Jared Yu

Computational Statistics

Problem Set 3

NOTE: All code for each of the problems is in the Code Appendix at the end of the document.

1. Problem 2.5. Only do parts (a), (b), (c), and (e).
    a. Let $N_i \mid b_{i1}, b_{i2} \sim Poisson(\lambda_i);\ \lambda_i = \alpha_1 b_{i1} + \alpha_2 b_{i2}, i = 1, \cdots, n_T$

The density function for the above Poisson distribution is as follows:

$$f(n_i) = \frac{\lambda_i^{n_i} e^{-\lambda_i}}{n_i!} = \frac{(\alpha_1 b_{i1} + \alpha_2 b_{i2})^{n_i} e^{-(\alpha_1 b_{i1} + \alpha_2 b_{i2})}}{n_i!}$$

The likelihood for the above p.d.f. is as follows:

$$L(\alpha_1, \alpha_2) = f\left(n_1, \cdots, n_{n_T}; \alpha_1, \alpha_2\right) = \prod_{i=1}^{n_T} \frac{(\alpha_1 b_{i1} + \alpha_2 b_{i2})^{n_i}}{n_i!} e^{-\alpha_1 b_{i1} - \alpha_2 b_{i2}}$$

$$= \frac{\prod_{i=1}^{n_T} (\alpha_1 b_{i1} + \alpha_2 b_{i2})^{n_i}}{\prod_{i=1}^{n_T} n_i!} e^{\sum_{i=1}^{n_T} -\alpha_1 b_{i1} - \alpha_2 b_{i2}}$$

The log-likelihood of the p.d.f. is as follows:

$$l(\alpha_1, \alpha_2) = \ln L(\alpha_1, \alpha_2) = \sum_{i=1}^{n_T} n_i \ln(\alpha_1 b_{i1} + \alpha_2 b_{i2}) + \sum_{i=1}^{n_T} (-\alpha_1 b_{i1} - \alpha_2 b_{i2}) - \sum_{i=1}^{n_T} \ln n_i!$$

The following are the first and second derivatives of the log-likelihood:

Let $\theta = (\alpha_1, \alpha_2)^\top$

$$\frac{\partial l(\theta)}{\partial \alpha_1} = \sum_{i=1}^{n_T} \frac{n_i b_{i1}}{\alpha_1 b_{i1} + \alpha_2 b_{i2}} - \sum_{i=1}^{n_T} b_{i1}$$

$$\frac{\partial l(\theta)}{\partial \alpha_2} = \sum_{i=1}^{n_T} \frac{n_i b_{i2}}{\alpha_1 b_{i1} + \alpha_2 b_{i2}} - \sum_{i=1}^{n_T} b_{i2}$$

$$\frac{\partial l(\theta)}{\partial \theta} = \left(\frac{\partial l(\theta)}{\partial \alpha_1}, \frac{\partial l(\theta)}{\partial \alpha_2}\right)^\top = \left(\sum_{i=1}^{n_T} \frac{n_i b_{i1}}{\alpha_1 b_{i1} + \alpha_2 b_{i2}} - \sum_{i=1}^{n_T} b_{i1}, \sum_{i=1}^{n_T} \frac{n_i b_{i2}}{\alpha_1 b_{i1} + \alpha_2 b_{i2}} - \sum_{i=1}^{n_T} b_{i2}\right)^\top$$

$$\frac{\partial^2 l(\theta)}{\partial \alpha_1^2} = -\sum_{i=1}^{n_T} \frac{n_i b_{i1}^2}{(\alpha_1 b_{i1} + \alpha_2 b_{i2})^2}$$

$$\frac{\partial^2 l(\theta)}{\partial \alpha_2^2} = -\sum_{i=1}^{n_T} \frac{n_i b_{i2}^2}{(\alpha_1 b_{i1} + \alpha_2 b_{i2})^2}$$

$$\frac{\partial^2 l(\theta)}{\partial \alpha_1 \partial \alpha_2} = -\sum_{i=1}^{n_T} \frac{n_i b_{i1} b_{i2}}{(\alpha_1 b_{i1} + \alpha_2 b_{i2})^2}$$

$$\frac{\partial^2 l(\theta)}{\partial \theta^2} = \begin{bmatrix} -\sum_{i=1}^{n_T} \frac{n_i b_{i1}^2}{(\alpha_1 b_{i1} + \alpha_2 b_{i2})^2} & -\sum_{i=1}^{n_T} \frac{n_i b_{i1} b_{i2}}{(\alpha_1 b_{i1} + \alpha_2 b_{i2})^2} \\ -\sum_{i=1}^{n_T} \frac{n_i b_{i1} b_{i2}}{(\alpha_1 b_{i1} + \alpha_2 b_{i2})^2} & -\sum_{i=1}^{n_T} \frac{n_i b_{i2}^2}{(\alpha_1 b_{i1} + \alpha_2 b_{i2})^2} \end{bmatrix}$$

The Newton-Raphson update can be shown as follows:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \boldsymbol{g}^{"}\left(\boldsymbol{\theta}^{(t)}\right)^{-1} \boldsymbol{g}'\left(\boldsymbol{\theta}^{(t)}\right)$$

where $\boldsymbol{\theta}^{(t)}$ is the $t'th$ iteration of the vector of parameters containing $\alpha_1^{(t)}$ and $\alpha_2^{(t)}$, $\boldsymbol{g}^{"}(\cdot)^{-1}$ is the inverse of the 2×2 Hessian matrix denoted above as $\frac{\partial^2 l(\theta)}{\partial \theta^2}$ and $\boldsymbol{g}'(\cdot)$ is the gradient of the log-likelihood denoted above as $\frac{\partial l(\theta)}{\partial \theta}$. Therefore, the above update for finding the MLE of this Poisson distribution can be expressed as:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \boldsymbol{g}^{"}\left(\boldsymbol{\theta}^{(t)}\right)^{-1} \boldsymbol{g}'\left(\boldsymbol{\theta}^{(t)}\right)$$

$$= \boldsymbol{\theta}^{(t)} - \begin{bmatrix} -\sum_{i=1}^{n_T} \frac{n_i b_{i1}^2}{(\alpha_1 b_{i1} + \alpha_2 b_{i2})^2} & -\sum_{i=1}^{n_T} \frac{n_i b_{i1} b_{i2}}{(\alpha_1 b_{i1} + \alpha_2 b_{i2})^2} \\ -\sum_{i=1}^{n_T} \frac{n_i b_{i1} b_{i2}}{(\alpha_1 b_{i1} + \alpha_2 b_{i2})^2} & -\sum_{i=1}^{n_T} \frac{n_i b_{i2}^2}{(\alpha_1 b_{i1} + \alpha_2 b_{i2})^2} \end{bmatrix}^{-1} \begin{pmatrix} \sum_{i=1}^{n_T} \frac{n_i b_{i1}}{\alpha_1 b_{i1} + \alpha_2 b_{i2}} - \sum_{i=1}^{n_T} b_{i1} \\ \sum_{i=1}^{n_T} \frac{n_i b_{i2}}{\alpha_1 b_{i1} + \alpha_2 b_{i2}} - \sum_{i=1}^{n_T} b_{i2} \end{pmatrix}$$

b. The method for finding the Fisher scoring update will be shown below.

