UNIVERSIDADE DE BRASÍLIA
INSTITUTO DE FÍSICA
PROGRAMA DE PÓS-GRADUAÇÃO EM FÍSICA

Estratégias computacionais no estudo do caos em sistemas hamiltonianos

Moises Fabiano Pereira da Silva Júnior

Tese de Doutorado

Brasília
Julho de 2020

Moises Fabiano Pereira da Silva Júnior

Estratégias computacionais no estudo do caos em sistemas hamiltonianos

Tese de Doutorado apresentada ao Instituto de Física da Universidade de Brasília, para obtenção do Título de Doutor em Física Teórica.
Orientador: Prof. Dr. Tarcísio Marciano da Rocha Filho

Brasília
2020

Educação não transforma o mundo.
Educação muda pessoas.
Pessoas transformam o mundo.

- Paulo Freire

à memória de minha mãe.

Agradecimentos

Agradeço a minha esposa Deborah, por estar sempre ao meu lado tornando mais fáceis os momentos difíceis e mais alegres os momentos felizes. Obrigado pelo amor, paciência e companheirismo.

Agradeço ao meu professor e orientador Tarcísio Marciano, pela confiança e paciência. Por estar sempre disposto a dedicar seu tempo às minhas dúvidas e questionamentos.

Aos meus pais, avós e familiares por terem ajudado a cuidar de mim. Em especial a minha mãe, pelo amor, carinho e por ter incentivado meus estudos desde cedo.

Aos meus amigos, pelos anos de carinho, incentivo e torcida.

A todos os meus professores, por todo o conhecimento e lições ensinadas a mim.

Ao CNPq, pelo apoio financeiro.

Resumo

No presente trabalho são apresentadas estimativas semi-analíticas e numéricas para o maior expoente de Lyapunov de um sistema de muitas partículas com interações de longo alcance, estendendo resultados anteriores do modelo da Hamiltoniana de campo médio com potencial cosseno. Os resultados evidenciam um expoente crítico associado ao decaimento com lei de potência para o maior expoente de Lyapunov para a transições de fase de segunda ordem, próximo do valor do modelo cosseno da Hamiltoniana de campo médio sugerindo a possível universalidade desse expoente. Também mostra que o expoente crítico para transições de fase de primeira ordem tem um valor diferente de estimativas teóricas e numéricas.

Palavras-chave: Expoente de Lyapunov, Longo Alcance, Transição de Fase

Abstract

In this work are presented semi-analytic and numerical estimates for the largest Lyapunov exponent in a many-particle system with long-range interactions, extending previous results for the Hamiltonian Mean Field model with a cosine potential. The results evidence a critical exponent associated to a power law decay of the largest Lyapunov exponent close to second-order phase transitions, close to the same value as for the cosine Hamiltonian Mean Field model, suggesting the possible universality of this exponent. It also show that the exponent for first-order phase transitions has a different value from both theoretical and numerical estimates.

Keywords: Lyapunov exponent, Long-Range, Phase transtions

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

S	Entropia
E	Energia
F	Energia Livre de Helmholtz
W	Número de microestados
Z	Função de Partição Canônica
N	Número de partículas
m	Magnetização
T	Temperatura
c	Calor Específico
H	Hamiltoniana de um sistema
$\boldsymbol{p}$	Momentos generalizados
$\boldsymbol{q}$	Posições generalizadas
K	Energia cinética
U	Energia potencial
λ	Maior Expoente de Lyapunov
K_R	Curvatura de Ricci
k_R	Curvatura média de Ricci
ξ	Expoente crítico associado ao maior expoente de Lyapunov

Lista de Abreviaturas

LLE	Maior Expoente de Lyapunov (*Largest Lyapunov Exponent*)
$cosHMF$	Hamiltoniana de campo médio cosseno (*cossine Hamiltonian Mean Field*)
$GHMF$	Hamiltoniana de campo médio generalizada (*Generalized Hamiltonian Mean Field*)
MMC	Monte Carlo Microcanônico (*Microcanonical Monte Carlo*)
ML	Aprendizado de Máquina (*Machine Learning*)
ANN	Rede Neural Artificial (*Artificial Neural Network*)
KDE	Estimativa de Densidade Kernel (*Kernel Density Estimator*)

Sumário

Introdução

A dinâmica de sistemas clássicos de muitas partículas com interações de longo alcance em um espaço de dimensão D, com potencial que decai a longas distâncias r como r^{-D} [1], possui descrição exata pela equação de Vlasov, onde a prescrição de Kac é usada com o propósito de termos um limite contínuo propriamente definido [2]. Neste limite, as partículas interagem apenas através de seus campos médios [3, 4, 5, 6, 7], e o sistema nunca atinge o equilíbrio termodinâmico, geralmente se estabelecendo em um regime de estado estacionário não-gaussiano [4]. Neste estado estacionário, os sistemas efetivamente tornam-se desacoplados aos pares, como partículas evoluindo através de um potencial estático. Isso implica que modelos unidimensionais com interações de longo alcance são integráveis e, consequentemente, não caóticos, neste limite. Por outro lado, uma situação mais complexa surge quando temos um número finito de partículas, onde a contribuição das colisões [8, 9, 10, 11] tornam-se relevantes para a dinâmica, corrigindo a simplicidade do campo médio e geralmente levando ao caos. Essas correções colisionais também são responsáveis por direcionar o sistema para o equilíbrio termodinâmico, embora com tempos de relaxação muito longos [12, 13].

Mostrar que a dinâmica de um sistema é caótica equivale a mostrar que seu maior expoente de Lyapunov (LLE, do inglês *largest Lyapunov exponent*) é positivo [14], o que tem sido realizado com sucesso em sistemas com interações de longo alcance [15, 16, 17, 18, 19]. Uma abordagem geométrica baseada em médias estatísticas da dinâmica microscópica foi desenvolvida por Casetti e colaboradores [20, 21, 22, 23]. Firpo [24] utilizou desta abordagem para mostrar que, para o modelo da Hamiltoniana de Campo Médio Cosseno (cosHMF, do inglês *cosine Hamiltonian Mean-Field*) [25], o LLE escala como $\lambda \propto |e - e_c|^{1/6}$ na transição de fase de segunda ordem, com e sendo a energia por partícula do sistema e e_c seu valor crítico. Esse resultado foi corroborado na Ref. [18] por simulações de dinâmica molecular, apesar dos valores obtidos numericamente para o LLE divergirem da previsão teórica em [24].

Neste trabalho, essa análise é extendida para a Hamiltoniana de Campo Médio Generalizada (GHMF, do inglês *Generalized Hamiltonian Mean Field*) [26], que possui um diagrama de fase mais rico do que o modelo cosHMF, com diferentes transições de segunda ordem além de transições de primeira ordem. Isso permite verificar se o expoente crítico do LLE depende da natureza da transição de fase ou do modelo estudado.

O trabalho está organizado da seguinte maneira: o Capítulo 1 é reservado para os sistemas com interações de longo alcance, o que inclui definições, propriedades dinâmicas e termodinâmicas, além do modelo que será objeto de estudo, o GHMF. No Capítulo 2 o conceito de caos é apresentado junto ao expoente de Lyapunov e seus métodos de obtenção, tanto computacionais quanto analíticos. Toda a metodologia aplicada e desenvolvida está no Capítulo 3, desde a obtenção semi-analítica do LLE com auxílio de Redes Neurais Artificiais (ANN, do inglês *Artificial Neural Network*) até as simulações de dinâmica molecular utilizando paralelização em GPU e técnicas de estimativa de densidade por Aprendizado de Máquina (ML, do inglês *Machine Learning*). Por fim, no Capítulo 4 temos os resultado teóricos e numéricos, com discussões sobre os expoentes críticos do LLE para transições de primeira e segunda ordem, seguido pelas conclusões e perspectivas.

Capítulo 1

Sistemas com Interações de Longo Alcance

Neste capítulo, é apresentada uma definição de sistemas com interações de longo alcance e suas principais propriedades dinâmicas e termodinâmicas. Ao final do capítulo, temos o cálculo analítico da entropia do modelo GHMF, que nos leva às suas características termodinâmicas, como seu espaço de fases.

1.1 Definição

Uma forma intuitiva de definirmos o que é uma interação de longo alcance, é através do cálculo da energia potencial u de uma partícula localizada no centro de uma esfera de raio R, onde todas as outras partículas estão homogeneamente distribuídas Figura 1.1 [27]. A longas distâncias, a interação de pares das partículas é dada por um potencial do tipo

$$V(r) = \frac{J}{r^{\alpha}}, \tag{1.1}$$

com r sendo a distância entre as partículas, J e α constantes. Uma vez que o interesse é a natureza de longo alcance do potencial, podemos excluir as contribuições para a energia u vinda de partículas localizadas a curtas distâncias $r < \delta$, além de evitar a divergência a curtas distâncias. Temos que, em d dimensões

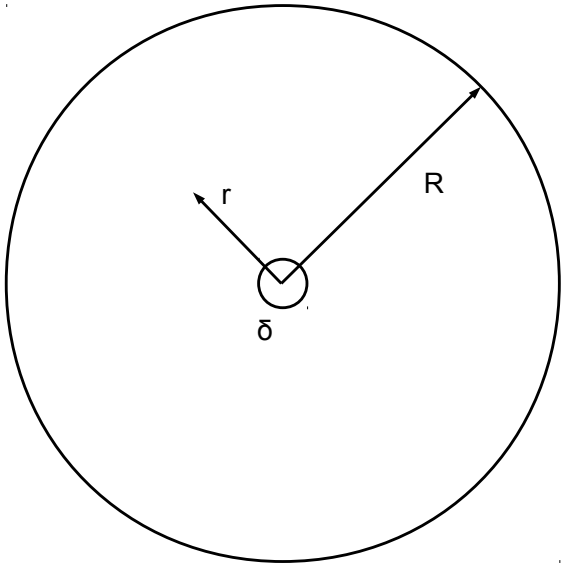


Figura 1.1: Partícula localizada no centro de uma esfera com distribuição homogênea de partículas. O raio da esfera exterior é R e da esfera interior é δ.

$$\begin{aligned} u &= \int_{\delta}^{R} \rho \frac{J}{r^{\alpha}} d^d r \\ &= \rho J \Omega_d \int_{\delta}^{R} r^{d-\alpha-1} dr \\ &= \frac{\rho J \Omega_d}{d-\alpha} \, r^{d-\alpha} \Big|_{\delta}^{R} \end{aligned}$$

$$\therefore \; u = \frac{\rho J \Omega_d}{d-\alpha} \left[R^{d-\alpha} - \delta^{d-\alpha} \right], \quad se \; \alpha \neq d, \tag{1.2}$$

onde ρ é uma densidade qualquer e Ω_d é o volume angular em d dimensões. Quando $\alpha = d$, temos

$$\begin{aligned} u &= \int_{\delta}^{R} \rho \frac{J}{r^{\alpha}} d^d r \\ &= \rho J \Omega_d \int_{\delta}^{R} r^{-1} dr \\ &= \rho J \Omega_d \ln r \Big|_{\delta}^{R} \end{aligned}$$

$$\therefore \; u = \rho J \Omega_d \ln \left(\frac{R}{\delta} \right), \quad se \; \alpha = d. \tag{1.3}$$

Com isso, podemos escrever

$$u = \begin{cases} \dfrac{\rho J \Omega_d}{d-\alpha}\left[R^{d-\alpha} - \delta^{d-\alpha}\right], & \alpha \neq d. \\ \rho J \Omega_d \ln\left(\dfrac{R}{\delta}\right), & \alpha = d. \end{cases} \tag{1.4}$$

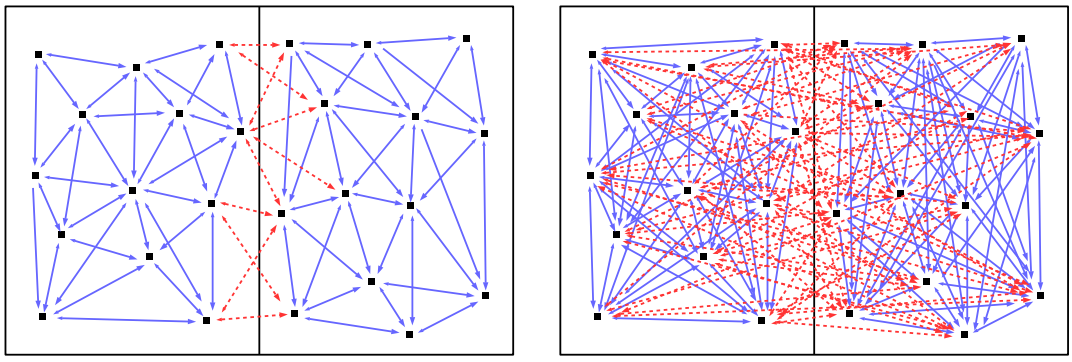
De (1.4), se $\alpha > d$ então a energia potencial é finita quando $R \to \infty$, neste caso a energia cresce linearmente com o tamanho do sistema e dizemos que a interação é de curto alcance. Se $\alpha \leq d$ a energia potencial diverge quando $R \to \infty$, ou seja, a energia cresce superlinearmente com o tamanho do sistema e a interação é dita ser de longo alcance. Sistemas com interações de longo alcance podem ser encontrados em diversas áreas da física, como astrofísica, física de plasma, hidrodinâmica, física atômica e nuclear. Sistemas desse tipo podem apresentar diversas peculiaridades, como calor específico negativo, susceptibilidade negativa, inequivalência entre ensembles e quebra de ergodicidade.

1.2 Propriedades termodinâmicas

Sistemas com interações de longo alcance são intrinsecamente não-aditivos. Essa é a principal diferença entre sistemas com interações de longo alcance e sistemas com interações de curto alcance. Uma forma intuitiva de entender a aditividade é imaginar um sistema dividido em dois subsistemas, 1 e 2, com energia total

$$E_{total} = E_1 + E_2 + E_{int}, \tag{1.5}$$

com E_1, E_2 e E_{int} sendo a energia do subsistema 1, a energia do subsistema 2 e a energia de interação entre os subsistemas, respectivamente. No caso em que as interações são de curto alcance, existem mais interações entre partículas de um mesmo subsistema do que partículas de subsistemas diferentes (como ilustrado na Figura 1.2(a), onde as linhas sólidas azuis representam interações entre partículas de um mesmo subsistema e as linhas tracejadas vermelhas representam interações entre partículas de subsistemas diferentes). Podemos notar que, quando o número de partículas cresce ($N \to \infty$), o número de linhas sólidas azuis se torna muito maior que o número de linhas tracejadas vermelhas, pois a interação entre os subsistemas fica restrita às partículas que estão próximas a barreira que os separa. Portanto, dizemos que no limite termodinâmico ($N \to \infty$) $E_{int} \ll E_1 + E_2$, ou seja,



(a) Interação de Curto Alcance (b) Interação de Longo Alcance

Figura 1.2: Representação de um sistema dividido em dois subsistemas que interagem entre si através de (a) interações de curto alcance e (b) interações de longo alcance. As linhas sólidas azuis representam interações entre partículas do mesmo subsistema e as linhas tracejadas vermelhas representam interações entre partículas de subsistemas diferentes.

$$E_{total} \simeq E_1 + E_2. \tag{1.6}$$

Como a energia do sistema é dada pela soma das energias de cada subsistema, dizemos que a energia é aditiva. Quando as interações são de longo alcance, o número de linhas tracejadas vermelhas é proporcional ao número de linhas sólidas azuis, Figura 1.2(b), porque a interação entre os subsistemas não fica restrita às partículas próximas à barreira. Neste caso, mesmo no limite termodinâmico, a energia de interação tem a mesma ordem de grandeza das energias dos subsistemas, $E_{int} \sim E_1 + E_2$, ou seja,

$$E_{total} \neq E_1 + E_2. \tag{1.7}$$

Neste caso, como a energia do sistema não é igual a soma das energias dos subsistemas, dizemos que a energia não é aditiva. Quando a energia de um sistema é aditiva, reduzir pela metade o tamanho do sistema implica em reduzir pela metade a energia do sistema, sendo assim um sistema também extensivo, [1]:

$$\text{Aditividade} \Rightarrow \text{Extensividade},$$

logo, também vale que:

$$\text{Não-extensividade} \Rightarrow \text{Não-aditividade}.$$

Sistemas com interação de longo alcance geralmente são não-aditivos e não-extensivos. Entretanto, a extensividade pode ser recuperada através da prescrição de Kac, [2]. A

prescrição de Kac é um artifício matemático de renormalização do potencial do sistema, com a finalidade de tornar a energia potencial proporcional ao tamanho do sistema, assim como é a energia cinética. Porém, mesmo recuperando a extensividade de um sistema, ele continua sendo não-aditivo.

Em sistemas com interações de curto alcance, os ensembles estatísticos são equivalentes. A equivalência entre os ensembles é caracterizada por duas propriedades importantes, [27]:

(i) No limite termodinâmico, as flutuações dos parâmetros termodinâmicos que não estão fixados desaparecem;
(ii) Um macroestado que pode ser realizado em um ensemble também pode ser realizado em outro ensemble.

Por exemplo, no caso da equivalência entre os ensembles microcanônico e canônico: um sistema isolado com uma energia fixa possui uma temperatura média. Se um mesmo sistema for colocado em contato térmico com um reservatório a essa mesma temperatura, então ele terá uma energia média igual a energia do sistema isolado. Portanto, existe uma correspondência um-para-um entre a energia e a temperatura. A vantagem da equivalência entre ensembles é a liberdade de poder escolher o ensemble onde os cálculos são mais fáceis e extrair dele toda a termodinâmica do sistema. Através da relação entre as funções de partição canônica e microcanônica com a energia livre de Helmholtz e a entropia de Boltzmann, temos que [28]

$$\begin{aligned} Z_N(T) &= \exp\left(-\frac{F_N(T)}{T}\right) = \sum_E W_N(E) \exp\left(-\frac{E}{T}\right) = \\ &= \sum_E \exp\left(S_N(E)\right) \exp\left(-\frac{E}{T}\right) \end{aligned}$$

$$\therefore \exp\left(-\frac{F_N(T)}{T}\right) = \sum_E \exp\left[-\frac{1}{T}\left(E - TS_N(E)\right)\right], \tag{1.8}$$

onde a constante de Boltzmann k_B é fixada como sendo um para simplificar as expressões. No equilíbro, o valor do somatório em (1.8) pode ser aproximado pelo maior termo da soma, com isso

$$F_N(T) \simeq min_E \left[E - TS_N(E)\right]. \tag{1.9}$$

Para satisfazer a condição de mínimo em (1.9), devemos ter

$$\frac{\partial}{\partial E}\left[E - TS_N(E)\right] = 0 \rightarrow 1 - T\frac{\partial S_N}{\partial E} = 0 \rightarrow \frac{\partial S_N}{\partial E} = \frac{1}{T}, \tag{1.10}$$

$$\frac{\partial^2}{\partial E^2}\left[E - TS_N(E)\right] > 0 \rightarrow -T\frac{\partial^2 S_N}{\partial E^2} > 0 \rightarrow \frac{\partial^2 S_N}{\partial E^2} < 0. \tag{1.11}$$

Quando as condições (1.10) e (1.11) são satisfeitas, temos

$$F_N(T) \simeq E - TS_N(E), \tag{1.12}$$

que é a transformada de Legendre da que define a energia livre de Helmholtz [29]. Ou seja, para existir equivalência entre os ensembles canônico e microcanônico, a entropia S deve ser uma função côncava da energia E, [1]. Quando a entropia de um sistema isolado possui uma região convexa, como a região $[E_1, E_2]$ da linha sólida da Figura 1.3(a), é possível substituir a região convexa por um "envelope côncavo", [27]. Partindo da energia E_1, a entropia do sistema com energia E_2, pode ser obtida através da integração da temperatura

$$S(E_2) = S(E_1) + \int_{E_1}^{E_2} \beta(E) dE, \tag{1.13}$$

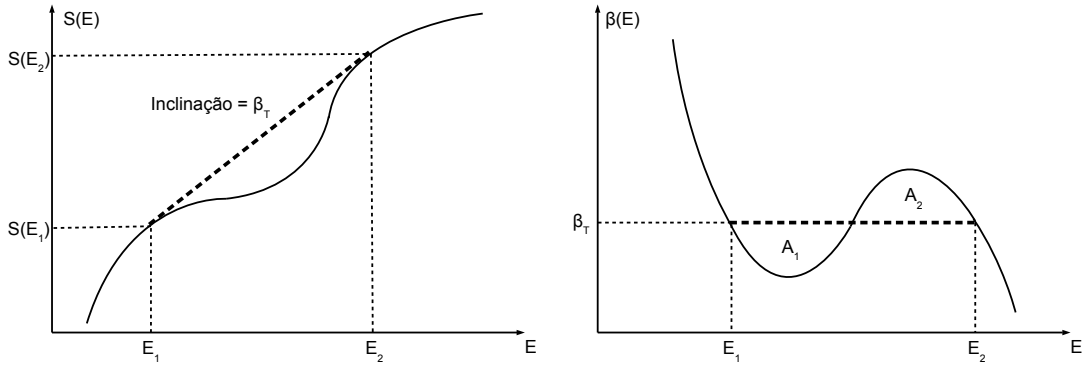
onde $\beta(E)$, representado na Figura 1.3(b), é o inverso da temperatura. A integração em (1.13) pode ser realizada sobre linha tracejada da Figura 1.3(b), chegando a

$$S(E_2) = S(E_1) + (E_2 - E_1)\beta_T. \tag{1.14}$$

A condição para que (1.14) seja igual a (1.13) é que as áreas A_1 e A_2, na Figura 1.3(b), sejam iguais. Integrando sobre a linha tracejada da Figura 1.3(b), de E_1 até um valor qualquer de $E \in [E_1, E_2]$, temos que a equação da reta tracejada com inclinação β_T, Figura 1.3(a) é dada por

$$S(E) = S(E_1) + (E - E_1)\beta_T, \quad E \in [E_1, E_2]. \tag{1.15}$$

Para entendermos qual o significado físico desta reta, considere que o sistema se separe em duas fases, 1 e 2, com energias $(1-\eta)E_1$ e ηE_2, onde $\eta \in [0, 1]$ é a fração do sistema



(a) Entropia *vs.* energia (b) Inverso da temperatura *vs.* energia

Figura 1.3: (a) Representação da entropia S em função da energia E (linha sólida) com uma região convexa, $[E_1, E_2]$, e um envelope côncavo (linha tracejada). (b) Representação do inverso da temperatura β em função da energia E (linha sólida).

na fase 2. No caso em que a energia é aditiva, a energia total do sistema é expressa como

$$E = (1 - \eta)E_1 + \eta E_2, \tag{1.16}$$

e a entropia total do sistema será

$$S = (1 - \eta)S(E_1) + \eta S(E_2). \tag{1.17}$$

Substituindo (1.14) em (1.17), temos

$$S = S(E_1) + \eta(E_2 - E_1)\beta_T. \tag{1.18}$$

Da equação (1.16), podemos escrever (1.18) como

$$S(E) = S(E_1) + (E - E_1)\beta_T, \quad E \in [E_1, E_2], \tag{1.19}$$

que é a mesma expressão para a reta (1.15). Ou seja, a reta tracejada da Figura 1.3(a) corresponde a uma separação de fases no sistema. Como a entropia da separação de fases é maior, o sistema tende a se separar nas duas fases. Neste caso, a região convexa é substituída pela reta tracejada, Figura 1.3(a), recuperando a equivalência entre os ensembles. Esse procedimento, de construção de um "envelope côncavo", é chamado de construção de Maxwell.

Sistemas com interações de longo alcance possuem energia não-aditiva, mesmo no limite termodinâmico. Portanto, não é possível existir uma separação de fases, pois a

equação (1.16) deixa de ser válida. Neste caso, quando não é possível recuperar a equivalência entre os ensembles, fica claro que não há uma correspondência um-para-um entre energia e temperatura (linha sólida da Figura 1.3(b)). Pela definição termodinâmica do calor específico de um sistema

$$\frac{\partial^2 S}{\partial E^2} = -\frac{1}{T^2 C}, \tag{1.20}$$

nas regiões onde a entropia é uma função convexa da energia, o calor específico C é negativo. Quando um sistema não possui energia aditiva, como sistemas com interações de longo alcance, é recomendado o uso do ensemble microcanônico, pois as construções do ensemble canônico exigem a aditividade da energia.

1.3 Propriedades dinâmicas

Apesar da não-aditividade e inequivalência entre ensembles, espera-se que a mecânica estatística de equilíbrio possa nos dar respostas sobre os estados estacionários de sistemas com interações de longo alcance. Porém, existem barreiras a serem enfrentadas quando tratamos da dinâmica de tais sistemas. Em sistemas com interações de curto alcance, o equilíbrio termodinâmico atingido tem sua função de distribuição de partículas dada pela mecânica estatística de Boltzmann-Gibbs [29, 30]. Nesses casos, o estado de equilíbrio termodinâmico do sistema não depende da distribuição inicial das partículas, apenas das variáveis globais conservadas como energia, volume, momento angular, etc. O que é diferente do caso de sistemas com interações de longo alcance, onde as condições iniciais desempenham um papel fundamental e levam o sistema a estados quasi-estacionários, em que as funções de distribuição não obedecem a mecânica de Boltzmann-Gibbs [31]. O grande desafio é exatamente prever quantitativamente esses estados estacionários atingidos por esses sistemas sem precisar resolver a dinâmica de N-corpos ou a equação de Vlasov:

$$\frac{\partial f}{\partial t} + \frac{\mathbf{p}}{m} \cdot \frac{\partial f}{\partial \mathbf{q}} + \mathbf{F}(\mathbf{q}, t) \cdot \frac{\partial f}{\partial \mathbf{p}} = 0, \tag{1.21}$$

onde $f \equiv f(\mathbf{q}, \mathbf{p}, t)$ é a função de distribuição de uma partícula e $\mathbf{F}(\mathbf{q}, t)$ a força de campo médio dada por:

$$\begin{aligned} \mathbf{F}(\mathbf{q},t) &= -\nabla\overline{V}(\mathbf{q},t) \\ \overline{V}(\mathbf{q},t) &\equiv \int V(\mathbf{q},\mathbf{q}')f(\mathbf{q}',\mathbf{p}',t)d\mathbf{q}'d\mathbf{p}'. \end{aligned} \tag{1.22}$$

com $V(\mathbf{q},\mathbf{q}')$ sendo o potencial do sistema.

Em 1967, Lynden-Bell propôs uma generalização da mecânica estatística de Boltzmann-Gibbs para lidar com sistemas com interações de longo alcance [32]. Ao perceber que a dinâmica da função de distribuição de tais sistemas era governada pela equação de Vlasov [3], ele utilizou a contagem de Boltzmann, mas ao invés de partículas, trabalhou diretamente com cortes discretos da função de distribuição. Uma vez que a distribuição inicial, pela equação de Vlasov, deve evoluir como um fluido incompressível no espaço de fase, Lynden-Bell definiu uma entropia de "caroço-grosso", na qual ele argumentou que a maximização deveria levar a distribuição mais provável. Entretanto, simulações computacionais mostraram que, em geral, a estatística de Lynden-Bell não era capaz de prever a distribuição de partículas de sistemas auto-gravitantes, por conta disso, a teoria foi abondonada no contexto da astrofísica. Mais tarde a teoria foi redescoberta pela comunidade da mecânica estatística, quando surgiram trabalhos mostrando que para alguns sistemas, como o cosHMF, a teoria de Lynden-Bell poderia fazer boas previsões [33, 34, 35, 36]. Um dos motivos atribuídos a falha da estatística de Lynden-Bell foi a quebra de ergodicidade [37]. A ergodicidade, um dos principais pressupostos da mecânica estatística de Boltzmann-Gibbs, nos diz que em um sistema isolado (ensemble microcanônico) sua distribuição inicial de partículas deve se espalhadar por todo o espaço de fase acessível, de tal forma que no equilíbrio todos os microestados correspondentes a um mesmo macroestado termodinâmico sejam equiprováveis [31]. Entretando, a quebra da ergodicidade vem sido observada em muitos sistemas com interações de longo alcance [38, 39, 40].

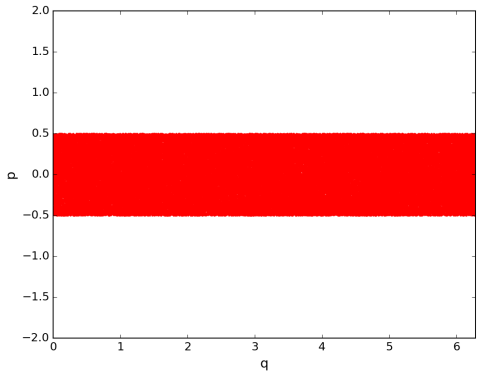
Surgiram trabalhos mostrando que quando a distribuição inicial do sistema satisfaz a condição do virial a estatística de Lynden-Bell prevê precisamente o estado estacionário de sistemas gravitacionais e Coulombianos [41, 42, 43, 44]. A generalização desse resultado para sistemas com potenciais que não são funções homogêneas, necessários para a condição do virial, veio com o trabalho de Benetti *et al.*, aplicado ao modelo cosHMF [45]. Quando os sistemas partem de uma distribuição inicial que não safistaz a condição do virial, o potencial de campo médio sofre fortes oscilações, algumas partículas entram em ressonância com estas oscilações e acabam recebendo muita energia, levando-as a regiões pouco

prováveis do espaço de fase, de acordo com a estatística de Boltzmann-Gibbs. Uma vez que as oscilações do campo médio desaparecem, as partículas sentem apenas um potencial de campo médio estático, onde a dinâmica se torna regular e as partículas não possuem meios de trocar energia entre si. As partículas altamente energéticas ficam presas em regiões improváveis do espaço de fase sem termalizar com o resto do sistema, resultando no sistema preso indefinidamente em um estado de não equilíbrio termodinâmico. Na Figura 1.4 temos a evolução do modelo cosHMF, partindo de uma distribuição inicial *waterbag* que rapidamente atinge uma distribuição estacionária que não corresponde ao equilíbrio termodinâmico. Esse sistema possui uma hamiltoniana dada por

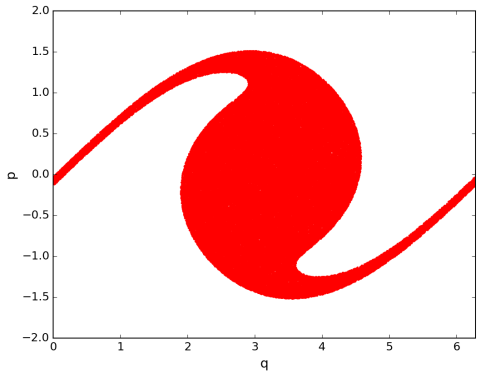
$$H = \sum_{i=1}^{N} \frac{p_i^2}{2} + \frac{1}{2N} \sum_{i,j=1}^{N} [1 - \cos(\theta_i - \theta_j)], \tag{1.23}$$

onde θ e p são as posições das partículas num círculo unitário e seus momentos conjugados, respectivamente.

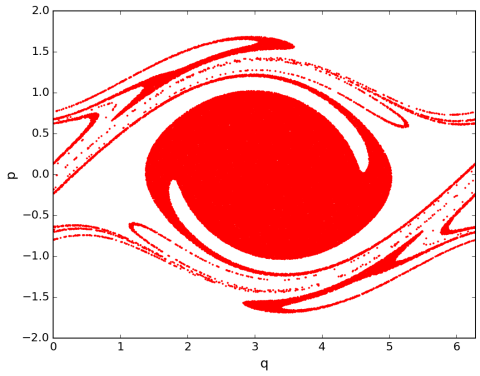
Esta situação extrema, onde o sistema nunca atinge o equilíbrio, só acontece no limite termodinâmico, $N \to \infty$. No mundo real, quando temos um número finito de partículas, as colisões entram em cena e as partículas podem então trocar energia entre si, fazendo com que o sistema atinja o equilíbrio termodiâmico previsto pela estatística de Boltzmann-Gibbs [8, 9, 10, 11]. Ainda assim, o tempo necessário para que se alcance o equilíbrio termodinâmico, em sistemas com interações de longo alcance, escala com o número de partículas do sistema, mais precisamente com N^2 [12], de tal forma que o sistema ainda passa um longo período de tempo em um estado quasi-estacionário, como podemos ver na Figura 1.5 para o cosHMF. Portanto, se tratando de simulações computacionais utilizando dinâmica molecular, para realizar alguma análise do sistema no equilíbrio esse tempo de relaxação é um dos problemas que precisa ser enfrentado, uma vez que um número cada vez maior de partículas é, geralmente, desejado para melhorar as precisão dos métodos aplicados.



