SYDE 556/750
Simulating Neurobiological Systems
Assignment 2

Due date: February 11, 2020

Important Notes – Please Read Carefully

- This assignment is worth 20 marks (20% of the final grade). The number of marks for each question is indicated in brackets to the left of each question.
- Please use Python 3 along with the `numpy` and `matplotlib` libraries to solve these assignments. In particular, fill out this Jupyter Notebook template:

  `https://github.com/astoeckel/syde556-w20/blob/master/assignments/assignment_02/syde556_assignment_02_template.ipynb`

- Clearly label any plot you produce, including the axes. Provide a legend if there are multiple graphs in the same plot.
- You won't be judged on the quality of your code, but writing reusable functions will greatly simplify this and future assignments.
- Make sure to execute the Jupyter command "Restart Kernel and Run All Cells" before submitting your solutions. You will lose marks if your code fails to run or produces results that differ significantly from what you've submitted.
- Rename the completed notebook to `syde556_assignment_02_<STUDENT ID>.ipynb` and email it to `astoecke@uwaterloo.ca` using your University of Waterloo email account, and make sure to include "[SYDE 556/750]", "Assignment 2" and your student ID in the subject line. The deadline is at `23:59 EST` on February 11, 2020.
- There is a late penalty of one mark per day late. You may be at most seven days late.
- **Do not use or refer to any code from Nengo!**

1 Generating a random input signal

1.1 Band-limited white noise

Create a function called that generates a randomly varying $x(t)$ signal chosen from a band limited white noise distribution. Call it `generate_signal` and ensure that it returns both $x(t)$ and $X(\omega)$, i.e., both its time- and Fourier-domain representation.

The inputs to this function are:

- `T`: the length of the signal in seconds.
- `dt`: the time step in seconds.
- `rms`: the root mean square power (RMS) level of the signal. The RMS σ of a time-continous signal $x(t)$ of length T is defined as

$$\sigma = \sqrt{\frac{1}{T}\int_0^T x(t)^2 \, \mathrm{d}t}$$

- `limit`: the maximum frequency for the signal.
- `seed`: the random number seed to use (so we can deterministically generate the same signal again).

[1] a) *Time-domain signals.* Plot $x(t)$ for three randomly generated signals with `limit` at 5, 10, and 20 Hz. For each of these, `T` = 1 s, `dt` = 1 ms and `rms` = 0.5.

[1] b) *Average power spectrum.* Plot the average $|X(\omega)|$ (the norm of the Fourier coefficients, or "power spectrum") over 100 signals generated with `T` = 1 s, `dt` = 1 ms, `rms` = 0.5, and `limit` = 0.5 (of course, each of these 100 signals should have a different `seed`). The plot should have the x-axis labeled "ω in radians" and the average $|X|$ value for that ω on the y-axis.

- For more information on how to do Fourier transformations in Python, see here.
- The transformation takes you from t to ω (or back the other way). Importantly, ω is frequency in radians, not in Hz. $\Delta\omega$ (i.e., the difference in frequency between two samples) will be $\frac{2\pi}{T}$.
- Ensure that the generated signal has no complex components in the time domain, i.e., is purely real. It holds $X(\omega) = \overline{X(-\omega)}$ exactly if the signal is real, where $\overline{a + \mathrm{i}b} = a - \mathrm{i}b$ denotes the complex conjugate.
- When randomly generating $X(\omega)$ values, sample them from a Normal distribution $\mathcal{N}(\mu = 0; \sigma = 1)$. Remember that the $X(\omega)$ are complex numbers, so sample twice from the distribution; once for the real component and once for the imaginary.
- To implement the limit, set all $X(\omega)$ components with frequencies above the limit to 0.
- To implement the `rms`, generate the signal, compute its RMS ($\sigma = \sqrt{\frac{1}{T}\int x(t)^2 dt}$) and rescale so it has the desired power. Make sure to rescale both the signal in the time- and Fourier-domain (you can use the same scaling factor, since the Fourier transformation is linear). Alternatively, if you want to rescale your initial coefficients, you could try to take advantage of Parseval's Theorem.

