Applicazione delle serie di Fourier

Manuel Zambelli

Indice

Capitolo 1

Introduzione

Durante i cinque laboratori previsti dal Piano Lauree Scientifiche, cui ho partecipato nell'anno in corso insieme a un gruppo di studenti di quinta che hanno aderito all'iniziativa, organizzata dal nostro istituto in collaborazione con l'Università, abbiamo trattato l'argomento dell'analisi armonica di segnali periodici mediante le serie di Fourier[1].
La proposta è stata fatta dal professor Sisto Baldo, analista presso il nostro Ateneo, Facoltà di Scienze MM.FF.NN., il quale ha introdotto l'argomento proponendo lo studio di suoni, in particolare vocali, che dovevano essere simulati attraverso l'applicazione della trasformata di Fourier discreta, DFT (Discrete Fourier Transform).

Tratterò in questo lavoro dell'applicazione delle serie di Fourier, cercando di far vedere lo svariato numero di ambiti in cui si possono applicare e commentando i programmi da me realizzati per mezzo del software *Mathematica* 7 [2].

Gli argomenti saranno i seguenti:

- approssimazione delle sette vocali della lingua italiana e riconoscimento delle stesse attraverso le prime due formanti;
- compressione di immagini importate in formato *.png*;

[1] Jean Baptiste Joseph Fourier, fisico e matematico francese, condusse esperimenti sulla propagazione del calore in un corpo e fu proprio lo studio di questi fenomeni che gli consentì di modellizzare l'evoluzione della temperature per mezzo di serie trigonometriche. Formulò cos quella che poi sarebbe diventata la legge di Fourier, cioè l'equazione generale della conduzione termica. Dal punto di vista matematico teorizzò la serie di Fourier e la sua conseguente trasformata.

[2] In altre versioni di Mathematica alcuni programmi potrebbero non funzionare, in quanto da una versione all'altra cambia il modo in cui file esterni vengono importati.

- risoluzione dell'*equazione della corda vibrante*, anche denominata *equazione delle onde unidimensionale*.

1.1 Perché LaTeX

Sono rimasto affascinato da questo linguaggio durante il progetto tandem, il cui responsabile era Enrico Gregorio (Algebrista presso l'Universitá Degli Studi di Verona), uno dei pionieri a livello internazionale di questo linguaggio che vista la mia curiositá mi ha indicato le procedure per scaricare tutto il *package* e le guide necessarie per cominciare ad utilizzarlo. Dopo aver giocato un po', sperimentando comandi, e dopo aver visto quanto sia professionale e bellissima la composizione tipografica, ho deciso di scrivere la tesina in LaTeX.

Capitolo 2

Un po' di teoria

Il seguente capitolo tratterà delle questioni teoriche salienti necessarie per una chiara comprensione di ciò che verrà spiegato in seguito.

2.1 Polinomi trigonometrici

Per comprendere a fondo il lavoro presentato nei prossimi capitoli è di fondamentale importanza rispondere alla seguente domanda: cos'è un polinomio trigonometrico [1]?
Un polinomio trigonometrico non è altro che una funzione del tipo

$$a_0 \cos 0t + a_1 \cos t + b_1 \sin t + a_2 \cos 2t + ... + a_n \cos nt + b_n \sin nt \qquad (2.1)$$

che può anche essere visto come

$$\sum_{k=0}^{n} a_k \cos kt + b_k \sin kt \qquad (2.2)$$

o anche

$$\frac{a_0}{2} + \sum_{k=1}^{n} a_k \cos kt + b_k \sin kt \qquad (2.3)$$

dove a_i e b_i sono costanti reali.

Un polinomio trigonometrico di questo tipo è anche detto polinomio trigonometrico di grado n. Ogni termine del polinomio è chiamato *armonica*.

[1] Molto spesso il termine polinomio trigonometrico è usato come sinonimo di serie trigonometrica.

2.2 La distanza tra due funzioni

Necessitiamo di uno strumento in grado di misurare la *distanza* tra due funzioni, in modo da poter essere in grado di dichiarare quando una funzione f approssima meglio di un'altra funzione g, una data funzione h. La misura della bontà della nostra approssimazione sarà chiamata *distanza* tra le due funzioni. Ora, noi sappiamo che la distanza tra due punti nel piano cartesiano si calcola con la formula

$$d((x_1, y_1), (x_2, y_2)) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{2.4}$$

mentre nello spazio

$$d((x_1, y_1, z_1), (x_2, y_2, z_2)) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{2.5}$$

e in generale, in un ipotetico spazio n-dimensionale, possiamo dire che la distanza tra due punti si calcola mediante questa formula generalizzata

$$d(P_1, P_2) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_n - y_n)^2} \tag{2.6}$$

Potremmo definire la distanza tra due funzioni come la distanza tra due punti in uno spazio a infinite dimensioni, in parole povere, la distanza in $(-\pi, \pi)$ tra due funzioni possiamo scriverla come segue

$$d(f, h) = \sqrt{\int_{-\pi}^{\pi} (f - h)^2} \tag{2.7}$$

La (2.7) è anche chiamata *distanza euclidea* tra due funzioni. Più è piccola, migliore è l'approssimazione.

2.3 Le relazioni di ortogonalità tra funzioni trigonometriche

Quanto valgono i seguenti integrali?

$$\int_{-\pi}^{\pi} \cos ntdt, \qquad \int_{-\pi}^{\pi} \sin ntdt \tag{2.8}$$

Prendiamo in considerazione l'integrale del coseno. L'antiderivata del coseno è il seno, e quindi in questo caso ($n \neq 0$)

$$\int_{-\pi}^{\pi} \cos ntdt = \frac{\sin \pi n}{n} - \frac{\sin (-\pi n)}{n} = 0 \tag{2.9}$$

Se $n = 0$, banalmente si ha:

$$\int_{-\pi}^{\pi} \cos 0 dt = \int_{-\pi}^{\pi} dt = 2\pi \tag{2.10}$$

Per il secondo integrale, sappiamo che l'antiderivata del seno è -coseno, da cui

$$\int_{-\pi}^{\pi} \sin ntdt = -\frac{\cos \pi n}{n} + \frac{\cos(-\pi n)}{n} = 0 \tag{2.11}$$

Complichiamoci un po' la vita, quanto varrà

$$\int_{-\pi}^{\pi} \cos nt \cos mtdt, \qquad m, n = 1, 2, 3, \ldots? \tag{2.12}$$

Non è così semplice come nei due casi precedenti, vediamo allora, con un piccolo trucco, come possiamo risolverlo...
Applicando la formula di addizione del coseno, possiamo scrivere $\cos nt \sin mt$ come

$$\frac{1}{2}[\cos (m-n)t + \cos (m+n)t] \tag{2.13}$$

da cui

$$\int_{-\pi}^{\pi} \cos nt \cos mtdt = \frac{1}{2}[\int_{-\pi}^{\pi} \cos (m-n)tdt + \int_{-\pi}^{\pi} \cos (m-n)tdt] \tag{2.14}$$

Come abbiamo visto precedentemente, i due integrali tra parentesi quadre valgono 0, a meno di un'importante eccezione, ovvero quando $m = n$, in quanto il primo dei due integrali tra quadre si annulla e stiamo calcolando l'integrale di 1, che vale 2π.

