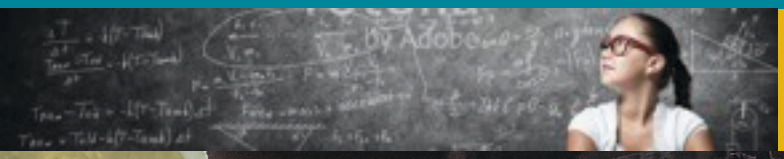
# Cancellazione del rumore

## Un esperimento tutt'altro che insonorizzato!



RESEARCH IN ACTION - RIA
RESEARCHINACTION.IT

In molte occasioni ci è capitato di dover produrre o ascoltare un segnale acustico in un luogo esposto a rumori di vario genere come una stazione ferroviaria, un negozio con musica alta o, più semplicemente, in mezzo al traffico di una congestionata città moderna. Per questo oggi è fondamentale l'utilizzo di sistemi capaci di eliminare il rumore indesiderato al fine di ricavare da un segnale sporco il corrispondente segnale utile pulito, senza interferenze. Il problema posto durante questo percorso è quindi quello di pulire un segnale acustico analogico eliminando il rumore che lo rende di difficile ascolto.

20 - Cancellazione del rumore - 05.23
Revisione 0 del 10.05.23

# RiA - Research in Action

La parola ria in inglese significa estuario, in particolare (dalla definizione che ne da l'Oxford Living Dictonaries):

> A long, narrow inlet formed by the partial submergence of a river valley ... the rias or estuaries contain very peculiar ecosystems which often contain important amounts of fish ... (a causa della loro natura, le rias o estuari contengono ecosistemi molto particolari che spesso contengono grandi quantità di pesce - www.eurotomic.com/spain/the-rias-altas-in-spain.php)

quindi questo prodotto che sarà realizzato grazie all'attività di alternanza scuola-lavoro di alcuni studenti del liceo scientifico G.B.Grassi di Latina - **www.liceograssilatina.org** - sarà un luogo virtuale da esplorare dove *pescare* molto materiale per la didattica laboratoriale.

## Fare scienza

La scienza non è solo identificabile con la formula, il modello, la teoria. In altre parole la scienza non rappresenta solo un corpo di conoscenze organizzate e formalizzate. La scienza è anche e fondamentalmente ricerca. Una ricerca volta a conoscere e a capire sempre più e sempre meglio come è fatto e come funziona questo nostro complicatissimo mondo.

Fare scienza si identifica con l'interrogarsi, con l'indagare ed esplorare fatti e cose. Questo tipo di lavoro i bambini lo fanno spontaneamente sin dalla loro nascita ma si perde nel corso del percorso scolastico. L'intervento educativo deve tener conto di ciò e fornire stimoli, occasioni e strumenti per far acquisire agli studenti capacità sempre più ampie e affinate per poter compiere questo lavoro di indagine mantenendo viva (o risvegliando) la curiosità cognitiva, la voglia di sapere e di scoprire, la fiducia di poter capire.

Pensare in senso creativo, in campo scientifico, significa aggredire i problemi, attivare processi vivi dei pensiero, alimentare l'evoluzione dinamica dell'intelligenza duttile, dell'esercizio dell'intuizione e dell'immaginazione, della capacità di progettare e formulare ipotesi, di controllare e verificare quanto prodotto e ricercato.

Per questo è necessario bandire forme di apprendimento consumate entro schemi rigidi di elaborazione del pensiero e puntare al recupero della congettura, dell'ipotesi, di una coscienza scientifica aperta a interrogare ogni problematica.


La società odierna deve far fronte ad un rinnovamento scientifico e tecnico accelerato in cui lo sviluppo delle conoscenze scientifiche e la creazione di prodotti di alta tecnologia *(hi-tech)*, come anche la loro diffusione subiscono un'accelerazione sempre più rapida.

È necessaria, quindi, una diffusione della conoscenza in genere ed è indispensabile promuovere una nuova cultura scientifica e tecnica basata sull'informazione e sulla conoscenza. E quanto più è solida la base di conoscenze scientifiche scolastiche, tanto più si può approfittare dell'informazione e della conoscenza scientifica e tecnica.

» https://www.facebook.com/Research-in-Action-341307966417448/
» https://www.youtube.com/channel/UC1PA7Zu78RUMBJnkaiOR8kA/

# Sommario dei contenuti

## Cancellazione del rumore - Un esperimento tutt'altro che insonorizzato

**Sommario dei contenuti**



## Materiale disponibile per questo laboratorio:

» il fascicolo (in formato PDF di circa 10MB): **http://researchinaction.it/materials/20-Cancellazione-del-rumore.pdf**;

» il materiale a supporto del laboratorio: il codice XML con l'applicazione sviluppata con l'aiuto di Blockly, il codice Python generato da Blockly, il file GeoGebra con l'esperimento virtuale: **http://researchinaction.it/materials/20-Cancellazione-del-rumore.zip**.

Per il materiale didattico a supporto del fascicolo visitare anche la pagina Download del sito dedicato al progetto: **http://researchinaction.it/download/**.

Per i videotutorial è possibile visitare il canale YouTube del progetto: **https://www.youtube.com/channel/UC1PA7Zu78RUMBJnkaiOR8kA**. In particolare, sul canale YouTube, sono presenti brevi videocorso introduttivi all'uso di xMaxima e di Blockly.


RiA Toolbox Blockly GeoGebra xMaxima G.B. GRassi

# Cancellazione del rumore

## Un esperimento tutt'altro che insonorizzato

# 1. Introduzione

*Questo laboratorio è stato sviluppato da … in collaborazione con il professor Danilo Comminiello (Dipartimento di Informatica, Elettronica e Telecomunicazioni La Sapienza di Roma), il progetto è stato coordinato dal professor Grassucci (Iss G.B. Grassi di Latina).*

In molte occasioni ci è capitato di dover produrre o ascoltare un segnale acustico in un luogo esposto a rumori di vario genere come una stazione ferroviaria, un negozio con musica ad alto volume o, più semplicemente, in mezzo al traffico di una congestionata città moderna. Per questo oggi è fondamentale l'utilizzo di sistemi capaci di eliminare il rumore indesiderato al fine di ricavare da un segnale sporco il corrispondente segnale utile pulito, senza interferenze. Il problema posto durante questo percorso è quindi quello di pulire un segnale acustico analogico eliminando il rumore che lo rende di difficile ascolto.

## 1.1. La cancellazione del rumore

Un sistema di cancellazione del rumore è costituito da due microfoni distanti fra loro che catturano rispettivamente il segnale disturbato, vicino all'ascoltatore, e il rumore vicino alla fonte da cui proviene. Quest'ultimo, in un certo senso, è il rumore puro.

Il primo microfono, quindi, registrerà un segnale *sporcato* che chiameremo $d[n]$, dato dalla sovrapposizione del segnale utile $s[n]$ e dal rumore registrato a distanza maggiore dalla fonte producente $v_1[n]$ ($d[n]$, $s[n]$, $v_1[n]$ sono liste di valori reali, dei campioni di suoni e rumori che, complessivamente, consentono di memorizzare l'audio registrato dai microfoni; qui usiamo gli stessi nomi dei vettori che useremo nel seguito per implementare l'algoritmo). I moderni sistemi audio infatti lavorano su campioni digitali del suono (catturato, trasmesso, riprodotto, …): parecchie volte al secondo si misura e si memorizza l'intensità del suono e quindi l'intero suono è trasformato in una sequenza di valori, una lista, un vettore, di numeri reali, e memorizzato in file secondo uno specifico formato.

L'applicazione che vogliamo realizzare, quindi, non farà altro che manipolare liste di numeri, quindi con $d[n]$, per esempio, intendiamo una lista di valori, positivi o negativi, che possono essere gestiti proprio come numeri, come detto poco fa!


Il secondo microfono, invece, registrerà il rumore più vicino alla sorgente. La lista di valori corrispondente la chiameremo $v_2[n]$. Nonostante $v_1[n]$ e $v_2[n]$ provengano dalla stessa sorgente di rumore e contengano informazioni simili, la loro forma d'onda è diversa perché la distanza dal microfono primario, le riflessioni delle onde acustiche nell'ambiente e gli oggetti che ostacolano il percorso tra i due microfoni modificano il rumore stesso quindi non è possibile ottenere il segnale pulito $s[n]$ semplicemente sottraendo al segnale registrato dall'ascoltatore $d[n]$ il rumore rilevato dal secondo microfono $v_2[n]$.

