**Apéndice C. Programas de MATLAB**

**C.1 Introducción**

Los programas que se muestran a continuación se utilizaron para la obtención de los resultados en el capítulo 8. Los programas para ráfagas y vórtices están programados con la filosofía de programación orientada a objetos el cual consiste en crear clases que se llaman temporalmente dentro del programa principal de ráfaga o vórtice donde dichas clases tiene asignadas las propiedades y funciones especificas.

Para los demás programas (excepto la función de integral IntegrarS) se utilizó la filosofía de programación imperativa el cual consiste en escribir el código paso por paso como se desea que se ejecuten los comandos, este tipo de filosofía es el más utilizado dentro de la programación aplicado a la ingeniería.

Para el programa IntegrarS, el cual se basa en la integral de Simpson para la obtención de una función discreta con longitudes no equidistantes se hizo por medio de la filosofía de programación funcional el cual se basa en la definición de las ecuaciones, este tipo de programación se asemeja mucho al lenguaje matemático. Para el programa IntegrarS, en específico, se utilizó la recursividad que consiste en que la función esta definida llamándose así mismo.

## C.2 Programa para obtener la respuesta ante efectos de ráfaga («Buffeting.m»)

Para que el programa funcione se requiere de los *scripts*: «Puente.m», «Viento.M», «Integrar.m» y «IntegrarS.m». Además, requiere de dos archivos de entrada: «Puente.txt» y «VientoRafaga.txt». Al final crea un archivo de nombre «Resultado.txt» donde muestra los valores máximos, la desviaciones estándar, los factores pico y los desplazamientos en el dominio del tiempo para las direcciones horizontales y verticales.

% Buffeting.m

clear all

close all

% Captura de datos

datosPuente = dlmread('Puente.txt','',1,0);

datosVientoR = dlmread('VientoRafaga.txt','',1,0);

% LLenado de datos

puente = Puente;

viento = Viento;

% Asigna los datos a cada objeto

puente.L = datosPuente (:,2);

puente.Lexp = datosVientoR(:,1);

puente.m = datosPuente (:,3);

puente.fiYP = datosPuente (:,4);

puente.fiZP = datosPuente (:,5);

puente.fiXP = datosPuente (:,6);

puente.fiY = datosVientoR(:,4);

puente.fiZ = datosVientoR(:,5);

puente.fiX = datosVientoR(:,6);

puente.w = [datosPuente(1,8); datosPuente(2,8)];

puente.c = [datosPuente(1,7); datosPuente(2,7)];

% Punto a revisar

yr = datosVientoR(1,15);

zr = datosVientoR(1,16);

B = datosVientoR(1,17);

% Geometría

puente.B = datosVientoR(:,2);

puente.D = datosVientoR(:,3);

% Coeficiente aoerdinmámicos

puente.cD = datosVientoR(:,7);

puente.cL = datosVientoR(:,8);

puente.dCd = datosVientoR(:,9);

puente.cdl = datosVientoR(:,10);

% Obtención de masa modal

puente.mModal();

% Integral de fi

puente.integrarFi()

% Datos del viento

viento.rho=1.25;

viento.v=datosVientoR(1,11);

viento.h=datosVientoR(1,12);

% Turbulencia

puente.Iu=datosVientoR(1,13);

puente.Iv=datosVientoR(1,14);

puente.integrarFi();

%Amortiguamiento aerodinámico

%Definición de integrales

intY = 0;

intFiY = 0;

intZ = 0;

intFiZ = 0;

%Buffeting

intY = integrarS(puente.fiY.^2.\*puente.D.\* puente.cD,puente.Lexp);

intZ = integrarS(puente.fiZ.^2.\*(puente.B.\*puente.cdl.\*puente.D.\*puente.cD),puente.Lexp);

%Puente

intFiY = integrarS(puente.fiYP.^2.,puente.L);

intFiZ = integrarS(puente.fiZP.^2.,puente.L);

%Amortiguamiento aerodinámico

zetaAeY = -(viento.rho\*viento.v) /(2\* puente.w(1)\* puente.m1) \* intY / intFiY ;

zetaAeZ = -(viento.rho\*viento.v) /(4\* puente.w(2)\* puente.m2) \* intZ / intFiZ ;

%Función de trasferencia mecánica

Hy = abs (1 - ( viento.w ./ puente.w(1) ) .^2 + 2\*1i \*(puente.c(1) - zetaAeY) \*(( viento.w ./ puente.w(1) ))) .^( -1) ;

Hz = abs (1 - ( viento.w ./ puente.w(2) ) .^2 + 2\*1i \*(puente.c(2) - zetaAeZ) \*(( viento.w ./ puente.w(2) ))) .^( -1) ;

%Función de densidad de kaimal

[sNu,sNw] = viento.densidadEspectralKaimal();

%Co espectro - Constantes

coU = zeros(1 ,length(viento.w));

coW = zeros(1 ,length(viento.w));

cuY = 9;

cwY = 6;

%Función de aceptancia constantes

jY = 0;

jZ = 0;

jm = 0;

%Expansión y prolongación de datos

div = 4;

puente.D = puente.prolong ( puente.D , div );

puente.B = puente.prolong ( puente.B , div );

puente.cD = puente.prolong ( puente.cD , div );

puente.cL = puente.prolong ( puente.cL , div );

puente.dCd = puente.prolong ( puente.dCd , div );

puente.cdl = puente.prolong ( puente.cdl , div );

puente.fiY = puente.prolong ( puente.fiY , div );

puente.fiZ = puente.prolong ( puente.fiZ , div );

puente.Lexp = puente.expand ( puente.Lexp, div );

puente.L = puente.expand ( puente.L , div );

n = length(puente.Lexp);

x=cumsum(puente.Lexp);

jY=0;

jZ=0;

% función de acepantcia conjunta

for i=1:n

for j=1:n

Dx=abs(x(i)-x(j));

coU = exp (- cuY\* ( viento.w\*Dx ) /(2 \* pi \* viento.v));

coW = exp (- cwY\* ( viento.w\*Dx ) /(2 \* pi \* viento.v));

jY = jY + puente.fiY(i) \* puente.fiY(j) .\* ( (((2\* puente.Iu )/ B) .^2\* puente.D(i) \* puente.cD(i) \* puente.D(j) \* puente.cD(j) ) .\* coU.\* sNu + ( (puente.D(i)/puente.B(i)\*puente.dCd(i)-puente.cL(i)) \* puente.Iv ) .^2 .\* coW.\* sNw ) .\* puente.L(i) .\* puente.L(j);

jZ = jZ + puente.fiZ(i) \* puente.fiZ(j) .\* ( (2\* puente.cL(i)\* puente.Iu) ^2 .\* coU.\* sNu + (( puente.Iv^2\*( puente.cdl(i) +( puente.D(i) \* puente.cD(i) )./B) \*(puente.cdl(j) +( puente.D(j) \* puente.cD(j) )./ B)) .\*coW.\* sNw )) \* puente.L(i) \* puente.L(j) ;

end

end

% Normalización de la función de aceptancia

jNormY = jY /( [puente.intFi1] ) ^2;

jNormZ = jZ /( [puente.intFi2] ) ^2;

%Espectro de respuesta

espResDivFiY = (( viento.rho \*viento.v ^2.\* B) /(2\* puente.m1 \* puente.w(1)^2) ) ^2 \* Hy.^2.\* jNormY;

espResDivFiZ = (( viento.rho \*viento.v ^2.\* B) /(2\* puente.m2 \* puente.w(2)^2) ) ^2 \* Hz.^2.\* jNormZ;

%Respuesta al extremo del puente

espResY = espResDivFiY\*yr^2;

espResZ = espResDivFiZ\*zr^2;

%Desviación estándar

Nomega = length ( viento.w);

Domega = ( viento.w ( Nomega ) - viento.w(1) ) /( Nomega -1) ;

intSy = integrar(Hy.^2.\*jNormY,Domega);

intSz = integrar(Hz.^2.\*jNormZ,Domega);

sRy = abs(yr)\*viento.rho \*viento.v^2\*B/(2\*puente.m1\*puente.w(1)^2)\*sqrt(intSy)

sRz = abs(zr)\*viento.rho \*viento.v^2\*B/(2\*puente.m2\*puente.w(2)^2)\*sqrt(intSz)

%Respuesta en el dominio del tiempo.

%Divisions de la frecuencia del viento

Nw = length ( viento.w );

dw = ( viento.w( Nw ) - viento.w(1)) / Nw ;

%Simulación de 10 minutos

t = linspace (0 ,600 ,600) ;

ry = zeros ( 1 , length(t) );

rz = zeros ( 1 , length(t) );

for i = 1: Nw

ry = ry+ sqrt(2\* espResY(i)\* dw ) .\* cos ( viento.w(i)\*t - rand (1) \*2\* pi);

rz = rz+ sqrt(2\* espResZ(i)\* dw ) .\* cos ( viento.w(i)\*t - rand (1) \*2\* pi);

end

%Factor pico

maxY = max(abs(ry))

maxZ = max(abs(rz))

kpY = maxY / sRy

kpZ = maxZ / sRz

%Gráfica de la respuesta

subplot (2 ,1 ,1)

plot (t , ry)

grid

xlabel ( 'T [s] ')

ylabel ( ' r\_y [m] ')

hold all

subplot (2 ,1 ,2)

plot (t , rz )

grid

xlabel ( 'T [s] ')

ylabel ( ' r\_z [m] ')

%Impresión de resultados

archivo= fopen ('Resultado.txt', 'w');

fprintf(archivo,'%s\r\n',datetime);

fprintf(archivo,'Desplazamiento máximo horizontal: %6.4f m\r\n',maxY);

fprintf(archivo,'Desplazamiento máximo vertical: %6.4f m\r\n',maxZ);

fprintf(archivo,'DesviaciÃ³n estándar horizontal: %6.4f m\r\n',sRy);

fprintf(archivo,'DesviaciÃ³n estándar vertical: %6.4f m\r\n',sRz);

fprintf(archivo,'Factor pico horizontal: %6.4f\r\n',kpY);

fprintf(archivo,'Factor pico vertical: %6.4f\r\n\r\n',kpZ);

fprintf(archivo,'%12s %12s %12s\r\n','T[s]', 'Desp Hor [m]', 'Desp Ver [m]');

fprintf(archivo,'%12.4f %12.4f %12.4f\r\n',[t;ry;rz]);

fclose(archivo);

A continuación se presenta un ejemplo de los archivos de entrada «Puente.txt» y «VientoRafaga.txt» para usar el programa «Buffeting».