1.2 Gaussian power spectrum noise

Create a modified version of your function from question 1.1 that produces noise with a different power spectrum. Instead of having the $X(\omega)$ values be 0 outside of some limit and sampled from $\mathcal{N}(\mu = 0; \sigma = 1)$ inside that limit, we want a smooth drop-off of power as the frequency increases. In particular, instead of the limit, we sample from $\mathcal{N}\left(\mu = 0; \sigma = e^{-\omega^2/(2b^2)}\right)$ where b (in radians) is the new `bandwidth` parameter that replaces the `limit` parameter.

[1] a) *Time-domain signals.* Plot $x(t)$ for three randomly generated signals with `bandwidth` at 5, 10, and 20 Hz. For each of these, `T` = 1 s, `dt` = 1 ms and `rms` = 0.5.

[1] b) *Average power spectrum.* Plot the average $|X(\omega)|$ (the norm of the Fourier coefficients, or "power spectrum") over 100 signals generated with `T` = 1 s, `dt` = 1 ms, `rms` = 0.5, and `bandwidth` = 10 (of course, each of these 100 signals should have a different `seed`). The plot should have the x-axis labeled "ω in radians" and the average $|X|$ value for that ω on the y-axis.

2 Simulating a spiking neuron

Write a program to simulate a single Leaky-Integrate and Fire neuron. The core differential equation is

$$\frac{\mathrm{d}v}{\mathrm{d}t} = \frac{1}{\tau_{\mathrm{RC}}}(J - v)\,,$$

where τ_{RC} is the membrane time constant, J is the input current and v is the current membrane potential. To simplify life, this is normalized so that the resting voltage is 0 and the thereshold voltage is $v_{\mathrm{th}} = 1$. This equation can be simulated numerically by taking small time steps (Euler's method). When the voltage reaches the threshold v_{th}, the neuron will "spike". It's voltage is reset to 0 and stays there for the refractory period τ_{ref} (to plot individual spike, place a dot or line at that time). Also, if the voltage goes below zero at any time, reset it back to zero. For this question, $\tau_{\mathrm{RC}} = 20\,\mathrm{ms}$ and $\tau_{\mathrm{ref}} = 2\,\mathrm{ms}$.

Since we want to have inputs in terms of represented values $\mathbf{x}$, we need to compute $J = \alpha\langle \mathbf{e}, \mathbf{x}\rangle + J^{\mathrm{bias}}$. For this neuron, set $\mathbf{e}$ to 1 and find α and J^{bias} such that the firing rate when $\mathbf{x} = 0$ is 40 Hz and when $\mathbf{x} = 1$ it is 40 Hz. To find these α and J^{bias} values, use the LIF rate approximation (see Assignment 1, Question 1.3):

$$G[J] = \frac{1}{\tau_{\mathrm{ref}} - \tau_{\mathrm{RC}} \ln\left(1 - \frac{1}{J}\right)}$$

[1] a) *Spike plots for constant inputs.* Plot the spike output for a constant input of $x = 0$ over 1 second. Report the number of spikes. Do the same thing for $x = 1$. Use a time step of $\Delta t = 1\,\mathrm{ms}$ for the simulation.

[1] b) *Discussion.* Does the observed number of spikes in the previous part match the expected number of spikes for $x = 0$ and $x = 1$? Why or why not? What aspects of the simulation would affect this accuracy?

[1] c) *Spike plots for white noise inputs.* Plot the spike output for $x(t)$ generated using your function from part 1.1. Use `T` = 1 s, `dt` = 1 ms, `rms` = 0.5, and `limit` = 30 Hz. Overlay on this plot $x(t)$.

[1] d) *Voltage over time.* Using the same $x(t)$ signal as in part c), plot the neuron's voltage over time for the first 0.2 seconds, along with the spikes over the same time.

[+1] e) ☆ *Bonus question.* How could you improve this simulation (in terms of how closely the model matches actual equation) without significantly increasing the computation time? 0.5 marks for having a good idea. Up to 1 mark for actually implementing it and showing that it works.

3 Simulating two spiking neurons

Write a program that simulates two neurons. The two neurons have exactly the same parameters, except one of them uses a positive encoder (i.e., $e = 1$) and the other one uses a negative encoder (i.e., $e = -1$). Other than that, use exactly the same settings as in question 2.

[1] a) *Spike plots for constant inputs.* Plot $x(t)$ and the spiking output for $x(t) = 0$ (both neurons should spike at about 40 spikes per second), as well as (in a separate plot) $x(t) = 1$ (one neuron should spike at ≈ 150 spikes per second, and the other should not spike at all).