Il risultato che vogliamo ottenere *cancellando il rumore*, $e[n]$, è quindi il segnale ripulito dal rumore chiamato *segnale di errore.*

Ciò che rende diversi i due segnali di rumore $v_1[n]$ e $v_2[n]$, pur provenienti dalla stessa sorgente, è il percorso acustico, o risposta impulsiva acustica $h[n]$, non perfettamente calcolabile ma solo stimabile approssimativamente attraverso un filtro $w[n]$ *convoluto* con $v_2[n]$. $v_1[n]$ corrisponde infatti al risultato della convoluzione tra tra $v_2[n]$ e $h[n]$.

## 1.2. Prerequisiti

### Competenze base

» conoscenze minime del calcolo matriciale
» conoscenze minime di rappresentazione di audio digitali
» capacità di tradurre in un linguaggio abbastanza formale una procedura, un algoritmo
» competenze base di programmazione (vedi box qui a lato)

### Soft skills

» capacità di problem solving
» capacità di lavoro di gruppo

Sul sito del progetto Research in Action è descritto un approccio semplice al coding: http://researchinaction.it/2022/06/25/programmare-senza-un-linguaggio-di-programmazione/. In particolare, faremo uso del generatore di codice Blockly: https://developers.google.com/blockly/. Sul sito è disponibile un breve corso introduttivo a Blockly: http://researchinaction.it/didattica/introduzione-a-blockly/.

## 1.3. Obiettivi

L'obiettivo del nostro progetto è elaborare un software capace di ripulire un segnale disturbato dal rumore proveniente da una seconda sorgente migliorandone la qualità.

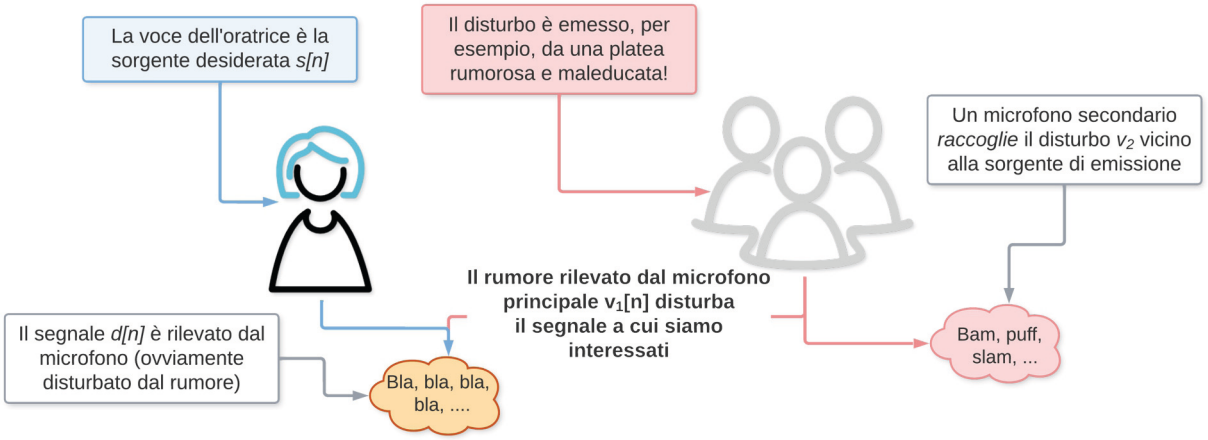
Se non siete avvezzi con la programmazione, non temete, seguiremo quello che noi chiamiamo un *approccio semplice* al *coding*, tre step che vi porteranno a realizzare il vostro programma per la cancellazione del rumore:

» sarete guidati alla scrittura di una o più procedure in linguaggio naturale (in italiano) cercando di rendere ogni frase quanto più possibile chiara e non ambigua (non necessariamente alla prima stesura, ci arriveremo per approssimazioni successive);
» le procedure saranno implementate utilizzando il generatore di codice Blockly, l'applicazione è testata all'interno del generatore di codice fino ad essere sicuri del suo funzionamento
» infine realizzeremo l'app copiando il codice Python generato da Blockly in un interprete e funzionerà senza bisogno di ulteriori interventi!

# 2. Al lavoro

Il problema della cancellazione del rumore è proprio la risposta impulsiva, altrimenti sarebbe sufficiente sottrarre al segnale rilevato il rumore registrato da un microfono secondario posto vicino alla sorgente di disturbo. Purtroppo, però, il disturbo, il rumore, viene modificato dall'ambiente e quando è rilevato dal microfono principale non è più lo stesso.

Lo scopo dell'algoritmo è quello di *costruire* una stima di questo rumore modificato avendo a disposizione quello originale, non modificato.




## 2.1. L'algoritmo in sintesi

L'algoritmo deve analizzare ogni campione del suono che abbiamo rilevato d[n], per ogni campione dobbiamo:

» aggiornare il buffer $v_2$temp che ci servirà per la convoluzione (il buffer è una sottolista del rumore rilevato $v_2$);
» applicare la convoluzione (la moltiplicazione del vettore riga w che contiene il filtro per il vettore colonna $v_2$temp che contiene il buffer), il risultato della convoluzione $v_1$stima è la stima del rumore da sottrarre al suono rilevato per ottenere quello pulito;
» calcolare il campione del suono pulito come differenza tra il suono rilevato d[n] e la stima del rumore e ...
» aggiornare il filtro.

## 2.2. I primi passi per la scrittura del codice

Tenete presente che ...

» Vogliamo scrivere un programma che non ci induca a variare più volte i valori numerici dei segnali di rumore o da ripulire ogni volta che vogliamo analizzarne di nuovi. Potrebbe esservi molto utile utilizzare delle variabili da attribuire ad ognuno di essi, cambieranno in automatico variando solo il blocco che attribuisce loro il valore iniziale.
» Questo filtro di cancellazione del rumore puó essere considerato una rete neurale a tutti gli effetti in quanto, iterazione per iterazione, aggiorna i suoi coefficienti finchè non riesce ad approssimare la risposta impulsiva h[n] stimata. Il filtro, quindi, impara di una *learning rate* μ che assume valori molto piccoli, vicini allo zero.
» Scriveremo un codice che sia capace di analizzare una situazione di prova (sostanzialmente costruita appositamente *a tavolino*), non un codice che utilizzi dati reali a noi forniti quindi creeremo artificialmente il nostro segnale sporcato. Successivamente applicheremo la nostra

implementazione a una situazione reale.

» Non abbiate paura di sbagliare, potete provare e riprovare più volte a far girare il programma, tutte le operazioni sono reversibili però ... salvate spesso il vostro lavoro!

il Toolbox suggerisce alcuni metodi per formalizzare una procedura con poche, pochissime, conoscenze di programmazione (cfr. 6. Algoritmi a pagina 24 - http://researchinaction.it/wp-content/uploads/2019/01/00-Toolbox.pdf). Sul sito del progetto si introduce un approccio morbido al coding (cfr. Programmare senza un linguaggio di programmazione: http://researchinaction.it/2022/06/25/programmare-senza-un-linguaggio-di-programmazione/)

## 2.3. Somma di vettori

Ora cercheremo di guidarvi verso la realizzazione del nostro filtro, passo dopo passo, seguiteci anche se non avete mai programmato e non conoscete quindi nessun linguaggio di programmazione. Iniziamo da una funzione semplice, anche apparentemente inutile, che ci servirà solo nei primi test ma che si presta bene come *sottoproblema* introduttivo.

**Cosa serve perché la procedura funzioni? In altre parole, cosa si deve fornire come input alla funzione SommaVettori?**

Ricorda che, per esempio, non è necessario indicare quanti elementi hanno le liste, i vettori, perché è un'informazione che si può ricavare conoscendo il vettore (ogni linguaggio di programmazione ha una specifica funzione per *contare* il numero degli elementi di una lista o la lunghezza di un vettore).

**Una volta stabilito cosa passare alla funzione per operare, concentra l'attenzione sul cuore della procedura, su un singolo passo, su una sola interazione. Cosa deve fare (ripetutamente) la procedura? Il risultato dell'operazione dove deve essere conservato, memorizzato?**

Se hai compreso il processo, dovresti aver scritto una o due righe solamente che descrivono una semplice operazione: la somma di due elementi corrispondenti (... che hanno lo stesso indice ...) dei due vettori. Una buona idea è quella di scrivere questa piccola porzione di codice in modo che sia *quanto più generale possibile*!

**Questa operazione basilare deve essere ripetuta più volte, per tutti gli elementi dei due vettori. Cerca di formalizzare questo concetto, come se racchiudessi le operazioni base dentro un contenitore più grande che conta le ripetizioni.**

La parola cruciale è *ripetuta*: la stessa operazione va eseguita più volte cambiando ogni volta l'indice dell'elemento nelle liste. Ora la procedura dovrebbe essere pronta o quasi …


**Ricorda che è necessario definire l'output dell'algoritmo. Cosa deve restituire la funzione SommaVettori? Qual è il risultato che ci aspettiamo?**

Un'ultima osservazione: probabilmente avrai dovuto usare una variabile in cui mettere da parte la somma delle addizioni mano a mano che veniva calcolata, se non lo hai ancora fatto, sarebbe una buona idea farlo ora.