«Puente.txt»

No. L Masa FiY FiZ FiX C w

1 3.4 9000 1 1 0 0.05 6.0822

2 3.4 15400 0.9444 0.9504 0 0.05 3.9671

3 3.4 13750 0.8884 0.8994 0

4 3.4 13500 0.8321 0.8472 0

5 3.4 15400 0.7757 0.7941 0

Este ejemplo consta de 5 elementos, con encabezado, la primera columna se refiere al número de elemento, la segunda a la lognitud del elemento, la tercera a la masa, las tres siguientes a las formas modales. Las últimas dos corresponden al amortiguamiento crítico y frecuencias en rad/s respectivamente, siendo la primera fila la correspondiente al modo horizontal y la segunda al modo vertical.

«VientoRafaga.txt»

L Ancho (B) Alto (D) fiy fiz fix Cd CL dCd dCL V H Iu Iv Yr Zr B

3.4 13.64 7.74 1 1 0 1.83 1.27 -0.3 0.5 38.24 192 0.12 0.06 1 1 13.64

3.4 13.64 7.74 0.9444 0.9504 0 1.83 1.27 -0.3 0.5

3.4 13.64 7.74 0.8884 0.8994 0 1.83 1.27 -0.3 0.5

3.4 13.64 7.74 0.8321 0.8472 0 1.83 1.27 -0.3 0.5

3.4 13.64 7.74 0.7757 0.7941 0 1.83 1.27 -0.3 0.5

Este ejemplo consta de 5 elementos, con encabezado, la primera columna se refiere al a la longitud del elemento, la segunda al ancho del tablero, la tercera al peralte del tablero, las tres siguientes a las formas modales, las columnas 7, 8, 9 y 10 corresponden los coeficientes de arrastre, levante, derivada del coeficiente de arrastre y derivada del coeficiente de levante respectivamente. Las últimas siete son valores independientes de cada elemento que corresponden a la velocidad media de diseño, la altura media del puente, el índice de turbulencia en la dirección u y w, el valor de la forma modal en el punto deseado en ambas direcciones y el ancho medio del puente.

## C.3 Programa para obtener la respuesta ante desprendimiento de vórtices «Vortices.m»

Para que el programa funciones se requieren de los programas Puente.m, Viento.M, Integrar.m, IntegrarS.m. Al final crea un archivo de nombre «Resultado.txt» donde muestra el desplazamiento máximos la desviación estándar, el factor pico y los desplazamientos en el dominio del tiempo para las vertical.

%Vortices.m

clear all

close all

%Captura de datos

datosPuente = dlmread('Puente3.txt','',1,0);

datosVientoV = dlmread('VientoVortices3.txt','',1,0);

%LLenado de datos

puente = Puente;

viento = Viento;

%Asigna los datos a cada objeto

puente.L = datosPuente (:,2);

puente.Lexp = datosVientoV(:,1);

puente.m = datosPuente (:,3);

puente.fiYP = datosPuente (:,4);

puente.fiZP = datosPuente (:,5);

puente.fiXP = datosPuente (:,6);

puente.fiZ = datosVientoV(:,2);

puente.fiX = datosVientoV(:,3);

puente.w = [datosPuente(1,8); datosPuente(2,8)];

puente.c = [datosPuente(1,7); datosPuente(2,7)];

puente.integrarFi();

puente.mModal();

%Propiedades del viento

viento.Kaz = datosVientoV(1,4);

viento.az = datosVientoV(1,5);

%Datos de espectro de carga

viento.lambdaZ = datosVientoV(1,6);

viento.sQz = datosVientoV(1,7);

viento.bZ = datosVientoV(1,8);

%Datos del viento

viento.rho = 1.25;

viento.v = datosVientoV(1,9);

puente.St = datosVientoV(1,10);

%Valor del modo al extremo del puente en direccion Z

fiI = datosVientoV(1,11);

wZ = puente.w(2);

mZ = puente.m2;

D = datosVientoV(1,13);

B = datosVientoV(1,12);

puente.intFi2=integrarS(puente.fiXP.^2+puente.fiZP.^2,puente.L);

intVor = integrarS(( puente.fiX.^2 + puente.fiZ.^2),puente.Lexp);

%Desviación estándar

Vr = D\* wZ /(2\* pi\* puente.St ); %Velocidad resonante

V = D\*puente.w(2)/(2\*pi\*puente.St);

z = ((4\* puente.m2 \* puente.c(2) ) /( viento.rho \*B^2\* viento.Kaz )) \*( puente.intFi2 / intVor );

b = fiI /(2^(5/2) \* pi ^(7/4) ) \* (( viento.rho \*D^3\* viento.lambdaZ ) /( puente.m2 \* viento.bZ \* viento.Kaz \* puente.intFi2) )^(1/2) \* ( viento.sigmaQz /( puente.St ^2\* viento.az ) ) \* (V / Vr ) ^(3/2) \* exp ( -(1/2) \*((1 -( Vr /V)) / viento.bZ )^2) ;

sAux = ((1 - z) /2 + (((1 - z ) /2) ^2 + b ^2) ^(1/2) ) ^(1/2) ;

sRz = sAux \* viento.az\*D

%Amortiguamiento aerodinámico

zAe=((viento.rho\*B^2\*viento.Kaz)/(4\*mZ)).\*((1-(sRz./(viento.az\*D)).^2)\*(puente.intFi2/intVor));

%Función de transferencia mecánica

H = abs (1 - ( viento.w ./ wZ ) .^2 + 2\*1i \*( puente.c(2) - zAe ) \*(( viento.w(2) ./ wZ ))).^( -1) ;

%Espectro de carga

wS = (2\* pi\* puente.St \*Vr)/D ; %Frecuencia resonante

sQ = 2\* viento.lambdaZ \* D \* (((0.5\* viento.rho \*Vr ^2\* B\* viento.sQz) ^2) /( sqrt (pi)\* wS \* viento.bZ )) \* exp( -(((1 - viento.w ./ wS ) ./ viento.bZ ) .^2)) .\* intVor ;

%Espectro de respuesta

espRes = (( fiI^2\* H .^2) ./(( wZ ^2\* mZ \* puente.intFi2) .^2) ) .\* sQ ;

%Función en el dominio del tiempo.

t = linspace (0 ,600 ,600) ;

n = length([viento.w]);

dw = ( viento.w (n)- viento.w(1)) / n ;

rz = zeros (1 ,length(t));

for i = 1: n

rz = rz + sqrt (2\* espRes(i)\* dw ) .\* cos ( viento.w(i)\*t - rand (1) \*2\* pi);

end

%Valor máximo

maxZ = max (abs (rz))

kpz = maxZ / sRz

%Gráfica de la respuesta en el tiempo

plot (t , rz)

grid

xlabel ( 'T [s] ')

ylabel ( ' r\_z [m] ')

%Impresión de resultados

archivo= fopen ('Resultado.txt', 'w');

fprintf(archivo,'%s\r\n',datetime);

fprintf(archivo,'Desplazamiento máximo vertical: %6.4f m\r\n',maxZ);

fprintf(archivo,'Desviación estándar vertical: %6.4f m\r\n',sRz);

fprintf(archivo,'Factor pico vertical: %6.4f\r\n\r\n',kpz);

fprintf(archivo,'%12s %12s \r\n','T[s]','Desp Ver [m]');

fprintf(archivo,'%12.4f %12.4f \r\n',[t;rz]);

fclose(archivo);

A continuación se presenta un ejemplo de los archivos de entrada «Puente.txt» y «VientoVortices.txt» para usar el programa «Vortices».

«Puente.txt»

No. L Masa FiY FiZ FiX C w

1 3.4 9000 1 1 0 0.05 6.0822

2 3.4 15400 0.9444 0.9504 0 0.05 3.9671

3 3.4 13750 0.8884 0.8994 0

4 3.4 13500 0.8321 0.8472 0

5 3.4 15400 0.7757 0.7941 0

Este ejemplo consta de 5 elementos, con encabezado, la primera columna se refiere al número de elemento, la segunda a la longitud del elemento, la tercera a la masa, las tres siguientes a las formas modales. Las últimas dos corresponden al amortiguamiento crítico y frecuencias en rad/s respectivamente, siendo la primera fila la correspondiente al modo horizontal y la segunda al modo vertical.

«VientoRafaga.txt»

Lexp FiZ FiX kAz aZ Lz sQz bZ Velocidad St Zr Ancho Alto

3.4 1 0 0.2 0.4 2 1 0.15 38.24 0.11 1 13.64 7.74

3.4 0.9504 0

3.4 0.8994 0

3.4 0.8472 0

3.4 0.7941 0

Este ejemplo consta de 5 elementos, con encabezado, la primera columna se refiere al a la longitud del elemento, la segunda y tercera columna son las formas modales verticales y horizontal a lo largo del puente respectivamente, las siguientes 5 propiedades corresponden a las propiedades del viento que se muestran también en la clase «Viento», la penúltima columna corresponde al valor de la forma modal vertical en el punto deseado y la última al ancho promedio del puente.

