project-01-demo

December 10, 2015

1 Project 1

1.1 B-IT Pattern Recognition

Presented on 10-Dec-2015 by:

- Abdullah Abdullah
- Can Güney Aksakallı
- Kang Cifong
- Umut Hatipoğlu

1.2 Task 1.1

1.2.1 Remove the outliers

1.2.2 Load the data first

- We went with the approach **1** of reading multi-typed data as in `whExample.py`
  - We like it because it is more explicit
  - Or use `pandas`
- We are using **Python 3.4**, so some modifications were made for compatibility

In [47]:
```
import numpy as np
import csv
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
%matplotlib inline

import pattrex.plotting_mpl as plt_rex
import pattrex.preprocessing as pre_rex
import pattrex.fitting as fit_rex
import pattrex.unit_circles as uc_rex
```

In [48]:
```
dt = np.dtype([('w', np.float), ('h', np.float), ('g', 'S1')])  # g is byte-string

data = np.loadtxt('data/whData.dat', dtype=dt, comments='#', delimiter=None)

ws = np.array([d[0] for d in data])
hs = np.array([d[1] for d in data])
```

```
gs = np.array([d[2].decode('utf-8') for d in data])

X = np.vstack((hs, ws, gs))  # data is going to be column-wise
# X.transpose()  # this will make it row-wise
X.shape
```

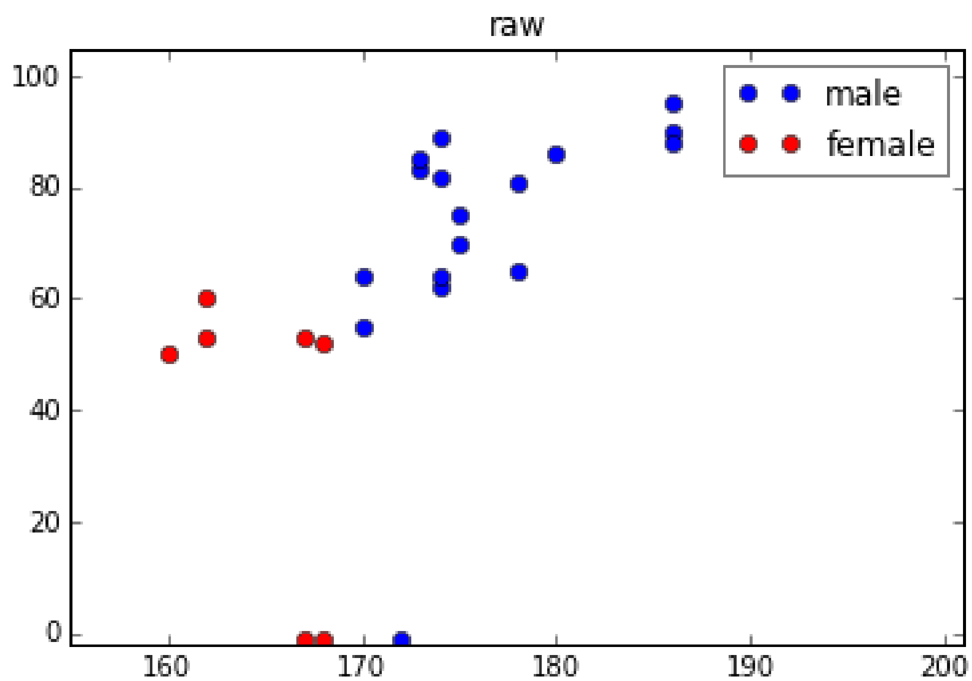
```
Out[48]: (3, 24)
```

1.2.3 Raw Data

- Now, let's just plot it without modifications
  - We split the data based on gender

```
In [49]: # split
         X_male, X_female = pre_rex.split_data(X, True, 2, ['m', 'f'])
         print("male :", X_male.shape[1], "; female :", X_female.shape[1])

         # plotting
         fig = plt.figure()
         axs = fig.add_subplot(111)

         # limits for the axes
         X_ = np.vstack((hs, ws))  # only the measurements; data is col-wise
         xmin, ymin = X_.min(axis=1)
         xmax, ymax = X_.max(axis=1)

         xlim = [xmin-5, xmax+15]  # purely for looks
         ylim = [-2, ymax+10]

         plt_rex.plot2d(X_male, colwise_data=True, hatch='bo', x_lim=xlim, y_lim=ylim,
                        show=False, axs=axs, set_aspect_equal=False, plotlabel="male")
         plt_rex.plot2d(X_female, colwise_data=True, hatch='ro', x_lim=xlim,
                         y_lim=ylim, show=False, axs=axs, set_aspect_equal=False,
                         plotlabel="female", title="raw")
```

```
male : 17 ; female : 7
```



1.2.4 Outliers!!

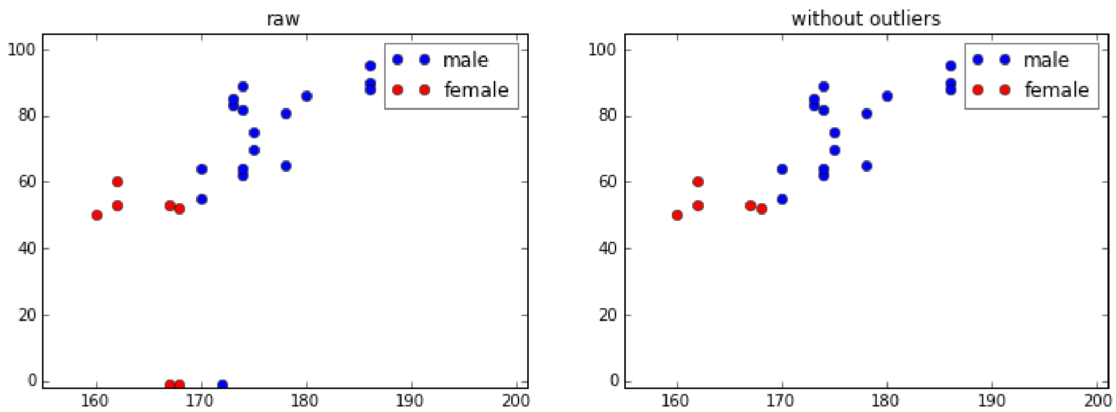
Outliers may give important insights

1.2.5 Dealing with the outliers

- Here, we just ignore them
  - We keep ... > only those data for which both measurements are positive.
- We find the *unique* columns/rows for which any of the measurements are negative
  - Then we delete the entire columns/rows that contain such measurements
    - using `numpy.delete(...)`

