Ecuaciones diferenciales parciales hiperbólicas

Laura Arango
Cristian Serna

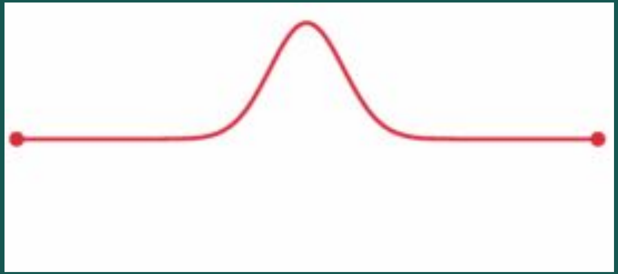
UNIVERSIDAD DE ANTIOQUIA
1 8 0 3

Índice:

Introducción

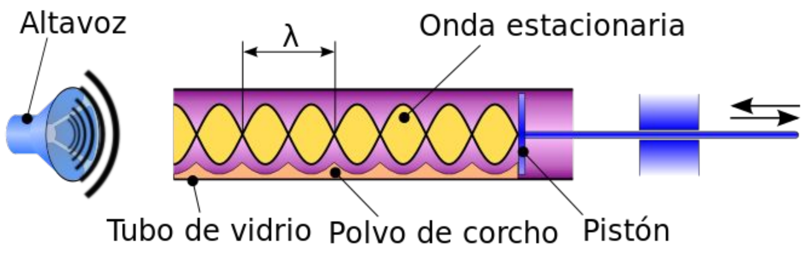
¿Qué son las ecuaciones diferenciales parciales hiperbólicas?

$$\frac{\partial^2 u}{\partial t^2}(x,t) - \alpha^2 \frac{\partial^2 u}{\partial x^2}(x,t) = 0.$$



Ecuación de ondas 1D $$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}$$

Describe...

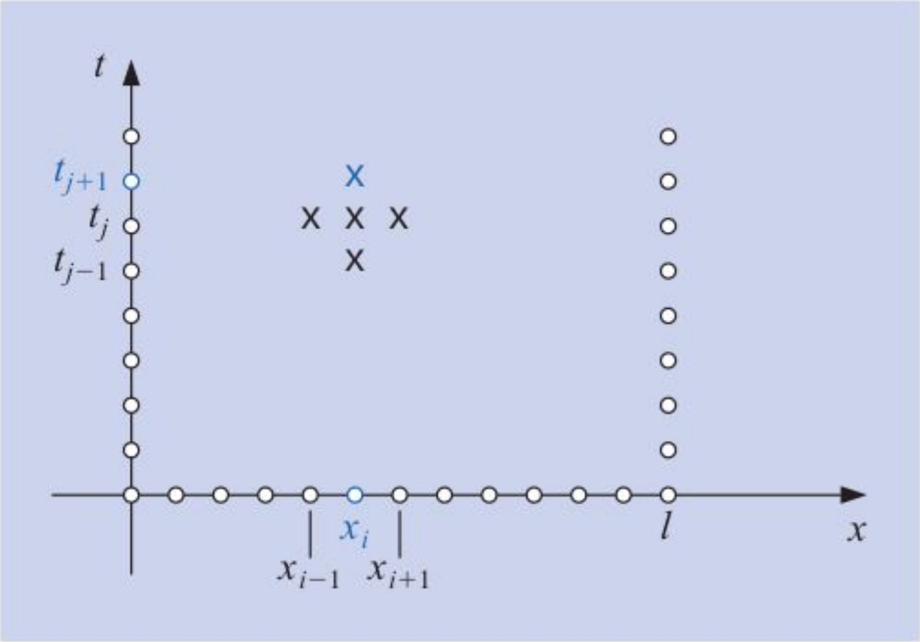


- Ondas en una cuerda
- Ondas en una cadena de monómeros
- Ondas sonoras en un tubo
- Propagación de la luz.

Diferencias finitas

$$\frac{\partial^2 u}{\partial t^2}(x_i, t_j) - \alpha^2 \frac{\partial^2 u}{\partial x^2}(x_i, t_j) = 0$$

$$(x_i, t_j) \qquad \begin{aligned} x_i &= ih \\ t_j &= jk \end{aligned}$$



Diferencias finitas para PDE hiperbólicas

Fórmula de punto medio para la segunda derivada:

$$f''(x_0) = \frac{1}{h^2}[f(x_0 - h) - 2f(x_0) + f(x_0 + h)]$$

$$\frac{\partial^2 u}{\partial t^2}(x_i, t_j) = \frac{u(x_i, t_{j+1}) - 2u(x_i, t_j) + u(x_i, t_{j-1})}{k^2}$$

$$\frac{\partial^2 u}{\partial x^2}(x_i, t_j) = \frac{u(x_{i+1}, t_j) - 2u(x_i, t_j) + u(x_{i-1}, t_j)}{h^2}$$