**C.4 Código de las clases**

Para el funcionamiento de los programas «Buffeting.m» y «Vortices.m» es necesario crear dos clases: «Viento.m» y «Puente.m». Estas dos clases permiten crear los objetos con sus propiedades y métodos utilizados en los programas para obtener la respuesta ante los efectos de ráfagas como de vórtices

**C.4.1 Clase Viento «Viento.m»**

%Viento.m

classdef Viento

properties

rho = 1.25; %Densidad del viento

v = 0; %Velocidad media del viento

h = 0; %Altura a revisar

w = linspace (0 ,20 ,2000) ; %frecuencias del viento

%Propiedades de desprendimiento de vï¿½rtices

Kaz = 0.2;

az = 0.4;

lambdaZ = 2;

sQz = 1;

bZ = 0.15;

sigmaQz = 1

end

methods

function [sNu,sNw]=densidadEspectralKaimal(obj)

%Obtiene la función de densidad espectral en todo el dominio w para las dos direcciones

%Constantes de la función

Au = 6.8/(2\*pi);

Aw = 9.4/(2\*pi);

H = [obj.h];

xLu = 100\*(H/10)^0.3;

xLw = xLu/12;

V = [obj.v];

%Definición de variables

w = [obj.w];

sNu = zeros(1,length(w));

sNw = zeros(1,length(w));

%Función de densidad espectral

sNu=Au.\*xLu./(V \*(1+1.5\* Au.\*w\*(xLu/V )).^(5/3) );

sNw=Aw.\*xLw./(V \*(1+1.5\* Aw.\*w\*(xLw/V )).^(5/3) );

end

end

end

**C.4.2 Clase Puente «Puente.m»**

%Puente.m

classdef Puente<handle

properties

%Formas modales expuestos

fiX = [];

fiY = [];

fiZ = [];

%Formas modales totales del puente

fiXP = [];

fiYP = [];

fiZP = [];

%Longitudes caracteristicas del puente

%B=ancho D=alto L=Longitud de cada elemento m=masa de cada elemento

%Lexp=Longitud expuesta

B = [];

D = [];

L = [];

Lexp = [];

%Propiedade dinámicas del viento

m = []; %masa de cada elemento

m1 = 0 ; %masa modal efectiva del modo1

m2 = 0 ; %masa modal efectiva del modo2

w = []; %Frecuencias de cada modo

c = []; %Coeficiente de amortiguamiento de cada modo

cD = []; %Coeficiente de arrastre

cL = []; %coelficiente de levante

dCd = []; %Derivada del coeficiente de arrastre

cdl = []; %Derivada del coelficiente de levante

St = 0 ; %Número de Strouhal

Iu = 0 ; %Intensidad de turvulencia en direecion x

Iv = 0 ; %Intensidad de turvulencia en direecion y

%Integrales de las formas modales al cuadrado

intFi1 = 0; %Integral del modo 1 direccion y

intFi2 = 0; %integral del modo 2 direccion x z

end

methods

function mModal(obj)

%Obtiene la masa modal efectiva para las dos direcciones

n = length([obj.L]);

intM1 = 0;

intPhi1 = 0;

intM2 = 0;

intPhi2 = 0;

%Datos del puente

%integral sobre L

intM1 = integrarS([obj.fiYP].^2.\* [obj.m],[obj.L]) ;

intPhi1 = integrarS([obj.fiYP].^2,[obj.L]);

intPhi2 = integrarS([obj.fiXP].^2+[obj.fiZP].^2,[obj.L]);

intM2 = integrarS(([obj.fiXP].^2+[obj.fiZP].^2).\*[obj.m],[obj.L]);

[obj.m1] = intM1/intPhi1;

[obj.m2] = intM2/intPhi2;

end

function integrarFi(obj)

n = length([obj.L]);

for i = 1 : n

% Longtiud total

[obj.intFi1] = integrarS([obj.fiYP].^2,[obj.L]);

[obj.intFi2] = integrarS([obj.fiXP].^2+[obj.fiZP].^2,[obj.L]);

end

end

function [ VarNueva ] = prolong (obj,var , div )

% aumenta la cantidad de datos

n = length (var);

VarNueva = zeros (0, (n -1) \* div +1) ;

VarNueva (1) = var (1) ;

for i = 0:n -2

delta = ( var (i +2) - var (i +1) )/ div ;

for j = 1: div

VarNueva ( div \*i +j +1) = VarNueva ( div \*i+j ) + delta ;

end

end

end

function [ VarNueva ] = expand (obj, var , div )

n = length (var);

VarNueva = zeros (0, (n - 1) \* div + 1) ;

VarNueva (2: div ) = (var (1) + var (2) \* 0.5) / div ;

VarNueva (1) = 0.5 \* VarNueva (2) ;

for i = 1 : n -3

delta = ( var (i +1) + var (i +2) ) /(2\* div );

VarNueva ( div \*i +1: div \*i+ div ) = delta ;

VarNueva ( div \*i + div ) = ( delta + ( var (i +2) + var (i +3) ) /(2\* div )) /2;

end

VarNueva ((n -2) \* div +1:( n -1) \* div ) = ( var (n -1) \*0.5+ var (n)) / div ;

VarNueva ((n -1) \* div +1) = 0.5\* VarNueva ((n -1) \* div ) ;

VarNueva ( div \*(n -3) + div ) = ( delta + ( var (i +2) +2\* var (i +3) ) /(2\* div )) /2;

end

end

end

**C. 5 Funciones usadas para los programas «Buffeting.m» y «Vortices.m»**

Para las funciones, se usaron las integrales de Simpson, sin embargo este método solo es válido para distancia equidistantes, por lo que hizo otro basado en la obtención del método Simpson.

**C.5.1 Integral de Simpson con distancias equidistantes «integrar.m»**

%integrar.m

function int=integrar(fx,Dx)

%integral de una serie de datos por medio del método de Simpson.

%Valido solamente para cantidad de valores impar.

n=length(fx);

int=0;

for i=1 : n

if i==1 || i==n

int=int+fx(i);

else

if mod(i,2)==0

int=int+4\*fx(i);

else

int=int+2\*fx(i);

end

end

end

int=int\*Dx/3;

end

Ejemplo de aplicación de la función en Matlab

%Aplicación de integrar en Matlab

x=lisnpace(0,1,10)

fx =sin(x);

dx=x(2)-x(1);

ifx=integar(fx,dx);

ifx=0.4296

**C.5.2 Integral de Simpson con distancias no equidistantes «integrar.s»**

%integrarS.m

function int=integrarS(fx,Dx)

%integral de una serie de datos por medio del mÃ©todo de simpson con distancias variables

n=length(fx);

if n>=3

x=fx(1:3);

dx=Dx(1:3);

fx(1:2)=[];

Dx(1:2)=[];

int=integralLocal(x,dx)+integrarS(fx,Dx);

else

int=integralLocal(fx,Dx)+0;

end

end

function int=integralLocal(fx,Dx)

int=0;

if length(fx)==3

h1=Dx(1);

h2=Dx(2);

y0=fx(1);

y1=fx(2);

y2=fx(3);

int=(h1+h2)/(6\*h1\*h2)\*(h2^2\*(y1-y0)+h1^2\*(y1-y2)+2\*h1\*h2\*(y0+y1+y2));

elseif length(fx)==2

h=Dx(1);

y0=fx(1);

y1=fx(2);

int=(y0+y1)\*h/2;

else

int=0;

end

end

Ejemplo de aplicación de la función en Matlab

%Aplicación de integrar en Matlab

x=lisnpace(0,1,10)

fx =sin(x);

dx=zeros(length(x),1)+x(2);

ifx=integar(fx,dx);

ifx=0.4596

## C.6 Programa para obtener la velocidad crítica de aleteo «FlutterVcr.m»

Para que el programa funciones requiere de los programas «derAer.m», «IntegrarS.m» y el archivo de entrada como se muestra en la Figura C.1. El archivo de entrada puede tener cualquier nombre siempre que el formato sea «.txt». En el archivo presentado a continuación se tiene limitada la búsqueda entre los rangos de 1 a 2 de la velocidad crítica con 200 intervalos entre estos rangos, esto se debe a que en ocasiones no existe solución a las raíces y Matlab arroja error por lo que el usuario tiene que variar el intervalo.

Al final crea un archivo de nombre «Resultado.txt» donde muestra la velocidad crítica y la frecuencia que corresponde además de los valores de las raíces reales e imaginarias.

|  |
| --- |
|  |
| Figura C.1 Archivo de entrada para obtención de la velocidad crítica |

%FlutterVcr.m

clc

clear all

[H1i,H2i,H3i,H4i,A1i,A2i,A3i,A4i]=derAer(1,2,200);

v=linspace(1,2,200);

[archivo,direccion] = uigetfile({'\*.txt'},'File Selector');

dirCom = strcat(direccion,archivo);

archivoID = fopen(dirCom,'r');

datos = textscan(archivoID, '%s','Delimiter', '\n');

matriz = datos{1};

varConst = zeros(1,7);

revConst = false;

revMat = false;

numConst = 0;

matConst = 0;

for i = 1 : length(matriz)

if revConst == false

constante = strsplit(matriz{i},'=');

if length (constante)== 2

num = str2double(strsplit(constante{2}));

numConst = numConst + 1;

varConst(numConst) = num(2);

if numConst == 7, revConst=true; end

end

elseif revMat == false && revConst==true

mat = strsplit(matriz{i});

matNum = str2double(mat);

if not(isnan(sum(matNum)))

switch matConst

case 0

L = matNum;

matConst = matConst + 1;

case 1

fiZ = matNum;

matConst = matConst + 1;

case 2

fiT = matNum;

matConst = matConst + 1;

revMat = true;

end

end

end

end

mz = varConst(1);

mt = varConst(2);

