Econ 675: HW 2

Erin Markiewitz

October 12, 2018

Contents

1 Kernel Density Estimation **3**
1.1 . . . 3
1.2 . . . 4
1.3 . . . 5
1.3.1 . . . 5
1.3.2 . . . 6
1.3.3 . . . 6

2 Linear Smoothing, Cross-Validation and Series **7**
2.1 . . . 7
2.2 . . . 8
2.3 . . . 9
2.4 . . . 10
2.5 . . . 11
2.5.1 . . . 11
2.5.2 . . . 11
2.5.3 . . . 11
2.5.4 . . . 11

3 Semiparametric Semi-Linear Model **12**
3.1 . . . 12
3.2 . . . 13
3.2.1 . . . 13
3.2.2 . . . 13
3.3 . . . 14
3.3.1 . . . 14
3.3.2 . . . 14
3.4 . . . 14

4 Code Appendix **16**

1 Kernel Density Estimation

1.1

First we consider the kernel density derivative estimator. $\hat{f}(x) = \frac{1}{nh}\sum_{i=1}^{N} k\left(\frac{X_i - x}{h}\right)$

The expectation of the estimator is:

$$\mathbb{E}[\hat{f}^{(s)}(x)] = \mathbb{E}[\hat{f}^{(s)}(x, h_n)] = \int_{-\infty}^{\infty} \frac{(-1)^s}{h^{1+s}} k^{(s)}\left(\frac{z-x}{h}\right) f(z)dz$$

where $k^{(s)}$ is the s^{th} derivative of the kernel function Now integrate by parts

$$\int_{-\infty}^{\infty} \frac{(-1)^s}{h^{1+s}} k^{(s)}\left(\frac{z-x}{h}\right) f(z)dz =$$

$$(-h)k^{(s-1)}\left(\frac{z-x}{h}\right) f^{(1)}(z)|_{-\infty}^{\infty} - \int_{-\infty}^{\infty} \frac{(-1)^{s-1}}{h^{1+s-1}} k^{(s)}\left(\frac{z-x}{h}\right) f^{(1)}(z)dz$$

As $k(.)$ is a P^{th} order kernel function and $s-1 < P$, the first term on the RHS of the equation above is equal to zero. Integrating by parts s-1 more times and changing the base, we get the following expression

$$\int_{-\infty}^{\infty} k\left(u\right) f^{(s)}(uh + x)du$$

So now we take a P^{th} order taylor expansion of $f^{(s)}(uh + x)$ around x, which gives us

$$f^{(s)}(x) + \frac{1}{P!}\int_{-\infty}^{\infty} k\left(u\right) f^{(s+P)}(uh + x)(uh + x - x)^p du + o(h_n^P)$$

$$= f^{(s)}(x) + \frac{1}{P!}\int_{-\infty}^{\infty} k\left(u\right) f^{(s+P)}(uh + x)(uh)^p du + o(h_n^P)$$

$$= f^{(s)}(x) + \frac{f^{(s+P)}(x)}{P!}\mu_P(K)h_n^p + o(h_n^p)$$

where $\mu_P(K) = \int_{\mathbb{R}} u^P K(u)du$ - which gives the result. (Note: the second term is the bias of the estimator)

Now consider the variance of the estimator

$$\mathbb{V}[\hat{f}^{(s)}(x)] = \frac{1}{nh^{2+2s}}\mathbb{V}\left[[k^{(s)}\left(\frac{z-x}{h}\right)\right] = \frac{1}{nh^{2+2s}}\mathbb{E}\left[k^{(s)}\left(\frac{z-x}{h}\right)\right]^2 - \frac{1}{n}\mathbb{E}\left[\frac{1}{nh^{1+s}}k^{(s)}\left(\frac{z-x}{h}\right)\right]^2$$

Now using our derivation of the expected value of our estimator we can rewrite the expression above as:

$$\frac{1}{nh^{2+2s}}\mathbb{E}\left[k^{(s)}\left(\frac{z-x}{h}\right)\right]^2 - \frac{1}{n}f^{(s)}(x)^2 + O\left(\frac{1}{n}\right)$$

(This comes from $\{\frac{f^{(s+P)}(x)}{P!}\mu_P(K)h_n^p + o(h_n^p)\}$ being bounded)

So continuing on, we just expand the first term a bit

$$\begin{aligned}
\mathbb{V}[\hat{f}^{(s)}(x)] &= \frac{1}{nh^{2+2s}}\int_{-\infty}^{\infty} k^{(s)}\left(\frac{z-x}{h}\right)^2 f(z)dz - \frac{1}{n}f^{(s)}(x)^2 + O\left(\frac{1}{n}\right) \\
&= \frac{1}{nh^{1+2s}}\int_{-\infty}^{\infty} k^{(s)}(u)\, f(uh+x)du - \frac{1}{n}f^{(s)}(x)^2 + O\left(\frac{1}{n}\right) \\
&= \frac{f(x)}{nh^{1+2s}}\int_{-\infty}^{\infty} k^{(s)}(u)\, du - \frac{1}{n}f^{(s)}(x)^2 + O\left(\frac{1}{n}\right) \\
&= \frac{f(x)\nu_s(k)}{nh^{1+2s}} - \frac{1}{n}f^{(s)}(x)^2 + O\left(\frac{1}{n}\right)
\end{aligned}$$

where $\nu_s(k) = \int_{\mathbb{R}} k^{(s)}(u)^2\, du$ is the roughness of the s^{th} derivative of a given function k. Not that the variance is $O(1/n)$ and as $1/n > 1/nh_n^{2s+1}$, so that means the statement above is equivalent to

$$\mathbb{V}[\hat{f}^{(s)}(x)] = \frac{f(x)\nu_s(k)}{nh^{1+2s}} - \frac{1}{n}f^{(s)}(x)^2 + O\left(\frac{1}{nh_n^{2s+1}}\right)$$

1.2

The optimal bandwith estimator solves the following problem

$$\min_h AIMSE[h] = \min_h \int_{-\infty}^{\infty}\left[\left(h_n^p\mu_p(k)\frac{f^{(P+s)}(x)}{P!}\right)^2 + \frac{\nu_s(k)f(x)}{nh_n^{1+2s}}\right]dx$$

Take first order conditions

$$0 = 2Ph^{2P-1} \int_{-\infty}^{\infty} \left[\left(\mu_p(k) \frac{f^{(P+s)}(x)}{P!} \right)^2 - \frac{(1+2s)\nu_s(k)f(x)}{nh^{2s}} \right] dx$$

$$\frac{2Pnh^{1-2P-2s}}{(1+2s)\nu_s(k)} = \left(\frac{P!}{\mu_p(k)\nu_{(P+s)}(f))} \right)^2$$

$$h_{AIMSE,s} = \left(\frac{(1+2s)\nu_s(k)(P!)^2}{2Pn\mu_p(k)^2\nu_{(P+s)}(f))} \right)^{\frac{1}{1-2P-2s}}$$

$$h_{AIMSE,s} = \left(\frac{(1+2s)(P!)^2}{2Pn} \frac{\nu_s(k)}{\mu_p(k)^2\nu_{(P+s)}(f))} \right)^{\frac{1}{1-2P-2s}}$$

Now for a consistent bandwith estimator we use cross validation procedure from the lecture notes. Cross-Validation minimizes the estimated mean-squared error through a choice of bandwith.

$$h^* = \text{argmin}_{h \in \mathbb{R}^{++}} \ CV(h) = \frac{1}{n^2 h} \sum_{i=1}^{n} \sum_{j=1}^{n} K\left(\frac{X_i - X_j}{h}\right) K\left(\frac{X_i - X_j}{h}\right) - \frac{2}{n} \sum_{i=1}^{n} \hat{f}_{-i}(X_i)$$

where $\hat{f}_{-i}(X_i)$ is the estimated density w/o x_i in the sample.

