No estudo de Matrizes Aleatórias

PIMENTA, J. V. A.

30 de setembro de 2023

Resumo

Resumo dos estudos de Iniciação Científica em Matrizes Aleatórias e Simulação de Gases de Coulomb.

Conteúdo

Capítulo 1

Introdução

1.1 Física e Mecânica Estatística

Estaremos lidando o tempo todo no nosso estudo com funções partições, de alguma forma, compartilhada com a física. Entenderemos um pouco mais sobre os desenvolvimentos dos ensembles na termodinâmica.

Em termos gerais os ensembles são sistemas aos quais se impõe vínculos arbitrários. Tanto matematicamente, quanto fisicamente. Da entropia de Shannon podemos deduzir as funções partições apenas adicionando relações de vínculo com multiplicadores de Lagrange. Dos sistemas físicos, usaremos um banho que possa atuar como vínculo.

1.1.1 Entropia de Shannon

Podemos fazer algo um pouco mais matemático. Definiremos a entropia de Shannon, que dá origem às expressões entrópicas gerais para qualquer ensemble.

$$\mathcal{S} = -k_b \sum_i p_i \log p_i$$

Um vínculo

Se impormos um vínculo do tipo

$$\sum_i p_i = 1$$

Podemos usar os multiplicadores de Lagrange para realizar a maximização da nossa função. Pela definição, escrevemos:

$$S(p_i, \lambda) \equiv k_b \sum_{i=1}^{N} p_i \log\left(p_i\right) - \lambda \left(\sum_{i=1}^{N} p_i - 1 \right)$$

Com diferencial

$$\delta S = -k_b \sum_{i=1}^{N} \left(p_i \log p_i + \frac{p_i}{p_i} \delta p_i \right) - \lambda \sum_{i=1}^{N} \delta p_i$$

Resolveremos agora o sistema

$$\begin{cases} \delta S = 0 \\ \sum_i p_i = 1 \end{cases}$$

ou seja,

$$\begin{cases} -k_b(\log p_i + 1) - \lambda \\ \sum_i p_i = 1 \end{cases}$$

Note que $p_i = cte\ \forall i$. Teremos então $p_i = \frac{1}{N}$. A entropia neste caso será

$$S(p_i{}^*) = -k_b \sum_{i=1}^{N} \left(\frac{1}{N} \log \frac{1}{N} \right) = k_b \log N$$

Dois vínculos

Faremos agora a implicação de um novo vínculo

$$\sum_\sigma p_\sigma E_\sigma = U$$

Desenvolvendo a equação

$$S(p_i, \lambda_1, \lambda_2) \equiv k_b \sum_{i=1}^{N} p_i \log(p_i) - \lambda_1 \left(\sum_{i=1}^{N} p_i - 1 \right) - \lambda_2 \left(\sum_\sigma p_\sigma E_\sigma - U \right)$$

Chegaremos no sistema

$$\begin{cases} -k_b(\log p_i + 1) - \lambda_1 - \lambda_2 E_\sigma = 0 \\ \sum_\sigma p_i = 1 \\ \sum_\sigma p_\sigma E_\sigma = U \end{cases}$$

E finalmente, da primeira equação tiramos

$$p_\sigma = e^{AE_\sigma + B} = Me^{-\beta E_\sigma}$$

De forma que podemos reescrever

$$1 = \sum_\sigma Me^{-\beta E_\sigma} = M \sum_\sigma e^{-\beta E_\sigma}$$

Onde nomearemos $M = \frac{1}{\sum_\sigma e^{-\beta E_\sigma}} = \frac{1}{\mathcal{Z}}$ **função partição**. Falta apenas definir β em

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

Para β podemos aplicar o último vínculo (da temperatura térmica)

$$U = \sum_\sigma p_\sigma E_\sigma = \sum_\sigma \left(\frac{1}{Z} e^{-\beta E_\sigma} \right) E_\sigma = \frac{1}{Z} \sum_\sigma E_\sigma e^{-\beta E_\sigma}$$

$$-\frac{1}{Z} \frac{\partial}{\partial \beta} \left(\sum_\sigma e^{-\beta E_\sigma} \right) = -\frac{1}{\mathcal{Z}} \frac{\partial Z}{\partial \beta} = -\frac{\partial (\log Z)}{\partial \beta}$$

De alguma forma esta equação transcendental nos define β.

Três vínculos

Introduziremos um terceiro novo vínculo

$$\sum_{\sigma} p_\sigma N_\sigma = N$$

Da mesma forma, teremos que desenvolver a equação

$$S(p_i, \lambda_1, \lambda_2, \lambda_3) \equiv k_b \sum_{i=1}^{N} p_i \log(p_i) - \lambda_1 \left(\sum_{i=1}^{N} p_i - 1 \right) - \lambda_2 \left(\sum_{\sigma} p_\sigma E_\sigma - U \right) - \lambda_3 \left(\sum_{\sigma} p_\sigma N_\sigma - N \right)$$

Do sistema associado aos vínculos de Lagrange

$$\begin{cases} -k_b(\log p_i + 1) - \lambda_1 - \lambda_2 E_\sigma - \lambda_3 N_\sigma = 0 \\ \sum_\sigma p_i = 1 \\ \sum_\sigma p_\sigma E_\sigma = U \\ \sum_\sigma p_\sigma N_\sigma = N \end{cases}$$

Da primeira equação tiramos

$$p_\sigma = e^{AE_\sigma + BN_\sigma + C} = Me^{-\beta E_\sigma + \beta \mu N_\sigma}$$

De forma que podemos reescrever

$$1 = \sum_{\sigma} Me^{-\beta E_\sigma + \beta \mu N_\sigma} = M \sum_{\sigma} e^{-\beta E_\sigma + \beta \mu N_\sigma}$$

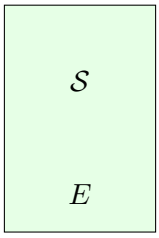
Novamente nomearemos Z em

$$M = \frac{1}{\sum_\sigma e^{-\beta E_\sigma + \beta \mu N_\sigma}} = \frac{1}{Z}$$

a função partição. As outras duas equações nos definem β e μ.

1.1.2 O ensemble Micro-Canônico

O ensemble Micro-Canônico devera ser o mais simples que desenvolveremos. Para este caso a energia é constante e todo microestados devem ser igualmente prováveis pela hipótese ergótica.



Sabemos então que vamos querer minimizar a entropia usual com

$$U(S, V, N) \tag{1.1}$$

Com

$$dU = TdS - PdV + \mu N$$

e, especialmente

$$\rho_E = \sum_\sigma \rho_\sigma = \Omega(E)\rho_\sigma$$

Ou ainda, se $\Omega(E)$ é a quantidade de microestados,

$$\rho_\sigma = \frac{1}{\Omega(E)}$$

De forma que nossa função partição será

$$Z = \sum_\sigma \frac{1}{\Omega(E)} = 1$$

E em relação a energia

$$Z = \sum_E \frac{1}{\Omega(E)}\Omega(E) = \exp\left\{\beta T k_b \log\frac{1}{\Omega(E)}\right\}\Omega(E)$$

Finalmente

$$\log(Z) = 0 = -\frac{S}{k_b} + \log(\Omega(E))$$

Ou melhor

$$S = k_b \log(\Omega(E)) \tag{1.2}$$

1.1.3 O ensemble Canônico

Quando tratamos destes sistemas no ensemble canônico a energia interna não mais será minimizada no nosso sistema termalizado. Introduzimos uma nova grandeza chamada Energia Livre de Helmholtz (F), definida como a transformada de Legendre (discutida no Apêndice B) da energia interna em relação à entropia, ou seja

$$U(S,V,N) \mapsto F(T,V,N) \tag{1.3}$$

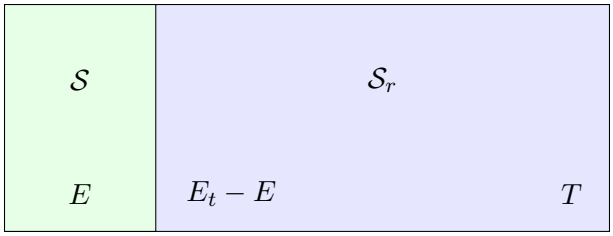
ou ainda mais especificamente

$$F(T,V,N) = U(S(T,V,N),V,N) - TS(T,V,N)$$

Note ainda as diferenciais

$$dU = TdS - PdV + \mu dN$$
$$dF = -SdT - PdV + \mu dN$$

Para o desenvolvimento da função partição estamos atentos aos requisitos do ensemble canônico. Um sistema termalizado por um banho térmico. Nosso sistema completo pode ser representado



Considere que os sistemas estão em contato térmico e o sistema completo, junto com o banho é um sistema isolado de energia E_t. Sabemos que a probabilidade de uma energia no nosso sistema termalizado (ρ_E) é expressa:

$$\rho_E = \sum_\sigma \rho_\sigma = \Omega(E)\rho_\sigma$$

Onde note, ρ_σ é a probabilidade de, dada uma temperatura, um microestado possível. Note por outro lado que podemos expressar:

$$\rho_E = \frac{\Omega(E)\Omega_r(E_t - E)}{\sum_i \Omega(E_i)\Omega_r(E_t - E)}$$

Note que isso é nada mais do que dizer que os microestados são equiprováveis e basta uma contagem (normalizada) para definir a probabilidade. Neste sentido podemos também escrever:

$$\rho_\sigma \propto \Omega_r(E_t - E)$$

Isso é, quanto mais formas o reservatório possa se organizar para determinada energia, mais provável é o microestado associado à energia de $\mathcal{S}$.

$$\begin{aligned}
\rho_\sigma &\propto e^{\beta T K_b \log\left(\Omega_r(E_t - E)\right)} \\
&\propto e^{\beta T\left(\mathcal{S}_r(E_t) - \frac{E}{T}\right)} \\
&\propto e^{-\beta E} e^{\beta T \mathcal{S}_r(E_t)} \\
&\propto e^{-\beta E}
\end{aligned}$$

Onde fizemos a expansão de $k_b \log\left(\Omega_r(E_t - E)\right)$ (entropia) em Taylor (apesar de que, em realidade, apenas $\frac{E}{E_t}$ é pequeno, não necessariamente E) e obtivemos

$$\mathcal{S}_r(E_t - E) \approx \mathcal{S}_r(E_t) + \left(\frac{\partial \mathcal{S}_r}{\partial E}\right)_{E=E_t} (-E)$$

Onde

$$\left(\frac{\partial \mathcal{S}_r}{\partial E}\right)_{E=E_t} = \frac{1}{T}$$

Um sistema termalizado vai querer minimizar essa nova grandeza da energia livre de Helmholtz. Em todo caso iniciaremos com a expressão já deduzida da equação de partição

$$\mathcal{Z} = \sum_\sigma e^{-\beta E_\sigma}$$

Que pode ser reescrito em termos de uma soma na energia

$$\begin{aligned}
\mathcal{Z} &= \sum_E e^{-\beta E}\Omega(E) \\
\mathcal{Z} &= \sum_E e^{-\beta T K_b \log\left(\Omega(E)\right)}\Omega(E)
\end{aligned}$$

Podemos argumentar que $K_b \log\left(\Omega(E)\right) = S(E)$ e usaremos o logaritmo de somas assintóticas (discutido no Apêndice C) para terminar o desenvolvimento. Note

$$\log(\mathcal{Z}) \approx \log\left(\max_x [e^{-\beta(E-TS(E))}]\right)$$
$$\log(\mathcal{Z}) \approx \log\left(e^{-\beta \min_E ((E-TS(E)))}\right)$$

Onde podemos reconhecer pela transformada de Legendre o termo referente à Energia livre de Helmholtz. Tendo assim

$$\log(\mathcal{Z}) \approx -\beta F$$

$$F = -K_b T \log(\mathcal{Z}) \tag{1.4}$$

1.1.4 O ensemble Grão Canônico

Supomos agora nosso sistema novamente controlado por um banho térmico. Desta vez permitiremos a troca de temperatura e partículas. Definiremos uma energia livre tal que $U(S,V,N) \mapsto \Phi(T,V,\mu)$. Chamaremos esta energia livre de Grão Potencial ou Potencial de Landau. Como sempre, definiremos o potencial como uma transformada de Legendre sob a Energia

$$\Phi = U - TS - \mu N \tag{1.5}$$

$$\Phi(T,V,\mu) = U(S(T,V,\mu),V,N(T,V,\mu)) - TS(T,V,\mu) - \mu N(T,V,\mu)$$

e é claro, na diferencial

$$d\Phi = dU - Tds - SdT - \mu dN - Nd\mu$$

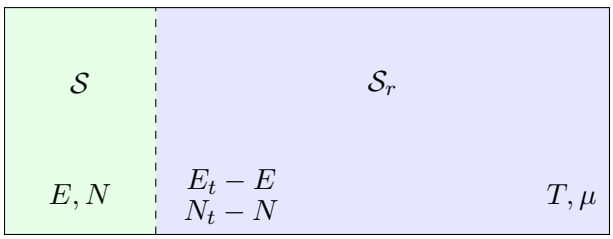
Onde lembramos que $d\Phi = TdS - PdV + \mu dN$ de forma que

$$d\Phi = -SdT - PdV - Nd\mu$$

Ou seja,

$$S = -\left|\frac{\partial \Phi}{\partial T}\right|_{V,N}$$
$$N = -\left|\frac{\partial \Phi}{\partial \mu}\right|_{T,V}$$

Tratamos do seguinte sistema,



Sabemos afirmar que $p_\sigma \propto \Omega_R(E_t - E, N_t - N)$, ou seja, cada microestado do nosso sistema é proporcional às formas que o banho pode se arranjar dado (E,N). Ou seja

$$
\begin{aligned}
p_\sigma &= \exp\left\{\log\left[\Omega_R(E_t - E, N_t - N)\right]\right\} \\
&\propto exp\left\{\frac{1}{k_b}\left[k_b \log\left(\Omega_R(E_T, N_T)\right) - E\frac{\partial}{\partial E'}(k_b log\Omega_R(E', N'))\Big|_{\substack{E'=E_T \\ N'=N_T}} - N\frac{\partial}{\partial N'}(k_b log\Omega_R(E', N'))\Big|_{\substack{E'=E_T \\ N'=N_T}}\right]\right\} \\
&\propto \exp\left\{\frac{1}{k_b}\left[-E\frac{\partial}{\partial E'}(k_b log\Omega_R(E', N'))\Big|_{\substack{E'=E_T \\ N'=N_T}} - N\frac{\partial}{\partial N'}(k_b log\Omega_R(E', N'))\Big|_{\substack{E'=E_T \\ N'=N_T}}\right]\right\}
\end{aligned}
$$

Onde retiramos o termo que não depende do nosso sistema de interesse e será constante, agregando ele na proporcionalidade. Agora faremos uso da ideia da entropia como $S = k_b \log\left(\Omega_R(E', N')\right)$ para escrever a relação acima em termos da temperatura e potencial químico. Usando das derivadas parciais da entropia em $dS = \frac{1}{T}dU + \frac{P}{T}dV - \frac{\mu}{T}dN$,

$$
\begin{aligned}
p_\sigma &\propto \exp\left\{\frac{1}{k_b}\left[-E\frac{1}{T} - N\frac{\mu}{T}\right]\right\} \\
&\propto e^{-\beta E + \beta\mu N}
\end{aligned}
$$

e

$$
p_\sigma = \frac{1}{\Xi}e^{-\beta E + \beta\mu N} \tag{1.6}
$$

Onde

$$
\Xi = \sum_\sigma e^{-\beta E + \beta\mu N} \tag{1.7}
$$

O log da nossa função partição deve resultar em uma expressão de energia livre.

$$
\begin{aligned}
\log\Xi &= \log\left(\sum_\sigma e^{-\beta E_\sigma + \beta\mu N_\sigma}\right) \\
&= \log\left(\sum_{E,N}\Omega(E, N)e^{-\beta E + \beta\mu N}\right) \\
&= \log\left(\sum_{E,N}\exp\left\{\frac{k_b}{k_b}\log\Omega(E, N)\right\}\exp\left\{(-\beta E + \beta N\mu)\right\}\right) \\
&= \log\left(\sum_{E,N}\exp\left\{\frac{T}{k_b T}S - \beta E + \beta N\mu\right\}\right) \\
&= \log\left(\sum_{E,N}e^{\beta(E - TS - N\mu)}\right)
\end{aligned}
$$

Para a aproximação desta expressão vamos considerar que o logaritmo de uma somatória pode ser aproximada por seu termo máximo,

$$
\approx \log e^{\beta(E^* - TS(E^*, N^*) - N^*\mu)} = \beta(E^* - TS(E^*, N^*) - N^*\mu)
$$

Ou seja,

$$\log \Xi = \beta(E^* - TS(E^*, N^*) - N^*\mu) = -\beta\Phi$$

e finalmente,

$$\Phi = -k_b T \log \Xi \tag{1.8}$$

Capítulo 2

Procura-se autovalores

A distribuição dos autovalores de uma matriz aleatória Gaussiana $N \times N$ é dada por:

$$\rho(x_1, x_2, \ldots, x_N) = \frac{1}{\mathcal{Z}_{N,\beta}} e^{-\frac{1}{2}\sum_{i=1}^{N} x_i^2} \prod_{j<k} |x_j - x_k|^\beta \tag{2.1}$$

onde a constante de normalização $\mathcal{Z}$ é dada por

$$\mathcal{Z}_{N,\beta} = (2\pi)^{\frac{N}{2}} \prod_{j=1}^{N} \frac{\Gamma(1 + j\frac{\beta}{2})}{\Gamma(1 + \frac{\beta}{2})} \tag{2.2}$$

β é o index de Dyson e indica o ensemble que utilizamos. Mais importante é notar que existe na nossa distribuição um fator de repulsão e um fator de campo de mínimo de energia em $x = 0$. De forma que mesmo centradas em 0 as partículas mantém uma distância entre si pela repulsão 'Coulombiana'.

Se quisermos computar o histograma dos autovalores podemos tomar a marginal

$$\rho(x) = \int \cdots \int dx_1 \cdot dx_2 \cdot \cdots \cdot dx_N \rho(x, x_2, \cdots, x_N) \tag{2.3}$$

Ou seja, o perfil da função partição de uma nova partícula dada o posicionamento das outras. Para a Equação 2.1, a densidade espectral $\mathbb{E}[n(x)] = \rho(x)$ é dada pelo limite

$$\lim_{N\to\infty} \sqrt{\beta N}\rho(\sqrt{\beta N}x) = \rho_{sc}(x) \tag{2.4}$$

Onde $\rho_{sc}(x) = \frac{1}{\pi}\sqrt{2 - x^2}$ compõe a famosa lei do semi-círculo de Wigner representada na Figura 2. Os pontos $\pm\sqrt{2\beta N}$ são as bordas espectrais. Um esquema útil para organização dos multiplos ensembles é a classificação de Layman. Denomina-se

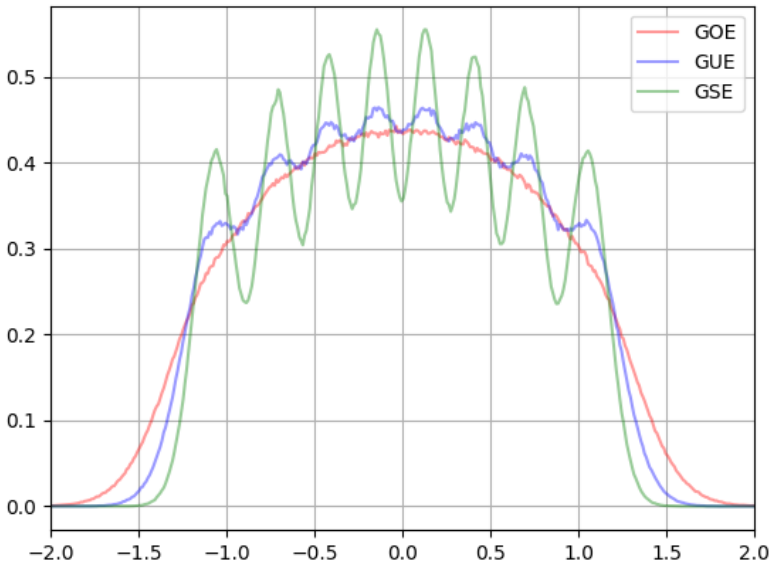
1. **Entradas Independentes**: Adicionada a necessidade de simetria, matrizes deste tipo são chamadas matrizes de Wigner.

