UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Sviluppo di un algoritmo per la determinazione automatica dei modi di vibrare mediante Continuous Scanning Laser Doppler Vibrometry

Development of an algorithm for the automatic determination of Operational Deflection Shapes by Continuous Scanning Laser Doppler Vibrometry

Relatore:

Chiar.mo Prof. Paolo Castellini

Tesi di Laurea di:

Elisa Corradini

A.A. 2019/2020

INDICE

1. INTRODUZIONE

Il vibrometro laser Doppler (LDV) è uno strumento che permette di effettuare una misurazione senza il bisogno di trasduttori in contatto fisico con l'oggetto di studio. Quest'ultimo aspetto è di fondamentale importanza nell'ambito della misurazione delle vibrazioni, in quanto rappresenta la possibilità di ridurre le interferenze legate allo strumento e all'installazione dei trasduttori, disturbi che corrodono la precisione dei risultati.

Le forme di deflessione operativa ("Operating Deflection Shapes") di una struttura vibrante possono, quindi, essere estratte mediante LDV, facendo uso della tecnica convenzionale di scansione per punti ("Step Scanning") o con scansione continua ("Continuous Scanning").

La misurazione può essere simulata, fornendo semplicemente i dati in ingresso ed ottenendo i modi di vibrare senza passare attraverso lo strumento e l'apparato sperimentale, cosa molto utile in condizioni particolari del corpo sottoposto all'indagine.

Disponendo di un programma di estrazione dei modi di vibrare mediante CSLDV ("Continuous Scanning Laser Doppler Vibrometry"), se ne vogliono migliorare le prestazioni, perfezionare l'efficacia delle estrazioni, migliorare la precisione dei risultati e rendere più agevoli i calcoli.

In questo studio sono stati perseguiti due obiettivi principali, entrambi con lo scopo del perfezionamento e della maggiore efficienza della misurazione simulata delle forme di deflessione operativa, effettuata con un sistema LDV.

Uno degli obiettivi operativi prefissati è raggiunto attraverso la scrittura di due programmi di definizione del segnale reale nel tempo, relativi uno alla scansione nella modalità "Step Scanning" e l'altro al "Continuous Scanning". Il segnale ottenuto dalla CSLDV è un input necessario al programma di estrazione delle forme di deflessione operativa, perché contenente le informazioni iniziali necessarie e completa il quadro della misurazione simulata dei modi di vibrare.

Il miglioramento dell'operatività del programma ed il perfezionamento dei risultati sfociano nella ricerca di algoritmi, che permettano di ridurre le approssimazioni del calcolo dei risultati e di accelerare lo svolgimento dei passaggi, due aspetti fondamentali nell'ambito della simulazione. È così delineato il secondo obiettivo di questo studio e il tentativo di perseguirlo ha preso forma in una ricerca svoltasi secondo il modello sperimentale convenzionale: dallo studio teorico dei possibili approcci è scaturita la formulazione di ipotesi, sino al raggiungimento delle conclusioni passando attraverso la sperimentazione pratica, cioè l'avviamento del programma una volta modificato opportunamente.

In conclusione, quello che si vuole ottenere è quanto citato nel titolo: lo sviluppo di un algoritmo per la determinazione automatica delle forme di deflessione operativa, che permetta, in tempi ragionevoli e con una precisione consistente, di raggiungere i risultati, disponendo in partenza solamente di pochi input, derivanti da una misurazione fisica o da una sua simulazione.

I programmi sono stati plasmati grazie al supporto dello script nel software MATLAB, di cui è stato fatto uso, sfruttando anche una delle sue funzionalità inserite nelle applicazioni (APP).

Nel lavoro svolto sono stati integrati diversi ambiti di conoscenza e competenza, quali vibrazioni e loro misura e conoscenze di algebra lineare e geometria, ma anche funzionalità dello script di MATLAB; perciò questo progetto permette di lambire disparati ambiti scientifici: dalle misurazioni alle simulazioni, dalle vibrazioni e al linguaggio di programmazione.

2. SVOGLIMENTO

2.1 Introduzione SSLDV e CSLDV

"CSLDV" o "Vibrometria Doppler laser a scansione continua" è una tecnica di misurazione delle vibrazioni basata sull'uso di un trasduttore, un vibrometro laser Doppler (LDV) di risposta senza contatto, che può essere impostato in scansione continua sulla superficie di una struttura eccitata sinusoidalmente.

Questa tecnica è un tipo di misurazione del "campo spaziale", che mira a superare la limitazione essenziale dei test di vibrazione convenzionali: per esempio, il numero di punti misurati è limitato dall'insieme di trasduttori che possono essere fisicamente attaccati alla struttura, di conseguenza la risoluzione spaziale dei modi di vibrare è limitata.
Con l'uso di un trasduttore ottico, a cui non è necessario un contatto materiale, le informazioni sulla vibrazione possono essere derivate in ciascun punto del modello acquisito senza alcuna restrizione fisica.

Le misurazioni con LDV convenzionali sono solitamente effettuate a partire dalla definizione di una serie di punti, individuabili sull'oggetto posto in vibrazione; il laser esegue un'acquisizione di dati sequenziale, spostandosi da un punto al successivo. Questa tecnica è detta "Step Scanning Laser Doppler Vibrometry (SSLDV)".

La "Continuous Scanning Laser Doppler Vibrometry (CSLDV)" si presenta, quindi, come una valida alternativa alla "SSLDV": se pur teoricamente più complessa, risulta, allo stesso tempo, di più rapida attuazione.

In CSLDV il punto laser viene spostato in modo continuo e sinusoidale sulla superficie target durante l'acquisizione dei dati di vibrazione: ciò consente di raccogliere contemporaneamente dati temporali e spaziali, utilizzando una risoluzione spaziale, che dipende dalla frequenza di campionamento utilizzata per raccogliere i dati.

Quando è effettuata una scansione continua con LDV lungo una linea arbitraria, l'uscita è un'onda sinusoidale, contenente i dati di vibrazione CSLDV e modulata in ampiezza secondo la struttura delle forme di deflessione operativa (combinazione di forme modali).
Esaminando lo spettro in uscita, si possono individuare bande laterali ("sidebands") rispetto alla frequenza di risonanza degli ODS.

Ampiezza e fase dello spettro alla frequenza di eccitazione ed alle bande laterali sono sfruttate per recuperare ampiezza e fase delle forme di deflessione operativa.
L'analisi dei dati in uscita si concentra, quindi, sull'output nel dominio della frequenza, cioè sullo spettro correlato con picchi alla frequenza di eccitazione e alle bande laterali. I picchi relativi alle *sidebands* devono essere distinguibili, centrati sulla frequenza di eccitazione e distanziati dalla frequenza di scansione del raggio laser, scelta in modo da evitare sovrapposizione delle bande laterali.

Il numero delle bande laterali è stimabile tramite: "`m=(length(f_scan)-1)/2`", dove "m" è il numero di sidebands ed "f_scan" è la frequenza di scansione.

Se l'estensione della linea scansionata si riduce fino a quando non diventa più corta della lunghezza d'onda del modello di vibrazione, il numero di bande laterali diminuisce a una sola coppia, da cui vibrazione angolare e traslazione possono essere recuperate.

Le forme di deflessione operativa sono ricavate mediante funzioni polinomiali derivanti dall'output nel dominio della frequenza.
Potrebbe essere difficile a volte estrarre gli ODS con il metodo "Peak-Picking" (o metodo "picco picco"), cioè con metodo polinomiale tradizionale: i modi potrebbero essere vicini o i dati molto rumorosi, motivo

per cui i picchi delle *sidebands* potrebbero sovrapporsi o essere mascherati dal rumore, rendendo difficile una ricostruzione efficiente delle forme di deflessione operativa.

È necessario, quindi, delineare un approccio per elaborare ciecamente i dati di vibrazione CSLDV, per l'estrazione dello spettro caratteristico CSLDV, chiamato “spettro della banda laterale”.

I metodi si basano sull'ipotesi che il modello del cluster di frequenze (frequenze centrali e bande laterali) che identifica un determinato ODS non cambi con le frequenze di scansione. L’utilizzo di “cluster” fa riferimento al fatto che, quando viene eccitata più di una frequenza caratteristica della struttura, ci sarà un gruppo di bande laterali per ognuna di tali frequenze.

Ciò significa che il contributo di ciascuna banda laterale al cluster ed anche la frequenza centrale sono invarianti rispetto alla frequenza di scansione e dipendono solo dall'evoluzione spaziale dell'ODS, che causa la modulazione dell'ampiezza sul segnale CSLDV.

Il riconoscimento delle frequenze di risonanza e l’individuazione delle forme di deflessione operativa nel trattamento dei dati CSLDV vengono eseguiti a partire dall'osservazione dello spettro di uscita CSLDV.

Mediante i programmi di estrazione “Estrai_SS” ed “Estrai_CS”, di seguito presentati, è possibile costruire i segnali SSLDV e CSLDV in uscita nel dominio del tempo, punto di partenza per le successive analisi appena introdotte.

2.2 Estrai_SS

Il programma di estrazione “Estrai_SS” permette di definire una funzione che sintetizza il segnale nel tempo, risultato della misurazione SSLDV, a partire dai dati iniziali relativi alla geometria della trave (oggetto posto in vibrazione, approssimato come monodimensionale), ai tempi di scansione e agli output registrati, derivanti da una simulazione preventiva.
I parametri sono stati acquisiti da una simulazione FEM di una trave, vincolata mediante incastro ad un estremo e sollecitata da uno shaker.

2.2.1 Testo del programma *Estrai_SS*

```
close all
clear
clc

%% Numero punti di misurazione
npoints=20;

%% Estrazione dati FEM trave
file_parameter='parameter_Ns8192_rand_undmap.mat';
file_pos_xyt='pos_xyt_Ns8192_rand_undmap.mat';

load(file_parameter)
width=parameter.width;
dt=parameter.dt;
tlist=parameter.tlist;
% x=parameter.x;
% x_tip_max=parameter.x_tip_max;
% load('sim_modal_disp.mat')

%load('pos_xyt_Ns8192_rand_undmap.mat','pos_xyt')
load(file_pos_xyt)
x=pos_xyt(:,:,1);
y=pos_xyt(:,:,2);

for kk=1:size(x,2)
    x(:,kk)=x(:,kk)+parameter.x';
end
%% Max spostamento in x al tip
[x_tip_max,in]=min(max(x));


%% Dati laser

%Rumore
a_noise=.1;


%Spazio
% x_tip_max=0.1;
% dx=(width-x_tip_max)/(npoints-1);
dx=x_tip_max/(npoints-1);
x_laser=(0:dx:x_tip_max);
figure,plot(x_laser);


%% Estrazione
for ii=1:npoints
```

```
    ind=find(x(:,ii)<=x_laser(ii));
    ind_x=max(ind);
    y_laser=y(ind_x,:);
    pippo(ii,:)=y_laser;  % da togliere dopo debug
    % Interpolazione di y_scan
    %x(ind_x)      x_laser(kk)     x(ind_x+1)
    %y(ind_x,:)       ?            y(ind_x+1,:)
%     y_interp=y(ind_x,:)+(x_laser(kk)-x(ind_x)/x(ind_x+1)-
x(ind_x))*y(ind_x+1,:)-y(ind_x,:);
    signal(ii,:)=y(ind_x,:)+(x_laser(ii)-x(ind_x,:))./(x(ind_x+1,:)-
x(ind_x,:)).*(y(ind_x+1,:)-y(ind_x,:));
end

noise=a_noise*(randn(size(signal,1),size(signal,2))-0.5);
signal_noise=signal+noise;

iop=1:size(pippo,2);
figure, plot(iop,signal(16,:),'b',iop,signal_noise(16,:),'y');
xlabel('time');
ylabel('signal');
figure, plot(iop,pippo(4,:),iop,signal(4,:),'-*')  % da togliere
figure, plot(x_laser,signal(:,3),'-*')

%%
fs=1/dt;
[Y,f]=fftic(y,fs);
figure
for jj=1:size(Y,1)
    plot(Y(jj,:));
end

%%
v=diff(y,1,2);  % velocità  nel tempo
figure,plot(v);
figure, plot(tlist(1:end-1),v(1:20,:))
xlabel('time');
ylabel('v')
```

2.2.2 Numero di punti

Nel programma, innanzitutto, si definisce il numero dei punti di misurazione, come si procederebbe in una misurazione reale e non simulata, in cui il primo step è la definizione della "nuvola" di punti interessati dal laser.
Fin dal primo passaggio, quindi, il programma di estrazione relativo alla modalità "Step Scanning" si differenzia dall'alternativa della scansione continua.