1.3

1.3.1

We can calculate the optimal bandwith by hand using the equation above for an Epanechnikov kernel, with n = 1000, s = 0 for the following gaussian mixture dgp.

$$x_i \ 0.5N(-1.5, 1.5) + 0.5N(1, 1)$$

So the optimal bandwith is

$$h_{AIMSE,s} = \left(\frac{1}{1000} \frac{\nu_0 s(k)}{\mu_2(k)^2 \nu_{(2)}(f))} \right)^{\frac{1}{1-4}}$$

From the first chapter of Hansen's chapter 1 notes: we now that $\nu(K) = 3/5$ and $\mu_2(K) = 1/5$. Next as the second derivative of the normal density is

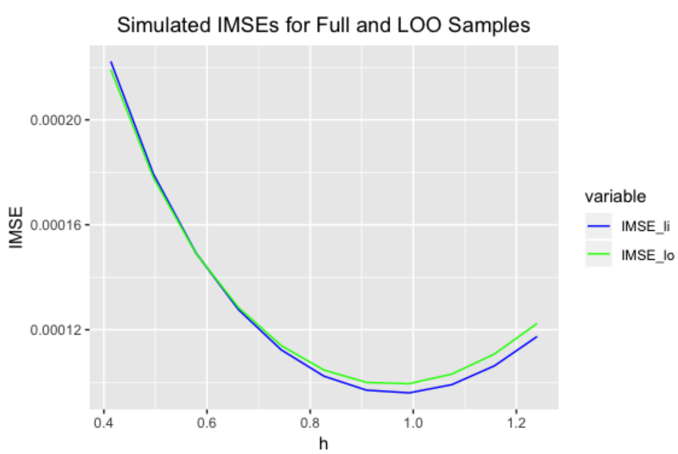
$$\phi^{(2)}(\mu, \sigma^2) = \frac{1}{2n\sigma^2} e^{-\frac{x-\mu}{\sigma}} \left(\left(\frac{x-\mu}{\sigma} \right)^2 - \frac{1}{\sigma^2} \right)$$

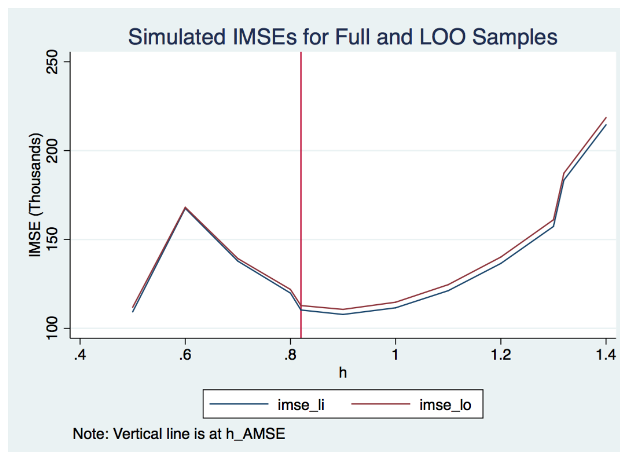
So we can rewrite

$$\nu_{(2)}(f)) = \int_{-\infty}^{\infty} \left(0.5\phi^{(2)}(-1.5, 1.5) + 0.5\phi^{(2)}(1, 1)\right)^2 dx$$

Which gives us a $h_{AIMSE,s} = 0.82$

1.3.2





It appears that the two monte carlo experiments in stata and r do a reasonable job at converging to the theoretically optimal bandwith. Although, they are both slightly off. Although theoretically the two numbers should converge as M $\to \infty$,

1.3.3

Considering a rule-of-thumbs estimate of the bandwith, we assume the DGP is gaussian, so using the equation from 1.3.1 we find that.

$$\bar{h}_{AIMSE} = M^{-1} \sum_{m=1}^{M} \hat{h}_{AIMSE,m} \approx 987147$$

which is significantly biased upwards relative to the asymptotic optimal or the simulation estimates.

2 Linear Smoothing, Cross-Validation and Series

2.1

Local polynomial regression solves the following problem:

$$\hat{\beta}_{LPR} = \text{argmin}_{\beta \in \mathbb{R}^P+1} \frac{1}{n}\sum_{i=1}^{N} (Y_i - r_p(x-x)\beta)^2 K(\frac{x_i - x}{h})$$

where $r_p(u) = (1, u, u^2, ..., u^p)'$ The true regression function $e(x_i)$ is estimated by $\hat{e}(x) = \hat{\beta}_{LPR}$, which can be rewritten as a weighted least-squares problem where $\hat{\beta}_{LPR}(x) = (\mathbf{R_p' W R_p)^{-1} R_p' W Y}$ where the weighting matrix is a diagonal matrix with the kernel functions of the x_i.

where

$$\mathbf{R}_p = \begin{bmatrix} 1 & (x_1 - x) & (x_1 - x)^2 & \cdots & (x_1 - x)^p \\ \vdots & \cdots & \ddots & \ddots & \vdots \\ 1 & (x_n - x) & \cdots & \cdots & (x_n - x)^p \end{bmatrix}$$

and $\mathbf{W}$ is a matrix with kernel weights of x_is on the diagonal

$$\mathbf{W} = \begin{bmatrix} K(\frac{x_1 - x}{h}) & 0 & 0 & 0 \\ 0 & K(\frac{x_2 - x}{h}) & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & K(\frac{x_n - x}{h}) \end{bmatrix}$$

So we can rewrite the estimator of our regression equation as
$\hat{e}(x) = \mathbf{e_1'} \hat{beta}_{LPR} = R_p' W R_p)^{-1} R_p' W Y$
where $\mathbf{e_1}$ is a basis vector of length $1 + p$.
Therefore we can rewrite the estimator above as a sum.

$$\hat{e}(x) = \mathbf{e_1'} \left(\sum_{i=1}^{n} r_p(x_i - x) r_p(x_i - x)' K\left(\frac{x_i - x}{h}\right)\right)^{-1} \left(\sum_{i=1}^{n} r_p(x_i - x) r_p(x_i - x) y_i K\left(\frac{x_i - x}{h}\right)\right)$$

Now we consider the series estimator, which solves the following problem

$$\hat{\beta}_s = \text{argmin}_{\beta \in \mathbb{R}^{k_n}} \frac{1}{n}\sum_{i=1}^{N} (Y_i - r_{k_n}(x)\beta)^2 K(\frac{x_i - x}{h})$$

where $r_{k_n}(x)$ is the basis of some series defined on x, so that

$$\hat{e}(x) = \mathbf{r_{k_n}(x)}'\hat{\beta}$$

where

$$\hat{\mathbf{beta_s}} = \left(\mathbf{R_p'R_p}\right)^{\mathbf{-1}} \mathbf{R_p Y}$$

and

$$\mathbf{R}_p = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^p \\ \vdots & \cdots & \ddots & \ddots & \vdots \\ 1 & x_n & \cdots & \cdots & x_n^p \end{bmatrix}$$

So we can rewrite the estimated regression function as

$$\hat{e}(x) = \mathbf{r_p(x)}' \left(\mathbf{R_p'R_p}\right)^{\mathbf{-1}} \mathbf{R_p Y}$$

and

$$\hat{e}(x) = \mathbf{r_p(x)}' \left(\sum_{i=1}^{n} r_p(x_i) r_p(x_i)'\right)^{-1} \left(\sum_{i=1}^{n} r_p(x_i) y_i\right)$$

2.2

Next, we need to show the following simplified cross-validation formula holds for local polynomial regression and series estimation:

$$CV(c) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{e}(x_i))^2 = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{e}_{(i)}(x_i)}{1 - w_{n,1}(x_i)}\right)^2$$

where c is a tuning parameter (h_n for LPR or a truncation K for series estimators)

Note from the first part of the question, we found that we can right both regressor estimators as a weighted average of the outcome variable

$$\hat{e}(x) = \frac{1}{n}\sum_{i=1}^{n} w_{n,i}(x) y_i$$

where $w_{n,i}(x) = w_{n,i}(x_1, x_s, \cdots, x_n; x)$

Now in estimation of the tuning parameter, we need our smoothing parameter $w_{n,i}$ to be consistent when we "leave one (x_i) out" for estimation. Our smoothing parameter sums to one in the case of the LPR and series estimators. So for cross validation, we need to adjust accordingly.