B = varConst(3);

wz = varConst(4);

wt = varConst(5);

zz = varConst(6);

zt = varConst(7);

rho = 1.25;

%Integral de formas modales

intFiZ = integrarS(fiZ.^2,L);

intFiT = integrarS(fiT.^2,L);

intFiZT = integrarS(fiZ.\*fiT,L);

%cambio de variable

g = wt/wz;

Bz = rho\*B^2/mz;

Bt = rho\*B^4/mt;

Y = intFiZT^2/(intFiT\*intFiZ);

dominio = length(v);

GrafRe = zeros(1,dominio);

GrafIm = zeros(1,dominio);

%Obtención de raices

for i=1:dominio

H1=H1i(i);

H2=H2i(i);

H3=H3i(i);

H4=H4i(i);

A1=A1i(i);

A2=A2i(i);

A3=A3i(i);

A4=A4i(i);

%Reales

R0 = 1;

R1 = 0;

R2 = -(1+g^2+4\*g\*zz\*zt+Bz/2\*g^2\*H4+Bt/2\*A3);

R3 = g\*(zt\*Bz\*g\*H1+zz\*Bt\*A2);

R4 = g^2\*(1+Bz/2\*H4+Bt/2\*A3+Bz\*Bt/4\*(A1\*H2\*Y-A2.\*H1+A3\*H4-A4\*H3\*Y));

raices = roots([R4 R3 R2 R1 R0]);

r = raices(imag(raices)==0);

GrafRe(i) = max(r);

%Imaginarios

I0 = -2\*(+zz\*g+zt);

I1 = -2\*(1/4\*(Bz\*g^2\*H1+Bt\*A2));

I2 = 2\*((zz\*(Bt/2\*A3+g)+zt\*g^2\*(Bz/2\*H4+1)));

I3 = 2\*(g^2\*((Bz\*Bt)/8\*(H1\*A3\*Y-H2\*A4\*Y-H3\*A1+H4\*A2)+1/4\*(Bz\*H1+Bt\*A2)));

raices = roots([I3 I2 I1 I0]);

r = raices(imag(raices)==0);

GrafIm(i) = max(r);

End

%Gráfica

fig1 = figure(1);

plot (v,GrafRe,'b-',v,GrafIm,'r-','LineWidth',1.25);

title ('Solucion para las ecuaciones de aleteo')

xlabel ('$\hat{V}\_{cr}$','Interpreter','latex','Fontsize',11)

ylabel ('$\hat{\omega}\_r$','Interpreter','latex','Fontsize',11)

grid on

h = legend('Raices reales','Raices imaginarias');

set(h,'Interpreter','latex','Location','SouthWest')

legend('boxoff')

%Punto de intersección

vcr=0;wr=0;

if GrafRe(1) - GrafIm(1) >= 0

dif=1;

else

dif=-1;

end

for i=1:length(GrafRe)

if (GrafRe(i) - GrafIm(i)) \* dif < 0

[vcr,wr]=interseccion(GrafRe(i-1),GrafRe(i),GrafIm(i-1),GrafIm(i),v(i-1),v(i));

end

end

%Escritura de resultados en .txt

archivo= fopen ('Resultado.txt', 'w');

fprintf(archivo,'%s\r\n',datetime);

fprintf(archivo,'Velocidad critica: %6.4f\r\n',vcr);

fprintf(archivo,'Frecuencia: %6.4f\r\n\r\n',wr);

fprintf(archivo,'%6s %6s %6s\r\n','Vcr', 'Re', 'Im');

fprintf(archivo,'%6.4f %6.4f %6.4f\r\n',v, GrafRe, GrafIm);

fclose(archivo);

%Funciones

function [x,y]=interseccion(f1,f2,g1,g2,x1,x2)

%pendientes de las rectas

mf=(f2-f1)/(x2-x1);

mg=(g2-g1)/(x2-x1);

%ordenada al origen

bf=f1-mf\*x1;

bg=g1-mg\*x1;

%Interseccion

x=(bf-bg)/(mg-mf);

y=mf\*x+bf;

end

## A continuación se presenta un ejemplo del archivo de entrada para el programa «FlutterVcr.m»

Archivo de entrada de datos para obtención de velocidad crítica de Flutter

Propiedades dinámicas de la estructura

m = 10000.00 [kg/m] Masa modal a flexión.

I0 = 600000.00 [kg-m2/m] Momento másico de inercia.

B = 20.00 [m] Ancho del puente.

wh = 0.8 [rad/s] Frecuencia angular dirección vertical.

wt = 1.6 [rad/s] Frecuencia angular a torsión.

ch = 0.005 Amortiguamiento crítico dirección Z.

Cm = 0.005 Amortiguamiento crítico a torsión.

Matrices

Longitud de tramo [m]

1 1 1 1 1

Formas modales

Vertical

1.0 0.8 0.6 0.4 0.2

Horizontal

1.0 0.8 0.6 0.4 0.2

## C.7 Programa para obtener las derivadas aerodinámicas para una placa «derAer.m»

A continuación se muestre el programa para obtener las derivadas aerodinámicas a flexión y torsión de una placa propuesta por Theodorsen (1935), además, se usa en conjunto con «FlutterVcr.m». Este programa se usa como función en Matlab por lo que el usuario puede cambiar las ecuaciones aquí presentes para obtener la velocidad crítica de un cuerpo geométrico diferente a la placa.

Esta función requiere de valores de entrada una velocidad mínima «vMin», una máxima «vMax» y la cantidad de datos entre la velocidad mínima y máxima «Datos». Como resuldato, almacena en 8 variables las derivadas aerodinámicas, siendo las primeras cuatro a flexión y las últimas cuatro a torsión.

%derAer.m

function [H1,H2,H3,H4,A1,A2,A3,A4]=derAer(vMin,vMax,Datos)

if vMin<=0, vMin=0.01; end

v=linspace(vMin,vMax,Datos);

x = 0.5./v;

j0 = besselj(0,x);

j1 = besselj(1,x);

y0 = bessely(0,x);

y1 = bessely(1,x);

G = -(j1.\*j0+y1.\*y0)./((j1+y0).^2+(y1-j0).^2);

F = (j1.\*(j1+y0)+y1.\*(y1-j0))./((j1+y0).^2+(y1-j0).^2);

H1 = -2\*pi\*v.\*F;

H2 = pi/2\*(1+F+4\*G.\*v).\*v;

H3 = 2\*pi\*(F.\*v-G/4).\*v;

H4 = pi/2\*(1+4\*G.\*v);

A1 = -pi/2\*F.\*v;

A2 = -pi/8\*(1-F-4\*G.\*v).\*v;

A3 = pi/2\*(F.\*v-G/4).\*v;

A4 = pi/2\*G.\*v;

end

A continuación se presenta un ejemplo de cómo usar la función en Matlab

%Aplicación de derAer en Matlab

VelMin = 1;

VelMax = 2;

NumDatos=5;

[H1,H2,H3,H4,A1,A2,A3,A4]=derAer(VelMin,VelMax,NumDatos)

## C.8 Programa para obtener la respuesta debido al aleteo «flutterOde45.m»

Para que el programa funciones requiere de los archivos de entrada como se muestra en la Figura C.2 y Figura C.3. En la Figura C.3 se muestra la velocidad turbulenta en la dirección horizontal y vertical que se obtienen del programa de «SimulacionesWAWS.m». Además requiere del programa «flut8a45.m».

|  |
| --- |
|  |
| Figura C.2 Archivo de entrada para obtención de la respuesta ante aleteo |

|  |
| --- |
|  |
| Figura C.3 Archivo de entrada de viento para obtención de la respuesta ante aleteo |

%flutterOde45.m

clear

close all

proDim = 9;

derAer = 8;

proVie = 2;

proSim = 5;

coeAer = 5;

propiedades = proDim + derAer + proVie + proSim + coeAer;

datos = importar(propiedades);

rho = datos(1);

m = datos(2);

I = datos(3);

B = datos(4);

wh = datos(5);

wa = datos(6);

w = datos(7);

ch = datos(8);

ca = datos(9);

H1 = datos(10);

H2 = datos(11);

H3 = datos(12);

H4 = datos(13);

A1 = datos(14);

A2 = datos(15);

A3 = datos(16);

A4 = datos(17);

Cd = datos(18);

Cl = datos(19);

Cm = datos(20);

dCl = datos(21);

dCm = datos(22);

V = datos(23);

h0 = datos(24);

a0 = datos(25);

vh = datos(26);

va = datos(27);

t = datos(28);

dh = datos(29);

Vr = V/(w\*B);

[velU,velV] = importarVel();

k=B\*w/V;

q=1/2\*rho\*V^2;

h1 = H1 / m \* q \* B \* k / V;

h2 = H2 / m \* q \* B \* k / V;

h3 = H3 / m \* q \* B \* k^2 ;

h4 = H4 / m \* q \* B \* k^2 / B;

a1 = A1 / I \* q \* B^2 \* k / V;

a2 = A2 / I \* q \* B^2 \* k / V;

a3 = A3 / I \* q \* B^2 \* k^2;

a4 = A4 / I \* q \* B^2 \* k^2 / B;

%constantes por rÃ¡fagas

b1 = q \* B / V / m \* 2 \* Cl;

b2 = q \* B / V / m \* (dCl + Cd);

b3 = q \* B ^2 / V / I \* 2\* Cm;

b4 = q \* B ^2 / V / I \* dCm;

constB=[b1 b2 b3 b4];

