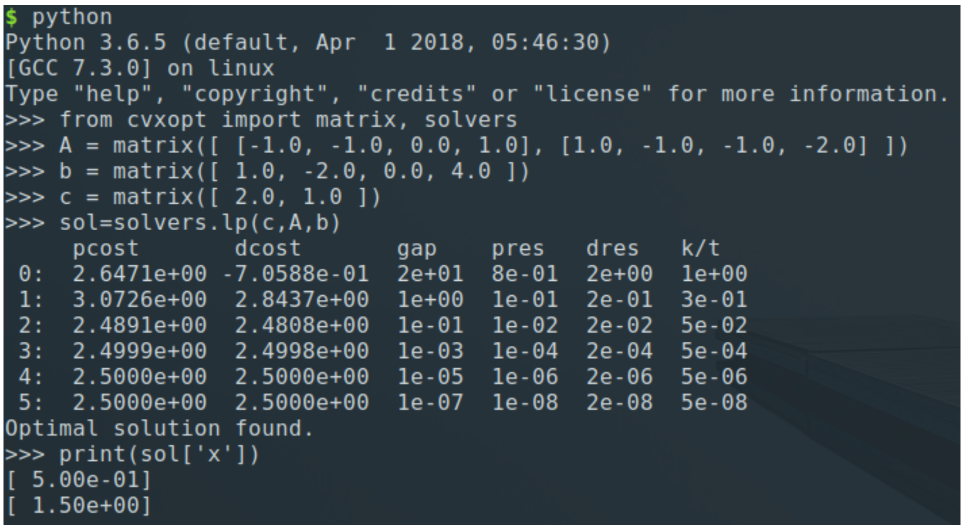
EE 127 Optimization Models in Engineering
Fall 2018 Nithin Raghavan

HW 1

1 CVX Installation



```
$ python
Python 3.6.5 (default, Apr  1 2018, 05:46:30)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from cvxopt import matrix, solvers
>>> A = matrix([ [-1.0, -1.0, 0.0, 1.0], [1.0, -1.0, -1.0, -2.0] ])
>>> b = matrix([ 1.0, -2.0, 0.0, 4.0 ])
>>> c = matrix([ 2.0, 1.0 ])
>>> sol=solvers.lp(c,A,b)
     pcost       dcost       gap    pres   dres   k/t
 0:  2.6471e+00 -7.0588e-01  2e+01  8e-01  2e+00  1e+00
 1:  3.0726e+00  2.8437e+00  1e+00  1e-01  2e-01  3e-01
 2:  2.4891e+00  2.4808e+00  1e-01  1e-02  2e-02  5e-02
 3:  2.4999e+00  2.4998e+00  1e-03  1e-04  2e-04  5e-04
 4:  2.5000e+00  2.5000e+00  1e-05  1e-06  2e-06  5e-06
 5:  2.5000e+00  2.5000e+00  1e-07  1e-08  2e-08  5e-08
Optimal solution found.
>>> print(sol['x'])
[ 5.00e-01]
[ 1.50e+00]
```

2 About general optimization

1. True. This is because rather than considering $\min_{x\in\mathbb{R}^n} f_0(x)$, we can let $f_0(x) \le C$ for some arbitrary C and add it to the list of f_i, so the new problem is $p^* = \min_{x\in\mathbb{R}^n} c : f_0(x) \le c, f_i(x) \le 0$.

2. True. This is just the reverse of the previous part; first, modify $f_0(x)$ so it is limited in the given constraints, piecewise. Then, let
$$g(x) = \begin{cases} f_0(x) \le & \text{if } \exists i, f_i(x) > 0 \\ \varnothing & \text{if } \nexists i, f_i(x) > 0. \end{cases}$$
Then, the problem can be rewritten as $p^* = \min_{x\in\mathbb{R}^n} g(x)$

3. True. Polynomials of higher degree can be considered vectors, which can be orthogonalized using Gram-Schmidt. Using a Taylor series expansion, any nonlinear constraint $f_i(x) \le 0$ can be divided up into an infinite amount of orthogonal vectors $\sum_{i=0}^{\infty} \frac{f_i^n(a)}{n!}(x-a)^n$ which each constrain the original problem linearly.

