ZTF-FCT
Zientzia eta Teknologia Fakultatea
Facultad de Ciencia y Tecnología

Zenbait sistema fisikoren konputazioaren bidezko modelizazioa eta bisualizazioa I: Sistema Termikoak

Gradu Amaierako Lana
Fisikako Gradua

Ander Arbide Bermudez

Zuzendaria
Josu Mirena Igartua

Leioa, 2018ko Otsilaren 12a

Aurkibidea

1 Sarrera

Graduaren zehar nabaritu dut zenbaitetan klasean emandako teoriaren adibiderik ez ditugula ikusi. Bakarrik ematen dizkigute formulak, eta guk ez dakigu zein den hauen itxura. Beraz pentsatu nuen irakaskuntzan erabiltzeko erabilgarria izango litzakela hurrengo lana egitea.

Lan hau hiru helburu ditu, lehenengoa zenbait sistema termodinamiko desberdinak aztertu eta metodo konputazionalak erabiliz beraien propietateak lortzea, hala nola energia, magnetizazioa, eboluzio tenporala eta abar. Helburu hau lortzeko sistema termodinamikoak estatistika erabiliz aztertuko ditugu. Hau Python lengoaia erabiliz egingo dugu simulazioak eta adierazpen grafikoak erabiliz.

Lan honen bigarren helburua hau da, gure sistema termodinamikoak aztertuko ditugun programak eta teoriak era interaktiboan eraiki irakaskuntzan erabiltzeko. Azkenik hirugarren helburua bigarrenarekin lotuta dago eta gaur egungo tresna informatikoak erabiltzean datza egindako lana biltzeko eta eskugarri edukitzeko.

Hau egiteko *Jupyter Notebook* eta *Github* erabiliko ditugu. Lehenengoa oso trensa erabilgarria da, honetan gure teoria garatu dezakegu eta aldi berean *Python*-en kodigoa idatzi dezakegu teoriako adibideak ikusteko eta hobeto ulertzeko. Behin hau dugula gure *Jupyter Notebook*-a *Github*-en igo dezakegu edorren eskura egoteko.

Azkenik esan behar dugu *Python*-en zenbait luzapen ez oso ezagunak erabiliko ditugula lan hau egiteko, hauek *Vphyton* eta *BQplot* dira. Lehenengoa erabiliz gure sistemen eboluzio denboralaren bideoak eraiki ditzakegu. Bigarrenarekin gure programa interaktibo bihurtu dezakegu erabiltzaileari uzten zenbait parametro aukeratzen.

2 Oreka Termodinamikoa

Sistema termodinamikoak estatistikaren ikuspuntutik aztertzeko momentuan oinarrizko ideia *Boltzman*-en banaketa da. Honek deskribatzen du energiaren banaketa sistema termiko batean. Sistema termodinamikoen azterketarekin hasteko, oreka ikertuko dugu *Monte Carlo* metodoa erabiliz.

2.1 Oreka termodinamikoa lortu

Sistema termodinamiko batean egoera bat edukitzeko probabilitatea ulertzeko, hasiko gara kontatzen maila mikroskopikoan ditugun egoera posibleak. Demagun sistema termiko bat dugula partikula mikroskopikoz osatuta, hauek energia gorde dezakete era bibrazionalean, rotazionalean edo higiduran. Sistema honen adibide bat *Einstein*-en solido eredua da, non solidoa N osziladore harmoniko sinple (OHS) eta independientez osatuta dagoen. Osziladore harmoniko bakoitzak eduki ditzake nahi dituen energi kuanto. Orain demagun gure sistema duen energia osoa $q\epsilon$ dela, non ϵ energi kuantoa den. Energia hau banatu daiteke era ezberdinetan osziladore harmonikoen artean, energia banaketa bakoitza mikroegoera bat izango da.

Maila makroskopikoan aztertu nahi duguna kanpotik neurtu ditzakegun efektuak dira, hauei makroegoerak deitzen zaie. Kontsidera ditzagun bi mikroegoera lehen azaldu dugun adibidean, lehenengo mikroegoeran osziladore harmoniko bakar batek q energia kuanto edukiko ditu eta beste $N-1$ osziladoreen energia nulua izango da, beste mikroegoeran q osziladoreek energi kuanto bakarra edukiko dute eta besteek zero. Bi mikroegoera hauek makroegoera berdina dute, q energia osoa duena. Mikroegoeren kopurua makroegoera bakoitzean $\Omega(N,q)$ da, kasu batzuetan zenbaki hau era analitikoan lortu dezakegu.

Behin kontzeptu hauek azaldu ditugula oreka aztertuko dugu, kontsidera ditzagun bi solido kontaktuan daudenak, A eta B, bakoitzak N_A eta N_B osziladore dituztenek, biek q energia partekatzen dute, non $q = q_A + q_B$ den. Gure makroegoerari $n = n(q_A, q_B)$ deituko diogu, non $q_B = q - q_A$ kalkulatu dezakegun q_A emanda. Bestalde hurrengo propietatea betetzen da:

$$\Omega(n) = \Omega_A(N_A, q_A) x \Omega_B(N_B, q_B) \tag{1}$$

hauei mikroegoeren kopurua diegu, q_A eta q_B energi banaketa bakoitzerako.

Gure Ω probabilitateekin lotzeko, erabiliko dugu estatistikaren postulatu oso garrantzitsu bat, postulatu honek esaten duena da sistema isolatu batean dauden mikroegoera guztiek probabilitate berbera dutela gertatzeko. Hau erabiliz esan dezakegu gure sistema n makroegoeran egoteko probabilitatea ondorengoa dela:

$$P(n) = \frac{\Omega(n)}{\Omega_T} \tag{2}$$

non $\Omega_T = \sum_n \Omega(n)$ den.

Probabilitatea teorikoki kalkulatu dezakegu edo bere konportamoldea simulatu, guk bigarren erara egingo dugu. Azterketa hau egiteko, kontsideratuko dugu solido txiki bat beste handi batekin kontaktuan dagoena, hau da, $N_A \ll N_B$ betetzen da, hau gertatzen da solido batek ingurunearekin kontaktuan dagoenean.

Gure simulazioa egiteko komenigarriena *'objetu'* bat erabiltzea da *Einstein*-en solidoa eraikitzeko funtzio bat erabiltzea baino. Objetu hau erabiliko dugu programa askotan eta bi solidoen arteko interakziorako hedatu dezakegu. Hona hemen *Einstein*-en solidoaren objetua:

```
import random as rnd
            #Einsteinen solidoen eredua:
class Einstein:
    def __init__ (self,N=400,q=10):
        self.N=N
            #q energia kuanto gelaxka bakoitzean:
        self.cell=[q]*N

            #bi solidoak konbinatu
    def __add__ (self,other):
        self.N += other.N
        self.cell += other.cell
        return cell

            #Energia trukaketa:
    def exchange(self,L=20000):
        for i in range(L):
            #Lortu bi zorizko zenbaki:
            take=rnd.randint(0,self.N-1)
            give=rnd.randint(0,self.N-1)
            while self.cell[take]==0:
            #Aurkitu energia>0 duen gelaxka:
                take=rnd.randint(0,self.N-1)
            #Energia trukatu:
            self.cell[take] -= 1
            self.cell[give] += 1
```

1. Programa. *Einstein*-en solidoen eredua.

Programa honetan *Python*-eko *init*() funtzioari deitzen zaio N osziladore harmonikoak eratzeko eta beraien energiak gordetzeko zerrenda batean. Hurrengo *add*() funtzioak batuketa eragilea definitzen du, zeinek bi solido konbinatzen baditugu non $C = A + B$ ematen den, orduan $N_C = N_A + N_B$ emango

da eta gainera gelaxkak kateatuko dira. Gure sisteman onartuta dauden mikroegora guztiak probabilitate berdina dutenez, osziladore harmonikoei utziko diegu energia trukatzea askatasunez, hau *exchange*() progamak egiten du. Iterazio bakoitzean, era aleatorioan hartzen ditu bi osziladore $randint(n_1, n_2)$ funtzioa erabiliz, zeinek zenbaki bat n_1 eta n_2-ren artean bueltatzen duen. Behin aurkituta osziladore harmoniko bat $E > 0$ duena, kentzen dio energi kuanto bat eta beste osziladore harmonikoari ematen dio.

