# SQP Methods for QN2 Quasi Newton Acceleration of the EM Algorithm

Final Project
CSCI-GA.2945 Numerical Optimization
Justin Mao-Jones
New York University

December 14, 2014

## 1 Introduction

In 1977, Dempster, Laird, and Rubin (henceforth abbreviated DLR) proposed the EM algorithm, which has since become, and continues to be, a popular method for computing maximum likelihood estimates from incomplete data. Wu [1983] showed that under certain properties, the sequence of EM iterates converge to a unique maximum likelihood estimate.

A drawback of the EM algorithm is its tendency towards slow convergence. In certain practical applications, the EM algorithm has been shown to exhibit sublinear convergence. For examples see Lange [1995], Jamshidian and Jennrich [1993, 1997].

Modifications and extensions of the EM algorithm have been proposed to speed up convergence, and are often referred to as "accelerators". In this project, I focus on the QN2 introduced by Jamshidian and Jennrich [1997]. QN2 has been referred to as a Quasi Newton accelerator of the EM algorithm. I explain the structure of the QN2 algorithm in relation to the underlying EM concepts and conventional Quasi Newton optimization concepts. I then exhibit and discuss the results of an application of the QN2 algorithm to deriving MLE parameter estimates of a mixture of poissons. Jamshidian and Jennrich [1997] applied QN2 to a mixture of 2 poissons. I generalize this to a mixture of $n$ poissons.

For the sake of the interested reader, I include the major steps of the derivation of both EM and QN2 algorithms. An additional goal of this project is to present QN2 in such a way that a reader familiar with Quasi Newton methods, but with little experience in either EM and QN2 could understand the underlying theory and implement both the EM and QN2 algorithms.

## 2 EM Algorithm

The EM algorithm, otherwise known as expectation-maximization, is a method for computing the maximum likelihood parameter estimates from incomplete data. Incomplete

data is a set of data in which some or all observations contain variables with missing data.

To use an illustrative example, consider a set of $m$ randomly sampled observations $\{x_1, ..., x_m\}$. Suppose that we would like to explore the possibility that each of these observations comes from one of $n$ poisson-distributed populations with unknown parameters $\theta = (\gamma_1, ..., \gamma_n, \lambda_1, ..., \lambda_n)^T$, where $\gamma_r$ and $\lambda_r$ are the parameters of population $r$. This construction is known as a mixture of poissons. We seek to estimate $\theta$ using maximum likelihood. We do not know which of the $n$ populations the observation $x_i$ corresponds to. Let $z_i \in \{1, ..., n\}$ denote the identity of the population that $x_i$ belongs to. In this example, $\{x_1, ..., x_m\}$ is incomplete data, $\{z_1, ..., z_m\}$ is missing data, and $\{(x_1, z_1), ..., (x_m, z_m)\}$ is the complete (unknown) data.

The probability distribution of an incomplete observation $x_i$ conditional on $z_i$ is:

$$p(x_i = j | z_i = r, \theta) = \frac{e^{-\lambda_r} \lambda_r^j}{j!} \qquad \lambda_r \geq 0$$

The probability distribution of the missing data is:

$$p(z_i = r|\theta) = \gamma_r \qquad \text{where} \qquad \sum_{r=1}^{n} \gamma_r = 1 \qquad \text{and} \qquad 0 \leq \gamma_r \leq 1$$

The probability distribution of the complete data is:

$$p(x_i = j, z_i = r|\theta) = p(x_i = j|z_i = r, \theta) p(z_i = r|\theta) = \gamma_r \frac{e^{-\lambda_r} \lambda_r^j}{j!} \tag{1}$$

We wish to compute the maximum likelihood estimates, and so we begin with the log likelihood function, which is defined as:

$$l(\theta) = \log(\prod_{i=1}^{m} p(x_i|\theta)) = \sum_{i=1}^{m} \log(p(x_i|\theta))$$

This is the point at which a method such as the EM algorithm becomes useful. Since $z_i$ are unobserved, solving for the roots of the derivative of $l(\theta)$ can be an intractable problem:

$$\sum_{i=1}^{m} \log(p(x_i|\theta)) = \sum_{i=1}^{m} \log \left( \sum_{r=1}^{n} p(x_i, z_i|\theta)) \right)$$