C1 = h3-wh^2; % h

C2 = h4; % a

C3 = h1 - 2 \*ch\*wh; % dh

C4 = h2; % da

C5 = a3; % h

C6 = a4 - wa^2; % a

C7 = a1; % dh

C8 = a2-2\*ca\*wa; % da

constHA=[C1 C2 C3 C4 C5 C6 C7 C8];

t0 = 0; tf = t; x0 = [h0 a0 vh va]; %start/final time and initial value

df = 'flut8a45';

tic

[t45,x45] = ode45(df,[t0 tf],x0,[],constHA,constB,velU,velV,dh);

toc

maxH=abs(max(x45(:,1)));

maxA=abs(max(x45(:,2)));

sH=std(x45(:,1));

sA=std(x45(:,2));

kpH=maxH/sH;

kpA=maxA/sA;

graficar(t45,x45);

%Impresión de resultados

A=[t45,x45]';

archivo= fopen ('Resultado.txt', 'w');

fprintf(archivo,'%s\r\n',datetime);

fprintf(archivo,'Desplazamiento máximo horizontal: %6.4f m\r\n',maxH);

fprintf(archivo,'Rotación máxima : %6.4f rad\r\n',maxA);

fprintf(archivo,'Desviación estándar horizontal: %6.4f m\r\n',sH);

fprintf(archivo,'Desviacion estámdar rotacional: %6.4f rad\r\n',sA);

fprintf(archivo,'Factor pico horizontal: %6.4f\r\n',kpH);

fprintf(archivo,'Factor pico vertical: %6.4f\r\n\r\n',kpA);

fprintf(archivo,'%12s %12s %12s %12s %12s \r\n','T [s]', 'Desp Hor [m]', 'Rot [rad]', 'Vel Hor [m/s]', 'Vel Rot [rad/s]');

fprintf(archivo,'%12.4f %12.4f %12.4f %12.4f %12.4f \r\n',A);

fclose(archivo);

function mDatos=importar(numDatos)

[archivo,direccion] = uigetfile({'\*.txt'},'File Selector');

dirCom = strcat(direccion,archivo);

archivoID = fopen(dirCom,'r');

datos = textscan(archivoID, '%s','Delimiter', '\n');

matriz = datos{1};

fclose(archivoID);

varConst = zeros(1,numDatos);

revConst = false;

numConst = 0;

for i = 1 : length(matriz)

if revConst == false

constante = strsplit(matriz{i},'=');

if length (constante)== 2

num = str2double(strsplit(constante{2}));

numConst = numConst + 1;

varConst(numConst) = num(2);

if numConst == numDatos, revConst=true; end

end

end

end

mDatos=varConst;

end

function [u,v]= importarVel()

[archivo,direccion] = uigetfile({'\*.txt'},'File Selector');

dirCom = strcat(direccion,archivo);

vel = dlmread(dirCom,'',1,0);

u = vel(:,1);

v = vel(:,2);

end

function graficar(t,x)

subplot(2,2,1)

plot(t,x(:,1),'b','LineWidth',1.25)

xlabel('$t$[s]','Interpreter','latex','Fontsize',11)

ylabel('$h$[m]','Interpreter','latex','Fontsize',11)

grid on

subplot(2,2,2)

plot(t,x(:,2),'b','LineWidth',1.25)

xlabel('$t$[s]','Interpreter','latex','Fontsize',11)

ylabel('$\alpha$[rad]','Interpreter','latex','Fontsize',11)

grid on

subplot(2,2,3)

plot(t,x(:,3),'b','LineWidth',1.25)

xlabel('$t$[s]','Interpreter','latex','Fontsize',11)

ylabel('$V\_h$[m/s]','Interpreter','latex','Fontsize',11)

grid on

subplot(2,2,4)

plot(t,x(:,4),'b','LineWidth',1.25)

xlabel('$t$[s]','Interpreter','latex','Fontsize',11)

ylabel('$V\_{\alpha}$[rad/s]','Interpreter','latex','Fontsize',11)

grid on

end

Ejemplo de archivo de entrada de datos del puente

Archivo de entrada de datos para obtención de la respuesta ante aleteo con 8 derivadas aerodinámicas

Propiedades dinámicas de la estructura

rho = 1.25 [kg/m] Densidad del viento

m = 5308.6815 [kg/m] Masa modal a flexión.

I0 = 50340.37 [kg-m2/m] Momento másico de inercia.

B = 13.67 [m] Ancho del puente.

wh = 1.33 [rad/s] Frecuencia angular dirección vertical.

wt = 13.62 [rad/s] Frecuencia angular a torsión.

wa = 1.66 [rad/s] Frecuencia del aleteo (o frecuencia en la dirección del flujo).

ch = 0.030 Amortiguamiento crítico dirección Z.

Cm = 0.030 Amortiguamiento crítico a torsión.

Derivadas aerodinámicas

H1\* = -1.0000

H2\* = -0.4000

H3\* = -0.8000

H4\* = 0.25

A1\* = 0.3000

A2\* = -0.1500

A3\* = 0.2000

A4\* = 0.007

Coeficientes aerodinámicos

Cd = 2.0000 Coeficiente de arrastre

Cl = 0.6500 Coeficiente de levante

Cm = 0.2100 Coeficiente de momento

dCl = 6.2070 Derivada del coeficiente de levante

dCm = 0.2387 Derivada del coeficiente de momento

V = 51.51 Velocidad media

Datos para la simulación

Valores iniciales

h = 0.0 [m] Desplazamiento inicial

a = 0.0 [rad] Rotación inicial

dh = 0.0 [m/s] Velocidad inicial

da = 0.0 [rad/s] Velocidad rotacional inicial

Simulación

t = 600 [s] Tiempo total de integración

dh = 120 [s] Intervalos de tiempo de la simulación del viento turbulento (este dato indica a que intervalo se hizo la simulación de velocidad u y w)

Ejemplo de archivo de entrada de las velocidades

u w

2.3506 -6.1623

-34.5543 -1.7441

12.3656 6.0699

2.5538 7.0565

-2.6009 1.3733

3.0441 -5.9770

Para que sea consistente el archivo de velocidades con el programa «fluterOde45.m» es necesario que el archivo de entrada dh sea el mismo que se uso para obtener la simulación de viento con el programa «SimulacionesWAWS.m» o «SimulacionesAR.m»

## C.8 Funciones auxiliares para usar el programa «flutterOde45.m»

## Para usar el programa «flutterOde45.m» solo se requiere de un archivo auxilar, el cual representa la ecuación diferencial de aleteo.

## C.8.1 Ecuación diferencial expresada en función de aleteo. «flut8a45.m»

% flute8a45.m

function dx=flut8a45(t,x,C0,C,B,U,V,dt)

% Ecuación de aleteo acoplada para resolver por medio de ode45

if not(mod(t,dt)==0)

n0 = floor(t/dt)+1;

if n0==length(U); n0=length(U)-1; end

n1 = n0+1;

t0 = dt\*(n0-1);

t1 = t0+dt;

u = [U(n0) U(n1)];

v = [V(n0) V(n1)];

T = [t0 t1];

F1 = griddedInterpolant(T(:),u(:),'linear');

F2 = griddedInterpolant(T(:),v(:),'linear');

U = F1(t);

V = F2(t);

else

n = floor(t/dt)+1;

if n > length(U) ; n=length(U); end

U = U(n);

V = V(n);

end

dx(1)=x(3);

dx(2)=x(4);

dx(3)=C(1)\*x(1)+C(2)\*x(2)+C(3)\*x(3)+C(4)\*x(4)+B(1)\*U + B(2)\*V;

dx(4)=C(5)\*x(1)+C(6)\*x(2)+C(7)\*x(3)+C(8)\*x(4)+B(3)\*U + B(4)\*V;

dx=dx(:);

## Ejemplo de aplicación

% aplicación de flute8a45.m

t = 1; % tiempo deseado en s

% variables en función del tiempo

h1= 0.1; %desplazamiento vertical en m

dh=0.2; % derivada del desplazamiento vertical en m/s

a1=0.3; % rotación en rad

a4=0.4; % derivada de la rotación en rad/s

x=[h1 a1 dh a2];

% Constantes de la ecuación diferencial

C1 = 1.75; C5=0;

C2 = -0.66; C6=-185.27;

C3 = -0.12; C7=0.02;

C4 = -0.01; C8=-0.83;

C=[C1 C2 C3 C4 C5 C6 C7 C8];

%Constantes de la ráfaga

b1=0.11; b2=0.68; b3=0.05; b4=0.03;

B=[B1 B2 B3 B4];

%Velocidad del viento en las dos direcciones

u=[2.51 2.52];

w=[2.69 2.70];

%intervalo de tiempo entre las simulaciones en s

dh = 1;

%Valor extra que se integró para emplear ode45.m

C0=0;

dx=flut8a45(t,x,C0,C,B,U,W,dh);

## C.9 Programa para obtener las simulaciones en el dominio del tiempo con el método WAWS «SimulacionesWAWS.m».

Para que el programa funcione se debe ingresar los datos como se muestra en la Figura C.4. Al final crea una archivo «vel.txt» que contiene las velocidades en el dominio del tiempo en las direcciones u y w.

|  |
| --- |
|  |
| Figura C.4 Archivo de entrada para obtención de las simulaciones en el dominio del tiempo |

%SimulacionesWAWS.m

clear

close

proVie = 3;

proSim = 5;

proLog = 3;

propiedades = proVie + proSim + proLog;

datos = importar(propiedades);

v = datos(1);

h = datos(2);

z0 = datos(3);

t = datos(4);

dt = datos(5);

separacion = datos(6);

simulaciones = datos(7);

cn = datos(8);

graficar = datos(9);

exportar = datos(10);

nodo = datos(11);

%Revisa si hay coherencia

if cn==true

cu=9; cw=6;

else

cu=0; cw=0;

end

[sNu,sNw,w] = densidadKaimal(h,v,z0);

intTiempo = [0 t];

N = t/dt;

dw = (w(length(w)) - w(1)) / length(w);

u = WindSimMult(sNu,simulaciones,separacion,v,dw,intTiempo,N,cu);

w = WindSimMult(sNw,simulaciones,separacion,v,dw,intTiempo,N,cw);