Diferencias finitas para PDE hiperbólicas

Sustituimos en: $$\frac{\partial^2 u}{\partial t^2}(x_i, t_j) - \alpha^2 \frac{\partial^2 u}{\partial x^2}(x_i, t_j) = 0$$

$$\frac{u(x_i, t_{j+1}) - 2u(x_i, t_j) + u(x_i, t_{j-1})}{k^2} - \alpha^2 \frac{u(x_{i+1}, t_j) - 2u(x_i, t_j) + u(x_{i-1}, t_j)}{h^2} = 0$$

Diferencias finitas para PDE hiperbólicas

Definimos: $u(x_i, t_j) = w_{i,j}$ $\lambda = \alpha k/h$

$$\frac{w_{i,j+1} - 2w_{i,j} + w_{i,j-1}}{k^2} - \alpha^2 \frac{w_{i+1,j} - 2w_{i,j} + w_{i-1,j}}{h^2} = 0$$

Diferencias finitas para PDE hiperbólicas

$$(x_i, t_j) \qquad \begin{aligned} x_i &= ih \\ t_j &= jk \end{aligned} \qquad \lambda = \alpha k/h$$

$$w_{i,j+1} = 2\left(1 - \lambda^2\right) w_{i,j} + \lambda^2 \left(w_{i+1,j} + w_{i-1,j}\right) - w_{i,j-1}$$

Condiciones iniciales y de frontera

$$u(0,t) = u(l,t) = 0 : \quad t > 0 \qquad w_{0,j} = w_{m,j} = 0, \quad j = 1,2,3,\ldots,$$

$$u(x,0) = f(x) \qquad w_{i,0} = f(x_i), \quad i = 1,2,\ldots,m-1$$

$$\frac{\partial u}{\partial t}(x,0) = g(x) : \quad 0 \leq x \leq l$$

Condiciones de frontera

Expansión de Taylor:

$$u(x_i, t_1) = u(x_i, 0) + k\frac{\partial u}{\partial t}(x_i, 0) + \frac{k^2}{2}\frac{\partial^2 u}{\partial t^2}(x_i, 0)$$

$$u(x_i, t_1) = u(x_i, 0) + kg(x_i) + \frac{\alpha^2 k^2}{2} f''(x_i)$$

Condiciones de frontera

$$f''(x_i) \approx \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{h^2}$$

$$u(x_i, t_1) = u(x_i, 0) + kg(x_i) + \frac{\lambda^2}{2}[f(x_{i+1}) - 2f(x_i) + f(x_{i-1})]$$

Condiciones de frontera

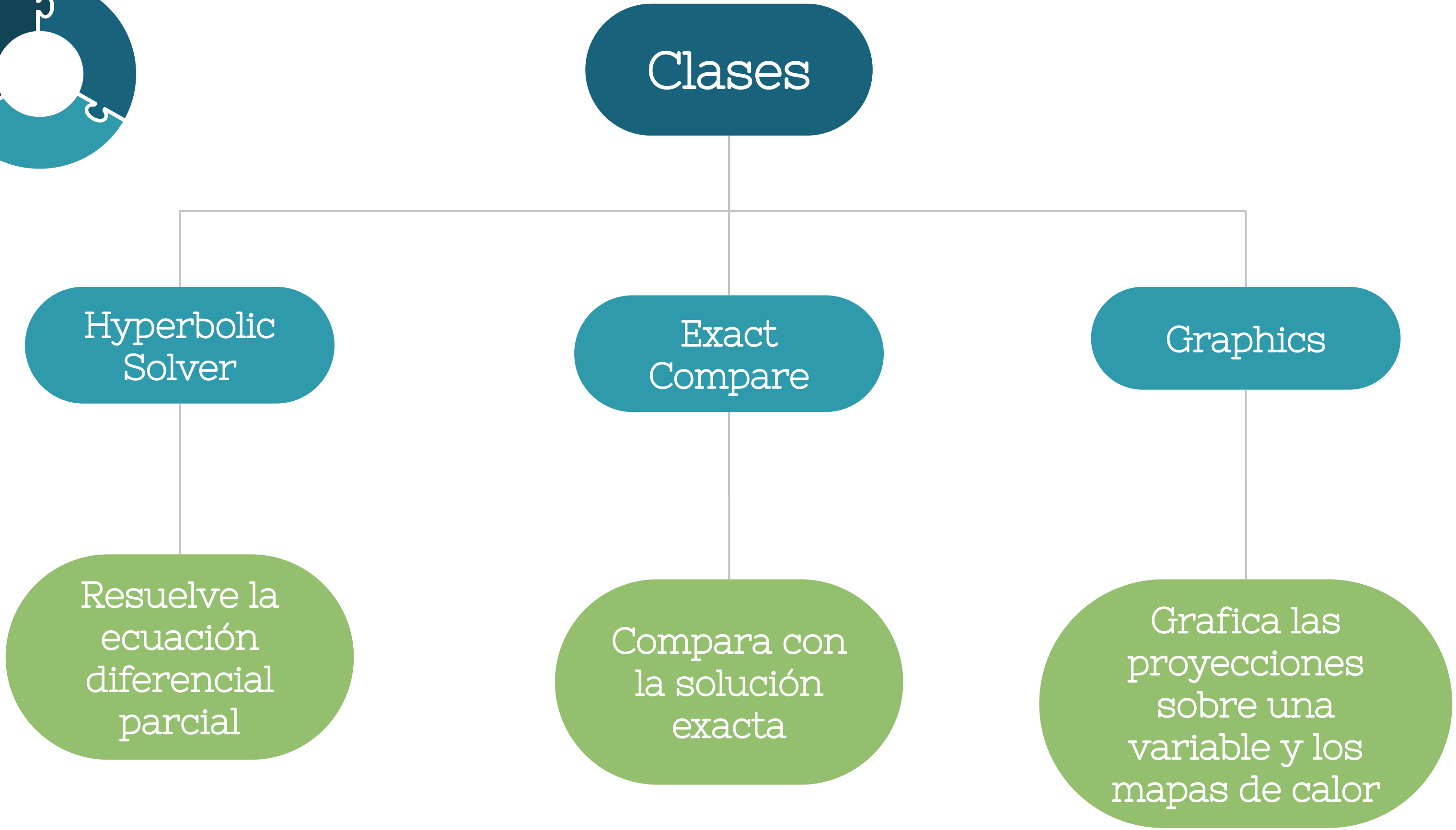
$$u(0,t) = u(l,t) = 0 : \quad t > 0 \qquad w_{0,j} = w_{m,j} = 0, \quad j = 1,2,3,\ldots,$$

