Изследване на дискретни във времето сигнали, ЦАП през LPT порт

Деян Левски
deyan.levski@eng.ox.ac.uk

2 април 2015 г.

1 Въведение

Целта на първата част от заниманието ни ще бъде да изследваме ефектите от дължината на бинарната дума върху шумовете, внасяни от един дискретизатор.

След като сме се убедили в истинността на теорията, през втората част от упражнението ни ще се опитаме да преобразуваме дискретите на една синусоида (която генерирахме по време на част първа) в аналогов сигнал посредством R2R ЦАП и паралелен порт на стандартен компютър.

Последно, освен синусоида, ще разгледаме възможностите за подаване на дискрети с друга информация (музика) с цел възпроизвеждането ѝ през високоговорител.

2 Част първа

Целта на заниманията ни през първа част е да построим идеален дискретизатор и подадем продължителни във времето сигнали на неговите входове. Matlab е изключително лек и интуитивен за работа език, което го прави идеален за целите ни в този случай. Тъй като Matlab не е безплатен софтуер, по време на упражнението ни ще използваме octave, който се явява почти пълен еквивалент.

2.1 Първо подзадание

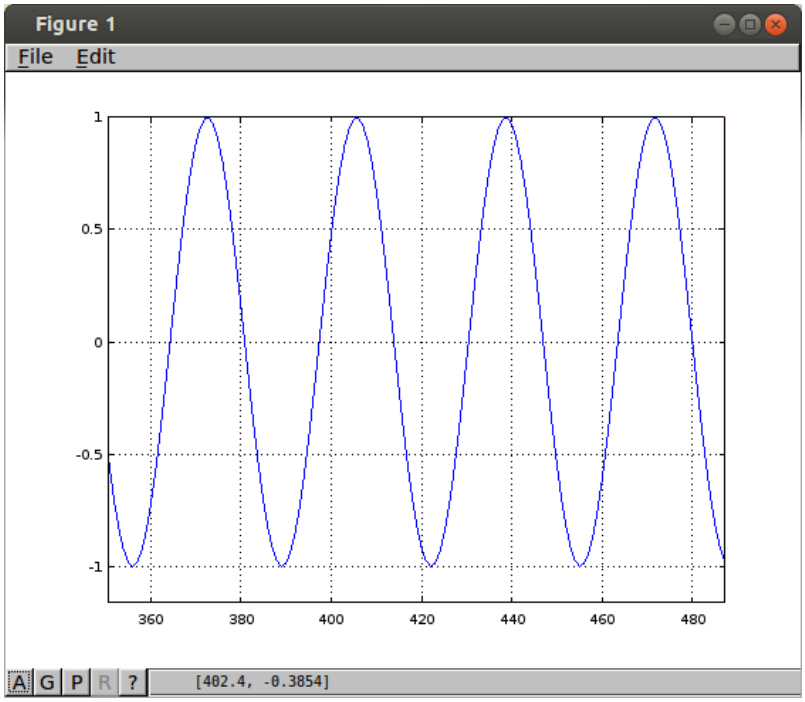
Като начало ни е необходим аналогов продължителен във времето сигнал. След това ще подадем този сигнал на идеалния дискретизатор, който ще построим. Това може да бъде сигнал с всякаква форма. Като за начало построете вектор X(n), съдържащ рамп сигнал (линейно нарастващ) с дължина 1024 дискрета. Към този момент амплитудата не е от значение, но за всеки случай е хубаво (и лесно) да вмъкнем един скалиращ коефициент. Ето и малка подсказка от къде да започнем:

```
clc;
clear all;
close all;

n = 1024; % Дължина на рамп сигнал
k = 1 % Скалиращ коефициент

y =   % Нека да генерираме сигналите тук

plot(y);
```

След като видяхме, че с octave не е никак трудно да създадем рамп вектор, нека да генерираме и една чиста синусоида. Същата ще ни е необходима през втората част от упражнението, където ще се опитаме да я възпроизведем на високоговорител. Нужно е да знаем, че дори и съвременните компютри да ни предлагат цифрова аритметика с 1.8e10 броя стъпки (64 бита), винаги ще имаме някаква определена дискретност. Не можем да генерираме идеален аналагов сигнал, но при такава финност можем просто да се абстрахираме от този факт за амплитудата на сигнала. Трудно е, обаче, да имаме толкова голям брой дискрети, тъй като те са свързани с оперативната памет на компютъра, която имаме. Нека и този път да ограничим дължината на сигнала до n=1024 и семплираща честота от 1MHz. Отново, за да спестим време ето малка подсказка:

```
x = sin(2*pi*f*(0:n)/fs)
```

Като fs в този случай се явява нашата семплираща честота, която е уместно да изберем доста по-голяма от честотата ни f, спомнете си теоремата на Найкуист. В крайна сметка трябва да получим нещо подобно на това:



2.2 Второ подзадание

След като вече имаме набор от синусоида и рамп сигнал, сега можем да пристъпим към създаването на дискретизатор. За по-лесно, нека да изберем референтна стойност с размер 1024 и нека като за начало да направим 3-битов дискретизатор. Припомням, стойността на един най-незначим бит LSB (least significant bit) е

$$LSB = \frac{V_{ref}}{2^N}$$

Малко помощ: в octave можем да генерираме вектор от всички кодове. Например:

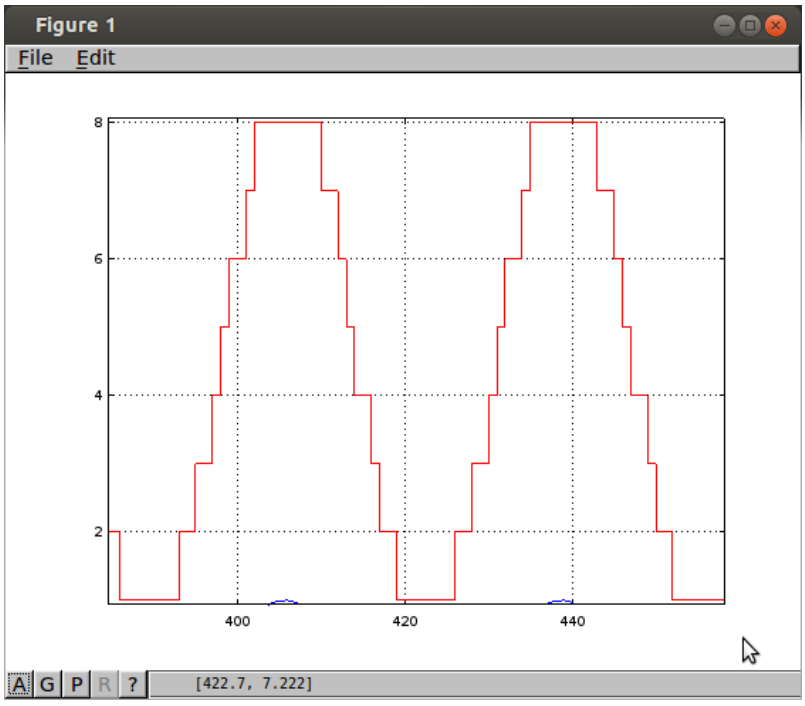
```
N = 2; % разрядност на дискретизатора
ref=[-(2^N):((2^N)-1)]/(2^N);
```

Има и много други методи и начини за съставяне на дискретизатор (Q). Оставям останалото на въображението ви. След като сме готови,

трябва да подадем сигналите, които генерирахме в първо подзадание към Q както и да изобразим неговия изход. За целта, освен функцията plot() доста удобна функция се явява stairs(). За повече информация относно функциите в octave:

```
help plot
help stairs

help funkciyataZaKoyatoNeZnam
```

Както и очакваме, дискретизираната синусоида след изобразяването ѝ с функцията stairs() трябва да изглежда подобна на това:



За да покажем ефектите от неидеалност и разрядността на дискретизация, можем да преобразуваме двете синусоиди от времевото в честотното пространство. По сигнали и системи сме учили, че това става с преобразуване на Фурие. В този случай - дискретно преобразуване на Фурие. Тук съм подготвил по-голяма част от необходимото за преобразуването на сигналите. Ако някой има затруднения да си представи до

какво всъщност се свежда дискретното преобразувание на Фурие, може да ме пита. Мога да дам много интуитивно обяснение за този "феномен".

```
NFFT = 2^nextpow2(n); % дължина на прозореца на преобразуването
(използваме правоъгълен прозорец)

Y = fft(y,NFFT)/n; % y се явява входен сигнал за преобразуване

f = fs/2*linspace(0,1,NFFT/2+1);
% генерираме честотните точки през които преобразувахме,
с цел изобразяването им на X оста

% Изобразяване на спектъра

figure(2);
stem(f,2*abs(Y(1:NFFT/2+1)));
title('Single-Sided Amplitude Spectrum of y(t)');
xlabel('Frequency (Hz)');
ylabel('|Y(f)|');
```