2.2.3 Estrazione dati

Dopo aver caricato i file dei parametri, si estraggono i singoli dati: lunghezza, incremento temporale, un array contenente i tempi di acquisizione, geometria della trave (matrix "x"), valori del segnale nel tempo in corrispondenza dei punti che definiscono la geometria (matrix "y").
Si individua la posizione dell'estremo della trave nel massimo valore assunto da x:

```
[x_tip_max,in]=min(max(x));
```

2.2.4 Laser

Dopo aver definito l'incremento dx, spostamento tra una posizione e la successiva, come rapporto tra posizione dell'estremo e numero di punti meno uno (il primo è lo zero), si procede esplicitando il movimento del laser: lo spostamento del laser è indicato come array ("x_laser") delle successive posizioni occupate da esso, intervallate da dx.

2.2.5 Costruzione del segnale

Per costruire il segnale bisogna partire da un'osservazione: i valori delle posizioni dei punti che definiscono la geometria potrebbero non coincidere con la "nuvola" di punti di misurazione occupati dal laser.
Mediante un *ciclo for* è possibile estrarre di volta in volta gli indici dei valori di ognuna delle colonne di "x" minori uguali dell'i-esimo elemento di "x_laser" ed individuarne il massimo: il segnale "y_laser" è il vettore costruito estraendo da ogni riga di "y" il valore di relativo all'indice massimo.
Il segnale così costruito è rigidamente legato ai punti ottenuti con la simulazione, tuttavia i punti di misurazione del laser non coincidono con essi, ma sono definiti dall'array "x_laser".
Si fa ricorso ad un'interpolazione, i cui termini sono:

```
%x(ind_x,:)       x_laser(ii,:)      x(ind_x+1,:)

%y(ind_x,:)       signal(ii,:)             y(ind_x+1,:)
```

Il segnale ottenuto è, quindi, definito come:

```
signal(ii,:)=y(ind_x,:)+(x_laser(ii)-x(ind_x,:))./(x(ind_x+1,:)-
x(ind_x,:)).*(y(ind_x+1,:)-y(ind_x,:));
```

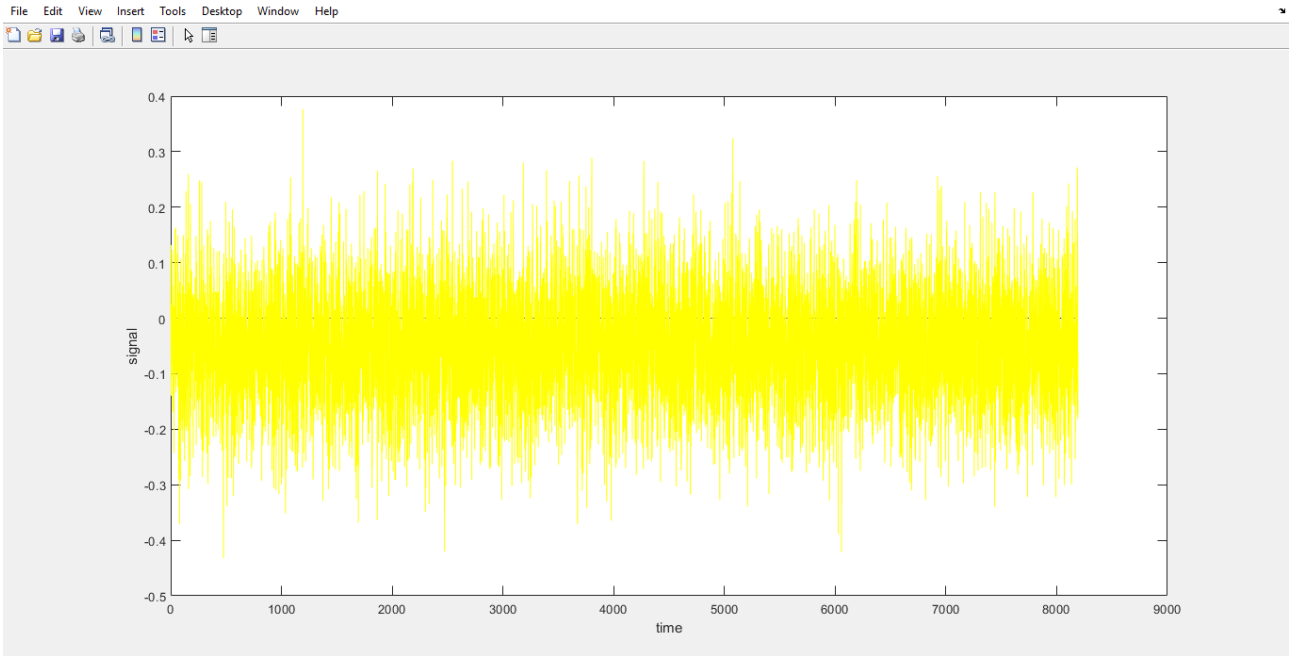

Figura 2.1. Confronto tra il segnale (in nero) e il segnale rumoroso (in giallo) in funzione del tempo

2.2.6 Rumore

L'output dell'interpolazione è il segnale teorico, ideale. Nella realtà non ci si può esimere dall'inevitabile contributo del rumore, fattore di disturbo: segnale indesiderato randomico che si sovrappone al segnale utile nella trasmissione.
La sintesi di un rumore randomico è stata possibile grazie alla funzione presente in MATLAB: "randn", "Normally distributed random numbers", che permette di costruire matrici o vettori della dimensione desiderata, i cui elementi sono numeri casuali generati dal programma.

Il segnale rumoroso è quindi:

```
noise=a_noise*(randn(size(signal,1),size(signal,2))-0.5);
```

dove “a_noise” è il valore dell’ampiezza.

Per ottenere il segnale reale in uscita “signal_noise” è stata sommata al segnale ideale interpolato “signal” la componente rumorosa “noise”, trasposizione di quanto accade nella realtà, cioè una sovrapposizione del rumore al segnale trasmesso.

2.2.7 Spettro in frequenza

Disponendo del segnale, nel dominio del tempo, mediante l’applicazione della trasformata di Fourier è possibile ottenere lo spettro in uscita, nel dominio delle frequenze.
In MATLAB la trasformata di Fourier è stata sintetizzata nell’algoritmo “Fast Fourier transform”, la cui corrispondente funzione è “fft”.
Nel programma di estrazione è stata usata un’alternativa più agile, la funzione “fftic”.
Tale algoritmo richiede un maggior numero di input rispetto all’alternativa “fft”: non è sufficiente fornire il segnale nel tempo, bensì è necessario assegnare anche la frequenza di campionamento.

2.2.8 Velocità

A partire dal segnale è possibile calcolare la risposta in termini di velocità mediante la funzione “diff”, “Differences and approximate derivatives”, presente in MATLAB e che fornisce come output la derivata dell’input.