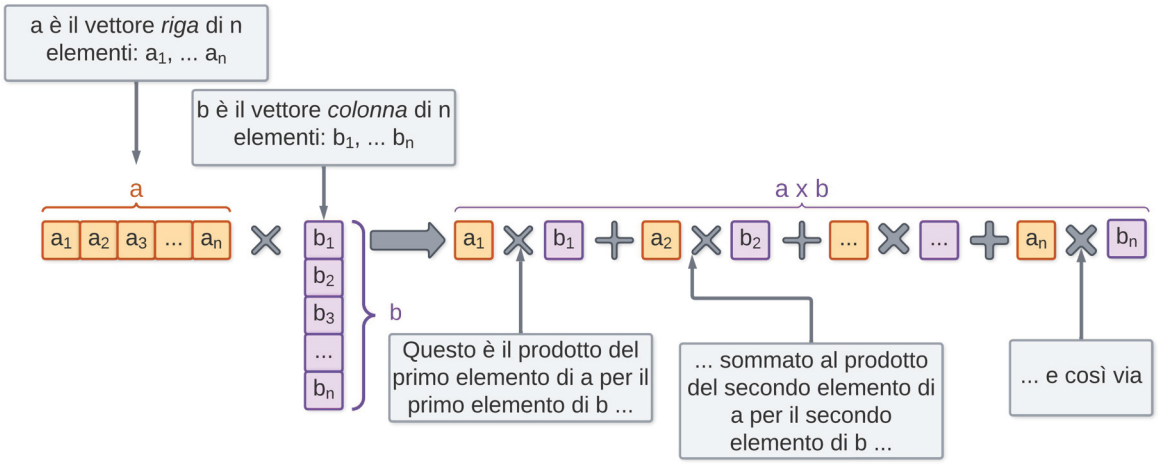
**Hai pensato a inizializzare questa variabile prima di avviare il ciclo? In altre parole, che valore deve avere questa variabile prima di iniziare a ripetere il cuore della procedura?**

Ora ci dovremmo essere. Se hai dubbi prova a leggere la nostra proposta di soluzione **(cfr. Soluzioni a pagina 15)**.

## 2.4. Convoluzione

Passiamo ora a qualcosa di molto simile, probabilmente anche più facile. Una delle operazioni fondamentali delle reti neurali, a cui somiglia il nostro algoritmo, è quella di *convoluzione*: si tratta di

moltiplicare un vettore (immaginato come una lista di numeri disposta su una riga) per un vettore colonna (pensato come una lista di numeri sistemati su una colonna).




Abbiamo un algoritmo, quello per moltiplicare due vettori, due liste, elemento per elemento (moltiplicazione riga per colonna) che sostanzialmente a ogni iterazione somma il prodotto dell'i-esimo elemento del primo vettore per l'i-esimo elemento del secondo vettore. Ora dobbiamo trasformare questa procedura in una funzione, chiamiamola Convoluzione, in cui ogni istruzione sia elementare e non ambigua anche se in un linguaggio naturale come l'italiano (se ancora non lo hai fatto, può essere di aiuto seguire l'esempio riportato sul sito del progetto, vedi il box a sinistra nella pagina precedente).

**Cosa serve perché la procedura possa operare correttamente? In altre parole, cosa si deve fornire come input alla funzione *Convoluzione*?**

Ricorda che, per esempio, non è necessario indicare quanti elementi hanno le liste, i vettori, perché è un'informazione che si può ricavare conoscendo il vettore (un po' come già visto in precedenza per la somma di due vettori **(cfr. 2.2 Somma di vettori a pagina 8)**.

**Una volta stabilito cosa passare alla funzione, concentra l'attenzione sul cuore della procedura, su un singolo passo, su una sola interazione. Cosa deve fare (ripetutamente) la procedura? Il risultato dell'operazione dove deve essere conservato, memorizzato?**

Se hai compreso il processo, dovresti aver scritto una o due righe solamente che descrivono una semplice operazione che somiglia molto, ma non è identica, a quella già vista per la somma.


**L'operazione basilare deve essere ripetuta più volte, per tutti gli elementi dei due vettori. Cerca di formalizzare questo concetto, come se *racchiudessi* le operazioni base dentro un contenitore più grande che conta le ripetizioni.**

Ora la procedura dovrebbe essere pronta o quasi, dobbiamo aggiungere solo un paio di *cosette*.

**Ricorda che è necessario definire l'output dell'algoritmo. Cosa deve restituire la funzione *Convoluzione*? Qual è il risultato che ci aspettiamo?**

Suggerimento: probabilmente hai dovuto usare una variabile dove mettere da parte la somma dei prodotti via via che veniva calcolata.

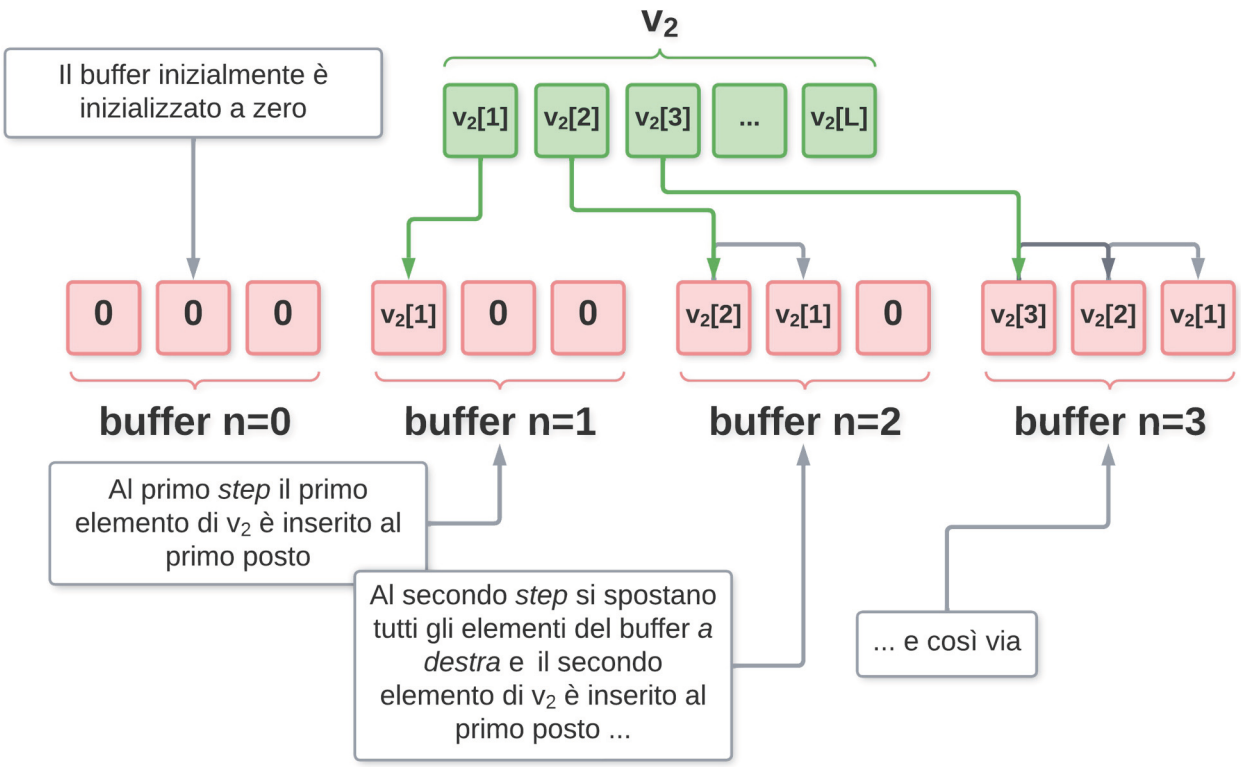
**Hai pensato a inizializzare questa variabile prima di avviare il ciclo?**

Ora ci dovremmo essere: il risultato della convoluzione è la stima del segnale di rumore che potremo utilizzare per *correggere* il segnale che abbiamo rilevato $d[n]$ in modo da ottenere il segnale *pulito*. Se hai dubbi prova a leggere la nostra proposta di soluzione **(cfr. 3.2 Convoluzione a pagina 16)**.

## 2.5. Aggiornamento Buffer

Il fine del buffer è la convoluzione con $v_2[n]$ per calcolare la stima del rumore di disturbo $v_1[n]$. Viene aggiornato ad ogni iterazione (come abbiamo già detto è una quasi-rete neurale).