2. **Invariantes por rotação**: Quaisquer duas matrizes que são relacionadas por uma transformação $\hat{H}' = \hat{U}\hat{H}\hat{U}^{-1}$ ocorrerão com a mesma probabilidade.

Só existe um tipo especial de matriz que se encontra na intersecção e são as classes Gaussianas.

Em geral no mundo das matrizes aleatórias três classes são muito importantes e serão atores centrais no nosso estudo. São as três classes associadas às matrizes que tem entradas gaussianas. Estas são: O Ensemble Gaussiano Ortogonal (GOE), O Ensemble Gaussiano Unitário (GUE) e o Ensemble Gaussiano Simplético (GSE). Em suma, eles tratam, em ordem, de matrizes com entradas gaussianas reais, complexas e quaterniônicas. Seus

nomes estão relacionados com a matriz necessária para a transformação de diagonalização das matrizes. Unitária para o caso complexo, por exemplo. Seus autovalores possuem distribuições distintas (ao menos para a escala usada) e é ilustrada abaixo na Figura (2).



Mais desenvolvimento sobre a forma desta distribuição e as diferenças será feito posteriormente. Usaremos a referência [4] para a maior parte dos desenvolvimentos. Algum material interessante pode ser consultado em um livro de física em [5].

2.1 Porquê exponencial?

2.1.1 Shannon-Quem?

O uso da p.d.f gaussiana pode ser justificado de algumas formas. Uma primeira abordagem a ser explorada é a de maximização entrópica ou minimização de informação similar aos trabalhos de Shannon-Kinchin. Definiremos uma grandeza $\mathcal{I}[\mathcal{P}(\hat{H})]$ associada à uma p.d.f tal que:

$$\mathcal{I}[\mathcal{P}(\hat{H})] = -\int d\mu(\hat{H})\mathcal{P}(\hat{H})\ln\mathcal{P}(\hat{H})$$

Que é uma extensão natural da definição discreta de informação $-\sum_{l=1}^{m} p_m \ln p_m$. Agora argumentaremos algo parecido com os argumentos usados em termodinâmica de maximização de entropia. Diremos que a incerteza sobre as matrizes será máxima, ou seja, teremos a maior aleatoriedade das matrizes quando a entropia for maximizada e a informação, minimizada. Assim como na entropia física impomos um vínculo de energia constante, aqui faremos algo do tipo $E(\operatorname{Tr}\hat{H}) = b$ e $E((\operatorname{Tr}\hat{H})^2) = a > 0$. Vamos introduzir esses vinculos como multiplicadores de lagrange com multiplicadores v_1 e v_2.

$$\mathcal{I}[\mathcal{P}(\hat{H})] = -\int d\mu(\hat{H})\mathcal{P}(\hat{H})\left(\ln\mathcal{P}(\hat{H}) - v_1\operatorname{Tr}\hat{H} - v_2\operatorname{Tr}\hat{H}^2\right)$$

que tem diferencial

$$\delta\mathcal{I}[\mathcal{P}(\hat{H})] = -\int d\mu(\hat{H})\delta\mathcal{P}(\hat{H})\left(1 + \ln\mathcal{P}(\hat{H}) - v_1\operatorname{Tr}\hat{H} - v_2\operatorname{Tr}\hat{H}^2\right) = 0$$

Que só vai ser mínimo se

$$\mathcal{P}(\hat{H}) \propto e^{-v_1 \operatorname{Tr} \hat{H} - v_2 \operatorname{Tr} \hat{H}^2}$$

Onde os multiplicadores são unicamente definidos pelas constantes do vínculo. Esse fato motiva de alguma forma o estudo da p.d.f gaussiana para as matrizes.

2.2 Independência ou Morte

Consideremos matrizes com entradas independentes. Qual a função densidade de probabilidade (F.P.D.) da matriz simétrica $\hat{H}_s$? Devemos fazer separadamente a diagonal da seção triangular que formos usar e teremos

$$\rho((\hat{H}_s)_{11}, \ldots, (\hat{H}_s)_{NN}) = \prod_{i=1}^{N} \left[\frac{e^{\frac{-(H_s)_{ii}^2}{2}}}{2\pi} \right] \prod_{i<j} \left[\frac{e^{-(H_s)_{ij}^2}}{\sqrt{\pi}} \right]$$

Que é a função distribuição da matriz. A desimetria da diagonal e da não diagonal nos permite escrever a distribuição como

$$\mathcal{P}(\mathcal{H}) = C_2 e^{-\frac{1}{2\sigma^2} \operatorname{Tr} \mathcal{H}^2} \tag{2.5}$$

Mais geralmente

$$\mathcal{P}(\mathcal{H}) = C_\beta e^{-\frac{\beta}{4\sigma^2} \operatorname{Tr} \mathcal{H}^2} \tag{2.6}$$

Suponha que queremos derivar a equação da distribuição dos autovalores para tais matrizes. Começamos com $\mathcal{H}_{N\times N}$

$$\begin{bmatrix} x_1 & x_{n+1} & \cdots & x_{2n} \\ x_{n+1} & x_2 & \vdots & \vdots \\ \vdots & \cdots & \ddots & \vdots \\ x_{2n} & \cdots & \cdots & x_n \end{bmatrix}$$

Notemos que teremos $x_1, x_2, \ldots, x_n \sim \mathcal{N}(0,1)$ e $x_{n+1}, \cdots, x_{n^2} \sim \mathcal{N}(0, \frac{1}{2})$. Qual a distribuição $p(s)$ de $s = \lambda_2 - \lambda_1$? Sabemos os autovalores soluções do polinômio característico

$$\lambda^2 - \operatorname{Tr} \mathcal{H}_s \lambda + \det \mathcal{H}_s$$

O resultado será uma função $s(x_1, x_2, \ldots, x_{n^2})$. Para a qual escrevemos

$$p(s) = \int_{-\infty}^{\infty} dx_1 \cdots dx_{n^2} \prod_{i=1}^{n} \frac{e^{-\frac{x_i^2}{2}}}{\sqrt{2\pi}} \prod_{i=n+1}^{n^2} \frac{e^{-x_i^2}}{\sqrt{\pi}} \delta(s - s(x_1, x_2, \ldots, x_{n^2}))$$

Que, desenvolvido, chega à

$$p(s) = -\frac{s}{2} e^{-f(s^2)}$$

Ou seja, teremos um resultado em que as partículas parecem se repelir!

2.3 Uma medida à Hermitiana

Consideraremos no nosso estudo para referencia matrizes quadradas de entradas complexas com dimensão N. Nosso objetivo é afinal ter uma forma de mensurar a distribuição de autovalores e para isso, faremos os seguintes desenvolvimentos.

Consideremos inicialmente um espaço de matrizes com entradas complexas $2N^2$ dimensional. Contido neste espaço temos um espaço de maior interesse correspondente ao espaço das matrizes *hermitianas* de dimensão N^2. A escolha do subespaço está relacionada com o fato que matrizes hermitianas são diagonalizáveis e a distribuição de seus autovalores estará diretamente relacionada (com uma mudança de base) à distribuição do traço da matriz diagonalizada. Note que para a matriz diagonal ter a mesma medida que nossa matriz inicial, nossa medida deve ser invariável por rotação.

Mais detalhadamente podemos escrever nossa matriz hermitiana $\hat{H}$ como

$$\hat{H} = \hat{U}\hat{\Lambda}\hat{U}^{-1} \ , \ \hat{\Lambda} = diag(\lambda_1, \ldots, \lambda_2) \ , \ \hat{U} \cdot \hat{U}^* = I$$

onde, claro, $\hat{\Lambda}$ é diagonal de autovalores e $\hat{U}$ é unitária e com colunas equivalentes aos autovetores de $\hat{H}$. Em geral, o conjunto de matrizes degeneradas tem medida nula e não é uma preocupação. Um cuidado deve ser tomado. A correspondência $\hat{H} \implies (\hat{U}\ U(N), \hat{\Lambda})$ não é injetora, podemos tomar $\hat{U}_1\hat{\Lambda}\hat{U}_1^{-1} = \hat{U}_2\hat{\Lambda}\hat{U}_2^{-1}$ se $\hat{U}_1^{-1}\hat{U}_2 = diag(e^i\phi_1, \ldots, e^i\phi_N)$ para qualquer escolha de fases $(\phi_1, \ldots, \phi_N)$. Para restringir nosso problema e tornar a função injetiva será necessário considerar as matrizes unitárias ao espaço de coset $U(N)/U(1) \times \cdots \times U(1)$ [1]. Uma outra restrição necessárias é ordenas os autovalores, ou seja, $\lambda_1 < \cdots < \lambda_n$. Temos que reescrever agora a medida $d\mu(\hat{H})$ em função de auvalores e da $\hat{U}$ de autovetores.

Para resumir o desenvolvimento, alguns resultados serão diretamente enunciados. Essa seção pode ser encontrada no relatório [2]. Em especial recuperaremos o elemento de distância e volume no subespaço que vamos tratar

$$(ds)^2 = \operatorname{Tr} d\hat{H} d\hat{H}^* = \sum_i (dx_{ii})^2 + 2\sum_{i<j} \left[(dx_{ij})^2 + (dy_{ij})^2\right] \tag{2.7}$$

$$d\mu(\hat{H}) = 2^{\frac{N(N-1)}{2}} \prod_i dx_{ii} \prod_{i<j} dx_{ij} dy_{ij} \tag{2.8}$$

Ambos vem de um desenvolvimento da métrica do espaço discutido. Note que nossa medida de comprimento é invariante em respeito à automorfismos (Calcule $\operatorname{Tr} H^2$). Especificamente, se tomarmos os elementos (2.7) e (2.8) na decomposição espectral, obteremos

$$(ds)^2 = \sum_i (d\lambda)^2 + \sum_{i<j} (\lambda_i - \lambda_j)^2 \overline{\delta U_{ij}} \delta U_{ij} \tag{2.9}$$

e

$$d\mu(\hat{H}) = \prod_{i<j} (\lambda_i - \lambda_j)^2 \prod_i d\lambda_i \times d\mu(\hat{U}) \tag{2.10}$$

Tendo a medida de integração pronta, podemos definir uma F.D.P $\mathcal{P}(\hat{H})$ neste espaço de matizes hermitianas tal que $\mathcal{P}(\hat{H})d\mu(\hat{H})$ é a probabilidade da matriz $\hat{H}$ estar no volume $d\mu(\hat{H})$. Queremos que nossa função seja invariante à rotação, ou seja, $\mathcal{P}(\hat{H}) = \mathcal{P}(\hat{U}^*\hat{H}\hat{U})$.

Conhecer os N primeiros traços $(\operatorname{Tr} \hat{H}^n)$ de $\hat{H}$ define unicamente o polinômio característico e junto com ele, os autovalores. Especificamente tomaremos

[1] Não tenho muita ideia de espaços de Coset. Pelo que entendo, existe um espaço onde toda $\hat{U}$ pode ser representada por $\hat{U}_c\hat{U}_d$, onde $\hat{U}_c$ compõe o espaço de coset e $\hat{U}_d$ é uma matriz diagonal unitária. Dessa forma matrizes equivalentes são aquelas que multiplicadas por $\hat{U}_d$ tem um mesmo resultado.

$$\mathcal{P}(\hat{H}) = Ce^{-\operatorname{Tr} Q(\hat{H})} \tag{2.11}$$

Onde Q deve ser um polinômio de até ordem $2j \leq N$ suficiente para garantir a convergência de

$$\mathcal{Z}_n = \int_{\mathcal{H}_n} e^{-\operatorname{Tr} Q(\hat{M})} d\hat{M}$$

Comumente uma condição suficiente é

$$\lim_{x \to \pm\infty} \frac{Q(x)}{\ln(1+x^2)} = \infty$$

Mas em especial, se tomarmos

$$Q(x) = ax^2 + bx + c$$

Nossa medida tomará a forma

$$\mathcal{P}(\hat{H}) = e^{-a\left[\sum_i x_{ii}^2 + 2\sum_{i<j}[x_{ij}^2 + y_{ij}^2]\right]} e^{-b\sum_i x_{ii}} e^{-cN} \tag{2.12}$$

$$= e^{-cN} \prod_{i=1}^{N} \left(e^{-ax_{ii}^2 - bx_{ii}}\right) \prod_{i<j} e^{-2ax_{ij}^2} \prod_{i<j} e^{-2ay_{ij}^2} \tag{2.13}$$

Onde podemos notar que a distribuição de probabilidade da matriz $\hat{H}$ pode ser representados por fatores independentes, cada um de forma gaussiana. Para este potencial, temos uma conexão entre as matrizes de entrada independentes e as matrizes invariáveis por rotação. Lembre-se que para as variáveis serem independentes $\mathcal{P}$ deve ter a forma $\mathcal{P} = Ce^{-\left(a \operatorname{Tr} \hat{H}^2 + b \operatorname{Tr} \hat{H} + cN\right)}$ para constantes $a > 0, b, c$. Em nota, sabemos então

$$e^{\operatorname{Tr} V(\hat{H})} d\mu(\hat{H}) = e^{-\sum_j V(\lambda_j)} \prod_{i<j} (\lambda_i - \lambda_j)^2 d\mu(\lambda) d\mu(\hat{U})$$

ou mais geralmente para o ensemble com

$$\frac{1}{\tilde{\mathcal{Z}}_n} e^{\operatorname{Tr}(V(\hat{M}))} dM$$

Dado λ_j os autovalores

$$\operatorname{Tr}(V(\hat{M})) = -\sum_{j=1}^{n} V(\lambda_j)$$

e finalmente podemos escrever

$$E[f] = \int_{\mathcal{H}_n} f(\hat{M}) e^{-\operatorname{Tr}(Q(\hat{M}))} d\hat{M} \tag{2.14}$$

$$= \frac{1}{\mathcal{Z}} \int \cdots \int f(\lambda_1, \ldots, \lambda_n) \prod_{i<j} (\lambda_i - \lambda_j)^2 \prod_{j=1}^{n} e^{-Q(\lambda_j)} d\lambda_1 \ldots d\lambda_n \tag{2.15}$$

Assim, a probabilidade conjunta nas matrizes induz uma densidade de probabilidade de autovalores

$$\frac{1}{\mathcal{Z}_n} \prod_{i<j} (\lambda_i - \lambda_j)^2 \prod_{j=1}^{n} e^{Q(\lambda_j)} \tag{2.16}$$

Alguns resultados foram resgatadas da nota do autor em [3].

Capítulo 3

Movimento Browniano

3.1 Processo Pontual

Um processo pontual pode ser interpretado como um conjunto aleatório de pontos ou como a medida de probabilidade associada a esse conjunto. Um processo pontual possui n pontos se

$$\mathcal{P}(\#X = n) = 1$$

Onde X é um conjunto enumerável de $\mathcal{X}$ ($\mathbb{R}$, $\mathbb{Z}$ ou um subconjunto destes). O conjunto de todas configurações possíveis é denominado $Conf(\mathcal{X})$. Se $P(x_1, \ldots, x_n)$ é uma função de densidade de probabilidade em $\mathbb{R}^n$ invariante por permutações

$$\mathbb{R}^n \to Conf(\mathbb{R})$$
$$(x_1, \ldots, x_n) \mapsto X = x_1, \ldots, x_n$$

define naturalmente um processo pontual com n pontos.

3.1.1 Poisson & fries

Tome $\{N(t)\}$ o número de eventos no intervalo de tempo $]0, t]$. $\{N(t)\}$ é um processo estocástico (de contagem). Se o processo de Poisson possui $\lambda > 0$, para um elemento fiox do espaço amostral a variável aleatória N assume valor k no tempo t com probabilidade

$$\mathcal{P}[N(t) = k] = \frac{(\lambda t)^k e^{-\lambda t}}{k!} \tag{3.1}$$

Onde λ é o número esperado de chagadas por unidade de tempo. Agora, como um processo pontual, a probabilidade de n eventos no intervalo $]a, b]$ é

$$\mathcal{P}(N]a, b] = n) = \frac{(\lambda(b-a))^n e^{-\lambda(b-a)}}{n!}$$

Podemos usar a independência de cada evento de Poisson em intervalos disjuntos para escrever

$$\mathcal{P}(N]a_1, b_1] = n_1, \ldots, N]a_k, b_k] = n_k) = \prod_{i=1}^{k} \frac{(\lambda(b_i - a_i))_i^n e^{-\lambda(b_i - a_i)}}{n_i!}$$

Podemos escrever para uma função f mensurável em $\mathbb{R}$

$$\sum_{x_i \in \mathcal{X}} f(x_i) = \int f(x) dN(x)$$

Onde a medida dN é

$$dN(x) = \sum_{x_i \in \mathcal{X}} \delta_{x_i}(x)$$

Onde notamos que podemos interpretar tanto quanto uma soma de um processo pontual quanto uma medida de probabilidade.

3.1.2 Funcão Correlação

Definimos uma variável aleatória N anteriormente. Naturalmente, poderíamos estar interessados em sua esperança. Mais especificamente, podemos procurar a esperança do número de pontos de uma ocnfiguração dentro de um intervalo $A \subset \mathbb{R}$.

$$A \mapsto \mathbb{E}[N(A)] = \mathbb{E}[\#(A \cap X)]$$

Que pode ser interpretada como uma medida com densidade p_1

$$\mathbb{E}[\#(A \cap X)] = \int_A p_1(x)dx \tag{3.2}$$

A equação 3.2 é conhecida como *função de correlação de 1 ponto*. Em grosso modo, $p_1(x)$ é a probabilidade de haver um ponto da configuração entre x e $x + dx$. Seja um configuração simples $X = \{x_1, \ldots, x_n\}$ e intervalos disjuntos na reta $A_1, A_2, \ldots, A_n$,

$$\int_A \cdots \int_A \rho_n(x_1, \ldots, x_n)dx_1, \ldots, dx_n = \mathbb{E}\left(\prod_{j=1}^{k} \#(X \cap A_j)\right)$$

é o número esperado de n-uplas $(x_1, x_2, \ldots, x_n) \in A_1 \times \cdots \times A_n$ tais que $x_i \in A_i, i = 1, \ldots, n$. Seja $\mathbb{P}(x_1, \ldots, x_n)$ uma densidade de probabilidade em $\mathbb{R}^n$, então o processo pontual de n pontos gerado possui funções de correlação dadas por

$$p_k(x_1, \ldots, x_k) = \frac{n!}{(n-k)!} \int \cdots \int \mathbb{P}(x_1, \ldots, x_n)dx_{k+1} \ldots dx_n$$

3.1.3 Pontual Determinantal

Um processo pontual vai ser chamado determinantal se dada uma função de correlação ρ_n, existe um núcleo $K(x, y)$ conhecido como núcleo de correlação tal que

$$\rho_n(x_1, \ldots, x_n) = det[K(x_i, x_j)]_{i,j=1}^n \tag{3.3}$$

onde

$$[K(x_i, x_j)]_{i,j=1}^n = \begin{bmatrix} K(x_1, x_1) & K(x_1, x_2) & \ldots & K(x_n, x_n) \\ K(x_2, x_1) & K(x_2, x_2) & \ldots & K(x_n, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ K(x_n, x_1) & K(x_n, x_2) & \ldots & K(x_n, x_n) \end{bmatrix}$$

Nos casos de baixa dimensão

$$p_1(x_1) = K(x_1, x_1), \qquad p_2(x_1, x_2) = \begin{vmatrix} K(x_1, x_1) & K(x_1, x_2) \\ K(x_2, x_1) & K(x_2, x_2) \end{vmatrix}$$

Para que o núcleo satisfaça a equação 3.3 enunciaremos um resultado de interesse.