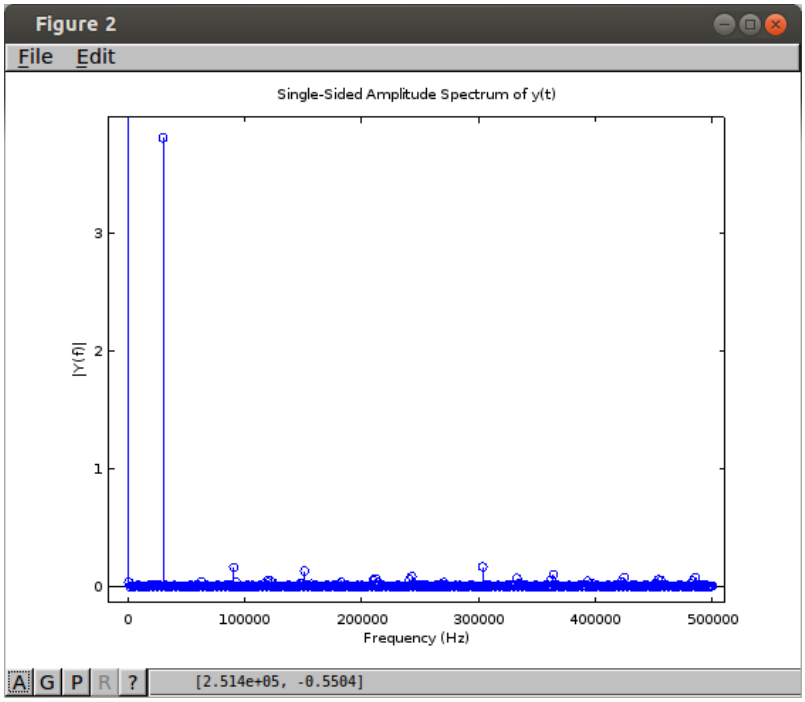
Входната честота на синусоидата трябва да е избрана така, че да спазим правилото за кохерентно семплиране, или:

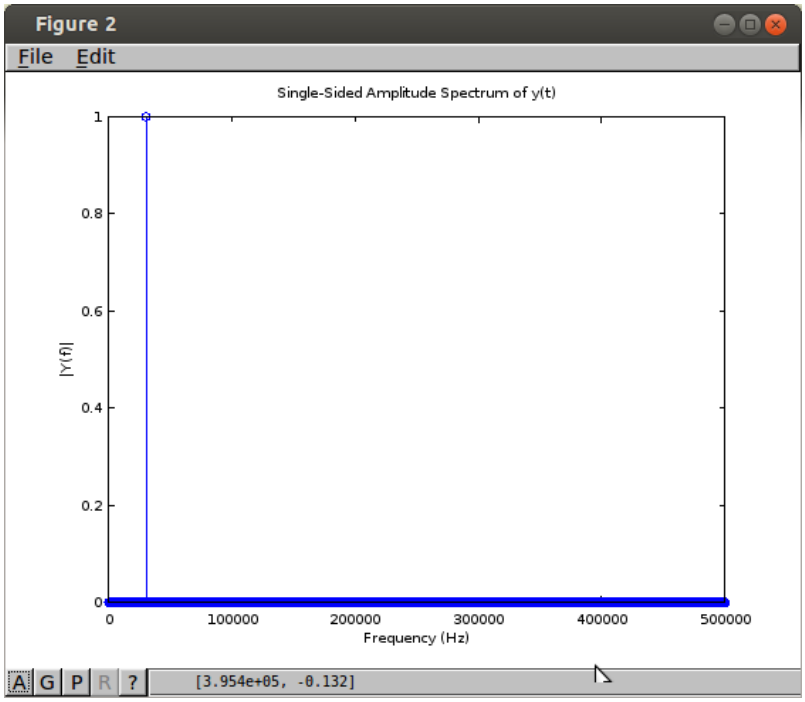
$$\frac{f_{in}}{f_s} = \frac{M_{cycles}}{N_{samples}}$$

Съответно, всяка една честота, която е просто число, може да свърши работа. В такъв случай за входна честота можем да изберем:

```
f = 31*fs/n
fs = 1e6;
```

От тук може да забележим разликите във високочестотните компоненти. При идеална синусоида не съществуват хармонични тонове, при дискретизираната обаче нещата далеч не стоят така.





Променете разрядността на дискретизатора, каква промяна настъпва

в хармоничните тонове? Дали промяната е линейна за синусоида? Една "обща"формула за размисъл:

$$SNDR = N6.02 + 1.76[dB]$$

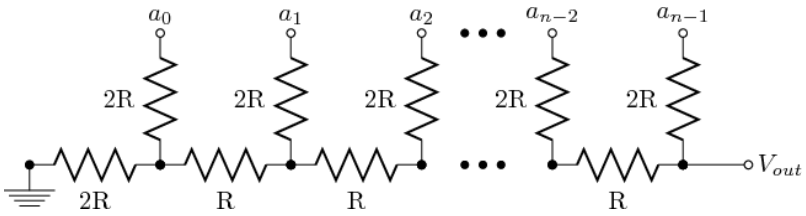
Тук завършваме с част първа и можем да преминем към възпроизвеждането на синусоидите които генерирахме през цифрово-аналогов преобразувател (ЦАП), който ще построим зеадно. Ще се опитаме да възпроизведем и музика през нашата примитивна (или не чак толкова) звукова карта.