In [50]:
```
X_male_new = pre_rex.only_all_positive(X_male[0:2, :].astype(np.float), True)
X_female_new = pre_rex.only_all_positive(X_female[0:2, :].astype(np.float), True)

# plotting
fig = plt.figure(figsize=(12, 4))
axs1 = fig.add_subplot(121)
axs2 = fig.add_subplot(122)

# using the old xlim, ylim

plt_rex.plot2d(X_male, colwise_data=True, hatch='bo', x_lim=xlim, y_lim=ylim,
               show=False, axs=axs1, plotlabel="male")
plt_rex.plot2d(X_female, colwise_data=True, hatch='ro',
               x_lim=xlim, y_lim=ylim, show=False, axs=axs1,
```

```
                  plotlabel="female", title="raw")

plt_rex.plot2d(X_male_new, colwise_data=True, hatch='bo', x_lim=xlim,
               y_lim=ylim, show=False, axs=axs2, plotlabel="male")
plt_rex.plot2d(X_female_new, colwise_data=True, hatch='ro', x_lim=xlim,
               y_lim=ylim, show=False, axs=axs2, plotlabel="female",
               title="without outliers")
```



1.3 Task 1.2

1.3.1 Fit normal distribution to data

1.3.2 Find the mean and standard Deviation of the height/weight data

- We used `numpy.mean(...)` and `numpy.std(...)`
- Then we use `scipy.stats.norm.pdf` to generate the normal distribution

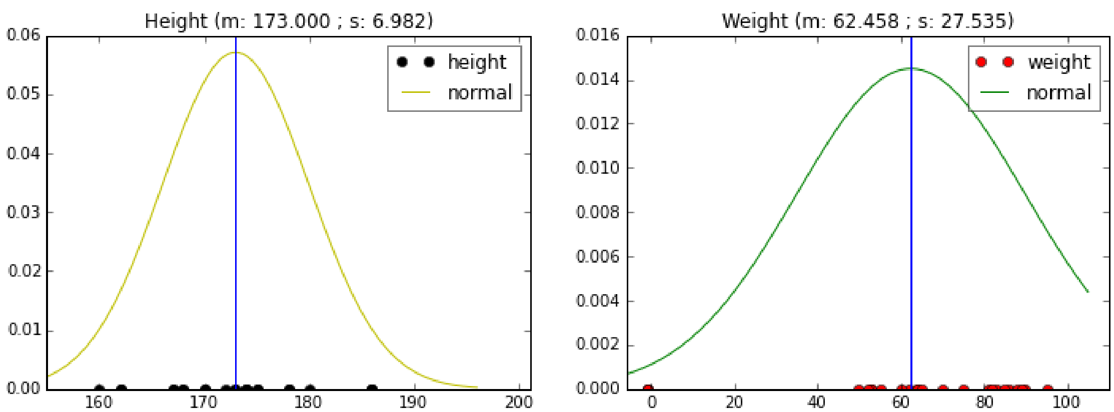
1.3.3 Plot

In [51]:
```
# fit normal distribution
h_mean, h_std, h_x, h_y = fit_rex.fit_normal_distribution(hs)

w_mean, w_std, w_x, w_y = fit_rex.fit_normal_distribution(ws)

# limits for the axes, and yes we are going to cheat by using X_
hmin, wmin = X_.min(axis=1)
hmax, wmax = X_.max(axis=1)

hlim = [hmin-5, hmax+15]  # purely for looks
wlim = [wmin-5, wmax+15]
```

In [52]:
```
# plotting
fig = plt.figure(figsize=(12, 4))
axs1 = fig.add_subplot(121)
axs2 = fig.add_subplot(122)

# height
plt_rex.plot2d(np.vstack((hs, np.zeros(hs.shape))), colwise_data=True,
```

```
                  hatch='ko', x_lim=hlim, show=False, axs=axs1,
                  plotlabel="height")
plt_rex.plot2d(np.vstack((h_x, h_y)), colwise_data=True, hatch='y',
                  x_lim=hlim, show=False, axs=axs1, plotlabel="normal",
                  title=("Height (m: %.3f ; s: %.3f)")%(h_mean, h_std))
axs1.axvline(x = h_mean)

# weight
plt_rex.plot2d(np.vstack((ws, np.zeros(ws.shape))), colwise_data=True,
                  hatch='ro', x_lim=wlim, show=False, axs=axs2,
                  plotlabel="weight")
plt_rex.plot2d(np.vstack((w_x, w_y)), colwise_data=True, hatch='g',
                  x_lim=wlim, show=False, axs=axs2, plotlabel="normal",
                  title=("Weight (m: %.3f ; s: %.3f)")%(w_mean, w_std))
axs2.axvline(x = w_mean)
```

Out[52]: <matplotlib.lines.Line2D at 0x121fd2a90>



1.3.4 Outliers!!

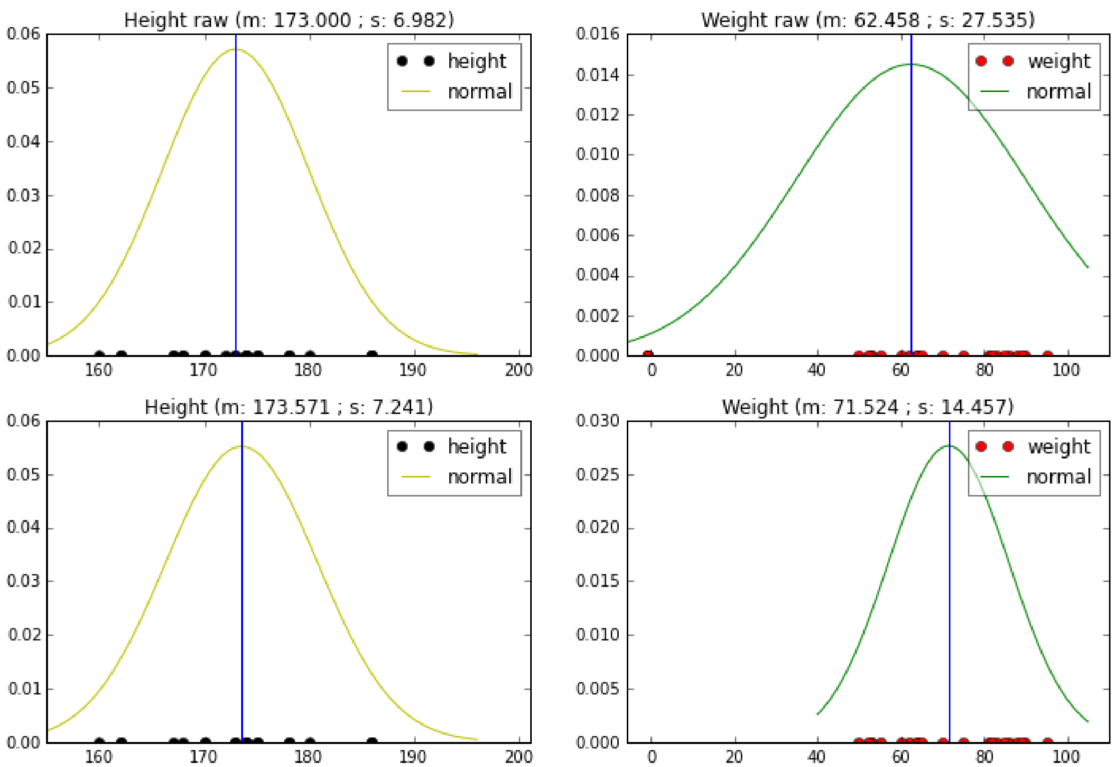
1.3.5 Plot (without outliers)