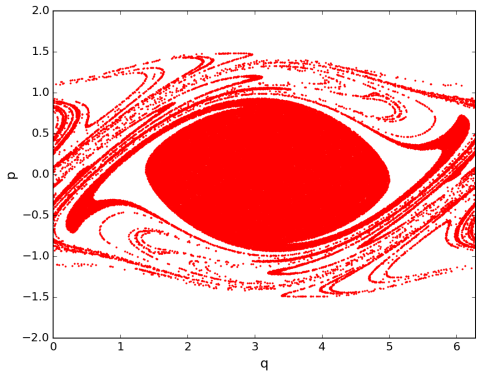
(a) t = 0.0



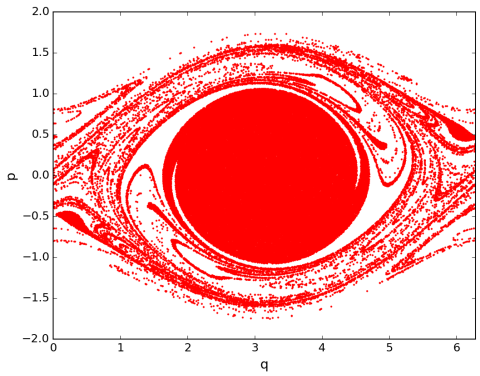
(b) t = 0.25



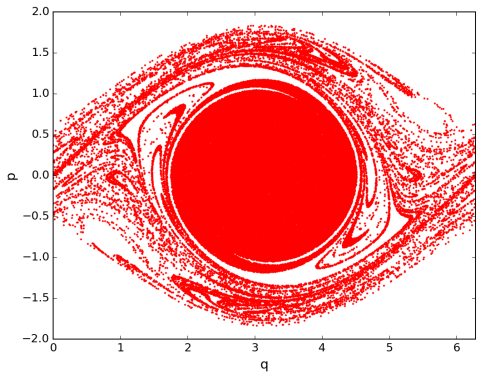
(c) t = 0.50



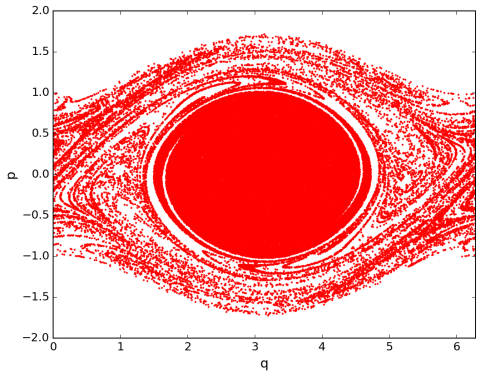
(d) t = 0.75



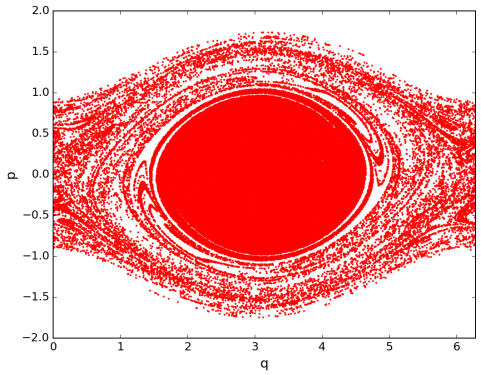
(e) t = 0.100



(f) t = 0.125

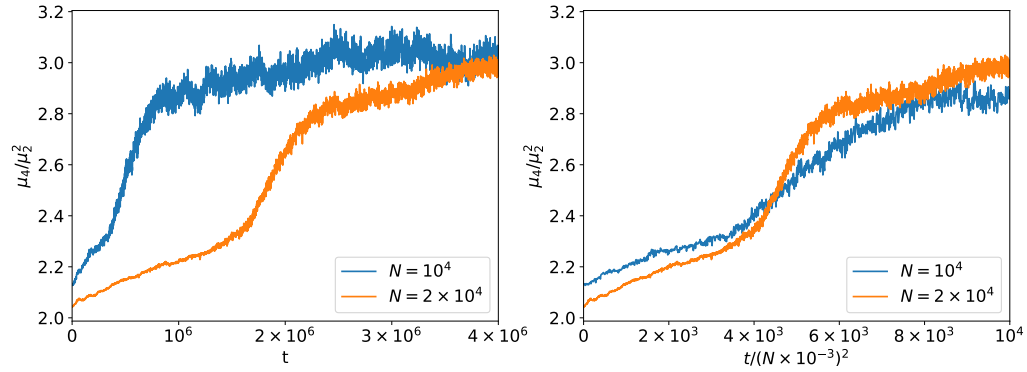


(g) t = 0.150

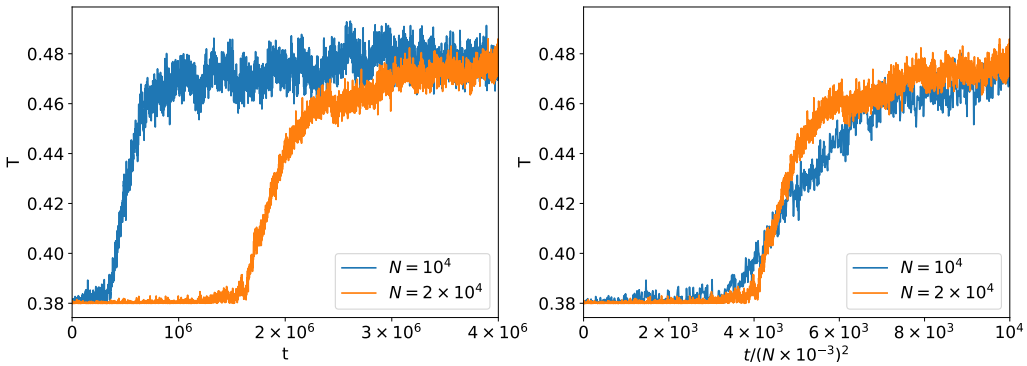


(h) t = 0.175

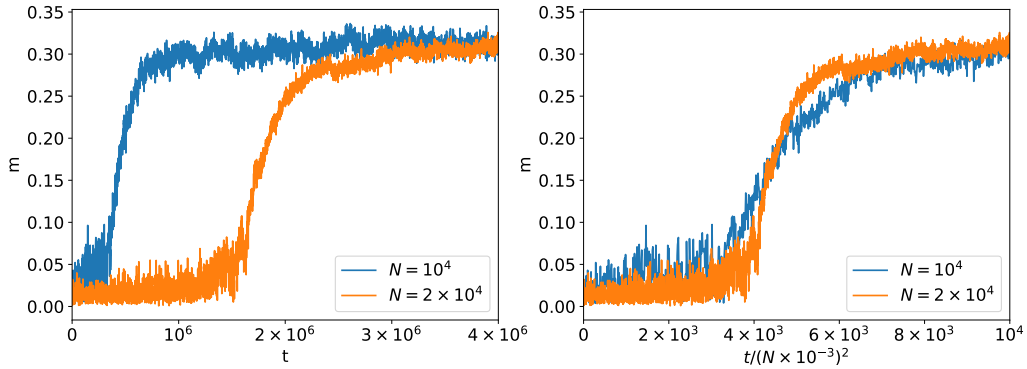
Figura 1.4: Evolução do cosHMF utilizando o integrador $leapfrog$, para $N = 10^5$, $e = 0.55$ e passo do integrador $h = 0.01$.



(a) Curtose dos momentos



(b) Temperatura



(c) Magnetização

Figura 1.5: Evolução do cosHMF para $e = 0.69$ com $N = 10^4$ e $N = 2 \times 10^4$ partindo de uma condição inicial de *waterbag*, nas Figuras da direita o tempo é reescalado como $t \to t/(N \times 10^{-3})^2$. (a) Curtose dos momentos das partículas. (b) Temperatura. (c) Magnetização.

1.4 Hamiltoniana de campo médio generalizada

O modelo GHMF é uma versão de longo alcance dos modelos estudados nas Refs. [46, 47], introduzido por Teles *et al.* [26]. O modelo consiste de N partículas com posições θ_i em um círculo unitário e momentos conjugados p_i, com a Hamiltoniana

$$H = \sum_{i=1}^{N} \frac{p_i^2}{2} + \frac{1}{2N} \sum_{i,j=1}^{N} V(\theta_i - \theta_j), \tag{1.24}$$

e com o potencial

$$V(\theta) = 1 - \Delta \cos\theta - (1 - \Delta)\cos(q\theta), \tag{1.25}$$

onde q é um inteiro positivo e $\Delta \in [0, 1]$. Existem diversas vantagens em se trabalhar com esse sistema, a primeira é que o modelo cosHMF, amplamente estudado, é recuperado com $\Delta = 1$. Além disso o modelo GHMF possui solução analítica tanto canônica quanto microcanônica e por ser tratar de um modelo de campo médio o esforço numérico de simulações escalam com N ao invés de N^2, o que permite simulações com grandes valores de N [26, 48]. Mas uma das principais características que torna esse modelo interessante, para o estudo proposto neste trabalho, é seu rico diagrama de fases, com transições de primeira e segunda ordem dependendo do valor de Δ escolhido. Definindo

$$\mathbf{m}_1 = (m_{1x}, m_{1y}) = (\langle\cos\theta\rangle, \langle\sin\theta\rangle) \tag{1.26}$$

e

$$\mathbf{m}_q = (m_{qx}, m_{qy}) = (\langle\cos q\theta\rangle, \langle\sin q\theta\rangle), \tag{1.27}$$

a Hamiltoniana pode ser reescrita como

$$H = \sum_{i=1}^{N} \frac{p_i^2}{2} + \frac{N}{2}\left[1 - \Delta\, m_1^2 - (1 - \Delta)\, m_q^2\right]. \tag{1.28}$$

Para obtermos o espaço de fase do sistema, primeiro temos que calcular sua entropia microcanônica. Pela estatística de Boltzmann-Gibbs, toda a informação termodinâmica do sistema está no seu volume acessível W no espaço de fase, dados os parâmetros macroscópicos conservados, nesse caso a energia E. Com isso a entropia é obtida por

$$S = k_B \ln W, \tag{1.29}$$

onde k_B é a constante de Boltzmann, que por simplicidade será ocultada por ter ser valor fixado como sendo 1. Sejam K e U as energias cinética e potencial, respectivamente, de um sistema com energia total E. O número de microestados desse sistema é dado por, [27],

$$
\begin{aligned}
W(E) &= \int \prod_{i=1}^{N} dp_i d\theta_i \delta(E-H) = \\
&= \int \prod_{i=1}^{N} dp_i d\theta_i \underbrace{\int dK \delta\left(K - \sum_{j=1}^{N} \frac{p_j^2}{2}\right)}_{=1} \delta(E-K-U(\{\theta_i\})) = \\
&= \int dK \underbrace{\int \prod_{i=1}^{N} dp_i \delta\left(K - \sum_{j=1}^{N} \frac{p_j^2}{2}\right)}_{W_{cin}(K)} \underbrace{\int \prod_{i=1}^{N} d\theta_i \delta(E-K-U(\{\theta_i\}))}_{W_{conf}(E-K)} \\
&= \int dK W_{cin}(K) W_{conf}(E-K). \qquad (1.30)
\end{aligned}
$$

Utilizando uma transformada de Laplace para resolver $W_{cin}(K)$, temos

$$
\begin{aligned}
Z_{cin}(\beta) &= \int_0^\infty e^{-\beta K} W_{cin}(K) dK = \\
&= \int_0^\infty \int_{-\infty}^\infty dK \prod_{i=1}^{N} dp_i e^{-\beta K} \delta\left(K - \sum_{j=1}^{N} \frac{p_j^2}{2}\right) = \\
&= \int_{-\infty}^\infty \prod_{i=1}^{N} dp_i \exp\left(-\beta \sum_{j=1}^{N} \frac{p_j^2}{2}\right) = \\
&= \int_{-\infty}^\infty \prod_{i=1}^{N} dp_i \prod_{j=1}^{N} \exp\left(-\beta \frac{p_j^2}{2}\right) = \\
&= \int_{-\infty}^\infty dp_1 \exp\left(-\beta \frac{p_1^2}{2}\right) \int_{-\infty}^\infty dp_2 \exp\left(-\beta \frac{p_2^2}{2}\right) \cdots \int_{-\infty}^\infty dp_N \exp\left(-\beta \frac{p_N^2}{2}\right) = \\
&= \left[\int_{-\infty}^\infty dp \exp\left(-\beta \frac{p^2}{2}\right)\right]^N . \qquad (1.31)
\end{aligned}
$$

Usando $\int_{-\infty}^{\infty} e^{-\alpha x^2} dx = \sqrt{\frac{\pi}{\alpha}}$ em (1.31), temos que

$$
Z_{cin}(\beta) = \left(\frac{2\pi}{\beta}\right)^{\frac{N}{2}} . \qquad (1.32)
$$

Aplicando a transformada inversa de Laplace $\mathcal{L}^{-1}\left\{\frac{\Gamma(n+1)}{s^{n+1}}\right\} = t^n \Theta(t)$, em (1.32), obtemos

$$W_{cin}(K) = \mathcal{L}^{-1}\left\{Z_{cin}(K)\right\} = \mathcal{L}^{-1}\left\{\left(\frac{2\pi}{\beta}\right)^{\frac{N}{2}}\right\} =$$
$$= \frac{(2\pi)^{\frac{N}{2}}}{\Gamma(\frac{N}{2})}\mathcal{L}^{-1}\left\{\frac{\Gamma(\frac{N}{2})}{\beta^{\frac{N}{2}}}\right\} = \frac{(2\pi)^{\frac{N}{2}}}{\Gamma(\frac{N}{2})}K^{\frac{N}{2}-1}\Theta(K),$$

como $K \geq 0$,

$$W_{cin}(K) = \frac{(2\pi)^{\frac{N}{2}}}{\Gamma(\frac{N}{2})}K^{\frac{N}{2}-1}$$

$$\therefore \ \ln W_{cin}(K) = \frac{N}{2}\ln 2\pi + \left(\frac{N}{2}-1\right)\ln K - \ln\Gamma\left(\frac{N}{2}\right), \tag{1.33}$$

usando a expressão assintótica da função Γ, $\ln\Gamma(N) \simeq \left(N-\frac{1}{2}\right)\ln N - N + \frac{1}{2}\ln 2\pi$, em (1.33),

$$\ln W_{cin}(K) \simeq \frac{N}{2}\left[\ln 2\pi + \left(1-\frac{2}{N}\right)\ln K - \left(1-\frac{1}{N}\right)\ln\frac{N}{2} + 1 - \frac{1}{N}\ln 2\pi\right]. \tag{1.34}$$

Desprezando os termos que vão a zero no limite em que $N \to \infty$, a equação (1.34) se torna

$$\ln W_{cin}(K) \simeq \frac{N}{2}\left[1 + \ln 2\pi + \ln\frac{2K}{N}\right]$$

$$\therefore \ W_{cin}(K) \simeq \exp\left[\frac{N}{2}\left(1 + \ln 2\pi + \ln\frac{2K}{N}\right)\right]. \tag{1.35}$$

Substituindo (1.35) em (1.30)

$$W(E) = \int dK \exp\left[\frac{N}{2}\left(1 + \ln 2\pi + \ln\frac{2K}{N}\right)\right] W_{conf}(E-K). \tag{1.36}$$

Definindo a densidade de energia cinética, potencial e total como $k = K/N = (E-U)/N = e - u$ e $u = \frac{U}{N}$ e a entropia configuracional como $s_{conf}(u) = \frac{1}{N}\ln W_{conf}(u)$. A função de partição microcanônica (1.36) pode ser reescrita como

$$W(Ne) = N\int dk \exp\left[N\left(\frac{1}{2} + \frac{1}{2}\ln 2\pi + \frac{1}{2}\ln 2k + s_{conf}(e-k)\right)\right]. \tag{1.37}$$

A expressão (1.37) nos dá o número de microestados do sistema com uma energia total $E = Ne$. Podemos interpretar a integração em k como a soma de todos os microestados com uma energia total $E = Ne$ e energia cinética $K = Nk$, ou seja,

$$W(Ne) = \int d(Nk) W(Ne, Nk). \tag{1.38}$$

Comparando (1.38) com (1.37), temos que

$$W(Ne, Nk) = \exp\left[N\left(\frac{1}{2} + \frac{1}{2}\ln 2\pi + \frac{1}{2}\ln 2k + s_{conf}(e-k)\right)\right]. \tag{1.39}$$

Substituindo $k = e - u$, em (1.39), temos o número de microestados do sistema com energia total $E = Ne$ e energia potencial $U = Nu$,

$$W(Ne, Nu) = \exp\left[N\left(\frac{1}{2} + \frac{1}{2}\ln 2\pi + \frac{1}{2}\ln\left(2e - 2u\right) + s_{conf}(u)\right)\right]. \tag{1.40}$$

Com isso, a entropia por partícula, em função de e, é obtida aplicando o limite termodinâmico e maximizando a expressão em relação a u

$$\begin{aligned} s(e) &= sup_u\left[\lim_{N\to\infty}\frac{1}{N}\ln W_N(Ne, Nu)\right] \\ &= \frac{1}{2} + \frac{1}{2}\ln 2\pi + sup_u\left[\frac{1}{2}\ln\left(2e - 2u\right) + s_{conf}(u)\right]. \end{aligned} \tag{1.41}$$

Para a maximização, é necessário ter uma expressão explícita da entropia configuracional $s_{conf}(u)$, ou seja, é preciso calcular $W_{conf}(U)$. No caso do GHMF, como a energia potencial depende apenas de m_1 e m_q, podemos definir

$$W(m_1, m_q) = \int \prod_i d\theta_i \delta\left(\sum_i \cos\theta_i - Nm_1\right)\delta\left(\sum_i \cos q\theta_i - Nm_q\right), \tag{1.42}$$

com $W(m_1, m_q)$ sendo proporcional a $W_{conf}(U)$ (a constante de proporcionalidade desaparece no limite termodinâmico). Existe uma degenerescência contínua da fase do sistema, portanto, não há perda de generalidade em definir as magnetizações apenas com componente x, como foi feito na expressão (1.42). Usando a representação de Fourier da função δ em (1.42), temos

$$
\begin{aligned}
W(m_1, m_q) &= \left(\frac{1}{2\pi}\right)^2 \int_{-\infty}^{\infty} dx \int_{-\infty}^{\infty} dy \int \prod_i d\theta_i \exp\left[ix\left(\sum_i \cos\theta_i - Nm_1\right)\right] \times \\
&\quad \times \exp\left[iy\left(\sum_i \cos q\theta_i - Nm_q\right)\right] = \\
&= \left(\frac{1}{2\pi}\right)^2 \int_{-\infty}^{\infty} dx \int_{-\infty}^{\infty} dy \int \prod_i d\theta_i e^{-ixNm_1 - iyNm_q} \times \\
&\quad \exp\left(ix\sum_i \cos\theta_i + iy\sum_i \cos q\theta_i\right) = \\
&= \left(\frac{1}{2\pi}\right)^2 \int_{-\infty}^{\infty} dx \int_{-\infty}^{\infty} dy \int \prod_i d\theta_i e^{-ixNm_1 - iyNm_q} \times \\
&\quad \exp\left[\sum_i (ix\cos\theta_i + iy\cos q\theta_i)\right] = \\
&= \left(\frac{1}{2\pi}\right)^2 \int_{-\infty}^{\infty} dx \int_{-\infty}^{\infty} dy e^{-ixNm_1 - iyNm_q} \times \\
&\quad \int \prod_i d\theta_i \left[e^{ix\cos\theta_1 + iy\cos q\theta_1}\right] \cdots \left[e^{ix\cos\theta_N + iy\cos q\theta_N}\right] \\
&= \left(\frac{1}{2\pi}\right)^2 \int_{-\infty}^{\infty} dx \int_{-\infty}^{\infty} dy e^{-ixNm_1 - iyNm_q} \left[\int_0^{2\pi} d\theta e^{i(x\cos\theta + y\sin\theta)}\right]^N \\
&= \left(\frac{1}{2\pi}\right)^2 \int_{-\infty}^{\infty} dx \int_{-\infty}^{\infty} dy \exp\left\{N\left[-ixm_1 - iym_q + \right.\right. \\
&\quad \left.\left. + \ln\left(\int_{-\pi}^{\pi} d\theta \exp\left(ix\cos\theta + iy\cos q\theta\right)\right)\right]\right\}. \qquad (1.43)
\end{aligned}
$$

As integrais em x e y podem ser aproximadas pelo método do ponto de sela. Para tal, o ponto de extremo correspondente a (x^*, y^*) e deve satisfazer as seguintes relações:

$$
m_1 = \frac{\int_{-\pi}^{\pi} d\theta \cos\theta \exp[ix\cos\theta + iy\cos q\theta]}{\int_{-\pi}^{\pi} d\theta \exp[ix\cos\theta + iy\cos q\theta]}, \qquad (1.44)
$$

$$
m_q = \frac{\int_{-\pi}^{\pi} d\theta \cos q\theta \exp[ix\cos\theta + iy\cos q\theta]}{\int_{-\pi}^{\pi} d\theta \exp[ix\cos\theta + iy\cos q\theta]}. \qquad (1.45)
$$

Definindo $a = ix^*$ e $b = iy^*$ e ignorando termos de ordem inferior a N, temos que a entropia configuracional, em função das magnetizações, dada por

$$s_{conf}(m_1, m_q) = -m_1 a(m_1, m_q) - m_q b(m_1, m_q) + \\ + \ln\left(\int d\theta \exp\left[a(m_1, m_q)\cos\theta + b(m_1, m_q)\cos q\theta\right]\right). \quad (1.46)$$

Como no limite termodinâmico podemos substituir $W_{conf}(U)$ por $W(m_1, m_q)$ e $u = (1 - \Delta m_1^2 - (1-\Delta)m_q^2)/2$, a maximização em (1.41) pode ser realizada em m_1 e m_q ao invés de u, sendo assim

$$s(e) = \frac{1}{2} + \frac{1}{2}\ln 2\pi + \\ + sup_{m_1, m_q}\left\{\frac{1}{2}\ln\left[2e - 1 + \Delta m_1^2 + (1-\Delta)m_q^2\right] - \right. \\ - m_1 a(m_1, m_q) - m_q b(m_1, m_q) + \\ \left. + \ln\left(\int d\theta \exp\left[a(m_1, m_q)\cos\theta + b(m_1, m_q)\cos q\theta\right]\right)\right\}. \quad (1.47)$$

A maximização em m_1 e m_q leva as equações

$$\frac{\Delta m_1^*}{2e - 1 + \Delta m_1^{*2} + (1-\Delta)m_q^{*2}} = a(m_1^*, m_q^*), \quad (1.48)$$

$$\frac{(1-\Delta)m_q^*}{2e - 1 + \Delta m_1^{*2} + (1-\Delta)m_q^{*2}} = b(m_1^*, m_q^*). \quad (1.49)$$

Substituindo as expressões (1.48) e (1.49) nas equações (1.44) e (1.45), temos os valores de equilíbrio para os parâmetros de ordem, m_1 e m_q, onde os símbolos $*$ foram removidos para simplificar a notação

$$m_1 = \frac{\int_{-\pi}^{\pi} d\theta \cos\theta \exp\left[\dfrac{\Delta m_1 \cos\theta + (1-\Delta)m_q \cos q\theta}{2e - 1 + \Delta m_1^2 + (1-\Delta)m_q^2}\right]}{\int_{-\pi}^{\pi} d\theta \exp\left[\dfrac{\Delta m_1 \cos\theta + (1-\Delta)m_q \cos q\theta}{2e - 1 + \Delta m_1^2 + (1-\Delta)m_q^2}\right]}, \quad (1.50)$$

$$m_q = \frac{\int_{-\pi}^{\pi} d\theta \cos q\theta \exp\left[\dfrac{\Delta m_1 \cos\theta + (1-\Delta)m_q \cos q\theta}{2e - 1 + \Delta m_1^2 + (1-\Delta)m_q^2}\right]}{\int_{-\pi}^{\pi} d\theta \exp\left[\dfrac{\Delta m_1 \cos\theta + (1-\Delta)m_q \cos q\theta}{2e - 1 + \Delta m_1^2 + (1-\Delta)m_q^2}\right]}. \quad (1.51)$$

Para o presente propósito, foi considerado aqui o caso $q = 2$, também considerado em [26, 31, 49], para o qual, existe uma fase paramagnética ($m_1 = m_2 = 0$) e uma

ferromagnética ($m_1 > 0,\ m_2 > 0$), além do modelo também apresentar uma fase nemática ($m_2 > m_1 = 0$). As transições são de segunda ordem exceto no intervalo de valores de Δ onde a transição ferromagnética-paramagnética é de primeira ordem, ou seja, mais de uma solução para as equações (1.50) e (1.51). O diagrama de fase do modelo GHMF pode ser visto na Figura 1.6, onde as linhas tracejadas correspondem a transições de fase de segunda ordem e a linha sólida transições de primeira ordem. Os círculos sólidos são os dois pontos tricríticos. As curvas calóricas e os parâmetros de ordem m_1 e m_2, para $\Delta = 0,\ 0.35,\ 0.5,\ 1$, em função da densidade de energia e, encontram-se nas Figuras 1.7 e 1.8.

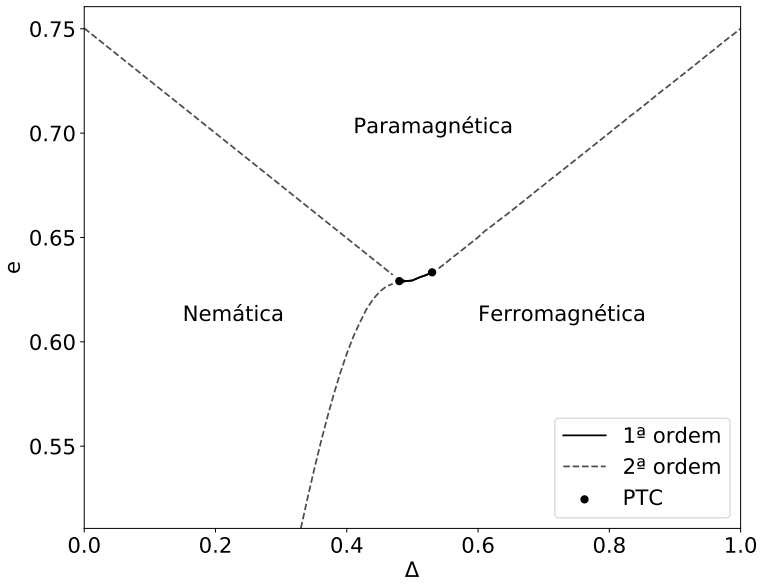


Figura 1.6: Diagrama de fase microcanônico do GHMF. Linhas tracejadas correspondem a transições de fase de segunda ordem e a linha sólida transições de primeira ordem. Círculos sólidos são os dois pontos tricríticos.

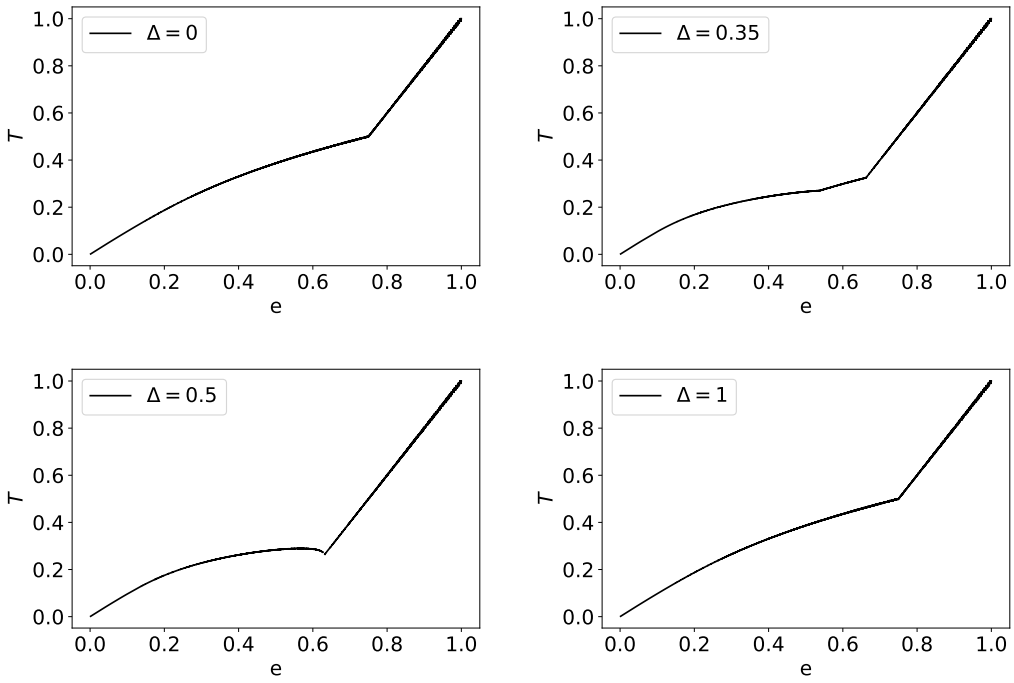


Figura 1.7: Curva calórica do modelo GHMF para $\Delta = 0,\ 0.35,\ 0.5,\ 1$ em função da densidade de energia.

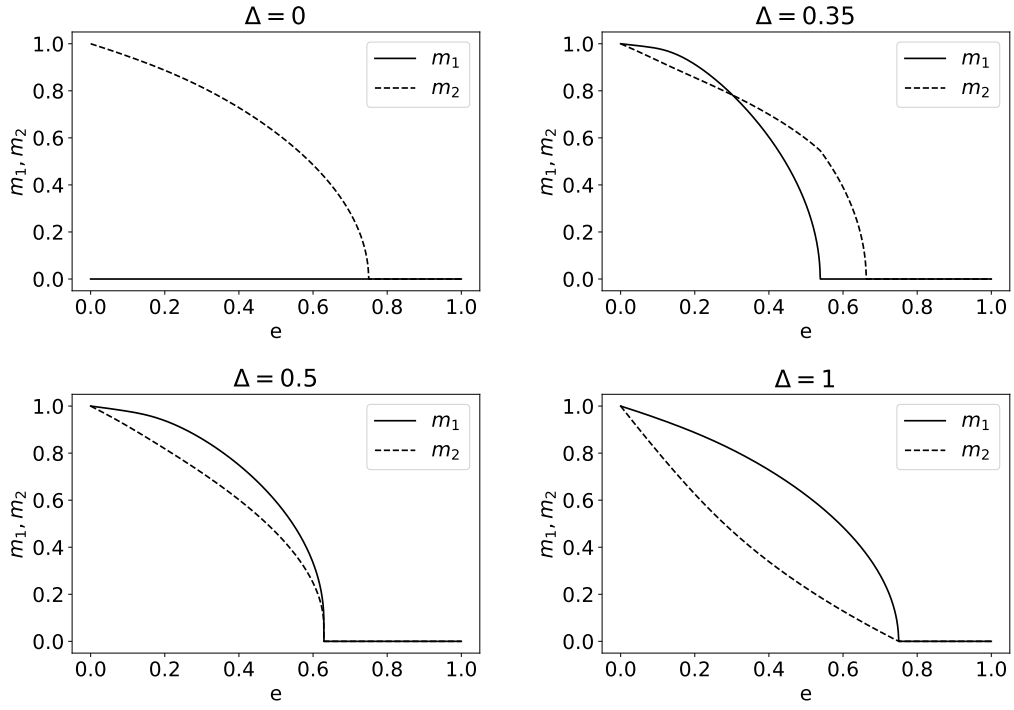


Figura 1.8: Parâmetros de ordem microcanônicos do modelo GHMF para $\Delta = 0,\ 0.35,\ 0.5,\ 1$ em função da densidade de energia.

Capítulo 2

Dinâmica não-Linear

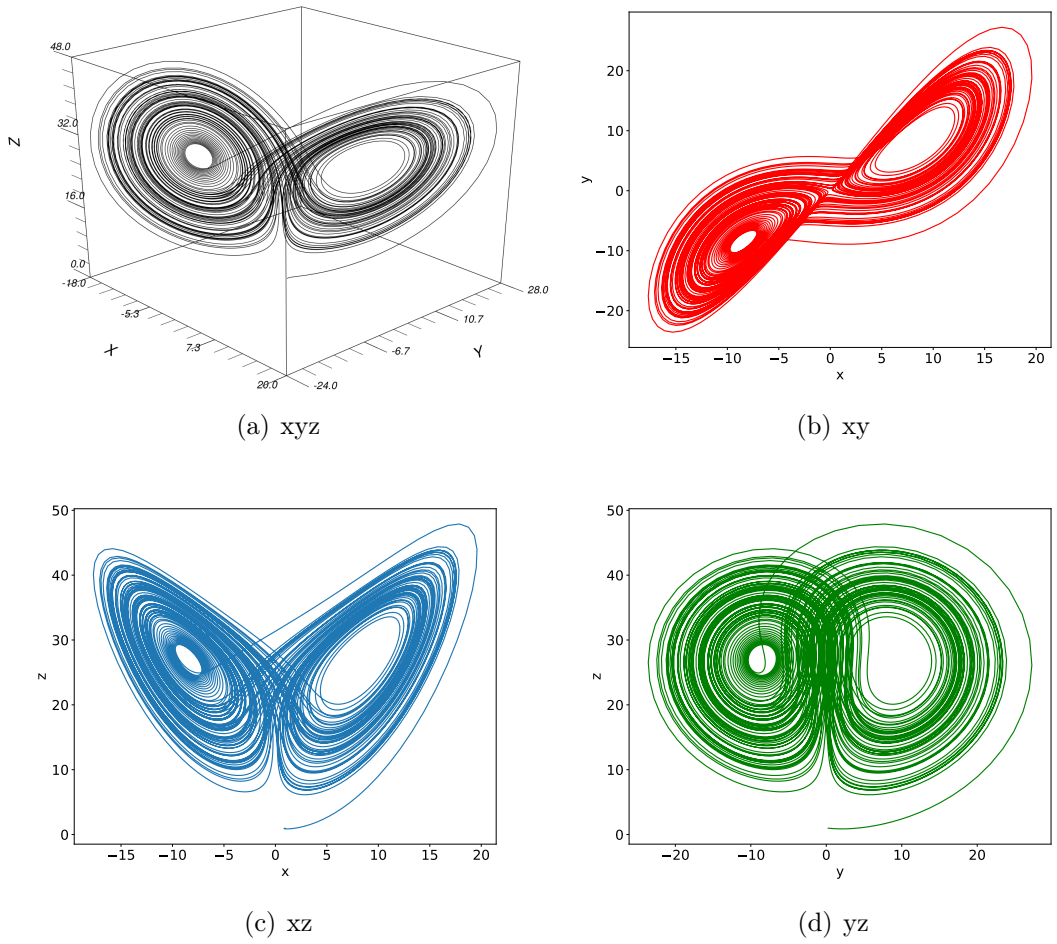
Neste capítulo vemos um pouco da história de sistemas caóticos, como mensurar a caoticidade de um sistema através dos expoentes de Lyapunov e como calcular esse expoente através de métodos númericos e analíticos.

