



**FACULTAD
DE INGENIERIA**

Universidad de Buenos Aires

75.12 - Análisis Numérico I
Trabajo Práctico N°1
Resolución de Sistemas de Ecuaciones Lineales

Nombre	Correo electrónico	Padrón
Gonzalo Ávila Alterach	gonzaloavilaalterach@gmail.com	94950
Nicolás Mariano Fernandez Lema	nicolasfernandezlema@gmail.com	93410

Fecha de entrega: 16 de octubre
2º cuatrimestre de 2013

Resumen

En este problema, el sistema a resolver se trata de $Ax = b$, siendo A una matriz cuadrada tridiagonal, de dimensiones $(n - 1)^2$.

El vector incógnita x a averiguar está formado por los elementos $(c_1, c_2, \dots, c_{n-1})$, ya que sabiendo su valor se pueden hallar el resto de los coeficientes de todos los polinomios. Debido a las ecuación 4 y la 7, se puede apreciar que la matriz A es tridiagonal, y sus elementos son los siguientes:

$$A_n = \begin{pmatrix} 2(h_0 + h_1) & h_1 & 0 & 0 & \cdots & 0 & 0 \\ h_1 & 2(h_1 + h_2) & h_2 & 0 & \cdots & 0 & 0 \\ 0 & h_2 & 2(h_2 + h_3) & h_3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & h_{n-1} & 2(h_{n-1} + h_n) \end{pmatrix}$$

Además, debido a que los datos de entrada utilizados la diferencia entre los θ_k consecutivos es constante, los h_k también lo son, y por lo tanto la matriz queda simétrica.

Los programas se desarrollaron para resolver matrices genéricas, sin tener en cuenta que los sistemas del problema a resolver son tridiagonales, por lo que es una resolución muy ineficiente: se está utilizando más memoria que la necesaria y también se están multiplicando muchas veces ceros por ceros.

Polinomios encontrados

Las coeficientes c_k encontrados, así como el resto de los coeficientes de los polinomios están en el apéndice A.

Radios espectrales

Utilizando la estimación dada, dividir el error relativo de una iteración respecto de la anterior, los radios espectrales son los siguientes:

Día 1

Iteración	ρ_J	ρ_{GS}
1	0.365217	0.262669
2	0.363875	0.280345
3	0.445632	0.291762
4	0.385874	0.288134
5	0.49069	0.263809
6	0.411277	0.317298
7	0.480461	
8	0.431252	

Día 2

Iteración	ρ_J	ρ_{GS}
1	0.336795	0.294195
2	0.31415	0.299836
3	0.482199	0.296196
4	0.397669	0.295461
5	0.475701	0.295959
6	0.442331	0.296806
7	0.462645	
8	0.464795	

Día 3

Iteración	ρ_J	ρ_{GS}
1	0.156788	0.250333
2	0.309514	0.255578
3	0.375905	0.247174
4	0.472542	0.245682
5	0.469129	0.302341
6	0.464291	0.283305
7	0.473524	

Día 4

Iteración	ρ_J	ρ_{GS}
1	0.237237	0.261134
2	0.430469	0.282262
3	0.387289	0.281608
4	0.486819	0.276875
5	0.432466	0.317498
6	0.485886	0.306247
7	0.461146	
8	0.476161	

Debido a que la matriz es tridiagonal, debería cumplirse la propiedad $\rho_J^2 = \rho_{GS}$. Sin embargo, se observa que el ρ_{GS} es aproximadamente un 30 % mayor que lo esperado. Suponemos que esto se debe a que la estimación del radio espectral solamente sirve para ciertas matrices y condiciones particulares, y que es más precisa para iteraciones i más grandes.