Fortunately, $l(\theta)$ can be converted into a more manageable form. Consider the probability distribution $p(\vec{z}|\vec{x}, \phi)$. Note that this function is parameterized by $\phi$ and not $\theta$. Per the EM literature:

$$\begin{aligned} l(\theta) = \log(p(\vec{x}|\theta)) &= \int_{\vec{z}} p(\vec{z}|\vec{x}, \phi) \log(p(\vec{x}|\theta)) d\vec{z} \\ &= E[\log(p(\vec{x}, \vec{z}|\theta))|\vec{x}, \phi] - E[\log(p(\vec{z}|\vec{x}, \theta))|\vec{x}, \phi] \\ &= Q(\theta|\phi) - H(\theta|\phi) \end{aligned} \tag{2}$$

where $Q(\theta|\phi)$ and $H(\theta|\phi)$ are defined as the left and right conditional expectations, respectively. In the words of Lange [1995], this derivation can seem "slightly mysterious." In order to shed some light on this result, I will show how it follows from our specific mixture of poissons problem. We will use the following identity, which follows from the assumption that our observations are independent:

$$1 = \int_{\vec{z}} p(\vec{z}|\vec{x}, \phi) d\vec{z} = \prod_{j=1}^{m} \int_{z_j} p(z_j|x_j, \phi) dz_j$$

The previous identity is true due to the identity $\int_{z_j} p(z_j|x_j, \phi) dz_j = 1$. Now, multiply $l(\theta)$ by $\int_{\vec{z}} p(\vec{z}|\vec{x}, \vec{\phi}) d\vec{z} = 1$:

$$l(\theta) = \left( \sum_{i=1}^{m} \log(p(x_i|\theta) \right) \prod_{j=1}^{m} \int_{z_j} p(z_j|x_j, \phi) dz_j$$

$$= \left( \sum_{i=1}^{m} \int p(z_i|x_i, \phi) \log(p(x_i|\theta) dz_i \right) \prod_{j=1}^{m} \int_{z_j} p(z_j|x_j, \phi) dz_j = \sum_{i=1}^{m} \int p(z_i|x_i, \phi) \log(p(x_i|\theta) dz_i$$

The previous result was derived by putting the $i^{th}$ integral inside of the summation and the log inside of the integral. Now we use Bayes' theorem to derive a useful result:

$$\sum_{i=1}^{m} \int p(z_i|x_i, \phi) \log(p(x_i|\theta)) dz_i = \sum_{i=1}^{m} \int p(z_i|x_i, \phi) \log \left( p(x_i, z_i|\theta) \frac{p(x_i|\theta)}{p(x_i, z_i|\theta)} \right) dz_i$$

$$= \sum_{i=1}^{m} \int p(z_i|x_i, \phi) \log \left( \frac{p(x_i, z_i|\theta)}{p(z_i|x_i, \theta)} \right) dz_i$$

$$= \sum_{i=1}^{m} \int p(z_i|x_i, \phi) \log \left( p(x_i, z_i|\theta) \right) dz_i - \sum_{i=1}^{m} \int p(z_i|x_i, \phi) \log \left( p(z_i|x_i, \theta) \right) dz_i$$

$$= Q(\theta|\phi) - H(\theta|\phi)$$

This result is useful, because $\nabla_\theta l(\theta)$ evaluated at $\theta = \phi$ makes $H(\theta|\phi)$ go to 0:

$$\nabla_\theta H(\theta|\phi)|_{\theta=\phi} = \nabla_\theta \sum_{i=1}^{m} \int p(z_i|x_i, \phi) \log \left( p(z_i|x_i, \theta) \right) dz_i$$

$$= \sum_{i=1}^{m} \int p(z_i|x_i, \phi) \nabla_\theta|_{\theta=\phi} \log \left( p(z_i|x_i, \theta) \right) dz_i = \sum_{i=1}^{m} \int \frac{p(z_i|x_i, \phi)}{p(z_i|x_i, \phi)} \nabla_\theta|_{\theta=\phi} p(z_i|x_i, \theta) dz_i$$

$$= \sum_{i=1}^{m} \nabla_\theta|_{\theta=\phi} \int p(z_i|x_i, \theta) dz_i = \sum_{i=1}^{m} \nabla_\theta|_{\theta=\phi} 1 = 0$$

Thus, we have the following EM identity:

$$\nabla_\theta l(\theta)|_{\theta=\phi} = \nabla_\theta Q(\theta|\phi)|_{\theta=\phi} \tag{3}$$

One way to think about this identity is that it shows where $\nabla_\theta l(\theta)|_{\theta=\phi}$ intersects with a more tractable function $\nabla_\theta Q(\theta|\phi)|_{\theta=\phi}$ in an extended parameter space. The EM algorithm moves along this intersection in the hopes of finding a zero.

This brings us to the specific steps of the EM algorithm, which works as follows. Given an iterate $\theta_k$, the next iterate $\theta_{k+1}$ is defined as the $\theta$ that maximizes $Q(\theta|\phi)$ where $\phi$ is set to $\theta_k$.

$$\theta_{k+1} = \arg\max_\theta Q(\theta|\theta_k) = M(\theta_k)$$

The function $M(\theta_k)$ is known as the EM operator. The algorithm begins with an initial parameter set $\theta_0$ and terminates when either $|M(\theta_k) - \theta_k|$ is small enough or a maximum number of iterations has been reached.