2.1 Sistemas caóticos

Sistemas caóticos são sistemas dinâmicos extremamente sensíveis às condições iniciais. Soluções (ou órbitas) de um sistema caótico não convergem para um função estacionária ou periódica, além disso, soluções próximas tendem a distanciar-se exponencialmente entre si. No final do século XIX, Henri Poincaré, investigando seções do espaço de fase, foi o primeiro a notar a possibilidade de caos. Ele observou que sistemas determinísticos com comportamento aperiódico podiam apresentar uma dependência sensível às condições iniciais. Em 1961, Lorenz usou um computador para simular padrões climáticos. Seu objetivo era visualizar uma sequência de dados novamente e, para poupar tempo, iniciou uma nova simulação com uma condição inicial que correspodia ao meio de uma simulação anterior, com dados que foram imprimidos pelo computador. Entretando, os resultados eram completamente diferentes, comparados a simulação anterior. A causa disso era o arredondamento realizado pelo computador ao imprimir os números. O computador trabalhava com precisão de 6 dígitos, mas para impressão arredondava as variáveis para 3 dígitos. Essa pequena diferença nas condições iniciais geravam resultados distintos no decorrer da simulação. O trabalho de Lorenz [50] sobre um modelo simplificado do problema de rolos de convecção na atmosfera, apresentou comportamento caótico. Suas equações, devido sua simplicidade, são estudadas até hoje por quem é introduzido na área de dinâmica não linear [51]:

$$\begin{aligned} \dot{x} &= \sigma(y - x), \\ \dot{y} &= x(\rho - z) - y, \\ \dot{z} &= xy - \beta z. \end{aligned} \tag{2.1}$$

Para um certo conjunto de parâmetros σ, ρ e β, o sistema de Lorenz, apresenta um comportamento caótico apresentado nas Figuras 2.1.



(a) xyz

(b) xy

(c) xz

(d) yz

Figura 2.1: (a) Trajetória da solução das equações de Lorenz para $\sigma = 10$, $\rho = 28$ e $\beta = 8/3$. Resultado obtido via Runge-Kutta de 4ª ordem, com passo do integrador $h = 0.001$. (b) Projeção na plano XY. (c) Projeção no plano XZ. (d) Projeção no plano YZ.

Para mostrar a divergência de soluções próximas com o tempo, na Figura 2.2 temos duas soluções com condições iniciais que diferem por 10^{-2}, para $\sigma = 10$, $\rho = 28$ e $\beta = 8/3$.

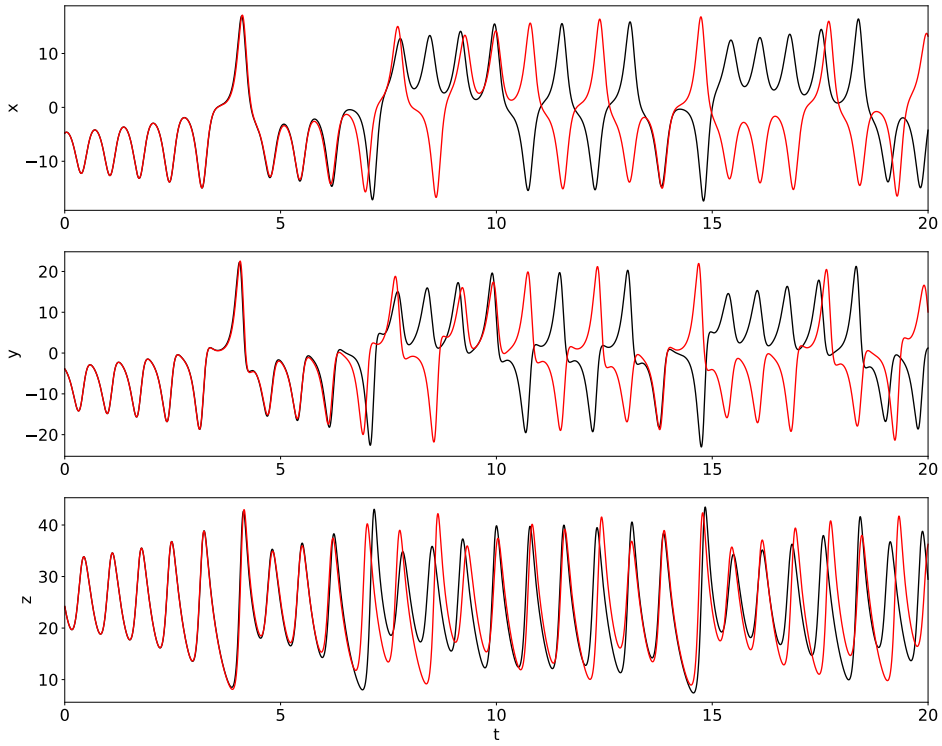


Figura 2.2: Soluções das equações de Lorenz para $\sigma = 10$, $\rho = 28$ e $\beta = 8/3$ cujas condições iniciais diferem por 10^{-2}.

Não devemos confundir caótico com aleatório, as equações que ditam a evolução de um sistema caótico são determinísticas, porém devido sua sensibilidade às condições inicias é praticamente impossível determinar indefinidamente sua trajetória no espaço de fase, uma vez que pequenas incertezas nas condições iniciais implicavam divergências exponenciais entre trajetórias.

2.2 Expoentes de Lyapunov

Uma vez entendido o caos de forma qualitativa, partimos para análises mais quantitativas. Duas trajetórias no espaço de fase, com separação inicial $\|\boldsymbol{w}_0\|$, divergem exponencialmente entre si como

$$\|\boldsymbol{w}(t)\| \approx \|\boldsymbol{w}_0\| e^{\lambda t}, \tag{2.2}$$

onde $\boldsymbol{w}$ é o vetor diferença entre as trajetórias e λ a taxa de separação entre elas, também chamado de expoente de Lyapunov. Podemos definir o expoente λ como

$$\lambda = \lim_{t\to\infty} \lim_{\|\boldsymbol{w}_0\|\to 0} \frac{1}{t} \ln \frac{\|\boldsymbol{w}(t)\|}{\|\boldsymbol{w}_0\|}. \tag{2.3}$$

O expoente de Lyapunov é uma estimativa da taxa de divergência ($\lambda > 0$) ou convergência ($\lambda < 0$) de soluções vizinhas. A taxa de separação pode ser diferente dependendo da orientação inicial do vetor $\boldsymbol{w}_0$, Portanto, existem n expoentes, sendo n igual a dimensionalidade do espaço de fase. O conjunto de todos os expoentes é denominado espectro de Lyapunov e pode ser organizado da seguinte forma

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n. \tag{2.4}$$

O maior expoente de Lyapunov (LLE) λ_1 nos da a noção de previsibilidade de um sistema dinâmico. Um LLE positivo é geralmente interpretado como um indicativo de que o sistema é caótico (dadas outras condições a serem satisfeitas, *e.g.*, órbitas confinadas em uma região do espaço de fase).

Para estimarmos os expoentes de Lyapunov, suponha um sistema qualquer, regido pelo seguinte conjunto de equações diferenciais:

$$\begin{aligned} \dot{x}_1 &= f_1(x_1, x_2, \ldots, x_n) \\ \dot{x}_2 &= f_2(x_1, x_2, \ldots, x_n) \\ &\vdots \\ \dot{x}_n &= f_n(x_1, x_2, \ldots, x_n), \end{aligned} \tag{2.5}$$

que pode ser rescrito de forma compacta como

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{F}(\boldsymbol{x}(t)). \tag{2.6}$$

A equação (2.6) nos dá a evolução do sistema, onde $\boldsymbol{F}$ é o campo de velocidades do fluído no espaço de fase. Para obter as divergências e convergências na vizinhança de $\boldsymbol{x}^*$, considere de duas soluções próximas, $\boldsymbol{x}_A(t)$ e $\boldsymbol{x}_B(t)$, e seu vetor diferença

$$\boldsymbol{w}(t) = \boldsymbol{x}_A(t) - \boldsymbol{x}_B(t). \tag{2.7}$$

A evolução do vetor diferença $\boldsymbol{w}(t)$, em primeira ordem, é dada por

$$\begin{aligned}\frac{d\boldsymbol{w}(t)}{dt} &= \boldsymbol{F}(\boldsymbol{x}_A(t)) - \boldsymbol{F}(\boldsymbol{x}_B(t)) \simeq \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}}\Big|_{\boldsymbol{x}=\boldsymbol{x}_A} \boldsymbol{w}(t) \\ &\equiv J\boldsymbol{w}(t), \end{aligned} \tag{2.8}$$

onde J é a matriz Jacobiana do campo vetorial $\boldsymbol{F}$ ao longo da trajetória $\boldsymbol{x}_A(t)$. Essa matriz nos fornece todos os coeficientes do sistema linearizado (2.8). Em um sistema Hamiltoniano, com N graus de liberdade, do tipo $H = K(p) + U(q)$ a matriz Jacobiana é do tipo $2N \times 2N$ e possui a forma

$$J = \begin{pmatrix} 0 & I \\ \tilde{J} & 0 \end{pmatrix}. \tag{2.9}$$

Cada elemento da representação (2.9) é uma matriz $N \times N$, onde I é a matriz identidade e $\tilde{J}$ é a matriz Hessiana do potencial, obtida através de

$$\tilde{J}_{ij} = -\frac{\partial^2 V}{\partial q_i \partial q_j}. \tag{2.10}$$

A solução do sistema linear (2.8) é dada por

$$\boldsymbol{w}(t) = \mathcal{H}(\boldsymbol{w}_0, t)\boldsymbol{w}(0), \tag{2.11}$$

$$\mathcal{H}(\boldsymbol{w}_0, t) = \exp\left(\int_0^t J(\boldsymbol{w}_0, t')dt'\right). \tag{2.12}$$

Com a definição (2.3) é possível estimar os valores dos expoentes de Lyapunov em conjunto da solução (2.11).

2.3 Método computacional

Um dos métodos computacionais de se obter os expoentes de Lyapunov é através da solução do sistema linearizado (2.8), mas para isso devemos evoluir simultaneamente a equação não linear (2.6), para calcularmos a matriz Jacobiana, com as n equações linearizadas (2.8), uma para cada dimensão do espaço de fase, e então aplicar a definição (2.3) para obtermos o espectro de Lyapunov, tal método é conhecido como Mapa Tangente.

Seja $\boldsymbol{x}$ meu sistema referência regido pela equação (2.8) e $\{\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_n\}$ uma base ortogonal com norma dos vetores $\epsilon \ll 1$, regidos pela equação (2.6). Então os expoentes

de Lyapunov são dados por

$$\lambda_i = \lim_{t\to\infty} \frac{1}{t} \ln \frac{\|\boldsymbol{w}_i(t)\|}{\epsilon}. \tag{2.13}$$

Como $\|\boldsymbol{w}(t)\|$ cresce exponencialmente, é necessário a realização de ortogonalização, seguido de uma renormalização, para impedirmos a divergência da norma dos vetores diferença, causando problemas de extrapolação de memória computacional. Por fim, o algoritmo segue os seguintes passos:

– Definir condições iniciais para o sistema referência $\boldsymbol{x}_0$ e para os vetores diferença $\{\boldsymbol{w}_{01}, \boldsymbol{w}_{02}, \ldots, \boldsymbol{w}_{0n}\}$, uma base ortonormal de norma ϵ, sendo ϵ um escalar muito pequeno;

– Integração simultânea de todos os sistemas, totalizando uma dinâmica de $n+1$ sistemas resultando em $\boldsymbol{x}_0 \to \boldsymbol{x}$ e $\{\boldsymbol{w}_{01}, \boldsymbol{w}_{02}, \ldots, \boldsymbol{w}_{0n}\} \to \{\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_n\}$;

– Em cada tempo T_{norm} aplicar a ortogonalização nos vetores diferença, levando $\boldsymbol{w} \to \boldsymbol{v}$, seguida de renormalização $\boldsymbol{v} \to \dfrac{\boldsymbol{v}}{\|\boldsymbol{v}\|}$;

– Redefinir as condições iniciais dos vetores diferença como $\epsilon\boldsymbol{v}$;

– Repetir os 3 processos anteriores K vezes;

– Obter os expoentes de Lyapunov atráves das médias

$$\lambda_i = \frac{1}{KT_{norm}} \sum_{k=1}^{K} \ln \frac{\|\boldsymbol{w}_i^{(k)}\|}{\epsilon}, \tag{2.14}$$

sendo K grande o suficiente para a convergência de λ_i.

O método de Mapa Tangente, na forma de um pseudocódigo, encontra-se no Algoritmo 1 [52, 53].

Algorithm 1 Mapa Tangente para o espectro de Lyapunov

procedure
  $\boldsymbol{x} \leftarrow initial\_conditions()$
  $\boldsymbol{w}_i \leftarrow initial\_conditions(),\ with\ ||\boldsymbol{w}_i|| = \epsilon\ and\ \epsilon \ll 1,\ \forall i$
  $\lambda_i \leftarrow 0,\ \forall i$
  for $k \leftarrow 1$ to K **do**
    $t \leftarrow 0$
    while $t < T_{norm}$ **do**
      $\boldsymbol{x} \leftarrow evolve(\boldsymbol{x})$
      for $i \leftarrow 1$ to n **do**
        $\boldsymbol{w}_i \leftarrow evolve(\boldsymbol{w}_i)$
      $t \leftarrow t + dt$
    for $i \leftarrow 1$ to n **do**
      $\lambda_i \leftarrow \lambda_i + \ln \frac{||\boldsymbol{w}_i||}{\epsilon}$
      $\boldsymbol{w}_i \leftarrow ortogonalization(\boldsymbol{w}_i)$
      $\boldsymbol{w}_i \leftarrow \frac{\epsilon}{||\boldsymbol{w}_i||}\boldsymbol{w}_i$
  $\lambda_i \leftarrow \frac{\lambda_i}{KT_{norm}},\ \forall i$

Todo o método descrito até agora é para o cálculo de todo o espectro de Lyapunov. Para obtermos apenas o LLE, que é o foco desse trabalho, o procedimento é parecido, mas com algumas simplificações. Por exemplo, ao invés de resolvermos $n + 1$ equações, precisamos resolver apenas duas, uma do sistema não-linear (2.8) e outra do linear (2.6) (apenas um vetor diferença) e, consequentemente, não é necessário a aplicação do processo de ortogonalização. Com isso, o algoritmo para obtermos apenas o LLE utilizando o método de Mapa Tangente fica como o apresentado no Algoritmo 2.

Algorithm 2 Mapa Tangente para o LLE

procedure
  $\boldsymbol{x} \leftarrow initial\_conditions()$
  $\boldsymbol{w} \leftarrow initial\_conditions(),\ with\ ||\boldsymbol{w}|| = \epsilon\ and\ \epsilon \ll 1$
  $\lambda \leftarrow 0$
  for $k \leftarrow 1$ to K **do**
    $t \leftarrow 0$
    while $t < T_{norm}$ **do**
      $\boldsymbol{x} \leftarrow evolve(\boldsymbol{x})$
      $\boldsymbol{w} \leftarrow evolve(\boldsymbol{w})$
      $t \leftarrow t + dt$
    $\lambda \leftarrow \lambda + \ln \frac{||\boldsymbol{w}||}{\epsilon}$
    $\boldsymbol{w} \leftarrow \frac{\epsilon}{||\boldsymbol{w}||}\boldsymbol{w}$
  $\lambda \leftarrow \frac{\lambda}{KT_{norm}}$

2.4 Método analítico

Um cálculo analítico para o LLE do cosHMF, que é o mesmo que o modelo GHMF com $\Delta = 1$, foi feito por Firpo [24] sob a luz dos trabalhos de Pettini e colaboradores [20, 21, 22, 23]. A ideia foi reformular a dinâmica Hamiltoniana em uma linguagem de geometria Riemanniana, onde as trajetórias correspondem a geodésicas de uma métrica subjacente. O caos surge de instabilidades no fluxo das geodésicas, que depende das propriedades da curvatura da variedade Riemanniana [20]. Assumindo que a curvatura efetiva sobre a trajetória é bem representada por um processo estocástico gaussiano, Caiani *et al.* [23] derivaram a seguinte expressão para λ:

$$\lambda = \frac{\Lambda}{2} - \frac{2\kappa_0}{3\Lambda}, \tag{2.15}$$

com

$$\Lambda = \left(2\sigma_\kappa^2 \tau + \sqrt{\frac{64}{27}\kappa_0^3 + 4\sigma_\kappa^4 \tau^2} \right)^{\frac{1}{3}}, \tag{2.16}$$

$$\tau = \frac{\pi\sqrt{\kappa_0}}{2\sqrt{\kappa_0}\sqrt{\kappa_0 + \sigma_\kappa} + \pi\sigma_\kappa}. \tag{2.17}$$

Onde $\kappa_0 \equiv \langle k_R \rangle_\mu$ e $\sigma_\kappa^2 \equiv \langle \delta^2 K_R \rangle_\mu$ (o índice μ indica média no *ensemble* microcanônico), sendo $K_R(q) = \nabla^2 U(q)$ a curvatura de Ricci e $k_R = \dfrac{K_R}{N-1}$ a média da curvatura de Ricci [22]. A energia potencial do cosHMF é a mesma do GHMF com $\Delta = 1$:

$$U = \frac{1}{2N} \sum_{i,j}^{N} [1 - \cos(\theta_i - \theta_j)], \tag{2.18}$$

como a média da curvatura de Ricci é dada por

$$k_R = \frac{1}{N-1} \sum_{k=1}^{N} \frac{\partial^2 U}{\partial \theta_k^2}, \tag{2.19}$$

devemos primeiro calcular $\dfrac{\partial^2 U}{\partial \theta_k^2}$. Temos que

$$\begin{aligned}
\frac{\partial^2 U}{\partial \theta_k^2} &= \frac{\partial}{\partial \theta_k}\frac{1}{2N}\sum_{i,j}^{N}\frac{\partial}{\partial \theta_k}[1-\cos(\theta_i-\theta_j)] \\
&= \frac{\partial}{\partial \theta_k}\frac{1}{2N}\sum_{i,j}^{N}\sin(\theta_i-\theta_j)\left(\delta_{ik}-\delta_{jk}\right) \\
&= \frac{\partial}{\partial \theta_k}\frac{1}{2N}\left[\sum_{j}^{N}\sin(\theta_k-\theta_j)-\sum_{i}^{N}\sin(\theta_i-\theta_k)\right] \\
&= \frac{1}{N}\sum_{i}^{N}\frac{\partial}{\partial \theta_k}\sin(\theta_k-\theta_i) \\
&= \frac{1}{N}\sum_{i}^{N}\cos(\theta_k-\theta_i)(1-\delta_{ik}) \\
&= \frac{1}{N}\sum_{i\neq k}^{N}\cos(\theta_k-\theta_i) \qquad (2.20)
\end{aligned}$$

Portanto, ignorando termos de ordem inferior a N que desaparecem no limite termodinâmico, a curvatura média de Ricci fica

$$k_R = \frac{1}{N-1}\frac{1}{N}\sum_{i,j}^{N}\cos(\theta_i-\theta_j). \qquad (2.21)$$

Da equação (2.18) podemos verificar que

$$\frac{1}{N}\sum_{i,j}^{N}\cos(\theta_i-\theta_j) = N-2U, \qquad (2.22)$$

sendo assim, podemos reescrever (2.21) como

$$k_R = \frac{N-2U}{N-1}. \qquad (2.23)$$

Na Ref. [24] Firpo utilizada as seguintes relações canônicas: $\langle U\rangle_c = -\partial_\beta \ln Z$ e $\langle (U-\langle U\rangle)^2\rangle_c = \partial_\beta^2 \ln Z$, para obter

$$\langle k_R\rangle_c = 1+\frac{2}{N}\partial_\beta \ln Z, \qquad (2.24)$$

$$\langle \delta^2 K_R\rangle_c \equiv \frac{1}{N}\langle (K_R-\langle K_R\rangle)^2\rangle_c = \frac{4}{N}\partial_\beta^2 \ln Z. \qquad (2.25)$$

junto da relação entre as flutuações de uma observável f nos *ensemble* microcanônico e

canônico [54],

$$\langle \delta^2 f \rangle_\mu = \langle \delta^2 f \rangle_c + \left(\frac{\partial \langle e \rangle_c}{\partial \beta} \right)^{-1} \left(\frac{\partial \langle f \rangle_c}{\partial \beta} \right)^2, \tag{2.26}$$

para obtenção de κ_0 e σ_κ. Aqui o cálculo é feito diretamente a partir de médias no *ensemble* microcanônico obtidas por Pearson *et al.* [55]:

$$\langle U \rangle_\mu = E - \frac{N}{2} T \tag{2.27}$$

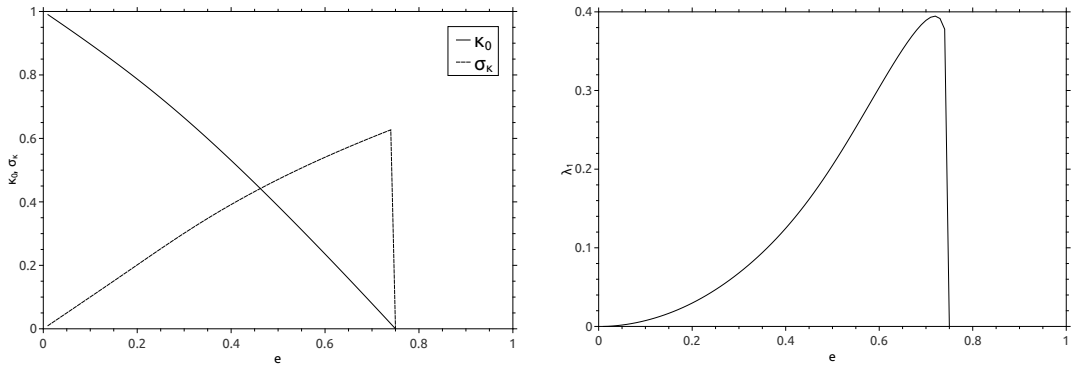
$$\langle (U - \langle U \rangle)^2 \rangle_\mu = \frac{N}{4} T^2 \left(2 - \frac{1}{c} \right), \tag{2.28}$$

onde E, T e c são a energia, temperatura e calor específico do sistema, respectivamente. Com isso,

$$\begin{aligned} \kappa_0 &= \langle k_R \rangle_\mu = \frac{N - 2\langle U \rangle_\mu}{N - 1} \\ &= 1 - 2e + T, \end{aligned} \tag{2.29}$$

$$\begin{aligned} \sigma_\kappa &= \sqrt{N \left(\langle k_R^2 \rangle_\mu - \langle k_R \rangle_\mu^2 \right)} \\ &= \sqrt{\frac{4}{N} \left(\langle U^2 \rangle_\mu - \langle U \rangle_\mu^2 \right)} \\ &= T \sqrt{2 - \frac{1}{c}}. \end{aligned} \tag{2.30}$$

Utilizando a temperatura e calor específico, temos o comportamento de κ_0 e σ_κ apresentados na Figura 2.3(a). Com as equações (2.15), (2.16) e (2.17), obtemos o gráfico de λ, Figura 2.3(b), de acordo com o resultado da Ref. [24]. Para o modelo cosHMF, Firpo encontrou um expoente crítico para λ, próximo da transição de segunda ordem em $e = 0.75$, com valor de $1/6$. Este resultado foi corroborado com simulações computacionais por Miranda *et al* [18], utilizando o método do Mapa Tangente.



(a) κ_0, σ_κ (b) λ

Figura 2.3: (a) Comportamento da média microcanônica da curvatura média de Ricci κ_0 e suas flutuações σ_κ para o cosHMF. (b) Comportamento do maior expoente de Lyapunov do cosHMF no limite $N \to \infty$.

Capítulo 3

Metodologia

Neste capítulo é apresentada a metodologia utilizada para o cálculo analítico e numerico do LLE para o modelo GHMF [56]. São apresentados os problemas encontrados em conjunto das soluções propostas para contornar esses obstáculos. Os códigos fonte desenvolvidos para este trabalho podem ser encontrados no Apêndice A.

3.1 Aproximação semi-analítica

Tendo como propósito obter λ para o GHMF através da Eq. (2.15), seguimos o mesmo procedimento apresentado na Seção 2.4 para o modelo cosHMF. Primeiramente, devemos encontrar a curvatura média k_R para então calcularmos κ_0 e σ_κ através de médias microcanônicas de k_R:

$$\kappa_0 = \langle k_R \rangle_\mu, \tag{3.1}$$

$$\sigma_\kappa^2 = N\left(\langle k_R^2 \rangle_\mu - \langle k_R \rangle_\mu^2\right). \tag{3.2}$$

Lembrando que k_R é obtido através do Laplaciano da energia potencial, Eq. (2.19), sendo que a energia potencial do GHMF é dada por

$$U = \frac{1}{2N}\sum_{i,j}^{N}[1 - \Delta\cos(\theta_i - \theta_j) - (1-\Delta)\cos(q\theta_i - q\theta_j)]. \tag{3.3}$$

Com isso,

$$
\begin{aligned}
\frac{\partial^2 U}{\partial\theta_k^2} &= \frac{\partial}{\partial\theta_k}\frac{1}{2N}\sum_{i,j}^{N}\frac{\partial}{\partial\theta_k}[1-\Delta\cos(\theta_i-\theta_j)-(1-\Delta)\cos(q\theta_i-q\theta_j)] \\
&= \frac{\partial}{\partial\theta_k}\frac{1}{2N}\sum_{i,j}^{N}[\Delta\sin(\theta_i-\theta_j)+(1-\Delta)q\sin(q\theta_i-q\theta_j)](\delta_{ik}-\delta_{jk}) \\
&= \frac{\partial}{\partial\theta_k}\frac{1}{2N}\left[\sum_{j}^{N}\Delta\sin(\theta_k-\theta_j)-\sum_{i}^{N}\Delta\sin(\theta_i-\theta_k)\right. \\
&\quad \left.+\sum_{j}^{N}(1-\Delta)q\sin(q\theta_k-q\theta_j)-\sum_{i}^{N}(1-\Delta)q\sin(q\theta_i-q\theta_k)\right] \\
&= \frac{1}{N}\sum_{i}^{N}\frac{\partial}{\partial\theta_k}\left[\Delta\sin(\theta_k-\theta_i)+(1-\Delta)q\sin(q\theta_k-q\theta_i)\right] \\
&= \frac{1}{N}\sum_{i}^{N}\left[\Delta\cos(\theta_k-\theta_i)+(1-\Delta)q^2\cos(q\theta_k-q\theta_i)\right](1-\delta_{ik}) \\
&= \frac{1}{N}\sum_{i\neq k}^{N}\left[\Delta\cos(\theta_k-\theta_i)+(1-\Delta)q^2\cos(q\theta_k-q\theta_i)\right].
\end{aligned}
$$

Temos então que, a menos de termos de ordem menor que N que desaparecem no limite termodinâmico, a curvatura média de Ricci k_R fica como

$$
\begin{aligned}
k_R &= \frac{1}{N-1}\frac{1}{N}\sum_{i\neq k}^{N}\left[\Delta\cos(\theta_k-\theta_i)+(1-\Delta)q^2\cos(q\theta_k-q\theta_i)\right] \\
&= \frac{N}{N-1}[\Delta m_1^2+(1-\Delta)q^2m_q^2]. \qquad (3.4)
\end{aligned}
$$

Para grandes valores de N, κ_0 pode ser aproximado por

$$
\begin{aligned}
\kappa_0 &= \langle k_R\rangle_\mu = \Delta\langle m_1^2\rangle_\mu+(1-\Delta)q^2\langle m_q^2\rangle_\mu. \\
&= \Delta m_1^{*2}+(1-\Delta)q^2m_q^{*2}, \qquad (3.5)
\end{aligned}
$$

onde recuperamos momentaneamente o símbolo $*$ das Eqs. (1.48)-(1.49) para diferenciarmos as magnetizações de equilíbrio das Eqs. (1.50)-(1.51) dos parâmetros de ordem definidos nas Eqs. (1.26)-(1.27). Os resultados da Ref. [24], obtidos por Firpo, para o modelo cosHMF são completamente recuperados aplicando $\Delta = 1$ na Eq. (3.5) e nas expressões subsequentes.

Em contraste com os resultados anteriores do cosHMF, o lado direito da Eq. (3.5) não pode, em geral, ser reescrito apenas como uma função da energia potencial do sistema $U = (N/2)\left[1 - \Delta\, m_1^2 - (1-\Delta)\, m_q^2\right]$ (devido ao q^2 multiplicando o segundo termo). Consequentemente, as aproximações (2.27)-(2.28) de [55] não podem ser extendidas diretamente para o caso atual. Para contornar essa dificuldade, determinamos o LLE a partir de (2.15) com κ_0 dado pela Eq. (3.5) e computamos σ_κ através de uma simulação de Monte Carlo Microcanônica (MMC, do inglês *Microcanonical Monte Carlo*) [56].

Em 1991, John R. Ray apresentou um método de Monte Carlo microcanônico [57], seguindo os mesmos princípios do método de Metropolis *et al.* [58], mas utilizando a densidade de probabilidade microcanônica obtida por Pearson *et al.* [55]:

$$\omega_E(\boldsymbol{q}) \propto (E - U(\boldsymbol{q}))^{\frac{dN}{2}-1}\, \Theta(E - U(\boldsymbol{q})), \tag{3.6}$$

com d a dimensão espacial do sistema, E sua energia e $\Theta(\,\cdot\,)$ a função de Heaviside. Podemos resumir o algoritmo de Ray nos seguintes passos:

− Definir uma energia E e uma condição inicial para o sistema, $\boldsymbol{q}$, com energia potencial $U(\boldsymbol{q}) < E$;

− Mudar aleatoriamente a posição de uma partícula, $\boldsymbol{q} \to \boldsymbol{q}'$, e computar a nova energia potencial $U(\boldsymbol{q}')$;

− Aceitar a nova configuração com a probabilidade:

$$A(\boldsymbol{q} \to \boldsymbol{q}') = min\left(1, \frac{\omega_E(\boldsymbol{q}')}{\omega_E(\boldsymbol{q})}\right). \tag{3.7}$$

com $\omega_E(\boldsymbol{q})$ dado pela Eq. (3.6). Caso a nova configuração seja aceita, a nova configuração $\boldsymbol{q}$ do sistema é $\boldsymbol{q}'$: $\boldsymbol{q} \leftarrow \boldsymbol{q}'$;

− Repetir os dois passos anteriores até um tempo T_0 onde o equilíbrio é atingido;

− Calcular as grandezas desejadas utilizando a distribuição atingida durante mais um tempo T.

Transformando o procedimento anterior em um pseudocódigo temo o Algoritmo 3.

Algorithm 3 Monte Carlo Microcanônico - Método de Ray

```
1: procedure
2:     do
3:         x ← initial_conditions()
4:     while U(x) > E
5:     for t ← 1 to T_0 do
6:         do
7:             x' ← change_state(x)
8:         while U(x') > E
9:         if A(x → x') > random_uniform(0, 1) then
10:            x ← x'
11:    for t ← 1 to T do
12:        calculate_averages()
```

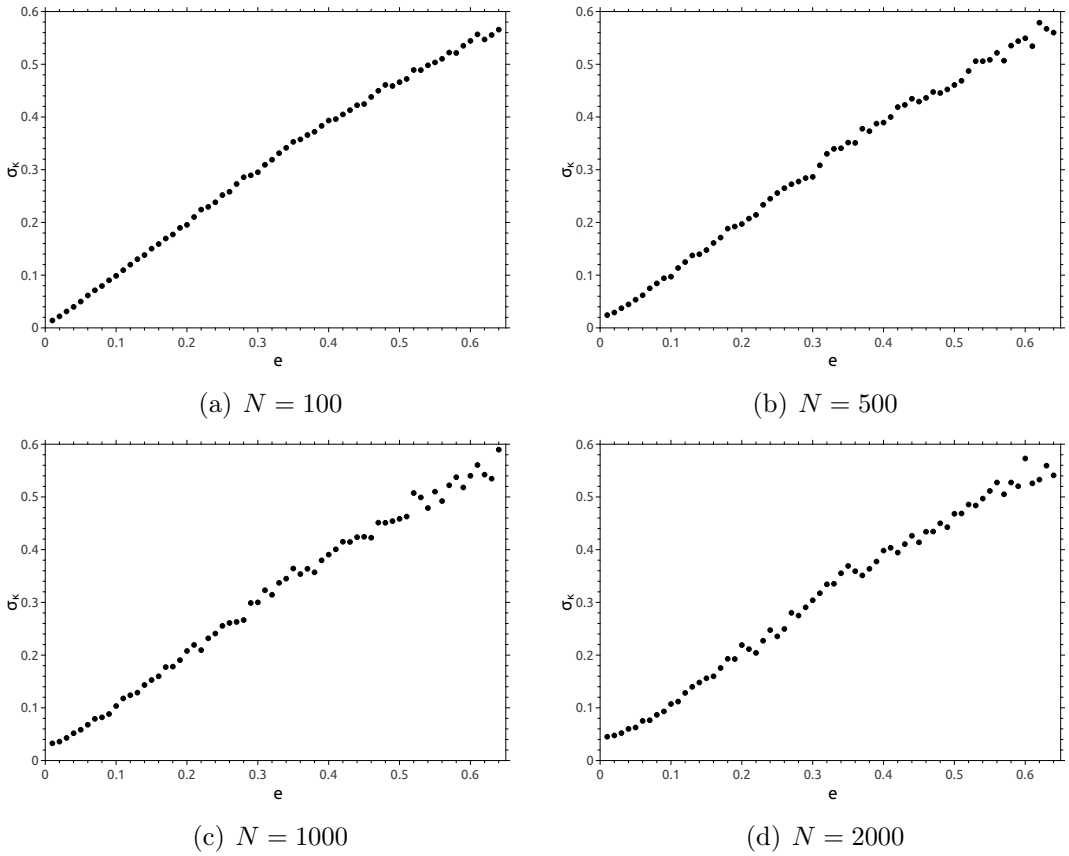
Desta forma, são gerados distribuições de $\mathbf{q}$ com probabilidades proporcionais a $(E - U(\mathbf{q}))^{\frac{dN}{2}-1}$, de onde é possível calcular todas as médias microcanônicas de grandezas macroscópicas relacionadas ao sistema isolado [59]. Isso permite que obtenhamos σ_κ^2 como

$$\sigma_\kappa^2 = N(\langle k_R^2 \rangle_\mu - \langle k_R \rangle_\mu^2). \tag{3.8}$$

3.2 Aumento de precisão via Redes Neurais

Apesar de conseguirmos σ_κ por meio das simulações MMC, ainda temos que lidar com os erros numéricos associado as essas simulações, que acabam se propagando para os valores finais do LLE, especialmente próximo das transições de fase, o que compromete o cálculo dos expoentes críticos. Podemos imaginar que simulações com um número elevado de partículas diminuiriam os erros, levando a uma maior precisão em σ_κ e, consequentemente, em λ. Mas como podemos ver na Figura 3.1, para os valores de N analisados, ocorre exatamente o contrário, o aumento de N leva a um maior erro numérico.

