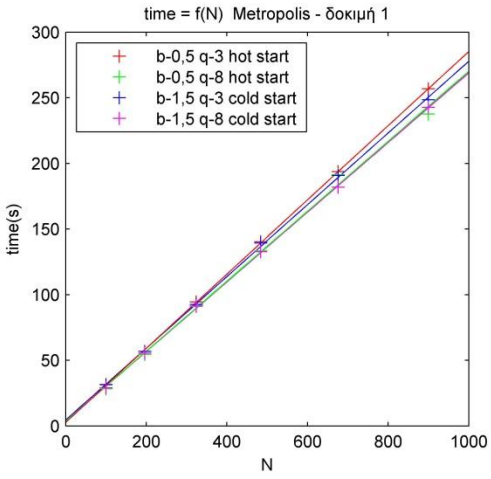
Οι διάφορες συγκρίσεις που παρουσιάζονταν στις δομές if – then – else τώρα αντικαταστάθηκαν με logical arrays που μέσω διαφόρων πολλαπλασιασμών και εντολών διαμόρφωσης μας δίνουν το ζητούμενο αποτέλεσμα. Αυτή η διαφοροποίηση όπως θα δούμε παρακάτω βελτιώνει σημαντικά τον χρόνο εκτέλεσης της εφαρμογής.

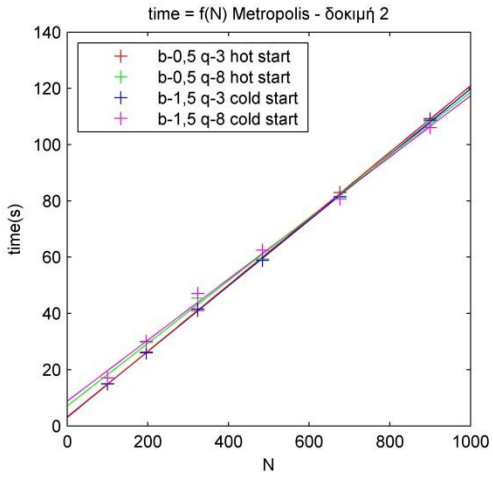
Ο αλγόριθμος Wolff από την άλλη, με την ανάγκη ανάπτυξης του cluster αντιστάθηκε στις προσπάθειες μου να εφαρμόσω ένα παρόμοιο τέχνασμα με το πιο πάνω. Αυτό που τελικά πέτυχα ήταν να γράψω ένα κώδικα που δεν έχει ανάγκη την δομή του stack για να λειτουργήσει, καθώς σε κάθε επανάληψη του while-loop, η εφαρμογή δρούσε ταυτόχρονα σε όλα τα υποψήφια για προσθήκη στο cluster σπιν, δίνοντας στο τέλος της τα υποψήφια σπιν για την επόμενη επανάληψη.

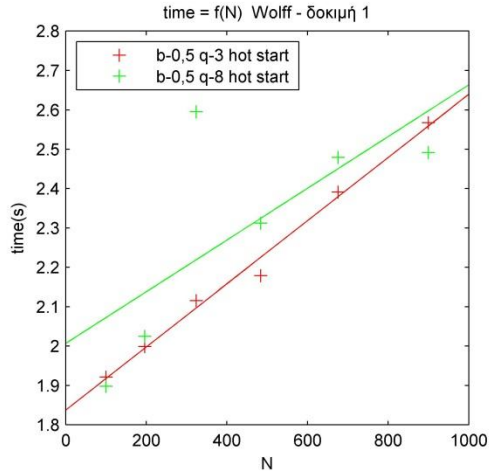
Σε αυτή την υλοποίηση έχει προστεθεί και μία επιπλέων μέθοδος που στην αρχή της προσομοίωσης αποθηκεύει σε ένα πίνακα $N \times 4$ τις διευθύνσεις των πλησιέστερων γειτόνων για όλες της πλεγματικές θέσεις. Αυτό κάνει την γραφή του κώδικα πιο απλή και καλυτερεύει ελαφρώς την απόδοση του.

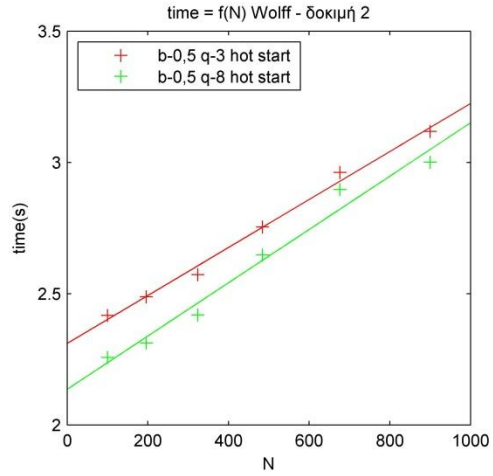
5.1 Εκτέλεση – Αποτελέσματα

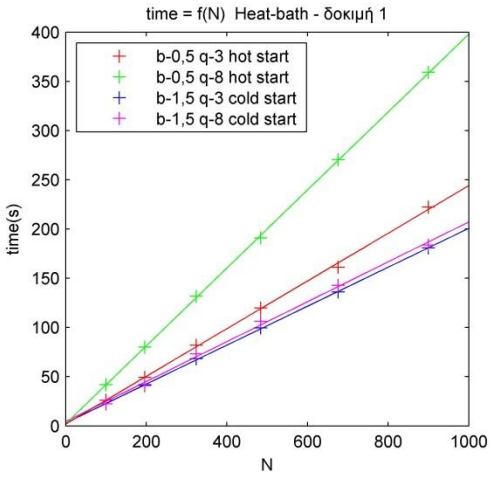
Εκτελώντας την δεύτερη υλοποίηση για τους αλγόριθμους διαπιστώνουμε ότι μας δίνει τα ίδια αποτελέσματα (στα πλαίσια του στατιστικού σφάλματος) με την πρώτη. Αυτό που διαφέρει όμως από την πρώτη εκτέλεση είναι ο χρόνος που παίρνει η διαδικασία της προσομοίωσης. Συγκεκριμένα για τον αλγόριθμο Metropolis έχουμε μειωμένο υπολογιστικό χρόνο κατά ένα παράγοντα 2,5 ενώ για τον αλγόριθμο Heat-bath κατά ένα παράγοντα μεταξύ 2 και 7 ανάλογα με τις παραμέτρους της προσομοίωσης (κυρίως του β). Για τον αλγόριθμο Wolff όμως, όπως ήταν αναμενόμενο, δεν έχουμε κάποιο κέρδος σε υπολογιστικό χρόνο. Στα παρακάτω διαγράμματα φαίνονται μαζεμένα αυτά τα αποτελέσματα. Παρόλη την βελτίωση που είχαμε στον υπολογιστικό χρόνο, η εφαρμογή στην C εξακολουθεί να είναι πολύ πιο γρήγορη.

