Università degli Studi di Perugia

Dipartimento di Matematica e Informatica

Report di Laboratorio – Tecniche di Acquisizione Dati

Relazione

Studenti

Anno Accademico 2022-2023

Sommario

- Identificare il «Picco» principale
- Mascherare tutti i picchi tranne quello principale, lasciando inalterati gli altri

Introduzione

In questa relazione sono illustrati e spiegati i quattro esercizi svolti in laboratorio durante il corso.
Nel primo capitolo utilizzeremo un FPGA (Field Programmable Gate Array) è un dispositivo a semiconduttore suddiviso in blocchi logici configurabili in cui è possibile definire le funzioni dopo la fabbricazione, utilizzando la suite software Vivado. Inoltre FPGA utilizza la proprietà parallela un esempio è possibile vederlo nell'esercizio 1.6.
Per le altre esercitazioni rimanenti utilizzeremo il linguaggio di programmazione Python.

Capitolo 1. FPGA e «KITT»

1.1 Accendere un LED alla pressione del pulsante centrale

```
module blink (
input clk,
input btnC,
output reg [7:0] led
);
 reg i;
initial
i = 0;
always @ (posedge clk) begin
led [0] = i;
if (btnC == 1) begin
   i <= 1;
end
end
endmodule
```

Nella prima riga troviamo l'istruzione `module` seguita da blink, il nome del programma. A seguire troviamo le istruzioni di `input` e `output`. Con l'`input` inseriamo le variabili che utilizzeremo nel programma, in questo caso `clk` e `btnC`. `Clk` è una variabile che rappresenta il clock mentre `btnC` il bottone centrale. Con l'`output` dichiariamo che si accenderà il primo led e con `reg` dichiariamo la variabile e il valore verrà mantenuto finchè non verrà assegnato di nuovo.
`always @ (posedge clk)` viene ripetuto a ogni ciclo di clock . All'interno il led prende il valore di `i`, ogni volta che il bottone viene premuto la variabile viene aggiornata. Perciò se il valore del led sarà diverso da 0 si accenderà.

1.2 Far lampeggiare un led con frequenza di 2 Hz

```
module blink2Hz (
input clk,
output reg [7:0] led
);
reg [32:0] i;
initial
i = 0;
```

```
always @ ( posedge clk ) begin
led [0] <= i [23];
i <= i + 1;
end
endmodule
```

Nella prima riga troviamo l'istruzione `module` seguita da `blink2Hz`, il nome del programma. A seguire troviamo le istruzioni di `input` e `output`. Con l'`input` inseriamo le variabili che utilizzeremo nel programma, in questo caso `clk`. Con l'`output` dichiariamo gli otto led mentre `i` è un array di 33 elementi che all'inizio del programma verrà inizializzato a 0. Nel blocco Always assegnavo il valere del ventiquattresimo bit di i al primo led. Infine a ogni ciclo di clock i viene incrementata di uno. Cosicché il ventiquattresimo bit viene aggiornato con una frequenza di 2Hz.

1.3 Far lampeggiare un LED con frequenza di 2 Hz quando il primo switch è ON

```
module blinkswon(
input clk,
input [15:0]sw,
output reg[7:0] led,
);
reg [32:0] i;
initial
i = 0;
always @ (posedge clk) begin
if( sw[0] == 1) begin
 led[0] <= i[23];
 i <= i + 1;
end
else begin
led[0] <= 0;
end
end
endmodule
```

Questo programma si chiama `blinkswon` e la differenza tra questo programma e quello precedente è: l'aggiunta dei sedici switch e l'incremento di i solo se il primo switch è uguale a uno.

1.4 Far lampeggiare un LED con frequenza di 2 Hz quando il primo switch è OFF, altrimenti farlo lampeggiare con frequenza doppia

```
module blinkswoff(
input clk,
input [15:0]sw,
output reg[15:0] led,
);
reg [32:0] counter;
 initial counter =0;
always @ (posedge clk) begin
if( sw[0]==1) begin
led[0]<=counter[22];
counter<=counter+1;
end
 else begin
 led[0]<=counter[23];
counter<=counter+1;
end
end
endmodule
```

La differenza con il programma precedente è nella parte finale. Infatti se lo switch è diverso da uno il led raddoppierà la frequenza di accensione.

