A Detailed assumptions, lemmas, and proofs

A.1 Tools

We begin by stating two general propositions that will be useful. First, we show that a version of Cauchy-Schwartz can be applied to weighted sums of tensors.

Proposition 1. *Tensor array version of Hölder's inequality. Let w be an array of scalars and let $a = (a_1, ..., a_N)$ be an array of tensors, were each a_n is indexed by $i = 1, \ldots, D_A$ (i may be a multi-index—e.g., if A is a $D \times D$ matrix, then $i = (j, k)$, for $j, k \in [D]$ and $D_A = D^2$). Let $p, q \in [1, \infty]$ be two numbers such that $p^{-1} + q^{-1} = 1$. Then*

$$\left\| \frac{1}{N} \sum_{n=1}^{N} w_n a_n \right\|_1 \le \frac{D_A^{\frac{1}{p}}}{N} \left\| w \right\|_p \left\| a \right\|_q .$$

In particular, with $p = q = 2$,

$$\left\| \frac{1}{N} \sum_{n=1}^{N} w_n a_n \right\|_1 \le \sqrt{D_A} \frac{\left\| w \right\|_2}{\sqrt{N}} \frac{\left\| a \right\|_2}{\sqrt{N}}.$$

Proof. The conclusion follows from the ordinary Hölder's inequality applied term-wise to n and Jensen's inequality applied to the indices i.

$$\begin{aligned}
\left\| \frac{1}{N} \sum_{n=1}^{N} w_n a_n \right\|_1 &= \sum_{i=1}^{D_A} \left| \frac{1}{N} \sum_{n=1}^{N} w_n \left(a_n\right)_i \right| \\
&\le \frac{1}{N} \sum_{i=1}^{D_A} \left| \left(\sum_{n=1}^{N} \left|w_n\right|^p \right)^{\frac{1}{p}} \left(\sum_{n=1}^{N} \left|\left(a_n\right)_i\right|^q \right)^{\frac{1}{q}} \right| \text{(Hölder)} \\
&= \frac{1}{N} \left\| w \right\|_p \frac{D_A}{D_A} \sum_{i=1}^{D_A} \left(\sum_{n=1}^{N} \left|\left(a_n\right)_i\right|^q \right)^{\frac{1}{q}} \\
&\le \frac{1}{N} \left\| w \right\|_p D_A \left(\frac{1}{D_A} \sum_{i=1}^{D_A} \sum_{n=1}^{N} \left|\left(a_n\right)_i\right|^q \right)^{\frac{1}{q}} \quad \text{(Jensen applied to } i\text{)} \\
&= \frac{1}{N} \left\| w \right\|_p D_A \left(\frac{1}{D_A} \sum_{n=1}^{N} \left\| a_n \right\|_q^q \right)^{\frac{1}{q}} \\
&= \frac{1}{N} \left\| w \right\|_p D_A^{1-\frac{1}{q}} \left\| a \right\|_q \\
&= \frac{D_A^{\frac{1}{p}}}{N} \left\| w \right\|_p \left\| a \right\|_q .
\end{aligned}$$

□

Next, we prove a relationship between the term-wise difference between matrices and the difference between their operator norms. It is well-known that the minimum eigenvalue of a non-singular matrix is continuous in the entries of the matrix. In the next proposition, we quantify this continuity for the L_1 norm.

Proposition 2. *Let A and B be two matrices. Let $\left\| A^{-1} \right\|_{op} \le C_{op}$ for some finite C_{op}, Then*

$$\left\| A - B \right\|_1 \le \frac{1}{2} C_{op}^{-1} \quad \Rightarrow \quad \left\| B^{-1} \right\|_{op} \le 2 C_{op}.$$

Proof. We will use Schott (2016, Theorem 5.20) and the associated discussion, which states the following general result. Take any matrix norm $\left\|\cdot\right\|$ that satisfies $\left\|I\right\| = 1$, where I is the identity matrix. Then if $\left\|A^{-1}\right\| \left\|A - B\right\| < 1$, then

$$\left\|A^{-1} - \left(A - B\right)^{-1}\right\| \le \frac{\left\|A^{-1}\right\| \left\|A - B\right\|}{1 - \left\|A^{-1}\right\| \left\|A - B\right\|} \left\|A^{-1}\right\|. \tag{2}$$

We will apply equation (2) using the operator norm $\left\|\cdot\right\|_{op}$, for which $\left\|I\right\|_{op} = 1$. First, note that

$$\begin{aligned}
\left\|A^{-1}\right\|_{op} \left\|A - B\right\|_{op} &\le \left\|A^{-1}\right\|_{op} \left\|A - B\right\|_{1} \text{ (ordering of matrix norms)} \\
&\le \frac{1}{2} C_{op} C_{op}^{-1} \\
&= \frac{1}{2},
\end{aligned}$$

so we can apply equation (2). Then

$$\begin{aligned}
\left\|B^{-1}\right\|_{op} &\le \left\|B^{-1} - A^{-1}\right\|_{op} + \left\|A^{-1}\right\|_{op} \text{ (triangle inequality)} \\
&\le \frac{\left\|A^{-1}\right\|_{op} \left\|A - B\right\|_{op}}{1 - \left\|A^{-1}\right\|_{op} \left\|A - B\right\|_{op}} \left\|A^{-1}\right\|_{op} + \left\|A^{-1}\right\|_{op} \text{ (Equation 2)} \\
&\le \left(\frac{\frac{1}{2}}{1 - \frac{1}{2}} + 1\right) \left\|A^{-1}\right\| \\
&\le 2 C_{op}.
\end{aligned}$$

□

A.2 Lemmas

We now prove some useful consequences of our assumptions. The proof roughly proceeds for all $w \in W_\delta$ by the following steps:

1. When δ is small we can make $\left\|\hat{\theta}\left(w\right) - \hat{\theta}_1\right\|_2$ small. (Lemma 2 below.)
2. When $\left\|\theta - \hat{\theta}_1\right\|_2$ is small, then the derivatives $H\left(\theta, w\right)$ are close to their optimal value, $H\left(\hat{\theta}_1, 1_w\right)$. (Lemma 3 and Lemma 4 below.)
3. When the derivatives are close to their optimal values, then $H\left(\theta, w\right)$ is uniformly non-singular. (Lemma 5 below.)
4. When the derivatives are close to their optimal values and $H\left(\theta, w\right)$ is uniformly non-singular we can control the error in $\hat{\theta}_{\mathrm{IJ}} - \hat{\theta}\left(w\right)$ in terms of δ. (Theorem 2 below.)

We begin by showing that the difference between $\hat{\theta}\left(w\right)$ and $\hat{\theta}_1$ for $w \in W_\delta$ can be made small by making δ from Condition 1 small.

Lemma 2. *Small parameter changes. Under Assumptions 1—3 and Condition 1,*

$$\textit{for all } w \in W_\delta, \quad \left\|\hat{\theta}\left(w\right) - \hat{\theta}_1\right\|_2 \le C_{op} \delta.$$

Proof. By a first-order Taylor expansion in θ, for some $\tilde{\theta}$ such that $\left\|\tilde{\theta} - \hat{\theta}_1\right\|_2 \le \left\|\hat{\theta}\left(w\right) - \hat{\theta}_1\right\|_2$,

$$G\left(\hat{\theta}\left(w\right), 1_w\right) = G\left(\hat{\theta}_1, 1_w\right) + H\left(\tilde{\theta}, 1_w\right)\left(\hat{\theta}\left(w\right) - \hat{\theta}_1\right).$$

By Assumption 2, $H\left(\tilde{\theta}, 1_w\right)$ is non-singular. A little manipulation, together with the fact that $G\left(\hat{\theta}\left(w\right), w\right) = G\left(\hat{\theta}_1, 1_w\right) = 0$ gives

$$\begin{aligned}
G\left(\hat{\theta}\left(w\right), 1_w\right) - G\left(\hat{\theta}\left(w\right), w\right) = H\left(\tilde{\theta}, 1_w\right)\left(\hat{\theta}\left(w\right) - \hat{\theta}_1\right) \Rightarrow \\
\hat{\theta}\left(w\right) - \hat{\theta}_1 = H\left(\tilde{\theta}, 1_w\right)^{-1}\left(G\left(\hat{\theta}\left(w\right), 1_w\right) - G\left(\hat{\theta}\left(w\right), w\right)\right).
\end{aligned}$$

Applying Condition 1 and Assumption 2,

$$\begin{aligned}
\left\|\hat{\theta}\left(w\right) - \hat{\theta}_1\right\|_2 &= \left\|H\left(\tilde{\theta}, 1_w\right)^{-1}\left(G\left(\hat{\theta}\left(w\right), 1_w\right) - G\left(\hat{\theta}\left(w\right), w\right)\right)\right\|_2 \\
&\le \left\|H\left(\tilde{\theta}, 1_w\right)^{-1}\right\|_{op}\left\|\left(G\left(\hat{\theta}\left(w\right), 1_w\right) - G\left(\hat{\theta}\left(w\right), w\right)\right)\right\|_2 \\
&\le \sup_{\theta\in\Omega_\theta}\left\|H\left(\theta, 1_w\right)^{-1}\right\|_{op}\left\|\left(G\left(\hat{\theta}\left(w\right), 1_w\right) - G\left(\hat{\theta}\left(w\right), w\right)\right)\right\|_2 \\
&\le C_{op}\left\|G\left(\hat{\theta}\left(w\right), 1_w\right) - G\left(\hat{\theta}\left(w\right), w\right)\right\|_2 \text{ (Assumption 2)} \\
&\le C_{op}\left\|G\left(\hat{\theta}\left(w\right), 1_w\right) - G\left(\hat{\theta}\left(w\right), w\right)\right\|_1 \text{ (relation between norms)} \\
&\le C_{op}\sup_{\theta\in\Omega_\theta}\left\|G\left(\theta, 1_w\right) - G\left(\theta, w\right)\right\|_1 \\
&\le C_{op}\delta. \text{ (Condition 1).}
\end{aligned}$$

□

Because we will refer to it repeatedly, we give the set of θ defined in Lemma 2 a name.

Definition 4. For a given δ, define the region around $\hat{\theta}_1$ given by Lemma 2 as

$$B_{C_{op}\delta} := \left\{\theta : \left\|\theta - \hat{\theta}_1\right\|_2 \le C_{op}\delta\right\}\bigcap\Omega_\theta.$$

In other words, Lemma 2 states that Condition 1 implies $\hat{\theta}\left(w\right) \in B_{C_{op}\delta}$ when $w \in W_\delta$.

Next, we show that closeness in θ will mean closeness in $H\left(\theta, w\right)$.

Lemma 3. *Boundedness and continuity. Under Assumptions 1–5 and Condition 1,*

$$\textit{for all } \theta \in B_{\Delta_\theta}, \quad \sup_{w\in W}\left\|H\left(\theta, w\right) - H\left(\hat{\theta}_1, w\right)\right\|_1 \le DC_wL_h\left\|\theta - \hat{\theta}_1\right\|_2.$$

Proof. For $\theta \in B_{\Delta_\theta}$,

$$\begin{aligned}
\sup_{w\in W}\left\|H\left(\theta, w\right) - H\left(\hat{\theta}_1, w\right)\right\|_1 &= \sup_{w\in W}\left\|\frac{1}{N}\sum_{n=1}^{N}w_n\left(h_n\left(\theta\right) - h_n\left(\hat{\theta}_1\right)\right)\right\|_1 \text{ (by definition)} \\
&\le D\sup_{w\in W}\frac{\left\|w\right\|_2}{\sqrt{N}}\frac{\left\|h\left(\theta\right) - h\left(\hat{\theta}_1\right)\right\|_2}{\sqrt{N}} \text{ (Proposition 1)} \\
&\le DC_w\frac{\left\|h\left(\theta\right) - h\left(\hat{\theta}_1\right)\right\|_2}{\sqrt{N}} \text{ (Assumption 5)} \\
&\le DC_wL_h\left\|\theta - \hat{\theta}_1\right\|_2 \text{ (Assumption 4 and } \theta \in B_{\Delta_\theta}\text{).}
\end{aligned}$$

□

We now combine Lemma 2 and Lemma 3 to show that $H(\theta, w)$ is close to its value at the solution $H\left(\hat{\theta}_1, 1_w\right)$ for sufficiently small δ and for all $\theta \in B_{C_{op}\delta}$.

Lemma 4. *Bounds for difference in parameters. Under Assumptions 1–5 and Condition 1, if $\delta \le \Delta_\theta C_{op}^{-1}$, then*

$$\sup_{\theta \in B_{C_{op}\delta}} \sup_{w \in W_\delta} \left\| H(\theta, w) - H\left(\hat{\theta}_1, 1_w\right) \right\|_1 \le \left(1 + D C_w L_h C_{op}\right) \delta.$$

Proof. By Lemma 2, $\delta \le \Delta_\theta C_{op}^{-1}$ implies that $C_{op}\delta \le \Delta_\theta$ and so $B_{C_{op}\delta} \subseteq B_{\Delta_\theta}$. Consequently, we can apply Lemma 3:

$$\begin{aligned}
\sup_{\theta \in B_{C_{op}\delta}} \sup_{w \in W_\delta} \left\| H(\theta, w) - H\left(\hat{\theta}_1, w\right) \right\|_1 &\le \sup_{\theta \in B_{\Delta_\theta}} \sup_{w \in W_\delta} \left\| H(\theta, w) - H\left(\hat{\theta}_1, w\right) \right\|_1 \\
&\le D C_w L_h \left\| \theta - \hat{\theta}_1 \right\|_2 \text{ (Lemma 3)} \\
&\le D C_w L_h C_{op}\delta \quad \text{(because } \theta \in B_{C_{op}\delta}\text{)}.
\end{aligned}$$

Next, we can use this to write

$$\begin{aligned}
&\sup_{\theta \in B_{C_{op}\delta}} \sup_{w \in W_\delta} \left\| H(\theta, w) - H\left(\hat{\theta}_1, 1_w\right) \right\|_1 \\
&\quad= \sup_{\theta \in B_{C_{op}\delta}} \sup_{w \in W_\delta} \left\| H(\theta, w) - H(\theta, 1_w) + H(\theta, 1_w) - H\left(\hat{\theta}_1, 1_w\right) \right\|_1 \\
&\quad\le \sup_{\theta \in B_{C_{op}\delta}} \sup_{w \in W_\delta} \left\| H(\theta, w) - H(\theta, 1_w) \right\|_1 + \sup_{\theta \in B_{C_{op}\delta}} \sup_{w \in W_\delta} \left\| H(\theta, 1_w) - H\left(\hat{\theta}_1, 1_w\right) \right\|_1 \\
&\quad\le \sup_{\theta \in \Omega_\theta} \sup_{w \in W_\delta} \left\| H(\theta, w) - H(\theta, 1_w) \right\|_1 + \sup_{\theta \in B_{C_{op}\delta}} \sup_{w \in W_\delta} \left\| H(\theta, 1_w) - H\left(\hat{\theta}_1, 1_w\right) \right\|_1 \\
&\quad\le \delta + \sup_{\theta \in B_{C_{op}\delta}} \sup_{w \in W_\delta} \left\| H(\theta, 1_w) - H\left(\hat{\theta}_1, 1_w\right) \right\|_1 \text{ (Condition 1)} \\
&\quad\le \delta + D C_w L_h C_{op}\delta.
\end{aligned}$$

□

The constant that appears multiplying δ at the end of the proof of Lemma 4 will appear often in what follows, so we give it the special name C_{IJ} in Definition 3.

Note that Lemma 4 places a condition on how small δ must be in order for our regularity conditions to apply. Lemma 2 will guarantee that $\hat{\theta}(w) \in B_{C_{op}\delta}$, but if we are not able to make δ arbitrarily small in Condition 1, then we are not guaranteed to ensure that $B_{C_{op}\delta} \subseteq B_{\Delta_\theta}$, will not be able to assume Lipschitz continuity, and none of our results will apply.

Next, using Lemma 4, we can extend the operator bound on H_1^{-1} from Assumption 2 to $H(\theta, w)^{-1}$ for all $w \in W_\delta$, not only for $w = 1_w$.

Lemma 5. *Uniform invertibility of the Hessian. Under Assumptions 1–5 and Condition 1, if $\delta \le \min\left\{\Delta_\theta C_{op}^{-1}, \frac{1}{2} C_{IJ}^{-1} C_{op}^{-1}\right\}$, then*

$$\sup_{\theta \in B_{C_{op}\delta}} \sup_{w \in W_\delta} \left\| H(\theta, w)^{-1} \right\|_{op} \le 2 C_{op}.$$

Proof. By Assumption 2, $\left\| H\left(\hat{\theta}_1, 1_w\right)^{-1} \right\|_{op} \le C_{op}$. So by Proposition 2, it will suffice to select δ so that

$$\sup_{\theta \in B_{C_{op}\delta}} \sup_{w \in W_\delta} \left\| H(\theta, w) - H\left(\hat{\theta}_1, 1_w\right) \right\|_1 \le \frac{1}{2} C_{op}^{-1}. \tag{3}$$

When we can apply Lemma 4, we have

$$\sup_{\theta \in B_{C_{op}\delta}} \sup_{w \in W_\delta} \left\| H(\theta, w) - H\left(\hat{\theta}_1, 1_w\right) \right\|_1 \le C_{\mathrm{IJ}} \delta.$$

So $H(\theta, w)$ will satisfy equation (3) if we can apply Lemma 4 and if

$$\delta \le \frac{1}{2} C_{op}^{-1} C_{\mathrm{IJ}}^{-1}.$$

To apply Lemma 4 we additionally require that $\delta \le \Delta_\theta C_{op}^{-1}$. By taking $\delta \le \min\left\{\Delta_\theta C_{op}^{-1}, \frac{1}{2} C_{op}^{-1} C_{\mathrm{IJ}}^{-1}\right\}$, we satisfy equation (3) and the result follows. □

At last, the upper bound on δ will be sufficient to control the error terms in our approximation. For compactness, we give it the name Δ_δ in Definition 3.

