Simulation

Assignment 2

Platon
Karageorgis
p3180068

① First we need to check if Laplace has an integral equal to 1. If not, we will multiply by $\frac{1}{c}$ where c is the value that we will find.

$$\int_{-1}^{2} \frac{1}{2b} \cdot e^{\frac{-|x|}{b}} dx = \int_{-1}^{0} \frac{1}{2b} \cdot e^{\frac{x}{b}} dx + \int_{0}^{2} \frac{1}{2b} \cdot e^{\frac{-x}{b}} dx$$

$$= \left[\frac{e^{\frac{x}{b}}}{2} \right]_{-1}^{0} + \left[-\frac{e^{\frac{-x}{2}}}{2} \right]_{0}^{2} = \underbrace{\left(\frac{e^{0}}{2} - \frac{e^{-1/b}}{2} \right)}_{c_1} - \underbrace{\left(\frac{e^{-2/b}}{2} + \frac{e^{0}}{2} \right)}_{c_2} =$$

$$= 1 - \left(\frac{e^{-1/b} + e^{-2/b}}{2} \right)$$

$$c = \frac{1}{1 - \left(\frac{e^{-1/b} + e^{-2/b}}{2} \right)} \Rightarrow \boxed{c = \frac{2}{2 - e^{-1/b} - e^{-2/b}}} \quad \boxed{c_1 = \frac{2}{1 - e^{-1/b}}} \quad \boxed{c_2 = \frac{2}{1 - e^{-2/b}}}$$

But for a reason that will be explained later, we will define g in 2 branches and each will get a chunk of c (c_1 and c_2 respectively).

$$g(x) = \begin{cases} \left(\frac{2}{1 - e^{-1/b}} \right) \cdot \frac{e^{\frac{x}{b}}}{2b} = \frac{e^{\frac{x}{b}}}{b \cdot (1 - e^{-1/b})}, & x \in [-1, 0] \\ \left(\frac{2}{1 - e^{-2/b}} \right) \frac{e^{\frac{-x}{b}}}{2b} = \frac{e^{\frac{-x}{b}}}{b \cdot (1 - e^{-2/b})}, & x \in [0, 2] \end{cases}$$

Now we need to multiply f as well and we will once more create separate branches.

$$f(x) = \begin{cases} \left(\frac{2}{1 - e^{-1/b}} \right) \cdot \frac{1}{\sqrt{2\pi}} \cdot e^{-x^2/2}, & x \in [-1, 0] \\ \frac{2}{2 - e^{-2/b}} \cdot \frac{1}{\sqrt{2\pi}} e^{-x^2/2}, & x \in [0, 2] \end{cases}$$

Moving on, we need to get a sample from g and we will use the method of inversion. To ~~[illegible]~~ cut short:

$$G(x)=\begin{cases}\dfrac{e^{\frac{x}{b}}-e^{\frac{-1}{b}}}{1-e^{\frac{-1}{b}}}, & x\in[-1,0]\\[2ex] \dfrac{1-e^{\frac{-x}{b}}}{1-e^{\frac{-2}{b}}}, & x\in[0,2]\end{cases}$$

$$G^{-1}(x)=\begin{cases} b\cdot\ln\left(e^{-\frac{1}{b}}+x\left(1-e^{-\frac{1}{b}}\right)\right)\\ -b\cdot\ln\left(1-x\left(1-e^{\frac{-2}{b}}\right)\right)\end{cases}$$

Now we need to find 'M'. We will maximise $h(x)=\frac{f(x)}{g(x)}$

$$h(x)=\begin{cases}\dfrac{\dfrac{2\cdot e^{-x^2/2}}{\sqrt{2\pi}\cdot(1-e^{-1/b})}}{\dfrac{e^{x/b}}{b\cdot(1-e^{-1/b})}}=\ldots=\dfrac{2\cdot b\cdot e^{\frac{-bx^2-2x}{2b}}}{\sqrt{2\pi}}, & x\in[-1,0]\\[4ex] \dfrac{\dfrac{2\cdot e^{-x^2/2}}{\sqrt{2\pi}\cdot(1-e^{-2/b})}}{\dfrac{e^{-x/b}}{b\cdot(1-e^{-2/b})}}=\ldots=\dfrac{2\cdot b\cdot e^{\frac{-bx^2+2x}{2b}}}{\sqrt{2\pi}}, & x\in[0,2]\end{cases}$$

After maximising we end up with $x_1<\frac{1}{b}$ and $x_1<-\frac{1}{b}$ for each branch respectively. We substitute in the place of x and the maximum of the $h_1(b)$, $h_2(b)$ will be M but first we need to find the best case b. We want M to be as little as possible since $\frac{1}{M}$ which is the acceptance ratio is critical for the speed of the algorithm. By plotting $h(b)$ we end up with $b=1$ (Python affirms this claim).
So finally our M will be:

$$\boxed{M=\frac{2\cdot e^{1/2}}{\sqrt{2\pi}}}$$

Before writing down the final algorithm, the explanation for the 2 branches in every case is that I wanted the whole exercise to have the same solution and the Python code was not working with ~~the~~ 1 branch. This method has a last trick, since f is symmetrical in [-1,2] we need to find a condition for u in order to choose the correct branch each time. This is done by creating a 'ratio' variable that calculates the percentage of the total integral in 1 of the 2 cases. So ratio is:

$$\text{ratio} = \frac{1-e^{-1/6}}{2-e^{-1/6}-e^{-2/6}} \overset{b=1}{=} \frac{1-e^{-1}}{2-e^{-1}-e^{-2}} \simeq 0.42$$

Algorithm

1. Generate a Y from $g(y)$
2. Generate a $u \sim U(0,1)$
3. Generate a $u_2 \sim U(0,1)$
4. If $u_2 \geq$ ratio:

    If $u_1 < \frac{h_1(Y)}{M}$ then accept; Else go back to 1

If $u_2 <$ ratio:

    If $u < \frac{h_2(Y)}{M}$ then accept; Else go back to 1

(3) This part allows us to choose our own method that accomplishes the best possible ratio. As a method I will use the Cubic Spline which is an interpolation method using n small polynoms. The splines will be natural which means that we set the second derivative equal to zero.

It is impossible to calculate an effective spline with many points but essentially the idea is that we take n points with an equal distance between them and between every pair we create a polynom that must pass through every single one of those points.

The rate of this algorithm will be $\frac{1}{C \cdot M} = \frac{1}{0.81 \cdot 1} = 1.23$

and the reason is that M will definitely be equal to 1 since cubic splines is a very precise method, but c is equal to 0.81 since we need to normalize the integral of $g \equiv f$ which is 0.81. So whilst doing that we also lose precision.

As for the algorithm, as long as we generate $g(x)$, we get a sample via the inversion method and then the rest of the algorithm is the same.

Algorithm

1. Create a Cubic Spline that mimics f and make the integral equal to 1.
2. Get a sample Y from g
3. Generate a $U \sim U(0,1)$
4. If $U < \frac{f(Y)}{g(Y) \cdot M}$ then accept; Else go back to (1)

Note: The code never runs, I ran out of time