EEE3092F
Computer Assignment 2

Instructions

- These exercises must be done with Julia in a Jupyter notebook.
- Before attempting the assignment exercises, you should first download the EEE3092F Julia sample code (as described on the next page) and study the code to become familia with Julia.
- Add plenty of comments to your code and results and to properly label your plots.
- You will be required to submit your work by uploading to Amathuba.

Instructions: Load Jupyter Notebook and Sample Code

- Download the sample code files.
  - 1_julia_complex_numbers_arrays_etc.ipynb
  - 2_julia_plotting_with_Plots_and_Plotly_backend.ipynb
  - 3_julia_signal_processing_demo.ipynb
- Start the Jupyter notebook. Load ipynb file. Try to execute a block (play button, or ctrl-Enter). If it says "Kernel Error", then you must change the kernel to the same version of Julia that you installed (probably 1.10.2). Click on Menu Kernel → Change Kernel
- Before attempting the assignment exercises, go through the above sample code files, running each code block to see what output is produced. These sample files will give you enough background to do the assignment exercises.

Submission of Assignment

- Put all your assignment exercises into a single Jupyter notebook file.
- Include a brief non-plagiarism declaration at the top of your notebook (type out something suitable).
- Make file look readable using "markdown text" boxes for headings and answers to questions - see example files.
- Make sure that all plots have labelled axes and titles, and that you add comments to your code.
- Submit a PDF version of the notebook for ease of marking:
  - Filename: EEE3092F_Assignment2_<student ID>.pdf
  - Run the entire notebook to ensure it runs fully and generates all plots
  - Use the browser's print command (ctrl-P on Firefox or Brave) to generate a PDF file, or use Jupyter's File->Download as PDF method — but this may require additional add-ons to work

Exercises

Exercise 1 - Energy via integration

The energy of a pulse x(t) is defined as $$E = \int_{-\infty}^{\infty} |x(t)|^2 \, dt$$

This can be approximated by $$E = \int_{t_1}^{t_2} p(t)\,dt \approx \sum_{n=0}^{N-1} p(t_1 + n\Delta t)\,\Delta t \quad \text{where} \quad p(t) = |x(t)|^2$$

Here we summing up the areas of rectangles (like in a Riemann sum), which can be made accurate by choosing a sufficiently small time step `Δt`. (Note: There are more accurate numerical methods, like Simpson's Rule.)

a) Write Julia code to calculate energy via numerical integration. Sample code below:

```
t1=  ; t2=  ; Δt=  ;
t=t1:Δt:t2;   # or use t=range(t1, t2, step=Δt)
x = my_function.(t)       # fill an array x with function values
InstPower = (abs.(x)).^2    # calculate instantaneous power
Energy = sum(InstPower)*Δt  # calculate energy by integration
using Plots; plotly();
fig = plot(t,x); display(fig);
fig = plot(t,InstPower); display(fig)
println("Energy = ",Energy);
```

Exercise 1

A different (neater) approach to writing the code is to make use of functions. A function that normally operates on a scaler can operate on a vector (array) if called using Julia's "dot-notation":

```
t1=  ; t2=  ; Δt=  ;
t=t1:Δt:t2;   # or use t=range(t1, t2, step=Δt)
myfunc(t) = ...     # Define a function myfunc(t) for the waveform
# Create a function to calculate the instantaneous power
InstPower(x) = (abs(x))^2    # x is normally a scalar value
# Create a fn to calculate the energy of the sampled signal stored in
array x with sample spacing Δt.
Energy(x,Δt) = sum( InstPower.(x) )*Δt  # Calculate energy by
integration. Note InstPower() must be called using dot-notation because
x is a vector. InstPower.(x) returns a vector.
using Plots; plotly();
x = myfunct.(t)    # Create a vector x of calculated values
fig = plot(t,x); display(fig);
fig = plot(t, InstPower.(x) ); display(fig)
println("Energy = ", Energy(x,Δt) );
```

Exercise 1

b) Use your code to calculate the energy of a sinusoidal pulse of length T=1 second, and frequency f_0=6 Hz, and amplitude A=10.

$$x(t) = A\,\mathrm{rect}\left(\frac{t}{T}\right)\cos(2\pi f_0 t)$$

Also show plots of $x(t)$ vs t, and instantaneous power $p(t)$ vs t.

