

**ECE367 PS2**  
**Benjamin Cheng (1004838045)**

**Q2.8a**

$$\nabla f_1 = \begin{bmatrix} 2 \\ 3 \end{bmatrix} \quad \nabla^2 f_1 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

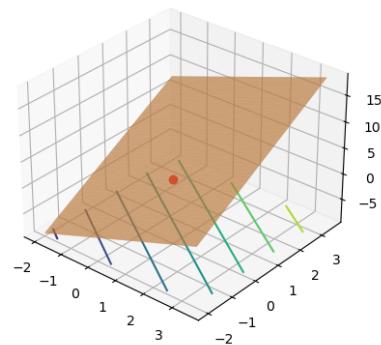
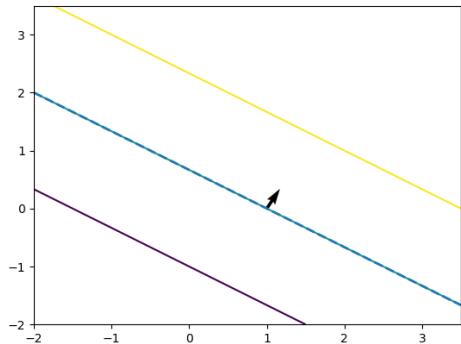
$$\nabla f_2 = \begin{bmatrix} 2x - y \\ 2y - x \end{bmatrix} \quad \nabla^2 f_2 = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$$

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

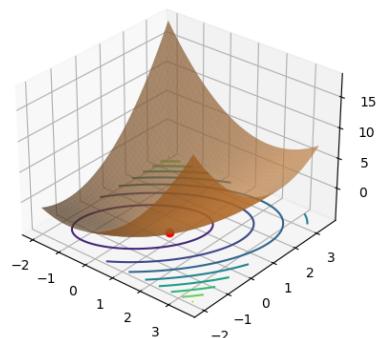
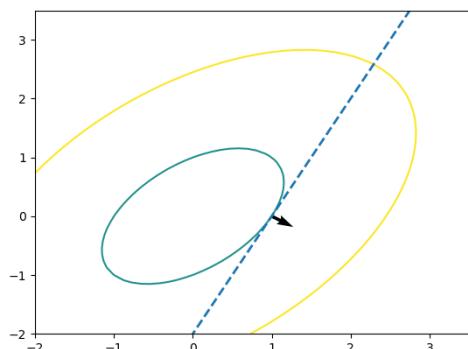
$$\nabla^2 f_3 = \begin{bmatrix} (y-5)\sin(x-5) & -\sin(y-5) - \cos(x-5) \\ -\sin(y-5) - \cos(x-5) & -(x-5)\cos(y-5) \end{bmatrix}$$

**Q2.9b**

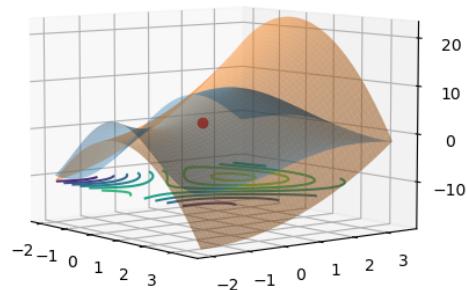
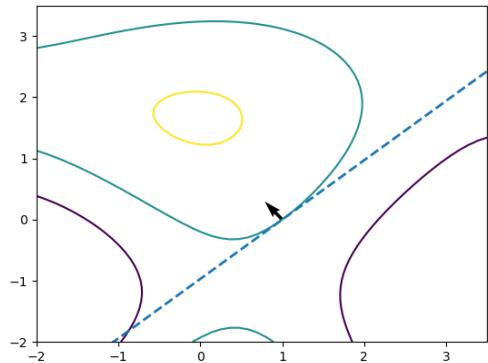
Function 1:



Function 2:

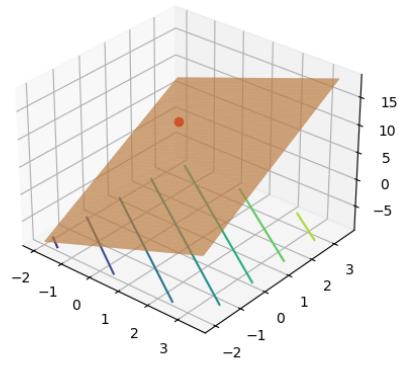
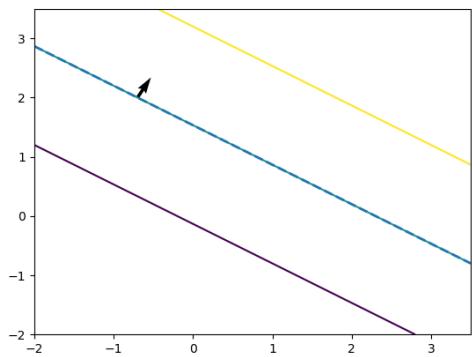


Function 3:

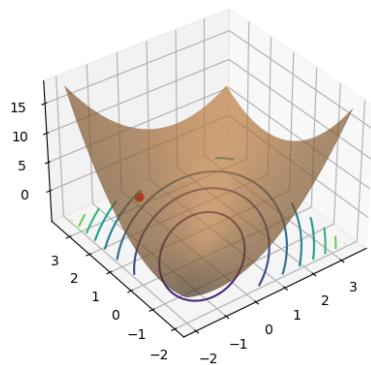
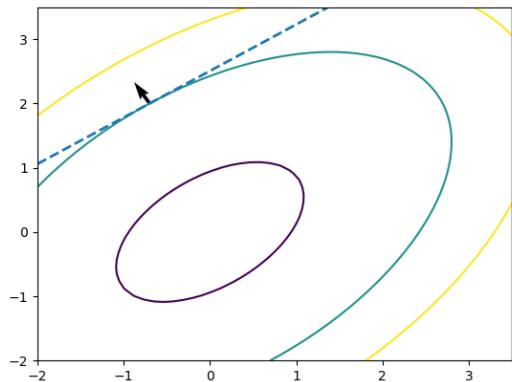


**Q2.9c**

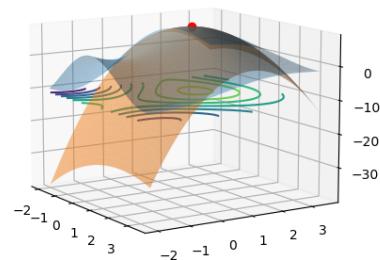
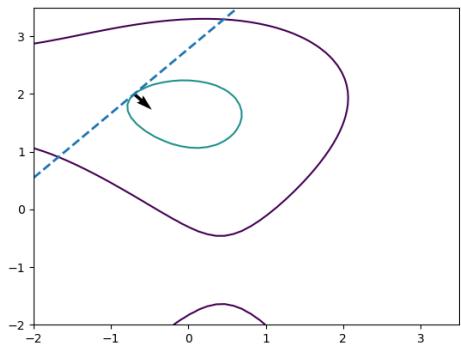
Function 1 (-.7, 2):



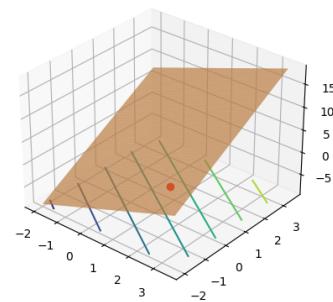
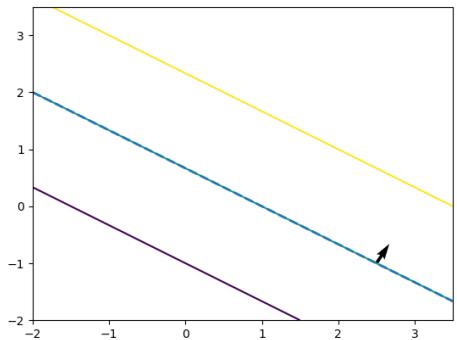
Function 2 (-.7, 2):