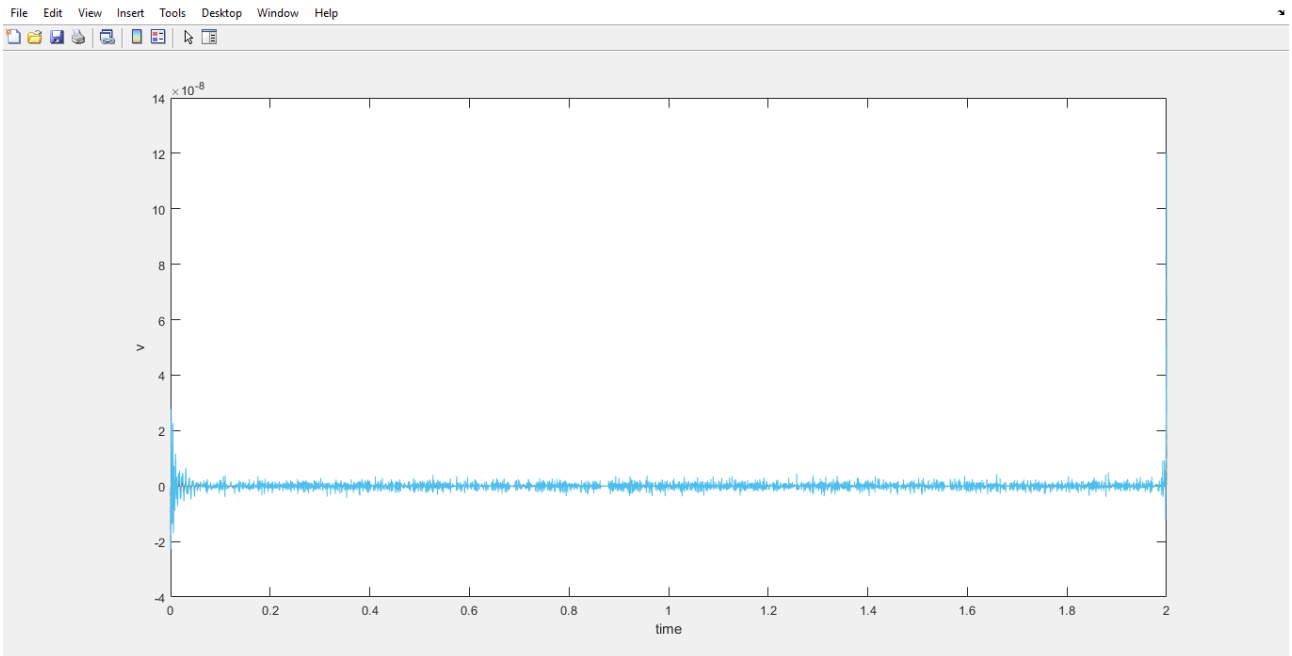

Figura 2.2. Velocità in funzione del tempo

2.3 Estrai_CS

2.3.1 Testo del programma *Estrai_CS*

```
close all
clear
clc
%cd('D:\TesiETirocini\Elisa Corradini\SWcompleto')
%path_data='D:\TesiETirocini\Elisa Corradini\dati';
%% Estrazione dati FEM trave
file_parameter='parameterNs8192';
file_data='pos_xytNs8192.mat';

%load(strcat(path_data,'\',file_parameter))
load(file_parameter)
load(file_data)
width=parameter.width;
dt=parameter.dt;
fs=1/dt;
tlist=parameter.tlist;
%x=parameter.x;
%load(strcat(path_data,'\',file_data))
x=pos_xyt(:,:,1);
y=pos_xyt(:,:,2);
%figure, plot(tlist,y(end,:))
%[Y,f]=fftic(y',fs);
%figure,plot(f,abs(Y(:,end)))
%% Dati laser
% Frequenza laser
n_scan=12;  %numero scansioni (ne metto molte e poi in lettura ne posso
selezionare solo quelle che mi servono
f_base=.5;
f_scan=f_base*(1:n_scan);
a_scan=.99*width/2;  % ampiezza scansione
%Rumore
noise_type=1;
a_speckle_noise=0; %0.001;
a_noise=0; %0.0005;

for ii=1:n_scan
    x_scan=a_scan+(a_scan)*sin(2*pi*f_scan(ii)*tlist);
    % figure, plot(tlist,x_scan)
        for jj=1:length(tlist)
        %ind=find(x_scan(jj)<=x,1,'first');
        ind=find(tlist<=tlist(jj));        %posizioni dei valori di tlist <= di
t_scan(ii)
        ind_t=max(ind);                     %posizione del massimo dei valori
minori (il pi? prossimo)
        indd=find(x(:,jj)<=x_scan(jj));
        ind_x=max(indd);
        y_scan(jj)=y(ind_x,ind_t);
        signal(jj)=y(ind_x,ind_t)+(x_scan(jj)-x(ind_x,jj))/(x(ind_x+1,jj)-
x(ind_x,jj))*(y(ind_x+1,ind_t)-y(ind_x,ind_t));

        %y_scan(jj)=y(ind,jj);

end
    noise=a_noise*(randn(1,size(y_scan,2))-0.5);
    v_signal=diff(y_scan,1,2);
    v_signal=[v_signal v_signal(end)];
    v_scan=diff(x_scan,1,2);
```

```
    switch noise_type
        case 0
            speckle_noise=speckle_armonic(a_speckle_noise,x_scan);
        case 1
            speckle_noise=speckle_abs(a_speckle_noise,x_scan);
    end
    signal_Noised(ii,:)=v_signal+noise+speckle_noise;
end
figure, plot(tlist,signal_Noised)
xlabel('time')
ylabel('signal noised')
[Signal_Noised,f]=fftic(signal_Noised',fs);
figure, plot(f,abs(Signal_Noised))
xlabel('frequency')
ylabel('Signal Noised')
figure, plot(tlist,y_scan)
[Y_scan,f]=fftic(y_scan',fs);
figure, plot(f,abs(Y_scan))

%% salvataggio dati
data.fs=fs;
data.fscan=f_scan;
data.Data=signal_Noised;

save(strcat(path_data,'\CSLVdataNs4097.mat'),'data')
```

La definizione di un segnale in uscita, ottenuto da una misurazione di tipo “Continuous Scanning”, può essere effettuata mediante il seguente programma di estrazione. È necessario, inoltre, ricavare, salvare e quindi conservare importanti output, che dovranno essere forniti nel programma di estrazione delle forme di deflessione operativa: il segnale in uscita e la frequenza di scansione.

A differenza del precedente programma di estrazione, facente riferimento allo “Step Scanning” e in cui è stato, innanzitutto, fondamentale indicare numero e posizione dei punti di misurazione, in questo programma il primo passaggio è rivolto direttamente al caricamento dati: infatti, come già detto in precedenza, la misurazione di tipo “Continuous Scanning Laser Doppler Vibrometry” non si effettua raccogliendo dati punto per punto, ma, sacrificando la semplicità di analisi degli output ottenuti, si procede in modo più spedito nella misurazione, muovendo il laser in modo continuo. Questo comporta, ovviamente, la presenza nella fase di analisi ed estrazione dei dati di nuove variabili legate al moto del laser, in aggiunta a quelle inevitabili derivanti dalla vibrazione del corpo, oggetto di esaminazione, in questo caso una trave.

Il laser segue una traiettoria sinusoidale: il suo spostamento è una funzione continua; in questo caso è possibile ricavare la velocità del laser, oltre che del segnale in uscita.

Da sottolineare è l’inevitabilità dei rumori che andranno a peggiorare la qualità della misurazione: in *Estrai_SS* è già stato affrontato l’argomento, definendo un segnale rumoroso randomico che è stato aggiunto a quello in uscita. Nel “Continuous Scanning” si palesa un altro problema: oltre al rumore già introdotto, relativo alla misurazione, se ne insinua anche un secondo, dovuto al moto del laser.

2.3.2 Estrazione dati

Come in “Estrai_SS”, devono essere, innanzitutto, introdotti gli input.
Dopo aver caricato i file dei parametri, si estraggono i singoli dati: lunghezza, incremento temporale, un array contenente i tempi di acquisizione, geometria della trave (matrix “x”), valori del segnale nel tempo in corrispondenza dei punti che definiscono la geometria (matrix “y”).

Nel precedente programma di estrazione si è proseguito con l'individuazione della posizione dell'estremo della trave nel massimo valore assunto da x, cosa non necessaria in questo caso: il laser procede in modo continuo.

2.3.3 Laser

2.3.3.1 Dati frequenza

Si definiscono i dati relativi al laser: numero di campioni e incremento della frequenza.
Il vettore frequenza è ottenuto come array dei valori ottenuti attraverso successive moltiplicazioni dell'incremento e del numero di campioni considerato di volta in volta.
Si indica, inoltre, l'ampiezza del laser, definita facendo riferimento alla lunghezza della trave.

2.3.3.2 Dati tempo

Si suppone che il vettore tempo sia uguale all'input "tlist".

2.3.3.3 Rumore

Si indicano le ampiezze dei due segnali rumorosi, presentati in precedenza.

Una delle ampiezze è indicata, non a caso, come "a_speckle": riguarda, infatti, un rumore tipico del laser, definito come "speckle noise", o, in alternativa, come "rumore a chiazze", o "a macchie", o "a granelli".

Sebbene questo fenomeno sia oggetto di studio sin dai tempi di Newton, è diventato rilevante dall'invenzione del laser. I *motivi a chiazze* si verificano in genere in riflessi diffusi di luce monocromatica, come la luce laser.

Tali riflessi possono verificarsi su materiali con superfici ruvide o in mezzi con un gran numero di particelle disperse nello spazio.

Questo tipo di interferenza è resa inevitabile dalla condizione di ruvidità ottica superficiale, che si realizza quando la micro-dimensione della rugosità della superficie è uguale o maggiore rispetto alla lunghezza d'onda della sorgente illuminante, secondo il "criterio di Rayleigh", che afferma che una superficie tende ad essere effettivamente regolare solo se il rapporto tra l'altezza delle sue irregolarità e la lunghezza d'onda del raggio incidente tende a zero. Solo le superfici ottiche lucide rappresentano un'eccezione.

Lo *speckle noise* è granulare poiché è prodotto da interferenze costruttive e distruttive della luce diffusa dalla superficie, o meglio, delle onde sparse delle fonti, in modo da produrre un punto luminoso o scuro.

Il segnale è deteriorato dallo *speckle noise*: quello che appare è un abbandono di segnale in corrispondenza di una macchiolina scura.

Lo *speckle noise* è di fondamentale importanza nella comunità della meccanica sperimentale, poiché è utile nella misurazione dei campi di spostamento tramite la correlazione di immagini digitali.

2.3.3.4 Spazio laser

Si definisce lo spostamento del laser, che, come già indicato, compie un moto descrivibile con una funzione sinusoidale di ampiezza "a_scan", elemento introdotto nel passaggio precedente:

```
x_scan=a_scan+(a_scan)*sin(2*pi*f_scan(jj)*t_scan);
```

Il termine "f_scan", la frequenza di scansione, dipende dal pedice "jj", relativo ad un *ciclo for* che viene iterato per ogni elemento di "n_scan", numero di campioni. In questo modo si valuta lo spostamento per ogni campione, variando di volta in volta la frequenza di scansione.

2.3.4 Estrazione trave

Mediante *ciclo for* si estraggono gli indici dei valori di *tlist* minori o uguali rispetto a quelli nel vettore tempo del laser, "t_scan", e se ne prende il massimo. Si fa lo stesso con i valori delle posizioni sulla trave, valutate rispetto allo spostamento "x_scan". Si definisce, quindi, il segnale ottenuto, estraendo dalla matrice *y* di output della simulazione i valori relativi alle posizioni indicate dagli indici di spazio e tempo.

Infine, si può ottenere un segnale meno vincolato mediante interpolazione, come già spiegato nel programma *Estrai_SS*. Infatti, l'operazione presenta la stessa struttura, ma richiede un'accortezza in più: considerare gli elementi delle matrici *x* ed *y* non solo relativamente agli indici di posizione ma anche rispetto a quelli di tempo.

```
signal(ii)=y(ind_x,ind_t)+(x_scan(ii)-x(ind_x,ii))/(x(ind_x+1,ii)-
x(ind_x,ii))*(y(ind_x+1,ind_t)-y(ind_x,ind_t));
```

2.3.5 Spettro Y

La velocità è stata ricavata mediante la funzione *diff*, precedentemente introdotta: si ottengono la velocità del segnale, derivando il segnale, e quella del laser, derivando invece lo spostamento dello stesso.

Alla prima, successivamente, si sovrappongono il rumore randomico e lo *speckle noise* attraverso una semplice operazione di addizione: questo permette di passare da una velocità ideale ad una reale, in cui non è possibile isolare la misurazione da rumori.

Il secondo rumore aggiunto è stato ricavato in due diversi modi. Sono state scritte due funzioni differenti ed integrate nel programma facendo uso di uno "switch", "Execute one of several groups of statements", elemento in MATLAB, che permette di eseguire nel *running* dell'algoritmo una fra le diverse alternative, inserite contemporaneamente nel testo, senza che ci siano interferenze o errori.
Lo schema della funzione *switch* è:
switch *switch_expression*
  case *case_expression*
    statement
  case *case_expression*
    statements
  ...
  otherwise
   statements
end

Innanzitutto, è necessario indicare il nome della variabile affiancata da un numero, che corrisponde al "caso" che si vuole eseguire; si procede nella scrittura seguendo il modello dello schema presentato.

Le due funzioni di calcolo dello *speckle noise*, alternative fra di esse, sono: *speckle_armonic* e *speckle_abs*.

Di seguito sono stati inseriti i due programmi delle due funzioni: entrambi richiedono due input, cioè l'ampiezza del segnale rumoroso, "n_speckle", e lo spostamento del laser, "x_scan", e forniscono in uscita il rumore.

2.3.5.1 speckle_armonic

2.3.5.1.1 Testo del programma

```
function speckle_noise=speckle_armonic(n_speckle,x_scan)
t=1:size(x_scan,2);
```

```
% x_scan=sin(2*pi*t/period);
% figure, plot(t,x_scan)

% da qui in poi
% n_speckle=0.01;
armonic=[.83 .35 .12 .03 .04 .1 .05];
%armonic_abs=[0.02665 0.005319 .002272 .001256 .0007943 .0005455 .0003961]
armonic=armonic/armonic(1);
%armonic_abs=armonic_abs/armonic_abs(1);
% figure, plot(1:7, armonic,1:7, armonic_abs)

v_scan=diff(x_scan,1,2);
% figure, plot(v_scan)
% figure, plot(abs(v_scan))
%
% [fft_v_scan freq]=fftic(abs(v_scan'),1/t(2));
% figure, plot(freq, abs(fft_v_scan))

t=t(1,1:end-1);   %??????????????????????????

% Set up fittype and options.
ft = fittype( 'fourier1' );
opts = fitoptions( 'Method', 'NonlinearLeastSquares' );
opts.Display = 'Off';
opts.Robust = 'LAR';
opts.MaxIter = 1200;
%opts.StartPoint = [0 m/size(v_scan,2)*2 97 1];
opts.StartPoint = [0 0 0 0.0628475649630366];
[fitresult,gof] = fit( t', v_scan', ft, opts );
% figure, h = plot( fitresult, t, v_scan ), hold on

a0=fitresult.a0;
a1=fitresult.a1;
b1=fitresult.b1;
w=fitresult.w;

for ii=1:length(armonic)
    val=a1*cos(ii*t*w) + b1*sin(ii*t*w);
    speckle_n(ii,:)=armonic(ii)*val;      %sin delle armoniche;
end

speckle_noise=sum(speckle_n,1);
speckle_noise=n_speckle*speckle_noise/max(speckle_noise);
speckle_noise=[speckle_noise speckle_noise(end)];

% figure, plot(speckle_noise)
```

2.3.5.1.2 Spiegazione programma

Forniti gli input, ampiezza del rumore e spostamento del laser, si estraggono le armoniche, che sono introdotte come dato da evidenze sperimentali, quindi dalla simulazione svolta.

