EEE3092F
Signals and Systems II
2022

A/Prof. A.J. Wilkinson

andrew.wilkinson@uct.ac.za

http://www.ee.uct.ac.za

Department of Electrical Engineering

University of Cape Town

Julia Simulation Exercises relating to Sections 2.2 and 2.3

Do <u>all</u> exercises inside this file.
ie do 2.x.1, 2.x.2, 2.x.3, 2.x.4, 2.x.5

Instructions

- These exercises must be done with Julia in a Jupyter notebook.
- Installation instructions for installing Julia and required libraries are inside:
  Vula → Resources → Assignments → Instruction Sheets → Julia Exercises
  "1_EEE3092F-Julia_Installation.pdf"
- To start Jyputer notebook form the Julia REPL (command line):
  using Ijulia <Enter>
  noteboook() <Enter>
- 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.
- You should put all you assignment exercises into a single Jupyter notebook file named:
  <student ID>_EEE3092F_Assignment1_Exercises_from_Section_2.2_2.3.ipynb
- 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 Vula. (Date to be announced)

Instructions: Load Jupyter Notebook and Sample Code

- Download the sample code files.
  Vula → Resources → Software → Julia-sample-code:
  `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 1.7.2). 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 exercies.

Additional Julia Language Tutorials

- Download the "`JuliaBoxTutorials-master.zip`" file from Vula → Resources → Software → Julia-tutorials-from-Juliabox.

  Unzip, and take a look inside:
  introductory-tutorials → intro-to-julia

  These tutorials must be loaded into Jupyter notebook.

  They contain examples of how to use the Julia language.

Quick help on a function (within Julia)

- To get a short help summary on any Julia function (either at the REPL command line or inside Jupyter notebook) insert a question mark in front of the function and execute.

  For example try:

  ```
  ?println <Enter>

  ?log10 <Enter>

  ?sum <Enter>
  ```

Notes on defining a "range" of time values

In Julia, a range of values to may be defined in several ways:

- `Specify constants t1=…; t2=…; Δt=…`
- `t=t1:Δt:t2    # the last value will be less than t2 if (t2-t1)/Δt is not an integer. To get the number of elements`
  `N = length(t) # t[1] is first value equal to t1. t[N] is the last value.`
- Another equivalent way to create a range is using the range() function:
  ```
  t=range(t1, t2, step=Δt) # same range as above
  ```
- Sometimes one would rather specify the exact number of samples N in the range, and calculate time step afterwards:
  ```
  N=100
  t=range(t1, t2, length=N)     # Note t[1] equals t1 and t[N] equals t2
  Δt = (t2-t1)/(N-1)
  ```
- Defining a "range" as above does not actually fill an array in memory unless you explicitly tell Julia to create the array using the funcion collect() as shown below:
  ```
  t=collect(t1:Δt:t2)
  ```
  Often, one does not need to convert a "range" into a memory array because evaluating a function like cos.(t) with work on both types, creating a memory array.

Julia Exercise 2.x.1 - Energy via integration

The energy in a pulse x(t) is given by $E = \int_{-\infty}^{\infty} |x(t)|^2 dt$

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

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 the 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);
```

Julia Exercise 2.x.1 - continued

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) );
```

Julia Exercise 2.x.1 - continued

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 $x(t)$ and instantaneous power $p(t)$.

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)
```

Julia Exercise 2.x.1 - continued

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

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

Note: You define a function: 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 a gunction 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.` Think of a way to avoid landing a sample on t=0, because sin(u)/u cannot be evaluated at u=0 (Returns "NaN" = "Not a Number").

For accurate results, choose a long enough interval and small enough time step.
Adjust spacing `Δt` and the interval [t1,t2] until result converges.
Note: Julia has a built-in function "sinc(u)=sin(πu)/(πu)" but is not the same definition as Sa(u)=sin(u)/u .

Answers (numerically calculated): b) 50.0 J c) 1.999 J

Julia Exercise 2.x.2 - Plotting filter impulse response

a) The impulse response $h(t)$ of an ideal LPF of bandwidth B Hz and unity gain is (see Drill Problem 2.6):

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

Write Julia code to plot the waveform for case $B = 1$ Hz.

b) By inspection of the waveform (and/or samples displayed) determine the approximate width of the main lobe, measured between the 3dB points.
(it should be approximately $\delta t \approx 0.44/B$)

c) Write Julia code to plot the impulse response of an ideal BPF, of bandwidth B and centre frequency ω_0. Specify $B = 1$ Hz and centre frequency $f_0 = 4$ Hz.
The formula was derived in Drill Problem 2.7

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

Julia Exercise 2.x.3 - Step response via integration

In Section 2.2 of the lecture notes, it is shown that the response $y(t)$ of a 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.