4. False. Consider, as a counterexample, the problem $\min_x x : 0 < x \le 1$. Removing the strict inequality at 0 leads to the problem $\min_x x : x \le 1$ which has its solution at $x = -\infty$, which is not allowed in the domain of the original problem.

5. True. Let $\arg\min_x \min_y F_0(x,y) = x^*$. By definition, $\min_y F_0(x^*, y) \le \min_y F_0(x,y)$ for all x. So, $\min_y F_0(x^*,y) \le \min_y \min_x F_0(x,y)$. Repeating this same argument with y^* gives $\forall y, \min_x F_0(x,y^*) \le \min_x F_0(x,y) \implies \min_x F_0(x,y^*) \le \min_x \min_y F_0(x,y)$. Because $\min_x \min_y F_0(x,y) \le \min_y \min_x F_0(x,y)$ and $\min_x \min_y F_0(x,y) \ge \min_y \min_x F_0(x,y)$, then $\min_x \min_y F_0(x,y) = \min_y \min_x F_0(x,y)$.

6. True. Consider the functions $f(x) = \max_{y'} F_0(x,y')$, and similarly, $g(y) = \min_{x'} F_0(x',y)$. Then, $p^* = \min_x \max_y F_0(x,y) = \min_x f(x)$ and $d^* = \max_y \min_x F_0(x,y) = \max_y g(y)$ by construction. By definition, $\forall y, f(x) \ge F_0(x,y)$ and $\forall x, g(y) \le F_0(x,y)$. Thus, $f(x) \ge g(y) \implies p^* \ge d^*$.

3 1D Convolution

1. The convolution can be rewritten as $y = h * x \implies y_i = \sum_{j+k=i+1} h_j x_k$. $i+1$ is bounded along $i+1 \leq n+m$ because of the upper bounds of the finite vector x (at n) and h (at m). Because $x_t = 0 : t < 1$, then the convolution for $j < 1$ and $k < 1$ will return 0, and so we can conclude that a is lower than the lowest bound of i: $a < 1 \implies a = 0$. Next, because $x_t = 0 : t > n$, then the convolution for $j + k > n + m$ will also return 0, and we can similarly conclude that b is greater than the highest bound of i: $b \geq n + m - 1 \implies b = n + m - 1$. The output sequence can therefore be represented by a $(b-a)$-dimensional vector, or $n+m-1$ dimensional vector.

2. $y = T(h) \cdot x$, with $T(h)$ dimensions of $(n+m-1) \times n$. An increase in the jth column corresponds to an increase in the index of x, which is compensated by a decrease in the index of h by the convolution equation given in part (a) (so for $j = 1$, $T(h)_{0,0} = h_1$, but this must be shifted downward with an increase in j, corresponding to an increase in x, because $i + j$ must equal a constant). Thus, $T(h)_{i,j} = h_{i-(j-1)}$ if $1 \leq i-(j-1) \leq m$. Because convolution is zero outside the band, then for all $i-(j-1)$ that do not satisfy this, the value of $T(h)_{i,j} = 0$. As an example, $n = m = 3$ gives $\begin{bmatrix} h_1 & 0 & 0 \\ h_2 & h_1 & 0 \\ h_3 & h_2 & h_1 \\ 0 & h_3 & h_2 \\ 0 & 0 & h_3 \end{bmatrix}$. The matrix has the structure that if an element h_i starts in a position on the first column, the element is maintained across its diagonal. The relationship between y and x becomes such that the convolution of h and x reduces to the matrix multiplication of h and x to produce y.

