ERHVERVSAKADEMI AARHUS

Opgave om Genetiske Algoritmer.

Sila.

Exercise – Cannonballs & Genetiske Algoritmer.

In groups of 2- 3.

Denne opgave handler om ved hjælp af Genetiske Algoritmer at finde en (flere) løsninger (hastighed, vinkel) hvormed kanonkugler sendes afsted så de kommer ud af en ”boks”.

Givet eksempel kode Cannonball_GA og Cannonball_Test_GA (for tests) (Canvas) skal man først arbejde med ændringer på kassen/boksen størrelse, populations størrelse antal iterationer man kører udregningen.

Det er opgave 2.a. – c.

Derefter:

Koden er visse steder lidt svært at forstå, selvom den jo sådan set gør som den skal. Så det handler derefter om at tage lave ændringer i koden, så der fremkommer en tydligere fitness funktion, og en tydelige selektionsmekanisme og mutations mekanisme. Kort sagt at der arbejdes med at gøre koden nemmere at forstå i forhold til begreberne forklaret i dagens lektion om genetiske algoritmer.

Endelig ønskes en ændring til fitness funktionen, så kuglerne i den endelige løsninger har en større tendens til at flyve mod højre.

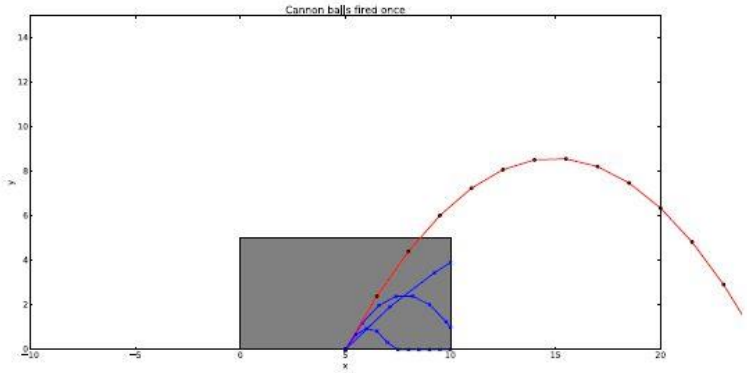
Det er opgave 2.d. – e.

Your Mission: Fire Cannonballs

Let's create a genetic algorithm for firing virtual cannonballs out of a paper bag. Let's see how cannonballs move when fired, and start thinking about which paths are better. This will tell us the criteria for the GA. There are two ways these cannonballs can move: straight up or at an angle.

When fired at an angle, cannonballs travel up, either left or right, eventually slowing down due to gravity, following a parabola. When fired straight up at 90 degrees, a similar thing happens. However, instead of a parabola, they come straight down. Cannonballs fired fast enough go into orbit, or reach escape velocity, which is certainly out of the paper bag, but hard to draw.

The trajectories of a few cannonballs are shown in the next figure. They start from an invisible cannon located at the bottom/middle of a gray bag. One cannonball travels up a little, then falls to the bottom of the bag and rolls along. Another two go up and stick to the edge of the bag. Finally, one manages to escape the bag by going high enough and fast enough:



1. Opgave formulering:

Opgave formuleringen er givet i teksten GA_ML_Ch3_Cannonballs (pdf), se Canvas.

Nedenfor følger et uddrag. Spørgsmålene til opgaven følger under punkt 2.

The higher up a cannonball gets at the edge of the bag, the better the angle, velocity pair. Any cannonball over the bag height at the edge escapes. You can use this height as the criteria, or fitness function. Let's see what equations to use to map these trajectories.

Encoding.

The first step in using a GA is encoding your problem. The cannonballs need pairs of numbers—angles and velocities—for their trajectories. Dividing up investments into bonds, property, foreign exchange, and shares finds four numbers—how much in each. Finding a seating plan can use an array with one element per guest. Other problems find letters or strings of bits. Any fixed-length solution is amenable to GAs.

Iterationer:

The first set of solutions of velocity/angle pairs are created randomly, giving a variety of solutions, some better than others. This is the first *generation*. The GA will use these to make better solutions over time. A GA is a type of guided random search, or *heuristic search*, using these to find improved solutions, using a loop:

```
generation = random_tries()
for a while:
    generation = get_better(generation)
```

Fitness:

To select fitter parents, the GA needs a way to compare solutions.
For the first approach, you can follow the arc of each cannonball and see which ones leave the bag. Does a ball hit an edge, land back inside the bag, or get out? For the second approach, you can do some math and figure out how far up the ball went, and how close to the edge it was.