The following is the Fisher Information matrix that will be used in the Fisher scoring update. The Fisher Information matrix is denoted $I(\theta)$.

$$I(\theta) = -E\left[\frac{\partial^2 l(\theta)}{\partial \theta^2}\right] = \begin{bmatrix} \sum_{i=1}^{n_T} \frac{E(n_i) b_{i1}^2}{(\lambda_i)^2} & \sum_{i=1}^{n_T} \frac{E(n_i) b_{i1} b_{i2}}{(\lambda_i)^2} \\ \sum_{i=1}^{n_T} \frac{E(n_i) b_{i1} b_{i2}}{(\lambda_i)^2} & \sum_{i=1}^{n_T} \frac{E(n_i) b_{i2}^2}{(\lambda_i)^2} \end{bmatrix}$$

$$= \begin{bmatrix} \sum_{i=1}^{n_T} \frac{b_{i1}^2}{\lambda_i} & \sum_{i=1}^{n_T} \frac{b_{i1}b_{i2}}{\lambda_i} \\ \sum_{i=1}^{n_T} \frac{b_{i1}b_{i2}}{\lambda_i} & \sum_{i=1}^{n_T} \frac{b_{i2}^2}{\lambda_i} \end{bmatrix}$$

Then the Fisher scoring update can be written as:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \mathbf{I}\left(\boldsymbol{\theta}^{(t)}\right)^{-1}\mathbf{l}'\left(\boldsymbol{\theta}^{(t)}\right),$$

where $\boldsymbol{\theta}^{(t)}$ is the $t'th$ iteration of the vector of parameters containing $\alpha_1^{(t)}$ and $\alpha_2^{(t)}$, $\mathbf{I}(\cdot)^{-1}$ is the inverse of the Fisher Information matrix written above as $I(\theta)$, and $\mathbf{l}'(\cdot)$ is the gradient of the log-likelihood function written above as $\frac{\partial l(\theta)}{\partial \theta}$. Therefore, the above update for finding the MLE through using the Fisher scoring update can be expressed as:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \mathbf{I}\left(\boldsymbol{\theta}^{(t)}\right)^{-1}\mathbf{l}'\left(\boldsymbol{\theta}^{(t)}\right)$$

$$= \boldsymbol{\theta}^{(t)} + \begin{bmatrix} \sum_{i=1}^{n_T} \frac{b_{i1}^2}{\alpha_1 b_{i1} + \alpha_2 b_{i2}} & \sum_{i=1}^{n_T} \frac{b_{i1}b_{i2}}{\alpha_1 b_{i1} + \alpha_2 b_{i2}} \\ \sum_{i=1}^{n_T} \frac{b_{i1}b_{i2}}{\alpha_1 b_{i1} + \alpha_2 b_{i2}} & \sum_{i=1}^{n_T} \frac{b_{i2}^2}{\alpha_1 b_{i1} + \alpha_2 b_{i2}} \end{bmatrix}^{-1} \begin{pmatrix} \sum_{i=1}^{n_T} \frac{n_i b_{i1}}{\alpha_1 b_{i1} + \alpha_2 b_{i2}} - \sum_{i=1}^{n_T} b_{i1} \\ \sum_{i=1}^{n_T} \frac{n_i b_{i2}}{\alpha_1 b_{i1} + \alpha_2 b_{i2}} - \sum_{i=1}^{n_T} b_{i2} \end{pmatrix}$$

c. Implementing the two algorithms was not difficult since the Newton-Raphson algorithm was recently completed in the univariate case. The task begins with calculating the math by hand to first figure out the log-likelihood of the function along with its related gradient and Hessian matrix. After the math has been derived, the functions are programmed as separate pieces so that they can be looped through efficiently in RStudio.

   The process for both with and without Fisher scoring is similar in that they require some initialization points, a while-loop performs the steps of re-calculating the gradient and Hessian, and a flag that checks for relative convergence. The initialization points are all combinations of values from -1 to 3. These are based upon the visual analysis of the graph of the log-likelihood shown below (Figures 1 and 2):

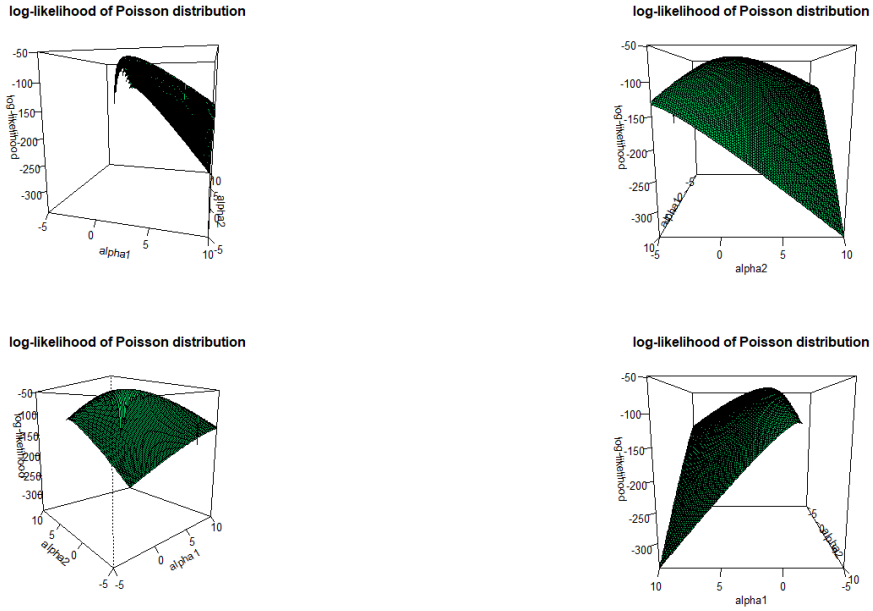

Figure 1 The above plot shows different angles of the log-likelihood of the Poisson distribution. By analyzing the hyperplane of the log-likelihood function, it can be estimated that the peak is roughly somewhere between -1 and 3.

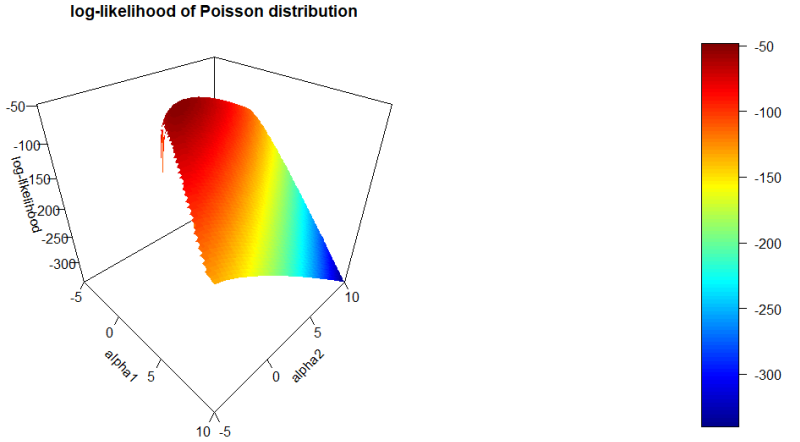

Figure 2 The above plot uses a heatmap of the data on the surface of the hyperplane to illustrate with color that the peak is in dark red. This indicates that the log-likelihood's max value is close to -50.