Teorema 1 *Seja K um núcleo tal que*

(a) $\int K(x,x) = n \in \mathbb{N}$,

(b) Para todo $x_1, x_2, \ldots, x_n \in \mathbb{R}$, *o determinante é não negativo*

(c) K possui a propriedade de ***núcleo reprodutor****, isto é;*

$$K(x,y) = \int_{-\infty}^{\infty} K(x,s)K(s,y)ds$$

Então

$$P(x_1, \ldots, x_n) = \frac{1}{n!} det[K(x_i, x_j)]_{i,j=1}^{n}$$

será uma densidade de probabilidade em $\mathbb{R}$ *cujo processo de n pontos associado é determinantal.*

3.2 Emsemble Biortogonal

Um n-ponto processo é um ensemble biortogonal se existem duas sequências $f_1, \ldots, f_n$ e $g_1, \ldots g_n$ em $L^2(R)$ e uma constante $\mathcal{Z}_n \neq$ tais que:

$$\mathcal{P}(x_1, \ldots, x_n) = \frac{1}{\mathcal{Z}_n} \det\left[f_i(x_j)\right]_{i,j=1}^{n} \cdot \det\left[g_i(x_j)\right]_{i,j=1}^{n}$$

Onde todo f_i e g_i é independente nos i's. Pode-se mostrar que se

$$\phi_j \in span(f_1, \ldots, f_n) \quad \psi_j \in span(g_1, \ldots, g_n)$$

tais que

$$\int_{-\infty}^{\infty} \phi_k(c)\psi_j(x)dx = \delta_{jk}$$

então

$$K_n(x,y) = \sum_{j=1}^{n} \phi_k(c)\psi_j(x)$$

onde $K_n(x,y)$ é um *Kernel* tal que

$$\mathcal{P}(x_1, \ldots, x_n) = \frac{1}{n!} \det\left[K_n(x_i, x_j)\right]_{i,j=1}^{n}$$

O processo é determinado e K_n é o kernel de correlação.

3.3 Karlin-McGregor

3.3.1 O teorema

Exploraremos os caminhos não cruzantes providos por processos de Markov. Considere uma partícula de movendo com uma regra qualquer, vamos descrever esse movimento de forma que denotaremos $p_t(a;x)$ a densidade de probabilidade de transição; isto é, a chance uma partícula em a ir para x em um próximo momento. Um teorema clássico enuncia a probabilidade de um certo número de caminhos não se intersectarem passado um tempo t.

O teorema diz: Considere $X_1(t), \ldots, X_n(t)$ cópias independentes de um processo forte de Markov com caminhos condicionados tais que

$$X_j(0) = a_j$$

onde $a_1 < a_2 < \cdots < a_n$ são valores dados. Notamos novamente $p_t(x, y)$ ser a densidade do processo de transição. Vamos definir regiões $E_1, E_2, \ldots, E_n$ onde E's vizinhos não se intersectam. Temos

$$\int_{E_1} \cdots \int_{E_n} \det\left[p_t(a_i, x_j)\right]_{i,j=1}^{n} dx_1 \ldots dx_n$$

vai ser a probabilidade de que os caminhos não tenham se intersectados no intervalo de tempo $[0, t]$ e $X_j(t)$ nos intervalos correspondentes. A demonstração está em [3]. Note que temos

$$\int_{E_1} \cdots \int_{E_n} \det\left[p_t(a_i, x_j)\right]_{i,j=1}^{n} dx_1 \ldots dx_n$$

$$= \int_{E_1} \cdots \int_{E_n} \begin{vmatrix} p_t(a_1, x_1) & p_t(a_2, x_1) & \ldots & p_t(a_{n-1}, x_1) & p_t(a_n, x_1) \\ p_t(a_1, x_2) & p_t(a_2, x_2) & \ldots & p_t(a_{n-1}, x_2) & p_t(a_n, x_2) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ p_t(a_1, x_{n-1}) & p_t(a_2, x_{n-1}) & \ldots & p_t(a_{n-1}, x_{n-1}) & p_t(a_n, x_{n-1}) \\ p_t(a_1, x_n) & p_t(a_2, x_n) & \ldots & p_t(a_{n-1}, x_n) & p_t(a_n, x_n) \end{vmatrix} dx_1 \ldots dx_n \tag{3.4}$$

$$= \sum_{\sigma} sgn(\sigma) \prod_{j=1}^{n} p_t(a_j, E_{\sigma(j)}) \tag{3.5}$$

$$= \sum_{\sigma} sgn(\sigma) \mathcal{P}(A_\sigma) \tag{3.6}$$

Onde denotamos

$$p_t(a_j, E_{\sigma(j)}) = \int_{E_j} p_t(a_i, x_j) dx_j$$

σ é uma permutação de $1, \ldots, n$ e A_σ é o evento que $X_j(t) \in E_{\sigma(j)}$ para todo j. Os caminhos devem ser independentes para (3.6).

De alguma forma o determinada permuta os caminhos em todas ordens possíveis e calcula a probabilidade de todos se manterem nos intervalos adequados. Um exemplo de baixas dimensões pode mostrar que

$$\begin{vmatrix} p_t(a_1, x_1) & p_t(a_2, x_1) & p_t(a_3, x_1) \\ p_t(a_1, x_2) & p_t(a_2, x_2) & p_t(a_3, x_2) \\ p_t(a_1, x_3) & p_t(a_2, x_3) & p_t(a_3, x_3) \end{vmatrix} = \tag{3.7}$$

$$+ p_t(a_1, x_1) p_t(a_2, x_2) p_t(a_3, x_3) \tag{3.8}$$

$$+ p_t(a_2, x_1) p_t(a_3, x_2) p_t(a_1, x_3) \tag{3.9}$$

$$+ p_t(a_3, x_1) p_t(a_1, x_2) p_t(a_2, x_3) \tag{3.10}$$

$$- p_t(a_3, x_1) p_t(a_2, x_2) p_t(a_1, x_3) \tag{3.11}$$

$$- p_t(a_2, x_1) p_t(a_1, x_2) p_t(a_3, x_3) \tag{3.12}$$

$$- p_t(a_1, x_1) p_t(a_3, x_2) p_t(a_2, x_3) \tag{3.13}$$

Logo

$$\int_{E_1}\cdots\int_{E_n}\det\left[p_t(a_i,x_j)\right]_{i,j=1}^{n}dx_1\ldots dx_n = + p_t(a_1,E_1)p_t(a_2,E_2)p_t(a_3,E_3) \tag{3.14}$$

$$+ p_t(a_2,E_1)p_t(a_3,E_2)p_t(a_1,E_3) \tag{3.15}$$

$$+ p_t(a_3,E_1)p_t(a_1,E_2)p_t(a_2,E_3) \tag{3.16}$$

$$- p_t(a_3,E_1)p_t(a_2,E_2)p_t(a_1,E_3) \tag{3.17}$$

$$- p_t(a_2,E_1)p_t(a_1,E_2)p_t(a_3,E_3) \tag{3.18}$$

$$- p_t(a_1,E_1)p_t(a_3,E_2)p_t(a_2,E_3) \tag{3.19}$$

Onde somamos os casos onde as partículas se matém ordenadas e subtraímos os casos onde elas se cruzam.

3.3.2 Consequências

Considere n cópias do processo de Markov condicionado para começar em $t = 0$ nas determinadas posições $a_1 < a_2 < \cdots < a_n$. Se condicionarmos estes processos para não intersectar no intervalo $[0, t]$, o teorema vai nos dizer que os caminhos em um tempo t vão ter uma densidade de probabilidade conjunta

$$\frac{1}{\mathcal{Z}_n}\det\left[p_t(a_i,x_j)\right]_{i,j=1}^{n}$$

Mas este não pode ser considerado um processo pontual determinado. Não é expresso por um produto de determinantes. Isso pode ser ajeitado se considerarmos um tempo $T > t$ no nosso processo. Tomaremos $b_1, b_2, \ldots, b_n$ posições finais e condicionaremos os caminhos a não intersectar no intervalo $[0, T]$ com $X_j(0) = a_j$ e $X_j(T) = b_j$ para todos. É possível mostrar que a distribuição conjunta deles será

$$\frac{1}{\mathcal{Z}'_n}\det\left[p_t(a_i,x_j)\right]_{i,j=1}^{n}\det\left[p_{T-t}(x_i,b_j)\right]_{i,j=1}^{n}$$

Que será biortogonal com as funções

$$f_j = p_t(a_j,x)\ ;\ g_j = p_{T-t}(x,b_j)$$

E nosso caso de interesse é quando $a_j \to a$ e $b_j \to b$. Note que usando as duas funções podemos forçar que o movimento browniano se inicie em um ponto e encerre em outro determinado. Em uma, reverteremos o tempo e, nos limites 0 e T, forçaremos que apenas uma das funções seja predominante de forma que a posição inicial de cada uma prevaleça. Podemos impor a posição inicial e final do movimento. No caso browniano teremos

$$p_t(a,x) = \frac{1}{\sqrt{2\pi t}}e^{-\frac{(x-a)^2}{2t}}$$

No caso dos limites de $a_i \to a$ e $b_j \to b$ ficamos, por consequência que quando mais de um a_i ou b_j convergem ao mesmo ponto teremos $\mathcal{Z} \to 0$ pela sua dependência destes termos (eles não serão mais 'ortogonais', pontos distintos). Por exemplo, para o limite

$$\lim_{a_n\to a_{n-1}}\frac{\det\left[p_t(a_i,x_j)\right]_{i,j=1}^{n}\det\left[p_{T-t}(x_i,b_j)\right]_{i,j=1}^{n}}{\mathcal{Z}'_n}$$

Queremos aplicar L'Hôpital para resolver a indeterminação. Para o determinante podemos notar que

$$\frac{\partial}{\partial a_n}\begin{vmatrix} p_t(a_1,x_1) & \dots & p_t(a_1,x_n) \\ \vdots & \ddots & \vdots \\ p_t(a_n,x_1) & \dots & p_t(a_n,x_n) \end{vmatrix} = \frac{\partial}{\partial a_n}[p_t(a_n,x_1)\left|\dots\right| + \dots + p_t(a_n,x_n)\left|\dots\right|]$$

$$[\frac{\partial}{\partial a_n}p_t(a_n,x_1)\left|\dots\right| + \dots + \frac{\partial}{\partial a_n}p_t(a_n,x_n)\left|\dots\right|]$$

$$= \begin{vmatrix} p_t(a_1,x_1) & \dots & p_t(a_1,x_n) \\ \vdots & \ddots & \vdots \\ \frac{\partial}{\partial a_n}p_t(a_n,x_1) & \dots & \frac{\partial}{\partial a_n}p_t(a_n,x_n) \end{vmatrix}$$

No limite que estamos tratando:

$$\begin{vmatrix} p_t(a_1,x_1) & \dots & p_t(a_1,x_n) \\ \vdots & \ddots & \vdots \\ \frac{\partial}{\partial a_{n-1}}p_t(a_{n-1},x_1) & \dots & \frac{\partial}{\partial a_{n-1}}p_t(a_{n-1},x_n) \end{vmatrix}$$

Para que todos $a_i \to a$ precisamos ainda fazer $a_{n-1} \to a_{n-2}$, $a_{n-2} \to a_{n-3}$ e assim por diante. Se fizermos assim, terminaremos com

$$det\left[\frac{\partial^{i-1}}{\partial a^{i-1}}p_t(a_i,x_j)\right]_{i,j=1}^{n}$$

Onde

$$\frac{\partial^{i-1}}{\partial a^{i-1}}p_t(a_i,x_j) = \left[\left(\frac{x_j-a_i}{t}\right)^{i-1} + O\left(\frac{x_j-a_i}{t}\right)^{i-2}\right]p_t(a_i,x_j)$$

Que nos permite afirmar que o determinante é proporcional à

$$\det\left[x_j^{i-1}\right]_{i,j=1}^{n}\prod_{j=1}^{n}e^{-\frac{x_j^2-2ax_j}{2t}}$$

quando $a_i \to 0$. Analogamente para o outro determinante

$$\det\left[x_j^{i-1}\right]_{i,j=1}^{n}\prod_{j=1}^{n}e^{-\frac{x_j^2-2bx_j}{2(T-t)}}$$

Ou seja;

$$f_j = F_{j-1}(x)e^{-\frac{(x-a)^2}{2t}} \;;\; g_j = G_{j-1}(x)e^{-\frac{(x-b)^2}{2(T-t)}}$$

onde F e G são polinômios em x de grau $j-1$. Tomaremos agora o limite em que $a, b \to 0$. Reescrevemos a distribuição conjunta destes pontos

$$P(x_1,\dots,x_n) = \frac{1}{\mathcal{Z}_n}\det\left[p_t(a_i,x_j)\right]_{i,j=1}^{n}\det\left[p_{T-t}(x_i,b_j)\right]_{i,j=1}^{n} \tag{3.20}$$

$$= \frac{1}{\mathcal{Z}_n}\det\left[x_j^{i-1}\right]_{i,j=1}^{n}\prod_{j=1}^{n}e^{-\frac{x_j^2}{2t}}\det\left[x_j^{i-1}\right]_{i,j=1}^{n}\prod_{j=1}^{n}e^{-\frac{x_j^2}{2(T-t)}} \tag{3.21}$$

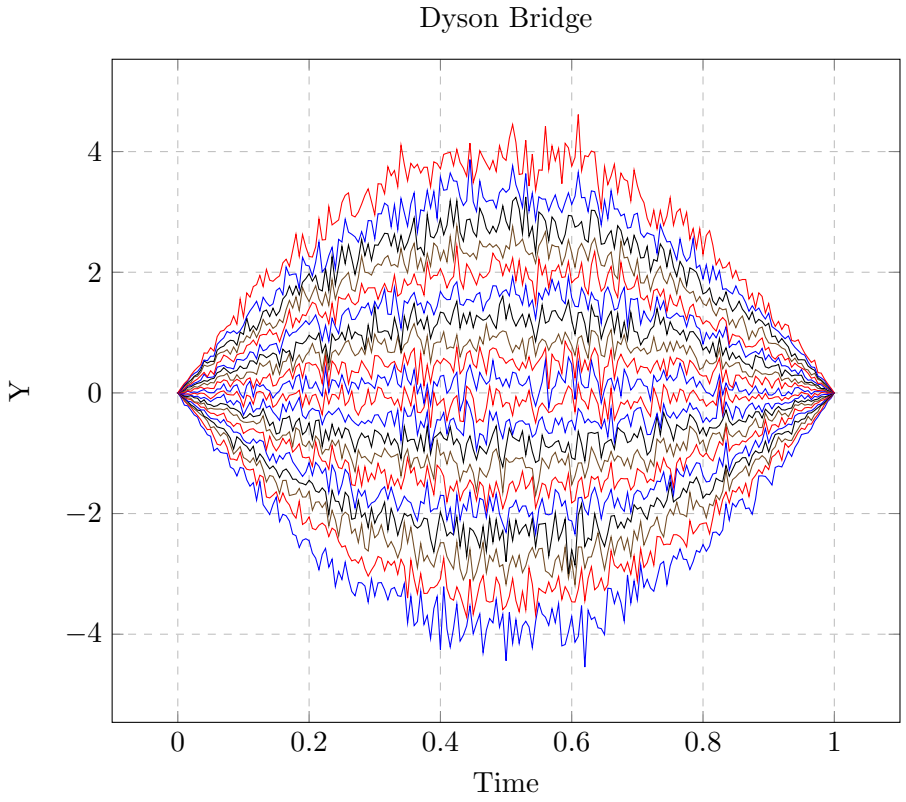
$$= \frac{1}{\mathcal{Z}_n}\det\left[x_j^{i-1}\right]_{i,j=1}^{n}\det\left[x_j^{i-1}\right]_{i,j=1}^{n}\prod_{j=1}^{n}e^{-\frac{Tx_j^2}{2t(T-t)}} \tag{3.22}$$

Onde o último passo é dado pelo valor do determinante de Vandermonde discutido no apêndice D tal que

$$P(x_1, \ldots, x_n) = \frac{1}{\mathcal{Z}_n} \prod_{1 \leq i < j \leq n} (x_j - x_i)^2 \prod_{j=1}^{n} e^{-\frac{Tx_j^2}{2t(T-t)}}$$

Relembrando a equação (2.16), podemos interpretar esta distribuição como uma distribuição de autovalores em um espaço de matrizes hermitianas do GUE com entradas de variância $\sigma^2 = \frac{2t(T-t)}{T}$ e média $\mu = 0$.

De forma que poderemos simular a evolução destes movimentos brownianos com o sistema de matrizes descrito.




```python
import numpy as np

n_particles = 20
steps = 1000

def GUE(N, sigma = 1, mu = 0, escale = False):
    beta = 2
    A = np.random.randn(N,N) * sigma + 1j*np.random.randn(N,N)*sigma
    A = (A + A.T.conj())/2
    eig = np.linalg.eigvalsh(A)
    if escale:
        return (eig + mu)/(np.sqrt(N*beta))
    return eig + mu

eig_GUE = np.zeros(n_particles)

```

```
final_time = 1
dt = final_time/steps
Memory = np.zeros((steps+1,n_particles))

for step in range(steps+1):
    partial_time = step * dt
    sigma = 2*partial_time*(final_time-partial_time)/final_time
    mu = 0
    eig_GUE = GUE(n_particles, sigma = sigma, mu = mu, escale = False)
    eig_order = np.argsort(eig_GUE)
    Memory[step,:] = eig_GUE[eig_order]

np.savetxt('Memory.txt', Memory)
```

3.4 Simulações Gerais

Uma outra pergunta seria: Como simular sistemas com dois pontos finais? Nesse caso pudemos retirar matrizes com entradas independentes porquê a distribuição era relacionada à $e^{\alpha H^2}$, mas agora teremos algo do tipo $e^{\alpha H^2 + H_0}$. Como fazer isso? Podemos tomar uma abordagem aproximada e checar por sua precisão posteriormente.