Ricapitolando

$$\int_{-\pi}^{\pi} \cos nt \cos mtdt = \begin{cases} 0 & \text{se } m \neq n \\ \pi & \text{se } m = n \end{cases}$$

e si potrebbe anche verificare che

$$\int_{-\pi}^{\pi} \sin nt \sin mt = \begin{cases} 0 & \text{se } m \neq n \\ \pi & \text{se } m = n \end{cases}$$

$$\int_{-\pi}^{\pi} \cos nt \sin mt = 0$$

Per la dimostrazione di questi risultati occorrerebbero degli altri conticini abbastanza tediosi. Se siete interessati alle dimostrazioni vi rimando ai testi in bibliografia.

Le tre relazioni qui sopra riportate sono dette *relazioni di ortogonalità del seno e del coseno.*

Capitolo 3

Analisi di Fourier

3.1 Giochiamo!

Nulla ci vieta di prendere un polinomio trigonometrico e assegnare ai coefficienti a_i e b_i i valori che vogliamo, quindi facciamolo!
Proviamo a sommare un numero arbitrario n di funzioni sinusoidali tutte con frequenza multipla di una frequenza base, come per esempio la funzione

$$f_n(t) = \sum_{k=1}^{n} \frac{1}{k} \sin kt \tag{3.1}$$

Con $n = 3$ la funzione assume questa forma

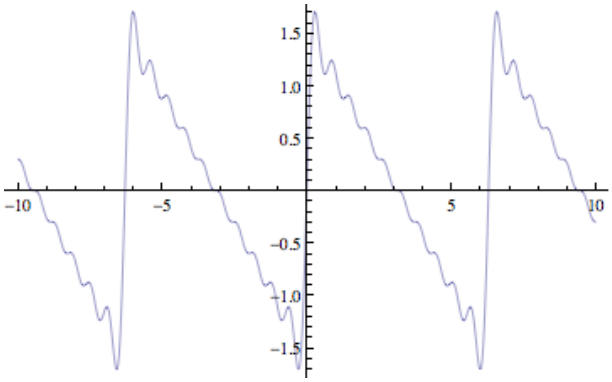
$$\sin t + \frac{1}{2} \sin 2t + \frac{1}{3} \sin 3t \tag{3.2}$$

mentre la forma generale con n arbitrario é

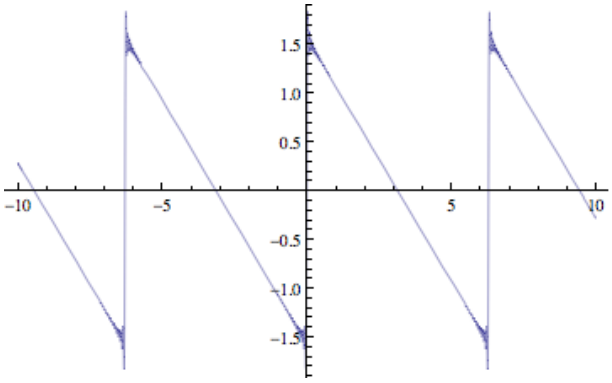
$$\sin t + \frac{1}{2} \sin 2t + \frac{1}{3} \sin 3t + ... + \frac{1}{n} \sin nt \tag{3.3}$$

Prima di procedere con lo studio della funzione, e in tutti gli esperimenti condotti nei laboratori, ho introdotto una misura del tempo in modo che un'unità di tempo corrisponda esattamente a 2π e corrisponda anche al periodo di oscillazione della funzione presa in considerazione. Questo accorgimento ci semplifica molto i calcoli.

Con *Mathematica* ho plottato il grafico della (3.1) con vari valori di n.



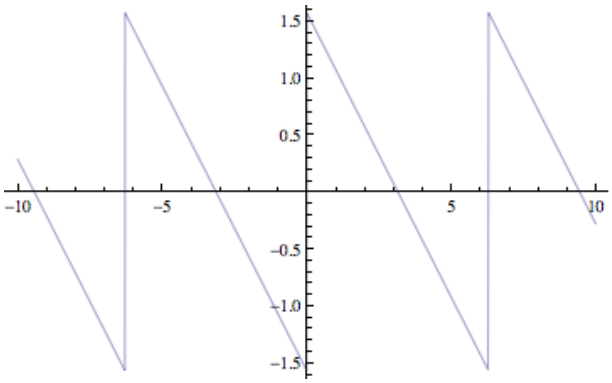
Questo é il grafico della (3.1) con $n = 10$ *nell'intervallo* $(-10, 10)$.



Questo é il grafico della (2.4) con $n = 100$ *nell'intervallo* $(-10, 10)$.

3.1.1 Approssimazione?

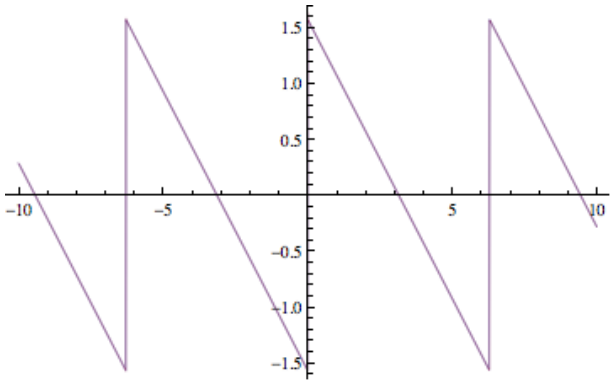
Guardando questi grafici potremmo intuitivamente presupporre che al crescere di n la nostra funzione tenda ad assumere una certa forma, che molto probabilmente sará questa:



Definire questa funzione a tratti risulterebbe abbastanza semplice ma una funzione algebrica che da lo stesso risultato é

$$f(t) = \arctan(\frac{1}{\tan(\frac{t}{2})}) \tag{3.4}$$

É lecito pensare che la (3.1) sia un'approssimazione della (2.7)?
L'intenzione é dare risposta a questa domanda. Per avere una conferma sperimentale della nostra congettura, visualizzo nello stesso grafico f_n e f per avere un raffronto visivo.



Come si puó notare dal grafico, le due funzioni sono indistinguibili e quindi, la nostra congettura non risulta essere del tutto infondata.

3.1.2 Quanto é buona questa approssimazione?

f, vista la forma, viene comunemente chiamata funzione a dente di sega. La nostra ipotesi dice che al crescere di n, f_n é un'approssimazione sempre migliore di f. In realtá anche con $n = 1$ é una buona approssimazione, perchè abbiamo i giusti coefficienti, ma approfondiremo questo punto piú avanti.
Se confrontiamo argutamente i due grafici notiamo che essi sono sempre vicini tranne che nei punti di discontinuitá. Infatti in $t = 0$, $f_n$passa per 0, mentre f non é definita in questo punto(il tratto verticale, infatti, non fa parte della funzione) e la funzione salta da $-\frac{\pi}{2}$ a $\frac{\pi}{2}$.
La nostra approssimazione é comunque una buona approssimazione? In realtá non é solamente una buona approssimazione, é la migliore di tutte!!

