Exercise Sheet 11

Exercise 2

Read in the data and have a look at it

In [46]:

```
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
%matplotlib inline

data = pd.read_csv('data/iris.data', sep = ',', header = None)
data = data.rename(columns = {0: 'sepal_length',
                              1: 'sepal_widgth',
                              2: 'petal_length',
                              3: 'petal_widgth',
                              4: 'class',
                                })
data
```

Out[46]:

	sepal_length	sepal_widgth	petal_length	petal_widgth	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

Visualize it using PCA, after finding an appropriate subspace from the eigen values and vectors.

In [47]:

```
X = data.iloc[:,:4].values
Y = data.iloc[:,4].values
color_map = {
    'Iris-setosa': 'green',
    'Iris-versicolor': 'red',
    'Iris-virginica': 'blue',
}
```

Using Eigendecomposition

At first we need to compute the eigenvalues and eigenvectors of the covariance matrix of X

In [48]:
```
covariance_matrix = np.cov(X.transpose())
L, V = np.linalg.eig(covariance_matrix)
print(L)
print(V)
```

```
[4.22484077 0.24224357 0.07852391 0.02368303]
[[ 0.36158968 -0.65653988 -0.58099728  0.31725455]
 [-0.08226889 -0.72971237  0.59641809 -0.32409435]
 [ 0.85657211  0.1757674   0.07252408 -0.47971899]
 [ 0.35884393  0.07470647  0.54906091  0.75112056]]
```

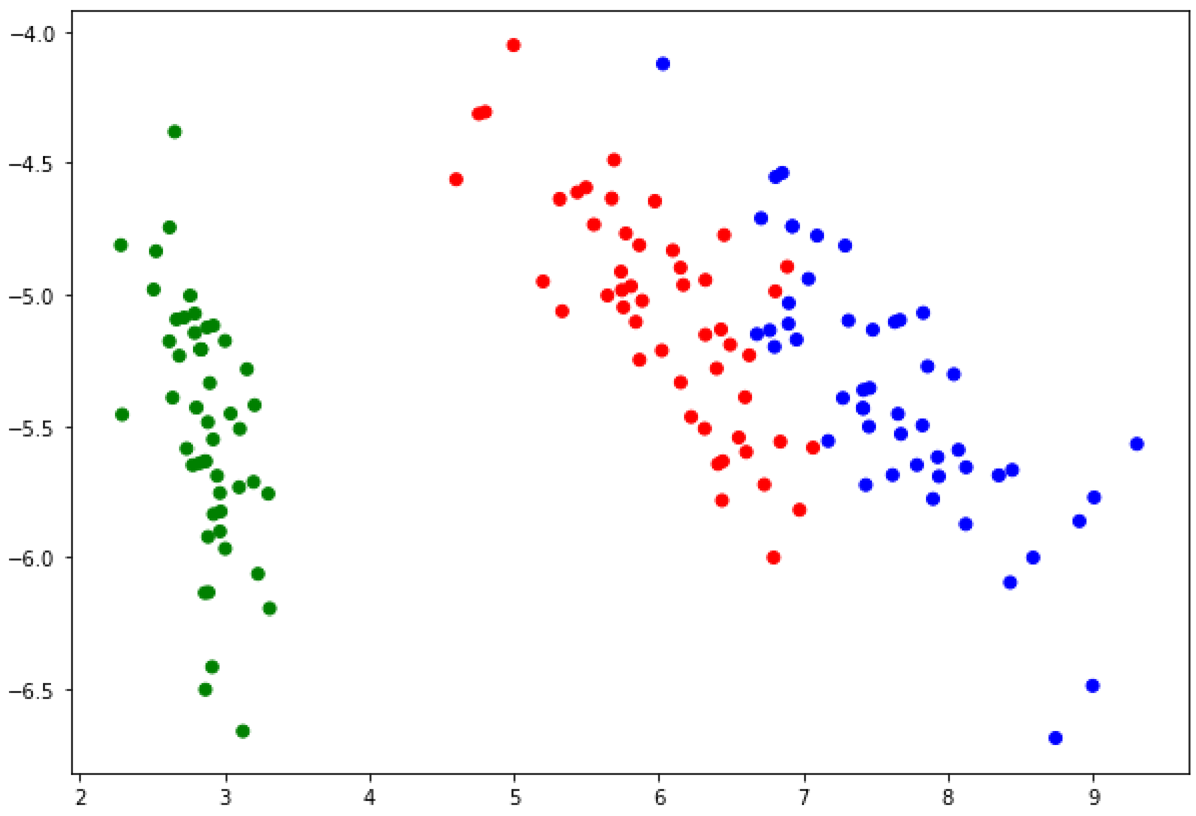
The first array are the sorted eigenvalues, the second is the Matrix V with the eigenvectors in the columns