Nello studio dei fenomeni oscillatori, le frequenze armoniche sono le frequenze il cui valore è multiplo intero della frequenza base (frequenza fondamentale) di un'onda.

Le armoniche vengono successivamente normalizzate.

Si calcola la velocità come derivata dello spostamento del laser, mediante la funzione *diff*.

A partire dalle armoniche, si effettua un “Curve Fitting”.

Il *Curve Fitting* è il processo di costruzione di una curva o di una funzione matematica, che abbia la migliore corrispondenza ad una serie di punti assegnati, possibilmente soggetti a limitazioni. Questa operazione può implicare sia l'interpolazione, dove è richiesta un'esatta corrispondenza con i punti dati, o lo spianamento (*smoothing*), dove viene costruita una funzione piana che combaci approssimativamente con i dati. Le curve approssimanti possono intervenire nella visualizzazione dei dati, nella rappresentazione dei valori di una funzione dove non sono disponibili i dati, e nel riassumere le relazioni tra due o più variabili.

Tra le applicazioni presenti in MATLAB (icona "APPS") si può trovare quella di *Curve Fitting*, denominata allo stesso modo. Dalla finestra "Curve Fitting Tool" si possono gestire le proprietà della curva analitica generata dal programma e verificarne la qualità della sovrapposizione rispetto a quella data.
Forniti i dati X, Y, Z, si sceglie il tipo di Fitting: *Interpolant, Costum Equations, Lowess* o *Polynomial*, gestendo i vari input richiesti, una volta scelta l'opzione più adeguata.

Dall'analisi di *Curve Fitting* si ottiene la curva:

```
val=a1*cos(ii*t*w) + b1*sin(ii*t*w)
```

necessaria in un *ciclo for*, in cui è moltiplicata di volta in volta per le armoniche.
Sommando tutti gli elementi ottenuti, si dispone del rumore *speckle*, in seguito completato attraverso la moltiplicazione per l'ampiezza (dato in ingresso) e la normalizzazione, dividendolo per il massimo tra i valori di *speckle noise* dal precedente passaggio.

Si ripete il valore finale a coda del segnale rumoroso, in modo tale che questo abbia una dimensione maggiore, necessaria nella successiva operazione di somma:

```
signal_Noised(ii,:)=v_signal+noise+speckle_noise;
```

presente nella parte finale dello *switch* nel programma *Estrai_CS*.

Lo *speckle noise* ottenuto con questo programma è rappresentato nella seguente figura.

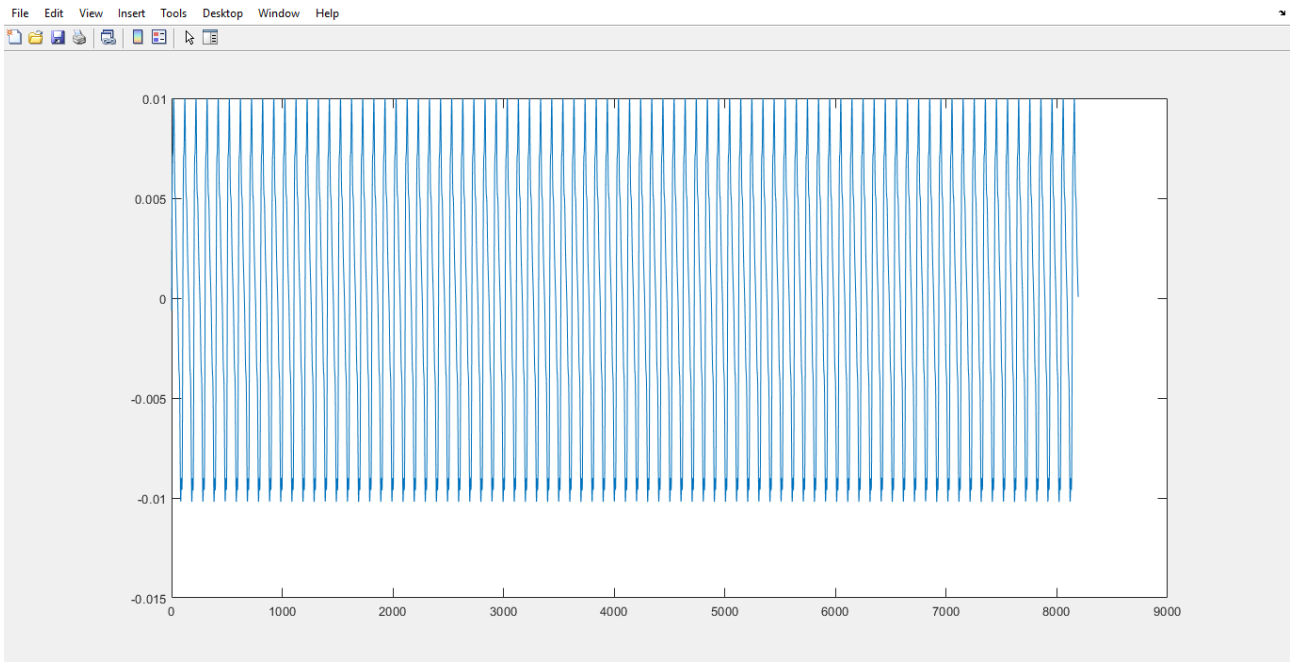

Figura 2.3. Speckle noise mediante speckle_armonic

2.3.5.2 speckle_abs

2.3.5.2.1 Testo del programma

```
function speckle_noise=speckle_abs(n_speckle,x_scan)

t=1:size(x_scan,2);

v_scan=diff(x_scan,1,2);

speckle_noise=n_speckle*abs(v_scan)/max(abs(v_scan));
speckle_noise=[speckle_noise speckle_noise(end)];

figure, plot(speckle_noise)
```

2.3.5.2.2 Spiegazione programma

Forniti i dati in ingresso e calcolata la derivata dello spostamento del laser, è possibile costruire il segnale rumoroso di tipo *speckle* a partire dalla velocità del laser. Il modulo della velocità è normalizzato, perché diviso per il massimo del modulo di velocità: il tipo di segnale ottenuto è svincolato dall'ampiezza della velocità, ma legato comunque alla sua modulazione nel tempo. A questo punto, si può procedere con la moltiplicazione per l'ampiezza del rumore, così da ottenere il segnale rumoroso legato al moto del laser. Queste operazioni sono riassunte in un semplice passaggio:

```
speckle_noise=n_speckle*abs(v_scan)/max(abs(v_scan));
```

Si ripete il valore finale a coda del segnale rumoroso, per gli stessi motivi indicati in precedenza in *speckle_armonic*.

Lo *speckle noise* che si ottiene è mostrato nella seguente figura.

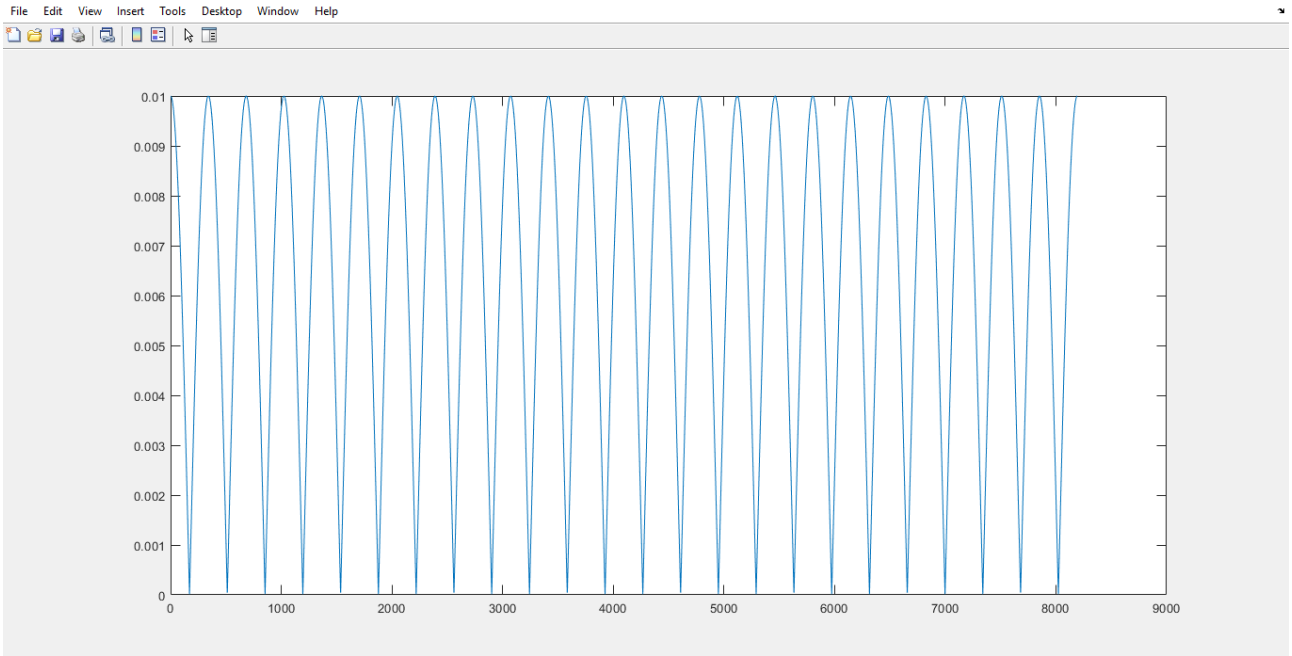

Figura 2.4. Speckle noise mediante speckle_abs

I rumori definiti sono sovrapposti alla velocità del segnale per ottenere il segnale reale nel tempo, come già spiegato.

I seguenti *plot* sono ottenuti dal *running* del programma *Estrai_CS* nella versione con *speckle_armonic*.

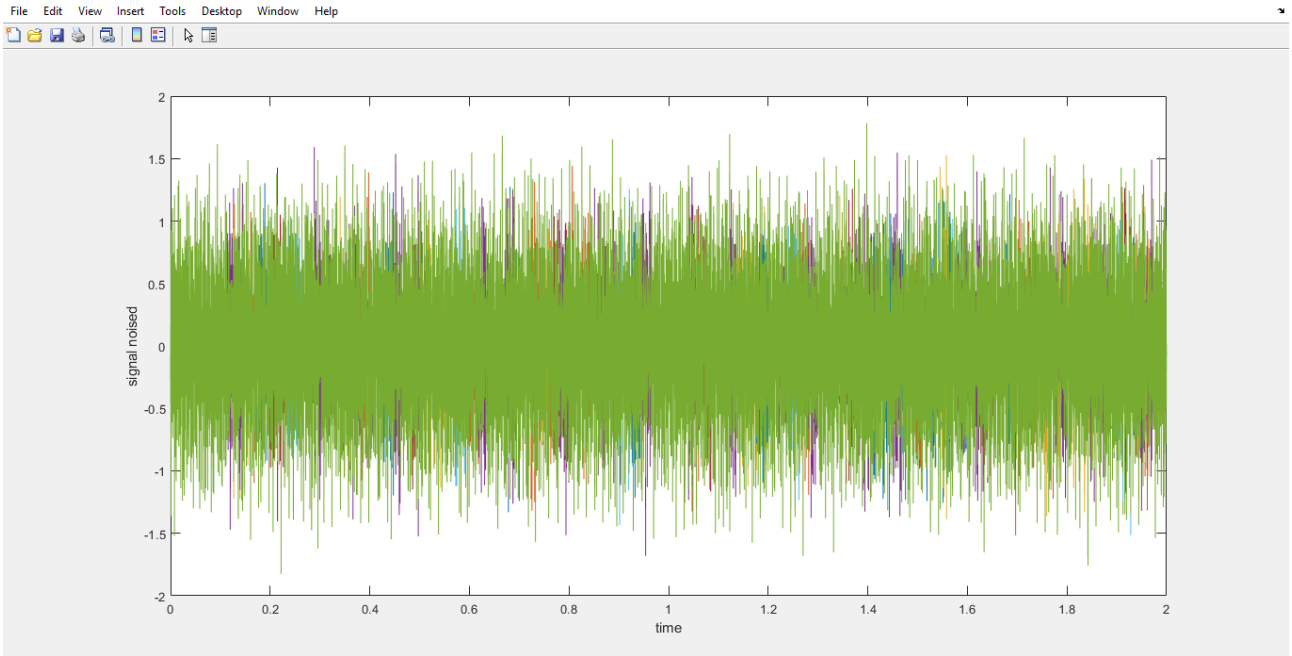

Figura 2.5. Segnale reale nel dominio del tempo

Al segnale ottenuto viene applicata la funzione già introdotta *fttic*, in modo da disporre del segnale reale in frequenza. Segue, quindi, il salvataggio dei dati destinati all'esportazione.