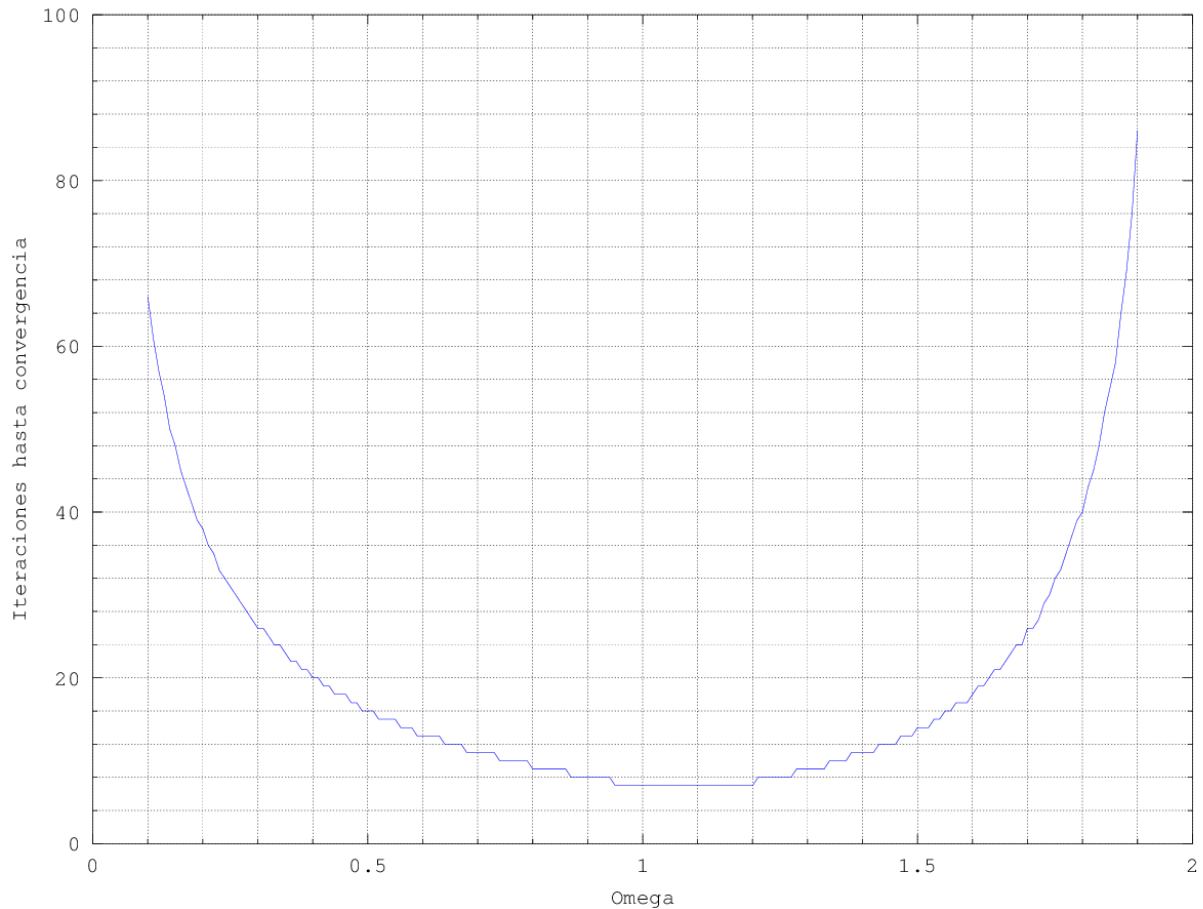
Se realizó un pequeño script en Octave (se incluye en el apéndice C) para calcular el radio espectral real de la matriz de la iteración de Jacobi y la de Gauss Seidel. Dichos radios hallados son aproximadamente 0,5 para la de Jacobi y 0,25 para la de Gauss Seidel. Observando los radios espectrales estimados, es posible ver que la estimación del ρ_J es bastante bueno (en las últimas iteraciones hay una diferencia pequeña entre lo esperado y lo calculado), mientras que la estimación del ρ_{GS} es muy distinta a la esperada. Esto explica por qué Gauss Seidel no esta convergiendo en la mitad de las iteraciones que Jacobi.