Both algorithms are run through all combinations of $-1, \cdots, 3$ and are represented in the Table 1 and 2 below. The columns on the far left, α_1 and α_2 represent the initial starting values of the algorithm in a certain trial. The columns $\hat{\alpha}_1$ and $\hat{\alpha}_2$ represent the approximated MLE's after utilizing the corresponding algorithm. The last column $Iterations$ represents the number of iterations required for an algorithm to converge.

Values of NA indicate that the algorithm failed to converge. One of the reasons is that there is a singularity error when trying to take the inverse of the matrix for the

update step. This was common for the Newton-Raphson algorithm. Another error is when dividing by 0 occurs somewhere leading to a failure to the algorithm to converge. The Fisher Scoring was more stable and less likely to output NA due to some calculation error. This is expected, since the Fisher Information matrix is a more efficient calculation compared to the Hessian matrix. Also, in terms of the stability, both methods appear to converge to the same value of $\hat{\alpha}_1 \approx 1.0972$ and $\hat{\alpha}_2 \approx 0.9376$.

There is a noticeable difference in speed when comparing the number of iterations required for convergence. The Newton-Raphson method would usually converge in under 10 iterations, while the Fisher Scoring method required at least 10 iterations to converge. This makes sense, since with optimization problems trade-offs are an expected part of any change in methodology. If the more stable Fisher Scoring method is used, then speed is the trade-off.

Table 1: Results for Newton-Raphson

α_1	α_2	$\hat{\alpha_1}$	$\hat{\alpha_2}$	Iterations
-1	-1	NA	NA	NA
0	-1	NA	NA	NA
1	-1	NA	NA	NA
2	-1	1.097152	0.9375546	8
3	-1	1.097152	0.9375546	8
-1	0	NA	NA	NA
0	0	NA	NA	NA
1	0	1.097152	0.9375546	6
2	0	1.097152	0.9375546	5
3	0	-5.112828	10.3490623	8
-1	1	NA	NA	NA
0	1	1.097152	0.9375546	8
1	1	1.097152	0.9375546	4
2	1	1.097152	0.9375546	7
3	1	NA	NA	NA
-1	2	-5.112828	10.3490623	8
0	2	1.097152	0.9375546	7
1	2	1.097152	0.9375546	6
2	2	3.004737	-1.9534767	10
3	2	NA	NA	NA
-1	3	-3.631235	8.1036416	7
0	3	1.097152	0.9375546	6
1	3	4.836828	-4.7300943	7
2	3	NA	NA	NA
3	3	NA	NA	NA

Table 2: Results for Newton-Raphson with Fisher Scoring

α_1	α_2	$\hat{\alpha_1}$	$\hat{\alpha_2}$	Iterations
-1	-1	1.097152	0.9375556	10
0	-1	1.097151	0.9375571	12
1	-1	NA	NA	NA
2	-1	1.097153	0.9375533	12
3	-1	1.097153	0.9375535	12
-1	0	1.097151	0.9375569	11
0	0	NA	NA	NA
1	0	1.097151	0.9375569	11
2	0	1.097151	0.9375569	11
3	0	1.097151	0.9375569	11
-1	1	NA	NA	NA
0	1	1.097151	0.9375571	12
1	1	1.097152	0.9375556	10
2	1	1.097154	0.9375519	10
3	1	1.097152	0.9375559	11
-1	2	1.097153	0.9375535	14
0	2	1.097151	0.9375571	12
1	2	1.097154	0.9375527	11
2	2	1.097152	0.9375556	10
3	2	1.097153	0.9375532	10
-1	3	1.097152	0.9375556	14
0	3	1.097151	0.9375571	12
1	3	1.097152	0.9375555	12
2	3	1.097153	0.9375534	11
3	3	1.097152	0.9375556	10

e. The steepest ascent method was also implemented for the same dataset. The technique to modify the learning rate α was to use step-halving, where if the approximated value of the log-likelihood function at the next iteration is not greater than or equal to the current iteration then $\alpha = 1$ will be halved until this happens. The reason that greater than or equal to is used is that otherwise the value of α will reduce to essentially 0 and the algorithm will fail to converge.

However, according to the lecture notes and the textbook, it seems to be implied that it should be strictly greater than. A possible explanation is that R is not capable of calculating and storing values to the tiny sizes required for the algorithm to function properly. Therefore, a greater than or equal to sign is required in the check for backtracking. This also occurs when Fisher scoring is used in place of the identity matrix in the update step.

To further clarify, the textbook mentions that in the Newton-Like methods such as steepest ascent, the Hessian matrix is replaced with another matrix that is simpler to calculate. A possibility is the identity matrix, and at the end of p.39 it mentions that, "backtracking with Fisher scoring would avoid stepping downhill." In an email conversation with the professor, it was made clear that the intention for this assignment is to utilize the identity matrix to replace the Hessian.

In the discussion problems for module 3, where the steepest ascent algorithm was used on the Normal distribution the results were ideal. The approximated MLE's in that case converged to the ideal values and the high number of iterations and backtracking steps coincides with what is understood to be the trade-off when implementing steepest ascent.

For this problem, steepest ascent was used in two different situations, the first will show the identity matrix as the substitute for the Hessian while the second will show the Fisher Information matrix as the substitute. Below are the two tables (Tables 3 and 4). From the tables it's evident that the identity matrix method fails to converge properly, while using Fisher scoring allows for convergence. The reason that convergence fails has been associated with the following calculation:

$$\log \lambda_i = \log \alpha_1 b_{i1} + \alpha_2 b_{i2}$$

The effect is that λ_i becomes negative, leading the log transformation evaluating to $-\infty$. However, the Fisher Information matrix prevents this from occurring as often.

Table 3: Results for Steepest Ascent with Identity Matrix

α_1	α_2	$\hat{\alpha_1}$	$\hat{\alpha_2}$	Iterations	Backtracks
-1	-1	NA	NA	NA	NA
0	-1	NA	NA	NA	NA
1	-1	NA	NA	NA	NA
2	-1	NA	NA	NA	NA
3	-1	NA	NA	NA	NA
-1	0	NA	NA	NA	NA
0	0	NA	NA	NA	NA
1	0	NA	NA	NA	NA
2	0	NA	NA	NA	NA
3	0	NA	NA	NA	NA
-1	1	NA	NA	NA	NA
0	1	NA	NA	NA	NA
1	1	1.097152	0.9375546	3416	13619
2	1	NA	NA	NA	NA
3	1	NA	NA	NA	NA
-1	2	NA	NA	NA	NA
0	2	NA	NA	NA	NA
1	2	NA	NA	NA	NA
2	2	NA	NA	NA	NA
3	2	NA	NA	NA	NA
-1	3	NA	NA	NA	NA
0	3	NA	NA	NA	NA
1	3	NA	NA	NA	NA
2	3	NA	NA	NA	NA
3	3	NA	NA	NA	NA