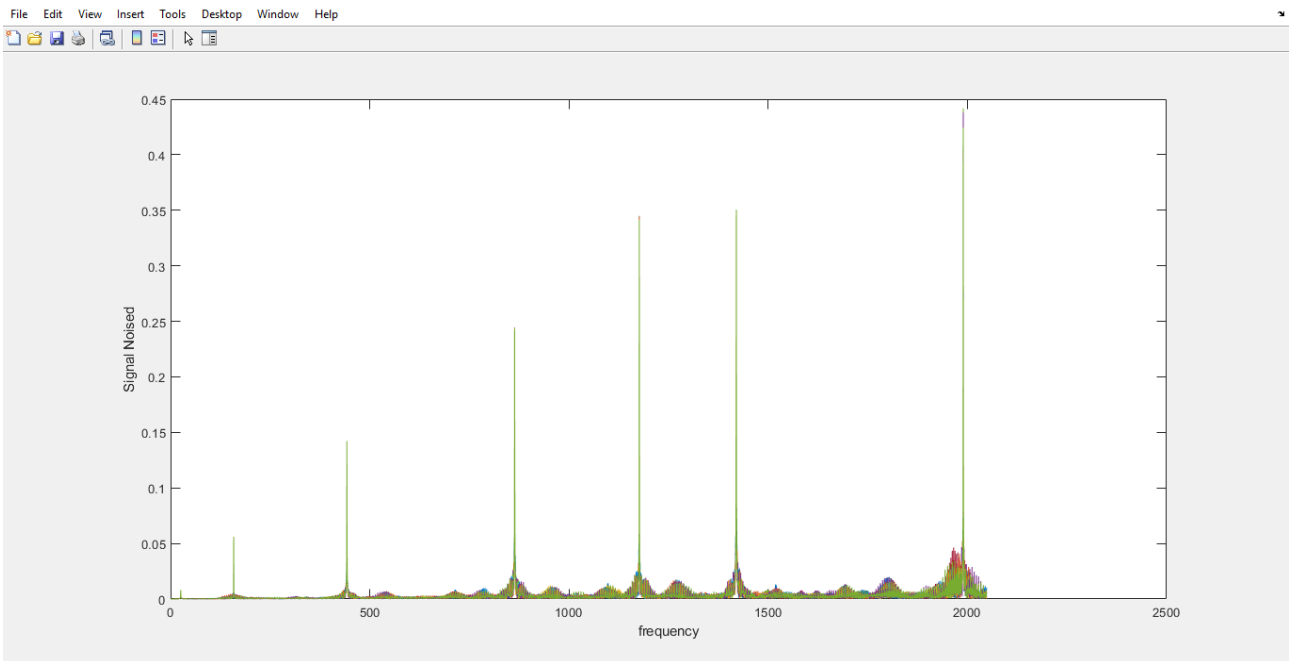

Figura 2.6. Segnale reale nel dominio della frequenza

2.4 Programma di estrazione Main_LauraJin_processor

L'algoritmo di estrazione delle forme di deflessione operativa si basa sulla soluzione, in senso almeno quadrato, di un insieme di equazioni lineari, che individua le relazioni tra i contributi di ciascun gruppo di frequenze (frequenza centrale e bande laterali), che potenzialmente definiscono il modello spettrale di un certo ODS con lo spettro complesso misurato.

L'elaborazione dell'algoritmo si basa sull'ipotesi che una forma di deflessione operativa possa essere modellata da un polinomio, i cui coefficienti sono direttamente correlati alle bande laterali: il numero di bande laterali è uguale all'ordine del polinomio che rappresenta l'ODS.

Il metodo proposto dal programma di estrazione sfrutta la rappresentazione matematica delle *sidebands* e la loro sovrapposizione per estrarre ampiezza e fase ad ogni frequenza dello spettro della banda laterale.

Quando due modi di vibrare della struttura sono vicini, lo spettro risultante mostrerà come la somma dei contributi complessi di ciascuno spettro (parti reali e immaginarie) e il modello di banda laterale, che in genere definisce la forma di deflessione operativa che si sta verificando a una certa frequenza di risonanza, si scompiglia e non è possibile ripristinare correttamente l'ODS.
Lo scopo dell'algoritmo di estrazione dei modi di vibrare è quello di separare il contributo di ciascun gruppo di frequenze (frequenza centrale e bande laterali correlate) e quindi consentire un corretto recupero degli ODS.
Il problema, per una singola combinazione di frequenza di scansione, può essere formulato come un sistema lineare: Ax=B.

Convenzionalmente, il riconoscimento delle frequenze di risonanza e il recupero degli ODS nel trattamento dei dati CSLDV viene eseguito a partire dall'osservazione dello spettro di uscita CSLDV.

Il programma è stato, perciò, modificato, inserendo come input il segnale elaborato nell'algoritmo *Estrai_CS*, da cui è importata anche la frequenza di scansione. È stato, inoltre, stimato il numero di *sidebands* a partire dalla stessa frequenza di scansione del laser mediante la già citata formula: `m=(length(f_scan)-1)/2.`

2.4.1 gen_CSLDV_func_SoR

L'algoritmo sfrutta una funzione ("gen_CSLDV_func"), che a partire dal segnale, dai dati relativi alla frequenza e dal numero di bande laterali stimato, permette di costruire e risolvere un sistema lineare, le cui soluzioni sono necessarie alla definizione degli ODS.

Anche questa funzione è stata modificata adeguatamente, ottenendo la versione "gen_CSLDV_func_SoR": la ricerca di soluzioni alternative a quella già presente, con lo scopo di migliorare l'efficienza di estrazione delle forme di deflessione operativa, ha condotto all'introduzione di uno *switch*, contente le diverse strategie.

Inoltre, è stato aggiunto il calcolo del resto.
Ottenuto "xx", soluzione del sistema lineare *AAxx=B*, il calcolo del resto permette di verificare la solidità della soluzione stessa. Attraverso la funzione "reshape" il resto è riorganizzato in ordine crescente e ridimensionato con "n" righe e "ns" colonne, dove *n* corrisponde alla lunghezza del vettore frequenza, mentre *ns* quella del vettore di frequenza di scansione "fscan".

```
res=(AA*xx)-B;
residuo=reshape(res,[n ns]);
```

Segue, quindi, la presentazione dei metodi di soluzione di un sistema lineare del tipo *Ax=B* approcciati nel corso dell'indagine.

2.5 Strategie risolutive di un sistema lineare

La risoluzione di un sistema lineare di equazione è conseguibile in MATLAB mediante diverse alternative di calcolo.

2.5.1 Metodo convenzionale

Un generico sistema lineare del tipo *Ax=b* si può sempre risolvere ricorrendo alla via più intuitiva:

```
x=inv(AA)* b;
```

o preferibilmente:

```
x=A\b;
```

Se il sistema è di dimensioni elevate, il calcolo potrebbe essere lento o, in alcuni casi, troppo pesante, con rischio di "Out of Memory" a causa del fill-in.

Il fill-in di una matrice è costituito da quei valori che da zero diventano diversi da zero durante l'esecuzione di un algoritmo. Per ridurre i requisiti di memoria e il numero di operazioni aritmetiche utilizzate in un algoritmo, è utile minimizzare il fill-in, scambiando righe e colonne della matrice o, in alternativa, si può optare per un'altra strategia, di seguito presentata.

In matematica, in particolare in analisi numerica, una matrice sparsa è una matrice i cui valori sono quasi tutti uguali a zero.
Quando delle matrici sparse vengono memorizzate e gestite su un computer, risulta proficuo, e spesso anche una necessità, utilizzare algoritmi specializzati e strutture dati, che tengono conto della natura sparsa della matrice. Svolgere delle operazioni utilizzando le strutture e gli algoritmi matriciali usuali risulta un'operazione molto lenta, da cui conseguono anche a grandi sprechi di memoria, se la matrice da gestire è di tipo sparso.
I dati sparsi sono, per loro natura, facilmente comprimibili e la loro compressione comporta quasi sempre un utilizzo significativamente inferiore di memoria.

In particolare, in MATLAB si può far ricorso alla funzione "sparse": *S = sparse (A)*, che converte il formato della memorizzazione della matrice da "full" a "sparse", cioè trascurando gli elementi nulli e registrando solo i valori diversi da zero con i relativi indici di posizione all'interno della matrice, riducendo quindi il fill-in. In poche parole, se una matrice contiene molti zeri, la conversione della matrice in *sparse* consente di risparmiare memoria, di ridurre il fill-in e quindi di rendere più agevole l'applicazione della matrice in successivi algoritmi e calcoli.

Disponendo di una matrice *A* in formato *sparse* è possibile ottenere la soluzione del sistema lineare *Ax=b* con il metodo già presentato in tempi ridotti, con l'accuratezza di convertire il risultato ottenuto in formato *full*.

```
x_sparse=A\b;      %soluzione formato sparse
x=full(x_sparse); %soluzione formato full
```

2.5.2 Fattorizzazione LU

Dato un sistema lineare *Ax = b*, le seguenti operazioni conducono ad un sistema equivalente, o, in altre parole, con la stessa soluzione: scambiare due equazioni; moltiplicare un'equazione per uno scalare; sostituire un'equazione con una combinazione lineare della stessa e di un'altra equazione. Motivi per cui ci si appella al "Metodo di eliminazione di Gauss", conosciuto anche come "Algoritmo di Gauss-Jordan", che

applica le operazioni precedentemente elencate al sistema per ottenerne uno equivalente a matrice triangolare superiore, risolvibile attraverso il metodo di sostituzione all'indietro già mostrato.

In MATLAB il *Metodo di eliminazione di Gauss* è alla base di svariati algoritmi volti alla risoluzione di sistemi lineari. Questo si deve al fatto che, in notazione matriciale, il *k-esimo* passo di eliminazione gaussiana, applicata ad un sistema del tipo $Ax=b$, può essere rappresentato come segue: $A(k+1) = MkA(k)$, con M matrice triangolare inferiore.

Si può dimostrare che il sistema di partenza equivale a $LUx = b$, dove L è l'inversa di M ed U è prodotto di M ed A: A è stata *fattorizzata* come LU.