$$u(x,0) = f(x) \qquad w_{i,0} = f(x_i), \quad i = 1,2,\ldots,m-1$$

$$\frac{\partial u}{\partial t}(x,0) = g(x) : \quad 0 \le x \le l \qquad w_{i,1} = (1-\lambda^2)f(x_i) + \frac{\lambda^2}{2}\left(f(x_{i+1}) + f(x_{i-1})\right) + kg(x_i)$$

Implementación

```
├── bin
│   └── plot
├── examples
│   ├── example0.cpp
│   ├── example1.cpp
│   ├── example2.cpp
│   └── plot.cpp
├── include
│   ├── colorcitos.h
│   ├── ExactCompare.h
│   ├── graphics.h
│   ├── HyperbolicSolver.h
│   └── matplotlibcpp.h
├── makefile
├── obj
│   ├── ExactCompare.o
│   ├── examples
│   │   └── plot.o
│   ├── graphics.o
│   └── HyperbolicSolver.o
└── src
    ├── ExactCompare.cpp
    ├── graphics.cpp
    └── HyperbolicSolver.cpp
```


Clases
Hyperbolic Solver
Exact Compare
Graphics
Resuelve la ecuación diferencial parcial
Compara con la solución exacta
Grafica las proyecciones sobre una variable y los mapas de calor

Hyperbolic solver

Constructor:

```
HyperbolicSolver::HyperbolicSolver(double (*f)(double), double (*g)(double),float alpha,
                                   float end_point, float maxtime,
                                   const unsigned int x_mesh_size,
                                   const unsigned int t_mesh_size):
                                   m(x_mesh_size), n(t_mesh_size)
```

Variables:

alpha→ Constante función de onda

m → Tamaño de la malla posiciones

n → Tamaño de la malla tiempo

l → Posición máxima

T →Tiempo total de integración

h, k →Variables de la malla

w → Vector multidimensional con las aprox.

posiciones→ Vector con todas las posiciones

tiempos →Vector con los valores del tiempos

lambda → Lambda calculado por alpha, h y k

Hyperbolic solver

Métodos

Funciones get y print:

Para todas las variables se tiene una función get.

Además hay funciones print para:

- Condiciones iniciales
- Condiciones de frontera
- Una tabla donde se incluyen todos los resultados para una variable fija.

También hay función savedata.

Función solve:

Esta función se encarga de solucionar la ecuación diferencial parcial y se compone de diferentes partes basadas en el algoritmo del libro Burden

```
Check of secondary functions:
x divisions 10
t divisions 20
alpha 2.0000000

Initial conditions:
y(0) = 0.0000000
y(l) = 0.0000000

Integration interval:
x ∈ [0, 1.0000000]
t ∈ [0, 1.0000000]
```

Hyperbolic solver

setInitialConditions:

Esta función prepara las condiciones iniciales i.e la primera columna de w.

```
void HyperbolicSolver::setInitialConditions(double (*f)(double), double (*g)(double)){
    /*This function prepares the initial conditions, meaning it sets
    the first column of the matrix w*/

    w.at(0).at(0) = f(0);
    w.at(m).at(0) = f(l);

    for(int i = 1; i < m; i++){
        w.at(i).at(0) = f(i*h);
        w.at(i).at(1) = (1 - std::pow(lambda, 2)) * f(i*h) + std::pow(lambda, 2)/2 *
                        (f((i + 1) * h) + f((i - 1) * h)) + k * g(i*h);
    }
}
```

Set $w_{0,0} = f(0)$;

$w_{m,0} = f(l)$.

