Answers to Empirical IO I: Problem Set 0

Part 0: Logit Inclusive Value

The logit inclusive value or $IV = \log \sum_{i=0}^{N} \exp[x_i]$.

1. Show that the this function is everywhere convex if $x_0 = 0$.
2. A common problem in practice is that if one of the $x_i > 600$ that we have an "overflow" error on a computer. In this case $\exp[600] \approx 10^{260}$ which is too large to store with any real precision, especially if another x has a different scale (say $x_2 = 10$). A common "trick" is to subtract off $m_i = \max_i x_i$ from all x_i. Show how to implement the trick and get the correct value of IV. If you get stuck take a look at Wikipedia.
3. Compare your function to `scipy.special.logsumexp`. Does it appear to suffer from underflow/overflow? Does it use the max trick?

1. We have the function:

$$IV = f(x_1, \ldots, x_N) \; = \; \log\Big(1 + e^{x_1} + \cdots + e^{x_N}\Big),$$

Fix $x = (x_1, \ldots, x_N)$ and a direction $h = (h_1, \ldots, h_N)$. Consider the single-variable function:

$$g(\theta) \; = \; f(x + \theta h) = \log\Big(1 + \sum_{i=1}^{N} e^{\,x_i + \theta h_i}\Big).$$

First derivative:

$$g'(\theta) = \frac{\sum_{i=1}^{N} h_i e^{x_i + \theta h_i}}{1 + \sum_{i=1}^{N} e^{x_i + \theta h_i}}.$$

Second derivative. Writing $a_i(\theta) = e^{x_i + \theta h_i}$,

$$g''(\theta) = \frac{\Big(1 + \sum_{i=1}^{N} a_i(\theta)\Big)\Big(\sum_{i=1}^{N} h_i^2 a_i(\theta)\Big) - \Big(\sum_{i=1}^{N} h_i a_i(\theta)\Big)^2}{\Big(1 + \sum_{i=1}^{N} a_i(\theta)\Big)^2}.$$

Expanding the numerator gives

$$\Big(1+\sum_{i=1}^{N} a_i(\theta)\Big)\Big(\sum_{i=1}^{N} h_i^2 a_i(\theta)\Big) - \Big(\sum_{i=1}^{N} h_i a_i(\theta)\Big)^2 = \sum_{1\leq i<j\leq N} a_i(\theta)a_j(\theta)\,(h_i - h_j)^2.$$

Hence

$$g''(\theta) = \frac{\sum_{1\leq i<j\leq N} e^{x_i+\theta h_i}\, e^{x_j+\theta h_j}\,(h_i-h_j)^2}{\Big(1+\sum_{k=1}^{N} e^{x_k+\theta h_k}\Big)^2} \;\geq\; 0.$$

Since $g''(\theta) \geq 0$ for all θ, g is convex in θ. Therefore IV is everywhere convex.

2. See Julia code. Used test vector $x = (10, 11, 12, 1000)$ and showed that naive implementation of inclusive value led to infinte result, whereas "trick" gave value ≈ 1000

3. Compared to the Julia equivalent, StatsFuns. This produced the same result and does not appear to suffer from overflow. Underflow should be harmless here: small terms that underflow to 0 won't contribute anything to the IV anyway.

Part 1: Markov Chains

Consider the following Markov TPM:
Let $P = \{p_{i,j}\}$ be an $n \times n$ transition matrix of a Markov process where $\{p_{i,j}\}$ is interpreted as the probability that the system, when it is in state i, will move to state j. If we denote by $\pi_t[\pi_{t,1}, \pi_{t,2}, \ldots, \pi_{t,n}]$ the probability mass function of the system over the n states then $\pi_{t,j}$ evolves according to

$$\pi_{t+1,j} = \sum_{i=1}^{n} p_{i,j}\pi_{t,i}$$

Then we can write the state to state transition matrix as :

$$\begin{aligned} \pi_{t+1} &= \pi_t P \\ P &= \begin{bmatrix} 0.2 & 0.4 & 0.4 \\ 0.1 & 0.3 & 0.6 \\ 0.5 & 0.1 & 0.4 \end{bmatrix} \end{aligned}$$

We're interested in the ergodic distribution $\pi P = \pi$. This is similar to the transition matrix infinitely many periods into the future P^∞. Write a function that computes the ergodic distribution of the matrix P by examining the properly rescaled eigenvectors and compare your result to P^{100}. Here I recommend `numpy.linalg.matrix_power` and `numpy.linalg.eig`. (A common mistake is element-wise exponentiation of the matrix).

See Julia code. Both approaches lead to the same values of π to 6 places: $(0.310345, 0.241379, 0.448276)$

Part 2: Numerical Integration

Note: The `scipy` quadrature routines may return a set of nodes/weights that correspond to integrating e^{-x^2} over (∞, ∞), while the nodes/weights available at `http://www.sparse-grids.de` may not. It is always important to make sure you understand what your nodes/weights correspond to. One way to do this is to integrate some simple functions $f(x) = 1, f(x) = x, f(x) = x^2$ where you know the analytic result and see what the quadrature routine gives as the answer.