Table 4: Results for Steepest Ascent with Fisher Matrix

α_1	α_2	$\hat{\alpha_1}$	$\hat{\alpha_2}$	Iterations	Backtracks
-1	-1	NA	NA	NA	NA
0	-1	NA	NA	NA	NA
1	-1	NA	NA	NA	NA
2	-1	1.097152	0.9375546	38	23
3	-1	1.097152	0.9375546	36	15
-1	0	NA	NA	NA	NA
0	0	NA	NA	NA	NA
1	0	1.097152	0.9375546	35	11
2	0	1.097152	0.9375546	32	10
3	0	1.097152	0.9375546	32	10
-1	1	NA	NA	NA	NA
0	1	1.097152	0.9375546	37	22
1	1	1.097152	0.9375546	32	19
2	1	1.097152	0.9375546	33	14
3	1	1.097152	0.9375546	34	14
-1	2	NA	NA	NA	NA
0	2	1.097152	0.9375546	35	14
1	2	1.097152	0.9375546	38	21
2	2	1.097152	0.9375546	31	9
3	2	1.097152	0.9375546	35	22
-1	3	NA	NA	NA	NA
0	3	1.097152	0.9375546	34	10
1	3	1.097152	0.9375546	36	19
2	3	1.097152	0.9375546	34	14
3	3	1.097152	0.9375546	34	22

2. Consider the following mixture of two normal densities:

$$\mathbf{p}(x;\theta) = \pi\phi(x;\ \mu_1,\sigma_1^2) + (1-\pi)\phi(x;\ \mu_2,\sigma_2^2)$$

where $\theta = (\pi,\mu_1,\mu_2,\sigma_1^2,\sigma_2^2)$

a. Show the complete-data log-likelihood function:

$$L(\theta;x,z) = \prod_{i=1}^{n}[\pi\phi(x_i;\mu_1,\sigma_1^2)]^{z_i}\,[(1-\pi)\phi(x_i;\mu_2,\sigma_2^2)]^{(1-z_i)}$$

$$= \prod_{i=1}^{n}[\pi\phi(x_i;\mu_1,\sigma_1^2)]^{z_i}\prod_{i=1}^{n}[(1-\pi)\phi(x_i;\mu_2,\sigma_2^2)]^{(1-z_i)}$$

$$l(\theta;x,z) = \log L(\theta;x,z) = \sum_{i=1}^{n}\log[[\pi\phi(x_i;\mu_1,\sigma_1^2)]^{z_i}] + \sum_{i=1}^{n}\log[(1-\pi)\phi(x_i;\mu_2,\sigma_2^2)]^{(1-z_i)}$$

$$= \sum_{i=1}^{n} z_i[\log\pi + \log\phi(x_i;\mu_1,\sigma_1^2)] + \sum_{i=1}^{n}(1-z_i)[\log(1-\pi) + \log\phi(x_i;\mu_2,\sigma_2^2)]$$

$$= \log\pi\sum_{i=1}^{n} z_i + \sum_{i=1}^{n} z_i\log\phi(x_i;\mu_1,\sigma_1^2) + \log(1-\pi)\left(n - \sum_{i=1}^{n} z_i\right) + \sum_{i=1}^{n}(1-z_i)\log\phi(x_i;\mu_2,\sigma_2^2)$$

b. Find $Q\left(\theta|\theta^{(k)}\right)$ in terms of $E\left[Z_i|x_i,\theta^{(k)}\right]$

$$Q\left(\theta|\theta^{(k)}\right) = E\left[l(\theta;y)|x,\theta^{(k)}\right]E\left[l(\theta;x,z)|x,\theta^{(k)}\right]$$

$$= E\left\{\log\pi^{(k)}\sum_{i=1}^{n} Z_i + \sum_{i=1}^{n} Z_i\log\phi\left(x_i;\mu_1^{(k)},(\sigma_1^2)^{(k)}\right)\right.$$
$$\left. + \log\left(1-\pi^{(k)}\right)\left(n - \sum_{i=1}^{n} Z_i\right) + \sum_{i=1}^{n}(1-Z_i)\log\phi\left(x_i;\mu_2^{(k)},(\sigma_2^2)^{(k)}\right)\right\}$$

$$= \log\pi^{(k)}\sum_{i=1}^{n} E\left[Z_i|x_i,\theta^{(k)}\right] + \log\phi\left(x_i;\mu_1^{(k)},(\sigma_1^2)^{(k)}\right)\sum_{i=1}^{n} E\left[Z_i|x_i,\theta^{(k)}\right] +$$
$$\log\left(1-\pi^{(k)}\right)\left(n - \sum_{i=1}^{n} E\left[Z_i|x_i,\theta^{(k)}\right]\right) + \log\phi\left(x_i;\mu_2^{(k)},(\sigma_2^2)^{(k)}\right)\left(n - \sum_{i=1}^{n} E\left[Z_i|x_i,\theta^{(k)}\right]\right)$$

c. Show $E\left[Z_i|x_i,\theta^{(k)}\right]$

$$E\left[Z_i|x_i,\theta^{(k)}\right] = P\left[Z_i = 1|x_i,\theta^{(k)}\right]\cdot 1 + P\left[Z_i = 0|x_i,\theta^{(k)}\right]\cdot 0$$

$$= P\left[Z_i = 1 | x_i, \theta^{(k)}\right] = \frac{P\left[x_i, Z_i = 1 | \theta^{(k)}\right]}{P[x_i; \theta^{(k)}]}$$

$$= \frac{\pi^{(k)} \phi\left(x_i; \mu_1^{(k)}, (\sigma_1^2)^{(k)}\right)^{(1)} \left[\left(1 - \pi^{(k)}\right) \phi\left(x_i; \mu_2^{(k)}, (\sigma_2^2)^{(k)}\right)\right]^{(1-(1))}}{\pi^{(k)} \phi\left(x_i; \mu_1^{(k)}, (\sigma_1^2)^{(k)}\right) + (1 - \pi^{(k)}) \phi\left(x_i; \mu_2^{(k)}, (\sigma_2^2)^{(k)}\right)}$$

$$\boxed{= \frac{\pi^{(k)} \phi\left(x_i; \mu_1^{(k)}, (\sigma_1^2)^{(k)}\right)}{\pi^{(k)} \phi\left(x_i; \mu_1^{(k)}, (\sigma_1^2)^{(k)}\right) + (1 - \pi^{(k)}) \phi\left(x_i; \mu_2^{(k)}, (\sigma_2^2)^{(k)}\right)}}$$

d. Show $\pi^{(k+1)}$

Let $\eta_i^{(k)} = E\left[Z_i | x_i, \theta^{(k)}\right]$ and $\eta^{(k)} = \sum_{i=1}^n \eta_i^{(k)}$.