Finally, we state a result that will allow us to define derivatives of $\hat{\theta}(w)$ with respect to w.

Lemma 6. *Inverse function theorem. Under Assumptions 1–5 and Condition 1, and for $\delta \le \Delta_\delta$, there exists a continuous, differentiable function of w, $\hat{\theta}(w)$, such that, for all $w \in W$, $G\left(\hat{\theta}(w), w\right) = 0$.*

Proof. This follows from Lemma 5 and the implicit function theorem. □

By definition, $\hat{\theta}(1_w) = \hat{\theta}_1$.

A.3 Bounding the errors in a Taylor expansion

We are now in a position to use Assumptions 1–5 and Condition 1 to bound the error terms in a first-order Taylor expansion of $\hat{\theta}(w)$. We begin by simply calculating the derivative $d\hat{\theta}(w)/dw$.

Proposition 3. *For any $w \in W$ for which $H\left(\hat{\theta}(w), w\right)$ is invertible, and for any vector $a \in \mathbb{R}^N$,*

$$\frac{d\hat{\theta}(w)}{dw^T}\Big|_w a = -H\left(\hat{\theta}(w), w\right)^{-1} G\left(\hat{\theta}(w), a\right).$$

Proof. Because $G\left(\hat{\theta}(w), w\right) = 0$ for all $w \in W$, by direct calculation,

$$\begin{aligned}
0 &= \frac{d}{dw^T} G\left(\hat{\theta}(w), w\right)\Big|_w a \\
&= \left(\frac{\partial G}{\partial \theta^T} \frac{d\hat{\theta}}{dw^T} + \frac{\partial G}{\partial w^T}\right)\Big|_w a \\
&= H\left(\hat{\theta}(w), w\right) \frac{d\hat{\theta}}{dw^T}\Big|_w a + \left(\frac{\partial}{\partial w^T} \frac{1}{N} \sum_{n=1}^N w_n g_n(\theta)\right)\Big|_w a \\
&= H\left(\hat{\theta}(w), w\right) \frac{d\hat{\theta}}{dw^T}\Big|_w a + \frac{1}{N} \sum_{n=1}^N g_n\left(\hat{\theta}(w)\right) a \\
&= H\left(\hat{\theta}(w), w\right) \frac{d\hat{\theta}}{dw^T}\Big|_w a + G\left(\hat{\theta}(w), a\right).
\end{aligned}$$

Because $H\left(\hat{\theta}(w), w\right)$ is invertible by assumption, the result follows. □

Definition 5. Define

$$\begin{aligned}
\hat{\theta}_{\mathrm{IJ}}(w) &:= \hat{\theta}_1 + \frac{d\hat{\theta}(w)}{dw^T}\Big|_{1_w} (w - 1_w) \\
&= \hat{\theta}_1 - H_1^{-1} G\left(\hat{\theta}_1, w\right). \text{ (because } G\left(\hat{\theta}_1, 1_w\right) = 0\text{)}
\end{aligned}$$

$\hat{\theta}_{\mathrm{IJ}}(w)$ in Definition 5 is the first term in a Taylor series expansion of $\hat{\theta}(w)$ as a function of w. We want to bound the error, $\hat{\theta}_{\mathrm{IJ}}(w) - \hat{\theta}(w)$.

Theorem 2. *Under Assumptions 1–5 and Condition 1, when $\delta \le \Delta_\delta$,*

$$\sup_{w \in W_\delta} \left\| \hat{\theta}_{IJ}(w) - \hat{\theta}(w) \right\|_2 \le 2 C_{op}^2 C_{IJ} \delta^2.$$

Proof. By a one-term Taylor series expansion of $G\left(\hat{\theta}(w), w\right) = 0$ in θ around $\hat{\theta}_1$, we have, for some $\tilde{\theta}$ such that $\left\|\tilde{\theta} - \hat{\theta}_1\right\|_2 \le \left\|\hat{\theta}(w) - \hat{\theta}_1\right\|_2$,

$$0 = G\left(\hat{\theta}(w), w\right) = G\left(\hat{\theta}_1, w\right) + H\left(\tilde{\theta}, w\right)\left(\hat{\theta}(w) - \hat{\theta}_1\right).$$

Because $\delta \in W_\delta$, Lemma 2 implies that $\hat{\theta}(w) \in B_{C_{op}\delta}$. Because $\left\|\tilde{\theta} - \hat{\theta}_1\right\|_2 \le \left\|\hat{\theta}(w) - \hat{\theta}_1\right\|_2$, $\tilde{\theta} \in B_{C_{op}\delta}$ as well. Because $\tilde{\theta} \in B_{C_{op}\delta}$, Lemma 5 implies that $H\left(\tilde{\theta}, w\right)$ is invertible, so we can solve for $\hat{\theta}(w) - \hat{\theta}_1$.

$$\begin{aligned}
\hat{\theta}(w) - \hat{\theta}_1 &= -H\left(\tilde{\theta}, w\right)^{-1} G\left(\hat{\theta}_1, w\right) \\
&= \left(-H\left(\tilde{\theta}, w\right)^{-1} + H\left(\hat{\theta}_1, 1_w\right)^{-1} - H\left(\hat{\theta}_1, 1_w\right)^{-1}\right) G\left(\hat{\theta}_1, w\right) \\
&= \left(H\left(\hat{\theta}_1, 1_w\right)^{-1} - H\left(\tilde{\theta}, w\right)^{-1}\right) G\left(\hat{\theta}_1, w\right) + \hat{\theta}_{\mathrm{IJ}}(w) - \hat{\theta}_1.
\end{aligned}$$

Eliminating $\hat{\theta}_1$ and taking the supremum of both sides we have that

$$\begin{aligned}
&\sup_{w \in W_\delta} \left\| \hat{\theta}_{\mathrm{IJ}}(w) - \hat{\theta}(w) \right\|_2 \\
&\quad = \sup_{w \in W_\delta} \left\| \left(H\left(\hat{\theta}_1, 1_w\right)^{-1} - H\left(\tilde{\theta}, w\right)^{-1}\right) G\left(\hat{\theta}_1, w\right) \right\|_2 \\
&\quad = \sup_{w \in W_\delta} \left\| H\left(\tilde{\theta}, w\right)^{-1} \left(H\left(\tilde{\theta}, w\right) - H\left(\hat{\theta}_1, 1_w\right)\right) H\left(\hat{\theta}_1, 1_w\right)^{-1} G\left(\hat{\theta}_1, w\right) \right\|_2 \\
&\quad \le 2C_{op} \sup_{w \in W_\delta} \left\| \left(H\left(\tilde{\theta}, w\right) - H\left(\hat{\theta}_1, 1_w\right)\right) H\left(\hat{\theta}_1, 1_w\right)^{-1} G\left(\hat{\theta}_1, w\right) \right\|_2 \text{ (Lemma 5)} \\
&\quad \le 2C_{op} \sup_{w \in W_\delta} \left\| H\left(\tilde{\theta}, w\right) - H\left(\hat{\theta}_1, 1_w\right) \right\|_{op} \left\| H\left(\hat{\theta}_1, 1_w\right)^{-1} G\left(\hat{\theta}_1, w\right) \right\|_2 \\
&\quad \le 2C_{op} \sup_{w \in W_\delta} \left\| H\left(\tilde{\theta}, w\right) - H\left(\hat{\theta}_1, 1_w\right) \right\|_2 \left\| H\left(\hat{\theta}_1, 1_w\right)^{-1} G\left(\hat{\theta}_1, w\right) \right\|_2 \text{ (ordering of matrix norms)} \\
&\quad \le 2C_{op} C_{\mathrm{IJ}} \delta \sup_{w \in W_\delta} \left\| H\left(\hat{\theta}_1, 1_w\right)^{-1} G\left(\hat{\theta}_1, w\right) \right\|_2 \text{ (Lemma 4)} \\
&\quad \le 2C_{op}^2 C_{\mathrm{IJ}} \delta \sup_{w \in W_\delta} \left\| G\left(\hat{\theta}_1, w\right) \right\|_2 \text{ (Assumption 2)} \\
&\quad = 2C_{op}^2 C_{\mathrm{IJ}} \delta \sup_{w \in W_\delta} \left\| G\left(\hat{\theta}_1, w\right) - G\left(\hat{\theta}_1, 1_w\right) \right\|_2 \text{ (because } G\left(\hat{\theta}_1, 1_w\right) = 0\text{)} \\
&\quad \le 2C_{op}^2 C_{\mathrm{IJ}} \delta^2 \text{ (Condition 1).}
\end{aligned}$$

□

A.4 Use cases

First, let us state a simple condition under which Assumptions 1–4 hold. It will help to have a lemma for the Lipschitz continuity.

Lemma 7. *Derivative Cauchy Schwartz. Let* $a(\theta) = (a_1(\theta), ..., a_N(\theta))$ *be an array of tensors with multi-index* $i \in [D_A]$*, and let* $\frac{\partial a(\theta)}{\partial \theta} = \left(\frac{\partial}{\partial \theta} a_1(\theta), ..., \frac{\partial}{\partial \theta} a_N(\theta)\right)$ *be an array of tensors of size* $D \times D_A$*. Then*

$$\left\| \frac{\partial}{\partial \theta} \|a(\theta)\|_2 \right\|_2 \leq D_A \left\| \frac{\partial a}{\partial \theta} \right\|_2.$$

Proof. By direct calculation,

$$\begin{aligned}
\left\| \frac{\partial}{\partial \theta} \|a(\theta)\|_2^2 \right\|_2^2 &= \sum_{r=1}^{D} \left(\frac{\partial}{\partial \theta_r} \sum_{n=1}^{N} \sum_{i=1}^{D_A} a_{n,i}(\theta)^2 \right)^2 \\
&= \sum_{r=1}^{D} \left(\sum_{n=1}^{N} \sum_{i=1}^{D_A} 2 a_{n,i}(\theta) \frac{\partial a_{n,i}(\theta)}{\partial \theta_r} \right)^2 \\
&\leq \sum_{r=1}^{D} \left(2 \sum_{i=1}^{D_A} \left(\sum_{n=1}^{N} a_{n,i}(\theta)^2 \right)^{\frac{1}{2}} \left(\sum_{n=1}^{N} \left(\frac{\partial a_{n,i}(\theta)}{\partial \theta_r} \right)^2 \right)^{\frac{1}{2}} \right)^2 \\
&\leq \sum_{r=1}^{D} \left(2 D_A^2 \left(\frac{1}{D_A} \sum_{i=1}^{D_A} \sum_{n=1}^{N} a_{n,i}(\theta)^2 \right)^{\frac{1}{2}} \left(\frac{1}{D_A} \sum_{n=1}^{N} \left(\frac{\partial a_{n,i}(\theta)}{\partial \theta_r} \right)^2 \right)^{\frac{1}{2}} \right)^2 \\
&= 4 D_A^2 \|a\|_2^2 \sum_{r=1}^{D} \left\| \frac{\partial a}{\partial \theta_r} \right\|_2^2 \\
&= 4 D_A^2 \|a\|_2^2 \left\| \frac{\partial a}{\partial \theta} \right\|_2^2.
\end{aligned}$$

By the chain rule,

$$\left\| \frac{\partial}{\partial \theta} \|a(\theta)\|_2 \right\|_2^2 = \frac{1}{4 \|a(\theta)\|_2^2} \left\| \frac{\partial}{\partial \theta} \|a(\theta)\|_2^2 \right\|_2^2 \leq D_A^2 \left\| \frac{\partial a}{\partial \theta} \right\|_2^2.$$

□

Lemma 8. *Let* $a(\theta) \in \mathbb{R}^{D \times D}$ *be a continuously differentiable random matrix with a* $D \times D \times D$ *derivative tensor. (Note that the function, not* θ*, is random. For example,* $\mathbb{E}[a(\theta)]$ *is still a function of* θ*.) Suppose that* $\mathbb{E}[\|a(\theta)\|_2]$ *is finite for all* $\theta \in \Omega_\theta$*. Then, for all* $\theta_1, \theta_2 \in \Omega_\theta$*,*

$$\left| \mathbb{E}[\|a(\theta_1)\|_2] - \mathbb{E}[\|a(\theta_2)\|_2] \right| \leq \sqrt{\mathbb{E}\left[\sup_{\theta \in \Omega_\theta} \left\| \frac{\partial a(\theta)}{\partial \theta} \right\|_2^2 \right]} \|\theta_1 - \theta_2\|_2.$$

Proof. For any tensor a with multi-index i,

$$\begin{aligned}
\left\| \frac{\partial}{\partial \theta} \left\|a\right\|_2^2 \right\|_2^2 &= \sum_{r=1}^{D} \left(\frac{\partial}{\partial \theta_r} \left\|a\right\|_2^2 \right)^2 \\
&= \sum_{r=1}^{D} \left(\frac{\partial}{\partial \theta_r} \sum_{i=1}^{D_A} a_i^2 \right)^2 \\
&= \sum_{r=1}^{D} \left(2 \sum_{i=1}^{D_A} a_i \frac{\partial a_i}{\partial \theta_r} \right)^2 \\
&\le 4 \sum_{r=1}^{D} \sum_{i=1}^{D_A} a_i^2 \sum_{i=1}^{D_A} \left(\frac{\partial a_i}{\partial \theta_r} \right)^2 \quad \text{(Cauchy-Schwartz)} \\
&= 4 \sum_{i=1}^{D_A} a_i^2 \sum_{r=1}^{D} \sum_{i=1}^{D_A} \left(\frac{\partial a_i}{\partial \theta_r} \right)^2 \\
&= 4 \left\|a\right\|_2^2 \left\| \frac{\partial a}{\partial \theta} \right\|_2^2 .
\end{aligned}$$

Consequently,

$$\begin{aligned}
\left\| \frac{\partial}{\partial \theta} \left\|a(\theta)\right\|_2 \right\|_2^2 &= \left\| \frac{1}{2\left\|a(\theta)\right\|_2} \frac{\partial}{\partial \theta} \left\|a(\theta)\right\|_2^2 \right\|_2^2 \\
&= \frac{1}{4\left\|a(\theta)\right\|_2^2} \left\| \frac{\partial}{\partial \theta} \left\|a(\theta)\right\|_2^2 \right\|_2^2 \\
&\le \frac{4\left\|a(\theta)\right\|_2^2}{4\left\|a(\theta)\right\|_2^2} \left\| \frac{\partial}{\partial \theta} a(\theta) \right\|_2^2 \\
&= \left\| \frac{\partial a(\theta)}{\partial \theta} \right\|_2^2 .
\end{aligned}$$

So for any $\theta_1, \theta_2 \in \Omega_\theta$,

$$\begin{aligned}
\left| \mathbb{E}\left[\left\|a(\theta_1)\right\|_2\right] - \mathbb{E}\left[\left\|a(\theta_2)\right\|_2\right] \right| &\le \mathbb{E}\left[\left| \left\|a(\theta_1)\right\|_2 - \left\|a(\theta_2)\right\|_2 \right|\right] \\
&\le \mathbb{E}\left[\left(\sup_{\theta \in \Omega_\theta} \left\| \frac{\partial}{\partial \theta} \left\|a(\theta)\right\|_2 \right\|_2 \right)\right] \left\|\theta_1 - \theta_2\right\|_2 \quad (\theta \text{ is not random}) \\
&\le \mathbb{E}\left[\left(\sup_{\theta \in \Omega_\theta} \left\| \frac{\partial a(\theta)}{\partial \theta} \right\|_2 \right)\right] \left\|\theta_1 - \theta_2\right\|_2 \\
&\le \sqrt{\mathbb{E}\left[\sup_{\theta \in \Omega_\theta} \left\| \frac{\partial a(\theta)}{\partial \theta} \right\|_2^2 \right]} \left\|\theta_1 - \theta_2\right\|_2 .
\end{aligned}$$

The result follows. Note that the bound still holds (though vacuously) if $\mathbb{E}\left[\sup_{\theta \in \Omega_\theta} \left\| \frac{\partial a(\theta)}{\partial \theta} \right\|_2^2\right]$ is infinite. ☐

Proposition 4. *Let Ω_θ be a compact set. Let $g_n(\theta)$ be twice continuously differentiable IID random functions. Define*

$$\begin{aligned}
h_n(\theta) &:= \frac{\partial g_n(\theta)}{\partial \theta} \\
r_n(\theta) &:= \frac{\partial^2 g_n(\theta)}{\partial \theta \partial \theta},
\end{aligned}$$

where $r_n(\theta)$ is a $D \times D \times D$ tensor. Assume that

1a) $\mathbb{E}\left[\sup_{\theta\in\Omega_\theta}\left\|g_n\left(\theta\right)\right\|_2^2\right]<\infty$*;*

1b) $\mathbb{E}\left[\sup_{\theta\in\Omega_\theta}\left\|h_n\left(\theta\right)\right\|_2^2\right]<\infty$*;*

1c) $\mathbb{E}\left[\sup_{\theta\in\Omega_\theta}\left\|r_n\left(\theta\right)\right\|_2^2\right]<\infty$;

2) $\mathbb{E}\left[h_n\left(\theta\right)\right]$ *is non-singular for all* $\theta\in\Omega_\theta$*;*

3) We can exchange expectation and differentiation.