In [49]:
```
V_2 = V[:, :2].copy()
V_2 = np.dot(V_2.transpose(),X.transpose()).transpose()
```

V_2 now contains the reduced dataset of X into the R^2. Therefore we can plot it and see wheter we can classify the 3 labels of Y, looking at the plot.

In [50]:
```
fig, ax = plt.subplots(1,1,figsize = (10,7))
ax.scatter(V_2[:,0], V_2[:,1] ,c = [color_map[i] for i in Y])
```

Out[50]:
```
<matplotlib.collections.PathCollection at 0x134b1f1f0>
```



Using Singular Value decomposition

deciding how many components:

here I am using Singular value decompositions, as I believe that to be the standard in PCA.

In [51]:
```
U, S, V = np.linalg.svd(X)
print(S)
```

```
[95.95066751 17.72295328  3.46929666  1.87891236]
```

S is an array 1x4 array with the Singular values, V is a 4x4 matrix with the right singular vectors and U is a 150x150 matrix with the left singular vectors.

Loosely spoken, the Singular values tell us how much information is in each of the columns of the matrix U.

In this case, the first transformed componend is with 95.95 significantly better than the others, yet 17.7 is still way better than 3.4.

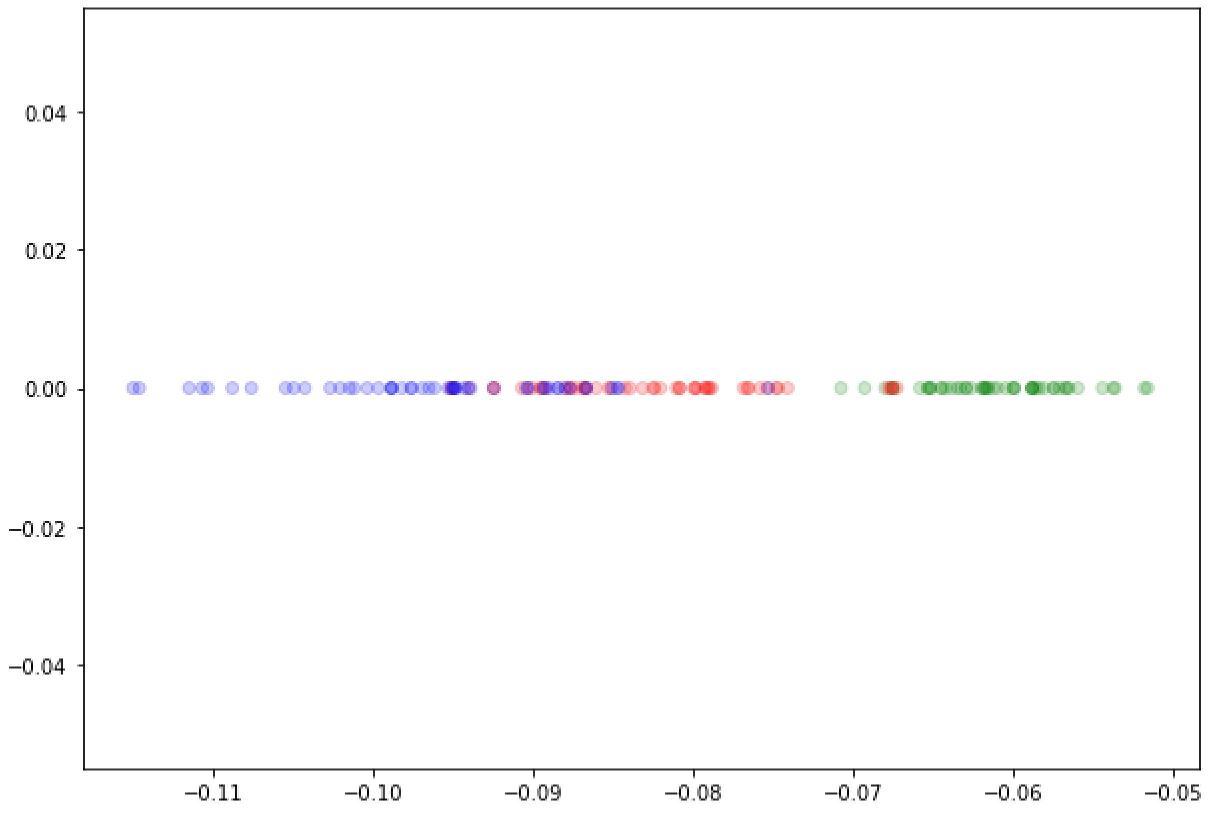
Therefore and **appropriate subspace would probably be 2**, although one could even argue to go into one dimension only. (That would only make sense for enourmous dataset, where the further processing takes ages, and dimensionality reduciton would yield massive performance boosts)