1.5 Realizzare un contatore binario e mostrarlo sui 16 LED a disposizione della FPGA

```
Module count(
Input clk,
Output reg [15:0] led,
);
```

```
 reg [38:0] i;
 reg [3:0] j;
initial
i=0;
j=0;
always @( posedge clk ) begin
if (led[j]!=i[j+23]) begin
led[j]<=i[j+23];
 j<=j+1;
 end
 else begin
 j<=0;
i<=i+1;
end
end
endmodule
```

In questo programma abbiamo due array, `i` con 39 bit e `j` con 4 che verranno inizializzati a zero. Nel blocco `always` i led vengono aggiornati con una differenza di ventitré bit dell'array `i`. Come risultato avremo che il led `j` si accenderà se `i` è uguale a uno altrimenti si spegnerà.

1.6 Iniziando con un solo LED acceso, far in modo che: - La luce si sposti a sinistra alla pressione del pulsante sinistro del joystick - La luce si sposti a destra alla pressione del pulsante destro del joystick - In entrambi i casi, fare in modo che lo spostamento si fermi al raggiungimento del LED più esterno

```
module joystick(
input clk,
input btnD,
input btnS,
output reg[15:0] led,
);
reg [4:0] i;
 reg j;
reg k;
always @ (posedge clk) begin
led[i] = 1;
```

```
   j<=btnD;
    if(j!=btnD & btnD==1 & led[0]=0) begin
   led[i] = 0;
   i <= i-1;
   end
   k<=btnS;
   if(k!=btnS & btnS==1 & led[15]==0) begin
   led[i]=0;
   i <= i+1
   end
   end
   endmodule
```

Il programma si chiama `joystick`, a seguire troviamo le istruzioni di `input` e `output`. Nell' `input`, oltre `clk` utilizziamo anche il bottone sinistro `btnS` e quello destro `btnD`. Nella riga 7 dichiariamo l'array `i` di 5 elementi e due variabili: `j` per lo scorrimento a sinistra e `k` per lo scorrimento a destra. Nel blocco `always` viene acceso il primo led di destra. Appena viene premuto un pulsante e l'ultimo led è spento allora aggiorniamo la variabile `i`: con un valore più alto per accendere il led a sinistra, con un valore più basso per accendere quello di destra.

1.7 Far scorrere autonomamente il LED acceso a destra e sinistra, facendolo <<rimbalzare>> al raggiungimento del bordo

```
module bounce(
 input clk,
output reg[15:0] led
);
reg [4:0] i;
reg [32:0] c;
reg j;
reg k;
initial
 led[0] = 1;
always @ (posedge clk) begin
c <= c+1;
 if(j!=c[26]) begin
j <= c[26];
```

```
if(k==1) begin
led[i]=0;
 led[i+1]=1;
i<=i+1;
end
else begin
 led[i]=0;
led[i-1]=1;
 i<=i-1;
end
if(led[15]==1 || led[0]==1) begin
k<=k+1;
end
end
end
endmodule
```

Nella riga 4 e 5 dichiariamo due array `i` e `c`, rispettivamente di 5 e 33 bit e due variabili `j` e `k`. Nella riga 8 il primo led viene acceso impostandolo a 1. Nel blocco `always` il contatore `c` viene incrementato per far accendere i led a una frequenza 2Hz. Nel primo `if` spegniamo il primo led e se `k` è uguale a 1 viene acceso il led a destra mentre se `k` è uguale a 0 viene acceso quello di sinistra. Nell'ultimo `if` confrontiamo se il primo o l'ultimo led è acceso e in questo caso aggiorniamo la variabile `k`.