In [53]:
```
X_new = pre_rex.only_all_positive(X_, True)
h_new = X_new[0, :]
w_new = X_new[1, :]

# fit normal distribution
h_mean_new, h_std_new, h_x_new, h_y_new = fit_rex.fit_normal_distribution(h_new)

w_mean_new, w_std_new, w_x_new, w_y_new = fit_rex.fit_normal_distribution(w_new)
```

In [54]:
```
# plotting
fig = plt.figure(figsize=(12, 8))
axs1 = fig.add_subplot(221)
axs2 = fig.add_subplot(222)
axs3 = fig.add_subplot(223)
axs4 = fig.add_subplot(224)
```

```python
# limits for the axes, and yes we are going to cheat by using X_
hmin_new, wmin_new = X_new.min(axis=1)
hmax_new, wmax_new = X_new.max(axis=1)

hlim_new = [hmin_new-5, hmax_new+15]  # purely for looks
wlim_new = [wmin_new-5, wmax_new+15]

# height raw
plt_rex.plot2d(np.vstack((hs, np.zeros(hs.shape))), colwise_data=True,
               hatch='ko', x_lim=hlim_new, show=False, axs=axs1,
               plotlabel="height")
plt_rex.plot2d(np.vstack((h_x, h_y)), colwise_data=True, hatch='y',
               x_lim=hlim_new, show=False, axs=axs1, plotlabel="normal",
               title=("Height raw (m: %.3f ; s: %.3f)")%(h_mean, h_std))
axs1.axvline(x = h_mean)

# weight raw
plt_rex.plot2d(np.vstack((ws, np.zeros(ws.shape))), colwise_data=True,
               hatch='ro', x_lim=wlim, show=False, axs=axs2,
               plotlabel="weight")
plt_rex.plot2d(np.vstack((w_x, w_y)), colwise_data=True, hatch='g',
               x_lim=wlim, show=False, axs=axs2, plotlabel="normal",
               title=("Weight raw (m: %.3f ; s: %.3f)")%(w_mean, w_std))
axs2.axvline(x = w_mean)

# height
plt_rex.plot2d(np.vstack((h_new, np.zeros(h_new.shape))), colwise_data=True,
               hatch='ko', x_lim=hlim_new, show=False, axs=axs3,
               plotlabel="height")
plt_rex.plot2d(np.vstack((h_x_new, h_y_new)), colwise_data=True, hatch='y',
               x_lim=hlim_new, show=False, axs=axs3, plotlabel="normal",
               title=("Height (m: %.3f ; s: %.3f)")%(h_mean_new, h_std_new))
axs3.axvline(x = h_mean_new)

# weight
plt_rex.plot2d(np.vstack((w_new, np.zeros(w_new.shape))), colwise_data=True,
               hatch='ro', x_lim=wlim, show=False, axs=axs4,
               plotlabel="weight")
plt_rex.plot2d(np.vstack((w_x_new, w_y_new)), colwise_data=True, hatch='g',
               x_lim=wlim, show=False, axs=axs4, plotlabel="normal",
               title=("Weight (m: %.3f ; s: %.3f)")%(w_mean_new, w_std_new))
axs4.axvline(x = w_mean_new)
```

Out[54]: <matplotlib.lines.Line2D at 0x12308b128>



1.3.6 Outliers

We see how the presence of outliers can mess things up for this simple model
Central Moments are not robust to outliers

1.4 Task 1.3

1.4.1 Fitting a Weibull Distribution

1.4.2 Weibull Distribution fitting using MLE

- Algorithm is simple to understand

  1. initialize
  2. keep updating while change is significant

- All equations were already given

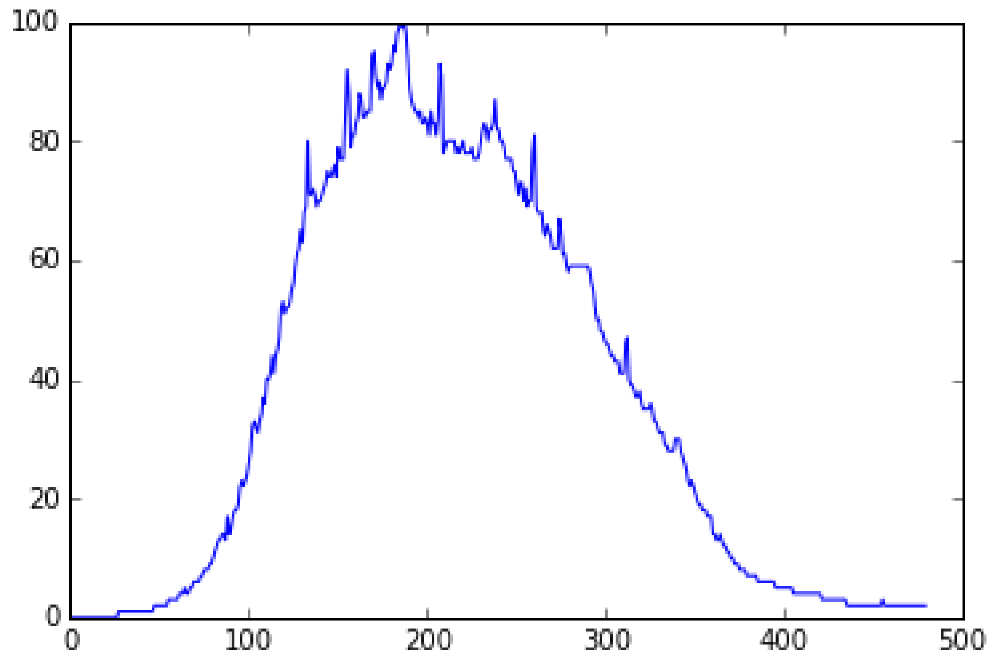
1.4.3 But first, get the data

```
In [55]: data_all = []
         data_ = []
         with open("data/myspace.csv") as f:
             read = csv.reader(f)
             for row in read:
                 data_all.append(row)
                 data_.append(int(row[1]))
```

```
data_ = np.array(data_)
print(len(data_all))
plt.figure(figsize=(6, 4))
plt.plot(data_)
```

```
480
```

Out[55]: [<matplotlib.lines.Line2D at 0x123179dd8>]