Behin hau egin dugula hurrengo programa egingo dugu *Einstein*-en solido baten energia banaketa adierazteko:

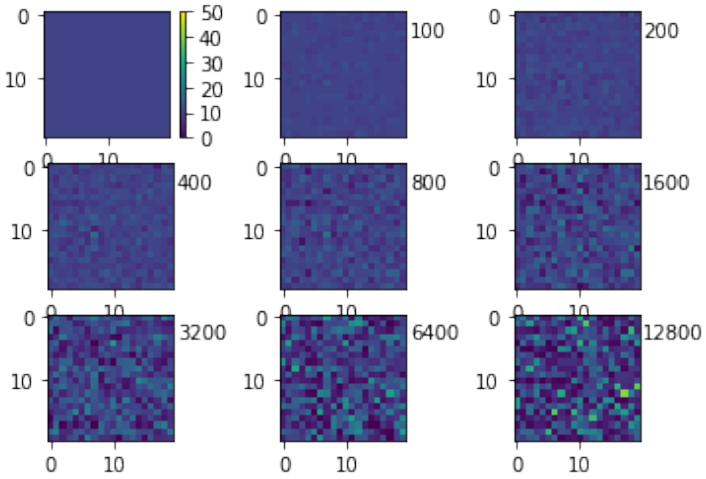
```
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.animation as am


def irudia(*args):
     #irudia berridatzi:
    plot.set_data(np.reshape(solid.cell,(k,k)))
    return [plot]

k=20
solid=Einstein(N=k*k,q=10)
L=100
fig=plt.figure()

for i in range(9):
     #irudi berri bat eraiki besteen alboan
    ax=fig.add_subplot(3,3,i+1)
     #Irudiaren itzura kxk izango da:
    img=np.reshape(solid.cell,(k,k))
    plot=plt.imshow(img,interpolation='none',vmin=0,vmax=50)
     #lehenengo irudian eskala jarri:
    if (i==0):
        plt.colorbar(plot)
     #irudia eraiki:
    anim=am.FuncAnimation(fig,irudia,interval=1,blit=True)
     #beste irudietan zenbat energi truke egon diren adierazi
        :
    if (i>0):
        plt.text(20,3.5,repr(L*2**(i-1)))
     #hurrengo irudian egongo diren energi trukeak:
    solid.exchange(L*2**i)
plt.show()
```

2. Programa. *Einstein*-en solido baten oreka.



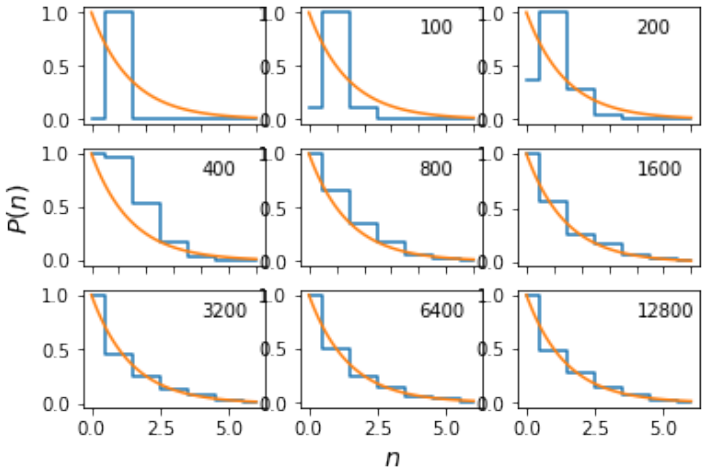
1. Irudia. Energien banaketa *Einstein*-en solido batean.

Hasieran osziladore guztiei eman diegu energia berdina, baina denbora pasa ahala ikusten dugu osziladore batzuk energia gehiago hartzen dutela, baina gehienak energia txikiarekin geratzen dira. Hau da, egoera probableena energia baxukoa da. Hau hobeto ulertzeko hurrengo progama irudikatuko dugu, kasu honetan $N = 1024$ osziladore erabiliko ditugu. Progama honek irudikatzen du n energia unitate dituzten osziladoreen zatidura erlatiboa histograma baten modura. Esan dugun modura hasieran osziladore guztiek $n = 1$ energia dute, baina gero $n = 0$ energia duen egoera bueltatzen da probableena.

```
import matplotlib.pyplot as plt
import numpy as np
    #Solidoa eratu:
solid=Einstein(N=1024,q=1)
L,M,kT=100,6,1./np.log(2.)
E,bin=np.linspace(0.,M,100),np.zeros(M+1)
fig=plt.figure()
    #irudi bakoitza eratu:
for i in range(9):
    ax=fig.add_subplot(3,3,i+1)
    for n in range(M+1):
        #Zenbat osziladore ditugu n energia kuantoekin:
        bin[n]=solid.cell.count(n)
    #histograma eraiki:
    plt.step(range(len(bin)),bin/max(bin),where='mid')
    #emaitza teorikoa:
    plt.plot(E,np.exp(-E/kT))
    if (i==3):
        plt.ylabel('$P(n)$',fontsize=14)
```

```
    if (i==7):
        plt.xlabel('$n$',fontsize=14)
    if (i<=5):
        plt.xticks(range(M+1),['' for j in range(M+1)])
    if (i>0):
        #iterazio kopurua adierazi:
        plt.text(M-2,0.8,repr(L*2**(i-1)))
    #iterazio berri bat kalkulatu:
    solid.exchange(L*2**i)
plt.show()
```

3. Programa. Energien banaketen probabilitateak.



2. Irudia. 3. progamako emaitza.

Ikusi dezakegunez azkenengo hiru irudietan banaketa ez da asko aldatzen, honek esan nahi duena da, gure sistema orekara heldu dela. Bestalde, nabaria denez probabilitatearen itxura exponentzial batena da:

$$P(n) = Ce^{-n\alpha} \tag{3}$$

Egin dugun simulazioan, daukagun datu bakarra bi sistemen osziladore harmonikoen kopurua $N = N_A + N_B$ da. Gure sistema, aztertu dugun modua dela eta, pentsa dezakegu bi solido ditugula kontaktuan: $N_A = 1$ eta $N_B = 1023$. Iterakzioak egitean, hartzen ditugu bi edozein osziladore eta beraien artean energia trukatzen dugu, baina osziladore guztiak berdinak direnez daukagun efektua osziladore bat (A solidoa) beste guztiekin (B solidoa) energia trukatzen duela da. Metodo honi *Monte Carlo* metodoa deitzen zaio.

Nahi izanez gero, gure programa egiteko hartu dezakegu osziladore bakar bat eta energia trukatu banan banan beste guztiekin, iterakzioak eginez emaitza berdina lortu behar da, baina programa pisutsuagoa izango litzateke.