Capitolo 2. «Whatsapp»

Realizzare programmi python per

• registrare e riprodurre un segnale audio dalla scheda audio

– provare ad acquisire per 10 secondi

– provare a riprodurre al doppio e alla metà della velocità

• Con matplotlib disegnare un semplice grafico (array arbitrario), provando a

– cambiare la dimensione della finestra

– aumentare il numero di punti

– cambiare il colore della linea

– cambiare il titolo del grafico

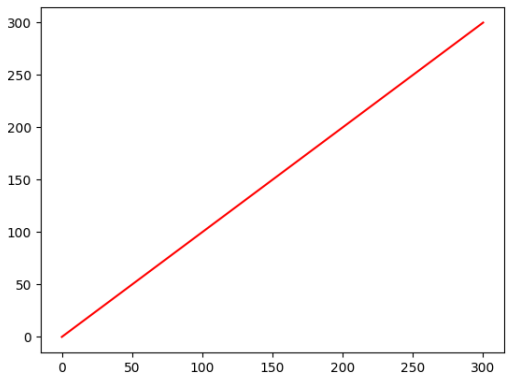
– aggiungere le label sugli assi

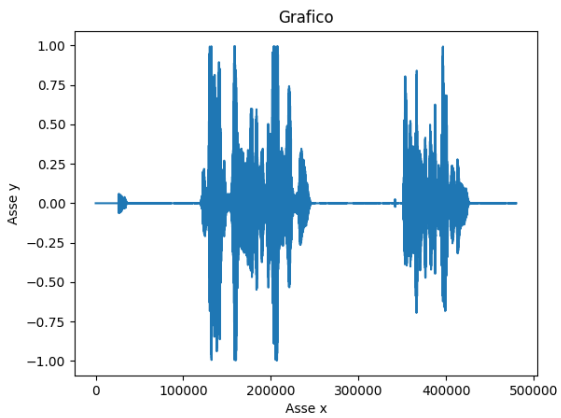
• registrare un segnale audio, riprodurlo e plottarne la «waveform»

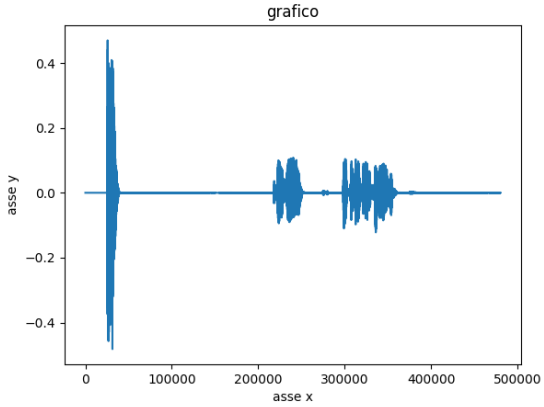
– Plottare solo un canale

```
import soundcard as sc
import numpy as np
import matplotlib.pyplot as plt
speakers=sc.all_speakers()
default_speaker=sc.default_speaker()
mics=sc.all_microphones()
default_mic=sc.default_microphone()

data= default_mic.record(samplerate=48000, numframes=480000)
default_speaker.play(data, samplerate=48000)

y=np.linspace(0,300)
plt.plot(y,y,'r')
plt.show()
plt.plot(data[:,0])
plt.title('Grafico')
plt.xlabel('Asse x')
plt.ylabel('Asse y')
plt.show()
```

Nelle prime tre righe del codice abbiamo importato tre librerie: `soundcard, numpy e matplotlib.` `Soundcard` registra e riproduce l'audio, `numpy` aggiunge supporto a grandi matrici e array multidimensionali mentre `matplotlib` permette di creare grafici. Nelle successive quattro righe vengono impostati gli speaker e gli autoparlanti di default. Nell'ottatava riga registriamo 10 secondi il suono grazie al rapporto tra i `samplerate`, cioè la frequenza di campionamento e i `numframes`, cioè la quantirà di frames acquisiti. Con la riga successiva possiamo decidere la velocità dell'audio prodotto, infatti portando la velocità di riproduzione a 280000 raddoppia. Nell'ultima parte creiamo il grafico utilizzando l'istruzione `plt.plot` e poi mostrato con il comando `plot.show.`







