Signals and Systems Project 1: Convolution

Author: Junwoo Hwang (2019170904)

About

This Jupyter Notebook details the code used for implementing the convolution specified in project document, and serves as a final submission file with answers to 3 different questions embedded in the sections organized below (each denoted with "Question X" style nomenclature).

It was a nice opportunity to get hands-on with the concept of convolution & learn the pitfalls & challenges of making it computationally effective, and gave me various foods for thought that I am still pondering about!

Signals definition

In [ ]:

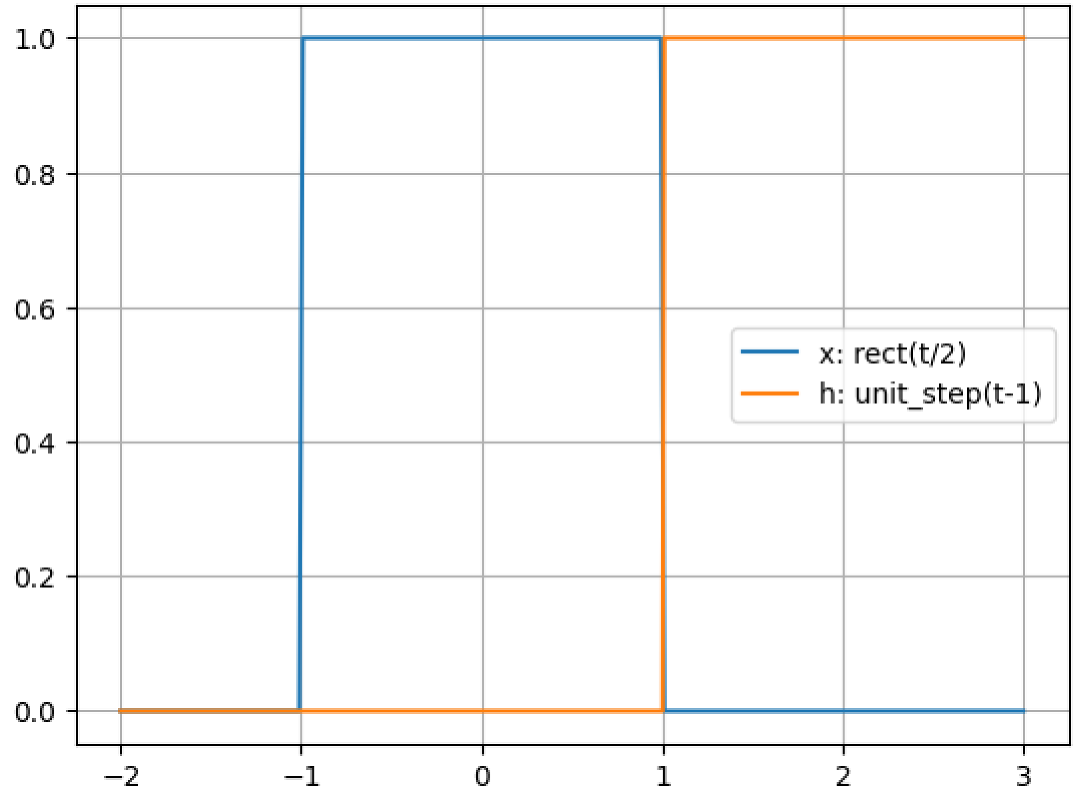
```
# Unit rect (rect)
def unit_rect(x):
  if abs(x) > 0.5:
    return 0.0
  elif abs(x) == 0.5:
    return 0.5
  else:
    return 1.0

# Half rect (rect(x/2))
def rect_x_2(x):
  return(unit_rect(x/2.0))

# Unit step function
def unit_step(x):
  if x > 0.0:
    return 1.0
  else:
    return 0.0

# Unit step with offset 1 (u(x-1))
def unit_step_1(x):
  return unit_step(x-1)
```

Draw the functions (Question 1)

In [ ]:

```
# Draw the functions
import matplotlib.pyplot as plt

x_range = [x/100.0 for x in range(-200, 301)]
```

```
# Our x and h functions
x_func = rect_x_2
h_func = unit_step_1

# Draw those functions
x_values = list(map(x_func, x_range))
h_values = list(map(h_func, x_range))

plt.plot(x_range, x_values, label='x: rect(t/2)')
plt.plot(x_range, h_values, label='h: unit_step(t-1)')
plt.legend()
plt.grid()
```