```
julia> cumsum([1,2,3,4])
4-element Array{Int64,1}:
  1
  3
  6
 10
```

```
t=t1:Δτ:t2
h = myfunction(t)    # impulse response function
y = cumsum(h)*Δτ    # cumulatively integrate
fig=plot(t,h); display(fig); 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 Section 2.2. of the lecture notes? i.e. $t_r = 0.446/B \approx 1/(2B)$.

Julia Exercise 2.x.4 - Plotting Periodic functions

You may recall Drill Problem 2.12 which involved determining the period of the function

$$v(t) = 4\cos(20\pi t) + 2\cos(30\pi t)$$

Write Julia code to:

a) plot the function $4\cos(20\pi t)$ as a function of t over a specified interval from t_1 to t_2 with a specified time step Δt.

b) plot the function $2\cos(30\pi t)$ over the same interval

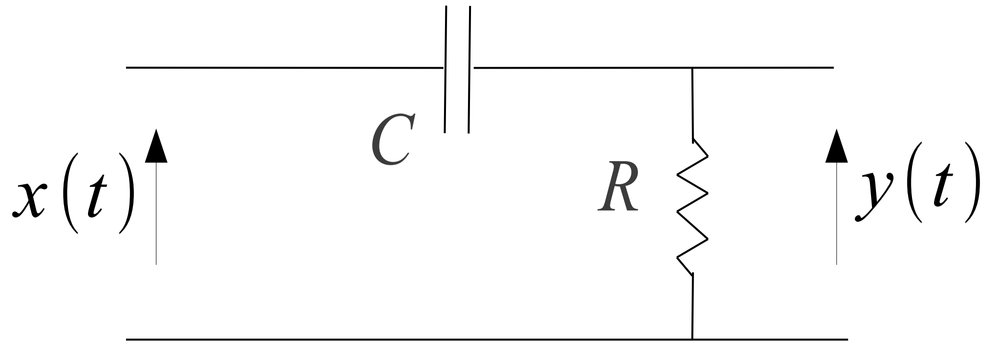
c) plot $v(t) = 4\cos(20\pi t) + 2\cos(30\pi t)$ over the same interval

d) By inspection of your plot, determine the period of the waveform $v(t)$ in seconds, and use it to calculate the fundamental frequency. Compare your answer with the value determined in Drill Problem 2.12

Julia Exercise 2.x.5 – Plotting Magnitude and Phase

You may recall Drill Problem 2.5 involved deriving the transfer function for an RC high pass filter:

$$H(\omega) = \frac{Y(\omega)}{X(\omega)} = \frac{j\omega RC}{1 + j\omega RC}$$



The breakpoint frequency is at $\omega_c = 1/(RC)$

Write some Julia code to plot the transfer function. Choose suitable parameters and a suitable plotting range, and show plots (label axes and give title) for:

a) Plot the magnitude using function call `abs.()`

b) Plot the phase using function `angle.()`

c) Plot the real part using function `real()`

d) Plot the imaginary part using function `imag()`

e) Determine $|H(\omega_c)|$ and calculate the ratio $|H(\text{inf})|/|H(\omega_c)|$ where $H(\text{inf}) = \lim_{\omega \to \infty} H(\omega)$ also convert the ratio to dB. For this you will need the `log10()` function.

EEE3092F
Signals and Systems II

End of handout