ECE367 PS1
Benjamin Cheng (1004838045)

Q1.10a
The pair with the smallest euclidean distance are 7 (James Forder) and 8 (Public image of George W. Bush) . This is different from the pair with the smallest angle are documents 9 (Barack Obama) and 10 (George W. Bush). Through visual inspection, article 7 did not similar to article 8 at all, however these are much shorter than the other articles. If we visualize two points on opposite sides of a circle, the larger the radius is the further they are apart. Extending this to a W-dim hypersphere, despite 7 and 8 being different, they happen to be the closest because their lengths are the smallest.

Source (Julia):

```julia
euclidean_dists = pairwise(Euclidean(), V, dims=2)
argmin(euclidean_dists + Inf*I)
```

```
CartesianIndex(8, 7)
```

```julia
angle_dists = pairwise(CosineDist(), V, dims=2)
argmin(angle_dists + Inf*I)
```

```
CartesianIndex(10, 9)
```

Q1.10b
After normalizing, both the smallest euclidian distance and angle were documents 9 (Barack Obama) and 10 (George W. Bush). Normalizing accounts for the number of words in the article, meaning the euclidean distances only measure distances linearly across points on the surface of a W-dim sphere. This linear distance is a close approximation to the arc-length, which is what the angle measures.

Source (Julia):

```julia
V_norm = V ./ sum(V, dims=1);
```

```julia
euclidean_dists = pairwise(Euclidean(), V_norm, dims=2);
argmin(euclidean_dists + Inf * I)
```

```
CartesianIndex(10, 9)
```

```julia
angle_dists = pairwise(CosineDist(), V_norm, dims=2);
argmin(angle_dists + Inf * I)
```

```
CartesianIndex(10, 9)
```

Q1.10c
With the new definition, the smallest euclidean distance is still Barack Obama and George W. Bush.

Source (Julia):

```
f_doc = sum(V .> 0.0, dims=2);
```

```
w = V_norm .* sqrt.(log10.(size(V)[2]./f_doc));
euclidean_dists = pairwise(Euclidean(), w, dims=2);
argmin(euclidean_dists + Inf * I)
```

```
CartesianIndex(10, 9)
```

Q1.10d
The TF-IDF definition is a scaling of the normalized word vectors for each document. Geometrically, only the magnitude of each vector is being adjusted. The scaling is inversely porportional to the number of documents the word appears in, reaching zero if the word appears in every document, and reaching a maximum if it only appears in one document. This may be done to make differences between documents more pronounced – unique words are "weighted" more than common ones.

Q1.10e
The fterm matrix is generated and compared to the given matlab representation:

```
In [1]: from collections import Counter
        from functools import reduce
        import numpy as np
        import scipy.io
```

```
In [2]: articles = [line.rstrip('\n') for line in open('data/wordVecArticles.txt')]
        wordcounts = [Counter(article.split()) for article in articles]
        wordset = reduce(lambda s, w: s | w.keys(), wordcounts, set())
        worddict = {w: i for i, w in enumerate(sorted(wordset))}

        fterm = np.zeros((len(worddict), len(articles)))
        for i, wordcount in enumerate(wordcounts):
            words = [worddict[word] for word in wordcount.keys()]
            fterm[words, i] = list(wordcount.values())
```

```
In [3]: reference = scipy.io.loadmat('data/wordVecV.mat')['V']
        np.array_equal(fterm, reference)

Out[3]: True
```

Q1.11a

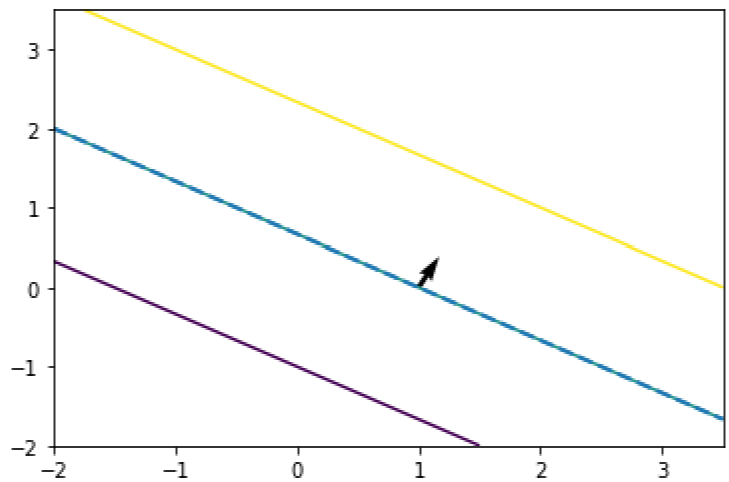
$$\nabla f_1 = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