Then $\lim_{N\to\infty}P\left(\textit{Assumptions 1–4 hold}\right)=1$.

Proof. The proof follows from Theorems 9.1 and 9.2 of Keener (2011). We will first show that the expected values of the needed functions satisfy Assumptions 1–4 , and then that the sample versions converge uniformly.

By Jensen's inequality,

$$\mathbb{E}\left[\sup_{\theta\in\Omega_\theta}\left\|g_n\left(\theta\right)\right\|_2\right]=\mathbb{E}\left[\sqrt{\sup_{\theta\in\Omega_\theta}\left\|g_n\left(\theta\right)\right\|_2^2}\right]\le\sqrt{\mathbb{E}\left[\sup_{\theta\in\Omega_\theta}\left\|g_n\left(\theta\right)\right\|_2^2\right]}.$$

Also, for the i^{th} component of $g_n\left(\theta\right)$

$$\mathbb{E}\left[\sup_{\theta\in\Omega_\theta}\left|g_{n,i}\left(\theta\right)\right|\right]\le\mathbb{E}\left[\sup_{\theta\in\Omega_\theta}\left\|g_n\left(\theta\right)\right\|_\infty\right]\le\mathbb{E}\left[\sup_{\theta\in\Omega_\theta}\left\|g_n\left(\theta\right)\right\|_2\right].$$

By Theorem 9.1 of Keener (2011), $\mathbb{E}\left[\left\|g_n\left(\theta\right)\right\|_2^2\right]$, $\mathbb{E}\left[\left\|g_n\left(\theta\right)\right\|_2\right]$, and $\mathbb{E}\left[g_n\left(\theta\right)\right]$ are continuous functions of θ, and because Ω_θ is compact, they are each bounded. Similar reasoning applies to $h_n\left(\theta\right)$ and $r_n\left(\theta\right)$. Consequently we can define

$$\sup_{\theta\in\Omega_\theta}\mathbb{E}\left[\left\|g_n\left(\theta\right)\right\|_2^2\right]=:Q_g^2<\infty$$
$$\sup_{\theta\in\Omega_\theta}\mathbb{E}\left[\left\|h_n\left(\theta\right)\right\|_2^2\right]=:Q_h^2<\infty.$$

Below, these constants will be used to satisfy Assumption 1 and Assumption 3 with high probability.

Because Ω_θ is compact, $\mathbb{E}\left[h_n\left(\theta\right)\right]$ is continuous, $\mathbb{E}\left[h_n\left(\theta\right)\right]$ is non-singular, and the operator norm is a continuous function of $\mathbb{E}\left[h_n\left(\theta\right)\right]$, we can also define

$$\sup_{\theta\in\Omega_\theta}\left\|\mathbb{E}\left[h_n\left(\theta\right)\right]^{-1}\right\|_{op}=:Q_{op}<\infty.$$

Below, this constant be used to satisfy Assumption 2 with high probability.

Finally, we turn to the Lipschitz condition. Lemma 8 implies that

$$\left|\mathbb{E}\left[\left\|h_n\left(\theta_1\right)\right\|_2\right]-\mathbb{E}\left[\left\|h_n\left(\theta_2\right)\right\|_2\right]\right|\le\sqrt{\mathbb{E}\left[\sup_{\theta\in\Omega_\theta}\left\|r_n\left(\theta\right)\right\|_2^2\right]}\left\|\theta_1-\theta_2\right\|_2.$$

Define

$$\Lambda_h=\sqrt{\mathbb{E}\left[\sup_{\theta\in\Omega_\theta}\left\|r_n\left(\theta\right)\right\|_2^2\right]},$$

so that we have shown that $\mathbb{E}\left[\left\|h_n\left(\theta\right)\right\|_2\right]$ is Lipschitz in Ω_θ with constant Λ_h, which is finite by assumption.

We have now shown, essentially, that the expected versions of the quantities we wish to control satisfy Assumptions 1–4 with $N=1$. We now need to show that the sample versions satisfy Assumptions 1–4 with high probability, which will follow from the fact that the sample versions converge uniformly to their expectations by Theorem 9.2 of Keener (2011).

First, observe that Assumption 1 holds with probability one by assumption. For the remaining assumption choose an $\epsilon > 0$, and define

$$\begin{aligned} C_g &:= \sqrt{Q_g^2 + \epsilon} \\ C_h &:= \sqrt{Q_h^2 + \epsilon} \\ C_{op} &:= 2Q_{op} \\ L_h &:= \sqrt{D^4 \Lambda_h^2 + \epsilon}. \end{aligned}$$

By Keener (2011) Theorem 9.2,

$$\sup_{\theta \in \Omega_\theta} \left| \frac{1}{N} \sum_{n=1}^{N} \left\| g_n(\theta) \right\|_2^2 - \mathbb{E}\left[\left\| g_n(\theta) \right\|_2^2 \right] \right| \xrightarrow[N \to \infty]{p} 0.$$

Because

$$\sup_{\theta \in \Omega_\theta} \left| \frac{1}{N} \sum_{n=1}^{N} \left\| g_n(\theta) \right\|_2^2 \right| > Q_g^2 + \epsilon \geq \sup_{\theta \in \Omega_\theta} \mathbb{E}\left[\left\| g_n(\theta) \right\|_2^2 \right] + \epsilon \Rightarrow$$
$$\sup_{\theta \in \Omega_\theta} \left| \frac{1}{N} \sum_{n=1}^{N} \left\| g_n(\theta) \right\|_2^2 - \mathbb{E}\left[\left\| g_n(\theta) \right\|_2^2 \right] \right| > \epsilon,$$

we have

$$P\left(\sup_{\theta \in \Omega_\theta} \left| \frac{1}{N} \sum_{n=1}^{N} \left\| g_n(\theta) \right\|_2^2 \right| \geq Q_g^2 + \epsilon \right) \leq$$
$$P\left(\sup_{\theta \in \Omega_\theta} \left| \frac{1}{N} \sum_{n=1}^{N} \left\| g_n(\theta) \right\|_2^2 - \mathbb{E}\left[\left\| g_n(\theta) \right\|_2^2 \right] \right| \leq \epsilon \right),$$

so

$$P\left(\sup_{\theta \in \Omega_\theta} \left| \frac{1}{N} \sum_{n=1}^{N} \left\| g_n(\theta) \right\|_2^2 \right| \geq C_g^2 \right) \xrightarrow[N \to \infty]{} 0.$$

An analogous argument holds for $\frac{1}{N} \left\| h_n(\theta) \right\|_2^2$. Consequently, $P(\text{Assumption 3 holds}) \xrightarrow[N \to \infty]{} 1$.

We now consider Assumption 2. Again, by Keener (2011) Theorem 9.2 applied to each element of the matrix $h_n(\theta)$, using a union bound over each of the D^2 entries,

$$\sup_{\theta \in \Omega_\theta} \left\| \frac{1}{N} \sum_{n=1}^{N} h_n(\theta) - \mathbb{E}\left[h_n(\theta) \right] \right\|_1 \xrightarrow[N \to \infty]{p} 0.$$

By the converse of Proposition 2, because $\left\| \mathbb{E}\left[h_n(\theta) \right]^{-1} \right\|_{op} \leq Q_{op}$,

$$\left\| \left(\frac{1}{N} \sum_{n=1}^{N} h_n(\theta) \right)^{-1} \right\|_{op} > 2Q_{op} = C_{op} \Rightarrow$$
$$\left\| \frac{1}{N} \sum_{n=1}^{N} h_n(\theta) - \mathbb{E}\left[h_n(\theta) \right] \right\|_1 > \frac{1}{2} Q_{op}^{-1}.$$

Consequently,

$$P\left(\left\|\left(\frac{1}{N}\sum_{n=1}^{N}h_n\left(\theta\right)\right)^{-1}\right\|_{op}\geq C_{op}\right)\leq$$
$$P\left(\left\|\frac{1}{N}\sum_{n=1}^{N}h_n\left(\theta\right)-\mathbb{E}\left[h_n\left(\theta\right)\right]\right\|_1\right)\xrightarrow[N\to\infty]{p}0,$$

and $P\left(\text{Assumption 2 holds}\right)\xrightarrow[N\to\infty]{}1$.

Finally, applying Lemma 8 to $\frac{1}{\sqrt{N}}\left\|h\left(\theta_2\right)\right\|_2$,

$$\begin{aligned}\left|\frac{1}{\sqrt{N}}\left\|h\left(\theta_1\right)\right\|_2-\frac{1}{\sqrt{N}}\left\|h\left(\theta_2\right)\right\|_2\right| &\leq \sup_{\theta\in\Omega_\theta}\left\|\frac{\partial}{\partial\theta}\frac{1}{\sqrt{N}}\left\|h\left(\theta\right)\right\|_2\right\|_2\left\|\theta_1-\theta_2\right\|_2\\ &\leq\frac{D^2}{\sqrt{N}}\sup_{\theta\in\Omega_\theta}\left\|r\left(\theta\right)\right\|_2\left\|\theta_1-\theta_2\right\|_2\\ &=D^2\sqrt{\sup_{\theta\in\Omega_\theta}\frac{1}{N}\left\|r\left(\theta\right)\right\|_2^2}\left\|\theta_1-\theta_2\right\|_2.\end{aligned}$$

Consequently,

$$\left|\frac{1}{\sqrt{N}}\left\|h\left(\theta_1\right)\right\|_2-\frac{1}{\sqrt{N}}\left\|h\left(\theta_2\right)\right\|_2\right|\geq L_h\left\|\theta_1-\theta_2\right\|_2\Rightarrow$$
$$D^2\sqrt{\sup_{\theta\in\Omega_\theta}\frac{1}{N}\left\|r\left(\theta\right)\right\|_2^2}\geq L_h\Rightarrow$$
$$\sup_{\theta\in\Omega_\theta}\frac{1}{N}\left\|r\left(\theta\right)\right\|_2^2-\sup_{\theta\in\Omega_\theta}\mathbb{E}\left[\left\|r_n\left(\theta\right)\right\|_2^2\right]\geq\frac{L_h^2}{D^4}-\sup_{\theta\in\Omega_\theta}\mathbb{E}\left[\left\|r_n\left(\theta\right)\right\|_2^2\right]\Rightarrow$$
$$\sup_{\theta\in\Omega_\theta}\left|\frac{1}{N}\left\|r\left(\theta\right)\right\|_2^2-\mathbb{E}\left[\left\|r_n\left(\theta\right)\right\|_2^2\right]\right|\geq\frac{L_h^2}{D^4}-\Lambda_h^2=\epsilon.$$

However, again by Keener (2011) Theorem 9.2,

$$\sup_{\theta\in\Omega_\theta}\left|\frac{1}{N}\left\|r\left(\theta\right)\right\|_2^2-\mathbb{E}\left[\left\|r_n\left(\theta\right)\right\|_2^2\right]\right|\xrightarrow[N\to\infty]{p}0,$$

so $P\left(\text{Assumption 4 holds}\right)\xrightarrow[N\to\infty]{}1$. □

B Genomics Experiments Details

We demonstrate the Python and R code used to run and analyze the experiments on the genomics data in a sequence of Jupyter notebooks. The output of these notebooks are included below, though they are best viewed in their original notebook form. The notebooks, as well as scripts and instructions for reproducing our analysis in its entirety, can be found in the git repository rgiordan/AISTATS2019SwissArmyIJ.

fit_model_and_save

February 21, 2019

1 Genomics experiment details.

We demonstrate the infinitesimal jackknife on a publicly available data set of mice gene expression in Shoemaker et al. [2015].

Mice were infected with influenza virus, and gene expression was assessed several times after infection, so the observed data consists of expression levels y_{gt} for genes $g = 1, ..., n_g$ and time points $t = 1, ..., n_t$, where in this case $n_g = 1000$ and $n_t = 42$.

This notebook contains the first of three steps in the analysis. In this notebook, we will first load the data and define a basis with a hyperparameter we wish to select with cross validation. We then describe the two stages of our analysis: a regression stage and a clustering stage. We then save the data for further analysis by the notebooks `load_and_refit` and `calculate_prediction_error`.

This notebook assumes you have already followed the instructions in `README.md` to install the necessary packages and create the dataset.

2 Step 1: Initial fit.

```
In [1]: import matplotlib.pyplot as plt
        %matplotlib inline

        import numpy as np
        import inspect
        import os
        import sys
        import time

        np.random.seed(3452453) # nothing special about this seed (we hope)!
```

```
In [2]: from aistats2019_ij_paper import regression_mixture_lib as rm_lib
        from aistats2019_ij_paper import regression_lib as reg_lib
        from aistats2019_ij_paper import sensitivity_lib as sens_lib
        from aistats2019_ij_paper import spline_bases_lib
        from aistats2019_ij_paper import transform_regression_lib as trans_reg_lib
        from aistats2019_ij_paper import loading_data_utils
        from aistats2019_ij_paper import saving_gmm_utils
        from aistats2019_ij_paper import mse_utils

        import plot_utils_lib
```

2.1 The first stage: Regression

2.1.1 Load data

```
In [3]: # Set bnp_data_repo to be the location of a clone of the repo
        # https://github.com/NelleV/genomic_time_series_bnp
        bnp_data_repo = '../../genomic_time_series_bnp'
        y_train, y_test, train_indx, timepoints = loading_data_utils.load_genomics_data(
            bnp_data_repo,
            split_test_train = True,
            train_indx_file = '../fits/train_indx.npy')
```

```
Loading data from:  ../../genomic_time_series_bnp/data/shoemaker2015reprocessed
```

```
In [4]: n_train = np.shape(y_train)[0]
        print('number of genes in training set: \n', n_train)

        n_test = np.shape(y_test)[0]
        print('number of genes in test set: \n', n_test)

        n_genes = n_train + n_test

        test_indx = np.setdiff1d(np.arange(n_genes), train_indx)
        gene_indx = np.concatenate((train_indx, test_indx))
```

```
number of genes in training set:
 700
number of genes in test set:
 300
```

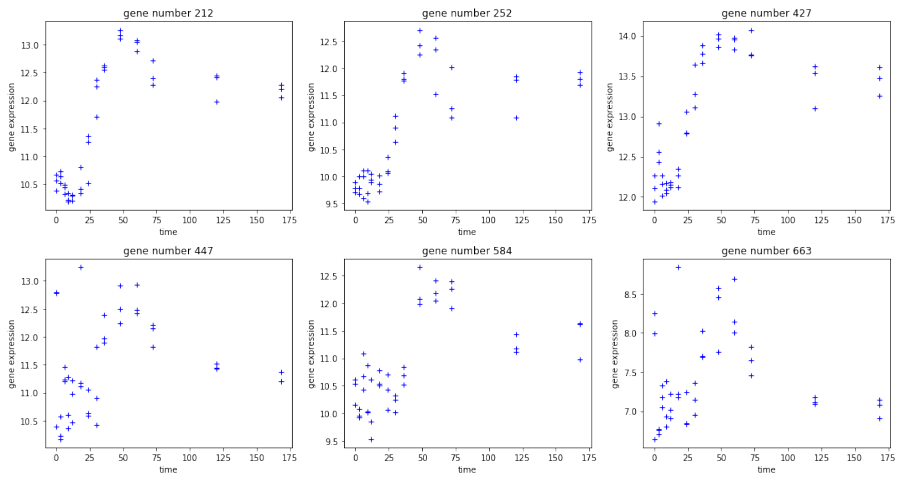
Each gene y_g has 42 observations. Observations are made at 14 timepoints, with 3 replicates at each timepoints.

```
In [5]: n_t = len(timepoints)
        n_t_unique = len(np.unique(timepoints))

        print('timepoints: \n ', timepoints, '\n')
        print('Distinct timepoints: \n', np.sort(np.unique(timepoints)), '\n')
        print('Number of distinct timepoints:', n_t_unique)
```

```
timepoints:
  [0, 0, 0, 3, 3, 3, 6, 6, 6, 9, 9, 9, 12, 12, 12, 18, 18, 18, 24, 24, 24, 30, 30, 30, 36, 36, 3

Distinct timepoints:
 [  0   3   6   9  12  18  24  30  36  48  60  72 120 168]

Number of distinct timepoints: 14
```

Here is the raw data for a few randomly chosen genes.

```
In [6]: f, axarr = plt.subplots(2, 3, figsize=(15,8))

        gene_indx = np.sort(np.random.choice(n_train, 6))

        for i in range(6):
            n = gene_indx[i]
            this_plot = axarr[int(np.floor(i / 3)), i % 3]
            this_plot.plot(timepoints, y_train[n, :].T, '+', color = 'blue');
            this_plot.set_ylabel('gene expression')
            this_plot.set_xlabel('time')
            this_plot.set_title('gene number {}'.format(n))

        f.tight_layout()
```



2.1.2 Define regressors

We model the time course using cubic B-splines. Let α be the degrees of freedom of the B-splines, and this is the parameter we seek to choose using cross-validation.