Note: You can define a rect() function as follows (two different methods are shown):

Method 1: define the function to accept a scalar argument (neater code)

```
rect(t)=(abs(t)<=0.5)*1.0    # Accepts a scaler argument t

# You can however still call the rect() function if t is a vector by using the
dot-notation which tells Julia to apply the rect function to each element in the
vector t.  The syntax is rect.(t)
so  x = A*rect.(t) .* cos.(2*pi*f0*t) will create a vector x of values.
You can also define a neat function which reads like the equation
myfunc(t) = A*rect(t)*cos(2*pi*f0*t)
x = myfunc.(t)   # This will create a vector x of calculated values
```

Method 2: define the function to accept a vector argument (needs "dots" inside the function definition)

```
rect(t)=(abs.(t).<=0.5).*1.0   # Works with a vector argument t
# With this definition, rect(t) will work if t is a vector e.g.:
y = A*rect(t).*cos.(2*pi*f0*t)
```

Exercise 1

c) Calculate the energy of the impulse response of an ideal LPF of unit-gain and bandwidth B=1 Hz.

$$h(t) = 2B\ \mathrm{Sa}(2\pi B t)$$

Note: You can define a function via a one-line statement: Sa(u) = sin(u)/u
which you can call via Sa(t) if t is a scalar, or call via Sa.(t) if t is a vector (array).
Alternatively you can define the function as: Sa(u) = sin.(u)./u
which can be called via Sa(t) where t can be a scalar or a vector.

Note: The sample times are t=t1:Δt:t2. If one of the t-values is exactly zero, then the Sa( ) function with complain because sin(u)/u cannot be evaluated at u=0 (Returns "NaN" = "Not a Number"). So:

- A quick way to get round this is to define the range of "t" to avoid landing a sample on t=0.
- Another option is to define a more complicated multi-line Sa(u) function which checks to see if the argument is "0" and returns the value "1" in such cases.
- Julia also has a built-in function "sinc(u)=sin(πu)/(πu)"; while this is not the same definition as Sa(u)=sin(u)/u, we could use the transform Sa(u) = sinc(u/π).

For accurate results, choose a long enough time interval [t1,t2] and small enough time step Δt.
Adjust spacing Δt and the interval [t1,t2] until the result converges.

Exercise 2 – Impulse response

a) The impulse response $h(t)$ of an ideal LPF of bandwidth B Hz and unity gain is:

$$h(t) = 2B\ \mathrm{Sa}(2\pi B t)$$

Write Julia code to plot the impulse response and instantaneous power (vs time) for case $B = 1$ Hz.

b) Write Julia code to plot the impulse response of an ideal BPF with bandwidth B and centre frequency ω_0. Specify $B = 1$ Hz and centre frequency $f_0 = 4$ Hz. The equation is given below:

$$h(t) = 2B\ \mathrm{Sa}(\pi B t)\cos(\omega_0 t)$$

Exercise 3 - Step response via integration

In the lecture notes, it is shown that the response $y(t)$ of an LTI filter to a unit-step function $u(t)$ can be found by integrating its impulse response $h(t)$:

$$y(t)=h(t)*u(t)=\int_{-\infty}^{\infty} h(\tau)u(t-\tau)d\tau=\int_{-\infty}^{t} h(\tau)d\tau$$

The integral may be evaluated by numerical integration:

$$y(t)=\int_{t_0}^{t} h(\tau)d\tau\approx\sum_{n=0}^{N-1} h\left(t_0+n\Delta\tau\right)\Delta\tau$$

This must be performed for a range of t values. Julia provides a special function "cumsum(X)" which cumulatively sums the values in array X (a column vector), creating an output array.

```
t=t1:Δτ:t2
h = myfunction(t)    # impulse response function
y = cumsum(h)*Δτ    # cumulatively integrate
fig=plot(t,y); display(fig)
```

a) Write Julia code to calculate and plot the step response of an ideal LPF for case B=1 Hz.
The impulse response is $h(t)=2B\ \mathrm{Sa}(2\pi Bt)$

b) By inspection/analysis of the step response, determine the 10% to 90% rise time. Does the value agree with the value stated in the lecture notes? i.e., $t_r = 0.446/B \approx 1/(2B)$.