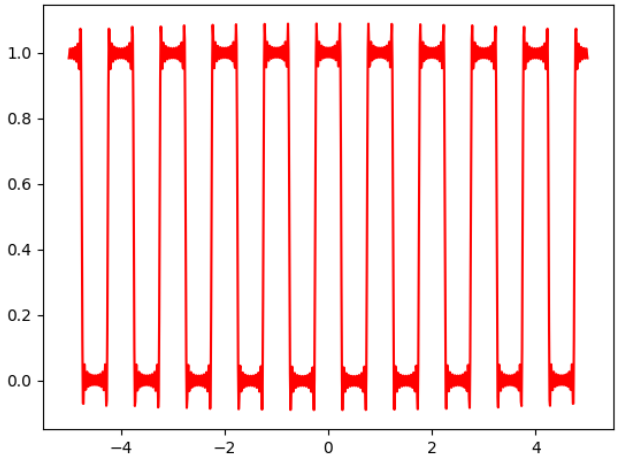
Capitolo 3. Anti-trasformata e FFT

Esercizio 3.1

Sintetizzare un segnale di treno di impulsi (onda quadra) a partire dai suoi coefficienti di Fourier e plottarne la waveform

```
import numpy
import matplotlib.pyplot as mat
ar=[]
t=numpy.linspace(-5,+5,1000)
ft=0
n=1
f=1
while n<20:
    ft=ft+(1/n)*(-1)**((n-1)/2)*numpy.cos(2*numpy.pi*n*t*f)
    n=n+2
ar=(1/2+(2/numpy.pi)*ft)
mat.plot(t,ar, 'r')
mat.show()
```

Dalla riga 3 ci sono le inizializzazioni: l'array ar che conterrà il seganle, t che genererà 1000 valori compresi tra -5 e 5 e ft che conterrà la sommatoria delle sinusoidi. Nel ciclo while calcoliamo la sommatoria delle sinusoidi e incrementiamo n di due. Con l'istruzione successiva al while effettuiamo l'ultima trasformata mentre con le ultime istruzioni plottiamo il grafico



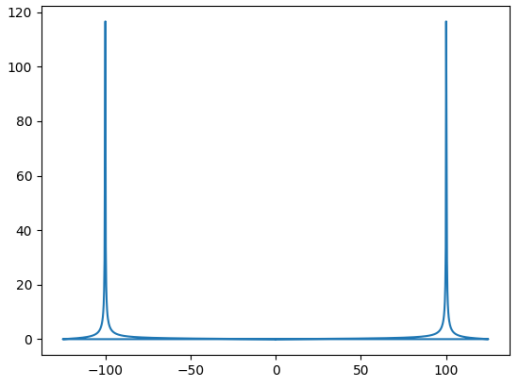
Esercizio 3.2

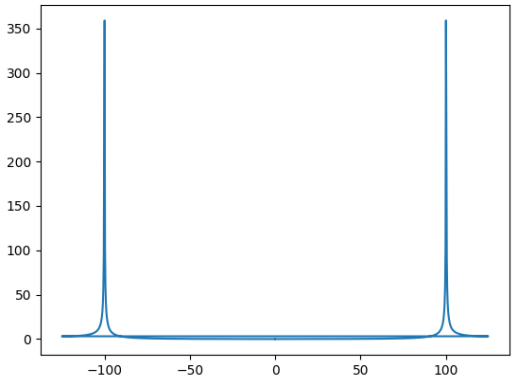
Realizzare lo studio in frequenza (potenza, parte reale e parte immaginaria dei coefficienti) per ciascun segnale - onda sinusoidale a 100 Hz, 200 Hz, 440 Hz - onda triangolare a 100 Hz, 200 Hz, 440 Hz - onda quadra a 100 Hz, 200 Hz, 440 Hz realizzare lo studio in frequenza per un segnale dato dalla somma delle tre onde sinusoidali di cui sopra