$$\nabla f_2 = \begin{bmatrix} 2x - y \\ 2y - x \end{bmatrix}$$

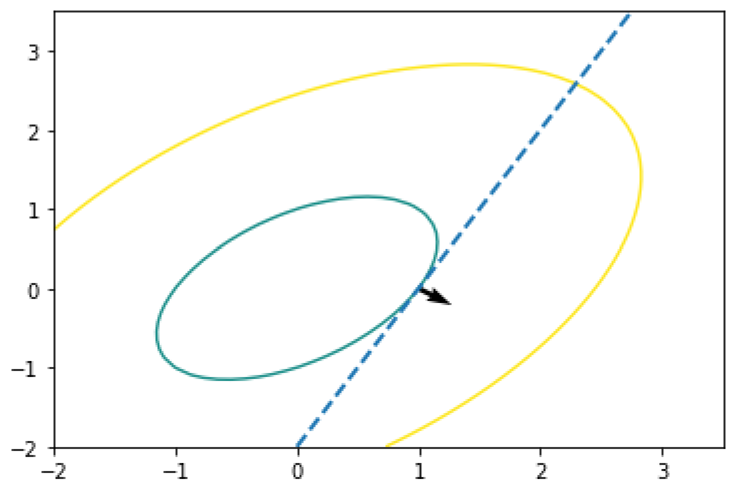
$$\nabla f_3 = \begin{bmatrix} \cos(y-5) - (y-5)\cos(x-5) \\ -(x-5)\sin(y-5) - \sin(x-5) \end{bmatrix}$$

Q1.11b

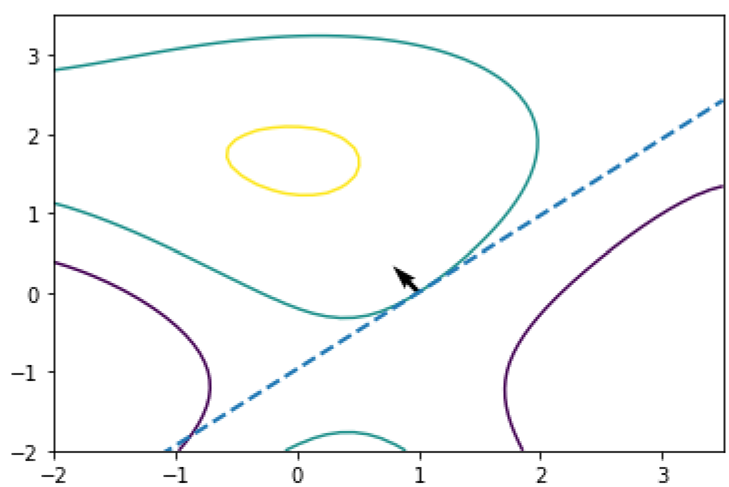
f_1



f_2



f_3

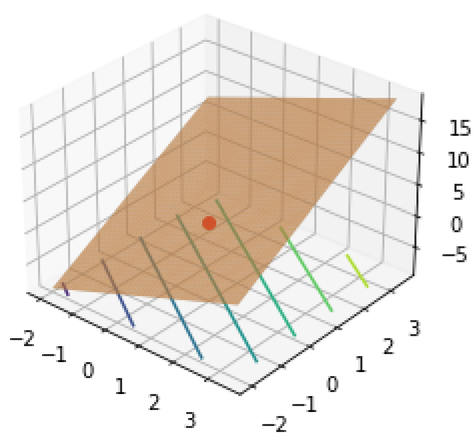


Source (Python):