La stima di $v_1[n]$ viene fatta su campioni del segnale di rumore $v_2[n]$ di lunghezza $M < L$ chiamati buffer. Quindi ogni buffer (ce n'è uno per ogni *step* dell'algoritmo) è una sottolista di M valori della lista $v_2[n]$ quindi, per esempio, se $L = 10$ e $M = 3$, al passo tre il buffer sarà una lista dei primi tre valori di $v_2[n]$ (il terzo, il secondo, il primo), al quarto *step* dell'algoritmosarà una lista dei valori dal quarto al secondo ... e così via, seguendo questo schema:





**Cosa serve perché la procedura funzioni? In altre parole, cosa si deve fornire come input alla nostra funzione di aggiornamento (che chiameremo, con poca fantasia ma grande chiarezza, AggiornamentoBuffer)?**

Ricorda che, per esempio, non è necessario indicare quanti elementi hanno le liste, i vettori, perché è un'informazione che si può ricavare conoscendo il vettore.

**Una volta stabilito cosa passare alla funzione (il suo input), concentra l'attenzione sul *cuore* della procedura. Cosa deve fare (ripetutamente) la procedura?**

In questa parte ti può essere utile ricordare qual è lo scopo del buffer (che abbiamo illustrato qui sopra). Quest'ultimo si *muove* tutte le volte di una posizione lungo la lista $v_2$, quindi, ad ogni successiva iterazione i valori, gli elementi, del buffer saranno spostati indietro (verso *sinistra*) di una

posizione per liberare l'ultima posizione del buffer che conterrà il successivo elemento. In questo modo il primo sarà eliminato ma non ne avremo più bisogno.

Questa operazione deve essere eseguita un elemento alla volta, partendo dal secondo per arrivare all'ultimo. Abbiamo già visto che queste ripetizioni possono essere implementate con un ciclo che racchiude lo *spostamento* di cui abbiamo parlato.

**Ora dobbiamo inserire, come ultimo elemento del buffer il valore del campione successivo: prendere dal vettore $v_2$ l'elemento corrispondente allo step dell'algoritmo e metterlo nel buffer in ultima posizione! Operazione che non dovrebbe presentare difficoltà.**

Ora la procedura dovrebbe essere pronta o quasi …

**Ricorda che è necessario definire l'output dell'algoritmo. Cosa deve restituire la funzione *AggiornamentoBuffer*? Qual è il risultato che ci aspettiamo?**

Ora ci dovremmo essere. Se hai dubbi prova a leggere la nostra proposta di soluzione **(cfr. 3.3 Aggiornamento Buffer a pagina 17, nelle Soluzioni)**.

## 2.6. Aggiornamento del filtro

Il filtro w va calcolato, aggiornato, per ogni campione del suono da processare. Il calcolo può sembrare complicato, ma non spetta a noi calcolarlo ogni volta, dobbiamo solo indurre l'algoritmo a eseguire le giuste operazioni.

**Cosa serve perché la procedura funzioni? In altre parole, cosa si deve fornire come input alla funzione *AggiornamentoFiltro*?**

L'aggiornamento del vettore del filtro di cancellazione è

$$w[n] = w[n-1] + \mu \cdot e[n] \cdot v_2[n]$$

dove w[n-1] è la lista dei campioni del filtro all'iterazione precedente mentre e[n] è il valore del suono ripulito, processato, a questo passo dell'algoritmo e μ è un valore costante. Ora dovrebbe essere più chiaro l'input da fornire alla funzione.

**Una volta stabilito cosa passare alla funzione, concentra l'attenzione sul cuore della procedura. Cosa deve fare (ripetutamente) la procedura? Il risultato dell'operazione dove deve essere conservato, memorizzato?**


Facendo riferimento alla formula qui sopra, dovresti aver scritto una semplice riga di codice che racchiuda la formula stessa e assegnare il risultato di questo calcolo a una variabile temporanea. Subito dopo è necessario inserire questo valore nella corrispondente posizione in modo che il filtro non contenga più i valori precedenti ma quelli aggiornati.

**Questa operazione, questo calcolo, e la successiva assegnazione devono essere ripetute più volte, per ciascun elemento del filtro, della lista w.**

È chiaro che si tratta di definire un ciclo (abbiamo incontrato questo tipo di procedura più volte in questo percorso) che *scorra* tutti gli elementi della lista w.

**Ricorda che è necessario definire l'output dell'algoritmo. Cosa deve restituire la funzione *AggiornamentoFiltro*? Qual è il risultato che ci aspettiamo?**

Ora la funzione dovrebbe essere completa. Se hai dubbi prova a leggere la nostra proposta di soluzione (cfr. Soluzioni al termine del fascicolo).

## 2.7. Un ultimo passo: la funzione principale

Ci siamo quasi. Ora dobbiamo *montare* tutti i pezzetti che abbiamo costruito in una funzione, una procedura, che implementi l'algoritmo. La chiameremo *CancellazioneRumore*!

**Per prima cosa devi precisare, come già fatto più volte, l'input della funzione: tutte le informazioni e i dati che dobbiamo passare perchè si riesca a ottenere il segnale ripulito.**

Ora la funzione vera e propria. Ci sara bisogno di una fase di inizializzazione, creare le liste che ci servono e impostare ogni elemento al valore iniziale. Attenzione: fino a ora non l'abbiamo mai nominata, ma ci serve anche una lista $e[n]$ in cui memorizzare, passo dopo passo, il segnale pulito!

**Scrivi le istruzioni, sempre in linguaggio naturale ma preciso e non ambiguo, che servono per creare le strutture dati e assegnare loro i valori iniziali.**

Ora ci serve un ciclo! Dobbiamo analizzare uno a uno i campioni del suono *sporco* e applicare il filtro.

**Definisci un ciclo che percorra la lista, il vettore d, quello che contiene i campioni del suono da ripulire.**

Ora ricorda: l'algoritmo richiede che, a ogni passo, sia:

» aggiornato il buffer,
» applicata la convoluzione,
» calcolato il campione pulito e ...
» aggiornato il filtro.

Ma abbiamo una funzione apposita per ciascuna di questi *step* (a parte il calcolo del campione pulito che vedremo tra poco).


**Specifica come richiamare la funzione *AggiornamentoBuffer* che abbiamo creato in precedenza e a quale variabile assegnare il risultato dell'aggiornamento.**

Ancora qualche passo ... si tratta di usare il buffer per applicare la convoluzione.

**Indica come applicare la funzione *Convoluzione*, implementata proprio per essere utilizzata in questo step dell'applicazione. Calcola il campione del segnale ripulito utilizzando la formula corrispondente.**

Ricorda che il risultato della funzione *Convoluzione* è la stima del campione di rumore che, quindi, possiamo sottrarre al segnale rilevato per ottenere quello ripulito.

**Scrivi una sola riga, sempre in linguaggio naturale ma precisa e non ambigua, per il calcolo del segnale pulito.**

Questa parte è facile. Una volta ottenuta la stima del segnale di rumore $v_1[n]$ si può ottenere il segnale pulito $e[n]$ sottraendo la stima del rumore al campione del suono rilevato $d[n]$:

$$e[n] = d[n] - v_{1temp}[n]$$

Ultimo passo, il filtro.

**Abbiamo una funzione che aggiorna, a ogni *step*, il filtro. Coma va usata? A chi va assegnato il risultato?**

Abbiamo concluso.

**Quali dati deve restituire la funzione?**

La risposta dovrebbe essere ovvia (proprio qualche pagina fa abbiamo risordato di creare una lista per memorizzare il segnale pulito, **cfr. 2.6 Un ultimo passo: a funzione principale a pagina 12**).

Bene! Il laboratorio termina qui. Confronta la tua schematizzazione del filtro per la cancellazione del rumore con le soluzioni che trovi nel seguito.

## 2.8. Un test virtuale

Per provare l'algoritmo *sul campo*, in una situazione reale, potrebbe essere utile creare un esperimento per così dire virtuale, costruito a tavolino.

**Usando un foglio di calcolo o un software CAS create una tabella simile a quella che trovate qui di seguito le cui colonne contengono alcuni dati sperimentali (virtuali, come preannunciato):**

» *t*: il tempo, a ogni istante corrisponderà un campione di suono o rumore;
» *s(t)*: il segnale, per i nostri scopi supponiamo di registrare il suono di un diapason che emette una sola specifica nota, suono che si può rappresentare come una sinusoide s(t) = A sin(t) e per ogni riga della tabella calcoliamo il valore della sinusoide nel corrispondente istante di tempo;
» $v_2(t)$: i campioni del disturbo come se fossero rilevati da un microfono, per quello che vogliamo fare qui potremmo costruire una funzione periodica più complicata della precedente e riempire la colonna con i valori di questa funzione nei corrispondenti istanti;
» $v_2(t)$ *random*: gli stessi campioni della colonna precedente modificati casualmente per simulare la risposta impulsiva;
» *d(t)*: la somma del segnale *s(t)* e del rumore simulato $v_2(t)$;
» *e(t)*: l'ultima colonna resterà inizialmente vuota, la riempiremo con i valori forniti dal nostro programma, dovrebbe rappresentare il suono ripulito e quindi ricostruire il segnale *s(t)* originale.