T = linspace(0,t,N);

if graficar; crearGrafica(T,u(nodo,:),w(nodo,:));end

if exportar; crearTxt(u(nodo,:),w(nodo,:));end

function [sNu,sNw,w]=densidadKaimal(z,V,z0)

%Obtiene la función de densidad espectral en todo el dominio w

U = 0.4\*V/(log(z/z0));

%Definicion de variables

w = linspace(0,10,2000);

sNu = U^2\*107\*(z/V)./(1+33\*w/(2\*pi)\*z/V).^(5/3);

sNw = U^2\*2\*(z/V)./(1+5.3\*w/(2\*pi)\*z/V).^(5/3);

end

function [x]=WindSimMult(Sw,numSim,separacion,velocidad,Dw,tiempo,Ndatos,coherencia)

numFreq = 2000;

dx = separacion;

t0 = tiempo(1);

tf = tiempo(2);

t = linspace(t0,tf,Ndatos);

w0 = 0;

w = w0;

x = zeros(numSim,length(t));

fi = 2 \* pi \* rand(numSim,numFreq);

if not(coherencia==0)

% Con coherencia

for ii = 1:numFreq

cx = coherencia;

Co = exp( -( cx \*w\* dx) / (2 \* pi \* velocidad));

%Cholesky decomposition

G = choleskySimplify(numSim,Co);

a = sqrt(2 \* Sw(ii) \* Dw);

for jj=1:numSim

for k=1:jj

x(jj,:) = x(jj,:) + G(jj,k) .\* a .\* cos( w .\* t + fi(k,ii));

end

end

w = w + Dw;

end

else

%Sin coherencia

for ii=1:numSim

for jj = 1: numFreq

x(ii,:) = x(ii,:) + sqrt(2\* Sw(jj)\* Dw ) .\* cos ( w.\*t + rand (1) \*2\* pi);

w = w + Dw;

end

end

end

end

function [G]=choleskySimplify(matDimM,Coh)

%Obtención de C

C = Coh;

G = zeros(matDimM);

for j=1:matDimM

for m = 1:j

if m == 1 && m<=j

G(j,m) = C ^ (abs(j-m));

else

G(j,m) = C ^ (abs(j-m)) \* sqrt(1-C ^ 2);

end

end

end

end

function mDatos=importar(numDatos)

[archivo,direccion] = uigetfile({'\*.txt'},'File Selector');

dirCom = strcat(direccion,archivo);

archivoID = fopen(dirCom,'r');

datos = textscan(archivoID, '%s','Delimiter', '\n');

matriz = datos{1};

fclose(archivoID);

varConst = zeros(1,numDatos);

revConst = false;

revMat = false;

numConst = 0;

for i = 1 : length(matriz)

if revConst == false

constante = strsplit(matriz{i},'=');

if length (constante)== 2

num = str2double(strsplit(constante{2}));

numConst = numConst + 1;

varConst(numConst) = num(2);

if numConst == numDatos, revConst=true; end

end

end

end

mDatos=varConst;

end

function crearTxt(u,w)

fileID = fopen('vel.txt','w');

fprintf(fileID,'%10s %10s\r\n','u','w');

fprintf(fileID,'%10.4f %6.4f\r\n',[u(:) w(:)]);

fclose(fileID);

end

function crearGrafica(T,u,w)

subplot(2,1,1)

plot(T,u,'b')

xlabel('T [s]')

ylabel('V [m/s]')

title('Velocidad fluctuante u');

subplot(2,1,2)

plot(T,w,'b')

xlabel('T [s]')

ylabel('V [m/s]')

title('Velocidad fluctuante w');

end

Ejemplo de archivo de entrada de datos para obtener la simulación WAWS en la dirección u y w.

﻿Archivo de entrada de datos para obtención de simulaciones

Propiedades dinámicas de la estructura

v = 50 [m/s] Velocidad media del viento.

h = 40 [m] Altura deseada

z0 = 0.05 [m] Rugosidad del terreno

Datos para la simulación

t = 600 [s] Tiempo total de integración

dh = 120 [s] Intervalos de tiempo

dx = 1 [m] Separación de tramos

ns = 5 Número de simulaciones

coherencia = 1 Revisa si hay coherencia {1 : con coherencia / 0 : Sin coherencia}

graficar = 1 Revisa si hay que graficar {1 : graficar / 0 no graficar}

exportar = 1 Revisa si se crea un .txt {1 : exportar / 0 no exportar}

nodo = 4 Nodo que se desea exportar o graficar

En este ejemplo se aplico el método WAWS para 5 elementos considerando coherencia, con un intervalo de tiempo de 120 s por lo que se obtienen 6 intervalos de tiempo (0 s, 120 s, 240 s, 360 s, 480 s y 600 s). El archivo que crea sirve para usar el programa «flutterOder45.m»

## C.10 Programa para obtener las simulaciones en el dominio del tiempo con el método AR «SimulacionesAR.m».

Para que el programa funcione se debe ingresar los datos como se muestra en la Figura C.4. Al final crea una archivo «vel.txt» que contiene las velocidades en el dominio del tiempo en las direcciones u y w. Proceso similar que se usó con el programa «SimulacionesWAWS.m»

% SimulacionesAR.m

% Simulaciones por el método AR

% Importar datos

clear

close

proVie=3; proSim=4; proLog=4; propiedades=proVie+proSim+proLog;

dato= importar(propiedades);

V = dato(1);

z = dato(2);

z0 = dato(3);

t = dato(4);

ti = dato(5);

ds = dato(6);

N = dato(7);

cn = dato(8);

graficar = dato(9);

exportar = dato(10);

nodo = dato(11);

% Inicio del programa

dx = 0;

U = 0.4\*V/(log(z/z0));

P = 4;

x = cumsum(zeros(N,1)+ds);

n = floor(t/ti);

% Revivisón de coherencia

if cn

cnu=9; cnw=6;

else

cnu=0; cnw=0;

end

Au=zeros(P\*N);

Aw=zeros(P\*N);

Ru=zeros(N);

Rw=zeros(N);

for p=1:P

R=zeros(N);

for i=1:N

for j=i:N

dx = abs(x(i)-x(j));

sNu = @(f)U^2\*105\*(z/V)./(1+33\*f/(2\*pi)\*z/V).^(5/3).\*exp(-cnu\*dx\*f/(2\*pi)/V).\*cos(2\*pi\*f/(2\*pi)\*(p-1)\*ti);

sNw = @(f)U^2\*2\*(z/V)./(1+5.3\*f/(2\*pi)\*z/V).^(5/3).\*exp(-cnw\*dx\*f/(2\*pi)/V).\*cos(2\*pi\*f/(2\*pi)\*(p-1)\*ti);

Ru(i,j) = integral(sNu,0,50);

Ru(j,i) = Ru(i,j);

Rw(i,j) = integral(sNw,0,50);

Rw(j,i) = Rw(i,j);

end

end

for k=1:P-p+1

Au(((k-1)\*N+1):(k\*N),((k+p-1-1)\*N+1):((k+p-1)\*N))=Ru;

Aw(((k-1)\*N+1):(k\*N),((k+p-1-1)\*N+1):((k+p-1)\*N))=Rw;

end

end

% Matriz simétrica

for i=1:P\*N

for j=1:i

Au(i,j)=Au(j,i);

Aw(i,j)=Aw(j,i);

end

end

% Correlación sin P

Ru=zeros(N);

Rw=zeros(N);

for i=1:N

for j=i:N

dx = abs(x(i)-x(j));

sNu = @(f)U^2\*105\*(z/V)./(1+33\*f/(2\*pi)\*z/V).^(5/3).\*exp(-cnu\*dx/(2\*pi)\*f/V).\*cos(2\*pi\*f/(2\*pi)\*P\*ti);

sNw = @(f)U^2\*2\*(z/V)./(1+5.3\*f/(2\*pi)\*z/V).^(5/3).\*exp(-cnw\*dx/(2\*pi)\*f/V).\*cos(2\*pi\*f/(2\*pi)\*P\*ti);

Ru(i,j) = integral(sNu,0,50);

Ru(j,i) = Ru(i,j);

Rw(i,j) = integral(sNw,0,50);

Rw(j,i) = Rw(i,j);

end

end

Bu = Au(1:N,(N+1):(N\*P)); Bw = Aw(1:N,(N+1):(N\*P));

Bu = [Bu,Ru]; Bw = [Bw,Rw];

Bu = Bu'; Bw = Bw';

Xu = Au\Bu; Xw = Aw\Bw;

R0u = Au(1:N,1:N); R0w = Aw(1:N,1:N);

RNu = R0u; RNw = R0w;

for i=1:P

RNu = RNu-(Xu(((i-1)\*N+1):(i\*N),1:N))'\*Bu(((i-1)\*N+1):(i\*N),1:N);

RNw = RNw-(Xw(((i-1)\*N+1):(i\*N),1:N))'\*Bw(((i-1)\*N+1):(i\*N),1:N);

end

Lu=chol(RNu); Lw=chol(RNw);

Lu=Lu'; Lw=Lw';

au=zeros(N,n); aw=zeros(N,n);

for i=1:N

au(i,:)=normrnd(0,1,1,n);

aw(i,:)=normrnd(0,1,1,n);

end

Vu(1:N,1) = Lu\*au(:,1);

Vw(1:N,1) = Lw\*aw(:,1);

for i=2:p

Vu(1:N,i) = Lu\*au(:,i); Vw(1:N,i) = Lw\*aw(:,i);

for j=1:i-1

Vu(1:N,i) = (Xu(((j-1)\*N+1):(j\*N),:))'\*Vu(1:N,(i-j))+Vu(1:N,i);