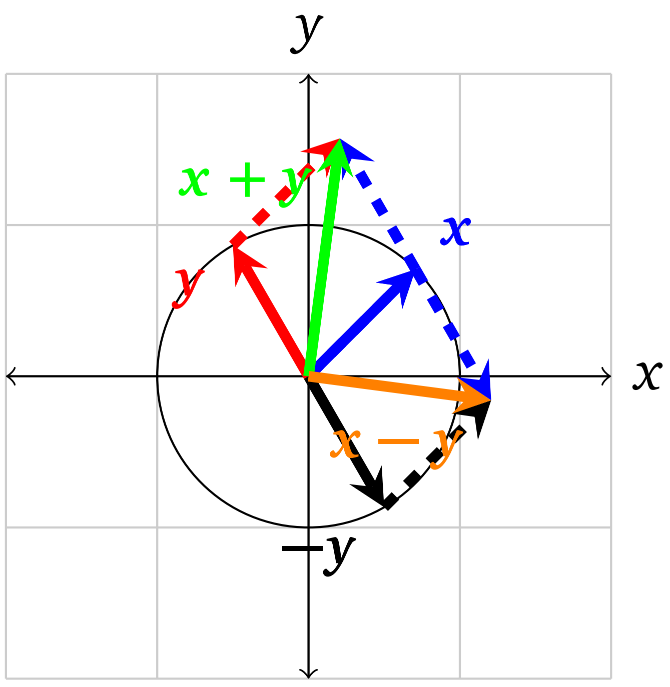
3. $p(s)q(s) = (p_0 s^0 + \cdots + p_n s^n)(q_0 s^0 + \cdots + q_n s^n) = p_0 q_0 + (p_0 q_1 + q_1 p_0)s + (p_0 q_2 + p_1 q_1 + p_2 q_0)s^2 + \cdots + p_n q_n s^{2n} = \sum_{i=0}^{n} \sum_{j=0}^{n} p_i q_j s^{i+j}$. Here we set $k = i + j$ and let $\sum_{k=0}^{2n} \sum_{i=0}^{k} p_i q_{k-i} x^k = (p * q)(s)$. Looking at the convolution, the coefficient of each x^k, the coefficient is the sum of the products that each result in a degree k. The amount of coefficients reach the maximum at $k = \frac{n}{2}$ and have their minimum at $k = 0$ and $k = 2n$, with the frequency of the

lower coefficients decreasing after $k = n$, and the frequency of the higher coefficents increasing after $k = n$. This can thus be written with a Toeplitz matrix as $p(s)q(s) = \left(\underbrace{\begin{bmatrix} p_0 & 0 & 0 & \cdots & 0 \\ p_1 & p_0 & 0 & \vdots & \vdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ p_{n-1} & p_{n-2} & \cdots & \ddots & p_0 \\ p_n & p_{n-1} & \cdots & \ddots & p_1 \\ 0 & p_n & \cdots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & p_n \end{bmatrix}}_{(2n-1)\times n} \underbrace{\begin{bmatrix} q_0 \\ \vdots \\ q_n \end{bmatrix}}_{n\times 1} \right)^T \begin{bmatrix} s^0 \\ \vdots \\ s^{2n} \end{bmatrix}$.

4. The equation $h = g * r$ can be rewritten for each component i of h to obtain $h_i = \sum_{j=1}^{4} r_{i-j+1} g_j$. This means that the current value of h at i is a linear combination of $r_i, r_{i-1}, r_{i-2}, r_{i-3}$, with coefficients g_1, g_2, g_3, g_4. So, g_3 can be interpreted as the factor by which the amount of rain at an arbitrary i will affect the river height 2 days into the future. So, if it is assumed that there is only one day of rain, then for the other days, $r_i = 0$, thus, the river height will be the most affected 2 days after the rainfall, because the coefficient of g is maximum. And because it takes 4 days for the input to stop affecting the output, then it takes 4 days for the river to return to its normal height.

4 Norm and angles

1. $||x||_2 = ||y||_2 = 1$. By the distributive property, $(x-y)\cdot(x+y) = (x-y)\cdot x + (x-y)\cdot y = x\cdot x - y\cdot x\cdot y - y\cdot y = 1-1=0$, so they are orthogonal. Drawing them out results in the following, using the parallelogram rule:



These form a parallelogram because the angle formed by the blue and red lines is supplementary to the angle formed by the blue and black lines. So, the angle formed by the green and dotted blue line is complementary to the angle formed by the orange and dotted blue line by the vertical angle theorem.