Para contornar essa limitação, aplicamos uma regressão não-linear nos resultados da simulação MMC para suavizar os erros numéricos associados [56]. Como a priori não sabemos qual a função que melhor se ajusta a σ_κ, nem se a mesma função trará bons resultados para qualquer valor de Δ, a solução foi utilizar um ajuste não-linear de uma Rede Neural Artificial (ANN) [60].



(a) $N = 100$ (b) $N = 500$

(c) $N = 1000$ (d) $N = 2000$

Figura 3.1: Resultado numérico de σ_κ em função da energia via simulação MMC para (a) $N = 100$, (b) $N = 500$, (c) $N = 1000$ e (d) $N = 2000$

Redes neurais podem ser utilizadas para diversos propósitos, inclusive regressões não-lineares. Na Figura 3.2 temos o esboço de uma rede neural genérica, onde podemos identificar três componentes essenciais: a camada de entrada, as camadas ocultas e a camada de saída. Cada camada contém "neurônios" que recebem informação das camadas anteriores e passam para as camadas posteriores, onde cada conexão entre neurônios possui um peso w_{ij}^l, onde l representa o número da camada com i e j sendo os números dos neurônios associados as camadas l e $l+1$, respectivamente. Além disso, cada neurônio, com exceção dos da primeira camada, possui um viés b_i^l. Um número que passa de um neurônio para outro é multiplicado pelo peso da conexão entre eles, depois é somado ao resultado o viés do neurônio destino, por fim, para quebrar a linearidade, o resultado é aplicado em um função de ativação não-linear, como uma tangente hipoerbólica ou sigmóide, então passado para a camada seguinte.

No caso de um problema de regressão, a ANN é alimentada pelos valores das variáveis independentes X, que neste caso são os valores de energia e, onde ao final esperamos que a rede nos dê os valores das variáveis dependentes y, no nosso caso σ_κ.

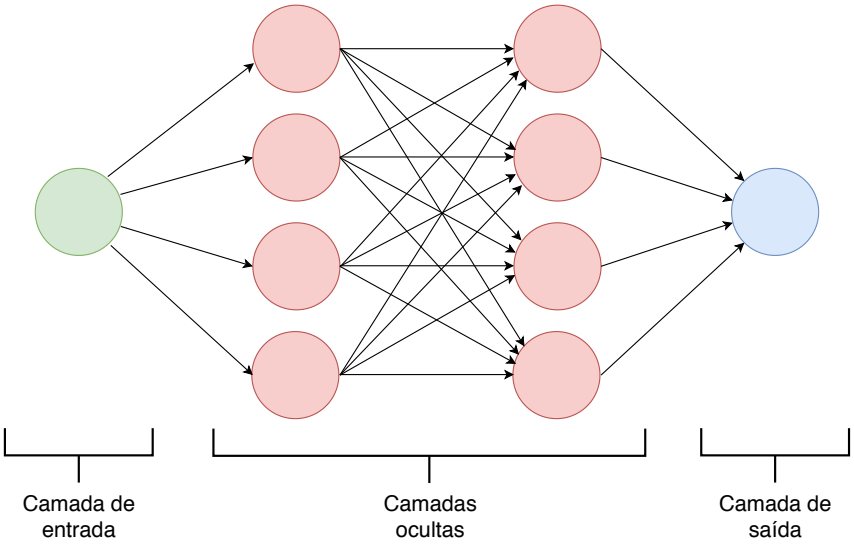


Figura 3.2: Esboço de uma Rede Neural Artificial.

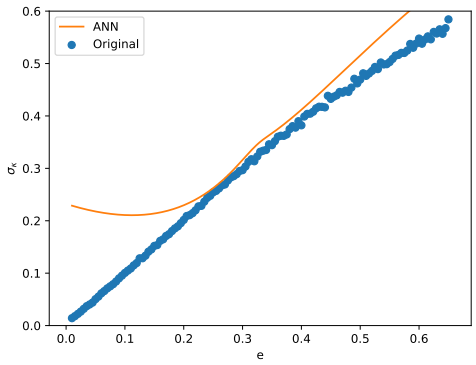
Para tal, precisamos minimizar o erro, também chamado de função custo. Existem diversas formas de definir a função custo, as mais comuns são o Erro Quadrático Médio (MSE, do inglês *Mean Squared Error*)

$$C_{MSE}(w_{ij}^l, b_i^l) = \frac{1}{n}\sum_{k=1}^{n}(y_k - \hat{y}_k)^2 \tag{3.9}$$

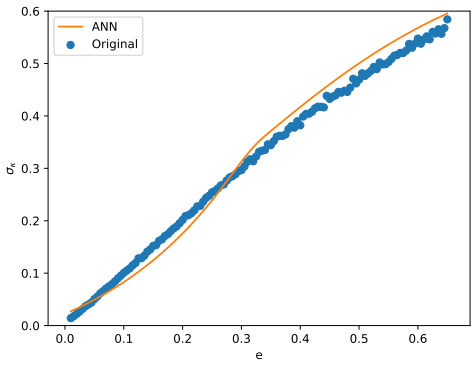
e o Erro Absoluto Médio (MAE, do inglês *Mean Absolute Error*)

$$C_{MAE}(w_{ij}^l, b_i^l) = \frac{1}{n}\sum_{k=1}^{n}|y_k - \hat{y}_k|, \tag{3.10}$$

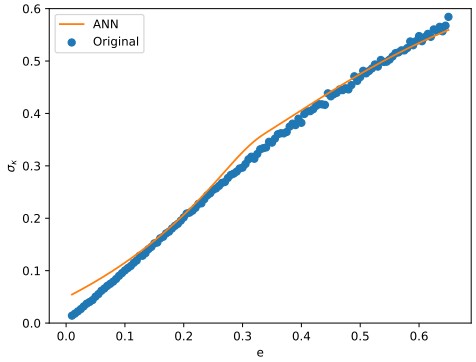
onde y são os valores reais, $\hat{y}$ os valores de saída da rede neural e n o número de pontos passados para a rede. Das Eqs. (3.9)-(3.10), vemos que o erro é função de todos os pesos w_{ij}^l e vieses b_i^l da rede. Portanto, o treinamento da rede neural consiste em encontrar os pesos e vieses que minimizam a função custo associada. Quando os dados passam por completo pela rede neural chamamos isso de uma época. Ao final de cada época atualizamos os pesos e vieses a partir da função custo. Existem diversas formas de realizar essa atualização, sendo um dos métodos mais difundidos o de retropropagação. A ideia desse método é atualizar os pesos e vieses no sentido oposto do gradiente da função custo [61] para irmos em direção ao mínimo da função. Espera-se que ao passar das épocas a função custo seja minimizada e temos então a rede neural treinada, Figura 3.3.



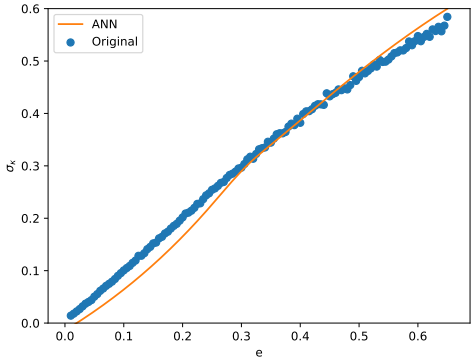
(a) Época = 1



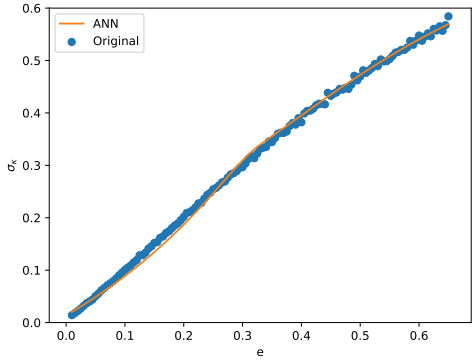
(b) Época = 5



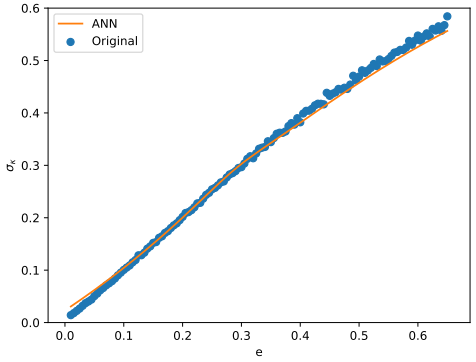
(c) Época = 10



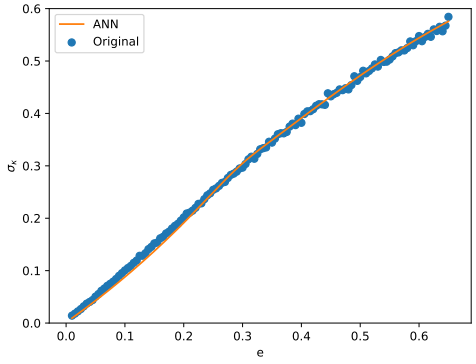
(d) Época = 15



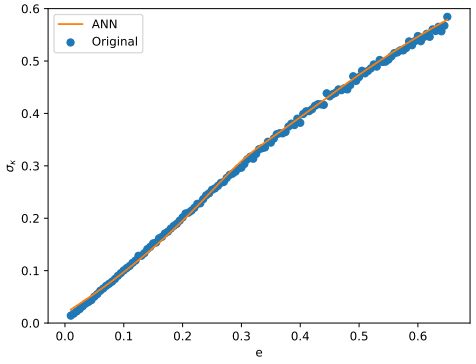
(e) Época = 20



(f) Época = 25



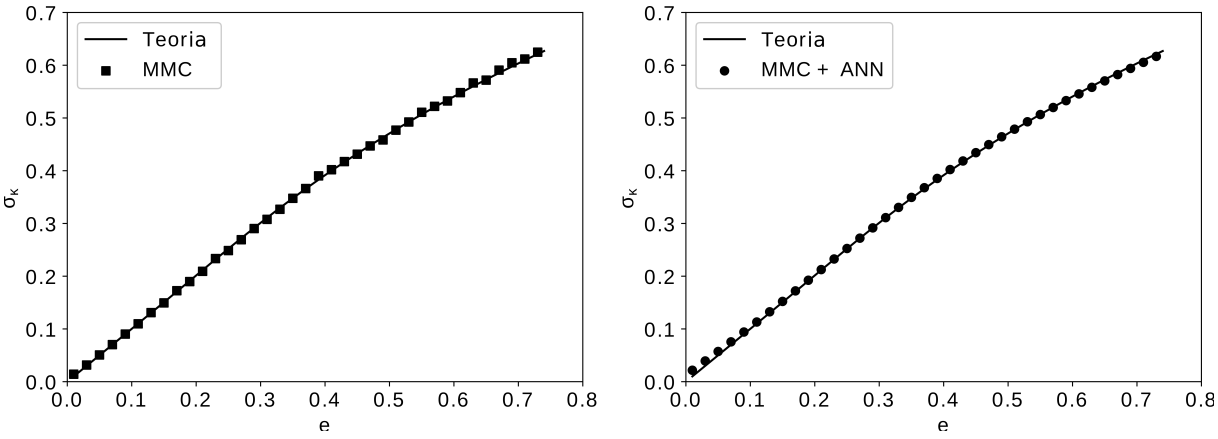
(g) Época = 30



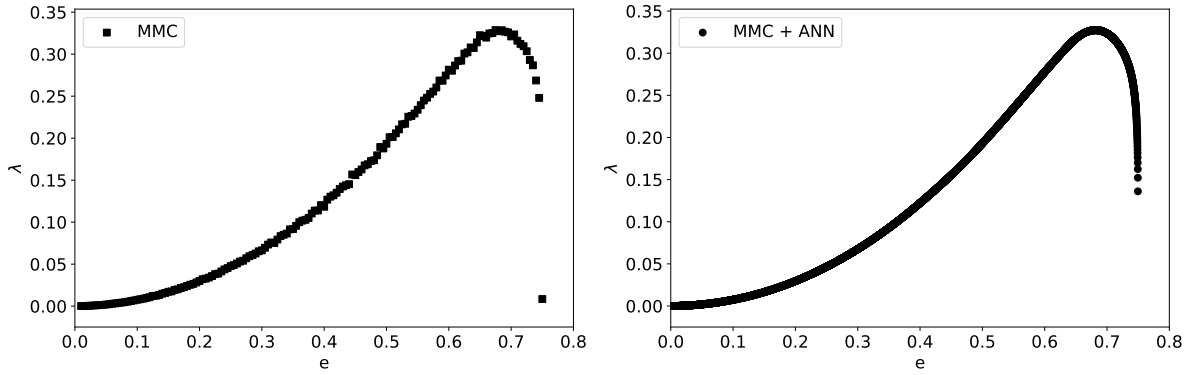
(h) Época = 35

Figura 3.3: Ajuste da rede neural em função das épocas.

A arquitetura da rede neural aplicada neste trabalho é de 4 camadas ocultas com 32 neurônios cada, onde cada neurônio utiliza uma função de ativação ELU (do inglês *Exponential Linear Unit*) [62]. A rede neural é treinada até verificarmos que não há mais diminuição significativa da função custo após certo número de épocas. A validade dessa aproximação é evidenciada na Figura 3.4, mostrando o resultado do processo de suavização via ANN para o GHMF com $\Delta = 1$, comparada com a previsão teórica correspondente, Figura 3.4(a).



(a) σ_κ



(b) λ

Figura 3.4: Resultado da suavização via ANN para o GHMF para $\Delta = 1$ e $N = 10^2$, onde na simulação MMC $N = 100$. A esquerda temos os resultados obtidos apenas pela simulação MMC e a direita após a suavização via ANN. (a) σ_κ. (b) λ.

A suavização da rede neural reproduz corretamente os valores de κ_0 enquanto reduz as oscilações do erro numérico, que é refletido diretamente no resultado final de λ, principalmente próximo a transição de fase, onde temos maior interesse. Para o GHMF com $\Delta = 1$ a previsão teórica do expoente crítico associado a λ é de 1/6 [24], assim podemos verificar a validade da nossa abordagem semi-analítica através da previsão desse expoente crítico, Tabela 3.1.

Analítico	MMC	MMC + ANN
0.166	0.131	0.160

Tabela 3.1: Previsões teóricas para o expoente crítico de λ do GHMF para $\Delta = 1$.

Após a correção da ANN nos resultados da simulação, o erro da estimativa fica em torno de 3%, enquanto que sem o ajuste da ANN temos um erro de mais de 20%. Vale salientar que na simulação MMC foram utilizadas apenas 100 partículas para calcular σ_κ, mostrando que com a suavização da ANN podemos realizar simulações com menos partículas e, consequentemente, mais rápidas.

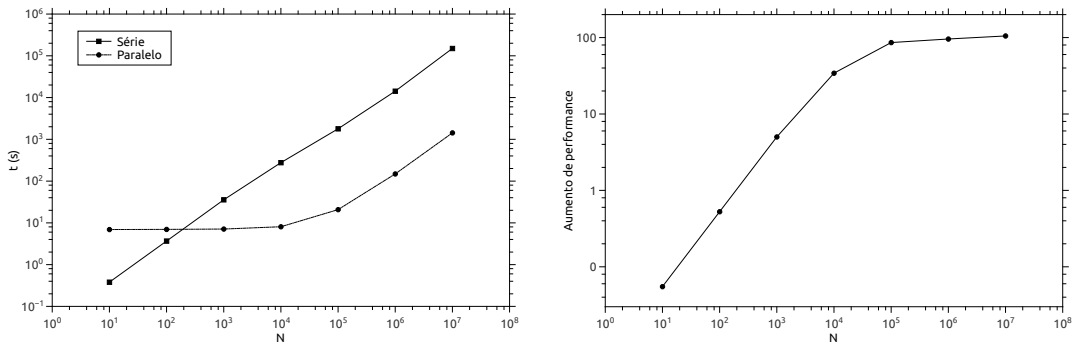
3.3 Otimização da simulação dinâmica

Tendo em vista a verificação da previsão analítica, devemos chegar o mais próximo possível em nossas simulações dinâmicas do limite termodinâmico. Em outras palavras, quanto maior o número de partículas nas simulações melhor. Para atingirmos tal objetivo, os códigos desenvolvidos usam programação em paralelo através de placas gráficas, GPU, utilizando a linguagem de programação CUDA da NVIDIA. Apesar dos processadores contidos nas CPU serem mais rápidos que os das GPU, a programação em paralelo tem o poder de agilizar os cálculos ao dividir a quantidade de operações matemáticas a serem realizadas entre diversos processadores. Na Tabela 3.2 temos os tempos de execução dos programas em série e em paralelo junto do aumento de performance, onde todos os parâmetros da simulação de dinâmica molecular foram fixados e apenas o número de partículas varia.

Número de	Tempo (s)		Aumento de
Partículas	Série	Paralelo	Performance
10^1	0.364	7.656	0.048
10^2	3.656	7.700	0.475
10^3	32.268	7.952	4.059
10^4	222.192	9.208	24.130
10^5	1525.640	24.084	63.347
10^6	14052.236	168.552	83.370
10^7	137946.664	1628.308	84.718

Tabela 3.2: Tempos de execução dos programas.

Os resultados da Tabela 3.2 estão na Figura 3.5 para visualizarmos melhor o efeito da paralelização.



(a) (b)

Figura 3.5: Resultados da paralelização do algoritmo computacional. (a) Comparação de tempo de execução. (b) Aumento de performance.

É notável que a paralelização para simulações com um número reduzido de partículas não há melhora na performance, muito pelo contrário. Sempre que paralelizamos algum procedimento, devemos fazer um gerenciamento de dados entre as memórias da GPU e RAM, além de reintegrar os resultados de cada processador individual. Por conta disso, quando há poucas partículas perdemos performance, uma vez que o cálculo em paralelo não compensa o tempo perdido com gerenciamento de memória e reintegração. Mas quando passamos de $N = 10^3$ começamos a observar a importância da paralelização do processamento. A performance fica cada vez maior a medida que N cresce chegando a uma otimização de até 85 vezes. A Figura 3.5(b), nos indica que existe um limite do quanto podemos melhorar a performance, afinal de contas a GPU também possui um limite de processadores.

Com um programa de alta performance é possível obter o espectro de Lyapunov de um sistema composto de várias partículas. Na Figura 3.6 temos o espectro do GHMF para $N = 300$ e $\Delta = 1$, o que implica na integração simultânea de 601 sistemas, Algoritmo 1. Uma verificação importante em sistemas hamiltonianos é a de que cada expoente positivo é acompanhado de um negativo de mesmo módulo para que haja conservação do volume no espaço de fases, ou seja,

$$\sum_{i=1}^{2N} \lambda_i = 0. \tag{3.11}$$

Outra grandeza importante é a entropia de Kolmogorov-Sinai, que possui informações gerais do sistema, além de também ser usada na caracterização de regimes caóticos. Para sistemas fechados e isolados a entropia de Kolmogorov-Sinai coincide com a soma dos expoentes de Lyapunov positivos [63].

$$S_{ks} = \sum_{\lambda_i > 0} \lambda_i. \tag{3.12}$$

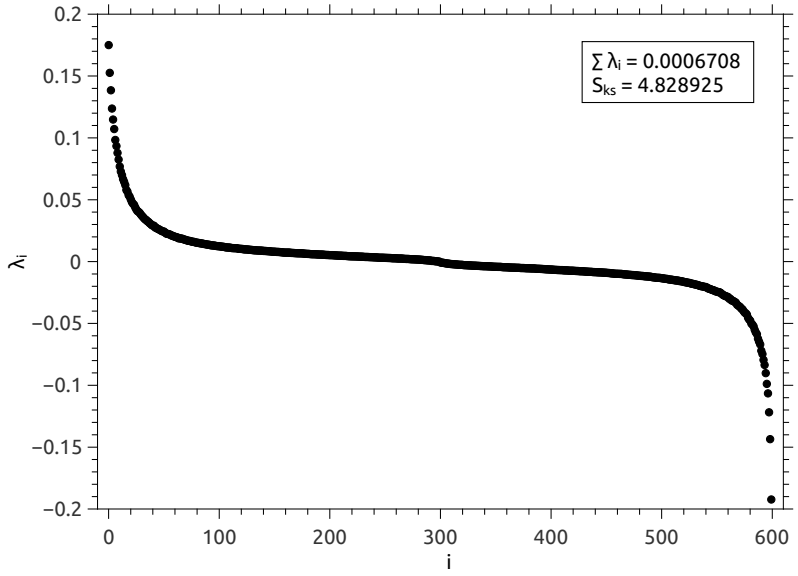


Figura 3.6: Espectro de Lyapunov do cosHMF com 300 partículas.

O resultado serve como demonstração do potencial alcançado com a paralelização através confirmação da equação (3.12), mas como não é o foco deste trabalho a análise do espectro de Lyapunov do modelo GHMF, o resultado não foi extendido para outros valores de Δ.

Apesar da paralelização nos poupar tempo nos cálculos proporcionais a N, como os de energia e força, não é possível paralelizar a dinâmica em si, ou seja, a evolução temporal. O que no caso de sistemas com interações de longo alcance é um grande problema, pois como já discutido na Seção 1.3, esses sistemas passam muito tempo em estados quasi-estacionários antes de atingir o equilíbrio termodinâmico. Uma forma inteligente de evitar a perda de horas de simulação, apenas para esperar o sistema entrar em equilíbrio, é escolher uma condição inicial que já esteja no equilíbrio, ou muito próxima dele. Sabemos que a distribuição dos momentos das partículas no equilíbrio deve seguir uma gaussiana. Então, utilizando a mecânica estatística de Boltzmann-Gibbs, podemos gerar uma condição inicial gaussiana de momentos que satisfaça a temperatura de equilíbrio do sistema. Para isso precisamos apenas do valor da temperatura de equilíbrio para uma energia desejada, que pode ser obtida tanto via simulação de Monte Carlo ou analiticamente. Felizmente, o modelo GHMF possui solução analítica de onde podemos obter essas temperaturas. Entretando, ainda resta o problema da distribuição de posição das partículas, que neste caso foi obtida através de simulações MMC, Algoritmo 3. Dada uma energia e e um

número de partículas N, executadamos uma simulação MMC até que o sistema atinja o equilíbrio térmico, para então salvar a posição de cada partícula em um arquivo que é utilizado como condição inicial para as posições das partículas na simulação dinâmica. Na Figura 3.7 podemos ver a curtose dos momentos, temperatura e magnetizações para o modelo GHMF com $e = 0.69$, $\Delta = 1$, $q = 2$ e $N = 2 \times 10^4$, tanto com uma condição inicial de *waterbag* quanto partindo do resultado da simulação MMC.

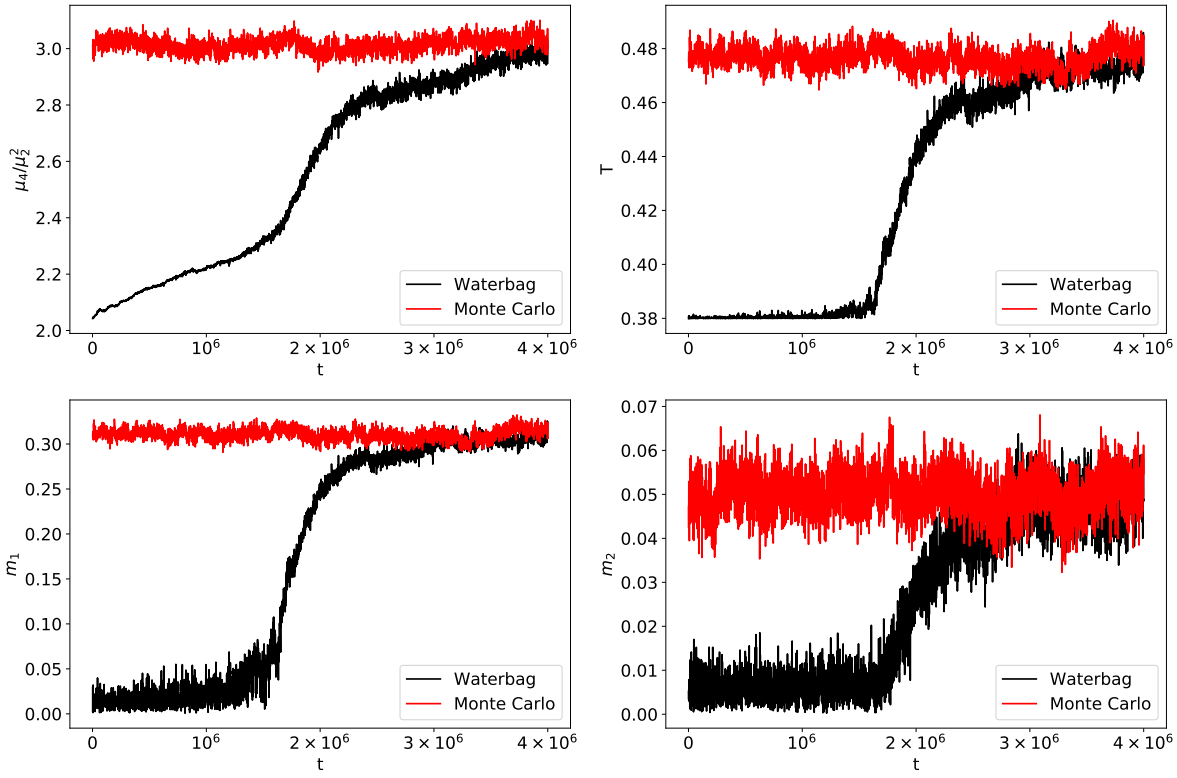


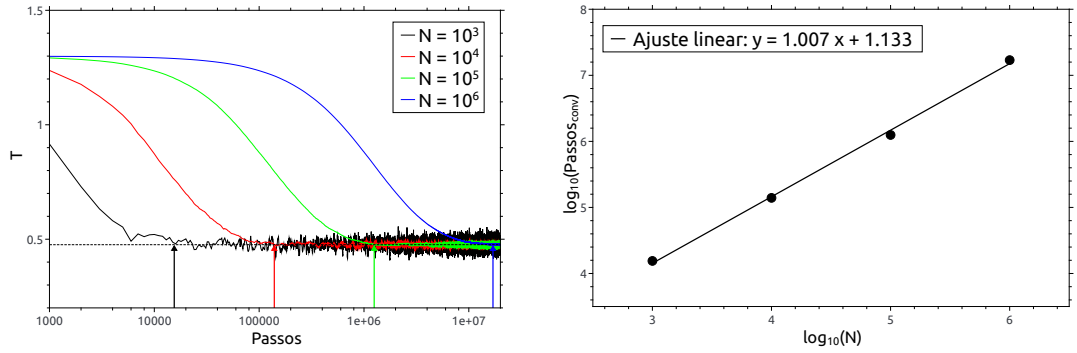
Figura 3.7: Evolução do GHMF para $e = 0.69$, $\Delta = 1$, $q = 2$ e $N = 2 \times 10^4$ com condição inicial *waterbag* e MMC. (Superior esquerdo) Curtose dos momentos. (Superior direito) Temperatura. (Inferior esquerdo) Magnetização m_1. (Inferior direito) Magnetização m_2.

O fato da curtose dos momentos sempre oscilar em torno de 3 (valor esperado de uma distribuição gaussiana) e de que a temperatura e magnetizações oscilam em torno dos seus valores de equilíbrio, deixam evidente que essa estratégia funciona bem para evitarmos o tempo de termalização de sistemas com interações de longo alcance em nossas simulações dinâmicas. Uma vez que a simulações de Monte Carlo tendem a ser muito mais rápidas, pois atinge o equilíbrio através da estatística e não da dinâmica do sistema.

3.4 Estimativa da condição inicial

Com a metodologia descrita na Seção 3.3 temos um ganho de tempo considerável, mas devemos lembrar que o que fazemos é substituir uma simulação dinâmica por uma simulação de Monte Carlo para levar o sistema a um estado de equilíbrio, que também fica mais demorada quando aumentamos N. A real vantagem está no fato do tempo de convergência da simulação de Monte Carlo escalar com N ao invés de N^2, Figura 3.8. Portanto, ainda temos o problema de tempo de espera até o equilíbrio, mas agora para um N muito maior.

Como o que nos interessa é uma configuração de partículas que satisfaça a distribuição de equilíbrio, podemos estimar a forma dessa distribuição com uma simulação MMC para um número menor de partículas e então realizar uma amostragem, baseada nessa distribuição, para o número N desejado.



(a) (b)

Figura 3.8: (a) Temperatura do GHMF para $e = 0.69$, $\Delta = 1$, $q = 2$ e $N = 10^3, 10^4, 10^5, 10^6$ em função do número de passos da simulação MMC. A linha tracejada marca a temperatura de equilíbrio e as setas indicam quando a temperatura a atingiu. (b) Número de passos até o equilíbrio da simulação MMC em função de N.

Em estatística, a Estimativa de Densidade Kernel (KDE, do inglês *Kernel Density Estimator*) é uma forma de estimar a densidade de probabilidade de uma variável aleatória de maneira não-paramétrica [64, 65]. Seja $(x_1, x_2, \ldots, x_n)$ uma amostra de uma variável aleatória independente com um densidade f desconhecida. A densidade estimada $\hat{f}$ pode ser feita através de:

$$\hat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right), \tag{3.13}$$

onde K é um kernel qualquer, como uma função gaussiana, e $h > 0$ é um parâmetro de distância chamado largura de banda. Se a densidade é de base gaussiana, a regra de

Silverman [66] nos diz que a escolha ideal para h é

$$h = \left(\frac{4\hat{\sigma}^5}{3n}\right)^{1/5} \approx 1.06\hat{\sigma}n^{-1/5}, \tag{3.14}$$

onde $\hat{\sigma}$ é o desvio padrão da amostra. A KDE possui uma relação com a construção de histogramas, a diferença é que para o histograma o eixo das abscissas é dividido em intervalos e sempre que um ponto cai dentro de um intervalo colocamos uma "caixa" de altura 1. Se mais pontos caem dentro do mesmo intervalo, as caixas são empilhadas uma em cima das outras. Para o cálculo de KDE, ao invés de uma caixa de altura 1, utilizamos um kernel gaussiano com média dada pela variável aleatória e desvio padrão h, Eq. (3.14). Podemos ver a diferença de suavidade da KDE em relação ao histograma pela Figura 3.9, onde ambos foram construídos com um conjunto de dados $x = [-2.1,\ -1.3,\ -0.4,\ 1.9,\ 5.1,\ 6.2]$. Isso faz com que as estimativas via KDE tenham uma convergência mais rápida em direção a verdadeira densidade para variáveis aleatórias contínuas [67].

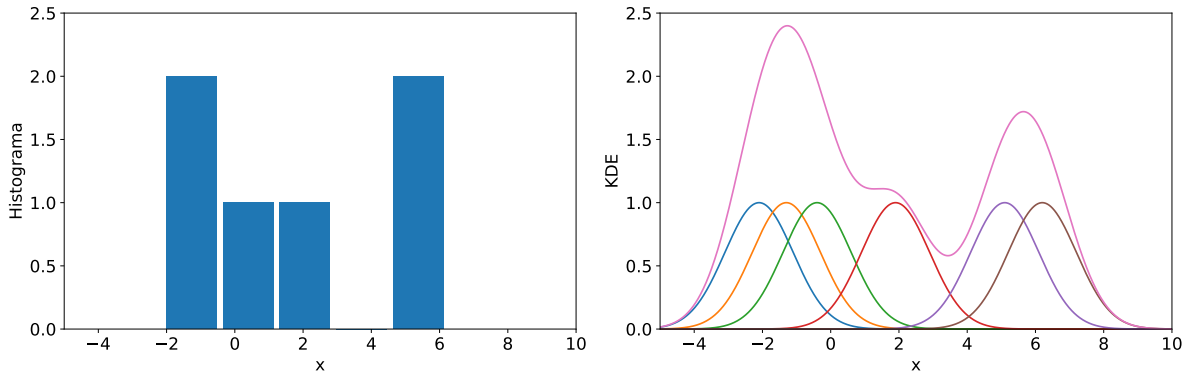


Figura 3.9: (Esquerda) Histograma. (Direita) KDE.

Com a aplicação da KDE nas posições de N_0 partículas em uma configuração de equilíbrio, gerada por uma simulação MMC, conseguimos estimar a densidade de probabilidade da posição, para então fazer uma amostragem com um número $N > N_0$ de partículas com a mesma distribuição inicial. Como o tempo de convergência escala proporcional a N, o aumento de performance ao definir a condição inicial do sistema com KDE é proporcional a N/N_0. Na Figura 3.10, temos o exemplo da distribuição original e estimada com $N/N_0 = 20$. Mas o mais importante é o fato de que utilizando como condição inicial a distribuição gerada pela KDE temos o mesmo efeito de deixar o sistema próximo do equilíbrio, Figura 3.11.