$$\frac{\delta Q\left(\theta | \theta^{(k)}\right)}{\delta \pi^{(k)}}$$

$$= \frac{\delta \left\{ \begin{matrix} \log \pi^{(k)} \sum_{i=1}^n E\left[Z_i | x_i, \theta^{(k)}\right] + \log \phi\left(x_i; \mu_1^{(k)}, (\sigma_1^2)^{(k)}\right) \sum_{i=1}^n E\left[Z_i | x_i, \theta^{(k)}\right] + \\ \log\left(1 - \pi^{(k)}\right)\left(n - \sum_{i=1}^n E\left[Z_i | x_i, \theta^{(k)}\right]\right) + \log \phi\left(x_i; \mu_2^{(k)}, (\sigma_2^2)^{(k)}\right)\left(n - \sum_{i=1}^n E\left[Z_i | x_i, \theta^{(k)}\right]\right) \end{matrix} \right\}}{\delta \pi^{(k)}}$$

$$= \frac{\eta^{(k)}}{\pi^{(k)}} - \frac{n - \eta^{(k)}}{1 - \pi^{(k)}} = \stackrel{set\ to}{\widehat{=}} 0$$

$$\Rightarrow \frac{\eta^{(k)}}{\pi^{(k)}} = \frac{n - \eta^{(k)}}{1 - \pi^{(k)}} \Rightarrow \frac{1 - \pi^{(k)}}{\pi^{(k)}} = \frac{n - \eta^{(k)}}{\eta^{(k)}} \Rightarrow \frac{1}{\pi^{(k)}} = \frac{n}{\eta^{(k)}} \Rightarrow \pi^{(k)} = \frac{\eta^{(k)}}{n}$$

$$\boxed{\Rightarrow \pi^{(k+1)} = \frac{\eta^{(k)}}{n} = \frac{1}{n} \sum_{i=1}^n E\left[Z_i | x_i, \theta^{(k)}\right]}$$

e. Show $\mu_1^{(k+1)}$, $(\sigma_1^2)^{(k+1)}$, $\mu_2^{(k+1)}$, and $(\sigma_2^2)^{(k+1)}$

$$\frac{\delta Q\left(\theta | \theta^{(k)}\right)}{\delta \mu_1^{(k)}}$$

$$= \frac{\delta \left\{ \begin{matrix} \log \pi^{(k)} \sum_{i=1}^n E\left[Z_i | x_i, \theta^{(k)}\right] + \log \phi\left(x_i; \mu_1^{(k)}, (\sigma_1^2)^{(k)}\right) \sum_{i=1}^n E\left[Z_i | x_i, \theta^{(k)}\right] + \\ \log\left(1 - \pi^{(k)}\right)\left(n - \sum_{i=1}^n E\left[Z_i | x_i, \theta^{(k)}\right]\right) + \log \phi\left(x_i; \mu_2^{(k)}, (\sigma_2^2)^{(k)}\right)\left(n - \sum_{i=1}^n E\left[Z_i | x_i, \theta^{(k)}\right]\right) \end{matrix} \right\}}{\delta \mu_1^{(k)}}$$

$$= \eta^{(k)} \left[-\frac{1}{2}\left((\sigma_1^2)^{(k)}\right)^{-1}(2)\left(x_i - \mu_1^{(k)}\right)(-1)\right] = \eta^{(k)} \frac{\left(x_i - \mu_1^{(k)}\right)}{(\sigma_1^2)^{(k)}} \stackrel{set\ to}{\widehat{=}} 0$$

$$\Rightarrow \eta^{(k)} x_i = \eta^{(k)} \mu_1^{(k)} \Rightarrow \mu_1^{(k)} = \frac{1}{\eta^{(k)}} \eta^{(k)} x_i$$

$$\boxed{\Rightarrow \mu_1^{(k+1)} = \frac{1}{\eta^{(k)}} \sum_{i=1}^{n} \eta_i^{(k)} x_i}$$

$$\frac{\delta Q\left(\theta|\theta^{(k)}\right)}{\delta (\sigma_1^2)^{(k)}}$$

$$= \frac{\delta \begin{Bmatrix} \log \pi^{(k)} \sum_{i=1}^{n} E\left[Z_i|x_i, \theta^{(k)}\right] + \log \phi\left(x_i; \mu_1^{(k)}, (\sigma_1^2)^{(k)}\right) \sum_{i=1}^{n} E\left[Z_i|x_i, \theta^{(k)}\right] + \\ \log\left(1-\pi^{(k)}\right)\left(n - \sum_{i=1}^{n} E\left[Z_i|x_i, \theta^{(k)}\right]\right) + \log \phi\left(x_i; \mu_2^{(k)}, (\sigma_2^2)^{(k)}\right)\left(n - \sum_{i=1}^{n} E\left[Z_i|x_i, \theta^{(k)}\right]\right) \end{Bmatrix}}{\delta (\sigma_1^2)^{(k)}}$$

$$= \eta^{(k)} \left\{ -\frac{1}{2(\sigma_1^2)^{(k)}} + \frac{\left(x_i - \mu_1^{(k)}\right)^2}{2((\sigma_1^2)^{(k)})^2} \right\} \overset{set\ to}{\widehat{=}} 0$$

$$\Rightarrow \frac{\eta^{(k)}}{2(\sigma_1^2)^{(k)}} = \eta^{(k)} \frac{\left(x_i - \mu_1^{(k)}\right)^2}{2((\sigma_1^2)^{(k)})^2} \Rightarrow (\sigma_1^2)^{(k)} = \frac{1}{\eta^{(k)}} \sum_{i=1}^{n} \eta_i^{(k)} \left(x_i - \mu_1^{(k)}\right)^2$$

$$\boxed{\Rightarrow (\sigma_1^2)^{(k+1)} = \frac{1}{\eta^{(k)}} \sum_{i=1}^{n} \eta_i^{(k)} \left(x_i - \mu_1^{(k+1)}\right)^2}$$

$$\frac{\delta Q\left(\theta|\theta^{(k)}\right)}{\delta \mu_2^{(k)}}$$

$$= \frac{\delta \begin{Bmatrix} \log \pi^{(k)} \sum_{i=1}^{n} E\left[Z_i|x_i, \theta^{(k)}\right] + \log \phi\left(x_i; \mu_1^{(k)}, (\sigma_1^2)^{(k)}\right) \sum_{i=1}^{n} E\left[Z_i|x_i, \theta^{(k)}\right] + \\ \log\left(1-\pi^{(k)}\right)\left(n - \sum_{i=1}^{n} E\left[Z_i|x_i, \theta^{(k)}\right]\right) + \log \phi\left(x_i; \mu_2^{(k)}, (\sigma_2^2)^{(k)}\right)\left(n - \sum_{i=1}^{n} E\left[Z_i|x_i, \theta^{(k)}\right]\right) \end{Bmatrix}}{\delta \mu_2^{(k)}}$$

$$= \left(n - \eta^{(k)}\right) \left\{ -\frac{1}{2(\sigma_2^2)^{(k)}} (2)\left(x_i - \mu_2^{(k)}\right)(-1) \right\} = \frac{\left(n - \eta^{(k)}\right)\left(x_i - \mu_2^{(k)}\right)}{(\sigma_2^2)^{(k)}} \overset{set\ to}{\widehat{=}} 0$$

