

Experiment 5: Representation of signals and computing the signal energy part2

1. Shifting and scaling operations.

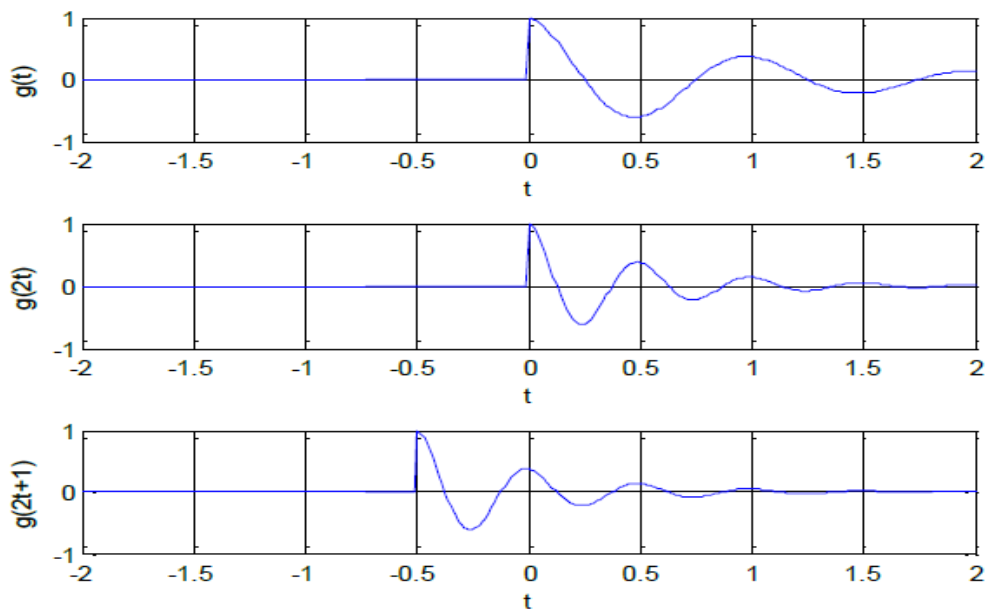
Consider

$$g(t) = e^{-t} \cos(2\pi t) u(t)$$

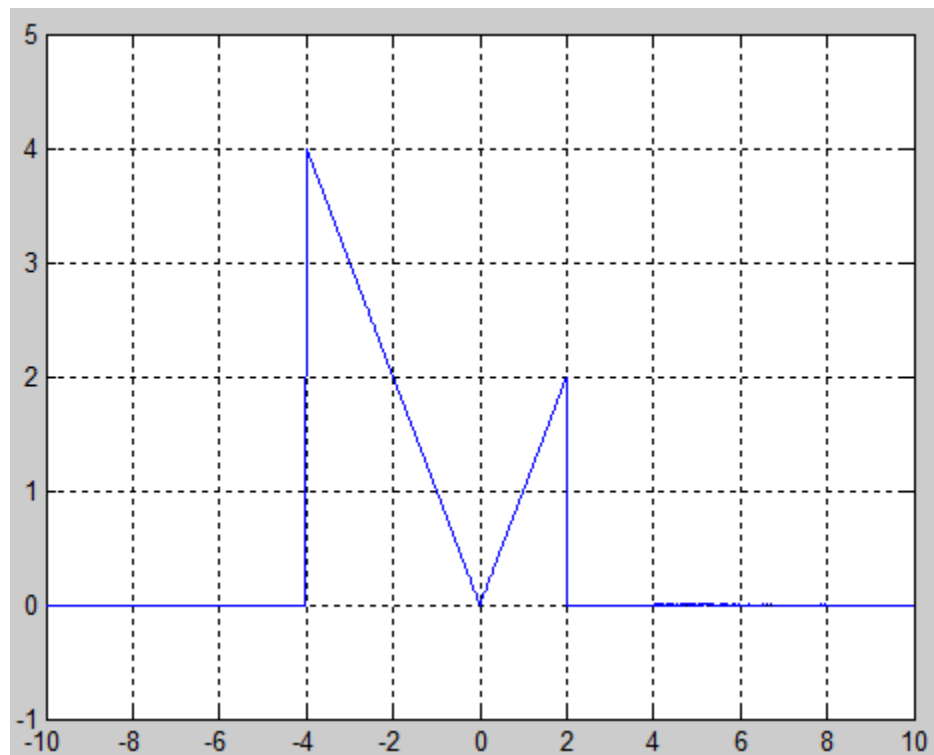
Unfortunately MATLAB cannot multiply inline objects. That is MATLAB reports an error for $g=f*u$ when f and u are inline objects, Rather $g(t)$ needs to be explicitly defined.

```
>> g = inline('exp(-t).*cos(2*pi*t).*(t>=0)','t')
g =
  Inline function:
  g(t) = exp(-t).*cos(2*pi*t).*(t>=0)
```

```
t = (-2:0.01:2);
subplot(3,1,1)
plot(t,g(t)); xlabel('t'); ylabel('g(t)'); grid;
subplot(3,1,2)
plot(t,g(2*t)); xlabel('t'); ylabel('g(2t)'); grid;
subplot(3,1,3)
plot(t,g(2*t+1)); xlabel('t'); ylabel('g(2t+1)'); grid;
```



