ESE2014 Week4

1) provide three different functions that satisfy the integral property of the continuous-time delta-function.

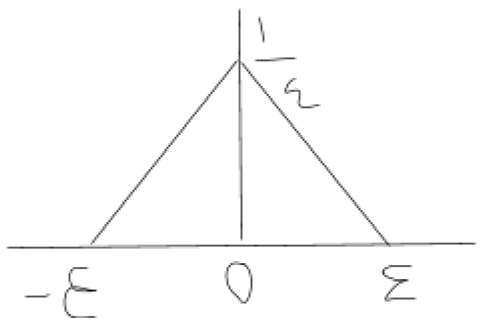
The integral property of the Dirac-delta function:

$$\int_{-\infty}^{\infty} \delta(t)\, dt = 1$$

We know that the (0 to infinity) line at t=0 has an area of 1.

So, any other function which has an area of 1 satisfies the integral property of the continuous-time delta-function.

Function 1:

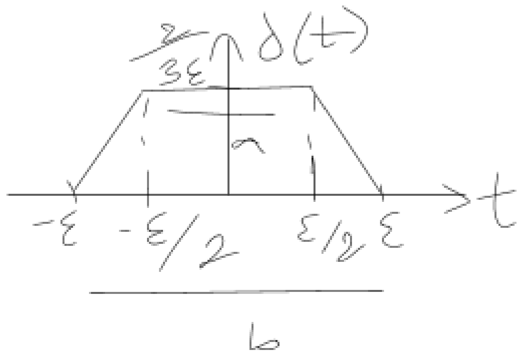


Area = ½(length)(height)

= ½(2ε)(1/ε)

= 1

Function 2:



Area = (ab)/2 x h

= (ε + 2ε)/2 x 2/3ε

= 3ε/2 x 3ε/2

= 1

Function 3:

Area = ½(πab)

=π/2(1/πε)x(2ε)

= 1

ESE2014 Week4

2) show that the unit step sequence can be expressed using the unit sample sequence using the expression shown in the notes (as an infinite sum)

Unit sample sequence / Dirac-delta

$$\delta(n) = \begin{cases} 0 & ; n \neq 0 \\ 1 & ; n = 0 \end{cases}$$

Any sequence can be expressed using delta function as:-

$$x(n) = \sum_{t=0}^{\infty} x(t)\,\delta(n-t)$$

Unit step sequence

$$u(n) = \begin{cases} 1 & ; n \geq 0 \\ 0 & ; n < 0 \end{cases}$$

So, unit step sequence is an impulse sequence from 0 to ∞.