[1] b) *Spike plots for a sinusodial input.* Plot $x(t)$ and the spiking output for $x(t) = \frac{1}{2}\sin(10\pi t)$.

[1] c) *Spike plot for a white noise signal.* Plot $x(t)$ and the spiking output for a random signal generated with your function for question 1.1 with `T` $= 2$ s, `dt` $= 0.5$ s, `rms` $= 0.5$, and `limit` $= 5$ Hz.

4 Computing an optimal filter

Compute the optimal filter for decoding pairs of spikes. Instead of implementing this yourself, use and complete the Python implementation that is included in the Jupyter notebook.

[1] a) *Document the code.* Fill in comments where there are `#`-signs in the Python code. Make sure that your comments (where this makes sense) describe the semantics of the code and do not just repeat what is obvious from the code itself. Run the function with what you wrote for part 3 above, so that it uses the spike signal generated in 3c).

[1] b) *Optimal filter.* Plot the time and frequency plots of the optimal filter for the signal you generated in question 3c). Make sure to use appropriate limits for the x-axis.

[1] c) *Decoded signal.* Plot the $x(t)$ signal, the spikes, and the decoded $\hat{x}(t)$ value for the signal from 3c).

[0.5] d) *Power spectra.* Plot the signal $|X(\omega)|$, spike response $|R(\omega)|$, and filtered signal $|\hat{X}(\omega)|$ power spectra for the signal from 3c).

[0.5] e) *Discussion.* How do these spectra relate to the optimal filter?

[0.5] f) *Filter for different signal bandwidths.* Plot the optmial filter $h(t)$ in the time domain when filtering spike trains for white noise signals with different `limit` values of 2 Hz, 10 Hz, and 30 Hz.

[0.5] g) *Discussion.* Describe the effects on the time plot of the optimal filter as `limit` increases. Why does this happen?

5 Using post-synaptic currents as a filter

Instead of using the optimal filter from the previous question, now we will use the post-synaptic current instead. This is of the form

$$h(t) = \begin{cases} \alpha^{-1} t^n e^{-\frac{t}{\tau}} & \text{if } t \geq 0\,, \\ 0 & \text{otherwise}\,. \end{cases}$$

where n is a non-negative integer, and α normalizes the filter to area one, i.e.

$$\alpha = \int_0^\infty t^n e^{-\frac{t}{\tau}}\,\mathrm{d}t\,.$$

[0.5] a) *Plotting the filter for different n.* Plot the normalized $h(t)$ for $n = 0$, 1, and 2, with $\tau = 7\,\text{ms}$.

- You should evaluate the above integral numerically to compute α. There is no closed-form solution.

[0.5] b) *Discussion.* What two things do you expect increasing n will do to $\hat{x}(t)$?

[0.5] c) *Plotting the filter for different τ.* Plot the normalized $h(t)$ for $\tau = 2\,\text{ms}$, $\tau = 5\,\text{ms}$, $\tau = 10\,\text{ms}$, $\tau = 20\,\text{ms}$ with $n = 0$.

[0.5] d) *Discussion.* What two things do you expect increasing τ will do to $\hat{x}(t)$?

[1] e) *Decoding a spike-train using the post-synaptic current filter.* Decode $\hat{x}(t)$ from the spikes generated in question 3c) using an $h(t)$ with $n = 0$ and $\tau = 7\,\text{ms}$. Do this by generating the spikes, filtering them with $h(t)$, and using that as your activity matrix A to compute your decoders. Plot the time and frequency plots for this $h(t)$. Plot the $x(t)$ signal, the spikes, and the decoded $\hat{x}(t)$ value.

- Do not use in-built convolution functions for any question in part 5. Instead, implement convolution this in the time domain by placing the filter when spikes occur, and then summing. This simulates a post-synaptic current occuring on the arrival of a spike from the pre-synaptic neuron.

[0.5] f) *Deocding a spike-train representing a low-frequency signal.* Use the same decoder and $h(t)$ as in part e), but generate a new $x(t)$ with `limit` $= 2\,\text{Hz}$. Plot the $x(t)$ signal, the spikes, and the decoded $\hat{x}(t)$ value.

[0.5] g) *Discussion.* How do the decodings from e) and f) compare? Explain.