Sobre relajación sucesiva

Debido a que la matriz del sistema es tridiagonal, hay una fórmula para calcular el omega óptimo: $\omega_{opt} = \frac{2}{1+\sqrt{1-\rho_{TGS}}}$

Utilizando los valores obtenidos experimentalmente de la sección anterior ($\rho_{TGS} \approx 0,3$) se obtiene un omega óptimo de aproximadamente 1,089.

Para calcular el mejor valor de omega, se iteró para todo ω entre 0,1 y 1,9, guardando cuantas iteraciones hizo falta para llegar a la convergencia con $RTOL < 0,001$.

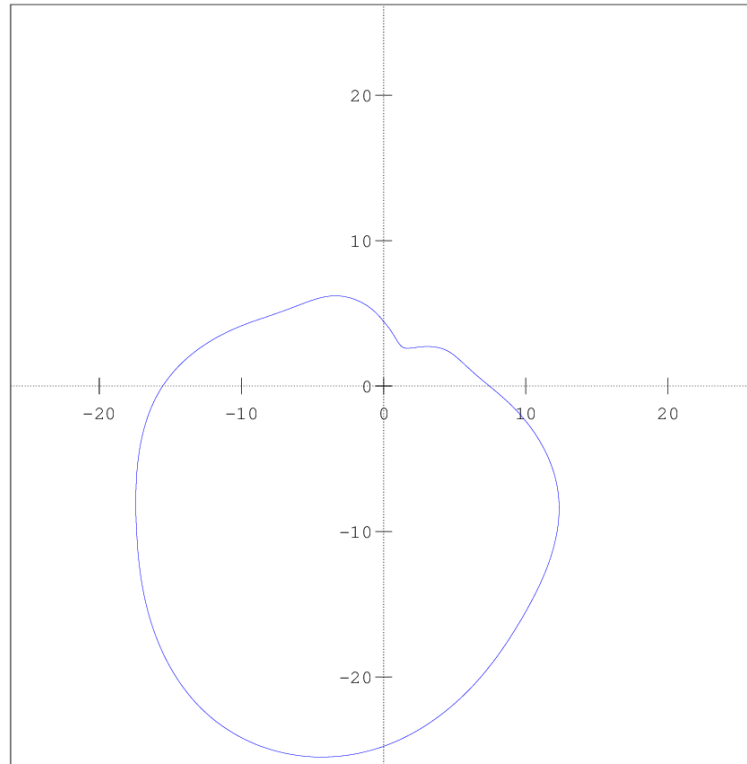


Se puede observar que el gráfico corresponde con lo esperado, habiendo un mínimo absoluto para $\omega \approx 1,1$.