Queremos tirar autovalores de uma matriz $\tilde{X}$ tal que sua distribuição seja equivalente à dos movimentos brownianos independentes que não se cruzam com dois pontos finais. Note que de alguma forma, podemos tomar como base a matriz aleatória hermetiana de entradas gaussianas já discutida X. Isso nos dá a situação anterior. Mas ainda mais queremos manter o começo parecido mas fazer que, no final, os autovalores estejam separados em dois núcleos. Como os autovalores de uma matriz $X_o^2 = UVU^*$ tal que

$$V = \begin{vmatrix} \lambda_1 & 0 & \cdots & \cdots & 0 \\ 0 & \lambda_1 & & & \vdots \\ \vdots & & \ddots & & \vdots \\ \vdots & & & \lambda_2 & 0 \\ 0 & \cdots & \cdots & 0 & \lambda_2 \end{vmatrix}$$

Dessa forma se fizermos a composição dos dois terbos com uma função linear em t, teremos:

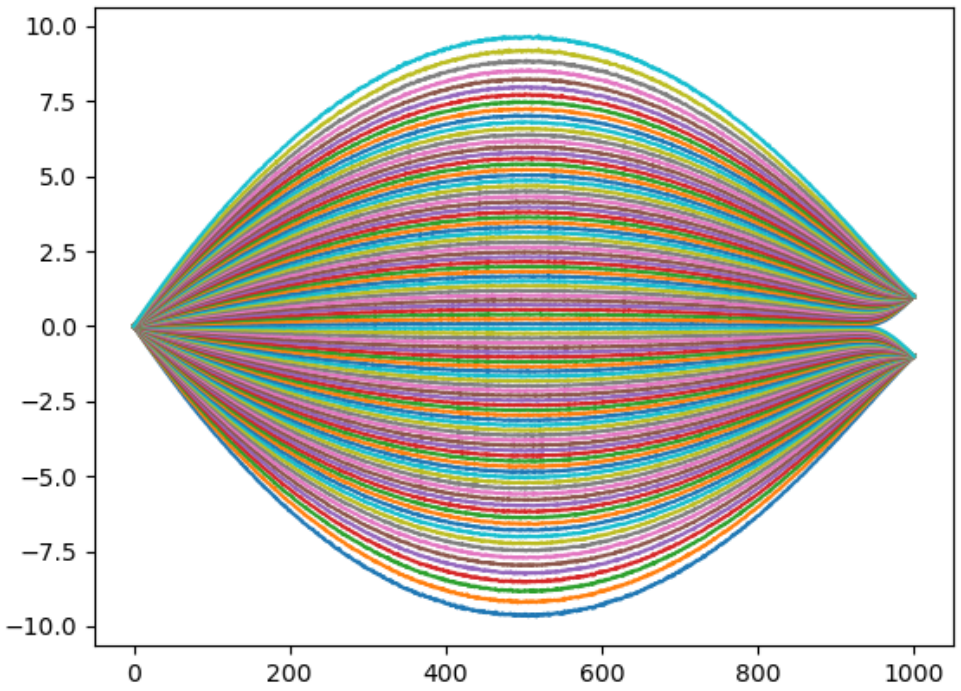
$$X + f(t)UVU^*$$

Que vai nos atender no sentido em que, quando $f(t) = 0$, em $t = 0$ o comportamento é dominado pelo primeiro termo. Enquanto isso, quando a variância volta a diminuir nas entradas de X, o segundo termo domina, e, no final, dita as posições das partículas. Podemos testar nosso método. Implementemos:

```
import numpy as np

n_particles = 100
n_groups = 2
steps = 100

```

```
def Mod_GUE(U, S, shift, N, sigma = 1, mu = 0, escale = False):
    beta = 2
    A = sigma * np.random.randn(N,N,) + sigma *1j*np.random.randn(N,N)
    A = (A + A.T.conj())/2
    A = A + (U @ S @ U.T.conj())*shift
    eig = np.linalg.eigvalsh(A)
    if escale:
        return (eig*sigma + mu)/(np.sqrt(N*beta))
    return eig

def USU(nGroups, n_particles):
    # U is the unitary matrix with the e1, e2, ..., eN vectors as columns
    # S is the diagonal matrix with the eigenvalues

    U = np.zeros((n_particles, n_particles), dtype = complex)
    S = np.zeros((n_particles, n_particles), dtype = complex)

    # fill U diagonal with 1's
    for i in range(n_particles):
        U[i,i] = 1

    # fill S diagonal with nGroups different values centered at 0
    divider = n_particles/nGroups
    shift = (nGroups-1)/2
    for i in range(n_particles):
        S[i,i] = (int(i/divider) - shift) * 10

    return U, S

def sigma_t(t):
    return 2*t*(1-t)

U, S = USU(n_groups, n_particles)
eig_GUE = np.zeros(n_particles)
dt = 1/steps

Memory = np.zeros((steps+1,n_particles))

for step in range(steps+1):
    print(step)
    partial_time = step * dt
    sigma = sigma_t(partial_time)
    shift = partial_time
    mu = 0
    eig_GUE = Mod_GUE(U, S, shift, n_particles, sigma = sigma, mu = mu, escale = False)
    eig_order = np.argsort(eig_GUE)
    Memory[step,:] = eig_GUE[eig_order]

# write Memory in file
np.savetxt('Memory.txt', Memory)
```

E teremos algo do tipo:



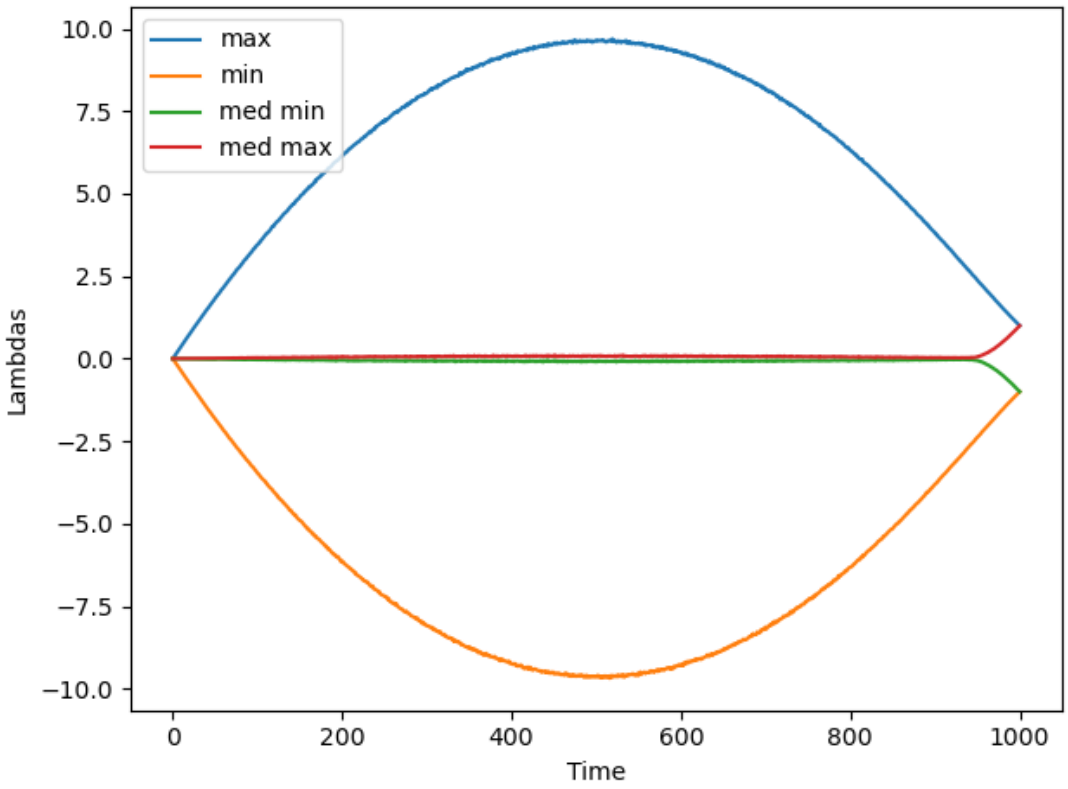
3.4.1 Tracy Widom

Uma das formas de validar nossa simulação é aproveitando de um resultado conhecido para esta distribuição. Tomamos $\gamma = \gamma(t) = \mathbb{E}[\lambda_{max}]$. Se definirmos a distribuição na borda para uma matriz de tamanho N quando $N \to \infty$, veremos que

$$\frac{\lambda_m ax - \mathbb{E}[\lambda_{max}]}{N^{\alpha}}$$

Deve ser a densidade da distribuição quando $\alpha = -\frac{2}{3}$ para toda a extensão externa da distribuição. Essa distribuição se nomeia Tracy Widom e tem formato bem conhecido. Note que se o expoente não for compatível com a distribuição observada, ele deve, no limite fazer com que ela se torne 'singular' em algum sentido, esmague ela no eixo em x ou y. Logo, idealmente, usaríamos matrizes extremamente grandes. O problema de autovalores, contudo, é complexo e leva tempo para tais matrizes. Faremos um compromisso e tentaremos observar a diferenciação da distribuição real.

Primeiro, a média. Faremos isso numericamente. Queremos algo do tipo,

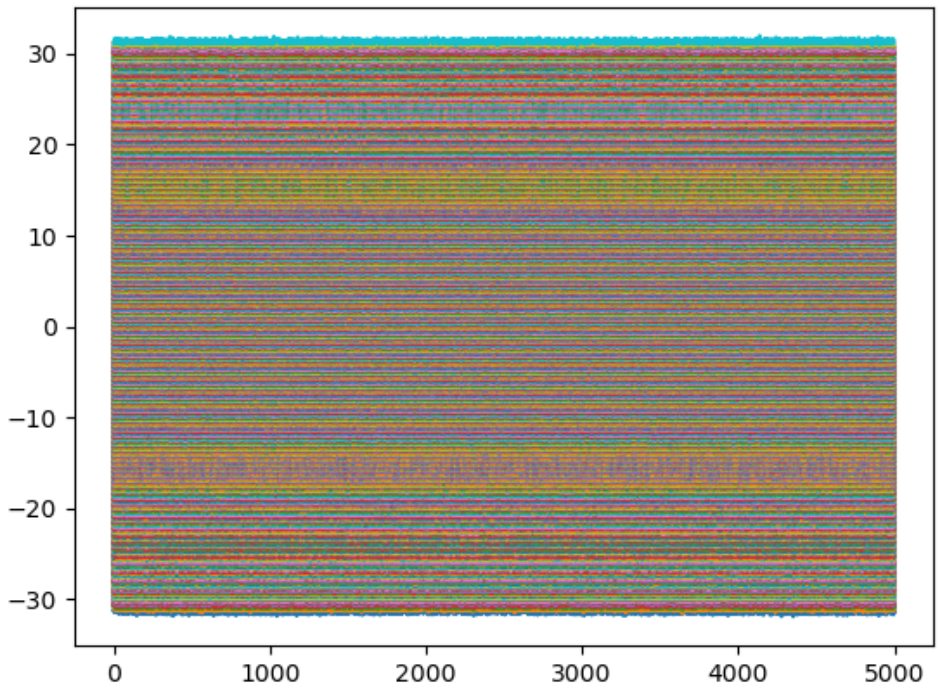


Para simplicar o gasto temporal calculemos para um passo específico múltiplos experimentos, uma vez que sabemos calcular a matriz que dá sua distribuição.

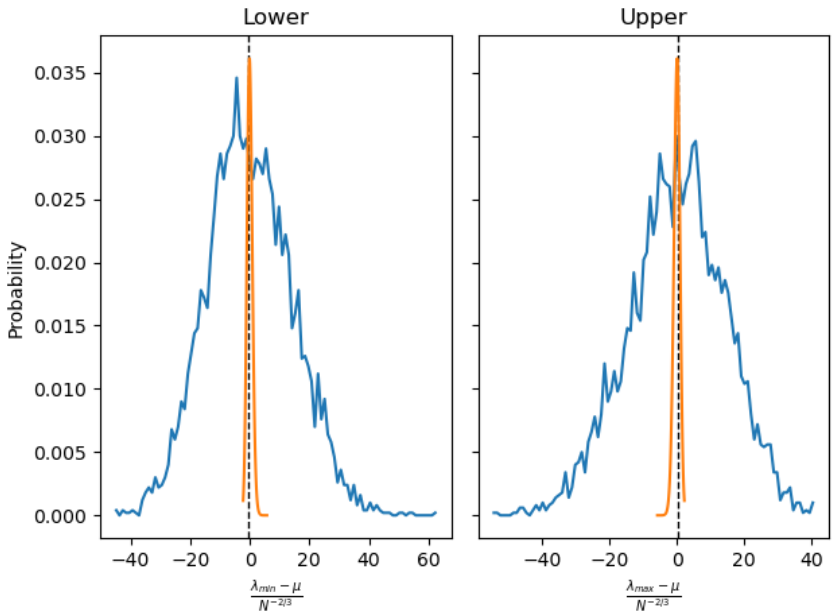
```
import numpy as np

n_particles = 1000
n_groups = 2

def Mod_GUE(S, shift, N, sigma = 1, mu = 0, escale = False):
    beta = 2
    A = sigma * np.random.randn(N,N,) + sigma *1j*np.random.randn(N,N)
    A = (A + A.T.conj())/2
    A = A + S*shift
    eig = np.linalg.eigvalsh(A)
    if escale:
        return (eig*sigma + mu)/(np.sqrt(N*beta))
    return eig

def USU(nGroups, n_particles):
    # U is the unitary matrix with the e1, e2, ..., eN vectors as columns
    # S is the diagonal matrix with the eigenvalues

    U = np.zeros((n_particles, n_particles), dtype = complex)
    S = np.zeros((n_particles, n_particles), dtype = complex)

    # fill U diagonal with 1's
    for i in range(n_particles):
        U[i,i] = 1
```

```
26
27      # fill S diagonal with nGroups different values centered at 0
28      divider = n_particles/nGroups
29      shift = (nGroups-1)/2
30      for i in range(n_particles):
31          S[i,i] = (int(i/divider) - shift) * 10
32
33      return U, S
34
35  def isHermitian(A):
36      return np.allclose(A, A.T.conj())
37
38  def sigma_t(t):
39      return 2*t*(1-t)
40
41  U, S = USU(n_groups, n_particles)
42  eig_GUE = np.zeros(n_particles)
43
44  exps = 1000
45
46  Memory = np.zeros((exps, n_particles))
47
48  for exp in range(exps):
49      partial_time = 0.5
50      sigma = sigma_t(partial_time)
51      shift = partial_time
52      mu = 0
53      eig_GUE = Mod_GUE(S, shift, n_particles, sigma = sigma, mu = mu, escale = False)
54      eig_order = np.argsort(eig_GUE)
55      Memory[exp,:] = eig_GUE[eig_order]
56
57      print(exp)
58
59  # write Memory in file
60  np.savetxt('Memory_test.txt', Memory)
```

Com $N = 1000$, para o momento $t = 0.5$ calculamos múltiplas iterações

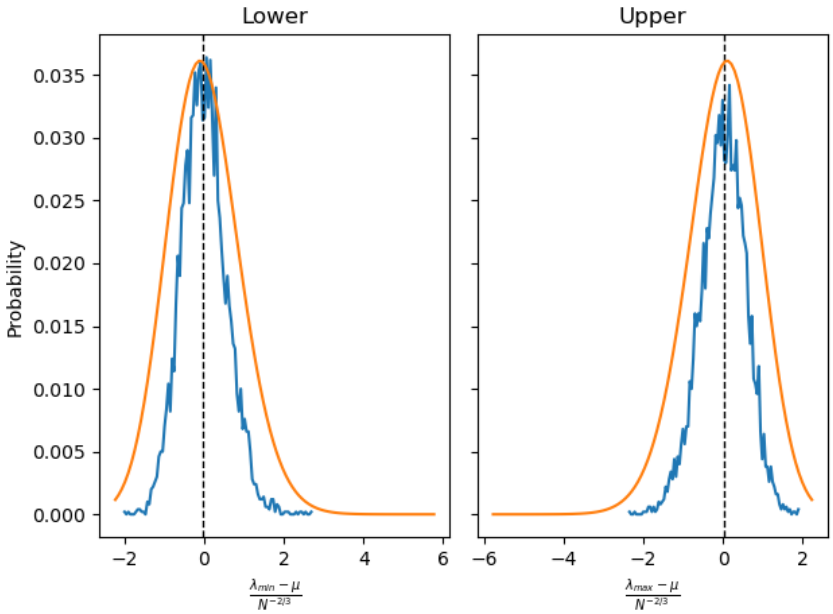


Agora podemos plotar o histograma para os extremos. Especificamente para o extremo superior:

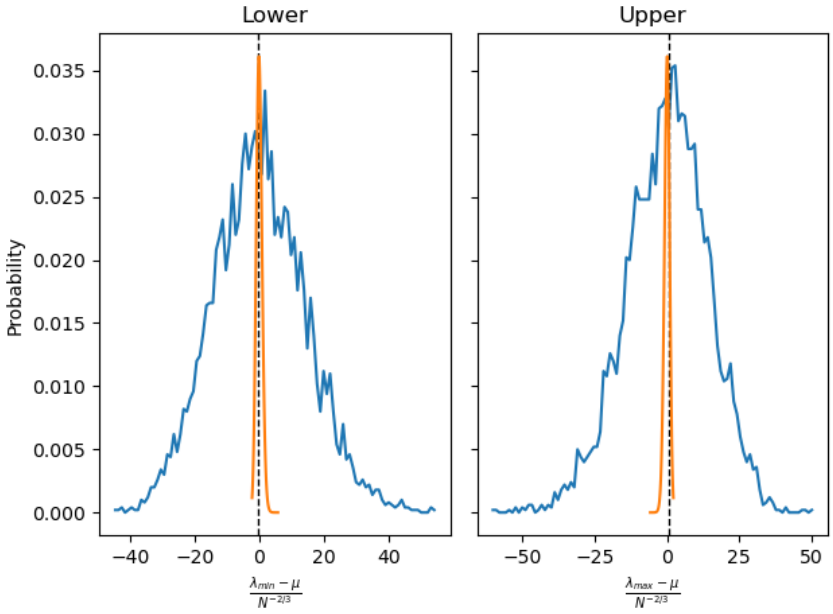


Algo está estranho. Parece que o histograma está mais espalhado do que o esperado. O que pode estar ocorrendo? Vemos nos dados que a variação está de acordo com o histograma. Os dados variam em média algo na ordem de ± 1. Escalado por $N^{\frac{2}{3}}$, dá o resultado observado. De alguma forma os dados parecem com o 'formato' correto, até sua assimetria pode ser observada. E os dados parecem simetrizados para o autovalor máximo e mínimo. Uma suspeita é que alguma escala que dependa de σ deve ser aplicada, isso se

dá principalmente pela pergunta: Isso se repete em outros momentos da simulação? Que tem ilustrada resposta:



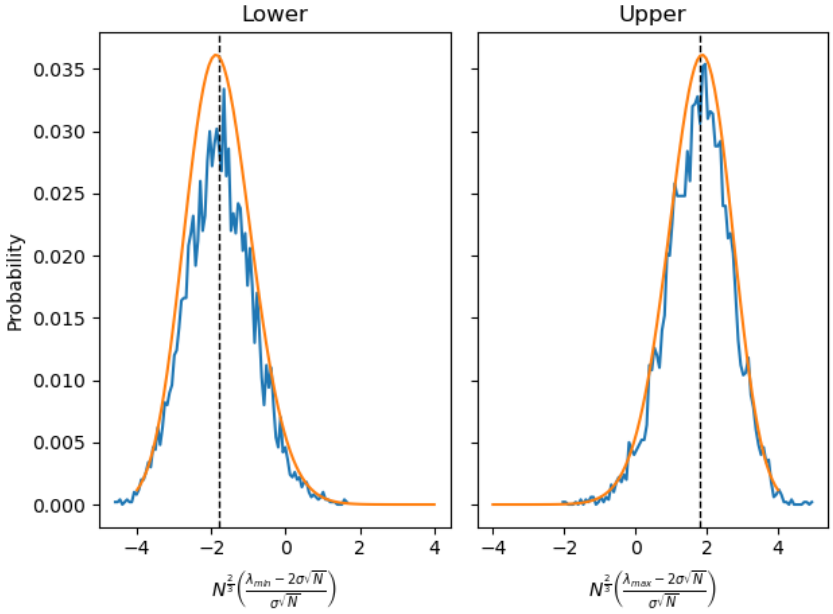
Isso se repetiria para o caso de borda de um único final? Se calcularmos como usualmente as Pontes de Dyson para um único ponto final, veremos a distribuição de Tracy-Widom? Isso indicaria um erro, não na escala, mas no nosso modelo (Principalmente porque os modelos de aproximam quando $t \to 0$). Para este, teríamos em $t = 0.5$:



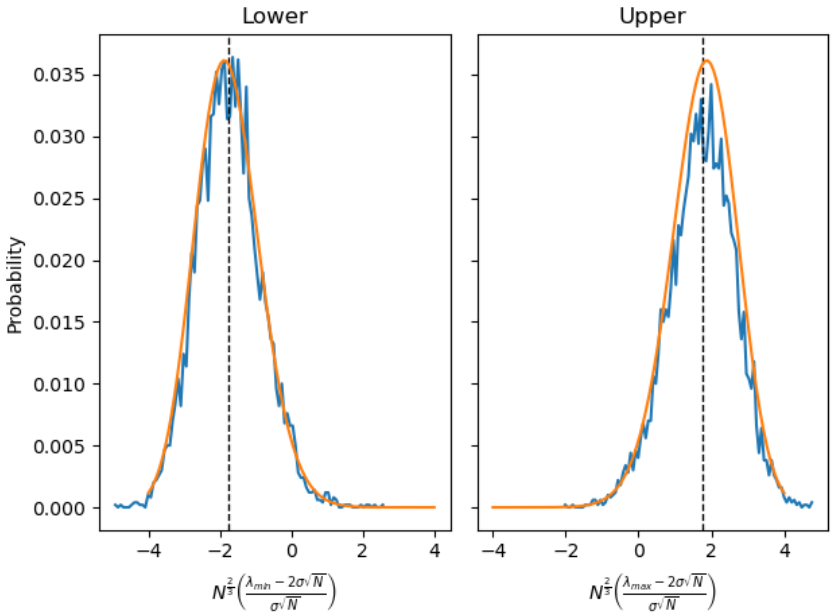
Que parece precisar da mesma escala que o caso anterior. De qual escala estamos falando? Note a expressão

$$N^{\frac{2}{3}}\left(\frac{\lambda_{max} - 2\sigma\sqrt{N}}{\sigma\sqrt{N}}\right)$$

Algo assim parece mais razoável. O λ médio depende de σ e N. Enquanto $\sigma\sqrt{N}$ escala com dependência das mesmas variáveis, que parecem ser importantes. Com essa nova distribuição podemos refazer a distribuição para $t = 0.5$,



e para $t = 0.01$,

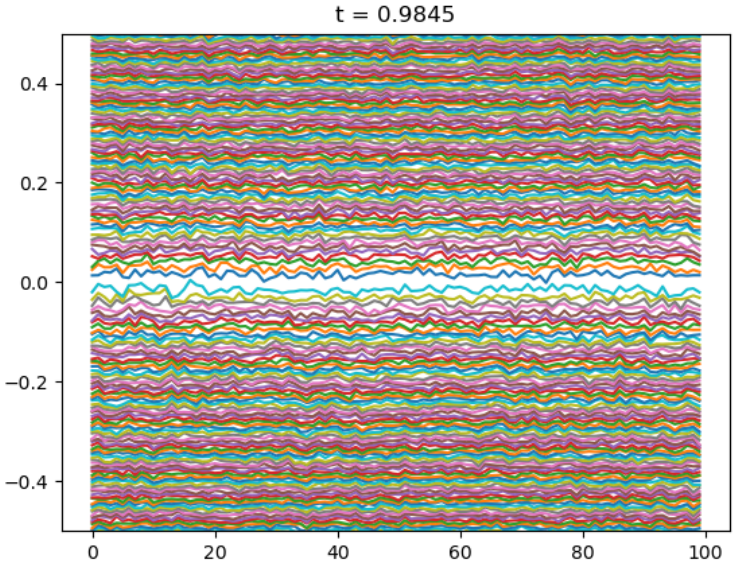


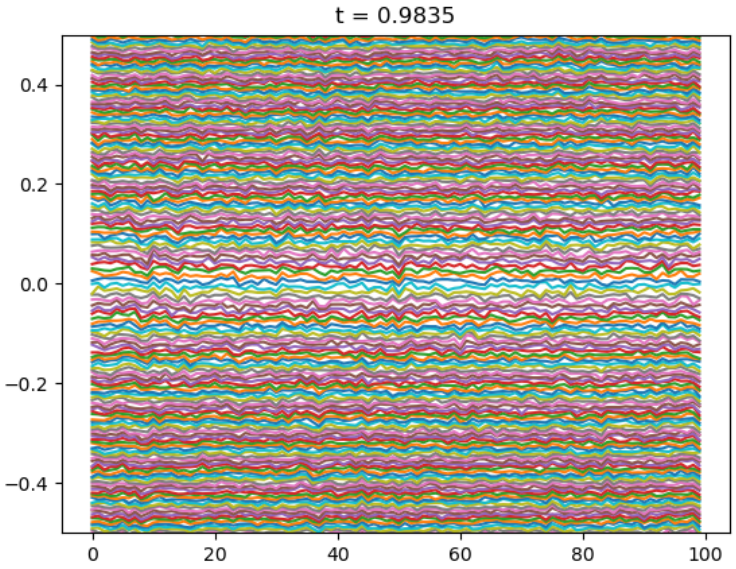
A média ainda poderia ser dada por experimento desde que se centralize a distribuição também.

De fato isso também vale para todo o contorno com exceção do ponto de bifurcação interno, onde gostaríamos de ver $\alpha = -\frac{3}{4}$. Ou seja,

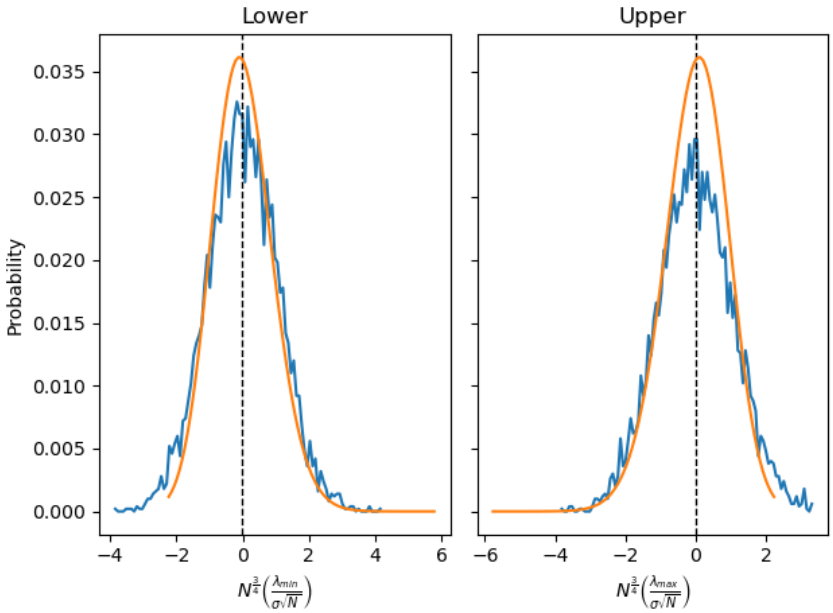
$$N^{\frac{3}{4}}\left(\frac{\lambda_{max} - 2\sigma\sqrt{N}}{\sigma\sqrt{N}}\right)$$

Para nosso caso de uma matriz $1000 \cdot 1000$ procuremos t apropriado para a distribuição dos autovalores médios. Note os autovalores para múltiplos experimentos:

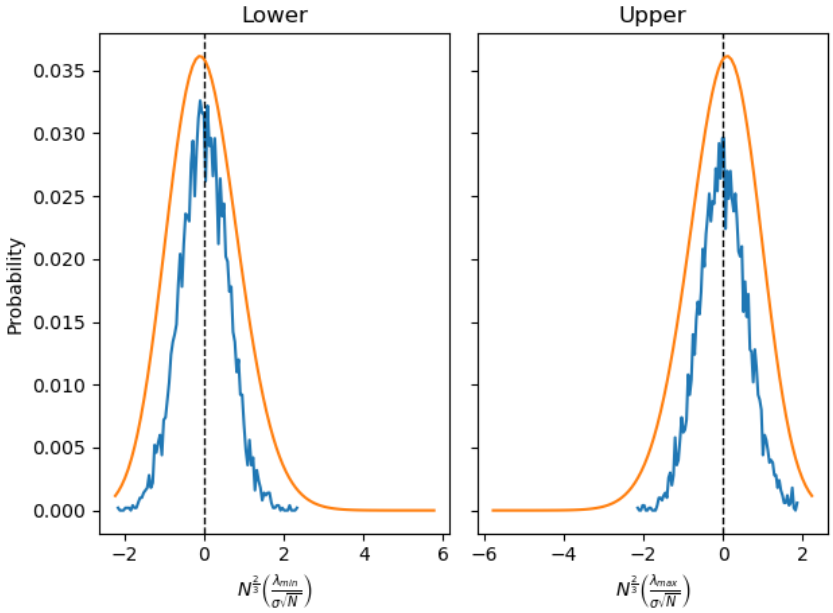




Para $t = 0.9840$ então, faremos para os autovalores médios:



E de fato, se utilizarmos $\alpha = \frac{2}{3}$,

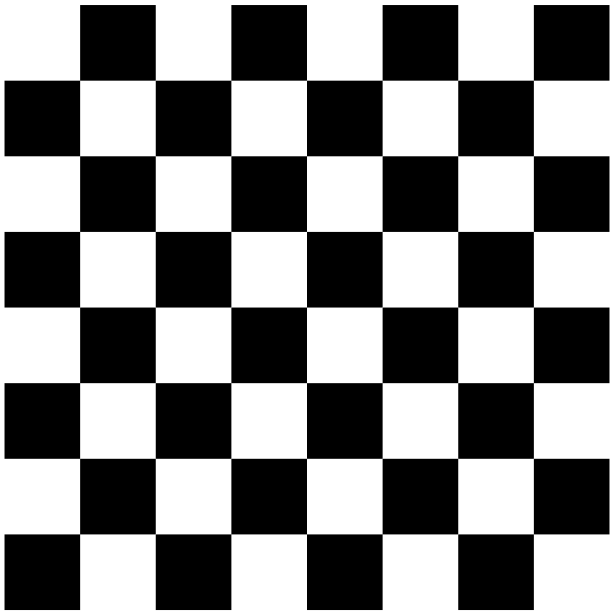


3.4.2 O diamante Asteca

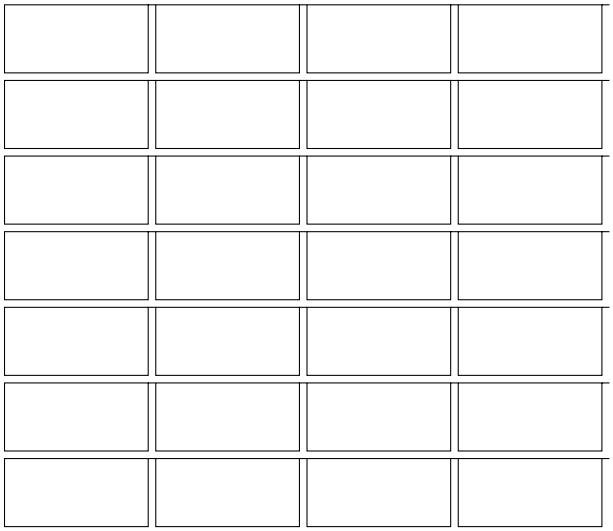
O tabuleiro de xadrez

Outros modelos estocásticos mantém de alguma forma a universalidade que aparece na distribuição de autovalores. Antes, uma introdução.

Se quisermos fazer o tiling de um tabuleiro de xadrez com dominós de $2x1$, sabemos que isso é possível, inclusive por tilings triviais.



Com tilings que podemos fazer trivialmente:

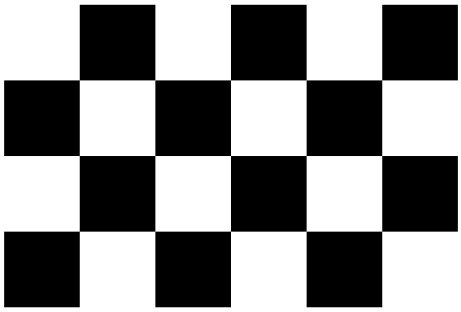


Um interessante detalhe é demonstrar se é possível fazer o mesmo retirando uma peça do tabuleiro. A demonstração é razoavelmente trivial e nos mostra que é impossível fazer tal coisa. Basta notar que todas peças do dominó devem ocupar uma peça branca e uma peça preta. De tal forma que, retirando uma peça, tal cobrimento não pode ser possível pois restará um número ímpar de alguma classe.

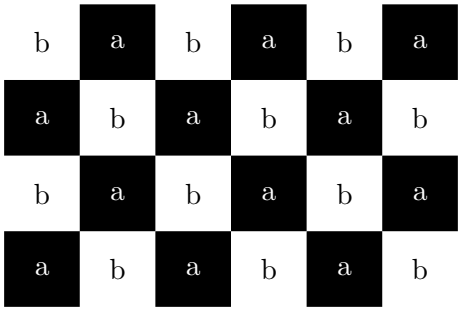
É sempre possível fazer o tiling de um tabuleiro onde se retire duas peças arbitrárias? Claro, se elas forem de mesma cor, a mesma demonstração anterior vale e não será possível fazer o cobrimento. Mas e se forem cores complementares? A demonstração nos diz que é possível retirando pares de cores. Como? É possível mostrar que é pode-se definir um contorno que sigam as células do tabuleiro. Ou seja, um caminho em loop no tabuleiro de peças alternadas que pode ser preenchido por dominós em sequência. Retiradas duas peças do caminho, temos dois casos, um degenerado quando se retira peças sequenciais e é trivial que pode-se preencher o resto (equivalente a retirar um dominó do caminho completo) ou separa-se e duas fitas independentes. Neste caso, basta notar que ambas fitas devem ter um quantidade par de casas alternadas e seguirá trivialmente que é possível as preencher.

De quantas formas preenchemos?

Imagine uma forma qualquer sem buracos.



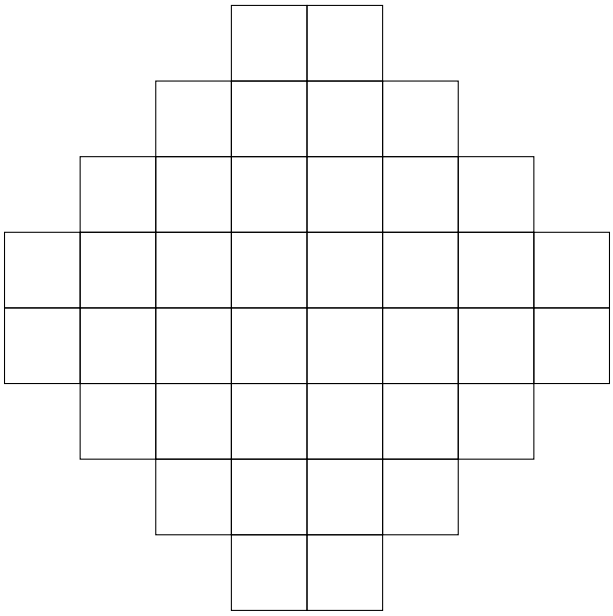
Para saber de quantas formas podemos fazer o preenchimento da forma, precisamos numerar as peças pretas e brancas em uma ordem arbitrária.



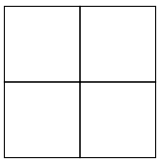
Se numerarmos eles e criarmos uma matriz "de adjacência"onde se preenche com 1 onde dois números são vizinhos horizontais no tabuleiro e i quando são vizinhos verticais. Preenchemos com 0 quando não o são. Teremos nosso resultado calculado quando fizermos o determinante de tal matriz. Isto é, o determinante desta matriz de vizinhança das casas ordenadas dentro das classes nos dá de quantas formas (em módulo e real) podemos preencher o plano com esses dominós.