$$\Rightarrow \frac{\left(n - \eta^{(k)}\right) x_i}{(\sigma_2^2)^{(k)}} = \frac{\left(n - \eta^{(k)}\right) \mu_2^{(k)}}{(\sigma_2^2)^{(k)}} \Rightarrow \mu_2^{(k)} = \frac{1}{n - \eta^{(k)}} \left(n - \eta^{(k)}\right) x_i$$

$$\boxed{\Rightarrow \mu_2^{(k+1)} = \frac{1}{n - \eta^{(k)}} \sum_{i=1}^{n} \left(1 - \eta_i^{(k)}\right) x_i}$$

$$\frac{\delta Q\left(\theta|\theta^{(k)}\right)}{\delta(\sigma_2^2)^{(k)}}$$

$$= \frac{\delta\begin{Bmatrix} \log\pi^{(k)}\sum_{i=1}^{n}E\left[Z_i|x_i,\theta^{(k)}\right] + \log\phi\left(x_i;\mu_1^{(k)},(\sigma_1^2)^{(k)}\right)\sum_{i=1}^{n}E\left[Z_i|x_i,\theta^{(k)}\right] + \\ \log\left(1-\pi^{(k)}\right)\left(n-\sum_{i=1}^{n}E\left[Z_i|x_i,\theta^{(k)}\right]\right) + \log\phi\left(x_i;\mu_2^{(k)},(\sigma_2^2)^{(k)}\right)\left(n-\sum_{i=1}^{n}E\left[Z_i|x_i,\theta^{(k)}\right]\right)\end{Bmatrix}}{\delta(\sigma_2^2)^{(k)}}$$

$$= \left(n-\eta^{(k)}\right)\left\{-\frac{1}{2(\sigma_2^2)^{(k)}} + \frac{\left(x_i-\mu_2^{(k)}\right)^2}{2((\sigma_2^2)^{(k)})^2}\right\} \overset{set\ to}{\hat{=}} 0$$

$$\Rightarrow \frac{\left(n-\eta^{(k)}\right)}{2(\sigma_2^2)^{(k)}} = \frac{\left(n-\eta^{(k)}\right)\left(x_i-\mu_2^{(k)}\right)^2}{2((\sigma_2^2)^{(k)})^2} \Rightarrow (\sigma_2^2)^{(k)} = \frac{1}{n-\eta^{(k)}}\left(n-\eta^{(k)}\right)\left(x_i-\mu_2^{(k)}\right)^2$$

$$\boxed{\Rightarrow (\sigma_2^2)^{(k+1)} = \frac{1}{n-\eta^{(k)}}\sum_{i=1}^{n}\left(1-\eta_i^{(k)}\right)\left(x_i-\mu_2^{(k+1)}\right)^2}$$

f. Below is the pseudocode for the EM-algorithm. It was asked in an email thread with the professor if such a format is acceptable to represent pseudocode and it was said that this is okay.

Let $\vec{x} = (x_1, x_2, \cdots, x_n)$ be an observed sample.

Initialize $k = 0$, $\vec{\theta}^{(0)} = \left(\pi^{(0)}, \mu_1^{(0)}, \mu_2^{(0)}, {\sigma_1^2}^{(0)}, {\sigma_2^2}^{(0)}\right)$

Epsilon = $1e^{-10}$

Convergence_flag = TRUE

While (Convergence_flag) {

E-step Compute $Q\left(\vec{\theta}|\vec{\theta}^{(k)}\right)$

$Q\left(\vec{\theta}|\vec{\theta}^{(k)}\right) = \log \pi^{(k)} \sum_{i=1}^{n} E\left[Z_i|x_i, \theta^{(k)}\right] + \log \phi\left(x_i; \mu_1^{(k)}, (\sigma_1^2)^{(k)}\right) \sum_{i=1}^{n} E\left[Z_i|x_i, \theta^{(k)}\right] +$
$\log\left(1 - \pi^{(k)}\right)\left(n - \sum_{i=1}^{n} E\left[Z_i|x_i, \theta^{(k)}\right]\right) + \log \phi\left(x_i; \mu_2^{(k)}, (\sigma_2^2)^{(k)}\right)\left(n - \sum_{i=1}^{n} E\left[Z_i|x_i, \theta^{(k)}\right]\right)$

M-step Maximize $Q\left(\vec{\theta}|\vec{\theta}^{(k)}\right)$ w.r.t. $\vec{\theta}$, set $\vec{\theta}^{(k+1)}$ equal to maximizer of Q

$\pi^{(k+1)} = \frac{\eta^{(k)}}{n} = \frac{1}{n}\sum_{i=1}^{n} E\left[Z_i|x_i, \theta^{(k)}\right]$

$\mu_1^{(k+1)} = \frac{1}{\eta^{(k)}}\sum_{i=1}^{n} \eta_i^{(k)} x_i$

$(\sigma_1^2)^{(k+1)} = \frac{1}{\eta^{(k)}}\sum_{i=1}^{n} \eta_i^{(k)}\left(x_i - \mu_1^{(k+1)}\right)^2$

$\mu_2^{(k+1)} = \frac{1}{n-\eta^{(k)}}\sum_{i=1}^{n}\left(1 - \eta_i^{(k)}\right)x_i$

$(\sigma_2^2)^{(k+1)} = \frac{1}{n-\eta^{(k)}}\sum_{i=1}^{n}\left(1 - \eta_i^{(k)}\right)\left(x_i - \mu_2^{(k+1)}\right)^2$

Check for convergence

If $\frac{\left|\left|\vec{\theta}^{(k)} - \vec{\theta}^{(k-1)}\right|\right|}{||\vec{\theta}^{(k-1)}||} <$ Epsilon {