Test only the first component (K=1):

In [52]:
```
U_1 = U[:, :1].copy()
U_1 *= S[:1]
```

In [53]:
```
fig, ax = plt.subplots(1,1,figsize = (10,7))
ax.scatter(U[:,0], np.zeros(150),c = [color_map[i] for i in Y], alpha = 0.2)
```

Out[53]:
```
<matplotlib.collections.PathCollection at 0x134d37f40>
```



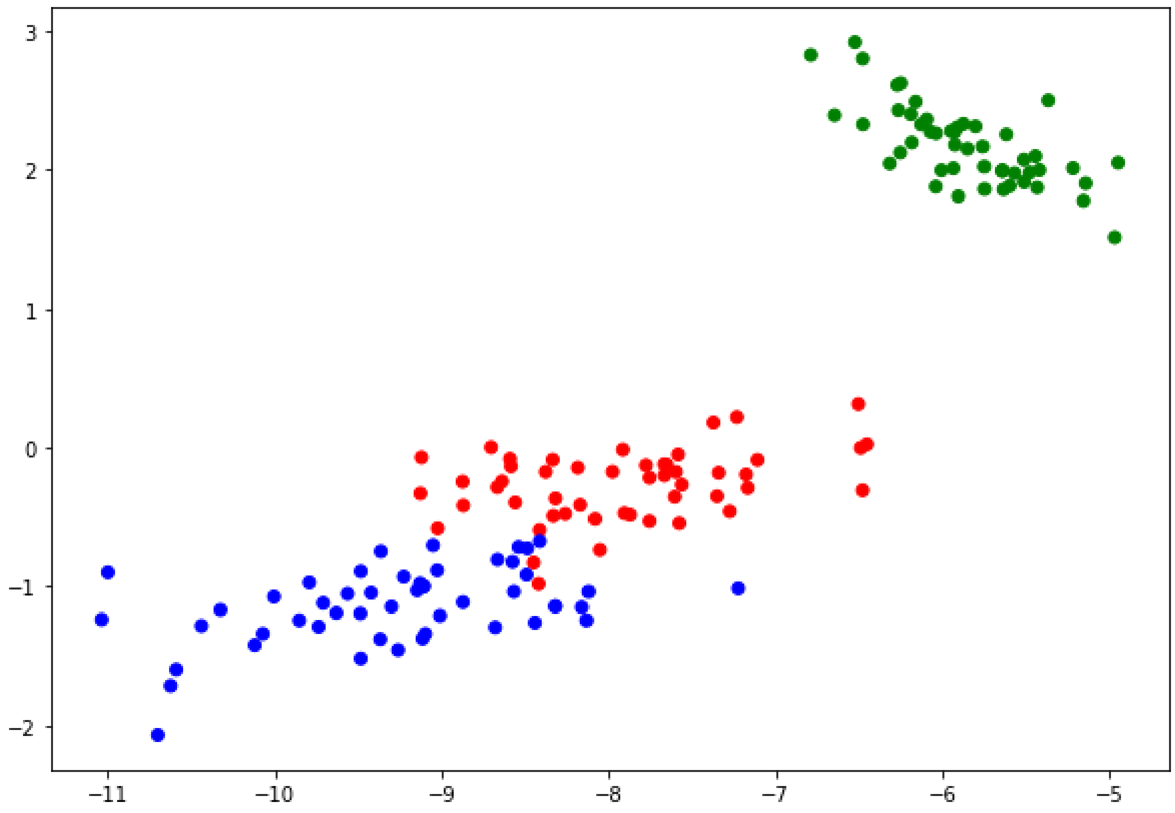
Here we can see visually, that the first copmonent is already a very good discriminator for the three classes. Lets see what happens when we increase the dimension by one.