For $i = 1, \ldots, m-1$ (*Initialize for $t = 0$ and $t = k$.*)

set $w_{i,0} = f(ih)$;

$$w_{i,1} = (1-\lambda^2) f(ih) + \frac{\lambda^2}{2}[f((i+1)h) + f((i-1)h)] + kg(ih).$$

$$
\begin{bmatrix} w_{1,j+1} \\ w_{2,j+1} \\ \vdots \\ w_{m-1,j+1} \end{bmatrix} = \begin{bmatrix} 2(1-\lambda^2) & \lambda^2 & 0 & \cdots & 0 \\ \lambda^2 & 2(1-\lambda^2) & \lambda^2 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \lambda^2 \\ 0 & \cdots & 0 & \lambda^2 & 2(1-\lambda^2) \end{bmatrix} \begin{bmatrix} w_{1,j} \\ w_{2,j} \\ \vdots \\ w_{m-1,j} \end{bmatrix} - \begin{bmatrix} w_{1,j-1} \\ w_{2,j-1} \\ \vdots \\ w_{m-1,j-1} \end{bmatrix}
$$

$$
w_{i,j+1} = 2\left(1-\lambda^2\right) w_{i,j} + \lambda^2 \left(w_{i+1,j} + w_{i-1,j}\right) - w_{i,j-1}
$$

Hyperbolic solver

matrixMultiplication:

Calcula los valores wij para la matriz de solución w.

```cpp
void HyperbolicSolver::matrixMultiplication(){
    /*This function performs the matrix multiplication for wij calculation*/

    for(int j = 1; j < n; j++){
        for(int i = 1; i < m; i++){
            w.at(i).at(j + 1) = 2 * (1 - std::pow(lambda, 2)) * w.at(i).at(j) + std::pow(lambda, 2) * (w.at(i + 1).at(j) + w.at(i - 1).at(j)) - w.at(i).at(j - 1);
        }
    }
}
```

For $j = 1, \ldots, N-1$ *(Perform matrix multiplication.)*

for $i = 1, \ldots, m-1$

set $w_{i,j+1} = 2(1-\lambda^2)w_{i,j} + \lambda^2(w_{i+1,j} + w_{i-1,j}) - w_{i,j-1}$.

Hyperbolic solver

solve:

Realiza el cálculo completo, saca tiempos y posiciones

```
void HyperbolicSolver::solve(double (*f)(double), double (*g)(double)){
    setInitialConditions(f, g);
    matrixMultiplication();
    for (int j = 0; j < n+1; j++){
        tiempos.at(j) = j*k;

        for (int i = 0; i < m+1; i++){
            posiciones.at(i) =i*h;
        }
    }
}
```

For $j = 0, \ldots, N$

  set $t = jk$;

  for $i = 0, \ldots, m$

    set $x = ih$;

    OUTPUT $(x, t, w_{i,j})$.

ExactCompare

Constructor:

```
ExactCompare::ExactCompare(const HyperbolicSolver& Obj_sol,
                           const double (*exact_solution)(double, double)):
                           sol_object(Obj_sol), exact_solution(exact_solution)
                           {calculateValues();}
```

Variables:

Obj_sol → recibe un objeto con la solución aprox.

exact_solution → Función de dos variables que es solución exacta del problema

tiempos→ Tiempos para solución analítica

posiciones→ Posiciones para solución analítica

aproximaciones→vector para almacenar la sol. aproximada por hyperbolicSolver

ExactCompare

Métodos

Funciones get:

Funciones para obtener tiempo, posición y solución aproximada de Hyperbolic solver.

Función print_table:

Imprime una tabla para un valor fijo de posición o tiempo comparando los valores esperados y los aproximados

Función calculateValues:

Evalúa los valores de posición y tiempo en la función exacta para obtener valor esperado

ExactCompare

calculateValues:

Evalúa en la función exacta para posiciones y tiempos dados.

calculateValues()

```

void ExactCompare::calculateValues(){
    /*Evaluates the function in all the points of the mesh.*/
    //Get the mesh
    tiempos = sol_object.getTime();
    posiciones = sol_object.getPositions();
    aproximaciones = sol_object.getW();

    // Vector for the exact solution in the mesh
    w = std::vector <std::vector <double>> (posiciones.size(),
std::vector <double> (tiempos.size()));

    // Evaluate the exact solution in the mesh
    for(int i = 0; i < posiciones.size(); i++){
    for(int j=0; j < tiempos.size(); j++){
        w.at(i).at(j) = exact_solution(posiciones.at(i),
tiempos.at(j));
    }
}
}
```

Graphics

Métodos

Constructor:

```
graphics::graphics(const HyperbolicSolver& sol_object)
```

Funciones plot slice:

Grafica una proyección de la solución dejando fija una de las variables, el tiempo o la posición, utilizando el wrapper matplotlib cpp.