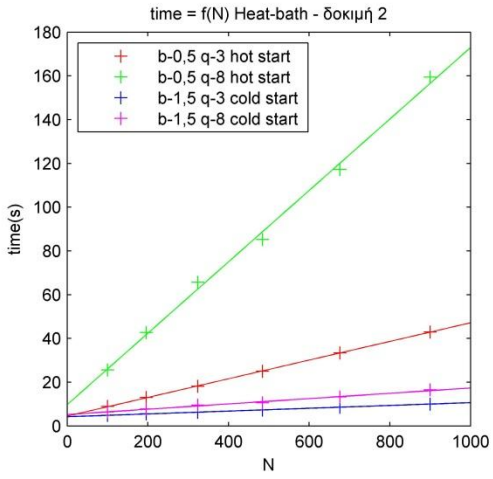

time = f(N) Metropolis - δοκιμή 1
b-0,5 q-3 hot start
b-0,5 q-8 hot start
b-1,5 q-3 cold start
b-1,5 q-8 cold start
time(s)
N


time = f(N) Metropolis - δοκιμή 2
b-0,5 q-3 hot start
b-0,5 q-8 hot start
b-1,5 q-3 cold start
b-1,5 q-8 cold start
time(s)
N


time = f(N) Wolff - δοκιμή 1
b-0,5 q-3 hot start
b-0,5 q-8 hot start
time(s)
N


time = f(N) Wolff - δοκιμή 2
b-0,5 q-3 hot start
b-0,5 q-8 hot start
time(s)
N


time = f(N) Heat-bath - δοκιμή 1
b-0,5 q-3 hot start
b-0,5 q-8 hot start
b-1,5 q-3 cold start
b-1,5 q-8 cold start
time(s)
N


time = f(N) Heat-bath - δοκιμή 2
b-0,5 q-3 hot start
b-0,5 q-8 hot start
b-1,5 q-3 cold start
b-1,5 q-8 cold start
time(s)
N

6

Τελικές βελτιώσεις

Για περεταίρω βελτίωση του κώδικα ανατρέχω στην εφαρμογή του profiler που μας λέει πώς μοιράζεται ο υπολογιστικός χρόνος στις διάφορες μεθόδους (ή και ακόμα εντολές). Μετά από ένα τρέξιμο της υλοποίηση χρησιμοποιώντας τον αλγόριθμο Metropolis με το πλάτος του πλέγματος ίσο με $L = 10$, διαπιστώνουμε όπως φυσικά και θα ήταν αναμενόμενο, ότι ο περισσότερος χρόνος αναλώνεται στην εκτέλεση των σαρώσεων που κάνει ο αλγόριθμος. Ζητώντας από τον profiler να μας δείξει πως ο χρόνος αυτός μοιράζεται στις διάφορες εντολές την μεθόδου αυτής παίρνουμε την παρακάτω αναφορά:

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
23	newstate(k) = avalstates(randi...	1000000	10.243 s	57.0%	
22	avalstates = find(ALLSTATES-ol...	1000000	4.400 s	24.5%	
24	end	1000000	1.334 s	7.4%	
26	Ei = sum((S(NN(n,:))==[oldstat...	20000	0.383 s	2.1%	
15	global Q N PROB S ALLSTATES NN	20000	0.373 s	2.1%	
All other lines			1.232 s	6.9%	
Totals			17.964 s	100%	

Αυτό που ουσιαστικά μας λέει είναι ότι το 81,5% του χρόνου αναλώνεται στην εύρεση νέων καταστάσεων σπιν διάφορων των αρχικών. Μία σκέψη για την αντιμετώπιση αυτού του προβλήματος είναι να διαλέγουμε τις νέες καταστάσεις χωρίς τον πιο πάνω περιορισμό. Αυτό όμως θα είχε σαν αποτέλεσμα ορισμένες αλλαγές στον σπιν να γίνονται προς την ίδια με την αρχική κατάσταση. Συγκεκριμένα (με πρόχειρους υπολογισμούς) μία στις κάθε q δράσεις του αλγορίθμου δεν θα συνεισφερε στην εξέλιξη του συστήματος, πράγμα που θα σήμαινε ότι, για να πάρουμε τα ίδια αποτελέσματα με την υλοποίηση που χρησιμοποιήσαμε πιο πάνω, θα πρέπει να αυξήσουμε τον χρόνο τρεξίματος της προσομοίωσης κατά ένα ποσοστό $1/q$. Ένα ποσό που για μικρά q είναι σημαντικό. Ένας άλλος τρόπος να λύσουμε το πρόβλημα αυτό είναι η εύρεση ενός άλλου, πιο αποδοτικού τρόπου δημιουργίας των νέων καταστάσεων έτσι ώστε να τηρείται η απαίτηση να είναι διάφορες από την αρχική. Αυτό επιτυγχάνεται με την αντικατάσταση των

γραμμών 19-24 στην μέθοδο metmat() με τον παρακάτω κώδικα:

```
newstate  = randi(Q,N/2,1);
newstate = abs(newstate-((oldstate==newstate)*double(Q+1)));
```

Χρησιμοποιώντας τον profiler διαπιστώνουμε ότι τώρα η γραμμή αυτή παίρνει μόλις το 9% του χρόνου τρεξίματος της μεθόδου

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
15	global Q N PROB S NN	20000	0.303 s	15.5%	
23	Ei = sum((S(NN(n,:))==[oldstat...	20000	0.282 s	14.4%	
24	Ef = sum((S(NN(n,:))==[newstat...	20000	0.217 s	11.1%	
19	newstate = randi(Q,N/2,1);	20000	0.212 s	10.8%	
27	accept = (delta<=0) | (rand...	20000	0.192 s	9.8%	
All other lines			0.752 s	38.4%	
Totals			1.958 s	100%	

Αυτό σημαίνει ότι για *q* μικρότερο του 10 έχουμε καλύτερη απόδοση σε σχέση με την περίπτωση που παίρνουμε τις νέες καταστάσεις ανεξάρτητα από τις αρχικές.

Επαναλαμβάνοντας την πιο πάνω διαδικασία και για την υλοποίηση του αλγόριθμου Heat-bath βλέπουμε ότι και πάλι υπάρχει μία γραμμή κώδικα που παίρνει δραματικά μεγαλύτερο μερίδιο του υπολογιστικού χρόνου σε σχέση με τις υπόλοιπες (84% μαζί με τις υπόλοιπες γραμμές που κάνουν την συγκεκριμένη εργασία).

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
93	qs = nonzeros(nonzeros(nonzero...	429322	22.300 s	81.2%	
96	S(nremain(ii)) = qs(bb(ii));	429322	0.763 s	2.8%	
20	global N Q S NN PROB ALLSTATES	20000	0.359 s	1.3%	
33	nnz = nnz + [(z12+z13+z14),(z1...	20000	0.357 s	1.3%	
69	logicarr = nonzeros(n .* tup	20000	0.301 s	1.1%	
All other lines			3.377 s	12.3%	
Totals			27.457 s	100%	

Αυτό είναι το κομμάτι του κώδικα που βρίσκει τις νέες καταστάσεις στην περίπτωση που δεν ταυτίζονται με τις καταστάσεις των πλησιέστερων γειτόνων. Αντικατέστησα το εν λόγω for loop της μεθόδου heat_mat() με το παρακάτω κομμάτι κώδικα:

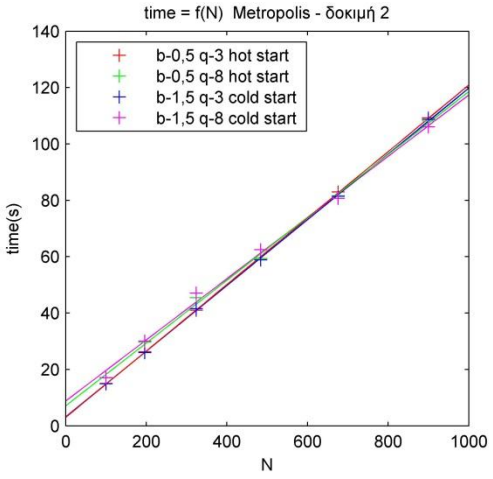
```
avalstates = repmat(1:Q,length(bb),1);
avalarray = logical((repmat(nnsremain(:,1),1,Q)==avalstates)+ ...
                    (repmat(nnsremain(:,2),1,Q)==avalstates)+ ...
                    (repmat(nnsremain(:,3),1,Q)==avalstates)+ ...
                    (repmat(nnsremain(:,4),1,Q)==avalstates));

avalstates(avalarray)=0;
avalstates = sort(avalstates,2,'descend');
indices_array = sub2ind(size(avalstates), 1:length(bb), bb')';
```

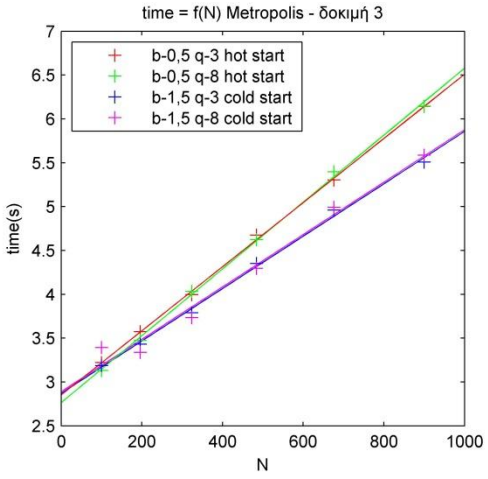
Με αυτό το κομμάτι κώδικα ο χρόνος που χιάζεται για την επιλογή των νέων καταστάσεων έχει μειωθεί στο 50% του χρόνου τρεξίματος της μεθόδου, όπως εξάλλου φαίνεται και στον παρακάτω πίνακα του profiler:

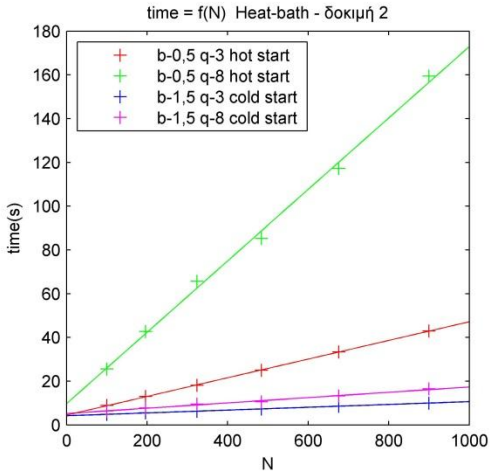
Lines where the most time was spent

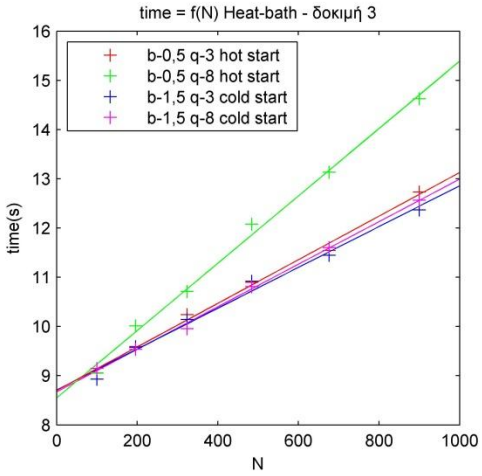
Line Number	Code	Calls	Total Time	% Time	Time Plot
92	avalarray = logical((repmat(nn...	20000	4.673 s	38.3%	
99	indices_array = sub2ind(size(a...	20000	1.491 s	12.2%	
91	avalstates = repmat(1:Q,length...	20000	1.061 s	8.7%	
20	global N Q S NN PROB ALLSTATES	20000	0.420 s	3.4%	
33	nnz = nnz + [(z12+z13+z14),(z1...	20000	0.368 s	3.0%	
All other lines			4.186 s	34.3%	
Totals			12.199 s	100%	

Τα αποτελέσματα και για τις δύο υλοποιήσεις παρουσιάζονται στα παρακάτω διαγράμματα:









Παρατηρούμε ότι έχουμε μία περαιτέρω μείωση του υπολογιστικού χρόνου κατά ένα παράγοντα 20 για την περίπτωση του αλγόριθμου Metropolis, και κατά ένα παράγοντα που κυμαίνεται στο διάστημα [1,10] ανάλογα με τις παραμέτρους της προσομοίωσης, για την περίπτωση του αλγορίθμου Heat-bath.

7

Συμπεράσματα

Σε αυτό το σημείο έχουμε αρκετά δεδομένα για να εξάγουμε τα τελικά μας συμπεράσματα. Ο αρχικός στόχος της εργασίας – η δημιουργία μιας εφαρμογής που να υλοποιεί προσομοιώσεις βάση το πρότυπο Potts στις δύο διαστάσεις στο υπολογιστικό περιβάλλον του MATLAB – επιτεύχθηκε σε κάποιο βαθμό. Τα αποτελέσματα που δίνει η εφαρμογή που αναπτύχτηκε εδώ είναι σε συμφωνία με τα αποτελέσματα άλλων δοκιμασμένων και επιβεβαιωμένων εφαρμογών. Στην πορεία του σχεδιασμού και ανάπτυξης του κώδικα έγινε επίσης κατορθωτό να μειωθεί δραματικά ο υπολογιστικός που αναλωνόταν στο τρέξιμο της εφαρμογής.

Παρόλα αυτά ,όσο αφορά τους χρόνους τρεξίματος της, η εφαρμογή μας υστερεί κατά πολύ έναντι της αντίστοιχης εφαρμογής που αναπτύχτηκε στην C. Καλύτερους χρόνους τρεξίματος πήραμε στην περίπτωση του αλγόριθμου Metropolis που προκύπτει να είναι πιο αργός από τον αντίστοιχο της C κατά ένα παράγοντα που κυμαίνεται στο διάστημα [9,14]. Τα πράγματα ήταν χειρότερα στην περίπτωση των αλγορίθμων Heat-bath και Wolff, που έδωσαν χρόνους κατά ~26 και ~37 φορές μεγαλύτερους αντίστοιχα. Οι διαφορές στον παράγοντα αυτό οφείλονται στην διαφορετική διαδικασία και στα διαφορετικά τεχνάσματα που χρησιμοποιήθηκαν κατά μετατροπή του κώδικα σε μια μορφή που να εκμεταλλεύεται τον τρόπο λειτουργίας του υπολογιστικού περιβάλλοντος του MATLAB.

Αυτό ίσος να αποθάρρυνε κάποιον από το να χρησιμοποιήσει την εφαρμογή μου ή και ακόμη το ίδιο το υπολογιστικό περιβάλλον του MATLAB για αυτή την δουλειά. Για αυτό το λόγο σε αυτό το σημείο νιώθω την ανάγκη να κάνω τον δικηγόρο του διαβόλου και να δώσω ένα αντίλογο στο πιο πάνω επιχείρημα.