O diamante

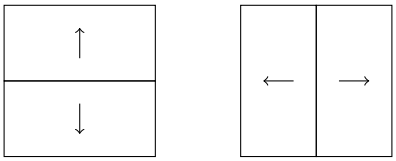
O algoritmo que seguiremos é descrito em [6] .Suponha agora o tabuleiro especial:



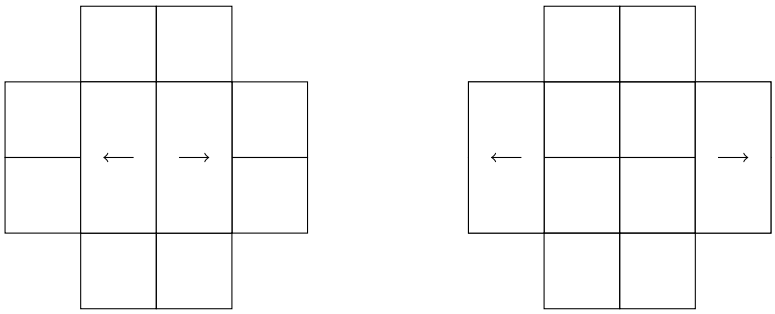
Conseguimos o preencher com os dominós? Podemos seguir o seguinte algoritmo. Suponho o caso mais básico do tabuleiro:



Este podemos fazer o tiling de duas formas triviais,



Realizaremos o tiling de uma grade expandida utilizando de uma técnica curiosa. Note as setas nos dois dominós que demonstramos. Elas vão indicar a direção de deslocamento que as peças devem tomar quando expandirmos o plano. Ou seja:



Note que ao final de cada procedimento sempre restará quadrados de $2x2$ completos que podem ser preenchidos independentemente com as duas opções que demos. Às vezes células terão celular que colidem. Devemos eliminar estas e é possível mostrar qus estes vem em pares e são equivalentes quando as peças que colidem são eliminadas. Podemos preencher ambos de volta os fazendo diferentes e vale algo curioso. Para cada malha de n-tamanho terão (mesmo que com o artifício descrito) n novos quadrados a se preencher de 2 maneiras. Ou seja, o tiling final pode ser preenchido de $2^{1+2+3+\cdots+n}$ formas. Escrevendo o modelo em código:

```
1       PROGRAM ASTECDIAMOND
2
3             DIMENSION IAD(-200:200,-200:200), IVAD(-400:400,-400:400)
4             INTEGER dim
5             COMMON /AD/ IAD, IVAD, dim
6
7             OPEN(UNIT=1, FILE='diamond.txt', STATUS='UNKNOWN')
8             OPEN(UNIT=2, FILE='diamond2.txt', STATUS='UNKNOWN')
9
10            dim = 0
11
12            CAll InitDiamond()
13
14            DO I = 1, 150
15                  CALL UpdateDiamond()
```

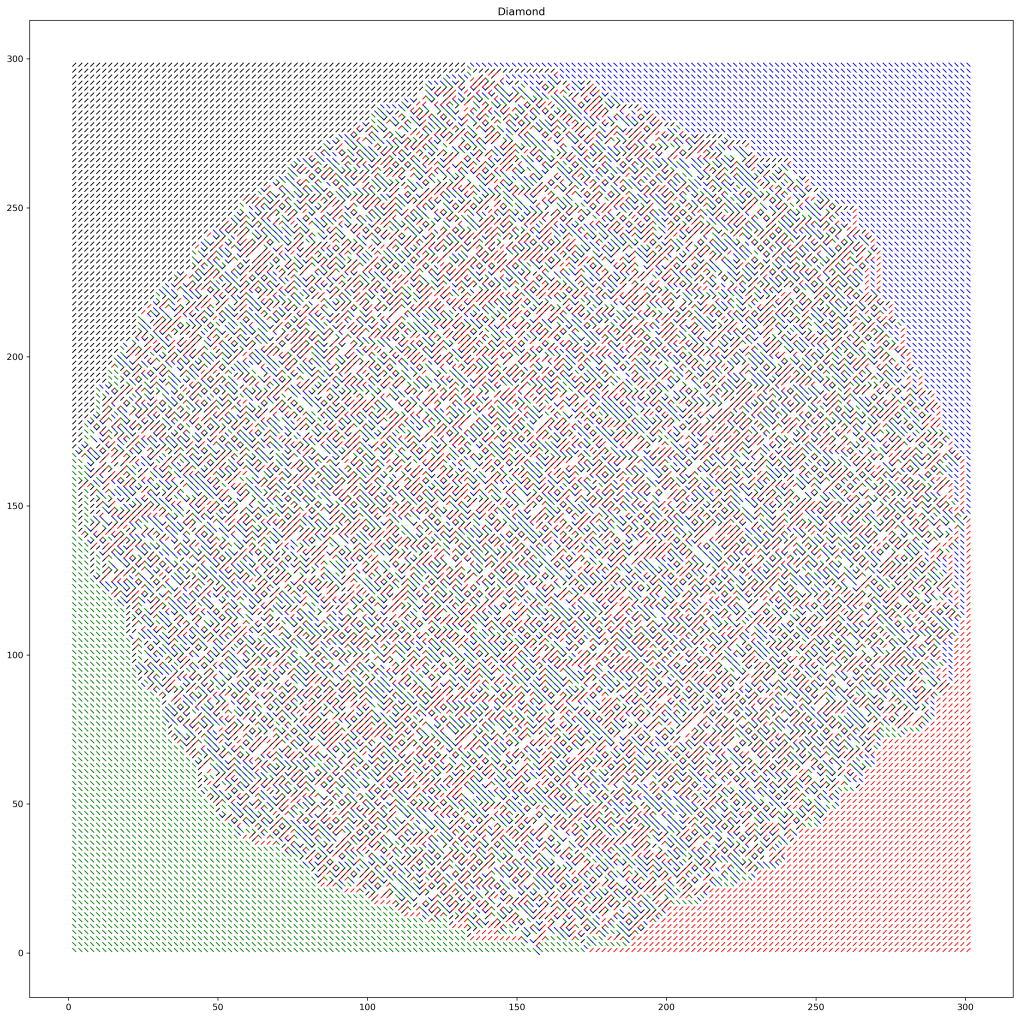
```
16              END DO
17
18  c           in active cells
19              DO I = -dim+1, dim, 2
20              DO J = -dim+1, dim, 2
21
22                    IF (IAD(I,J) .EQ. 0) THEN
23                          CALL InitSquare(I, J)
24                    END IF
25
26              END DO
27              END DO
28
29              DO I = -dim+1, dim+1
30                    WRITE(1,*) IVAD(I,-dim-1:dim+1)
31              END DO
32
33        END PROGRAM ASTECDIAMOND
34
35        SUBROUTINE InitDiamond()
36              DIMENSION IAD(-200:200,-200:200), IVAD(-400:400,-400:400)
37              COMMON /Ad/ IAD, IVAD, dim
38
39  c           init all cells with 0
40              DO I = -100, 100
41                    DO J = -100, 100
42                          IAD(I,J) = 0
43                    END DO
44              END DO
45
46              DO I = -200, 200
47                    DO J = -200, 200
48                          IVAD(I,J) = 0
49                    END DO
50              END DO
51
52
53        END SUBROUTINE InitDiamond
54
55        SUBROUTINE UpdateDiamond()
56              DIMENSION IAD(-200:200,-200:200),IaVAD(-200:200,-200:200),
57       +      IVAD(-400:400,-400:400)
58              INTEGER dim
59              COMMON /AD/ IAD, IVAD, dim
60
61  c           in active cells
62              DO I = -dim+1, dim, 2
63              DO J = -dim+1, dim, 2
64
65                    IF (IAD(I,J) .EQ. 0) THEN
66                          CALL InitSquare(I, J)
67                    END IF
```

```
68
69              END DO
70              END DO
71
72  c           Update grid size
73              dim = dim + 1
74
75  c           in active cells
76              DO I = -dim+1, dim, 2
77              DO J = -dim+1, dim, 2
78
79                    IF (IAD(I,J) .GT. 1) THEN
80                          CALL EraseSquare(I, J)
81                    END IF
82
83              END DO
84              END DO
85
86  c           equalize aux grid to zero
87              DO I = -dim-10, dim+10
88              DO J = -dim-10, dim+10
89                    IaVAD(I,J) = 0
90              END DO
91              END DO
92
93  c           Move vertices
94              DO I = -dim+1, dim
95              DO J = -dim+1, dim
96
97                    IF (IVAD(I,J) .EQ. 1) THEN
98
99                          IaVAD(I+1,J+1) = 1
100
101                         IAD(I+1,J+1) = IAD(I+1,J+1) + 1
102                         IAD(I-1,J-1) = IAD(I-1,J-1) - 1
103
104                   ELSE IF (IVAD(I,J) .EQ. 2) THEN
105
106                         IaVAD(I-1,J+1) = 2
107
108                         IAD(I-2,J+1) = IAD(I-2,J+1) + 1
109                         IAD(I,J-1) = IAD(I,J-1) - 1
110
111                   ELSE IF (IVAD(I,J) .EQ. -1) THEN
112
113                         IaVAD(I-1,J-1) = -1
114
115                         IAD(I-2,J-2) = IAD(I-2,J-2) + 1
116                         IAD(I,J) = IAD(I,J) - 1
117
118                   ELSE IF (IVAD(I,J) .EQ. -2) THEN
119
120                         IaVAD(I+1,J-1) = -2
```

```
121
122                         IAD(I+1,J-2) = IAD(I+1,J-2) + 1
123                         IAD(I-1,J) = IAD(I-1,J) - 1
124
125                   END IF
126             END DO
127             END DO
128
129   c         copy aux grid to main grid
130             DO I = -dim-10, dim+10
131             DO J = -dim-10, dim+10
132                   IVAD(I,J) = IaVAD(I,J)
133             END DO
134             END DO
135
136       END SUBROUTINE UpdateDiamond
137
138       SUBROUTINE InitSquare(i, j)
139             DIMENSION IAD(-200:200,-200:200), IVAD(-400:400,-400:400)
140             COMMON /AD/ IAD, IVAD, dim
141
142
143             toss = RAND(0)
144
145             IF (toss .LT. 0.5) THEN
146                   IVAD(i,j) = -1
147                   IVAD(i+1,j+1) = 1
148
149                   IAD(i-1,j-1) = IAD(i-1,j-1) + 1
150                   IAD(i+1,j+1) = IAD(i+1,j+1) + 1
151             ELSE
152                   IVAD(i+1,j) = -2
153                   IVAD(i,j+1) = 2
154
155                   IAD(i-1,j+1) = IAD(i-1,j+1) + 1
156                   IAD(i+1,j-1) = IAD(i+1,j-1) + 1
157             END IF
158
159             IAD(i,j) = IAD(i,j) + 2
160
161       END SUBROUTINE InitSquare
162
163       SUBROUTINE EraseSquare(i, j)
164             DIMENSION IAD(-200:200,-200:200), IVAD(-400:400,-400:400)
165             COMMON /AD/ IAD, IVAD, dim
166
167             DO k = i, i+1
168             DO l = j, j+1
169                   IF (IVAD(k,l) .NE. 0) THEN
170
171                   IF ((IVAD(k,l) .EQ. 1) .OR. (IVAD(k,l) .EQ. -1)) THEN
172                         IAD(k-1,l-1) = IAD(k-1,l-1) - 1
173                         IAD(k,l) = IAD(k,l) - 1
```

```
174                 ELSE IF ((IVAD(k,l) .EQ. 2).OR.(IVAD(k,l).EQ.-2)) THEN
175                       IAD(k-1,l) = IAD(k-1,l) - 1
176                       IAD(k,l-1) = IAD(k,l-1) - 1
177                 END IF
178
179                 IVAD(k,l) = 0
180
181                 END IF
182           END DO
183           END DO
184
185     END SUBROUTINE EraseSquare
```

E temos de saída:



Capítulo 4

Coulombolas! Gases Aleatórios

Retomamos agora os resultados sobre a distribuição dos autovalores para matrizes hermitianas para introduzir o tópico principal deste resumo: Os Gases de Coulomb.

Um dos primeiros resultados foi que para o ensemble do GUE, teríamos que

$$p(M) = \frac{1}{\tilde{\mathcal{Z}}_N} e^{-\operatorname{Tr} \tilde{V}(M)} dM$$

Mas estamos mais interessados na distribuição relativa aos autovalores, que podemos expressar escrevendo o jacobiano da transformação,

$$p(\lambda_1, \lambda_2, \ldots, \lambda_N) = \frac{1}{\mathcal{Z}_N} \prod_{i<j} (\lambda_i - \lambda_j)^\beta \prod_{i=1}^{N} e^{-V(\lambda_i)} d\lambda$$

Onde

$$\mathcal{Z}_N = \int_{R^N} \prod_{i<j} (\lambda_i - \lambda_j)^\beta \prod_{i=1}^{N} e^{-V(\lambda_i)} d\lambda$$

Lembrando que devemos ter medida uniforme nos autovetores. Por isso podemos expressar apenas nos autovalores. Assim como na física, para resolver nosso Gás de Coulomb, vamos precisar minimizar a energia livre do nosso ensemble. Lembre-se que se trata do ensemble canônico e que devemos ter a energia livre de Helmholtz

$$F = -\frac{1}{k_b} \ln (\mathcal{Z}_N)$$

Teremos que os estados mais prováveis serão aqueles em que for maximizada a expressão

$$\begin{aligned}
&\prod_{i<j} (\lambda_i - \lambda_j)^2 \prod_{i=1}^{N} e^{V(\lambda_i)} \\
&= \exp\left[-N^2 \left(\frac{1}{N^2} \sum_{i \neq j} \log \frac{1}{|\lambda_i - \lambda_j|} + \frac{1}{N^2} \sum_{i=1}^{N} V(\lambda_i) \right) \right] \\
&= \exp\left(-N^2 \tilde{\mathcal{H}}_N(\lambda)\right)
\end{aligned}$$

Que, como soma assintótica, importaremos apenas com o maior termo da soma de logs. E novamente como na física, vamos então precisar minimizar o Hamiltoniano do sistema $\tilde{\mathcal{H}}_N(\lambda)$!

4.1 Hamiltonianozinho

Vamos introduzi uma função contagem para facilitar o tratamento do conjunto de pontos em $\mathbb{R}$. Definiremos

$$\upsilon_\lambda = \frac{1}{N}\sum_1^N \delta_{\lambda_i} \tag{4.1}$$

Notamos a propriedade de nossa função

$$\int f(x)d\upsilon_\lambda(x) = \frac{1}{N}\sum f(x) \tag{4.2}$$

Com isso, se quisermos escrever $\mathcal{H}_N(\upsilon_\lambda)$, precisamos notar que

$$\int V(x)d\upsilon_\lambda(x) = \frac{1}{N}\sum V(x)$$

e

$$\int\int_{x\neq y} \log\frac{1}{|\lambda_i-\lambda_j|}d\upsilon_\lambda(x)d\upsilon_\lambda(y) = \frac{1}{N^2}\sum_{i\neq j}\log\frac{1}{|\lambda_i-\lambda_j|}$$

Unindo esses resultados,

$$\mathcal{H}_N(\upsilon_\lambda) = \int\int_{x\neq y}\log\frac{1}{|\lambda_i-\lambda_j|}d\upsilon_\lambda(x)d\upsilon_\lambda(y) + \frac{1}{N}\int \tilde{V}(x)d\upsilon_\lambda(x) \tag{4.3}$$

O que acontece quando tratamos do limite termodinâmico? Ou seja, quando $N\to\infty$. Estaremos transicionando da nossa função υ_λ para uma densidade $\phi(x)dx$, ou seja,

$$\int f(x)d\upsilon_\lambda(x) = \int f(x)\phi(x)dx$$

Note que, para justificar isso, escrevemos

$$\int f(x)d\upsilon_\lambda(x) = \frac{1}{N}\sum_{i=1}^N f(\lambda_j)$$

Mas é claro, nosso lambdas são desigualmente espaçados, definimos uma função (ϕ) que mapeie pontos igualmente distribuídos nos nossos autovalores de forma que

$$\frac{1}{N}\sum_{i=1}^N f(\lambda_j) = \frac{1}{N}\sum_{i=1}^N f(\Phi(x_j))$$

Assim podemos retomar e escrever

$$\int f((\Phi(s))ds = \int f(x)\frac{1}{\Phi'(\Phi^{-1}(x))} = \int f(x)(\Phi^{-1})'(x)dx = \int f(x)\phi(x)dx = \int f(x)d\psi(x)$$

Onde $\Phi(s)=x$, $ds=\frac{dx}{\Phi'(s)}$ e $\psi=\Phi^{-1}$. Em suma, podemos afirmar que nossa função converge para a densidade de forma

$$\int f(x)d\upsilon_\lambda(x) = \int f(x)\phi(x)dx$$

Finalmente podemos pensar em minimizar nossa energia livre, ou equivalentemente minimizar o hamiltoniano. Note que pela equação 4.3, nosso potencial externo é limitado

e deve ir à zero com o limite aplicado. Teremos uma situação sem equilíbrio!! Precisamos garantir um potencial $V(x) = N\tilde{V}(x)$. Nossa nova distribuição será

$$\frac{1}{\tilde{\mathcal{Z}}_{\mathcal{N}}} e^{-N \operatorname{Tr}(V(M))} dM$$

e equivalentemente,

$$\frac{1}{\mathcal{Z}_N} \prod_{i<j} (\lambda_i - \lambda_j)^2 \prod_{i=1}^{N} e^{-NV(\lambda_i)} d\lambda = \frac{1}{\mathcal{Z}_N} e^{-N^2 \mathcal{H}_N(\lambda)}$$

E finalmente poderemos expressar de forma correta

$$\mathcal{H}_N(\upsilon_\lambda) \to \int \int_{x \neq y} \log \frac{1}{|\lambda_i - \lambda_j|} d\phi(x) d\phi(y) dx dy + \int V(x) d\phi(x) dx \equiv E^V(\phi) \tag{4.4}$$

E $\phi(x)$ será aquela que minimize $E^V(\phi)$ que limita a energia livre discutida.

Capítulo 5

Simulações e o Artigo

Nos referenciaremos aqui aos desenvolvimento do artigo citado em [1]. Vamos compilar algum desenvolvimento teórico necessário e explicitar os resultados e métodos do artigo.

5.1 Introdução Teórica

Vamos esquematizar as notações a serem usadas. O artigo toma um subespaço S de dimensão d em $\mathbb{R}^n$. O subespaço toma a métrica de Lebesgue, denotado dx. O campo externo é denominado $V : S \mapsto \mathbb{R}$ e a interação entre partículas $W : S \mapsto (-\infty, \infty]$. Para qualquer $N \geq 2$ consideramos P_N em $S^N = S \times \cdots \times S$ definida

$$P_N(dx) = \frac{e^{-\beta_N H_N(x_1,\cdots,x_N)}}{Z_N} dx_1 \cdots dx_N$$

Onde $\beta_n > 0$ é uma constante e Z_N é contante de normalização. Por último,

$$H_N(x_1, \cdots, x_N) = \frac{1}{N} \sum_{i=1}^{N} V(x_i) + \frac{1}{2N^2} \sum_{i \neq j} W(x_i - x_j)$$

Note que P_N é invariante por permutação e que H_N depende somente da medida empírica

$$\mu_N = \frac{1}{N} \sum_{i=1}^{N} \delta_{x_i}$$

Note que as partículas vivem em S^N de dimensão dN.

5.1.1 Gases de Coulomb

Eu não entendo Gases de Coulomb. O importante aqui é notar que tomando o subespaço S como um condutor em $\mathbb{R}^n$ e $W = g$, onde g é o kernel de Coulomb ou função de Green em $\mathbb{R}^n$ onde

$$g(x) = \begin{cases} \log \frac{1}{|x|} & sen = 2 \\ \frac{1}{|x|^{n-2}} & sen \geq 2 \end{cases}$$

Em termos de física, interpretamos $H_N(x_1, x_2, \ldots, x_N)$ é energia eletrostática da configuração dos N elétrons em $\mathbb{R}^n$ contidos em S nas posições $x_1, x_2, \ldots, x_N$ em um campo externo de potencial V. g expressa a repulsão de coulomb da interação entre dois corpos. P_N pode ser visto como a medida de Boltzmann-Gibbs, com β_N sendo o inverso da temperatura. P_N é o denominado gás de Coulomb.

5.1.2 Log-Gases

Log-Gases são caracterizados pela escolha de $n = d$ e W tal que

$$W(x) = \log \frac{1}{|x|} = -\frac{1}{2}\log(x_1^2 + \cdots + x_d^2)$$

note que os gases coincidem quando $n = d = 2$.

5.1.3 Medidas de Equilíbrio

É sabido que a medida empírica μ_N tende, quando $N \lim \infty$ para uma medida de probabilidade não aleatória

$$\mu^* = \arg\inf \epsilon$$

sendo o minimizante único da função 'energia' ϵ convexa e semi continua definida por

$$\mu \mapsto \epsilon(\mu) = \int V d\mu + \int\int W(x-y)\mu(dx)\mu(dy)$$

Se $W = g$ for o kernel de Coulomb, $\epsilon(\mu)$ é a energia eletrostática da distribuição de cargas μ.

5.2 Simulando Coulomb-Log Gases

Para alguns modelos de Gases, sabemos que é conveniente usar modelos de matrizes aleatórias quando estas estiverem disponíveis. Processos determinantais também podem ser de ajuda quando $\beta = 2$. Fora esses casos, existem ainda métodos alternativos como o *Overdamped Langevin Difusion Algorithm*, *Metropolis-Hastingss algorithm*, *Metropolis adjusted Langevin algorithm* e versões cinéticas chamadas *Hybrid or Hamiltonian Monte Carlo* baseada em uma versão cinética (*underdamped*) da difusão de Langevin.