For a given degrees of freedom, the B-spline basis is given by an $n_t \times n_x$ matrix X_{df}, where the each column of X_{df} is a B-spline basis vector evaluated at the n_t timepoints. Note that n_x increases with increasing degrees of freedom.

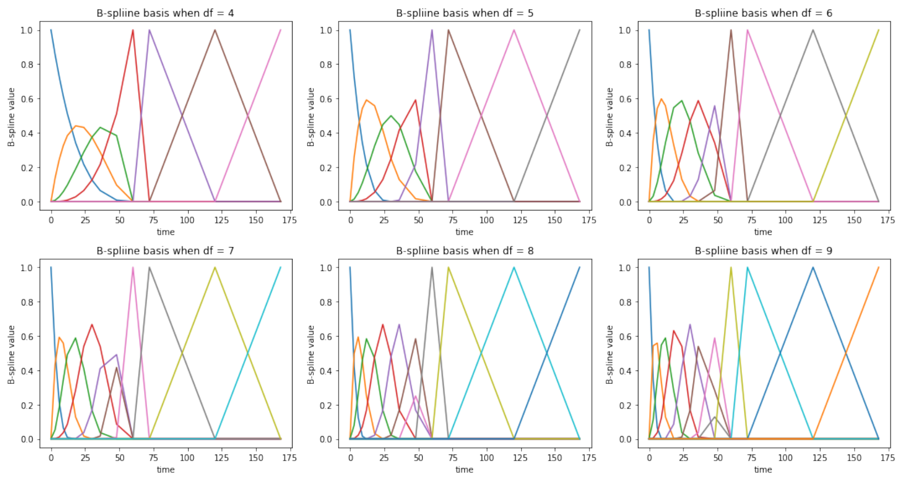
Note that we only use B-splines to smooth the first 11 timepoints. For the last three timepoints, $t = 72, 120, 168$, we use indicator functions on each timepoint as three extra basis vectors. In other words, we append to the regressor matrix three columns, where each column is 1 if $t = 72, 120$, or 168, respectively, and 0 otherwise. We do this to avoid numerical issues in the matrix $X^T X$. Because the later timepoints are more spread out, the B-spline basis are close to zero at the later timepoints, leading to matrices close to being singular.

```
In [7]: # Simulate passing arguments in on the command line so that the notebook
        # looks more like those in ``cluster_scripts``.
        class Args():
            def __init__(self):
                pass

        args = Args()
        args.df = 7
        args.degree = 3
        args.num_components = 10
```

```
In [8]: regressors = spline_bases_lib.get_genomics_spline_basis(
            timepoints, df=args.df, degree=3)

        regs = reg_lib.Regressions(y_train, regressors)
```

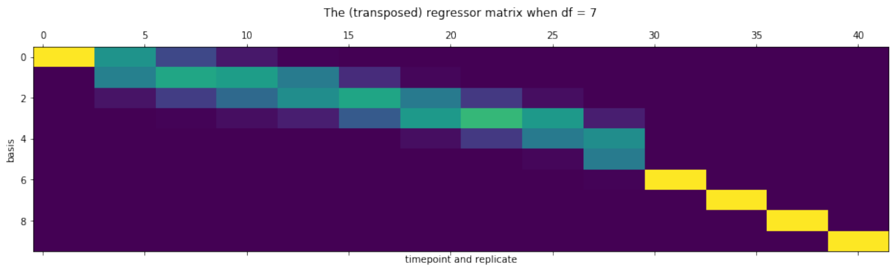
We plot the B-spline matrix for several degrees of freedom below:

```
In [9]: f, axarr = plt.subplots(2, 3, figsize=(15,8))

        i = 0
        for df in [4, 5, 6, 7, 8, 9]:
            _regressors = spline_bases_lib.get_genomics_spline_basis(
                    timepoints, exclude_num=3, df=df)

            this_plot = axarr[int(np.floor(i / 3)), i % 3]
            this_plot.plot(timepoints, _regressors);
            this_plot.set_xlabel('time')
            this_plot.set_ylabel('B-spline value')
            this_plot.set_title('B-spliine basis when df = {}'.format(df))

            i += 1

        f.tight_layout()
```



We display the regressor matrix below.

```
In [10]: plt.matshow(regs.x.T)
         plt.ylabel('basis')
         plt.xlabel('timepoint and replicate')
         plt.title('The (transposed) regressor matrix when df = {}\n'.format(args.df));
```



With the regressor X defined above, for each gene g we model $P\left(y_g|\beta_g,\sigma_g^2\right) = \mathcal{N}\left(y_g|X\beta_g,\sigma_g^2\right)$. In the second stage, we will want to cluster β_g taking into account its uncertainty on each gene. To do this, we wish to estimate the posterior mean $\mathbb{E}[\beta_g|y_g]$ and covariance $\mathrm{Cov}(\beta_g|y_g)$ with flat priors for both β_g and σ_g^2.

For each gene, we estimate the posterior with a mean field variational Bayes (MFVB) approximation $q\left(\sigma_g^2,\beta_g;\hat{\eta}_g\right)$ to the posterior $P\left(\beta_g,\sigma_g^2|y_g\right)$.

In particular, we take $q\left(\sigma_g^2,\beta_g;\hat{\eta}_g\right) = q^*\left(\sigma_g^2\right) q^*\left(\beta_g\right)$, where $q^*\left(\sigma_g^2\right)$ is a dirac delta function, and we optimize over its a location parameter; $q^*\left(\beta_g\right)$ is a Gaussian density and we optimize over its mean and covariance.

The optimal variational approximation has a closed form that is formally identical to the standard frequentist mean and covariance estimate for linear regression. Explicitly, the optimal variational distribution is,

$$q^*(\beta_g) = \mathcal{N}\left(\beta_g \,\middle|\, (X^TX)^{-1}X^Ty_g,\ \hat{\tau}_g(X^TX)^{-1}\right)$$
$$q^*(\sigma_g^2) = \delta\{\sigma_g^2 = \hat{\tau}_g\}$$

where $\hat{\tau}_g = \frac{1}{n_t - n_x}\|y_g - X(X^TX)^{-1}X^Ty_g\|_2^2$.

The advantage of the MVFB construction is that $\hat{\eta}_g$ for $g = 1, ..., n_g$ satisfies set of n_g independent M-estimation objectives, allowing us to apply our infinitesimal jackknife results. Specifically, defining $\theta_{reg} := (\eta_1, ..., \eta_{n_g})$, we wish to minimize

$$\begin{aligned} F_{reg}\left(\theta_{reg}, \alpha\right) &= \sum_{g=1}^{n_g} KL\left(q\left(\sigma_g^2, \beta_g; \eta_g\right) \|| P\left(\beta_g, \sigma_g^2 | y_g\right)\right) \\ &= -\sum_{g=1}^{n_g} \mathbb{E}_q\left[\log P\left(\beta_g, \sigma_g^2 | y_g\right)\right] + \mathbb{E}_q\left[\log q\left(\beta_g, \sigma_g^2 | \eta_g\right)\right] \\ &:= \sum_{g=1}^{n_g} F_{reg,g}\left(\eta_g, \alpha\right). \end{aligned}$$

Our M-estimator, then, is

$$\frac{\partial F_{reg}\left(\theta_{reg}, \alpha\right)}{\partial \theta_{reg}} = 0.$$

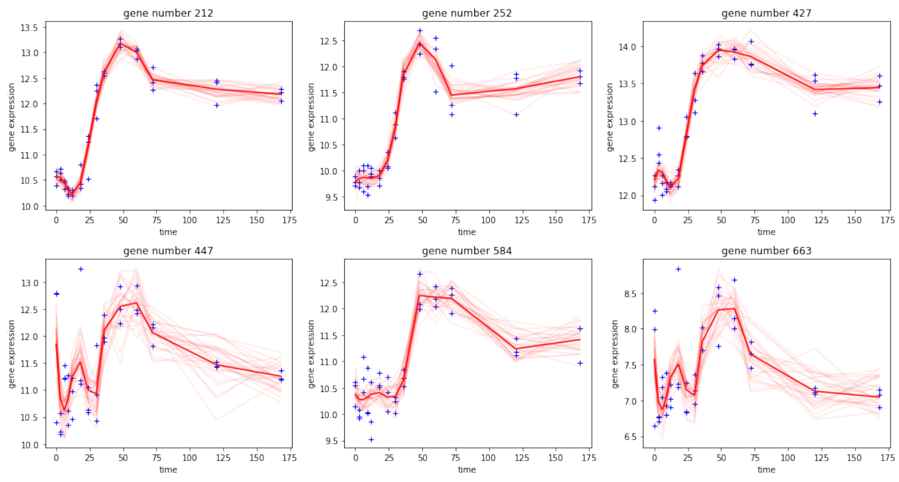
The class `regs` can calculate the optimal variational parameters for each gene. In particular, the variational parameters η_g consist of a variational mean and covariance for β_g, as well as a location estimate for σ_g^2.

```
In [11]: reg_time = time.time()
         opt_reg_params = regs.get_optimal_regression_params()
         reg_time = time.time() - reg_time
         print('Regression time: {} seconds'.format(reg_time))
```

```
Regression time: 0.029132366180419922 seconds
```

Here are what some of the fits look like. Each regression produces a prediction $\hat{y}_g := X\mathbb{E}_q\left[\beta_g\right]$, plotted with the heavy red line above. The light red are predictions when β_g is drawn from $q^*(\beta_g)$; the spread of the light red is intended to give a sense of the covariance of β_g.

```
In [12]: f, axarr = plt.subplots(2, 3, figsize=(15,8))

         for i in range(6):
             n = gene_indx[i]
             this_plot = axarr[int(np.floor(i / 3)), i % 3]
             plot_utils_lib.PlotRegressionLine(
                 timepoints, regs, opt_reg_params, n, this_plot=this_plot)

         f.tight_layout()
```



We also define and save data for the test regressions, which we will use later to evaluate out-of-sample performance. The training regressions will be saved below with the rest of the fit.

```
In [13]: regs_test = reg_lib.Regressions(y_test, regressors)
         test_regression_outfile = '../fits/test_regressions.json'
         with open(test_regression_outfile, 'w') as outfile:
             outfile.write(regs_test.to_json())
```

2.2 The second stage: fit a mixture model.

2.2.1 Transform the parameters before clustering

We are interested in the pattern of gene expression, not the absolute level, so we wish to cluster $\hat{y}_g - \bar{\hat{y}}_g$, where $\bar{\hat{y}}_g$ is the average over time points. Noting that the $n_t \times n_t$ matrix $\mathrm{Cov}_q\left(\hat{y}_g - \bar{\hat{y}}_g\right)$ is rank-deficient because we have subtracted the mean, the final step is to rotate $\hat{y}_g - \bar{\hat{y}}_g$ into a basis where the zero eigenvector is a principle axis and then drop that component.

Call these transformed regression coefficients γ_g and observe that $\mathrm{Cov}_q\left(\gamma_g\right)$ has a closed form and is full-rank. It is these γ_gs that we will cluster in the second stage.

We briefly note that the re-centering operation could have been equivalently achieved by making a constant one of the regressors. We chose this implementation because it also allows the user to cluster more complex, non-linear transformations of the regression coefficients, though we leave this extension for future work.

We note that the transformations described in this section are done automatically in the `GMM` class. We are only calculating these transformations here for exposition.

```
In [14]: # Get the matrix that does the transformation.
         transform_mat, unrotate_transform_mat = \
             trans_reg_lib.get_reversible_predict_and_demean_matrix(regs.x)
         trans_obs_dim = transform_mat.shape[0]
```

If T is the matrix that effects the transformation, then

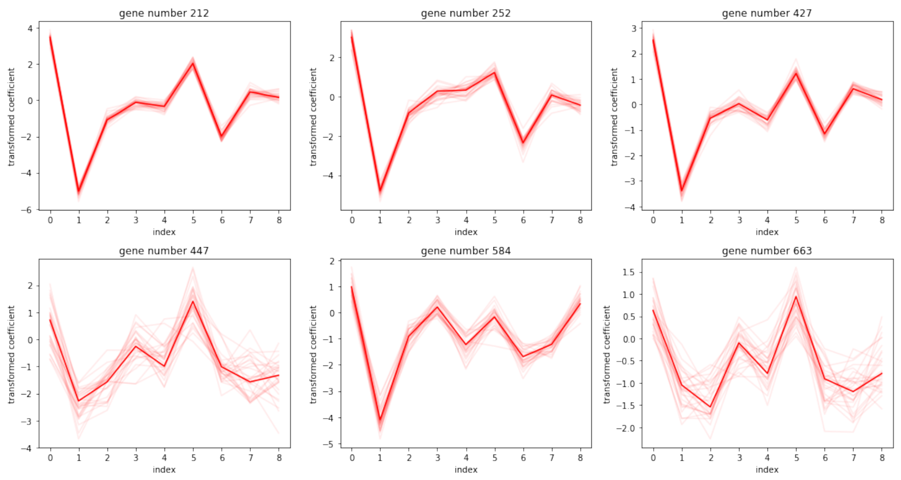
$$\mathbb{E}_q[\gamma_g] = T\mathbb{E}_q[\beta_g]$$
$$\mathrm{Cov}_q(\gamma_q) = T\mathrm{Cov}_q(\beta_g)T^T$$

The transformed parameters are also regression parameters, just in a different space.

```
In [15]: # Apply the transformation
         transformed_reg_params = \
             trans_reg_lib.multiply_regression_by_matrix(
                 opt_reg_params, transform_mat)
```

We now visualize the transformed coefficients and their uncertainty.

```
In [16]: f, axarr = plt.subplots(2, 3, figsize=(15,8))

         transformed_beta = transformed_reg_params['beta_mean']
         transformed_beta_info = transformed_reg_params['beta_info']

         for i in range(6):
             n = gene_indx[i]
             this_plot = axarr[int(np.floor(i / 3)), i % 3]
             this_plot.plot(transformed_beta[n, :], color = 'red');
             this_plot.set_ylabel('transformed coefficient')
             this_plot.set_xlabel('index')
             this_plot.set_title('gene number {}'.format(n))

             # draw from the variational distribution, to plot uncertainties
             for j in range(30):
                 transformed_beta_draw = np.random.multivariate_normal(
                     transformed_beta[n, :], \
                      np.linalg.inv(transformed_beta_info[n]))

                 axarr[int(np.floor(i / 3)), i % 3].plot(transformed_beta_draw,
                                                         color = 'red', alpha = 0.08);

         f.tight_layout()
```



The heavy red lines are the means of the transformed regression coefficients; shaded lines are draws from the variational distribution.

It is these transformed coefficients, γ_g, that we cluster in the second stage.

2.2.2 Estimate an optimal clustering.

We now define a clustering problem for the γ_g. Let n_k be the number of clusters, and $\mu_1, ..., \mu_{n_k}$ be the cluster centers. Also let z_{gk}be the binary indicator for the gth gene belonging to cluster k. We then define the following generative model

$$\begin{aligned}
P(\pi) &= Dirichlet(\omega) \\
P(\mu_k) &= \mathcal{N}(\mu_k|0,\Sigma_0) \quad \text{for} \quad k = 1, ..., n_k \\
P(z_{gk} = 1|\pi_k) &= \pi_k \quad \text{for} \quad k = 1, ..., n_k;\ n = 1, ..., n_g \\
P(\gamma_g|z_{gk} = 1, \mu_k, \eta_g) &= \mathcal{N}\left(\gamma_g|\mu_k, \mathrm{Cov}_q(\gamma_g) + \epsilon I_{n_t-1}\right) \quad \text{for} \quad k = 1, ..., n_k;\ n = 1, ..., n_g.
\end{aligned}$$

where ϵ is a small regularization parameter, which helped our optimization produce more stable results.

We will estimate the clustering using the maximum a posteriori (MAP) estimator of $\theta_{clust} := (\mu, \pi)$. This defines an optimization objective that we seek to minimize:

$$F_{clust}(\theta_{clust}, \theta_{reg}) = -\sum_{g=1}^{n_g} E_{q_z^*}\left\{\log P(\gamma_g|\eta_g, \mu, \pi, z_g) - \log P(z_g|\pi)\right\} - \log P(\mu) - \log P(\pi)$$

which, for every value of θ_{reg}, we expect to satisfy

$$\frac{\partial F_{clust}(\theta_{clust}, \theta_{reg})}{\partial \theta_{clust}} = 0.$$

Note that θ_{clust} involves only the ''global'' parameters μ and π. We did take a variational distribution for the z_{gk}s, represented by independent Bernoulli distribution, but the optimal q_z^* can be written as a function of μ and π. Hence, our optimization objective only involves these global parameters.

```
In [17]: # Define prior parameters.
         num_components = args.num_components
         epsilon = 0.1
         loc_prior_info_scalar = 1e-5

         trans_obs_dim = regs.x.shape[1] - 1
         prior_params = \
             rm_lib.get_base_prior_params(trans_obs_dim, num_components)
         prior_params['probs_alpha'][:] = 1
         prior_params['centroid_prior_info'] = loc_prior_info_scalar * np.eye(trans_obs_dim)
```

```
In [18]: gmm = rm_lib.GMM(args.num_components,
                          prior_params, regs, opt_reg_params,
                          inflate_coef_cov=None,
                          cov_regularization=epsilon)
```

In our experiment, the number of clusters n_k was chosen to be 10. We set ω to be the ones vector of length n_k. The prior info for the cluster centers Σ_0 is 1e-05$\times I$. ϵ was set to be 0.1.