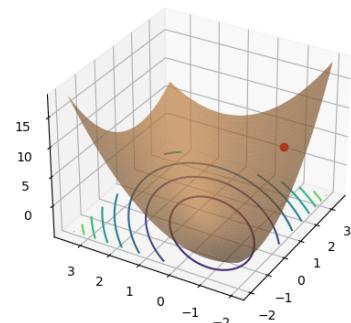
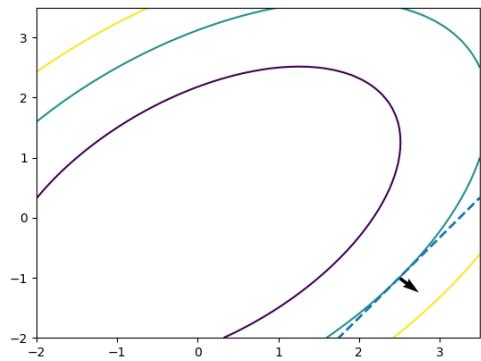
Function 3 (-.7, 2):



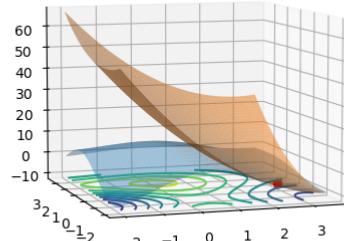
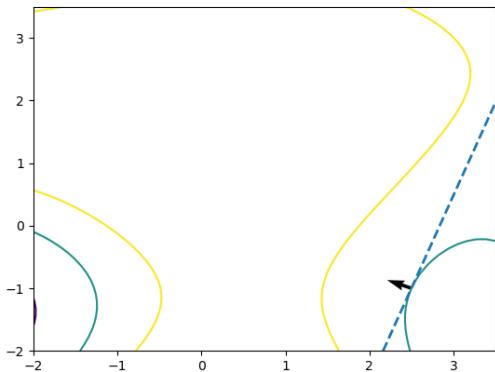
Function 1 (2.5, -1)



Function 2 (2.5, -1)



## Function 3 (2.5, -1)



### Source (Python):

```

1 def quadraticapprox(xs, ys, x, y, f, grad_f, hessian_f):
2     def f_hat(xn, yn):
3         grad = grad_f(x, y)
4         hessian = hessian_f(x, y)
5         xs = np.array([[xn], [yn]])
6         x0 = np.array([[x], [y]])
7         return (f(x, y) + np.dot(grad.T, (xs - x0)) + 1/2 * (xs-x0).T.dot(hessian).dot(xs - x0))
8     fig = plt.figure()
9     ax = fig.gca(projection='3d')
10    ax.view_init(30, -50)
11    gridx, gridy = np.meshgrid(xs, ys)
12    gridz = f(gridx, gridy)
13    ax.plot_surface(gridx, gridy, f(gridx, gridy), zorder=1, alpha=0.4)
14    ax.contour(xs, ys, gridz, offset=np.min(gridz), zorder=-1)
15    ax.plot_surface(gridx, gridy, np.vectorize(f_hat)(gridx, gridy), zorder=2, alpha=0.5)
16    ax.plot3D(x, y, f(x, y), zorder=2, marker='o', c='r')

```

Levelset and specific functions to be plotted (incl. gradients and hessians) are as in PS1.

### Q2.9d

Functions 1 and 2 are always accurate, which is why all the points look like they have only one surface visible. This is because function 1 is a plane, which is exactly equal to a plane. The hessian is zero for this case because a plane perfectly models the function. Function 2 is quadratic in  $x$  and  $y$ , which is perfectly fitted by the 2<sup>nd</sup> order approximation, which is also quadratic in  $x$  and  $y$ . However function 3 is only approximately accurate for values close to the approximation point. This is because function 3 is not linear or quadratic in  $x$  or  $y$ . Recalling Taylor Series for single variable functions, sine and cosine need to be approximated by infinite polynomial terms. Thus function 3 is only accurately approximated by a 2<sup>nd</sup> order approximation for a small area around the approximation point.