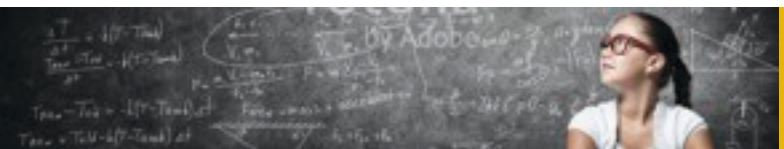

Insomma, qualcosa di simile a questo **(una tabella già completa si puo trovare a pagina 22)**.

| t | s(t) | $v_2$(t) | $v_2$(t) random | d(t) | e |
|---|---|---|---|---|---|
| 0.0 | 0.00000 | 3.00000 | 2.51244 | 2.51244 | |
| 0.2 | 2.86942 | 2.55383 | 2.23796 | 5.10738 | |
| 0.4 | 3.99829 | 3.38587 | 3.65475 | 7.65304 | |
| ... | ... | ... | ... | ... | |

Ora lanciate il programma con i dati virtuali, raccogliete i valori che otterrete in output e confrontateli con il segnale originale ... se, magari dopo qualche istante di tempo, le due liste saranno più o meno simili avremo una prima, parziale, verifica del funzionamento dell'algoritmo.

Il passo successivo sarà una prova sul campo, con il suono registrato da due microfoni, il primario e il secondario, ma questo sarà argomento di un articolo sul nostro sito!


14




# 3. Soluzioni

## 3.1. Somma di vettori

Vediamo di scrivere una breve funzione - in linguaggio naturale ma preciso e non ambiguo - che useremo nel nostro programma. Usate questo passaggio come esercizio, come esempio per capire come lavorare in seguito.

Sul sito del progetto Research in Action è descritto un approccio semplice al coding: http://researchinaction.it/2022/06/25/programmare-senza-un-linguaggio-di-programmazione/. In particolare, faremo uso del generatore di codice Blockly: https://developers.google.com/blockly/. Sul sito è disponibile un breve corso introduttivo a Blockly: http://researchinaction.it/didattica/introduzione-a-blockly/.

**Cosa serve perché la procedura funzioni? In altre parole, cosa si deve fornire come input alla funzione *SommaVettori*?**

La funzione deve operare su due vettori e questo sarà proprio l'input che ci serve. È bene precisare, ci sarà utile nel seguito, che i due vettori (le due liste) hanno lo stesso numero di elementi e quindi la stessa lunghezza. Il codice dovrebbe essere simile a questo:

```
SommaVettori(a, b)
      ...
```

dove *a* e *b* saranno proprio le due liste.

**Una volta stabilito cosa passare alla funzione per funzionare, concentra l'attenzione sul cuore della procedura, su un singolo passo, su una sola interazione. Cosa deve fare (ripetutamente) la procedura? Il risultato dell'operazione dove deve essere conservato, memorizzato?**

Si tratta di sommare l'i-esimo elemento della lista *a* con l'i-esimo elemento (stesso indice) della lista *b*. L'elemento lo inseriamo come ultimo della lista *somma* che conterrà il risultato, la lista con la somma dei due vettori su cui stiamo lavorando (questo inserimento come ultimo elemento sarà più chiaro tra breve).

```
SommaVettori(a, b)
      ...
            in somma metti a[i] + b[i] come ultimo elemento
      ...
```


dove con a[i] indichiamo l'i-esimo elemento della lista *a* e con b[i] il corrispondente elemento del vettore *b*.

**L'operazione basilare deve essere ripetuta più volte, per tutti gli elementi dei due vettori. Cerca di formalizzare questo concetto, come se racchiudessi le operazioni base dentro un contenitore più grande che conta le ripetizioni.**

Dobbiamo ripetere l'operazione per ogni valore di i, dal primo elemento (diciamo che contiamo gli elementi indicando con *uno* il primo) all'ultimo. Vogliamo

```
SommaVettori(a, b)
      ...
      ripeti per i = 1 a lunghezza di a
```

```
            in somma metti a[i] + b[i] come ultimo elemento
      ...
```

**Ricorda che è necessario definire l'output dell'algoritmo. Cosa deve restituire la funzione *SommaVettori*? Qual è il risultato che ci aspettiamo?**

Dobbiamo fare in modo che la funzione restituisca la variabile contenente la somma dei due elementi corrispondenti dei due vettori, *somma*.

```
SommaVettori(a, b)
      ...
      ripeti per i = 1 a lunghezza di a
            in somma metti a[i] + b[i] come ultimo elemento
      restituisci somma
```

**Hai pensato a inizializzare questa variabile prima di avviare il ciclo? In altre parole, che valore deve avere questa variabile prima di iniziare a ripetere il cuore della procedura?**

Eh, già! La variabile *somma* deve essere inizializzata e visto che la *costruiremo* via via, inserendo un elemento alla volta, la poniamo uguale a una lista vuota:

```
SommaVettori(a, b)
      somma = lista vuota
      ripeti per i = 1 a lunghezza di a
            in somma metti a[i] + b[i] come ultimo elemento
      restituisci somma
```

Ora abbiamo la nostra funzione. A questo punto puoi cercare di tradurla usando un generatore di codice come Blockly e scrivere un brevissimo programma per controllarne il funzionamento.

## 3.2. Convoluzione

**Cosa serve perché la procedura funzioni? In altre parole, cosa si deve fornire come input alla funzione Convoluzione?**

Somiglia a quanto fatto per la somma:

```
Convoluzione(a, b)
      ...
```


dove, come già visto, *a* e *b* sono le due liste.

**Una volta stabilito cosa passare alla funzione, concentra l'attenzione sul cuore della procedura, su un singolo passo, su una sola interazione. Cosa deve fare (ripetutamente) la procedura? Il risultato dell'operazione dove deve essere conservato, memorizzato?**

Si devono moltiplicare tra loro due elementi corrispondenti delle due liste e sommare questo risultato al prodotto dei precedenti:

```
Convoluzione(a, b)
      ...
            somma = somma + a[i] * b[i]
      ...
```

**L'operazione basilare deve essere ripetuta più volte, per tutti gli elementi dei due vettori. Cerca di formalizzare questo concetto, come se racchiudessi le operazioni base dentro un contenitore più grande che conta le ripetizioni.**

Racchiudiamo semplicemente questa istruzione in un ciclo che opera su tutti gli elementi delle due liste:

```
Convoluzione(a, b)
      ...
      ripeti per i = 1 a lunghezza di a
            somma = somma + a[i] * b[i]
      ...
```

**Ricorda che è necessario definire l'output dell'algoritmo. Cosa deve restituire la funzione Convoluzione? Qual è il risultato che ci aspettiamo?**

Il risultato che ci serve è proprio la somma dei prodotti che abbiamo calcolato nel ciclo.

```
Convoluzione(a, b)
      ...
      ripeti per i = 1 a lunghezza di a
            somma = somma + a[i] * b[i]
      restituisci somma
```

Attenzione: probabilmente hai dovuto usare una variabile dove mettere da parte la somma dei prodotti via via che veniva calcolata.

**Hai pensato a inizializzare questa variabile prima di avviare il ciclo?**

La *somma*, inizialmente, deve essere chiaramente nulla per non modificare in modo imprevedibile la convoluzione dei due vettori.

```
Convoluzione(a, b)
      somma = 0
      ripeti per i = 1 a lunghezza di a
            somma = a[i] + b[i]
      restituisci sommaù
```

Ricorda, con poco, pochissimo lavoro puoi testare questa funzione in Blockly così da verificare la correttezza del risultato.