2. Let $\text{card}(x) = n \geq 1$. Then, $||x||_1 = \sum_{i=1}^{n} |x_i|$. From here, Cauchy-Schwarz implies $\sum_{i=1}^{n} |x_i| \leq \left(\sum_{i=1}^{n} |x_i|\right)^{1/2} \left(\sum_{i=1}^{n} 1\right)^{1/2} = ||x||_2\sqrt{n} \implies ||x||_1 \leq \sqrt{n}||x||_2$. This further implies $\frac{1}{n}||x||_1 \leq \frac{1}{\sqrt{n}}||x||_2 \leq ||x||_2$. Similarly, $||x||^2 = \sum_{i=1}^{n} x_i^2 \leq n \max_{1\leq i\leq n} x_i^2 = n\left(\max_{1\leq i\leq n} |x_i|\right)^2 = n||x||_\infty^2 \implies ||x||_2 \leq$

$\sqrt{n}||x||_\infty \implies \sqrt{n}||x||_2 \le n||x||_\infty$. And additionally, because $||x||_2^2 = \sum_{i=1}^{n} x_i^2 \le \sum_{i=1}^{n} x_i^2 + 2\sum_{j\neq i}^{n} |x_i||x_j| = ||x||_1^2$, we finally have $\frac{1}{n}||x||_1 \le \frac{1}{\sqrt{n}}||x||_2 \le ||x||_\infty \le ||x||_2 \le ||x||_1 \le \sqrt{n}||x||_2 \le n||x||_\infty$.

3. Let $\text{card}(x) = n \ge 1$. We have $||x||_1^2 \le n||x||_2^2$ by the previous problem, and so $\frac{||x||_1^2}{||x||_2^2} \le n = \text{card}(x)$.

5 Comparing text

1. The frequency vectors are:

   (a): [1, 2, 0, 0, 0, 0]
   (b): [1, 0, 3, 2, 4, 0]
   (c): [0, 0, 0, 2, 0, 2]
   (d): [0, 0, 0, 0, 0, 1]

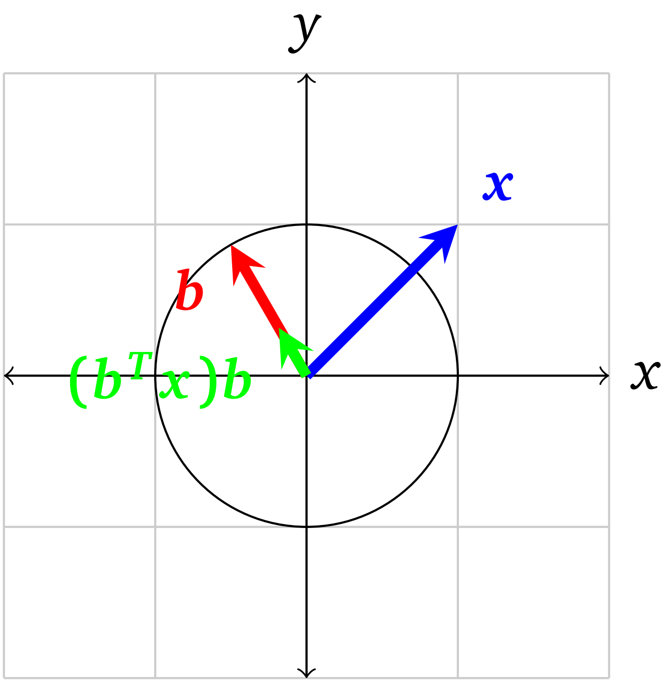
2. The cosine distance $\cos\theta = \dfrac{\langle a, b\rangle}{||a||_2||b||_2}$ between the following:

(a, a):	1
(a, b):	$\sqrt{6}/30$
(a, c):	0
(a, d):	0
(b, b):	1
(b, c):	$\sqrt{15}/15$
(b, d):	0
(c, c):	1
(c, d):	$\sqrt{2}/2$
(d, d):	1

6 Linear functions and projections

1. $f(x) = \begin{bmatrix} -1/n & -1/n & \cdots & 1-1/n & -1/n & \cdots & -1/n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \\ x_{m+1} \\ \vdots \\ x_n \end{bmatrix}$

2.