$$\hat{e}_{(i)}(x) = \frac{1}{1 - w_{ii}} \sum_{i=1}^{n} w_{i,j}(x) y_i$$

So to get our result:

$$\begin{aligned}
(1 - w_{ii})\hat{e}_{(i)}(x) &= \sum_{j \neq i, j=1}^{n} w_{i,j}(x) y_j \\
\hat{e}_{(i)}(x) &= \sum_{j \neq i, j=1}^{n} w_{i,j}(x) y_j + w_{i,i}\hat{e}_{(i)}(x) \\
&= \sum_{i=1}^{n} w_{i,j}(x) y_i + w_{ii}\hat{e}_{(i)}(x) - w_{i,i} y_i \\
&= \hat{e}(x) + w_{ii}\hat{e}_{(i)}(x) - w_{i,i} y_i
\end{aligned}$$

Which gives us

$$\begin{aligned}
y_i - \hat{e}_{(i)}(x) &= y_i - \hat{e}_(x) - w_{i,i}\hat{e}_{(i)}(x) + w_{i,i} y_i \\
y_i - \hat{e}_{(i)}(x) &= y_i - \hat{e}_(x) w_{i,i}(y_i - \hat{e}_{(i)}(x)) \\
(1 - w_{i,i})(y_i - \hat{e}_{(i)}(x)) &= y_i - \hat{e}_(x) \\
y_i - \hat{e}_{(i)}(x) &= \frac{y_i - \hat{e}_(x)}{1 - w_{i,i}}
\end{aligned}$$

So it follows

$$\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{e}(x_i))^2 = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{y_i - \hat{e}_{(i)}(x_i)}{1 - w_{n,1}(x_i)} \right)^2$$

2.3

If we assume the data is iid and a finite first moment, we can show consistency:

$$\begin{aligned}
\mathbb{E}[\hat{e}(x)|x] &= \mathbb{E}\left[\sum_{i=1}^{n} w_{n,i}(x_i) y_i \right] \\
&= \sum_{i=1}^{n} w_{n,i}(x_i) \mathbb{E}[y_i|x] \\
&= \mathbb{E}[y_i|x]
\end{aligned}$$

as $\sum_{i=1}^{n} w_{n,i}(x_i) = 1$
Now if we assume a finite second moment of our regressor estimator

$$\begin{aligned}\mathbb{V}[\hat{e}(x)|x] &= \mathbb{V}\left[\sum_{i=1}^{n} w_{n,i}(x_i)y_i\right] \\ &= \sum_{i=1}^{n} \mathbb{V}[w_{n,i}(x_i)y_i|x] \\ &= \sum_{i=1}^{n} w_{n,i}(x_i)^2\mathbb{V}[y_i|x] \\ &= \mathbb{V}[y_i|x]\sum_{i=1}^{n} w_{n,i}(x_i)^2\end{aligned}$$

So the variance estimator is

$$\hat{V}(x) = \left(\frac{1}{1-n}\sum_{i=1}^{n}(y_i - \hat{e}(x_i))^2\right)\left(\sum_{i=1}^{n} w_{n,i}(x_i)^2\right)$$

Asymptotic normality follows from the CLT

2.4

The
A pointwise asymptotically valid confidence interval for a fixed x is

$$CI_{95} = \left[\hat{e}(x) - 1.96\sqrt{\hat{V}(x)/n}; \hat{e}(x) + 1.96\sqrt{\hat{V}(x)/n}\right]$$

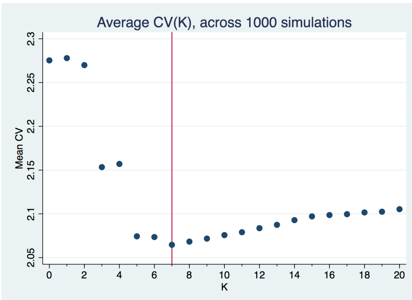
but this convidence invterval is not asymptotically valid across the entire support. In order for the confidence interval to be uniformly aymptotically valid we need the interval to hold across the entire support of x

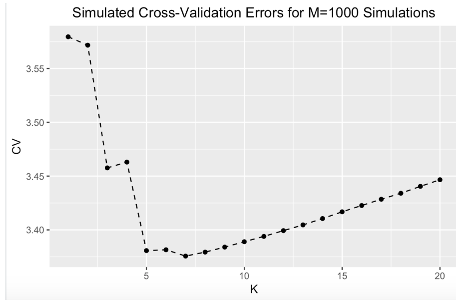
$$\text{sup}_{x\in X}\left|\frac{\hat{e}(x) - e(x))}{\sqrt{\hat{V}(x)}}\right| \leq q_{1-\alpha/2}$$

2.5

2.5.1

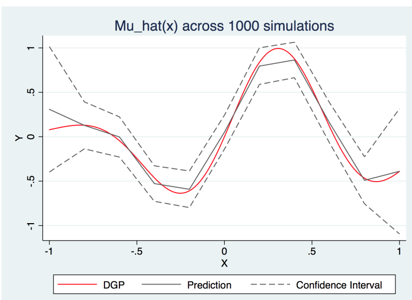
2.5.2

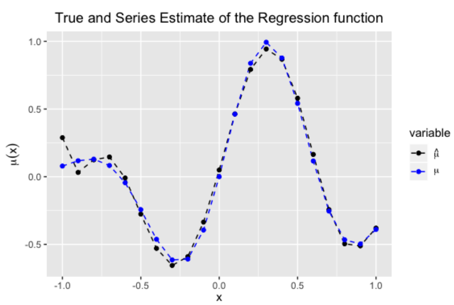




The two cross validation results from stata and R match up pretty closely.

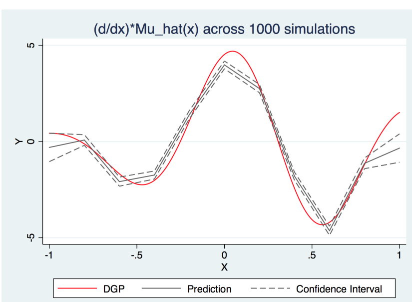
2.5.3

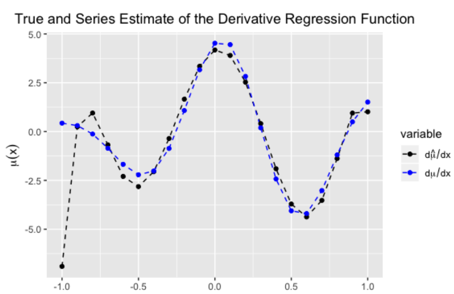




The two kernel estimatation results from stata and R match up pretty closely.

2.5.4





The two kernel estimatation results from stata and R match up pretty closely. Although, the confidence interval was off for the stata code and I was unable to fix it before the deadline.

3 Semiparametric Semi-Linear Model

3.1

The following question concerns this moment condition:

$$\mathbb{E}[(t_i - h_0(x_i))(y_i - t_i\theta)] = 0 \text{ , where } h_0(x_i) = \mathbb{E}[t_i|x_i]$$

As long as t_i is not collinear with x_i then θ_0 will be identifiable. Assuming that θ_0 is identifiable, it satisfies the moment condition above:

$$\begin{gathered}
\mathbb{E}[t_i y_i] + \mathbb{E}[h_0(x_i)t_i\theta] - \mathbb{E}[h_0(x_i)y_i] - \mathbb{E}[t_i t_i\theta)] = 0 \\
\mathbb{E}[\mathbb{E}[t_i y_i|t_i, x_i]] + \mathbb{E}[\mathbb{E}[h_0(x_i)t_i\theta|t_i, x_i]] - \mathbb{E}[\mathbb{E}[h_0(x_i)y_i|t_i, x_i]] + \mathbb{E}[\mathbb{E}[t_i t_i\theta)|t_i, x_i]] = 0 \\
\mathbb{E}[h_0(x_i)\mathbb{E}[y_i|t_i, x_i]] + \mathbb{E}[h_0(x_i)h_0(x_i)]\theta - \mathbb{E}[h_0(x_i)\mathbb{E}[y_i|t_i, x_i]] + \mathbb{E}[h_0(x_i)h_0(x_i)]\theta = 0 \\
0 = 0
\end{gathered}$$