**Algorithm 2.1** Expectation-Maximization

1: Initialize $\theta_0$; Define **maxit**, **ftol**
2: $k = 0$
3: **while** $|M(\theta_k) - \theta_k| <$ **ftol** and $k <$ **maxit do**
4: $\quad k = k + 1$
5: $\quad \theta_{k+1} = M(\theta_k)$
6: **end while**

# 3 QN2 Algorithm

The QN2 Algorithm by Jamshidian and Jennrich [1997] is described as a Quasi Newton method using Broyden-Fletcher-Goldfarb-Shanno (BFGS) symmetric rank 2 updating. I will show how it can be applied with Sequential Quadratic Programming to estimate n-dimensional mixture models with equality constraints.

To begin, we define the following:

$$g(\theta_k) = \nabla_\theta Q(\theta|\phi)|_{\theta=\theta_k,\phi=\theta_k} = \nabla_\theta Q(\theta_k|\theta_k) \tag{4}$$

$$\tilde{g}(\theta_k) = M(\theta_k) - \theta_k \tag{5}$$

The first function $g(\theta_k)$ is the gradient of $Q$ with respect to $\theta$ evaluated at $\theta = \theta_k$ and $\phi = \theta_k$. To simplify notation, I re-write this as $\nabla_\theta Q(\theta_k|\theta_k)$. Note that we will never take a derivative with respect to $\phi$ The second function is the would-be step size of the EM operator $M$.

Jamshidian and Jennrich [1993] showed the following property of the EM step:

$$\tilde{g}(\theta_k) \approx -(\nabla^2_\theta Q(\hat{\theta}|\hat{\theta}))^{-1} g(\theta_k) \tag{6}$$

where $\nabla^2_\theta Q$ is the Hessian of $Q$ and $\hat{\theta}$ is a local maximum of $l(\theta)$.

We are using a Quasi Newton method, and so would like to approximate $(\nabla^2_\theta l(\theta))^{-1}$, the inverse Hessian of the log likelihood function,

$$(\nabla^2_\theta l(\theta))^{-1} = (\nabla^2_\theta Q(\theta|\phi) - \nabla^2_\theta H(\theta|\phi))^{-1}$$

which, using equations (2) and (6), can be approximated as follows:

$$(\nabla^2_\theta l(\theta))^{-1} g(\theta) \approx -\tilde{g}(\theta) + Sg(\theta)$$

where we define $S$ as a Quasi-Newton approximation to

$$(\nabla^2_\theta l(\theta))^{-1} - (\nabla^2_\theta Q(\hat{\theta}|\hat{\theta}))^{-1}$$

This approximation will be used to define the Quasi-Newton update step. Notice that this is a modification to the conventional Quasi-Newton method, because it contains a floating inverse Hessian approximation of $Q(\hat{\theta}|\hat{\theta})$. The presumption here is that this floating value should aid the overall approximation of the inverse Hessian of the log likelihood, and thus speed up convergence.

The resulting update step to $\theta_k$ defined below and assumes a line search methodology is used to determine $\alpha_k$, which is described in the next section. For ease of reading, I make abbreviations such as $g_k = g(\theta_k)$.

$$\theta_{k+1} = \theta_k - \alpha_k d_k \quad \text{where} \quad d_k = \tilde{g}_k(\theta_k) - S_k g(\theta_k)$$

NOTE: Jamshidian and Jennrich [1997] p.575 contains an error in step a, which incorrectly defines $d_k = -\tilde{g}_k(\theta_k) + S_k g(\theta_k)$.

Jamshidian and Jennrich [1997] point out that $d_k$ can be viewed as a modification to the EM step, which is why they label QN2 as an EM accelerator.

The BFGS update is then derived as follows. Using the notation used by Jamshidian and Jennrich [1997], define:

$$\Delta\tilde{g}_k = \tilde{g}_{k+1} - \tilde{g}_k$$
$$\Delta g_k = g_{k+1} - g_k$$

The classic BFGS update of the inverse Hessian is (see Nocedal and Wright [a], p. 136-140):

$$H_{k+1} = (I - \frac{\Delta\theta_k \Delta g_k^T}{\Delta g_k^T \Delta\theta_k}) H_k (I - \frac{\Delta g_k \Delta\theta_k^T}{\Delta g_k^T \Delta\theta_k}) + \frac{\Delta\theta_k \Delta\theta_k^T}{\Delta g_k^T \Delta\theta_k}$$

$$= H_k - H_k \frac{\Delta g_k \Delta\theta_k^T}{\Delta g_k^T \Delta\theta_k} - \frac{\Delta\theta_k \Delta g_k^T}{\Delta g_k^T \Delta\theta_k} H_k + \frac{\Delta\theta_k \Delta g_k^T H_k \Delta g_k \Delta\theta_k^T}{(\Delta g_k^T \Delta\theta_k)^2} + \frac{\Delta\theta_k \Delta\theta_k^T}{\Delta g_k^T \Delta\theta_k}$$