Test for the first two components (K=2):

In [54]:
```
U_2 = U[:, :2].copy()
U_2 *= S[:2]
```

In [55]:
```
fig, ax = plt.subplots(1,1,figsize = (10,7))
ax.scatter(U_2[:,0], U_2[:,1] ,c = [color_map[i] for i in Y])
```

Out[55]:
```
<matplotlib.collections.PathCollection at 0x10c58f730>
```



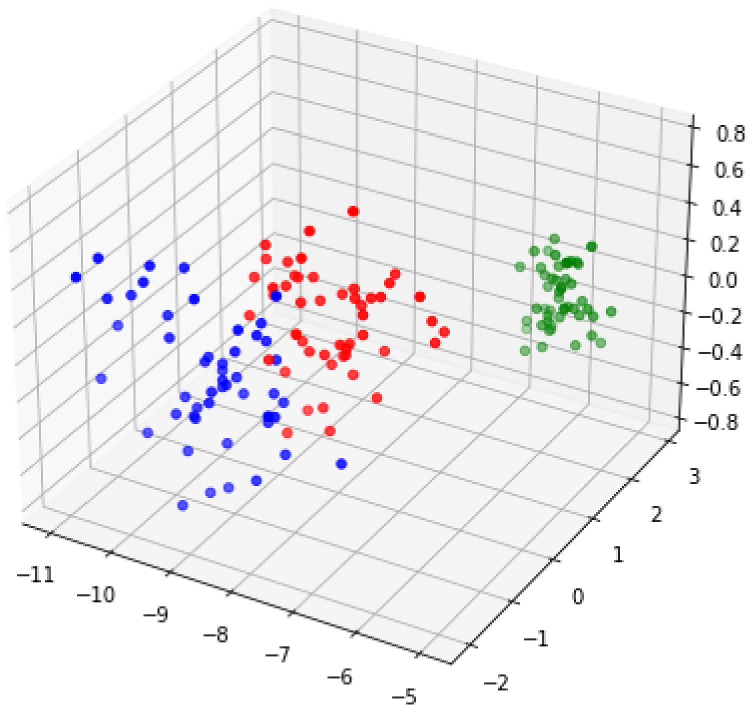
here he can see an even better separation of the three classes. Lastly lets look at the threedimensional Case.

Test only the first three components (K=3):

In [56]:
```
U_3 = U[:, :3].copy()
U_3 *= S[:3]
```

In [57]:
```
fig = plt.figure(figsize = (10,7))
ax = fig.gca(projection='3d')
ax.scatter(U_3[:,0], U_3[:,1], U_3[:,2] ,c = [color_map[i] for i in Y])
```

Out[57]:
```
<mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x134e6c370>
```



A third dension doesnt really add value for the classification task. The discrimnation of not better as in the 2D Case.