We can use dirac-delta to express this as:-

$$u(n) = \delta(n) + \delta(n-1) + \delta(n-2) + \ldots \text{so on}$$

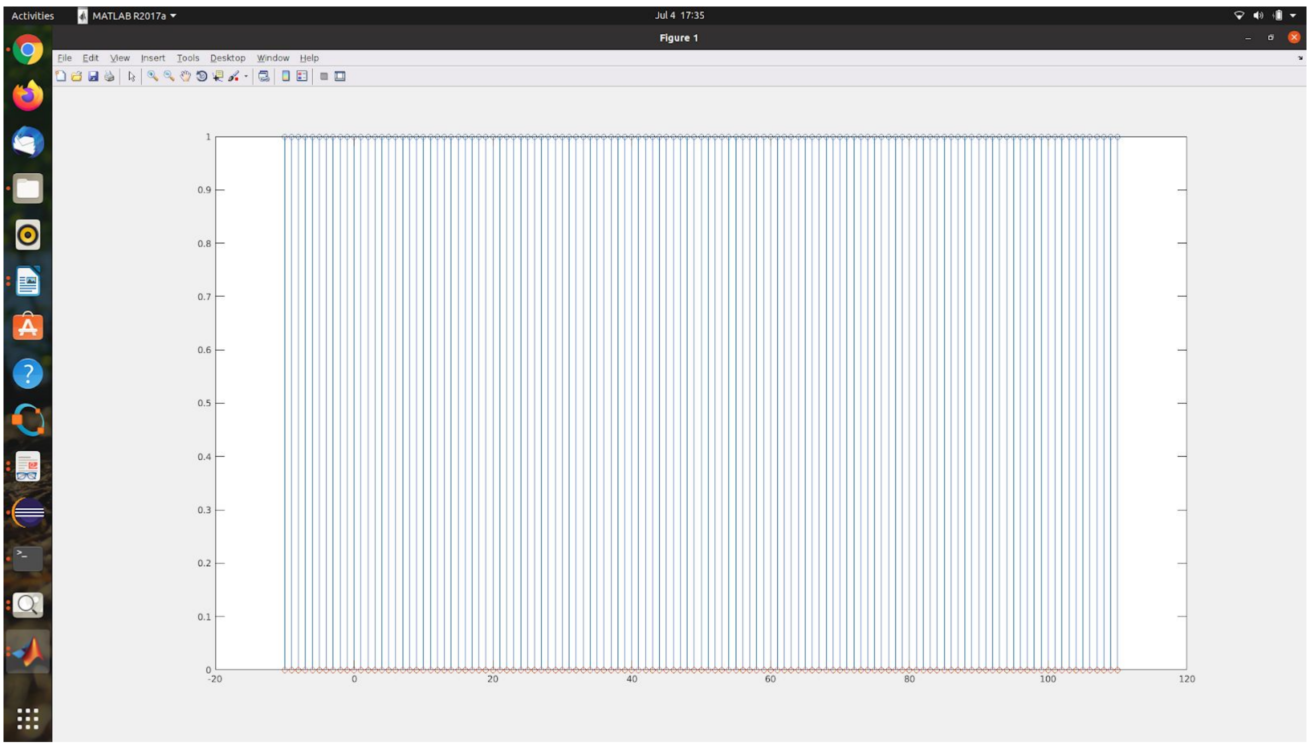
$$\Rightarrow u(n) = \sum_{t=0}^{\infty} \delta(n-t)$$

ESE2014 Week4

We can also prove this in Matlab:

```
x = -10:110;
y = zeros(121);

for n = -10:110
y(n+11) = dirac(0);
end
idx = y == Inf; % find Inf
y(idx) = 1;      % set Inf to finite value
stem(x,y);
```

Result: As we can see we can plot a finite unit step function using 120 Dirac-delta functions.



3) Do continuous-time signals possess the "periodic in frequency" property of discrete-time signals? Why or why not?

ESE2014 Week4

A continous signal $x(t)$ is said to be periodic if $x(t) = x(t \pm nT)$; $n \in Z$, $T \rightarrow$

Consider a complex exponential function.

$$x(t) = A e^{j(\omega_0 + 2\pi r)t} \quad \forall\ t \in \text{real no.}$$

$$\forall\ r \in Z$$

$$= A e^{j\omega_0 t} \cdot e^{j2\pi r t}$$

$$x(t) = A e^{j\omega_0 t} \underbrace{\left[\cos(2\pi r t) + j \sin(2\pi r t)\right]}_{\neq 1}$$

$$\therefore\ A e^{j(\omega_0 + 2\pi r)t} \neq A e^{j\omega_0 t}$$

Therefore, this function is not periodic in frequency.

4) P2.9 and P2.10 from the course textbook

P2.9) Using the conv_m function, determine the autocorrelation sequence r xx () and the cross-correlation sequence r xy () for the following sequences:

$x(n) = (0.9)^n$ $\quad 0 \le n \le 20$;