Let us examine the optimization objective. First, we'll inspect the likelihood terms. What follows is the likelihood given that gene g belongs to cluster k.

```
In [19]: print(inspect.getsource(rm_lib.get_log_lik_nk))

def get_log_lik_nk(centroids, probs, x, x_infos):
    loc_log_lik = \
        -0.5 * (-2 * np.einsum('ni,kj,nij->nk', x, centroids, x_infos) +
                np.einsum('ki,kj,nij->nk', centroids, centroids, x_infos))

    log_probs = np.log(probs[0, :])
    log_lik_by_nk = loc_log_lik + log_probs.T

    return log_lik_by_nk
```

We can then optimize for q_z^*, which can be parametrized by its mean $\mathbb{E}_{q_z^*}[z]$. We note that this update has a closed form given θ_{clust}, so there is no need to solve an optimization problem to find $q_z^*(z)$. We additionally note that we do not use the EM algorithm, which we found to have exhibit extremely poor convergence rates. Rather, we set $q_z^*(z)$ to its optimal value given θ_{clust} and return the objective as a function of θ_{clust} alone, allowing the use of more general and higher-quality optimization routines.

```
In [20]: print(inspect.getsource(rm_lib.get_e_z))
```

```
def get_e_z(log_lik_by_nk):
    log_const = paragami.simplex_patterns.logsumexp(log_lik_by_nk, axis=1)
    e_z = np.exp(log_lik_by_nk - log_const)
    return e_z
```

With the optimal parameters for z_{nk}, we combine the likelihood term with the prior and entropy terms.

```
In [21]: print(inspect.getsource(rm_lib.wrap_get_loglik_terms))
         print(inspect.getsource(rm_lib.wrap_get_kl))
```

```
def wrap_get_loglik_terms(gmm_params, transformed_reg_params):
    log_lik_by_nk = get_log_lik_nk(
        centroids=gmm_params['centroids'],
        probs=gmm_params['probs'],
        x=transformed_reg_params['beta_mean'],
        x_infos=transformed_reg_params['beta_info'])

    e_z = get_e_z(log_lik_by_nk)

    return log_lik_by_nk, e_z

def wrap_get_kl(gmm_params, transformed_reg_params, prior_params):
    log_lik_by_nk, e_z = \
        wrap_get_loglik_terms(gmm_params, transformed_reg_params)
    log_prior = get_log_prior(
        gmm_params['centroids'], gmm_params['probs'], prior_params)
    return get_kl(log_lik_by_nk, e_z, log_prior)
```

This objective function is wrapped in the `GMM` class method `get_params_kl`.

```
In [22]: print(inspect.getsource(gmm.get_params_kl))
```

```
    def get_params_kl(self, gmm_params):
        """Get the optimization objective as a function of the mixture
        parameters.
        """
        return wrap_get_kl(
            gmm_params, self.transformed_reg_params, self.prior_params)
```

2.2.3 Optimization

For optimization we make extensive use of the autograd library for automatic differentiation and the paragami library for parameter packing and sparse Hessians. These packages' details are beyond the scope of the current notebook.

First, we do a k-means initialization.

```
In [23]: print('Running k-means init.')
         init_gmm_params = \
             rm_lib.kmeans_init(gmm.transformed_reg_params,
                                gmm.num_components, 50)
         print('Done.')
         init_x = gmm.gmm_params_pattern.flatten(init_gmm_params, free=True)
```

```
Running k-means init.
Done.
```

We note that the match between "exact" cross-validation (removing time points and re-optimizing) and the IJ was considerably improved by using a high-quality second-order optimization method. In particular, for these experiments, we employed the Newton conjugate-gradient trust region method (Chapter 7.1 of Wright et al [1999]) as implemented by the method `trust-ncg` in `scipy.optimize`, preconditioned by the Cholesky decomposition of an inverse Hessian calculated at an initial approximate optimum.

We found that first-order or quasi-Newton methods (such as BFGS) often got stuck or terminated at points with fairly large gradients. At such points our method does not apply in theory nor, we found, very well in practice.

The inverse Hessian used for the preconditioner was with respect to the clustering parameters only and so could be calculated quickly, in contrast to the H_1 matrix used for the IJ, which includes the regression parameters as well.

First, run with a low tolerance to get a point at which to evaluate an initial preconditioner.

```
In [24]: gmm.conditioned_obj.reset() # Reset the logging and iteration count.
         gmm.conditioned_obj.set_print_every(1)

         opt_time = time.time()
         gmm_opt, init_x2 = gmm.optimize(init_x, gtol=1e-2)
         opt_time = time.time() - opt_time
```

```
Iter 0: f = -159.11834165
Iter 1: f = -159.67926278
Iter 2: f = -159.97782885
Iter 3: f = -160.15878320
Iter 4: f = -159.59447036
Iter 5: f = -160.19209687
Iter 6: f = -160.27259154
Iter 7: f = -160.29486553
Iter 8: f = -160.33460656
Iter 9: f = -160.34154288
Iter 10: f = -160.32382096
Iter 11: f = -160.34447865
Iter 12: f = -160.34634639
Iter 13: f = -160.34692896
```

Next, set the preconditioner using the square root inverse Hessian at the point `init_x2`.

```
In [25]: tic = time.time()
         h_cond = gmm.update_preconditioner(init_x2)
         opt_time += time.time() - tic
```

The method `optimize_fully` repeats this process of optimizing and re-calculating the preconditioner until the optimal point does not change.

```
In [26]: gmm.conditioned_obj.reset()
         tic = time.time()
         gmm_opt, gmm_opt_x = gmm.optimize_fully(
             init_x2, verbose=True)
         opt_time += time.time() - tic
         print('Optimization time: {} seconds'.format(opt_time))
```

```
Preconditioned iteration 1
  Running preconditioned optimization.
Iter 0: f = -160.34692896
Iter 1: f = -160.34694250
Iter 2: f = -160.34694250
Preconditioned iteration 2
  Getting Hessian and preconditioner.
  Running preconditioned optimization.
Iter 3: f = -160.34694250
Iter 4: f = -160.34694250
Converged.
Optimization time: 8.438910484313965 seconds
```

`paragami` patterns allow conversion between unconstrained vectors and dictionaries of parameter values. After "folding" the optimal `gmm_opt_x`, `opt_gmm_params` contains a dictionary of optimal cluster centroids and cluster probabilities.

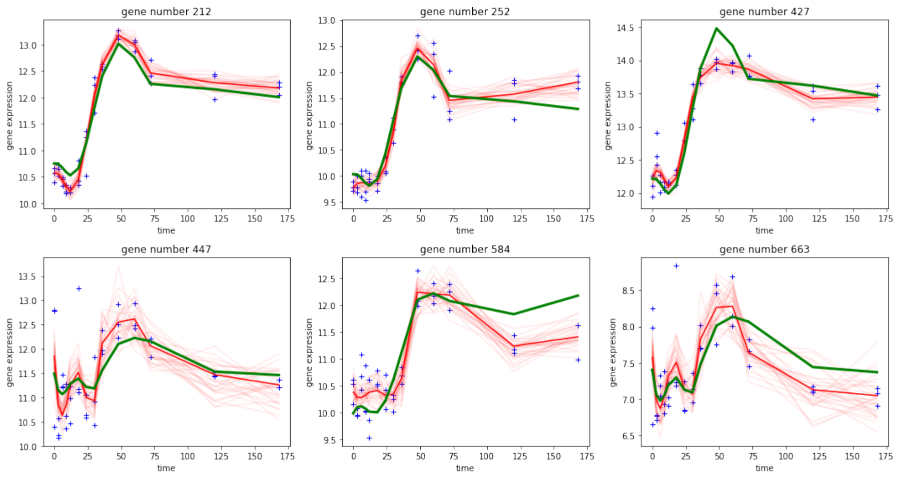
```
In [27]: opt_gmm_params = gmm.gmm_params_pattern.fold(gmm_opt_x, free=True)
         print(opt_gmm_params.keys())
         print(np.sort(opt_gmm_params['probs']))
```

```
odict_keys(['centroids', 'probs'])
[[0.01567608 0.04016882 0.06955236 0.07427946 0.09373695 0.0947442
  0.09653288 0.12626624 0.15739176 0.23165127]]
```

Each gene's regression line has an inferred cluster membership given by $\mathbb{E}_{q_z^*}[z_g]$, and an expected posterior centroid given by $\sum_k \mathbb{E}_{q_z^*}[z_{gk}]\mu_k$. This expected posterior centroid can be untransformed to give a prediction for the observation.

It is the difference between this prediction line — which is a function of the clustering — and the actual data that we consider to be the "error" of the model.

```
In [28]: gmm_pred = mse_utils.get_predictions(gmm, opt_gmm_params, opt_reg_params)

         f, axarr = plt.subplots(2, 3, figsize=(15,8))

         for i in range(6):
             n = gene_indx[i]
             this_plot = axarr[int(np.floor(i / 3)), i % 3]
             plot_utils_lib.PlotRegressionLine(
                 timepoints, regs, opt_reg_params, n, this_plot=this_plot)
             plot_utils_lib.PlotPredictionLine(
                 timepoints, regs, gmm_pred, n, this_plot=this_plot)

         f.tight_layout()
```



2.2.4 Calculating H_1 for the IJ

We seek to choose the degrees of freedom α for the B-splines using cross-validation. We leave out one or more timepoints, and fit using only the remaining timepoints. We then estimate the test error by predicting the value of the genes at the held out timepoints.

To do this, we define time weights w_t by observing that, for each g, the term $\mathbb{E}_q\left[\log P\left(\beta_g, \sigma_g^2 | y_g\right)\right]$ decomposes into a sum over time points:

$$F_{reg,g}\left(\eta_g, \alpha, w\right) := -\sum_{t=1}^{n_t} w_t \left(-\frac{1}{2}\sigma_g^{-2}\left(y_{g,t} - \left(X\beta_g\right)_t\right)^2 - \frac{1}{2}\log\sigma_g^2\right) + \mathbb{E}_q\left[\log q\left(\beta_g, \sigma_g^2 | \eta_g\right)\right].$$

We naturally define $F_{reg}\left(\theta_{reg}, \alpha, w\right) := \sum_{g=1}^{n_g} F_{reg,g}\left(\eta_g, \alpha, w\right)$.

By defining $\theta = (\theta_{clust}, \theta_{reg})$, we then have an M-estimator

$$G(\theta, w, \alpha) := \begin{pmatrix} \frac{\partial F_{reg}(\theta_{reg}, w, \alpha)}{\partial \theta_{reg}} \\ \frac{\partial F_{clust}(\theta_{clust}, \theta_{reg})}{\partial \theta_{clust}} \end{pmatrix} = 0.$$

We can then apply the IJ to approximate the leaving out of various timepoints.

Note that what we call the "Hessian" for this two-step procedure is not really a Hessian, as it is not symmetric. More precisely, it is the Jacobian of G, or what we defined as H_1 in the text.

Calculating H_1 is the most time-consuming part of the infinitesimal jackknife, since the H_1 matrix is quite large (though sparse). However, once H_1 is computed, calculating each $\theta_{IJ}(w)$ is extremely fast.

H_1 can be computed in blocks:

$$H_1 = \begin{pmatrix} \nabla^2_{\theta_{reg}} F_{reg} & 0 \\ \nabla_{\theta_{reg}} \nabla_{\theta_{clust}} F_{clust} & \nabla^2_{\theta_{clust}} F_{clust} \end{pmatrix}$$

The code refers to $\nabla^2_{\theta_{clust}} F_{clust}$ as the "GMM Hessian". It refers to $\nabla_{\theta_{reg}} \nabla_{\theta_{clust}} F_{clust}$ as the "cross Hessian". And it refers to $\nabla^2_{\theta_{reg}} F_{reg}$ as the "regression Hessian", which itself is block diagonal, with each block an observation. Due to details of the implementation of block sparse Hessians using forward mode automatic differnetiation in the class `vittles.SparseBlockHessian`, the code below confusingly refers to each regression parameter as a "block".

When the `FitDerivatives` class is initialized, it calculates these blocks separately and stacks them into the attribute `full_hess`, which is a sparse matrix representing H_1.

```
In [29]: # Even though $H_1$ is not a Hessian, by force of habit we call the time to
         # compute it ``hess_time``.
         hess_time = time.time()
         fit_derivs = sens_lib.FitDerivatives(
             opt_gmm_params, opt_reg_params,
             gmm.gmm_params_pattern, regs.reg_params_pattern,
             gmm=gmm, regs=regs,
             print_every=10)
         hess_time = time.time() - hess_time
         print('Total hessian time: {} seconds'.format(hess_time))
```

```
Initializing FitDerivatives.
Getting t Jacobian.
Getting full Hessian.
   Getting GMM Hessian...
   GMM Hessian time: 2.1917014122009277
   Getting cross Hessian...
   Cross Hessian time: 34.25235605239868
   Getting regression Hessian...
Block index 0 of 66.
Block index 10 of 66.
Block index 20 of 66.
Block index 30 of 66.
Block index 40 of 66.
Block index 50 of 66.
Block index 60 of 66.
Done differentiating.
   Regression Hessian time: 121.74362897872925
Done with full Hessian.
Total hessian time: 169.59288716316223 seconds
```

2.2.5 Save results as a compressed file.

The results, including H_1, are now saved. To calculate the exact CV, these results (including the preconditioner) will be loaded and the model will be refit with timepoints left out. To calculate the IJ, the same results will be loaded and H_1 will be used to calculate the IJ.

```
In [30]: extra_metadata = dict()
         extra_metadata['opt_time'] = opt_time
         extra_metadata['reg_time'] = reg_time
         extra_metadata['hess_time'] = hess_time
         extra_metadata['df'] = args.df
         extra_metadata['degree'] = args.degree

         npz_outfile = '../fits/initial_fit.npz'
         saving_gmm_utils.save_initial_optimum(
             npz_outfile,
             gmm=gmm,
             regs=regs,
             timepoints=timepoints,
             fit_derivs=fit_derivs,
             extra_metadata=extra_metadata)
```

2.2.6 Bibliography

J. E. Shoemaker, S. Fukuyama, A. J. Eisfeld, D. Zhao, E. Kawakami, S. Sakabe, T. Maemura, T. Gorai, H. Katsura, Y. Muramoto, S. Watanabe, T. Watanabe, K. Fuji, Y. Matsuoka, H. Kitano, and Y. Kawaoka. An Ultrasensitive Mechanism Regulates Influenza Virus-Induced Inflammation. PLoS Pathogens, 11(6):1–25, 2015

S. Wright and J. Nocedal. Numerical optimization. Springer Science, 35(67-68):7, 1999.

load_and_refit

February 21, 2019

1 Step 2: Refit.

In this notebook, we calculate the parameters used for exact CV by refitting the model initially fit in step one, the notebook `fit_model_and_save`.

For expository purposes this notebook calculates the refit for only one weight vector. To compute exact CV, one would perform the corresponding computation for all leave-k-out weight vectors.

```
In [1]: from copy import deepcopy
        import inspect
        import matplotlib.pyplot as plt
        %matplotlib inline
        import numpy as np
        import sys
        import time

        np.random.seed(3452453)

        import paragami

        from aistats2019_ij_paper import regression_mixture_lib as rm_lib
        from aistats2019_ij_paper import saving_gmm_utils
        from aistats2019_ij_paper import mse_utils

        import plot_utils_lib
```

```
In [2]: # Load the initial fit.
        # This file was produced by the notebook ``fit_model_and_save``.
        initial_fit_infile = '../fits/initial_fit.npz'
        full_fit, gmm, regs, metadata = \
            saving_gmm_utils.load_initial_optimum(initial_fit_infile)
        timepoints = metadata['timepoints']
```

```
Initializing FitDerivatives.
Using provided t_jac.
Using provided full_hess.
```

First, choose some timepoints to leave out.

```
In [3]: # Simulate passing arguments in on the command line.
        class Args():
            def __init__(self):
                pass

        args = Args()
        args.num_times = 1
        args.which_comb = 1
        args.max_num_timepoints = 7
```

The number of points left out (that is, k) is given by `num_times`, which is 1. The largest timepoint we leave out is given by `max_num_timepoints`, which is 7. Because later timepoints are not affected by the smoothing, there is no reason to leave them out.

There are a certain number of ways to leave k out of 7 timepoints, and `which_comb` chooses one of them in the order given by the function `itertools.combinations`. Of course, when $k = 1$, `which_comb` simply chooses which timepoint to leave out. `mse_utils.get_indexed_combination` maps `which_comb` to particular timepoints in a consistent way.

Full exact CV would run this script for all 7 choose k values of `which_comb`.