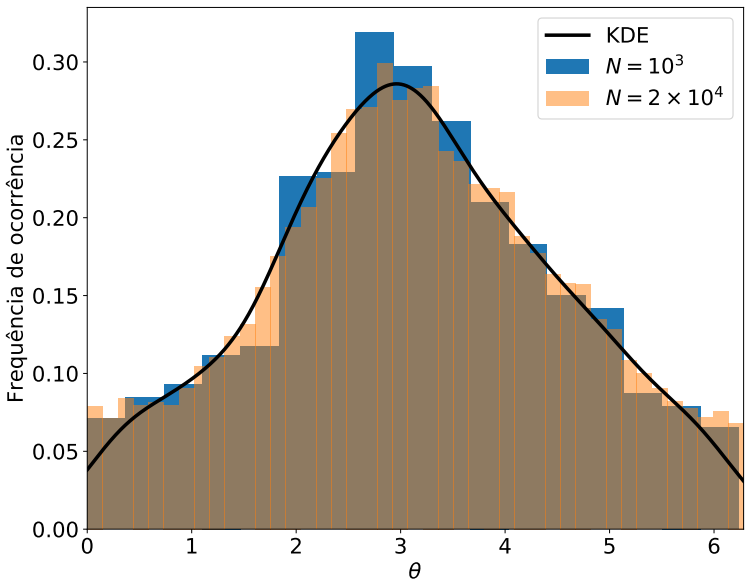


Figura 3.10: Comparação da distribuição original de posições das N_0 partículas, obtidas com simulação MMC, com a distribuição gerada após KDE da distribuição original para $N = 20 \times N_0$ partículas. A linha sólida é a densidade de probabilidade estimada.

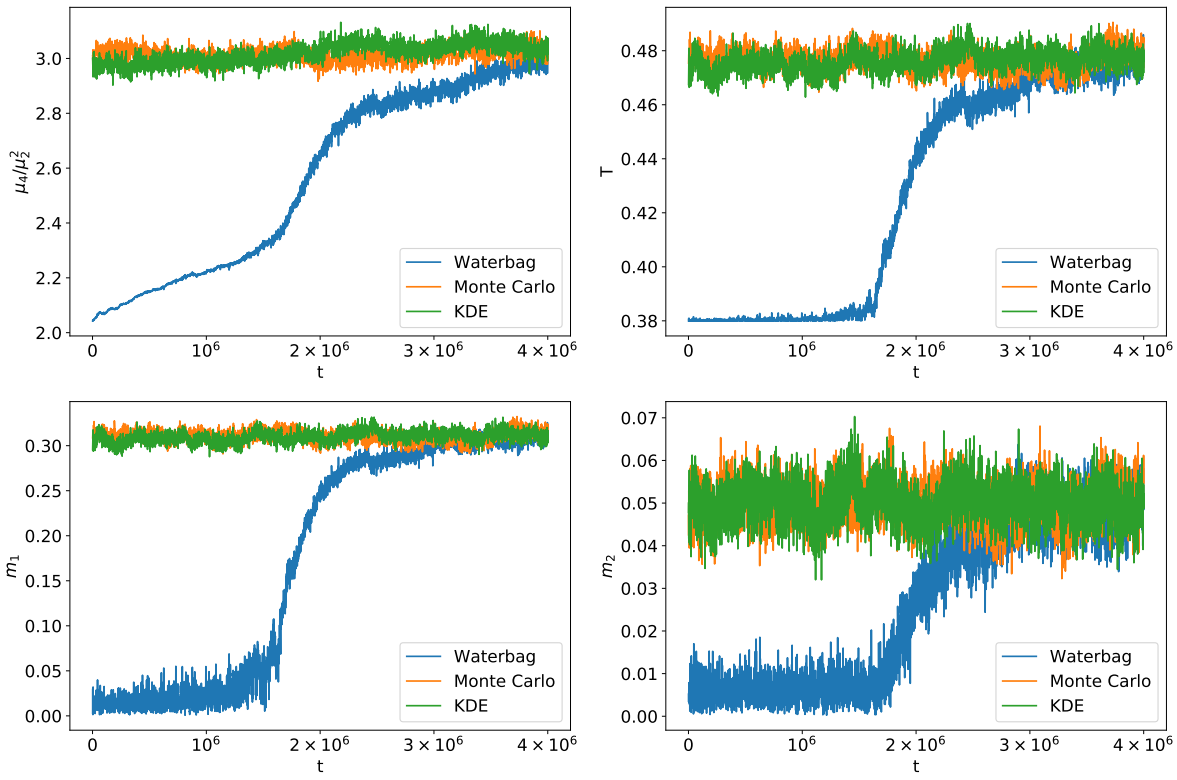


Figura 3.11: Evolução do GHMF para $e = 0.69$, $\Delta = 1$, $q = 2$ e $N = 2 \times 10^4$ com condição inicial *waterbag*, MMC e KDE. (Superior esquerdo) Curtose dos momentos. (Superior direito) Temperatura. (Inferior esquerdo) Magnetização m_1. (Inferior direito) Magnetização m_2.

Independente do método adotado para as condições iniciais, chegamos nos mesmos resultados para λ com o método de Mapa Tangente, Tabela 3.3. Monstrando que podemos melhorar a performance sem perder precisão nos resultados finais.

Condição Inicial	$N = 10^3$	$N = 10^4$	$N = 10^5$
Waterbag	0.182	0.160	0.142
Monte Carlo	0.183	0.160	0.142
KDE	0.183	0.160	0.142

Tabela 3.3: LLE obtido dinamicamente para $e = 0.55$ e $N = 10^3,\ 10^4,\ 10^5$ com diferentes condições iniciais.

Capítulo 4

Resultados e Discussões

Neste capítulo veremos as previsões teóricas vinda da aproximação semi-analítica e os resultados atingidos do maior expoente de Lyapunov através de simulações dinâmicas. Além de disso, expandimos a análise do expoente crítico de λ para transições de primeira e segunda ordem em busca de uma possível universalidade.

4.1 Previsões teóricas

Na Seção 3.1 é discutido que a curvatura média de Ricci k_R para o GHMF, não pode ser expressa como uma função da energia potencial do sistema, como feito no trabalho de Firpo [24]. Entretanto, para alguns valores de Δ isso é sim possível, como o caso $\Delta = 1$ (cosHMF) e também quando $\Delta = 0$ [56]. Temos então que k_R como uma função da energia potencial para $\Delta = 0$ fica

$$k_R = q^2 m_q^2 = q^2 \frac{N - 2U}{N - 1}, \tag{4.1}$$

o que nos leva a

$$\begin{aligned} \kappa_0 &= q^2 m_q^2 \\ \sigma_\kappa &= q^2 T \sqrt{2 - \frac{1}{c}}, \end{aligned} \tag{4.2}$$

que é a mesma estrutura de quando $\Delta = 1$, Eqs (2.29)-(2.30), com a adição do termo multiplicativo q^2. Portanto, se tivermos

$$\begin{aligned} m_q^{(\Delta=0)} &= m_1^{(\Delta=1)} \\ T^{(\Delta=0)} &= T^{(\Delta=1)}, \end{aligned} \tag{4.3}$$

podemos dizer que

$$\begin{aligned} \kappa_0^{(\Delta=0)} &= q^2 \kappa_0^{(\Delta=1)} \\ \sigma_\kappa^{(\Delta=0)} &= q^2 \sigma_\kappa^{(\Delta=1)}. \end{aligned} \tag{4.4}$$

Na Figura 4.1, temos a verificação visual da validade das Eqs. (4.3) para $q = 2$. Com isso, podemos aplicar o resultado (4.4) nas Eqs. (2.15)-(2.17) para concluir que nesse caso [56]

$$\lambda^{(\Delta=0)} = 2\lambda^{(\Delta=1)}. \tag{4.5}$$

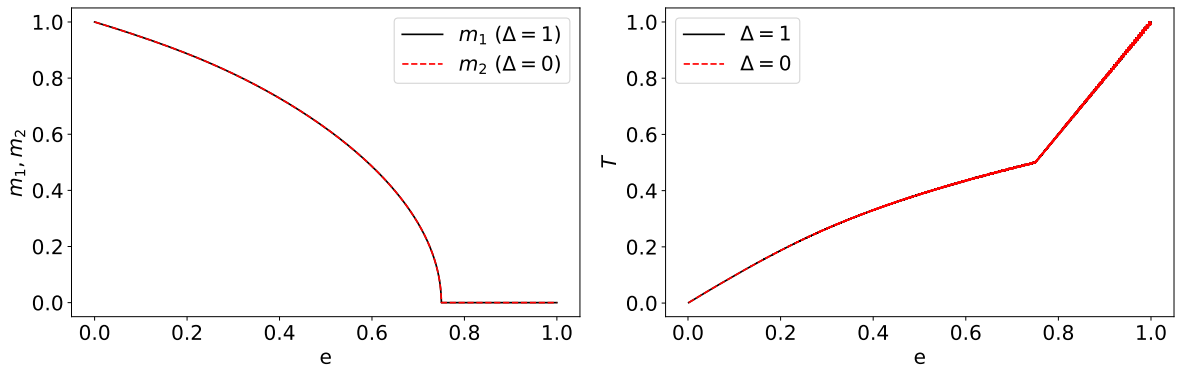


Figura 4.1: Comparação das magnetizações e temperatura para $\Delta = 0, 1$. (Esquerda) $m_2^{(\Delta=0)}$ e $m_1^{(\Delta=1)}$. (Direita) $T^{(\Delta=0)}$ e $T^{(\Delta=1)}$.

A partir das Eqs. (2.15)-(2.17), com κ_0 vindo de (3.5) e σ_κ calculado através da metodologia descrita na Seção 3.1, encontramos a previsão teórica do maior expoente de Lyapunov λ para o modelo GHMF, com $q = 2$ e $\Delta = 0, 0.35, 0.5$ e 1, Figura 4.2. A escolha desses valores de Δ é justificada pelo diagrama de fase do modelo, Figura 1.6, onde podemos encontrar 4 tipos de transições de fase diferentes: nemática-paramagnética de segunda ordem ($\Delta = 0, 0.35$), ferromagnética-nemática de segunda ordem ($\Delta = 0.35$), ferromagnética-paramagnética de primeira ordem ($\Delta = 0.5$) e ferromagnética-paramagnética de segunda ordem ($\Delta = 1$).

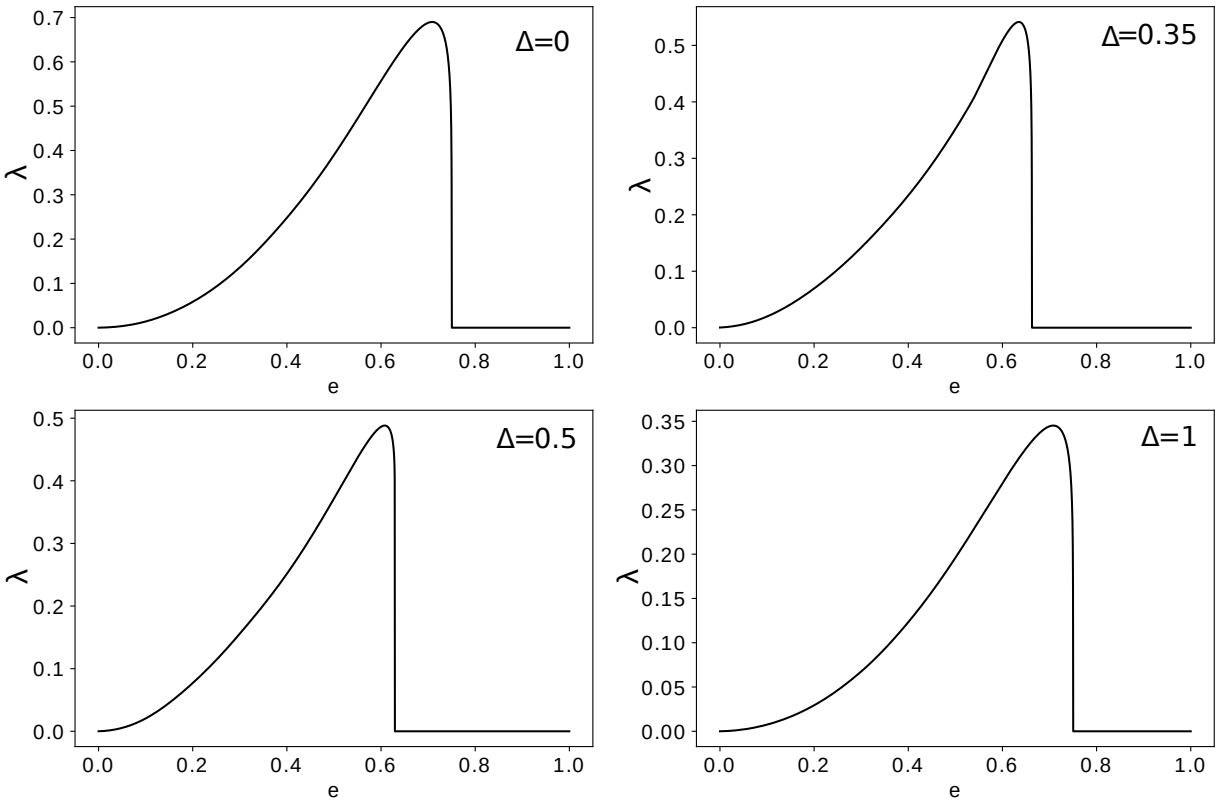


Figura 4.2: LLE para $\Delta = 0$ (superior esquerdo), $\Delta = 0.35$ (superior direito), $\Delta = 0.5$ (inferior esquerdo) e $\Delta = 1$ (inferior direito) em função da energia.

Para todos os valores de Δ escolhidos observamos o mesmo formato da curva de λ em função da energia, semelhante ao do modelo cosHMF. A valor do LLE tem seu máximo próximo da energia crítica onde cai abruptamente para zero ao atingir a fase paramagnética. Diferente das curvas de magnetização e temperatura, onde podemos identificar as transições de fase ferromagnética-nemática e nemática-paramagnética quando $\Delta = 0.35$, aqui não notamos nenhum efeito distinto em λ na transição ferromagnética-nemática. Uma outra verificação interessante está na Figura 4.3. Vemos que assim como previsto na Eq. (4.5), o LLE correspondente a $\Delta = 1$ é metade de quando $\Delta = 0$, com exceção dos valores de baixas energias que provavelmente decorrem de possíveis flutuações numéricas.

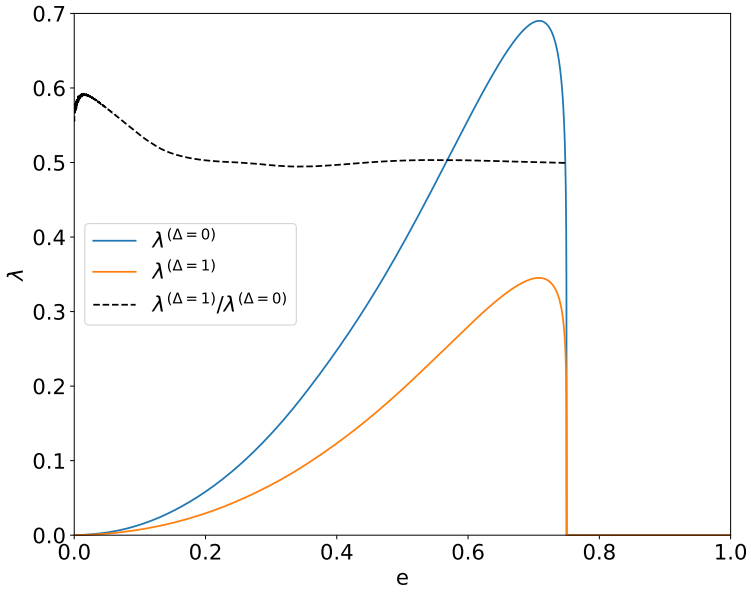


Figura 4.3: LLE para $\Delta = 0,\ 1$ e razão $\lambda^{(\Delta=1)}/\lambda^{(\Delta=0)}$.

4.2 Resultados numéricos

Uma vez obtidas as previsões teóricas do comportamento do LLE, o próximo passo é comparar com resultados obtidos através da dinâmica do sistema. Os valores de λ são obtidos através do Algoritmo 2. Para o modelo GHMF, as equações de Hamilton são:

$$\begin{aligned}
\dot{\theta}_i &= p_i, \\
\dot{p_i} &= \Delta\left(m_{1y}\cos\theta_i - m_{1x}\sin\theta_i\right) \\
&\quad + (1-\Delta)q\left[m_{qy}\cos(q\theta_i) - m_{qx}\sin(q\theta_i)\right].
\end{aligned} \tag{4.6}$$

As equações linearizadas em torno da solução (θ_i^*, p_i^*) das Eqs. (4.6) ficam

$$\begin{aligned}
\delta\dot{\theta}_i &= \delta p_i, \\
\delta\dot{p}_i &= \Delta\left(\delta m_{1y}\cos\theta_i^* - \delta m_{1x}\sin\theta_i^*\right) \\
&\quad - \Delta\left(m_{1y}^*\sin\theta_i^* + m_{1x}^*\cos\theta_i^*\right)\delta\theta_i \\
&\quad + (1-\Delta)q\left[\delta m_{qy}\cos(q\theta_i^*) - \delta m_{qx}\sin(q\theta_i^*)\right] \\
&\quad - (1-\Delta)q^2\left[m_{qy}^*\sin(q\theta_i^*) + m_{qx}^*\cos(q\theta_i^*)\right]\delta\theta_i,
\end{aligned} \tag{4.7}$$

onde m^*_{1x}, m^*_{1y}, m^*_{qx} e m^*_{qy} são as componentes das magnetizações computadas em θ^*_i e

$$\begin{aligned}
\delta m_{1x} &\equiv -\frac{1}{N}\sum_{j=1}^{N}\delta\theta_j \sin\theta^*_j, \\
\delta m_{1y} &\equiv \frac{1}{N}\sum_{j=1}^{N}\delta\theta_j \cos\theta^*_j, \\
\delta m_{qx} &\equiv -\frac{q}{N}\sum_{j=1}^{N}\delta\theta_j \sin(q\theta^*_j), \\
\delta m_{qy} &\equiv \frac{q}{N}\sum_{j=1}^{N}\delta\theta_j \cos(q\theta^*_j).
\end{aligned} \tag{4.8}$$

As simulações de dinâmica molecular foram implementadas para executar na GPU em paralelo, permitindo simulações com grandes valores de N [18, 68] e estimativas precisas para os expoentes de Lyapunov. A Figura 4.4 mostra os resultados de λ em função do tempo de integração para $N = 10^5$, $e = 0.5$ e $\Delta = 0, 0.35, 0.5$ e 1, com uma boa convergência para um tempo total de $T_f = 10^5$ [56].

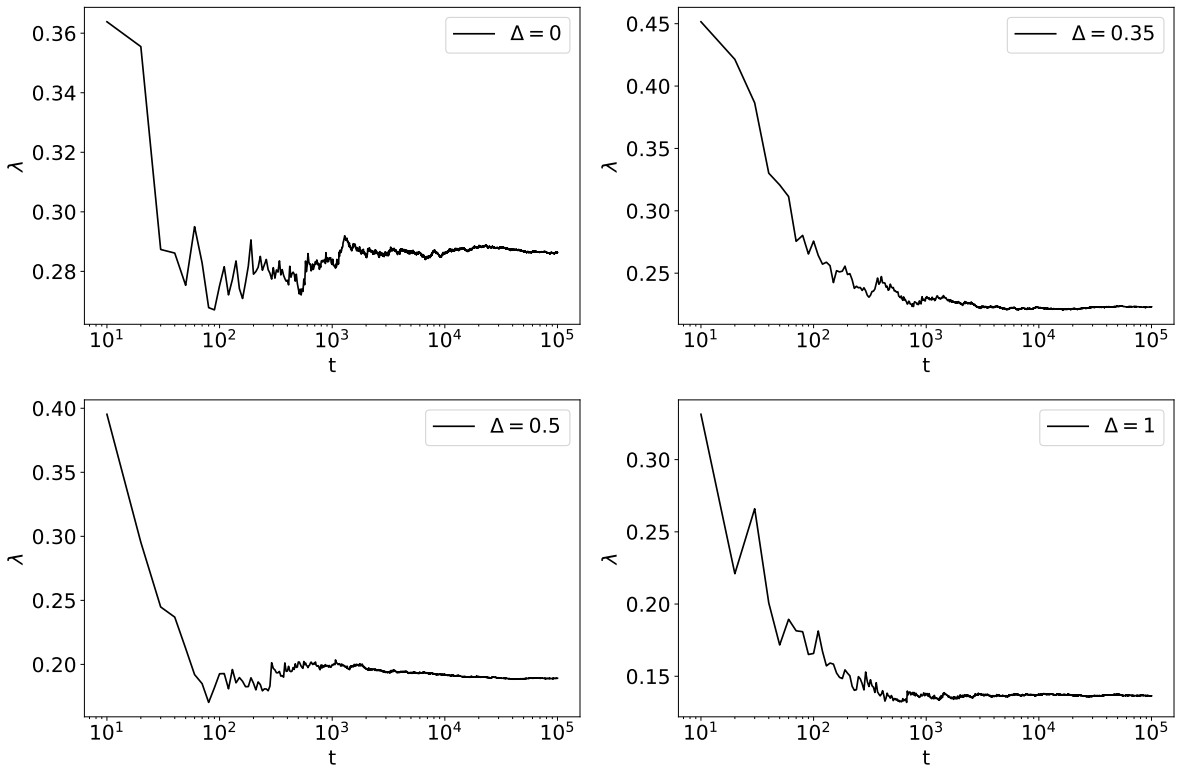


Figura 4.4: LLE para $N = 10^5$, $e = 0.5$ e $\Delta = 0$ (superior esquerdo), $\Delta = 0.35$ (superior direito), $\Delta = 0.5$ (inferior esquerdo) e $\Delta = 1$ (inferior direito) em função do tempo de integração. Passo do tempo de integração é $\Delta t = 0.05$.

O tempo total de integração, que é o tempo em que λ é computado, é contabilizado apenas a partir do momento que o sistema encontra-se em equilíbrio. Como discutido na Seção 1.3, sistemas com interações de longo alcance possuem tempos de relaxação para o equilíbrio muito longos [1, 25, 26, 31], mas com a metodologia apresentada na Seção 3.4 podemos preparar o sistema em condições iniciais bem próximas do estado de equilíbrio. Então deixamos o sistema evoluir por um intervalo t_0, apenas para garantir a termalização antes de iniciar o cálculo das grandezas de interesse.

Para garantir que o sistema está de fato no estado correto de equilíbrio, são computadas grandezas termodinâmicas, como temperatura e magnetizações, com objetivo de checar seus valores com as respectivas previsões teóricas. As grandezas termodinâmicas computadas no início do cálculo de λ, estão nas Figuras 4.5 e 4.6 junto das curvas teóricas, mostrando que o sistema encontra-se em equilíbrio quando λ começa a ser calculado.

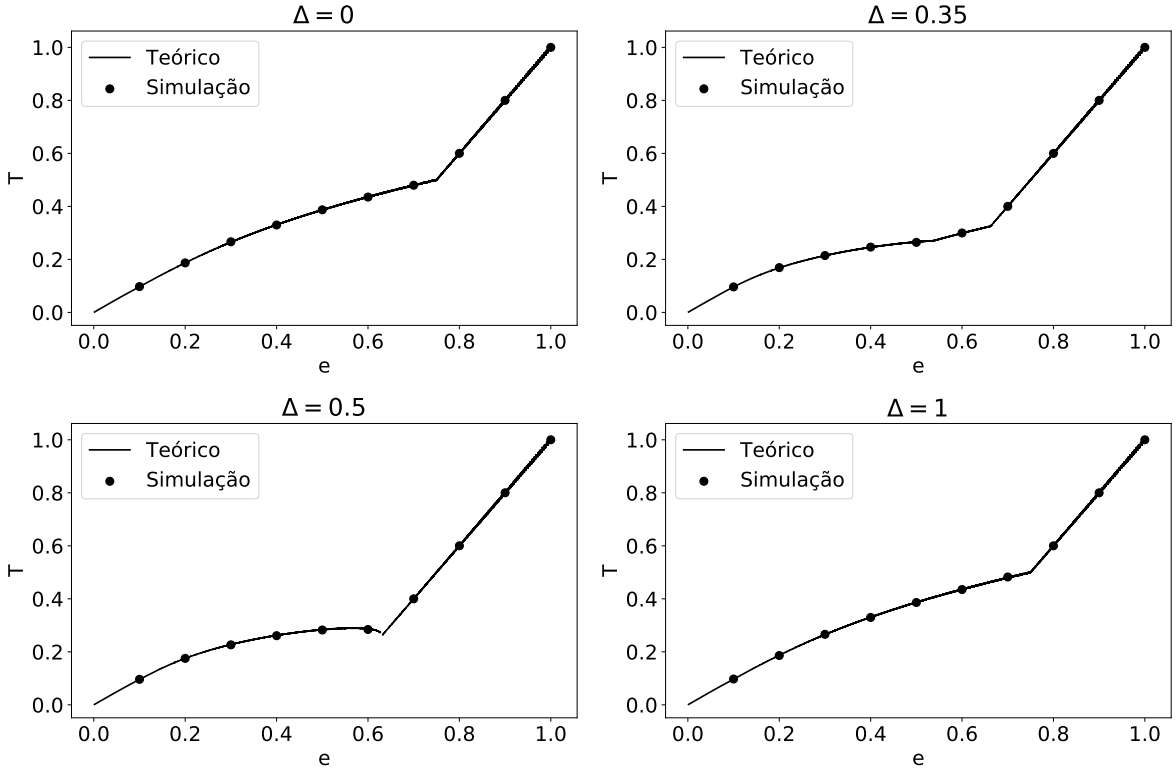


Figura 4.5: Temperatura com $N = 10^5$ e $t_0 = 10^4$ para $\Delta = 0$ (superior esquerdo), $\Delta = 0.35$ (superior direito), $\Delta = 0.5$ (infeior esquerdo) e $\Delta = 1$ (inferior direito) em função da energia. As barras de erro são menores que os tamanhos dos símbolos.

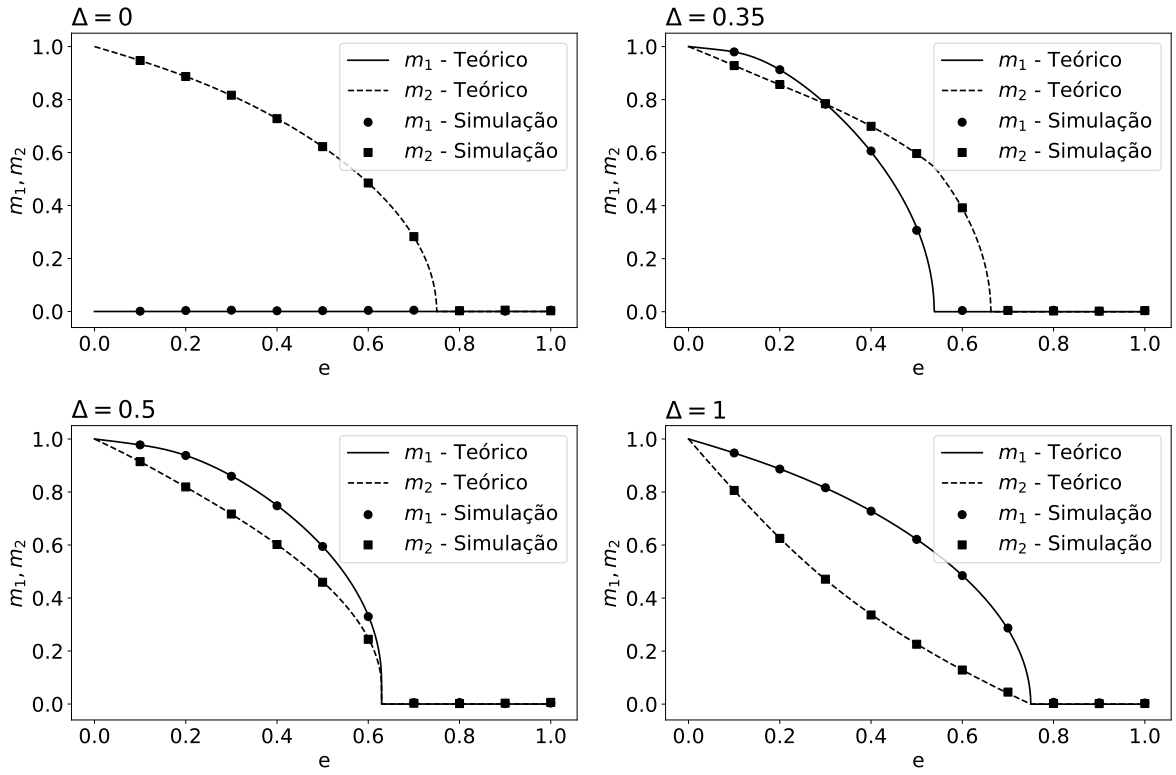


Figura 4.6: Magnetizações com $N = 10^5$ e $t_0 = 10^4$ para $\Delta = 0$ (superior esquerdo), $\Delta = 0.35$ (superior direito), $\Delta = 0.5$ (infeior esquerdo) e $\Delta = 1$ (inferior direito) em função da energia. As barras de erro são menores que os tamanhos dos símbolos.

Os resultados pelo método de Mapa Tangente foram obtidos para $N = 10^5$, $\epsilon = 10^{-6}$ e tempo total de simulação $T_f = 10^5$, com renormalização do vetor diferença $\boldsymbol{w}$ em intervalos de tempo $T_{norm} = 10$. O LLE em função da energia por partícula para os mesmos parâmetros usados na Seção 4.1 estão na Figura 4.7, em conjunto das respectivas estimativas analíticas. Similarmente do que foi mostrado para o modelo cosHMF na Ref. [18], a dependência do LLE com a energia difere significamente da sua estimativa (semi-)analítica. Uma possível explicação vem do fato de que uma das suposições usadas na aproximação analítica é de que as flutuações são δ-correlacionadas, o que espera-se que seja válido apenas a altas energias [18, 20, 21].

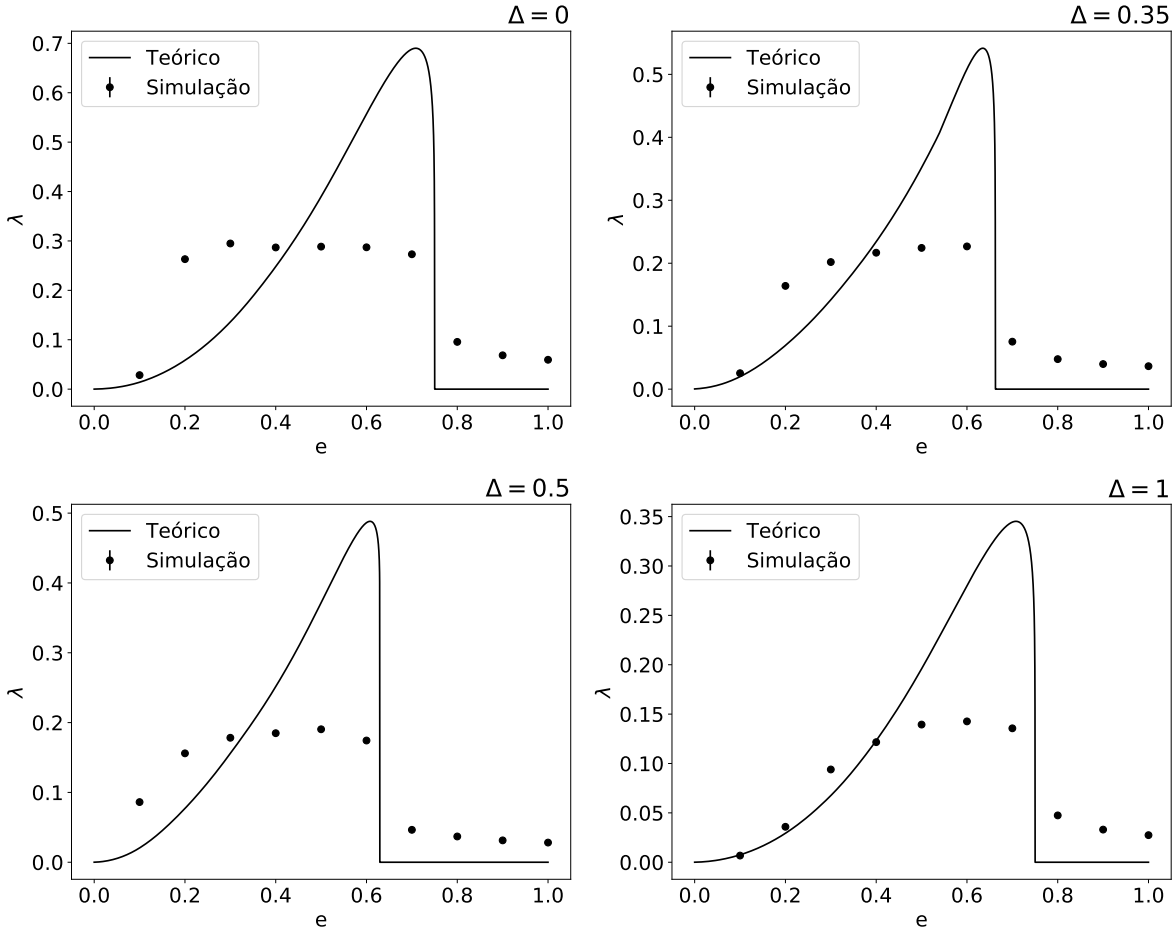


Figura 4.7: LLE com método de Mapa Tangente com $N = 10^5$ e previsões teóricas correspondentes para $\Delta = 0$ (superior esquerdo), $\Delta = 0.35$ (superior direito), $\Delta = 0.5$ (infeior esquerdo) e $\Delta = 1$ (inferior direito) em função da energia. As barras de erro são menores que os tamanhos dos símbolos.