3.2 Ricostruzione dei coefficienti di una somma di sinusoidi: la trasformata di Fourier

Poniamo il caso che siamo in possesso di una funzione che sappiamo essere un polinomio trigonometrico di questo tipo:

$$f(t) = \frac{a_0}{2} + a_1 \cos t + b_1 \sin t + a_2 \cos 2t + b_2 \sin 2t + \ldots + a_n \cos nt + b_n \sin nt \tag{3.5}$$

Siamo in grado di risalire ai a_k, b_k senza procedere per tentativi? Se integriamo entrambi i membri tra $-\pi$ e π otteniamo:

$$\int_{-\pi}^{\pi} f(t) = \int_{-\pi}^{\pi} \frac{a_0}{2} + \int_{-\pi}^{\pi} a_1 \cos t + \int_{-\pi}^{\pi} b_1 \sin t + \ldots + \int_{-\pi}^{\pi} a_n \cos nt + \int_{-\pi}^{\pi} b_n \sin nt \tag{3.6}$$

Da cui, ricordandoci le *relazioni di ortogonalitá tra funzioni trigonometriche*, si ottiene che

$$\int_{-\pi}^{\pi} f(t) = \pi a_0 \tag{3.7}$$

In modo analogo possiamo determinarci gli $n-$esimi coefficienti a_n e b_n. Se volessimo calcolarci b_3, basta moltiplicare da entrambe le parti per $\sin 3t$[1] per ottenere che

$$\int_{-\pi}^{\pi} f(t) \sin 3t = \pi b_3 \tag{3.8}$$

In generale, possiamo affermare che

$$\int_{-\pi}^{\pi} f(t) \sin nt = \pi b_n \quad e \quad \int_{-\pi}^{\pi} f(t) \cos nt = \pi a_n \tag{3.9}$$

Funzioni qualsiasi...

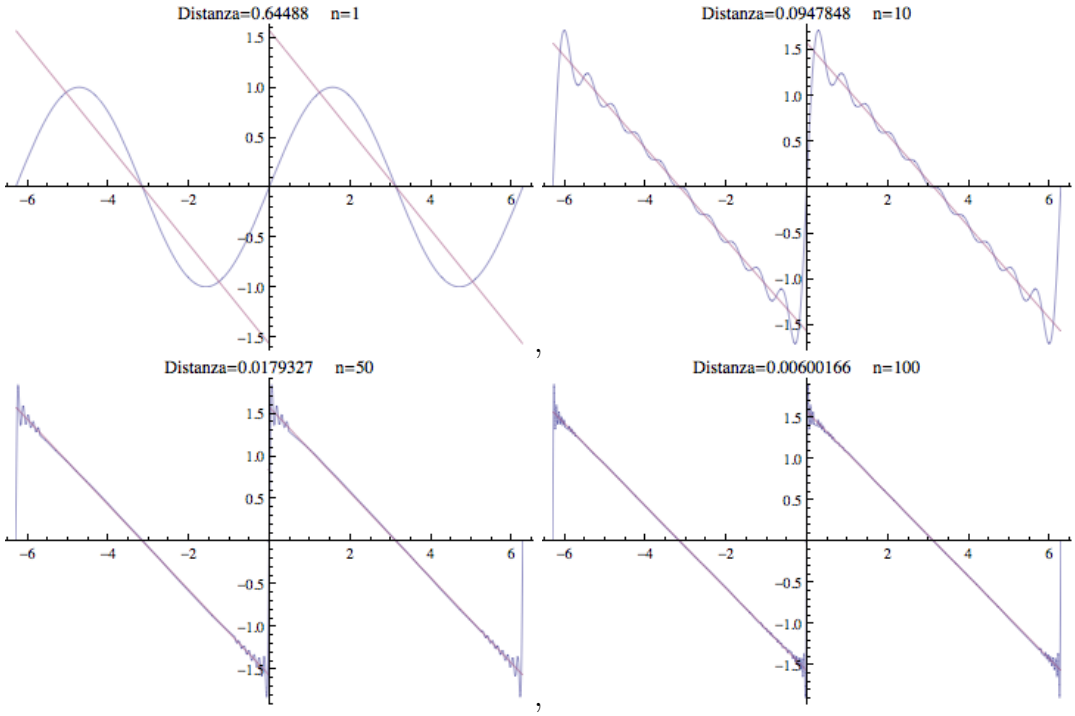
La cosa interessante é che lo stesso identico principio vale per qualsiasi funzione. Abbiamo quindi trovato un metodo che approssima nel miglior modo possibile i coefficienti a_k, b_k.
Cosa significa per noi *nel miglior modo possibile*?
Significa che scegliendo cosí i coefficienti si *minimizza* la distanza[2] tra f e f_n.

[1] $\cos 3t$ se volessimo calcolarci a_3

[2] si intende la distanza euclidea che si é definita nel secondo capitolo

3.3 Convergenza dell'approssimazione al crescere di n

Una questione di fondamentale importanza é proprio la convergenza dell'approssimazione al crescere di n. Questa questione richiede una dimostrazione abbastanza noiosa che puó benissimo essere sostituita dai grafici che troverete di seguito dove approssimo con il polinomio di Fourier la (3.4)[3]

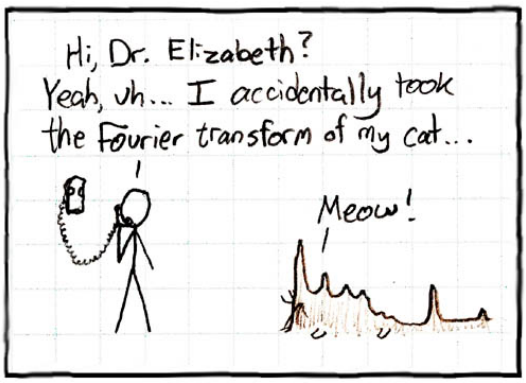


[3] *Un buon esempio vale due libri, C. F. Gauss*

Capitolo 4

Approssimazione di vocali

Una delle cose piú interessanti dell'analisi di Fourier é la possibilitá di comprimere una quantitá impressionante di dati e di informazione, che molte volte risulta superflua.

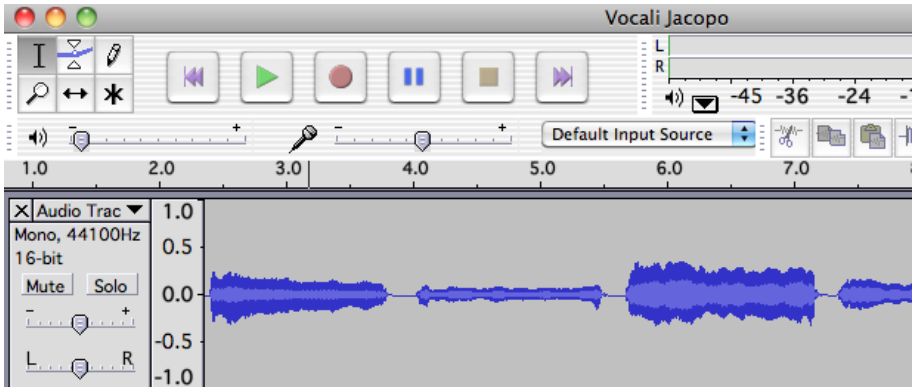


Dal punto di vista acustico questo fatto é parecchio evidente. Il nostro orecchio, infatti, é insensibile alla fase e alle frequenze troppo alte. Questo ci puó giá suggerire un buon metodo per comprimere segnali acustici, quali ad esempio le sette vocali della lingua italiana che sono **I E É A Ó O U**.