1.4.4 As noted

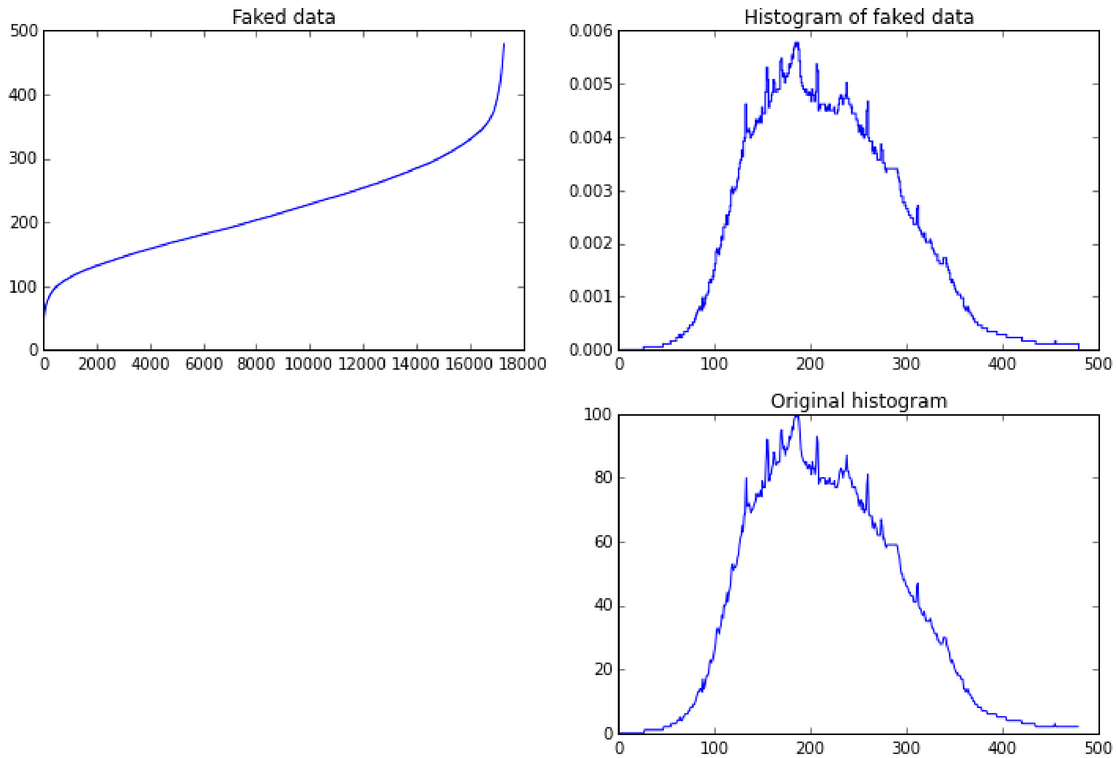
- This is a histogram (which is basically a scaled probability distribution)
- It is easy (but wrong) to use this as the real data (/measurements)
- We wrapped our head around this by assuming that the data represented number of weekly visits to the site
- So, to use the given equations, we have to generate some fake data
  - keep in mind that the data should be > 0

In [56]:
```
fake_data = []
v = 0
for h in data_:
    if h != 0:
        for n in range(h):
            fake_data.append(v)  # same user visiting h times


    v += 1
```

```
data = np.array(fake_data)
print(data.size)

fig = plt.figure(figsize=(12, 8))
axs1 = fig.add_subplot(221)
axs2 = fig.add_subplot(222)
axs3 = fig.add_subplot(224)

axs1.plot(data)
axs1.set_title("Faked data")
_ = axs2.hist(data, bins=np.linspace(0, v, data_.size),
              histtype='step', normed=True)
axs2.set_title("Histogram of faked data")
axs3.plot(data_)
axs3.set_title("Original histogram")
```

```
17293
```

Out [56]: <matplotlib.text.Text at 0x1235f1860>



1.4.5 Now...

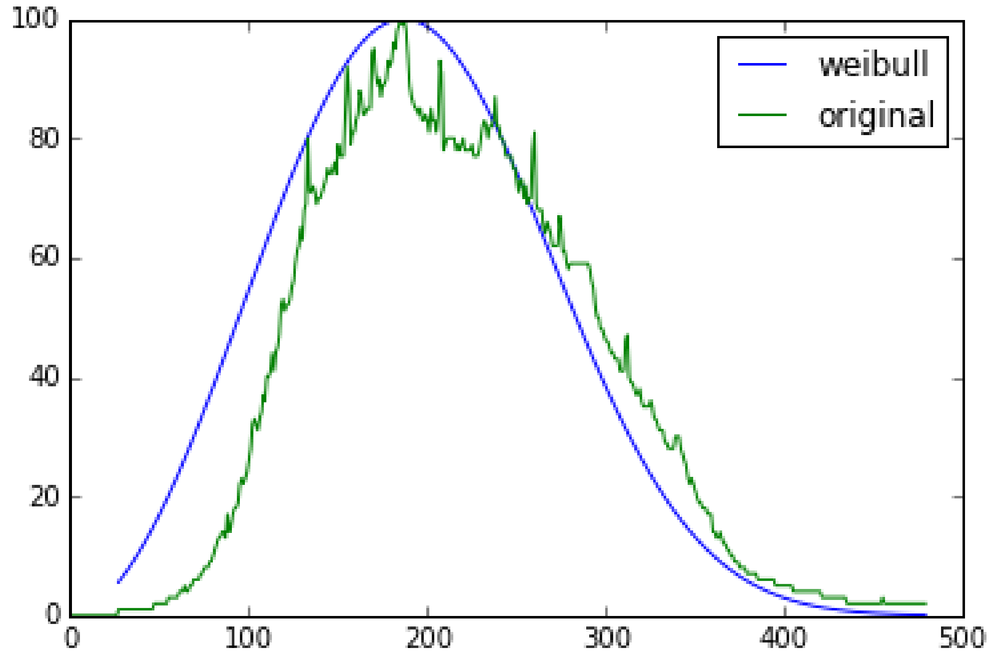
we find the parameters of the Weibull Distribution that fits this *probability distribution*
And as instructed, we use the Newton Raphson method for MLE

In [57]:
```
k, a, _, weibull_probs = fit_rex.fit_weibull_distribution(data)
print("k :", k, "; a :", a)
```

```
fig = plt.figure(figsize=(6, 4))
axs = fig.add_subplot(111)
axs.plot(data, data_.max() * weibull_probs/weibull_probs.max())
axs.plot(data_)
axs.legend(["weibull", "original"])
```

```
k : 2.82724539954 ; a : 217.753098459
```