Vw(1:N,i) = (Xw(((j-1)\*N+1):(j\*N),:))'\*Vw(1:N,(i-j))+Vw(1:N,i);

end

end

for i=(P+1):n

Vu(1:N,i) = Lu\*au(:,i); Vw(1:N,i) = Lw\*aw(:,i);

for j=1:P

Vu(1:N,i)=(Xu(((j-1)\*N+1):(j\*N),:))'\*Vu(1:N,(i-j))+Vu(1:N,i);

Vw(1:N,i)=(Xw(((j-1)\*N+1):(j\*N),:))'\*Vw(1:N,(i-j))+Vw(1:N,i);

end

end

t=(1:n)\*ti;

if graficar; crearGrafica(t,Vu(nodo,:),Vw(nodo,:));end

if exportar; crearTxt(Vu(nodo,:),Vw(nodo,:));end

% Función para importar datos

function mDatos=importar(numDatos)

[archivo,direccion] = uigetfile({'\*.txt'},'File Selector');

dirCom = strcat(direccion,archivo);

archivoID = fopen(dirCom,'r');

datos = textscan(archivoID, '%s','Delimiter', '\n');

matriz = datos{1};

fclose(archivoID);

varConst = zeros(1,numDatos);

revConst = false;

numConst = 0;

for i = 1 : length(matriz)

if revConst == false

constante = strsplit(matriz{i},'=');

if length (constante)== 2

num = str2double(strsplit(constante{2}));

numConst = numConst + 1;

varConst(numConst) = num(2);

if numConst == numDatos, revConst=true; end

end

end

end

mDatos=varConst;

end

% Función para exportar

function crearTxt(u,w)

fileID = fopen('vel.txt','w');

fprintf(fileID,'%10s %10s\r\n','u','w');

fprintf(fileID,'%10.4f %10.4f\r\n',[u(:) w(:)]);

fclose(fileID);

end

% Función para graficar

function crearGrafica(T,u,w)

subplot(2,1,1)

plot(T,u,'b')

xlabel('T [s]')

ylabel('V [m/s]')

title('Velocidad fluctuante u');

subplot(2,1,2)

plot(T,w,'b')

xlabel('T [s]')

ylabel('V [m/s]')

title('Velocidad fluctuante w');

end

Ejemplo de archivo de entrada de datos para obtener la simulación WAWS en la dirección u y w.

﻿Archivo de entrada de datos para obtención de simulaciones

Propiedades dinámicas de la estructura

v = 50 [m/s] Velocidad media del viento.

h = 40 [m] Altura deseada

z0 = 0.05 [m] Rugosidad del terreno

Datos para la simulación

t = 600 [s] Tiempo total de integración

dh = 120 [s] Intervalos de tiempo

dx = 1 [m] Separación de tramos

ns = 5 Número de simulaciones

coherencia = 1 Revisa si hay coherencia {1 : con coherencia / 0 : Sin coherencia}

graficar = 1 Revisa si hay que graficar {1 : graficar / 0 no graficar}

exportar = 1 Revisa si se crea un .txt {1 : exportar / 0 no exportar}

nodo = 4 Nodo que se desea exportar o graficar

En este ejemplo se puede ver que el archivo de entrada es el mismo que el que se usó para el programa «SimulacionesWAWS.m».

## C.11 Programa para obtener la respuesta debido al aleteo por el método de Hamming «flutter8Ham.m»

Este archivo funciona del mismo modo que el programa «flutterOde45.m» aplicando el siguiente código. Los archivos de entrada y de salida son los mismos. Para que funcione este programa requiere del programa auxiliar «flut8a.m» el cual sustituye a «flut8a45.m», además de los programas «ode\_Ham.m» y «ode\_RK4.m».

%flutter8Ham.m

clear

close all

proDim = 9;

derAer = 8;

proVie = 2;

proSim = 5;

coeAer = 5;

propiedades = proDim + derAer + proVie + proSim + coeAer;

datos = importar(propiedades);

rho = datos(1);

m = datos(2);

I = datos(3);

B = datos(4);

wh = datos(5);

wa = datos(6);

w = datos(7);

ch = datos(8);

ca = datos(9);

H1 = datos(10);

H2 = datos(11);

H3 = datos(12);

H4 = datos(13);

A1 = datos(14);

A2 = datos(15);

A3 = datos(16);

A4 = datos(17);

Cd = datos(18);

Cl = datos(19);

Cm = datos(20);

dCl = datos(21);

dCm = datos(22);

V = datos(23);

h0 = datos(24);

a0 = datos(25);

vh = datos(26);

va = datos(27);

t = datos(28);

dh = datos(29);

Vr = V/(w\*B);

[velU,velW] = importarVel();

k=B\*w/V;

q=1/2\*rho\*V^2;

h1 = H1 / m \* q \* B \* k / V;

h2 = H2 / m \* q \* B \* k / V;

h3 = H3 / m \* q \* B \* k^2 ;

h4 = H4 / m \* q \* B \* k^2 / B;

a1 = A1 / I \* q \* B^2 \* k / V;

a2 = A2 / I \* q \* B^2 \* k / V;

a3 = A3 / I \* q \* B^2 \* k^2;

a4 = A4 / I \* q \* B^2 \* k^2 / B;

%constantes por rÃ¡fagas

b1 = q \* B / V / m \* 2 \* Cl;

b2 = q \* B / V / m \* (dCl + Cd);

b3 = q \* B ^2 / V / I \* 2\* Cm;

b4 = q \* B ^2 / V / I \* dCm;

constB=[b1 b2 b3 b4];

kc = 1; %factor corrector de incio

C1 = h3-wh^2; % h

C2 = h4; % a

C3 = h1 - 2 \*ch\*wh; % dh

C4 = h2; % da

C5 = a3; % h

C6 = a4 - wa^2; % a

C7 = a1; % dh

C8 = a2-2\*ca\*wa; % da

constHA=[C1 C2 C3 C4 C5 C6 C7 C8];

t0 = 0; tf = t; x0 = [h0 a0 vh va];

N = 100000;

df = ‘flut8a’;

tic

[tH,xH] = ode\_Ham(df,[t0 tf],x0,N,kc,constHA,constB,velU,velW,dh);

toc

maxH=abs(max(xH(:,1)));

maxA=abs(max(xH(:,2)));

sH=std(xH(:,1));

sA=std(xH(:,2));

kpH=maxH/sH;

kpA=maxA/sA;

graficar(tH,xH);

%Impresión de resultados

A=[t45,x45]';

archivo= fopen ('Resultado.txt', 'w');

fprintf(archivo,'%s\r\n',datetime);

fprintf(archivo,'Desplazamiento máximo horizontal: %6.4f m\r\n',maxH);

fprintf(archivo,'Rotación máxima : %6.4f rad\r\n',maxA);

fprintf(archivo,'Desviación estándar horizontal: %6.4f m\r\n',sH);

fprintf(archivo,'Desviacion estámdar rotacional: %6.4f rad\r\n',sA);

fprintf(archivo,'Factor pico horizontal: %6.4f\r\n',kpH);

fprintf(archivo,'Factor pico vertical: %6.4f\r\n\r\n',kpA);

fprintf(archivo,'%12s %12s %12s %12s %12s \r\n','T [s]', 'Desp Hor [m]', 'Rot [rad]', 'Vel Hor [m/s]', 'Vel Rot [rad/s]');

fprintf(archivo,'%12.4f %12.4f %12.4f %12.4f %12.4f \r\n',A);

fclose(archivo);

function mDatos=importar(numDatos)

[archivo,direccion] = uigetfile({'\*.txt'},'File Selector');

dirCom = strcat(direccion,archivo);

archivoID = fopen(dirCom,'r');

datos = textscan(archivoID, '%s','Delimiter', '\n');

matriz = datos{1};

fclose(archivoID);

varConst = zeros(1,numDatos);

revConst = false;

numConst = 0;

for i = 1 : length(matriz)

if revConst == false

constante = strsplit(matriz{i},'=');

if length (constante)== 2

num = str2double(strsplit(constante{2}));

numConst = numConst + 1;

varConst(numConst) = num(2);

if numConst == numDatos, revConst=true; end

end

end

end

mDatos=varConst;

end

function [u,v]= importarVel()

[archivo,direccion] = uigetfile({'\*.txt'},'File Selector');

dirCom = strcat(direccion,archivo);

vel = dlmread(dirCom,'',1,0);

u = vel(:,1);

v = vel(:,2);

end

function graficar(t,x)

subplot(2,2,1)

plot(t,x(:,1),'b','LineWidth',1.25)

xlabel('$t$[s]','Interpreter','latex','Fontsize',11)

ylabel('$h$[m]','Interpreter','latex','Fontsize',11)

grid on

subplot(2,2,2)

plot(t,x(:,2),'b','LineWidth',1.25)

xlabel('$t$[s]','Interpreter','latex','Fontsize',11)

ylabel('$\alpha$[rad]','Interpreter','latex','Fontsize',11)

grid on

subplot(2,2,3)

plot(t,x(:,3),'b','LineWidth',1.25)

xlabel('$t$[s]','Interpreter','latex','Fontsize',11)

ylabel('$V\_h$[m/s]','Interpreter','latex','Fontsize',11)

grid on

subplot(2,2,4)

plot(t,x(:,4),'b','LineWidth',1.25)

xlabel('$t$[s]','Interpreter','latex','Fontsize',11)

ylabel('$V\_{\alpha}$[rad/s]','Interpreter','latex','Fontsize',11)

grid on

end

Ejemplo de archivo de entrada de datos del puente

Archivo de entrada de datos para obtención de la respuesta ante aleteo con 8 derivadas aerodinámicas

Propiedades dinámicas de la estructura

rho = 1.25 [kg/m] Densidad del viento

m = 5308.6815 [kg/m] Masa modal a flexión.

I0 = 50340.37 [kg-m2/m] Momento másico de inercia.