Η εμπειρία μου στο περιβάλλον του MATLAB κατά το γράψιμο της εφαρμογής μου ήταν ιδιαίτερα ευχάριστη. Η δυνατότητα που μου παρείχε να δημιουργώ δοκιμαστικές μεταβλητές στον χώρο εργασίας (workspace) με ευκολία και να τις χρησιμοποιώ για να τρέξω μεμονωμένες γραμμές κώδικα ήταν ένας μεγάλος βοηθητικός παράγοντας στο γράψιμο. Πολύ βοηθητική στη διαδικασία της βελτίωσης του κώδικα ήταν και εφαρμογή του profiler που

όπως είδαμε πιο πάνω δείχνει το πως μοιράζεται ο υπολογιστικό χρόνος στις διάφορες γραμμές του κώδικα κατά το τρέξιμο του. Πέραν των πιο πάνω ευκολιών για την διαδικασία γραψίματος του κώδικα, βρήκα πολύ βοηθητική και την δυνατότητα ανάλυσης και παρουσίασης δεδομένων που μου παρείχε το περιβάλλον του MATLAB. Συγκεκριμένα, επειδή τα περισσότερα δεδομένα ήταν αποθηκευμένα σε arrays και πίνακες, μου ήταν πολύ εύκολο να δράσω πάνω σε αυτά. Εύκολη ήταν και η διαδικασία δημιουργίας των γραφημάτων που παρουσιάζω στα προηγούμενα κεφάλαια. Φτιάχνοντας ένα πρόχειρο γράφημα με μία απλή εντολή, μπορούσα στην συνέχεια να το διαμορφώσω με γραφικό τρόπο και στην συνέχεια να πάρω τον κώδικα που δίνει το γράφημα αυτό και μετά από κατάλληλη τροποποίηση να τον χρησιμοποιήσω για να φτιάξω όλα τα υπόλοιπα.

Εν κατακλείδι, το αν το υπολογιστικό περιβάλλον του MATLAB είναι κατάλληλο για την ανάπτυξη μίας εφαρμογής που να προσομοιώνει ένα μαγνητικό σύστημα βάση του προτύπου Potts, είναι κάτι που θα το αποφασίσει ο εκάστοτε ερευνητής ανάλογα με τις ανάγκες του προβλήματος που πρέπει να λύσει. Αν ο υπολογιστικός χρόνος δεν είναι σημαντικός παράγοντας ή αν το πρόβλημα ασχολείται με μικρά συστήματα τότε συνιστώ την χρήση του περιβάλλοντος του MATLAB για την ανάπτυξη των σχετικών εφαρμογών. Στην αντίθετη περίπτωση καλύτερο θα ήταν να γίνει χρίση μιας καταλληλότερης γλώσσας προγραμματισμού όπως παραδείγματος χάριν η C/C++.

Παραρτήματα

A

Κώδικας υλοποίησης

Σε αυτό το παράστημα παραθέτω τον κώδικα που χρησιμοποιήθηκε στις σχετικές προσομοιώσεις. Σχετικά με την επεξήγηση των διεργασιών που τελούνται, τα σχόλια που περιλαμβάνονται είναι αρκετά κατατοπιστικά.

