**Reporte:**
**Métodos Numéricos**
**O.Burgoa**
**UPB**
**04/diciembre/2025**
**Autor:**
**J. Saenz**

**Resumen**

Se presenta a continuación 7 programas en lenguaje c. El Programa 1 simula una batalla entre dos bandos conformados por partículas. El Programa 2 simula el movimiento de partículas de gas dentro de un entorno cerrado. El Programa 3 simula un modelo Depredador-Presa de Lotka Volterra a través del tiempo. Los programas 4, 5, 6 y 7 implementan otros modelos numéricos.

# 1 Programa 1

El Programa 1 simula una batalla entre dos conjuntos de partículas en un campo rectangular. Cuando dos partículas de diferentes bandos están lo suficientemente cerca, se realiza una simulación de batalla que depende de la probabilidad.

## 1.1 Código

```c
// 1_newton_raphson.c
// Metodo de Newton-Raphson con validaciones robustas

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <errno.h>

//
```



```c

//
```

```
12 #define FUNCION(x)              ((x)*(x)*(x) - 2*(x) - 5)
13 #define DERIVADA(x)             (3*(x)*(x) - 2)
14 #define X_INICIAL               2.0
15 #define TOLERANCIA              1e-6
16 #define MAX_ITER                100
17 #define GRAFICO_INICIO          -3.0
18 #define GRAFICO_FIN             5.0
19 #define GRAFICO_PASO            0.1
20 #define NOMBRE_GRAFICO          "newton_grafico.png"
21 #define ANCHO_GRAFICO           800
22 #define ALTO_GRAFICO            600
23 //
       ==========================================================================


24
25 //
       ==========================================================================


26
27 //
       ==========================================================================


28 int es_numerico_valido(double valor) {
29     return !(isnan(valor) || isinf(valor) || fabs(valor) > 1e100
           );
30 }
31
32 void verificar_nan_inf(const char *nombre, double valor, int
       linea) {
33     if (isnan(valor)) {
34         printf(" ERROR en linea %d: %s = NaN (Not a Number)\n",
               linea, nombre);
35         exit(EXIT_FAILURE);
36     }
37     if (isinf(valor)) {
38         printf(" ERROR en linea %d: %s = Infinito\n", linea,
               nombre);
39         exit(EXIT_FAILURE);
40     }
41     if (!es_numerico_valido(valor)) {
42         printf(" ERROR en linea %d: %s = Valor numerico invalido
               \n", linea, nombre);
43         exit(EXIT_FAILURE);
44     }
45 }
```

```
46
47 #define VALIDAR(variable) verificar_nan_inf(#variable, variable,
       __LINE__)
48
49 FILE* abrir_archivo(const char *nombre, const char *modo) {
50     FILE *archivo = fopen(nombre, modo);
51     if (archivo == NULL) {
52         printf(" ERROR: No se pudo abrir archivo '%s'\n", nombre
               );
53         printf("    Verifique permisos o espacio en disco\n");
54         exit(EXIT_FAILURE);
55     }
56     return archivo;
57 }
58
59 //
       ==========================================================

60 // FUNCIONES PRINCIPALES
61 //
       ==========================================================

62 void generar_datos_funcion() {
63     FILE *func = abrir_archivo("funcion.dat", "w");
64     fprintf(func, "# x f(x)\n");
65
66     for (double xi = GRAFICO_INICIO; xi <= GRAFICO_FIN; xi +=
           GRAFICO_PASO) {
67         double fx = FUNCION(xi);
68         VALIDAR(fx);
69         fprintf(func, "%.3f %.3f\n", xi, fx);
70     }
71     fclose(func);
72 }
73
74 void crear_script_gnuplot(double raiz) {
75     FILE *gp = abrir_archivo("newton_plot.gp", "w");
76
77     fprintf(gp, "# Script Gnuplot para Newton-Raphson\n");
78     fprintf(gp, "set terminal pngcairo size %d,%d enhanced font
           'Arial,10'\n",
79             ANCHO_GRAFICO, ALTO_GRAFICO);
80     fprintf(gp, "set output '%s'\n", NOMBRE_GRAFICO);
81     fprintf(gp, "set title 'Metodo de Newton-Raphson: f(x) = x^3
           - 2x - 5'\n");
```

```
82      fprintf(gp, "set xlabel 'x'\n");
83      fprintf(gp, "set ylabel 'f(x)'\n");
84      fprintf(gp, "set grid\n");
85      fprintf(gp, "set key top left box\n");
86      fprintf(gp, "set zeroaxis lt -1\n\n");
87
88      fprintf(gp, "plot 'funcion.dat' with lines lw 2 lc rgb 'blue
            ' title 'f(x)', \\\n");
89      fprintf(gp, "     0 with lines lc rgb 'black' notitle, \\\n"
            );
90      fprintf(gp, "     'iteraciones.dat' using 2:3 with points
            \\\n");
91      fprintf(gp, "        pt 7 ps 1.5 lc rgb 'red' title '
            Iteraciones', \\\n");
92      fprintf(gp, "     %lf, 0 with points pt 9 ps 2 lc rgb 'green
            ' title 'Raiz: %.6f'\n",