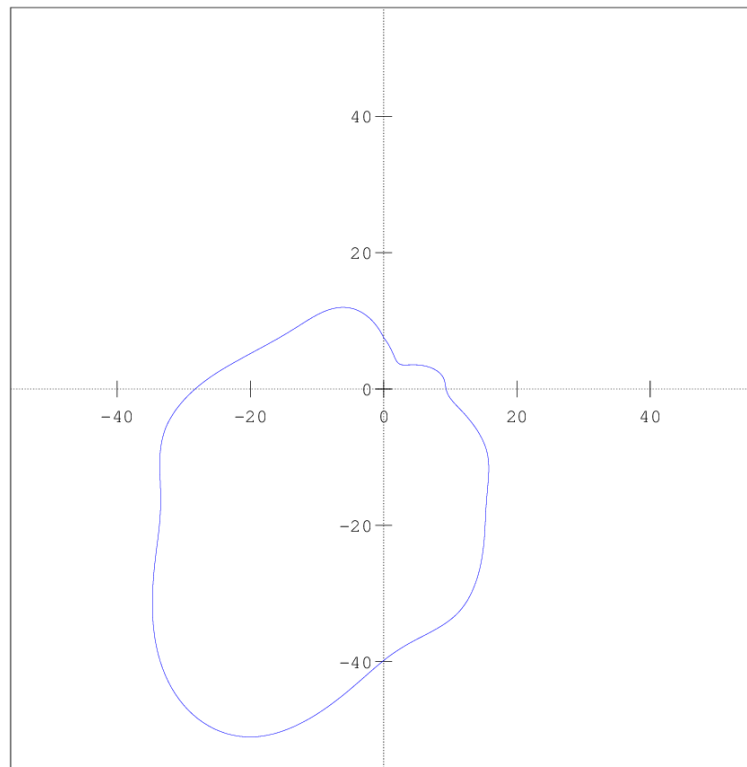
Gráfico de splines obtenidos

Se utilizó como referencia de 0° el eje vertical, y ángulo aumentando en sentido horario. Para evitar tener una curva abierta, se repitió el primer punto. Debido a la no inclusión de condiciones para mantener continuidad de la derivada entre el principio y el final, es posible ver que en $\rho = 0^\circ$ la curva no es suave.