Em geral amostrar a medida resulta em dificuldades. A computação de forças de e energias escala com N^2 pelo Hamiltoniano tratar de interações par a par. Outra dificuldade são as singularidades em W, que resultam em instabilidades numéricas.

5.2.1 Os típicos

A ideia explorada é que P_N é medida de probabilidade invariante reversível do processo de difusão de Markov $(X_t)_{t>0}$ solução de

$$dX_t = -\alpha_N \nabla H_N(X_t)dt + \sqrt{2\frac{\alpha_N}{\beta_N}dB_t}$$

Sob alguma condição em β_N e V podemos afirmar que

$$X_t \xrightarrow[t\to\infty]{Law} P_N$$

discretizado, tomaríamos o processo

$$x_{k+1} = x_k - \nabla H_N(x_k)\alpha_N \Delta t + \sqrt{2\frac{\alpha_N}{\beta_N}\Delta t}G_k$$

onde G_k é a família de variáveis gaussianas usuais. Uma forma de contornar o viés embutido é amenizar a dinâmica com a forma

$$x_{k+1} = x_k - \frac{\nabla H_N(x_k)\alpha_N \Delta t}{1 + |\nabla H_N(x_k)|\alpha_N \Delta t} + \sqrt{2\frac{\alpha_N}{\beta_N}\Delta t}G_k$$

Ainda assim, precisamos otimizar o processo. A ideia do algoritmo de Metropolis é adicionar um processo de seleção para evitar passos irrelevantes, algo do tipo:

- defina $\tilde{x}_{k+1}$ de acordo com o kernel $K(x_k,)$ gaussiano;
- defina p_k

$$p_k = 1 \wedge \frac{K(\tilde{x}_{k+1}, x_k)e^{\beta_N H_N(\tilde{x}_{k+1})}}{K(x_k, \tilde{x}_{k+1})e^{\beta_N H_N(\tilde{x}_k)}}$$

- defina

$$x_{k+1} = \begin{cases} \tilde{x}_{k+1} & w/p_k \\ x_k & w/1 - p_k \end{cases}$$

5.2.2 O Híbrido de Monte Carlo

O algoritmo híbrido de Monte Carlo é baseado o algoritmo anterior mas adicionando uma variável de momento para melhor explorar o espaço. Defina $E = \mathbb{R}^{dN}$ e deixe $U_N : E \to \mathbb{R}$ ser suave para que $e^{-\beta_N U_N}$ seja Lebesgue integrável. Seja ainda $(X_t, Y_t)_{t>0}$ ser o processo de difusão em $E \times E$ solução de

$$\begin{cases} dX_t = \alpha_N \nabla U_N(Y_t)dt \\ dY_t = \alpha_N \nabla H_N(X_t)dt - \gamma_N \alpha_N \nabla U_N(Y_t)dt + \sqrt{2\frac{\gamma_N \alpha_N}{\beta_N} dB_t} \end{cases}$$

Onde $(B_t)_{t>0}$ é o movimento browniano em E e $\gamma_N > 0$ parâmetro representando atrito.

Quando $U_N(y) = \frac{1}{2}|y|^2$ temos $Y_t = dX_t/dt$ e teremos que X_t e Y_t poderão ser interpretados como posição e velocidade do sistema de N pontos em S no tempo t. Nesse caso, U_n é energia cinética.

O algoritmo discreto

Descrevemos agora o algoritmo discretizado. Inicie de uma configuração (x_0, y_0) e para todo $k \geq 0$ faça

1. atualize as velocidades com

$$\tilde{y}_k = \eta y_k + \sqrt{\frac{1-\eta^2}{\beta_N}} G_k, \ \eta = e^{-\gamma_N \alpha_N \Delta t}$$

2. calcule os termos

$$\begin{cases} \tilde{y}_{k+\frac{1}{2}} = \tilde{y}_k - \nabla H_N(x_k)\alpha_N \frac{\Delta t}{2} \\ \tilde{x}_{k+1} = x_k + \tilde{y}_{k+\frac{1}{2}} \alpha_N \Delta t \\ \tilde{y}_{k+1} = \tilde{y}_{k+\frac{1}{2}} - \nabla H_N(x_{k+1})\alpha_N \frac{\Delta t}{2} \end{cases}$$

3. definir p_k

$$p_k = 1 \wedge \exp\left[-\beta_N \left(H_N(\tilde{x}_{k+1}) + \frac{\tilde{y}_{k+1}^2}{2} - H_N(x_k) - \frac{\tilde{y}_k^2}{2}\right)\right]$$

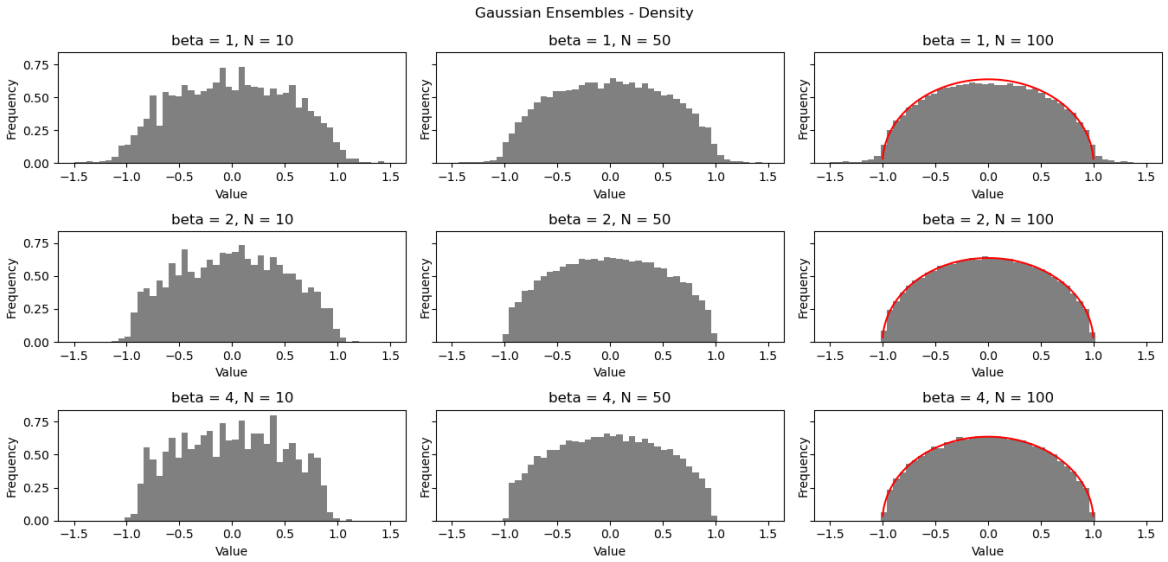


Figura 5.1: Validação para ensembles clássicos, utilizamos 200000 passos registrando a cada 500 a partir da metade dos passos. $\Delta t = 0.1$, $\gamma = 1$, $\alpha = 1.0$. Para replicar a semente foi 987991650.

4. defina

$$(x_{k+1}, y_{k+1}) = \begin{cases} (\tilde{x}_{k+1}, \tilde{y}_{k+1}) \ w/ \ p_k \\ (x_k, -\tilde{y}_k) \ w/ \ 1 - p_k \end{cases}$$

5.3 Validando a implementação

5.3.1 Validando nosso programa

A implementação simples do algoritmo descrito está disponível em GitHub - Algoritmo Artigo. Também fica disponível no anexo A. Podemos ainda checar os ensembles clássicos e ver se podemos observar a formação da lei do semicírculo de Wigner.

5.3.2 Outras distribuições

Potencial Quártico

Potencial Mônico

5.3.3 Paralelização

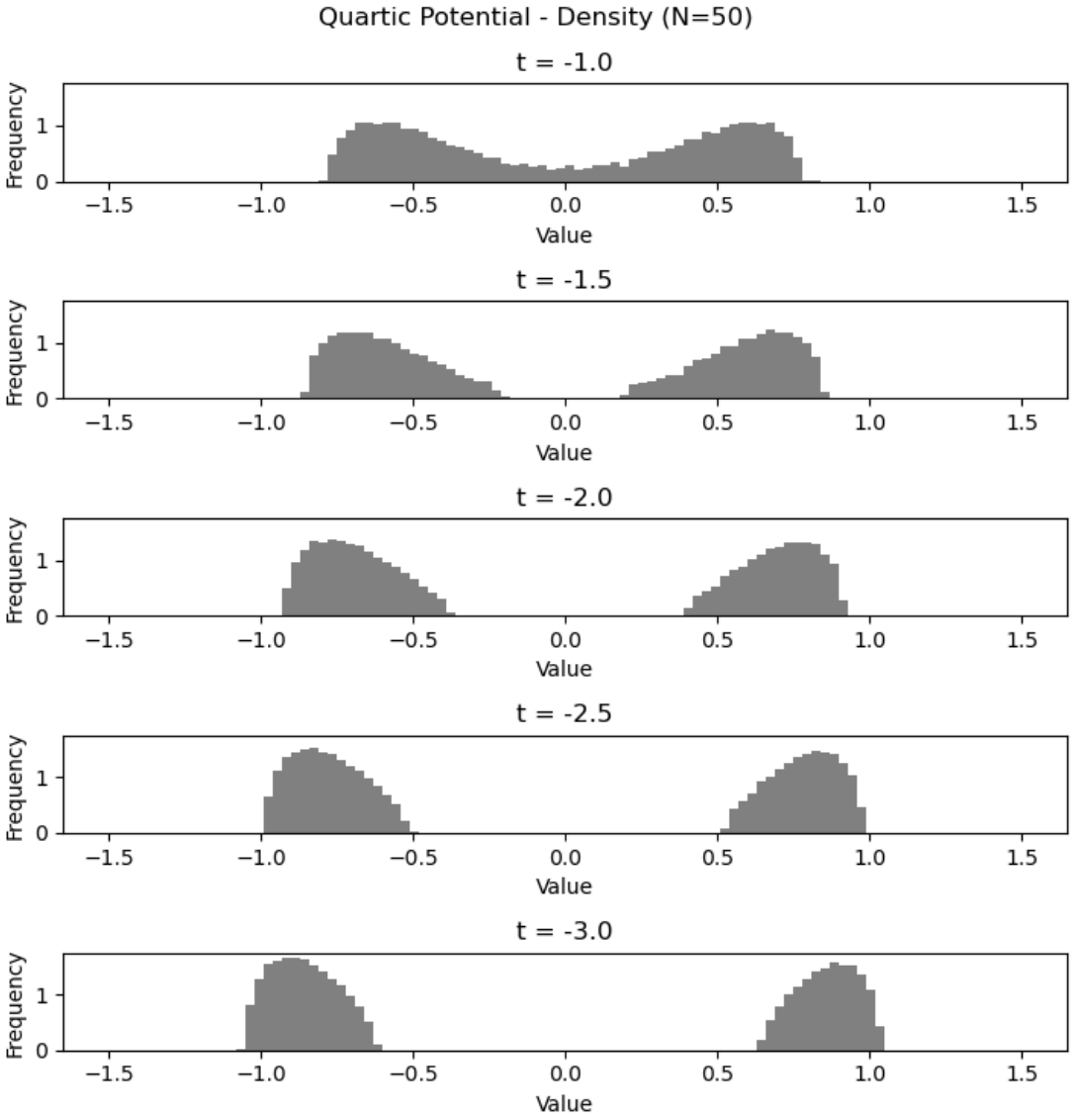


Figura 5.2: Validação para potencial quártico, $V(x) = \frac{1}{4}x^4 + \frac{1}{2}x^2$. Utilizamos 1000000 passos registrando a cada 500 a partir da metade dos passos. $\Delta t = 0.1$, $\gamma = 10$, $\alpha = 0.1$. Para replicar a semente foi 987991650.

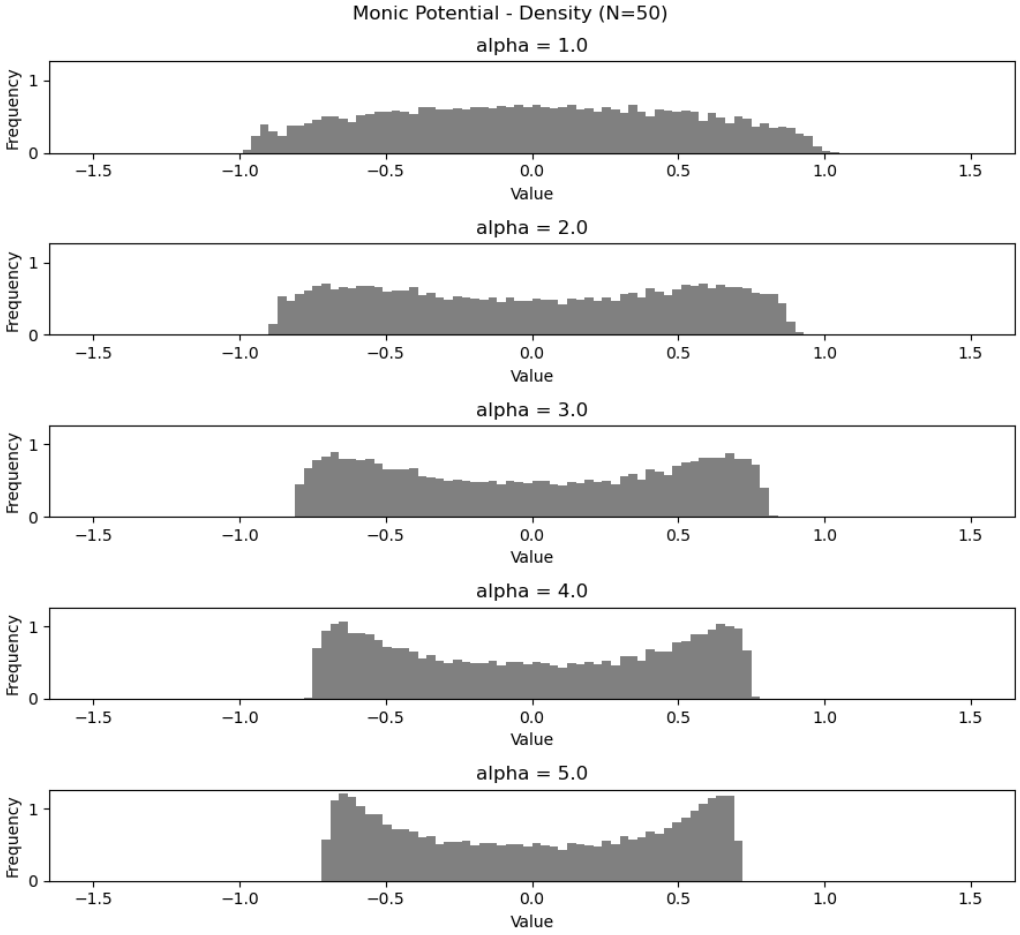


Figura 5.3: Validação para potencial mônico, $V(x) = \frac{t}{2\alpha}x^{2\alpha}$. Utilizamos 1000000 passos registrando a cada 500 a partir da metade dos passos. $t = 1$, $\Delta t = 0.1$, $\gamma = 10$, $\alpha = 0.1$. Para replicar a semente foi 987991650.

Bibliografia

[1] Djalil Chafaï and Grégoire Ferré. Simulating coulomb and log-gases with hybrid monte carlo algorithms. *Journal of Statistical Physics*, 174(3):692–714, nov 2018.

[2] Yan V. Fyodorov. Introduction to the random matrix theory: Gaussian unitary ensemble and beyond, 2010.

[3] Arno Kuijlaars. Lecture notes on riemann-hilbert problems and multiple orthogonal polynomials.

[4] Giacomo Livan, Marcel Novaes, and Pierpaolo Vivo. *Introduction to Random Matrices*. Springer International Publishing, 2018.

[5] M.L. MEHTA. Preface. In M.L. MEHTA, editor, *Random Matrices and the Statistical Theory of Energy Levels*. Academic Press, 1967.

[6] James Propp. Generalized domino-shuffling, 2002.