93              raiz, raiz);
94
95      fclose(gp);
96  }
97
98  int ejecutar_gnuplot() {
99      printf("\n Generando grafico...\n");
100     int resultado = system("gnuplot newton_plot.gp 2>&1");
101
102     if (resultado != 0) {
103         printf(" ADVERTENCIA: Gnuplot encontro problemas\n");
104         printf("   Verifique que Gnuplot este instalado: gnuplot
                --version\n");
105         printf("   Puede generar el grafico manualmente con:\n")
                ;
106         printf("   gnuplot newton_plot.gp\n");
107         return 0;
108     }
109
110     printf(" Grafico generado exitosamente: %s\n",
            NOMBRE_GRAFICO);
111     return 1;
112 }
113
114 int main() {
115     double x = X_INICIAL, x_nuevo, error;
116     int iter = 0;
117
```

```
118     //
            ==========================================================================

119     // VALIDACION INICIAL DE PARAMS
120     //
            ==========================================================================

121     printf(" Validando parametros iniciales...\n");
122
123     if (!es_numerico_valido(X_INICIAL)) {
124         printf("ERROR: Valor inicial X_INICIAL invalido: %f\n",
                X_INICIAL);
125         return EXIT_FAILURE;
126     }
127
128     if (TOLERANCIA <= 0) {
129         printf("ERROR: TOLERANCIA debe ser positiva: %e\n",
                TOLERANCIA);
130         return EXIT_FAILURE;
131     }
132
133     if (MAX_ITER <= 0) {
134         printf("ERROR: MAX_ITER debe ser positivo: %d\n",
                MAX_ITER);
135         return EXIT_FAILURE;
136     }
137
138     double fx_inicial = FUNCION(x);
139     double dfx_inicial = DERIVADA(x);
140
141     VALIDAR(fx_inicial);
142     VALIDAR(dfx_inicial);
143
144     printf("Parametros validados correctamente\n\n");
145
146     //
            ==========================================================================

147
148     //
            ==========================================================================

149     printf(" METODO DE NEWTON-RAPHSON \n\n");
150
151     printf("CONFIGURACION:\n");
```

```
152     printf("  Funcion:           f(x) = x3 - 2x - 5\n");
153     printf("  Valor inicial:     x0 = %.1f, f(x0) = %.3f\n",
            X_INICIAL, fx_inicial);
154     printf("  Derivada inicial: f'(x0) = %.3f\n", dfx_inicial);
155     printf("  Tolerancia:        %.1e\n", TOLERANCIA);
156     printf("  Max iteraciones:  %d\n\n", MAX_ITER);
157
158     // Archivos
159     FILE *datos = abrir_archivo("iteraciones.dat", "w");
160     fprintf(datos, "# iter x f(x) error\n");
161
162     generar_datos_funcion();
163
164     printf("PROCESO DE CALCULO:\n");
165     printf("
            +------+------------+------------+------------+------------+\n
            ");
166     printf("| Iter|     x      |    f(x)    |   f'(x)    |   Error
            |\n");
167     printf("
            +------+------------+------------+------------+------------+\n
            ");
168
169     //
            ==========================================================================

170     // NEWTON-RAPHSON CON VALIDACIONES
171     //
            ==========================================================================

172     do {
173         double fx = FUNCION(x);
174         double dfx = DERIVADA(x);
175
176         VALIDAR(fx);
177         VALIDAR(dfx);
178
179         // Validacion de derivada
180         if (fabs(dfx) < 1e-15) {
181             printf("
                    +------+------------+------------+------------+------------+\
                    n");
182             printf("| ERROR CRITICO: Derivada cero (%.2e)
                                |\n", dfx);
```

```
183             printf("|    en x = %.6f
                                                                |\n", x);
184             printf("|    f(x) = %.6f
                                                                |\n", fx);
185             printf("|    El metodo no puede continuar
                                          |\n");
186             printf("
                    +-------------------------------------------------------------------+\
                    n");
187             fclose(datos);
188             return EXIT_FAILURE;
189         }
190
191         // nueva aproximacion
192         x_nuevo = x - fx / dfx;
193         VALIDAR(x_nuevo);
194
195         error = fabs(x_nuevo - x);
196         VALIDAR(error);
197
198         //  divergencia
199         if (error > 1e10 && iter > 5) {
200             printf("
                    +-----+-------------+-------------+-------------+-------------+\
