Monte Carlo - Ackley Function - Übung für SuS – May 2024 (part 2)

This is the second part of the Übung with Monte Carlo and Ackley's function. The aim is to get the software working. A week later, the aim will be to use it to look at sampling from a distribution. You need a working implementation of Ackley's function from the previous Übung.

There are two aims

- Write a program which implements Metropolis Monte Carlo simulation using Ackley's function as the cost (like energy) – this week
- Run some calculations to see that the program works and find a non-intuitive result (next week)

1. Introduction

We had the description of Ackley's function in the last Übung. Now we pretend that Ackley's function is our energy. If we talk about energy E as a function of a vector of x's, we write $E(\mathbf{x})$. We will treat E as a potential energy. We use $\mathbf{x}$ to mean a vector of $x_1, x_2, .. x_n$ for n dimensions.

We do not have absolute probabilities as in the previous Übung. This would be $p_i = \frac{\exp\left(\frac{-E(\mathbf{x}_i)}{kT}\right)}{Z}$.
We can only calculate relative probabilities, which look like

$$\frac{p_i}{p_j} = \frac{\exp\left(\frac{-E(\mathbf{x}_i)}{kT}\right)}{\exp\left(\frac{-E(\mathbf{x}_j)}{kT}\right)} = \exp\left(\frac{-\Delta E_{ij}}{kT}\right) \quad (1)$$

where ΔE_{ij} is the energy difference between states i and j.

We do not have real energies, so our temperature units are arbitrary. We set $k = 1$ and the temperatures we work with will be in a range from 0.01 to about 1 or 2.

2. Programming

Programming Monte Carlo is easy, but you will find you need the same amount of code to control the program. You can write in any language that you think I can read (C, go, python, R, perl…)

2.1. Getting parameters into the program

You need a temperature T. Later, the program will be extended to have an initial temperature and final temperature, so let us call the temperature `ini_tmp`. Let us name all the parameters

name in parameter file		type
`ini_tmp`	temperature	float
`final_tmp`	final temperature (not used this week)	float
`n_step`	number of steps	integer
`x_ini`	initial coordinates	set of floats, separated by commas
`x_delta`	maximum step size	float
`seed`	random number seed	integer
`foutname`	name of an output file	string

You could squash this into command-line arguments, but it would be inconvenient and error prone.

Write a function that reads a file that looks like this and assigns the value to a variable in your program. Note that there are different data types, so you will need different functions for the conversions.

```
ini_temp 0.1
final_temp 0.1
n_step 100000
x_ini -1.5,0.5,1
x_delta 0.5
seed 1637
foutname ex1.csv
```

We will use `x_ini` to determine the number of dimensions. If you have `x_ini 1`, it means run in one dimension with an initial $x = 1$. If you have `x_ini 2,2,2,2,2` it will mean you want five dimensions and the initial coordinates are $x_1 = 2, x_2 = 2, \ldots$. Note that this means you see the word x_ini, you have to break the next word into a list/array of numbers.

This will take longer to program than you think.

2.2. Core Monte Carlo

This is simple and corresponds to the recipe in the lectures.

Start by initialising your random number generator with the seed you choose in your input. You do not want to use the time of day or `/dev/random`. To debug and test, you want reproducibility. Initialise the code and then run the Monte Carlo loop. The code should look like this, depending on your language and style

```
Initialise
  read the parameter file and assign variables
  decide on how many dimensions you have
  set the random number seed
  open any files you need for output
  set x ≔ ini_x
  store the initial energy E for this x
function get_trial_x (x: set of floats, n: number of dimensions) {
  idim ≔ random(0, number_of_dimensions)
  x_trial ≔ x   // copy the old set of x
  step ≔  x_delta * (2*random(0,1) – 1) // scale the random number by the max step size
  x_trial[idim] ≔ x_trial[idim] + step
}
monte_carlo() {
  FOR num_step
    x_trial ≔ get_trial(x, n)
    E_trial ≔ energy(x_trial) // from Ackley function
    flag := False
    IF (E_trial ≤ E)
      flag ≔ True
    ELSE
      IF E_trial > E
        deltaE ≔ E – E_trial
        IF random(0,1) < exp(deltaE/Temperature)
          flag ≔ True
    IF (flag == True)
      print out energies and x
      update x, update E
```