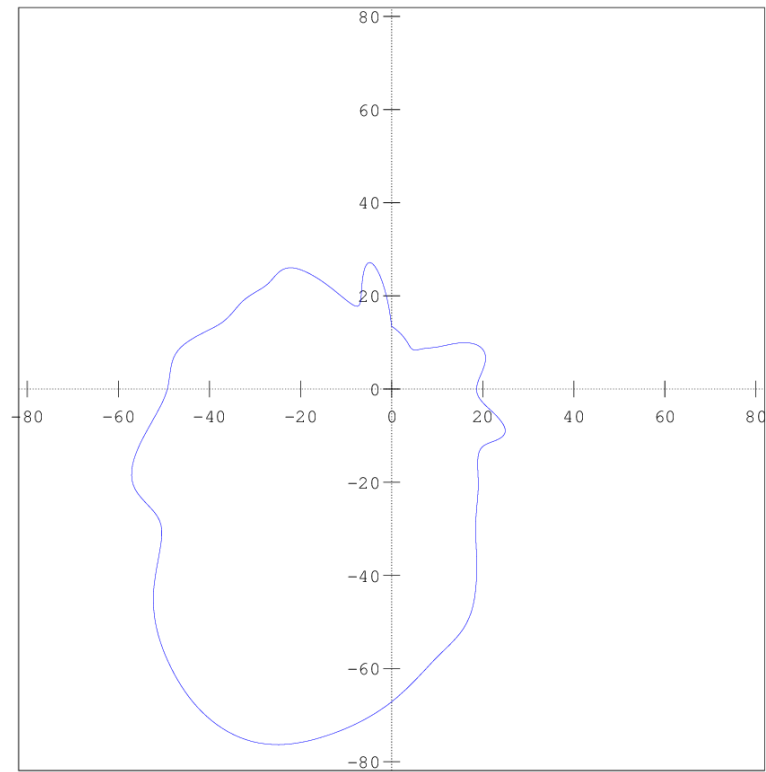
Día 1



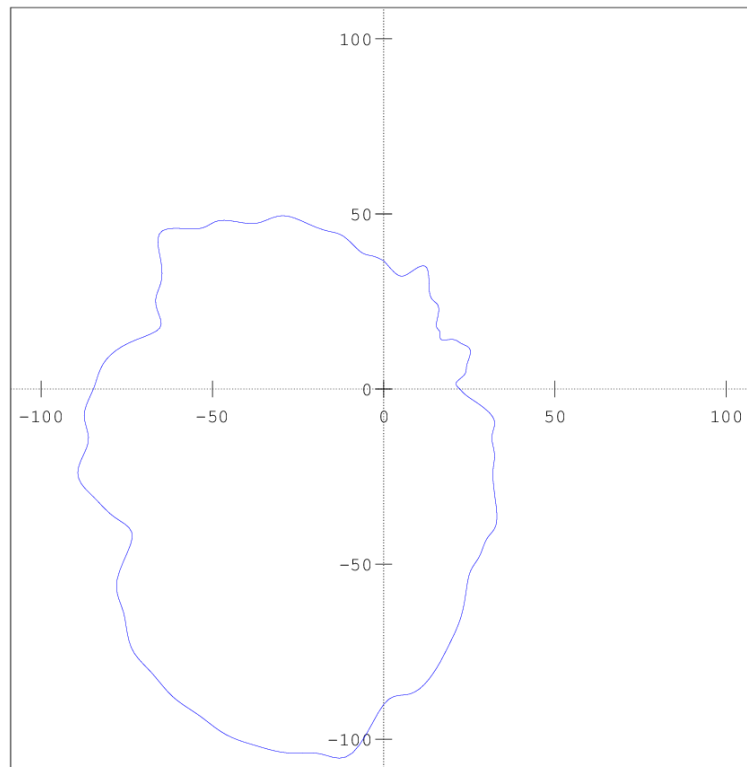
Día 2



Día 3



Día 4



Máxima velocidad de avance

La velocidad máxima de avance del fuego fue calculada observando la diferencia entre los radios de dos días consecutivos, y encontrando el ángulo para el cual el radio varía la mayor cantidad (en módulo).

(COMPLETAR)

Vivienda alcanzada por el fuego

Se busca saber si una vivienda en una posición dada es alcanzada por el fuego. Para averiguar esto primero hay que ver en los cuatro días cuáles fueron los radios del fuego en el ángulo de la casa: si siempre fueron menores entonces la casa no fue alcanzada por el fuego.

Si en algún día el radio del fuego es mayor al de la casa entonces la misma fue alcanzada por el fuego. Para saber en qué tiempo fue alcanzada utilizamos una interpolación lineal con los radios del fuego entre el día que no fue alcanzada y el día que fue alcanzada. Es decir, se asume que la velocidad del fuego es constante para un ángulo dado.

En nuestro caso la casa estaba ubicada en las coordenadas (188,36°; 94,18). La misma fue alcanzada por el fuego entre los días 3 y 4, con radio de 73,942 y 106,11 respectivamente. Entonces el momento en que fue alcanzado por el fuego es: $t \approx (94,18 - 73,942)/(106,11 - 73,942) = 0,62903$. Expresado en días y horas, la casa fue alcanzada aproximadamente el día 3 a las 15 hs.

Conclusiones

El problema físico era analizar el frente de llama de un incendio, utilizando como datos mediciones discretas separadas un cierto ángulo entre sí. El problema numérico era interpolar los datos para poder analizar la posición de las llamas entre dos mediciones. El método utilizado, los splines, en este caso la interpolación segmentaria cúbica plantea un sistema lineal para hallar los coeficientes de los polinomios.

En este caso, dicho sistema lineal se resolvió mediante métodos iterativos: Jacobi, Gauss Seidel y SOR. Como la matriz del sistema es simétrica, tridiagonal, diagonal dominante, la convergencia de dichos métodos está garantizada.

Como se esperaba por ser tridiagonal, el método de Jacobi converge en más iteraciones que Gauss Seidel. Además, SOR permite mejorar la convergencia, forzando los valores. En este caso, debido a que el omega óptimo no está tan alejado del 1, no hay tanta diferencia en cuanto a iteraciones hasta la convergencia entre Gauss Seidel y SOR.

Los tipos de errores involucrados son los siguientes:

- Errores de entrada: son los que provienen de las incertezas en las mediciones de los radios del fuego.
- Errores de truncamiento: debido a que no se pueden hacer infinitas iteraciones de los métodos, las soluciones obtenidas no son las soluciones exactas.
- Errores de redondeo: es el error por trabajar en una computadora, utilizando tipos de datos con precisión finita. Se manifiesta en cada operación pero en este problema es muchos órdenes de magnitud menor al error de truncamiento.