To derive a closed form equation for θ_0 we follow the steps outlined in Hansen's notes on nonparametrics (chapter 7), which describes Robinson (Econometrica, 1988).

$$y_i = t_i\theta_0 + g(x_i) + \epsilon_i$$

First we take the conditional expectation with respect to the treatment and other covariates. (We assume the treatment is not collinear with the other covariates.)

$$\mathbb{E}[y_i|t_i x_i] = \mathbb{E}[t_i|t_i x_i]\theta_0 + \mathbb{E}[g(x_i)|t_i x_i] + 0\mathbb{E}[y_i|t_i x_i] = h_o(x_i)\theta_0 + g(x_i) + 0$$

Next, let's define $g_{y,x} \coloneqq \mathbb{E}[y_i|t_i x_i]$, and subtract the equation above from the original regression.

$$y_i - g_{y,x} = (t_i - h_o(x_i))\theta_0 + g(x_i) - g(x_i) + \epsilon_i$$

Now, we can rewrite the regression as a residual regression:

$$\begin{gathered}
\epsilon_{yi} = \epsilon_{ti}\theta_0 + \epsilon_i \\
y_i = g_{y,x} + \epsilon_{yi} \\
t_i = h_o(x_i) + \epsilon_{ti}
\end{gathered}$$

Which produces the infeasible estimtor:

$$\beta = \left(\sum_{i=1}^{n} \epsilon_{ti}\epsilon_{ti}'\right)^{-1} \left(\sum_{i=1}^{n} \epsilon_{ti}\epsilon_{yi}'\right)$$

Note that we can rewrite the residual regression as :

$$M_{yx}y_i = M_{tx}t_i\theta_0 + \epsilon_i$$

Which is the second stage of an IV regression that partials out the effects of X_i on y_i and t_i using anhilation matrixes.

3.2

3.2.1

If the treatment is undetermined by the power series of the covariates, θ_0 is simply

$$\theta_0 = (T'T)^{-1}(T'Y)$$

which has a feasible estimator of

$$\hat{\theta}(K) = (\sum_{i=1}^{n} t_i t_i)^{-1}(\sum_{i=1}^{n} t_i y_i)$$

3.2.2

If the treatment is correlated to the other covariates, in order to estimate a feasible estimator, one must run Nadaraya - Watson kernel regressions of the outcome and treatment variables onto the power series.

$$\hat{y}_i = \frac{\sum\limits_{i=1}^{n} k\left(\frac{p^{K_n}(x_i) - p^{K_n}(x)}{h}\right) y_i}{\sum\limits_{i=1}^{n} k\left(\frac{p^{K_n}(x_i) - p^{K_n}(x)}{h}\right)}$$

$$h_0(x_i) = \frac{\sum\limits_{i=1}^{n} k\left(\frac{p^{K_n}(x_i) - p^{K_n}(x)}{h}\right) t_i}{\sum\limits_{i=1}^{n} k\left(\frac{p^{K_n}(x_i) - p^{K_n}(x)}{h}\right)}$$

Now, construct residualize

$$\hat{\epsilon}_{yi} = y_i - \hat{y}_i = M_{yx}y_i$$
$$\hat{\epsilon}_{ti} = t_i - h_0(x_i) = M_{tx}t_i$$

Which produces the feasible estimator

$$\hat{\theta}(K) = \left(\sum_{i=1}^{n} \hat{\epsilon}_{ti}\hat{\epsilon}'_{ti}\right)^{-1}\left(\sum_{i=1}^{n} \hat{\epsilon}_{ti}\hat{\epsilon}'_{yi}\right)$$

3.3

3.3.1

Fixing K, the reason this approach is called a "flexible parametric" estimation because you are estimating θ_0, while letting

If $K \to \infty$ does not invalidate the "fixed K" assumption as long as the ratio between the observations and covariates is fixed $\left(\frac{K_n}{n} = \frac{\bar{K}}{\bar{n}}\right)$

3.3.2

Using the results above the confidence interval is

$$CI_{95} = \left[\hat{\theta}(K) - 1.96\sqrt{\hat{V}_{HCO}/n}; \hat{\theta}(K) + 1.96\sqrt{\hat{V}_{HCO}/n}\right]$$

3.4

k	coverage rate	mean $\hat{V}$	mean $\hat{\theta}$	SD of $\hat{\theta}$	mean bias	$\mathbb{V}[\hat{\theta}]$
1	0	.0121515	3.038587	.5610881	2.038587	.3148198
2	.001	.0192048	.5193815	.4861081	-.4806185	.2363011
3	.001	.0192048	.5193815	.4861081	-.4806185	.2363011
4	.001	.0191404	.5346238	.5019035	-.4653762	.2519072
5	.001	.0191404	.5346238	.5019035	-.4653762	.2519072
6	0	.0190032	.5710299	.4311689	-.4289701	.1859066
7	0	.0190032	.5710299	.4311689	-.4289701	.1859066
8	.001	.0189843	.5708849	.4300738	-.4291151	.1849635
9	.001	.0189843	.5708849	.4300738	-.4291151	.1849635
10	.001	.0189318	.5813037	.4359568	-.4186963	.1900583
11	.001	.0189036	.584183	.453701	-.415817	.2058446
12	.001	.0187963	.6144832	.4626188	-.3855168	.2140162
13	0	.0187873	.6093753	.4969279	-.3906247	.2469373
14	.001	.0188136	.5898626	.4697601	-.4101374	.2206745

At this point my estimates for stata are way off. I get a terrible coverage rate. I think it could be due to a number of factors. I attempted to build in a cross validation excercise into my program, but it was giving me weird results to I suppressed the output. I am confident that the optimal k by CV estimates is around 126.

k	mean $\hat{V}$	bias $\hat{\theta}$	SD of $\hat{\theta}$	$\mathbb{V}[\hat{\theta}]$	coverage rate
6	2.856	1.856	0.481	0.465	0.972
11	0.827	-0.173	0.227	0.216	0.113
21	0.833	-0.167	0.227	0.214	0.113
26	0.834	-0.166	0.229	0.214	0.112
56	0.852	-0.148	0.226	0.201	0.131
61	0.867	-0.133	0.22	0.195	0.127
126	1.009	0.009	0.113	0.101	0.089
131	1.01	0.01	0.114	0.101	0.092
252	1.007	0.007	0.123	0.094	0.134
257	1.007	0.007	0.124	0.094	0.131
262	1.007	0.007	0.125	0.094	0.141
267	1.007	0.007	0.126	0.094	0.145
272	1.007	0.007	0.128	0.095	0.145
277	1.008	0.008	0.13	0.095	0.152

The R code produces more reasonable results. This shows that including more terms in the polynomial does not neccesarly decrease bias. The cross validation exercise in R says that the optimal k is 126.

