Pattern Recognition

Project 1: warm up

Group members:

Alina Arunova
Mohammad Ali Ghasemi
Aditya Kela
Aleksandr Korovin
Marina Mircheska
Sattar Rahimbeyli

Task 1.1: acquainting yourself with python for pattern recognition

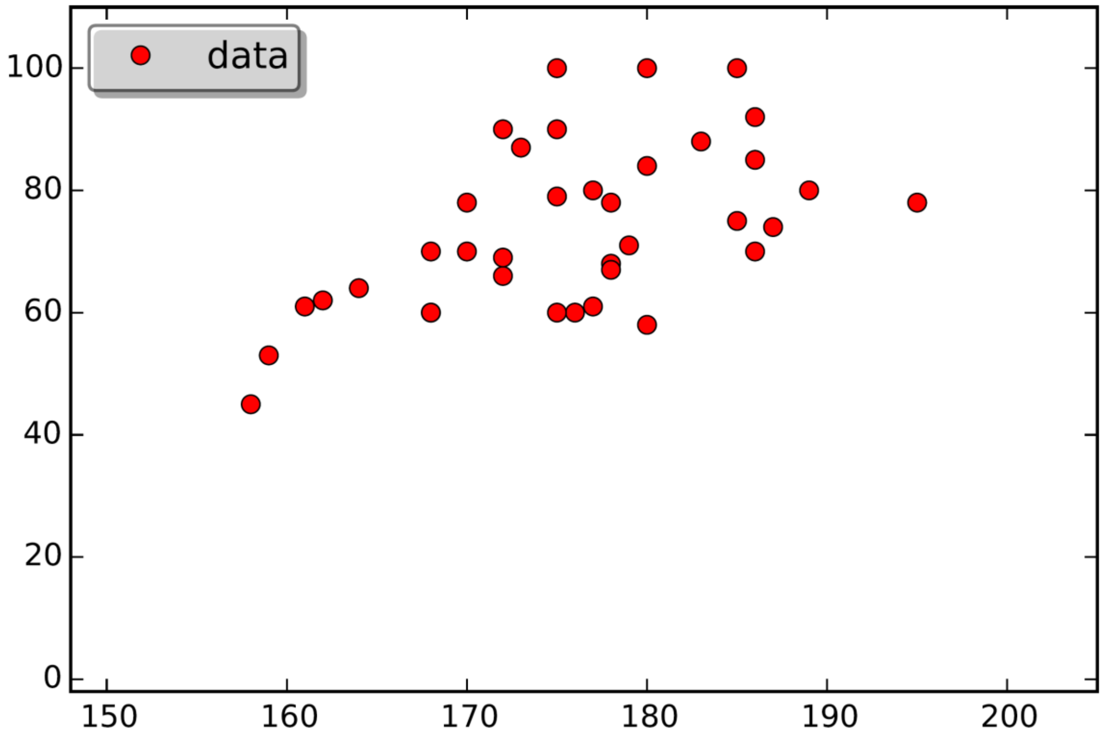
Solution

- given a dataset in whData.dat, containing student data (height, weight)
- the task is to **remove the outliers** – so called cleaning of the dataset
- plot heights vs. weights and weights vs. heights
- our code for cleaning the data:

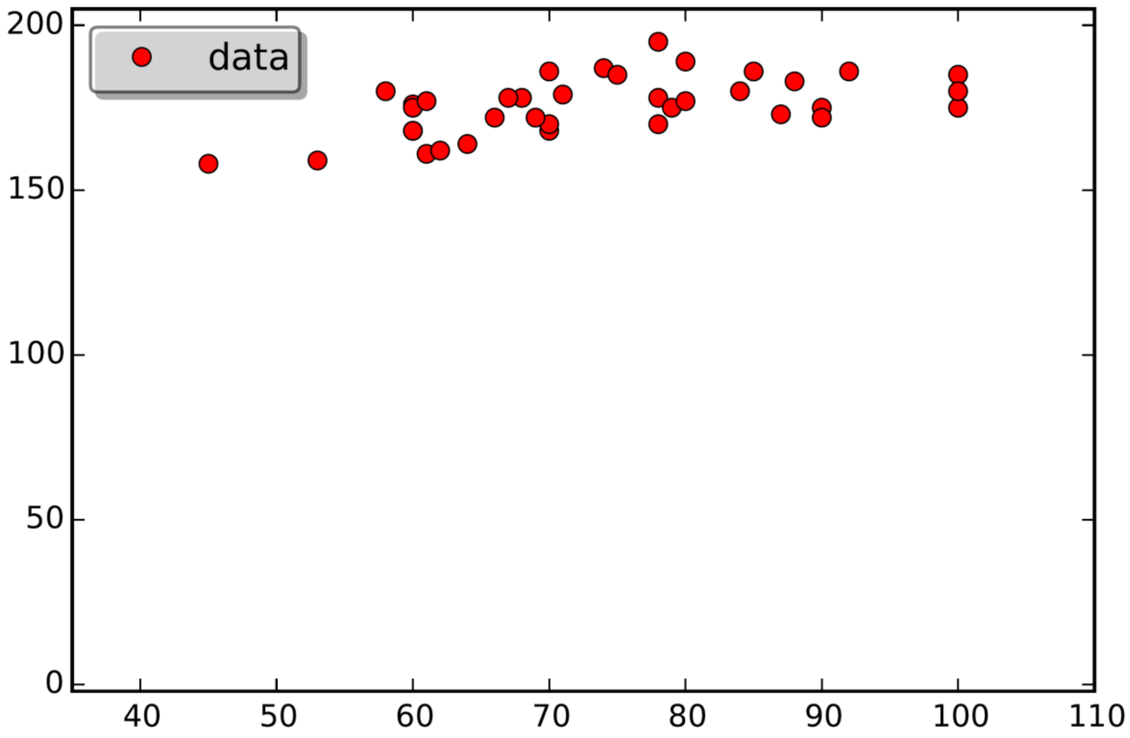
```
# read data as 2D array of data type 'object'
data = np.loadtxt('whData.dat',dtype=np.object,comments='#',delimiter=None)

# read height and weight data into 2D array (i.e. into a matrix)
X = data[:,0:2].astype(np.float)

# remove the outliers
X = X[X[:,0] >= 0]
```

Task 1.1: acquainting yourself with python for pattern recognition

Plots



plot heights vs. weights



plot weights vs. heights

Task 1.2: fitting a Normal distribution to 1D data

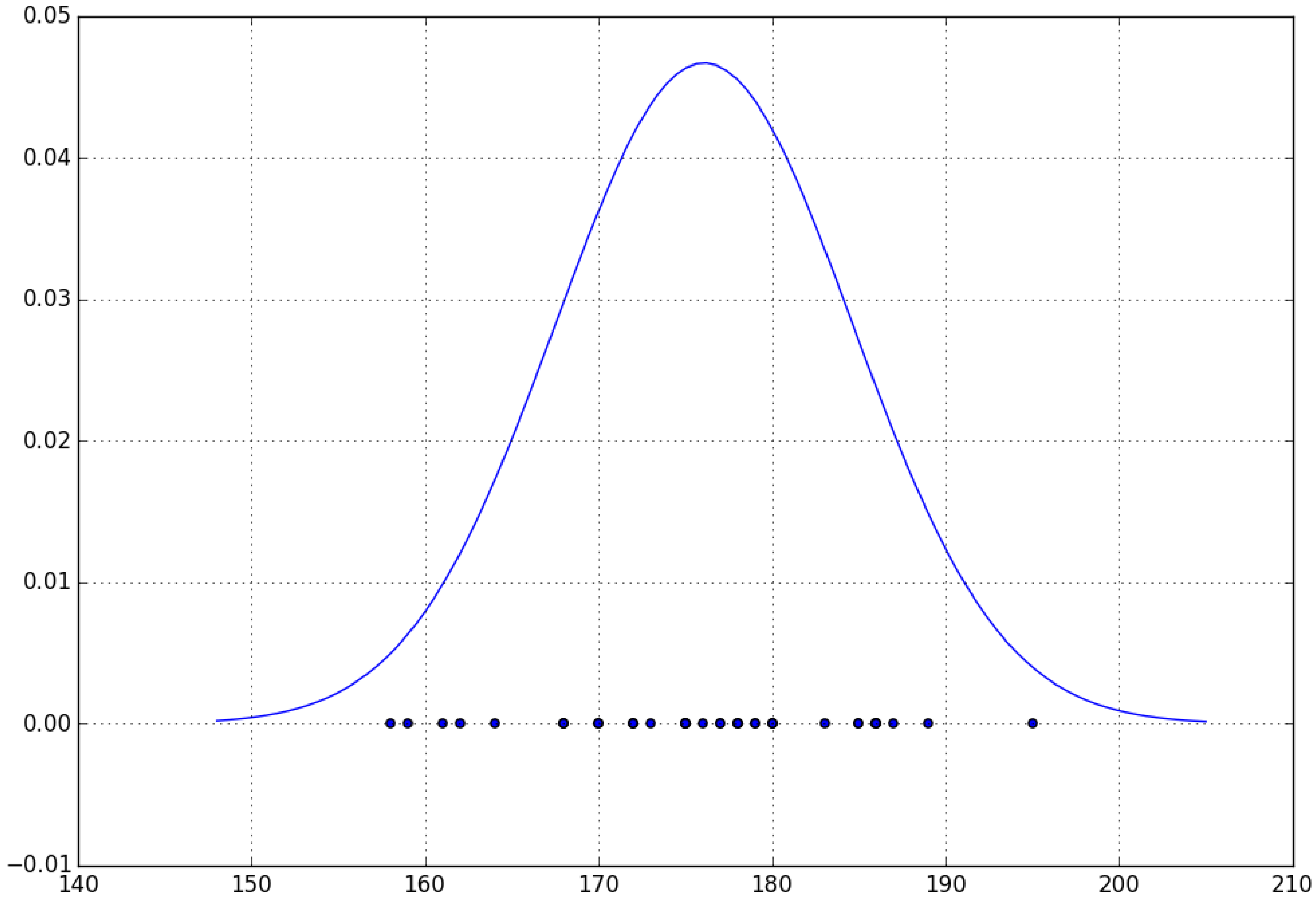
Solution

- **task**: fit Norml distribution on student data (heights) from whData.dat
- **solution**:
  - compute mean and standard deviation
  - plot the data and its Normal distribution
- our code:

```
# compute the mean and standard deviation using built-in numpy functions
mean = np.mean(hs)
std = np.std(hs)

# draw the points on x axis
x = np.linspace(hs.min()-10, hs.max()+10, 100)

# for each point in x, plot the Normal distribution using built-in mlab function
plt.plot(x, mlab.normpdf(x, mean, std))

# plot the 1D heigh data
plt.scatter(hs, np.zeros(hs.size))
```

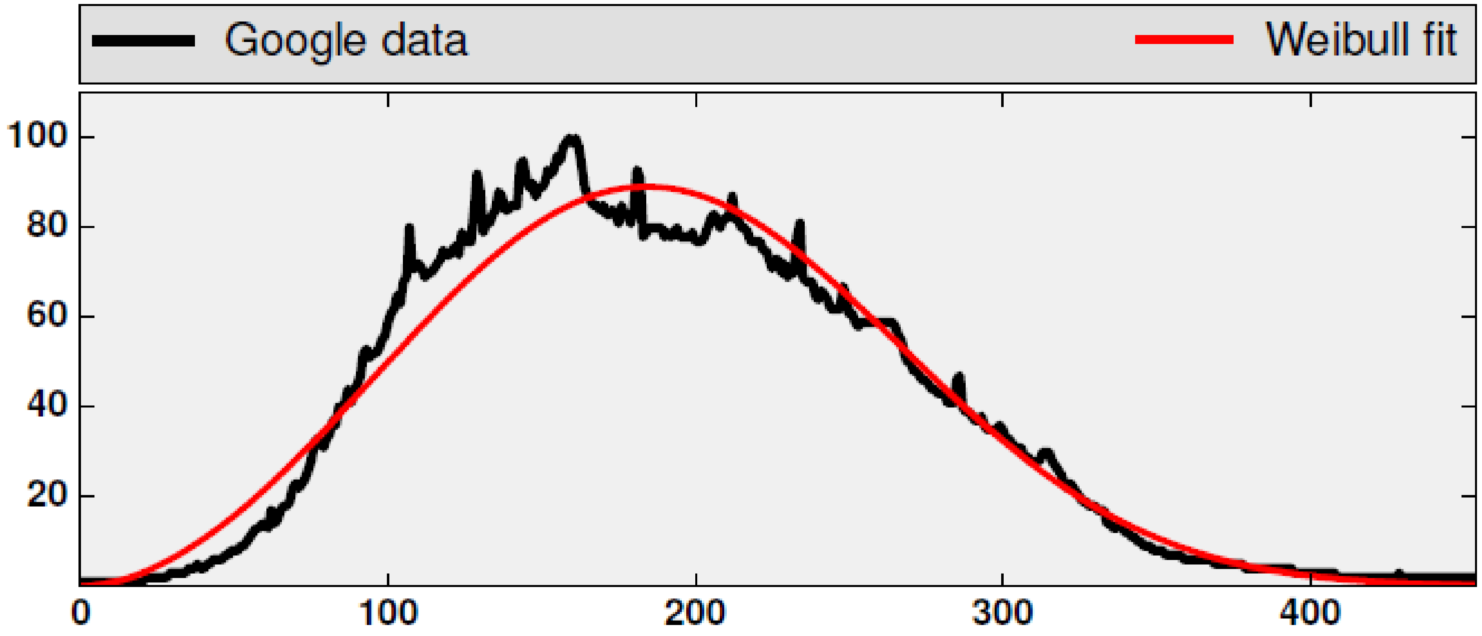
Task 1.2: fitting a Normal distribution to 1D data

Plot



Task 1.3: fitting a Weibull distribution to 1D data

Data and scaled version of the fitted distribution look like this:



Task: Fit a Weibull distribution to the histogram h(x)

Given: Histogram h(x) data (Google trends)

Task 1.3: fitting a Weibull distribution to 1D data

Solution

Probability density function (PDF) of Weibull distribution:

$$f(x \mid \kappa, \alpha) = \frac{\kappa}{\alpha} \left(\frac{x}{\alpha}\right)^{\kappa-1} e^{-\left(\frac{x}{\alpha}\right)^{\kappa}}$$

How to find $\boldsymbol{k}$ and $\boldsymbol{\alpha}$? – Use Log-likelihood for the parameters

$$L(\alpha, \kappa \mid D) = N\big(\log \kappa - \kappa \log \alpha\big) + (\kappa - 1)\sum_i \log d_i - \sum_i (d_i/\alpha)^{\kappa}.$$

To **solve L use Newton's method** for simultaneous equations

$$\begin{bmatrix} \kappa^{\text{new}} \\ \alpha^{\text{new}} \end{bmatrix} = \begin{bmatrix} \kappa \\ \alpha \end{bmatrix} + \begin{bmatrix} \frac{\partial^2 L}{\partial \kappa^2} & \frac{\partial^2 L}{\partial \kappa \partial \alpha} \\ \frac{\partial^2 L}{\partial \kappa \partial \alpha} & \frac{\partial^2 L}{\partial \alpha^2} \end{bmatrix}^{-1} \begin{bmatrix} -\frac{\partial L}{\partial \kappa} \\ -\frac{\partial L}{\partial \alpha} \end{bmatrix}$$

Hessian matrix (elements are gradients) to compute $\boldsymbol{k}$ and $\boldsymbol{\alpha}$ new iteratively

Scale factor:

$$\begin{aligned} \int_0^{+\infty} a\text{Weibull}\, dx &= \text{Area Under The Histogram} \\ \Rightarrow a \int_0^{+\infty} \text{Weibull}\, dx &= \sum_i h[i] \\ \Rightarrow a \times 1 &= 17293 \\ \Rightarrow a &= 17293 \end{aligned}$$

Area under the curve for both, the scaled Weibull and the histogram should be equal

Task 1.3: fitting a Weibull distribution to 1D data

Solution

Deriving Scale and Shape parameter from Newton's method:

$$\begin{bmatrix} \kappa^{\text{new}} \\ \alpha^{\text{new}} \end{bmatrix} = \begin{bmatrix} \kappa \\ \alpha \end{bmatrix} + \begin{bmatrix} \frac{\partial^2 L}{\partial \kappa^2} & \frac{\partial^2 L}{\partial \kappa \partial \alpha} \\ \frac{\partial^2 L}{\partial \kappa \partial \alpha} & \frac{\partial^2 L}{\partial \alpha^2} \end{bmatrix}^{-1} \begin{bmatrix} -\frac{\partial L}{\partial \kappa} \\ -\frac{\partial L}{\partial \alpha} \end{bmatrix}$$

Where the entries can be calculated:

$$\frac{\partial L}{\partial \kappa} = N/\kappa - N \log \alpha + \sum_i \log d_i - \sum_i (d_i/\alpha)^\kappa \log(d_i/\alpha)$$

$$\frac{\partial L}{\partial \alpha} = \kappa/\alpha \left(\sum_i (d_i/\alpha)^\kappa - N \right)$$

$$\frac{\partial^2 L}{\partial \kappa^2} = -N/\kappa^2 - \sum_i (d_i/\alpha)^\kappa \left(\log(d_i/\alpha)\right)^2$$

$$\frac{\partial^2 L}{\partial \alpha^2} = \kappa/\alpha^2 \left(N - (\kappa+1) \sum_i (d_i/\alpha)^\kappa \right)$$

$$\frac{\partial^2 L}{\partial \kappa \partial \alpha} = 1/\alpha \sum_i (d_i/\alpha)^\kappa + \kappa/\alpha \sum_i (d_i/\alpha)^\kappa \log(d_i/\alpha) - N/\alpha.$$

→

```
def newton(k,a,dataList):
    #We Input parameters 'k' and 'a' (alpha) into the function.
    N=len(dataList)

   #Calculated all the matrix elements of the Newtonian Method.
    B1=N/k-N*math.log(a)+np.sum(np.log(dataList))-np.sum(((dataList/a)**k)*np.log(dataList/a))
    B2=(k/a)*(np.sum((dataList/a)**k)-N)
    M11=-N/(k**2)-np.sum(((dataList/a)**k)*(np.log(dataList/a))**2)
    M22=(k/((a)**2))*(N-(k+1)*np.sum((dataList/a)**k))
    M12=M21=(1/a)*np.sum((dataList/a)**k)+(k/a)*np.sum(((dataList/a)**k)*np.log(dataList/a))-N/a

    return np.array(np.matmul(np.linalg.inv(np.matrix([[M11,M12],[M21,M22]])),np.array([-B1,-
B2]))+np.array([k,a]))[0]
```

Task 1.3: fitting a Weibull distribution to 1D data

Solution: code

Important!: we should make a dataset for MLE procedure and we know that H is the frequency of x. Thus we generate dataset to work with Newton's function.

After 20 Iteration we've found the exact value of $\boldsymbol{k}$ and $\boldsymbol{\alpha}$:

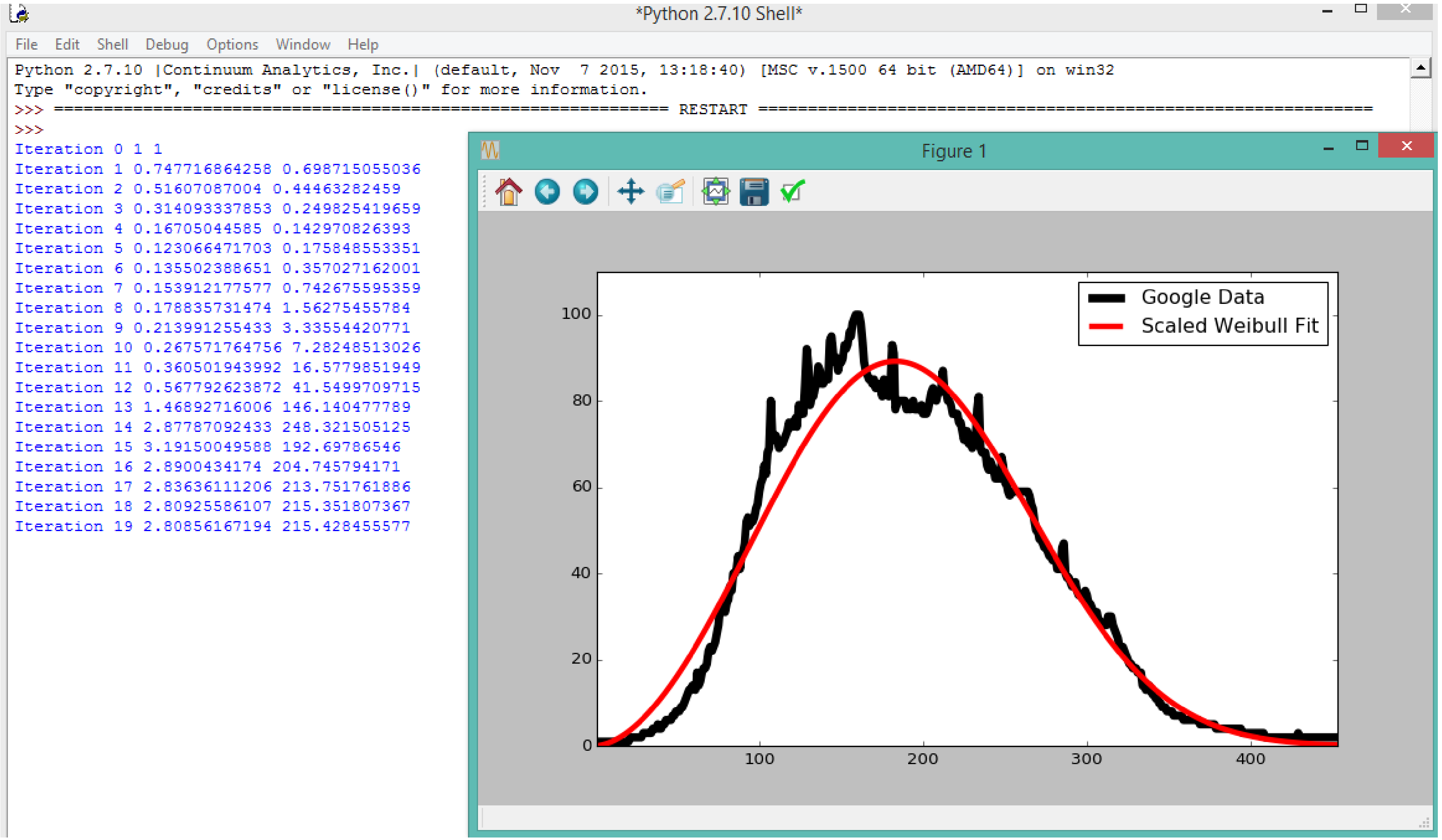
```
Iteration 0 1 1
Iteration 1 0.747716864258 0.698715055036
Iteration 2 0.51607087004 0.44463282459
Iteration 3 0.314093337853 0.249825419659
Iteration 4 0.16705044585 0.142970826393
Iteration 5 0.123066471703 0.175848553351
Iteration 6 0.135502388651 0.357027162001
Iteration 7 0.153912177577 0.742675595359
Iteration 8 0.178835731474 1.56275455784
Iteration 9 0.213991255433 3.33554420771
Iteration 10 0.267571764756 7.28248513026
Iteration 11 0.360501943992 16.5779851949
Iteration 12 0.567792623872 41.5499709715
Iteration 13 1.46892716006 146.140477789
Iteration 14 2.87787092433 248.321505125
Iteration 15 3.19150049588 192.69786546
Iteration 16 2.8900434174 204.745794171
Iteration 17 2.83636111206 213.751761886
Iteration 18 2.80925586107 215.351807367
Iteration 19 2.80856167194 215.428455577
```

```python
#Data generation for Newton's Method.
data=[];
for i in range(len(histData)):
  data=np.append(data,[xValues[i]]*int(histData[i]))

....

def iters(k,a,n,dataList):
    oldPara=np.array([k,a])
    for i in range(n):
        newPara=newton(oldPara[0],oldPara[1],dataList)
        oldPara=newPara
    return newPara
```

Task 1.3: fitting a Weibull distribution to 1D data

Solution: plots

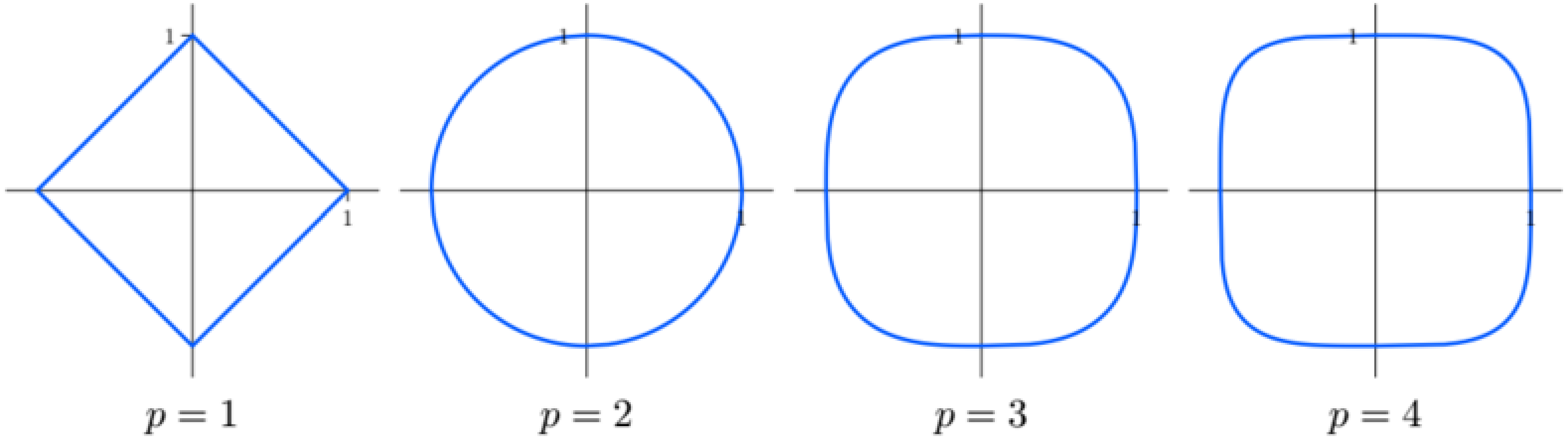
Finally!: This is how it we generated fitted Weibull Dist. to 1D data



Task 1.4: drawing unit circles

L_p norms for R^m, for different *p*;

The corresponding unit spheres may look different. For instance, the following examples show unit circles in R^2:



Task 1.4: drawing unit circles

Solution

Consider the L_p norm for $p = 0.5$ and plot the corresponding R^2 unit circle.

In R^2 for $n > 1$, the following expression describes the L_p norm:

$$\|x\|_p = (|x_1|^p + |x_2|^p + \ldots + |x_n|^p)^{\frac{1}{p}}$$

From the formula above, we derive the expression for the y coordinate:

$$\|x\|_p = 1$$
$$|x|^p + |y|^p = 1$$
$$y = (1 - |x|^p)^{\frac{1}{p}}$$

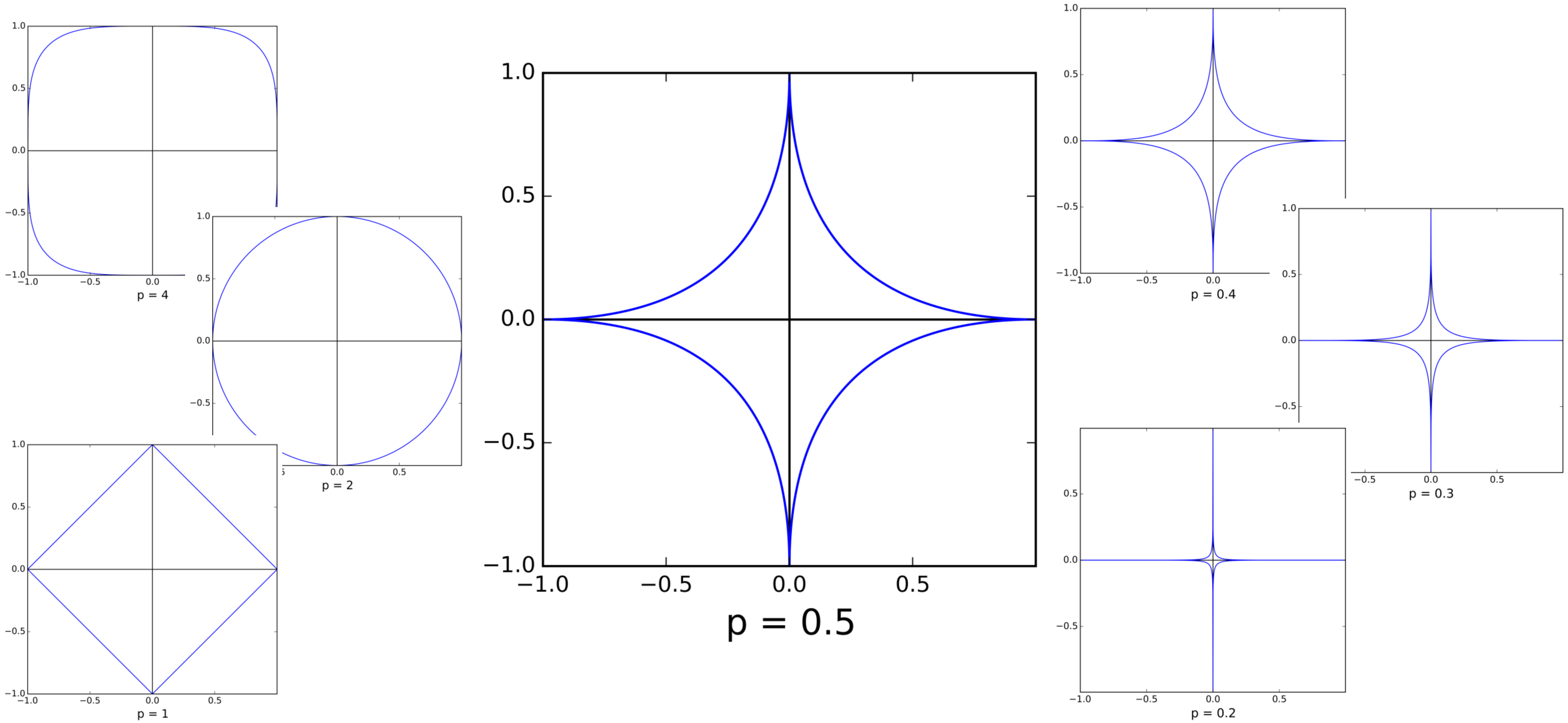
Task 1.4: drawing unit circles

Solution: code

```
def generateUnitCircle(p):
    '''
    Generates a unit circle of norm p
    '''
    exponent = np.true_divide(1,p)

    x = np.arange(-1.0, +1.0, 0.001) # to accelerate the processing, increase the last parameter

    uppercircle_y = np.power(1 - np.abs(np.power(np.abs(x), p)), exponent)
    lowercircle_y = -np.power(1 - np.abs(np.power(np.abs(x), p)), exponent)

    coordinates_x = np.append(x, x[::-1])
    coordinates_y = np.append(uppercircle_y, lowercircle_y[::-1])

    return coordinates_x, coordinates_y

-----------------------------------------------------------------------------------------------------------

    # generating a unit circle of norm p as a parameter
    pnorm = 0.5
    coord_x, coord_y = generateUnitCircle(pnorm)
```

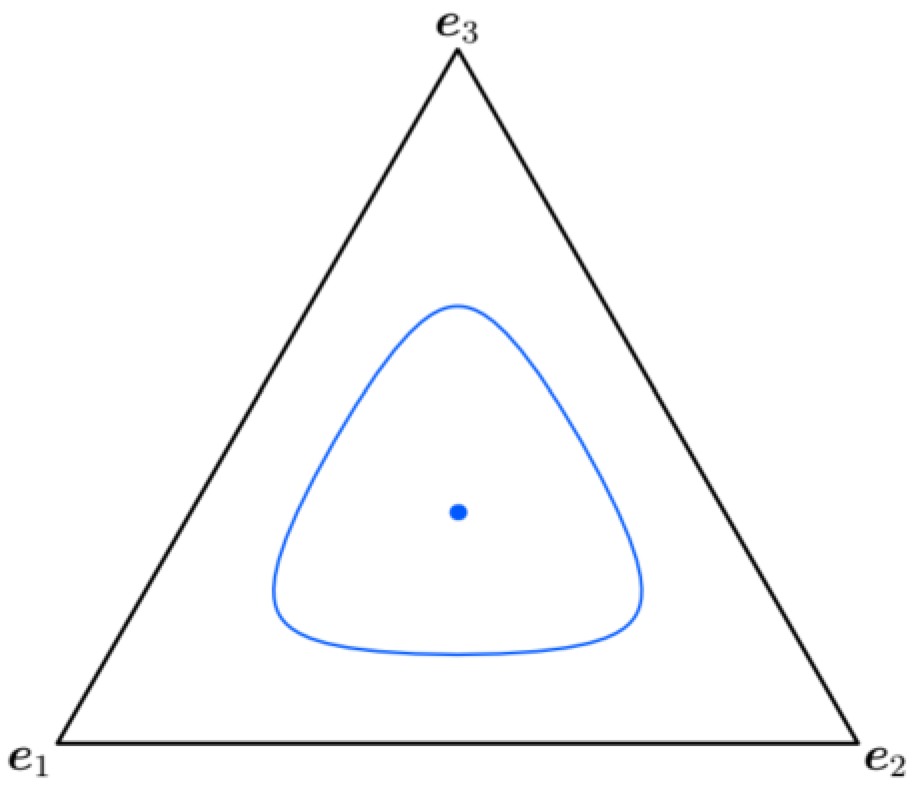
Task 1.4: drawing unit circles

Plots



Bonus task: drawing circles in S^3

Aitchinson geometry of the vector space S^3: unit circle looks like this:



Task: plot a circle centered at $0 \in S^3$ with a radius $r = 3$

Bonus task: drawing circles in S^3

Theory (Egozcue et al. Modeling and Analysis of Compositional Data, 2015)

Let:

- $\boldsymbol{x} \in S^D$ - a vector in the simplex S^D (a composition)
- $\{\boldsymbol{e}_1, \boldsymbol{e}_2, \dots, \boldsymbol{e}_{D-1}\}$ – orthonormal basis of the simplex S^D
- $\mathrm{clr}(\boldsymbol{x})$ - transformation that gives the expression of a composition in centered logratio coefficients
- $g_m(\boldsymbol{x})$ - componentwise geometric mean of composition

$$\mathrm{clr}(\boldsymbol{x}) = C\left[\ln\frac{x_1}{g_m(\boldsymbol{x})}, \ln\frac{x_2}{g_m(\boldsymbol{x})}, \dots, \ln\frac{x_D}{g_m(\boldsymbol{x})}\right] = \boldsymbol{\xi}$$

$$g_m(\boldsymbol{x}) = \exp\left(\frac{1}{D}\sum_{i=1}^{D} \ln x_i\right)$$

- the clr coordinates:
  - are not coordinates with respect to a basis of a simplex
  - allow to translate operations and metrics from the simplex into real space
  - clr^{-1} gives the coefficients in the canonical basis of the real space

- $\boldsymbol{\Psi}$– contrast matrix, $\boldsymbol{\Psi}_i = \mathrm{clr}(\mathbf{e}_i), i = 1,2, \dots, D-1$
  - using clr^{-1} on each row of the matrix, we can recover the compositions of the basis

Bonus task: drawing circles in S^3

Solution

An algorithm to recover $\boldsymbol{x}$ from its coordinates $\boldsymbol{x}^*$:

1. Construct the contrast matrix $\boldsymbol{\Psi}$ of the basis;
   Egozcue et al. (2003) obtained the following expression for the contrast matrix:

$$\boldsymbol{\Psi}_{ij} = \begin{cases} +\sqrt{\dfrac{1}{(D-i)(D-i+1)}}, & j \leq D-i; \\ -\sqrt{\dfrac{D-i}{D-i+1}}, & j = D-i+1; \\ 0, & \text{otherwise.} \end{cases}$$

2. Compute the matrix product $\boldsymbol{x}^*\boldsymbol{\Psi}$;

3. Apply clr^{-1} to obtain $\boldsymbol{x}$.

$$\mathrm{clr}^{-1}(\boldsymbol{\xi}) = \mathcal{C}[\exp(\boldsymbol{\xi}_1), \exp(\boldsymbol{\xi}_2), \dots, \exp(\boldsymbol{\xi}_D)] = \boldsymbol{x}$$

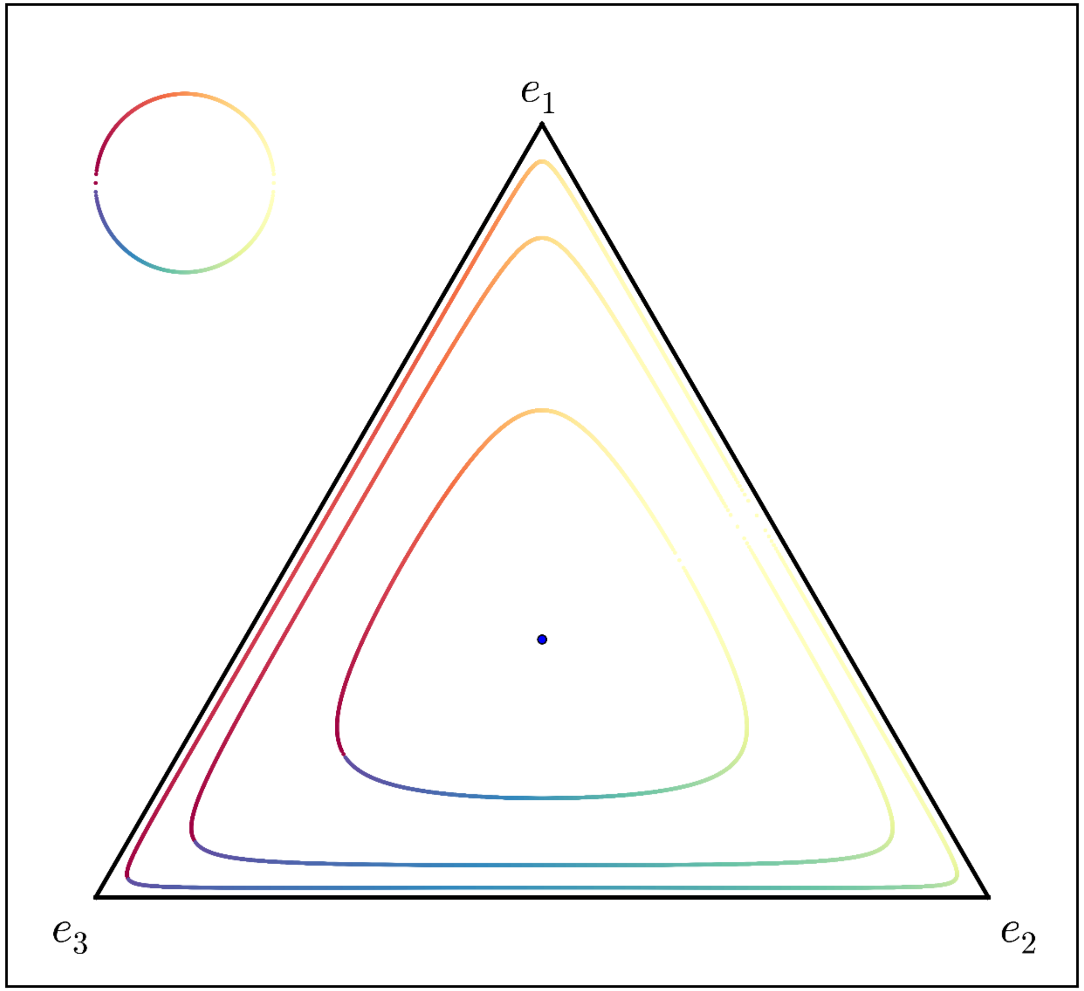
Bonus task: drawing circles in S^3

Solution: code

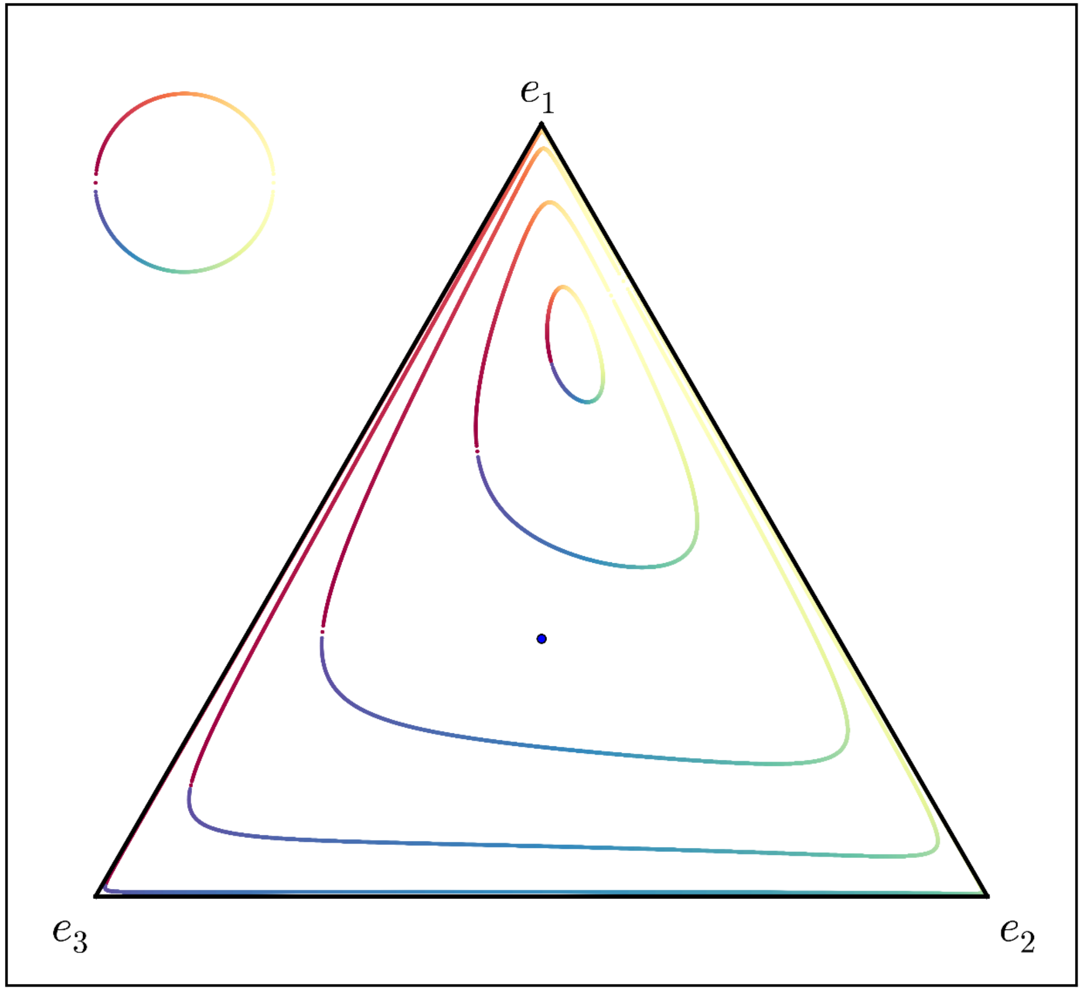
```
def transformToCompositions(euclead_x, euclead_y):

    contrastMatrix = np.matrix([[1/np.sqrt(6), 1/np.sqrt(6), -np.sqrt(np.true_divide(2,3))], [1/np.sqrt(2), -1/np.sqrt(2), 0]])
    vector_x = np.array([])
    vector_y = np.array([])

    for i in range(euclead_x.size):
        # each pair of Eucledean coordinates is multiplied by the contrast matrix
        pair = np.array([euclead_x[i], euclead_y[i]])
        composition = np.asarray(np.dot(pair,contrastMatrix))[0]
        sum = 0

        # the inverse closure operator is applied to the result
        for i in range(composition.size):
            composition[i] = np.exp(composition[i])
            sum += composition[i]

        # the coordinates are normalized and compositions are obtained
        for i in range(composition.size):
            composition[i] = composition[i]/sum

        # calculating the compositions' coordinates for a ternary plot
        # y = sqrt(3) * (x - coord[1])
        y = composition[0] * height
        x = y / np.sqrt(3) + composition[1]
        vector_x = np.append(vector_x, x)
        vector_y = np.append(vector_y, y)

    return vector_x, vector_y
```

Bonus task: drawing circles in S^3

Plots



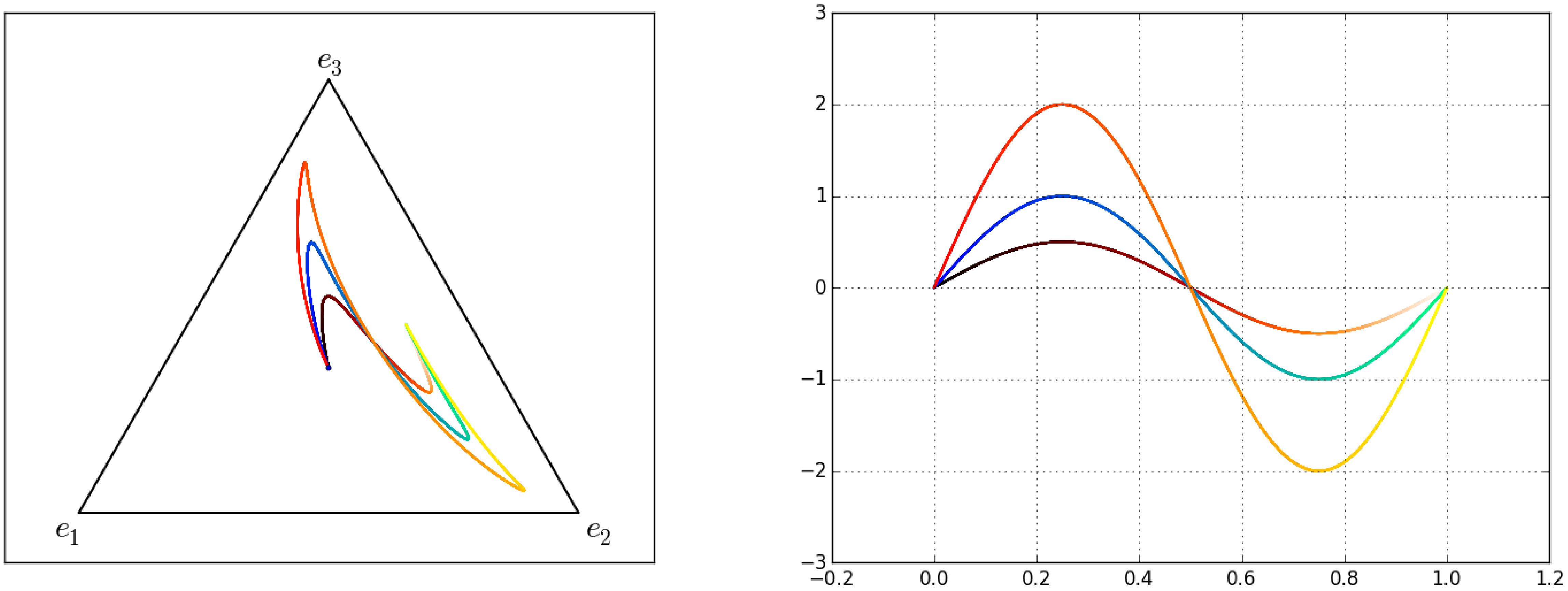
circles centered at $0 \in S^3$ with radii $r = 1, 2, 3$



circles centered at $[0.72046295 \;\; 0.1751566 \;\; 0.10438044] \in S^3$ (corresponds to $(1, 1) \in \mathbb{R}^2$) with radii $r = 0.5, 1, 2, 3, 5$

Bonus task: drawing circles in S^3

Plots



Three sinusoids in $\mathbb{R}^2$ and their corresponding transformations onto vector space S^3