Out[57]: <matplotlib.legend.Legend at 0x1236ee6d8>



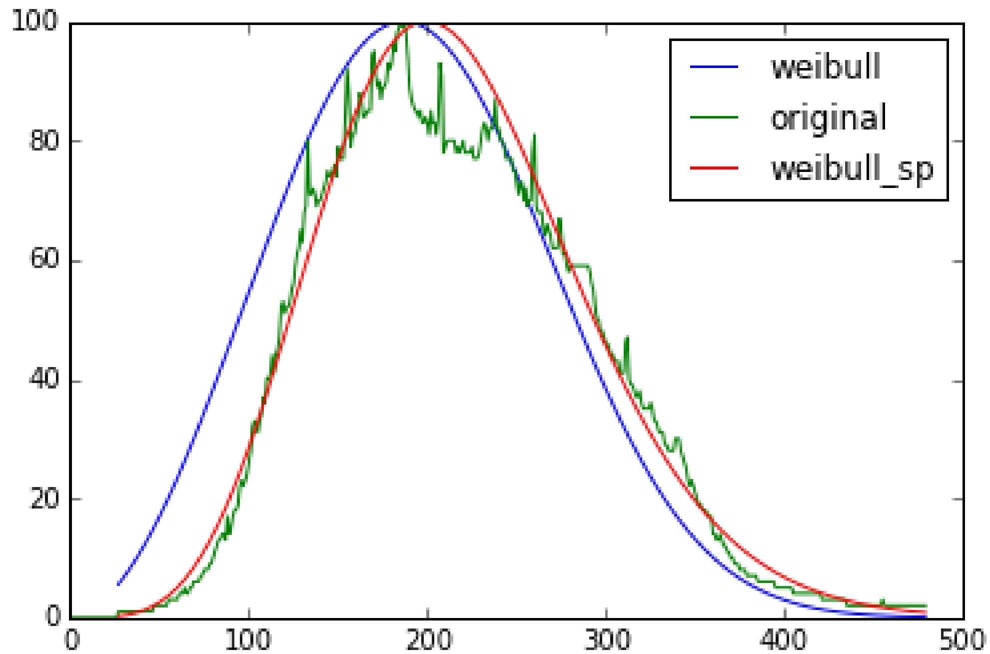
1.5 Hmm...

1.5.1 Let's check what a scipy.stats has to offer

In [58]:
```
vals, _, weibull_probs_sp = fit_rex.fit_weibull_distribution_sp(data)
print(vals)

fig = plt.figure(figsize=(6, 4))
axs = fig.add_subplot(111)
axs.plot(data, data_.max() * weibull_probs/weibull_probs.max())
axs.plot(data_)
axs.plot(data, data_.max() * weibull_probs_sp/weibull_probs_sp.max())
axs.legend(["weibull", "original", "weibull_sp"])
```

```
(2.5567578325685147, 2.0333230524292247, -2.0387030220680509, 178.85627834902073)
```

Out[58]: <matplotlib.legend.Legend at 0x123a23470>



- There might problem with our implementation.
- Our internal initialization of $\kappa = 1$ and $\alpha = mean(Data)$ might be wrong.
- May be we are stopping the iterations too soon.

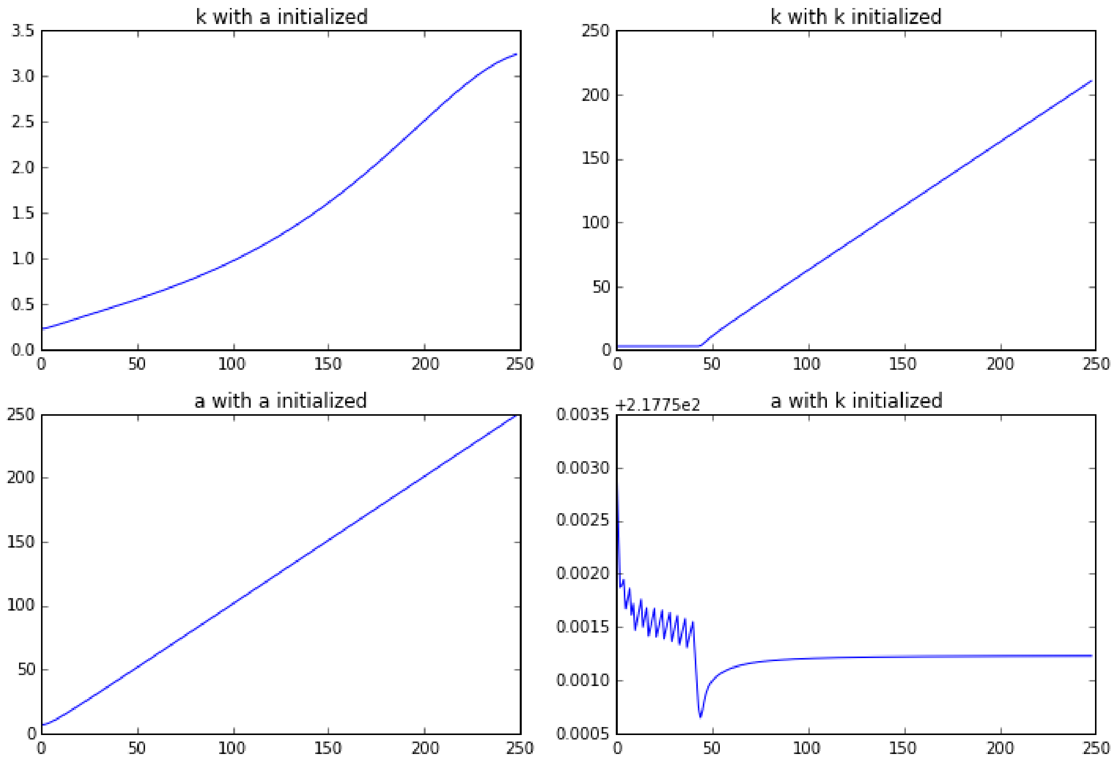
1.5.2 Initializing with different values of κ and α

```
In [59]: # initializing a with different values
         k_a = []
         a_a = []
         for a in range(1, 250):
             k_, a_, _, _ = fit_rex.fit_weibull_distribution(data, init_a=a)
             k_a.append(k_)
             a_a.append(a_)
```

```
In [60]: # initializing k with different values
         k_k = []
         a_k = []
         for k in range(1, 250):
             k_, a_, _, _ = fit_rex.fit_weibull_distribution(data, init_k=k)
             k_k.append(k_)
             a_k.append(a_)
```

```
In [61]: fig = plt.figure(figsize=(12, 8))
         s = 220

         for i, tv in enumerate(zip(["k with a initialized", "k with k initialized",
                                     "a with a initialized", "a with k initialized"],
                                    [k_a, k_k, a_a, a_k])):
             s += 1
             axs = fig.add_subplot(s)
```

```
axs.plot(tv[1])
axs.set_title(tv[0])
```



1.5.3 Shouldn't the lines be horizontal?

- But again, this approach (a type of Gradient Descent (?)) is known to get "stuck" in local minima
  - and get severely impacted by the initializations