A.1 Δοκιμή 1

```
function [E,M] = potts_naive(l,beta,q,nsweep,start,algorithm)
% POTTS_NAIVE(l,beta,q,nsweep,start,algoritm) simulates a magnet
% according to the 2-D Potts model
%
% l         - side length of  our square lattice
% beta      - inverse temperature
% q         - number of different spin states
% nsweep    - number of iterations
% start     - 0 for cold start
%             1 for hot start
%             2 for old configuration. Load the old conf in the global
%               variable PRESET
% algorithm - 0 for Metropolis algorithm
%             1 for Wolff algorithm
%             2 for Heat-bath algorithm
%
global L Q N XNN YNN PROB PADD S ALLSTATES NN IMGCOUNT PRESET

%% Initialize variables
Q = int32(q);
L = l;
N = L*L;
XNN = 1;
YNN = L;
IMGCOUNT = 1;

switch start
    case 0 %cold start
        S = int32(ones(N,1));
    case 1 %hot start
        S = int32(randi(Q,N,1));
    case 2 % preset conf
        S = PRESET;
end
switch algorithm
    case 0
        PROB = exp(-((1:4)*beta));
```

```
    case 1
        PADD = 1-exp(-beta);
    case 2
        PROB = exp(((1:4)*beta));
        NN = nnfinder_hel(L);
        ALLSTATES = 1:Q;
end

E = zeros(nsweep,1);
M = zeros(nsweep,1);

%% Simulation and measurements
for isweep = 1:nsweep,
    switch algorithm
        case 0 % Metropolis algorithm
            met_naive();
            [e,m] = measurements();
        case 1 % Wolff algorithm
            wolff_naive();
            [e,m] = measurements();
        case 2 % Heat-bath algorithm
            heat_naive();
            [e,m] = measurements();
    end
    E(isweep) = -e;
    M(isweep) = m/N;
    %visualizer(S,L,Q);
end
```

```
function [nn] = nnfinder_hel(L)
% NNFINDER(L) finds the lattice locations of the nearest neighbors for
% each node of the lattice
%
% L  - the length of the lattice side
%
% nn - each row of nn contains the location of the four nearest
%      neighbors of the node indicated by the row index
%      column 1 -> right
%      column 2 -> left
%      column 3 -> up
%      column 4 -> down
%
% helicoid boundary conditions assumed

N   = L*L;
XNN = 1;
YNN = L;
nn  = zeros(N,4);

for i=1:N,
    nright = i+XNN;
    if nright > N,
        nright = nright - N;
    end
    nleft = i-XNN;
    if nleft <= 0,
        nleft = nleft + N;
    end
    ndown = i+YNN;
    if ndown > N,
        ndown = ndown - N;
    end
    nup = i-YNN;
    if nup <= 0,
        nup = nup + N;
    end
    nn(i,:) = [nright,nleft,nup,ndown];
end
```

```
function [] = met_naive()
% MET_NAIVE() implements a Metropolis algorithm sweep on the lattice.
% It works with global variables as parameters, so no I/O parameters
% is required.
%
% Its operation is based on calculating the energy difference DE
% between the initial lattice state and the candidate new state. If DE
% is negative or equal to zero the new state is accepted. In any other
% case it is accepted with probability equal to e^(-DE*beta).
% Calculating exponentials is a computational expensive task, so they
% are calculated in the beginning of the simulation and stored in the
array PROB.
%
global N Q S PROB XNN YNN


for k = 1:N,
    %% pick a site at random
    i = randi(N,1);
    %% choose a new state different from the initial one
    oldstate = S(i);
    newstate = S(i);
    while newstate == oldstate,
        newstate = randi(Q,1);
    end
    %% calc DE
    nn = i+XNN;
    if nn > N,
        nn = nn-N;
    end;
    Ei = (oldstate == S(nn));
    Ef = (newstate==S(nn));

    nn = i-XNN;
    if nn <= 0,
        nn = nn+N;
    end;
    Ei = Ei+(oldstate == S(nn));
    Ef = Ef+(newstate == S(nn));

    nn  = i+YNN;
    if nn > N,
        nn = nn-N;
    end;
    Ei = Ei+(oldstate == S(nn));
    Ef = Ef+(newstate == S(nn));

    nn = i-YNN;
    if nn <= 0,
        nn = nn+N;
    end;
    Ei = Ei+(oldstate == S(nn));
    Ef = Ef+(newstate == S(nn));

    delta = -(Ef-Ei);

    %% accept new state with probability 1 if DE<=0, and with
    % probability e^(-DE*beta) if DE>0
    if delta <= 0,
        S(i) = newstate;
    else
```

```
        if rand(1) < PROB(delta),
            S(i) = newstate;
        end;
    end;
end % for k
```

```
function [] = wolff_naive()
% WOLFF_NAIVE() implements a Wolff algorithm sweep on the lattice. It
% works with global variables as parameters, so no I/O parameters is.
% required
% It begins by creating a stack to hold the spins that form the
% cluster and adding the seed spin to it while giving it a new value.
% Using a while loop, it goes over the nearest neighbors of the spins
% contained in the stack, adding them to the stack and giving them the
% new spin value with probability 1-e^(-beta) if they are of the same
% value with the seed spin. The exponential appearing here is
% calculated and stored in the variable PADD prior to the sweep
%
global N S Q PADD XNN YNN

stack = zeros(N,1);
%% choose the seed spin for the cluster, put it on the stack and find
%  a new value for it
i = randi(N,1);
stack(1) = i;
sp = 1;
oldspin = S(i);
newspin = S(i);
while newspin == oldspin,
    newspin = randi(Q,1);
end
S(i) = newspin;
ncluster = 1;

%% start loop on spins on the stack:
while sp,
    % Pull a site off the stack
    sp = sp-1;
    current = stack(sp+1);
    % Check the neighboring spins. If it's value is equal to the one
    % of the seed spin, add it to the stack with probability equal to
    % PADD = 1-e^(-beta)
    nn = current+XNN;
    if nn > N,
        nn = nn-N;
    end;

    if S(nn) == oldspin,
        if rand(1) < PADD,
            stack(sp+1) = nn;
            sp = sp+1;
            S(nn) = newspin;
            ncluster = ncluster+1;
        end;
    end;
    nn = current-XNN;
    if nn <= 0,
        nn = nn+N;
    end
    if S(nn) == oldspin,
        if rand(1) < PADD,
            stack(sp+1) = nn;
            sp = sp+1;
            S(nn) = newspin;
            ncluster = ncluster+1;
        end;
    end;
```

```
    nn = current+YNN;
    if nn > N,
        nn = nn-N;
    end
    if S(nn) == oldspin,
        if rand(1) < PADD,
            stack(sp+1) = nn;
            sp = sp+1;
            S(nn) = newspin;
            ncluster = ncluster+1;
        end
    end
    nn = current - YNN;
    if nn <= 0,
        nn = nn+N;
    end
    if S(nn) == oldspin;
        if rand(1) < PADD,
            stack(sp+1) = nn;
            sp = sp+1;
            S(nn) = newspin;
            ncluster = ncluster+1;
        end
    end
end % while sp
```

```
function [] = heat_naive()
% HEAT_NAIVE() implements a Heat-bath algorithm sweep on the lattice.
% It works with global variables as parameters, so no I/O parameters
% is required.
%
% The new state of the lattice in this implementation is chosen
% according to its Boltzmann weight. Firstly we check the states which
% result to a lower-laying energy state as they are more probable, and
% then continuing with the remaining ones. All the exponentials are
% calculated prior to the sweep and are stored in the PROBS array
%
% Here, the lattice positions of the nearest neighbors for all the
% nodes of the lattice are taken from a Nx4 matrix created in the
% beginning of the simulation run
%
global N Q S NN PROB ALLSTATES

for ii=1:N,
    current = randi(N,1); % Pick a spin at random

    nns = S(NN(current,:)); % Get the spin values of the nn of that
                            % spin

    %% calculate the number of different nn states
    %nn - 1:right, 2:left, 3:up, 4:down
    z12 = nns(1)==nns(2);
    z34 = nns(3)==nns(4);
    z13 = nns(1)==nns(3);
    z23 = nns(3)==nns(2);
    z24 = nns(2)==nns(4);
    z14 = nns(4)==nns(1);

    nnz = ones(1,4);
    nnz = nnz +
[(z12+z13+z14),(z12+z23+z24),(z34+z13+z23),(z34+z24+z14)];

    % The number of different states found in the nn
    states = ((nnz(3)+nnz(4)).*nnz(1).*nnz(2) + ...

(nnz(1)+nnz(2)).*nnz(3).*nnz(4))./(nnz(1).*nnz(2).*nnz(3).*nnz(4));

    %% calculate the weights for moves which set the spin equal to one
    %  of its nearest neighbors
    nnb = [PROB(nnz(1))./nnz(1) , PROB(nnz(2))./nnz(2) , ...
        PROB(nnz(3))./nnz(3) , PROB(nnz(4))./nnz(4)];

    %% choose the new state for the spin using the heat-bath algorithm
    summ = sum(nnb);
    normconst = (summ+double(Q)-states);
    randno = rand(1);

    %% find which state this corresponds to
    %% check first to see of it's one of the states which is aligned
    %  with a nn spin
    testval = normconst.*randno;

    weight = nnb(1);
    if testval<weight,
        S(current) = nns(1);
    else
        weight = weight + nnb(2);
```

```
        if testval<weight,
            S(current) = nns(2);
        else
            weight = weight + nnb(3);
            if testval<weight,
                S(current) = nns(3);
            else
                weight = weight + nnb(4);
                if testval<weight,
                    S(current) = nns(4);
                else
                    %% if not choose a new value from the non-aligned
ones
                    %  with equal probability
                    avalstates =
nonzeros(nonzeros(nonzeros(nonzeros( ...
                        ALLSTATES-nns(1))+nns(1)-nns(2))+nns(2)-
nns(3)) ...
                        +nns(3)-nns(4))+nns(4);
                    % here I am renormalizing the remainder of the
                    % probability to be used in the following spin
                    % state selection process
                    % a new random number could have been used, but I
                    % do this to remain thorough
                    newstate = ceil(((1-randno)/(1-
weight/normconst))* ...
                        (double(Q)-states));
                    S(current) = avalstates(newstate);
                end
            end
        end
    end
end
```

```
function [e,m] = measurements()
% MEASUREMENT() Calculates the energy and magnetism of the current
% instance of our lattice. It uses the global variables as parameters,
% so no input parameters needed
%
global S Q N XNN YNN
%% Energy
e = 0;
for po = 1:N, % only forward nn necessary
    nn = po+XNN;
    if nn > N,
        nn = nn-N;
    end;
    e = e+(S(po) == S(nn));
    nn = po+YNN;
    if nn > N,
        nn = nn-N;
    end;
    e = e+(S(po) == S(nn));
end

%% Magnetism
[~,Nmax]=mode(single(S));
m=Nmax-floor((N-Nmax)/(Q-1));
```

```
function [] = visualizer(s,L,q)
% VISUALIZER(s,L,q) creates a JPEG image file with a visualization of
% the 2-D Potts model lattice instance provided in the input
%
% s - 2-D Potts model lattice instance
% L - the lattice side length
% q - the number of different spin states (with the default
%     colormap it can support up to 64 different spin states)
%
global IMGCOUNT
clrmap = length(get(gcf,'Colormap')); % default = 64
lat = figure;
image(reshape(s,L,L)*clrmap/q,'CDataMapping','direct');
axis off
axis image
print(lat,'-r600','-djpeg',int2str(IMGCOUNT));
IMGCOUNT = IMGCOUNT + 1;
```