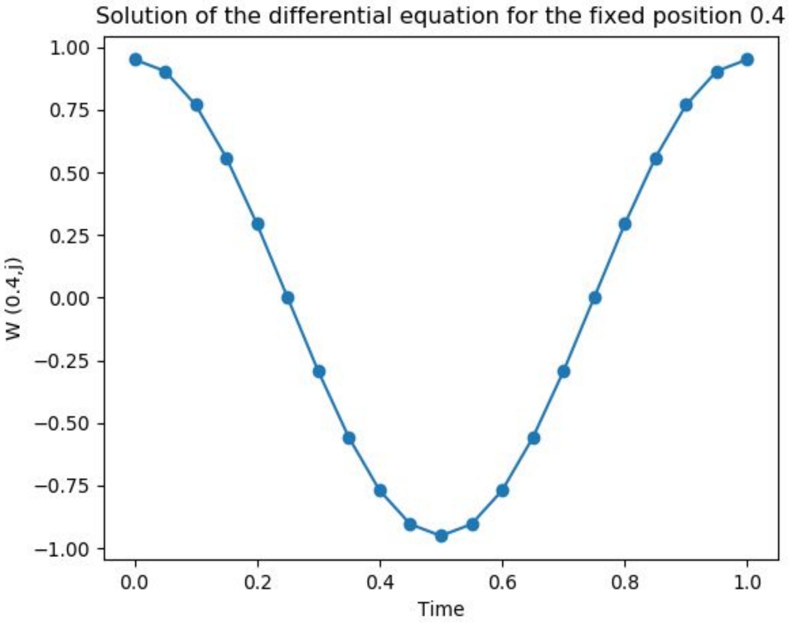
Las funciones están sobre cargadas para recibir un objeto

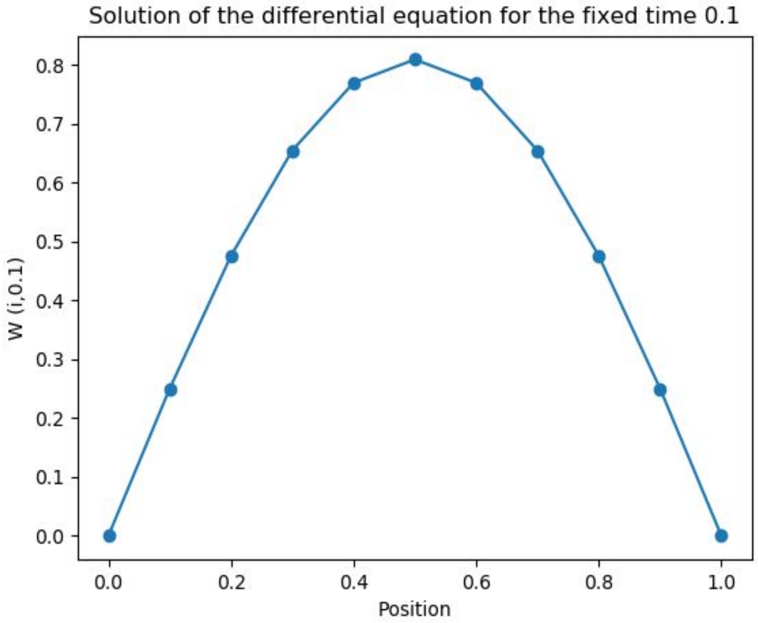
Funciones Heatmap:

Grafica un mapa de calor con todas las variables

Graphics

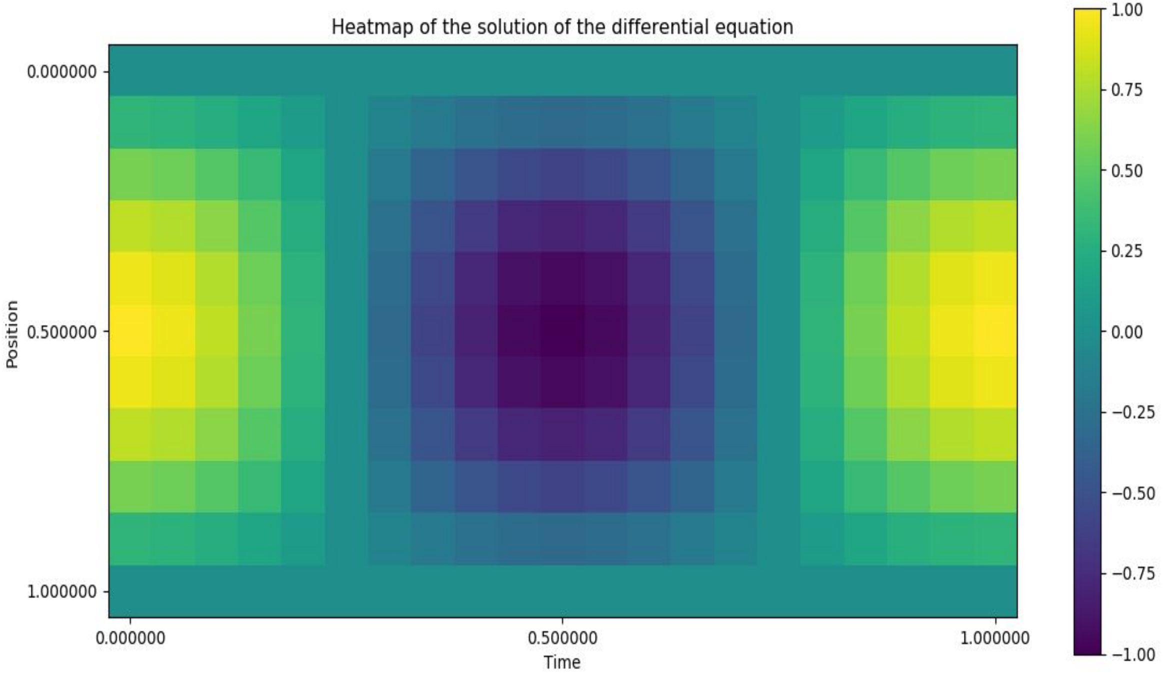
Métodos





Graphics

Métodos



Ejemplos de uso

Solución a ecuaciones de onda y gráficas

Ejemplo 0: Comprobación

$$\frac{\partial^2 u}{\partial t^2}(x,t) - 4\frac{\partial^2 u}{\partial x^2}(x,t) = 0$$

$$u(0,t) = u(1,t) = 0; \qquad 0 < t,$$

$$u(x,0) = \sin(\pi x); \quad 0 \leq x \leq 1,$$

Definición funciones

```
float f(float x){ return std::sin(x*pi);}
float g(float x){ return 0;}

const float example0_sol(float x,float t){ return std::sin(x*pi)*std::cos(2*pi*t);}
```

$$\frac{\partial u}{\partial t}(x,0) = 0; \quad 0 \leq x \leq 1,$$

$$u(x,t) = \sin \pi x \cos 2\pi t$$

Ejemplo 0: Comprobación

Declaración y tablas básicas

```
const HyperbolicSolver example0 ( f, g, 2, 1, 1, 10, 20);
ExactCompare example0_exact (example0, example0_sol);
graphics example0_graph (example0);
example0.print_table(POSITION, 4);
example0.print_table(TIME, 10);

example0.info();
```

```
Check of secondary functions:
x divisions 10
t divisions 20
alpha 2.0000000

Initial conditions:
y(0) = 0.0000000
y(l) = 0.0000000

Integration interval:
x ∈ [0, 1.0000000]
t ∈ [0, 1.0000000]
```

```
Time fixed at:
Index: 10
Time: 0.5000000
```

Position	W(i,10)
0.0000000	0.0000000
0.1000000	-0.3090169
0.2000000	-0.5877853
0.3000000	-0.8090171
0.4000000	-0.9510564
0.5000000	-1.0000000
0.6000000	-0.9510564
0.7000000	-0.8090168
0.8000000	-0.5877852
0.9000000	-0.3090170
1.0000000	0.0000000