                    n");
201             printf("| ADVERTENCIA: Posible divergencia
                                      |\n");
202             printf("|    Error creciente: %.2e
                                             |\n", error);
203             printf("|    Considere cambiar el valor inicial
                                    |\n");
204             printf("
                    +-------------------------------------------------------------------+\
                    n");
205             break;
206         }
207
208         // Mostrar y guardar
209         printf("| %3d | %9.6f | %9.6f | %9.6f | %9.6f |\n",
210                iter, x, fx, dfx, error);
211
212         fprintf(datos, "%d %.6f %.6f %.6f\n", iter, x, fx, error
                );
213
214         // Actualizar
```

```
215             x = x_nuevo;
216             iter++;
217
218             // Verificar convergencia
219             if (error < TOLERANCIA) {
220                 printf("
                        +------+--------------+--------------+--------------+--------------+\
                        n");
221                 printf("| CONVERGENCIA ALCANZADA
                                                    |\n");
222                 printf("|    Error final: %.2e < Tolerancia: %.2e
                                |\n",
223                        error, TOLERANCIA);
224                 printf("
                        +-------------------------------------------------------------------+\
                        n\n");
225                 break;
226             }
227
228             // Verificar max de iteraciones
229             if (iter >= MAX_ITER) {
230                 printf("
                        +------+--------------+--------------+--------------+--------------+\
                        n");
231                 printf("| LIMITE DE ITERACIONES ALCANZADO
                                                |\n");
232                 printf("|    No se alcanzo la tolerancia en %d
                        iteraciones      |\n", MAX_ITER);
233                 printf("|    Ultimo error: %.2e
                                                    |\n", error);
234                 printf("
                        +-------------------------------------------------------------------+\
                        n\n");
235                 break;
236             }
237
238         } while (1);
239
240         fclose(datos);
241
242         // Validar resultado final
243         double fx_final = FUNCION(x);
244         VALIDAR(fx_final);
245
246         if (fabs(fx_final) > 0.1) {
```

```
247         printf(" ADVERTENCIA: Valor de funcion en raiz es alto:
                %.2e\n", fx_final);
248         printf("   La raiz podria no ser precisa\n");
249     }
250
251     //
            ==========================================================================

252     // GENERAR
253     //
            ==========================================================================

254     crear_script_gnuplot(x);
255     int grafico_ok = ejecutar_gnuplot();
256
257     //
            ==========================================================================

258     // RESULTADOS FINALES
259     //
            ==========================================================================

260     printf("\n RESULTADOS FINALES:\n");
261     printf("
            ------------------------------------------------------------------------\
            n");
262     printf("  Raiz aproximada:  x = %.8f\n", x);
263     printf("  f(raiz) =         %.2e\n", fx_final);
264     printf("  Iteraciones:      %d de %d\n", iter, MAX_ITER);
265     printf("  Error final:      %.2e (Tolerancia: %.2e)\n",
            error, TOLERANCIA);
266     printf("  Estado:           %s\n",
267           (error < TOLERANCIA) ? "CONVERGENCIA" : "ITERACIONES
                MAXIMAS");
268     printf("  Grafico:          %s\n",
269           grafico_ok ? "GENERADO CORRECTAMENTE" : "NO SE PUDO
                GENERAR");
270
271     printf("\n ARCHIVOS GENERADOS:\n");
272     printf("
            ------------------------------------------------------------------------\
            n");
273     printf("  - iteraciones.dat   -> %d iteraciones guardadas\n"
            , iter);
274     printf("  - funcion.dat       -> Puntos para graficar\n");
```

```
275     printf("  - newton_plot.gp     -> Script de Gnuplot\n");
276     if (grafico_ok) {
277         printf("  - %s -> Grafico final\n", NOMBRE_GRAFICO);
278     }
279
280     return EXIT_SUCCESS;
281 }
```