In MATLAB, la "Fattorizzazione LU" è tradotta nella funzione "lu":

```
[L,U]=lu(A);

y=L\b;

x=U\y;
```

2.5.3 Metodi iterativi

I metodi diretti consentono la costruzione della soluzione esatta, a meno degli errori di arrotondamento, in un numero finito di passi. Per sistemi con matrici dense i metodi diretti sono generalmente preferibili. Per sistemi con matrici sparse e di ordine elevato, i metodi diretti diventano impraticabili a causa dei tempi di calcolo e degli errori per il fill-in della matrice (è possibile applicare alcune strategie già introdotte). Pertanto tali metodi risultano particolarmente convenienti quando la matrice A del sistema è di grandi dimensioni e sparsa. Infatti, quando la matrice A è sparsa, cioè il numero degli elementi non nulli è di molto inferiore al numero degli elementi nulli, applicando i metodi diretti può accadere che vengano generati elementi diversi da zero in corrispondenza degli elementi nulli della matrice di partenza (fenomeno del fill-in). Questo non avviene, applicando i metodi iterativi, in quanto essi si limitano ad utilizzare gli elementi non nulli della matrice, lasciando inalterati gli elementi nulli.

I metodi iterativi lasciano immutata la struttura di A e costruiscono una successione infinita di vettori, che sotto opportune ipotesi converge alla soluzione x. Per i metodi iterativi è necessario valutare condizioni di applicabilità e costo computazionale (di ogni iterazione), problemi relativi annessi.

2.5.3.1 Metodi iterativi di decomposizione

Idea base: data una stima iniziale $x^{(0)}$ della soluzione del sistema lineare $Ax = b$, con $A \in R^{n\times n}$, $det(A) \neq 0$, $b \in R^n$, si costruisce una successione di vettori $\{x^{(k)}\}$ con $k\geq 0$, che converga (si spera) ad x, risolvendo ad ogni iterazione dei sistemi lineari molto più semplici di quello di partenza.

Si consideri una decomposizione (splitting) della matrice A del tipo $A = M + N$, $det(M) \neq 0$. Allora si ha $Ax = (M + N)\, x = b \Longleftrightarrow Mx = -Nx + b$.

Si supponga di essere all'iterazione $k + 1$, di avere quindi a disposizione $x(k)$ e di voler calcolare $x(k+1)$. Si definisce $x(k+1)$, in modo che valga $Mx^{(k+1)} = -Nx^{(k)} + b$. Allora $x^{(k+1)} = -M^{-1}Nx^{(k)} + M^{-1}\, b$, che può essere scritto in forma compatta come $x^{(k+1)} = Bx^{(k)} + c$, avendo posto $B = -M^{-1}N$ e $c = M^{-1}b$ (processo iterativo 2).

La matrice $B = -M^{-1}N$ è la matrice d'iterazione ed individua il particolare metodo.

È necessario verificare le condizioni di convergenza del metodo iterativo.

Definizione: Convergenza
"Il procedimento iterativo (2) si dice convergente se, per ogni vettore iniziale $x^{(0)}$, la successione $\{x^{(k)}\}$ converge ad un vettore limite y; $\lim_{k\to\infty} x^{(k)}=y$ cioè se, per ogni $\varepsilon>0$, esiste un indice v tale che per ogni $k>v$ si ha $\|x^{(k)} - y\| \leq \varepsilon$."

Teorema: Condizione necessaria e sufficiente alla Convergenza

"Sia $A=M+N$ una matrice di ordine n, con $det(A) \neq 0$, e $B=-M^{-1}N$ la matrice di iterazione del procedimento iterativo (2). Condizione necessaria e sufficiente per la convergenza del procedimento iterativo, comunque si scelga il vettore iniziale $x^{(0)} \in R^n$, al vettore $x=A^{-1}b$, è che $\rho(B) < 1$, ovvero che il raggio spettrale (che corrisponde all'autovalore di modulo massimo) della matrice di iterazione T sia minore di 1."

Il raggio spettrale indica non solo se il metodo iterativo converge, ma anche quanto velocemente converge. Intuitivamente, la velocità di convergenza è una misura di quanto velocemente l'errore tende a zero. La velocità di convergenza di un metodo iterativo è tanto maggiore quanto più il raggio spettrale della matrice di iterazione è vicino a 0; viceversa è tanto più lenta quanto più è vicino a 1.

Il calcolo del raggio spettrale della matrice di iterazione B è, anche utilizzando algoritmi idonei, piuttosto oneroso. Si preferisce, quindi, considerare condizioni anche più restrittive, ma facilmente verificabili, o individuare classi di matrici, per le quali la convergenza è garantita da risultati teorici. Individuare una condizione di convergenza sufficiente serve a questi scopi, giustificato anche dalla difficoltà di verifica della condizione necessaria e sufficiente.

Teorema: Condizione sufficiente alla Convergenza

"Se, per una qualche norma, risulta $\|B\|<1$, allora il processo iterativo $x^{(k)} = -Bx^{(k-1)} + M^{-1} b$ per $k=1,2,\ldots$ è convergente per ogni $x^{(0)}$."

Se $\|B\| < 1$ il metodo iterativo converge, dove $\|B\|$ è la *norma matriciale* di B. Un concetto, quello della norma matriciale, da accostare alla ben più nota norma vettoriale: una norma è una funzione che assegna ad ogni vettore di uno spazio vettoriale, tranne lo zero, una lunghezza positiva.

La dimostrazione della condizione sufficiente è la seguente: per ogni norma naturale $\rho(B) \leq \|B\|$.

2.5.3.1.1 Metodo di Jacobi

Il "Metodo di Jacobi" è anche definito come "metodo delle sostituzioni simultanee" o "metodo degli spostamenti simultanei".
Dato un sistema lineare $Ax=b$, il metodo di Jacobi prevede uno splitting della matrice A del tipo $A=E+D+F$, dove E è la matrice triangolare inferiore ricavata da A e avente diagonale nulla, D la matrice diagonale avente gli stessi valori della diagonale di A ed F la matrice triangolare superiore di A, avente diagonale nulla. Nel metodo di Jacobi le matrici M ed N, introdotte nella spiegazione generica dei metodi iterativi, corrispondono rispettivamente a D ed $E+F$. La matrice d'iterazione B_J sarà ottenuta come $B_J = -D^{-1} (E + F)$.

Pertanto il procedimento iterativo (2) diviene $x (k) = -D^{-1} (E + F) x^{(k-1)} + D^{-1} b$ per $k=1,2,\ldots$.

2.5.3.1.2 Metodo di Gauss-Seidel

Il "Metodo di Gauss-Seidel", conosciuto anche come "metodo degli spostamenti successivi", prevede uno splitting alternativo rispetto al metodo precedente.
Dato il sistema lineare $Ax=b$, A può essere pensata come somma di E, D, F. Considerate le matrici M ed N

come $M = E + D$ e $N = F$, la matrice di iterazione B_{GS} risulterà $B_{GS} = -(E + D)^{-1} F$ secondo il metodo di Gauss-Seidel. Pertanto, il procedimento iterativo (2) diviene $x(k) = -(E+D)^{-1} F x^{(k-1)} + (E+D)^{-1} b$ per $k=1,2,\ldots$.

Osservazioni

Per matrici a predominanza diagonale stretta (per righe o per colonne indifferentemente) il metodo di Jacobi converge.
Per matrici a predominanza diagonale stretta (per righe o per colonne) il metodo di Gauss-Seidel converge.

Per matrici simmetriche definite positive il metodo di Gauss-Seidel converge.

Teorema:

"Se la matrice *A* è a diagonale strettamente dominante, cioè $|a_{ij}| > \sum_{k=1,k\neq i}^{n} |a_{ik}|$ $i=1,2,\ldots,n$. Allora sia il metodo di Jacobi che quello di Gauss-Seidel convergono e si ha: $\| B_{GS} \| < \| B_J \| < 1$."

Teorema:

"Se la matrice *A* è simmetrica e definita positiva, il metodo di Gauss-Seidel è convergente."

In generale la convergenza di uno dei metodi non implica la convergenza dell'altro e viceversa. Tuttavia, quando entrambi convergono, la velocità di convergenza del metodo di Gauss-Seidel è generalmente superiore. Si possono fare le seguenti considerazioni, confrontando i due metodi:
$0 < \rho(BGS) < \rho(BJ) < 1$, $1 < \rho(BJ) < \rho(BGS)$, $\rho(BGS) = \rho(BJ) = 0$, $\rho(BGS) = \rho(BJ) = 1$.

2.5.3.1.3 Metodo di SoR

È possibile accelerare la convergenza di un metodo iterativo e addirittura rendere convergente un metodo che non lo è: proprietà relative ad una famiglia di metodi, nota come "metodi di rilassamento". L'idea alla base di questi metodi è la seguente: poiché la velocità di convergenza di un metodo iterativo dipende dal raggio spettrale della matrice di iterazione associata al metodo, un modo per cercare di accelerare la convergenza è quello di far dipendere la matrice di iterazione da un parametro, detto "parametro di rilassamento", e di scegliere tale parametro in modo tale che la matrice abbia minimo raggio spettrale.

Un metodo di rilassamento può essere ricavato a partire dal metodo di Gauss-Seidel.
Il metodo di Gauss-Seidel può essere riscritto nella forma: $x^{(k)}=x^{(k-1)}+r^{(k)}$, dove $r^{(k)}=x^{(k)}-x^{(k-1)}=-D^{-1}[Ex^{(k)}+Fx^{(k-1)}-b]-x^{(k-1)}$. Modificando il metodo come $x^{(k)}=x^{(k-1)}+\omega r^{(k)}$.
Scegliendo opportunamente il parametro $\omega>0$, si può accelerare la convergenza in modo significativo. A seconda di come viene scelto tale parametro, i metodi di rilassamento si distinguono in "under-relaxation methods", se $0<\omega<1$, e "over-relaxation methods", se $\omega<1$.
I metodi *under-relaxation* possono essere usati per ottenere convergenza su certi sistemi per cui non si ha convergenza con il metodo di Gauss-Seidel, mentre i metodi *over-relaxation* possono essere usati per accelerare la convergenza in sistemi in cui il metodo di Gauss-Seidel converge ma lentamente. Questi ultimi metodi vengono chiamati metodi "SOR" o "Successive Over-Relaxation".

La matrice di iterazione del metodo è $T_\omega = (D + \omega E)^{-1}[(1-\omega)D - \omega F]$.

Nell'applicazione di un metodo di rilassamento dipendente dal parametro nasce il problema della scelta ottimale del parametro ω, che, oltre ad assicurare la convergenza, renda minimo il raggio spettrale della matrice di iterazione T_ω, in modo da ottenere la massima velocità di convergenza. È noto infatti che, per matrici simmetriche definite positive e per $0 < \omega < 2$, il valore di $\rho(T_\omega)$ si mantiene <1, ma c'è un punto in cui è minimo. Il valore del parametro corrispondente al minimo del raggio spettrale è detto ω_{ottimo}, che richiede un calcolo impegnativo.

Condizioni di convergenza del metodo SoR

Il metodo "Successive Over Relaxation", comunemente detto "SOR" o "metodo del sovrarilassamento" richiede, innanzitutto, determinante di *A* non nullo, dove *A* è la matrice dei coefficienti in un sistema lineare del tipo *Ax=b*. Inoltre, presenta una condizione necessaria di convergenza*: 0< ω <2*; la sufficienza è ottenuta nel caso in cui *A* sia diagonale strettamente dominante, proprietà descritta nel teorema sopraccitato, di seguito riproposto.

Teorema:

"Se la matrice A è a diagonale strettamente dominante, cioè $|a_{ij}| > \sum_{k=1,k\neq i}^{n} |a_{ik}|$ *i=1,2,…,n*. Allora sia il metodo di Jacobi che quello di Gauss-Seidel convergono e si ha: $\|B_{GS}\| < \|B_J\| < 1$."