Ejemplo 0: Comprobación

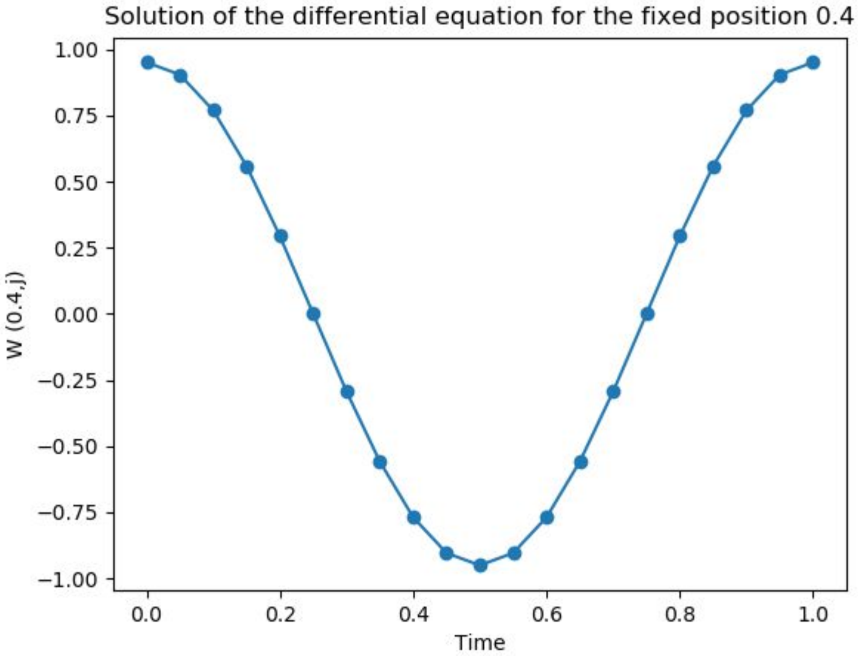
Tablas comparativas y gráficas

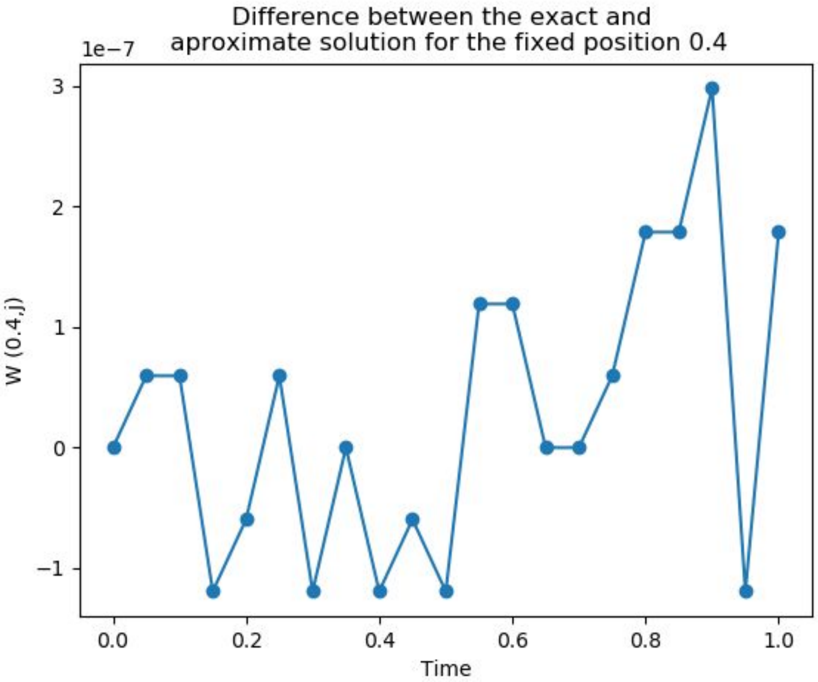
```
example0_exact.print_comp_table(TIME, 10);
example0_exact.print_comp_table(POSITION, 4);

example0_graph.plot_slice(POSITION, 4);
example0_graph.plot_slice(example0_exact, POSITION, 4);
example0_graph.plot_heatmap();
example0_graph.plot_heatmap(example0_exact);
```

```
Time fixed at:
Index: 10
Time: 0.5000000
```

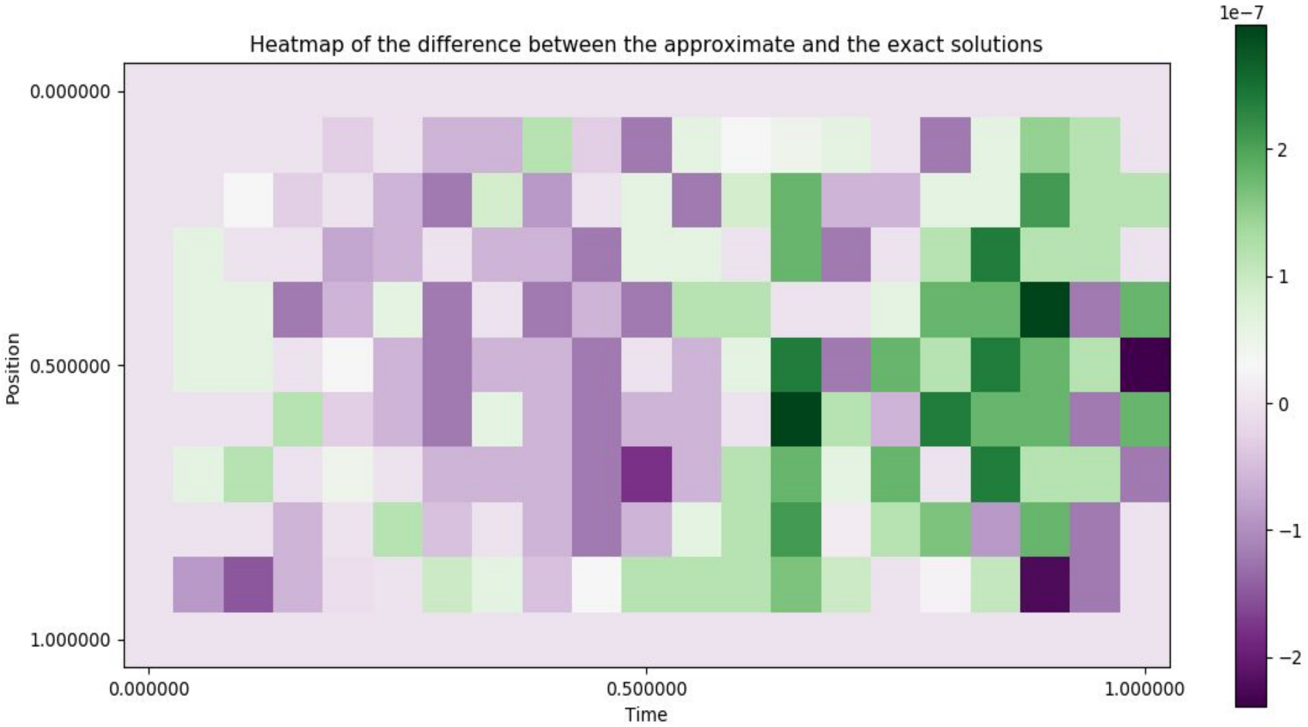
Position	W(i,10)	Exact value
0.0000000	0.0000000	-0.0000000
0.1000000	-0.3090169	-0.3090170
0.2000000	-0.5877853	-0.5877852
0.3000000	-0.8090171	-0.8090170
0.4000000	-0.9510564	-0.9510565
0.5000000	-1.0000000	-1.0000000
0.6000000	-0.9510564	-0.9510565
0.7000000	-0.8090168	-0.8090170
0.8000000	-0.5877852	-0.5877852
0.9000000	-0.3090170	-0.3090169
1.0000000	0.0000000	-0.0000000