2.2 Entropia

Aurreko atalean ikusi dugun modura, gure sistema termikoa hasierako egoera ordenatu batetatik beste egorea nahasi batera joaten da. Orain egingo duguna da, nabaritu dugun joera hau parametro fisiko batekin erlazionatuko dugu: *Entropia*. Entropiak sistema baten desorden maila ematen digu. *Boltzmann*-ek honela definitu zuen entropia:

$$S = k_b ln\Omega \tag{4}$$

non k_b *Boltzmann*-en konstatea den. Entropiaren propietate garrantzitsu bat da batukorra dela, hau da, sistema konposatu baten entropia osoa sistema bakoitzaren entropien batura dela. Behin hau definitu dugula lehen deskribatu dugun solidoaren entropia kalkula dezakegu. Horretarako, demagun gure sistema iturri batekin kontaktuan dagoela, beraz esan dugun modura $S + S_i = S_T$ izango da, eta $S = P_1S(1) + P_2S(2) + ... = \sum_n P_nS(n)$ denez iturriaren entropia idatzi dezakegu:

$$S_R = k_b \sum_n P_n ln\Omega_n \tag{5}$$

Bestalde lehen definitu dugun modura $P(n) = \dfrac{\Omega}{\Omega_T}$ da, beraz:

$$S_R = k_b \sum_n P_n(lnP_n + ln\Omega_T) = k_b \sum_n P_n lnP_n + k_b ln\Omega_T \tag{6}$$

Gogoratu probabilitateak normalizatuta daudela: $\sum_n P_n = 1$. Dena bere lekuan ordezkatuz:

$$S = S_T - S_R = S_T - k_b \sum_n P_n lnP_n - k_b ln\Omega_T = -k_b \sum_n P_n lnP_n \tag{7}$$

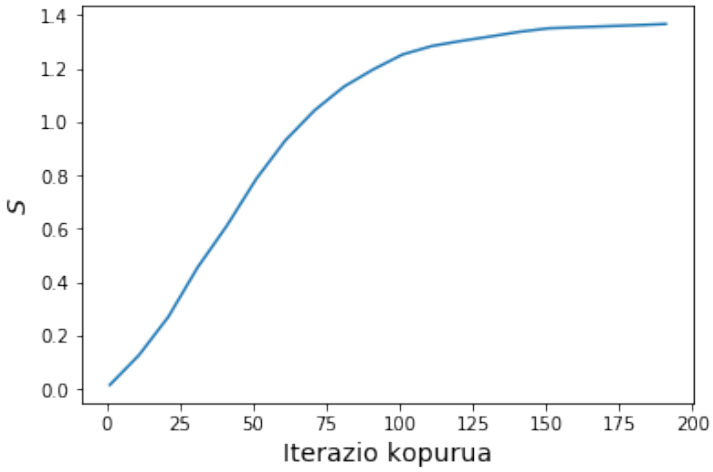
Behin hau lortu dugula, hurrengo funtzioa eraiki dezakegu osziladore baten entropia kalkulatzeko sistema baten barruan:

```
def entropia(cell):
    #Einstein-en solidoaren entropia
    N,n,nt,s=len(cell),0,0,0.
```

```
    #Gelaxka guztiak kontatu hartu:
    while nt<N:
        #Zenbat gelaxka En energia dute:
        cn=cell.count(n)
        n,nt=n+1,nt+cn
        #En energia edukitzeko probabilitatea:
        p=cn/float(N)
        if (cn != 0):
            #Entropia kalkulatu 7. ekuazioa erabiliz:
            s=s-p*np.log(p)
    return s
```

4. Programa. *Einstein*-en solidoaren entropiaren kalkulua.

Funtzio hau lehen eraiki dugun 2.*programan* inplementatu dezakegu gure sistemaren entropia lortzeko, hau eginda dugun emaitza hau da:



3. Irudia. Osziladore harmoniko sinplez osatutako solido baten entropia.

Ikusi dezakegu nola hasieran entropia nulua dela, hau da, gure sistema guztiz ordenatuta dagoela, eta denbora aurrera egitean entropia handitzen da. Hemendik atera dezakegun ondorioa ondokoa da: entropia handitzera jotzen du. Hau termodinamikaren bigarren legea da, eta lehen ezarri dugun erlazioa dela eta (4.*Ekuazioa*) sistema bat orekan egongo da Ω handieneko makroegoeran dagoenean, hau da, entropia maximoa denean oreka lortu dugu.

2.3 Tenperatura

Intuizioak esaten digun moduan bi sistema orekan daude tenperatura berdina dutenean. Beraz pentsa dezakegu erlazio bat egongo dela entropia eta tenperaturaren artean. Dakigun modura bi sistemen arteko oreka entropia totala maximoa denean ematen da, hau da:

$$\frac{\partial S_T}{\partial q_B} = \frac{\partial S_B}{\partial q_B} + \frac{\partial S_A}{\partial q_B} = 0 \Rightarrow \frac{\partial S_B}{\partial q_B} = \frac{\partial S_A}{\partial q_A} \tag{8}$$

$q = q_A + q_B$ (hau da $S_A \neq S_A(q)$) erabili dugu azkeneko pausuan. Beraz lortu dugu magnitude bat berdina izan behar dena bi sistemak orekan egoteko, hau esan dugun modura tenperatura izan behar da. Bestalde, kontsideratuz tenperatura energiarekin handitzen dela honela definituko dugu:

$$\frac{1}{T} = \frac{\partial S}{\partial E} \tag{9}$$

Ikusi dugun modura gure *Einstein*-en solidoan entropia handitzen da energiarekin eta ondorioz tenperatura baita, baina hau ez da beti honela, zenbait sistema paramagnetikoek bakarrik gorde dezakete energi mugatu bat, kasu hauetan entropia txikitzen da eta ondorioz $\partial S/\partial E < 0$ izango da eta sistemaren tenperatura baita.

3 Ising eredua

Orain arte *Einstein*-en solidoen eredua aztertu dugu. Behin hau egin dugula aztertuko dugu eredu bat zeinetan partikulak beraien artean menpekoak diren. Era honetako eredurik sinpleena $Ising$ eredua da, honek oso erabilgarria da solido ferromagnetikoak aztertzeko. Gainera eredu hau nahiko sinplea denez ondo azaldu eta inplementatu dezakegu $Metropolis$ algoritmoan.

3.1 1 dimentsioko Ising-en eredua

Kontsideratuko dugu 1 dimentsioko $Ising$-en eredua, hau N momentu dipolar berdinetan oinarritzen da, spin hauek kate bat osatzen dute. Momentu dipolar (spin) bakoitzak bi egoeretan egon daiteke: gorantz: $s_i = 1$ edo beherantz: $s_i = -1$, beraz bi spinen arteko interakzio energia ondokoa da:

$$E_{i,j} = -\epsilon s_i s_j \tag{10}$$

Non ϵ parekatze energia den. $\epsilon > 0$ bada, interakzio energiak lagunduko du spin paraleloak izatera ferromagnetismoa sortuz; ϵ-en balioa atomoen

arteko indarrek zehazten dute. Guk artuko dugu gure analisia egiteko $\epsilon > 0$ eta tratatuko dugu sistema ferromagnetiko baten energi kuanto bezala. Sistemaren energi osoa kalkulatu nahi badugu bikote guztiak kontuan hartu behar ditugu, baina bakarrik kontuan hartuko ditugu ondoz ondoko bikoteak, kontuan hartuz distantzia handiko elkarrekintza ahula dela. Beraz 1D-ko $Ising$-en ereduaren energia osoa ondokoa da:

$$E = \sum_i E_{i,i+1} = -\epsilon \sum_{i=1}^{N} s_i s_{i+1} \tag{11}$$

Ikusten dugunez s_{N+1} terminoa daukagu gure garapenean, definitu gabe dagoena. Bi konponbide ditugu, bat termino hau kendu edo bestela muga baldintza periodikoak erabili. Lehenengo konponbidea ondo dago N oso handia bada. Guk muga baldintza periodikoak erabiliko ditugu:

$$s_{N+1} = s_1 \tag{12}$$

Imaginatu dezakegu muga baldintza hauekin gure katearen amaieran beste identiko bat jarri dugula edo gure katea eraztun bat osatzen duela.