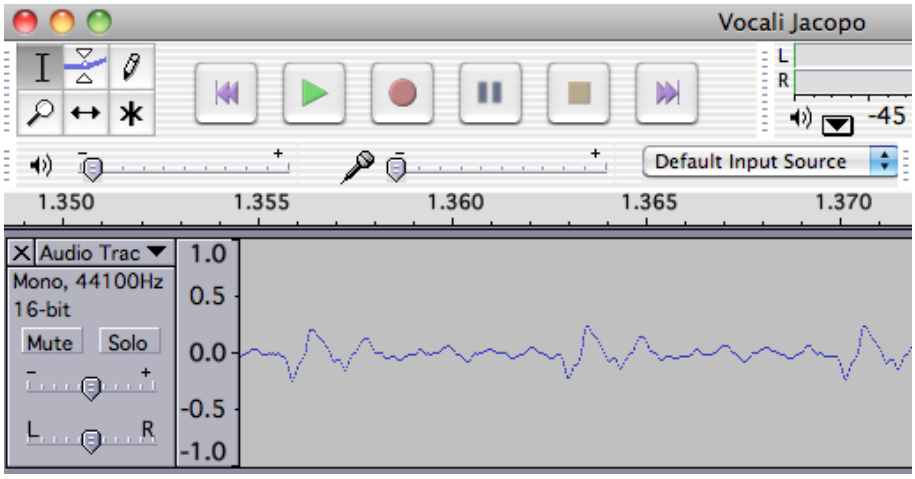
4.1 L'acquisizione delle vocali

Mi sono servito dell'ottimo programma gratuito Audacity per registrare e per far registrare a quante piú persone ho potuto, dei secondi di AAA EEE IIII OOOO UUUU[1]. Una volta ottenuto questo risultato

[1]Durante le registrazioni si é cercato di essere il piú possibile isolati da rumori di sottofondo, per evitare "impuritá" nel lavoro



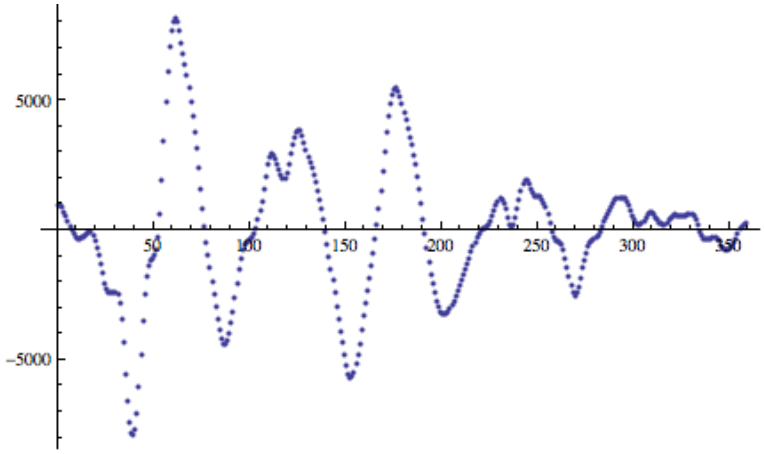
ho ingrandito la schermata fino a che non era riconoscibile la forma d'onda che cercavo, in questo caso la forma d'onda della A



e dopo aver selezionato un unico periodo l'ho salvato in formato .wav, pronto per essere importato nel mio programma in Mathematica! Ovviamente la continuitá di quella linea blu é frutto di un'interpolazione, un periodo é determinato da circa 300 punti di campionamento.

4.2 Simuliamole!!

La nostra A, viene vista in Mathematica come una lista di interi. Grazie al comando `ListPlot` riesco a visualizzarla:



Ora il problema é riuscire a trovare un polinomio trigonometrico che approssimi questa lista di interi. Come possiamo considerare una qualsiasi lista finita di interi una funzione 2π-periodica definita in $(-\pi, \pi)$?
Basta usare la lista per costruire un insieme ordinato di coppie $(-\pi + h * i - 1, \quad L_i)$ dove $h = \frac{2\pi}{numeroElementiLista}$ e L_i rappresenta l'i-esimo elemento della lista.[2]

4.2.1 Programma

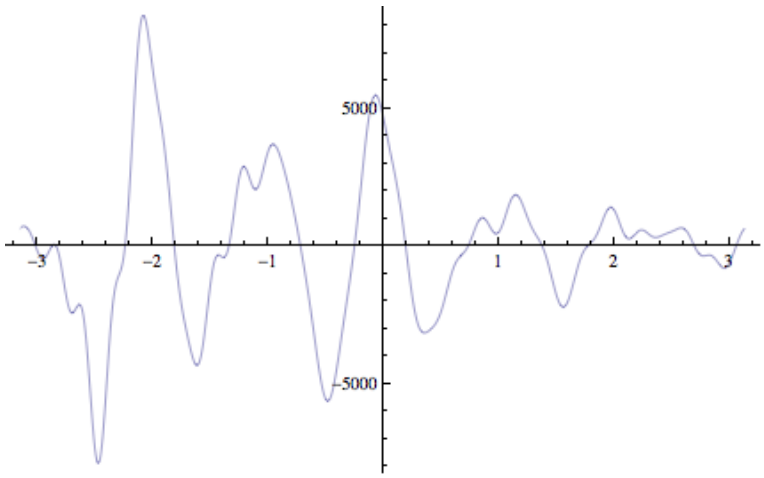
Il programma che ho implementato per costruire la vocale simulata é il seguente:

```
<< Miscellaneous`Audio`
A = ReadSoundFile["a.wav"][[2]];
fine = 30;
lA = {}; lB = {};
h = 2 π / numeroDivisioni;
lnodi = {};
For[i = 0, i < Length[A], AppendTo[lnodi, -π + i * h]; i++];

For[j = 0, j < fine + 1, j++, {
  AppendTo[lA, 1/π * h * Sum[A[[i]] Cos[j * lnodi[[i]]], {i, Length[A]}] // N];
  AppendTo[lB, 1/π * h * Sum[A[[i]] Sin[j * lnodi[[i]]], {i, Length[A]}] // N];}]

q[t_] = lA[[1]]/2 + Sum[(lA[[f + 1]] * Cos[f t] + lB[[f + 1]] * Sin[f t]), {f, Length[listaA] - 1}];

ListPlot[a]
Plot[q[t], {t, -π, π}]
```

[2]Il primo elemento della lista é identificato dal numero 1

I coefficienti a_n e b_n vengono determinati dalle relazioni (3.9). Quegli integrali, trovandoci in ambiente discreto quale il calcolatore, li calcolo con il metodo dei rettangoli, come si puó denotare dal codice.

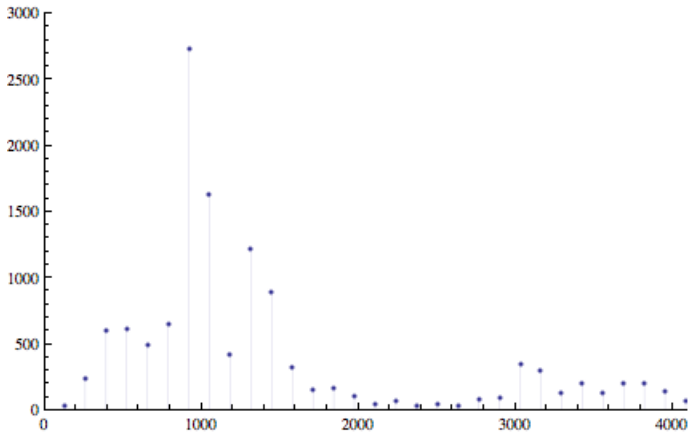
Il comando `Plot[q[t], {t,-Pi,Pi}]` permette di visualizzare il polinomio trigonometrico di Fourier `q[t]` appena determinato, e questo é il grafico:



In questo caso, il miglior modo di misurare la bontá dell'approssimazione sarebbe "suonare" la funzione, e, per fortuna, Mathematica lo permette!
Le vocali maggiormente distinguibili sono indubbiamente la A e la É, per le altre serve un pelino in piú di orecchio, ma alla fine ci si arriva.