Thus,

$$\Delta H_k = H_{k+1} - H_k = (1 + \frac{\Delta g_k^T H_k \Delta g_k}{\Delta g_k^T \Delta\theta_k}) \frac{\Delta\theta_k \Delta\theta_k^T}{\Delta g_k^T \Delta\theta_k} - \frac{\Delta\theta_k \Delta g_k^T H_k + (\Delta\theta_k \Delta g_k^T H_k)^T}{\Delta g_k^T \Delta\theta_k}$$

If we construct the inverse Hessian approximation as

$$H_k = (\nabla^2_\theta Q(\hat{\theta}|\hat{\theta}))^{-1} + S_k$$

then, using Eqn (6), it follows that

$$H_k g_k = -\tilde{g}_k + S_k g_k$$
$$H_k g_{k+1} = -\tilde{g}_{k+1} + S_k g_k$$
$$H_k \Delta g_k = -\Delta\tilde{g}_k + S_k \Delta g_k$$

The QN2 BFGS update step is:

$$\Delta S_k = H_{k+1} - H_k = \left(1 + \frac{\Delta g_k^T \Delta \theta_k^*}{\Delta g_k^T \Delta \theta}\right) \frac{\Delta \theta_k \Delta \theta_k^T}{\Delta g_k^T \Delta \theta_k} - \frac{\Delta \theta_k^* \Delta \theta_k^T + (\Delta \theta_k^* \Delta \theta_k^T)^T}{\Delta g_k^T \Delta \theta_k} \tag{7}$$

where

$$\Delta \theta_k^* = -\Delta \tilde{g}_k + S_k \Delta g_k$$

As will be seen in the next section, it is important to initialize $S_0$ to $\mathbf{0}$.

## 3.1 Constraints

The problem of finding the maximum likelihood estimate is often a constrained optimization problem. For example, in the mixture of poissons, we have the following constraints:

$$(\lambda_k)_r \geq 0 \qquad 0 \leq (\gamma_k)_r \leq 1 \qquad r \in \{1, ..., n\}$$
$$\sum_{r=1}^{n} (\gamma_k)_r = 1 \tag{8}$$

Here $(\lambda_k)_r$ and $(\gamma_k)_r$ are defined as the $\lambda$ and $\gamma$ parameters, respectively, of population $r$ at iteration $k$. While I have not performed an exhaustive literature search, it seems there is a tendency in the literature to ignore all but the equality constraint. The presumption then must be that in practical problems, the maximum likelihood estimate exists strictly inside the feasible region.

In the mixture of poissons example, this would make sense. $(\lambda_k)_r = 0$ would require that all observations have value 0, and thus maximum likelihood estimation would not be useful. $(\gamma_k)_r$ cannot become zero, because, as will be shown later, the derivative would go to infinity. Similarly, $(\gamma_k)_r$ cannot become 1 either, since a value of 1 in any $(\gamma_k)_r$ would imply a zero in the other $\gamma$'s.

In order to enforce the inequality parameter constraints, we use the following pseudo line search method:

**Algorithm 3.1** Constraint Enforcement

1: Given $\theta_k$, $d_k$
2: $\alpha_k = 1$
3: **while** $\theta_k + \alpha_k d_k$ violates constraints **do**
4: $\quad \alpha_k = \alpha_k / 2$
5: **end while**
6: return $\alpha_k$

The equality parameter constraint can be handled by appropriate initialization of $\theta_0$ and $S_0$. To explain why, consider that at iteration $k$, we assume the relationship $\sum_{r=1}^{n} [\gamma_k]_r = 1$ holds. Then, in order for Eqn (8) to be satisfied at each successive iteration $k+1$, the step along the $\gamma$ parameters $\Delta\gamma_k = \gamma_{k+1} - \gamma_k = \alpha_k (d_k)_\gamma$ must satisfy the relationship

$$\sum_{r=1}^{n} [\Delta \gamma_k]_r = \sum_{r=1}^{n} [\gamma_{k+1}]_r - [\gamma_k]_r = 0 \tag{9}$$

Thus, $\sum_{r=1}^{n}[\gamma_{k+1}]_r = 1$ if and only if $\sum_{r=1}^{n}[\Delta\gamma_k]_r = 0$. In order for this to be true, it must be true that:

$$\sum_{r=1}^{n}[(d_k)_\gamma]_r = \sum_{r=1}^{n}[(\tilde{g}_k)_\gamma]_r - [(S_k g_k)_\gamma]_r = 0 \tag{10}$$

First, assume that the following is true for $S_k$ and any vector $v \in \Re^{2n}$:

$$\sum_{r=1}^{n}[(S_k v)_\gamma]_r = 0 \tag{11}$$

Second, consider $\sum_{r=1}^{n}[(\tilde{g}_k)_\gamma]_r$:

$$\sum_{r=1}^{n}[(\tilde{g}_k)_\gamma]_r = \sum_{r=1}^{n}[(M(\theta_k) - \theta_k)_\gamma]_r = \sum_{r=1}^{n}[(M(\theta_k))_\gamma]_r - \sum_{r=1}^{n}[\gamma_k]_r = 1 - 1 = 0 \tag{12}$$

which holds true for all $k$. By definition, $M(\theta_k)$ is another estimate of $\theta$ that satisfies all parameter constraints. An explicit example of $M(\theta_k)$ is provided in Section 4

By assumption (11) and Eqn (12), Eqn (10) holds true, and thus Eqn (9) holds for $k$. However, this does not yet prove that Eqn (9) will hold for iteration $k+1$ or any other subsequent iterations.