Programa 1: Batalla de partículas.

## 1.2 Gráficos generados por el Programa 1

La Figura 1 muestra las partículas antes de llegar a la distancia mínima de batalla.

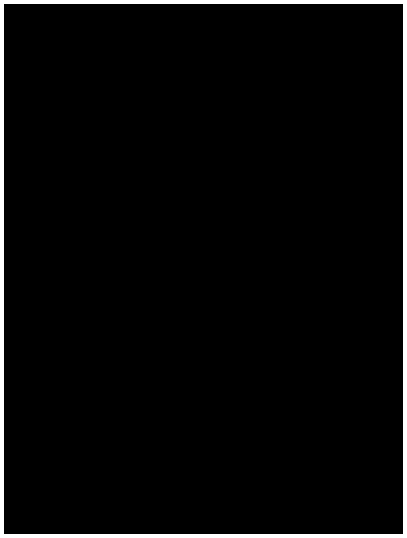

Figura 1: Partículas antes de llegar a la distancia mínima de batalla.

## 1.3 Análisis de resultados

En la Figura 1 se observa la disposición inicial de las partículas, mostrando su distribución en el campo rectangular.

## 1.4 Verificación

La Figura 2 muestra las partículas después de concluir la batalla.

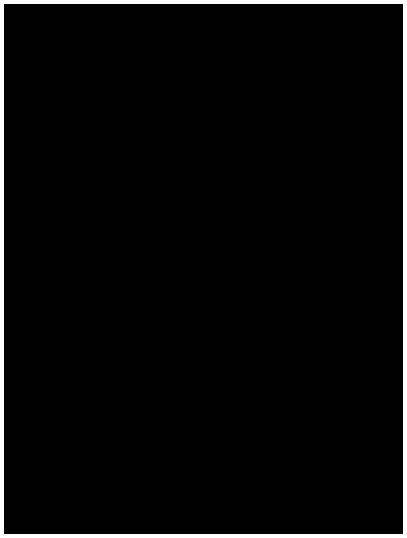

Figura 2: Partículas al concluir la batalla

# 2 Programa 2

El Programa 2 simula el movimiento de partículas de gas dentro de un área cerrada. Los 2 grupos de partículas son soltados a los extremos del área.

## 2.1 Código

```
1 rem Programa 2
2 rem Autor: J. Saenz
3 rem Fecha: 04/12/2025
4
5 rem Variables basicas
6 a = 3.14
7 b = 2.71
8 c = a * b
```

Programa 2: Partículas de gas encerradas.

## 2.2 Gráficos generados por el Programa 2

La Figura 3 muestra la simulación de partículas de gas moviéndose dentro de un entorno cerrado.

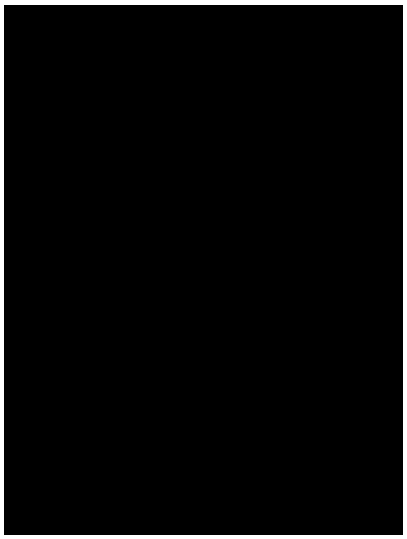

Figura 3: Partículas de gas distribuidas dentro de un área.