- Consider the signal $x(t)$ as shown in figure:



Find, a) $x(t-4)$

b) $x(t/1.5)$

c) $x(2t-4)$

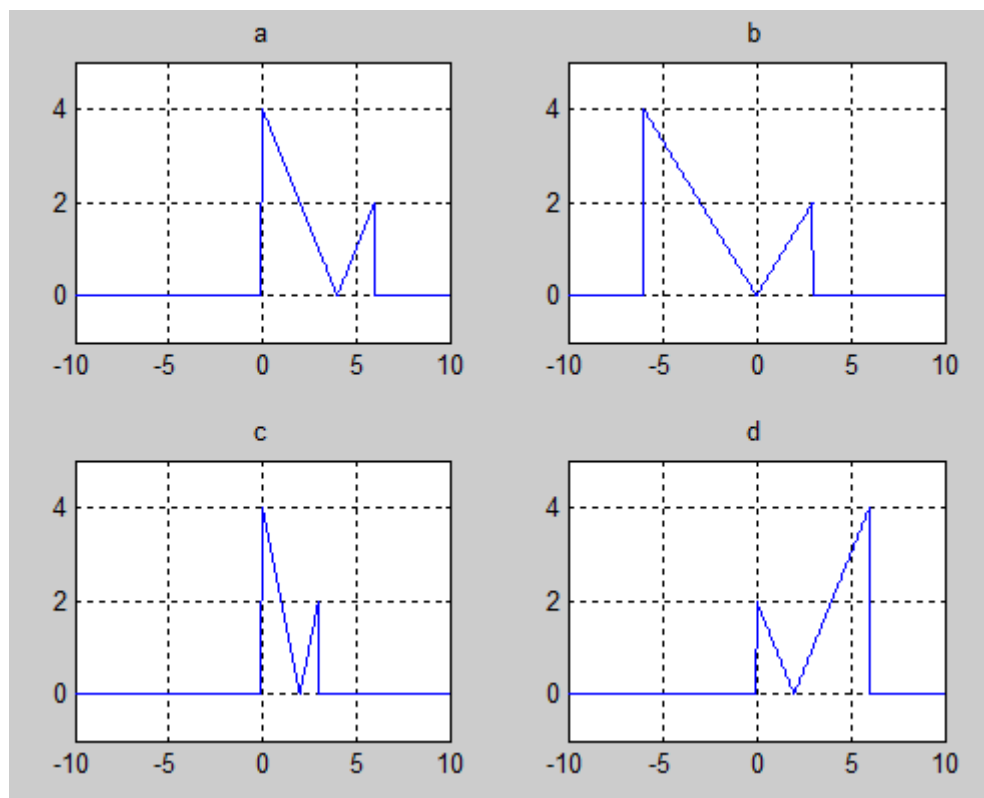
d) $x(-t+2)$

```
t = -10 : 0.01 : 10;
x = inline('-poslin(t+4) + 4*hardlim(t+4) + 2*poslin(t) -  
poslin(t-2) -2*hardlim(t-2)', 't');
subplot 221
plot(t,x(t-4))
grid;
axis([-10 10 -1 5]);
title('a')
subplot 222
plot(t,x(t/1.5))
grid;
```

```

axis([-10 10 -1 5]);
title('b')
subplot 223
plot(t,x(2*t-4))
grid;
axis([-10 10 -1 5]);
title('c')
subplot 224
plot(t,x(-t+2))
grid;
axis([-10 10 -1 5]);
title('d')

```



2. Numerical Integration and Estimating Signal Energy

Interesting signals often have nontrivial mathematical representation, Computing signal energy ,which involve integrating the square of these expressions , can be a daunting task.