3.1.1 Metropolis simulazioa

Gure $Ising$ ereduaren dinamika aztertzeko, gure sistema T tenperatura iturriarekin kontaktuan jarri behar dugu,tenperatura iturrien abantila da beroa hartu eta eman dezaketela beraien tenperatura aldatu gabe. Inolako kalkulorik egin gabe badakigu hasierako egoera bat emanez gure sistemak iturriarekin energia trukatuko duela spinen orientazioa aldatuz. Momentu dipolarren biraketa gertatuko da gure sistema orekara heldu arte, gero espin batzuen fluktuazioak gerta daiteke baina energiaren aldaketa oso txikia izango da. Behin hau argi dugula, hurrengo galdera da, nola simulatuko dugu prozesu hau? Egin dezakegu gure kalkulua era zuzenean, eratuz gure s_i mikroegoera aleatorioki. Arazoa da, mikroegoeren kopurua handiegia izan daitekela, adibidez $N = 1000$ badaukagu mikroegoeren kopurua $2^N \sim 10^{300}$ dela, beraz metodo hau ez da erabilgarria. Baina irtenbide bat daukagu: $Metropolis$ algoritmoa. Honen funtzionamendua ondokoa da:

1. Hasierako s_i mikroegoera zehaztu behar dugu. Edozein mikroegoera balio du, baina komenigarriagoa da egoera hotz batekin hastea, hau da, spin paralelo gehiago edukitzea antiparalelo baino.

2. Hartu behar dugu edozein i espin eta biratuko dugu gertatutako energia aldaketa(ΔE) neurtuz:

$$\Delta E = E_{bukaeran} - E_{hasieran} = 2\epsilon s_i(s_{i-1} + s_{i+1}) \tag{13}$$

3. Orain $\Delta E < 0$ bada espinaren biraketa honartuko dugu, kontuan izan prozesu fisikoetan naturak energia baxuko egoerak nahiago dituela, $s_i \to -s_i$ aldaketa eginez. Bestalde, x zenbaki aleatorio bat hartuko dugu eta $x < \exp(-\Delta E/k_b T)$ bada baita onartuko dugu biraketa. Hau ez badugu egiten gure sistema energia nuluko egoerara joango da zuzenean.

4. Azkenik 2. eta 3. pasusoak errepikatu behar ditugu oreka lortu arte.

Konturatu $\epsilon = 1$ egin dezakegula informazioa galdu gabe. Behin *Metropolis* algoritmoa deskribatu dugula funtzio bat egingo dugu hau egiten duena:

```
def metropolis(N, spin,kT,E,M):
    #Metropolis algoritmoa:
    i, flip=rnd.randint(0,N-1),0
    #%N=12. ekuazioa muga baldintza periodikoak hartuz:
    dE=2*spin[i]*(spin[i-1]+spin[(i+1)%N])
    if (dE<0.0):
        #biraketa onartu:
        flip=1
    else:
        p=np.exp(-dE/kT)
        if (rnd.random()<p):
            flip=1
    if (flip==1):
        #Energia berria, biraketa onartu eta gero:
        E=E+dE
        #Magnetizazio berria:
        M=M-2*spin[i]
        spin[i]=-spin[i]
    return E,M
```

5. **Programa**. Metropolis algoritmoa 1 dimentsioko espinen katean.

Honekin gure sistemaren evoluzioa aztertu dezakegu iterakzio bakoitzean edo orekaren inguruan ditugun oszilazioak. *Monte Carlo Metropolis* simulazioan oso erraz kalkulatu ditzakegu batezbestekoak, adibidez energiaren eta magnetizazioaren batezbestekoak honela kalkulatzen dira:

$$\langle E \rangle = \frac{1}{N} \sum_{n=1}^{N} E(n) \tag{14}$$

$$\langle M \rangle = \frac{1}{N} \sum_{n=1}^{N} M(n) \tag{15}$$

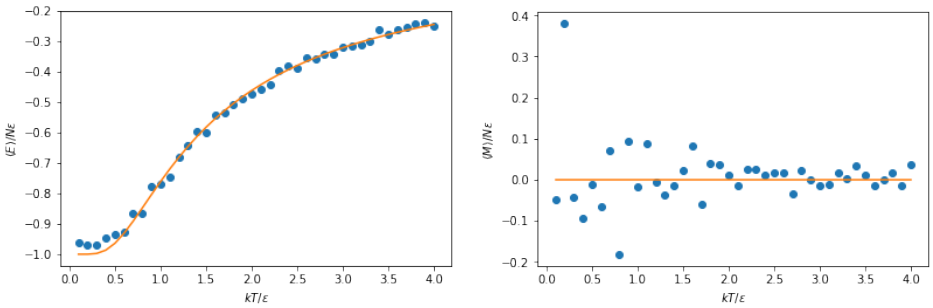
Non $E(n)$ eta $M(n)$ n. mikroegoeraren energia eta magnetizazioa (hau da espin totala) diren. Beras hau ondorengo programan ordezkatuz bi hauen itxura lortu ditzakegu:

```
#Gure spinen katea egiten dugu:
def hasiera(N):
    p,spin,E,M=0.5,[1]*N,0.,0.
    for i in range(1,N):
        if (rnd.random()<p):
            spin[i]=-1
        E=E-spin[i-1]*spin[i]
        M=M+spin[i]
    return spin,E-spin[N-1]*spin[0],M+spin[0]

N,pausuak=1000,10
iterak,Nmc=N*pausuak*10,N*pausuak
T,Ebat,Mbat,z=[],[],[],[]
for i in range(1,41):
    kT=0.1*i
    spin,E,M=hasiera(N)
    #Oreka lortzen dugu tenperatura bakoitzeko
    for k in range(iterak):
        E,M=metropolis(N,spin,kT,E,M)
    E1,M1=0.,0.
    for k in range(Nmc):
        E,M=metropolis(N,spin,kT,E,M)
        E1,M1=E1+E,M1+M
    #Batezbesteko kalkulatzen dugu:
    E1,M1=E1/Nmc,M1/Nmc
    T.append(kT),Ebat.append(E1/N),Mbat.append(M1/N)
    z.append(0.0)

plt.figure()
plt.plot(T,Ebat,'o',T,-np.tanh(1./np.array(T)))
plt.xlabel('$kT/\epsilon$')
plt.ylabel(r'$\langle E \rangle /N\epsilon$')
plt.figure()
plt.plot(T,Mbat,'o',T,z)
plt.xlabel('$kT/\epsilon$')
plt.ylabel(r'$\langle M \rangle /N\epsilon$')
plt.show()
```

6. Programa.



(a) Energiaren batezbestekoa (b) Magnetizazioaren batezbestekoa

4. Irudia. 6. programa erabiliz orekako energia eta magnetizazioaren batezbestekoa tenperaturaren funtzioan kalkulatu ditugu.