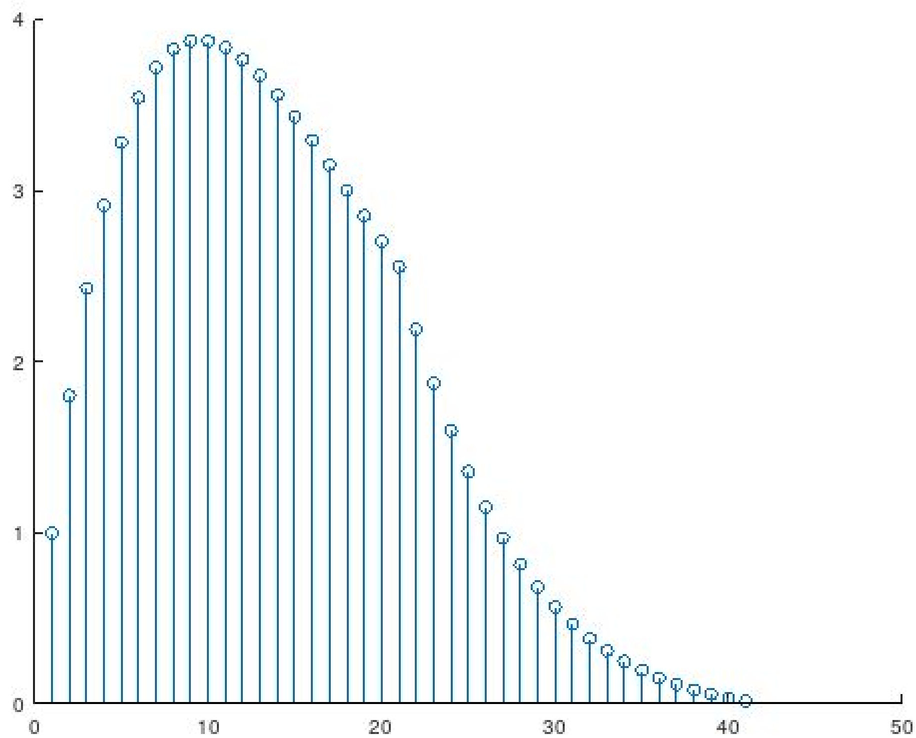
$y(n) = (0.8)^{-n}$ $\quad -20 \le n \le 0$

Describe your observations of these results.

```
x0=0:20
for n1 = 0:20
x(n1+1)= 0.9.^n1;
end;

yt= -20:0
for n2 = 0:20
```

```
y(n2+1)= 0.8^(-1*n2);
end;

%Since conv(f(t),g(t)) = cross correlation (f(t),g(-t))
cross = conv(x,y)
%stem(cross)

autocorrX = conv(x,x)
stem(autocorrX)
```

Result:



P2.10)In a certain concert hall, echoes of the original audio signal x(n) are generated due to the reflections at the walls and ceiling. The audio signal experienced by the listener y(n) is a combination of x(n) and its echoes. Let

$$y(n) = x(n) + \alpha x(n - k)$$

where k is the amount of delay in samples and α is its relative strength. We want to estimate the delay using the correlation analysis.

ESE2014 Week4

1. Determine analytically the cross-correlation r yx () in terms of the autocorrelation r xx ().

$$r_{yx}(l) = y(l) * x(-l)$$

$$= [x(l) + \alpha x(l-k)] * x(-l)$$

$$= x(l) * x(-l) + \alpha x(l-k) * x(-l)$$

$$r_{yx}(l) = r_{xx}(l) + \alpha r_{xx}(l-k)$$

w.k.t

$$r_{xx} = \sum_{n=-\infty}^{\infty} x(n)x(n-l)$$

$$x(l-k) * x(-l) = \sum_{n} x(n-k)x(n-l)$$

let $m = n-k$

$$\Rightarrow n = m+k$$

$$\therefore x(l-k) * x(-l) = \sum_{m=-\infty}^{\infty} x(m) * x(m-(l+k))$$

$$= r_{xx}(l-k)$$

5) Build and execute the simple_1.c program that uses the GSL. Explain the program and provide your output.