$b^T x = \begin{bmatrix} -1/2 & \sqrt{3}/2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{\sqrt{3}-1}{2}$. This is the scaling magnitude of the projection of x onto b.

3. PDF attached at the end of the HW. The idea is to dot the a-vectors with each row of the matrix X, the senator voting data. This results in four separate visualizations of the score that depend on the direction of the scoring function.

4. I would prefer a_3's direction, because it is centered at 0 and because it separates out the blue vs. red voting patterns much more than the other three directions, which are more muddled together.

7 Customer purchase history matrix

1. $C\mathbf{1}$ represents $\sum_{j=1}^{N} \begin{bmatrix} C_{1j} \\ \vdots \\ C_{Nj} \end{bmatrix}$ which is an N-vector, with each cell i containing the total number of dollar purchases made by the ith customer.

2. $C^T\mathbf{1}$ represents $\sum_{i=1}^{N} \begin{bmatrix} C_{i1} \\ \vdots \\ C_{iK} \end{bmatrix}$ which is a K-vector, with each cell i containing the total number of dollar purchases made of the ith category.

3. The sum of the total dollar amount is just $\underbrace{\begin{bmatrix} 1 & \cdots & 1 \end{bmatrix}}_{N} C\mathbf{1}_{K\times 1}$.

4. This means that the kth customer did not share any purchases with the ℓth customer, because the value of the dot product of the vectors C_k and C_ℓ is zero (so they are orthogonal).

5. Row centering C provides the equation $C_m w = \left(C - \frac{\overline{C\mathbf{1}}^T}{K} \right) w$
$= Cw - \frac{\overline{C\mathbf{1}}^T}{K} w$, where $\overline{C\mathbf{1}}$ represents a $K \times N$ matrix whose columns are all $C\mathbf{1}$. So, to efficiently compute Cw, then store all the nonzero values in 3 arrays that each contain the value of the nonzero elements (A), their row index (B), and their column index (C). Then, to perform the multiplication, create a zeroed-array called ans, and for each element $0 \le k < N$, then add to the $B[k]$th index of ans the result of $A[k] \times v_{C[k]}$. For the second operation, the only calculations that are required are the first row of $\overline{C\mathbf{1}}^T$ dotted with w, because all the rows of $\overline{C\mathbf{1}}$ are the same.

6. The naive method would have a runtime of $O(NK)$, while the runtime of my method would be $O(KNd)$; since $d < 1$, this results in a smaller coefficient and so a more efficient runtime.

7. My method has 26.6 times lesser (more efficent) runtime than the naive method.

8. If r_{new} represents the new row to be added to C, then first update the three arrays with the nonzero information contained in r_{new}. Next, by that method, multiply r_{new} by w, and then add it to the matrix Cw. By the equation for $C_m w$ in part (5), $\overline{C\mathbf{1}}w$ must therefore be recalculated, and then finally subtracted from the updated version of Cw. This is how to efficiently update the vector v.