4.3 Expoentes críticos - Analíticos

Um dos questionamentos motivadores ao desenvolvimento deste trabalho é se o valor 1/6 para o expoente crítico do LLE, encontrado por Firpo [24] para o cosHMF, é universal para sistemas de campo médio. Além disso, verificar se também existe um expoente para as transições de primeira ordem e qual seu valor. Como temos a relação entre $\lambda^{(\Delta=0)}$ e $\lambda^{(\Delta=1)}$, podemos estimar teoricamente qual deve ser o valor do expoente crítico para $\lambda^{(\Delta=0)}$ a partir do expoente de $\lambda^{(\Delta=1)}$. Assumindo que, próximo as transições de fase, $\lambda^{(\Delta=0)}$ e $\lambda^{(\Delta=1)}$ obedecem uma lei de escala da forma:

$$\begin{aligned} \lambda^{(\Delta=0)} &\propto |e - e_{\rm c}|^{\xi_0}, \\ \lambda^{(\Delta=1)} &\propto |e - e_{\rm c}|^{\xi_1}, \end{aligned} \tag{4.9}$$

com ξ_0 e ξ_1 os expoentes críticos para $\Delta = 0$ e $\Delta = 1$, respectivamente, obtemos da Eq. (4.5) que [56]

$$\xi_0 = \xi_1. \tag{4.10}$$

Também podemos esperar valores diferentes do expoente crítico ξ para transições de fase de primeira e segunda ordem, observando que k_R como indicado na Eq. (3.4) depende apenas dos parâmetros de ordem do sistema e, consequentemente, os expoentes críticos associados as magnetizações pertinentes à transição de fase determinam os valores dos expoentes críticos do LLE. Os expoentes críticos das magnetizações nas transições de fase podem ser determinados diretamente da solução analítica do modelo em equilíbrio, dada na Ref. [26], como representado na Figura 4.6. Como m_1 e m_2 têm expoentes críticos distintos, próximo à transição de fase, o maior termo na Eq. (3.4) (aquele com o menor expoente crítico) determina o valor de ξ. Para transições de segunda ordem (casos com $\Delta = 0,\ 0.35,\ 1$), o termo dominante (m_1 para $\Delta = 1$ e m_2 para $\Delta = 0,\ 0.35$) tem um expoente crítico de 1/2. Por outro lado, os expoentes críticos de m_1 e m_2 na transição de primeira ordem são aproximadamente 0.150. Portanto, diferentes valores de ξ nas transições de primeira e segunda ordem são uma consequência do comportamento distinto dos parâmetros de ordem correspondentes [56].

Para aplicar a Eq. (2.15), observamos que os expoentes críticos de κ_0 são 1, de acordo com a Eq. (3.5), para as transições de fase de segunda ordem, mas próximo de 0.3 para o caso de primeira ordem (consultar a Figura 4.8). Além disso, σ_κ tem um expoente muito menor que κ_0, que pode ser aproximado de zero quando substituído na Eq. (2.15). Então $\tau \approx \sqrt{\kappa_0}/\sigma_\kappa$ por Eq. (2.17), e Eq. (2.16) se reduz a $\Lambda^3 \approx 4\sigma_\kappa^2\tau \approx 4\sigma_\kappa\sqrt{\kappa_0}$, e Eq. (2.15) para [56]

$$\lambda \approx (\sigma_\kappa/2)^{1/3}\kappa_0^{1/6}\,. \tag{4.11}$$

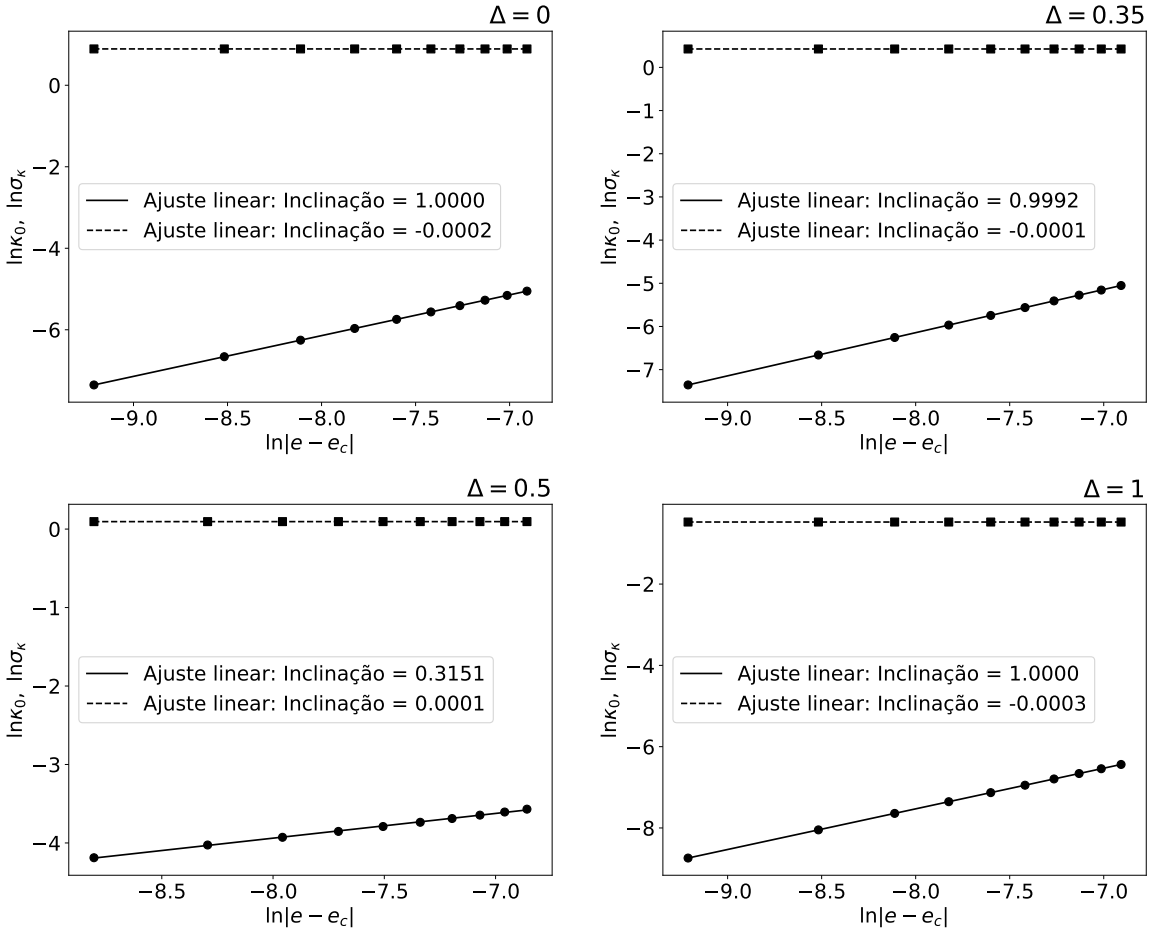


Figura 4.8: Estimativas analíticas de κ_0 (linha sólida) e σ_κ (linha tracejada) próximas as transições de fases para $\Delta = 0$ (superior esquerdo), $\Delta = 0.35$ (superior direito), $\Delta = 0.5$ (inferior esquerdo) e $\Delta = 1$ (inferior direito).

Para estimarmos os expoentes críticos utilizamos os valores de λ próximos da energia crítica e_c e para investigar se eles seguem uma lei de potência da forma $\lambda \propto |e - e_{\rm c}|^{\xi}$, reescrevemos

$$\ln \lambda = \xi \ln |e - e_{\rm c}| + C, \tag{4.12}$$

com C uma constante, com isso podemos realizar uma regressão linear onde a inclinação da reta é o expoente crítico associado a λ. A Figura 4.9 mostra o gráfico log-log para os mesmos dados da Figura 4.2 próximos da transição de fase. Como teste de consistência, observamos que Eq. (4.10) é satisfeita. Mais importante, os expoentes críticos associados a transição de segunda ordem ($\Delta = 0,\ 0.35,\ 1$) estão todos bem próximos do valor 1/6 vindo das estimativas analíticas.

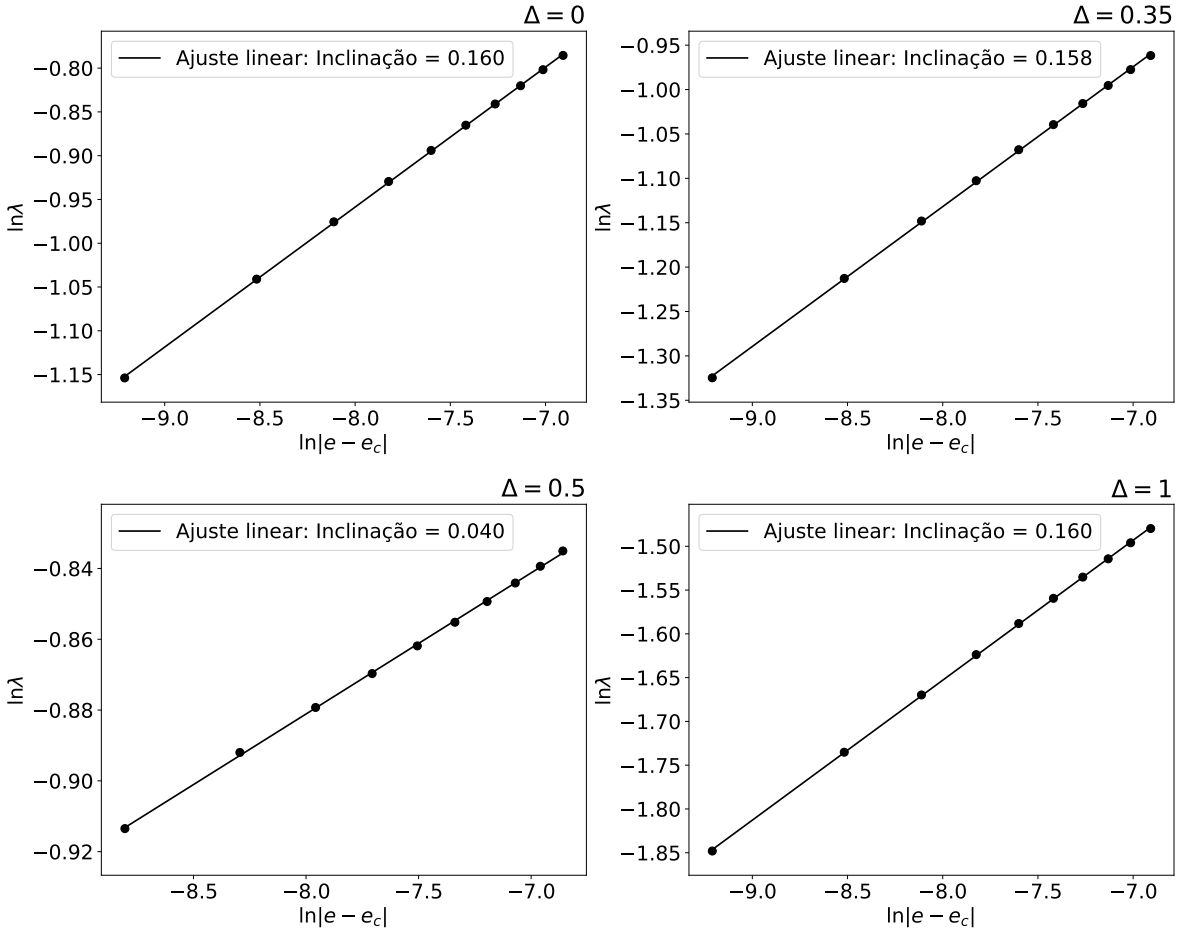


Figura 4.9: LLE próximo da transição de fase para $\Delta = 0$ (superior esquerdo), $\Delta = 0.35$ (superior direito), $\Delta = 0.5$ (inferior esquerdo) e $\Delta = 1$ (inferior direito).

Para o caso $\Delta = 0.5$, transição de primeira ordem, o LLE também obedece uma lei de potência similar, mas com um expoente diferente $\xi = 0.040$. Estimativas analíticas próximas ao valor $\Delta = 0.5$ estão na Figura 4.10, com os valores do expoente da lei de potência variando de 0.039 até 0.048, longe dos valores da transição contínua mas de acordo com a Eq. (4.11). Também foi investigado o expoente crítico ξ com a variação de Δ para visualizarmos como ξ varia quando mudamos de uma transição de segunda ordem para uma de primeira. Os pontos tricríticos ocorrem em $\Delta \approx 0.545,\ e \approx 0.636$ e $\Delta \approx 0.477,\ e \approx 0.628$ [49].

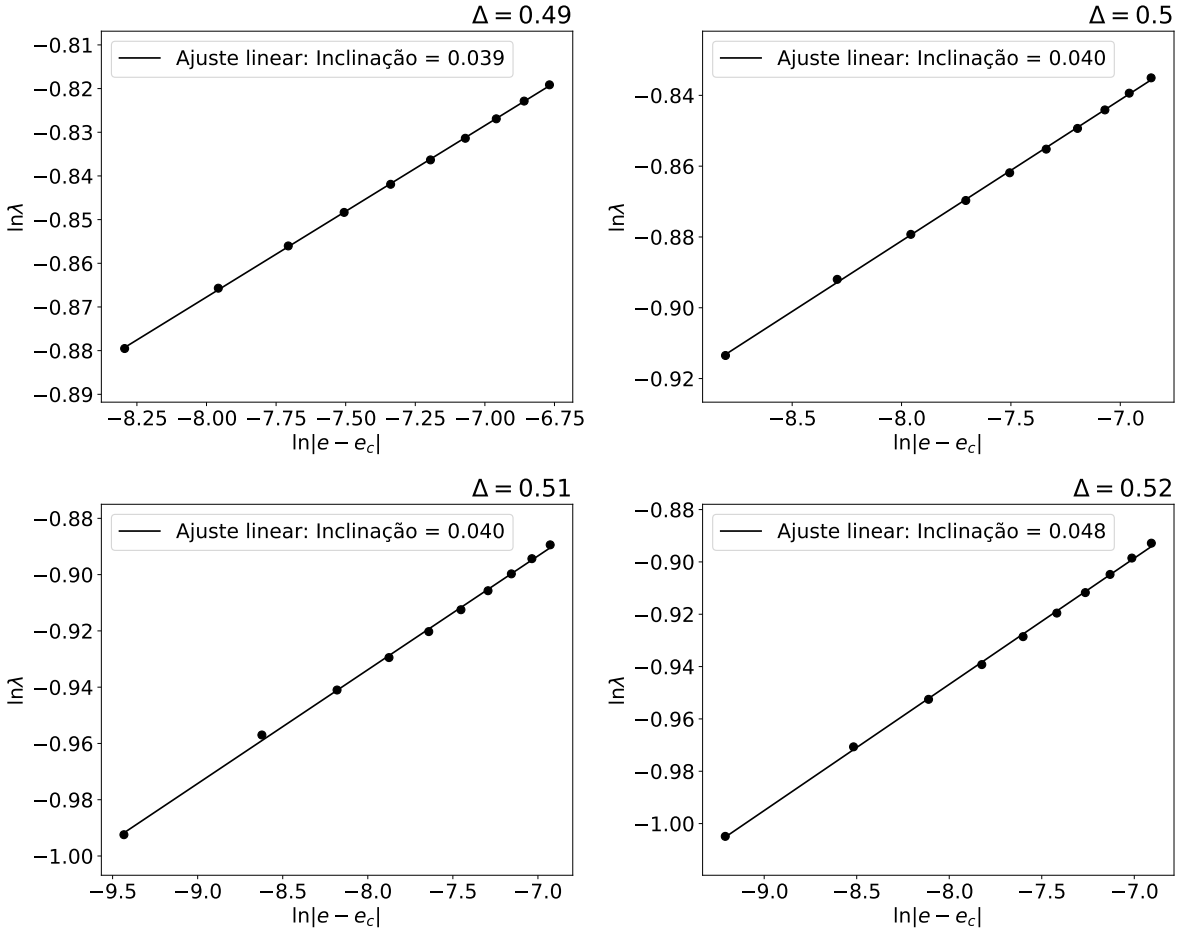


Figura 4.10: Estimativas analíticas de λ próxima a transição de fase para $\Delta = 0.49$ (superior esquerdo), $\Delta = 0.5$ (superior direito), $\Delta = 0.51$ (inferior esquerdo) e $\Delta = 0.52$ (superior direito).

A Figura 4.11 mostra que quando aumentamos Δ, começando de $\Delta = 0.46$, o valor do expoente crítico cai abruptamente de $\xi \approx 0.160$, valor da transição de segunda ordem, para $\xi \approx 0.040$, então cresce (de forma suave mas não linear) para 0.160 mais uma vez, mas diferindo levemente do seu valor para a transição de segunda ordem quando $\Delta \gtrsim 0.545$.

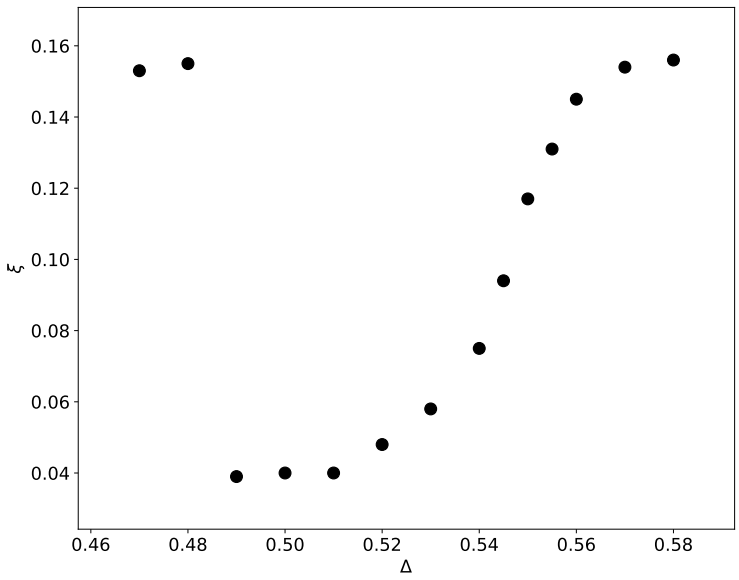


Figura 4.11: Expoente crítico do LLE para valores de Δ em torno da região de transição de primeira ordem.

4.4 Expoentes críticos - Numéricos

Como próximo passo natural temos as estimativas numéricas, obtidas via método Mapa Tangente, para os mesmo parâmetros usados nos resultados analíticos. Os resultados para $N = 10^4$, $N = 10^5$ e $N = 10^6$ e as previsões analíticas correspondentes estão agrupadas na Tabela 4.1. A Figura 4.12 exibe o comportamento do LLE próximo a energia crítica e os valores para o expoente da lei de potência ξ do ajuste.

Com o aumento do número de partículas, a estimativa numérica se aproxima da analítica, exceto para o caso de transição de primeira ordem. Para $\Delta = 0$ e $\Delta = 1$ (ambos para transição de segunda ordem), a estimativa da dinâmica é aproximadamente 10% menor do que a analítica, enquanto é apenas 1% menor para $\Delta = 0.35$.

Δ	Teórico	$N = 10^4$	$N = 10^5$	$N = 10^6$
0	0.160	0.144	0.125	0.148
0.35	0.158	0.079	0.151	0.156
0.5	0.040	0.064	0.058	0.069
1	0.160	0.081	0.144	0.144

Tabela 4.1: Expoentes críticos obtidos teoricamente e dinamicamente.

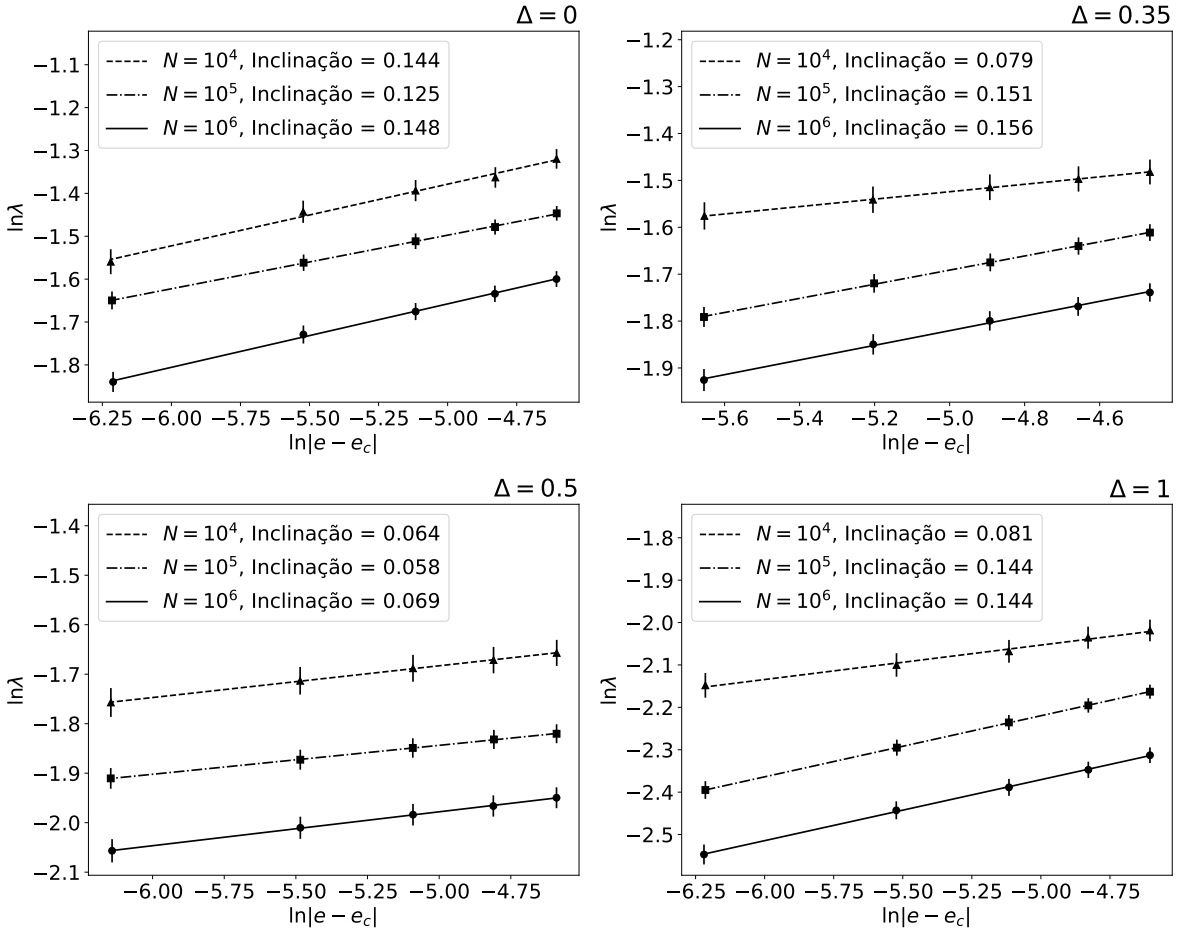


Figura 4.12: LLE do método de Mapa Tangente para $N = 10^4, 10^5, 10^6$ próximo das transições de fase para $\Delta = 0$ (superior esquerdo), $\Delta = 0.35$ (superior direito), $\Delta = 0.5$ (inferior esquerdo) e $\Delta = 1$ (superior direito).

Analisamos agora com mais atenção o comportamento do LLE para a transição de fase de primeira ordem concentrando-se nos valores do parâmetro nas proximidades de $\Delta = 0.5$. Os resultados estão resumidos na Tabela 4.2 e mostram que as previsões analíticas não concordam com os resultados das simulações neste intervalo de parâmetros. Os gráficos da Figura 4.13 devem ser comparados com os da Figura 4.10 para as estimativas analíticas. Por outro lado, verificamos o fenômeno que ξ aumenta com Δ, como esperado conforme mostrado na Figura 4.11.

Δ	Teórico	$N = 10^4$	$N = 10^5$	$N = 10^6$
0.49	0.039	0.051	0.061	0.056
0.50	0.040	0.064	0.058	0.069
0.51	0.040	0.083	0.072	0.073
0.52	0.048	0.074	0.073	0.096

Tabela 4.2: Expoentes críticos obtidos teoricamente e pelo Mapa Tangente na região de transições de primeira ordem.

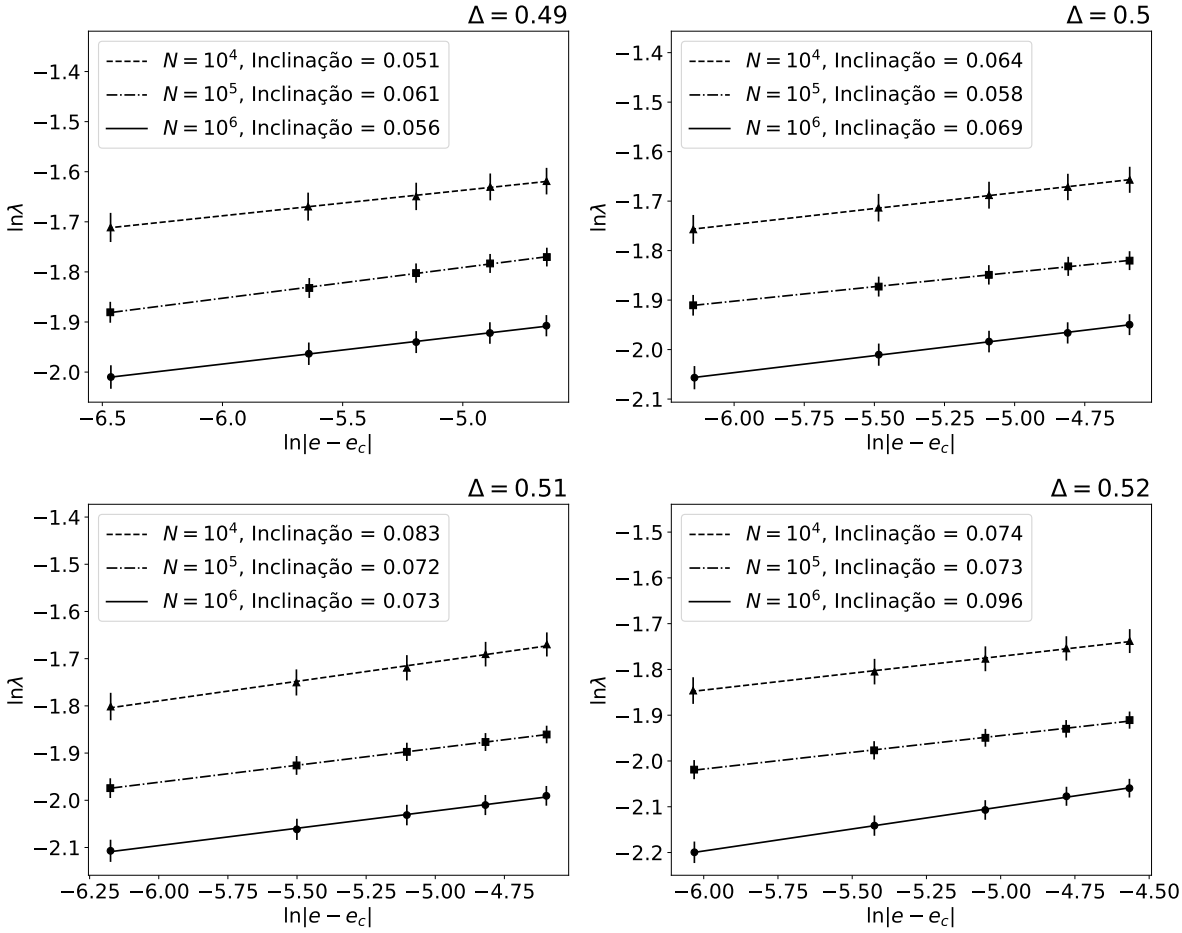


Figura 4.13: LLE do método de Mapa Tangente para $N = 10^4, 10^5, 10^6$ próximo das transições de fase para $\Delta = 0.49$ (superior esquerdo), $\Delta = 0.5$ (superior direito), $\Delta = 0.51$ (inferior esquerdo) e $\Delta = 0.52$ (inferior direito).

Conclusões

Neste trabalho mostramos que o método geométrico das Refs. [20, 21, 22, 23, 69] pode ser aplicado a um modelo mais geral do que o modelo cosHMF, considerado por Firpo [24]. Isso exigiu o uso de uma abordagem semi-analítica para estimar médias microcanônicas das flutuações da curvatura ao longo trajetórias (geodésicas) no espaço de configuração, que em princípio podem ser estendidas a outros modelos [56]. A utilização de regressões não-lineares via Redes Neurais é útil para suavizar resultados numéricos de simulações de Monte Carlo, permitindo uma maior precisão nas médias microcanônicas que impactam diretamente na estimativa do LLE [56]. Já para as estimativas numéricas com o método de Mapa Tangente, vimos que estimar a distribuição de partículas com Estimativa de Densidade Kernel é útil para preparar amostras do sistema com muitas partículas em equilíbrio a partir de uma distribuição com número reduzido de partículas. Em geral, aplicar estas técnicas de *Machine Learning* ao estudo de caos em sistemas com interações de longo alcance reduziu o tempo de simulação sem a perda de precisão dos valores obtidos.

Também foi investigado o comportamento da lei de potência do LLE próximo às diferentes transições de fase do modelo GHMF [56]. Embora o valor exato do LLE obtido pela abordagem analítica seja diferente da estimativa numérica com método de Mapa Tangente, que também ocorre para o modelo cosHMF [18], as estimativas para o expoente da lei de potência para o LLE estão em acordo razoável para transições de fase de segunda ordem, com o mesmo valor $\xi = 1/6$ previsto e observado para o modelo cosHMF. Isso é uma indicação de que pode ser um expoente universal, mas requer muito mais investigação [56]. Para a transição de primeira ordem, as estimativas numéricas diferem do valor previsto, mas tendem a concordar quando Δ cresce em direção ao valor em que a transição se torna de segunda ordem [56]. Simulações com um número maior de partículas e mais próximas da energia crítica podem confirmar se as previsões são imprecisas nesse caso.

Como perspectiva, o presente trabalho pode ser estendido a outros modelos unidimensionais, *e.g.* Anel Auto-Gravitante (SGR, do inglês *Self-Gravitating Ring*) [70], e de dimensões superiores, *e.g.* cosHMF em 2D [71, 72], para verificar se o expoente crítico $\xi = 1/6$ realmente se qualifica como um expoente crítico universal para transições de fase de segunda ordem em sistemas de interação de longo alcance.