4.3 Distinguiamo le vocali...

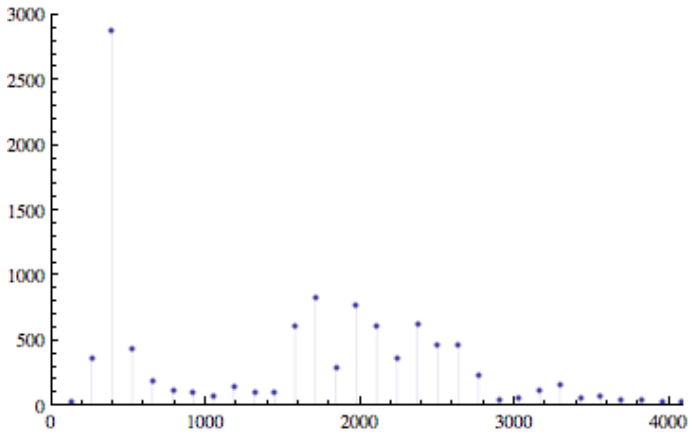
Come possiamo distinguere le vocali? Per distinguere le vocali, basta conoscere le prime due formanti, non una di piú, solamente le prime due.



Questo é lo spettro della nostra A simulata.

Le formanti si possono riconoscere ad occhio nello spettro delle vocali. Esse infatti corrispondono ai due primi picchi che in questo caso corrispondono all'incirca a 900 H_z e 1400 H_z.

Qui invece vediamo applicato alla vocale E ciò che abbiamo visto prima applicato alla vocale A.



Questo é lo spettro della nostra E simulata.

In questo caso le formanti si possono riconoscere a 400 H_z e 1800 H_z.

Ovviamente tutti questi sono dei calcoli approssimativi, che però trovano riscontro in quest'immagine:

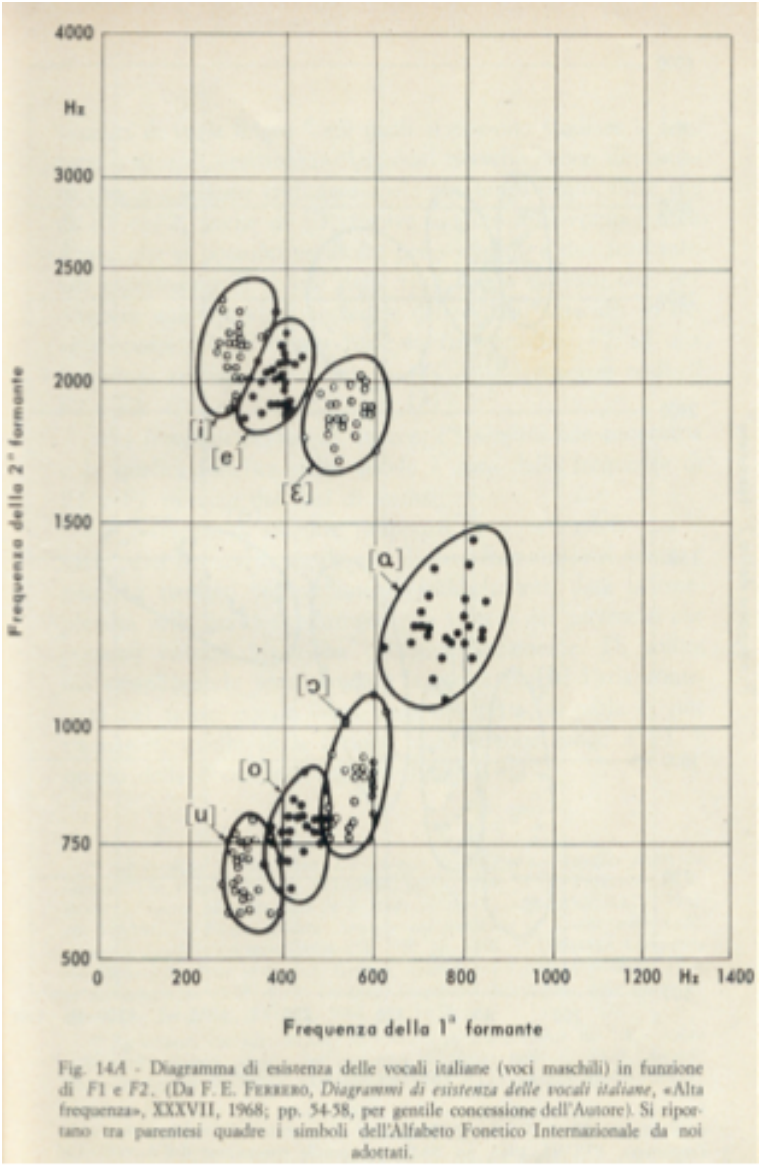


Fig. 14A - Diagramma di esistenza delle vocali italiane (voci maschili) in funzione di F1 e F2. (Da F. E. Ferrero, *Diagrammi di esistenza delle vocali italiane*, «Alta frequenza», XXXVII, 1968; pp. 54-58, per gentile concessione dell'Autore). Si riportano tra parentesi quadre i simboli dell'Alfabeto Fonetico Internazionale da noi adottati.

Essa mostra come sia possibile distinguere le vocali conoscendo la frequenza della prima e della seconda formante. Guardando questo grafico e provando a tracciare le nostre due vocali di cui prima abbiamo calcolato le formanti, si vede che effettivamente esse si trovano devo devono trovarsi, ovvero all'interno dei cerchietti che circoscrivono ogni singola vocale.

Capitolo 5

Compressione di immagini

Dopo lo studio sulla compressione delle vocali, ho pensato che lo stesso principio potesse essere applicato allo studio simulato d'immagini in formato *.png*[1]

Con il software Mathematica 7.0 ho realizzato un programma per mezzo del quale, a partire da un'immagine che viene importata nel software, si sviluppa una procedura che comprime l'immagine e ne simula la costruzione attraverso la DFT (Discrete Fourier Transform). In questo breve articolo presento il lavoro da me fatto e i risultati che ho ottenuto.

5.1 *.png* in Mathematica

Data un'immagine in formato *.png* di $n \times m$ pixel adiacenti (dove m sta per il numero di righe di pixel, n per il numero di pixel in ogni riga e dove ogni pixel é rappresentato come una terna di interi), quando essa viene importata in Mathematica é rappresentata come un vettore di vettori di vettori costituito nel seguente modo:

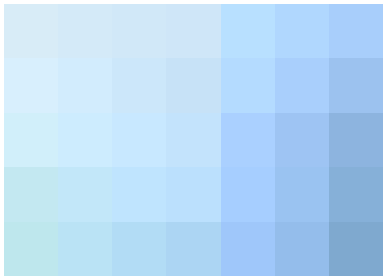
$$\{\ \{\{a_{11}, b_{11}, c_{11}\}, \{a_{12}, b_{12}, c_{12}\}, \ldots, \{a_{1n}, b_{1n}, c_{1n}\}\},$$
$$\{\{a_{21}, b_{21}, c_{21}\}, \{a_{22}, b_{22}, c_{22}\}, \ldots, \{a_{2n}, b_{2n}, c_{2n}\}\},$$
$$\ldots$$
$$\ldots$$
$$\ldots$$
$$\{\{a_{m1}, b_{m1}, c_{m1}\}, \{a_{m2}, b_{m2}, c_{m2}\}, \ldots, \{a_{mn}, b_{mn}, c_{mn}\}\}\ \} \qquad (1)$$

[1] In merito a questo programma ho scritto un articolo che è stato pubblicato su Matematicamente numero 160, uscito il 30 giugno 2011

I numeri a_{ji}, b_{ji}, c_{ji} rappresentano l'intensità delle componenti cromatiche rossa, verde e blu del pixel di posto ji. Questo puó considerarsi il prototipo di qualsiasi immagine in formato *.png* importata in Mathematica.