Α.2 Δοκιμή 2

```
function [E,M] = potts_mat(l,beta,q,nsweep,start,algorithm)
% POTTS_MAT(l,beta,q,nsweep,start,algoritm) simulates a magnet
% according to the 2-D Potts model, optimized for MATLAB
%
% l         - side length of  our square lattice
% beta      - inverse temperature
% q         - number of different spin states
% nsweep    - number of iterations
% start     - 0 for cold start
%             1 for hot start
%             2 for old configuration. Load the old conf in the global
%               variable PRESET
% algorithm - 0 for Metropolis algorithm
%             1 for Wolff algorithm
%             2 for Heat-bath algorithm
%
%% Initialize variables
global L Q N XNN YNN PROB PADD S ALLSTATES NN N1 N2 IMGCOUNT PRESET
Q = int32(q);
L = l;
N = L*L;
XNN = 1;
YNN = L;
IMGCOUNT = 1;

ALLSTATES = 1:Q;
switch start
    case 0 %cold start
        S = int32(ones(N,1));
    case 1 %hot start
        S = int32(randi(q,N,1));
    case 2 % preset conf
        S = PRESET;
end
switch algorithm
    case 0
        % Here i have extended the PROB array so it can cope with
        % negative delta values
        PROB = zeros(9,1);
        PROB(1:5) = 2;
        PROB(6:9) = exp(-((1:4)*beta));
    case 1
        PADD = 1-exp(-beta);
        S = S'; % It is easier to work with a row array
    case 2
        PROB = exp(((1:4)*beta))';
end

NN = nnfinder_hel(L);
[N1,N2] = splitter(L);

E = zeros(nsweep,1);
M = zeros(nsweep,1);

%% Simulation and measurements
for isweep = 1:nsweep,
    switch algorithm
        case 0 % Metropolis algorithm
```

```
            met_mat(N1);
            met_mat(N2);
            [e,m] = measurements();
        case 1 % Wolff algorithm
            wolff_mat();
            [e,m] = measurements_wolff();
        case 2 % Heat-bath algorithm
            heat_mat(N1);
            heat_mat(N2);
            [e,m] = measurements();
    end
    E(isweep) = -e;
    M(isweep) = m/N;
    %visualizer(S,L,Q);
end
```

```
function [nn] = nnfinder_per(L)
% NNFINDER_PER(L) finds the lattice locations of the nearest neighbors
% for each node of the lattice
%
% L  - the length of the lattice side
%
% nn - each row of nn contains the location of the four nearest
%      neighbors of the node indicated by the row index
%      column 1 -> right
%      column 2 -> left
%      column 3 -> up
%      column 4 -> down
%
% periodic boundary conditions assumed

N   = L*L;
XNN = 1;
YNN = L;
nn  = zeros(N,4);

for i=1:N,
    if mod(i,L)==0,
        nright = i+XNN-L;
    else
        nright = i+XNN;
    end

    if mod((i-1),L)==0,
        nleft = i-XNN+L;
    else
        nleft = i-XNN;
    end

    nup = i-YNN;
    if nup<=0,
        nup = nup+N;
    end

    ndown = i+YNN;
    if ndown>N;
        ndown = ndown-N;
    end

    nn(i,:) = [nright,nleft,nup,ndown];
end
```

```
function [n1,n2] = splitter(L)
% SPLITTER(L) Splits our lattice into two sets of nodes, in such way
% that the nearest neighbors for the nodes in one of the sets are all
% contained in the other one. Periotic boundary conditions assumed
%
% n1 contains the lattice locations of the first set
% n2 contains the lattice locations of the second set
%
% L is the length of the lattice side
% L must be even for this function to work

%% n1
n1 = zeros(L/2,1);
for i=1:L,
    if mod(i,2)~=0,
        n1((1:L/2)+(i-1)*(L/2)) = (1:2:(L-1))+(i-1)*L;
    else
        n1((1:L/2)+(i-1)*(L/2)) = (2:2:L)+(i-1)*L;
    end
end

%% n2
n2 = zeros(L/2,1);
for i=1:L,
    if mod(i,2)==0,
        n2((1:L/2)+(i-1)*(L/2)) = (1:2:(L-1))+(i-1)*L;
    else
        n2((1:L/2)+(i-1)*(L/2)) = (2:2:L)+(i-1)*L;
    end
end
```

```
function [] = met_mat(n)
% MET_MAT() implements a Metropolis algorithm sweep on the lattice.
% What is different in this implementation is the way it accesses the
% lattice nodes If our square lattice has an even side length, it can
% be divided in to 2 separate groups with one containing the nearest
% neighbors of the other. And because each change of spin value only
% depends on the spin value of the nearest neighbors, we can change
% the spins of either of the groups all "at once" (what is actually
% happening here is using matrix notation for the commands, which
% works much better in the MATLAB environment furthermore it is easier
% to parallelize the execution on multiple processing units). Compared
% to the naive implementation of the same algorithm, we have more
% actions taking place per run. But the gain in processing time is
% still considerable.
%
global Q N PROB S ALLSTATES NN

%% Recover the spin values of the current state of the lattice
oldstate  = S(n);
newstate  = zeros(N/2,1);

%% Find new spin values, different from the initial ones
for k = 1:(N/2),
    avalstates = find(ALLSTATES-oldstate(k));
    newstate(k) = avalstates(randi(Q-1,1,1));
end

%% Calculate the energy difference locally resulting after a change in
%  the spin values from the initial state to the candidate new one
Ei = sum((S(NN(n,:))==[oldstate,oldstate,oldstate,oldstate]),2);
Ef = sum((S(NN(n,:))==[newstate,newstate,newstate,newstate]),2);
delta = -(Ef-Ei);

%% Create a logical array containing information concerning the
%  acceptance of the candidate new spin states in accordance with the
%  Metropolis algorithm
accept = (delta<=0) | (rand(N/2,1)<PROB(delta+5));

%% Pass the new spin values to the latice
nn1 = nonzeros(n.*accept);
s1 = nonzeros(newstate.*accept);
S(nn1)=s1;
```

```
function [e,m] = wolff_mat()
% WOLFF_MAT() implements a Wolff algorithm sweep on the lattice. It
% works with global variables as parameters, so no I/O parameters
% needed.
% Here I used matrix notation to increase the efficiency of the
% implementation. Although the gain in processing time isn't great,
% the code is can profit from some form of parallelization. The stack
% structure was removed and in its place we use a variable size array
% to store the "under-consideration" spins which we process at the
% same time.
%
global S N Q ALLSTATES NN PADD
%% Choose a spin at random from the lattice to serve as the seed spin
current = randi(N,1);


%% Find a diffent new spin state
oldstate = S(current);
avalspins = find(ALLSTATES-oldstate);
newstate = avalspins(randi(Q-1,1));
%% Evolve the cluster, while changing the spins to the new value
%  the current variable holds the location(s) of the spin(s) to be