Apêndice A

Algoritmo Artigo

```
C########################################################################
C           Implementation of Algorith described in the paper
C  "Simulating Coulomb and Log-Gases with Hybrid Monte Carlo Algorithms"
C                 by Djalil Chafaï and Gregoire Ferré
C            DOI: https://doi.org/10.1007/s10955-018-2195-6
C
C     Author: Pimenta, J. V. A.
C########################################################################
C########################################################################
      PROGRAM HMC

      IMPLICIT REAL*8 (A-H,O-Z)

C########################################################################
C########################################################################
C     Considerations:
C
C     We onsider gases that in a subspace S of R^n, with dimension m
C
C     The external potential is given by V : S -> R
C
C     The interaction potential is given by W : S -> (-inf, inf]
C
C     We take N particles in S, and for any N >= 2, we define P_n to be
C     P_N = 1/Z_N exp(-beta H_N(x_1, ..., x_N)) dx_1 ... dx_N
C     .: Z_N is the partition function, beta is the inverse temperature
C     and H_N is the Hamiltonian
C
C     The Hamiltonian is given by
C     H_N(x_1,...,x_N) = 1/N sum_{i=1}^N V(x_i) +
C                        1/2N^2 sum_{i \neq j} W(x_i - x_j)
C
C     Note that the Hamiltonian is invariant by permutation and func of
C     \mu_N = 1/N sum_{i=1}^N \delta_{x_i}
C
C     It is know that the empirical measure converges to an non random
C     measure, i.e. \mu_N -> arg inf \epsilon(\mu) where
C     \epsilon(\mu) = \int V(x)d\mu(x)
```

```
39  C                   + 1/2 \int\int W(x-y)d\mu(x)d\mu(y)
40  C
41  C ---------------------------------------------------------------------
42  C
43  C     The Hybrid Monte Carlo algorithm is described as follows:
44  C
45  C     Define a space E = R^mn and let U_N : E -> R be smooth such that
46  C     exp{-\beta_N U_N} is a density w.r.t. the Lebesgue measure on E
47  C
48  C     Let (X_t, V_t) be a Markov chain on E x E with invariant measure
49  C     \pi_N(dx dv) = exp{-\beta_N U_N(x)} dx dp solution of
50  C
51  C     dX_t = \alpha_N \Nabla U_N(Y_t) dt
52  C     dY_t = - \alpha_N \Nabla H_N(X_t) dt
53  C            - \gamma_N \alpha_N \Nabla U_N(X_t) dt
54  C            + \sqrt{2 ((\gamma_N \alpha_N) / \beta_N) dB_t}
55  C
56  C     where (B_t)_{t>0} is a Brownian motion on E
57  C     \gamma_N is a coeficient that represents the friction
58  C     \alpha_N is a coeficient that represents the step size
59  C
60  C#####################################################################
61  C#####################################################################
62  C     Parameters:
63  C
64  C     N: number of particles, scalar
65  C     m: dimension of the space, scalar
66  C     beta: inverse temperature, scalar
67  C     alpha: step size, scalar
68  C     gamma: friction, scalar
69  C     nsteps: number of steps, scalar
70  C     eta: coeficient exp{-\gamma_N \alpha_N \delta t}, scalar
71  C     sdn: coeficient \sqrt{(1 - \eta^2) / \beta_N}, scalar
72  C     pi: coeficient \pi, scalar
73  C     tstep: time step, scalar
74  C     niter: number of iterations to register, scalar
75  C     p: acceptance probability, scalar
76  C
77  C     xk: position at k, vector of size (N,m)
78  C     xtildek1: candidate positions for k+1, vector of size (N,m)
79  C     vk: velocities at k, vector of size (N,m)
80  C     vtilde: gaussian modified velocities, vector of size (N,m)
81  C     vtildek1: candidate velocities for k+1, vector of size (N,m)
82  C     F: force, vector of size (N,m)
83  C     GVe: gradient of the external potential, vector of size (m)
84  C     GW: gradient of the interaction potential, vector of size (m)
85
86        PARAMETER (N = 100, m = 1, beta = 4.0, alpha = 1)
87
88        DIMENSION xk(N,m), xtildek1(N,m),
89       &      vk(N,m),vtilde(N,m),vtildek1(N,m),
90       &      F(N,m), GVe(m), GW(m)
```

```
91
92          COMMON /X/ xk, xtildek1
93          COMMON /V/ vk, vtilde, vtildek1
94          COMMON /G/ F, GVe, GW
95
96          nsteps = 200000
97          niter = 500
98          tstep = 0.1
99          gamma = 1.0
100
101         eta = EXP(-gamma * alpha * tstep)
102         sdn = SQRT((1 - eta**2) / beta) / N
103         pi = ACOS(-1.d0)
104
105         call srand(987991650)
106
107 C#########################################################################
108 C#########################################################################
109 C     Step 1: Initialization of (x0, v0)
110 C     We take x0 = 0 and v0 = 0
111 C     We also initialize xk = x0 and vk = v0
112
113         CALL INIT()
114
115 C -----------------------------------------------------------------------
116
117         OPEN(1,FILE='dataX.txt',STATUS='UNKNOWN')
118         OPEN(2,FILE='dataV.txt',STATUS='UNKNOWN')
119
120         DO 10 k = 1, nsteps
121
122 C -----------------------------------------------------------------------
123 C     Step 2: Update the velocities with
124 C     vtilde_k = \eta vk_k + \sqrt{(1 - \eta^2) / \beta_N} \G_k
125 C     where \G_k is a standard Gaussian vector
126 C     and \eta = \exp{-\gamma_N \alpha_N \delta t}
127
128         CALL GaussianV(eta, sdn, pi)
129
130 C -----------------------------------------------------------------------
131 C     Step 3: Calculate
132 C     vtildehalf_k = vtilde_k - \alpha_N \Nabla H_N(xk) timestep/2
133 C     xtildek1_k = xk + \alpha_N vtildehalf_k timestep
134 C     vtilde = vtildehalf_k - \alpha_N \Nabla H_N(xtildek1_k) timestep/2
135 C
136 C     Note that we update a half step of the velocities then we update
137 C     the positions and then we update the velocities again
138
139         CALL UPDATE(tstep, alpha, beta)
140
141 C -----------------------------------------------------------------------
142 C     Step 4: define the acceptance probability
143 C     prob=1 ^ exp{-\beta_N (H_N(xtildek1_k)-H_N(xk)+vtilde_k^2/2-vk^2/2)}
```

```
144
145         p = PROB(beta)
146
147   C -------------------------------------------------------------------
148   C     Step 5: accept or reject the candidate with probability p
149
150         IF (RAND(0) <= p) THEN
151               xk = xtildek1
152               vk = vtildek1
153         ELSE
154               vk = -vtildek1
155               CALL GRAD_H(.FALSE., beta)
156         END IF
157
158   C -------------------------------------------------------------------
159   c     Save some data every 1000 steps
160         IF (k > nsteps/2) THEN
161               IF (MOD(k,niter) == 0) THEN
162                     WRITE(1,*) xk / SQRT(2*beta)
163                     WRITE(2,*) vk
164                     WRITE(*,*) k, XK(1,1), VK(1,1)
165               END IF
166         END IF
167
168   C -------------------------------------------------------------------
169
170    10   END DO
171
172         CLOSE(1)
173         CLOSE(2)
174
175   C#########################################################################
176
177         END PROGRAM HMC
178
179   C#########################################################################
180   C     Subroutines:
181
182   C     INIT: initialization of (x0, v0)
183   c           modifies xk, vk, F
184         SUBROUTINE INIT()
185               PARAMETER(N = 100, m = 1)
186               IMPLICIT REAL*8 (A-H,O-Z)
187               DIMENSION xk(N,m), xtildek1(N,m),
188        &                vk(N,m),vtilde(N,m),vtildek1(N,m),
189        &                F(N,m), GVe(m), GW(m)
190               COMMON /X/ xk, xtildek1
191               COMMON /V/ vk, vtilde, vtildek1
192               COMMON /G/ F, GVe, GW
193
194               DO i = 1, N
195               DO j = 1, m
196
```

```
197                   xk(i,j) = -1.0 + 2*RAND(0)
198                   vk(i,j) = 0.0
199                   F(i,j) = 0.0
200
201             END DO
202             END DO
203
204       END SUBROUTINE INIT
205
206 C     GaussianV: update the velocities with the gaussian variable
207 c           modifies vtilde
208       SUBROUTINE GaussianV(eta, sdn, pi)
209             PARAMETER(N = 100, m = 1)
210             IMPLICIT REAL*8 (A-H,O-Z)
211             DIMENSION vk(N,m),vtilde(N,m),vtildek1(N,m)
212             COMMON /V/ vk, vtilde, vtildek1
213
214             DO i = 1, N
215             DO j = 1, m
216                   vtilde(i,j) = eta * vk(i,j) + sdn * Gauss(pi)
217             END DO
218             END DO
219
220       END SUBROUTINE GaussianV
221
222 C     UPDATE: update the positions and velocities
223 c           modifies xtildek1 and vtildek1
224       SUBROUTINE UPDATE(tstep, alpha, beta)
225             PARAMETER(N = 100, m = 1)
226             IMPLICIT REAL*8 (A-H,O-Z)
227             DIMENSION xk(N,m), xtildek1(N,m),
228      &                vk(N,m),vtilde(N,m),vtildek1(N,m),
229      &                F(N,m), GVe(m), GW(m)
230             COMMON /X/ xk, xtildek1
231             COMMON /V/ vk, vtilde, vtildek1
232             COMMON /G/ F, GVe, GW
233
234             DO i = 1, N
235                   vtilde(i,:) = vtilde(i,:) + alpha*F(i,:)*tstep/2
236                   xtildek1(i,:) = xk(i,:) + alpha*vtilde(i,:)*tstep
237             END DO
238
239             CALL GRAD_H(.TRUE., beta)
240
241             DO i = 1, N
242                   vtildek1(i,:) = vtilde(i,:) + alpha*F(i,:)*tstep/2
243             END DO
244
245       END SUBROUTINE UPDATE
246
247 C     GRAD_H: gradient of the Hamiltonian (force)
248 c           modifies F, GVe and GW
```

```
249         SUBROUTINE GRAD_H(next, beta)
250               PARAMETER(N = 100, m = 1)
251               IMPLICIT REAL*8 (A-H,O-Z)
252               DIMENSION x(N,m), xk(N,m), xtildek1(N,m),
253        &                F_aux(N,N,m), F(N,m), GVe(m), GW(m)
254               LOGICAL next
255               COMMON /G/ F, GVe, GW
256               COMMON /X/ xk, xtildek1
257
258               IF (next) THEN
259                     x = xtildek1
260               ELSE
261                     x = xk
262               END IF
263
264               DO i = 1, N
265                     DO j = 1, i-1
266                           CALL GRAD_W(x(i,:), x(j,:))
267                           F_aux(i,j,:) = - GW
268                     END DO
269               END DO
270
271               DO i = 1, N
272
273                     F(i,:) = 0.0
274
275                     DO j = 1,i-1
276                           F(i,:) = F(i,:) + F_aux(i,j,:)
277                     END DO
278
279                     DO j = i+1,N
280                           F(i,:) = F(i,:) - F_aux(j,i,:)
281                     END DO
282
283                     F(i,:) = F(i,:) / N
284
285                     CALL GRAD_Ve(x(i,:), beta)
286                     F(i,:) = F(i,:) - GVe
287
288                     F(i,:) = F(i,:) / N
289
290               END DO
291
292         END SUBROUTINE GRAD_H
293
294   C     GRAD_Ve: gradient of the external potential
295   c           modifies GVe
296         SUBROUTINE GRAD_Ve(x, beta)
297               PARAMETER(N = 100, m = 1)
298               IMPLICIT REAL*8 (A-H,O-Z)
299               DIMENSION x(m), F(N,m), GVe(m), GW(m)
300               COMMON /G/ F, GVe, GW
```

```
301
302   c                 Gradient of V(x) = ||x||^2 / (2 * beta) Beta - Hermite
303               GVe = x / beta
304
305         END SUBROUTINE GRAD_Ve
306
307   C     GRAD_W: gradient of the interaction potential
308   c           modifies GW
309         SUBROUTINE GRAD_W(x, y)
310               PARAMETER(N = 100, m = 1)
311               IMPLICIT REAL*8 (A-H,O-Z)
312               DIMENSION x(m), y(m), v(m),
313        &                F(N,m), GW(m), GVe(m)
314               COMMON /G/ F, GVe, GW
315
316   c                 Gradient of W(x) = -log(||x||)
317               v = x-y
318               GW = -v / NORM2(v)**2
319
320         END SUBROUTINE GRAD_W
321   C#########################################################################
322   C#########################################################################
323   C     Functions:
324   C     1-FUNCTION ARE CALLED IN MAIN
325   C     2-FUNCTION ARE CALLED IN SUBROUTINES
326   C     3-FUNCTION ARE CALLED IN FUNCTIONS
327
328   C     (2-FUNCTION) Gauss: gaussian variable
329   c           return a standard gaussian variable, scalar
330         FUNCTION Gauss(pi)
331               IMPLICIT REAL*8 (A-H,O-Z)
332               Gauss = 1000
333               DO WHILE (Gauss > 100)
334                     Gauss = sqrt(-2.*LOG(RAND()))*COS(2.*pi*RAND())
335               END DO
336               RETURN
337         END FUNCTION Gauss
338
339   C     (1-FUNCTION) PROB: calculate the acceptance probability
340   c           return the acceptance probability, scalar
341         FUNCTION PROB(beta)
342               PARAMETER(N = 100, m = 1)
343               IMPLICIT REAL*8 (A-H,O-Z)
344               DIMENSION vk(N,m),vtilde(N,m),vtildek1(N,m)
345               COMMON /V/ vk, vtilde, vtildek1
346
347               Hi = H(.FALSE.,beta)
348               Hf = H(.TRUE.,beta)
349
350               PROB = EXP(-beta * (Hf-Hi))
351
352               DO i = 1, N
353                     dtilde = DOT_PRODUCT(vtildek1(i,:),vtildek1(i,:))
```

```
354                   dk = DOT_PRODUCT(vk(i,:),vk(i,:))
355                   PROB = PROB * EXP(-beta * (dtilde - dk) / 2)
356             END DO
357
358             RETURN
359             END FUNCTION PROB
360
361   C     (3-FUNCTION) H: Hamiltonian
362   c           return the Hamiltonian, scalar
363         FUNCTION H(next, beta)
364             PARAMETER(N = 100, m = 1)
365             IMPLICIT REAL*8 (A-H,O-Z)
366             LOGICAL next
367             DIMENSION x(N,m), xk(N,m), xtildek1(N,m)
368             COMMON /X/ xk, xtildek1
369
370             if (next) then
371                   x = xtildek1
372             else
373                   x = xk
374             end if
375
376             H = 0.0
377
378             DO i = 1, N
379                   H = H + Ve(x(i,:), beta)
380                   DO j = i+1, N
381                         H = H + W(x(i,:), x(j,:)) / (2*N)
382                   END DO
383             END DO
384
385             H = H / N
386
387             RETURN
388         END FUNCTION H
389
390   C     (3-FUNCTION) Ve: external potential
391   c           return the external potential, scalar
392         FUNCTION Ve(x, beta)
393             PARAMETER(N = 100, m = 1)
394             IMPLICIT REAL*8 (A-H,O-Z)
395             DIMENSION x(m)
396
397   c                   V(x) = ||x||^2 / (2*beta) Beta - Hermite
398             Ve = DOT_PRODUCT(x,x) / (2*beta)
399
400             RETURN
401         END FUNCTION Ve
402
403   C     (3-FUNCTION) W: interaction potential
404   c           return the interaction potential, scalar
405         FUNCTION W(x, y)
```

```
406               PARAMETER(N = 100, m = 1)
407               IMPLICIT REAL*8 (A-H,O-Z)
408               DIMENSION x(m), y(m)
409
410   c                   W(x) = -log(||x||)
411               W = -LOG(NORM2(x-y))
412
413               RETURN
414         END FUNCTION W
415
416   C#########################################################################
```

Apêndice B

Transformação Legendre

Transformações de Legendre tem ampla aplicação e interpretação. Faremos uma breve dissertação de duas possíveis visualizações do processo.

B.1 Tangentes

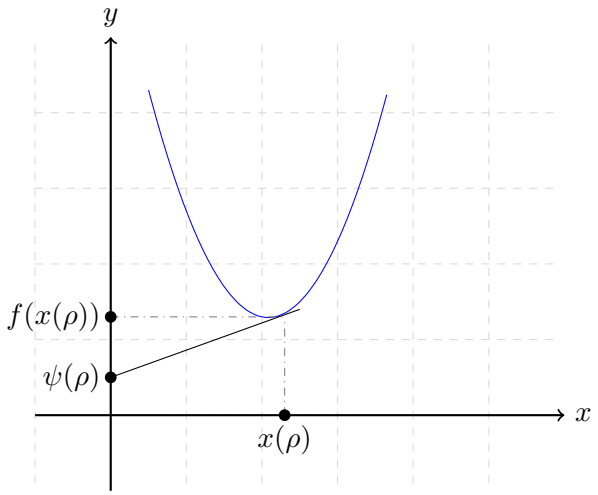
Uma primeira interpretação do processo representa uma mudança clara de variável a partir de tangentes de uma função de concavidade bem definida. Faremos

$$y = f(x) \mapsto \psi(\rho) = f(x(\rho)) - \rho x(\rho)$$

onde

$$\rho \equiv \frac{y - f(x(\rho))}{x - x(\rho)} = \frac{\psi(\rho) - f(x(\rho))}{0 - x(\rho)}$$

Que podemos visualizar



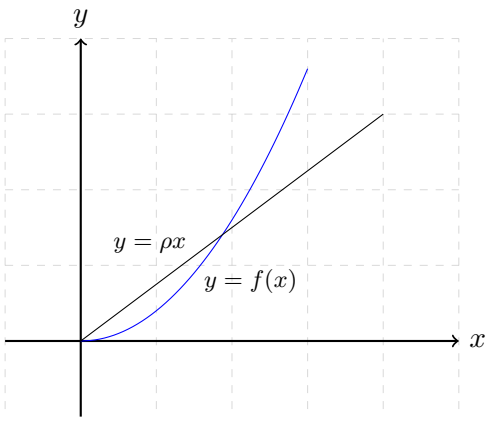
De forma que a transformada é criada pela projeção desta tangente no eixo x=0.

B.2 Otimização

Outra forma de visualizar a transformada é por um problema mais conveniente de otimização. Definiremos

$$\psi(\rho) = \max_x [\rho x - f(x)]$$

Onde teremos



Se definirmos

$$g(x) = \rho x - f(x)$$

Minimizaremos $g(x)$ e teremos a condição $f'(x^*) = \rho$. Ou seja

$$f'(x(\rho)) = \rho \implies \max_x [\rho x - f(x)] = \rho x(\rho) - f(x(\rho))$$

Que é equivalente à dizer

$$\psi(\rho) = \min_x [f(x) - \rho x] = f(x(\rho)) - \rho x(\rho)$$

Apêndice C

Soma Assintóticas

Existe uma aproximação a se fazer no logaritmo de somas assintóticas que pode ser de interesse. Assuma

$$S = \sum_{i=1}^{M} a_i$$

Se $a_i(N) \sim e^{\phi_i N}$ ou $\log(a_i) \approx \phi_i N$, podemos afirmar

$$\log(S) \sim \log(a_{max})$$

Para demonstrar isso, notamos

$$a_{max} < S < M_{a_m ax}$$
$$\frac{\log(a_{max})}{N} < \frac{\log S}{N} < \frac{\log(a_{max})}{N} + \frac{\log M}{N}$$

Ou seja, desde que $\frac{\log M}{N} \to 0$. Isto ocorrerá desde que M seja sub-exponencial. Contudo **NOTE** que dizer

$$\log(n!) \sim n \log n - n$$

Não implica que

$$n! \sim! \left(\frac{n}{e}\right)^n$$

Em algumas situações é possível afirmar contudo

$$n! \sim \left(\frac{n}{e}\right)^n \sqrt{2\pi n}$$

Apêndice D

Det Vandermonde

A formulação nos diz

$$|V| = \prod_{1 \leq i < j \leq n} (\alpha_j - \alpha_i) \tag{D.1}$$

Note que podemos iniciar com uma matriz $n \cdot n$. Seja c_i a coluna i, multiplicamos a coluna c_i por $-\alpha_1$ e somamos com a coluna c_{i+1}

$$V = \begin{bmatrix} 1 & \alpha_1 & \alpha_1^2 & \dots & \alpha_1^{n-1} \\ 1 & \alpha_2 & \alpha_2^2 & \dots & \alpha_2^{n-1} \\ 1 & \alpha_3 & \alpha_3^3 & \dots & \alpha_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_n & \alpha_n^2 & \dots & \alpha_n^{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & \alpha_2 - \alpha_1 & \alpha_2(\alpha_2 - \alpha_1) & \dots & \alpha_2^{n-2}(\alpha_2 - \alpha_1) \\ 1 & \alpha_3 - \alpha_1 & \alpha_3(\alpha_3 - \alpha_1) & \dots & \alpha_3^{n-2}(\alpha_3 - \alpha_1) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_n - \alpha_1 & \alpha_n(\alpha_n - \alpha_1) & \dots & \alpha_n^{n-2}(\alpha_n - \alpha_1) \end{bmatrix}$$

Utilizando do Teorema de Laplace, o determinante vai ser definido simplesmente por

$$|V| = \begin{vmatrix} \alpha_2 - \alpha_1 & \alpha_2(\alpha_2 - \alpha_1) & \dots & \alpha_2^{n-2}(\alpha_2 - \alpha_1) \\ \alpha_3 - \alpha_1 & \alpha_3(\alpha_3 - \alpha_1) & \dots & \alpha_3^{n-2}(\alpha_3 - \alpha_1) \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_n - \alpha_1 & \alpha_n(\alpha_n - \alpha_1) & \dots & \alpha_n^{n-2}(\alpha_n - \alpha_1) \end{vmatrix}$$

De onde é claro, podemos fatorar os coeficientes e ter

$$|V| = (\alpha_2 - \alpha_1)(\alpha_3 - \alpha_1) \dots (\alpha_n - \alpha_1) \begin{vmatrix} 1 & \alpha_2 & \alpha_2^2 & \dots & \alpha_2^{n-2} \\ 1 & \alpha_3 & \alpha_3^2 & \dots & \alpha_3^{n-2} \\ 1 & \alpha_4 & \alpha_4^3 & \dots & \alpha_4^{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_n & \alpha_n^2 & \dots & \alpha_n^{n-2} \end{vmatrix}$$

E assim por diante.