1.5.4 But there's another weird observation

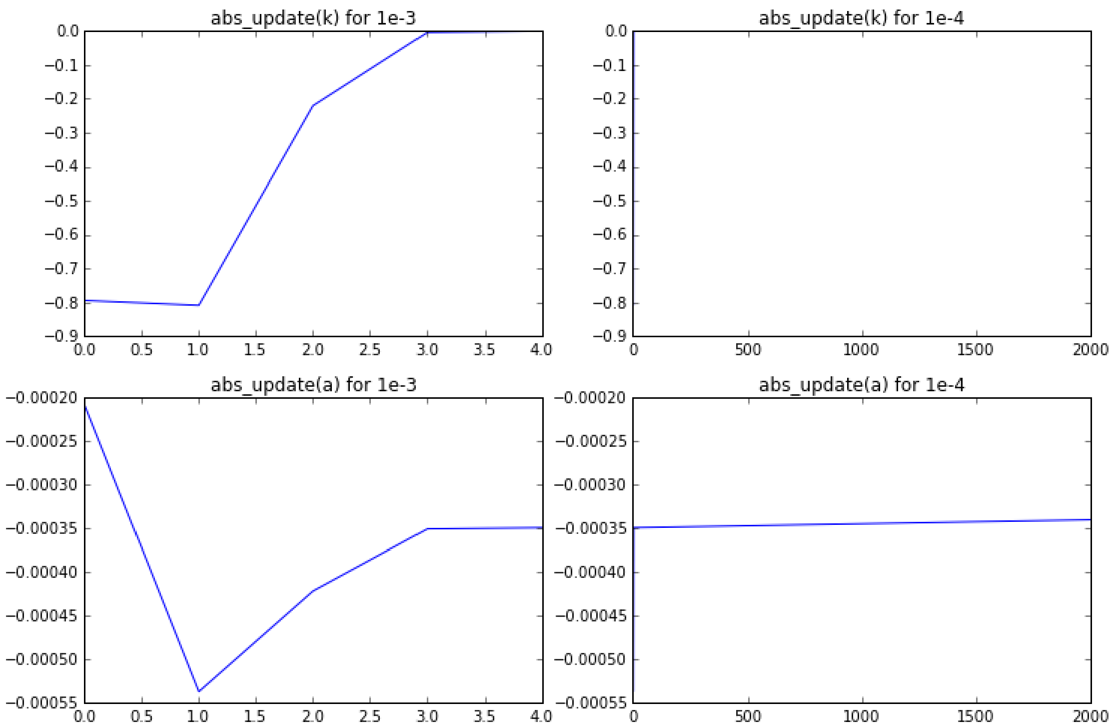
```
In [62]: _, _, _, _, errors_e3 = fit_rex.fit_weibull_distribution(data, n_iter=2000,
                                                                   change_thresh=1e-3,
                                                                  return_err=True)


         e_k_e3 = np.array([e[0] for e in errors_e3])
         e_a_e3 = np.array([e[1] for e in errors_e3])
```

```
In [63]: _, _, _, _, errors_e4 = fit_rex.fit_weibull_distribution(data, n_iter=2000,
                                                                   change_thresh=1e-4,
                                                                  return_err=True)


         e_k_e4 = np.array([e[0] for e in errors_e4])
         e_a_e4 = np.array([e[1] for e in errors_e4])
```

```
In [64]: print("set max iterations = 2000")
```

```python
fig = plt.figure(figsize=(12, 8))
s = 220

for i, tv in enumerate(zip(["abs_update(k) for 1e-3", "abs_update(k) for 1e-4",
                            "abs_update(a) for 1e-3", "abs_update(a) for 1e-4"],
                           [e_k_e3, e_k_e4, e_a_e3, e_a_e4])):
    s += 1
    axs = fig.add_subplot(s)
    axs.plot(tv[1])
    axs.set_title(tv[0])
```

```
set max iterations = 2000
```



1.5.5 Here...

and during further experiments we see that :

- κ zig-zigs and stabilizes quickly
- α updates at a snail's pace relative to it's own value
- **Both** behave weirdly for different initializations

1.5.6 Also...

- Our implementation might have a problem that we were not able to find in >4 complete re-writes
- `scipy.stats.exponweib` was able to handle a lot many cases where our implementation started facing numerical problems.

- for example, generating fake data in range $(0 \ldots 1)$

- but we do get a *not-that-bad* solution

1.6 Task 1.4

1.6.1 Drawing L^p Unit Circles in $\mathbb{R}^2$

1.6.2 L^p norm for $\mathbb{R}^m$

We use the following definition for m-dimensional x:

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

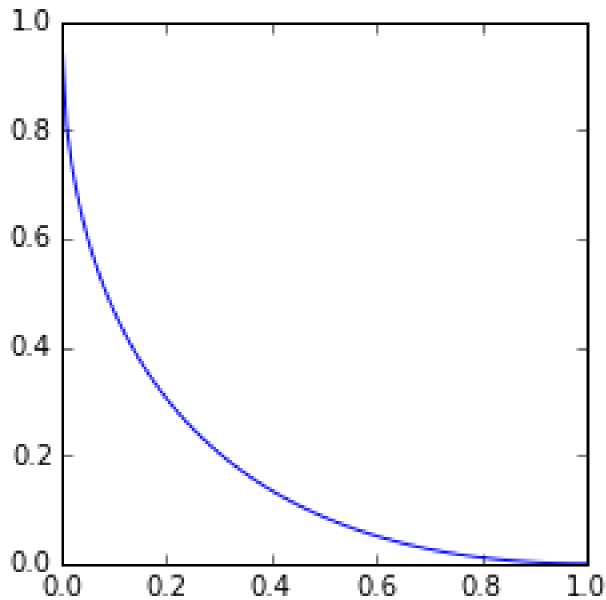
For $\mathbb{R}^2$,

- We choose 100 values for x_1 in range [0...1]
  - We exploit the symmetry of unit circles
- Then find corresponding x_2 as :

$$x_2 = (1 - x_1^p)^{\frac{1}{p}}$$

  - since we are going to cheat, we only use one solution for x_2

In [77]:
```
fig = plt.figure(figsize=(12, 8))
subplot_num = 230
for p in [0.5, 1, 2, 3, 4]:

    x, y = uc_rex.lp_unit_circle(p=p)

    subplot_num += 1
    ax = fig.add_subplot(subplot_num)
    plt_rex.plot2d(np.vstack((x, y)), colwise_data=True, axs=ax, hatch='-')
    plt_rex.plot2d(np.vstack((-x, y)), colwise_data=True, axs=ax, hatch='-')
    plt_rex.plot2d(np.vstack((x, -y)), colwise_data=True, axs=ax, hatch='-')
    plt_rex.plot2d(np.vstack((-x, -y)), colwise_data=True, axs=ax,
                   x_lim=[-1.2, 1.2], y_lim=[-1.2, 1.2],
                   set_aspect_equal=True, show=False,
                   show_axes_through_origin=True,
                   title="p=%.1f" % (p), hatch='-')
```