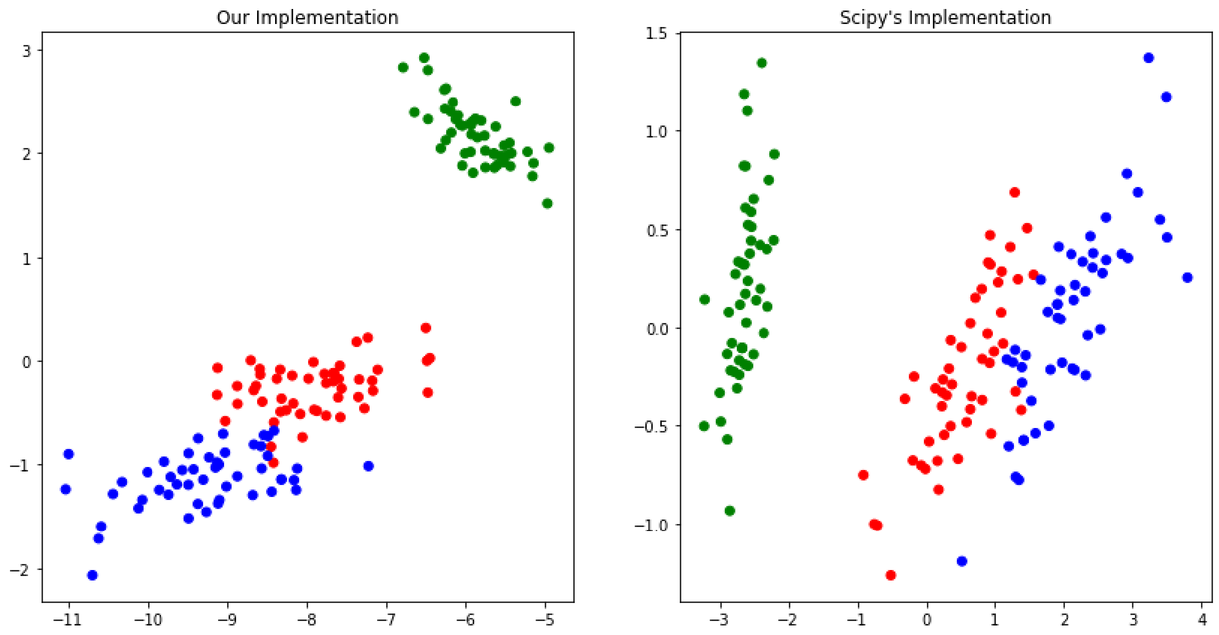
Using the SciPy library: (out of curiosity and comparison)

In [58]:
```
pca = PCA(n_components = 2).fit(X)
print(pca.transform(X)[:5])
print(U_2[:5])
```

```
[[-2.68420713  0.32660731]
 [-2.71539062 -0.16955685]
 [-2.88981954 -0.13734561]
 [-2.7464372  -0.31112432]
 [-2.72859298  0.33392456]]
[[-5.91220352  2.30344211]
 [-5.57207573  1.97383104]
 [-5.4464847   2.09653267]
 [-5.43601924  1.87168085]
 [-5.87506555  2.32934799]]
```

In [59]:
```
fig, ax = plt.subplots(1,2,figsize = (14,7))
ax[0].scatter(U_2[:,0], U_2[:,1] ,c = [color_map[i] for i in Y])
ax[1].scatter(pca.transform(X)[:,0], pca.transform(X)[:,1] ,c = [color_map[i] for i
ax[0].set_title('Our Implementation')
ax[1].set_title('Scipy\'s Implementation')
```

Out[59]:
```
Text(0.5, 1.0, "Scipy's Implementation")
```



We can see, that these values divere, despite clear clusteres emerging in both versions. Reason for that is, that in the scipy funciton, the features get centered around 0. When we also do this, we should get the same result.