Source code:

```
/*
 * simplegsl.c
 *
 *  Created on: Jul. 3, 2020
 *      Author: Takis
 *      Revised on: Jul. 16, 2020
 *      Revised by: Shreya
 */
```

```c
/*
 * standard includes
 */
	#include <stdio.h>
	#include <stdlib.h>
/*
 *  includes for GSL components
 *    - use double precision
 */
	#include <gsl/gsl_vector_double.h>
	#include <gsl/gsl_matrix_double.h>
	#include <gsl/gsl_rng.h>
	#include <gsl/gsl_randist.h>

/*
 * FUNCTIONS
 */

/*
 * simple Fibonacci sequence generator function, using recursion
 * */
size_t fib(size_t k)
{
	if (k==0)
	{
		return 0;
	}
	else if (k==1)
	{
		return 1;
	}
	else /* k >= 2 */
	{
		return fib(k-1) + fib(k-2);
	}
}

gsl_matrix *embt_mm(const gsl_matrix *U, const gsl_matrix *V, size_t N)
{
	gsl_matrix *W = gsl_matrix_alloc(N,N);
	double dp;
```

```
	double uk,vk;

	for (size_t i=0; i != N; ++i)
	{
		for (size_t j=0; j != N; ++j)
		{
			/* compute element (i,j) of W */
			dp = 0;
			for (size_t k=0; k != N; ++k)
			{
				uk = gsl_matrix_get(U,i,k);
				vk = gsl_matrix_get(V,k,j);
				dp += uk*vk;
			}
			gsl_matrix_set(W,i,j,dp);
		}
	}
	return W;
}

void embt_print_vector(const gsl_vector *q)
{
	int N=5;
	for (size_t i=0; i!=N; ++i)
	{
		printf("q(i)=%f\t",gsl_vector_get(q, i));
	}
	printf("\n");
}

void embt_print_matrix(const gsl_matrix *q)
{
	int N=5;

	for (size_t i=0; i!=N; ++i)
	{
		for (size_t j=0; j!=N;++j)
		{
			printf ("q(%zu,%zu) = %f\n", i, j, gsl_matrix_get (q,i,j));
		}
	}
	printf("\n");
```

```
}

/*
 * Main Code
 */

int main()
{
	/*
	 * INITIALIZE PARAMETERS
	 */
		/* vectors parameters */
		size_t      N=5; /* index type, vector sizes */
		gsl_vector *a = gsl_vector_alloc(N); /* allocate vector from heap of size N */
		gsl_vector *b = gsl_vector_alloc(N); /* allocate vector from heap of size N */
		gsl_vector *c = gsl_vector_calloc(N); /* alloc vect of size N but initialize
entries to 0 */

		/* random number generator parameters */
		const gsl_rng_type *T;
		gsl_rng *r; /* handle for our random number generator */

		/* matrix parameters */
		gsl_matrix *A = gsl_matrix_alloc(N,N);
		gsl_matrix *B = gsl_matrix_alloc(N,N);
		gsl_matrix *C = gsl_matrix_calloc(N,N);

	/*
	 * SET UP RANDOM NUMBER GENERATION
	 */
		gsl_rng_env_setup();
		T = gsl_rng_default;
		r = gsl_rng_alloc(T);

	/*
	 *  VECTOR OPERATIONS
	 */
		/* set the vector elements */
		for (size_t i = 0; i != N; ++i)
		{
		gsl_vector_set(a, i, fib(i)); /* set element i of vector a to Fibonacci number i
*/
```

```
            gsl_vector_set(b, i, gsl_ran_flat(r,-1.0,+1.0)); /*set element of vector b to
random no. */
            }

            /* c = a + b */
            gsl_vector_add(c, a); /* c += a */
            gsl_vector_add(c, b); /* c += b */

            /* print results */
            for (size_t i =0; i<N; ++i)
            {
                  printf("i=%zu, a(i)=%f, b(i)=%f, c(i)=%f\n", i,
                  gsl_vector_get(a, i),
                  gsl_vector_get(b, i),
                  gsl_vector_get(c, i));
            }

      /*
       *    MATRIX OPERATIONS - your homework!! :)
       */
/* fill A with first N*N Fibonacci numbers, starting with row 1 (cols 1-10), then row 2,
etc. */
            for (size_t i=0; i != N; ++i)
            {
                  for (size_t j = 0; j != N; ++j)
                  {
                        gsl_matrix_set(A, i, j, (double) fib(j+i*N));
                  }
            }

      /* fill B with N*N random numbers, uniformly distributed over the interval (-100, 100)
*/
            for (size_t i=0; i != N; ++i)
            {
                  for (size_t j = 0; j != N; ++j)
                  {
                        gsl_matrix_set(B, i, j, gsl_ran_flat(r,-100.0,+100.0));
                  }
            }

            /* make C the product of A and B */
            C = embt_mm(A,B,N);
```

```
        /* print the results */
        printf("\nembt_print_vector(a)\n");
        embt_print_vector(a);
        printf("embt_print_vector(b)\n");
        embt_print_vector(b);
        printf("embt_print_vector(c)\n");
        embt_print_vector(c);
        printf("\nembt_print_matrix(A)\n");
        embt_print_matrix(A);
        printf("embt_print_matrix(B)\n");
        embt_print_matrix(B);
        printf("embt_print_matrix(C)\n");
        embt_print_matrix(C);

    /* de-allocate memory */
    gsl_vector_free(a);
    gsl_vector_free(b);
    gsl_vector_free(c);
    gsl_matrix_free(A);
    gsl_matrix_free(B);
    gsl_matrix_free(C);
    return EXIT_SUCCESS;
}
```