- Errores del modelo matemático: es el error causado por el modelo en sí, en este caso sería que la interpolación spline no aproxime correctamente al frente de llama entre dos mediciones.

A. Polinomios hallados

B. Código (C++)

```
1  #include <iostream>
2  #include <fstream>
3  #include <cmath>
4  #include <string.h>
5  #include <cstdlib>
6  #include <climits>
7  using namespace std;
8
9  #define PADRON1 0.94950
10 #define PADRON2 0.93410
11
12 double normaDos(double *x, int n){
13     double suma = 0.0;
14     for(int i = 0; i < n; i++){
15         suma += (x[i] * x[i]);
16     }
17     return sqrt(suma);
18 }
19 double normaInf(double *x, int n){
20     double maximo = 0.0;
21     for(int i = 0; i < n; i++){
22         maximo = max(maximo, abs(x[i]));
23     }
24     return maximo;
25 }
26
27 int cargarDatos(const char *archivo, double *&x, double *&y){
28     int n;
29     ifstream in(archivo);
30     if(!in.is_open()) return -1;
31
32     //Primera línea: cantidad de puntos
33     in >> n;
34     x = new double[n];
35     y = new double[n];
36
37     for(int i=0;i<n;i++){
38         in >> x[i] >> y[i]; //Cada línea tiene un par (x,y)
39         y[i] *= (PADRON1+PADRON2)/(2.0);
40     }
41
42     return n;
43 }
44
45 void prepararMatriz(double *a, double *h, int n){
46     for(int y=0;y<n;y++){
47         for(int x=0;x<n;x++){
48             int valor = 0;
49
50             if(x == y){
51                 valor = 2.0*(h[y]+h[y+1]); //Diagonal
52             }else if(x == y+1){
53                 valor = h[y+1]; //Arriba de la diagonal
54             }else if(x == y-1){
55                 valor = h[y]; //Abajo de la diagonal
```

```

56     }
57     a[x+y*n] = valor;
58 }
59 }
60
61 //Resuelve  $Ax = b$  por SOR, Siendo  $n$  la dimensión
62 //w la constante de relajación (?) y rtol el error mínimo deseado
63 int sor(double *a, double *x, double *b, int n, double w, double rtol){
64     int it=0;
65     bool termino = false;
66     double *xError = new double[n];
67     double *r = new double[100]; // Para almacenar los errores relativos
68     while(!termino && it<100){
69         for(int nFila=0; nFila<n ;nFila++){
70             double suma = 0.0;
71             // sum desde j = 1 hasta n-1 de  $a_{nj} * x_j$ 
72             for(int j = 0; j < nFila; j++)
73                 suma += a[nFila*n+j] * x[j];
74             for(int j = nFila+1; j < n; j++)
75                 suma += a[nFila*n+j] * x[j];
76             xError[nFila] = x[nFila];
77             //Supongo que la diagonal no es cero
78             x[nFila] = w* ((b[nFila] - suma)/a[nFila*n+nFila]) + (1.0 - w)*x[nFila];
79             xError[nFila] = x[nFila] - xError[nFila];
80         }
81         // Calculo el error para saber si se termino de iterar
82         r[it] = normaInf(xError,n)/normaInf(x,n);
83         termino = r[it] <= rtol;
84
85         /*if(it != 0)
86             cout << "R_GSS[" << it << "]=" << r[it]/r[it-1] << endl;*/
87
88         it++;
89     }
90     delete []r;
91     delete []xError;
92     return it;
93 }
94 int jacobi(double *a, double *x, double *b, int n, double rtol){
95     int it=0;
96     bool termino = false;
97     double *xAnterior = x;
98     double *xActual = new double[n]; // Para cambiar los punteros entre si
99     double *xError = new double[n];
100     double *r = new double[100]; // Para almacenar los errores relativos
101     while(!termino && it<100){
102         for(int nFila=0; nFila<n ;nFila++){
103             double suma = 0.0;
104             // sum desde j = 1 hasta n-1 de  $a_{nj} * x_j$ 
105             for(int j = 0; j < nFila; j++)
106                 suma += a[nFila*n+j] * xAnterior[j];
107             for(int j = nFila+1; j < n; j++)
108                 suma += a[nFila*n+j] * xAnterior[j];
109
110             //Supongo que la diagonal no es cero
111             xActual[nFila] = (b[nFila] - suma)/a[nFila*n+nFila];
112             xError[nFila] = xActual[nFila] - xAnterior[nFila];
113         }
114         // Calculo el error para saber si se termino de iterar
115         r[it] = normaInf(xError,n)/normaInf(xActual,n);