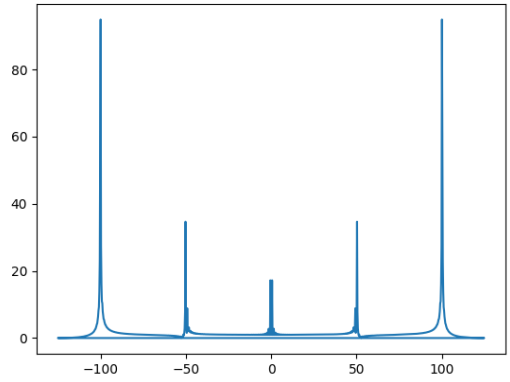
```
import matplotlib.pyplot as plt
import numpy
import scipy.fft as sc
import scipy.signal as sci
f0=[]
f1=[]
f2=[]
f3=[]
f4=[]
f5=[]
f6=[]
f7=[]
t=numpy.linspace(-2,+2,1000)
f=100
sinusoidale=numpy.sin(2*numpy.pi*t*f)
f0=numpy.real(sc.fft(sinusoidale))
n=f0.size
freq=sc.fftfreq(n,4/1000)
plt.plot(freq,numpy.abs(f0))
plt.show()
f1=numpy.imag(sc.fft(sinusoidale))
n0=f1.size
freq0=sc.fftfreq(n0,4/1000)
plt.plot(freq0,numpy.abs(f1))
plt.show()
triangle=sci.sawtooth(2*numpy.pi*f*t, 0.5)
f2=numpy.real(sc.fft(triangle))
n1=f2.size
freq1=sc.fftfreq(n1,4/1000)
plt.plot(freq1,numpy.abs(f2))
plt.show()
```

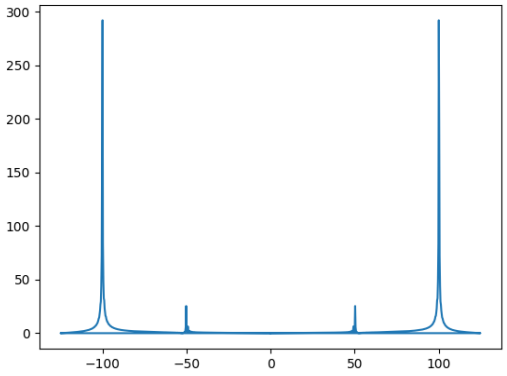
```
f3=numpy.imag(sc.fft(triangle))
n2=f3.size
freq2=sc.fftfreq(n2,4/1000)
plt.plot(freq2,numpy.abs(f3))
plt.show()
quadra=sci.square(2*numpy.pi*f*t)
f4=numpy.real(sc.fft(quadra))
n3=f4.size
freq3=sc.fftfreq(n3,4/1000)
plt.plot(freq3,numpy.abs(f4))
plt.show()
f5=numpy.imag(sc.fft(quadra))
n4=f5.size
freq4=sc.fftfreq(n4,4/1000)
plt.plot(freq4,numpy.abs(f5))
plt.show()
signal=numpy.sin(2*numpy.pi*100*t)+numpy.sin(2*numpy.pi*200*t)+numpy.sin(
2*numpy.pi*440*t)
f6=numpy.real(sc.fft(signal))
n5=f6.size
freq5=sc.fftfreq(n5,4/1000)
plt.plot(freq5,numpy.abs(f6))
plt.show()
f7=numpy.imag(sc.fft(signal))
n6=f7.size
freq6=sc.fftfreq(n6,4/1000)
plt.plot(freq6,numpy.abs(f7))
plt.show()
```

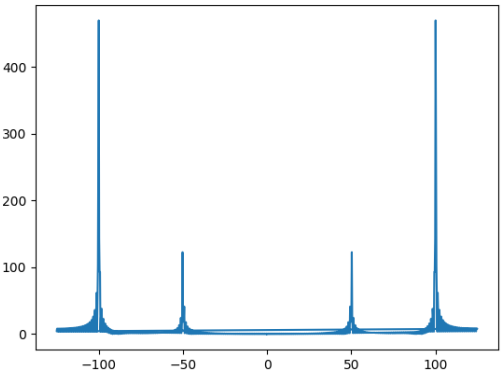
La libreria `scipy` contiene algoritmi e strumenti matematici. Fino alla riga 12 c'è l'inizializzazione di otto array poi con t generiamo 1000 numeri compresi tra -2 e +2 e con f la frequenza. La trasformata di Fourier vien calcolata con `fft` e `scipy` mentre frequenza viene calcolata con `fftfreq` aggiungendo la grandezza dell'array e il samplerate. La stessa cosa vale per la parte reale. Infine viene calcolato il segnale con le frequenze 100 Hz, 200 Hz, 440 Hz come le precedenti.