## 3.3. Aggiornamento Buffer

**Cosa serve perché la procedura funzioni? In altre parole, cosa si deve fornire come input alla nostra funzione di aggiornamento (che chiameremo, con poca fantasia ma grande chiarezza, AggiornamentoBuffer)?**

Abbiamo bisogno di conoscere sicuramente il passo dell'algoritmo di cancellazione del rumore in cui ci troviamo, diciamo che lo chiamiamo n. Ci servono anche la lista del suono campionato $v_1$ e, chiaramente, il buffer da aggiornate buffer. Quindi l'intestazione della nostra funzione è un poco più complicata di quanto visto fino a ora:

```
AggiornamentoBuffer(n, v2, buffer)
      ...
```

**Una volta stabilito cosa passare alla funzione (il suo input), concentra l'attenzione sul *cuore* della procedura. Cosa deve fare (ripetutamente) la procedura?**

Si tratta di spostare ogni elemento, a partire dall'ultimo, nella posizione successiva.

```
AggiornamentoBuffer(n, v2, M)
      ripeti per j = lunghezza di buffer a 2 con passo -1
            buffer[j] = buffer[j - 1]
      ...
```

dove con la frase ... `con passo -1` ... intendiamo dire che il ciclo si ripeterà dall'ultimo elemento fino al secondo *sottraendo* uno al contatore *j* a ogni iterazione (o, in altre parole, aggiungendo uno).

**Ora dobbiamo inserire, come ultimo elemento del buffer il valore del campione successivo. Operazione che non dovrebbe presentare difficoltà ...**

In pratica, l'ultimo elemento del buffer è il campione del segnale $v_2$ attualmente processato.

```
AggiornamentoBuffer(n, v2, M)
      ripeti per j = lunghezza di buffer a 2 con passo -1
            buffer[j] = buffer[j - 1]
      buffer[1] = v2[n]
      ...
```

**Ricorda che è necessario definire l'output dell'algoritmo. Cosa deve restituire la funzione AggiornamentoBuffer? Qual è il risultato che ci aspettiamo?**

Beh! Semplice, dobbiamo restituire il buffer aggiornato.

```
AggiornamentoBuffer(n, v2, M)
      ripeti per j = lunghezza di buffer a 2 con passo -1
            buffer[j] = buffer[j - 1]
      buffer[1] = v2[n]
      restituisci buffer
```

## 3.4. Aggiornamento del filtro


**Cosa serve perché la procedura funzioni? In altre parole, cosa si deve fornire come input alla funzione *AggiornamentoFiltro*?**

Abbiamo bisogno innanzitutto del filtro costruito al passaggio precedente, del campione del suono ripulito attuale, del valore del passo dell'aggiornamento e del buffer. Quindi l'intestazione della nostra funzione può essere:

```
AggiornamentoFiltro(w, mu, en, v2temp)
      ...
```

**Una volta stabilito cosa passare alla funzione, concentra l'attenzione sul cuore della procedura. Cosa deve fare (ripetutamente) la procedura? Il risultato dell'operazione dove deve essere conservato, memorizzato?**

Si tratta di implementare semplicemente la formula

$$w[n] = w[n-1] + \mu \cdot e[n] \cdot v_2[n]$$

e immagazzinare il risultato in una variabile temporanea. Dopo di ciò è necessario aggiornare il k-esimo elemento del filtro a esso il valore di questa variabile temporanea.

```
AggiornamentoFiltro(w, mu, en, v2temp)
      ...
            temp = mu * en * v2temp[k]
            w[k] = w[k] + temp
      ...
```

**Questa operazione, questo calcolo, e la successiva assegnazione devono essere ripetute più volte, per ciascun elemento del filtro, della lista w.**

Dobbiamo ripetere queste operazioni per ogni elemento del ciclo.

```
AggiornamentoFiltro(w, mu, en, v2temp)
      ripeti per k = 1 a lunghezza di w
            temp = mu * en * v2temp[k]
            w[k] = w[k] + temp
      ...
```

**Ricorda che è necessario definire l'output dell'algoritmo. Cosa deve restituire la funzione AggiornamentoFiltro? Qual è il risultato che ci aspettiamo?**

Ovviamente dobbiamo restituire il filtro aggiornato (la lista w).

```
AggiornamentoFiltro(w, mu, en, v2temp)
      ripeti per k = 1 a lunghezza di w
            temp = mu * en * v2temp[k]
            w[k] = w[k] + temp
      restituisci w
```

## 3.5. Un ultimo passo: la funzione principale

**Per prima cosa devi precisare, come già fatto più volte, l'input della funzione: tutte le informazioni e i dati che dobbiamo passare perchè si riesca a ottenere il segnale ripulito.**


Beh! Ci serve un po' tutto. In particolare il segnale disturbato d, il rumore campinato dal microfono secondario v2 e, non dimentichiamola, la lunghezza della sottolista M. Le prime due saranno liste mentre l'ultima sarà un valore intero minore della lunghezza delle due precedenti.

```
CancellazioneRumore(d, v2, M)
      ...
```

**Scrivi le istruzioni, sempre in linguaggio naturale ma preciso e non ambiguo, che servono per creare le strutture dati e assegnare loro i valori iniziali.**

Dobbiamo calcolare la lunghezza del campione L. Ci serve la lista e di lunghezza *L* che conterrà il segnale pulito e le liste w e v2temp in cui memorizzare il filtro e il buffer, entrambe di lunghezza M.

```
CancellazioneRumore(d, v2, M)
      L = lunghezza di d
      crea la lista e di lunghezza L con tutti gli elementi nulli
      crea la lista w di lunghezza M con tutti gli elementi nulli
```

```
        crea la lista v2temp di lunghezza M con tutti gli elementi nulli
        ...
```

**Definisci un ciclo che percorra la lista, il vettore d, quello che contiene i campioni del suono da ripulire.**

```
CancellazioneRumore(d, v2, M)
        L = lunghezza di d
        crea la lista e di lunghezza L con tutti gli elementi nulli
        crea la lista w di lunghezza M con tutti gli elementi nulli
        crea la lista v2temp di lunghezza M con tutti gli elementi nulli
        ripeti per i = 1 a L
        ...
```

**Specifica come richiamare la funzione *AggiornamentoBuffer* che abbiamo creato in precedenza e a quale variabile assegnare il risultato dell'aggiornamento.**

La funzione *AggiornamentoBuffer* fa quello che dice il nome. Il risultato è il nuovo buffer.

```
CancellazioneRumore(d, v2, M)
        L = lunghezza di d
        crea la lista e di lunghezza L con tutti gli elementi nulli
        crea la lista w di lunghezza M con tutti gli elementi nulli
        crea la lista v2temp di lunghezza M con tutti gli elementi nulli
        ripeti per i = 1 a L
                v2temp = AggiornamentoBuffer(i, v2, v2temp)
        ...
```

**Indica come applicare la funzione *Convoluzione*, implementata proprio per essere utilizzata in questo step dell'applicazione. Calcola il campione del segnale ripulito utilizzando la formula corrispondente.**

Il risultato della comvoluzione tra il filtro e il buffer è il cuore dell'algortimo e fornisce la stima del rumore originale.

```
CancellazioneRumore(d, v2, M)
        L = lunghezza di d
        crea la lista e di lunghezza L con tutti gli elementi nulli
        crea la lista w di lunghezza M con tutti gli elementi nulli
        crea la lista v2temp di lunghezza M con tutti gli elementi nulli
        ripeti per i = 1 a L
                v2temp = AggiornamentoBuffer(i, v2, v2temp)
                v1stima = Convoluzione(w, v2temp)
        ...
```


**Scrivi una sola riga, sempre in linguaggio naturale ma precisa e non ambigua, per il calcolo del segnale pulito.**

Questa, alla fine, è forse la cosa più facile di tutto il processo.

```
CancellazioneRumore(d, v2, M)
        L = lunghezza di d
        crea la lista e di lunghezza L con tutti gli elementi nulli
        crea la lista w di lunghezza M con tutti gli elementi nulli
```

```
        crea la lista v2temp di lunghezza M con tutti gli elementi nulli
        ripeti per i = 1 a L
              v2temp = AggiornamentoBuffer(i, v2, v2temp)
              v1stima = Convoluzione(w, v2temp)
              e[i] = d[i] - v1stima
        ...
```

**Abbiamo una funzione che aggiorna, a ogni *step*, il filtro. Coma va usata? A chi va assegnato il risultato?**

Dobbiamo usare i valori appena calcolati, compreso il campione del suono pulito, per aggiornare il filtro.

```
CancellazioneRumore(d, v2, M)
      L = lunghezza di d
      crea la lista e di lunghezza L con tutti gli elementi nulli
      crea la lista w di lunghezza M con tutti gli elementi nulli
      crea la lista v2temp di lunghezza M con tutti gli elementi nulli
      ripeti per i = 1 a L
            v2temp = AggiornamentoBuffer(i, v2, v2temp)
            v1stima = Convoluzione(w, v2temp)
            e[i] = d[i] - v1stima
            w = AggiornaFiltro(w, mu, e[i], v2temp)
      ...
```

**Quali dati deve restituire la funzione?**

Adesso ci siamo! Al termine del ciclo la lista e conterrà i campioni sel suono pulito: è proprio questo che la funzione deve restituire.

```
CancellazioneRumore(d, v2, M)
      L = lunghezza di d
      crea la lista e di lunghezza L con tutti gli elementi nulli
      crea la lista w di lunghezza M con tutti gli elementi nulli
      crea la lista v2temp di lunghezza M con tutti gli elementi nulli
      ripeti per i = 1 a L
            v2temp = AggiornamentoBuffer(i, v2, v2temp)
            v1stima = Convoluzione(w, v2temp)
            e[i] = d[i] - v1stima
            w = AggiornaFiltro(w, mu, e[i], v2temp)
      restituisci e
```


Puoi implementare tutto il programma con un generatore di codice (come Blockly) per provare a testare l'algoritmo.