Now we will focus on Senator Voting data. This data provides information about senator vote x and senator political affiliation y. We provide you with four different vectors (a_1, a_2, a_3, a_4) precomputed by the EECS127 staff. Each of these vectors can be used to define a linear function $f_a : x \rightarrow a^T x$.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
senator_df =  pd.read_csv('data_matrix.csv', index_col=0)
a_vectors = np.loadtxt('vectors.txt').reshape(4, 542)
affiliation_file = open("politician_labels.txt", "r")
affiliations = np.array([line.split('\n')[0].split(' ')[1] for line in affiliation_file.readlines()])
```

In [3]:
```
senator_df.head()
```

Out[3]:

	bill_type bill_name bill_ID	missing_votes	Arlen Specter (PA)	Barack H. Obama (IL)	BarbaraA Mikulski(MD)	BarbaraBoxer((
2	Appropriations_Transit Security Amendment_3866	0.0	1.0	1.0	1.0	
3	Budget_Spending_and_Taxes_2007 Budget Resoluti...	0.0	1.0	-1.0	-1.0	-
4	"Budget, Spending and Taxes_Debt Limit Increas...	0.0	1.0	-1.0	-1.0	-
5	"Budget, Spending and Taxes_Education Funding ...	0.0	-1.0	1.0	1.0	
6	Budget, Spending and Taxes_Reinstate Pay-As-Yo...	0.0	-1.0	1.0	1.0	

5 rows × 102 columns

Now that we have collected all the data, we will clean the senator voting data and convert it to numerical format

In [49]:
```
X = np.array(senator_df.values[:-1, 2:], dtype='float64')
```

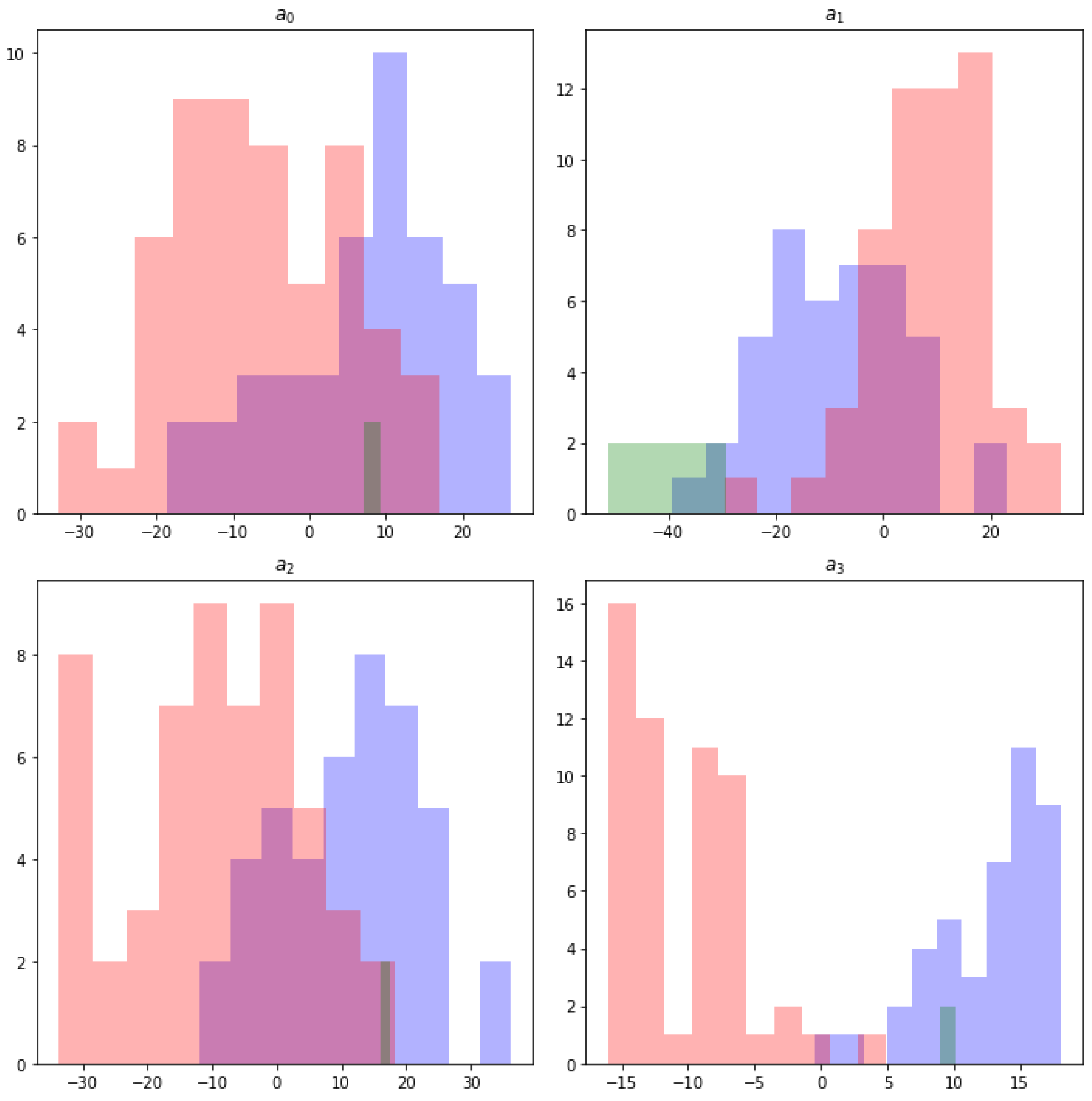
In [50]:
```
# TODO: Center the data matrix X by removing to each column its mean
X_bar = X.mean(axis=1)
for i in range(X.shape[1]):
   X[:, i] = X[:, i] - X_bar
len(X[0])
```

Out[50]:
```
100
```

In [51]:
```
# TODO: compute for each vector a the score of each senator
senator_scores = np.zeros(shape=(4, 100))
print(len(senator_scores), len(senator_scores[0]))
for i in range(4):
    senator_scores[i] = a_vectors[i].dot(X)
```