5.1.1 Esempio

Questa immagine di 5×7 pixel,



importata in mathematica viene cosí rappresentata:

```
{{{216,236,247},{212,234,248},{209,232,248},{207,230,248},{184,224,255},{176,215,254},{168,206,251}},{{
  216,239,253},{210,236,253},{204,231,250},{199,226,247},{180,219,255},{169,207,252},{156,194,239}},{{
  209,239,250},{205,236,254},{200,232,255},{195,227,252},{170,208,255},{158,196,243},{141,180,223}},{{
  195,232,241},{194,231,249},{191,228,254},{187,224,253},{166,206,255},{154,195,241},{134,176,216}},{{
  190,231,237},{186,227,245},{178,220,245},{172,213,243},{159,199,250},{148,189,235},{127,169,207}}}
```

5.2 Algoritmo

Riclassifico il vettore prima presentato in modo tale da formare la seguente nuova terna di vettori A, B, C:

$A = \{\{a_{11}, a_{12}, \ldots, a_{1n}\}, \{a_{21}, a_{22}, \ldots, a_{2n}\}, \ldots, \{a_{m1}, a_{m2}, \ldots, a_{mn}\}\};$
$B = \{\{b_{11}, b_{12}, \ldots, b_{1n}\}, \{b_{21}, b_{22}, \ldots, b_{2n}\}, \ldots, \{b_{m1}, b_{m2}, \ldots, b_{mn}\}\};$
$C = \{\{c_{11}, c_{12}, \ldots, c_{1n}\}, \{c_{21}, c_{22}, \ldots, c_{2n}\}, \ldots, \{c_{m1}, c_{m2}, \ldots, c_{mn}\}\};$

Questa trasformazione (dove il numero di interi contenuti in ognuno dei vettori A, B, C è mn) permette di rendere più agevole l'accesso ai dati nel mio algoritmo, rendendo molto più chiaro il codice.
Il primo elemento (vettore) di A contiene la componente cromatica rossa di tutti i pixel della prima riga, il secondo elemento tutte le componenti rosse della seconda e così via fino all'm-esima riga. La stessa cosa vale anche per il vettore B (verde) e per il vettore C (blu).
Per ricostruire l'immagine originale con la DFT basta allora considerare ogni

elemento della lista A (e in modo analogo per le liste B, C) come campionamento di una funzione f_{Aj} 2π-periodica.
Per l'approssimazione delle f_Aj possiamo utilizzare polinomi trigonometrici nella forma

$$\begin{cases} w_{Aj}(t) = \frac{p_{Aj0}}{2} + \sum_{k=1}^{N} p_{Ajk}\cos(k\cdot t) + q_{Ajk}\sin(k\cdot t) \\ j = 1,\ldots,m \end{cases} \tag{3}$$

dove p_{Ajk} sta a indicare il k-esimo coefficiente del coseno del j-esimo segnale 2π-periodico del vettore A.

Dobbiamo però determinare i coefficienti q_{Ajk} e p_{Ajk}.
Qui entra in gioco Fourier. Infatti, posta h la distanza tra ogni punto di campionamento (questa distanza è equivalente per ogni nostra f_Aj, essendo il numero di campionamenti equivalente per tutti e uguale a n), dove $h = \frac{2\pi}{n}$ i coefficienti q_{Ajk} e p_{Ajk} vengono calcolati come segue:

$$\begin{cases} p_{Ajk} = \frac{1}{\pi}\int_{-\pi}^{\pi} f_{Aj}(t)\cos(kt)\,dt \\ q_{Ajk} = \frac{1}{\pi}\int_{-\pi}^{\pi} f_{Aj}(t)\sin(kt)\,dt \end{cases} \qquad k \in \{0,1,\ldots,N\} \tag{5}$$

Le (5) danno i coefficienti di Fourier esatti delle funzioni f_{Aj}. Mathematica, essendo un software molto sofisticato e molto ben sviluppato, permette di calcolare integrali, se possibile, anche in modo esatto (simbolico). Noi però non abbiamo a disposizione una funzione periodica definita in tutti i punti da passare a Mathematica, bensì una funzione campionata. Calcolandoci gli integrali delle (5) con il metodo dei rettangoli, dove il numero di rettangoli sui quali effettuare il calcolo dell'approssimazione numerica dell'integrale è n (essendo n il numero di campionamenti delle nostre f_{Aj}, la trasformata, nel nostro problema, diventa

$$\begin{cases} p_{Ajk} = \frac{1}{\pi}\cdot h\cdot \sum_{i=1}^{n} a_{ji}\cos\left(k\left(-\pi + h\cdot(i-1)\right)\right) \\ q_{Ajk} = \frac{1}{\pi}\cdot h\cdot \sum_{i=1}^{n} a_{ji}\sin\left(k\left(-\pi + h\cdot(i-1)\right)\right) \end{cases}, \quad k \in \{0,1,\ldots,N\} \tag{6}$$

detta anche Discrete Fourier Transfom (DFT).
Potremmo anche calcolarci i coefficienti con l'algoritmo della Fast Fourier

Transform, che renderebbe molto più veloce tutto il processo, ma per semplicità ho deciso di implementare le (6).
Quanto grande è opportuno prendere N? O meglio, a quale coefficiente conviene fermarsi?
Empiricamente, ho notato che un N opportuno si ottiene da $6m\sqrt{n}$, (7) ma il programma permette anche di poter scegliere un N arbitrario, per poter vedere come cambia l'immagine simulata al variare di N. Il vettore formato dai coefficienti q_{Ajk} e p_{Ajk} costituisce la mia immagine compressa, che posso ricostruire tramite un algoritmo di decompressione.
Infatti, avendo i coefficienti q_{Ajk} e p_{Ajk}, è immediato il calcolo dei $w_{Aj}(t)$ (vedi (3)) e, applicando questo procedimento anche ai vettori B e C, otteniamo questi 3 vettori:

$$\begin{aligned} X &= \{\, w_{A1}(t), w_{A2}(t), \ldots\ldots, w_{Am}(t) \,\} \\ Y &= \{\, w_{B1}(t), w_{B2}(t), \ldots\ldots, w_{Bm}(t) \,\} \\ Z &= \{\, w_{C1}(t), w_{C2}(t), \ldots\ldots, w_{Cm}(t) \,\} \end{aligned} \qquad (8)$$