## 3.6. Un test virtuale

**Usando un foglio di calcolo o un software CAS create una tabella simile a quella che trovate qui di seguito le cui colonne contengono alcuni dati sperimentali (virtuali, come preannunciato):**

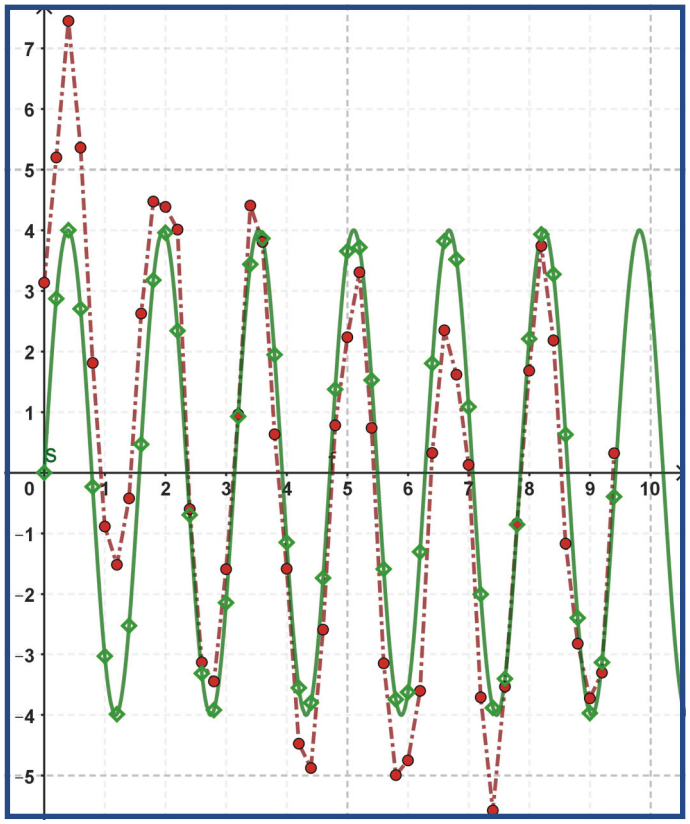
Per completare il laboratorio mettiamo alla prova il programma, che abbiamo realizzato con il generatore di codice Blockly, su una serie di dati virtuali. La tabella che si trova nella pagina successiva contiene questi dati.

| t | s(t) | $v_2(t)$ | $v_2(t)$ random | d(t) | e |
|---|---|---|---|---|---|
| 0.0 | 0.00000 | 3.00000 | 2.51245 | 2.51245 | 3.13338 |
| 0.2 | 2.86942 | 2.55384 | 2.23796 | 5.10738 | 5.19867 |
| 0.4 | 3.99829 | 3.38588 | 3.65475 | 7.65305 | 7.44988 |
| 0.6 | 2.70185 | 2.92388 | 3.00047 | 5.70232 | 5.36046 |
| 0.8 | -0.23350 | 2.39167 | 2.68130 | 2.44780 | 1.80942 |
| 1.0 | -3.02721 | 3.16255 | 3.01643 | -0.01078 | -0.88901 |
| 1.2 | -3.98466 | 2.66763 | 3.15636 | -0.82830 | -1.51926 |
| 1.4 | -2.52507 | 2.05433 | 1.83249 | -0.69257 | -0.42495 |
| 1.6 | 0.46620 | 2.76874 | 2.91949 | 3.38569 | 2.62419 |
| 1.8 | 3.17467 | 2.24815 | 2.53523 | 5.70990 | 4.47224 |
| 2.0 | 3.95743 | 1.56226 | 1.06263 | 5.02006 | 4.38348 |
| 2.2 | 2.33967 | 2.22826 | 2.53287 | 4.87254 | 4.00836 |
| 2.4 | -0.69731 | 1.69244 | 1.50152 | 0.80422 | -0.60247 |
| 2.6 | -3.31131 | 0.94553 | 0.83104 | -2.48026 | -3.12932 |
| 2.8 | -3.91671 | 1.57399 | 1.88346 | -2.03326 | -3.44414 |
| 3.0 | -2.14629 | 1.03596 | 1.43143 | -0.71487 | -1.59342 |
| 3.2 | 0.92604 | 0.24195 | -0.20502 | 0.72101 | 0.95941 |
| 3.4 | 3.43665 | 0.84584 | 0.76187 | 4.19852 | 4.40546 |
| 3.6 | 3.86263 | 0.32039 | -0.17604 | 3.68659 | 3.80481 |
| 3.8 | 1.94559 | -0.50525 | -0.83891 | 1.10668 | 0.63274 |
| 4.0 | -1.15161 | 0.08827 | 0.56867 | -0.58294 | -1.58583 |
| 4.2 | -3.55027 | -0.40894 | -0.47821 | -4.02848 | -4.47098 |
| 4.4 | -3.79538 | -1.25010 | -1.55526 | -5.35064 | -4.87199 |
| 4.6 | -1.73826 | -0.65248 | -0.51050 | -2.24876 | -2.58983 |
| 4.8 | 1.37326 | -1.10584 | -0.88718 | 0.48608 | 0.77985 |
| 5.0 | 3.65178 | -1.94675 | -2.04661 | 1.60517 | 2.23321 |
| 5.2 | 3.71518 | -1.33120 | -1.62611 | 2.08907 | 3.30412 |
| 5.4 | 1.52500 | -1.72614 | -1.51663 | 0.00837 | 0.73761 |
| 5.6 | -1.59022 | -2.55231 | -2.06056 | -3.65079 | -3.14633 |
| 5.8 | -3.74084 | -1.90660 | -2.27824 | -6.01908 | -4.98944 |
| 6.0 | -3.62231 | -2.23044 | -2.17676 | -5.79908 | -4.74732 |
| 6.2 | -1.30654 | -3.02948 | -2.74951 | -4.05605 | -3.60004 |
| 6.4 | 1.80176 | -2.34377 | -1.97977 | -0.17801 | 0.32532 |
| 6.6 | 3.81714 | -2.58651 | -2.24624 | 1.57090 | 2.34759 |
| 6.8 | 3.51709 | -3.34891 | -3.30533 | 0.21176 | 1.61577 |
| 7.0 | 1.08362 | -2.61640 | -2.45818 | -1.37455 | 0.12826 |
| 7.2 | -2.00716 | -2.77134 | -2.52903 | -4.53619 | -3.70622 |
| 7.4 | -3.88042 | -3.49095 | -3.30744 | -7.18786 | -5.57289 |
| 7.6 | -3.39988 | -2.70842 | -2.95010 | -6.34998 | -3.52964 |
| 7.8 | -0.85701 | -2.77252 | -2.40808 | -3.26509 | -0.83981 |
| 8.0 | 2.20571 | -3.44694 | -3.69677 | -1.49106 | 1.68087 |
| 8.2 | 3.93047 | -2.61493 | -2.34318 | 1.58729 | 3.74053 |
| 8.4 | 3.27107 | -2.58905 | -3.00942 | 0.26165 | 2.18217 |
| 8.6 | 0.62747 | -3.21970 | -2.79501 | -2.16753 | -1.17175 |
| 8.8 | -2.39673 | -2.34257 | -1.94189 | -4.33862 | -2.82161 |
| 9.0 | -3.96712 | -2.23139 | -2.42058 | -6.38770 | -3.72349 |
| 9.2 | -3.13110 | -2.82337 | -2.40817 | -5.53927 | -3.29999 |
| 9.4 | -0.39580 | -1.90907 | -1.45507 | -1.85087 | 0.32030 |

Sul sito del progetto http://researchi-naction.it/ è disponibile il materiale a supporto di questo laboratorio: http://researchinaction.it/materials/20-Cancellazione-del-rumore.zip in cui si può trovare il file GeoGebra utilizzato per il test virtuale, nel foglio di calcolo di GeoGebra sono riportati tutti i dati che vedete in tabella.