4 Code Appendix

Stata

```
clear all
set more off, perm
set seed 12345

global dir "/Users/erinmarkiewitz/Dropbox/Phd_Coursework/Econ675/hw2"
global datadir $dir\data
global resdir $dir\results

cap log close
log using $resdir\pset2_stata.smcl, replace


***********************************
************ Question 1 ************
***********************************
/*
   * Some values
   global M = 1000 //number of iterations
   global n = 1000
   global hvalues .5 .6 .7 .8 0.8199 .9 1 1.1 1.2 1.3 1.4 1.5
   mat hvalues = (0.8199, .5, .6, .7, .8, .9, 1, 1.1, 1.2, 1.3, 1.4, 1.5)

   *DGP Values
   global mu1 = -1.5
   global mu2 = 1
   global sd1 = sqrt(1.5)
   global sd2 = 1

   mata:
   //**********FUNCTIONS***********
   // function for calculating kernel
   real scalar function kern(real scalar u){
   return(.75*(1-u^2)*(abs(u)<=1))
   }

   // function for calculating true density
   real scalar function f_true(real scalar u){
   return(.5*normalden(u,-1.5,sqrt(1.5)) + .5*normalden(u,1,1))
   }

   // function for calculating MSE (LI & LO)
   real vector function mse(real vector xdata, real scalar hvalue){
   //Construct two matrices of xdata
   M1 = J($n,$n,.) // n x n matrix with one column for each observation
   M2 = J($n,$n,.) // n x n matrix with one row for each observation
   for (i=1; i<= $n; i++) {
   v = J($n,1,xdata[i])
   M1[,i] = v
   M2[i,] = v'
   }

   M3 = (M1-M2)/hvalue //object to be evaluated by kernel
   M4 = J($n,$n,.)
   M5 = J($n,$n,.)
   fx = J($n,1,.)

   for (i=1; i<=$n; i++){
   for (j=1; j<=$n; j++){
   M4[i,j] = kern(M3[i,j])
   }
   M5[i,] = M4[i,]
   M5[i,i]=0

   fx[i,1] = f_true(xdata[i])
   }

   fhat_LI = rowsum(M4)/($n*hvalue)
   fhat_LO = rowsum(M5)/(($n-1)*hvalue)

   sqe_LI = (fhat_LI-fx):^2
   sqe_LO = (fhat_LO-fx):^2

   mse_LI = mean(sqe_LI)
   mse_LO = mean(sqe_LO)

   return((mse_LI,mse_LO))
   }

   // function for importing/exporting to mata for mse calculation
   void iteration(real scalar m){
   x= st_data(.,.)
   hvalues = st_matrix("hvalues")

   mse = J(12,2,.)
   for (h=1; h<=12; h++){
```

```
mse[h,] = mse(x,hvalues[1,h])
}
st_matrix("msetemp",mse)
}
end


*Empty matrix to be filled
mat msesum = J(12,2,0)

*Loop through iterations
timer on 1
forval m = 1/$M{
disp `m'
set obs $n

*equally weight two normal distributions
gen comps = uniform() >= .5

*generate sample
gen x = comps*rnormal($mu1,$sd1) + (1-comps)*rnormal($mu2,$sd2)
drop comps

*call mata function to calculate mse
mata iteration(`m')
drop x
mat msesum = msesum + msetemp
}
timer off 1
timer list

mat imse = msesum*1000
svmat imse
rename imse1 imse_li
rename imse2 imse_lo

egen h = fill(.5, .6, .7, .8, 0.8199, .9, 1, 1.1, 1.2, 1.3, 1.4, 1.5)

twoway(line imse_li h)(line imse_lo h), ytitle("IMSE (Thousands)") ///
xtitle("h") xline(0.8199) caption("Note: Vertical line is at h_AMSE")
graph export $resdir/pset2q1.png, replace

*?

**************
**** Problem 2
**************

**************
**** Problem 2a-b
***************
set obs 1000


* Define cross validation function: CV(list, i): vars=variable list, i = max polynomial
mata
void CV(vars, i) {
st_view(y=., ., "y")
st_view(X=., ., tokens(vars))
XpX  = cross(X, X)
XpXinv  = invsym(XpX)
b  = XpXinv*cross(X, y)
w = diagonal(X*XpXinv*X')
muhat = X*b
num = (y - muhat):*(y - muhat)
den= (J(1000,1,1) - w):*(J(1000,1,1) - w)
div = num:/den
CV = mean(div)
CV
st_numscalar("mCV"+strofreal(i), CV)
}
end


* Program which runs the monte-carlo experiment
program CVsim, rclass
drop _all
set obs 1000
forvalues i = 0/20 {
gen CV`i' = 0
}
gen x = runiform(-1,1)
gen e = x^2*(rchi2(5)-5)
gen y = exp(-0.1*(4*x-1)^2)*sin(5*x)+e
forvalues i = 0/20 {
gen x`i' = x^`i'
}
forvalues i = 0/20 {
global xlist = "x0-x`i'"
di "$xlist"
mata CV("$xlist", `i')
replace CV`i' = mCV`i'
```

```
}
end



* Run the experiment
set seed 12345
simulate CV0=CV0 CV1=CV1 CV2=CV2 CV3=CV3 CV4=CV4 CV5=CV5 CV6=CV6 CV7=CV7 CV8=CV8 ///
CV9=CV9 CV10=CV10 CV11=CV11 CV12=CV12 CV13=CV13 CV14=CV14 CV15=CV15 ///
CV16=CV16 CV17=CV17 CV18=CV18 CV19=CV19 CV20=CV20, reps(100) nodots: CVsim
collapse *
gen i = 1
reshape long CV, i(i) j(k)
sort CV
local min = k[1]
twoway scatter CV k, ytitle("Mean CV") xtitle("K") xlabel(0(2)20) xmtick(0(1)20) xline(`min') title("Average CV(K), across 1000 simulations")
graph export $resdir\pset2q2b.png, replace


**************
***Problem 2c
***************

* Program which runs the monte-carlo experiment for mu_0
program muhatsim, rclass
drop _all
set obs 1000
gen x = runiform(-1,1)
gen e = x^2*(rchi2(5)-5)
gen y = exp(-0.1*(4*x-1)^2)*sin(5*x)+e
forvalues p = 0/7 {
gen x`p' = x^`p'
}
reg y x0-x7, nocons
clear
set obs 11
gen n = _n
gen foo = 1
gen x = -1+(_n-1)/5
forvalues p = 0/7 {
gen x`p' = x^`p'
}
predict muhat
predict se, stdp
generate lb = muhat - invnormal(0.975)*se
generate ub = muhat + invnormal(0.975)*se



keep n muhat foo lb ub
reshape wide muhat lb ub, i(foo) j(n)
end



set seed 12345
simulate muhat1=muhat1 muhat2=muhat2 muhat3=muhat3 muhat4=muhat4 muhat5=muhat5 ///
muhat6=muhat6 muhat7=muhat7 muhat8=muhat8 muhat9=muhat9 muhat10=muhat10 muhat11=muhat11 ///
ub1=ub1 ub2=ub2 ub3=ub3 ub4=ub4 ub5=ub5 ub6=ub6 ub7=ub7 ub8=ub8 ub9=ub9 ub10=ub10 ub11=ub11 ///
lb1=lb1 lb2=lb2 lb3=lb3 lb4=lb4 lb5=lb5 lb6=lb6 lb7=lb7 lb8=lb8 lb9=lb9 lb10=lb10 lb11=lb11, reps(1000) nodots: muhatsim
gen i = _n
reshape long muhat ub lb, i(i) j(grid)
collapse muhat ub lb, by(grid)
gen x = -1+ (grid-1)/5
twoway (function y = exp(-0.1*(4*x-1)^2)*sin(5*x), range(-1 1) lcolor(red)) ///
(line muhat x, lcolor(gs6)) (line lb x, lcolor(gs6) lpattern(dash)) (line ub x, lcolor(gs6) lpattern(dash)), ///
legend(order(1 "DGP" 2 "Prediction" 3 "Confidence Interval") rows(1)) ytitle(Y) xtitle(X) title("Mu_hat(x) across 1000 simulations")
graph export $resdir\pset2q2c.png, replace


**************
***Problem 2d
***************

* Program which runs the monte-carlo experiment for mu_1
program dmuhatsim, rclass
drop _all
set obs 1000
gen x = runiform(-1,1)
gen e = x^2*(rchi2(5)-5)
gen y = exp(-0.1*(4*x-1)^2)*((0.8-3.2*x)*sin(5*x)+5*cos(5*x)) + e
forvalues p = 0/7 {
gen x`p' = x^`p'
}
reg y x0-x7, nocons
clear
set obs 11
gen n = _n
```

```
    gen foo = 1
    gen x = -1+(_n-1)/5
    forvalues p = 0/7 {
    gen x`p' = x^`p'
    }
    predict dmuhat
    predict se, stdp
    generate lb = dmuhat - invnormal(0.975)*se
    generate ub = dmuhat + invnormal(0.975)*se



    keep n dmuhat foo lb ub
    reshape wide dmuhat lb ub, i(foo) j(n)
    end


    set seed 12345
    simulate dmuhat1=dmuhat1 dmuhat2=dmuhat2 dmuhat3=dmuhat3 dmuhat4=dmuhat4 dmuhat5=dmuhat5 ///
    dmuhat6=dmuhat6 dmuhat7=dmuhat7 dmuhat8=dmuhat8 dmuhat9=dmuhat9 dmuhat10=dmuhat10 dmuhat11=dmuhat11 ///
    ub1=ub1 ub2=ub2 ub3=ub3 ub4=ub4 ub5=ub5 ub6=ub6 ub7=ub7 ub8=ub8 ub9=ub9 ub10=ub10 ub11=ub11 ///
    lb1=lb1 lb2=lb2 lb3=lb3 lb4=lb4 lb5=lb5 lb6=lb6 lb7=lb7 lb8=lb8 lb9=lb9 lb10=lb10 lb11=lb11, reps(1000)
         nodots: dmuhatsim
    gen i = _n
    reshape long dmuhat ub lb, i(i) j(grid)
    collapse dmuhat ub lb, by(grid)
    gen x = -1+ (grid-1)/5
    twoway (function y = exp(-0.1*(4*x-1)^2)*((0.8-3.2*x)*sin(5*x)+5*cos(5*x)), range(-1 1) lcolor(red))
        ///
    (line dmuhat x, lcolor(gs6)) (line lb x, lcolor(gs6) lpattern(dash)) (line ub x, lcolor(gs6) lpattern(
        dash)), ///
    legend(order(1 "DGP" 2 "Prediction" 3 "Confidence Interval") rows(1)) ytitle(Y) xtitle(X) title("(d/dx)
        *Mu_hat(x) across 1000 simulations")
    graph export $resdir\pset2q2d.png, replace




*/ */

*******
*** Problem 3
*******

drop _all
set obs 1000
local theta = 1
local d = 5
local n = 500

forvalues p = 1/14 {
        gen v_hat`p' = .
        gen theta_hat`p' = .

}


mata:
        void polyloop(i) {
                X       = uniform(`n',`d'):*2 :-1
                ep      = invnormal(uniform(`n',1)):*0.3637899:*(1 :+ rowsum(X:^2))
                gx      = exp(rowsum(X:^2))
                T       = invnormal(uniform(`n',1)) + rowsum(X:^2):^.5 :>= 0
                Y   = T + gx + ep
                cons= J(500,1,1)
                /*Raising to single powers */
                X2      = X:^2
                X3      = X:^3
                X4      = X:^4
                X5      = X:^5
                X6      = X:^6
                X7      = X:^7
                X8      = X:^8
                X9      = X:^9
                X10 = X:^10
                /*Kronekering, but this creates some duplicates*/
                X1k = X#X
                X2k = X2#X2
                X3k = X3#X3
                X4k = X4#X4
                /* Manually removing duplicates...might be a better way to do this */
                X1k = X1k[1::`n',2::5], X1k[1::`n', 8::10], X1k[1::`n',14::15], X1k[1::`n', 20]
                X2k = X2k[1::`n',2::5], X2k[1::`n', 8::10], X2k[1::`n',14::15], X2k[1::`n', 20]
                X3k = X3k[1::`n',2::5], X3k[1::`n', 8::10], X3k[1::`n',14::15], X3k[1::`n', 20]
                X4k = X4k[1::`n',2::5], X4k[1::`n', 8::10], X4k[1::`n',14::15], X4k[1::`n', 20]
                A = asarray_create("real",1)
                asarray(A,1,X)
                asarray(A,2,(asarray(A,1),X2))
                asarray(A,3,(asarray(A,2),X1k))
                asarray(A,4,(asarray(A,3),X3))
                asarray(A,5,(asarray(A,4),X2k))
                asarray(A,6,(asarray(A,5),X4))
```

```
                asarray(A,7,(asarray(A,6),X3k))
                asarray(A,8,(asarray(A,7),X5))
                asarray(A,9,(asarray(A,8),X4k))
                asarray(A,10,(asarray(A,9),X6))
                asarray(A,11,(asarray(A,10),X7))
                asarray(A,12,(asarray(A,11),X8))
                asarray(A,13,(asarray(A,12),X9))
                asarray(A,14,(asarray(A,13),X10))
                theta_hat = I(1,14):*0
                se_hat = I(1,14):*0
                k_hat = I(1,14):*0

                for (j=1; j<=14; j++) {
                        Z = qrsolve(cons,(T,asarray(A,j)))
                        ZZ  = Z*Z'
                        Yhat = ZZ*Y
                        W = diag(ZZ)
                        ///CV_out = diag(mean_vec*(Y - Yhat) / (1 - W)^2)//

                        ZQ = (cons,asarray(A,j))*invsym((cons,asarray(A,j))'*(cons,asarray(A,j)))*(cons,
                            asarray(A,j))'
                        M = I(`n') - ZQ
                        YM = M*Y
                        TM = M*T
                        theta_hat[1,j] = (TM'*YM) / (TM'*TM)
                        sigma = diag(ZQ*(Y-T*theta_hat[1,j]))
                        se_hat[1,j] = sqrt(invsym(T'*ZQ*T)*(T'*ZQ*sigma*ZQ*T)*invsym(T'*ZQ*T))
                        st_store(i, "v_hat"+strofreal(j), se_hat[1,j])
                        st_store(i, "theta_hat"+strofreal(j), theta_hat[1,j])

                        }
                        }
                        end
forvalues i = 1/10 {
        mata polyloop(`i')
}

save output_q3_temp.dta, replace

use output_q3,clear
gen obs = _n
reshape long v_hat theta_hat, i(obs) j(k)

gen abs_theta_hat = abs(theta_hat)
egen sd_theta_hat = sd(theta_hat) ,by(k)
gen coverage_rate = 1 if abs_theta_hat <= 1.96 * sd_theta_hat/sqrt(1000)
replace coverage_rate = 0 if coverage_rate == .

collapse (mean) coverage_rate mean_v_hat= v_hat  mean_theta_hat=theta_hat (sd) sd_theta_hat = theta_hat,
    by(k)
gen mean_bias = mean_theta_hat  - 1
gen v_theta_hat = sd_theta_hat^2

*coverage rate test
log close
translate $resdir\pset2_stata.smcl $resdir\pset2_stata.pdf, replace
```

R

```
## ECON675: PS 2
## Erin Markiewitz
## 10/12/2018
#########################################################################
# Load packages, clear workspace
#########################################################################
rm(list = ls())              #clear workspace
library(foreach)
library(dplyr)               #for data manipulation
library(data.table)          #for data manipulation
library(ggplot2)             #for pretty plots
library(boot)                #for bootstrapping
options(scipen = 999)        #forces R to use normal numbers instead of scientific notation



#########################################################################
# Q3 (a): compute theoretically optimal bandwidth
#########################################################################
# NB. This code only makes sense with the associated tex file...

# Write function to compute second derivative of normal density
d2norm  <- function(x, mu=0, v=1) {
  dnorm(x,mu,sqrt(v))*(((x-mu)/v)^2-1/v)
}

# Second derivative, squared of given Gaussian mixture
myf     <- function(x){
  (0.5*d2norm(x,-1.5,1.5)+0.5*d2norm(x,1,1))^2
}

```

```r
# Compute required integral
k1      <-integrate(myf, -Inf, Inf)$val

# Compute optimal bandwidth
n       <- 1000
k2      <- 1/5
k3      <- 3/5
P       <- 2

h_aimse <- ((1/(2*P*n))*(factorial(P)/k2)^2*(k3/k1))^(1/(1+2*P))


######################################################################
# Q3 (b): monte carlo
######################################################################

# Function for EP kernel
K.ep     <- function(x){
  y <- .75 * (1-x^2) * (abs(x) <= 1)
  return(y)
}

# Function to compute true density value
f.true  <- function(x){
  y<-0.5*dnorm(x,-1.5,sqrt(1.5))+0.5*dnorm(x,1,1)
  return(y)
}

# Create vector of bandwidths
h.list = h_aimse*seq(0.5,1.5,0.1)

# Generate big matrix of random draws from the given Gaussian DGP
N           <- 1000
M           <- 1000

components <- sample(1:2,prob=c(0.5,0.5),size=n,replace=TRUE)
mu.vec      <- c(-1.5,1)
sd.vec      <- sqrt(c(1.5,1))

set.seed(5290)
X.mat       <- replicate(M,rnorm(n=N,mean=mu.vec[components],sd=sd.vec[components]))

# Function for computing LOO imse for a given bandwidth and random sample
imse.lo          <- function(x.rand=randx, h=h_aimse){

  # Compute leave-one-out fhats for each x_i
  y    = sapply(1:N,function(i) 1/(1000*h)*sum(K.ep((as.matrix(x.rand)[-i,]-x.rand[i])/h)))

  # Convert y to data.table for easy manipulation
  y    = as.data.table(y)

  # Add true density values
  y[, y.true := f.true(x.rand)]

  # Compute squared errors
  y[, sq_er.lo := (y - y.true)^2]

  # Compute imse.lo
  imse.lo <- y[, mean(sq_er.lo)]

  output <- imse.lo

  return(output)
}

# Function for computing full-sample imse for a given bandwidth and random sample
imse.li          <- function(x.rand, h=h_aimse){

  # First compute vector of density estimates at each x_i
  y    = sapply(x.rand,function(x) 1/(1000*h)*sum(K.ep((x.rand-x)/h)))

  # Convert y to data.table for easy manipulation
  y    = as.data.table(y)

  # Add true density values
  y[, y.true := f.true(x.rand)]

  # Compute squared errors
  y[, sq_er.li := (y - y.true)^2]

  # Compute imse.li
  imse.li <- y[, mean(sq_er.li)]

  output <- imse.li

  return(output)
}

# RUN SIMULATIONS - TOTAL RUNTIME APPROX 13-15 MINS
 IMSE_LI <- foreach(h=h.list, .combine='cbind') %:%
   foreach(i=1:1000, .combine='c') %do% {
     imse.li(X.mat[,i],h)
```

```r
    }

 IMSE_LO <- foreach(h=h.list , .combine='cbind') %:%
    foreach(i=1:1000, .combine='c') %do% {
      imse.lo(X.mat[,i],h)
    }

# Plot IMSEs
 IMSE_comb <- as.data.frame(cbind(h.list,colMeans(IMSE_LI),colMeans(IMSE_LO)))
 colnames(IMSE_comb) <- c("h", "IMSE_li","IMSE_lo")
 g <- melt(IMSE_comb, id="h")

 ggplot(g) +
    geom_line(aes(x=h, y=value, colour=variable)) +
    scale_colour_manual(values=c("blue","green")) +
    labs(title="Simulated IMSEs for Full and LOO Samples",y="IMSE") +theme(plot.title = element_text(hjust = 0.5))


######################################################################
# Q3 (d): rule-of-thumb bandwidth
######################################################################

# Write function to compute squared second derivative of normal density
d2normsq  <- function(x, mu=0, v=1) {
  (dnorm(x,mu,sqrt(v))*(((x-mu)/v)^2-1/v))^2
}


# Write function to compute ROT bandwidth for random sample
h.rot <- function(x.rand){

  # Compute sample mean and variance
  mu = mean(x.rand)
  v  = var(x.rand)

  # Compute second derivative of normal density
  k1      <-integrate(d2normsq,mu=mu,v=v, -Inf, Inf)$val

  # Compute ROT bandwidth
  h <- ((1/N)*(1/k2)^2*(k3/k1))^(1/5)

}

# Run simulation using foreach
h.rot.vec <- foreach(i=1:1000, .combine='c') %do% h.rot(X.mat[,i])

# Run simulation using sapply - FASTER!
h.rot.vec2<- sapply(1:M,function(i) h.rot(X.mat[,i]))

# Compute mean h.rot.vec
mean(h.rot.vec2)
```

```r
## ECON675: ASSIGNMENT 2
## Erin Markiewitz
## 10/10/2018
######################################################################
# Load packages, clear workspace
######################################################################
rm(list = ls())             #clear workspace
library(foreach)            #for looping
library(dplyr)              #for data manipulation
library(data.table)         #for data manipulation
library(ggplot2)            #for pretty plots
library(boot)               #for bootstrapping
library(Matrix)             #fast matrix calcs
options(scipen = 999)       #forces R to use normal numbers instead of scientific notation


######################################################################
#Q2 (a): generate random data
######################################################################
N   = 1000
M   = 1000

# Generate x's
x.mat   = replicate(M,runif(N,-1,1))

# Generate chi-squared r.v
chi = replicate(M,rchisq(N,5))

# Generate errors
u.mat   = (chi-5)*x.mat

# Generate y's
y.mat   = exp(-0.1*(4*x.mat-1)^2)*sin(5*x.mat) + u.mat

######################################################################
#Q2 (b): cross-validation series estimator
######################################################################