Thus, now we need to show that:

$$\sum_{r=1}^{n}[(d_{k+1})_\gamma]_r = \sum_{r=1}^{n}[(\tilde{g}_{k+1})_\gamma]_r - [(S_{k+1} g_{k+1})_\gamma]_r = 0 \tag{13}$$

Eqn (12) implies that:

$$\sum_{r=1}^{n}[(\tilde{g}_{k+1})_\gamma]_r = 0$$

And so all that is left is to show that:

$$0 = \sum_{r=1}^{n}[(S_{k+1} g_{k+1})_\gamma]_r = \sum_{r=1}^{n}[((S_k + \Delta S_k) g_{k+1})_\gamma]_r = \sum_{r=1}^{n}[(S_k g_{k+1})_\gamma]_r + \sum_{r=1}^{n}[(\Delta S_k g_{k+1})_\gamma]_r$$

By the assumption in Eqn (11), we already have that $\sum_{r=1}^{n}[(S_k g_{k+1})_\gamma]_r = 0$. Thus, to show that $\sum_{r=1}^{n}[(\Delta S_k g_{k+1})_\gamma]_r = 0$, we need to evaluate the BFGS update, which has three matrix terms: $\Delta\theta_k \Delta\theta_k^T$, $\Delta\theta_k \Delta\theta_k^{*T}$, and $\Delta\theta_k^* \Delta\theta_k^T$. Consider any vector $u \in \Re^{2n}$

$$\Delta\theta_k \Delta\theta_k^T u = \begin{bmatrix} \Delta\gamma_k \\ \Delta\lambda_k \end{bmatrix} \Delta\theta_k^T u \quad \text{which implies} \quad \sum_{r=1}^{n}[(\Delta\theta_k \Delta\theta_k^T u)_\gamma]_r = \Delta\theta_k^T u \sum_{r=1}^{n}[\Delta\gamma_k]_r = 0$$

$$\Delta\theta_k \Delta\theta_k^{*T} u = \begin{bmatrix} \Delta\gamma_k \\ \Delta\lambda_k \end{bmatrix} \Delta\theta_k^{*T} u \quad \text{which implies} \quad \sum_{r=1}^{n}[(\Delta\theta_k \Delta\theta_k^{*T} u)_\gamma]_r = \Delta\theta_k^{*T} u \sum_{r=1}^{n}[\Delta\gamma_k]_r = 0$$

As for the third matrix term:

$$\Delta\theta_k^* \Delta\theta_k^T u = (-\Delta\tilde{g}_k + S_k \Delta g_k) \Delta\theta_k^T u$$

Eqn (12) implies:

$$\sum_{r=1}^{n}[(\Delta\tilde{g}_k)_\gamma]_r = 0$$

And assumption (11) allows that:

$$\sum_{r=1}^{n}[(S_k\Delta g_k)_\gamma]_r = 0$$

Therefore,

$$\sum_{r=1}^{n}[(\Delta S_k u)_\gamma]_r = \sum_{r=1}^{n}[(a\Delta\theta_k\Delta\theta_k^T u + b\Delta\theta_k^*\Delta\theta_k^T u + c\Delta\theta_k\Delta\theta_k^{*T} u)_\gamma]_r = 0 \qquad (14)$$

where $a, b, c$ are constants (which need not be derived). It follows that

$$\sum_{r=1}^{n}[(S_{k+1}u)_\gamma]_r = 0$$

Ergo, Eqn (9) must also be true for $k+1$ and, by induction, must hold for all $k$, as long as $S_0$ is chosen such that:

$$\sum_{r=1}^{n}[(S_0 u)_\gamma]_r = 0$$

Hence, we should choose $S_0 = \mathbf{0}$. This is an interesting result of the BFGS update to QN2 on mixture problems that assume the construction $p(z_i = r|\theta) = \gamma_r$ that allows us to maintain feasibility in the equality constraint. The next section discusses how to construct QN2 as a Newton Lagrange method.