Costruisco ora, nel modo seguente, la mia immagine decom- pressa:

$$\begin{aligned} \{ \quad & \{\{ w_{A1}(-\pi), w_{B1}(-\pi), w_{C1}(-\pi) \}, \\ & \quad \{ w_{A1}(-\pi+h), w_{B1}(-\pi+h), w_{C1}(-\pi+h) \}, \ldots\ldots, \\ & \quad \{ w_{A1}(-\pi+(n-1)\,h), w_{B1}(-\pi+(n-1)\,h), w_{C1}(-\pi+(n-1)\,h) \}\}, \\ & \{\{ w_{A2}(-\pi), w_{B2}(-\pi), w_{C2}(-\pi) \}, \\ & \quad \{ w_{A2}(-\pi+h), w_{B2}(-\pi+h), w_{C2}(-\pi+h) \}, \ldots\ldots, \\ & \quad \{ w_{A2}(-\pi+(n-1)\,h), w_{B2}(-\pi+(n-1)\,h), w_{C2}(-\pi+(n-1)\,h) \}\}, \\ & \vdots \\ & \vdots \\ & \{\{ w_{Am}(-\pi), w_{Bm}(-\pi), w_{Cm}(-\pi) \}, \\ & \quad \{ w_{Am}(-\pi+h), w_{Bm}(-\pi+h), w_{Cm}(-\pi+h) \}, \ldots\ldots, \\ & \quad \{ w_{Am}(-\pi+(n-1)\,h), w_{Bm}(-\pi+(n-1)\,h), w_{Cm}(-\pi+(n-1)\,h) \}\} \\ \} & \end{aligned}$$

Le immagini seguenti presentano i diversi output del programma al variare di N, con immagine in input costante.

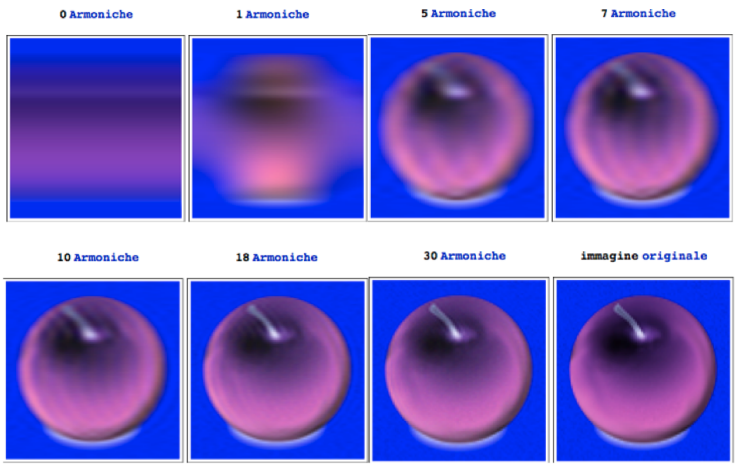


fig.1

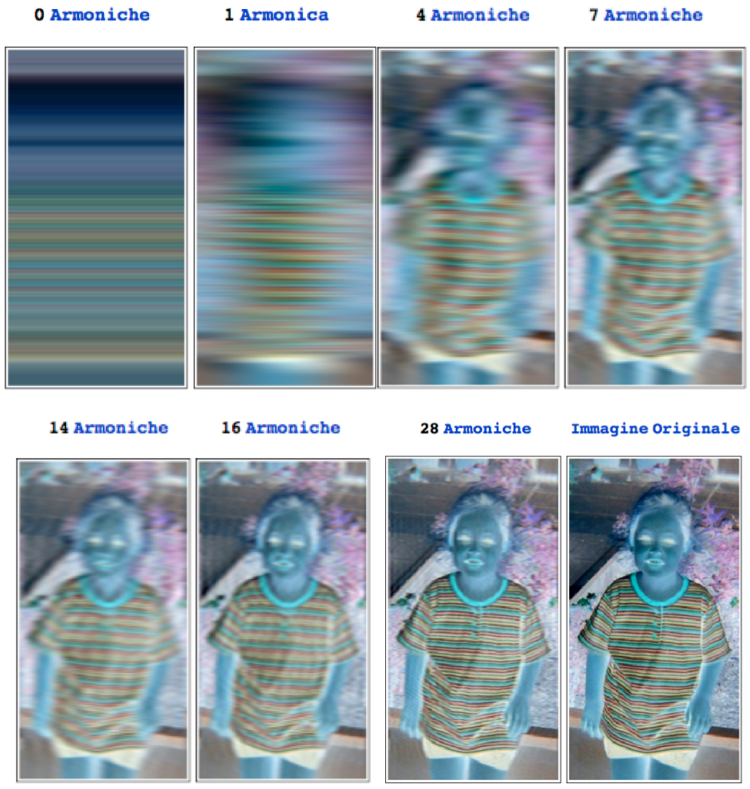


fig.2

Siccome per un'immagine reale non c'è ragione di supporre che $f_{Aj}(-\pi) = f_{Aj}(\pi)$, la funzione periodica f_{Aj} potrebbe risultare discontinua ai bordi dell'intervallo $[\pi, \pi]$: se partiamo da una immagine che non abbia uno sfondo omogeneo, dobbiamo aspettarci degli artefatti di compressione vicino al bordo dell'immagine. Questo effettivamente è ciò che accade, come possiamo

vedere nella figura sottostante (l'immagine in alto è l'originale, quella in basso la sua ricostruzione con il mio algoritmo).

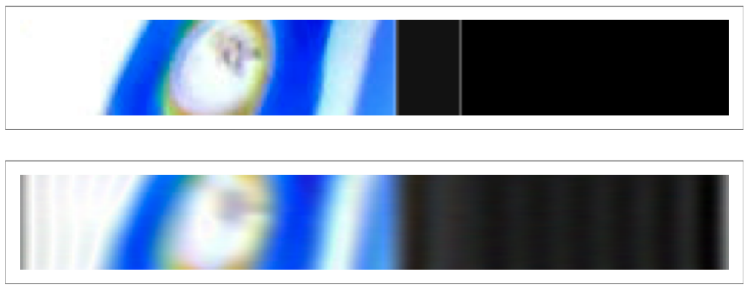


fig.3

Nella parte sinistra dell'immagine simulata, dove dovremmo aspettarci solo il bianco, compare invece una forte componente di grigio. A destra, l'originale nero è mescolato con il bianco. Queste stranezze sono proprio gli artefatti di cui parlavamo prima.
Il mio algoritmo comprime separatamente ciascuna riga dell'immagine.
Gli algoritmi usati nel mondo reale (per esempio .jpeg), invece, comprimono contemporaneamente tutta l'immagine usando una versione bidimensionale della DFT: in questo modo il file compresso risulta notevolmente più piccolo, ma ovviamente l'algoritmo è molto più complicato.
Il vantaggio di questo algoritmo di compressione sta nel fatto che, scegliendo di fermarsi a $\sqrt{n}$ coefficienti, per memorizzare l'immagine ho bisogno di salvare $6m\sqrt{n}$ coefficienti di Fourier, al posto dei $3nm$ numeri interi presenti inizialmente nel vettore (1).
Sostanzialmente, riesco a rimpiazzare il fattore n con un multiplo piccolo di $\sqrt{n}$, cosa molto conveniente per n grande!