In this part we will look to calculation the logit choice probability $p(X, \theta)$ by numerical integration:
$p(X, \theta) = \int_{-\infty}^{\infty} \frac{\exp(\beta_i X)}{1+\exp(\beta_i X)} f(\beta_i|\theta) \partial\beta_i$.
Assume $f \sim N(0.5, 2)$ and that $X = 0.5$.

1. Create the function in an Python called `binomiallogit`. (It should take β the item you integrate over as its argument, it should take the PDF `scipy.stats.norm.pdf` as an optional argument).
2. Integrate the function using Python's `scipy.integrate.quad` command and setting the tolerance to 1×10^{-14}. Treat this a the "true" value.
3. Integrate the function by taking 20 and 400 Monte Carlo draws from f and computing the sample mean.
4. Integrate the function using Gauss-Hermite quadrature for $k = 4, 12$ (Try some odd ones too). Obtain the quadrature points and nodes from the internet. Gauss-Hermite quadrature assumes a weighting function of $\exp[-x^2]$, you will need a change of variables to integrate over a normal density.[See my notes] You also need to pay attention to the constant of integration.

1. See Julia code.

2. "True" value ≈ 0.551

3. See Julia code.

4. To apply Gauss–Hermite quadrature, recall that the standard formula approximates

$$\int_{-\infty}^{\infty} e^{-x^2} g(x)\, dx \approx \sum_{i=1}^{k} w_i g(x_i),$$

where $\{x_i, w_i\}$ are the Hermite nodes and weights. Our target integral is an expectation with respect to a normal density:

$$\int_{-\infty}^{\infty} \sigma(\beta X)\, f(\beta)\, d\beta, \quad f(\beta) = \frac{1}{\sqrt{2\pi}\,\sigma}\, \exp\Big(-(\beta-\mu)^2 2\sigma^2\Big).$$

First set $z = (\beta - \mu)/\sigma$, so that $\beta = \mu + \sigma z$ and $z \sim N(0, 1)$. This gives

$$\int_{-\infty}^{\infty} \sigma\big((\mu + \sigma z)X\big)\, \phi(z)\, dz, \quad \phi(z) = 1\sqrt{2\pi} e^{-z^2/2}.$$

Next substitute $z = \sqrt{2}\, x$, yielding

$$\int_{-\infty}^{\infty} \sigma\big((\mu + \sigma\sqrt{2}x)X\big)\, \frac{1}{\sqrt{\pi}} e^{-x^2}\, dx.$$

This now matches the Gauss–Hermite weight e^{-x^2}. Therefore,

$$\int \sigma(\beta X) f(\beta)\, d\beta \;\approx\; \frac{1}{\sqrt{\pi}} \sum_{i=1}^{k} w_i\, \sigma\big((\mu + \sigma\sqrt{2}\, x_i)X\big).$$

Hence the Hermite nodes x_i are rescaled to $\beta_i = \mu + \sigma\sqrt{2}\, x_i$, and the effective weights are $w_i/\sqrt{\pi}$, which sum to one. This ensures the quadrature formula is consistent with integration under the normal density.

See Julia code for implementation.

5. Compare results to the Monte Carlo results. *Make sure your quadrature weights sum to 1!*
6. Repeat the exercise in two dimensions where $\mu = (0.5, 1), \sigma = (2, 1)$, and $X = (0.5, 1)$.
7. Put everything into two tables (one for the 1-D integral, one for the 2-D integral). Showing the error from the "true" value and the number of points used in the evaluation.
8. Now Construct a new function `binomiallogitmixture` that takes a vector for X and returns a vector of binomial probabilities (appropriately integrated over $f(\beta_i|\theta)$ for the 1-D mixture). It should be obvious that Gauss-Hermite is the most efficient way to do this. *Do NOT use loops*

5. All values are reasonably close to the "true" value (i.e. 2 places) but the GH values are even closer (3 places +).
6. See Julia code.

7.

Table 1: Numerical Integration Results

1-D Results (true value: 0.551493)		
Method	Points	Error
Monte Carlo	20	8.756e−3
Monte Carlo	400	5.735e−3
Gauss–Hermite	4	1.794e−4
Gauss–Hermite	9	9.517e−7
Gauss–Hermite	12	6.857e−8

2-D Results (true value: 0.714484)		
Method	Points	Error
Monte Carlo	20	1.418e−2
Monte Carlo	400	4.135e−4
Gauss–Hermite (4×4)	16	1.124e−4
Gauss–Hermite (9×9)	81	4.193e−8
Gauss–Hermite (12×12)	144	4.464e−9

Note: For Monte Carlo, "Points" = random draws; for Gauss–Hermite, "Points" = quadrature nodes.

8. See Julia code.