## 3.2 Sequential Quadratic Programming Method of QN2

Another nice property of $p(z_i = r|\theta) = \gamma_r$ for QN2 is that the Lagrange multiplier $\lambda^*$ at the optimal solution is easy to compute. The Lagrangian function of Eqn (2) is:

$$\mathcal{L}(\theta, \lambda) = l(\theta) - \lambda(\sum_{r=1}^{n}\gamma_r - 1)$$

Using Eqn (3), the gradient of the Lagrangian is:

$$\nabla_\theta\mathcal{L}(\theta,\lambda)|_{\theta=\phi} = \nabla_\theta Q(\theta|\phi) - \lambda J^T \quad \text{where} \quad J^T = \begin{bmatrix} \mathbf{1}_n \\ \mathbf{0}_n \end{bmatrix} \begin{matrix} \leftarrow \gamma \\ \leftarrow \lambda \end{matrix}$$

Using our concrete mixture of poissons example:

$$Q(\theta|\phi) = E[\log(p(\vec{x},\vec{z}|\theta)|\vec{x},\phi] = \sum_{r=1}^{n}\sum_{i=1}^{m} p(z_i = r|x_i,\phi)\log(p(x_i, z_i = r|\theta))$$

$$= \sum_{r=1}^{n} \sum_{i=1}^{m} p(z_i = r|x_i, \phi) \log(p(z_i|\theta) p(x_i|z_i = r, \theta))$$

$$= \sum_{r=1}^{n} \sum_{i=1}^{m} p(z_i = r|x_i, \phi) \left(\log(\gamma) + \log(p(x_i|z_i = r, \theta))\right)$$

The gradient of the Lagrangian is:

$$\frac{\partial}{\partial \gamma_r} \mathcal{L}(\theta, \lambda)|_{\theta=\phi} = \frac{1}{\gamma_r} \left[ \sum_{i=1}^{m} p(z_i = r|x_i, \phi) \right] - \lambda$$

Solving for the zero of the gradient yields:

$$\gamma_r = \frac{1}{\lambda} \sum_{i=1}^{m} p(z_i = r|x_i, \phi)$$

Summing over all $\gamma_r$ and setting to one yields:

$$1 = \sum_{r=1}^{n} \gamma_r = \sum_{r=1}^{n} \frac{1}{\lambda} \sum_{i=1}^{m} p(z_i = r|x_i, \phi) = \frac{1}{\lambda} \sum_{i=1}^{m} \sum_{r=1}^{n} p(z_i = r|x_i, \phi) = \frac{m}{\lambda}$$

Thus,

$$\lambda^* = \frac{1}{m}$$

This is a well-known result of mixtures. With it, we can show how to formulate QN2 within the SQP construct. The SQP can be formulated as follows. See Nocedal and Wright [b] for more details.

$$\begin{bmatrix} \nabla^2_{\theta_k} \mathcal{L}(\theta_k, \lambda_k) & -J^T \\ J^T & 0 \end{bmatrix} \begin{bmatrix} d_k \\ \lambda_{k+1} \end{bmatrix} = - \begin{bmatrix} g_k \\ c(\theta_k) \end{bmatrix} \quad \text{where} \quad c(\theta_k) = \sum_{r=1}^{n} [\gamma_k]_r - 1$$

If we initialize the multiplier $\lambda_0 = \lambda^* = 1/m$, as well as initialize $S_0 = \mathbf{0}_{2n \times 2n}$ and $\sum_{r=1}^{n} [\gamma_0]_r = 1$, then $\lambda_k = \lambda^* = 1/m$ for all iterations. Thus, the SQP formulation becomes:

$$\begin{bmatrix} \nabla^2_{\theta_k} l(\theta_k) & -J^T \\ J^T & 0 \end{bmatrix} \begin{bmatrix} d_k \\ \lambda^* \end{bmatrix} = - \begin{bmatrix} g_k \\ 0 \end{bmatrix}$$

Which reduces to:

$$\nabla^2_{\theta_k} l(\theta_k) d_k = -(g_k - \lambda^* J^T)$$

Which is approximated by:

$$d_k = \tilde{g}_k - S_k \bar{g} \quad \text{where} \quad \bar{g}_k = g_k - \lambda^* J^T$$

Note that this has not altered our approximation of $S_k$, since at each BFGS update we are concerned with $\Delta g_k$. Furthermore, it was shown in Eqn (14) that the equality constraints will still hold in all iterations with this new formulation.

## 3.3 Line Search

Jamshidian and Jennrich [1997] use the line search described by Jamshidian and Jennrich [1993], with initial step length equal to 2. I will not refer to that here and instead use the Armijo and Wolfe sufficient increase conditions (i.e. the Strong Wolfe conditions) to perform the line search. These steps are outlined below. The step size $\alpha_k$ is initialized using the output from the constraint enforcement step (3.1) to ensure that all line search function evaluations occur within the feasible region.