B = 13.67 [m] Ancho del puente.

wh = 1.33 [rad/s] Frecuencia angular dirección vertical.

wt = 13.62 [rad/s] Frecuencia angular a torsión.

wa = 1.66 [rad/s] Frecuencia del aleteo (o frecuencia en la dirección del flujo).

ch = 0.030 Amortiguamiento crítico dirección Z.

Cm = 0.030 Amortiguamiento crítico a torsión.

Derivadas aerodinámicas

H1\* = -1.0000

H2\* = -0.4000

H3\* = -0.8000

H4\* = 0.25

A1\* = 0.3000

A2\* = -0.1500

A3\* = 0.2000

A4\* = 0.007

Coeficientes aerodinámicos

Cd = 2.0000 Coeficiente de arrastre

Cl = 0.6500 Coeficiente de levante

Cm = 0.2100 Coeficiente de momento

dCl = 6.2070 Derivada del coeficiente de levante

dCm = 0.2387 Derivada del coeficiente de momento

V = 51.51 Velocidad media

Datos para la simulación

Valores iniciales

h = 0.0 [m] Desplazamiento inicial

a = 0.0 [rad] Rotación inicial

dh = 0.0 [m/s] Velocidad inicial

da = 0.0 [rad/s] Velocidad rotacional inicial

Simulación

t = 600 [s] Tiempo total de integración

dh = 120 [s] Intervalos de tiempo de la simulación del viento turbulento (este dato indica a que intervalo se hizo la simulación de velocidad u y w)

Ejemplo de archivo de entrada de las velocidades

u w

2.3506 -6.1623

-34.5543 -1.7441

12.3656 6.0699

2.5538 7.0565

-2.6009 1.3733

3.0441 -5.9770

## C.12 Función auxiliar para obtener la respuesta debido al aleteo. «flut8a.m»

function dx=flut8a(t,x,C,B,U,V,dt)

if not(mod(t,dt)==0)

n0 = floor(t/dt)+1;

if n0==length(U); n0=length(U)-1; end

n1 = n0+1;

t0 = dt\*(n0-1);

t1 = t0+dt;

u = [U(n0) U(n1)];

v = [V(n0) V(n1)];

T = [t0 t1];

F1 = griddedInterpolant(T(:),u(:),'linear');

F2 = griddedInterpolant(T(:),v(:),'linear');

U = F1(t);

V = F2(t);

else

n = floor(t/dt)+1;

if n > length(U) ; n=length(U); end

U = U(n);

V = V(n);

end

dx(1)=x(3);

dx(2)=x(4);

dx(3)=C(1)\*x(1)+C(2)\*x(2)+C(3)\*x(3)+C(4)\*x(4)+B(1)\*U + B(2)\*V;

dx(4)=C(5)\*x(1)+C(6)\*x(2)+C(7)\*x(3)+C(8)\*x(4)+B(3)\*U + B(4)\*V;

## Ejemplo de aplicación

% aplicación de flute8a.m

t = 1; % tiempo deseado en s

% variables en función del tiempo

h1= 0.1; %desplazamiento vertical en m

dh=0.2; % derivada del desplazamiento vertical en m/s

a1=0.3; % rotación en rad

a4=0.4; % derivada de la rotación en rad/s

x=[h1 a1 dh a2];

% Constantes de la ecuación diferencial

C1 = 1.75; C5=0;

C2 = -0.66; C6=-185.27;

C3 = -0.12; C7=0.02;

C4 = -0.01; C8=-0.83;

C=[C1 C2 C3 C4 C5 C6 C7 C8];

%Constantes de la ráfaga

b1=0.11; b2=0.68; b3=0.05; b4=0.03;

B=[B1 B2 B3 B4];

%Velocidad del viento en las dos direcciones

u=[2.51 2.52];

w=[2.69 2.70];

%intervalo de tiempo entre las simulaciones en s

dh = 1;

dx=flut8a(t,x,C,B,u,w,dh);

## C.13 Función para obtener la solución para una ecuación diferencia ordinaria por el método de Hamming «ode\_Ham» (Chung et al, 2005)

% ode\_Ham.m

function [t,y] = ode\_Ham(f,tspan,y0,N,KC,varargin)

% Hamming method to solve vector d.e. y'(t) = f(t,y(t))

% for tspan = [t0,tf] and with the initial value y0 and N time steps

% using the modifier based on the error estimate depending on KC = 1/0

if nargin < 5, KC = 1; end %with modifier by default

if nargin < 4 | N <= 0, N = 100; end %default maximum number of iterations

if nargin < 3, y0 = 0; end %default initial value

y0 = y0(:)'; %make it a row vector

h = (tspan(2)-tspan(1))/N; %step size

tspan0 = tspan(1)+[0 3]\*h;

[t,y] = ode\_RK4(f,tspan0,y0,3,varargin{:}); %Initialize by Runge-Kutta

t = [t(1:3)' t(4):h:tspan(2)]';

for k = 2:4

F(k - 1,:) = feval(f,t(k),y(k,:),varargin{:});

end

p = y(4,:);

c = y(4,:);

h34 = h/3\*4;

KC11 = KC\*112/121;

KC91 = KC\*9/121;

h312 = 3\*h\*[-1 2 1];

for k = 4:N

p1 = y(k - 3,:) + h34\*(2\*(F(1,:) + F(3,:)) - F(2,:)); %Eq.(6.4.9a)

m1 = p1 + KC11\*(c - p); %Eq.(6.4.9b)

c1 = (-y(k - 2,:) + 9\*y(k,:) +...

h312\*[F(2:3,:); feval(f,t(k + 1),m1,varargin{:})])/8; %Eq.(6.4.9c)

y(k+1,:) = c1 - KC91\*(c1 - p1); %Eq.(6.4.9d)

p = p1; c = c1; %update the predicted/corrected values

F = [F(2:3,:); feval(f,t(k + 1),y(k + 1,:),varargin{:})];

end

## C.14 Función para obtener la solución para una ecuación diferencia ordinaria por el método de Runge Kutta «ode\_RK4.m» (Chung et al, 2005)

% ode\_RK4.m

function [t,y] = ode\_RK4(f,tspan,y0,N,varargin)

%Runge-Kutta method to solve vector differential eqn y’(t) = f(t,y(t))

% for tspan = [t0,tf] and with the initial value y0 and N time steps

if nargin < 4 || N <= 0, N = 100; end

if nargin < 3, y0 = 0; end

y(1,:) = y0(:)'; %make it a row vector

h = (tspan(2) - tspan(1))/N; t = tspan(1)+[0:N]'\*h;

for k = 1:N

f1 = h\*feval(f,t(k),y(k,:),varargin{:});

f1 = f1(:)'; %(6.3.2a)

f2 = h\*feval(f,t(k) + h/2,y(k,:) + f1/2,varargin{:}); f2 = f2(:)';%(6.3.2b)

f3 = h\*feval(f,t(k) + h/2,y(k,:) + f2/2,varargin{:}); f3 = f3(:)';%(6.3.2c)

f4 = h\*feval(f,t(k) + h,y(k,:) + f3,varargin{:}); f4 = f4(:)'; %(6.3.2d)

y(k + 1,:) = y(k,:) + (f1 + 2\*(f2 + f3) + f4)/6; %Eq.(6.3.1)

end

Ejemplo de aplicación para el programa «ode\_Ham.m» y «ode\_RK4.m».

Para ello se requiere crear un programa que contenga la ecuación diferencial para ello se empleara. En este ejemplo resuelve la ecuación

|  |  |  |
| --- | --- | --- |
|  |  |  |

Sin embargo, para resolverla es necesaria transformarla en ecuaciones de primer grado

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

Por lo que el código del programa que contiene la ecuación diferencial es.

%df651.m

dx = zeros(size(x));

dx(1) = x(2); dx(2) = -4\*x(1);

Este archivo se guarda con el nombre «df651.m». Por lo que para obtener la solución de esta ecuación queda:

%ejemplo.m

t0 = 0; tf = 1; x0 = [2 0];

N = 5;

df = 'df651';

[tH2,xH2] = ode\_RK4(df,[t0 tf],x0,N);

[tH2,xH2] = ode\_Ham(df,[t0 tf],x0,N);

En este ejemplo se emplea un tiempo inicial de 0 y uno final de 1 con 5 valores, las condiciones iniciales de desplazamiento es 2 y de velocidad es 0.

**C.15 Función de las derivadas aerodinámicas del puente San Cristóbal**

% derAerSC.m

% Función para obtener los las derivadas aerodinámicas

function [H,A]=derAerSC2(v)

H(1) = -0.003125\*v.^6+ 0.086458\*v.^5-0.95521\*v.^4+5.3344\*v.^3-15.642\*v.^2+21.929\*v-12.25;

H(2) = -0.00040793\*v.^4+0.0054312\*v.^3+0.0081643\*v.^2-0.18343\*v-0.085;

H(3) = -0.025893\*v.^2+-0.34875\*v-0.045;

H(4) = 0.00021759\*v.^6-0.0069284\*v.^5+0.094617\*v.^4-0.71349\*v.^3+3.0093\*v.^2-6.5022\*v+ 5.6111;

A(1) = -0.00067308\*v.^5+0.017599\*v.^4-0.16925\*v.^3+ 0.74685\*v.^2-1.424\*v+1.225;

A(2) = -0.035394\*v-0.073333;

A(3) = -0.0074074\*v.^3+ 0.084921\*v.^2+ -0.13624\*v+0.16667;

A(4) = 0.00060897\*v.^5+-0.015268\*v.^4+0.14468\*v.^3+ -0.62721\*v.^2+ 1.2872\*v+ -0.89167;

end

**C.16 Archivos de entrada para el puente San Cristóbal**

Debido que los archivos son muy extensos, se subieron a una plataforma para poder incluir todos.