Convergence_flag = FALSE

}

}

Return $\vec{\theta}^{(k+1)}$

Code Appendix (*NOTE: Towards the end in the steepest ascent algorithm there are two lines commented to allow for switching of updating methods. They alternate between utilizing the identity matrix and the Fisher Information matrix.*)

```
library(plot3D); library(knitr) # Libraries

### Problem 2.5
oilspills <- read.table(file = "oilspills.dat", header = TRUE)

### part a
# log-likelihood of Poisson
g0 <- function(x = oilspills, alpha1, alpha2) {
  n <- x[,2]; b1 <- x[,3]; b2 <- x[,4]
  sum(n * log(alpha1 * b1 + alpha2 * b2)) +
    sum(-alpha1 * b1 - alpha2 * b2) -
    sum(log(factorial(n)))
}

# dg/dalpha1
g1 <- function(x = oilspills, alpha1, alpha2) {
  n <- x[,2]; b1 <- x[,3]; b2 <- x[,4]
  sum((n * b1) / (alpha1 * b1 + alpha2 * b2)) - sum(b1)
}

# dg/dalpha2
g2 <- function(x = oilspills, alpha1, alpha2) {
  n <- x[,2]; b1 <- x[,3]; b2 <- x[,4]
  sum((n * b2) / (alpha1 * b1 + alpha2 * b2)) - sum(b2)
}

# gradient(alpha1, alpha2)
g3 <- function(x = oilspills, alpha1, alpha2) {
  return(c(g1(x, alpha1, alpha2), g2(x, alpha1, alpha2)))
}

# d^2g/dalpha1^2
g4 <- function(x = oilspills, alpha1, alpha2) {
  n <- x[,2]; b1 <- x[,3]; b2 <- x[,4]
  -sum((n * b1^2) / (alpha1 * b1 + alpha2 * b2)^2)
}

# d^2g/dalpha2^2
g5 <- function(x = oilspills, alpha1, alpha2) {
  n <- x[,2]; b1 <- x[,3]; b2 <- x[,4]
  -sum((n * b2^2) / (alpha1 * b1 + alpha2 * b2)^2)
}

# d^2g/(dalpha1 * dalpha2)
g6 <- function(x = oilspills, alpha1, alpha2) {
  n <- x[,2]; b1 <- x[,3]; b2 <- x[,4]
  -sum((n * b1 * b2) / (alpha1 * b1 + alpha2 * b2)^2)
}

# 2x2 Hessian
g7 <- function(x = oilspills, alpha1, alpha2) {
  return(matrix(c(
    g4(x, alpha1, alpha2), g6(x, alpha1, alpha2),
    g6(x, alpha1, alpha2), g5(x, alpha1, alpha2)), ncol = 2))
}

h <- function(x = oilspills, alpha1, alpha2) {  # h function
  return(-solve(g7(x, alpha1, alpha2)) %*% g3(x, alpha1, alpha2))
}

relative_convergence <- function(alpha1_old, alpha1_new,  # Test for convergence
                                 alpha2_old, alpha2_new, epsilon) {
  old_vec <- c(alpha1_old, alpha2_old)
```

```r
  new_vec <- c(alpha1_new, alpha2_new)
  relative_convergence_criterion <- sqrt(sum((new_vec - old_vec)^2)) /
    (sqrt(sum(old_vec^2)) + epsilon)
  bool_flag <- ifelse(relative_convergence_criterion < epsilon, FALSE, TRUE)
  return(c(relative_convergence_criterion, bool_flag))
}

# Plot 3D graph of the log-likelihood function
# Reference: https://www.datamentor.io/r-programming/3d-plot/
xs <- seq(-5, 10, length.out = 100)
ys <- seq(-5, 10, length.out = 100)

# Reference: https://stackoverflow.com/questions/52317124/dims-product-xx-do-not-match-the-length-of-object-xx-error-in-using-r-func
g0_vectorize <- Vectorize(g0, vectorize.args = c("alpha1", "alpha2"))
z <- outer(xs, ys, function(xs, ys) g0_vectorize(alpha1 = xs, alpha2 = ys))

# Reference: https://stackoverflow.com/questions/17606906/find-row-and-column-index-of-maximum-value-in-a-matrix
max_x_y <- which(z == max(z, na.rm = TRUE), arr.ind = TRUE) # 41,40
# xs[max_x_y[1]]; ys[max_x_y[2]] # 1.060606, 0.9090909

persp3D(xs, ys, z,
      main = "log-likelihood of Poisson distribution",
      zlab = "log-likelihood", xlab = "alpha1", ylab = "alpha2",
      theta = 45, phi = 35, ticktype = "detailed")

par(mfrow = c(2,2))
persp(xs, ys, z,
      main = "log-likelihood of Poisson distribution",
      zlab = "log-likelihood", xlab = "alpha1", ylab = "alpha2",
      theta = 15, phi = 10, col = "springgreen", shade = 0.3,
      ticktype = "detailed")

persp(xs, ys, z,
      main = "log-likelihood of Poisson distribution",
      zlab = "log-likelihood", xlab = "alpha1", ylab = "alpha2",
      theta = 90, phi = 10, col = "springgreen", shade = 0.3,
      ticktype = "detailed")

persp(xs, ys, z,
      main = "log-likelihood of Poisson distribution",
      zlab = "log-likelihood", xlab = "alpha1", ylab = "alpha2",
      theta = -40, phi = 10, col = "springgreen", shade = 0.3,
      ticktype = "detailed")

persp(xs, ys, z,
      main = "log-likelihood of Poisson distribution",
      zlab = "log-likelihood", xlab = "alpha1", ylab = "alpha2",
      theta = -180, phi = 10, col = "springgreen", shade = 0.3,
      ticktype = "detailed")
dev.off()

# Newton-Raphson
nr <- function(x = oilspills, alpha1_init, alpha2_init, epsilon = 1e-5) {
  alpha1_new <- alpha1_init; alpha2_new <- alpha2_init  # Initialize variables
  convergence_flag <- TRUE; count <- 0

  while(convergence_flag) {  # Loop through algorithm
    alpha1_old <- alpha1_new; alpha2_old <- alpha2_new

    alpha_vec <- c(alpha1_old, alpha2_old) + h(x, alpha1_old, alpha2_old)
    alpha1_new <- alpha_vec[1]; alpha2_new <- alpha_vec[2]; count <- count + 1

    convergence_vec <- relative_convergence(alpha1_old, alpha1_new, alpha2_old,
                                            alpha2_new, epsilon)
    convergence_value <- convergence_vec[1]; convergence_flag <- convergence_vec[2]
  }
  cat('Iterations: ', count, '\n')
```

```r
  return(c(alpha_vec, count))
}

# Test values for Newton-Raphson
test_initializations <- expand.grid(-1:3, -1:3)
# Reference: https://stackoverflow.com/questions/4227223/convert-a-list-to-a-data-frame
test_initializations <- matrix(unlist(test_initializations),
                               ncol = length(test_initializations))

test_results <- mapply(function(a1, a2) {
    tryCatch(nr(alpha1_init = a1, alpha2_init = a2),
             error = function(e) NA,
             warning = function(w) NA,
             message = function(c) NA
    )
  }, a1 = test_initializations[,1], a2 = test_initializations[,2]
)

test_results <- do.call(rbind, test_results)
test_results <- as.data.frame(test_results)
test_results <- cbind(test_initializations, test_results)
colnames(test_results) <- c("alpha1_init", "alpha2_init", "alpha1", "alpha2", "Iterations")
kable(test_results, format = "latex", caption = "Results for Newton-Raphson")

nr(alpha1_init = -1, alpha2_init = -1)
nr_fisher(alpha1_init = 1, alpha2_init = -1)

# part b
fisher_information <- function(x = oilspills,
                               alpha1, alpha2) {  # Fisher Information matrix
  n <- x[,2]; b1 <- x[,3]; b2 <- x[,4]
  lambda_i <- alpha1 * b1 + alpha2 * b2
  top_left <- -sum(b1^2 / lambda_i)
  bottom_left <- top_right <- -sum(b1 * b2 / lambda_i)
  bottom_right <- -sum(b2^2 / lambda_i)
  matrix(c(top_left, bottom_left, top_right, bottom_right), ncol = 2)
}

h_fisher <- function(x, alpha1, alpha2) {  # h function with Fisher scoring
  return(-solve(fisher_information(x, alpha1, alpha2)) %*%
           g3(x, alpha1, alpha2))
}

nr_fisher <- function(x = oilspills, alpha1_init, alpha2_init, epsilon = 1e-5) {
  alpha1_new <- alpha1_init; alpha2_new <- alpha2_init  # Initialize variables