```
4 100
```

In [52]:
```
# Then we help you visualizing the scores with the library matplotlib
f, axarr = plt.subplots(2, 2,figsize=(10, 10))
for i in range(4):
    axarr[i // 2,  i % 2].hist(senator_scores[i, affiliations == "Blue"], color="Blue", alpha=0.3)
    axarr[i // 2,  i % 2].hist(senator_scores[i, affiliations == "Red"], color="Red", alpha=0.3)
    axarr[i // 2,  i % 2].hist(senator_scores[i, affiliations == "Yellow"], color="Green", bins=1, alpha=0
    axarr[i // 2,  i % 2].set_title(r'$a_'+ str(i)  +'$')
plt.tight_layout()
plt.show()
```



In [ ]:

Customer purchase history

We are now interested in efficient computations. In our setting, note that the data matrix C is large but very sparse. The number of non zero-valued elements divided by the total number of elements is called the density d of the matrix C. Let w in $\mathbb{R}^K$ be a given weighting vector. Assume that we center the rows (removing the average row to every row), obtaining a new row-centered matrix C_m.

In [1]:
```
import numpy as np
import time
from scipy.sparse import csr_matrix
```

In [2]:
```
lmbda = 0.1
K = 10000
N = 10000
dims = [N, K]
```

Generate sparse matrix using poisson distribution

In [3]:
```
# in dense matrix format, no performance improvement
C = np.random.poisson(lmbda, dims)

# in sparse matrix format, certain operations should be faster
C_sparse = csr_matrix(C)

# average of the rows of C
r_avg = np.mean(C, axis=0)

# w, in this case we use the one vector
w = np.ones(K)
```

In [4]:
```
print("The sparsity is around", C_sparse.count_nonzero() / N / K)
```

```
The sparsity is around 0.09521792
```

Naive implementation:

In [5]:
```
# The jupyter-notebook's magic commands, %t expr,
# will print the amount of time needed to evaluate expr
%time (C - r_avg) @ w
```

```
CPU times: user 682 ms, sys: 1.04 s, total: 1.73 s
Wall time: 5.22 s
```

Out[5]:
```
array([-28.5493,  -3.5493,  25.4507, ...,  48.4507, -21.5493,  -5.5493])
```

Efficient implementation:

In [6]:
```
# TODO: Implement your proposed procedure here to compute the desired quantity.
# Make sure you always get the same results as the naive implementation
%time C_sparse @ w - C_sparse.mean(axis=0) @ w
```

```
CPU times: user 83.2 ms, sys: 105 ms, total: 188 ms
Wall time: 196 ms
```

Out[6]:
```
matrix([[-28.5493,  -3.5493,  25.4507, ...,  48.4507, -21.5493,  -5.5493]])
```

In [ ]: