Econ 675 Assignment 1

Nathan Mather*

October 12, 2018

Contents

1 Kernal Density Estimation

1.1 Q1 Part 1

Start by noting that

$$\hat{f}^{(s)}(x) = \frac{(-1)^s}{nh^{1+s}} \sum_{i=1}^{n} k^{(s)}\left(\frac{x_i - x}{h}\right)$$

Now taking the expectation of $\hat{f}^{(s)}(x)$ that we can apply the linearity of expectations to move the expectation inside the sum. Then we can use the i.i.d. assumption to show the sum is just n times the expectation. This leaves us with

*Shouts out to Ani, Paul, Tyler, Erin, Caitlin and others for all the help with this

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

Where the second equality is just by the definition of the expectation. Next we use integration by parts. Note that

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

Iterating this s times gives us

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

Next we apply change of variables. let $u = \frac{z-x}{h}$ Note that $du = \frac{1}{h}dz$ so we get

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

Next we Taylor expand $f^{(s)}(x+hu)$ to the P^{th} order about x. Recall from properties of the kernal estimator that $\int_{-\infty}^{\infty} k(u)du = 1$ and that $\int_{-\infty}^{\infty} k(u)u^j du = 0$ for all $j \neq p$ This gives us

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

which is the desired result.

Now for the variance recall again that

$$\hat{f}^{(s)}(x) = \frac{(-1)^s}{nh^{1+s}} \sum_{i=1}^{n} k^{(s)}\left(\frac{x_i - x}{h}\right)$$

So by the i.i.d. assumption we can get that

$$\mathrm{V}\left(\hat{f}^{(s)}(x)\right) = \frac{1}{nh^{2+2s}} \mathrm{V}\left(k^{(s)}\left(\frac{x_i - x}{h}\right)\right)$$

$$\begin{aligned}
\mathrm{V}\left(\hat{f}^{(s)}(x)\right) &= \frac{1}{nh^{2+2s}} \mathrm{V}\left(k^{(s)}\left(\frac{x_i - x}{h}\right)\right) && (1)\\
&= \frac{1}{n2h^{2+2s}} \mathrm{E}\left[\left(k^{(s)}\left(\frac{x_i - x}{h}\right)\right)^2\right] - \frac{1}{nh^{2+2s}} \mathrm{E}\left[\left(k^{(s)}\left(\frac{x_i - x}{h}\right)\right)^2\right]^2 && (2)\\
&= \frac{1}{nh^{2+2s}} \mathrm{E}\left[\left(k^{(s)}\left(\frac{x_i - x}{h}\right)\right)^2\right] - \frac{1}{n}\left(\frac{1}{h^{1+s}} \mathrm{E}\left[\left(k^{(s)}\left(\frac{x_i - x}{h}\right)\right)^2\right]\right)^2 && (3)\\
&= \frac{1}{nh^{2+2s}} \int_{-\infty}^{\infty} k^{(s)}\left(\frac{x_i - x}{h}\right)^2 f(z)dz + \frac{1}{nh^{2+2s}} f^{(n)}(X)^2 && (4)\\
&= \frac{1}{nh^{1+2s}} \int_{-\infty}^{\infty} k^{(s)}(u)^2 f(x+hu)du + o\left(\frac{1}{nh^{2+2s}}\right) && (5)\\
&= \frac{1}{nh^{1+2s}} \cdot \vartheta_s(K) + o\left(\frac{1}{nh^{2+2s}}\right) && (6)
\end{aligned}$$

1.2 Q1 part 2

We start with the following AMISE

$$AIMSE[h] = \int \left[\left(h_n^P \cdot \mu_P(K) \cdot \frac{f^{(P+s)}(x)}{P!} \right)^2 + \frac{1}{nh_n^{1+2s}} \cdot \vartheta_s(K) \cdot f(x) \right] dx$$

Using the ϑ notation so $\vartheta_{P+s}(f) = \int (f^{(P+s)}(x))^2$ and recalling that f(x) integrates to 1 we can rewrite this as

$$= h_n^{2P} \left(\frac{\mu_P(K)}{P!} \right)^2 \vartheta_{P+s}(f) + \frac{\vartheta_s(K)}{nh_n^{1+2s}}$$

Now taking first order conditions and solving for h

$$\frac{d}{dh} AIMSE[h] = 2Ph_n^{2p-1} \left(\frac{\mu_P(K)}{P!} \right)^2 \vartheta_{P+s}(f) - (1+2s) \frac{\vartheta_s(K)}{nh_n^{2+2s}} = 0$$

$$\implies 2Ph^{1+2P+2s} \left(\frac{\mu_P(K)}{P!} \right)^2 \vartheta_{P+s}(f) = (1+2s) \frac{\vartheta_s(K)}{n}$$

Thus, we get the AIMSE-optimal bandwidth choice.

$$h_{AIMSE_s} = \left[\frac{(2s+1)(P!)^2}{2P} \frac{\vartheta_s(K)}{\vartheta_{s+P}(f) \cdot \mu_P(K)^2} \frac{1}{n} \right]^{\frac{1}{1+2P+2s}}$$

Least squares cross-validation is a fully automatic data-driven method of selecting the smoothing parameter h. THis is based on the principle of selecting bandwidth that minimizes the integrated squared error of the resulting estimate.The estimate used is

$$\hat{h}_{CV} = \arg\min_h \frac{1}{n^2 h} \sum_{i=1}^{n} \sum_{j=1}^{n} \bar{k} \left(\frac{X_i - X_j}{h} \right) - \frac{2}{n(n-1)h} \sum_{i=1}^{n} \sum_{j=1, i \neq j}^{n} k \left(\frac{X_i - X_j}{h} \right)$$

1.3 Monte Carlo experiment

1.3.1 Q1 Part 3 a

First, we want to compute the theoretically optimal bandwidth for $s = 0$, $n = 1000$, using the Epanechnikov kernel ($P = 2$), with the following Gaussian DGP:

$$x_i \sim 0.5\mathcal{N}(-1.5, -1.5) + 0.5\mathcal{N}(1, 1)$$

Filling in what we know so far we have :

$$h_{AIMSE_s} = \left[\frac{\vartheta_0(K)}{\vartheta_2(f) \cdot \mu_2(K)^2} \frac{1}{1000} \right]^{\frac{1}{5}}$$

So we need the second moment of K and the first moment of the second derivative of k squared. We can get two of these values from the table in Bruce Hanson's nonparametric notes. Giving us.

$$h_{AIMSE_s} = \left[\frac{\frac{3}{5}}{\vartheta_2(f) \cdot \frac{1}{5}^2} \frac{1}{1000} \right]^{\frac{1}{5}}$$

The second derivative of the normal density φ with mean μ variance σ^2 is

$$\varphi''_{\mu,\sigma^2}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \left[\left(\frac{(x-\mu)}{\sigma^2} \right)^2 - \frac{1}{\sigma^2} \right]$$

now useing the linearity of integrals we can find $\vartheta_2(f)$

$$\vartheta_2(f) = \int_{-\infty}^{\infty} [0.5\varphi''_{1,1}(x) + 0.5\varphi''_{-1.5,1.5}(x)]^2 dx \approx 0.03883397$$

Where the approximation comes from R
Finally, pluging this in gives the theoretically optimal bandwidth is:

$$h* = 0.8267532$$

1.3.2 Q1 Part 3 b

Below Is the table of $\widehat{IMSE}^{lI}$ results and $\widehat{IMSE}^{LO}$ results by bandwidth h. My stata code was significantly slower and so I only ran 10 repetitions. Even with that the plots give us generally the same idea.

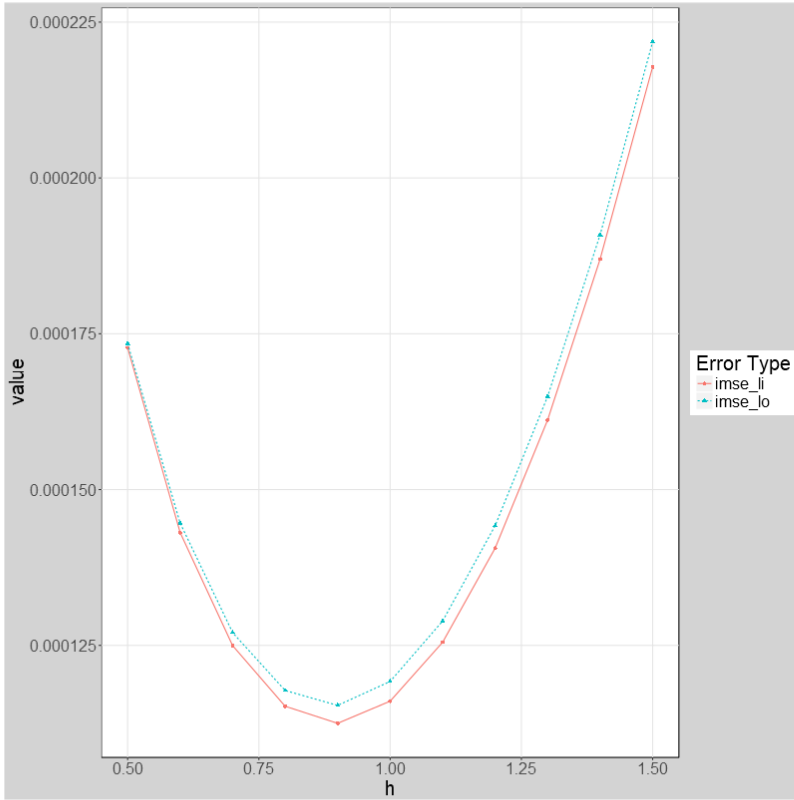
R TABLE

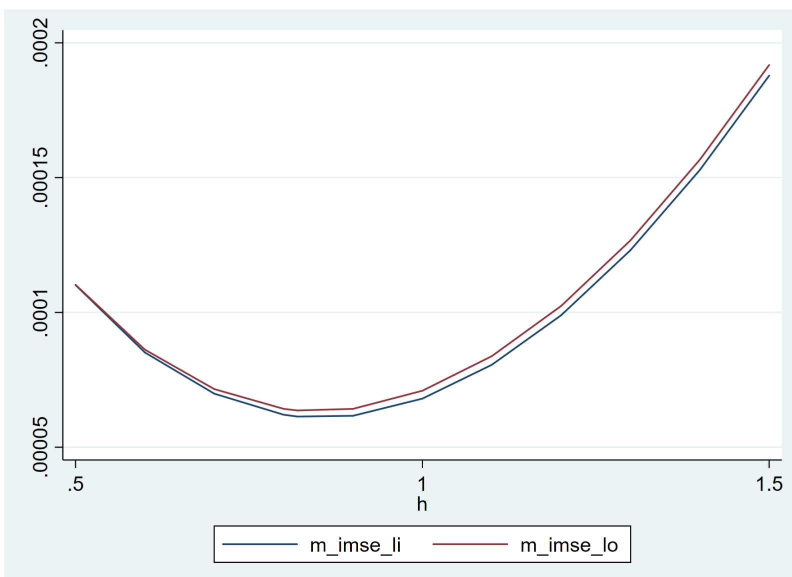
h	imse_li	imse_lo	d_h_hat
0.5	0.000173	0.000173	0.988
0.6	0.000143	0.000145	0.988
0.7	0.000125	0.000127	0.988
0.8	0.000115	0.000118	0.988
0.9	0.000112	0.000115	0.988
1	0.000116	0.000119	0.988
1.1	0.000126	0.000129	0.988
1.2	0.000141	0.000144	0.988
1.3	0.000161	0.000165	0.988
1.4	0.000187	0.000191	0.988
1.5	0.000218	0.000222	0.988

STATA TABLE

h	m_imse_li	m_imse_lo
0.500	0.000110	0.000110
0.600	8.51e-05	8.62e-05
0.700	6.99e-05	7.16e-05
0.800	6.21e-05	6.42e-05
0.820	6.14e-05	6.36e-05
0.900	6.17e-05	6.42e-05
1	6.80e-05	7.09e-05
1.100	8.06e-05	8.38e-05
1.200	9.89e-05	0.000102
1.300	0.000123	0.000127
1.400	0.000153	0.000157
1.500	0.000188	0.000192

My graphs from both programs are below





1.3.3 Q1 Part 3 c

Intuitively the difference between the two estimators, LI and LO, is that the LI includes the extra zero term in the sum since we include $x_i - x_i$. As the size of the sample increases this contribution to the overal average will go to zero. Meaning that the LI IMSE will also converge to the correct estimate. s

1.3.4 Q1 Part 3 d

The "d_h_hat" column of the graph in part c is my calculation of this over the 1000 iterations. The value it came up with was 1.04. This is somewhat close but, as expected, not exactly correct.

2 Question 2: Linear Smoothers, Cross-validation and Series

2.1 Q2 Part 1

For local polynomial regression we want to estimate $e(x) = \mathrm{E}[y_i|x_i = x]$. The idea of a local polynomial regression is to estimate e(x) around the point x using a polynomial of degree p. We estimate this polynomial with weighted least squares. For a given x, we want to solve.

$$\hat{\boldsymbol{\beta}}_{LP}(x) = \arg\min_{\beta\in\mathbb{R}^{p+1}} \sum_{i=1}^{n} [y_i - \boldsymbol{r}_p(x_i - x)'\boldsymbol{\beta}]^2 K(\frac{x_i - x}{h})$$

where $\boldsymbol{r}_p(x) = (1, x, x^2, ..., x^p)'$ and $\hat{e}(x) = \hat{\beta}_0$
from the lecture notes we can get that

$$\hat{\boldsymbol{\beta}}_{LP}(x) = (\boldsymbol{R_p'}\boldsymbol{W}\boldsymbol{R_p'})^{-1}\boldsymbol{R_p'}\boldsymbol{W}\boldsymbol{y}$$

I think This simplifies to the following

$$\hat{e}(x) = e_1' \left(\sum_{i=1}^{n} \boldsymbol{r}_p(x_i - x)\boldsymbol{r}_p(x_i - x)' w_i\right)^{-1} \left(\sum_{i=1}^{n} \boldsymbol{r}_p(x_i - x) w_i y_i\right)$$

where $wi = K(\frac{x_i - x}{h})$
Now for the series estimation.

$$\hat{\boldsymbol{\beta}}_{\boldsymbol{series}} = \arg\min_{\beta\in\mathbb{R}^{p+1}} \sum_{i=1}^{n} (y_i - \boldsymbol{r}_p(x_i)'\boldsymbol{\beta})^2$$

where $\boldsymbol{r}_p(x_i) = (1, x_x, x_i^2, ..., x_i^p)$ and

$$\hat{e}(x) = \boldsymbol{r}_p(x)'\hat{\boldsymbol{B}}_{series}$$

Together we get

$$\hat{\boldsymbol{B}}_{series} = (\boldsymbol{R}`_p\boldsymbol{R}_p)^{-1}\boldsymbol{R}_p\boldsymbol{y}$$

so

$$\hat{e}(x) = \boldsymbol{r}_p(x)'(\boldsymbol{R}`_p\boldsymbol{R}_p)^{-1}\boldsymbol{R}_p\boldsymbol{y}$$

Writing this in linear summation form I believe we get

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

2.2 Q2 Part 2

We want to choose the tuning parameter to minimize the mean squared leave one out error which is

$$\hat{c} = \arg\min_c \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{e}_(i)(x_i; c))^2$$

where $\hat{e}_{(i)}(x_i)$ is the estimator of the regression function that leaves out x_i
We can write the local polynomial series estimator as

$$\hat{\boldsymbol{e}}(x) = \boldsymbol{S}\boldsymbol{y}$$

Where S is the smoothing matrix. Note that the rows of S sum to one so $\boldsymbol{S1} = \boldsymbol{1}$. For the leave one out estimator we want to use $\boldsymbol{S}$ but with the i_{th} row and column removed. If we let the elements of $\boldsymbol{s}$ be denoted by w_{ij} than deleting the i_{th} column means that the i_{th} row will now sum to $1 - w_{ij}$. So, we divide by $1 - w_{ij}$ to renormalize and get the the leave-one-out estimator is

$$\hat{e}_{(i)}(x_i) = \frac{1}{1 - w_{ij}} \sum_{j=1 j\neq i}^{n} w_{ij}y_i$$

The full sample estimator is

$$\hat{e}(x_i) = \sum_{j=1}^{n} w_{ij}y_i$$

Together we can get that

$$\hat{e}_{(i)}(x_i)(1 - w_{ij}) = \sum_{j=1 j\neq i}^{n} w_{ij}y_i$$

$$\implies \hat{e}_{(i)}(x_i) = \sum_{j=1 j\neq i}^{n} w_{ij}y_i + w_{ij}\hat{e}_{(i)}(x_i) = \sum_{j=1}^{n} w_{ij}y_i + w_{ij}\hat{e}_{(i)}(x_i) - w_{ij}y_i = \hat{e}_{(i)}(x_i) + w_{ij}\hat{e}_{(i)}(x_i) - w_{ij}y_i$$

$$\begin{aligned} \implies y - \hat{e}_{(i)}(x_i) &= y - \hat{e}_{(i)}(x_i) - w_{ij}\hat{e}_{(i)}(x_i) + w_{ij}y_i \\ &= y - \hat{e}_{(i)}(x_i) + w_{ij}(y_i - \hat{e}_{(i)})) \end{aligned}$$

$$\implies y_i - \hat{e}_{(i)}(x_i) = \frac{1}{1 - w_{ij}}(y_i - \hat{e}(x_i))$$

Which is what we wanted

2.3 Q2 part 3

Note that we have iid data and the $\sum_{i=1}^{n} w_{n,i}(x_i) = 1$ first we want to find

$$\mathrm{E}[\hat{e}(x)|\boldsymbol{x}] = \mathrm{E}\left[\sum_{i=1}^{n} w_{n,i}(x_i)y_i|\boldsymbol{x}\right] = \sum_{i=1}^{n} \mathrm{E}\left[w_{n,i}(x_i)y_i|\boldsymbol{x}\right] = \sum_{i=1}^{n} w_{n,i}(x_i)\mathrm{E}\left[y_i|\boldsymbol{x}\right] = \mathrm{E}[y_i|\boldsymbol{x}]$$

Now as long as we have a bounded second moment we can use CLT to get asymptotic normality. Now to calculate the variance:

$$\mathrm{V}[\hat{e}(x)|\boldsymbol{x}] = \mathrm{V}\left[\sum_{i=1}^{n} w_{n,i}(x_i)y_i|\boldsymbol{x}\right] = \sum_{i=1}^{n} \mathrm{V}\left[w_{n,i}(x_i)y_i|\boldsymbol{x}\right] = \sum_{i=1}^{n} w_{n,i}(x_i)^2 \mathrm{V}\left[y_i|\boldsymbol{x}\right]$$

Then if we assume homoroskedasticity we get the estimator

$$\hat{\mathrm{V}}(x) = \hat{\sigma}^2 \sum_{i=1}^{n} w_{n,i}(x)^2$$

Where $\hat{\sigma}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(y_i - \hat{e}(x_i))^2$

2.4 Q2 part 4

The pointwise asymptotically valid 95% convidence interval for $e(x)$ is

$$CI_{95}(x) = [\hat{e}(x) - 1.96\sqrt{\hat{\mathrm{V}}(x)}, \hat{e}(x) + 1.96\sqrt{\hat{\mathrm{V}}(x)}]$$

This is just a confidence interval for a given point. applying this to a grid of points across the line and interpreting that as a band for the function is incorrect. For uniformly valid inference we need that the estimate is less that the cutoff for all values of x, not just one specific x.

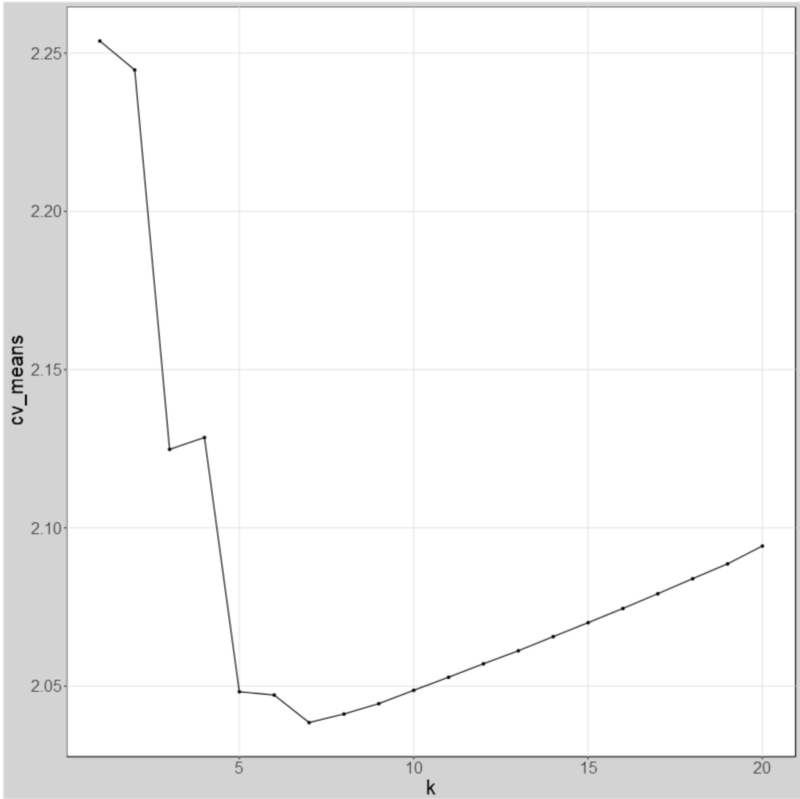
2.5 Q2 part 5

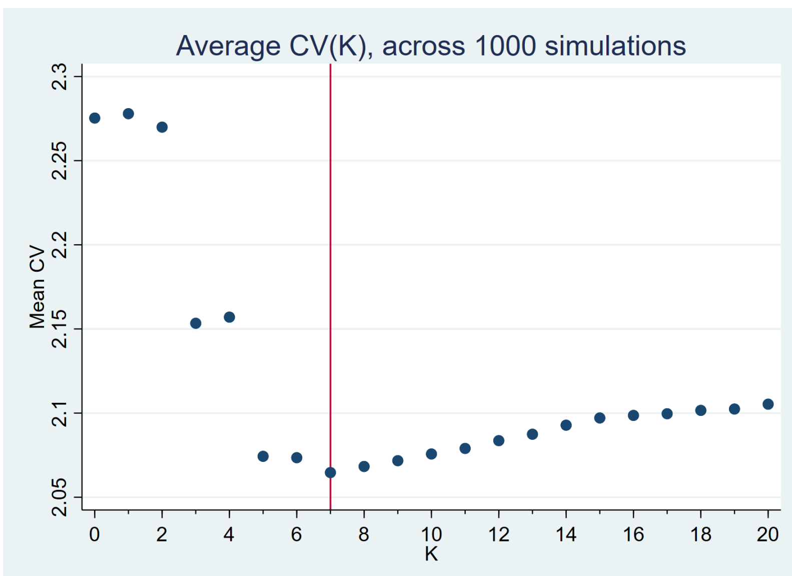
2.5.1 Q2 part 5 a

See the code in appendix

2.5.2 Q2 part 5 B

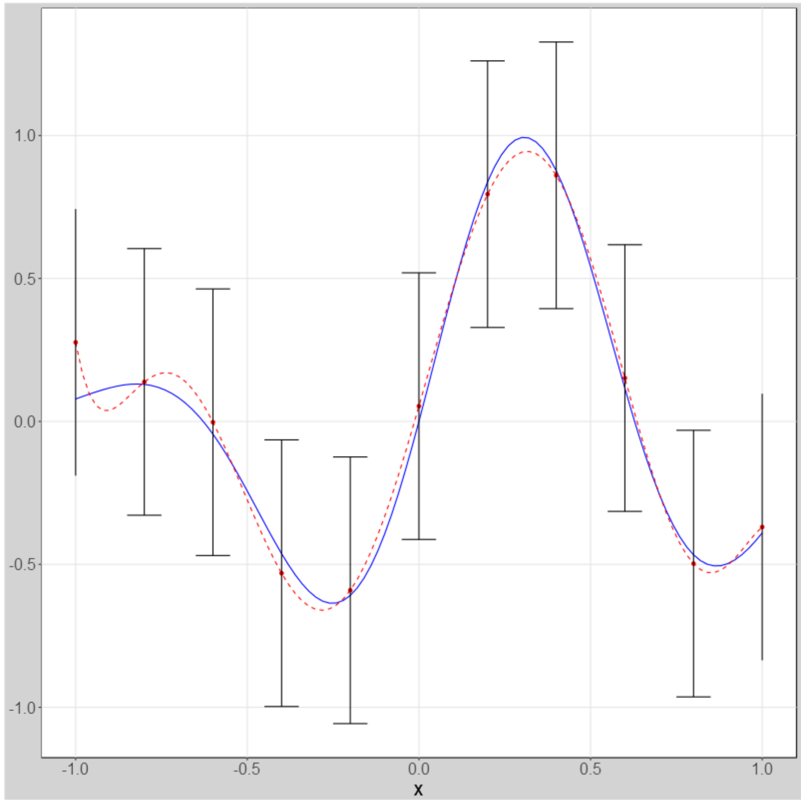
The plot of the CV(K) simulations is below

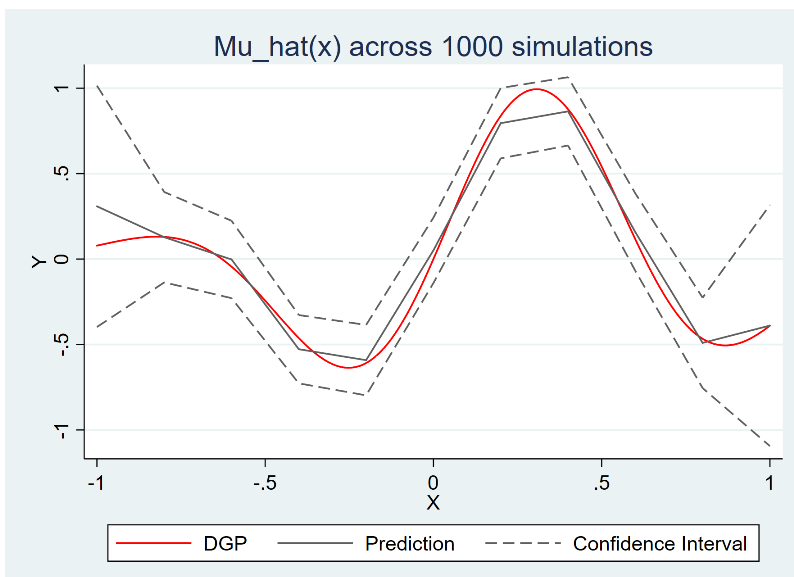




2.5.3 Q2 part 5 C

My plot is below. I used homoscedastic standard errors. The dotted line is my estimate





2.5.4 Q2 part 5 D

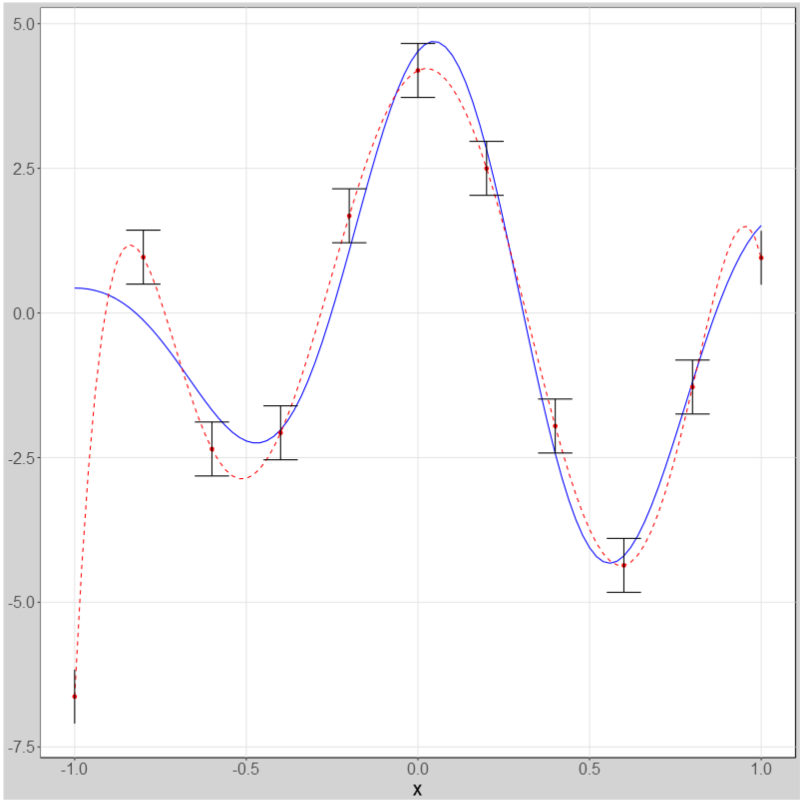
Calculating the derivative of $u(x)$ we get

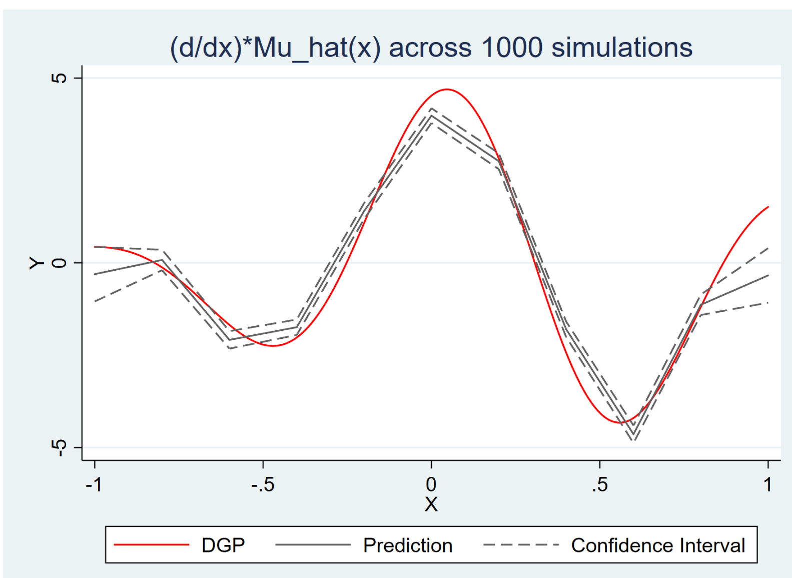
$$e^{(0.1\cdot(4x-1)^2)}[5\cdot cos(5x) - 0.8\cdot(4x-1)sin(5x)]$$

Taking the derivative of the estimated euqation we get

$$\hat{u}'(x) = \beta_1 + 2\beta_2 x + 3\beta_3 x^2 + 4beta_4 x^3 + 5\beta_5 x^4 + 6\beta_6 x^5 + 7\beta_7 x^6$$

I plot the corresponding curves below. The dotted line is my estimate





3 Question 3: Semiparametric Semi-Linear Model

3.1 Q3 part 1

first note that θ_0 cannot contain a constant. since $\alpha + g(z) = [\alpha + c] + [g(z) - c] \equiv \alpha_{new} + g_{new}(z)$ the sum of the new g and intercept are observationally equivalent to the old ones so they cannot be identified. From Li and Racine page 223 we can see that the requirements for identifiability are that $\mathrm{E}[(t_i - h_0(x_i))(t_i - h_0(x_i))']$ is positive definite. 3

To prove the moment condition let's start with the expectation of interest and apply the law of iterated expectations.

$$\mathrm{E}[(t_i - h_o(x_i))(y_i - t_i\theta_0)] = \mathrm{E}[\mathrm{E}[(t_i - h_o(x_i))(y_i - t_i\theta_0)|x_i]] = \mathrm{E}[\mathrm{E}[(t_i - h_o(x_i))(g_0(x_i) + \epsilon_i)|x_i]]$$

$$= \mathrm{E}[\mathrm{E}[(t_i - h_o(x_i))g_o(x_i)|x_i]] + \mathrm{E}[\mathrm{E}[(t_i - h_o(x_i))\epsilon_i|x_i]]$$
$$= \mathrm{E}[g_o(x_i)\mathrm{E}[(t_i - h_o(x_i))|x_i]] + \mathrm{E}[(t_i - h_o(x_i))\mathrm{E}[\epsilon_i|x_i, i_i]] = 0$$

Now to find a closed form solution for θ_0.

$$\mathrm{E}[(t_i - h_o(x_i)y_i)] - \mathrm{E}[(t_i - h_o(x_i)t_i)]\theta_0 = 0$$
$$\implies \theta_0 = \frac{\mathrm{E}[(t_i - h_o(x_i)y_i)]}{\mathrm{E}[(t_i - h_o(x_i)t_i)]}$$

The IV interpretation can be given as follows. Let $yi = t_i\theta_0 + g_0(x_i) + \epsilon_i = t_i\theta_0 + \mu_i$. Now μ_i is uncorrelated with t_i so we can define and instrument $z_i = t_i - h_0(x_i)$ which has the property of $\mathrm{E}[z_i\mu_i] = 0$ and $\mathrm{E}[t_i z_i] \neq 0$ so it is a valid instrument.

3.2 Q3 part 2

3.2.1 Q3 part 2 a

As the question asks we will consider the power series approximation.

$$\mathrm{E}[y_i|\boldsymbol{x}_i] \approx t_i\theta_0 + \boldsymbol{p}^K(\boldsymbol{x}_i)'\gamma_k$$

Next, as the question instructs, we can use the partition regression formula and get the OLS estimator

$$\hat{\theta}(K) = (\boldsymbol{t'M_x t})^{-1}\boldsymbol{t'M_x Y}$$

Where here $\boldsymbol{t} = (t_1, ..., t_n)'$ and $\boldsymbol{M}_p = \boldsymbol{I} - \boldsymbol{P_{r_p(x)}}$

and $\boldsymbol{P_{r_p(x)}} = \boldsymbol{R_p(R_p'R_p)^{(-1)}R_p'}$

$$\boldsymbol{R}_p = \begin{bmatrix} 1 & (\boldsymbol{x}_1) & (\boldsymbol{x}_1)^2 & \dots & (\boldsymbol{x}_1)^p \\ 1 & (\boldsymbol{x}_2) & (\boldsymbol{x}_2)^2 & \dots & (\boldsymbol{x}_2)^p \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & (\boldsymbol{x}_n) & (\boldsymbol{x}_n)^2 & \dots & (\boldsymbol{x}_n)^p \end{bmatrix}$$

3.2.2 Q3 part 2 b

We used the moment condition when discussing the IV estimate interpretation in part 1 to find

$$\theta_0 = \frac{\mathrm{E}[(t_i - h_o(x_i)y_i)]}{\mathrm{E}[(t_i - h_o(x_i)t_i)]}$$

So we can estimate this with

$$\left(\frac{1}{n}\sum_{i=1}^{n} t_i - h_o(x_i)t_i\right)^{-1}\left(\frac{1}{n}\sum_{i=1}^{n} t_i - h_o(x_i)y_i\right)$$

3.3 Q3 part 3

3.3.1 Q3 part 3 a

WE can just use the partialing out method above to get

$$\hat{\theta}(K) = (\boldsymbol{t'M_pt})^{-1}\boldsymbol{t'M_p}(\boldsymbol{t}\theta_0 + \boldsymbol{R_p\gamma_k} + \boldsymbol{e}) = \theta + (\boldsymbol{t'M_pt})^{-1}\boldsymbol{t'M_pe}$$

Now with iid data and conditional l heteroskedasticity, we can use the WLLN and CLT as usual to get normality and the sandwich form variance matrix

3.3.2 Q3 part 3 b

The asymptotically valid 95% confidence interval is just the same as usual then

$$CI_{95} = [\hat{\theta}(K) - 1.96\sqrt{V_{HCO}}, \hat{\theta}(K) + 1.96\sqrt{V_{HCO}}]$$

3.4 Q3 part 4

3.4.1 Q3 part 4 a

see code in appendix

3.4.2 Q3 part 4 b

My results from R are in the table below

K	Theta	Bias	S.D	V_HCO	Rejection rate
6.000	1.841	0.467	3.607	0.462	0.980
11.000	-0.167	0.231	0.082	0.218	0.093
21.000	-0.161	0.232	0.080	0.214	0.093
26.000	-0.162	0.235	0.081	0.214	0.102
56.000	-0.140	0.227	0.071	0.199	0.128
61.000	-0.124	0.221	0.064	0.193	0.122
126.000	0.008	0.113	0.013	0.100	0.094
131.000	0.008	0.114	0.013	0.100	0.093
252.000	0.008	0.128	0.017	0.094	0.149
257.000	0.008	0.130	0.017	0.094	0.153
262.000	0.008	0.132	0.017	0.094	0.154
267.000	0.009	0.132	0.018	0.094	0.162
272.000	0.009	0.133	0.018	0.094	0.162
277.000	0.008	0.135	0.018	0.094	0.170

theta_hat	se_hat	bias	cov	svar	K
3.002	0.114	2.002	0	0.688	1
0.734	0.140	-0.266	0.00500	0.249	2
0.734	0.140	-0.266	0.00500	0.249	3
0.766	0.139	-0.234	0.00600	0.223	4
0.766	0.139	-0.234	0.00600	0.223	5
0.796	0.139	-0.204	0.00500	0.256	6
0.796	0.139	-0.204	0.00500	0.256	7
0.790	0.139	-0.210	0.00600	0.259	8
0.790	0.139	-0.210	0.00600	0.259	9
0.793	0.139	-0.207	0.00500	0.250	10
0.791	0.139	-0.209	0.00600	0.230	11
0.779	0.139	-0.221	0.00600	0.242	12
0.771	0.139	-0.229	0.00700	0.227	13
0.795	0.138	-0.205	0.00700	0.220	14

3.4.3 Q3 part 4 c

Using cross-validation, I get $\hat{K}_{cv} = 126$. We can see from Table 1, across the simulations, $\hat{K}_{cv}$ gives a low rejection rate, but other estimators have lower bias and variance.

4 Appendix

4.1 R Code

pset 2 Labor

```r
#=======================================#


   #===========================#
# ==== Metrics 675 ps 2 ====
#===========================#




#=======================================#
# ==== load packages and clear data ====
#=======================================#

library(data.table)
library(doParallel)
library(foreach)
library(ggplot2)
library(Matrix)

# clear data and consol
rm(list = ls(pos = ".GlobalEnv"), pos = ".GlobalEnv")
options(scipen = 999)
cat("\f")

# set options
opt_test_run <- TRUE

# set attributes for plot to default ea theme
plot_attributes <- theme( plot.background = element_rect(fill = "lightgrey"),
                          panel.grid.major.x = element_line(color = "gray90"),
                          panel.grid.minor  = element_blank(),
                          panel.background = element_rect(fill = "white", colour = "black") ,
                          panel.grid.major.y = element_line(color = "gray90"),
                          text = element_text(size= 20),
                          plot.title = element_text(vjust=0, colour = "#0B6357",face = "bold", size = 4

#================================================#
# ==== Question 1: Kernel Density Estimation ====
#================================================#

#=================#
# ==== Part a ====
#=================#


# now to find the theoretically optimal H I need to calculate integral of second derivative.
# second dericative of normal function is
phi_2 <- function(x, mean, v){

  dnorm(x=x,mean=mean,sd=sqrt(v))*(((x - mean)/v)^2-(1/v))
```

```
}

# now create the function to integrate
f_int <- function(x){

  f_out <- (.5*phi_2(x=x, -1.5,1.5) + .5*phi_2(x=x, 1,1))^2
  return(f_out)
}

# and the integral is
v2k <- integrate(f_int, lower = -Inf, upper = Inf)$val

# so optimal bandwith is
h_opt <- (15/(v2k*1000))^(1/5)

#=================#
# ==== part b/d ====
#=================#

# set parms
n     <- 1000
M <- ifelse(opt_test_run, 10, 1000)


# kernal function
K0 <- function(u){

  out <- .75 * (1-u^2) * (abs(u) <= 1)
  return(out)
}

# define the true f(x) function
f_x <- function(x){

  .5*dnorm(x, -1.5,sqrt(1.5))+.5*dnorm(x,1,1)
}


#============================#
# ==== Make imse function ====
#============================#

# define variables for debug
# in_data <- r_dt
# x_v <- "rdraw"

# generate data for debugging functions
# start data.table for random data, take a random draw for weighted normals
# r_dt <- data.table( r1 = sample(1:2,prob=c(.5,.5),size=n,replace=T) )
#
# # draw a random number from appropriate normal dist according to r1
# r_dt[r1 == 1, rdraw := rnorm(.N,-1.5,1.5)]
# r_dt[r1 == 2, rdraw := rnorm(.N,1,1)]
```

```
# r_dt[, r1 := NULL]
# in_data <- r_dt
# h_v <- c(.5,.6)
# x_v <- "rdraw"
# i <- 1

imse_f <- function(in_data, x_v, h_v = NULL, f_x = f_x){

  # copy the data to aviod editing it in global enviorment
  data <- copy(in_data)

  # add a constant for the merge
  in_data[, const := 1]

  # cartesian merge to get all pairs
  paired_dt <- merge(in_data, in_data, by = "const", allow.cartesian = TRUE)

  # get new variable names after the merge. This kind of annoyingly general for a HW assingment. I regr
  x_vx <- paste0(x_v, ".x")
  x_vxi <- paste0(x_v, ".y")

  # initialize a list for output from each h
  ouput_list <- vector("list", length= length(h_v))

  # now do the imse calculations for each h in h_v
  for(i in 1:length(h_v)){

    h <- h_v[[i]]

    # get the kernal thing for each pair
    paired_dt[, k_x := K0((get(x_vxi) - get(x_vx))/h)]

    # now mean the kernal by rdraw.x and devide by h
    f_hats <- paired_dt[, list(f_hat_x = mean(k_x)/h), by = x_vx]

    # now get the f_hats for the leave one out by deleating the observation where x= xi. This will be r
    # 1, M+2, 2M+3, 3M+4 ... so eq(1, M*M, M+1) should take care of those
    paired_dt_lo <- paired_dt[-c(seq(1, n*n, n+1)), ]

    # now get the mean of the f_hats leacing out the x
    f_hats_lo <- paired_dt_lo[, list(f_hat_x = mean(k_x)/h), by = x_vx]

    # now add in f_x for each
    f_hats[, f_x := f_x(get(x_vx))]
    f_hats_lo[, f_x := f_x(get(x_vx))]

    # now do squared error
    f_hats[, sq_er := (f_hat_x - f_x)^2]
    f_hats_lo[, sq_er := (f_hat_x - f_x)^2]

    # now get imse
    imse_li <- f_hats[, mean(sq_er)]
    imse_lo <- f_hats_lo[, mean(sq_er)]
```

```r
    # now put into a data.table and put in list
    ouput_list[[i]] <- data.table(imse_li = imse_li, imse_lo= imse_lo, h = h)
  }

  output <- rbindlist(ouput_list)

  return(output[])

}


#==========================#
# ==== run simulations ====
#==========================#

# note: pretty sure it would be faster yet to just include the simulations in the by group of the data
# operations in the IMSE function. Probably marginally faster but kind of hard to wrap my head around.
# update: I tried this an it exceeded R's vector length limit. Might be a workaround, unsure.

# define squared phi_2 function for part d
phi_2_sq <- function(x , mean, v){

  phi_2(x =x , mean = mean, v = v)^2
}


# now set up function to run simulations, make sure to pass in user defined functons/vars or foreach ca
sim_function <- function(i, n, f_x, phi_2, h_v){

  # generate data
  # start data.table for random data, take a random draw for weighted normals
  r_dt <- data.table( r1 = sample(1:2,prob=c(.5,.5),size=n,replace=T) )

  # draw a random number from appropriate normal dist according to r1
  r_dt[r1 == 1, rdraw := rnorm(.N,-1.5,1.5)]
  r_dt[r1 == 2, rdraw := rnorm(.N,1,1)]
  r_dt[, r1 := NULL]


  # get IMSE
  results_i <- imse_f(in_data = r_dt, x_v = "rdraw" ,f_x = f_x ,h_v =h_v)
  results_i[, sim := i]

  # now get mean and SE or part d
  mean_i <- r_dt[, mean(rdraw)]
  var_i <- r_dt[, var(rdraw)]

  # calculate "optimal bandwidth" under the procdure from part D
  vok <- 3/5
  u2k2 <- (1/5)^2

  # and the integral is
  v2phi <- integrate(phi_2_sq, mean = mean_i, v = var_i, lower = -Inf, upper = Inf)$val
```

```
  # now calculate h optimal
  h_opt <- (vok/ (u2k2 *v2phi*n))^(1/5)

  # put that bad boy in the table
  results_i[, d_h_hat := h_opt]

  # return the rsults for all of q2
  return(results_i[])

}


# make a vector of h's
h_v <- seq(.5, 1.5,.1)

# lets time this sucker
start_t <- Sys.time()


# parallel setup
cl <- makeCluster(4, type = "PSOCK")
registerDoParallel(cl)

# run simulations in parallel
output_list <- foreach(sim = 1 : M,
                       .inorder = FALSE,
                       .packages = "data.table",
                       .options.multicore = list(preschedule = FALSE, cleanup = 9)) %dopar% sim_functio

# stop clusters
stopCluster(cl)


# AND TIME
run_time1 <- Sys.time() - start_t

  # bind list
  output_dt <- rbindlist(output_list)

  # now take the mean of imse
  part_b_res <- output_dt[, list(imse_li = mean(imse_li), imse_lo = mean(imse_lo), d_h_hat = mean(d_h_ha

  # make them pretty
  part_b_res_pretty <- signif(part_b_res, 3)
  part_b_res_pretty[, colnames(part_b_res_pretty)] <- lapply(part_b_res_pretty[,colnames(part_b_res_pre

  # make the graph
  # melt the data to work better with ggplot
  part_b_res[, d_h_hat := NULL ]
  plot_data <-  melt.data.table(part_b_res, id.vars = "h", variable.name = "Error Type")
  plot_1_3_b <- ggplot(data = plot_data, aes(x = h, y = value, color = `Error Type`, shape = `Error Typ

  plot_1_3_b <- plot_1_3_b + geom_point() + geom_line() + plot_attributes
```

```
  plot_1_3_b




#====================#
# ==== save data ====
#====================#

  # only save data if this isn't a test run
  if(!opt_test_run){
    # save IMSE by h results
    print(xtable(part_b_res_pretty, type = "latex",
                 digits = 3),
          file = "C:/Users/Nmath_000/Documents/Code/courses/econ 675/PS_2_tex/Q1_p3_b.tex",
          include.rownames = FALSE,
          floating = FALSE)

    # save the plot
    png("c:/Users/Nmath_000/Documents/Code/courses/econ 675/PS_2_tex/plot_1_3_b.png", height = 800, wid
    print(plot_1_3_b)
    dev.off()

  }



  #=====================#
  # ==== Question 2 ====
  #=====================#


  #===========================#
  # ==== A: generate data ====
  #===========================#

    gen_data_2.5.a <- function(){

      # start data.table with random x's. get a chi squared too cause i need that for the epsilon
      r_dt <- data.table(x = runif(n,-1,1), chi_sq = rchisq(n,5))

      # create a noise clumn epsilon,
      r_dt[, eps := x^2*(chi_sq-5)]

      # now calcualte y
      r_dt[, y := exp(-0.1*(4*x-1)^2)*sin(5*x) + eps]

      # drop the chi_sq column
      r_dt[, chi_sq := NULL]

      # return the random data
      return(r_dt[])
```

```
  }


#==========================#
# ==== B do experiment ====
#==========================#

# generate some random data
r_dt <- gen_data_2.5.a()


# write a function to apply accross simulations
power_s_fun <- function(sim = NULL){

  r_dt <- gen_data_2.5.a()
  r_dt[, const :=1]

  # store results in a list
  results <- vector("list", length = 20)
  # make the 20 squared variables
  #note: im makeing an extra column. Ill fix this if I have time but this is easy for now
  for(i in 1:20){

  r_dt[, temp := x^i]
  setnames(r_dt, "temp", paste0("x_exp_", i))


  # conver things to matrices to get the y hats
  x_mat <- as.matrix(r_dt[, c(grep("x_exp", colnames(r_dt), value = TRUE), "const"), with = FALSE])
  y_mat <- as.matrix(r_dt[, y])

  # get the projection matrix
  X.Q <- qr.Q(qr(x_mat))
  XX <- tcrossprod(X.Q)
  Y.hat <- XX %*% y_mat

  # now put this crap in a data.table to calculate cv
  res <- data.table(y_hat = Y.hat, w = diag(XX), y = r_dt[, y])

  # now calculate cv
  res[, cv_n := ((y - y_hat.V1)/(1-w))^2]

  # now get the mean of cv_i to get cv
  res <-  data.table(cv = res[, mean(cv_n)], k = i)
  setnames(res, "cv", paste0("cv_", sim))
  results[[i]] <- res

  }

  print(sim)
  # bind results
return(rbindlist(results))
```

```
}

start_t <- Sys.time()

# parallel setup
cl <- makeCluster(4, type = "PSOCK")
registerDoParallel(cl)

# run simulations in parallel
all_out <- foreach(sim_i = 1 : M,
                   .inorder = FALSE,
                   .packages = "data.table",
                   .options.multicore = list(preschedule = FALSE, cleanup = 9)) %dopar% power_s_fu


# now merge all results
all_out_dt <-Reduce(function(x, y) merge(x, y, by = "k"), all_out)

# stop clusters
stopCluster(cl)

# check time
run_time2 <- Sys.time() - start_t

# row sum my data to get the averaage cv for each k
all_out_dt[, k := NULL]
mean_cv <- data.table( cv_means = rowMeans(all_out_dt), k = 1:20)

# now plot that bad boy

# initialize base data mapping for plot
plot_2_5_b <- ggplot(data = mean_cv, aes(x = k, y = cv_means))

plot_2_5_b <- plot_2_5_b + geom_point(size = 1) + geom_line() + plot_attributes
plot_2_5_b

#=================#
# ==== part c ====
#=================#


    # write a function to apply accross simulations
    B_fun <- function(sim = NULL){

      r_dt <- gen_data_2.5.a()
      r_dt[, const :=1]


      # make the y vars
      for(i in 1:7){

        r_dt[, temp := x^i]
        setnames(r_dt, "temp", paste0("x_exp_", i))
```

```
    }

      # conver things to matrices to get the y hats
      x_mat <- as.matrix(r_dt[, c(grep("x_exp", colnames(r_dt), value = TRUE), "const"), with = FALS
      y_mat <- as.matrix(r_dt[, y])

      # get betas
      B <- Matrix::solve(Matrix::crossprod(x_mat, x_mat))%*%(Matrix::crossprod(x_mat, y_mat))

      # get weights
      X.Q <- qr.Q(qr(x_mat))
      XX <- tcrossprod(X.Q)
      weights <- diag(XX)
      Y.hat <- XX %*% y_mat

      # now square the weights
      weights_sq <- weights^2

      # now get se
      se <- sqrt(sum(weights_sq) * var(y_mat - Y.hat))



      # put the stuff in a list
      output <- list()
      output[["B"]] <- B
      output[["se"]] <- se

    # return the betas
    return(output)

  }

  start_t <- Sys.time()

  # okay now run this shit 1000 times
 bw_stuff <- lapply(c(1:M), B_fun)

 run_time3 <- Sys.time() - start_t

 # now do some dumb stuff because its late
 b_list <- list()
 se_list <- list()
 for(i in 1:M){

   b_list[[i]] <- bw_stuff[[i]][["B"]]
   se_list[[i]] <- bw_stuff[[i]][["se"]]
 }

 b_mat <- do.call(cbind, b_list)
 se_mat <- do.call(cbind, se_list)

 # sum the rows
```

```r
  betas <- rowMeans(b_mat)
  se <-   rowMeans(se_mat)

  # now write a function to plot the u hat funciton
  u_hat_fun <- function(x){

    betas[[8]] + betas[[1]]*x + betas[[2]]*x^2 + betas[[3]]*x^3 + betas[[4]]*x^4 + betas[[5]]*x^5 +
  }


  # write out true function
  true_fun <- function(x){

    exp(-0.1*(4*x-1)^2)*sin(5*x)

  }


 #=====================#
 # ==== part c plot ====
 #=====================#


  # plot the true functin
  plot_2_5_c <- ggplot(data = data.frame(x = 0), mapping = aes(x = x))
  plot_2_5_c <- plot_2_5_c + stat_function(fun = true_fun,
                                           color = "blue")
  plot_2_5_c <- plot_2_5_c + plot_attributes + xlim(-1,1)


  # now add u hat function
  plot_2_5_c <- plot_2_5_c + stat_function(fun = u_hat_fun,
                                             color = "red", linetype = 2)

  plot_2_5_c <- plot_2_5_c + scale_colour_identity("Function", guide="legend",
                                                labels = c("U hat", "True U"),
                                                breaks = c("red", "blue")) + theme(axis.title.y=element_
  # create some data to plot with the standard errors
  plot_data <- data.table(x = seq(-1,1,.2))
  plot_data[, y_hat := u_hat_fun(x)]
  plot_data[, se := se]

  plot_2_5_c <- plot_2_5_c + geom_point(data = plot_data, mapping = aes(x = x, y = y_hat),
                                        color = "red")

  plot_2_5_c <- plot_2_5_c + geom_errorbar(data = plot_data, aes(ymin=y_hat-se, ymax=y_hat+se), wid

  # print it out to see if it looks alright
  plot_2_5_c

#=====================#
# ==== part d pot ====
#=====================#
```

```
    # create derivative funciton
    # write out true function
    true_fun_d <- function(x){

      exp(-0.1*(4*x-1)^2)*(5*cos(5*x) - 0.8*(4*x-1)*sin(5*x))

    }

    # write out estimated polynomial
    est_fun_d <- function(x){
     betas[[1]] + 2*betas[[2]]*x + 3*betas[[3]]*x^2 + 4*betas[[4]]*x^3 + 5*betas[[5]]*x^4 + 6*betas[[(

    }

    # plot the true functin
    plot_2_5_d <- ggplot(data = data.frame(x = 0), mapping = aes(x = x))
    plot_2_5_d <- plot_2_5_d + stat_function(fun = true_fun_d,
                                              color = "blue")
    plot_2_5_d <- plot_2_5_d + plot_attributes + xlim(-1,1)


    # now add u hat function
    plot_2_5_d <- plot_2_5_d + stat_function(fun = est_fun_d,
                                              color = "red", linetype = 2)

    plot_2_5_d <- plot_2_5_d + scale_colour_identity("Function", guide="legend",
                                                      labels = c("U hat", "True U"),
                                                      breaks = c("red", "blue")) + theme(axis.title.y=(
    # create some data to plot with the standard errors
    plot_data <- data.table(x = seq(-1,1,.2))
    plot_data[, y_hat := est_fun_d(x)]
    plot_data[, se := se]

    plot_2_5_d <- plot_2_5_d + geom_point(data = plot_data, mapping = aes(x = x, y = y_hat),
                                           color = "red")

    plot_2_5_d <- plot_2_5_d + geom_errorbar(data = plot_data, aes(ymin=y_hat-se, ymax=y_hat+se), wid

    # print it out to see if it looks alright
    plot_2_5_d

#=====================#
# ==== save plots ====
#=====================#

    # only save data if this isn't a test run
    if(!opt_test_run){

      # save the plot
      png("c:/Users/Nmath_000/Documents/Code/courses/econ 675/PS_2_tex/plot_2_5_b.png", height = 800,
      print(plot_2_5_b)
      dev.off()
```

```
      # save the plot
      png("c:/Users/Nmath_000/Documents/Code/courses/econ 675/PS_2_tex/plot_2_5_c.png", height = 800,
      print(plot_2_5_c)
      dev.off()

      # save the plot
      png("c:/Users/Nmath_000/Documents/Code/courses/econ 675/PS_2_tex/plot_2_5_d.png", height = 800,
      print(plot_2_5_d)
      dev.off()



    }



#======================#
# ==== question 3 ====
#=====================#

  #=================#
  # ==== Part a ====
  #=================#

    d   = 5
    theta_n = 1


    data_gen <- function(n) {
      X <- matrix(runif(n*d,-1,1), n, d)
      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,
    ## RETURN POLYNOMIAL BASIS
    return(out)
  }

#=================#
# ==== part b ====
#=================#

  n   <- 500
  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)
  M <- ifelse(opt_test_run, 10, 1000)
  theta.hat <- matrix(NaN, ncol=nK, nrow=M)
  se.hat    <- theta.hat

  set.seed(123)
  ptm <- proc.time()
  for (m in 1:M) {
    data <- data_gen(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))
      MP     <- diag(rep(1,n)) - X.Q %*% t(X.Q)
      Y.M <- MP %*% Y
      tt.M <- MP %*% tt
      theta.hat[m, k] <- (t(tt.M) %*% Y.M) / (t(tt.M) %*% tt.M)
      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)
    }
  }
  proc.time() - ptm

  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]) - 1                              # bias
    table[k, 3] <- sd(theta.hat[, k])                                    # standard deviation
    table[k, 4] <- table[k, 2]^2 + table[k, 3]^2                         # mse
    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
  }

  table <- data.table(table)
  setnames(table, colnames(table), c("K", "Theta", "Bias", "S.D", "V_HCO", "Rejection rate"))

#====================#
```

```r
# ==== save table ====
#=====================#

    # save IMSE by h results
    print(xtable(table, type = "latex",
                 digits = 3),
          file = "C:/Users/Nmath_000/Documents/Code/courses/econ 675/PS_2_tex/Q3_4_b.tex",
          include.rownames = FALSE,
          floating = FALSE)


#=====================#
# ==== Q3. 4. (c) ====
#=====================#



    # 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

    set.seed(123)
    ptm <- proc.time()
    for (m in 1:M) {
      data <- data_gen(n)
      X <- data$X; Y <- data$Y; tt <- data$tt
      k.opt <- K.CV(tt, X, Y)
      X.pol <- cbind(1, gen.P(X, K[k.opt]))
      X.Q   <- qr.Q(qr(X.pol))
      MP    <- diag(rep(1,n)) - X.Q %*% t(X.Q)
      Y.M   <- MP %*% Y
      tt.M  <- MP %*% tt
      theta.hat2[m] <- (t(tt.M) %*% Y.M) / (t(tt.M) %*% tt.M)
      Sigma         <- diag((as.numeric((Y.M - tt.M*theta.hat[m, k])))^2)
      se.hat2[m]    <- sqrt(t(tt.M) %*% Sigma %*% tt.M) / (t(tt.M) %*% tt.M)
      K.hat2[m]     <- K.r[k.opt]
    }
    time4 <- proc.time() - ptm
```

```r
# summary of the cross validation
table(K.hat2)

# estimator
summary(theta.hat2)
sd(theta.hat2)
summary(se.hat2)
sd(se.hat2)

par(mfrow=c(1,2))
hist(theta.hat2, freq=FALSE, xlab="theta-hat", ylab="", main="")
lines(c(mean(theta.hat2), mean(theta.hat2)), c(-1, 20), col="red", lwd=3)
hist(se.hat2, freq=FALSE, xlab="s.e.", ylab="", main="")
lines(c(mean(se.hat2), mean(se.hat2)), c(-1, 80), col="red", lwd=3)


par(mfrow=c(1,2))
CI.l <- theta.hat2 - 1.96 * se.hat2
CI.r <- theta.hat2 + 1.96 * se.hat2

# rejection rate
mean(1 < CI.l | 1 > CI.r)
plot(1:M, CI.l, type="l", ylim=c(0,2), xlab="simulations", ylab="CI")
lines(1:M, CI.r)
abline(1, 0, col="red", lwd=2)

temp <- sort(CI.l, index.return=TRUE)
CI.l <- temp$x
CI.r <- CI.r[temp$ix]
plot(1:M, CI.l, type="l", ylim=c(0,2), xlab="simulations", ylab="CI")
lines(1:M, CI.r)
abline(1, 0, col="red", lwd=2)
```

4.2 STATA Code

```
* NOTES:
* My code fricking CRAAAWLS. It is extremely slow. I guess trying to jerry rig how
* I did this in R into the weird world of stata was not a great idea.


clear all
set more off, perm

global dir "c:\Users\Nmath_000\Documents\Code\courses\econ 675\PS_2_tex\"

cap log close
log using $pset2_stata_log.smcl, replace
*************************************
************ Question 1 ************
*************************************

 global hvalues .5 .6 .7 .8 0.8199 .9 1 1.1 1.2 1.3 1.4 1.5
* global hvalues .5
local h = .5
local i = 1
global n = 1000

* I need to only do 10 simulations because of how slow this thing is
global m = 10
set obs $n

* replace with for loop eventually
forvalues i = 1/10{
di `i'
* start loop
clear
set obs $n
* generate random data
gen z_o = uniform()
gen xi = rnormal(-1.5, sqrt(1.5)) if z_o < .5
replace xi = rnormal(1,1) if z_o >= .5

* drop zero one var
drop z_o

* gen constaant for merge
gen const = 1

* save as temp file for merge
tempfile rand_i
 save "`rand_i'"


 * rename variable
 rename xi x

* try merging this with teacher level enr_staff file
joinby const using `rand_i'

* now loop over h values
foreach h in $hvalues {

    di `h'
    * make h for file names
    local h_n: subinstr local h "." "", all

    *preserve data before I mess with is
    preserve

    * gnerate u
    gen u =  (xi-x )/`h'

    * calculate kernal for pairs
    gen kern = (.75*(1-u^2)*(abs(u)<=1))/`h'

```

```

	* get means
	replace x = round(x,.00001)
	bys x: egen fhats = mean(kern)
	egen tag = tag(x)
	keep if tag == 1
	drop xi const u kern tag

	* add in f_x
	gen f_x = .5*normalden(x, -1.5, sqrt(1.5)) + .5*normalden(x, 1, 1)

	* find sq error
	gen sq_er = (fhats-f_x)^2

	* now get  imse_li
	egen imse_li = mean(sq_er)
	egen tag2 = tag(imse_li)
	keep if tag2 == 1

	keep imse_li

	* fill in sum info
	gen sim = `i'
	gen h = `h'


	* save temp data
	tempfile imseli_`h_n'_`i'
	 save "imseli_`h_n'_`i'", replace

	* restore data, preserve it for next thing
	restore

	preserve
	* now do the leave on out, drop columns with the same x xi
	* this is bad coding but STATA is terrible so this is what it deserves
	keep if x != xi

	* gnerate u
	gen u =  (xi-x )/`h'

	* calculate kernal for pairs
	gen kern = (.75*(1-u^2)*(abs(u)<=1))/`h'

	* collaps data to get means \* collapse data
	replace x = round(x,.00001)
	bys x: egen fhats = mean(kern)
	egen tag = tag(x)
	keep if tag == 1
	drop xi const u kern tag

	* add in f_x
	gen f_x = .5*normalden(x, -1.5, sqrt(1.5)) + .5*normalden(x, 1, 1)

	* find sq error
	gen sq_er = (fhats-f_x)^2

	* now get  imse_li
	egen imse_lo = mean(sq_er)
	egen tag2 = tag(imse_lo)
	keep if tag2 == 1

	keep imse_lo


	* fill in sum info
	gen sim = `i'
	gen h = `h'


```

```
        * save temp data
        tempfile imselo_`h_n'_`i'
        quietly save "imselo_`h_n'_`i'" , replace

        * restore data for next h
        restore

    }
    }
    * now, because I dont think stata has lists we just load all that back in and stack it
    * clear out data
    clear

    forvalues i = 1/$m{
    foreach h in $hvalues {

        * make h for file names
        local h_n: subinstr local h "." "", all

        append using "imseli_`h_n'_`i'.dta"
        append using "imselo_`h_n'_`i'.dta"

    }
    }

    * Now collapse data to get mean leave on in and out across iteratiosn by h
        bys h: egen m_imse_li = mean(imse_li)
        bys h: egen m_imse_lo = mean(imse_lo)
        egen tag = tag(h)
        keep if tag == 1
        keep h m_imse_li m_imse_lo


    * graph this stuff
    line m_imse_li m_imse_lo h

    graph export "$dir\stata_plot_1_3_b.png", replace

    dataout, save($dir\stata_table_1_3_b.tex) tex replace


    **************
    **** Problem 2
    **************
    **************
    **** Problem 2.5.b
    ***************
    clear all
    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 "$dir\stata_plot_2_5_b.png", replace

****************
***Problem 2.5.c
****************

* 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 "$dir\stata_plot_2_5_c.png", replace


***************
* poblem 2.5.d
****************


* 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 "$dir\stata_plot_2_5_d.png", replace
```

```

***************************
******** Question 3 ********
***************************
* DGP
clear all
drop _all
local theta = 1
local d = 5
local n = 500

set obs 1000

forvalues p = 1/14 {
gen se_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)
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, "se_hat"+strofreal(j), se_hat[1,j])
st_store(i, "theta_hat"+strofreal(j), theta_hat[1,j])
}

}
end

forvalues i = 1/1000 {
mata polyloop(`i')
}

gen theta = _n

reshape long se_hat theta_hat, i(theta) j(K)

replace theta = 1

gen bias = theta_hat - theta
gen cov = ((theta_hat - invnormal(.975)*abs(se_hat) <= 1) & (theta_hat + invnormal(.975)*abs
(se_hat) >= 1))

collapse se_hat theta_hat bias cov (sd) svar = theta_hat, by(K)


label var se_hat "SE"
label var theta_hat "Thetahat"
label var bias "Bias"
label var cov "Coverage"
label var svar "Sample Standard Dev."

order theta_hat se_hat bias cov svar
dataout, save($dir\stata_table_3_4_d.png) tex replace
```