2.3. output

You have a choice. You can either

a. write a .csv file or

b. build plotting into your program

You want to be able to

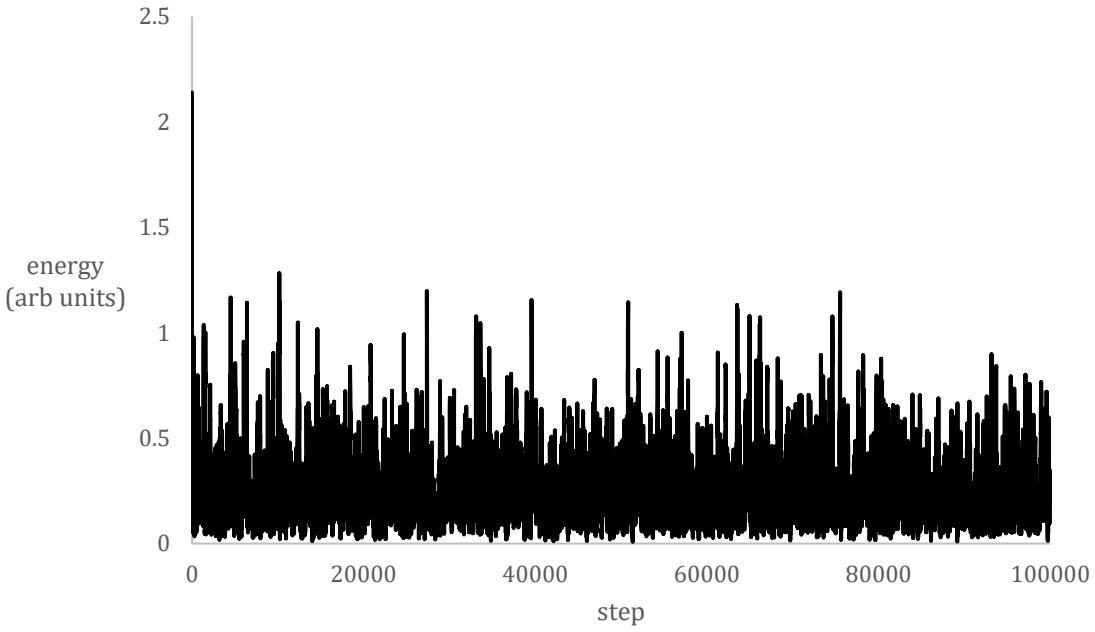
a. plot the energy (function value from Ackley's function)

b. plot the actual x-coordinates

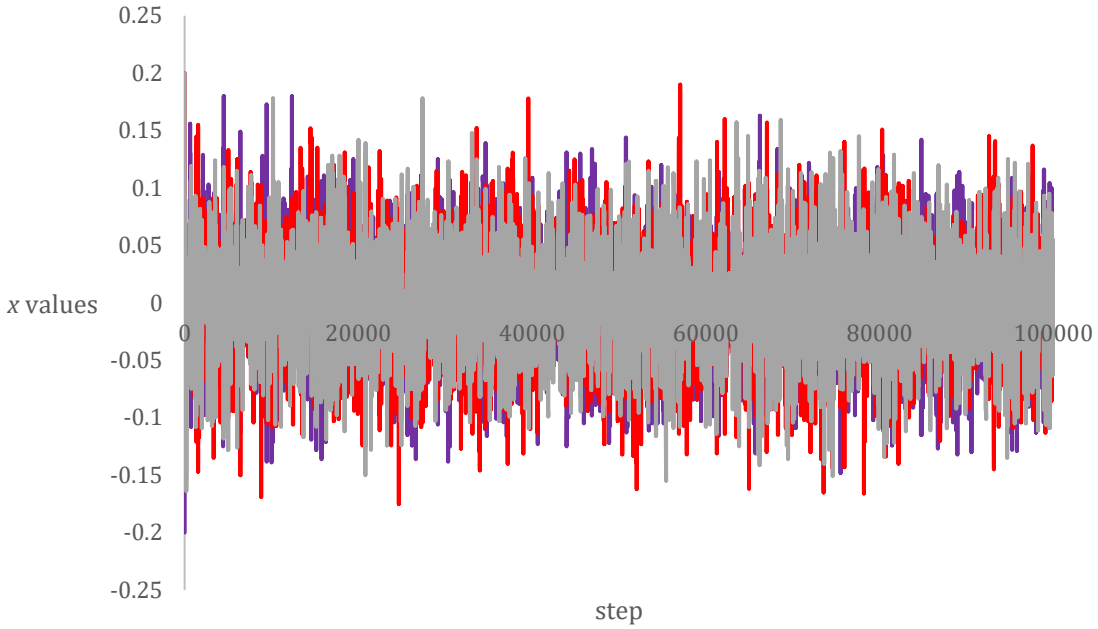
If I start a run with

`ini_tmp`	0.1
`n_step`	100000
`x_ini`	-0.2,0.2,0.2
`x_delta`	0.5

I get an energy plot like this



and the x values in the three dimensions follow this path:



3. To Do

- Write a program to run the calculations. The command line should look like `ackley_mc input_file` or maybe `ackley_mc [options] input_file` or maybe even, `python ackley_mc.py input_file`
- Do not hesitate to play with one of the binaries for linux or windows that are linked to from moodle. The graphical user interface lets you very quickly have a look at behaviour of the system.
- Play with your program to convince yourself that it works.
    - Start with one or two dimensions
    - If you have a temperature $T = 0.1$ and your initial x values are at zero, they should quickly reach equilibrium and move in a range from ~-1.5 to 1.5
    - If you have a temperature $T = 0.1$ and your initial x values are something like 3,3, they should quickly reach equilibrium and move in a range from ~-1.5 to 1.5
    - Check that the code works in ten dimensions and the plots look reasonable
- Do you know how to measure the runtime of a program ? If your command line looks like
  `python yourprog.py input_file`, you could try
  `/usr/bin/time python yourprog.py input_file` or under windows, it might look like `measure-command { python yourprog.py input_file }`
- Go to moodle and answer a few specific questions.

Do not worry if you have not learnt much about Monte Carlo this week. That is the idea of the next part of the Übung.