```

```r
# Write function to compute CV errors for K in 1:20
cross.val <- function(i){

  data = data.table(y=y.mat[,i],x=x.mat[,i],const=1)
  temp = rep(NaN, 20)

  for (k in 1:20) {

    data[, temp := x^k]
    setnames(data, "temp", paste0("x_", k))

    X <- as.matrix(data[, c("const",grep("x_", colnames(data), value = TRUE)), with = FALSE])
    Y <- as.matrix(data[,y])

    # Compute projection matrix using QR decomp
    X.Q   <- qr.Q(qr(X))
    XX <- X.Q %*% t(X.Q)
    Y.hat <- XX %*% Y
    W <- diag(XX)

    temp[k] <- mean(((Y-Y.hat) / (1-W))^2)

  }
  return(temp)
}

# RUN SIMULATION -- RUNTIME 5 MINS
results     <- sapply(1:M,function(i) cross.val(i))

# Get average CV errors across simulations
results.avg <- rowMeans(results)

# Get the optimal K
K.hat       <- which.min(results.avg)


# #Plot CV
g <- as.data.frame(cbind(1:20,results.avg))
colnames(g) <- c("K", "CV")


ggplot(g,aes(x=K, y=CV)) +
  geom_line(linetype = "dashed")+
  geom_point()+
  labs(title="Simulated Cross-Validation Errors for M=1000 Simulations") +theme(plot.title = element_text(hjust = 0.5))

########################################################################
#Q2 (c): diagnostics
########################################################################

# Generate grid of x-values for plot
x.grid = seq(-1,1,0.1)

# Write function to compute optimal beta's (i.e. for K=7)
cv.beta <- function(i){

  data = data.table(y=y.mat[,i],x=x.mat[,i],const=1)

  for (k in 1:7) {
    data[, temp := x^k]
    setnames(data, "temp", paste0("x_", k))
  }

  X <- as.matrix(data[, c("const",grep("x_", colnames(data), value = TRUE)), with = FALSE])
  Y <- as.matrix(data[,y])

  beta <- solve(crossprod(X))%*%crossprod(X,Y)

  return(t(beta))

}

# Write function to compute optimal standard errors (i.e for K=7)
cv.se <- function(i){

  data = data.table(y=y.mat[,i],x=x.mat[,i],const=1)

  for (k in 1:7) {
    data[, temp := x^k]
    setnames(data, "temp", paste0("x_", k))
  }

  X <- as.matrix(data[, c("const",grep("x_", colnames(data), value = TRUE)), with = FALSE])
  Y <- as.matrix(data[,y])

  X.Q   <- qr.Q(qr(X))
  XX <- X.Q %*% t(X.Q)
  Y.hat <- XX %*% Y
  W <- diag(XX)

  s <- (1/(N-1))*sum((Y-Y.hat)^2)
```

```
  V <- s*(t(W)%*%W)

  se <- sqrt(V)

  return(se)

}

# Get optimal betas for each m
results.beta <- sapply(1:M, function(i) cv.beta(i))

# Get associated standard errors
results.se <- sapply(1:M, function(i) cv.se(i))

# Compute averages over M
opt.beta  <- rowMeans(results.beta)
opt.se    <- rowMeans(results.se)

# Compute regressors for each number in the x.grid
X.new     <- t(sapply(x.grid, function(x) return(cbind(1,x,x^2,x^3,x^4,x^5,x^6,x^7))))

# Compute y.hats
y.hats    <- as.numeric(X.new%*%as.vector(opt.beta))

# Write the true regression function and compute for x.grid values
f.true <- function(x) exp(-0.1*(4*x-1)^2)*sin(5*x)
y.true <- f.true(x.grid)


# MAKE PLOT

# Get data in right format for ggplot
plot.data = melt(as.data.frame(cbind(x.grid,y.hats,y.true)),id="x.grid")

ggplot(plot.data,aes(x=x.grid,y=value,color=variable))+
  geom_line(linetype = "dashed")+geom_point()+
  labs(title="True and Series Estimate of the Regression function")+
  labs(y=expression(paste(mu(x))),x=expression(paste(x))) +theme(plot.title = element_text(hjust = 0.5))+
  scale_color_manual(values=c("black", "blue"),labels = c(expression(paste(hat(mu))),expression(paste(mu))
      ))

######################################################################
#Q2 (d): derivative of the regression function
######################################################################


# Compute regressors for each number in the x.grid
X.der     <- t(sapply(x.grid, function(x) return(cbind(0,1,2*x,3*x^2,4*x^3,5*x^4,6*x^5,7*x^6))))

# Compute y.hats
dy.hats    <- as.numeric(X.der%*%as.vector(opt.beta))

# Write the true derivative function and compute for x.grid values
df.true <- function(x) exp(-0.1*(4*x-1)^2)*(5*cos(5*x)-0.8*sin(5*x)*(4*x-1))
dy.true <- df.true(x.grid)


# MAKE PLOT
dplot.data = melt(as.data.frame(cbind(x.grid,dy.hats,dy.true)),id="x.grid")

ggplot(dplot.data,aes(x=x.grid,y=value,color=variable))+
  geom_line(linetype = "dashed")+geom_point()+
  labs(title="True and Series Estimate of the Derivative Regression Function")+
  labs(y=expression(paste(mu(x))),x=expression(paste(x))) +theme(plot.title = element_text(hjust = 0.5))+
  scale_color_manual(values=c("black", "blue"),labels = c(expression(paste(d*hat(mu)/dx)),expression(paste
      (d*mu/dx))))
```

```
## ECON675: ASSIGNMENT 2
## Erin Markiewitz
## 10/10/2018
######################################################################
# Load packages, clear workspace
######################################################################
rm(list = ls())             #clear workspace
library(foreach)            #for looping
library(dplyr)              #for data manipulation
library(data.table)         #for data manipulation
library(ggplot2)            #for pretty plots
library(boot)               #for bootstrapping
library(Matrix)             #fast matrix calcs
options(scipen = 999)       #forces R to use normal numbers instead of scientific notation

######################################################################
# Q3 (a): data generation, ploynomial basis
######################################################################
d   = 5
N   = 500
M   = 1000

DGP    = function(n=N){
```

```
  x      = t(as.matrix(replicate(n,runif(d,-1,1))))
  v      = rnorm(n)
  x.norm = sapply(1:n,function(i) t(x[i,])%*%x[i,])
  e      = 0.3637899*(1+x.norm)*v
  g0.x   =exp(x.norm)
  u      = rnorm(n)
  tt     = as.numeric((sqrt(x.norm)+u)>1)
  y      = tt + g0.x + e

  return(list(y=y, x=x, tt=tt))
}

# generate the polynomial basis
gen.P = function(Z,K) {
  if (K==0)   out = NULL;
  if (K==1)   out = poly(Z,degree=1,raw=TRUE);
  if (K==2)  {out = poly(Z,degree=1,raw=TRUE); for (j in 1:ncol(Z)) out = cbind(out,Z[,j]^2);}
  if (K==2.5) out = poly(Z,degree=2,raw=TRUE);
  if (K==3)  {out = poly(Z,degree=2,raw=TRUE); for (j in 1:ncol(Z)) out = cbind(out,Z[,j]^3);}
  if (K==3.5) out = poly(Z,degree=3,raw=TRUE);
  if (K==4)  {out = poly(Z,degree=3,raw=TRUE); for (j in 1:ncol(Z)) out = cbind(out,Z[,j]^4);}
  if (K==4.5) out = poly(Z,degree=4,raw=TRUE);
  if (K==5)  {out = poly(Z,degree=4,raw=TRUE); for (j in 1:ncol(Z)) out = cbind(out,Z[,j]^5);}
  if (K==5.5) out = poly(Z,degree=5,raw=TRUE);
  if (K>=6)  {out = poly(Z,degree=5,raw=TRUE); for (k in 6:K) for (j in 1:ncol(Z)) out = cbind(out,Z[,j]^k);}
  ## RETURN POLYNOMIAL BASIS
  return(out)
}

########################################################################
# Q3 (b): monte carlo simulation
########################################################################
K   <- c(1, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6, 7, 8, 9, 10)
K.r <- c(6, 11, 21, 26, 56, 61, 126, 131, 252, 257, 262, 267, 272, 277)
nK  <- length(K)
theta.hat <- matrix(NaN, ncol=nK, nrow=M)
se.hat    <- theta.hat


for (m in 1:M) {
  data <- DGP(N)
  X    <- data$x
  Y    <- data$y
  TT   <- data$tt

  for (k in 1:nK) {
    X.pol <- cbind(1, gen.P(X, K[k]))
    X.Q   <- qr.Q(qr(X.pol))

    # Compute annihalator matrix
    MP     <- diag(rep(1,N)) - X.Q %*% t(X.Q)

    # Pre-multiplly by MP
    Y.M <- MP %*% Y
    TT.M <- MP %*% TT

    # Get theta.hat using partition regression
    theta.hat[m, k] <- (t(TT.M) %*% Y.M) / (t(TT.M) %*% TT.M)

    # Get standard errors
    Sigma <- diag((as.numeric((Y.M - TT.M*theta.hat[m, k])))^2)
    se.hat[m, k] <- sqrt(t(TT.M) %*% Sigma %*% TT.M) / (t(TT.M) %*% TT.M)
  }
}

# Tabulate results
table <- matrix(NaN, ncol=6, nrow=nK)
for (k in 1:nK) {
  table[k, 1] <- K.r[k]
  table[k, 2] <- mean(theta.hat[, k])                                # point estimate
  table[k, 3] <- mean(theta.hat[, k]) - 1                            # bias
  table[k, 4] <- sd(theta.hat[, k])                                  # standard deviation
  table[k, 5] <- mean(se.hat[, k])                                   # mean standard error
  table[k, 6] <- mean((theta.hat[, k] - 1.96 * se.hat[, k] > 1) |
                      (theta.hat[, k] + 1.96 * se.hat[, k] < 1))     # rejection rate
}
write.table(round(table,3), "PhD_Coursework/econ675/hw2/partial_linear.txt", sep=" & ", eol="\\\\ \n", col.names = FALSE, row.names = FALSE)

########################################################################
# Q3 (c): cross-validation
########################################################################

# cross validation function
K.CV <- function(tt, X, Y) {
  temp <- rep(NaN, nK)
  for (k in 1:nK) {
    X.pol <- cbind(1, tt, gen.P(X, K[k]))
    X.Q   <- qr.Q(qr(X.pol))
    XX <- X.Q %*% t(X.Q)
    Y.hat <- XX %*% Y
```

```
      W <- diag(XX)
      temp[k] <- mean(((Y-Y.hat) / (1-W))^2)
    }
    return(which.min(temp))
}

theta.hat2 <- rep(NaN, M)
se.hat2    <- theta.hat2
K.hat2     <- theta.hat2



for (m in 1:M) {
   data <- DGP(N)
   X     <- data$x
   Y     <- data$y
   tt    <- data$tt

   k.opt <- K.CV(tt, X, Y)
   K.hat2[m]       <- K.r[k.opt]
}

table(K.hat2)
```