Random Tries:

To get started, the GA needs a population of candidate solutions. A list fits the bill, so make a list and append pairs of random numbers; `theta` for the angle and `v` for velocity.

Selection Process:

How do you select parent solutions? A genetic algorithm tends to pick better, stronger, fitter solutions for breeding, as happens in Darwinian evolution. This means the fitter solutions can pass on their genes to future generations. In this case, the genes are angles and velocities.

Recall how to find the x-coordinate of a cannonball:

$$x = vt\cos(\theta)$$

Since you know how wide the paper bag is, you can work out when it gets to the edge, and then use the equation for y to see how high up it gets at time t. Suppose the cannon is positioned in the middle of a bag that is 10 units wide, as it was in the arcs considered on page 35. If the bottom left of the bag is (0, 0), then a cannonball at 0 or +10 across is at the edge. Since it started in the middle, it has traveled 0.5 * 10 = 5 units left or right when it gets to the edge. If it goes right, anything under 90 degrees, it has traveled 5 units across when it gets to the edge, so the time it took is

$$t = 5/(v \times \cos(\theta))$$

You know v and theta, so can calculate t. If it goes left, anything over 90 degrees, it travels -5 units horizontally, so you can do a similar sum using -5 instead of 5.

Having found t, find the height it gets to at the edge of the bag using the equation for y:

$$y = vt\sin(\theta) - \frac{1}{2}gt^2$$

You can use this to make a fitness function returning a boolean by checking whether or not the cannonball goes over the bag height. Alternatively, you can use the y value it returns as a fitness. Higher values will be better.

Selection Process:

Armed with a list of potential solutions, and a fitness function, the GA can now select parents. It could take the best two, but to give your GA a chance to explore a variety of possible solutions, you need to be slightly less elitist. You can pick the top few by sorting or *ranking* the population and pick two at random from the top few.

Crossover:

You now have a population and a way to pick better parents. Armed with two parents, breed new solutions in a crossover function. Not all genetic algorithms use two parents; some just use one, and there are others that use more. However, two is a conventional approach:

In order to breed, the information is split—half from one parent and half from another. Since there are two bits of information, each child will have a velocity from one parent and an angle from the other. There is only one way to

split this. In general, when the solution contains more information, the genetic algorithm needs to be a bit smarter

Mutation:

Let's potentially change everything in a population, to keep it general. Suppose you want something to happen on average, one time out of ten. One approach is to get a random number between 0 and 1 and do that "something" if you get less than 0.1. This is called *probabilistic* mutation. If you mutated values every time, this would be *deterministic*. Mutate either (or both) value(s) whenever you draw a random number less than 0.1.

2. Spørgsmål:

a)

I den udleverede kode laves der display af 1 og sidste generation. Lav en kode ændring i programmet så det bliver muligt at følge bedre i progress. Du kan vælge at lave plot for hver generation, eller evt. se på plot mellem 2 efterfølgende generation, for derved at se progress fra generation til generation.

b)