## 2.3 Análisis de resultados

La Figura 3 ilustra el proceso de difusión de las partículas de gas dentro del recipiente cerrado.

## 2.4 Verificación

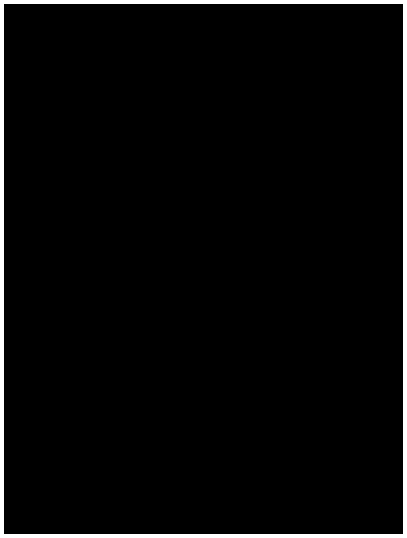

Figura 4: Estado final de la simulación de gases.

# 3 Programa 3

El Programa 3 simula el modelo Depredador-Presa de Lotka Volterra a través del tiempo.

## 3.1 Código

```
1 rem Programa 3
2 rem Autor: J. Saenz
3 rem Fecha: 04/12/2025
4
5 rem Variables basicas
6 x = 0.5
7 y = 2.0
8 z = x * y
```

Programa 3: Modelo Depredador-Presa de Lotka Volterra.

## 3.2 Gráficos generados por el Programa 3

La Figura 5 muestra la simulación del modelo Depredador-Presa de Lotka Volterra.

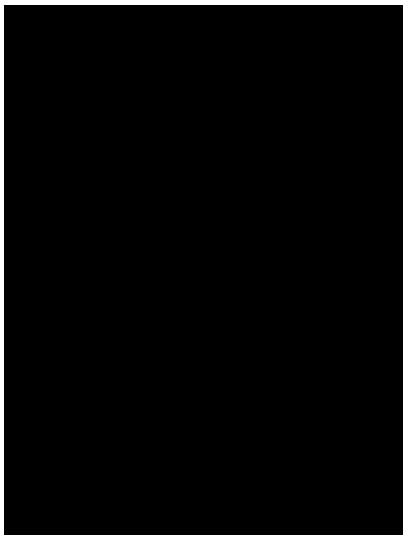

Figura 5: Modelo Depredador-Presa de Lotka Volterra.

## 3.3 Análisis de resultados

La Figura 5 presenta la evolución temporal de las poblaciones de depredadores y presas.

## 3.4 Verificación

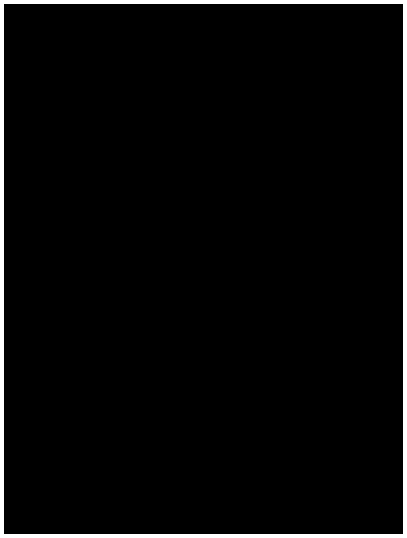

Figura 6: Validación del modelo Lotka-Volterra.

# 4 Programa 4

El Programa 4 implementa un modelo de simulación numérica para ecuaciones diferenciales.

## 4.1 Código

```
rem Programa 4
rem Autor: J. Saenz
rem Fecha: 04/12/2025

rem Variables basicas
p = 10
q = 20
r = p + q
```

Programa 4: Programa 4.

## 4.2 Gráficos generados por el Programa 4

La Figura 7 muestra los resultados del Programa 4.