%  considered for addition in the cluster
S(current) = newstate;
while size(current,1)>=1,
    temp = nonzeros((S(NN(current,:))==oldstate).* ...
        (rand(size(current,1),4)<PADD).*NN(current,:));
    S(temp) = newstate;
    current = temp;
end
```

```
function [] = heat_mat(n)
% HEAT_MAT() implements a Heat-bath algorithm sweep on the lattice.
% What is different in this implementation is the way it accesses the
% lattice nodes. If our square lattice has an even side length, it can
% be divided in to 2separate groups with one containing the nearest
% neighbors of the other. And because each change of spin value only
% depends on the spin value of the nearest neighbors, we can change
% the spins of either of the groups all "at once" (what is actually
% happening here is using matrix notation for the commands, which
% works much better in the MATLAB environment furthermore it is easier
% to parallelize the execution on multiple processing units). Compared
% to the naive implementation of the same algorithm, we have more
% actions taking place per run. But the gain in processing time is
% still considerable.
%
global N Q S NN PROB ALLSTATES
%% find the spins of the nn of the current node group
nns = S(NN(n,:));

%% calculate the number of different nn states
%nn - 1:right, 2:left, 3:up, 4:down
z12 = nns(:,1)==nns(:,2);
z34 = nns(:,3)==nns(:,4);
z13 = nns(:,1)==nns(:,3);
z23 = nns(:,3)==nns(:,2);
z24 = nns(:,2)==nns(:,4);
z14 = nns(:,4)==nns(:,1);

nnz = ones(N/2,4);
nnz = nnz + [(z12+z13+z14),(z12+z23+z24),(z34+z13+z23),(z34+z24+z14)];

% The number of different states found in the nn
states = ((nnz(:,3)+nnz(:,4)).*nnz(:,1).*nnz(:,2) +
(nnz(:,1)+nnz(:,2)) ...
    .*nnz(:,3).*nnz(:,4))./(nnz(:,1).*nnz(:,2).*nnz(:,3).*nnz(:,4));

%% calculate the weights for moves which set the spin equal to one of
%  its nearest neighbors
nnb = [PROB(nnz(:,1))./nnz(:,1) , PROB(nnz(:,2))./nnz(:,2) , ...
    PROB(nnz(:,3))./nnz(:,3) , PROB(nnz(:,4))./nnz(:,4)];

%% choose the new state for the spin using the heat-bath algorithm
summ = sum(nnb,2);
rn = (summ+double(Q)-states);
randno = rand(N/2,1);

%% find which state this corresponds to
%% check first to see of it's one of the states which is aligned with
%  a nn spin
testval = rn.*randno;

weightsval = nnb(:,1);
tright  = testval<weightsval;
ntright = ~tright;
logicarr = nonzeros(n.*tright);
S(logicarr) = S(NN(logicarr,1));

weightsval = weightsval + nnb(:,2);
tleft   = testval<weightsval;
ntleft  = ntright .* ~tleft;
logicarr = nonzeros(n .* tleft .* ntright);
```

```
S(logicarr) = S(NN(logicarr,2));

weightsval = weightsval + nnb(:,3);
tup      = testval<weightsval;
ntup     = ntleft .* ~tup;
logicarr = nonzeros(n .* tup .* ntleft);
S(logicarr) = S(NN(logicarr,3));

weightsval = weightsval + nnb(:,4);
tdown  = testval<weightsval;
ntdown = ntup .* ~tdown;
logicarr = nonzeros(n .* tdown .* ntup);
S(logicarr) = S(NN(logicarr,4));

%% if not choose a new value from the non-aligned ones
logicarr = find(ntdown);

% here I am requesting the needed variables only for the spins that
% didn't change during the first leg of the algorithm
nnsremain = nns(logicarr,:);
rnremain  = rn(logicarr);
weightsremain = weightsval(logicarr);
randremain = randno(logicarr);
statesremain = states(logicarr);
nremain = n(logicarr);

% here I am renormalizing the remainder of the probability to be used
% in the following spin state selection process
% a new random number could have been used, but I do this to remain
% thorough
bb = ceil(((1-randremain)./(1-weightsremain./rnremain)).* ...
    (double(Q)-statesremain));

%% apply the new values to the lattice
for ii=1:length(bb),
    qs = nonzeros(nonzeros(nonzeros(nonzeros(ALLSTATES-
nnsremain(ii,1)) ...
        +nnsremain(ii,1)-nnsremain(ii,2))+nnsremain(ii,2) ...
        -nnsremain(ii,3))+nnsremain(ii,3)-
nnsremain(ii,4))+nnsremain(ii,4);
    S(nremain(ii)) = qs(bb(ii));
end
```

```
function [e,m] = measurements()
% MEASUREMENT() Calculates the energy and magnetism of the current
% instance of our lattice. It uses the global variables as parameters,
% so no input parameters needed
%
global S Q N NN N1
%% Energy
%  only one of the 2 sets of nodes has to be accounted for
e = sum(sum((S(NN(N1,:))==[S(N1),S(N1),S(N1),S(N1)]),2));

%% Magnetism
[~,Nmax]=mode(single(S));
m=Nmax-floor((N-Nmax)/(Q-1));
```

```
function [e,m] = measurements_wolff()
% MEASUREMENT() Calculates the energy and magnetism of the current
% instance of our lattice. It uses the global variables as parameters,
% so no input parameters needed
%
global S Q N NN N1
%% Energy
%  only one of the 2 sets of nodes has to be accounted for
e = sum(sum((S(NN(N1,:))==[S(N1)',S(N1)',S(N1)',S(N1)']),2));

%% Magnetism
[~,Nmax]=mode(single(S));
m=Nmax-floor((N-Nmax)/(Q-1));
```

```
function [] = visualizer(s,L,q)
% VISUALIZER(s,L,q) creates a JPEG image file with a visualization of
% the 2-D Potts model lattice instance provided in the input
%
% s - 2-D Potts model lattice instance
% L - the lattice side length
% q - the number of different spin states (with the default
%     colormap it can support up to 64 different spin states)
%
global IMGCOUNT
clrmap = length(get(gcf,'Colormap')); % default = 64
lat = figure;
image(reshape(s,L,L)*clrmap/q,'CDataMapping','direct');
axis off
axis image
print(lat,'-r600','-djpeg',int2str(IMGCOUNT));
IMGCOUNT = IMGCOUNT + 1;
```

```
function  result = jack(indata,binno)
% JACK(INDATA,BINNO) calculates the expectation value and the standard
% deviation of a given quantity together with their statistical
% errors, using the Jackknife method
%
% INDATA - a set of values of the quantity under consideration, from
%          which the expected value and the standard deviation will be
%          extracted
% BINNO  - the number of bins to be used by the method
%
%
ndat=size(indata,1);
O=zeros(binno,1);
Chi=zeros(binno,1);

binw=floor(ndat/binno);
xdat=binno*binw;
X=reshape(indata(ndat-xdat+1:ndat),binw,binno);
div=xdat-binw;

% Compute averages
for i=1:binno,
%    O(i)=(sum(X)-X(i))/div;
    temp=[X(1:binw,1:i-1),X(1:binw,i+1:end)];
    O(i)=sum(sum(temp))/div;
    Chi(i)=sum(sum((temp-O(i)).^2))/div;
end
avO=mean(O);
avChi=mean(Chi);

% Compute errors
erO=sqrt(sum((O-avO).^2));
erChi=sqrt(sum((Chi-avChi).^2));

result = [avO,erO,avChi,erChi];
```