Because we have repeated measurements at each timepoint, leaving out a single timepoint will correspond to leaving out multiple row of the observation matrix. Those rows are determined by `mse_utils.get_time_weight`, which also returns a weight vector setting these observations' weights to zero.

```
In [4]: lo_inds = mse_utils.get_indexed_combination(
            num_times=args.num_times, which_comb=args.which_comb,
            max_num_timepoints=args.max_num_timepoints)
        new_time_w, full_lo_inds = mse_utils.get_time_weight(lo_inds, timepoints)

        print('Left out timepoint: {}'.format(lo_inds))
        print('Left out observations: {}'.format(full_lo_inds))
        print('Leave-k-out weights: {}'.format(new_time_w))

Left out timepoint: [1]
Left out observations: [3 4 5]
Leave-k-out weights: [1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1]
```

We now re-optimize with the new weights.

Note that we could either start the optimization at the initial optimum (a "warm start") or do a fresh start from k-means. A fresh start is more time consuming but a more stringent test for the accuracy of the IJ. We calculate both, but report results from the fresh start in the paper. In the notebook `examine_and_save_results`, you can choose to examine either set of results.

Here, for consistency with the paper, we re-initialize with k-means.

```
In [5]: regs.time_w = deepcopy(new_time_w)
        reg_params_w = regs.get_optimal_regression_params()
        gmm.set_regression_params(reg_params_w)

        init_gmm_params = \
            rm_lib.kmeans_init(gmm.transformed_reg_params,
                               gmm.num_components, 50)
        init_x = gmm.gmm_params_pattern.flatten(init_gmm_params, free=True)

        opt_time = time.time()
        gmm_opt, init_x2 = gmm.optimize(init_x, gtol=1e-2)

        print('\tUpdating preconditioner...')
        kl_hess = gmm.update_preconditioner(init_x2)

        print('\tRunning preconditioned optimization...')
        gmm.conditioned_obj.reset()
        reopt, gmm_params_free_w = gmm.optimize_fully(init_x2, verbose=True)
        print(gmm_opt.message)
        opt_time = time.time() - opt_time

        print('Refit time: {} seconds'.format(opt_time))
```

```
Iter 0: f = -153.38003431
Iter 1: f = -152.49438715
Iter 2: f = -153.69147895
Iter 3: f = -153.83779915
Iter 4: f = -154.02397812
Iter 5: f = -153.41393391
Iter 6: f = -154.10396420
Iter 7: f = -154.14366282
Iter 8: f = -154.14261201
Iter 9: f = -154.16417745
Iter 10: f = -154.18307547
Iter 11: f = -154.20711481
Iter 12: f = -154.22118064
Iter 13: f = -154.27402715
Iter 14: f = -154.28739474
Iter 15: f = -154.33849929
Iter 16: f = -154.03580241
Iter 17: f = -154.35421130
Iter 18: f = -154.36910489
Iter 19: f = -154.36872458
Iter 20: f = -154.37238982
Iter 21: f = -154.37722095
Iter 22: f = -154.38186985
Iter 23: f = -154.38410992
        Updating preconditioner...
        Running preconditioned optimization...
Preconditioned iteration 1
```

```
  Running preconditioned optimization.
Iter 0: f = -154.38410992
Iter 1: f = -154.38423176
Iter 2: f = -154.38584092
Iter 3: f = -154.21889674
Iter 4: f = -154.42200228
Iter 5: f = -154.39603234
Iter 6: f = -154.39957947
Iter 7: f = -154.41374585
Iter 8: f = -154.43397491
Iter 9: f = -154.43484046
Iter 10: f = -154.43484816
Iter 11: f = -154.43484816
Preconditioned iteration 2
  Getting Hessian and preconditioner.
  Running preconditioned optimization.
Iter 12: f = -154.43484816
Iter 13: f = -154.43484816
Converged.
Optimization terminated successfully.
Refit time: 14.35115647315979 seconds
```

We now save the results.

```
In [6]: gmm_params_w = \
            full_fit.comb_params_pattern['mix'].fold(
                gmm_params_free_w, free=True)
        refit_comb_params = {
            'mix': gmm_params_w,
            'reg': reg_params_w }
        refit_comb_params_free = \
            full_fit.comb_params_pattern.flatten(refit_comb_params, free=True)
```

```
In [7]: save_filename = \
            '../fits/refit__num_times{}__which_comb{}.npz'.format(
                    args.num_times, args.which_comb)
        print('Saving to {}'.format(save_filename))
        saving_gmm_utils.save_refit(
            outfile=save_filename,
            comb_params_free=refit_comb_params_free,
            comb_params_pattern=full_fit.comb_params_pattern,
            initial_fit_infile=initial_fit_infile,
            time_w=new_time_w,
            lo_inds=lo_inds,
            full_lo_inds=full_lo_inds)
```

```
Saving to ../fits/refit__num_times1__which_comb1.npz
```

calculate_prediction_errors

February 21, 2019

1 Step 3: Calculate the IJ and prediction errors.

In this notebook, for a single weight vector, we calculate the IJ itself as well as the prediction errors for exact CV and IJ. This notebook uses the output of the notebooks `load_and_refit` and `fit_model_and_save`.

```
In [1]: import numpy as np
        import paragami
        import vittles
        import scipy as sp
        from scipy import sparse
        import time

        import seaborn as sns
        import pandas as pd

        import matplotlib.pyplot as plt
        %matplotlib inline

        np.random.seed(3452453)

        from aistats2019_ij_paper import regression_lib as reg_lib
        from aistats2019_ij_paper import sensitivity_lib as sens_lib
        from aistats2019_ij_paper import saving_gmm_utils
        from aistats2019_ij_paper import mse_utils

        import plot_utils_lib
```

```
In [2]: # Simulate passing arguments in on the command line.
        class Args():
            def __init__(self):
                pass

        args = Args()
        args.num_times = 1
        args.which_comb = 1
        args.max_num_timepoints = 7
```

```
In [3]: ################################
        # Load the original fit.

        print('Loading original fit.')
        initial_fit_infile = '../fits/initial_fit.npz'
        full_fit, gmm, regs, initial_metadata = \
            saving_gmm_utils.load_initial_optimum(initial_fit_infile)

        opt_comb_params = full_fit.get_comb_params()
```

```
Loading original fit.
Initializing FitDerivatives.
Using provided t_jac.
Using provided full_hess.
```

```
In [4]: ################################
        # Load the test data

        test_regression_infile = '../fits/test_regressions.json'
        with open(test_regression_infile) as infile:
            regs_test = reg_lib.Regressions.from_json(infile.read())

        ###########################################
        # Load a refit as specfified by ``args``.

        refit_filename = \
            '../fits/refit__num_times{}__which_comb{}.npz'.format(
                    args.num_times, args.which_comb)

        comb_params_free_refit, comb_params_pattern_refit, refit_metadata = \
            saving_gmm_utils.load_refit(refit_filename)

        time_w = refit_metadata['time_w']
        lo_inds = refit_metadata['lo_inds']
        full_lo_inds = refit_metadata['full_lo_inds']

        assert(comb_params_pattern_refit == full_fit.comb_params_pattern)
        comb_params_refit = comb_params_pattern_refit.fold(
            comb_params_free_refit, free=True)

        time_w = refit_metadata['time_w']
        lo_inds = refit_metadata['lo_inds']
        full_lo_inds = refit_metadata['full_lo_inds']
```

The objects named `comb_params` refer to both the regression and clustering parameters. The name `free` refers to the unconstrained flat value for the parameters as calculated by `paragami`.

```
In [5]: print('Regression pattern: ',
              comb_params_pattern_refit['reg'])
```

```
        print('Clustering pattern: ',
              comb_params_pattern_refit['mix'])

Regression pattern:  OrderedDict:
        [beta_mean] = NumericArrayPattern (700, 10) (lb=-inf, ub=inf)
        [beta_info] = PatternArray (700,) of PDMatrix 10x10 (diag_lb = 0.0)
        [y_info] = NumericArrayPattern (700,) (lb=0.0, ub=inf)
Clustering pattern:  OrderedDict:
        [centroids] = NumericArrayPattern (10, 9) (lb=-inf, ub=inf)
        [probs] = SimplexArrayPattern (1,) of 10-d simplices
```

1.0.1 Calculate the infinitesimal jackknife.

The `vittles` package makes it easy to calculate linear approximations to the sensitivity of M-estimators to hyperparameters, of which the IJ is a special case. Here, the `HyperparameterSensitivityLinearApproximation` uses the sparse value of H_1 calculated earlier.

Note that H_1 is factorized during the initialization of `weight_sens`, and that it takes relatively little time.

```
In [6]: # Note that if you don't cast the jacobian to a numpy array from
        # a numpy matrix, the output is a 2d-array, causing confusion later.
        weight_sens = vittles.HyperparameterSensitivityLinearApproximation(
            objective_fun=lambda: 0,
            opt_par_value=full_fit.comb_params_free,
            hyper_par_value=regs.time_w,
            hessian_at_opt=sp.sparse.csc_matrix(full_fit.full_hess),
            cross_hess_at_opt=np.array(full_fit.t_jac.todense()))
```

We now use the `weight_sens` object to approximate the "free" value of the combined parameters at `time_w`. The IJ operates in unconstrained space, so we use `paragami` to fold the unconstrained vector back into a dictionary of parameters.

```
In [7]: # Get the infinitesimal jackknife for the refit weight vector.
        lr_time = time.time()
        comb_params_free_lin = \
            weight_sens.predict_opt_par_from_hyper_par(time_w)
        lr_time = time.time() - lr_time
        print('Infinitesimal jackknife time: {}'.format(lr_time))

        comb_params_lin = full_fit.comb_params_pattern.fold(comb_params_free_lin, free=True)

Infinitesimal jackknife time: 0.0011603832244873047
```

1.0.2 Calculate various prediction errors.

Recall that the prediction error is the difference between the data and the posterior expected cluster centroid for a particular gene. Let us consider the original optimal clustering parameters, `opt_comb_params['mix']`. To get the test set error on gene g for these parameters, we need to do the following steps:

1. Run the regression for gene g in the test set
2. Classify the regression, calculating $\mathbb{E}_{q_z^*}[z_g]$. This is a function of the clustering parameters and the regression line for gene g.
3. Calculate the expected posterior cluster centroid for gene g, which is $\mu_g^* = \sum_k \mathbb{E}_{q_z^*}[z_{gk}]\mu_k$.
4. Because the transformation discards the mean information, compare the de-meaned data to the estimated centroid: $error_{gt} = \left(y_{gt} - \frac{1}{T}\sum_{t'=1}^{T} y_{gt'}\right) - \mu_{gt}^*$.

Note that step one could re-run the regression either with the original weights or the new weights. We found that this decision does not matter qualitatively. Here and in the paper, we simply classify the original regression, but the notebook `examine_and_save_results` can produce results for oth the original and re-weighted regressions.

We will examine prediction error on the time points that are left out, that is, for observations in `full_lo_inds`.

```
In [8]: print('Calculating prediction error.')

        # Get the training set error on the full data.
        train_error = mse_utils.get_lo_err_folded(
            opt_comb_params,
            keep_inds=full_lo_inds,
            mse_regs=regs,
            mse_reg_params=opt_comb_params['reg'],
            gmm=gmm)

        ############
        # Original fit.

        # Get the optimal test set regressions.
        reg_params_test = regs_test.get_optimal_regression_params()

        # Get the test error for the original fit.
        orig_test_error = mse_utils.get_lo_err_folded(
            opt_comb_params,
            keep_inds=full_lo_inds,
            mse_regs=regs_test,
            mse_reg_params=reg_params_test,
            gmm=gmm)

        orig_pred = mse_utils.get_predictions(
            gmm, opt_comb_params['mix'], reg_params_test)

        # Get the test error for the CV refit.
        cv_error = mse_utils.get_lo_err_folded(
            comb_params_refit,
            keep_inds=full_lo_inds,
            mse_regs=regs_test,
            mse_reg_params=reg_params_test,
            gmm=gmm)

        cv_pred = mse_utils.get_predictions(
            gmm, comb_params_refit['mix'], reg_params_test)

        # Get the test error for the IJ approximation.
        ij_error = mse_utils.get_lo_err_folded(
            comb_params_lin,
            keep_inds=full_lo_inds,
            mse_regs=regs_test,
            mse_reg_params=reg_params_test,
            gmm=gmm)

        ij_pred = mse_utils.get_predictions(
            gmm, comb_params_lin['mix'], reg_params_test)
```

```
Calculating prediction error.
```

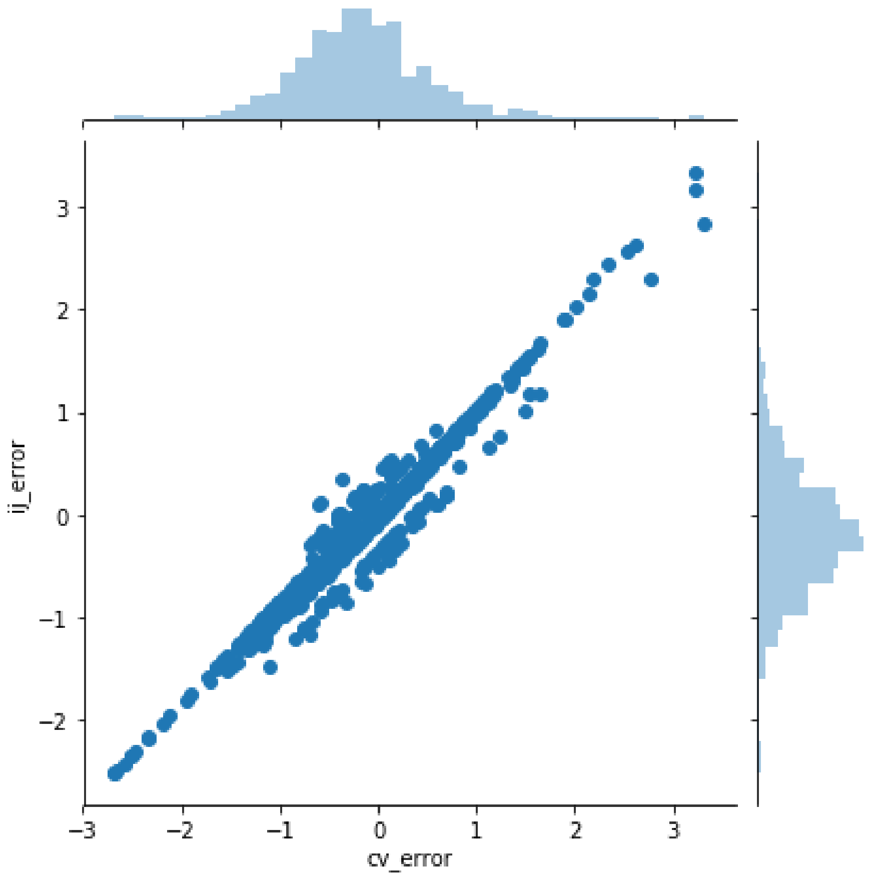
1.0.3 Selected results.

We now make a cursory comparison of the results. For a more detailed analysis, including the results that went into the paper, see the notebook `examine_and_save_results`.

```
In [9]: cv_excess_error = cv_error - orig_test_error
        ij_excess_error = ij_error - orig_test_error

        def GetColDf(col):
            return pd.DataFrame(
                {'cv_error': cv_error[:, col],
                 'cv_excess': cv_excess_error[:, col],
                 'ij_error': ij_error[:, col],
                 'ij_excess': ij_excess_error[:, col],
                 'col': col})

        result = pd.concat([ GetColDf(col) for col in range(len(full_lo_inds)) ])
```

If we simply look at the point-by-point error, CV and IJ are highly correlated.

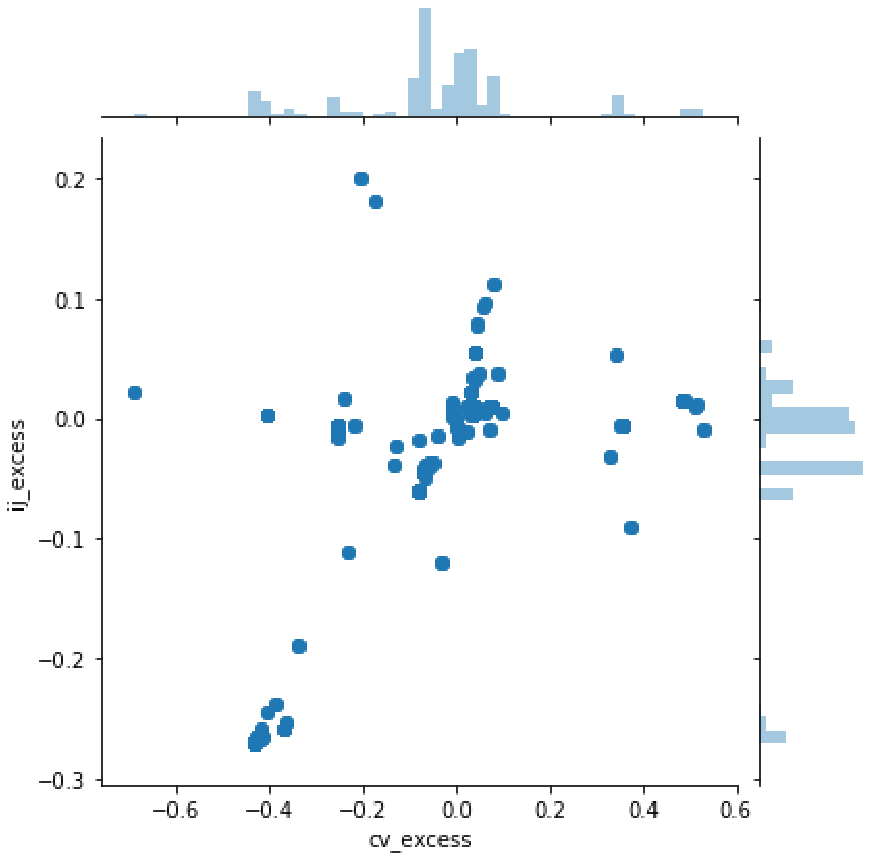
```
In [10]: sns.jointplot(x='cv_error', y='ij_error', data=result);
```



However, this is because the error in each point is dominated by the error at the original optimum. To meaningfully compare the IJ to CV, we should compare the difference between the IJ and CV error and the error at the original optimum. The distribution of these "difference-in-difference" errors is shown in the next plot.