2.6 Applicazione e risoluzione

2.6.1 Testo del programma *Find_matrix*

```
close all
clear
clc

%%
load('AA')
load('B')
% AA=[1,1,1,;1,-1,1;1,1,-1;2,2,2;2,0,0];
% B=[1,2,3,2,5];
AAB=[AA,B'];
% rango_AA=rank(AA);
% rango_AAB=rank(AAB);

%% Sparse
S=sparse(AA);
s=sparse(B);
SS=sparse(AAB);

%% Rango
rango_AA=sprank(S);
rango_AAB=sprank(SS);

%% TEOREMA (Rouché - Capelli).
%Un sistema lineare ha almeno una soluzione se e solo se il rango della sua
matrice incompleta è uguale al rango della sua matrice completa.
%rank(A)=rank(A,b)
rank=rank_AA-rank_AAB; %rank=0 allora almeno una soluzione
if(rank<0)
    disp('Teorema di Rouché-Capelli verificato');
end

%% Soluzione1  %https://it.mathworks.com/help/matlab/ref/mldivide.html
%x1=AA\B;
x_sparse=SS\s;     %soluzione formato sparse
x1=full(x_sparse); %soluzione formato full

%% Soluzione2:Metodo con fattorizzazione LU
[L,U]=lu(AA);
y=L\B;
x2=U\y;

%% Ordinamento delle matrice
% RREF
% [R,p]=rref(AAB);
% AA_R=R(1:rango_AA,1:rango_AA);
% B_R=R(1:rango_AA,end);

%
[R,t]=elimgauss03(AAB);
AA_R=R(1:rango_AA,1:rango_AA);
B_R=R(1:rango_AA,end);


%% Verifica convergenza metodi iterativi (Gauss-Seidel,Jacobi)
% Matrice quadrata, anche se non diagonalizzata, con determinante diverso da
zero
```

```
det_AA_R=det(AA_R); %diverso da 0
%AA_R=E+D+F;
aa=diag(AA_R);  %vettore elementi diagonali AA
D=diag(aa);  %matrice diagonale
EE=tril(AA_R);
E=EE-D;
FF=triu(AA_square);
F=FF-D;

%% Convergenza Metodo di Jacobi o metodo delle sostituzioni simultanee. M = D, N
= E + F.
ansM_J=det(D); %diverso da 0
B_J=-(inv(D))*(E+F);
% Necessario e sufficiente
rhoJ=max(abs(eig(B_J)));
ansJ=sign(rhoJ-1); %deve essere negativo
%sufficiente= norma indotta di B minore di 1
ans_J=norm(B_J)-1;    %deve essere negativo

if(ansJ<0)
    disp('il Metodo di Jacobi va a convergenza (condizione necessaria e
sufficiente)');
end
%% Convergenza Metodo di Gauss-Seidel. M = E + D, N = F.
ansM_GS=det(E+D); %diverso da 0
B_GS=-(inv(E+D))*F;
% Necessario e sufficiente
rhoGS=max(abs(eig(B_GS)));
ansGS=sign(rhoGS-1); %deve essere negativo
% Sufficiente= norma di B minore di 1
ans_GS=norm(B_GS)-1; %deve essere negativo

if(ansGS<0)
    disp('il Metodo di Metodo di Gauss-Seidel va a convergenza (condizione
necessaria e sufficiente)');
end

%% Convergenza Metodo di SoR. (DDM)
AA_R_abs=abs(AA_R);   %matrice valore assoluto
aa=diag(AA_R_abs); %vettore elementi diagonali AA
D=diag(aa);  %matrice diagonale
BB=AA_R_abs-D;     %matrice AA con diagonale nulla
bb=sum(BB,2);
DDM=aa-bb;  %verificare sign(DDM)

%% Verifica SoR
if(max(sing(DDM))<0)
    disp('il Metodo di Metodo di SoR va a convergenza (condizione
sufficiente)');
end
```

Confrontandosi con un sistema lineare del tipo *Ax=b* di grandi dimensioni, il metodo di rilassamento SoR sembra essere il più appetibile fra le varianti introdotte precedentemente, pur presentando condizioni stringenti.

2.6.2 Sparsità

Sono introdotte e definite le matrici incompleta *A* e completa *A|b*, matrici *full*.

In MATLAB, come già mostrato in precedenza, si può far ricorso alla funzione “sparse”, “Create sparse matrix”: *S = sparse (A),* che converte il formato della memorizzazione della matrice da *full* a *sparse*, cioè trascurando gli elementi nulli e registrando solo i valori diversi da zero con i relativi indici di posizione all’interno della matrice. Oltre ad assicurare un minor impiego di memoria, rende più agevole l’applicazione della matrice in successivi algoritmi e calcoli.

La differenza del costo computazionale nelle operazioni di calcolo del rango e del determinante con matrice in formato *full* e in *sparse* è considerevole.

Dato un sistema lineare *Ax=b*, il calcolo del rango in MATLAB è possibile mediante l’applicazione della funzione “rank”, “Rank of Matrix”.

Si convertono le matrici *A* e *A|b* dal formato *full* in sparso mediate la funzione *sparse*, ottenendo le matrici denominate nel programma come *S* ed *SS*.
Calcolare il rango di matrici sparse in MATLAB è reso possibile dalla funzione “sprank”, “Structural Rank”.

Il rango strutturale di una matrice è il rango massimo di tutte le matrici con lo stesso modello diverso da zero. Una matrice ha un rango strutturale completo se può essere permutata in modo tale che la diagonale non abbia zero ingressi. Il rango strutturale ha un limite superiore rispetto al rango di una matrice, quindi soddisfa: *sprank (A)> = rank (full (A)).*

2.6.3 Teorema di Rouché-Capelli

La prima verifica effettuata sul sistema lineare in uscita dal programma riguarda, innanzitutto, il Teorema di Rouché-Capelli.

2.6.3.1 Enunciato

Teorema:

“Dato un sistema lineare di equazioni *Ax=b*, detta matrice incompleta *A* e completa *A|b*, esiste almeno una soluzione per il sistema se e solo se il rango della matrice completa è uguale al rango della matrice incompleta”

2.6.3.2 Programma

Nel programma proposto la verifica del teorema si effettua con la presa visione del risultato di una semplice differenza di ranghi di matrice incompleta *A* e completa `AAB=[AA,B']`, dove *B* è l’array dei termini noti, con dimensioni *(1,0),* di cui si fa il trasposto per ottenere un vettore colonna, *(0,1):*

```
rank=rank_AA-rank_AAB; %rank=0 allora almeno una soluzione
if(rank<0)
    disp('Teorema di Rouché-Capelli verificato');
end
```

2.6.3.3 Ciclo if

Nelle righe di programma relative al Teorema d Rouché-Capelli spicca un *ciclo if*, ossia una struttura condizionale del tipo *“se… allora…”*
La struttura condizionale semplice si rifà alla seguente impostazione:

```
if      <espressione logica è verificata>
        <esegui processo>
end
```

La presa visione della verifica di una condizione è immediata grazie all'utilizzo di un *ciclo if*, in cui sono inseriti il requisito da soddisfare e la funzione "disp(X)".
Esplicabile come "Display value of variable", "disp (X)" visualizza il valore della variabile *X* senza registrare il nome della variabile. Un altro modo per visualizzare una variabile è digitare il suo nome, che visualizza una "X =" iniziale prima del valore.
In breve, questo ciclo permette di avere un feedback della condizione, se soddisfatta, direttamente nella *Command Window*, una volta concluso il *running*.

2.6.3.4 Conclusione

Il Teorema di Rouché-Capelli non è risultato soddisfatto dal sistema lineare su cui si opera.

2.6.4 Condizioni di Convergenza di SoR

Successivamente si procede al controllo delle condizioni del metodo SoR, nonostante l'esito negativo della precedente verifica.

La scrittura del programma denominato "DDM" ha reso possibile la verifica della proprietà di diagonale strettamente dominante della matrice $A|b$ del sistema.
In breve, in ogni riga il modulo dell'elemento appartenente alla diagonale della matrice di riferimento deve risultare maggiore della somma dei moduli di tutti gli altri elementi della riga perché la condizione sia soddisfatta.
Il programma *DDM* inizialmente impone la funzione "abs", "Absolute value and complex magnitude", alla matrice $A|b$, equivalente all'applicazione del modulo a tutti i singoli elementi, in modo da operare in seguito esclusivamente con valori assoluti.
Mediante la funzione "diag", "Create diagonal matrix or get diagonal elements of matrix", si estrae un array corrispondente alla diagonale.
Avvalendosi della stessa funzione, si costruisce la matrice avente come diagonale un dato array. Definita, quindi, la matrice diagonale a partire dal vettore precedentemente estratto, la si sottrae alla matrice in valore assoluto per ottenerne una priva della diagonale. Si effettua la somma orizzontale delle colonne della matrice ottenuta: il vettore risultante viene sottratto all'array diagonale.
Il dato in uscita è un vettore, i cui elementi dovranno essere positivi per vedere soddisfatta la condizione. Attraverso un *ciclo if*, contenente la funzione *disp*, come già spiegato, si mostra il risultato della verifica.

Il risultato del controllo è stato negativo; la verifica relativa alla condizione di non nullità del determinante della matrice *A* è stata, comunque, portata a termine, mediante la funzione di calcolo del determinante presente in MATLAB "Matrix determinant", inseribile nello script come "det".

Il determinante di *A* è risultato nullo: anche la seconda condizione non è soddisfatta.

Il metodo di SoR non converge in questo caso di studio.

2.6.5 Programmi di ordinamento

Da questo si può ricavare che la matrice *A* ha una struttura sparsa, che cela un numero di righe nulle equivalente alla differenza tra il numero di righe della matrice e il rango calcolato.

Si può procedere in due modi: eliminare le righe completamente nulle, poco interessanti ai fini del sistema lineare, trascurando le frequenze corrispondenti; utilizzare altre strategie e funzioni che possano restituire una soluzione sostanziale, anche se approssimata in alcuni casi.

Potenzialmente è possibile eliminare righe nulle una volta individuate nella matrice *A*: ciò significa anche eliminare soluzioni, quindi frequenze che devono essere estratte dal programma una volta portata a termine la risoluzione del sistema lineare. Avere una riga nulla nella matrice corrisponde ad avere

un'equazione del tipo 0x=0, con tipo di soluzione indeterminata, oppure del tipo 0x=n con n≠0, con tipo di soluzione impossibile; si può quindi procedere al ritaglio della matrice $A|b$, avendo comunque l'accortezza di tenere sempre presente quali sono le frequenze deliberatamente trascurate effettuando questa operazione.
Eliminare semplicemente le righe nulle non basta: il risultato sarebbe una matrice rettangolare, di cui non è possibile, ovviamente, calcolare il determinante, a cui non è possibile applicare alcun metodo iterativo, compresi i metodi di rilassamento, come SoR, e, soprattutto, corrispondente ad un sistema con un numero di incognite maggiore del numero di equazioni.

Modificando il numero di frequenze incognite e di *sidebands* da fornire al programma, è stato possibile generare un sistema lineare con un numero di righe maggiore, quindi un sistema che dispone di più equazioni che incognite.

È necessario trovare a questo punto un programma di ordinamento per eliminare le righe nulle e rendere la matrice quadrata, requisiti necessari per l'applicazione dei metodi iterativi. Operando in questo modo, si spera anche di assicurare un determinante non nullo alla matrice quadrata, output del processo.