Il segnale originale è simulato con una sinusoide $s(t) = 4\sin(4t)$ mentre il rumore di disturbo è

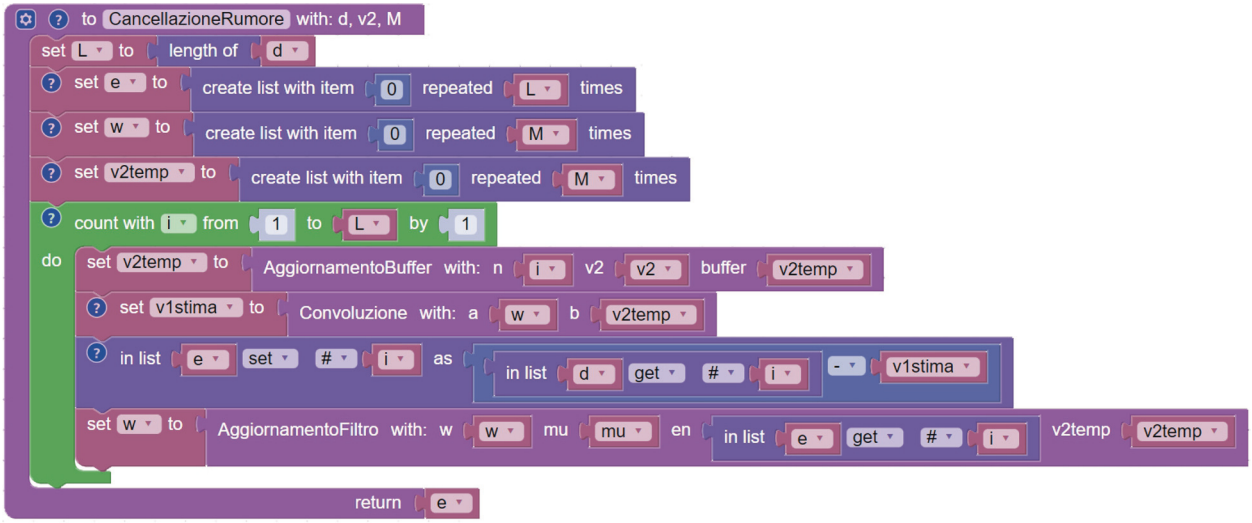
realizzato con la funzione goniometrica, $v_2(t) = 2\sin(21t) + 3\cos(31t)$. Questo disturbo è stato *sporcato* aggiungendo un valore casuale compreso tra *-0.5* e *+0.5*. Il segnale rilevato è ovviamente ottenuto come somma tra quello originale e il disturbo: $d(t) = s(t) + v_2 random(t)$. Una volta lanciata l'applicazione, con input il vettore, la lista *d*, il vettore *$v_2$random* e fissati $M = 15$ e $\mu = 0.001$, abbiamo ottenuto la lista e che abbiamo riportato nell'ultima colonna della tabella.

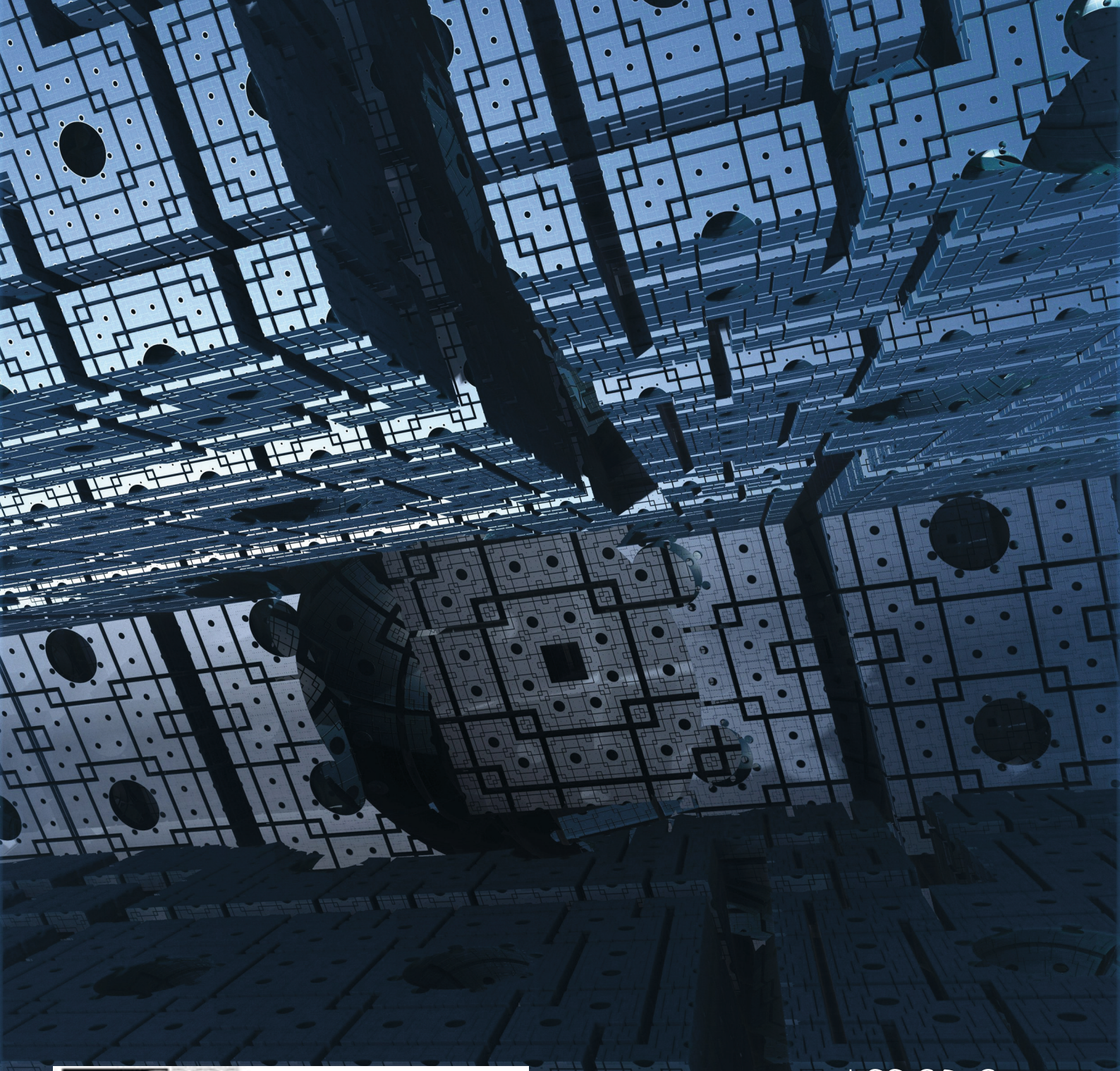



Nel grafico qui accantoabbiamo infine rappresentato la funzione *s(t)*, in colore verde, con i corrispondenti campioni (nello stesso colore) e i campioni del suono ripulito *e* (in colore rosso). Si nota abbastanza bene come, dopo qualche istante di tempo, la spezzata che lega i valori ottenuti dall'applicazione si avvicina a rappresentare il suono originale ricalcandone il tracciato.

Un buon lavoro ... che però non termina qui!

In fondo a questa pagina la funzione principale dell'applicazione, *CancellazioneRumore*, realizzata con Blockly. Si distingue chiaramente il ciclo principale, il blocco in colore verde, che *costruisce* uno a uno i campioni del suono ripulito dal disturbo (e li memorizza nella lista *e*). Il ciclo svolge le fasi dell'algoritmo che abbiamo visto inizialmente: *AggiornamentoBuffer*, *Convoluzione*, *AggiornamentoFiltro* **(cfr. 2.1 L'algoritmo in sintesi a pagina 7)**. I primi quattro blocchi della funzione CancellazioneRumore si occupano dell'inizializzazione creando le liste e impostando tutti gli elementi a zero.



LICEO SCIENTIFICO GRASSI LATINA

CNR IAC
Istituto per le Applicazioni del Calcolo

CNR IFN
Istituto di Fotonica e Nanotecnologie

Marine Technology Research Institute
INSEAN

DIET
Dipartimento di Ingegneria dell'Informazione, Elettronica e Telecomunicazioni

## LSS G.B. Grassi
Liceo scientifico statale G.B. Grassi di Latina
www.liceograssilatina.org

## CNR - IAC
Istituto per le Applicazioni del Calcolo Mauro Picone
www.iac.cnr.org

## CNR - IFN Roma
Istituto di Fotonica e Nanotecnologie
www.roma.ifn.cnr.org

## CNR - INSEAN
Istituto Nazionale Studi Esperienze e Architettura Navale
www.insean.cnr.org

## DIET
Dip. di Informatica, Elettronica e Telecominicazioni
web.uniroma1.it/dip_diet/