Define Convolution

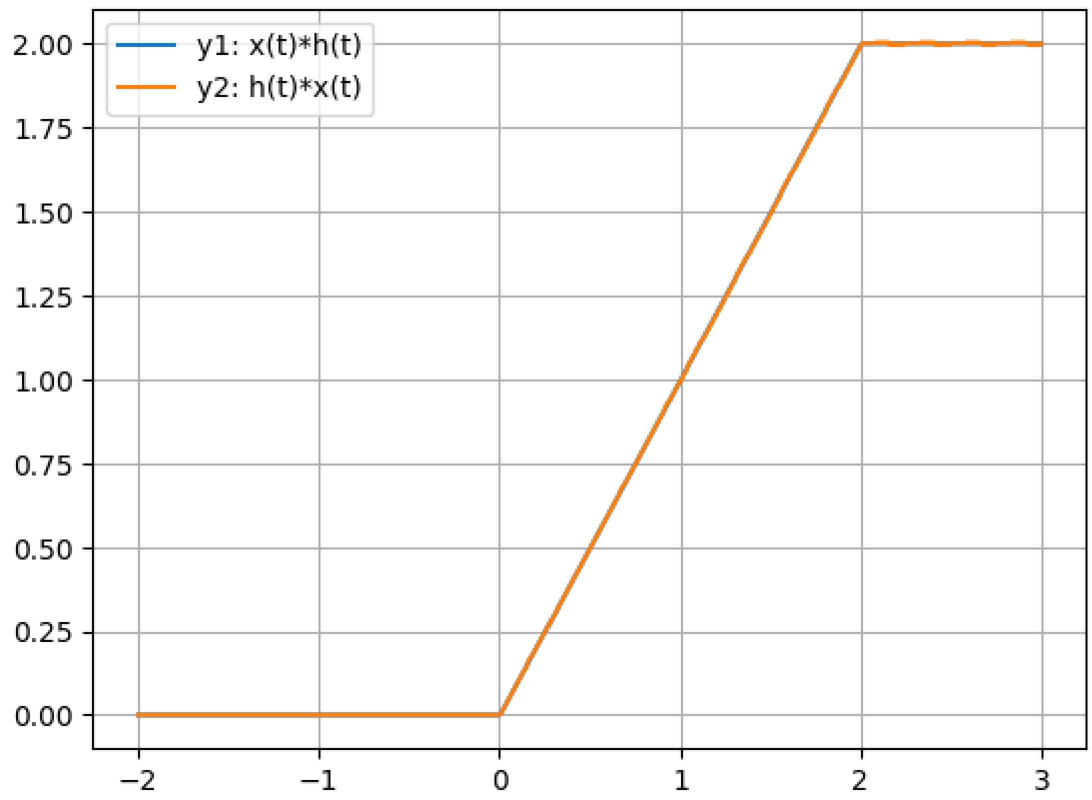
In [ ]:

```
# Default range to include x values that results in non-zero value of the
RANGE_DEFAULT = [-100.0, 100.0]

def convolve(f, g, x_range, dx = 0.01, f_nonzero_range = RANGE_DEFAULT, g
  '''
  Convolve two functions (f, g) and output the result in specified x_rang
  Result will be a list of numbers the length equal to x_range itself.

  dx = granuality of the variable x for the integration itself. By defaul
  is in a sane range of 0.01, assuming that functions are mostly rectangu
  unit step, etc, which doesn't change much within the range of dx, excep
  at non-continuous boundaries

  NOTE:
  To make computation more efficient, we can optionally supply the nonzer
  functions (f, g) being convoluted. This provides a sensible `tau` range
  since without this limit, it can literally range from -inf to +inf, whi
  computation (since it's not symbolic math at this point).
```

```
  But to remove the need to integrate -inf to +inf, the arbitrary default
  '''
  # Uncomment to enable debugging
  # DEBUG = True

  result = [0] * len(x_range)

  for x_idx in range(len(x_range)):
    t = x_range[x_idx]

    # tau range that results in non-zero value of f
    f_range = f_nonzero_range

    # tau range that results in non-zero (significant) value of g
    g_range = [t-g_nonzero_range[1], t-g_nonzero_range[0]]

    # Common tau range that results in both f and g resulting in non-zero
    tau_range = [max(f_range[0], g_range[0]), min(f_range[1], g_range[1])

    # if DEBUG:
    #   print('x_range[{}]={} | Tau_range={}'.format(x_idx, x_range[x_idx

    if tau_range[0] >= tau_range[1]:
      # No valid tau range avilable (integral is 0)
      result[x_idx] = 0.0
    else:
      # Integral of f(tau) * g(t - tau). Discretized by unit of `dx`
      result[x_idx] = 0.0
      for tau_idx in range(int(tau_range[0]/dx), int(tau_range[1]/dx) + 1
        tau = tau_idx * dx
        result[x_idx] += (f(tau) * g(t - tau)) * dx

  return result
```

Execute Convolution (Question 2)

In [ ]:
```
# Convolution test
import matplotlib.pyplot as plt

x_range = [x/100.0 for x in range(-200, 301)]

# Our x and h functions
x_func = rect_x_2
h_func = unit_step_1

# Convolve
y_one_values = convolve(x_func, h_func, x_range)
y_two_values = convolve(h_func, x_func, x_range)

# Plot
plt.plot(x_range, y_one_values, label='y1: x(t)*h(t)')
plt.plot(x_range, y_two_values, label='y2: h(t)*x(t)')

plt.legend()
plt.grid()
```



Is Y_1(t) and Y_2(t) the same? (Question 3)

It is. Because convolution by definition is commutative, meaning the order of the functions for the arguments doesn't matter: <f, g> = <g, f>.

Thought Experiment

This can be proven simply with a thought experiment: When we integrate f(tau) and g(t-tau), we are essentially integrating every single pair of t that sums up to the specified value `t`.

And it should be obvious that it doesn't matter whether we have f in the first or the second, as the resulting pair of t would stay the same for both f and g.

Also, put it simply, every single pair of f(`tau`) and g(`t - tau`) equals to: f(`t - (t - tau)`) and g(`(t - tau)`), which means that regardless of the order of f and g function, the resulting pair matches 1 to 1 by just modulating the value of `tau`.

Resources

1. https://scicoding.com/convolution-in-python-3-essential-packages/
2. https://en.wikipedia.org/wiki/Convolution
3. https://en.wikipedia.org/wiki/Heaviside_step_function
4. https://en.wikipedia.org/wiki/Rectangular_function