Contents

1 ./edosys.m

```
1  #!/usr/bin/octave-cli
2
3  ## Instituto Tecnológico de Costa Rica
4  ## Área Académica de Ingeniería en Computadores
5  ## CE-3102 Análisis Numérico para Ingeniería
6  ## Prof. Pablo Alvarado
7  ## II Semestre 2018
8  ## Examen Final
9
10 ## PROBLEMA 1
11
12 ## NOMBRE: Alexis Gavriel Gómez
13 ## CARNE: 2016085662
14
15 1;
16
17 global m = 0.1;    ## Masa de la partícula
18 global b = 0.05;   ## Coeficiente de atenuación
19 global k = 1;      ## Constante de Hook
20
21 ## ###################
22 ## ## Problema 1.1 ##
23 ## ###################
24 ## Fuerza aplicada en la partícula
25 global F=@(x,v,t) -b*v - k*x ;
26
27
28
29 ## Resuelva el sistema atenuado masa resorte usando Euler
30 ## tn el último instante de tiempo
31 ## Dt paso temporal
32 function [t,x]=eulersys(tn,Dt)
33
34   global m b k F;
35
36   t=0:Dt:tn; ## Intervalo de simulación
37
38   ## Pre-reserve la memoria utilizada.
39   x=zeros(size(t));
40   v=zeros(size(t));
41
42   ## Condiciones iniciales
43   x(1)=-1;
44
45   ## ###################
46   ## ## Problema 1.2 ##
47   ## ###################
48
49   ## Resuelva el sistema de ecuaciones con Euler
50
51   ## Da un paso en cada itearación
52   for i = 2:size(t)(2)
53     x(i) = x(i-1) + v(i-1)*Dt;
54     v(i) = v(i-1) + F(x(i-1),v(i-1), tn )/m*Dt;
55   endfor
56
57
58
59 endfunction
60
61 figure(1,"name","Euler");
62 hold off;
63 [t,x]=eulersys(10,0.05);
64 plot(t,x,"r;\\Delta␣t=0.05;");
65
```

3/3

```
66  hold on;
67  [t,x]=eulersys(10,0.01);
68  plot(t,x,"m;\\Delta␣t=0.01;");
69
70  [t,x]=eulersys(10,0.001);
71  plot(t,x,"b;\\Delta␣t=0.001;");
72
73  xlabel("t");
74  ylabel("x(t)");
75  axis([0,10,-2,2]);
76  grid on;
77
78  ## Resuelva el sistema de ecuaciones con Runge-Kutta 4to orden
79  ## tn Último instante de tiempo
80  ## Dt Paso temporal (delta t)
81  function [t,x] = rksys(tn,Dt)
82    global m b k F;
83
84    t=0:Dt:tn; ## Intervalo de simulación
85
86    ## Pre-reserve la memoria utilizada.
87    x=zeros(size(t));
88    v=zeros(size(t));
89
90    ## Condiciones iniciales
91    x(1)=-1;
92
93    ## ###################
94    ## ## Problema 1.4 ##
95    ## ###################
96
97
98    ## Velocidad
99    g=@(x,v,t) v;
100
101   ## Aceleración
102   h=@(x,v,t) F(x,v,t)/m ;
103
104   ## Calcula los coeficientes del RK4 y al final hace un paso en los vectores: x , v
105   ## Da un paso en cada itearación
106   for i = 2:size(t)(2)
107     l1 = h(x(i-1), v(i-1), tn) * Dt;
108     k1 = g(x(i-1), v(i-1), tn) * Dt;
109
110     l2 = h(x(i-1) + k1/2, v(i-1) + l1/2, tn) * Dt;
111     k2 = g(x(i-1) + k1/2, v(i-1) + l1/2, tn) * Dt;
112
113     l3 = h(x(i-1) + k2/2, v(i-1) + l2/2, tn) * Dt;
114     k3 = g(x(i-1) + k2/2, v(i-1) + l2/2, tn) * Dt;
115
116     l4 = h(x(i-1) + k3, v(i-1) + l3, tn) * Dt;
117     k4 = g(x(i-1) + k3, v(i-1) + l3, tn) * Dt;
118
119     v(i) = v(i-1) + (l1 + 2*l2 + 2*l3 + l4)/6;
120     x(i) = x(i-1) + (k1 + 2*k2 + 2*k3 + k4)/6;
121   endfor
122
123
124 endfunction
125
126 figure(2,"name","RK");
127 hold off;
128 [t,x]=rksys(10,0.05);
129 plot(t,x,"r;\\Delta␣t=0.05;");
130
131 hold on;
132 [t,x]=rksys(10,0.01);
133 plot(t,x,"m;\\Delta␣t=0.01;");
```

1.3 2/2

10/10

```
134
135 [t,x]=rksys(10,0.001);
136 plot(t,x,"b;\\Delta␣t=0.001;");
137
138 xlabel("t");
139 ylabel("x(t)");
140 axis([0,10,-2,2]);
141 grid on;
```

2 ./posfromdist.m

¿No es sospechoso que todo le quede en el piso?

```
#!/usr/bin/octave-cli

## Instituto Tecnológico de Costa Rica
## Área Académica de Ingeniería en Computadores
## CE-3102 Análisis Numérico para Ingeniería
## Prof. Pablo Alvarado
## II Semestre 2018
## Examen Final

## PROBLEMA 3

## NOMBRE: Alexis Gavriel Gómez
## CARNE: 2016085662

3;
## Cargue los datos de distancias
##
## Cada columna de D tiene las distancias a un sensor fijo en el
## espacio.
D=load("-ascii","dists.dat");

## Las posiciones de los emisores están dadas por las _columnas_
## de la matriz S
E=load("-ascii","emisors.dat");

## Función calcula posición a partir de distancias a emisores.
##
## dists:     Distancias del punto a cada emisor como vector fila.
## emisorPos: Posiciones de los emisores, cada emisor en una columna.
## option:    Si solo hay 3 emisores, cuál de las soluciones retorna:
##            1: use + en la cuadrática
##            2: use - en la cuadrática
##            3: decida cuál signo usar dependiendo de la velocidad y
##               aceleración
##
## La posición i de dists contiene la distancia al emisor en la
## columna i.
##
function p=calcPosition(dists,emisorPos,option=1)

  ## Número de emisores usados
  dim = min(columns(emisorPos),length(dists));

  ## ###################
  ## ## Problema 3.1 ##
  ## ###################
  ## Construya la matriz M
  M  = zeros(dim,4);

  ## llena la primera columna con 1
  for j = 1:rows(M)-1
        M(j,1) = 1;
  endfor

  for i = 1:rows(M)-1
    for j = 2:columns(M)-1
      M(i,j)=abs(-2*emisorPos(j-1,i));
    endfor
  endfor

  ## ###################
  ## ## Problema 3.2 ##
  ## ###################
  ## Construya el vector b

```

¡ evite ciclos en octave !

0.1

```
66      b = zeros(dim,1); ## <<< Ponga su solución aquí
67    for k = 1:rows(b)
68      b(k,1) = dists(k)^2 - norm([emisorPos(1,k) emisorPos(2,k) emisorPos(3,k)])^2;
69    endfor
70
71
72    ## ###################
73    ## ## Problema 3.3 ##
74    ## ###################
75
76    ## Calcule la matriz seudo-inversa utilizando SVD
77    iM=pinv(M); ## <<< Ponga su solución aquí (se busca calcular esto con SVD)
78    [U,W,V] = svd(M);
79
80    Urez = resize(U,rows(M),columns(M));
81
82    Wrez = resize(W,columns(M));
83
84    Vrez = resize(V,columns(M));
85
86    pinv(Wrez);
87
88
89    iM = Vrez * pinv(Wrez) * Urez'; ##'
90
91    ## Verifique que iM y pinv(M) son lo mismo
92    if (norm(iM-pinv(M),"fro") > 1e-6)
93      error("Matriz inversa calculada con SVD incorrecta");
94    endif
95
96    ## ###################
97    ## ## Problema 3.4 ##
98    ## ###################
99
100   ## Calcule la solución particular
101   hatp= iM * b;
102
103
104   ## El caso de 3 dimensiones tiene dos posibles soluciones:
105   if (dim==3)
106     ## Con 3 emisores, calcule las dos posibles posiciones
107
108     ## ###################
109     ## ## Problema 3.5 ##
110     ## ###################
111
112     ## >>> Ponga su solución aquí <<<
113     ## obtiene la columna 4 del la matriz
114     col4 = Vrez(:,4);
115
116     vector1 = (col4(2,1)^2) + (col4(3,1)^2) + (col4(4,1)^2);
117     vector2 = ((2* hatp(2,1) * col4(2,1)) + (2*hatp(3,1) * col4(3,1)) + (2* hatp(4,1) *
          col4(4,1))) - col4(1,1);
118     vector3 = ((hatp(2,1)^2) + (hatp(3,1)^2) + (hatp(4,1)^2)) - hatp(1,1);
119
120
121     q = 2*vector1;
122     dis = 4* vector1 * vector3;
123
124     if(option == 1)
125       l = (-vector2 + sqrt((vector2^2) - (dis))) / q;
126     else
127       l = (-vector2 - sqrt((vector2^2) - (dis))) / q;
128     endif
129
130     p = (hatp + l )(1:3);
131
132   else
```

```
133       ## Caso general de más de tres dimensiones
134
135       ## ##################
136       ## ## Problema 3.6 ##
137       ## ##################
138
139       p = hatp(2:4);
140     endif
141
142   endfunction
143
144   ## Calcule la trayectoria de posiciones para una matriz de
145   ## distancias que contiene en cada fila el vector de distancias
146   ## a cada emisor.
147   ##
148   ## Sea N el número de datos y S el número de emisores
149   ## dists:     matriz de distancias de tamaño N x S
150   ## emisorPos: matrix de tamaño 3xS
151   ## option:    si S==3, entonces cuál de las dos soluciones devolver
152   ##            option=1 implica usar + en cuadrática, y otra cosa usa -
153   function p=calcPositions(dists,emisorPos,option=1)
154     n=min(columns(emisorPos),columns(dists));
155     k=rows(dists);
156
157     ## Reserve memoria para todos los puntos en la trayectoria
158     p=zeros(k,3);
159
160     if (option==3)
161
162       predPos=[2;0;1]; ## Predicción inicial de posición
163
164       predVel=[0;0;0];
165       predAcc=[0;0;0];
166       ## Para cada punto en la secuencia
167       for i=1:k
168
169         ## Calcule las dos posibilidades
170         sol1=calcPosition(dists(i,1:n),emisorPos(:,1:n),1)';
171         sol2=calcPosition(dists(i,1:n),emisorPos(:,1:n),2)';
172
173         ## Verifique cuál solución está más cerca de la predicción
174         if (norm(predPos-sol1) < norm(predPos-sol2))
175           p(i,:) = sol1;
176         else
177           p(i,:) = sol2;
178         endif
179
180         ## ##################
181         ## ## Problema 3.7 ##
182         ## ##################
183         ## >>> Complete su solución aquí <<<
184
185         ## Actualice la predicción
186
187
188       endfor
189     else
190       ## Para todos los puntos en la trayectoria
191       for i=1:k
192         ## Calcule la posición del punto, dadas las distancias a los emisores
193         p(i,:)=calcPosition(dists(i,1:n),emisorPos(:,1:n),option)';
194       endfor
195     endif
196
197   endfunction
198
199   ## Función calcula distancia de un punto a todos los emisores
200   ## pos:       Posición del objeto (en las filas)
```

```
## emisorPos: Posiciones de los emisores, en las columnas.
function d=calcDistances(pos,emisorPos)
  if (columns(pos)!=3)
    error("Position must be a 3D vector");
  endif

  ## Calcule la distancia a cada emisor, dada la posición "pos" del
  ## objeto y las posiciones de los emisores.

  ## ##################
  ## ## Problema 3.8 ##
  ## ##################

  ## >>> Ponga su solución aquí <<<
  #d=zeros(rows(pos),1);
  d = zeros(rows(pos),5);

  for i = 1:rows(pos)
    for j = 1:columns(pos)
      d(i,j) = sqrt(((pos(i,1)-emisorPos(1,j))^2) + ((pos(i,2)-emisorPos(2,j))^2) + ((pos
          (i,3)-emisorPos(3,j))^2));
    endfor
  endfor
endfunction


## ##################################
## ## Pruebas con varios emisores ##
## ##################################

## Pruebe todo el esquema usando el número de emisores dado
## n:    número de emisores que debe ser 3, 4 o 5
## D:    datos de distancia
## E:    posición de emisores
## fig1: número de primera figura a utilizar
function testCase(n,D,E,fig1)

  assert(n>2 && n<=columns(D));

  printf("Probando el caso de %i emisores\n",n);

  ## Prueba unitaria: calcule la posición del primer dato
  p=calcPosition(D(1,1:n),E)

  ## Calcule las posiciones a partir de las distancias
  p=calcPositions(D(:,1:n),E);

  ## Distancias a posiciones estimadas
  ed=calcDistances(p,E);

  ## Errores
  err=(ed-D);

  ## Promedio de errores
  error_average = sum(err)/rows(err)

  ## Muestre los errores de las distancias a los primeros tres emisores
  figure(fig1,"name",cstrcat("Errores con ",num2str(n)," emisores"));
  hold off;
  plot(err(:,1),"r");
  hold on;
  plot(err(:,2),"b");
  plot(err(:,3),"k");

  ## Grafique las posiciones encontradas
  figure(fig1+1,"name",...
         cstrcat("Trayectoria estimada con ",num2str(n)," emisores"));
  hold off;
```

2.2

```
268
269     ## Grafique la trayectoria en azul
270     plot3(p(:,1),p(:,2),p(:,3),'b');
271     xlabel("x");
272     ylabel("y");
273     zlabel("z");
274
275     hold on;
276
277     if (n==3)
278       ## Si n==3 hay otros posibles valores para la trayectoria
279
280       ## Use - en la cuadrática
281       p=calcPositions(D(:,1:n),E,2);
282       plot3(p(:,1),p(:,2),p(:,3),'r');
283
284       ## Use la predicción para auto-seleccionar cuál solución usar y
285       ## píntela con cuadrados magenta
286       p=calcPositions(D(:,1:n),E,3);
287       plot3(p(:,1),p(:,2),p(:,3),'ms');
288
289       ## A la fuerza use n=5 para ver la trayectoria verdadera
290       ## y píntela usando circulos verdes
291       p=calcPositions(D(:,1:5),E);
292       plot3(p(:,1),p(:,2),p(:,3),'go');
293     endif
294
295     ## Grafique los emisores en postes
296     plot3([E(1,1:n);E(1,1:n)],...
297           [E(2,1:n);E(2,1:n)],...
298           [E(3,1:n);zeros(1,n)],...
299           "linewidth",3);
300
301     plot3(E(1,1:n),E(2,1:n),E(3,1:n),'rx');
302   endfunction;
303
304   ## Fuerce probar con 3, 4 y 5 emisores
305   testCase(3,D,E,1); ## 3 emisores, en las figuras 1 y 2
306   testCase(4,D,E,3); ## 4 emisores, en las figuras 3 y 4
307   testCase(5,D,E,5); ## 5 emisores, en las figuras 5 y 6
```

No funciona

3 ./splines2d.m

```
1  #!/usr/bin/octave-cli
2
3  ## Instituto Tecnológico de Costa Rica
4  ## Área Académica de Ingeniería en Computadores
5  ## CE-3102 Análisis Numérico para Ingeniería
6  ## Prof. Pablo Alvarado
7  ## II Semestre 2018
8  ## Examen Final
9
10 ## PROBLEMA 2
11
12 ## NOMBRE: Alexis Gavriel Gómez
13 ## CARNE: 2016085662
14 2;
15
16
17 ## Construya algunos datos 2D para el problema
18 ## N: Número de datos.  Cada fila de la matriz tendrá un punto.
19 ##    La primera columna tendrá la coordenada x y la segunda columna
20 ##    la coordenada y.
21 function points = createData(N)
22   astep = 360/N;
23
24   angles = (0:astep:360-astep)';  ##'
25   anoise = rand(size(angles))*astep/4;
26   rnoise = rand(size(angles))*1.5;
27
28   radii  = 2+rnoise;
29   angles = deg2rad(angles + anoise);
30
31   points = [radii.*cos(angles) radii.*sin(angles)];
32 endfunction
33
34 ## Calcule las segundas derivadas
35 ## x: posiciones x de las muestras
36 ## f: valores de la función en cada x
37 ## retorne fpp con los valores de la segunda derivada en cada posición t
38 function fpp=findDerivs(t,f)
39   assert(size(f)==size(t));
40
41   N=length(t)-1; # Número de subintervalos
42
43   ## Arme el sistema de ecuaciones
44   M=eye(N+1,N+1);
45   fpp=zeros(N+1,1);
46   b  =zeros(N+1,1);
47
48   ## ##################
49   ## ## Problema 2.4 ##
50   ## ##################
51
52
53   ## Coloca las n-1 ecuaciones en la matriz
54   ## Coloca el valor en la diagonal y luego el valor en las dos posiciones a la derecha
55   for i=1:(N-1)
56     M(i,i)   = t(i+1) - t(i);
57     M(i,i+1) = 2*(t(i+2) - t(i));
58     M(i,i+2) = t(i+2) - t(i+1);
59     b(i) = 6*((f(i+2) - f(i+1))/(t(i+2) - t(i+1))) - 6*((f(i+1) - f(i))/(t(i+1)-t(i)));
60   endfor
61
62   ## Se agregan los puntos virtuales
63   x = [t(:); t(N+1)+1; t(N+1)+2];
64   y = [f(:); f(N+1)+1; f(N+1)+2];
65
```

? no tiene sentido.

f(1) f(2)?

```
66    ## Coloca la ecuación n centrada en el último dato
67    M(N,N) = x(N+1) - x(N);
68    M(N, N+1 ) = 2*(x(N+2)- x(N)  );
69    M(N, 1 ) = (   x(N+1) - x(N));
70    b(N) = 6*( y(1)-y(N+1) )/( x(N+2)-x(N+1) ) - 6*(y(N+1)-y(N) )/( x(N+1)-x(N) );
71
72    ## Coloca la ecuación n+1 centrada en el primer dato, el punto virtual, accesado por t(
          N+2)
73    M(N+1,N+1) = x(N+3) - x(N+2);
74    M(N+1,1) = 2*(x(N+3) - t(N+1));
75    M(N+1,2) = x(N+3) - x(N+2);
76    b(N+1) = 6*((y(2) - y(1))/(x(N+3) - x(N+2))) - 6*((y(1) - y(N+1))/(x(N+2)-x(N+1)));
77
78    ## Resuelva el sistema
79
80    fpp = M\b;
81
82  endfunction
83
84  ## Interpole los valores fi(xi) usando los puntos x y sus valores f(x)
85  ## t: valores de soporte conocidos
86  ## f: valores de la función en los x conocidos
87  ## ts: valores en donde debe encontrarse la función interpolada
88  ## retorna fs: valores de la función en los xs dados
89  function fs=interpole(t,f,ts)
90    assert(size(t)==size(f));
91
92    ts=ts(:); ## Asegúrese de que es un vector columna
93
94    ## Encuentre las segundas derivadas
95    fpp=findDerivs(t,f);
96
97    ## ###################
98    ## ## Problema 2.5 ##
99    ## ###################
100
101   ## >>> Ponga su solución aquí <<<
102   fs=zeros(size(ts));
103
104   ## Inicializa los vectores que guardan los 4
105   ##      coeficientes que se utlizan para cada intervalo
106   ## co : coeficiente
107   co1=zeros(size(ts));
108   co2=zeros(size(ts));
109   co3=zeros(size(ts));
110   co4=zeros(size(ts));
111
112   ## LLena los vectores de coeficientes
113   for i = 1:length(t)
114     ## Indica la posición del siguiente valor del vector
115     j = i+1;
116
117     ## Cuando se llega al final, el valor siguiente es el inicial
118     if i == length(t)
119       j = 1;
120     endif
121
122     ## Se calcula cada uno de los coeficientes
123     co1(j) = fpp(i) / (t(i)-t(j))/6;
124     co2(j) = fpp(j) / (t(j)-t(i))/6;
125     co3(j) = ((f(i))/(t(i)-t(j))) + (fpp(i)*(t(i)-t(j))/-6);
126     co4(j) = ((f(j))/(t(j)-t(i))) + (fpp(j)*(t(j)-t(i))/-6);
127   endfor
128
129   ## Itera sobre los datos para interpolarlos usando los coeficientes
130   for k = 1:length(ts)
131     ## Se busca el intervalo al que pertenece el punto a interpolar
132     i = lookup(t,ts(k));
```

```
133
134       ## Indica la posición del siguiente valor del vector
135       j = i+1;
136       ## Cuando se llega al final, el valor siguiente es el inicial
137       if i == length(t)
138         j = 1;
139       endif
140
141       ## Se divide la operación que calcula el valor
142       ##       interpolado en 4 operaciones para facilitar la lectura
143       part1 = co1(j)* ((ts(k)-t(j))^3);
144       part2 = co2(j)* ((ts(k)-t(i))^3);
145       part3 = co3(j)* (ts(k)-t(j));
146       part4 = co4(j)* (ts(k)-t(i));
147
148       fs(k) = part1 + part2 + part3 + part4;
149     endfor
150
151
152
153
154     ## Sugerencia: Puede serle muy útil el uso de la función 'lookup'
155     ##              para encontrar cuál subintervalo utilizar.
156
157
158   endfunction
159
160   ## Depuración
161   figure(2,"name","Interpolación simple cerrada (depuración)");
162   x=[0,1,2,3];
163   f=[1,2,1,0.5];
164
165   hold off;
166   plot(x,f,'rx-;original;',"linewidth",2);
167
168   step=0.1;
169   xs=0:step:4-step;
170
171   fs=interpole(x,f,xs);
172
173
174
175
176   hold on;
177   plot(xs,fs,'bo-;interpolado;');
178   grid on;
179   xlabel("t");
180   ylabel("f(t)");
181
182   ## El caso completo
183   N=10;
184   D = createData(N);
185   figure(1,"name","Interpolación 2D cerrada");
186   hold off;
187   plot(D(:,1),D(:,2),'rx-',"linewidth",2);
188
189   step=0.1;
190   t=0:step:N-step;
191   xs=interpole([0:N-1]',D(:,1),t); ##')
192   ys=interpole([0:N-1]',D(:,2),t); ##')
193
194   ## ###################
195   ## ## Problema 2.6 ##
196   ## ###################
197
198   ## >>> Ponga su solución aquí <<<
199
200   xs = [xs; D(1,1)];
```

6/7 (e.c).

```
ys = [ys; D(1,2)];


hold on;
plot(xs,ys,'bo-');
xlabel("x");
ylabel("y");
grid;
```