Ikusi dezakegunez energiaren batezbestekoa $\langle E\rangle = -1$ da tenperatura zerorantz doaenean eta tenperatura igotzen dugunean $\langle E\rangle \to 0$-rantz doa. Energiaren batezbestekoa analitikoki kalkulatu dezakegu, kontuan izanda gure sistemaren partizio funtzioa ondokoa dela eta muga baldintza periodikoak artuz:

$$Z = (\sum_{s_1 \dots s_N} e^{\beta\epsilon s_i s_{i+1}})^n = [2cosh(\beta\epsilon)]^n \tag{16}$$

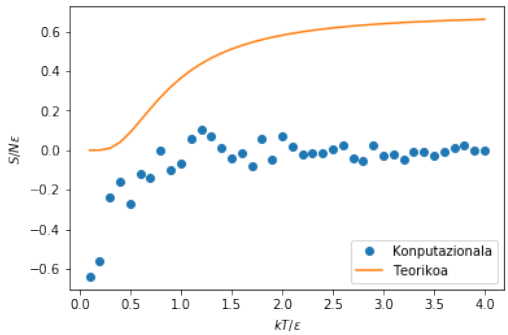
Hemendik energiaren batezbestekoa lortu dezakegu:

$$\langle E\rangle = -\frac{\partial lnZ}{\partial\beta} = -N\epsilon tanh(\beta\epsilon) \tag{17}$$

Hau gure simulazioarekin bat dator. Bestalde teoriak esaten digu magnetizazioa nulua izan behar dela bi arrazoiengatik. Lehenegoa, ez dugu inolako kanpo eremurik beraz spinek ez dute norabide pribilegiaturik. Bigarrena, ia mikroegora guztietan spin bat biratu daiteke energia totala aldatu gabe, ondorioz magnetizazioaren batezbestekoa zeroren inguruan oszilatuko du.
Behin hau dugula aurreko adibidean bezala gure sistemaren entropia kalkulatu dezakegu tenperaturaren funtzioan. Hau egiteko kontuan izan behar dugu $\partial S = \partial E/T$ erlazio termodinamikoa dugula, non ∂S entropiaren aldaketa den ∂E energia xurgatzean. Honekin kalkulatu dezakegu sistemaren entropia T_2 tenperaturan:

$$S_2 \simeq S_1 + \frac{\partial E}{\bar{T}} \tag{18}$$

non $\bar{T} = (T_1 + T_2)/2$ den. Onartuko dugu $S(T = 0) = S_1 = 0$ dela, gogoratu tenperatura nulua denean gure sistema guztiz ordenatuta dagoela. Beraz hau 6. programan inplementatzen badugu ondorengo emaitza lortzen dugu:

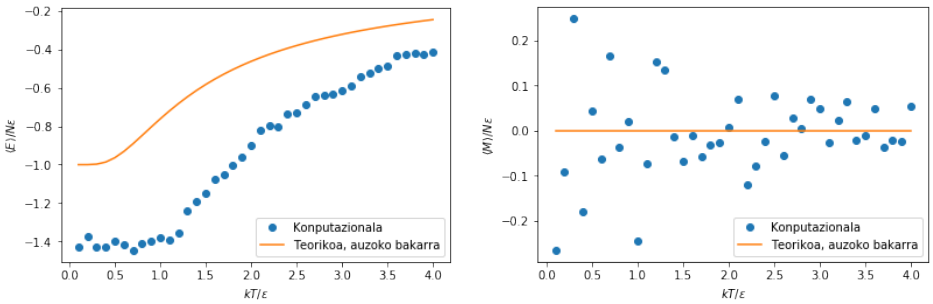


5. Irudia. Dimentsio bakarreko espinen katearen entropia.

Ikusten den modura lortutako emaitza teoriaren itxura dauka baina ez dator guztiz bat, hau gertatzen da entropia kalkulatzeko erabilidugun hurbilketa, hau da, 18. ekuazioa ez delako guztiz zehatza, baina uzten digu entropiaren konportamoldea ulertzea. Gure dimentsio bakarreko espinen katearen analisia bukatzeko kontuan izango dugu espin bakoitza ez duela bakarrik bere lehenengo ondokoarekin energia trukatzen, honen ordez bigarren auzokoak kontuan hartuko ditugu. Hau burutzeko gure *Metropolis* funtzioa aldatu behar dugu, energiaren aldaketa berria honakoa izango da:

$$\Delta E = 2\epsilon s_i(rs_{i-2} + s_{i-1} + s_{i+1} + rs_{i+2}) \tag{19}$$

non r murrizketa faktorea den, hau simulatzen du interakzioak ahulagoak izango direla bigarren auzokideekin. Guk $r = 0.5$ artuko dugu:



(a) Energiaren batezbestekoa (b) Magnetizazioaren batezbestekoa

6. Irudia.Orekako energia eta magnetizazioaren batezbestekoa tenperaturaren funtzioan bigarren auzokiedeen interakzioa kontuan izanda.

Bigarren ondokoen eragina sartuz energiaren itxura ez da aldatzen, bakarrik mugitzen da beherantz eta magnetizazioaren batazbestekoa zerotik aldentzen da. Behin hau dugula eta espinen katearen konportamoldea kuantizatu

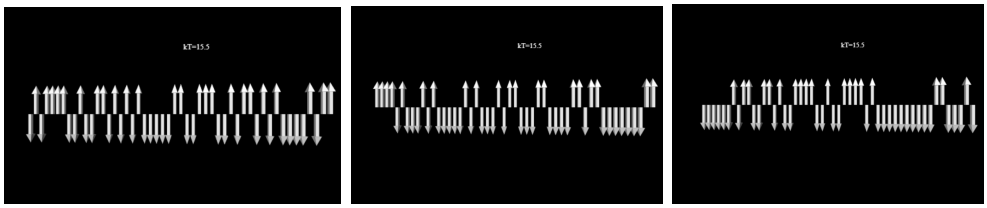
dugula nahi dugun beste aldaketa egin dezakegu, hala nola kanpo eremu bat sartu, auzokide geiago kontuan hartu eta abar.

Gure lanaren helburuak betetzeko egingo dugu hurrengo programa gure katearen eboluzio denporalaren bideo bat lortzeko:

```
import ivisual as vp

N=50
#espinene katea eraiki
spin,E,M=hasiera(N)
L=1.0
#Bideoa hasierazi
scene=vp.display(title='1D ising',background=(.2,.5,1),center
    =(0,0,1.0),forward=(-.4,-.3,-1))
ay=(0,1,0)
x=np.linspace(-5*L,5*L,N)
s=np.arange(N)
#spind:gure espinen katearen marrazkia
spind=[]
for i in s:
    spind.append(vp.arrow(pos=(x[i],0,0),axis=(0,spin[i],0),
        length=1.0,color=(1,1,1)))
vp.label(pos=(0,2.,0),text='kT=15.5',box=True)
#Bideoa mantendu:
while True:
    vp.rate(100)
    kT=15.5
    #eboluzio denporala egin:
    spin=metropolis(N,spin,kT)
    for i in s:
        spind[i].axis=(0,spin[i],0)
```

7. Programa. Ising-en ereduaren eboluzio denporala.



(a) t_1=0.0s (b) t_2=25.0s (c) t_3=50.0s

7. Irudia. Dimentsio bakarreko katea denbora desberdinetan.

3.2 2 Dimentsioko Ising-en eredua

Ikusi dugun modura 1D-ko *Ising*-en ereduak zenbait emaitz termodinamiko interesgarri ematen digu baina ez du balio fase trantsizioak aztertzeko, horretarako 2D-ko *Ising*-en eredua eraiki behar dugu. Eredu honetan spinez osatuatako sare bat, NxN dimentsiokoa, aztertuko dugu. 1D-ko adibidean bezala bakarrik kontuan izango dugu ondoz ondoko espinen arteko interakzioak, baina kasu honetan lau auzoko izango ditugu: goian, behean , ezkerrean eta eskuinean. Beraz mikroegora baten energia honela idatziko dugu:

$$E = -\epsilon \sum_{(i,j)} s_i s_j \tag{20}$$

Non batukaria bakarrik den (i, j) bikote auzokideentzako. Bestalde *Monte Carlo Metropolis* simulazioa erabiltzeko behar dugu espin bat biratzean gertatzen den energiaren aldaketa:

$$\Delta E = 2\epsilon s_{i,j}(s_{i-1,j} + s_{i+1,j} + s_{i,j-1} + s_{i,j+1}) \tag{21}$$