```

```

116     termino = r[it] < rtol;
117     // Cambio los punteros de lugar para la proxima iteracion,
118     // sino x queda con el contenido correcto
119     x = xActual;
120     xActual = xAnterior;
121     xAnterior = x;
122
123     if(it != 0)
124         cout << "R_J[" << it << "]=" << r[it]/r[it-1] << endl;
125
126     it++;
127 }
128 delete []r;
129 delete []xError;
130 delete []xActual;
131 return it;
132 }
133
134 //Dados los Ck, hk los pares de puntos y los h,
135 //imprime todos los polinomios con sus intervalos
136 void poly(double *c, double *x, double *y, double *h, int n){
137     ofstream outCSV("salida.csv");
138
139     for(int k=0;k<n;k++){
140         double a = y[k];
141         double b = (y[k+1]-y[k])/h[k] - (h[k]/3.0) * (2.0*c[k]+c[k+1]);
142         double d = (c[k+1]-c[k])/(3.0*h[k]);
143
144         cout << "y = "
145              << a << "+"
146              << b << "*(x-<<x[k]<<")+"
147              << c[k] << "*(x-<<x[k]<<")^2+"
148              << d << "*(x-<<x[k]<<")^3"
149              << " {" << x[k] << "," << x[k+1] << "}" << endl;
150
151         for(int dx=0;dx<h[k];dx++){
152             //Imprimo los valores de los polinomios cada 1°
153             outCSV << dx+x[k] << "\t" << (a + b*dx + c[k]*dx*dx + d*dx*dx*dx) << endl;
154         }
155     }
156 }
157 void calcW(double *a, double *x, double *b, int n, double rtol){
158     ofstream outCSV("w-optimo.csv");
159     int maxIter = INT_MAX;
160     float maxIterW;
161     outCSV << "W\tIteraciones" << endl;
162     outCSV << "Jacobi\t" << jacobi(a,x,b,n,rtol) << endl;
163     for(int k=0;k<n;k++){
164         x[k] = 0;
165         for(float currentW = 0.1; currentW < 1.9; currentW = currentW + 0.01){
166             int currentIter = sor(a,x,b,n,currentW, rtol);
167             outCSV << currentW << "\t" << currentIter << endl;
168             if(currentIter < maxIter){
169                 maxIter = currentIter;
170                 maxIterW = currentW;
171             }
172             for(int k=0;k<n;k++){
173                 x[k] = 0;
174             }
175             cout << "El w optimo es " << maxIterW << " con " << maxIter << " iteraciones" <<