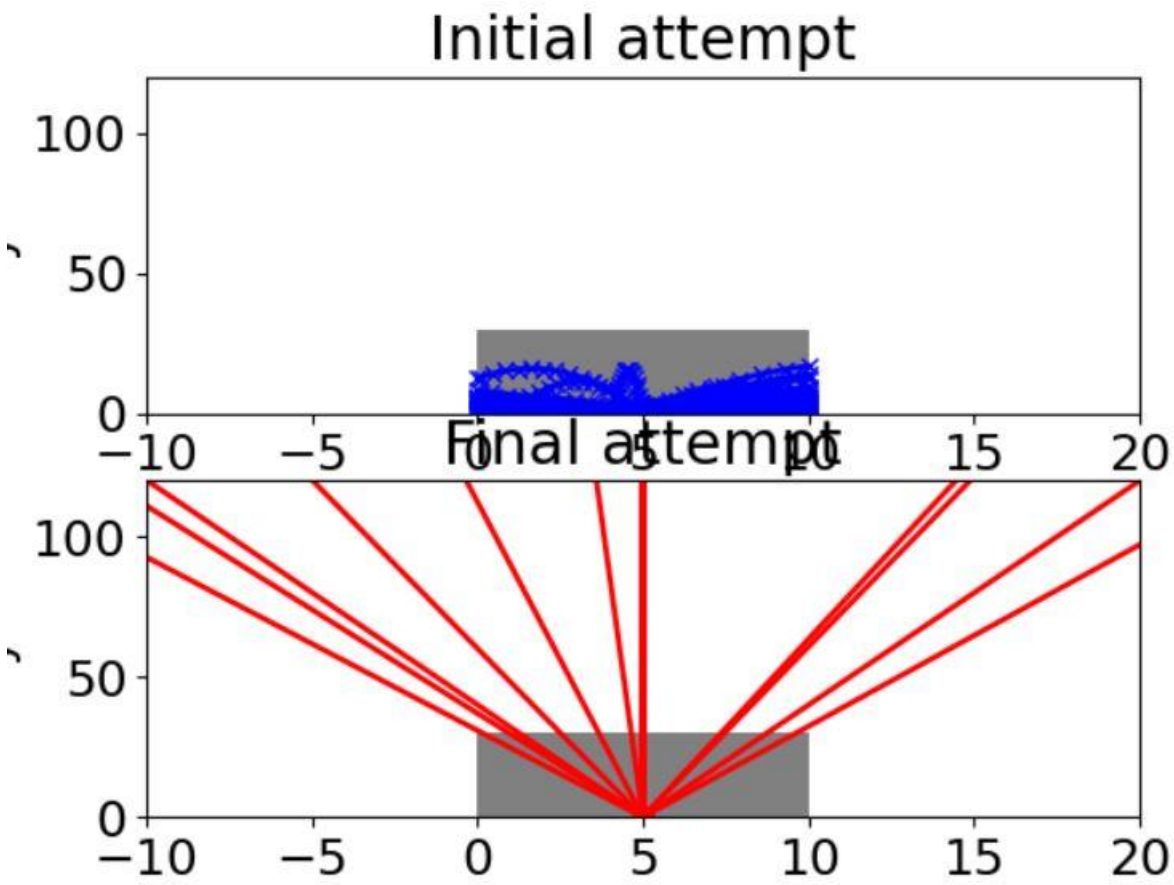
Hvis højden på kassen sættes op (height) til f.eks. 30, hvor mange generationer skal der så køres (epochs) for at alle kanonkugler kommer ud af boksen?

c)

Eksperimenter med løsningens parametre— antal generationer (epoker), antal elementer i en population (items).

```
epochs = ?
items = ?
height = 30
width = 10
```

Hvis height på boksen er 30 hvad er så en god setting (minimal) for items og epochs?



Tilsvarende for height = 10. Hvad er så en god setting (minimal) for items og epochs?

d) Ændringer til programmet – så du laver din egen fitness funktion.

Som sagt er koden korrekt, men ikke helt super let at forstå i forhold til dagens lektion om begreberne der indgår i en genetisk algoritmer.

Derfor:

Lav en ny fitness funktion – dvs. en fitness funktion – hvor koden er nemmere at forstå … (for dig, fordi du selv har designet den …)

I.e. det er forklaret i teksten (GA_ML_Ch3_Cannonballs, se Canvas) hvordan programmet nu udvælger hvilke items der skal være i næste generation v.hj.af fitness funktion.

```
choices = selection(generation, width)
next_generation = []
```

Husk du kan ikke ødelægge noget…
så prøv at lav nogle ændringer til koden som gør koden mere i overenstemmelse med en simpel, mere intuitiv (for dig) forståelse af udvælgelse til ny generation på baggrund fitness funktion.

Dvs. der ønskes nu kodeændringer, så vi får en tydelig funktion kaldet fitness – som giver en scorer til en løsning (hastighed, vinkel – e.g. en liste med hvilket hit koordinat det giver, som derefter sorteres. Hvorfra man kan udvælge sig løsninger blandt de f.eks. 20 % bedste. For yderligere kontrol kan man forestille sig mulighed for at vælge løsninger på højden, hvor der gælder noget om vinlen?). Derefter en (tydeligere) metode som udvælger de bedste løsninger – F.eks. at kunne vælge mom and dad fra de 20 procent løsninger. Og derfra lader nogle af dem gå videre til næste generation, suppleret med nye løsninger (lavet ved hjælp af ”breeding”, mom and dad), samt endelig mutation i nogle af populationens løsninger.

e) ekstra.

Ved hjælp af den nye funktion – gør det så mere attraktivt at sende kanonkuglerne mod højre.