  convergence_flag <- TRUE; count <- 0

  while(convergence_flag) {  # Loop through algorithm
    alpha1_old <- alpha1_new; alpha2_old <- alpha2_new  # Update old variables

    # Update new variables
    alpha_vec <- c(alpha1_old, alpha2_old) + h_fisher(x, alpha1_old, alpha2_old)
    alpha1_new <- alpha_vec[1]; alpha2_new <- alpha_vec[2]; count <- count + 1

    # Check for convergence
    convergence_vec <- relative_convergence(alpha1_old, alpha1_new, alpha2_old,
                                            alpha2_new, epsilon)
    convergence_value <- convergence_vec[1]; convergence_flag <- convergence_vec[2]
  }
  cat('Iterations: ', count, '\n')
  return(c(alpha_vec, count))
}

test_results_fisher <- mapply(function(a1, a2) {
  tryCatch(nr_fisher(alpha1_init = a1, alpha2_init = a2),
           error = function(e) NA,
           warning = function(w) NA,
           message = function(c) NA
  )
```

```
  }, a1 = test_initializations[,1], a2 = test_initializations[,2]
)

test_results_fisher <- do.call(rbind, test_results_fisher)
test_results_fisher <- as.data.frame(test_results_fisher)
test_results_fisher <- cbind(test_initializations, test_results_fisher)
colnames(test_results_fisher) <- c("alpha1_init", "alpha2_init", "alpha1", "alpha2", "Iterations")
kable(test_results_fisher, format = "latex", caption = "Results for Newton-Raphson with Fisher Scoring")

# part e
backtrack_check <- function(x = oilspills,
                            alpha1_old_btc,
                            alpha1_test,
                            alpha2_old_btc,
                            alpha2_test,
                            alpha_btc,
                            backtracks_btc,
                            backtrack_flag_btc) {
  step_old <- g0(x = x,  # Calculate steps
                 alpha1 = alpha1_old_btc,
                 alpha2 = alpha2_old_btc)
  step_test <- g0(x = x,
                  alpha1 = alpha1_test,
                  alpha2 = alpha2_test)

  if(step_test < step_old) {  # Check if "downhill" or sideways
    alpha_btc <- alpha_btc / 2
    backtracks_btc <- backtracks_btc + 1
  } else {  # Stop tuning alpha
    backtrack_flag_btc <- FALSE
  }
  return(c(alpha_btc, backtracks_btc, backtrack_flag_btc))
}

alpha_tuner <- function(x = oilspills,
                        alpha1_old_at,
                        alpha2_old_at,
                        alpha_at,
                        backtracks_at) {
  backtrack_flag <- TRUE  # (Re-)Initialize blacktracking variables

  while (backtrack_flag) {  # Loop until a suitable alpha is found
    # theta_test <- c(alpha1_old_at, alpha2_old_at) + # trying to find first update
      # alpha_at * h_fisher(x, alpha1_old_at, alpha2_old_at)
    theta_test <- c(alpha1_old_at, alpha2_old_at) + # trying to find first update
      alpha_at * g3(x, alpha1_old_at, alpha2_old_at)
    backtrack_vec <- backtrack_check(x = x,
                                     alpha1_old_btc = alpha1_old_at,
                                     alpha1_test = theta_test[1],
                                     alpha2_old_btc = alpha2_old_at,
                                     alpha2_test = theta_test[2],
                                     alpha_btc = alpha_at,
                                     backtracks_btc = backtracks_at,
                                     backtrack_flag_btc = backtrack_flag)
    alpha_at <- backtrack_vec[1]; backtracks_at <- backtrack_vec[2]
    backtrack_flag <- backtrack_vec[3]
  }
  return(c(alpha_at, backtracks_at))
}

steepest_ascent <- function(x = oilspills,
                            alpha1_init,
                            alpha2_init,
                            epsilon = 1e-10) {
  alpha1_new <- alpha1_old <- alpha1_init  # Initialize variables
  alpha2_new <- alpha2_old <- alpha2_init
  convergence_flag <- TRUE; count <- 0; backtracks <- 0; alpha <- 1

  while(convergence_flag) {  # Loop through algorithm
```

```
    alpha1_old <- alpha1_new; alpha2_old <- alpha2_new; alpha <- 1

    # Tune alpha
    alpha_tune_vec <- alpha_tuner(x = x,
                                  alpha1_old_at = alpha1_old,
                                  alpha2_old_at = alpha2_old,
                                  alpha_at = alpha,
                                  backtracks_at = backtracks)
    alpha <- alpha_tune_vec[1]; backtracks <- alpha_tune_vec[2]

    # Update old / new variables
    # theta_vec <- c(alpha1_old, alpha2_old) + alpha * h_fisher(x, alpha1_old, alpha2_old)
    theta_vec <- c(alpha1_old, alpha2_old) + alpha * g3(x, alpha1_old, alpha2_old)
    alpha1_new <- theta_vec[1]; alpha2_new <- theta_vec[2]; count <- count + 1

    # Check for convergence
    convergence_vec <- relative_convergence(alpha1_old, alpha1_new, alpha2_old,
                                            alpha2_new, epsilon)
    convergence_flag <- convergence_vec[1]; convergence_value <- convergence_vec[2]
  }
  cat('Iterations: ', count, '\n')
  cat('Backtracks: ', backtracks, '\n')

  return(c(theta_vec, count, backtracks))
}

test_results_steepest_ascent_identity <- mapply(function(a1, a2) {
  tryCatch(steepest_ascent(alpha1_init = a1, alpha2_init = a2),
           error = function(e) NA,
           warning = function(w) NA,
           message = function(c) NA
  )
  }, a1 = test_initializations[,1], a2 = test_initializations[,2]
)

test_results_steepest_ascent_identity <- do.call(rbind, test_results_steepest_ascent_identity)
test_results_steepest_ascent_identity <- as.data.frame(test_results_steepest_ascent_identity)
test_results_steepest_ascent_identity <- cbind(test_initializations, test_results_steepest_ascent_identit
y)
colnames(test_results_steepest_ascent_identity) <- c("alpha1_init", "alpha2_init", "alpha1", "alpha2", "It
erations", "Backtracks")
kable(test_results_steepest_ascent_identity, format = "latex", caption = "Results for Steepest Ascent with
 Identity Matrix")

test_results_steepest_ascent_fisher <- mapply(function(a1, a2) {
  tryCatch(steepest_ascent(alpha1_init = a1, alpha2_init = a2),
           error = function(e) NA,
           warning = function(w) NA,
           message = function(c) NA
  )
  }, a1 = test_initializations[,1], a2 = test_initializations[,2]
)

test_results_steepest_ascent_fisher <- do.call(rbind, test_results_steepest_ascent_fisher)
test_results_steepest_ascent_fisher <- as.data.frame(test_results_steepest_ascent_fisher)
test_results_steepest_ascent_fisher <- cbind(test_initializations, test_results_steepest_ascent_fisher)
colnames(test_results_steepest_ascent_fisher) <- c("alpha1_init", "alpha2_init", "alpha1", "alpha2", "Iter
ations", "Backtracks")
kable(test_results_steepest_ascent_fisher, format = "latex", caption = "Results for Steepest Ascent with F
isher Matrix")
```