```

```

    endl;
176 }
177 int main(int argc, char **args){
178     double *x=NULL, *y=NULL;
179     char* file;
180     if(strcmp(args[1], "-v") == 0){
181         cout << "Version 1.0";
182         return 1;
183     }
184     if(strcmp(args[1], "-h") == 0){
185         cout << "Comandos: " << endl;
186         cout << "-h: ayuda sobre los comandos" << endl;
187         cout << "-v: version del programa" << endl;
188         cout << "-sor [tol] [file] [w]: se utiliza el metodo sor con el w indicado
            sobre el archivo indicado con la tolerancia indicada" << endl;
189         cout << "-jcb [tol] [file]: se utiliza el metodo de jacobi sobre el archivo
            indicado con la tolerancia indicada" << endl;
190         cout << "-g-s [tol] [file]: se utiliza el metodo de gauss-seidel sobre el
            archivo indicado con la tolerancia indicada" << endl;
191         cout << "-calcW [tol] [file]: se calcula el w optimo para el archivo indicado
            con la tolerancia indicada" << endl;
192         return 1;
193     }
194     char* func;
195     float w = 0.0;
196     if(strcmp(args[1], "-jcb") == 0){
197         func = "jcb";
198     } else if((strcmp(args[1], "-g-s") == 0) || (strcmp(args[1], "-sor") == 0)){
199         if(strcmp(args[1], "-g-s") == 0){
200             w = 1.0;
201         } else {
202             w = strtod(args[4], NULL);
203         }
204         func = "sor";
205     } else if(strcmp(args[1], "-calcW") == 0) {
206         func = "calcW";
207     } else {
208         cerr << "No se selecciono un comando valido, intenta con -h" << endl;
209         return 1;
210     }
211     float tol = strtod(args[2], NULL);
212     file = args[3];
213     int n = cargarDatos(file, x, y)-1;
214     if(n<0){
215         cerr << "No se pudo abrir el archivo." << endl;
216         return 1;
217     }
218
219     //N es la cantidad de polinomios, no de puntos totales!
220     cout << "N = " << n << endl;
221
222     double *h = new double[n];
223     for(int k=0;k<n;k++)
224         h[k] = x[k+1] - x[k];
225
226     //Creo el sistema a resolver
227     double *aSist = new double[(n-1)*(n-1)];
228     double *xSist = new double[n+1];
229     double *bSist = new double[n-1];
230

```

```

231 //Semilla inicial para el SOR
232 for(int i=0;i<=n;i++)
233     xSist[i] = 0;
234
235 prepararMatriz(aSist, h, n-1);
236
237 //Preparo el B del sistema
238 for(int k=0;k<n-1;k++)
239     bSist[k] = (3.0/h[k+1]) * (y[k+2]-y[k+1]) - (3.0/h[k+1]) * (y[k+1]-y[k]);
240
241 //Resuelvo el sistema, teniendo en cuenta que xSist son los Ck, por lo que 'salteo'
    el primer elemento
242 if(strcmp(func,"calcW") == 0){
243     calcW(aSist,xSist+1,bSist,n-1, tol);
244 } else {
245     if(strcmp(func,"jcb") == 0){
246         cout << "Iteraciones jacobi: " << jacobi(aSist,xSist+1,bSist,n-1, tol) <<
            endl;
247     } else if(strcmp(func,"sor") == 0){
248         cout << "Iteraciones SOR: " << sor(aSist,xSist+1,bSist,n-1,w,tol) << endl;
249     }
250     poly(xSist, x, y, h, n);
251 }
252 delete []aSist;
253 delete []xSist;
254 delete []bSist;
255 delete []x;
256 delete []y;
257 return 0;
258 }

```

C. Código (Octave/MATLAB)

```

1 LARGO = 70;
2 VALOR = 5;
3
4 D = 4*VALOR*eye(LARGO);
5 U = -diag(VALOR*ones(1,LARGO-1),1);
6 L = -diag(VALOR*ones(1,LARGO-1),-1);
7
8 %GS
9 M = D-L-U;
10 autovalores_GS = eig(inv(D-L)*U);
11 radio_GS = max(abs(autovalores_GS))
12 wopt = 2/(1+sqrt(1-radio_GS))
13
14 %Jacobi
15 autovalores_J = eig(inv(D)*(L+U));
16 radio_J = max(abs(autovalores_J))

```