## Q2.9

The J matrix is loaded and the link matrix A is computed using the following Python code:

```
1 import numpy as np
2 import scipy.io
3 import matplotlib.pyplot as plt

1 J = scipy.io.loadmat('data/pagerank_adj.mat')['J']
2 A = J/J.sum(axis=0)
```

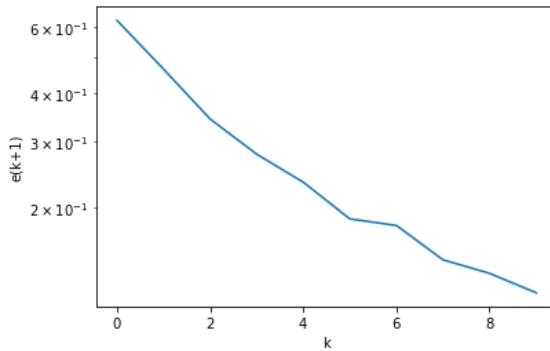
## Q2.9a

The link matrix in the PageRank matrix has the property that the element in the  $i$ -th row and  $j$ -th column represents the probability that a link will be followed from the  $i$ -th page to the  $j$ -th page under the assumption that all links on a page are equally likely to be clicked on. The sum of the columns is essentially summing over  $j$ , which is all the pages that  $i$  links to. This results in 1.

**Source (Python):**

```
: 1 np.allclose(A.sum(axis=0), 1)
: True
```

## Q2.9b



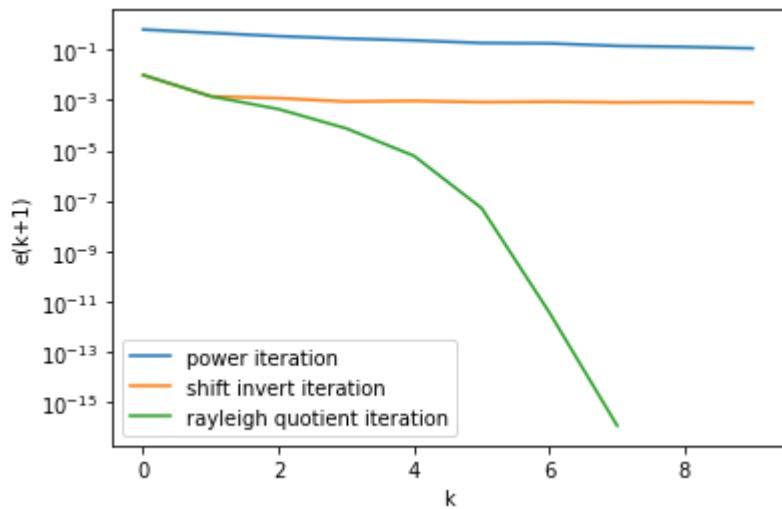
**Source (Python):**

```
1 def power_iteration(A, x, num_iters=10):
2     x = [x]
3     y = []
4     l = []
5     for k in range(num_iters):
6         y.append(A.dot(x[-1]))
7         x.append(y[-1]/np.linalg.norm(y[-1]))
8         l.append(x[-1].conj().T.dot(A).dot(x[-1]))
9     return x[1:], y, l

1 x0 = np.ones(A.shape[0])# + 1j*np.random.rand(A.shape[0]) # Random complex x vector
2 #x0 = x0 / np.linalg.norm(x0)
3 xs, ys, ls = power_iteration(A, x0)

1 error = [np.linalg.norm(A.dot(x) - x) for x in xs]
2 plt.ylabel('e(k+1)')
3 plt.xlabel('k')
4 plt.yscale('log')
5 plt.plot(error)
6 plt.show()
```

## Q2.9c



The Rayleigh Quotient error follows a trend similar to the one displayed in OptM. It is equal to the first 2 iterations of Shift Invert Iteration, which is expected since they are the same for those two. The Power Iteration error decreasing very slowly in comparison, as shown in OptM. However, the error in Shift Invert Iteration doesn't decrease exponentially (should appear linear due to logarithmic scale), as shown in by Calafiore. The Rayleigh Quotient Iteration was stopped after 8 iterations ( $k=7$ ) due to further iterations resulting in numerical inaccuracies causing singular matrices.