Apêndice A

Códigos Fonte

Método de Mapa Tangente - CUDA

```
/*
================================================================================
    Program to obtain the biggest Lyapunov exponent using Tangent Map
    by Moises F. P. da Silva Jr - moisesfabianojr@gmail.com - Oct 2018
================================================================================
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <curand.h>
#include <curand_kernel.h>
#include <iostream>

// Functions prototypes

void cudaMem();
void cudaError();
void progress_bar(double percentage);
void initial_conditions(long nBlocks, long nThreads, long N, double norm0,
                        double M0, double beta, double q[], double p[],
                        double dev_q[], double dev_p[], double dev_w[]);
void energy_GHMF(long nBlocks, long nThreads, long N, double dev_q[],
                 double dev_p[], double *E, double *M, double *Mq,
                 double *beta, double delta, long Q);
void evolve_GHMF(long nBlocks, long nThreads, long N, double t0, double T,
                 double dt, double dev_q[], double dev_p[], double delta,
                 long Q);
void lyapunov_calc(long nBlocks, long nThreads, long N, double norm0,
                   long steps, double T, double dt, double *lambda,
                   double dev_q[], double dev_p[], double dev_w[],
                   double delta, long Q);

int main() {
    long i, N, steps, nBlocks, nThreads, Q;
    double norm0, E, M, Mq, beta, E0, M0, beta0, t0, T, dt, lambda=0.0, delta;
    double *dev_q, *dev_p, *dev_w;
    FILE *input, *output, *space_phase;

    srand(time(NULL));

// Reads parameters

    input = fopen("input.txt", "r");
    fscanf(input, "%ld", &N);
    fscanf(input, "%lf", &norm0);
    fscanf(input, "%ld", &steps);
    fscanf(input, "%lf", &t0);
    fscanf(input, "%lf", &T);
    fscanf(input, "%lf", &dt);
    fscanf(input, "%lf", &E0);
    fscanf(input, "%lf", &M0);
```

```
54      fscanf(input, "%lf", &beta0);
55      fscanf(input, "%lf", &delta);
56      fscanf(input, "%ld", &Q);
57      fscanf(input, "%ld", &nBlocks);
58      fscanf(input, "%ld", &nThreads);
59      fclose(input);
60      beta0 = 1.0/beta0;
61
62  // Allocates memory on the CPU
63
64      double *q = (double*)malloc(N*sizeof(double));
65      double *p = (double*)malloc(N*sizeof(double));
66
67  // Allocates memory on the GPU
68
69      cudaMalloc((void**)&dev_q, N*sizeof(double)); cudaError();
70      cudaMalloc((void**)&dev_p, N*sizeof(double)); cudaError();
71      cudaMalloc((void**)&dev_w, 2*N*sizeof(double)); cudaError();
72      cudaMem();
73
74  // Set the initial configuration
75      do {
76          initial_conditions(nBlocks, nThreads, N, norm0, M0, beta0, q, p,
77                             dev_q, dev_p, dev_w);
78          energy_GHMF(nBlocks, nThreads, N, dev_q, dev_p, &E, &M, &Mq, &beta,
79                      delta, Q);
80      }
81      while(fabs(E-E0) > 0.00001);
82
83  // Thermalize the system
84
85      evolve_GHMF(nBlocks, nThreads, N, 0.0, t0, dt, dev_q, dev_p, delta, Q);
86
87      energy_GHMF(nBlocks, nThreads, N, dev_q, dev_p, &E, &M, &Mq, &beta,
88                  delta, Q);
89
90  // Print the parameters of the simulation
91
92      printf("-------------------------------------------------\n");
93      printf("Number of particles: %ld\n", N);
94      printf("Initial norm: %lf\n", norm0);
95      printf("Number of steps: %ld\n", steps);
96      printf("Thermalization time: %.1lf\n", t0);
97      printf("Evolution time: %.1lf\n", T);
98      printf("Step size: %.3lf\n", dt);
99      printf("Equilibrium temperature: %lf\n", 1.0/beta0);
100     printf("Equilibrium magnetization: %lf\n", M0);
101     printf("Initial temperature: %lf\n", 1.0/beta);
102     printf("Initial magnetization: %lf\n", M);
103     printf("Initial q-magnetization: %lf\n", Mq);
104     printf("Initial energy: %lf\n", E);
105     printf("Number of blocks: %ld\n", nBlocks);
106     printf("Threads per block: %ld\n", nThreads);
107     printf("-------------------------------------------------\n");
108
109 // Obtain the Lyapunov exponent
110
111     lyapunov_calc(nBlocks, nThreads, N, norm0, steps, T, dt, &lambda,
112                   dev_q, dev_p, dev_w, delta, Q);
113
114     lambda /= steps*T;
115
116 // Saves the outputs
117
118     cudaMemcpy(q, dev_q, N*sizeof(double), cudaMemcpyDeviceToHost); cudaError();
119     cudaMemcpy(p, dev_p, N*sizeof(double), cudaMemcpyDeviceToHost); cudaError();
120
121     output = fopen("output.txt", "w");
122     space_phase = fopen("pxq.txt", "w");
123     for(i=0;i<N;i++)
124         fprintf(space_phase, "%lf\t%lf\n", q[i], p[i]);
125     fprintf(output, "%lf\t%lf\t%lf\t%lf\n", E, 1.0/beta, M, lambda);
126     fclose(output);
127     fclose(space_phase);
128
129     energy_GHMF(nBlocks, nThreads, N, dev_q, dev_p, &E, &M, &Mq, &beta,
```

```
130                 delta, Q);
131
132 // Print the parameters of the simulation
133
134     printf("-------------------------------------------------\n");
135     printf("Number of particles: %ld\n", N);
136     printf("Initial norm: %lf\n", norm0);
137     printf("Number of steps: %ld\n", steps);
138     printf("Thermalization time: %.1lf\n", t0);
139     printf("Evolution time: %.1lf\n", T);
140     printf("Step size: %.3lf\n", dt);
141     printf("Equilibrium temperature: %lf\n", 1.0/beta0);
142     printf("Equilibrium magnetization: %lf\n", M0);
143     printf("Final temperature: %lf\n", 1.0/beta);
144     printf("Final magnetization: %lf\n", M);
145     printf("Final q-magnetization: %lf\n", Mq);
146     printf("Final energy: %lf\n", E);
147     printf("Biggest Lyapunov exponent: %lf\n", lambda);
148     printf("Number of blocks: %ld\n", nBlocks);
149     printf("Threads per block: %ld\n", nThreads);
150     printf("-------------------------------------------------\n");
151
152     free(q);
153     free(p);
154     cudaFree(dev_q);
155     cudaFree(dev_p);
156     cudaFree(dev_w);
157
158     return 0;
159 }
```

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include <time.h>
5  #include <curand.h>
6  #include <curand_kernel.h>
7  #include <iostream>
8
9  void cudaError();
10 void condini_gauss(long N, double beta, double mag, double q[], double p[]);
11 //void waterbag_create(long N, double dq, double dp, double q[], double p[]);
12 __global__ void cuda_scalar_product(long N, double x[], double y[],
13                                     double *out);
14 __global__ void cuda_random_create(long N, double a, double b, double x[]);
15 __global__ void cuda_renorm(long N, double x[], double norm);
16
17 void initial_conditions(long nBlocks, long nThreads, long N, double norm0,
18                         double M0, double beta, double q[], double p[],
19                         double dev_q[], double dev_p[], double dev_w[]) {
20     long i;
21     double norm, zero=0.0;
22     double *dev_norm;
23     size_t shared = nThreads*sizeof(double);
24     FILE *initial_config;
25
26     initial_config = fopen("initial_config.txt", "r");
27
28     cudaMalloc((void**)&dev_norm, sizeof(double)); cudaError();
29
30 //  waterbag_create(N, 1.84, 1.84, q, p);
31     condini_gauss(N, beta, M0, q, p);
32     for(i=0;i<N;i++)
33         fscanf(initial_config, "%lf", &q[i]);
34     cudaMemcpy(dev_q, q, N*sizeof(double), cudaMemcpyHostToDevice);
35     cudaError();
36     cudaMemcpy(dev_p, p, N*sizeof(double), cudaMemcpyHostToDevice);
37     cudaError();
38
39     cuda_random_create<<<nBlocks,nThreads>>>(2*N, 2.0, -1.0, dev_w);
40     cudaError();
41
42     cudaMemcpy(dev_norm, &zero, sizeof(double), cudaMemcpyHostToDevice);
43     cudaError();
44     cuda_scalar_product<<<nBlocks,nThreads,shared>>>(2*N, dev_w, dev_w,
45                                                     dev_norm);
```

```
46      cudaError();
47      cudaMemcpy(&norm, dev_norm, sizeof(double), cudaMemcpyDeviceToHost);
48      cudaError();
49      cuda_renorm<<<nBlocks,nThreads,shared>>>(2*N, dev_w, sqrt(norm)/norm0);
50      cudaError();
51
52      cudaFree(dev_norm);
53      fclose(initial_config);
54  }
```

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <math.h>
4   #include <time.h>
5   #include <curand.h>
6   #include <curand_kernel.h>
7   #include <iostream>
8
9   void waterbag_create(long N, double dq, double dp, double q[], double p[]) {
10      long i;
11      double rq, rp;
12
13      for(i=0;i<N;i++) {
14          rq = (double)rand()/RAND_MAX;
15          rp = (double)rand()/RAND_MAX;
16          q[i] = rq*dq;
17          p[i] = (2.0*rp-1.0)*dp;
18      }
19  }
```

```
1   #include <stdlib.h>
2   #include <math.h>
3   #include <stdio.h>
4   #include <time.h>
5   #include <curand.h>
6   #include <curand_kernel.h>
7   #include <iostream>
8
9   #define TPI 6.283185307179586
10
11  // Function prototypes
12
13  double random_gaussian();
14
15  void condini_gauss(long N, double beta, double mag, double q[], double p[]) {
16      long i;
17      double R, lt, pr, pr2, mm, ebm, ss;
18
19      ss = 1.0/sqrt(beta);
20      ebm = exp(beta*mag);
21      lt = 0.0;
22      i=0;
23      while(i < N) {
24          R = (double)rand()/RAND_MAX;
25          q[i] = TPI*(R-0.5);
26          mm = cos(q[i]);
27          pr = exp(beta*mag*mm)/ebm;
28          pr2 = (double)rand()/RAND_MAX;
29          if(pr2 < pr) {
30              lt += mm;
31              i++;
32          }
33      }
34      lt /= (double)N;
35
36      for(i=0;i<N;i++)
37          p[i] = ss*random_gaussian();
38  }
```

```
1   #include <stdlib.h>
2   #include <math.h>
3   #include <stdio.h>
4   #include <time.h>
5   #include <curand.h>
6   #include <curand_kernel.h>
7   #include <iostream>
```

```

double random_gaussian() {
    long fl=0;
    double R1, R2, x, y, r;

    while(fl == 0) {
        R1 = (double)rand()/RAND_MAX;
        R2 = (double)rand()/RAND_MAX;

        x = 2.0*R1-1.0;
        y = 2.0*R2-1.0;

        r = x*x+y*y;

        if(r > 0.0 && r < 1.0) {
            r = x*sqrt(-2.0*log(r)/r);
            fl=1;
        }
    }

    return r;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <curand.h>
#include <curand_kernel.h>
#include <iostream>

void cudaError();
void progress_bar(double percentage);
void evolve_GHMF_linear(long nBlocks, long nThreads, long N, double t0,
                        double T, double dt, double dev_q[], double dev_p[],
                        double dev_deltaq[], double dev_deltap[],
                        double delta, long Q);
__global__ void cuda_scalar_product(long N, double x[], double y[],
                                    double *out);
__global__ void cuda_renorm(long N, double x[], double norm);

void lyapunov_calc(long nBlocks, long nThreads, long N, double norm0,
                   long steps, double T, double dt, double *lambda,
                   double dev_q[], double dev_p[], double dev_w[],
                   double delta, long Q) {
    long i;
    double norm, zero=0.0;
    double *dev_norm;
    size_t shared = nThreads*sizeof(double);
    FILE *lyapunov;

    lyapunov = fopen("lyapunov.txt", "w");

    cudaMalloc((void**)&dev_norm, sizeof(double)); cudaError();

    printf("Lyapunov calculation:\n");
    for(i=1;i<=steps;i++) {
        progress_bar((double)i/steps);

        evolve_GHMF_linear(nBlocks, nThreads, N, 0.0, T, dt, dev_q, dev_p,
                           &dev_w[0], &dev_w[N], delta, Q);

        cudaMemcpy(dev_norm, &zero, sizeof(double), cudaMemcpyHostToDevice);
        cudaError();
        cuda_scalar_product<<<nBlocks,nThreads,shared>>>(2*N, dev_w, dev_w,
                                                         dev_norm);
        cudaError();
        cudaMemcpy(&norm, dev_norm, sizeof(double), cudaMemcpyDeviceToHost);
        cudaError();
        cuda_renorm<<<nBlocks,nThreads,shared>>>(2*N, dev_w, sqrt(norm)/norm0);
        cudaError();

        *lambda += log(sqrt(norm)/norm0);
        fprintf(lyapunov, "%lf\t%lf\n", i*T, *lambda/(i*T));
    }
    progress_bar(1.0);
```

```
54      printf("\n");
55
56      fclose(lyapunov);
57      cudaFree(dev_norm);
58  }
```

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <math.h>
4   #include <time.h>
5   #include <curand.h>
6   #include <curand_kernel.h>
7   #include <iostream>
8
9   void cudaError();
10  void progress_bar(double percentage);
11  void leapfrog_GHMF(long nBlocks, long nThreads, long N, double dt,
12                     double *dev_m1x, double *dev_m1y, double *dev_m2x,
13                     double *dev_m2y, double dev_F[], double dev_q[],
14                     double dev_p[], double delta, long Q);
15  void magnetization_GHMF(long nBlocks, long nThreads, long N, double dev_q[],
16                          double *m1x, double *m1y, double *m2x, double *m2y,
17                          long Q);
18  __global__ void cuda_scalar_product(long N, double x[], double y[],
19                                      double *out);
20  __global__ void cuda_stats_moment(long N, long n, double mean, double x[],
21                                    double *out);
22
23  void evolve_GHMF(long nBlocks, long nThreads, long N, double t0, double T,
24                   double dt, double dev_q[], double dev_p[], double delta,
25                   long Q) {
26      long i=0;
27      double t, Ekin, M, Mq, m1x, m1y, m2x, m2y, m1, m2, m4, kurtosis, zero=0.0;
28      double *dev_m1x, *dev_m1y, *dev_m2x, *dev_m2y, *dev_F, *dev_Ekin, *dev_m1,
29             *dev_m2, *dev_m4;
30      FILE *therm;
31      size_t shared = nThreads*sizeof(double);
32
33      therm = fopen("thermalization.txt", "w");
34
35      cudaMalloc((void**)&dev_m1x, sizeof(double)); cudaError();
36      cudaMalloc((void**)&dev_m1y, sizeof(double)); cudaError();
37      cudaMalloc((void**)&dev_m2x, sizeof(double)); cudaError();
38      cudaMalloc((void**)&dev_m2y, sizeof(double)); cudaError();
39      cudaMalloc((void**)&dev_F, N*sizeof(double)); cudaError();
40      cudaMalloc((void**)&dev_Ekin, sizeof(double)); cudaError();
41      cudaMalloc((void**)&dev_m1, sizeof(double)); cudaError();
42      cudaMalloc((void**)&dev_m2, sizeof(double)); cudaError();
43      cudaMalloc((void**)&dev_m4, sizeof(double)); cudaError();
44
45      t = t0;
46      printf("Thermalizing:\n");
47      while(t < T) {
48          progress_bar((double)t/T);
49          leapfrog_GHMF(nBlocks, nThreads, N, dt, dev_m1x, dev_m1y, dev_m2x,
50                        dev_m2y, dev_F, dev_q, dev_p, delta, Q);
51
52          if(i%N == 0) {
53              cudaMemcpy(dev_Ekin, &zero, sizeof(double), cudaMemcpyHostToDevice);
54              cudaError();
55              cuda_scalar_product<<<nBlocks,nThreads,shared>>>(N, dev_p, dev_p,
56                                                               dev_Ekin);
57              cudaError();
58              cudaMemcpy(&Ekin, dev_Ekin, sizeof(double), cudaMemcpyDeviceToHost);
59              cudaError();
60
61              magnetization_GHMF(nBlocks, nThreads, N, dev_q, &m1x, &m1y, &m2x,
62                                 &m2y, Q);
63
64              M = sqrt(m1x*m1x+m1y*m1y);
65              Mq = sqrt(m2x*m2x+m2y*m2y);
66
67              Ekin *= 0.5;
68
69              cudaMemcpy(dev_m1, &zero, sizeof(double), cudaMemcpyHostToDevice);
70              cudaError();
```

```
71              cuda_stats_moment<<<nBlocks,nThreads,shared>>>(N, 1, 0, dev_p,
72                                                            dev_m1);
73              cudaError();
74              cudaMemcpy(&m1, dev_m1, sizeof(double), cudaMemcpyDeviceToHost);
75              cudaError();
76
77              cudaMemcpy(dev_m4, &zero, sizeof(double), cudaMemcpyHostToDevice);
78              cudaError();
79              cuda_stats_moment<<<nBlocks,nThreads,shared>>>(N, 4, m1, dev_p,
80                                                            dev_m4);
81              cudaError();
82              cudaMemcpy(&m4, dev_m4, sizeof(double), cudaMemcpyDeviceToHost);
83              cudaError();
84
85              cudaMemcpy(dev_m2, &zero, sizeof(double), cudaMemcpyHostToDevice);
86              cudaError();
87              cuda_stats_moment<<<nBlocks,nThreads,shared>>>(N, 2, m1, dev_p,
88                                                            dev_m2);
89              cudaError();
90              cudaMemcpy(&m2, dev_m2, sizeof(double), cudaMemcpyDeviceToHost);
91              cudaError();
92
93              kurtosis = m4/(m2*m2);
94
95              fprintf(therm, "%lf\t%lf\t%lf\t%lf\t%lf\t%lf\n",
96                      t, m4, kurtosis, 2*Ekin/(N-2.0), M, Mq);
97          }
98
99          t += dt;
100         i += 1;
101     }
102     progress_bar(1.0);
103     printf("\n");
104
105     cudaFree(dev_m1x);
106     cudaFree(dev_m1y);
107     cudaFree(dev_m2x);
108     cudaFree(dev_m2y);
109     cudaFree(dev_F);
110     cudaFree(dev_Ekin);
111     cudaFree(dev_m1);
112     cudaFree(dev_m2);
113     cudaFree(dev_m4);
114     fclose(therm);
115 }
```

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include <time.h>
5  #include <curand.h>
6  #include <curand_kernel.h>
7  #include <iostream>
8
9  void cudaError();
10 void progress_bar(double percentage);
11 void leapfrog_GHMF(long nBlocks, long nThreads, long N, double dt,
12                    double *dev_m1x, double *dev_m1y, double *dev_m2x,
13                    double *dev_m2y, double dev_F[], double dev_q[],
14                    double dev_p[], double delta, long Q);
15 void leapfrog_GHMF_linear(long nBlocks, long nThreads, long N, double dt,
16                           double *dev_m1x, double *dev_m1y, double *dev_m2x,
17                           double *dev_m2y, double *dev_deltam1x,
18                           double *dev_deltam1y, double *dev_deltam2x,
19                           double *dev_deltam2y, double dev_F[], double dev_q[],
20                           double dev_deltaq[], double dev_deltap[],
21                           double delta, long Q);
22
23 void evolve_GHMF_linear(long nBlocks, long nThreads, long N, double t0,
24                         double T, double dt, double dev_q[], double dev_p[],
25                         double dev_deltaq[], double dev_deltap[],
26                         double delta, long Q) {
27     double t;
28     double *dev_m1x, *dev_m1y, *dev_m2x, *dev_m2y, *dev_deltam1x, *dev_deltam1y,
29            *dev_deltam2x, *dev_deltam2y, *dev_F;
30
```

```
    cudaMalloc((void**)&dev_m1x, sizeof(double)); cudaError();
    cudaMalloc((void**)&dev_m1y, sizeof(double)); cudaError();
    cudaMalloc((void**)&dev_m2x, sizeof(double)); cudaError();
    cudaMalloc((void**)&dev_m2y, sizeof(double)); cudaError();
    cudaMalloc((void**)&dev_deltam1x, sizeof(double)); cudaError();
    cudaMalloc((void**)&dev_deltam1y, sizeof(double)); cudaError();
    cudaMalloc((void**)&dev_deltam2x, sizeof(double)); cudaError();
    cudaMalloc((void**)&dev_deltam2y, sizeof(double)); cudaError();
    cudaMalloc((void**)&dev_F, N*sizeof(double)); cudaError();

    t = t0;
    while(t < T) {
        leapfrog_GHMF(nBlocks, nThreads, N, dt, dev_m1x, dev_m1y, dev_m2x,
                      dev_m2y, dev_F, dev_q, dev_p, delta, Q);
        leapfrog_GHMF_linear(nBlocks, nThreads, N, dt, dev_m1x, dev_m1y,
                             dev_m2x, dev_m2y, dev_deltam1x, dev_deltam1y,
                             dev_deltam2x, dev_deltam2y, dev_F, dev_q,
                             dev_deltaq, dev_deltap, delta, Q);
        t += dt;
    }

    cudaFree(dev_m1x);
    cudaFree(dev_m1y);
    cudaFree(dev_m2x);
    cudaFree(dev_m2y);
    cudaFree(dev_deltam1x);
    cudaFree(dev_deltam1y);
    cudaFree(dev_deltam2x);
    cudaFree(dev_deltam2y);
    cudaFree(dev_F);
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <curand.h>
#include <curand_kernel.h>
#include <iostream>

#define C1 0.5
#define C2 1.0
#define C3 0.5

void cudaError();
void force_GHMF(long nBlocks, long nThreads, long N, double *dev_m1x,
                double *dev_m1y, double *dev_m2x, double *dev_m2y,
                double dev_F[], double dev_q[],
                double delta, long Q);
__global__ void cuda_boost(long N, double x[], double a, double y[]);

void leapfrog_GHMF(long nBlocks, long nThreads, long N, double dt,
                   double *dev_m1x, double *dev_m1y, double *dev_m2x,
                   double *dev_m2y, double dev_F[], double dev_q[],
                   double dev_p[], double delta, long Q) {
    double c1, c2, c3;

    c1 = C1*dt;
    c2 = C2*dt;
    c3 = C3*dt;

    cuda_boost<<<nBlocks,nThreads>>>(N, dev_q, c1, dev_p); cudaError();

    force_GHMF(nBlocks, nThreads, N, dev_m1x, dev_m1y, dev_m2x, dev_m2y, dev_F,
               dev_q, delta, Q);

    cuda_boost<<<nBlocks,nThreads>>>(N, dev_p, c2, dev_F); cudaError();

    cuda_boost<<<nBlocks,nThreads>>>(N, dev_q, c3, dev_p); cudaError();
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <curand.h>
```

```
6  #include <curand_kernel.h>
7  #include <iostream>
8
9  #define C1 0.5
10 #define C2 1.0
11 #define C3 0.5
12
13 void cudaError();
14 void force_GHMF_linear(long nBlocks, long nThreads, long N, double *dev_m1x,
15                        double *dev_m1y, double *dev_m2x, double *dev_m2y,
16                        double *dev_deltam1x, double *dev_deltam1y,
17                        double *dev_deltam2x, double *dev_deltam2y,
18                        double dev_F[], double dev_q[],
19                        double dev_deltaq[], double delta, long Q);
20 __global__ void cuda_boost(long N, double x[], double a, double y[]);
21
22 void leapfrog_GHMF_linear(long nBlocks, long nThreads, long N, double dt,
23                           double *dev_m1x, double *dev_m1y, double *dev_m2x,
24                           double *dev_m2y, double *dev_deltam1x,
25                           double *dev_deltam1y, double *dev_deltam2x,
26                           double *dev_deltam2y, double dev_F[], double dev_q[],
27                           double dev_deltaq[], double dev_deltap[],
28                           double delta, long Q) {
29     double c1, c2, c3;
30
31     c1 = C1*dt;
32     c2 = C2*dt;
33     c3 = C3*dt;
34
35     cuda_boost<<<nBlocks,nThreads>>>(N, dev_deltaq, c1, dev_deltap);
36     cudaError();
37
38     force_GHMF_linear(nBlocks, nThreads, N, dev_m1x, dev_m1y, dev_m2x, dev_m2y,
39                       dev_deltam1x, dev_deltam1y, dev_deltam2x, dev_deltam2y,
40                       dev_F, dev_q, dev_deltaq, delta, Q);
41
42     cuda_boost<<<nBlocks,nThreads>>>(N, dev_deltap, c2, dev_F);
43     cudaError();
44
45     cuda_boost<<<nBlocks,nThreads>>>(N, dev_deltaq, c3, dev_deltap);
46     cudaError();
47 }
```

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include <time.h>
5  #include <curand.h>
6  #include <curand_kernel.h>
7  #include <iostream>
8
9  void cudaError();
10 __global__ void cuda_mx(long N, double x[], long Q, double *out);
11 __global__ void cuda_my(long N, double x[], long Q, double *out);
12 __global__ void cuda_force_GHMF(long N, double m1x, double m1y, double m2x,
13                                 double m2y, double x[], double F[],
14                                 double delta, long Q);
15
16 void force_GHMF(long nBlocks, long nThreads, long N, double *dev_m1x,
17                 double *dev_m1y, double *dev_m2x, double *dev_m2y,
18                 double dev_F[], double dev_q[], double delta, long Q) {
19     double m1x, m1y, m2x, m2y, zero=0.0;
20     size_t shared = nThreads*sizeof(double);
21
22     cudaMemcpy(dev_m1x, &zero, sizeof(double), cudaMemcpyHostToDevice);
23     cudaError();
24     cudaMemcpy(dev_m1y, &zero, sizeof(double), cudaMemcpyHostToDevice);
25     cudaError();
26     cudaMemcpy(dev_m2x, &zero, sizeof(double), cudaMemcpyHostToDevice);
27     cudaError();
28     cudaMemcpy(dev_m2y, &zero, sizeof(double), cudaMemcpyHostToDevice);
29     cudaError();
30
31     cuda_mx<<<nBlocks,nThreads,shared>>>(N, dev_q, 1, dev_m1x);
32     cudaError();
33     cuda_my<<<nBlocks,nThreads,shared>>>(N, dev_q, 1, dev_m1y);
```

```
34      cudaError();
35      cuda_mx<<<nBlocks,nThreads,shared>>>(N, dev_q, Q, dev_m2x);
36      cudaError();
37      cuda_my<<<nBlocks,nThreads,shared>>>(N, dev_q, Q, dev_m2y);
38      cudaError();
39
40      cudaMemcpy(&m1x, dev_m1x, sizeof(double), cudaMemcpyDeviceToHost);
41      cudaError();
42      cudaMemcpy(&m1y, dev_m1y, sizeof(double), cudaMemcpyDeviceToHost);
43      cudaError();
44      cudaMemcpy(&m2x, dev_m2x, sizeof(double), cudaMemcpyDeviceToHost);
45      cudaError();
46      cudaMemcpy(&m2y, dev_m2y, sizeof(double), cudaMemcpyDeviceToHost);
47      cudaError();
48
49      cuda_force_GHMF<<<nBlocks,nThreads>>>(N, m1x, m1y, m2x, m2y, dev_q, dev_F,
50                                            delta, Q);
51  }
```

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <math.h>
4   #include <time.h>
5   #include <curand.h>
6   #include <curand_kernel.h>
7   #include <iostream>
8
9   void cudaError();
10  __global__ void cuda_mx(long N, double x[], long Q, double *out);
11  __global__ void cuda_my(long N, double x[], long Q, double *out);
12  __global__ void cuda_deltamx(long N, double x[], double deltax[], long Q,
13                               double *out);
14  __global__ void cuda_deltamy(long N, double x[], double deltax[], long Q,
15                               double *out);
16  __global__ void cuda_force_GHMF_linear(long N, double m1x, double m1y,
17                                         double m2x, double m2y,
18                                         double deltam1x, double deltam1y,
19                                         double deltam2x, double deltam2y,
20                                         double x[], double deltax[], double F[],
21                                         double delta, long Q);
22
23  void force_GHMF_linear(long nBlocks, long nThreads, long N, double *dev_m1x,
24                         double *dev_m1y, double *dev_m2x, double *dev_m2y,
25                         double *dev_deltam1x, double *dev_deltam1y,
26                         double *dev_deltam2x, double *dev_deltam2y,
27                         double dev_F[], double dev_q[], double dev_deltaq[],
28                         double delta, long Q) {
29      double deltam1x, deltam1y, deltam2x, deltam2y, m1x, m1y, m2x, m2y, zero=0.0;
30      size_t shared = nThreads*sizeof(double);
31
32      cudaMemcpy(dev_m1x, &zero, sizeof(double), cudaMemcpyHostToDevice);
33      cudaError();
34      cudaMemcpy(dev_m1y, &zero, sizeof(double), cudaMemcpyHostToDevice);
35      cudaError();
36      cudaMemcpy(dev_m2x, &zero, sizeof(double), cudaMemcpyHostToDevice);
37      cudaError();
38      cudaMemcpy(dev_m2y, &zero, sizeof(double), cudaMemcpyHostToDevice);
39      cudaError();
40      cudaMemcpy(dev_deltam1x, &zero, sizeof(double), cudaMemcpyHostToDevice);
41      cudaError();
42      cudaMemcpy(dev_deltam1y, &zero, sizeof(double), cudaMemcpyHostToDevice);
43      cudaError();
44      cudaMemcpy(dev_deltam2x, &zero, sizeof(double), cudaMemcpyHostToDevice);
45      cudaError();
46      cudaMemcpy(dev_deltam2y, &zero, sizeof(double), cudaMemcpyHostToDevice);
47      cudaError();
48
49      cuda_mx<<<nBlocks,nThreads,shared>>>(N, dev_q, 1, dev_m1x); cudaError();
50      cuda_my<<<nBlocks,nThreads,shared>>>(N, dev_q, 1, dev_m1y); cudaError();
51      cuda_mx<<<nBlocks,nThreads,shared>>>(N, dev_q, Q, dev_m2x); cudaError();
52      cuda_my<<<nBlocks,nThreads,shared>>>(N, dev_q, Q, dev_m2y); cudaError();
53      cuda_deltamx<<<nBlocks,nThreads,shared>>>(N, dev_q, dev_deltaq, 1,
54                                               dev_deltam1x);
55      cudaError();
56      cuda_deltamy<<<nBlocks,nThreads,shared>>>(N, dev_q, dev_deltaq, 1,
57                                               dev_deltam1y);
```

```
58      cudaError();
59      cuda_deltamx<<<nBlocks,nThreads,shared>>>(N, dev_q, dev_deltaq, Q,
60                                               dev_deltam2x);
61      cudaError();
62      cuda_deltamy<<<nBlocks,nThreads,shared>>>(N, dev_q, dev_deltaq, Q,
63                                               dev_deltam2y);
64      cudaError();
65
66      cudaMemcpy(&m1x, dev_m1x, sizeof(double), cudaMemcpyDeviceToHost);
67      cudaError();
68      cudaMemcpy(&m1y, dev_m1y, sizeof(double), cudaMemcpyDeviceToHost);
69      cudaError();
70      cudaMemcpy(&m2x, dev_m2x, sizeof(double), cudaMemcpyDeviceToHost);
71      cudaError();
72      cudaMemcpy(&m2y, dev_m2y, sizeof(double), cudaMemcpyDeviceToHost);
73      cudaError();
74      cudaMemcpy(&deltam1x, dev_deltam1x, sizeof(double), cudaMemcpyDeviceToHost);
75      cudaError();
76      cudaMemcpy(&deltam1y, dev_deltam1y, sizeof(double), cudaMemcpyDeviceToHost);
77      cudaError();
78      cudaMemcpy(&deltam2x, dev_deltam2x, sizeof(double), cudaMemcpyDeviceToHost);
79      cudaError();
80      cudaMemcpy(&deltam2y, dev_deltam2y, sizeof(double), cudaMemcpyDeviceToHost);
81      cudaError();
82
83      cuda_force_GHMF_linear<<<nBlocks,nThreads>>>(N, m1x, m1y, m2x, m2y,
84                                                   deltam1x, deltam1y,
85                                                   deltam2x, deltam2y,
86                                                   dev_q, dev_deltaq, dev_F,
87                                                   delta, Q);
88      cudaError();
89  }
```

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <math.h>
4   #include <time.h>
5   #include <curand.h>
6   #include <curand_kernel.h>
7   #include <iostream>
8
9   void cudaError();
10  void magnetization_GHMF(long nBlocks, long nThreads, long N, double dev_q[],
11                          double *m1x, double *m1y, double *m2x, double *m2y,
12                          long Q);
13  __global__ void cuda_scalar_product(long N, double x[], double y[],
14                                      double *out);
15
16  void energy_GHMF(long nBlocks, long nThreads, long N, double dev_q[],
17                   double dev_p[], double *E, double *M, double *Mq,
18                   double *beta, double delta, long Q) {
19      double m1x, m1y, m2x, m2y, zero=0.0, Ekin, Epot;
20      double *dev_Ekin;
21      size_t shared = nThreads*sizeof(double);
22
23      cudaMalloc((void**)&dev_Ekin, sizeof(double)); cudaError();
24
25      magnetization_GHMF(nBlocks, nThreads, N, dev_q, &m1x, &m1y, &m2x, &m2y, Q);
26
27      *M = sqrt(m1x*m1x+m1y*m1y);
28      *Mq = sqrt(m2x*m2x+m2y*m2y);
29      Epot = 0.5*(1.0 - delta*(*M)*(*M) - (1.0-delta)*(*Mq)*(*Mq));
30
31      cudaMemcpy(dev_Ekin, &zero, sizeof(double), cudaMemcpyHostToDevice);
32      cudaError();
33      cuda_scalar_product<<<nBlocks,nThreads,shared>>>(N, dev_p, dev_p, dev_Ekin);
34      cudaError();
35      cudaMemcpy(&Ekin, dev_Ekin, sizeof(double), cudaMemcpyDeviceToHost);
36      cudaError();
37
38      Ekin *= 0.5/N;
39      *beta = 1.0/(2.0*Ekin);
40      *E = Ekin + Epot;
41
42      cudaFree(dev_Ekin);
43  }
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <curand.h>
#include <curand_kernel.h>
#include <iostream>

void cudaError();
__global__ void cuda_mx(long N, double x[], long Q, double *out);
__global__ void cuda_my(long N, double x[], long Q, double *out);

void magnetization_GHMF(long nBlocks, long nThreads, long N, double dev_q[],
                        double *m1x, double *m1y, double *m2x, double *m2y,
                        long Q) {
    double zero=0.0;
    double *dev_m1x, *dev_m1y, *dev_m2x, *dev_m2y;
    size_t shared = nThreads*sizeof(double);

    cudaMalloc((void**)&dev_m1x, sizeof(double)); cudaError();
    cudaMalloc((void**)&dev_m1y, sizeof(double)); cudaError();
    cudaMalloc((void**)&dev_m2x, sizeof(double)); cudaError();
    cudaMalloc((void**)&dev_m2y, sizeof(double)); cudaError();

    cudaMemcpy(dev_m1x, &zero, sizeof(double), cudaMemcpyHostToDevice);
    cudaError();
    cudaMemcpy(dev_m1y, &zero, sizeof(double), cudaMemcpyHostToDevice);
    cudaError();
    cudaMemcpy(dev_m2x, &zero, sizeof(double), cudaMemcpyHostToDevice);
    cudaError();
    cudaMemcpy(dev_m2y, &zero, sizeof(double), cudaMemcpyHostToDevice);
    cudaError();

    cuda_mx<<<nBlocks,nThreads,shared>>>(N, dev_q, 1, dev_m1x); cudaError();
    cuda_my<<<nBlocks,nThreads,shared>>>(N, dev_q, 1, dev_m1y); cudaError();
    cuda_mx<<<nBlocks,nThreads,shared>>>(N, dev_q, Q, dev_m2x); cudaError();
    cuda_my<<<nBlocks,nThreads,shared>>>(N, dev_q, Q, dev_m2y); cudaError();

    cudaMemcpy(m1x, dev_m1x, sizeof(double), cudaMemcpyDeviceToHost);
    cudaError();
    cudaMemcpy(m1y, dev_m1y, sizeof(double), cudaMemcpyDeviceToHost);
    cudaError();
    cudaMemcpy(m2x, dev_m2x, sizeof(double), cudaMemcpyDeviceToHost);
    cudaError();
    cudaMemcpy(m2y, dev_m2y, sizeof(double), cudaMemcpyDeviceToHost);
    cudaError();

    cudaFree(dev_m1x);
    cudaFree(dev_m1y);
    cudaFree(dev_m2x);
    cudaFree(dev_m2y);
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <curand.h>
#include <curand_kernel.h>
#include <iostream>

__global__ void cuda_random_create(long N, double a, double b, double x[]) {
    long index;
    double R;
    curandState state;

    index = threadIdx.x + blockIdx.x*blockDim.x;

    curand_init(clock64(), index, 0, &state);

    while(index < N) {
        R = curand_uniform(&state);
        x[index] = a*R + b;
        index += blockDim.x*gridDim.x;
    }
    __syncthreads();
```

```
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <curand.h>
#include <curand_kernel.h>
#include <iostream>

__global__ void cuda_force_GHMF(long N, double m1x, double m1y, double m2x,
                                double m2y, double x[], double F[],
                                double delta, long Q) {
    long index;

    index = threadIdx.x + blockIdx.x*blockDim.x;
    while(index < N) {
        F[index] = delta*(m1y*cos(x[index]) - m1x*sin(x[index]))
                 + (1.-delta)*Q*(m2y*cos(Q*x[index]) - m2x*sin(Q*x[index]));
        index += blockDim.x*gridDim.x;
    }
    __syncthreads();
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <curand.h>
#include <curand_kernel.h>
#include <iostream>