2.6.5.1 Mosse di Gauss: *rref*

Un modo semplice di agire è quello di applicare le "mosse di Gauss" ad $A|b$.
In MATLAB è già presente un algoritmo che permette di ottenere l'ordinamento mediante le mosse di Gauss, cioè l'iter di passaggi da applicare per ridurre a gradini una matrice, già esplicato precedentemente. La funzione nota con il nome di "ref" ricalca perfettamente le "mosse di Gauss"; in MATLAB è possibile usare "rref", "Reduced row echelon form".
In breve, questa funzione compie un *pivoting parziale*, secondo l'algoritmo di Gauss-Jordan.
In matematica, e più specificamente in algebra lineare, il *pivot* (in francese *perno*) o *elemento pivotale* di una matrice è l'elemento della matrice che viene scelto per primo da un algoritmo e che si richiede rispetti determinate proprietà allo scopo di far funzionare correttamente l'algoritmo, o più semplicemente per renderne l'esecuzione più efficiente.
Quando ci si riferisce a matrici a scalini, solitamente nell'ambito dell'eliminazione gaussiana, con *pivot* di una riga si intende il primo elemento non nullo della riga.
Per far sì che l'elemento in posizione di *pivot* rispetti le proprietà necessarie ad assicurare il funzionamento o l'efficienza dell'algoritmo, vengono effettuate quelle che sono definite come *mosse di Gauss*, già elencate nel paragrafo *Fattorizzazione LU*: questa serie di scambi volta ad individuare il *pivot* prende il nome di *pivoting* o *pivotazione*.
Quando vengono scambiate solo le righe o solo le colonne della matrice si parla di pivoting parziale o pivotazione parziale, quando vengono scambiate sia le righe che le colonne si parla invece di pivoting totale o pivotazione totale.

Ottenuta la matrice in uscita dall'algoritmo, di semplice svolgimento è il ridimensionamento della stessa: si procede alla definizione di una nuova matrice, stavolta quadrata, a partire da quella rettangolare, di cui si va a considerare un numero di righe di pari valore del rango della matrice completa già calcolato in precedenza. In breve, si prendono le righe fino all'ultimo *pivot*, eliminando tutte le altre a seguire, cioè quelle nulle, le cui equazioni corrispondenti sono identità o impossibili.

Il *running* dell'*rref* ha richiesto lunghi tempi di calcolo, dovuti anche alle ingenti dimensioni della matrice, conducendo ad un "Out of Memory".

Si abbandona, quindi, quest'ultima strategia, che presentava un costo computazionale importante, ma non la possibilità di usare programmi fondati sull'algoritmo di Gauss-Jordan.

2.6.5.2 Ordina

La successiva mossa è quella di scrivere un programma ad hoc per il caso in esame: invece di attuare una strategia di pivoting, si ordinano le righe della matrice in modo da ottenere una matrice diagonalizzata o approssimabile ad una diagonale.
Da qui il programma "Ordina".

2.6.5.2.1 Testo del programma *Ordina*

```
function [AA_ord, B_ord]=ordina(AA,B)

%%
for ii=1:size(AA,1)
    indi=find(abs(AA(ii,:))>0);
%     plot(AA(ii,:),'-*')
%     pause(0.01)
%     [ind(ii) m]=min(indi);
    [ind_min m]=min(indi);
    [ind_max m]=max(indi);
    ind(ii)=(ind_min+ind_max)/2;
    dime(ii)=size(indi,2);
end

% figure, plot(dime,'-*')

indici=[(1:length(ind))',ind'];

% indici(end,2)=0;     % !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
indici=sortrows(indici,2);

AA_ord=AA(indici(:,1),:);
B_ord=B(indici(:,1));
```

2.6.5.2.2 Spiegazione del programma

Il programma si sviluppa come una funzione: forniti gli input *AA* e *B*, li restituisce ordinati, in modo da avere una matrice completa diagonalizzata.

Si procede con un *ciclo for*, in cui avviene l'estrazione degli indici delle righe con valori maggiori di zero e di cui se ne prende il massimo ed il minimo di volta in volta nei passaggi successivi, scrivendo due array di indici.

L'ultimo passaggio all'interno del *ciclo for* prevede la definizione del vettore di indici *ind*, come media dei due array precedentemente calcolati.

Si introduce la variabile *indici*, matrice con due colonne, ottenuta dai trasposti di due vettori: uno costituito dalla dimensione dell'array di indici *ind*, l'altro costituito dal vettore stesso.

In seguito, è applicata la funzione "sortrows", "Sort rows of matrix or table". *B = sortrows (A)* ordina le righe di una matrice in ordine crescente in base agli elementi nella prima colonna. Quando la prima colonna contiene elementi ripetuti, l'ordinamento è in base ai valori nella colonna successiva.

Si ricavano, quindi, i nuovi *AA_ord* e *B_ord*, applicando gli indici ordinati.

L'applicazione di "issymmetric", "Determine if matrix is symmetric or skew-symmetric", consente una verifica sulla condizione desiderata: *tf = issymmetric (A)* restituisce 1 logico (*true*), se la matrice quadrata *A* è simmetrica; in caso contrario, restituisce 0 logico (*false).*

tf = issymmetric (A, skewOption) specifica il tipo di test. Specificare *skewOption* come 'skew' per determinare se A è *skew-simmetrico*.

2.6.5.2.3 Conclusioni

L'applicazione del programma conduce ad una matrice poco approssimabile al formato diagonale, bensì piuttosto vicina a quello triangolare.
Un ulteriore limite è derivato dall'estrema difficoltà a tracciare la posizione iniziale di ogni riga e, quindi, anche la sua corrispondenza con la frequenza relativa, cosa che rende inaccettabile l'eliminazione di una o più righe.

2.6.5.3 Mosse di Gauss

La via più giusta da intraprendere sembra ora quella dell'algoritmo di Gauss-Jordan.
In MathWorks è possibile trovare algoritmi che traducono le mosse di Gauss: la scelta è ricaduta sul programma "elimingauss03".
Il risultato del *running*: come in *rref*, la lunghezza di calcolo è un problema insuperato.

2.6.6 Alternative di risoluzione

La scelta di strategie di risoluzione alternative sono cadute sui metodi più usati: quello convenzionale e la Fattorizzazione LU.

2.6.6.1 Metodo convenzionale

Il primo ha un ingente costo computazionale, pur fornendo risultati concreti.
La lunghezza dei tempi di calcolo è stata superata effettuando l'operazione in formato *sparse,* pur dovendo riconvertire in *full* i risultati.
I valori delle incognite, ricavati dal *running*, non hanno soddisfatto le aspettative: applicato a questo caso di studio, il metodo tradizionale di risoluzione è piuttosto insoddisfacente.

2.6.6.2 Fattorizzazione LU

La Fattorizzazione LU, invece, presenta un problema di tempistica: il *running* può durare giorni.

La possibilità di applicazione degli altri metodi iterativi perde di significato, sapendo che le condizioni di convergenza non sono soddisfatte: il determinante della matrice ricavata inizialmente dal programma è non nullo; il formato rettangolare per ovvi motivi non può vedere applicato alcun metodo iterativo; non è stato possibile fornire matrici quadrate e riordinate soddisfacenti a cui applicare metodi iterativi.

2.6.6.3 linsolve

Un'ulteriore alternativa è vista in *linsolve*, "Solve linear system of equations", una funzione in MATLAB che permette la risoluzione dei sistemi lineari attraverso il solutore più adatto secondo il programma stesso.

X = linsolve (A, B) risolve il sistema lineare *AX = B* usando uno di questi metodi: quando *A* è quadrata, *linsolve* utilizza la Fattorizzazione LU con pivoting parziale; per tutti gli altri casi, *linsolve* utilizza la Fattorizzazione QR con la rotazione della colonna.

linsolve avverte se *A* è mal condizionato (per matrici quadrate) o rango carente (per matrici rettangolari).

X = linsolve (A, B, opts), è l'esplicitazione nello script per la risoluzione in MATLAB di un sistema *AX=B*; il tipo di risoluzione più appropriata è determinata dalle opzioni della struttura.
I campi in *opts* sono valori logici che descrivono le proprietà della matrice *A*. Ad esempio, se *A* è una matrice triangolare superiore, è possibile impostare *opts.UT = true* per fare in modo che *linsolve* utilizzi un

solutore progettato per matrici triangolari superiori.
linsolve non prova a verificare che *A* abbia le proprietà specificate in *opts*.

2.6.7 Conclusioni sulla risoluzione

In conclusione, i risultati più soddisfacenti, per ora, sono quelli forniti dal metodo risolutivo già presente nel programma di estrazione delle forme di deflessione operativa.

```
xx=(inv(AA'*AA+cc*eye(size(AA'*AA)))*AA')*B;
```

dove *AA* è la matrice dei coefficienti e *B* il vettore dei termini noti.

3. CONCLUSIONI

L'elaborazione dei programmi di estrazione dei segnali ottenuti dalle misurazioni SSLDV e CSLDV permette un confronto tra i segnali stessi.

Inoltre, l'individuazione del segnale reale nel tempo ricavato mediante CSLDV collabora in modo attivo alla completezza del quadro della simulazione e permette di procedere con l'estrazione dei modi di vibrare.

I due programmi possono essere comunque usati indipendentemente dallo scopo prefissato, cioè di supporto al programma preesistente di individuazione delle forme di deflessione operativa: sono cellule autosufficienti che permettono di ottenere il segnale in uscita indipendentemente dal conseguente bisogno o meno di calcolare i modi di vibrare; possono essere utilizzati, quindi, anche in funzione di altre ricerche legate allo strumento LDV.

In questi programmi è stata considerata minima interferenza causata dal rumore: in possibili simulazioni future si potrà aumentare l'ampiezza del rumore, considerando così un disturbo più presente, più realistico in genere.

La ricerca di metodi alternativi di risoluzione del sistema lineare ha portato ad escludere altri metodi più rapidi per il calcolo ed a giustificare l'uso della funzione già presente nel programma.
È stata dimostrata la complessità della matrice output della funzione "gen_CSLDV_func", che presenta proprietà non usuali (per esempio, il determinante non nullo).
Un aspetto da sottolineare è la difficoltà nel trattare una matrice di grandi dimensioni in MATLAB, cosa inevitabile al fine di raggiungere risultati con piccole approssimazioni.

L'obiettivo sarà quello di snellire programmi di riordinamento e di accelerare gli algoritmi di risoluzione senza comprometterne l'efficacia.

Inoltre, il programma di verifica del Teorema di Rouché-Capelli e delle condizioni di convergenza dei metodi iterativi non è strettamente vincolato a questo caso di studio: potrà essere impiegato in funzione di un qualsiasi sistema lineare del tipo $Ax=b$.

4. RIFERIMENTI BIBLIOGRAFICI E SITOGRAFICI

L. Mignanelli, P. Chiariotti, P. Castellini, M. Martarelli, “Blind identification of Operational Deflection Shapes from Continuous Scanning Laser Doppler Vibrometry data”

Milena Martarelli, “Exploiting the Laser Scanning Facility for Vibration Measurements”

https://it.mathworks.com/help/search.html

http://calvino.polito.it/~sberrone/Faculty/01ILRFW.2011/2_SistemiLineari.pdf

5. Ringraziamenti

Vorrei concludere l’elaborato rivolgendo al mio relatore, il professor Paolo Castellini, i miei più sentiti ringraziamenti per l’ingente quantità di tempo e dedizione dedicata, per il supporto mai mancato, per gli insegnamenti, non solo di tipo didattico, impartiti con grande competenza, professionalità, arguzia, pazienza.

Ringrazio immensamente la professoressa Milena Martarelli per il suo appoggio, fornito con puntualità e competenza, e per la sua costante disponibilità.

Infine, ma non per ultimo, un ringraziamento alla mia famiglia, che adoro, ed ai miei cari amici.