**Algorithm 3.2** Strong Wolfe (i.e. Armijo/Wolfe) Line Search

Given $\theta_k$
$\alpha_k =$<constraint enforcement output>
$\eta_s = 10^{-4}$
$\eta_w = 0.99$
$T = 10$
$t = 0$
**while** $l(\theta_k + \alpha_k d_k) - l(\theta_k) \geq \eta_s \alpha_k \bar{g}_k^T d_k$ and $\bar{g}_k^T d_k \leq \eta_w \bar{g}(\theta_k + \alpha_k d_k)^T d_k$ and $t < T$ **do**
    $\alpha_k = \alpha_k / 2$
    $t = t + 1$
**end while**
**if** $t < T$ **then**
    foundalpha = True
**else**
    foundalpha = False
**end if**
return $\alpha_k$, foundalpha

This line search runs until it either finds an $\alpha_k$ that satisfies the Strong Wolfe Conditions or else terminates if the number of line search attempts exceeds the maximum 10 attempts, in which case we consider the line search a failure. If the line search fails, we "restart" QN2 by re-initializing $S$ to $S_0$.

## 3.4 Termination

Jamshidian and Jennrich [1997] use a "relative gradient" merit function proposed by Khalfan et al. [1993] shown below.

$$rg = \max_i \left[ |\bar{g}(\theta_k)|_i \frac{\max\{|\theta_k + \Delta\theta_k|_i, 1\}}{max\{|l(\theta_k + \Delta\theta_k|), 1\}} \right] < f_{tol} \tag{15}$$

An alternative merit function is the 2-norm of $\bar{g}_k$.

## 3.5 Initialization

Given an initial set of parameters $\theta_0$, Jamshidian and Jennrich [1993] recommend running a few iterations of EM before beginning QN2. We do the same here.

As discussed above, set:

$$S_0 = \mathbf{0}_{2n \times 2n} \qquad \theta_0 \text{ feasible} \qquad \lambda^* = \frac{1}{m}$$

## 3.6 Algorithm Summary

For completion, I summarize the major steps of my implementation of the QN2 algorithm below.

**Algorithm 3.3** QN2 Implementation

Given $\theta_0$ feasible
Define $\lambda^* = \frac{1}{m}$
Define maxit
Define $f_{tol}$
$\theta = \text{EM}(\theta_0; \text{ 6 iterations})$ *Algorithm (2.1)*
$\bar{g} = \bar{g}(\theta) = g(\theta) - \lambda^* J^T$ *Eqn (4)*
$\tilde{g} = \tilde{g}(\theta)$ *Eqn (5)*
$S = \mathbf{0}_{2n \times 2n}$
$rg = rg(\theta)$ *Eqn (15)*
$k = 0$
**while** $rg \geq f_{tol}$ and $k <$ maxit **do**
    $k = k + 1$
    $d = \tilde{g} - S\bar{g}$
    $\alpha = \text{ConstraintEnforcement}(\theta, d)$ *Algorithm (3.1)*
    $\alpha$, foundalpha = StrongWolfeLineSearch$(\theta, d, \alpha)$ *Algorithm (3.2)*
    **if** foundalpha == True **then**
        $\Delta\theta = \alpha d$
        $\Delta g = g(\theta + \Delta\theta) - g$
        $\Delta\tilde{g} = \tilde{g}(\theta + \Delta\theta) - \tilde{g}$
        $\Delta\theta^* = \tilde{g} - S\bar{g}$
        $\Delta S = \text{BFGS Update}$ *Eqn (7)*
        $\theta = \theta + \Delta\theta$
        $rg = rg(\theta, g)$
        $\bar{g} = \bar{g} + \Delta g$
        $\tilde{g} = \tilde{g} + \Delta\tilde{g}$
        $S = S + \Delta S$
    **else**
        $S = \mathbf{0}$
    **end if**
**end while**

# 4 Mixture of Poissons

Jamshidian and Jennrich [1997] provide the gradient and EM operator solutions for a two mixture Poisson, while using $\gamma, 1-\gamma$ to model $\gamma_1, \gamma_2$. This approach is fine in 2 dimensions, but becomes significantly more complicated in higher dimensions. Here I list the computations for the generalized n-mixture Poisson problem.

Given a mixture of $n$ poissons parameterized by $\gamma_1, ..., \gamma_n, \lambda_1, ..., \lambda_n$, the gradient of the Lagrange function is as follows:

$$\frac{\partial \mathcal{L}(\theta)}{\partial \gamma_r} = \frac{\partial Q(\theta|\phi)}{\partial \gamma_r}|_{\phi=\theta} - \lambda^* = \frac{1}{\gamma_r}\sum_{j=0}^{J} w_{rj}c_j - \lambda^*$$

$$\frac{\partial \mathcal{L}(\theta)}{\partial \lambda_r} = \frac{\partial Q(\theta|\phi)}{\partial \lambda_r} = \frac{1}{\lambda_r}\sum_{j=0}^{J} j w_{rj}c_j - \sum_{j=0}^{J} w_{rj}c_j$$

$$w_{rj} = p(z=r|x=j,\phi) = \frac{[\gamma_\phi]_r e^{-[\lambda_\phi]_r}[\lambda_\phi]_r^j}{\sum_{s=1}^{n}[\gamma_\phi]_s e^{-[\lambda_\phi]_s}[\lambda_\phi]_s^j}$$

$$c_j = \sum_{i=1}^{m} 1_{x_i=j} \qquad J = \max(x_i)$$

$[\gamma_\phi]$ and $[\lambda_\phi]$ are the $\gamma$ and $\lambda$ elements of $\phi$.