ESE2014 Week4

Result:

```
shreya@ShreyasPC: ~/eclipse-workspace/SimpleGSL

embt_print_vector(a)
q(i)=0.000000   q(i)=1.000000   q(i)=1.000000   q(i)=2.000000   q(i)=3.000000
embt_print_vector(b)
q(i)=0.999483   q(i)=-0.674180  q(i)=-0.434764  q(i)=0.894402   q(i)=-0.536687
embt_print_vector(c)
q(i)=0.999483   q(i)=0.325820   q(i)=0.565236   q(i)=2.894402   q(i)=2.463313

embt_print_matrix(A)
```

```
embt_print_matrix(A)
q(0,0) = 0.000000
q(0,1) = 1.000000
q(0,2) = 1.000000
q(0,3) = 2.000000
q(0,4) = 3.000000
q(1,0) = 5.000000
q(1,1) = 8.000000
q(1,2) = 13.000000
q(1,3) = 21.000000
q(1,4) = 34.000000
q(2,0) = 55.000000
q(2,1) = 89.000000
q(2,2) = 144.000000
q(2,3) = 233.000000
q(2,4) = 377.000000
q(3,0) = 610.000000
q(3,1) = 987.000000
q(3,2) = 1597.000000
q(3,3) = 2584.000000
q(3,4) = 4181.000000
q(4,0) = 6765.000000
q(4,1) = 10946.000000
q(4,2) = 17711.000000
q(4,3) = 28657.000000
q(4,4) = 46368.000000
```

```
embt_print_matrix(B)
q(0,0) = -3.005277
q(0,1) = 91.495391
q(0,2) = 48.861069
q(0,3) = 8.008732
q(0,4) = 47.990596
q(1,0) = 51.988760
q(1,1) = 31.727323
q(1,2) = -36.872476
q(1,3) = 60.880603
q(1,4) = 3.934423
q(2,0) = -66.285516
q(2,1) = -4.894054
q(2,2) = -21.537201
q(2,3) = -55.666463
q(2,4) = -57.361908
q(3,0) = -93.932959
q(3,1) = -33.292150
q(3,2) = -61.170230
q(3,3) = 88.743356
q(3,4) = 15.986335
q(4,0) = 79.660972
q(4,1) = 33.112786
q(4,2) = -0.277938
q(4,3) = 12.125651
q(4,4) = -63.543071
```

```
embt_print_matrix(C)
q(0,0) = 36.820240
q(0,1) = 59.587328
q(0,2) = -181.583951
q(0,3) = 219.077807
q(0,4) = -212.084028
q(1,0) = 275.052879
q(1,1) = 1074.372422
q(1,2) = -1624.682799
q(1,3) = 2079.307097
q(1,4) = -2299.027817
q(2,0) = 3062.401908
q(2,1) = 11877.683967
q(2,2) = -18053.094737
q(2,3) = 23091.455872
q(2,4) = -25501.390012
q(3,0) = 33961.473871
q(3,1) = 131728.896055
q(3,2) = -200208.724906
q(3,3) = 256085.321690
q(3,4) = -282814.317946
q(4,0) = 376638.614493
q(4,1) = 1460895.540571
q(4,2) = -2220349.068700
q(4,3) = 2840029.994459
q(4,4) = -3136458.887423

shreya@ShreyasPC:~/eclipse-workspac
```