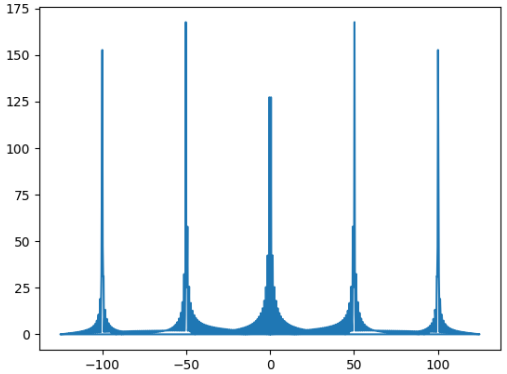


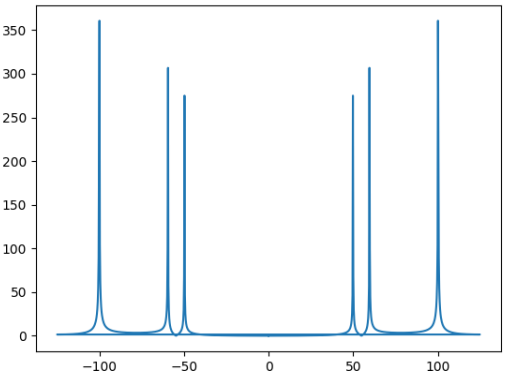


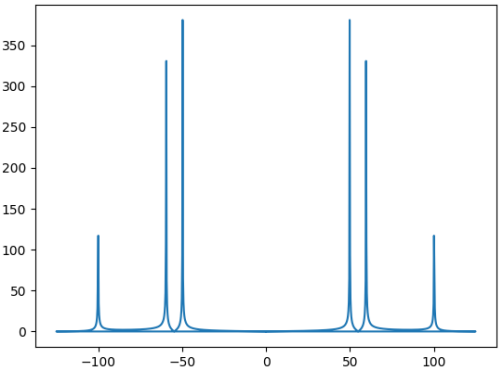












(GRAFICI SBAGLIATI, AUMENTARE VALORI NEL CODICE)