Eredu honetan baita erabiliko ditugu muga baldintza periodikoak:$s_{n+1,j} = s_{1,j}$ eta $s_{i,N+1} = s_{i,1}$. Behin aldaketa hauek azalduta idatzi dezakegu funtzio bat 1D-ko kasuan egin dugun modu berean:

```
def metropolis2d(N,spin,kT,E,M):
    i,j,flip=rnd.randint(0,N-1),rnd.randint(0,N-1),0
    #Energiaren aldaketa berria:
    dE=2*spin[i][j]*(spin[i-1][j]+spin[(i+1)%N][j]+spin[i][j
       -1]+spin[i][(j+1)%N])
    if (dE<0.0):
        flip=1
    else:
        p=np.exp(-dE/kT)
        if (rnd.random()<p):
            flip=1
    if (flip==1):
        E=E+dE
        M=M-2*spin[i][j]
        spin[i][j]=-spin[i][j]
    return E,M,spin
```

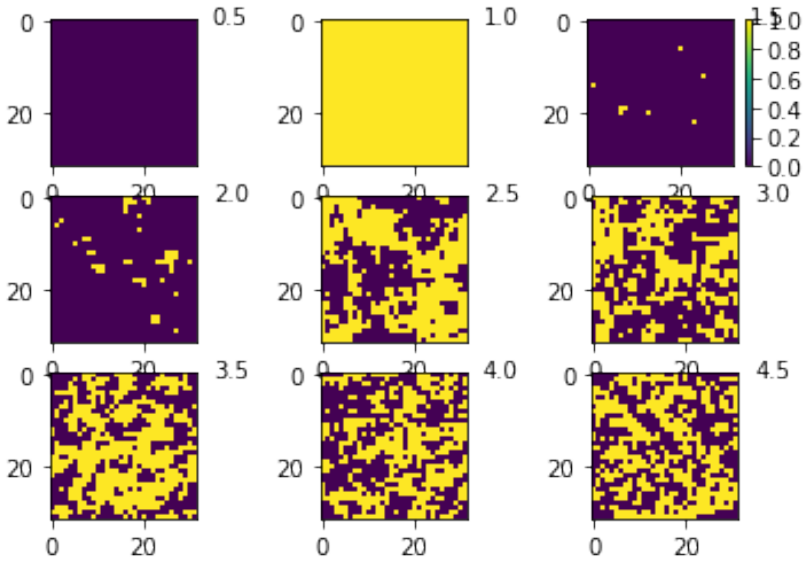
7. Programa. 2 dimentsioko *Metropolis* algoritmoa espinen sarearako.

Behin hau dugula idatzi dugun funtzioa hurrengo programan inplementatuko dugu $32x32$-ko espinen sarearen oreka konfigurazioak aztertzeko tenperatura desberdinetarako:

```
def hasiera(N):
    #Gure spinen sarea eraiki:
    p,E,M=0.5,-0.6,0.
    spin=np.zeros((N,N))
    for i in range(0,N):
        for j in range(0,N):
            spin[i][j]=1
            if (rnd.random()<p):
                spin[i][j]=-1
            E=E-(spin[i-1][j]+spin[i][j-1])*spin[i][j]
            M=M+spin[i][j]
    return spin,E,M

N=32
kT=0.0
iterazio=100000*N
fig=plt.figure()
for j in range(9):
    kT=kT+0.5
    spin,E,M=hasiera(N)
    #Tenperatura honetako oreka lortu:
    for i in range(iterazio):
        E,M,spin=metropolis2d(N,spin,kT,E,M)
    if (j==3):
        plt.colorbar(plot)
    #irudia eraiki:
    ax=fig.add_subplot(3,3,j+1)
    img=np.reshape(spin,(N,N))
    plot=plt.imshow(spin,interpolation='none',vmin=0,vmax=1)
    plt.text(35,0.5,repr(0.5+0.5*j))
plt.show()
```

8. Programa.



5. Irudia. 7. programa erabiliz 2 dimentsioko Ising-en sarearen oreka tenperatura desberdinetarako.

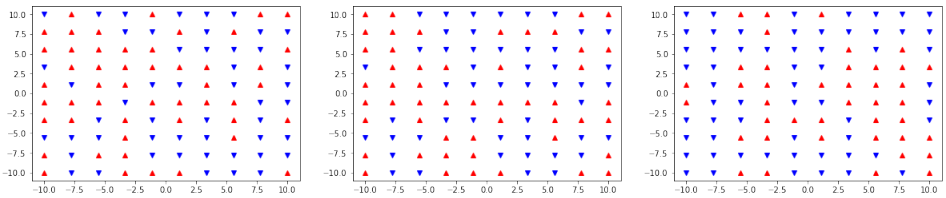
Ikusi dezakegunez tenperatura txikia denean espin guztiak lerrokatuta daude eta tenperatura igotzean nahasten dira. Baina $kT \sim 2.5$ inguruan aldaketazakar bat ikusten dugu, hau esaten digu tenperatura horren inguruan fase trantsizioa gertatzen dela. Fenomeno honen inguruko propietateak aztertu baino lehen 2 dimentsioko eredu hau era interaktiboan programatuko dugu.

Programa interatibo hau A eranskinean dago, bestalde ezin dugunez gure emaitza interatiboa hemen jarri, eranskinetan dagoen estekan lan honen programa eta emaitza guztiak egongo dira era interaktiboan. Baina hurrengo programa erabiliz lortu ditzakegu denbora desberdinetan gure bi dimentsioko sistemaren eboluzioa:

```
N=10
kT=2.5
t=0.0
iterazio=100
fig=plt.figure()
spin,E,M=hasiera(N)
L=2.0
x=np.linspace(-5*L,5*L,N)
y=np.linspace(-5*L,5*L,N)
fig=plt.figure()
#Hasierako egoera irudikatu:
for i in range(N):
```

```
    for j in range(N):
        if (spin[i][j]==1.0):
            plt.plot(x[i],y[j],marker='^',color='r')
        else:
            plt.plot(x[i],y[j],marker='v',color='b')
plt.figure()
#t aldiunea irudikatu:
for k in range(iterazio):
    t=t+0.5
    E,M,spin=metropolis2d(N,spin,kT,E,M)
    if (t==25.0):
        for i in range(N):
            for j in range(N):
                if (spin[i][j]==1.0):
                    plt.plot(x[i],y[j],marker='^',color='r')
                else:
                    plt.plot(x[i],y[j],marker='v',color='b')
plt.show()
```

9. Programa. Bi dimentsioko sistemaren eboluzioa.

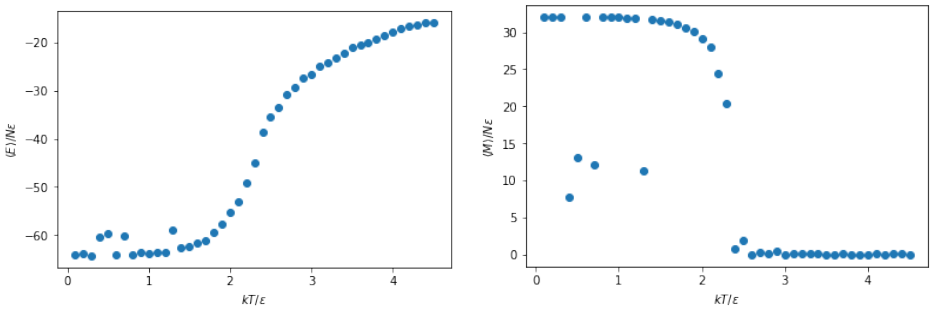


(a) t_1=0.0s (b) t_2=25.0s (c) t_3=50.0s

7. Irudia. Bi dimentsioko sistema denbora desberdinetan.

3.2.1 Fase trantsizioa

Fenomeno hau ikertzeko orekako energia eta magnetizazioaren batesbeztekoa kalkulatuko dugu tenperaturaren funtzioan, horretarako bakarrik moldatu behar dugu 6. programa gure 2 dimentsioko *Metropolis* funtzioa, hona hemen lortutako emaitza:



(a) Energiaren batezbestekoa (b) Magnetizazioaren batezbestekoa

6. Irudia. Orekako energiaren eta magnetizazioaren batezbestekoak tenperatura desberdinetan.

Ikusten dugun modura energiak 1D-ko ereduaren konportamolde berdina dauka, baina magnetizazioa aldaketa bortitz bat dauka $T_k \sim 2.2$ denean, honek esaten digu fase trantsizioa dugula. Hau da, spin guztiak zorizko norabidea daukate tenperatura kritikotik gora eta hortik behera guztiz ordenatuta daude. Gainera ikusi dezakegu ferromagnetikoen beste propietate bat, tenperatura kritikoa gainditu ondoren norabide berdina duten spinak elkarrekin geratzen dira domeinuak eratuz eta tenperatura igotzean multzo hauen tamaina txikituz doa.

Orain konparatu ditzakegu 1 eta 2 dimentsioko ereduak eta ulertu zergatik $1D$-ko ereduan ez dugun ikusi fase trantsizioa. Dimentsio bakarreko ereduan spinen kate osoari buelta eman ahal diogu energiarik eman gabe, tenperatura nulua ez den bitartean.Baina bi dimentsioko ereduan ezin dugu hau egin sistemari energia eman gabe. Gure sistemak energia behar du domeinuen arteko indarra gainditzeko. Kontsidera dezagun zutabe bat, hau $1D$-ko ereduko katea da, baina honi ezin diogu buelta eman alboko zutabeen arteko elkarrenkintza dugulako.

4 Molekulen dinamika

Atal honetan aztertuko dugu molekulen dinamika N gorputzen simulazio klasikoa eginez. Atomoen masa elektroien masa bainoz askoz handiagoa denez *Newton*-en legeak erabili ditzakegu beraien higidura ikertzeko tenperatura baxuetan. Gure helburua gasen zinetika eta propietate termikoak ikertzea da, horretarako atomo bakoitzaren higidura klasikoki aztertuko dugu eta gero

lortuko ditugu propietate termikoak estatistikoki.

4.1 Iterazio potentzialak

Molekulen dinamika N-gorputzen problema bat bezala aztertu dezakegu, non higidura ekuazioak *Newton*-en ekuazioak diren, hau da:

$$\frac{d\vec{r_i}}{dt} = \vec{v_i} \tag{22}$$

$$\frac{d\vec{v_i}}{dt} = \frac{1}{m_i}\sum_{j\neq i}\vec{f_{ij}} \tag{23}$$

Non $\vec{f_{ij}}$ i eta j molekulen arteko indarra den. Normalean hartzen ditugu partikula identikoz osatutako sistemak, beraz beraien masa $m_i = m$ berdina izango da. Indar grabitazionala ez bezala naturan dauden potentzialak ezin dira idatzi zehazki, beraz normalean erabiltzen dira potentzial hurbilduak, ala nola $Morse$ potentzaiala edo $Lennard-Jones$ potentziala. Guk gehien erabiliko dugun potentziala $Lennard-Jones$ potentziala izango da:

$$V(r) = V_0[(\frac{r_0}{r})^{12} - 2(\frac{r_0}{r})^6] \tag{24}$$

Non V_0 konstante positiboa eta r_0 oreka distantzia diren. Lehenengo $1/r^{12}$ terminoa elektroien arteko alderatze indarra deskribatzen du (*Coulom*-en indarra eta *Pauli*-ren kanporatze printzipioa), bigarren terminoa $1/r^6$ distantzia handiko dipolo-dipolo elkartze indarra deskribatzen d (*van der Waals* indarra). Hau edukita i. atomoak j. atomoan eragindako indarra kalkulatu dezakegu gradientea kalkulatuz:

$$\vec{f_{ij}} = -\Delta\vec{V} = 12\frac{V_0}{r_0}[(\frac{r_0}{r_{ij}})^6 - 1](\frac{r_0}{r_{ij}})^8\frac{\vec{r_{ij}}}{r_0} \tag{25}$$

$\vec{r_{ij}} = \vec{r_i} - \vec{r_j}$ i eta j atomoen arteko posizio erlatiboa eta $r_{ij} = |\vec{r_i} - \vec{r_j}|$ bekoteraren modulua dira.

4.2 Muga baldintza periodikoak

Molekulen dinamikaren simulazioa egin aurretik muga baldintza egokiak aukeratu behar ditugu. Aurreko ataletan bezala muga baldintza periodikoak aukeratu ditzakegu, gure sistemaren kopia birtualak eratuz honen inguruan. Modu honetan edozein partikularentzako edukiko dugu beste bat posizio erlatibo berekin eta abiadura berdinarekin. Aukeraketa hau egiten badugu,

partikula bat gure sistematik irtetzen bada beste bat sartuko da. Baina honekin arazo bat daukagu, partikula bat gure sistematik ateratzen denean eta beste bat sartzean sisteman dauden beste partikulek pairatzen duten energia potentziala aldatzen da asko,$r \to r'$ aldaketa gertatzen baita $r' < r$ izanik. Honek sortzen du jauzi bat potentzialean eta indarrean. Honek eragingo duena da gure sistema hustuko dela.

Baina, partikula batetik bestera dagoen distatzia r bada eta bere partikula birtualara dagoen distantzia r' bada, beti aukaretzen badugu bi hauen arteko txikiena, indarran dagoen etena ekidituko dugu. Hau da, beti aukeratu behar dugu distantziarik txikiena $x - x_r$ eta $x - x_{r'}$ artean. Fisikoki esan nahi duena da, partikula batek beti aukeratzen duela bere posiziotik hurbilen dagoena, erreala edo kopia birtuala izan. Honi ondoko-hurbilen elkarrekintza deituko diogu.

Hau eginda erraz ikusi dezakegu bi partikulen arteko distantziarik handiena sistemaren luzeraren erdia izango dela. Beraz L sistemaren luzera(x ardatzean) izanda eta $\Delta = x_1 - x_2$ izanda, $|\Delta x| \leq 1/2L$ bada distantzia hau onartuko dugu, baina ez bada betetzen ondorengo aldaketa egin behar dugu:

$$\Delta x \to \begin{cases} \Delta x - L & , \quad \Delta x > L/2 \\ \\ \Delta x + L & , \quad \Delta x < L/2 \end{cases} \tag{26}$$

Berdina egingo dugu y eta z norabideekin. Hauek izango dira gure muga baldintza periodikoak.