Capitolo 6

Corde vibranti

$$\begin{cases} u_{tt} = c^2 u_{xx} & \text{se } 0 < x < \pi, \quad t > 0 \\ u(0,t) = u(\pi,t) = 0 & \text{se } t > 0 \\ u(x,0) = f(x) & \text{se } 0 < x < \pi \\ u_t(x,0) = 0 & \text{se } 0 < x < \pi \end{cases} \tag{6.1}$$

La (6.1) é anche detta *equazione della corda vibrante* (o *equazione delle onde* unidimensionale). Essa rappresenta il movimento di una corda vibrante che a riposo giace sull'asse delle x tra 0 e π:

- $u(x,t)$ rappresenta lo spostamento verticale (rispetto a 0) del punto della corda di ascissa x, al tempo t
- $u(0,t) = u(\pi,t) = 0$ sono le condizioni al contorno e dicono che gli estremi della corda sono fissati
- $f(x)$ rappresenta la *posizione iniziale* della corda
- $u_t(x,0) = 0$ dice che la corda viene lasciata andare con velocitá iniziale nulla (problema della *corda pizzicata*)

Risolvendo questo problema, e ponendo $f(x) = 0$ in π e in 0 é possibile descrivere il moto di una corda al variare del tempo t che ha per posizione iniziale $f(x)$! La cosa interessante é che é possibile scrivere la soluzione del problema come serie trigonometrica del tipo

$$u(x,t) = \sum_{n=1}^{\infty} b_n \sin nx \cos cnt \tag{6.2}$$

con

$$b_n = \frac{2}{\pi} \int_0^{\pi} f(x) \sin x \quad dx \tag{6.3}$$

6.1 Un esempio pratico

Se scegliamo di prendere come $f(x)$ la seguente funzione:

$$f(x) = \begin{cases} \dfrac{x}{c} & \text{se } 0 \leq x \leq c \\ \dfrac{1}{\pi - c}(x - c) + 1 & \text{se } c \leq x \leq \pi \end{cases} \tag{6.4}$$

la soluzione in serie sará (fermata al terzo termine):

$$u(x,t) = 0.79 \cos t \sin x + 0.21 \cos 2t \sin 2x + 0.01 \cos 3t \sin 3x \tag{6.5}$$

Questo rende la rappresentazione grafica da parte di Mathematica molto piú veloce e dimostra come sia possibile risolvere equazioni alle derivate parziali molto complicate tramite le serie di Fourier.

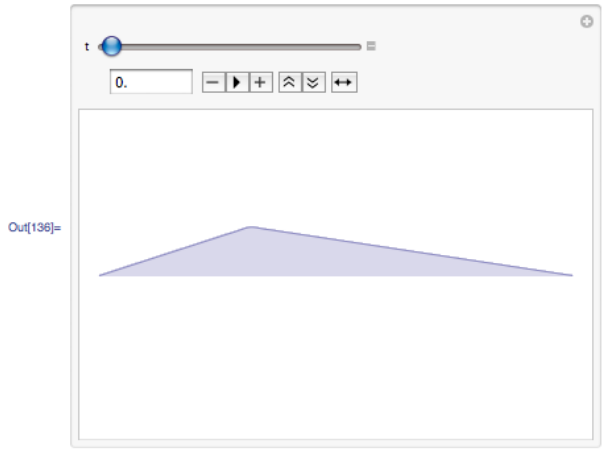
6.1.1 Programma

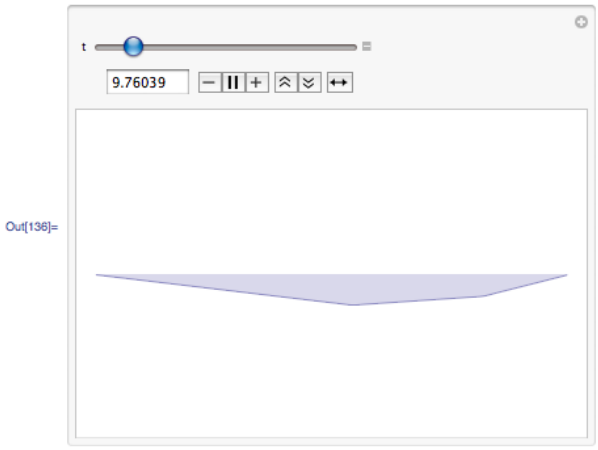
Per l'implementazione in *Mathematica* ho ritenuto oppurtuno utilizzare il comando `Manipulate` per poter avere un'animazione carina ed efficace. Di seguito presento il codice.

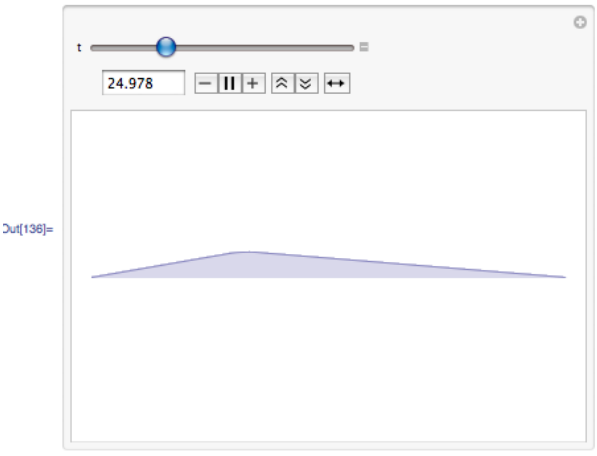
```
f[x_] := x/x0 /; 0 ≤ x ≤ x0; f[x_] := -1/(π - x0) (x - x0) + 1 /; x0 ≤ x ≤ π;
fine = 3; numeroDivisioni = 301;
h = π/numeroDivisioni; l = {}; listaCn = {};
For[i = 0, i < numeroDivisioni, AppendTo[l, i*h]; i++];
For[k = 0, k ≤ fine, k++, {
   AppendTo[listaCn, 2/π h* Sum[f[l[[i]]] Sin[(k + 1) * l[[i]]] // N, {i, 1, numeroDivisioni}]];
  }];
newCn = Chop[listaCn];
u[x_, t_] = Sum[(newCn[[n]] Cos[n t] Sin[n x]), {n, 1, Length[newCn]}];
Manipulate[
 Plot[u[x, t] * E^(-t/39), {x, 0, π}, PlotRange → {{0, π}, {-3, 3}}],
 {t, 0, 50 π}]
```

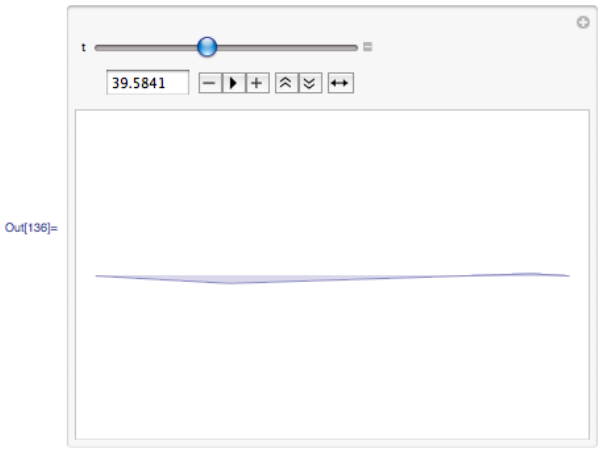
Come si può vedere, all'interno del comando Plot moltiplico la serie $u(x,t)$ per una componente esponenziale molto piccola che decresce all'aumentare di t. Questo per avere un effetto di smorzamento che rende più reale la simulazione di una corda pizzicata.

Queste immagini presentano l'output del programma con in input la (6.4)









Bibliografia

[1] Smirnov Vladimir Ivanovic. *Corso di matematica superiore.* Vol. II, pagg. 466 e segg., Editori Riuniti, Roma, 1977.

[2] Baldo Sisto. *Appunti per il laboratorio di scienza del suono: parte matematica.* Dipartimento di Matematica, Universitá degli Studi di Trento, anno 2010.

[3] Baudoin Marc. *Impara LaTeX!(...e mettilo da parte!).* École Nationale supérieure de techniques avancées (ENSTA), Parigi, 1996.