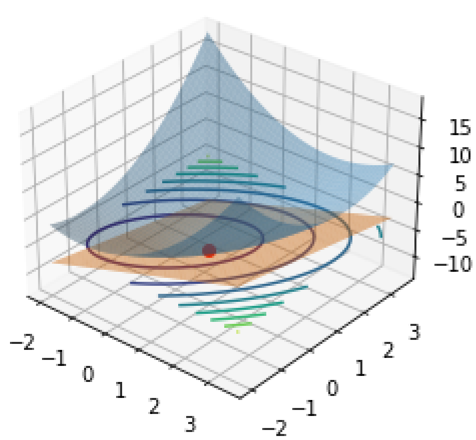
```
def levelset(xs, ys, x, y, f, grad_f):
    gridx, gridy = np.meshgrid(xs, ys)
    c = f(x, y)
    grad = grad_f(x, y)
    tangent = -grad[0]/grad[1] * (xs - 1)
    plt.contour(xs, ys, f(gridx, gridy), [c-5, c, c+5])
    plt.quiver(1, 0, grad[0], grad[1])
    plt.plot(xs, tangent, '--', linewidth=2)
    plt.ylim([np.min(ys), np.max(ys)])
```

Q.11c

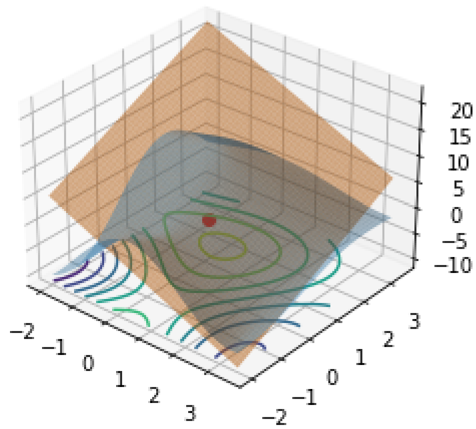
f_1



f_2



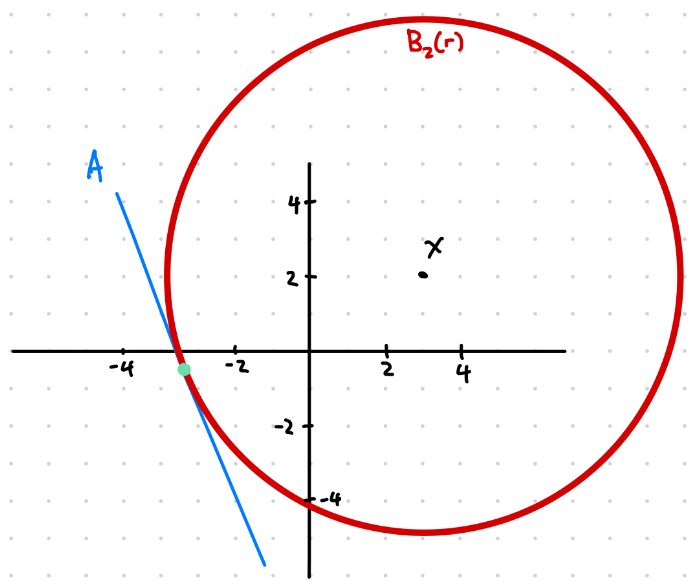
f_3



Source (Python):

```python
def linearapprox(xs, ys, x, y, f, grad_f):
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    ax.view_init(30, -50)
    gridx, gridy = np.meshgrid(xs, ys)
    gridz = f(gridx, gridy)
    grad = grad_f(x, y)
    tangent = lambda xn, yn: f(x, y) + (xn - x) * grad[0] + (yn - y) * grad[1]
    ax.plot_surface(gridx, gridy, f(gridx, gridy), zorder=1, alpha=0.4)
    ax.contour(xs, ys, gridz, offset=np.min(gridz), zorder=-1)
    ax.plot_surface(gridx, gridy, tangent(gridx, gridy), zorder=2, alpha=0.5)
    ax.plot3D(x, y, f(x, y), zorder=2, marker='o', c='r')
```