__global__ void cuda_force_GHMF_linear(long N, double m1x, double m1y,
                                       double m2x, double m2y,
                                       double deltam1x, double deltam1y,
                                       double deltam2x, double deltam2y,
                                       double x[], double deltax[], double F[],
                                       double delta, long Q) {
    long index;

    index = threadIdx.x + blockIdx.x*blockDim.x;
    while(index < N) {
        F[index] = delta*(deltam1x*sin(x[index]) + deltam1y*cos(x[index])
                 - m1x*cos(x[index])*deltax[index]
                 - m1y*sin(x[index])*deltax[index])
                 + (1.0-delta)*Q*(deltam2x*sin(Q*x[index])
                 + deltam2y*cos(Q*x[index])
                 - m2x*Q*cos(Q*x[index])*deltax[index]
                 - m2y*Q*sin(Q*x[index])*deltax[index]);
        index += blockDim.x*gridDim.x;
    }
    __syncthreads();
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <curand.h>
#include <curand_kernel.h>
#include <iostream>

__device__ void cuda_atomicAdd_mx(double *address, double val) {
    double assumed,old=*address;

    do {
        assumed=old;
        old= __longlong_as_double(atomicCAS((unsigned long long int*)address,
                    __double_as_longlong(assumed),
                    __double_as_longlong(val+assumed)));
    }while(assumed != old);
}

__global__ void cuda_mx(long N, double x[], long Q, double *out) {
    extern __shared__ double sdata[];
    long I, index;
```

```
23
24      sdata[threadIdx.x] = 0.0;
25
26      index = threadIdx.x + blockIdx.x*blockDim.x;
27      while(index < N) {
28          sdata[threadIdx.x] += cos(Q*x[index]);
29          index += blockDim.x*gridDim.x;
30      }
31      __syncthreads();
32
33      for(I=blockDim.x/2;I>0;I>>=1) {
34          if(threadIdx.x < I)
35              sdata[threadIdx.x] += sdata[threadIdx.x + I];
36          __syncthreads();
37      }
38      __syncthreads();
39      if(threadIdx.x == 0)
40          cuda_atomicAdd_mx(out,sdata[0]/N);
41  }
```

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <math.h>
4   #include <time.h>
5   #include <curand.h>
6   #include <curand_kernel.h>
7   #include <iostream>
8
9   __device__ void cuda_atomicAdd_my(double *address, double val) {
10      double assumed,old=*address;
11
12      do {
13          assumed=old;
14          old= __longlong_as_double(atomicCAS((unsigned long long int*)address,
15                      __double_as_longlong(assumed),
16                      __double_as_longlong(val+assumed)));
17      }while(assumed != old);
18  }
19
20  __global__ void cuda_my(long N, double x[], long Q, double *out) {
21      extern __shared__ double sdata[];
22      long I, index;
23
24      sdata[threadIdx.x] = 0.0;
25
26      index = threadIdx.x + blockIdx.x*blockDim.x;
27      while(index < N) {
28          sdata[threadIdx.x] += sin(Q*x[index]);
29          index += blockDim.x*gridDim.x;
30      }
31      __syncthreads();
32
33      for(I=blockDim.x/2;I>0;I>>=1) {
34          if(threadIdx.x < I)
35              sdata[threadIdx.x] += sdata[threadIdx.x + I];
36          __syncthreads();
37      }
38      __syncthreads();
39      if(threadIdx.x == 0)
40          cuda_atomicAdd_my(out,sdata[0]/N);
41  }
```

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <math.h>
4   #include <time.h>
5   #include <curand.h>
6   #include <curand_kernel.h>
7   #include <iostream>
8
9   __device__ void cuda_atomicAdd_deltamx(double *address, double val) {
10      double assumed,old=*address;
11
12      do {
13          assumed=old;
14          old= __longlong_as_double(atomicCAS((unsigned long long int*)address,
```

```
                    __double_as_longlong(assumed),
                    __double_as_longlong(val+assumed)));
    }while(assumed != old);
}

__global__ void cuda_deltamx(long N, double x[], double deltax[], long Q,
                             double *out) {
    extern __shared__ double sdata[];
    long I, index;

    sdata[threadIdx.x] = 0.0;

    index = threadIdx.x + blockIdx.x*blockDim.x;
    while(index < N) {
        sdata[threadIdx.x] += Q*sin(Q*x[index])*deltax[index];
        index += blockDim.x*gridDim.x;
    }
    __syncthreads();

    for(I=blockDim.x/2;I>0;I>>=1) {
        if(threadIdx.x < I)
            sdata[threadIdx.x] += sdata[threadIdx.x + I];
        __syncthreads();
    }
    __syncthreads();
    if(threadIdx.x == 0)
        cuda_atomicAdd_deltamx(out,sdata[0]/N);
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <curand.h>
#include <curand_kernel.h>
#include <iostream>

__device__ void cuda_atomicAdd_deltamy(double *address, double val) {
    double assumed,old=*address;

    do {
        assumed=old;
        old= __longlong_as_double(atomicCAS((unsigned long long int*)address,
                    __double_as_longlong(assumed),
                    __double_as_longlong(val+assumed)));
    }while(assumed != old);
}

__global__ void cuda_deltamy(long N, double x[], double deltax[], long Q,
                             double *out) {
    extern __shared__ double sdata[];
    long I, index;

    sdata[threadIdx.x] = 0.0;

    index = threadIdx.x + blockIdx.x*blockDim.x;
    while(index < N) {
        sdata[threadIdx.x] += Q*cos(Q*x[index])*deltax[index];
        index += blockDim.x*gridDim.x;
    }
    __syncthreads();

    for(I=blockDim.x/2;I>0;I>>=1) {
        if(threadIdx.x < I)
            sdata[threadIdx.x] += sdata[threadIdx.x + I];
        __syncthreads();
    }
    __syncthreads();
    if(threadIdx.x == 0)
        cuda_atomicAdd_deltamy(out,sdata[0]/N);
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
```

```
#include <curand.h>
#include <curand_kernel.h>
#include <iostream>

__device__ void cuda_atomicAdd_stats(double *address, double val) {
    double assumed,old=*address;

    do {
        assumed=old;
        old= __longlong_as_double(atomicCAS((unsigned long long int*)address,
                    __double_as_longlong(assumed),
                    __double_as_longlong(val+assumed)));
    }while(assumed != old);
}

__global__ void cuda_stats_moment(long N, long n, double mean, double x[],
                                  double *out) {
    extern __shared__ double sdata[];
    long I, index;

    sdata[threadIdx.x] = 0.0;
    index = threadIdx.x + blockIdx.x*blockDim.x;
    while(index < N) {
        sdata[threadIdx.x] += powf(x[index] - mean, n);
        index += blockDim.x*gridDim.x;
    }
    __syncthreads();

    for(I=blockDim.x/2;I>0;I>>=1) {
        if(threadIdx.x < I)
            sdata[threadIdx.x] += sdata[threadIdx.x + I];
        __syncthreads();
    }
    __syncthreads();

    if(threadIdx.x == 0)
        cuda_atomicAdd_stats(out,sdata[0]/N);
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <curand.h>
#include <curand_kernel.h>
#include <iostream>

__global__ void cuda_boost(long N, double x[], double a, double y[]) {
    long index;

    index = threadIdx.x + blockIdx.x*blockDim.x;
    while(index < N) {
        x[index] += a*y[index];
        index += blockDim.x*gridDim.x;
    }
    __syncthreads();
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <curand.h>
#include <curand_kernel.h>
#include <iostream>

__global__ void cuda_renorm(long N, double x[], double norm) {
    long index;

    index = threadIdx.x + blockIdx.x*blockDim.x;
    while(index < N) {
        x[index] /= norm;
        index += blockDim.x*gridDim.x;
    }
    __syncthreads();
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <curand.h>
#include <curand_kernel.h>
#include <iostream>

__device__ void cuda_atomicAdd_scalar(double *address, double val) {
    double assumed,old=*address;

    do {
        assumed=old;
        old= __longlong_as_double(atomicCAS((unsigned long long int*)address,
                    __double_as_longlong(assumed),
                    __double_as_longlong(val+assumed)));
    }while(assumed != old);
}

__global__ void cuda_scalar_product(long N, double x[], double y[],
                                    double *out) {
    extern __shared__ double sdata[];
    long I, index;

    sdata[threadIdx.x] = 0.0;
    index = threadIdx.x + blockIdx.x*blockDim.x;
    while(index < N) {
        sdata[threadIdx.x] += x[index]*y[index];
        index += blockDim.x*gridDim.x;
    }
    __syncthreads();

    for(I=blockDim.x/2;I>0;I>>=1) {
        if(threadIdx.x < I)
            sdata[threadIdx.x] += sdata[threadIdx.x + I];
        __syncthreads();
    }
    __syncthreads();

    if(threadIdx.x == 0)
        cuda_atomicAdd_scalar(out,sdata[0]);
}
```

Monte Carlo Microcanônico - C

```c
/*
================================================================================
    Program to obtain microcanonical averages using Ray's algorithm
    by Moises F. P. da Silva Jr - moisesfabianojr@gmail.com - Jun 2017
================================================================================
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

// Functions prototypes

void progress_bar(double percentage);
void initial_condition(long N, long J, double E0, double dE, double V0,
                       double *E, double *K, double X[], double delta, long q);
void microcanonical_metropolis(long N, double V, double E, double steps0,
                               double steps, double *K, double *T, double *M,
                               double *Mq, double *KR, double *KR2, double X[],
                               double delta, long q);
double lyapunov_theoretical(long N, double KR, double KR2);

int main() {
    long I, N, n, steps0, steps, q;
    double E, dE, E0, Ef, V, V0, K, T, M, Mq, KR, KR2, lambda, delta;
    FILE *input, *tem, *mag, *lam;

    tem = fopen("temperature.txt", "wt");
    mag = fopen("magnetization.txt", "wt");
    lam = fopen("lyapunov.txt", "wt");

    srand(time(NULL));

// Reads parameters

    input = fopen("input.txt", "rt");
    fscanf(input, "%ld\n", &N);
    fscanf(input, "%lf\n", &V);
    fscanf(input, "%lf\n", &V0);
    fscanf(input, "%lf\n", &E0);
    fscanf(input, "%lf\n", &Ef);
    fscanf(input, "%lf\n", &dE);
    fscanf(input, "%lf\n", &delta);
    fscanf(input, "%ld\n", &q);
    fscanf(input, "%ld\n", &steps0);
    fscanf(input, "%ld\n", &steps);
    fclose(input);

    n = (long)((Ef - E0)/dE);

// Allocates memory on the CPU

    double *X = (double*)malloc(N*sizeof(double));

// Print the parameters of the simulation

    printf("-------------------------------------------------\n");
    printf("Number of particles: %ld\n", N);
    printf("Volume: %lf\n", V);
    printf("Initial volume: %lf\n", V0);
    printf("Initial Energy: %.3lf\n", E0);
    printf("Final Energy: %.3lf\n", Ef);
    printf("Energy variation: %.3lf\n", dE);
    printf("Delta: %.2lf\n", delta);
    printf("Intervals of energy: %ld\n", n);
    printf("Steps to converge: %ld\n", steps0);
    printf("Steps: %ld\n", steps);
    printf("-------------------------------------------------\n");

// Obtain the caloric curve

    printf("Calculation:\n");
    for(I=0;I<=n;I++) {
```

```
75          progress_bar((double)I/n);
76          initial_condition(N, I, E0, dE, V0, &E, &K, X, delta, q);
77          if(K >= 0) {
78              microcanonical_metropolis(N, V, E, steps0, steps, &K, &T, &M, &Mq,
79                                        &KR, &KR2, X, delta, q);
80              lambda = lyapunov_theoretical(N, KR, KR2);
81              fprintf(tem, "%f\t%f\n", E, T);
82              fprintf(mag, "%f\t%f\t%f\n", E, M, Mq);
83              fprintf(lam, "%f\t%f\t%f\t%f\n", E, KR, KR2, sqrt((KR2 - KR*KR)*N));
84          }
85      }
86      printf("\n");
87
88      fclose(tem);
89      fclose(mag);
90      fclose(lam);
91      free(X);
92
93      return(0);
94  }
```

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <math.h>
4   #include <time.h>
5
6   double magnetization_GHMF(long N, double X[], long q);
7   double energy_GHMF(long N, double M, double Mq, double delta);
8
9   void initial_condition(long N, long J, double E0, double dE, double V0,
10                         double *E, double *K, double X[], double delta,
11                         long q) {
12      long I;
13      double U, M, Mq, R;
14
15      *E = E0 + J*dE;
16      for(I=0;I<N;I++) {
17          R = (double)rand()/RAND_MAX;
18          X[I] = (double)V0*R;
19      }
20
21      M = magnetization_GHMF(N, X, 1);
22      Mq = magnetization_GHMF(N, X, q);
23      U = energy_GHMF(N, M, Mq, delta);
24      *K = *E - U;
25  }
```

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <math.h>
4   #include <time.h>
5
6   double temperature(long N, double K);
7   double magnetization_GHMF(long N, double X[], long q);
8   void propose_energy(long N, double V, double E, long *P, double *X2,
9                       double *K2, double X[], double delta, long q);
10  double Ricci_curvature(double delta, long q, double m, double mq);
11  //void progress_bar(double percentage);
12
13  void microcanonical_metropolis(long N, double V, double E, double steps0,
14                                 double steps, double *K, double *T, double *M,
15                                 double *Mq, double *KR, double *KR2, double X[],
16                                 double delta, long q) {
17      long I, P;
18      double K2, X2, PA, m, mq, sum_T, sum_M, sum_Mq, sum_KR, sum_KR2, Ricci;
19  //  FILE *evol;
20
21  //  evol = fopen("evol.txt", "wt");
22
23      sum_T = 0.0;
24      sum_M = 0.0;
25      sum_Mq = 0.0;
26      sum_KR = 0.0;
27      sum_KR2 = 0.0;
28      for(I=0;I<steps;I++) {
29  //      progress_bar((double)I/steps);
```

```
30          propose_energy(N, V, E, &P, &X2, &K2, X, delta, q);
31
32          PA = exp((0.5*N-1.0)*log(K2/(*K)));
33          if((double)rand()/RAND_MAX < PA)
34              *K = K2;
35          else
36              X[P] = X2;
37
38          if(I > steps0) {
39              m = magnetization_GHMF(N, X, 1);
40              mq = magnetization_GHMF(N, X, q);
41              sum_T += N*temperature(N, *K);
42              sum_M += m;
43              sum_Mq += mq;
44              Ricci = Ricci_curvature(delta, q, m, mq);
45              sum_KR += Ricci;
46              sum_KR2 += Ricci*Ricci;
47          }
48  //      fprintf(evol, "%ld\t%lf\n", I,
49  //              Ricci_curvature(delta, q, magnetization_GHMF(N, X, 1),
50  //              magnetization_GHMF(N, X, q)));
51      }
52  //  progress_bar(1.0);
53
54      *T = sum_T/(steps-steps0);
55      *M = sum_M/(steps-steps0);
56      *Mq = sum_Mq/(steps-steps0);
57      *KR = sum_KR/(steps-steps0);
58      *KR2 = sum_KR2/(steps-steps0);
59  //  fclose(evol);
60  }
```

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <math.h>
4   #include <time.h>
5
6   void new_state_GHMF(long N, double V, long *P, double *X2, double X[]);
7   double magnetization_GHMF(long N, double X[], long q);
8   double energy_GHMF(long N, double M, double Mq, double delta);
9
10  void propose_energy(long N, double V, double E, long *P, double *X2,
11                      double *K2, double X[], double delta, long q) {
12      double M, Mq, U;
13
14      do {
15          new_state_GHMF(N, V, P, X2, X);
16          M = magnetization_GHMF(N, X, 1);
17          Mq = magnetization_GHMF(N, X, q);
18          U = energy_GHMF(N, M, Mq, delta);
19          *K2 = E - U;
20
21          if(*K2 < 0)
22              X[*P] = *X2;
23      }
24      while(*K2 < 0);
25  }
```

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <math.h>
4   #include <time.h>
5
6   void new_state_GHMF(long N, double V, long *P, double *X2, double X[]) {
7       double R;
8
9       do {
10          R = (double)rand()/RAND_MAX;
11          *P = (long)N*R;
12      }
13      while(*P >= N);
14
15      R = (double)rand()/RAND_MAX;
16      *X2 = X[*P];
17      X[*P] = (double)V*R;
18  }
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

double energy_GHMF(long N, double M, double Mq, double delta) {
    double Epot;

    Epot = 0.5*(1.0 - delta*M*M - (1.0-delta)*Mq*Mq);

    return Epot;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

double magnetization_GHMF(long N, double X[], long q) {
    long i;
    double m, mx, my;

    mx = 0.0;
    my = 0.0;
    for(i=0;i<N;i++) {
        mx += (double)cos(q*X[i]);
        my += (double)sin(q*X[i]);
    }
    mx /= (double)N;
    my /= (double)N;

    m = sqrt(mx*mx + my*my);

    return m;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

double Ricci_curvature(double delta, long q, double m, double mq) {
    double Ricci;

    Ricci = delta*m*m + (1.0-delta)*q*q*mq*mq;

    return Ricci;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

double temperature(long N, double K) {
    return 2.0*K/((double)N-2.0);
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

double lyapunov_theoretical(long N, double KR, double KR2) {
    double kappa, sigma, tau, Lambda;

    kappa = KR;
    sigma = sqrt((KR2 - KR*KR)*N);

    tau = 3.14159265359*sqrt(kappa)/(2.0*sqrt(kappa)*sqrt(kappa+sigma)
        + 3.14159265359*sigma);
    Lambda = pow(2.0*sigma*sigma*tau + sqrt(64.0*pow(kappa,3)/27.0
                                            + 4.0*pow(sigma,4)*tau*tau),
                 1.0/3.0);
    return Lambda/2.0 - 2.0*kappa/(3.0*Lambda);
}
```

Regressão via Rede Neural - Python

```
# -*- coding: utf-8 -*-
"""
Created on Thu Feb 7 2019

@author: Moises
"""
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from keras.layers import Dense
from keras.models import Sequential
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint, EarlyStopping

dataset = np.loadtxt('./DADOS.txt')
X = dataset[:, 0:1]
y = dataset[:, 3]

sc = StandardScaler()
X_sc = sc.fit_transform(X)

def build_nn():
    model = Sequential()
    model.add(Dense(32, activation='elu', input_dim=1))
    model.add(Dense(32, activation='elu'))
    model.add(Dense(32, activation='elu'))
    model.add(Dense(32, activation='elu'))
    model.add(Dense(1))
    model.compile(optimizer=Adam(lr=0.001), loss='mae')

    return model

model = build_nn()
print(model.summary())
history = model.fit(X_sc, y, batch_size=X.size,
                    epochs=1000000, validation_split=0.2,
                    callbacks=[ModelCheckpoint('./model.hdf5',
                                               monitor='val_loss',
                                               save_best_only=True),
                               EarlyStopping('val_loss', patience=1000)])
model.load_weights('./model.hdf5')

x_new = np.linspace(X[0,0], X[-1,0], 1000).reshape(-1,1)
x_new_sc = sc.transform(x_new)
y_pred = model.predict(x_new_sc)
```

Estimativa de distribuição via KDE - Python

```python
# -*- coding: utf-8 -*-
"""
Created on Wed Aug 7 2019

@author: Moises
"""
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KernelDensity

dados = np.loadtxt('initial_config.txt')

def kde_sklearn(X, y):
    h = 1.06*np.std(y)*len(y)**(-0.2)
    kde = KernelDensity(bandwidth=h)
    kde.fit(y[:, np.newaxis])
    log_pdf = kde.score_samples(X[:, np.newaxis])
    y_new = kde.sample(20*len(y), random_state=0)
    y_new = np.array([y - 2*np.pi if y > 2*np.pi else 2*np.pi + y
                      if y < 0 else y for y in y_new])
    return np.exp(log_pdf), y_new

y = dados
X = np.linspace(0, 2*np.pi, 1000)
pdf, y_new = kde_sklearn(X, y)

plt.figure(figsize=(10, 8))
plt.hist(y, bins='auto', density=True, alpha=1)
plt.hist(y_new, bins='auto', density=True, alpha=0.5)
plt.plot(X, pdf, c='black', lw=3, label='KDE')
plt.xlabel(r'$\theta$')
plt.xlim(0, 2*np.pi)
plt.legend()
plt.savefig('kde_result.pdf', format='pdf', dpi=1200)
plt.clf()

np.savetxt('initial_config_new.txt', y_new, fmt='%.6f')

def energy_GHMF(x, E, delta, q=2):
    m = np.sqrt(np.cos(x).sum()**2 + np.sin(x).sum()**2)/len(x)
    mq = np.sqrt(np.cos(q*x).sum()**2 + np.sin(q*x).sum()**2)/len(x)
    U = 0.5*(1 - delta*m**2 - (1-delta)*mq**2)
    T = 2*(E-U)

    return 0.5*T + U, T

E = 0.69
U, T = energy_GHMF(y_new, E, 1.0)
print('{:.6f}'.format(U))
print('{:.6f}'.format(T))
```

Referências Bibliográficas

[1] Alessandro Campa, Thierry Dauxois, Duccio Fanelli, and Stefano Ruffo. *Physics of long-range interacting systems*. OUP Oxford, 2014.

[2] M Kac, GE Uhlenbeck, and PC Hemmer. On the van der waals theory of the vapor-liquid equilibrium. i. discussion of a one-dimensional model. *Journal of Mathematical Physics*, 4(2):216–228, 1963.

[3] Werner Braun and K Hepp. The vlasov dynamics and its fluctuations in the 1/n limit of interacting classical particles. *Communications in mathematical physics*, 56(2):101–113, 1977.

[4] TM Rocha Filho, MA Amato, AE Santana, A Figueiredo, and JR Steiner. Dynamics and physical interpretation of quasistationary states in systems with long-range interactions. *Physical Review E*, 89(3):032116, 2014.

[5] Herbert Spohn. *Large scale dynamics of interacting particles*. Springer Science & Business Media, 2012.

[6] Pierre-Emmanuel Jabin. A review of the mean field limits for vlasov equations. *Kinetic & Related Models*, 7(4):661–711, 2014.

[7] Michael K-H Kiessling. The microscopic foundations of vlasov theory for jellium-like newtonian n-body systems. *Journal of Statistical Physics*, 155(6):1299–1328, 2014.

[8] JL Rouet and MR Feix. Relaxation for one-dimensional plasma: Test particles versus global distribution behavior. *Physics of Fluids B: Plasma Physics*, 3(8):1830–1834, 1991.

[9] Y Chaffi, Rocha TM Filho, and L Brenig. A convergent kinetic equation for gravitational and coulomb systems. *arXiv preprint arXiv:1711.07353*, 2017.

[10] Dominique F Escande, Yves Elskens, and Fabrice Doveil. Uniform derivation of coulomb collisional transport thanks to debye shielding. *Journal of Plasma Physics*, 81(1), 2015.

[11] DF Escande, Didier Bénisti, Yves Elskens, David Zarzoso, and Fabrice Doveil. Basic microscopic plasma physics from n-body mechanics. *Reviews of Modern Plasma Physics*, 2(1):9, 2018.

[12] TM Rocha Filho, AE Santana, MA Amato, and A Figueiredo. Scaling of the dynamics of homogeneous states of one-dimensional long-range interacting systems. *Physical Review E*, 90(3):032133, 2014.

[13] CR Lourenço and TM Rocha Filho. Scaling of the dynamics of a homogeneous one-dimensional anisotropic classical heisenberg model with long-range interactions. *Physical Review E*, 92(1):012117, 2015.

[14] Edward Ott. *Chaos in dynamical systems*. Cambridge university press, 2002.

[15] Raúl O Vallejos and Celia Anteneodo. Largest lyapunov exponent of long-range xy systems. *Physica A: Statistical Mechanics and its Applications*, 340(1-3):178–186, 2004.

[16] Marie-Christine Firpo and Stefano Ruffo. Chaos suppression in the large size limit for long-range systems. *Journal of Physics A: Mathematical and General*, 34(37):L511, 2001.

[17] Celia Anteneodo and Raúl O Vallejos. Scaling laws for the largest lyapunov exponent in long-range systems: A random matrix approach. *Physical Review E*, 65(1):016210, 2001.

[18] LH Miranda Filho, MA Amato, and TM Rocha Filho. Lyapunov exponent and criticality in the hamiltonian mean field model. *Journal of Statistical Mechanics: Theory and Experiment*, 2018(3):033204, 2018.

[19] LH Miranda Filho, MA Amato, Yves Elskens, and TM Rocha Filho. Contribution of individual degrees of freedom to lyapunov vectors in many-body systems. *Communications in Nonlinear Science and Numerical Simulation*, 74:236–247, 2019.

[20] Lapo Casetti, Roberto Livi, and Marco Pettini. Gaussian model for chaotic instability of hamiltonian flows. *Physical review letters*, 74(3):375, 1995.

[21] Lapo Casetti, Marco Pettini, and EGD Cohen. Geometric approach to hamiltonian dynamics and statistical mechanics. *Physics Reports*, 337(3):237–341, 2000.

[22] Lapo Casetti, Cecilia Clementi, and Marco Pettini. Riemannian theory of hamiltonian chaos and lyapunov exponents. *Physical Review E*, 54(6):5969, 1996.

[23] Lando Caiani, Lapo Casetti, Cecilia Clementi, and Marco Pettini. Geometry of dynamics, lyapunov exponents, and phase transitions. *Physical review letters*, 79(22):4361, 1997.

[24] Marie-Christine Firpo. Analytic estimation of the lyapunov exponent in a mean-field model undergoing a phase transition. *Physical Review E*, 57(6):6599, 1998.

[25] Mickael Antoni and Stefano Ruffo. Clustering and relaxation in hamiltonian long-range dynamics. *Physical Review E*, 52(3):2361, 1995.

[26] Tarcísio N Teles, Fernanda P da C Benetti, Renato Pakter, and Yan Levin. Nonequilibrium phase transitions in systems with long-range interactions. *Physical review letters*, 109(23):230601, 2012.

[27] Alessandro Campa, Thierry Dauxois, and Stefano Ruffo. Statistical mechanics and dynamics of solvable models with long-range interactions. *Physics Reports*, 480(3-6):57–159, 2009.

[28] Markus Deserno. Microcanonical and canonical two-dimensional ising model: An example. *UCLA, USA*, 2004.

[29] Herbert B Callen. *Thermodynamics and an Introduction to Thermostatistics*. John wiley & sons, 1985.

[30] JW Gibbs. The collected works of j. willard gibbs. thermodynamics. new york: Longmans, green and co., 1928. v. 1. p. 55-349. 1982.

[31] Yan Levin, Renato Pakter, Felipe B Rizzato, Tarcísio N Teles, and Fernanda PC Benetti. Nonequilibrium statistical mechanics of systems with long-range interactions. *Physics Reports*, 535(1):1–60, 2014.

[32] Donald Lynden-Bell. Statistical mechanics of violent relaxation in stellar systems. *Monthly Notices of the Royal Astronomical Society*, 136:101, 1967.

[33] Andrea Antoniazzi, Francesco Califano, Duccio Fanelli, and Stefano Ruffo. Exploring the thermodynamic limit of hamiltonian models: Convergence to the vlasov equation. *Physical review letters*, 98(15):150602, 2007.

[34] Julien Barré, Freddy Bouchet, Thierry Dauxois, and Stefano Ruffo. Out-of-equilibrium states as statistical equilibria of an effective dynamics in a system with long-range interactions. *Physical review letters*, 89(11):110601, 2002.

[35] TM Rocha Filho, A Figueiredo, and M A Amato. Entropy of classical systems with long-range interactions. *Physical review letters*, 95(19):190601, 2005.

[36] PH Chavanis. Lynden-bell and tsallis distributions for the hmf model. *The European Physical Journal B-Condensed Matter and Complex Systems*, 53(4):487–501, 2006.

[37] D Mukamel, Stefano Ruffo, and N Schreiber. Breaking of ergodicity and long relaxation times in systems with long-range interactions. *Physical review letters*, 95(24):240604, 2005.

[38] Toshio Tsuchiya, Tetsuro Konishi, and Naoteru Gouda. Quasiequilibria in one-dimensional self-gravitating many-body systems. *Physical Review E*, 50(4):2607, 1994.

[39] F Borgonovi, GL Celardo, M Maianti, and E Pedersoli. Broken ergodicity in classically chaotic spin systems. *Journal of Statistical Physics*, 116(5-6):1435–1447, 2004.

[40] Peilong Chen and MC Cross. Mixing and thermal equilibrium in the dynamical relaxation of a vortex ring. *Physical review letters*, 77(20):4174, 1996.

[41] Yan Levin, Renato Pakter, and Tarcisio N Teles. Collisionless relaxation in non-neutral plasmas. *Physical review letters*, 100(4):040604, 2008.

[42] Tarcísio N Teles, Yan Levin, Renato Pakter, and Felipe B Rizzato. Statistical mechanics of unbound two-dimensional self-gravitating systems. *Journal of Statistical Mechanics: Theory and Experiment*, 2010(05):P05007, 2010.

[43] Michael Joyce and Tirawut Worrakitpoonpon. Quasistationary states in the self-gravitating sheet model. *Physical Review E*, 84(1):011139, 2011.

[44] Tarcísio Nunes Teles, Yan Levin, and Renato Pakter. Statistical mechanics of 1d self-gravitating systems: the core—halo distribution. *Monthly Notices of the Royal Astronomical Society: Letters*, 417(1):L21–L25, 2011.

[45] Fernanda P da C Benetti, Tarcísio N Teles, Renato Pakter, and Yan Levin. Ergodicity breaking and parametric resonances in systems with long-range interactions. *Physical review letters*, 108(14):140601, 2012.

[46] DH Lee and G Grinstein. Strings in two-dimensional classical xy models. *Physical review letters*, 55(5):541, 1985.

[47] Fábio C Poderoso, Jeferson J Arenzon, and Yan Levin. New ordered phases in a class of generalized x y models. *Physical review letters*, 106(6):067202, 2011.

[48] Mickaël Antoni, Yves Elskens, and Caroline Sandoz. Weak turbulence and structure evolution in n-body hamiltonian systems with long-range force. *Physical Review E*, 57(5):5347, 1998.

[49] Arkady Pikovsky, Shamik Gupta, Tarcisio N Teles, Fernanda PC Benetti, Renato Pakter, Yan Levin, and Stefano Ruffo. Ensemble inequivalence in a mean-field x y model with ferromagnetic and nematic couplings. *Physical Review E*, 90(6):062141, 2014.

[50] Edward N Lorenz. Deterministic nonperiodic flow. *Journal of the atmospheric sciences*, 20(2):130–141, 1963.

[51] Steven H Strogatz. *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering*. Westview press, 2014.

[52] Giancarlo Benettin, Luigi Galgani, and Jean-Marie Strelcyn. Kolmogorov entropy and numerical experiments. *Physical Review A*, 14(6):2338, 1976.

[53] Thomas S Parker and Leon Chua. *Practical numerical algorithms for chaotic systems*. Springer Science & Business Media, 2012.

[54] JL Lebowitz, JK Percus, and L Verlet. Ensemble dependence of fluctuations with application to machine computations. *Physical Review*, 153(1):250, 1967.

[55] Eric M Pearson, Timur Halicioglu, and William A Tiller. Laplace-transform technique for deriving thermodynamic equations from the classical microcanonical ensemble. *Physical Review A*, 32(5):3030, 1985.

[56] Moises Fabiano Silva, Tarcisio Marciano Rocha Filho, and Yves Elskens. Critical exponent for the lyapunov exponent and phase transitions–the generalized hamiltonian mean-field model. *Journal of Physics A: Mathematical and Theoretical*, 2020.

[57] John R Ray. Microcanonical ensemble monte carlo method. *Physical Review A*, 44(6):4061, 1991.

[58] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

[59] Moises Fabiano Pereira da Silva Júnior. Comparação de métodos computacionais para o estudo da termodinâmica de sistemas com interações de longo alcance. Master's thesis, Universidade de Brasília, 2016.

[60] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre GR Day, Clint Richardson, Charles K Fisher, and David J Schwab. A high-bias, low-variance introduction to machine learning for physicists. *Physics Reports*, 2019.

[61] Michael A Nielsen. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA:, 2015.

[62] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[63] Astrid S de Wijn. Kolmogorov-sinai entropy for dilute systems of hard particles in equilibrium. *Physical Review E*, 71(4):046211, 2005.

[64] Murray Rosenblatt. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, pages 832–837, 1956.

[65] Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076, 1962.

[66] Bernard W Silverman. *Density estimation for statistics and data analysis*. Routledge, 2018.

[67] David W Scott. On optimal and data-based histograms. *Biometrika*, 66(3):605–610, 1979.

[68] Tarcisio M Rocha Filho. Molecular dynamics for long-range interacting systems on graphic processing units. *Computer Physics Communications*, 185(5):1364–1369, 2014.

[69] Marco Pettini. *Geometry and topology in Hamiltonian dynamics and statistical mechanics*, volume 33. Springer Science & Business Media, 2007.

[70] Yasuhide Sota, Osamu Iguchi, Masahiro Morikawa, Takayuki Tatekawa, and Keiichi Maeda. Origin of scaling structure and non-gaussian velocity distribution in a self-gravitating ring model. *Physical Review E*, 64(5):056133, 2001.

[71] Alessandro Torcini and Mickaël Antoni. Equilibrium and dynamical properties of two-dimensional n-body systems with long-range attractive interactions. *Physical Review E*, 59(3):2746, 1999.

[72] Thierry Dauxois, Vito Latora, Andrea Rapisarda, Stefano Ruffo, and Alessandro Torcini. The hamiltonian mean field model: from dynamics to statistical mechanics and back. In *Dynamics and Thermodynamics of Systems with Long-Range Interactions*, pages 458–487. Springer, 2002.