Α.3 Δοκιμή 3

Το μεγαλύτερο μέρος του κώδικα που χρησιμοποιείται κατά την τρίτη δοκιμή είναι πανομοιότυπος με αυτόν που χρησιμοποιείται και κατά την δεύτερη. Παρακάτω παραθέτω τις μεθόδους που έχουν διαφοροποιηθεί.

```
function [] = met_mat(n)
% MET_MAT() implements a Metropolis algorithm sweep on the lattice.
% What is different in this implementation is the way it accesses the
% lattice nodes If our square lattice has an even side length, it can
% be divided in to 2 separate groups with one containing the nearest
% neighbors of the other. And because each change of spin value only
% depends on the spin value of the nearest neighbors, we can change
% the spins of either of the groups all "at once" (what is actually
% happening here is using matrix notation for the commands, which
% works much better in the MATLAB environment furthermore it is easier
% to parallelize the execution on multiple processing units). Compared
% to the naive implementation of the same algorithm, we have more
% actions taking place per run. But the gain in processing time is
% still considerable.
%
global Q N PROB S NN

%% Recover the spin values of the current state of the lattice
oldstate  = S(n);
newstate  = randi(Q,N/2,1);

%% Find new spin values, different from the initial ones
newstate = abs(newstate-((oldstate==newstate)*double(Q+1)));

%% Calculate the energy difference locally resulting after a change in
%  the spin values from the initial state to the candidate new one
Ei = sum((S(NN(n,:))==[oldstate,oldstate,oldstate,oldstate]),2);
Ef = sum((S(NN(n,:))==[newstate,newstate,newstate,newstate]),2);
delta = -(Ef-Ei);

%% Create a logical array containing information concerning the
%  acceptance of the candidate new spin states in accordance with the
%  Metropolis algorithm
accept = (delta<=0) | (rand(N/2,1)<PROB(delta+5));

%% Pass the new spin values to the lattice
nn1 = nonzeros(n.*accept);
s1 = nonzeros(newstate.*accept);
S(nn1)=s1;
```

```
function [] = heat_mat(n)
% HEAT_MAT() implements a Heat-bath algorithm sweep on the lattice.
% What is different in this implementation is the way it accesses the
% lattice nodes. If our square lattice has an even side length, it can
% be divided in to 2 separate groups with one containing the nearest
% neighbors of the other. And because each change of spin value only
% depends on the spin value of the nearest neighbors, we can change
% the spins of either of the groups all "at once" (what is actually
% happening here is using matrix notation for the commands, which
% works much better in the MATLAB environment furthermore it is easier
% to parallelize the execution on multiple processing units). Compared
% to the naive implementation of the same algorithm, we have more
% actions taking place per run. But the gain in processing time is
% still considerable.
%
global N Q S NN PROB
%% find the spins of the nn of the current node group
nns = S(NN(n,:)); % the spins of the nn of the current node group

%% calculate the number of different nn states
%nn - 1:right, 2:left, 3:up, 4:down
z12 = nns(:,1)==nns(:,2);
z34 = nns(:,3)==nns(:,4);
z13 = nns(:,1)==nns(:,3);
z23 = nns(:,3)==nns(:,2);
z24 = nns(:,2)==nns(:,4);
z14 = nns(:,4)==nns(:,1);

nnz = ones(N/2,4);
nnz = nnz + [(z12+z13+z14),(z12+z23+z24),(z34+z13+z23),(z34+z24+z14)];

% The number of different states found in the nn
states = ((nnz(:,3)+nnz(:,4)).*nnz(:,1).*nnz(:,2) +
(nnz(:,1)+nnz(:,2)) ...
    .*nnz(:,3).*nnz(:,4))./(nnz(:,1).*nnz(:,2).*nnz(:,3).*nnz(:,4));

%% calculate the weights for moves which set the spin equal to one of
%  its nearest neighbors
nnb = [PROB(nnz(:,1))./nnz(:,1) , PROB(nnz(:,2))./nnz(:,2) , ...
    PROB(nnz(:,3))./nnz(:,3) , PROB(nnz(:,4))./nnz(:,4)];

%% choose the new state for the spin using the heat-bath algorithm
summ = sum(nnb,2);
rn = (summ+double(Q)-states);
randno = rand(N/2,1);

%% find which state this corresponds to
%% check first to see of it's one of the states which is aligned with
%  a nn spin
testval = rn.*randno;

weightsval = nnb(:,1);
tright  = testval<weightsval;
ntright = ~tright;
logicarr = nonzeros(n.*tright);
S(logicarr) = S(NN(logicarr,1));

weightsval = weightsval + nnb(:,2);
tleft   = testval<weightsval;
ntleft  = ntright .* ~tleft;
logicarr = nonzeros(n .* tleft .* ntright);
```

```
S(logicarr) = S(NN(logicarr,2));

weightsval = weightsval + nnb(:,3);
tup      = testval<weightsval;
ntup     = ntleft .* ~tup;
logicarr = nonzeros(n .* tup .* ntleft);
S(logicarr) = S(NN(logicarr,3));

weightsval = weightsval + nnb(:,4);
tdown  = testval<weightsval;
ntdown = ntup .* ~tdown;
logicarr = nonzeros(n .* tdown .* ntup);
S(logicarr) = S(NN(logicarr,4));

%% if not choose a new value from the non-aligned ones
logicarr = find(ntdown);

% here I am requesting the needed variables only for the spins that
% didn't change during the first leg of the algorithm
nnsremain = nns(logicarr,:);
rnremain  = rn(logicarr);
weightsremain = weightsval(logicarr);
randremain = randno(logicarr);
statesremain = states(logicarr);
nremain = n(logicarr);

% here I am renormalizing the remainder of the probability to be used
% in the following spin state selection process
% a new random number could have been used, but I do this to maintain
% thoroughness
bb = ceil(((1-randremain)./(1-weightsremain./rnremain)).* ...
    (double(Q)-statesremain));

%% apply the new values to the lattice
avalstates = repmat(1:Q,length(bb),1);
avalarray = logical((repmat(nnsremain(:,1),1,Q)==avalstates)+ ...
                    (repmat(nnsremain(:,2),1,Q)==avalstates)+ ...
                    (repmat(nnsremain(:,3),1,Q)==avalstates)+ ...
                    (repmat(nnsremain(:,4),1,Q)==avalstates));

avalstates(avalarray)=0;
avalstates = sort(avalstates,2,'descend');
indices_array = sub2ind(size(avalstates), 1:length(bb), bb')';

S(nremain) = avalstates(indices_array);
```

Βιβλιογραφία

[1] Gibbs, J. W. 1902 *Elementary Principles in Statistical Mechanics.* επανέκδοση 1981, Ox. Bow Press, Woodridge

[2] Newman, M. E. J. και Barkema, G. T. 1999 *Monte Carlo Methods in Statistical Physics.* επανέκδοση 2002, Oxford University Press, New York

Αναγνωστόπουλος Κ. Ν. 2010 *Σημειώσεις Υπολογιστικής Φυσικής ΙΙ*