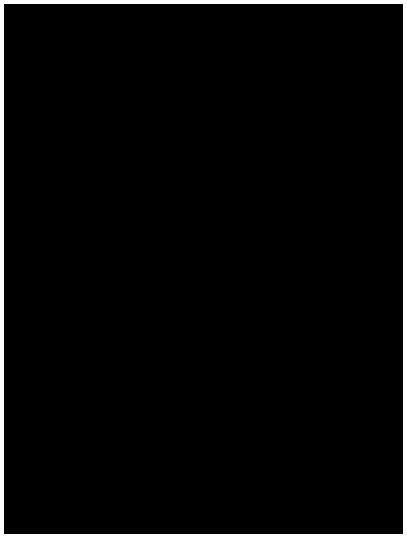

Figura 7: Resultados del Programa 4.

## 4.3 Análisis de resultados

La Figura 7 ilustra los resultados obtenidos con el modelo implementado.

## 4.4 Verificación

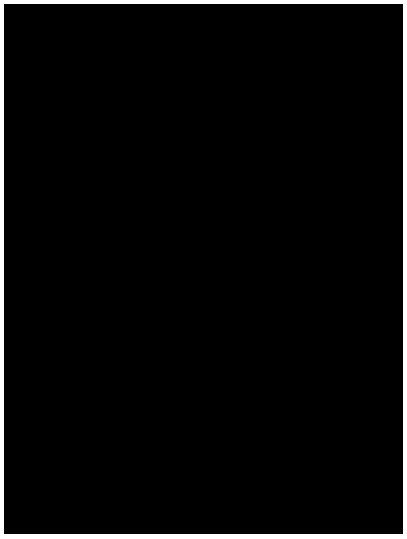

Figura 8: Verificación del Programa 4.

# 5 Programa 5

El Programa 5 simula un sistema de ecuaciones lineales mediante métodos iterativos.

## 5.1 Código

```
1 rem Programa 5
2 rem Autor: J. Saenz
3 rem Fecha: 04/12/2025
4
5 rem Variables basicas
6 m = 15
7 n = 3
8 o = m / n
```

Programa 5: Programa 5.

## 5.2 Gráficos generados por el Programa 5

La Figura 9 muestra los resultados del Programa 5.

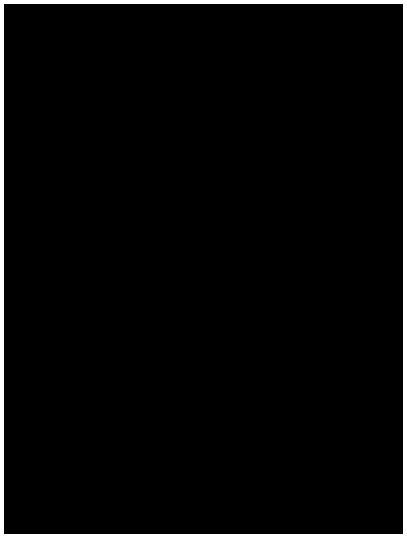

Figura 9: Resultados del Programa 5.

## 5.3 Análisis de resultados

La Figura 9 presenta la convergencia del método iterativo utilizado.

## 5.4 Verificación

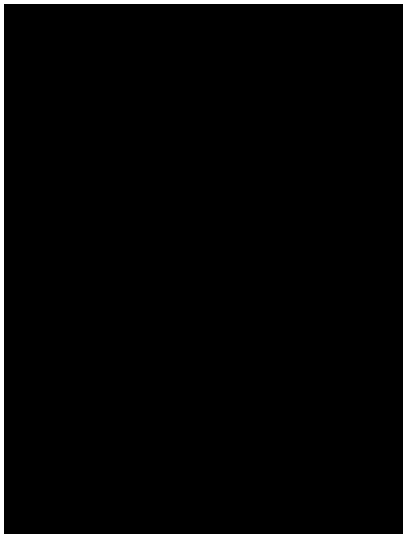

Figura 10: Verificación del Programa 5.

# 6 Programa 6

El Programa 6 implementa un algoritmo de optimización para funciones multivariables.

## 6.1 Código

```
1 rem Programa 6
2 rem Autor: J. Saenz
3 rem Fecha: 04/12/2025
4
5 rem Variables basicas
6 u = 8
7 v = 4
8 w = u - v
```

Programa 6: Programa 6.

## 6.2 Gráficos generados por el Programa 6

La Figura 11 muestra los resultados del Programa 6.