4.3 Simulazioa

Behin muga baldintzak eta partikulen higidura ekuazioak ditugula gure sistemaren dinamika simulatzen duen programa idatzi dezakegu:

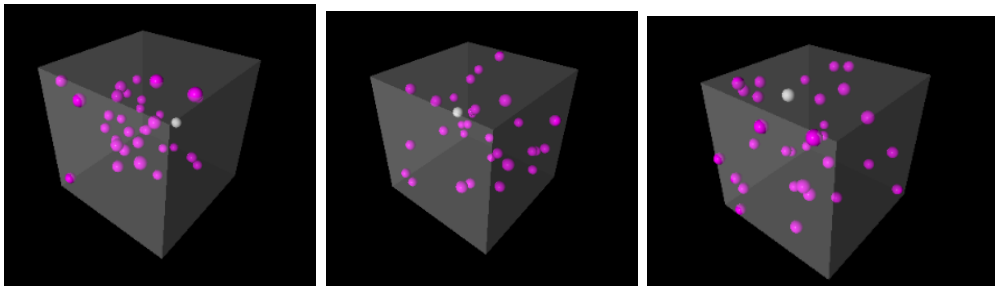
```
import ode, vpm
import random as rnd
import numpy as np,ivisual as vp
#n. partikularen higidura ekuazioa:
def ngorputz(id,r,v,t):
    if (id==0):
        return v
    a=np.zeros((N,3))
    for i in range(N):
        rij=r[i]-r[i+1:]
        rij[rij>HL]-=L
```

```
        rij[rij<-HL]+=L
        r2=np.sum(rij*rij,axis=1)
        r6=r2*r2*r2
        for k in [0,1,2]:
            fij=12.*(1.-r6)*rij[:,k]/(r6*r6*r2)
            a[i,k]+=np.sum(fij)
            a[i+1:,k]
    return a

L,N=10.0,32
atoms,HL,t,h=[],L/2.,0.,0.002
r,v=np.zeros((N,3)),np.zeros((N,3))
#Gure irudia eraikitzen dugu:
scene=vp.display(background=(.2,.5,1),center=(L/2,L/3,L/2))
vp.box(pos=(HL,HL,HL),length=L,height=L,width=L,opacity=0.3)
for i in range(N):
    for k in range(3):
        r[i,k]=L*rnd.random()
        v[i,k]=1-2*rnd.random()
    if (i==3):
        atoms.append(vp.sphere(pos=r[i],radius=0.04*L,color
            =(1,1,1)))
    else:
        atoms.append(vp.sphere(pos=r[i],radius=0.04*L,color
            =(1,0,1)))

v-=np.sum(v,axis=0)/N
#Gure sistemaren eboluzio denborala adierazi:
while True:
    vp.rate(1000)
    #higidura ekuazioa ebatzi:
    r,v=ode.leapfrog(ngorputz,r,v,t,h)
    r[r>L]-=L
    r[r<0.]+=L
    for i in range(N):
        atoms[i].pos=r[i]
```

n. Programa. Sistemaren eboluzioa ematen digu.



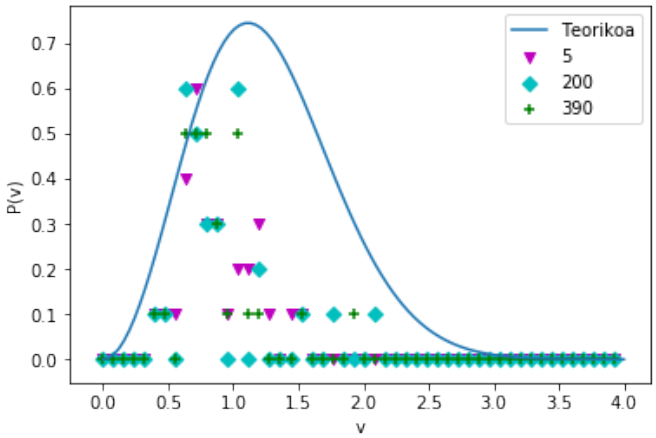
(a) t_1 (b) t_2 (c) t_3

n. Irudia. Gure sistema denbora desberdinetan.

Behin hau dugula, programa hau aldatu dezakegu gure partikulen sistemaren abiaduren diagrama irudikatzeko. Hau egitean ikusten dugu nola *Boltzman*-en distribuzioarekin bat datorren, teoriak aurresaten duen modura. Bestalde hemendik atera dezakegu sistemaren tenperatura, gogoratuz honelako sistemetan hurrengoa dugula:

$$\frac{1}{2}m\langle v^2\rangle = \frac{3}{2}kT \tag{27}$$

hau egiten badugu ondorengo emaitza lortzen dugu:



n+1. Irudia. Abiaduren diagrama aldiune desberdinetarako.

Hemendik lehen esan dugun modura gure sistemaren tenperatura lortu dezakegu $\langle v^2\rangle = np.mean()$ funtzioa erabiliz eta $m = 1.0$ hartuz. Aukeraketa honekin hau dugu:

$$\langle v^2\rangle \simeq 103.41 m^2/s^2 \Rightarrow kT \simeq 0.62J \tag{28}$$

5 Ondorioak

Lan honekin zenbait sistema fisikoak aztertu ditugu termodinamikaren ikuspuntutik beraien propietateak lortuz. Bestalde tresna konputazionalak eraiki ditugu bestelako teoriak eta proiektuak adierazteko, baita irakaskuntzan erabili ahal izateko.

Gainera lan hau egiteko *Jupyter Notebook*-a erabili egin dugu, tresna honekin erabili dugun teoria idatzi dezakegu eta aldi berean egindako programak egin, konpilatu eta emaitzak ikusi ditzakegu. Hau oso interesgarria izan daiteke irakaskuntzan, adibidez teoria *pdf* formatuan eman beharrean *Jupyter Notebook* batean eman daiteke eta hor ikaslea apunteak geitu ditzake edo teoriako programekin ikasi.

A Eranskinak

A.1 Programa interaktiboa

```
import random as rnd
import ivisual as vp
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import *
from ipywidgets import interact, interactive, fixed,
   interact_manual
from ipywidgets import *
import ipywidgets as widgets
from IPython.display import *

get_ipython().magic('matplotlib nbagg')

def update(t1):
    def hasiera(N):                                          #
        Gure spinen sarea eraiki
        p,E,M=0.5,-0.6,0.
        spin=np.zeros((N,N))
        for i in range(0,N):
            for j in range(0,N):
                spin[i][j]=1
                if (rnd.random()<p):
                    spin[i][j]=-1
                E=E-(spin[i-1][j]+spin[i][j-1])*spin[i][j]
                M=M+spin[i][j]
        return spin,E,M

    def metropolis2d(N,spin,kT,E,M):
        i,j,flip=rnd.randint(0,N-1),rnd.randint(0,N-1),0
        dE=2*spin[i][j]*(spin[i-1][j]+spin[(i+1)%N][j]+spin[i
           ][j-1]+spin[i][(j+1)%N])
        if (dE<0.0):
            flip=1
        else:
            p=np.exp(-dE/kT)
            if (rnd.random()<p):
                flip=1
        if (flip==1):
            E=E+dE
            M=M-2*spin[i][j]
            spin[i][j]=-spin[i][j]
        return E,M,spin
    N=10
    kT=2.5
```

```
    t=0
    iterazio=100
    fig=plt.figure()
    spin,E,M=hasiera(N)
    L=2.0
    x=np.linspace(-5*L,5*L,N)
    y=np.linspace(-5*L,5*L,N)

    for k in range(iterazio):
        t=t+1
        E,M,spin=metropolis2d(N,spin,kT,E,M)
        if (t==t1):
            for i in range(N):
                for j in range(N):
                    if (spin[i][j]==1.0):
                        plt.plot(x[i],y[j],marker='^',color='
                            m')
                    else:
                        plt.plot(x[i],y[j],marker='v',color='
                            c')
    print(t)

def play():

    t1=widgets.IntSlider(min=1,max=100,value=50,description="
        espinen kopurua")

    play_button= widgets.Play(value=50,min=0,max=100,step=1,
        description="Press play",disabled=False)
    q=widgets.interactive(update,t1=t1)
    display(q)

play()

t1=widgets.IntSlider(min=1,max=100,value=50,description="
    denbora")
play_button= widgets.Play(value=1,min=0,max=100,step=1,
    description="Press play",disabled=False)
jslink((play_button,'value'),(t1,'value'))
VBox([HBox([play_button,t1])])
```

A.2 *Github*-eko esteka

Bibliografia

[1] Neil W. Ashcroft & N. David Mermin, *Solid State Physics* (1976)