### Source(Python):

```
1 def shift_invert_power_iteration(A, x, sigma, num_iters=10):
2     x = [x]
3     y = []
4     l = []
5     A_inv = np.linalg.inv(A - sigma * np.eye(A.shape[0]))
6     for k in range(num_iters):
7         y.append(A.dot(x[-1]))
8         x.append(y[-1]/np.linalg.norm(y[-1]))
9         l.append(x[-1].conj().T.dot(A).dot(x[-1]))
10    return x[1:], y, l
```

```
: 1 def rayleigh_quotient_iteration(A, y, num_iters=10):
2     x = [y]
3     y = []
4
5     for i in range(num_iters):
6         sigma = x[-1].conj().T.dot(A).dot(x[-1]) / x[-1].conj().dot(x[-1]) if i >= 2 else 0.99
7         y.append(np.linalg.solve(A - sigma * np.eye(A.shape[0]), x[-1]))
8         x.append(y[-1]/np.linalg.norm(y[-1]))
9
10    return x[1:], y
```

```

1 si_xs, si_ys, si_ls = shift_invert_power_iteration(A, x0, 0.99)
2 rq_xs, rq_ys = rayleigh_quotient_iteration(A, x0)

1 p_error = [np.linalg.norm(A.dot(x) - x) for x in xs]
2 si_error = [np.linalg.norm(A.dot(x) - x) for x in si_xs]
3 rq_error = [np.linalg.norm(A.dot(x) - x) for x in rq_xs]
4
5 plt.ylabel('e(k+1)')
6 plt.xlabel('k')
7 plt.yscale('log')
8 plt.plot(p_error, label='power iteration')
9 plt.plot(si_error, label='shift invert iteration')
10 plt.plot(rq_error, label='rayleigh quotient iteration')
11 plt.legend()
12 plt.show()

```

## Q2.9d

Taking the eigenvector from the last iteration of the Rayleight Quotient Iteration gives the PageRank scores of all pages. After sorting the vector, the following pages were determined (index, URL):

Top 5:

1. 424, <http://www1.hollins.edu/homepages/hammerpw/qrhomepage.htm>
2. 987, <http://www1.hollins.edu/homepages/hammerpw/qrcourses2.htm>
3. 986, <http://www1.hollins.edu/homepages/hammerpw/qrcourses.htm>
4. 985, <http://www1.hollins.edu/homepages/hammerpw/qractivities.htm>
5. 930, <http://www1.hollins.edu/homepages/godardrd/homepage.htm>

Bottom 5 (Starting from last):

1. 1, <http://www1.hollins.edu/>
2. 48, <http://www.hollins.edu/academics/library/libtoc.htm>
3. 5, <http://www1.hollins.edu/Docs/GVCalendar/gvmain.htm>
4. 10, <http://www1.hollins.edu/Docs/comptech/comptech.htm>
5. 23, <http://www1.hollins.edu/security/Default.htm>

These links are all broken but they appear to be somewhat resonable. The top 4 are under a professor's (assuming hammerpw is someone's name) site, which would likely lead to each other and to other resources. The 5<sup>th</sup> one is another professor's homepage. The absolute lowest score has the homepage, but it seems like there are two homepages, one under www1 and one onder www. It is possible that the www1 homepage is a legacy one that is no longer linked anywhere except a few outliers. It is quite weird that the actual homepage isn't on the top 5 list, but many professors don't link to the University website on their page (see <https://www.comm.utoronto.ca/~sdraper/> as an example).

## Source (Python)

```
1 pagerank = np.absolute(rq_xs[-1]).argsort()  
  
1 pagerank[:5]  
array([ 0,  3, 19,  4, 86])  
  
1 pagerank[-5:]  
array([929, 984, 985, 986, 423])
```