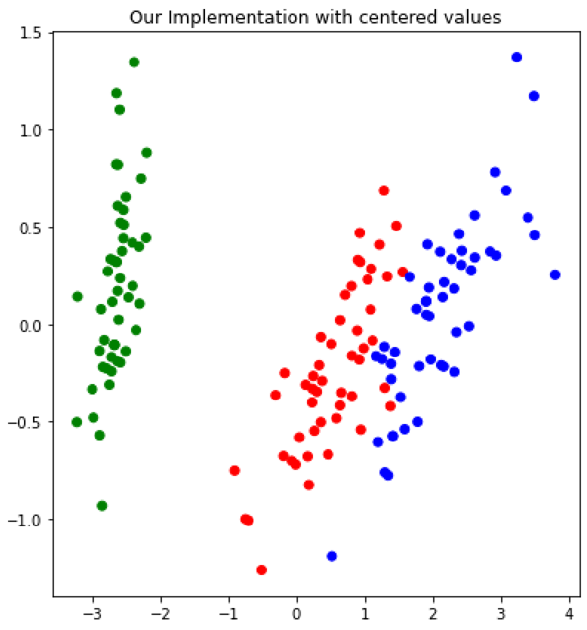
In [60]:
```
mean_ = np.mean(X, axis=0)
X_norm = X - mean_
U_2_norm, S_2_norm, V_2_norm  = np.linalg.svd(X_norm)
U_2_norm = U_2_norm[:, :2]
U_2_norm *= S_2_norm[:2]
print(pca.transform(X)[:5])
print(U_2_norm[:5])
```

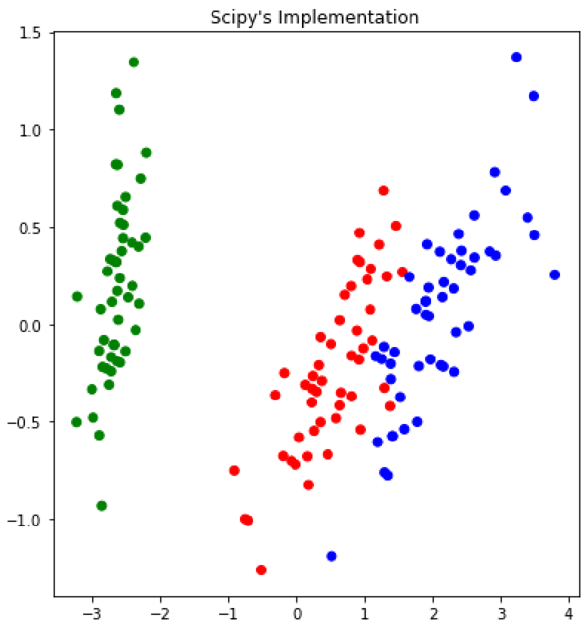
```
[[-2.68420713  0.32660731]
 [-2.71539062 -0.16955685]
 [-2.88981954 -0.13734561]
 [-2.7464372  -0.31112432]
 [-2.72859298  0.33392456]]
[[-2.68420713 -0.32660731]
 [-2.71539062  0.16955685]
 [-2.88981954  0.13734561]
 [-2.7464372   0.31112432]
 [-2.72859298 -0.33392456]]
```

Interestingly, the second feature is multiplied by -1, compared to the componets, we calculated before.

In [61]:
```
fig, ax = plt.subplots(1,2,figsize = (14,7))
ax[0].scatter(U_2_norm[:,0], -1 * U_2_norm[:,1] ,c = [color_map[i] for i in Y])
ax[1].scatter(pca.transform(X)[:,0], pca.transform(X)[:,1] ,c = [color_map[i] for i
ax[0].set_title('Our Implementation with centered values')
ax[1].set_title('Scipy\'s Implementation')
```

Out[61]:
```
Text(0.5, 1.0, "Scipy's Implementation")
```





Now the values are aligned, which is what we assumed. Also, the discrimitation of the three casses with uncentered data is just as good as with centered data, at least for this example.

It is likely that this is not the general Case, especially with Dimensions larger than 10.