Genetic Bike

Group:
Fábio Köpp Nóbrega, Oscar Augusto de Aguiar Francisco, Cecilia Jarne

Implementation of a genetic algorithm to design a two dimensional bike with two wheels and two mass points

https://users.hepforge.org/~dgrell/ICTP14/
https://bitbucket.org/bicicletagenetica/bicicletagenetica
Statements of the problem

- The bike must have two wheels and two loads.
- The initial positions of these masses and wheels can be freely chosen by the algorithm.
- The elements are connected by springs whose length, with fix damping constant and spring constant.
- The loads must never touch the ground.
- The optimality of a particular candidate solution (the fitness function) is determined by how far it travels before a mass touches the ground or reaches a fix maximum distance.
What we need to model?

- Geometry of the problem (two dimensions+time)
- Bicycle object
- Ground shape
- Physics (spring, masses, wheel, gravity)
- Time evolution (Runge-Kutta)
- Evolution (genetic algorithm to find the best bike)
- Visualization (visual interface)
How do we split the work?

Meeting

- Cecilia
  - Bike creation

- Oscar
  - Bike Component

- Fabio
  - Animation

Starting point
How do we split the work?

Meeting

Cecilia
Bike creation
Bike creation

Oscar
Bike Component

Fabio
Animation
Animation

Starting point
How do we split the work?

Meeting
- Cecilia
  - Bike creation
- Oscar
  - Bike Component
- Fabio
  - Animation

Force (spring interaction)
- Genetic Algorithm
- Runge-Kutta

Starting point
How do we split the work?

Meeting

Cecilia
- Bike creation
- Meeting

Oscar
- Bike Component
- Meeting

Fabio
- Animation
- Meeting

Force (spring interaction)
- Meeting

Genetic Algorithm
- Meeting

Runge-Kutta
- Meeting

MAIN
- Meeting

Starting point
Class Structure Definition and shearing code design
Example: generating the random bikes

- **class Bike:**
  - ```
    class Bike:
    
    # Bike class
    
    def __init__(self, wheels_tmp = [], mass_points_tmp = []):
        self.wheels = wheels_tmp
        self.mass_points = mass_points_tmp
        self.spring = (spring_constant, Damping)
        self.fitness_value = fitness_value_tmp
  ```

- ```
    def random_bike():
        mp1 = MassPoint(x0 = random_position(), y0 = random_position())
        print "First one set"
        mp2 = MassPoint()
        mp2.Position = mp1.Position
        while np.linalg.norm(mp1.Position - mp2.Position) < 0.5:
            mp2 = MassPoint(x0 = random_position(), y0 = random_position())
            print "Second one set"
        w1 = Wheel(x0 = random_position(), y0 = random_position())
        while np.linalg.norm(mp1.Position - w1.Position) < 0.5 or np.linalg.norm(w1.Position - mp2.Position) < 0.75:
            w1 = Wheel(x0 = random_position(), y0 = random_position())
            print "Third one set"
        w2 = Wheel(x0 = random_position(), y0 = random_position())
        while np.linalg.norm(w2.Position - w1.Position) < 1.0 or np.linalg.norm(w2.Position - mp1.Position) < 0.75 or np.linalg.norm(w2.Position - mp2.Position) < 0.75:
            w2 = Wheel(x0 = random_position(), y0 = random_position())
        random_bike = Bike([w1, w2], [mp1, mp2])
        return random_bike
  ```
Example 2: the genetic algorithm

import numpy as np
from operator import itemgetter

import matplotlib.pyplot as plt
from math import sqrt

number_fittest = 20
population_size = 200

def fitness(x):
    return sqrt(x)

def avg_fitness(population):
    fit_sum = 0
    for i in population:
        fit_sum += fitness(i)
    return fit_sum/len(population)

def max_fitness(population):
    map_pop_fitness = []
    for i in population:
        # /
        ....
Documentation Example: using Pydoc at the genetic algorithm code

- **NAME**
  - genetic_simple

- **FUNCTIONS**
  - avg_fitness(population)
    - Average fitness of the population
  - crossover(father, mother)
    - Crossover (Mean)
  - fitness(x)
    - Fitness calculation
  - max_fitness(population)
    - Max fitness of the population

- **DATA**
  - number_fittest = 20
  - population_size = 200
Down the hill
Meeting

Cecilia

Bike creation

Force (spring interaction)

Oscar

Bike Component

Genetic Algorithm

Fabio

Animation

Runge-Kutta / leapfrog

Not yet... (adhoc)

MAIN

Starting point

How far we get

Meeting

Bike creation

OK

OK

OK

OK
To Do

- Improve the ground interaction.
- Implement the genetic algorithm for the bikes
Conclusion

• It was a great opportunity to learn more about numerical evaluation of a problem (Euler, Runge kutta forth order, leapfrog and genetic code.)

• Heterogeneous group
Thank you!
Back Up:

- The Euler problem