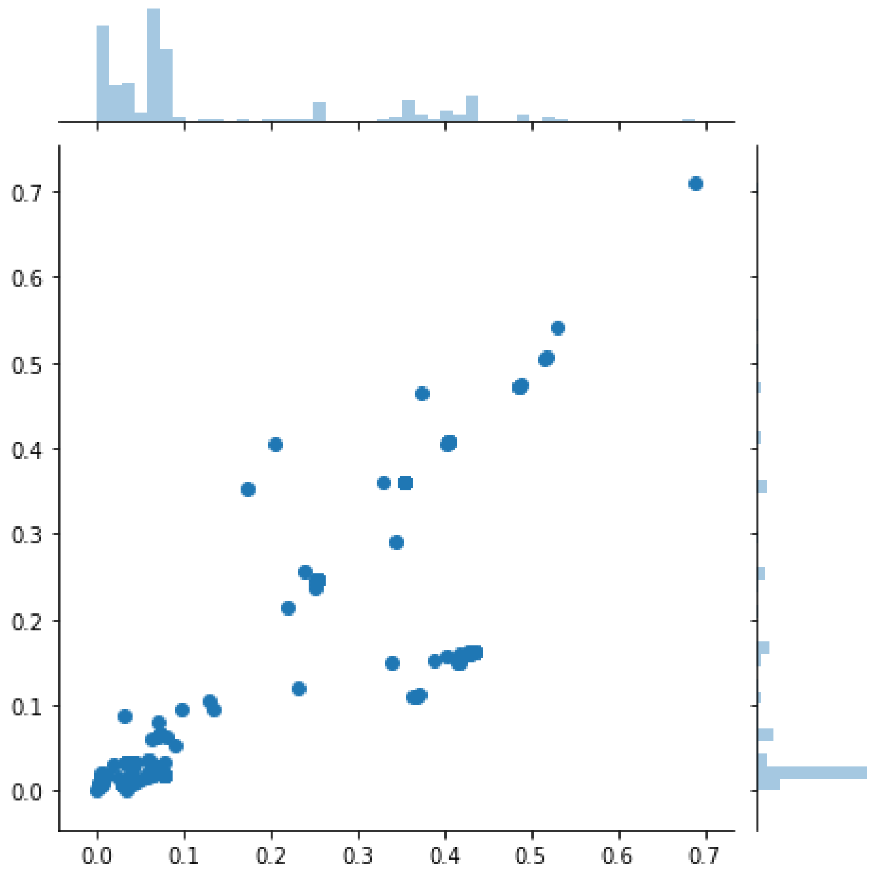
Some clear outliers can be seen. However, note that, in this case, overplotting makes IJ looks worse than it is – in the histograms you can see that most differences are very small.

```
In [11]: sns.jointplot(x='cv_excess', y='ij_excess', data=result);
```



As you might expect from a linear approximation, the IJ does the worst when the predicted change for CV is large.

```
In [12]: misfit = np.max(np.abs(cv_excess_error - ij_excess_error), axis=1)
         abs_cv_excess_error = np.max(np.abs(cv_excess_error), axis=1)

         sns.jointplot(abs_cv_excess_error, misfit)

Out[12]: <seaborn.axisgrid.JointGrid at 0x7f3f74fe1908>
```



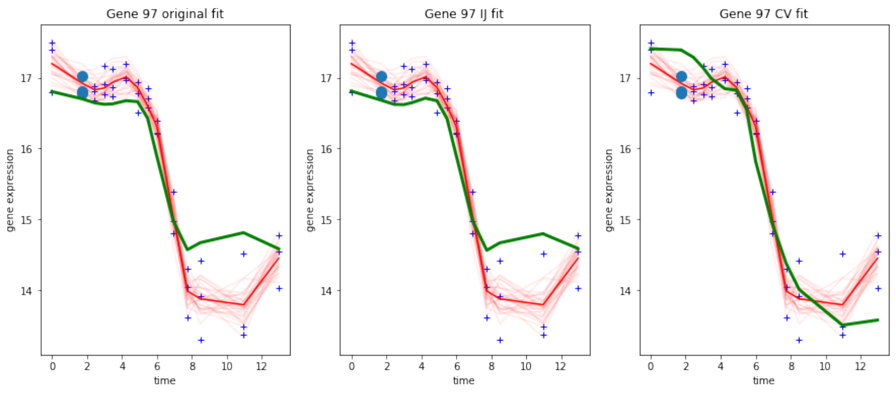
Finally, we visualize some of the genes where IJ badly misestimates the CV error. Clearly, in these cases, re-fitting with the left-out points (shown with large dots) produced large changes that the IJ did not capture. In general, it appears that the IJ errs relative to CV by not moving far enough from the original optimum.

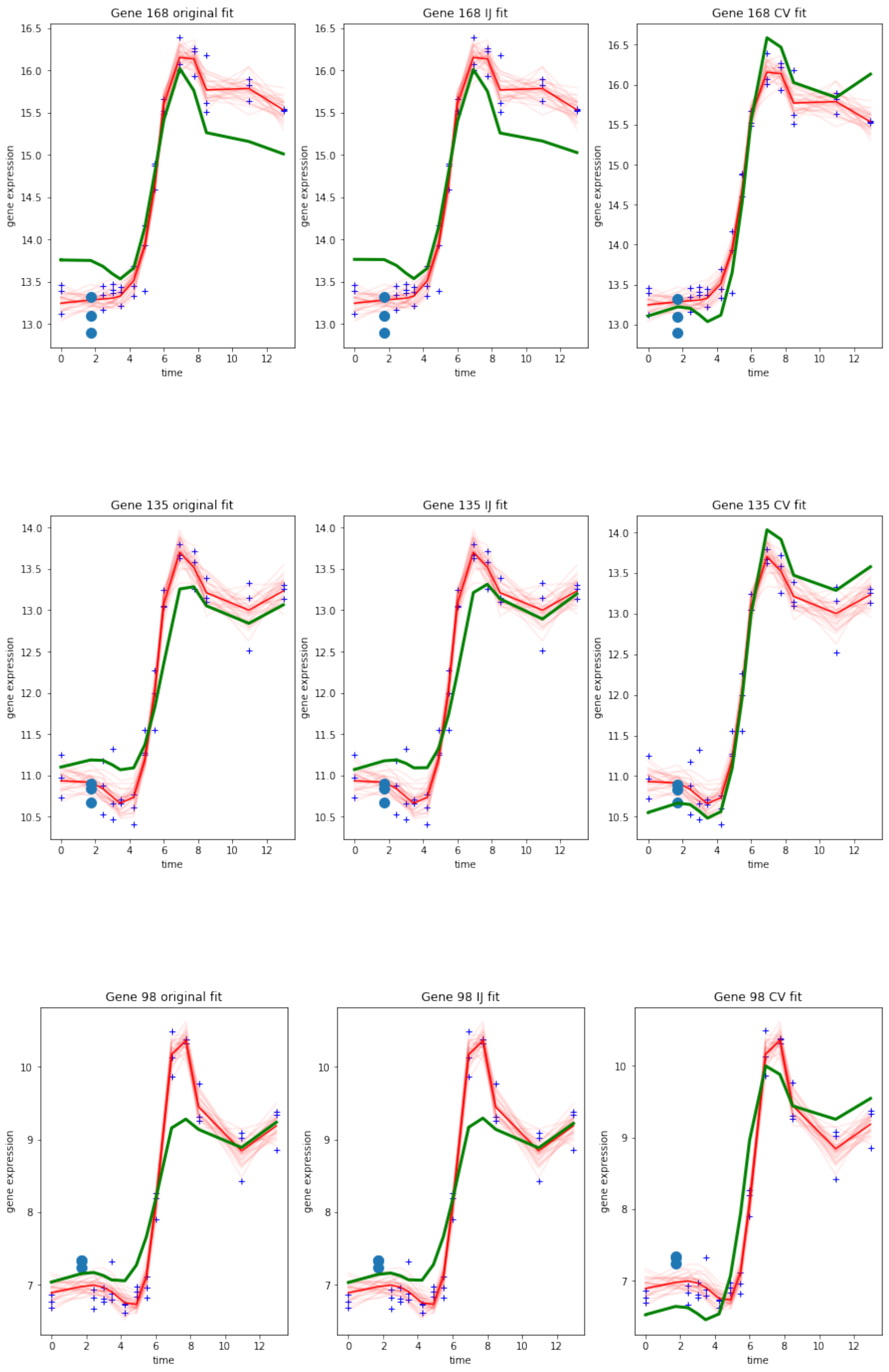
Despite the poor fit on these extreme genes, we stress that most genes exhibited small changes in both CV and IJ. For these genes, IJ performs well enough to capture salient aspects of the estimated out-of-sample error. For more detailed analysis of this point, see the notebook `examine_and_save_results`.

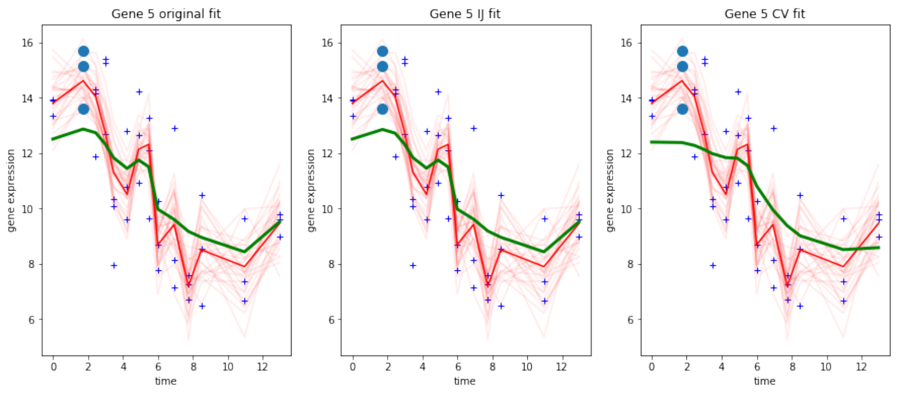
```
In [13]: timepoints = initial_metadata['timepoints']
         timepoints_stretch = np.sqrt(timepoints)

         def PlotGenePredictions(gene_ind):
             _, figs = plt.subplots(1, 3, figsize=(15,6))
```

```
    for i in range(3):
        np.random.seed(42)
        plot_utils_lib.PlotRegressionLine(
            timepoints_stretch, regs_test, reg_params_test, gene_ind, this_plot=figs[i]
        figs[i].plot(timepoints_stretch[full_lo_inds],
                     regs_test.y[gene_ind, full_lo_inds], 'o', markersize=10)

    plot_utils_lib.PlotPredictionLine(
        timepoints_stretch, regs_test, orig_pred, gene_ind, this_plot=figs[0])
    figs[0].set_title('Gene {} original fit'.format(gene_ind))

    plot_utils_lib.PlotPredictionLine(
        timepoints_stretch, regs_test, ij_pred, gene_ind, this_plot=figs[1])
    figs[1].set_title('Gene {} IJ fit'.format(gene_ind))

    plot_utils_lib.PlotPredictionLine(
        timepoints_stretch, regs_test, cv_pred, gene_ind, this_plot=figs[2])
    figs[2].set_title('Gene {} CV fit'.format(gene_ind))
```

```
In [14]: worst_fits = np.argsort(-1 * misfit)

         for gene in worst_fits[0:5]:
             PlotGenePredictions(gene)
```




Gene 168 original fit
Gene 168 IJ fit
Gene 168 CV fit
Gene 135 original fit
Gene 135 IJ fit
Gene 135 CV fit
Gene 98 original fit
Gene 98 IJ fit
Gene 98 CV fit
gene expression
time


Gene 5 original fit
Gene 5 IJ fit
Gene 5 CV fit
gene expression
time

examine_and_save_results

February 21, 2019

1 Detailed analysis of results.

This notebook loads the output of the scripts in the directory `cluster_scripts` (particularly, the final script, `run_slurm_pred_error.py`). It produces the Rdata file that is used for the graphs in the paper as well as a number of supplemental analyses.

```
In [1]: library(tidyverse)
        library(gridExtra)
        library(repr) # For setting plot sizes
        source("load_python_data_lib.R")
        py_main <- InitializePython()
```

```
 Attaching packages  tidyverse 1.2.1
 ggplot2 3.1.0      purrr   0.2.5
 tibble  1.4.2      dplyr   0.7.8
 tidyr   0.8.1      stringr 1.3.1
 readr   1.1.1      forcats 0.3.0
 Conflicts  tidyverse_conflicts()
 dplyr::filter() masks stats::filter()
 dplyr::lag()    masks stats::lag()

Attaching package: gridExtra

The following object is masked from package:dplyr:

    combine


Attaching package: reshape2

The following object is masked from package:tidyr:

    smiths
```

```
In [2]: # Choose the initialization method.
        init_method <- "kmeans" # This is the choice for the paper.
```

```
#init_method <- "warm"

# Choose whether or not to re-run the regressions before calculating test error.
use_rereg <- FALSE # This is the choice for the paper.
#use_rereg <- TRUE

# This is the file that is used in the paper's knitr.
save_dir <- "../../fits"
save_filename <- sprintf("paper_results_init_%s_rereg_%s.Rdata", init_method, use_rereg)
```

1.0.1 Load the saved data for all dfs and k

```
In [3]: dfs <- list()
        metadata_dfs <- list()

        for (lo_num_times in 1:3) {
          cat("lo_num_times ", lo_num_times)
          for (df in 4:8) {
            cat(".")
            load_res <- LoadPredictionError(df, lo_num_times, init_method)
            this_refit_err_df <- load_res$refit_err_df
            this_metadata_df <- load_res$metadata_df
            this_refit_err_melt <- MeltErrorColumns(this_refit_err_df)
            dfs[[length(dfs) + 1]] <- this_refit_err_melt
            metadata_dfs[[length(metadata_dfs) + 1]] <- this_metadata_df
          }
          cat("\n")
        }
        cat("Done.\n")
        refit_err_melt <- do.call(bind_rows, dfs)
        metadata_df <- do.call(bind_rows, metadata_dfs)
```

```
lo_num_times  1...
lo_num_times  2...
lo_num_times  3...
Done.
```

1.0.2 Metadata (timing, parameter dimensions)

Make a tidy dataframe with the metadata. The parameter length, Hessian time, and initial optimization time are all reported in the text of the paper. Their values will be derived from this dataframe in knitr.

```
In [4]: metadata_df <-
            metadata_df %>%
            mutate(lr_hess_time=total_lr_time + initial_hess_time,
                   avg_lr_time=total_lr_time / num_comb,
                   avg_refit_time=total_refit_time / num_comb,
                    param_length=gmm_param_length + reg_param_length)
        print(names(metadata_df))

        select(metadata_df, df, param_length) %>%
            group_by(df) %>%
            summarize(param_length=unique(param_length))

        select(metadata_df, df, initial_hess_time, initial_opt_time) %>%
            group_by(df) %>%
            summarize(initial_hess_time=median(initial_hess_time),
                      initial_opt_time=median(initial_opt_time))

        round(median(metadata_df$initial_opt_time), digits=-1)
```

```
 [1] "num_comb"          "total_lr_time"     "total_refit_time"
 [4] "initial_opt_time"  "initial_reg_time"  "initial_hess_time"
 [7] "gmm_param_length"  "reg_param_length"  "df"
[10] "lo_num_times"      "init_method"       "lr_hess_time"
[13] "avg_lr_time"       "avg_refit_time"    "param_length"
```

df	param_length
4	25325
5	31643
6	38661
7	46379
8	54797

df	initial_hess_time	initial_opt_time
4	275.7295	31.44656
5	295.0325	41.84182
6	359.6855	35.11145
7	478.7345	50.88843
8	584.4987	77.02919