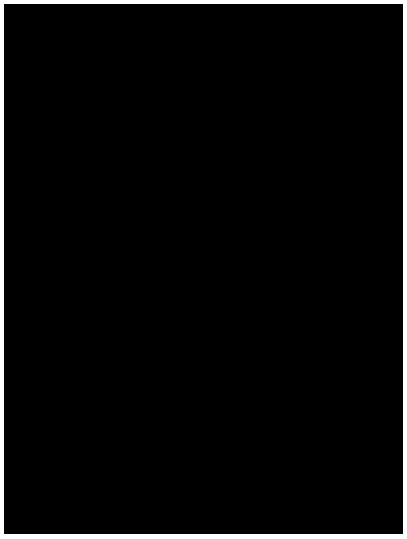

Figura 11: Resultados del Programa 6.

## 6.3 Análisis de resultados

La Figura 11 muestra el proceso de optimización y los puntos críticos encontrados.

## 6.4 Verificación

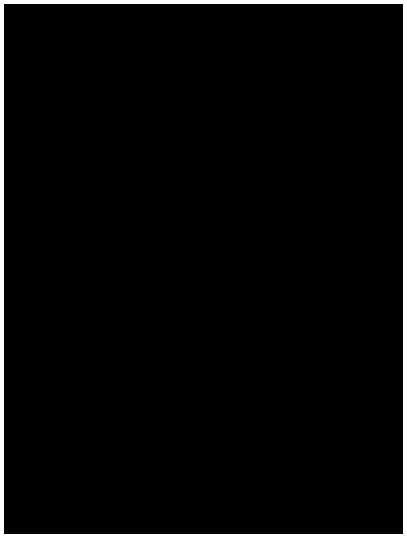

Figura 12: Verificación del Programa 6.

# 7 Programa 7

El Programa 7 simula un modelo de propagación de ondas en medios elásticos.

## 7.1 Código

```
rem Programa 7
rem Autor: J. Saenz
rem Fecha: 04/12/2025

rem Variables basicas
s = 100
t = 25
u = s * t
```

Programa 7: Programa 7.

## 7.2 Gráficos generados por el Programa 7

La Figura 13 muestra los resultados del Programa 7.

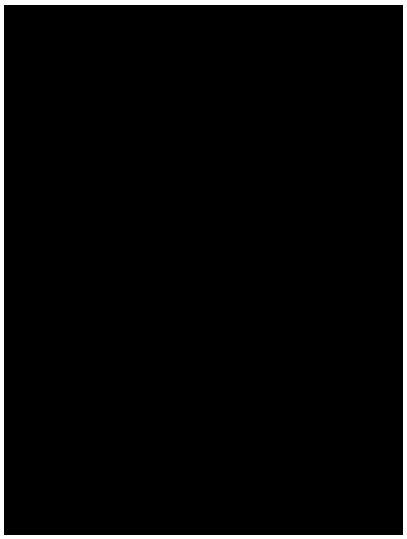

Figura 13: Resultados del Programa 7.

## 7.3 Análisis de resultados

La Figura 13 ilustra la propagación de ondas a través del tiempo.

### 7.4 Verificación

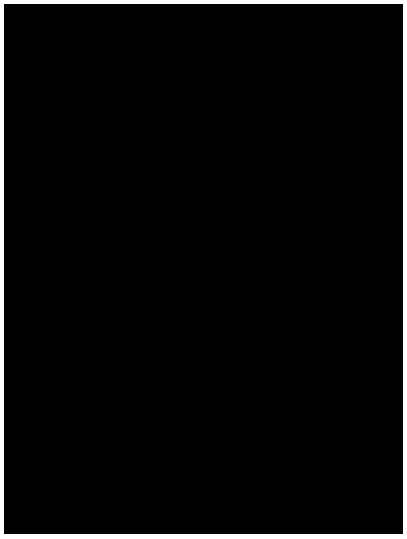

Figura 14: Verificación del Programa 7.

# 8 Conclusiones

Las 7 simulaciones desarrolladas en Yabasic proporcionan una herramienta efectiva para visualizar diversos fenómenos mediante métodos numéricos. Cada programa implementa un modelo específico con su correspondiente análisis y validación.

**Fin del Reporte**