Capitolo 4. <Garage band>

```
import soundfile as sf
import matplotlib.pyplot as plt
import numpy as np
import scipy.fft as sc
```

```
def matching_freq(freq):
note=""
if(freq>15 and freq<17.32):
note= "C0"
return note
elif(freq>17.32 and freq<19.45):
note="D0"
 return note
 elif(freq>19.45 and freq<20.8):
note= “E0”
return note
elif(freq>20.8 and freq<23.12):
note="F0"
 return note
elif(freq>23.12 and freq<25.96):
note="G0"
 return note
elif(freq>25.96 and freq<29.14):
note="A0"
 return note
elif(freq>29.14 and freq<31):
note="B0"
 return note
elif(freq>31 and freq<34.65):
note="C1"
 return note
elif(freq>34.65 and freq<38.89):
note="D1"
 return note
elif(freq>38.89 and freq<42):
note="E1"
 return note
elif(freq>42 and freq<46.25):
note="F1"
 return note
elif(freq>46.25 and freq<51.91):
note="G1"
 return note
```

```
elif(freq>51.91 and freq<58.27):
note="A1"
 return note
elif(freq>58.27 and freq<63):
note="B1"
 return note
elif(freq>63 and freq<69.30):
note="C2"
 return note
elif(freq>69.30 and freq<77.78):
note="D2"
 return note
elif(freq>77.78 and freq<85):
note="E2"
 return note
elif(freq>85 and freq<92.50):
note="F2"
 return note
elif(freq>92.50 and freq<103.83):
note="G2"
 return note
elif(freq>103.83 and freq<116.54):
note="A2"
 return note
elif(freq>116.54 and freq<126):
note="B2"
 return note
elif(freq>126 and freq<138.59):
note="C3"
 return note
elif(freq>138.59 and freq<155.56):
note="D3"
 return note
elif(freq>155.56 and freq<168):
note="E3"
 return note
elif(freq>168 and freq<185):
note="F3"
```

```
 return note
elif(freq>185 and freq<207.65):
note="G3"
 return note
elif(freq>207.65 and freq<233.08):
note="A3"
 return note
elif(freq>233.08 and freq<253):
note="B3"
 return note
elif(freq>253 and freq<277.18):
note="C4"
 return note
elif(freq>277.18 and freq<311.13):
note="D4"
 return note
elif(freq>311.13 and freq<338):
note="E4"
 return note
elif(freq>338 and freq<369.99):
note="F4"
 return note
elif(freq>369.99 and freq<451.3):
note="G4"
 return note
elif(freq>451.3and freq<466.16):
note="A4"
 return note
elif(freq>466.16 and freq<500):
note="B4"
 return note
elif(freq>500 and freq<554.37):
note="C5"
 return note
elif(freq>554.37 and freq<622.25):
note="D5"
 return note
elif(freq>622.25 and freq<675):
```

```
note="E5"
 return note
elif(freq>675 and freq<740):
note="F5"
 return note
elif(freq>740 and freq<830):
note="G5"
 return note
elif(freq>740 and freq<830):
note="G5"
 return note
elif(freq>830 and freq<932):
note="A5"
 return note
elif(freq>932 and freq<1000):
note="B5"
 return note
elif(freq>1000 and freq<1108):
note="C6"
 return note
elif(freq>1108 and freq<1244):
note="D6"
 return note
elif(freq>1244 and freq<1350):
note="E6"
 return note
elif(freq>1350 and freq<1480):
note="F6"
 return note
elif(freq>1480 and freq<1661.22):
note="G6"
 return note
elif(freq>1661.22 and freq<1864):
note="A6"
 return note
elif(freq>1864 and freq<2000):
note="B6"
 return note
```

