Lab 7

Computational Mathematics

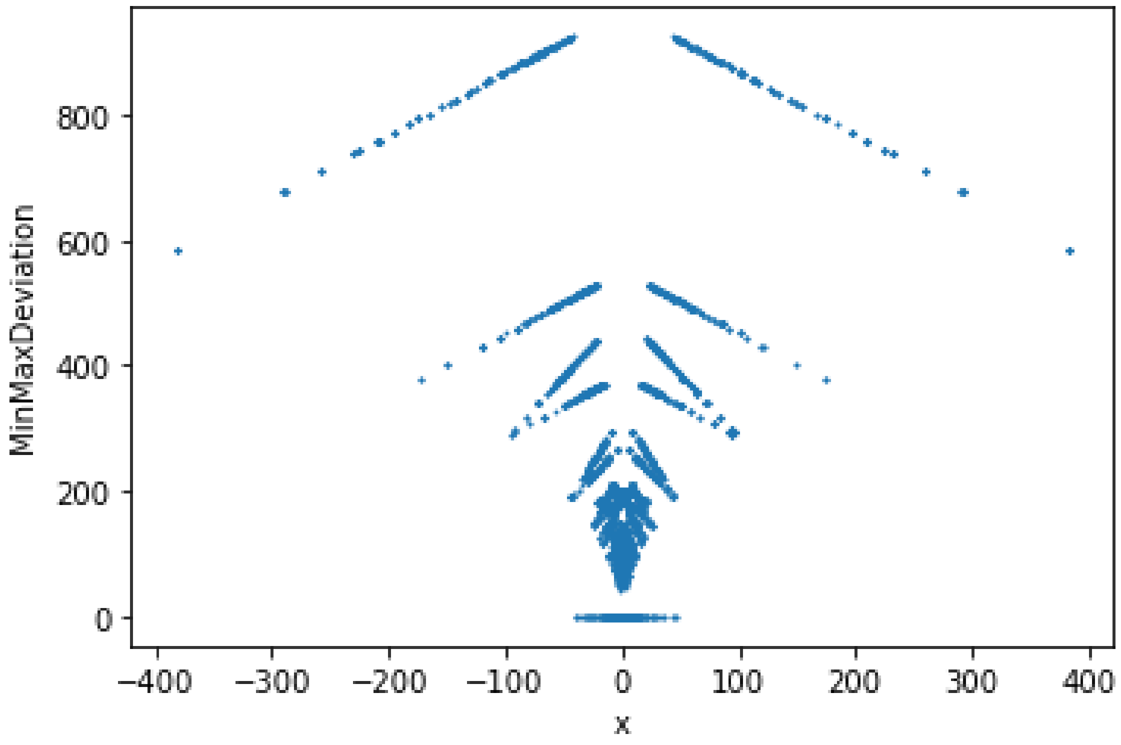
Daniel de las Heras

April 24, 2021

Question 1

In class we read the paper *Optimal tuning of multi rank keyboards*. We were exposed to the basics of music theory and Secor's temperament. We found out that there are difficulties to find an ideal temperament so that given a constant separation of discrete keys, the frequencies of the 11 first harmonics can be played by a precise key.
Hence, we are willing to find the ideal frequency difference of keys such that given a base frequency, x, can approximate the first 11 odd harmonics from keys in the keyboard at the same time as good as possible.
For this problem we are trying to minimize the maximum deviation, hence we will end up with the absolute value of 4 different constraints, corresponding to each of the odd harmonics, of the form $|y| \geq m(x - x_0)$ where $|y|$ is the absolute value of the deviation of that harmonic, m is the steps of the key selected for the harmonic, x is the frequency difference between keys and x_0 is the target frequency or harmonic.
As we are looking at the absolute value of the inequality we must evaluate $|y| \geq m(x - x_0)$ as $y \geq m(x - x_0)$ and $-y \geq m(x - x_0)$.

MinMax Scatter Plot (x, y)



By running the program we will get an (x,y) solution, which depends on the assigned steps to the harmonics. ”x” represent the ideal separation to get the minimax solution ”y” which is the deviation of the obtained frequency from the target harmonic’s frequency. We know that the lower the deviation the more ”Pleasant” the sound will be as it approximates better to every harmonic sound.

```python
import numpy as np
from scipy import optimize
import cvxopt
import random
import matplotlib.pyplot as plt

# Function to calculate the deviation of a harmonic
def harmonic(n, step):

    return ((1200/step) * np.log2(n))


# Generates a random step for each harmonic
def steps():
    fundamental = []
    steps = []
    position = list(range(1,24))

    # Generates a list with the position of the
    # Fundamental harmonic and the other 3 harmonics
    for i in range(5):
        x = random.randrange(0,len(position))
        fundamental.append(float(position[x]))
        position.pop(x)

    # Calculates the difference in position between
    # The harmonics and the fundamental and returns
    # A list with the relative positions of the harmonics
    # With respect to the fundamental
    for i in range(1,5):
        steps.append(float(fundamental[i]-fundamental[0]))
    return steps

# Function that finds the minmax solution
# Output -- An array with the minimum harmonic deviation
# Given the random steps
def findMinMax(harmonic, step):

    # Steps of the constraints


    #harmonics = [5/4, 7/4, 9/8, 11/8]

    # Objective Function
    c = np.array([0.0, 1.0])

    # Constraints
    b = np.array([harmonic(5/4, step[0]),
                   -1*harmonic(5/4, step[0]),
                   harmonic(7/4, step[1]),
                   -1*harmonic(7/4, step[1]),
                   harmonic(9/8, step[2]),
                   -1*harmonic(9/8, step[2]),
                   harmonic(11/8, step[3]),
                   -1*harmonic(11/8, step[3])])
    # Constraints of variables
    A = np.array([[step[0], -1.0],
                  [step[0], -1.0],
                  [step[1], -1.0],
                  [step[1], -1.0],
                  [step[2], -1.0],
                  [step[2], -1.0],
                  [step[3], -1.0],
                  [step[3], -1.0]])

    A_ = cvxopt.matrix(A)
    b_ = cvxopt.matrix(b)
    c_ = cvxopt.matrix(c)

    sol = cvxopt.solvers.lp(c_, A_, b_)
    xy = np.array(sol['x'])

    return xy
#print(steps())

#
x = []
y = []
positions = []

# Runs the function findMinMax(harmonic, steps) 10,000 times
for i in range(10000):
    step = steps()
    sol = findMinMax(harmonic, step)
    # Appends the distance between keys to x
    x.append(sol[0][0])

    # Appends the deviation to y list
    y.append(sol[1][0])

    # Appends the positions to the positions list
    positions.append(step)

print(step)

# Create the vectors X and Y
```

```python
x = np.array(x)
y = np.array(y)


# Create the plot
plt.scatter(x, y, s = 0.5)

plt.xlabel("x")
plt.ylabel(" MinMaxDeviation ")
# Show the plot
plt.show ()
```