Q1.12a

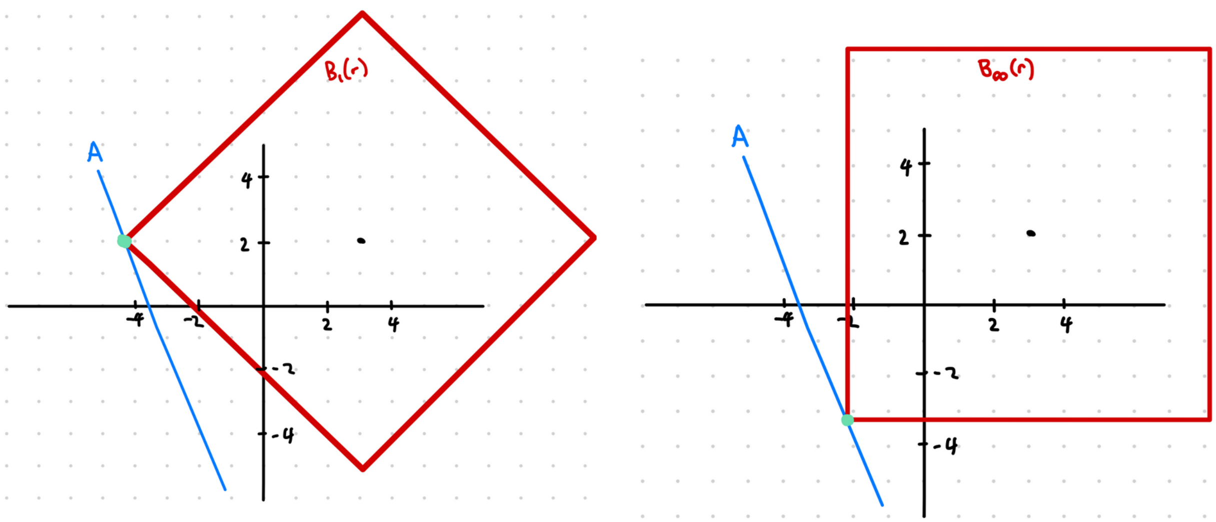


Q1.12b

$$\lambda = \frac{\langle x - v^{(0)}, v\rangle}{\|v\|_2^2} = \frac{1}{29}\langle \begin{bmatrix}5\\6\end{bmatrix}, \begin{bmatrix}-2\\5\end{bmatrix}\rangle = \frac{20}{29}$$

$$\begin{aligned} y^{(2)} &= \lambda v + v^{(0)} \\ &= \frac{20}{29}\begin{bmatrix}-2\\5\end{bmatrix} + \begin{bmatrix}-2\\-4\end{bmatrix} \\ &= \begin{bmatrix}-\frac{98}{29}\\-\frac{16}{29}\end{bmatrix} \\ &\approx \begin{bmatrix}-3.38\\-0.55\end{bmatrix}\end{aligned}$$

Q1.12c



Q1.12d

```
: proj(x, v0, v, norm2)[1]

: 2×1 Array{Float64,2}:
   -3.379308580231317
   -0.5517197152730516
```

Source (Julia)

```julia
function proj(x, v0, v, norm_fn)
    solver = () -> SCS.Optimizer(verbose=0)
    y = Variable(2)
    t = Variable()
    p = minimize(norm_fn(x - y), y == t * v + v0)
    solve!(p, solver)
    return y.value, p.optval
end

v0 = [-2; -4]
v = [-2; 5]
x = [3; 2];
```

Q1.12e
Using function defined in Q1.12d:

$y^{(1)}$:

```
proj(x, v0, v, x -> norm(x, 1))[1]
```

```
2×1 Array{Float64,2}:
 -4.4000000005330495
  2.0000000000497464
```

$y^{(\infty)}$:

```
proj(x, v0, v, x -> maximum(x))[1]
```

```
2×1 Array{Float64,2}:
 -2.2857142854941315
 -3.2857142853456933
```