1.6.3 But, there is one more way to do this

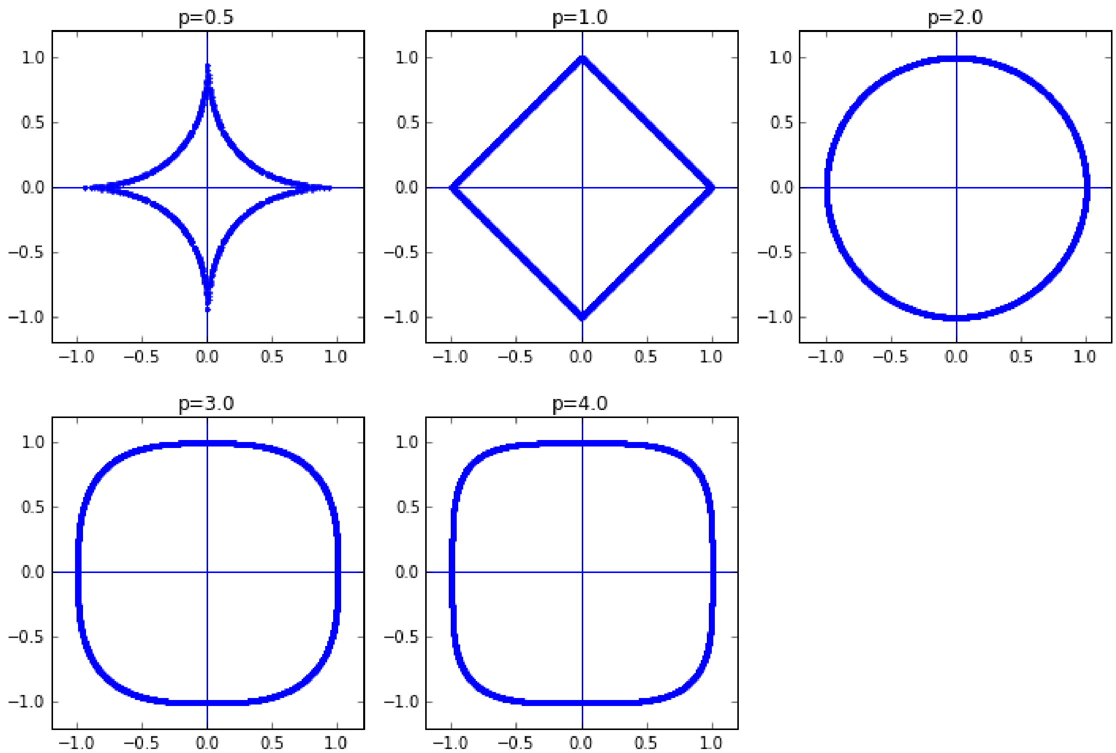
We can take the *Sieve Approach*

1. Calculate the p-norm for **all*** points in the [-1. . . 1] square
   - we used `numpy.linalg.norm(...)`
   - and some indexing tricks
2. Filter out where the value is 1

```
In [66]: x = np.linspace(-1, 1, 1000)  # not so smart
         y = np.linspace(-1, 1, 1000)

         # for convenience
         m = np.meshgrid(x, y)

         fig = plt.figure(figsize=(12, 8))
         subplot_num = 230
         for p in [0.5, 1, 2, 3, 4]:

             # calculate p-norms for **all** points
             norms = np.linalg.norm(m, axis=0, ord=p)

             # find where p-norm is 1, with some tolerance
             xi, yi = np.where(np.isclose(norms, 1, atol=0.001))

             print("p :", p, "; points", xi.size, "(", 100*xi.size/1000000, "% )")

             subplot_num += 1
             ax = fig.add_subplot(subplot_num)
```

```
            plt_rex.plot2d(np.vstack((x[xi], y[yi])), colwise_data=True, axs=ax,
                           x_lim=[-1.2, 1.2], y_lim=[-1.2, 1.2],
                           set_aspect_equal=True, show=False,
                           show_axes_through_origin=True,
                           title="p=%.1f" % (p), hatch='.')
```

```
p : 0.5 ; points 688 ( 0.0688 % )
p : 1 ; points 3996 ( 0.3996 % )
p : 2 ; points 3196 ( 0.3196 % )
p : 3 ; points 3548 ( 0.3548 % )
p : 4 ; points 3728 ( 0.3728 % )
```



This method is *extremely* wasteful And, ***some plots will just not appear*** (when an appropriate number of points to sieve through is not chosen)

But

- All we needed was a function that calculates the desired norms
- No pen-paper required to find the *solutions* first
- Very quick and easy to reason about (if you don't crash your machine)

1.6.4 Using that

1.7 Bonus Task

1.7.1 Drawing Aitchison Unit Circles in $\mathbb{S}^3$

1.7.2 Aitchison Norm

Norm of $\mathbf{x} \in \mathbb{S}^D$ is defined as:

$$\|\mathbf{x}\|_a = \sqrt{\frac{1}{2D}\sum_{i=1}^{D}\sum_{j=1}^{D}\left(ln\frac{x_i}{x_j}\right)^2}$$

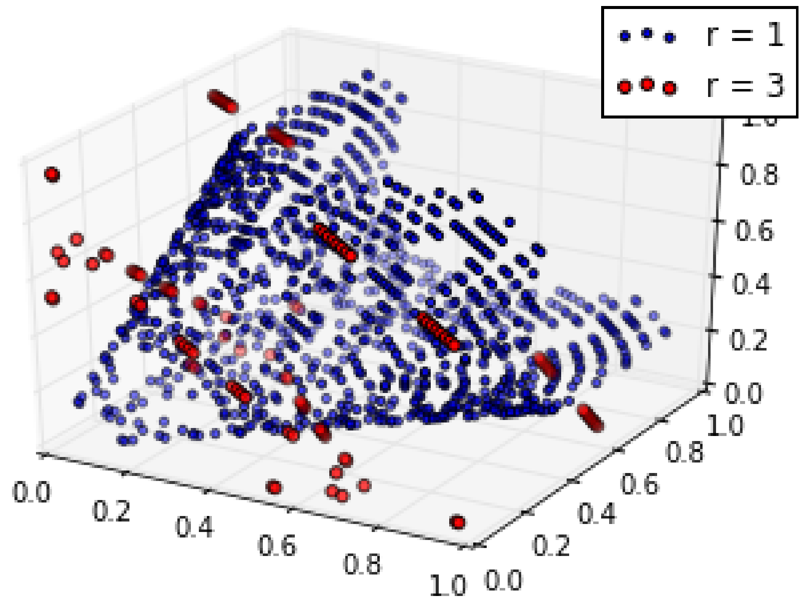
In practice, there is an alternative that can be used,

$$\|\mathbf{x}\|_a = \sqrt{\sum_{i=1}^{D}\left(ln(x_i) - ln(g_m(\mathbf{x}))\right)^2}$$

where $g_m(\cdot)$ is the geometric mean of the arguments
$ln(g_m(\cdot))$ can be replaced by mean of logs

In [67]:
```
### WARNING ### WARNING ### WARNING ###
# the slightest of changes to n can CRASH YOUR SYSTEM
n = 100

x = np.linspace(1e-9, 1, n)  # zero is not used due to log
xx, yy, zz = np.meshgrid(x, x, x)

# calculate all Aitchison Norms
norms = uc_rex.aitchison_norm((xx, yy, zz))

# find where norms is 1, 3
xyzi_1 = np.where(np.isclose(norms, 1, atol=0.001))
xyzi_3 = np.where(np.isclose(norms, 3, atol=0.001))

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.scatter3D(xx[xyzi_1], yy[xyzi_1], zz[xyzi_1], c='b', s=10)
ax.scatter3D(xx[xyzi_3], yy[xyzi_3], zz[xyzi_3], c='r')
ax.set_zlim(0, 1)
ax.set_ylim(0, 1)
ax.set_xlim(0, 1)
ax.legend(["r = 1", "r = 3"])
```

Out[67]:
```
<matplotlib.legend.Legend at 0x125ab2320>
```

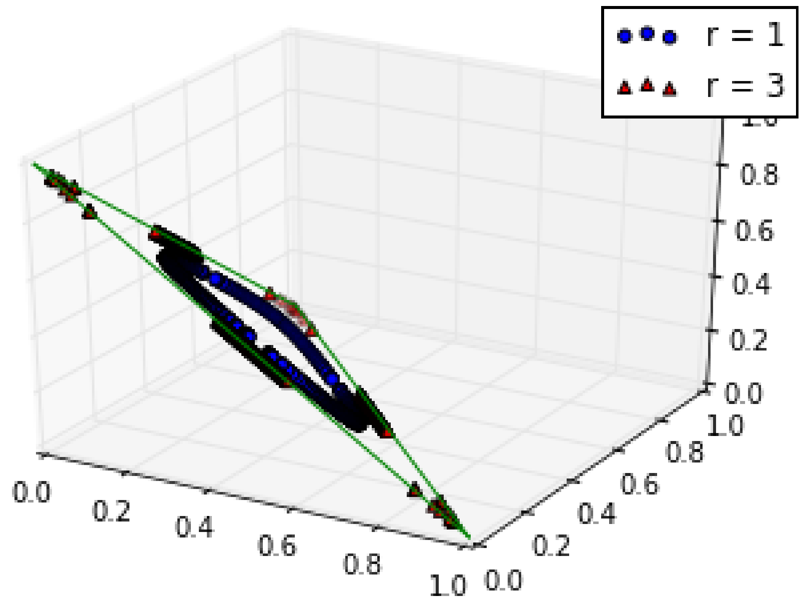


We can be a bit smarter and do something like this

```
In [68]: x = np.linspace(1e-9, 1-(1e-9), 1200)
         y = (1 - 1e-8) - x
         xx, yy = np.meshgrid(x, y)
         zz = (1 - 1e-7) - xx - yy
         # weird numbers above for numerical stability  # doesn't work

         # calculate all Aitchison Norms
         norms = uc_rex.aitchison_norm((xx, yy, zz))

         # find where norms is 1, 3
         xyzi_1 = np.where(np.isclose(norms, 1, atol=0.001))
         xyzi_3 = np.where(np.isclose(norms, 3, atol=0.001))

         fig = plt.figure()
         ax = plt.axes(projection='3d')
         ax.scatter3D(xx[xyzi_1], yy[xyzi_1], zz[xyzi_1], c='b', marker='o')
         ax.scatter3D(xx[xyzi_3], yy[xyzi_3], zz[xyzi_3], c='r', marker='^')
         ax.set_zlim(0, 1)
         ax.set_ylim(0, 1)
         ax.set_xlim(0, 1)
         ax.legend(["r = 1", "r = 3"])
         ax.plot(np.linspace(0, 1, 100), np.linspace(1, 0, 100), np.zeros(100), 'g')
         ax.plot(np.zeros(100), np.linspace(0, 1, 100), np.linspace(1, 0, 100), 'g')
         ax.plot(np.linspace(1, 0, 100), np.zeros(100), np.linspace(0, 1, 100), 'g')
```

```
/Users/myrmidon/Delve/studies-mi/patt-rex/pattrex/unit_circles.py:18: RuntimeWarning: invalid value enco
  lxyz = np.log(xyz)
```

```
Out[68]: [<mpl_toolkits.mplot3d.art3d.Line3D at 0x121c6da58>]
```

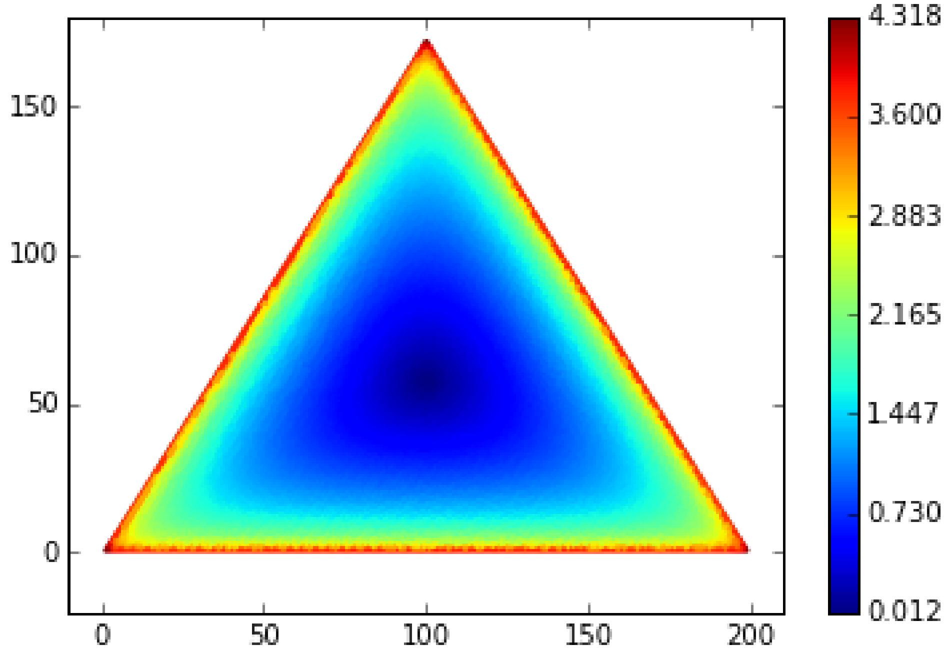


The *Sieve Approach* at least lifted the mystery

- it was very easy, and quick to code
  - Safety? (no comments)
- **but**, we're pretty sure this is not the best way to do it

- We found a library for Python https://github.com/marcharper/python-ternary (Python Ternary)
- And were able to plotted a heatmap of the Aichison Norm

In [69]:
```
import ternary
figure, tax = ternary.figure(scale=200)
tax.heatmapf(uc_rex.aitchison_norm, boundary=False, style='hexagonal')
```



1.7.3 We have not been able to do it the *right* way, yet

- We understand that we will have to do some projection mapping
  - Something we were not able to figure out, yet
- But also,
  - We believe it will hide the true beauty

1.8 References

- The slides of course
- scipy.org
- Pawlowsky-Glahn, Egozcue, Tolosana-Delgado, “Lecture Notes on Compositional Data Analysis”, 2007; http://www.sediment.uni-goettingen.de/staff/tolosana/extra/CoDa.pdf ***

1.9 Questions?