Symbolic Integration in MATLAB

Evaluate the integral $\int_1^2 x \cos(x) dx$

```
>>int('x*cos(x)',1,2)
ans =
cos(2)+2*sin(2)-cos(1)-sin(1)
>>eval(ans)
ans =
0.0207
```

Notice that since MATLAB is working symbolically here the answer it gives is in terms of the sine and cosine of 1 and 2 radians. In order to force MATLAB to evaluate this, we have to use the *eval* command. ?

For many functions, the anti-derivative cannot be written down in a closed form (as the sum of a finite number of terms), and so the *int* command cannot give a result. As an example, the function

$$f(x) = e^{-x^2}$$

falls into this category of functions. If we try *int* on this function, we get:

```
int('exp(-x^2)')
ans =
1/2*pi^(1/2)*erf(x)
```

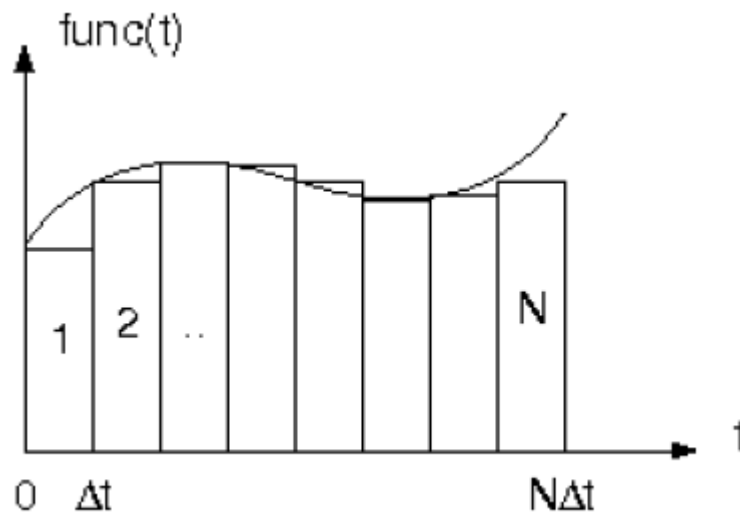
where by *erf(x)* MATLAB is referring to the function

$$\text{erf}(x) = \frac{2}{\pi} \int_0^x e^{-y^2} dy$$

which is to say, MATLAB hasn't actually told us anything. In cases like this, we can proceed by evaluating the integral numerically.

Numerical Integration in MATLAB

Evaluate the functions which we want to integrate at points uniformly separated by Δt , multiply by Δt to compute rectangle areas, and then sum over all rectangles.



Example

Compute the energy of

$$x(t) = e^{-t}(u(t) - u(t - 1))$$
$$E_x = \int_{-\infty}^{\infty} |x(t)|^2 dt = \int_0^1 e^{-2t} dt = .4323$$

```
>> x = inline('exp(-t).*((t>=0)&(t<1))','t');  
t = (0:0.01:1);  
E_x = sum(x(t).*x(t)*0.01)  
E_x =  
0.4367
```

The result is not perfect, but at 1% relative error it is close. By reducing Δt the approximation is improved. For example, $\Delta t = 0.001$ yields $E_x = 0.4328$ or .1% relative error.

Although simple to visualize, rectangular approximation is not the best numerical integration technique. The MATLAB function `quad` implements a better numerical integration technique called recursive adaptive Simpson quadrature.

```
x_squared = inline('exp(-2*t).*((t>=0)&(t<1))','t');  
E_x = quad(x_squared,0,1)  
E_x =  
0.4323
```

In this case the relative error is -.0026%