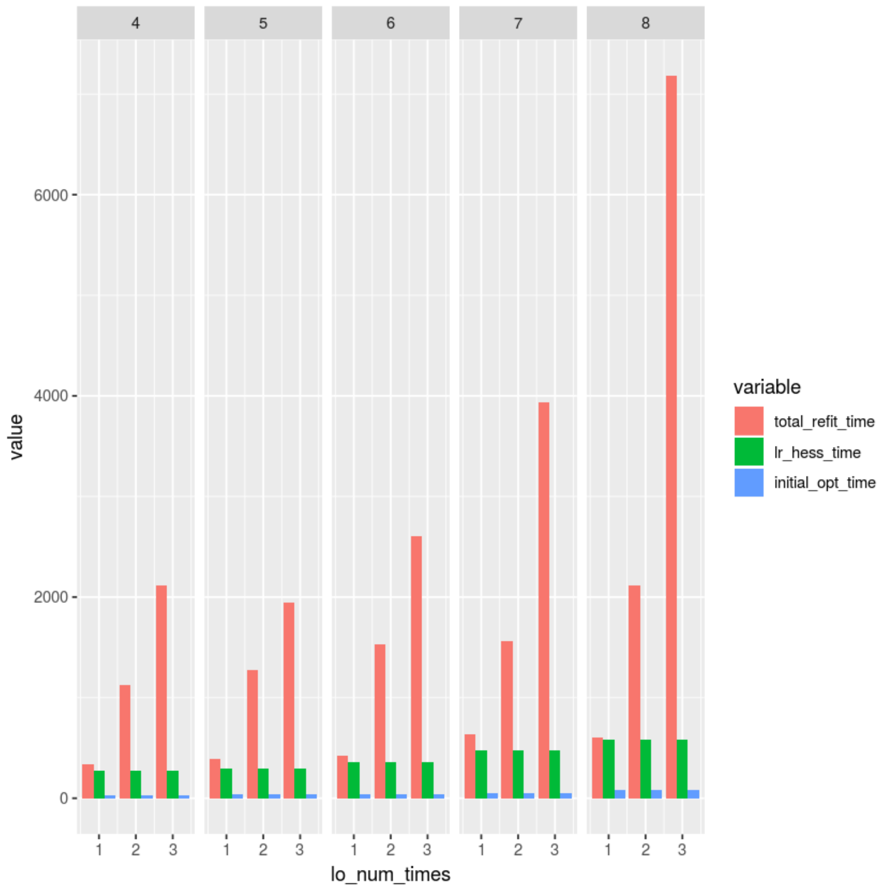
40

Make a dataframe for the timing plot from the metadata.

```
In [5]: metadata_graph_df <-
            metadata_df %>%
            select(df, lo_num_times, total_refit_time, lr_hess_time, initial_opt_time) %>%
            melt(id.vars=c("lo_num_times", "df"))
        head(metadata_graph_df)
```

lo_num_times	df	variable	value
1	4	total_refit_time	338.1638
1	5	total_refit_time	391.6006
1	6	total_refit_time	423.8322
1	7	total_refit_time	632.2635
1	8	total_refit_time	599.0894
2	4	total_refit_time	1123.7316

```
In [6]: ggplot(metadata_graph_df) +
            geom_bar(aes(x=lo_num_times, y=value, fill=variable, group=variable),
                     stat="identity", position=position_dodge()) +
            facet_grid( ~ df)
```



1.0.3 Calculate prediction errors

Make summaries of prediction error for various methods and datasets.

```
In [7]: # In-sample IJ error.
        lr_df <-
          refit_err_melt %>%
          filter(rereg==use_rereg, method=="lin", test==FALSE, measure=="err") %>%
          rename(error=value) %>%
          mutate(output="lin_in_sample")

        # In-sample CV error.
        cv_df <-
          refit_err_melt %>%
          filter(rereg==use_rereg, method=="ref", test==FALSE, measure=="err") %>%
          rename(error=value) %>%
          mutate(output="cv_in_sample")

        # In-sample training error (no points left out).
        train_df <-
          refit_err_melt %>%
          filter(rereg==use_rereg, method=="ref", test==FALSE, measure=="train_err") %>%
          rename(error=value) %>%
          mutate(output="train_error")

        # Out-of-sample test error.
        test_df <-
          refit_err_melt %>%
          filter(rereg==use_rereg, method=="ref", test==TRUE, measure=="train_err") %>%
          rename(error=value) %>%
          mutate(output="test_error")

        refit_for_df_choice <- bind_rows(
          lr_df, cv_df, test_df, train_df)
```

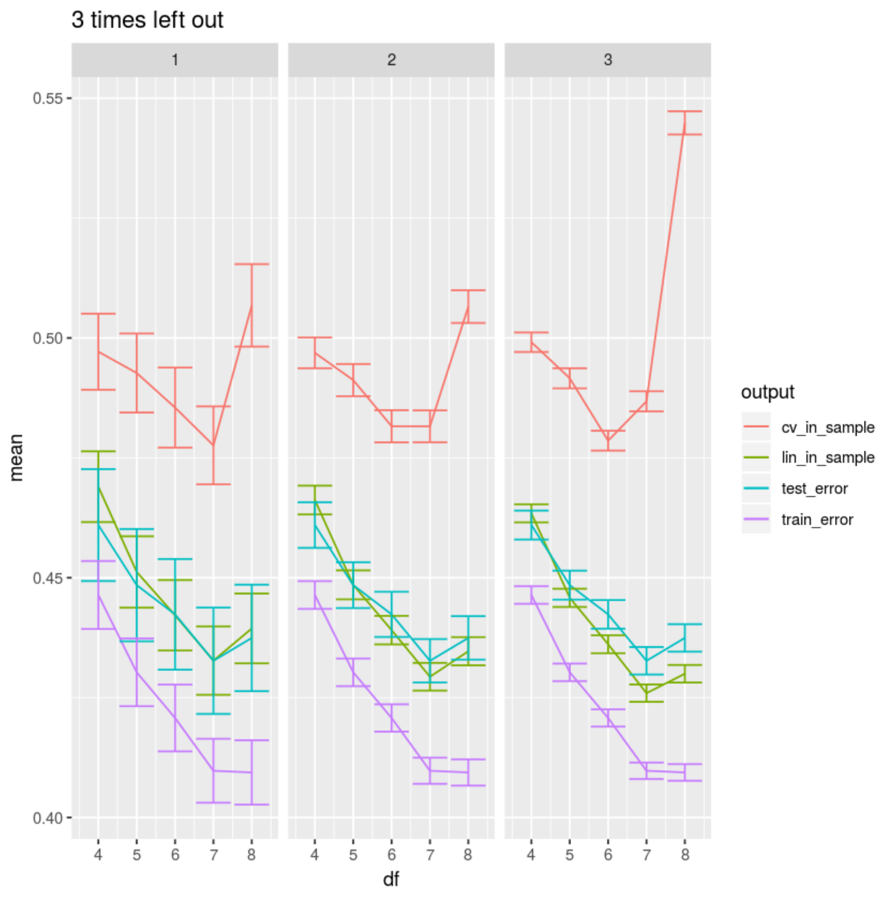
```
In [8]: head(refit_for_df_choice)
```

test	method	comb	rereg	gene	df	lo_num_times	time	measure	error	output
FALSE	lin	0	FALSE	0	4	1	0	err	1.0088933	lin_in_sam
FALSE	lin	0	FALSE	1	4	1	0	err	0.1243607	lin_in_sam
FALSE	lin	0	FALSE	2	4	1	0	err	-0.4340983	lin_in_sam
FALSE	lin	0	FALSE	3	4	1	0	err	-0.2203431	lin_in_sam
FALSE	lin	0	FALSE	4	4	1	0	err	1.9032786	lin_in_sam
FALSE	lin	0	FALSE	5	4	1	0	err	-0.2876837	lin_in_sam

Make a tidy dataframe for choosing `df`. The graph in the paper will be based on this dataframe.

Note that most of the signal for choosing `df` is already in the training data error. However, there is an uptick in error in both CV and IJ for `df=8` which is not captured by the training data error.

```
In [9]: refit_err_summary <-
          refit_for_df_choice %>%
          group_by(output, df, lo_num_times) %>%
          mutate(esize=abs(error)) %>%
          summarize(med=median(esize),
                    mean=mean(esize),
                    n_obs=n(),
                    se=sd(esize) / sqrt(n_obs),
              qlow=quantile(esize, 0.25),
              qhigh=quantile(esize, 0.75))

        ggplot(refit_err_summary) +
          geom_line(aes(x=df, y=mean, group=output, color=output)) +
          geom_errorbar(aes(x=df, ymin=mean - 2 * se, ymax=mean + 2 * se,
                           group=output, color=output)) +
          facet_grid(~ lo_num_times) +
          ggtitle(sprintf("%d times left out", lo_num_times))
```



1.0.4 Gene-by-gene accuracy measures.

```
In [10]: refit_err_plot <-
           refit_err_melt %>%
          filter(rereg==use_rereg) %>%
          dcast(df + lo_num_times + test + comb + rereg + gene + time ~ method + measure,
                value.var=c("value"))
```

We now look at the correlation between the CV and IJ prediction errors across genes. For each `df` and `k`, there are a number of different combinations of left-out points. We report the median, min, and max correlation coefficients across these combinations of left-out points.

First, we show the correlation between the raw prediction errors. Although the correlation is quite high, this is because the training error at the original optimum is the principle source of variation in the errors across genes, and this quantity is common to both CV and IJ.

```
In [11]: err_corr <- refit_err_plot %>%
           filter(test==FALSE, rereg==use_rereg) %>%
           group_by(df, lo_num_times, comb) %>%
           summarize(r=cor(lin_err, ref_err)) %>%
           group_by(df, lo_num_times) %>%
           summarize(med_r=median(r), min_r=min(r), max_r=max(r))

         print("Correlation between error: ")
         print(err_corr)
```

```
[1] "Correlation between error: "
# A tibble: 15 x 5
# Groups:   df [?]
      df lo_num_times med_r min_r max_r
   <int>        <int> <dbl> <dbl> <dbl>
 1     4            1 0.974 0.949 0.984
 2     4            2 0.975 0.902 0.992
 3     4            3 0.967 0.871 0.991
 4     5            1 0.963 0.856 0.983
 5     5            2 0.966 0.860 0.984
 6     5            3 0.947 0.759 0.981
 7     6            1 0.980 0.807 0.985
 8     6            2 0.968 0.835 0.986
 9     6            3 0.929 0.759 0.983
10     7            1 0.962 0.794 0.974
11     7            2 0.952 0.737 0.976
12     7            3 0.914 0.599 0.974
13     8            1 0.962 0.703 0.971
14     8            2 0.941 0.663 0.974
15     8            3 0.829 0.251 0.958
```

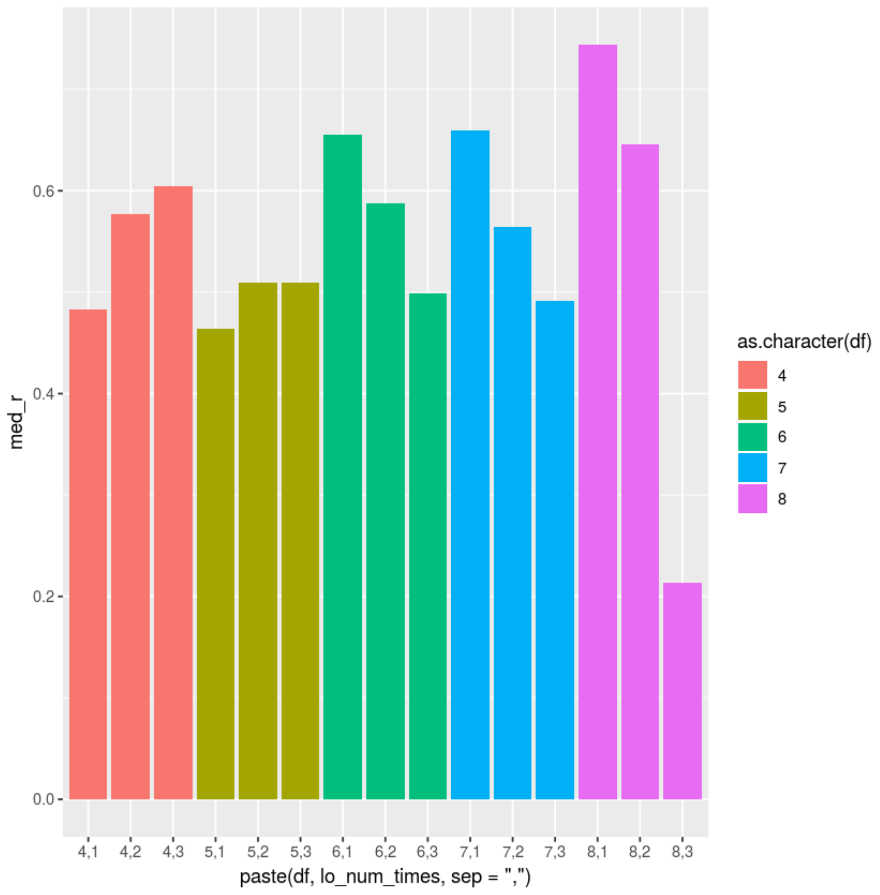
A more meaningful measure is the correlation in the excess error for IJ and CV over the error at the original fit.

```
In [12]: diff_corr <- refit_err_plot %>%
           filter(test==FALSE, rereg==use_rereg) %>%
           group_by(df, lo_num_times, comb) %>%
            summarize(r=cor(lin_e_diff, ref_e_diff)) %>%
            group_by(df, lo_num_times) %>%
            summarize(med_r=median(r), min_r=min(r), max_r=max(r))

          print("Correlation between difference from train error: ")
          print(diff_corr)
```

```
[1] "Correlation between difference from train error: "
# A tibble: 15 x 5
# Groups:   df [?]
      df lo_num_times med_r   min_r max_r
   <int>        <int> <dbl>   <dbl> <dbl>
 1     4            1 0.483  0.0956 0.844
 2     4            2 0.577  0.277  0.828
 3     4            3 0.605  0.303  0.833
 4     5            1 0.464  0.143  0.728
 5     5            2 0.510  0.330  0.709
 6     5            3 0.510  0.312  0.671
 7     6            1 0.655  0.368  0.783
 8     6            2 0.588  0.218  0.845
 9     6            3 0.499  0.0701 0.737
10     7            1 0.660  0.512  0.760
11     7            2 0.564  0.224  0.863
12     7            3 0.491  0.0344 0.801
13     8            1 0.744  0.380  0.900
14     8            2 0.646  0.166  0.862
15     8            3 0.214 -0.226  0.767
```

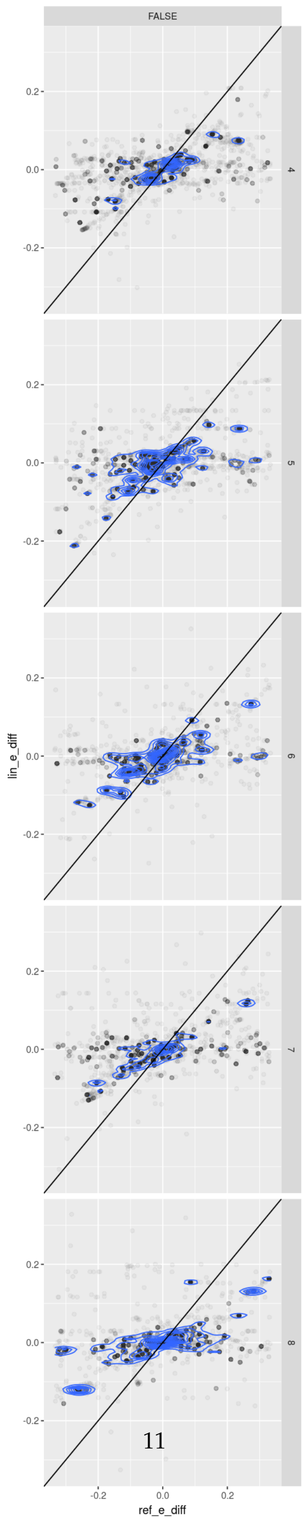
For higher degrees of freedom, increasing the number of left-out points seems to decrease the IJ's accuracy, as you might expect.

```
In [13]: ggplot(diff_corr) +
             geom_bar(aes(x=paste(df, lo_num_times, sep=","),
                          y=med_r, fill=as.character(df)), stat="identity")
```



Plot the densities of the IJ and CV with points to show outliers. This is a graphical version of the results summarized by the correlation tables above.

```
In [14]: # There are a few outliers, so limit the extent of the plot so that
         # the bulk of the distribution is visible.
         qlim <- quantile(refit_err_plot$ref_e_diff, c(0.1, 0.9))

         options(repr.plot.width=4, repr.plot.height=20)

         # This plot, or ones like it, is probably the best measure of
         # the accuracy of the IJ.
         ggplot(filter(refit_err_plot, test == FALSE, lo_num_times==1)) +
             geom_point(aes(x=ref_e_diff, y=lin_e_diff), alpha=0.01) +
             geom_density2d(aes(x=ref_e_diff, y=lin_e_diff)) +
            geom_abline(aes(slope=1, intercept=0)) +
            facet_grid(df ~ rereg) +
            xlim(qlim[1], qlim[2]) + ylim(qlim[1], qlim[2])

Warning message:
Removed 10770 rows containing non-finite values (stat_density2d).Warning message:
Removed 10770 rows containing missing values (geom_point).
```


FALSE
4
5
6
7
8
0.2
0.0
-0.2
-0.2
0.0
0.2
lin_e_diff
ref_e_diff

1.0.5 Save results for plotting in the paper.

```
In [15]: print(sprintf("Saving to %s", file.path(save_dir, save_filename)))
         save(refit_err_summary,
              metadata_df,
              diff_corr,
              err_corr,
              file=file.path(save_dir, save_filename))
```

```
[1] "Saving to ../../fits/paper_results_init_kmeans_rereg_FALSE.Rdata"
```