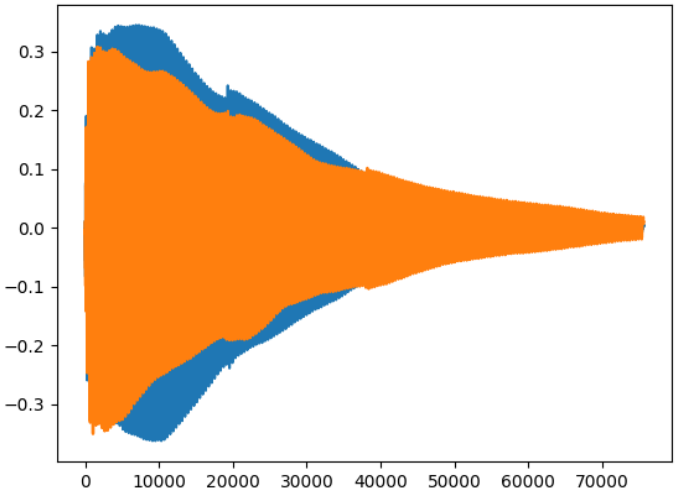
```
elif(freq>2000 and freq<2217.46):
note="C7"
 return note
elif(freq>2217.46 and freq<2489.02):
note="D7"
 return note
elif(freq>2489.02 and freq<2700):
note="E7"
 return note
elif(freq>2700 and freq<2960):
note="F3"
 return note
elif(freq>2960 and freq<3322.4):
note="G7"
 return note
elif(freq>3322.4 and freq<3729):
note="A7"
 return note
elif(freq>3729.31 and freq<4040):
note="B7"
 return note
elif(freq>4040 and freq<4435):
note="C8"
 return note
elif(freq>4435 and freq<4978):
note="D8"
 return note
elif(freq>4978 and freq<5350):
note="E8"
 return note
elif(freq>5350 and freq<5919):
note="F8"
 return note
elif(freq>5919 and freq<6644):
note="G8"
 return note
elif(freq>6644 and freq<7458):
note="A8"
```

```
 return note
elif(freq>7458 and freq<8000):
note="B8"
 return note
file='C:\\Users\\fufif\\Desktop\\diapason.wav'
data,sampler=sf.read(file)
 plt.plot(data)
plt.show()
sf.write('test.wav', data, sampler)
datamono=data[:,0]
f0pow=sc.fft(datamono, norm="forward")
f0=np.abs(sc.fft(datamono))
n=len(f0) plt.plot(datamono)
plt.show()
 freq=sc.fftfreq(n,1/sampler)
plt.plot(freq,np.abs(f0))
plt.show()
f1=np.imag(sc.fft(datamono))
n1=f1.size
freq1=sc.fftfreq(n1,1/sampler)
plt.plot(freq1,np.abs(f1))
plt.show()
 c1=0 freqmax=freq[np.array(f0).argmax()]
while(max(f0)>450):
 if(freq[np.array(f0).argmax()]>0):
c1=c1+1                                  //da rivedere
print(freq[np.array(f0).argmax()])
 print(matching_freq((freq[np.array(f0).argmax()])))
 a=matching_freq((freq[np.array(f0).argmax()]))
c=0
 c2=0
while(a==matching_freq((freq[np.array(f0).argmax()]))):
c=c+1
a=matching_freq((freq[np.array(f0).argmax()+c]))
a=matching_freq((freq[np.array(f0).argmax()]))
while(a==matching_freq((freq[np.array(f0).argmax()]))):
 c2=c2+1
 a=matching_freq((freq[np.array(f0).argmax()-c2]))
```

```
print(c+c2)
f0[np.array(f0).argmax()]=0
for i in range(0,n,1):
 if(freq[i]<freqmax-100 or freq[i]>freqmax+100):
f0[i]=0
f0pow[i]=0
plt.plot(freq,np.abs(f0))
plt.show()
if(c1>=3):
 print("è un accordo")
else:
print("è una nota")
data2=3*np.real(sc.ifft(f0pow, norm="forward"))
plt.plot(data2)
plt.show()
sf.write('test2.wav',data2, sampler)
```

In questo codice oltre alle `librerie numpy, matplotlib e scipy` abbiamo aggiunto la libreria `soundfile` che legge e scrive file audio. Con la funzione `matchinfreq` otteniamo la nota corrispondente inserendo in input una frequenza

Il `.wav` viene memorizzato nella variabile file e letto per estrarre i dati e il samplerate attraverso il comando `sf.read` tramite `sf.write`, il `wav` viene plottato e duplicato con il nome di `test.wav` e memorizzato nella variabile `datamono` (uno dei 2 canali del file audio). La parte reale e la parte immaginaria vengono estratte dal singolo canale con `fft` e `scipy` e con `fftfreq` vengono generati due grafici. Nel ciclo while confrontiamo `max(f0)` ovvero i picchi massimi con un valore minimo di 450 e con `cl` memorizziamo il numero di picchi per vedere se corrisponde a una nota o a un accordo. Nell'istruzione successiva selezioniamo i valori con frequenza positiva. La funzione `matchinfreq` individua la frequenza del picco che viene salvata in a e confrontandola con `matchinfreq` scopriamo quanto è largo il picco. Il for azzera tutte le frequenze diverse da 100 cosicché nellìarray rimarranno solo i valori del picco principale. Infine, con il comando `ifft` prendiamo solo i valori reali dell'array e plottiamo il grafico con `plt.plot`.


0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0
10000
20000
30000
40000
50000
60000
70000

Bibliografia

https://pypi.org/project/SoundCard/

https://pypi.org/project/soundfile/

https://pypi.org/project/numpy/

https://pypi.org/project/matplotlib/

https://pypi.org/project/scipy/