The EM operator $M(\theta_k)$ for the mixture of poissons is determined by solving for the zero of the derivative of the Lagrange function with respect to $\theta$ conditional on $\phi = \theta_k$. At the maximum likelihood estimate, $M(\theta_k) = \theta_k$.

$$[\gamma_{k+1}]_r = \frac{\sum_{j=0}^{J} w_{rj}c_j}{\sum_{j=0}^{J} c_j} \qquad [\lambda_{k+1}]_r = \frac{\sum_{j=0}^{J} j w_{rj}c_j}{\sum_{j=0}^{J} w_{rj}c_j}$$

The log likelihood function is:

$$l(\theta) = \sum_{j=1}^{J} c_j \log\left(\sum_{r=1}^{n} \gamma_r \frac{e^{-\lambda_r}\lambda_r^j}{j!}\right)$$

## 4.1 Computing Poisson Densities

The natural structure of the Poisson density function runs into issues when computed numerically due to the terms $\lambda_r^j$ and $j!$. Therefore, I convert these densities to a different form that is more numerically stable:

$$\frac{e^{-\lambda_r}\lambda_r^j}{j!} \qquad \rightarrow \qquad e^{-\lambda_r + j\log(\lambda_r) - \sum_{i=0}^{j}\log(j)}$$

# 5 Comparison of EM and QN2 SQP on Mixtures of Poissons

My implementation of EM and QN2 SQP is written in the Python programming language version 2.7. A copy of the code is provided as a supplement to this paper.

The scope of my comparison EM and QN2 SQP encompasses "practical problems", rather than edge-case problems. I build a random problem generator, which I use to generate a large sample of problems on which I run EM and QN2 SQP.

From a practical perspective, two high level criteria for comparing algorithms are speed and reliability. From the user's perspective, speed is execution time, and reliability is whether or not the algorithm converges to the solution. Therefore, I compare convergence rates, execution times, convergence failures, and solutions.

I run comparisons on a mixtures of 2 Poissons (sample size 3000), 5 Poissons (sample size 3000), and 10 Poissons (sample size 30000).

## 5.1 Random Problem Generator

I generate hidden samples $z_1, ..., z_m$ from a multinomial distribution parameterized by random p-values. Next, I generate a $\lambda$ values for each of the $n$ Poisson distributions using an exponential distribution with mean 1/10. Then I generate sample observations $x_1, ..., x_m$ where observation $x_i$ generated from Poisson distribution $z_i$, which is parameterized by $\lambda_{z_i}$. Finally, only the sample observations $x_1, ..., x_m$ are passed to each algorithm.

## 5.2 Initialization

For each randomly generated problem, I test three initial starting conditions:

Initialization A:

$$\gamma_0 = [\frac{1}{n(n+1)/2}, ..., \frac{n}{n(n+1)/2}]^T$$
$$\lambda_0 = [1, ..., n]^T$$

Initialization B:

$$\gamma_0 = \{\text{parameters used for random problem generator}\}$$
$$\lambda_0 = \{\text{parameters used for random problem generator}\}$$

Initialization C:

$$\gamma_0 = [\frac{1}{n}, ..., \frac{1}{n}]^T$$
$$\lambda_0 = [1, ..., n]^T$$

As an aside, any point in which $\lambda_1 = \cdots = \lambda_n$ is a stationary point, and very likely not the maximum likelihood estimator.

## 5.3 Results

# References

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *J. R. Statist. Soc.*, B(39):1–38, 1997.

Mori Jamshidian and Robert I. Jennrich. Conjugate gradient acceleration of the em algorithm. *J. Am. Statist. Ass.*, 88:221–228, 1993.

Mortaza Jamshidian and Robert I. Jennrich. Acceleration of the em algorithm by using quasi newton methods. *Journal of the Royal Statistical Society*, 1997.

H. F. Khalfan, R. H. Byrd, and R. B. Schnabel. A theoretical and experimental study of the symmetric rank-one update. *SIAM Journal of Optimization*, (3):1–24, 1993.

Kenneth Lange. A quasi newton acceleration of the em algorithm. *Statist. Sin.*, 5:1–18, 1995.

Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. 2nd edition, a.

Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. 2nd edition, b.

C. F. Jeff Wu. On the convergence properties of the em algorithm. *Ann. Statist.*, 11: 95–103, 1983.