3 Част втора

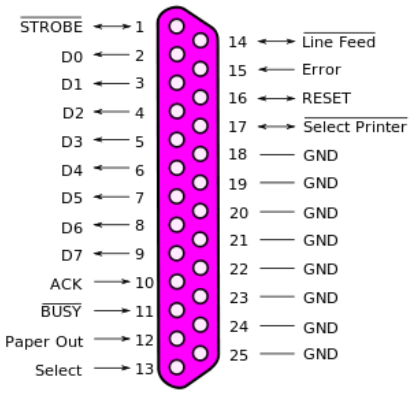
За да можем да възпроизведем звук от бинарни числа, ще ни е необходим (както бе споменато на няколко места по-горе) цифрово-аналогов преобразувател (ЦАП). Има много видове ЦАП и методи за преобразуване.

3.1 Първо подзадание

Много добър баланс между ниска сложност (малък брой елементи) и задоволително качество се явява т.н. R2R ЦАП. По-долу е показана примерна схема, която можем да използваме. Преди това е хубаво да си обясним как всъщност (чисто на схемно ниво / аналогово) става преобразуването. За целта - запознайте се с Приложение 1.



След като свържем необходимите резистори (10 и 20 kOhm), нека да се запознаем с периферията на LPT портовете. Google + Wikipedia са доста добро начало, но за да обобщим, ето и диаграма с изводите:



Ще използваме информационните изводи D0 - D7, което ни позволява да построим 8-битов ЦАП. Така ще постигнем приблизително 50 dB съотношение сигнал - шум (формулката за SNDR от част първа).

3.2 Второ подзадание

За да можем да управляваме информационните изводи, обаче, ще е необходимо да използваме instrument-control-package за octave. Можем да го заредим по следния начин:

```
pkg load instrument-control
```

За да отворим паралелния порт за "писане"използваме следните команди:

```
pp = parallel("/dev/parport0", 0);
% отваряме parport0 (на повечето линукс машини това е стандартното име)

pp_data(pp,255); % задаваме стойност на вектор D0-D7
 (8-бита съответно от 0 до 255)

pp_close(pp); % затваряме
```

По този начин можем да "изпеем"всички семпли на нашата синусоида или музика, намиращи се във вектор. Тъй като instrument-control не ни осигурява функции за контрол на семплиращата честота, ще бъде необходимо да я измерим ръчно и да нагласим броя на семплите на синусоидата или музиката, които подаваме.

```
for k = 1:100000
pp_data(pp,255);
pp_data(pp,0);
end
```

Като при запускането измерваме честотата с осцилоскоп напр на D0. След като сме се уверили в честотата можем да подадем определена синусоида с такъв брой семпли в период, така че да се чува верен тон. За прочитане на wav файл можем да използваме функцията wavread(). За да оразмерим входния вектор така, че да се побере от 0 до 255, можем да използваме следното закръгление:

```
[y0 rate bits] = wavread("/usr/share/skype/sounds/CallRingingIn.wav");

y=round(y0*(2^N/2)+(2^N/2)); % скалиране от 0 до 1 към 0 до 255

nSmpl = length(y); % намиране на дължината на y

pp = parallel("/dev/parport0", 0); % отвори LPT

for k = 1:nSmpl % пеем
pp_data(pp,y(k));
end

pp_close(pp);
```

След като имаме всичко необходимо, не е лошо да можем да изпеем и други файлове освен CallRingingIn.wav. За целта можем да използваме командата sox в линукс за да преобразуваме любимата си mp3ка в wav формат с точно определена семплираща честота и разрядност. Ето малко примери:

```
sox Side\ 3,\ Pt.\ 6\ Allons-Y\ (2).mp3 output.wav channels 1 rate 64000
sox -v 0.95 Nervana.mp3 output.wav channels 1 rate 44000
```

При промяна на rate, sox или ще направи линейна интерполация за достигане на необходимата семплираща честота, или в случай на по-ниски семплиращи честоти от mp3ката ще осъществи децимиране на семпли до достигане на зададената честота. Интересни опити:

1. Какво става ако шунтираме MSB резисторите? Как се изменя качеството на звука?

2. А ако откачим LSB (D0)?

3. Как се изменя качеството, ако поставим нископропускащ филтър на изхода на ЦАП?

4. Има ли разлика в съотношението сигнал-шум на музика с висок динамичен обхват и такава с нисък?

5. Освен музика с какво още бихме могли да експериментираме? Как може тази техника да ни бъде полезна за в бъдеще, дори и ако не се занимаваме с микроелектроника?