Ejemplo 0: Comprobación





Ejemplo 0: Comprobación



Ejemplo 1

$$\frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} = 0$$

$$u(0,t) = u(1,t) = 0$$

Declaración y tablas básicas

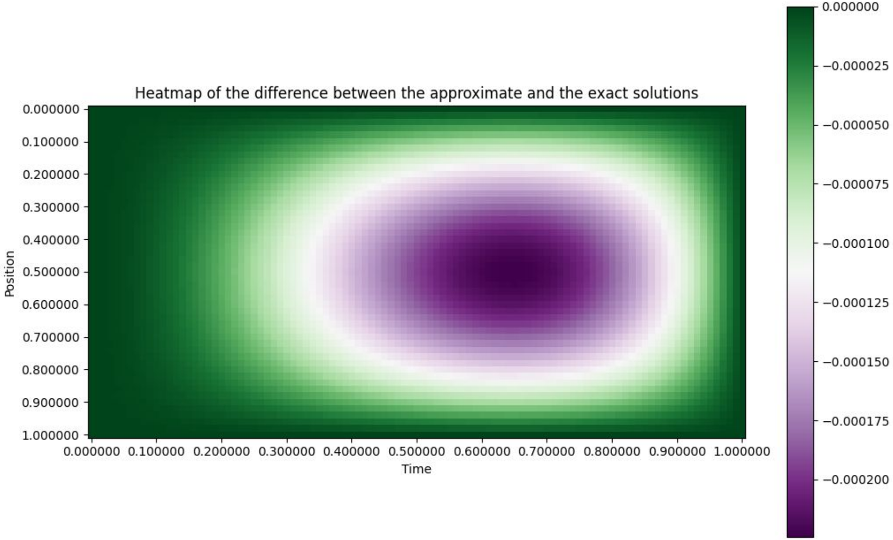
```
float f(float x){ return std::sin(x*pi);}
float g(float x){ return 0;}

const float example1_sol(float x,float t){
  return std::sin(x*pi)*std::cos(pi*t);}
```

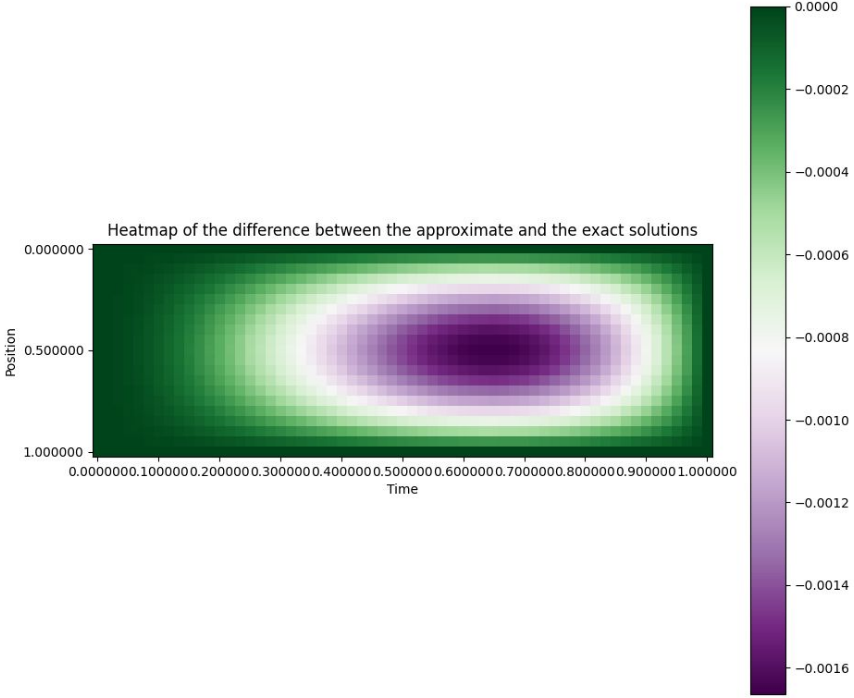
$$u(x,0) = \sin(\pi x)$$

$$\frac{\partial u}{\partial t}(x,0) = 0; \quad 0 \le x \le 1,$$

$$u(x,t) = \cos(\pi t)\sin(\pi x)$$

Ejemplo 1


Heatmap of the difference between the approximate and the exact solutions
Position
0.000000
0.100000
0.200000
0.300000
0.400000
0.500000
0.600000
0.700000
0.800000
0.900000
1.000000
Time
0.000000 0.100000 0.200000 0.300000 0.400000 0.500000 0.600000 0.700000 0.800000 0.900000 1.000000
0.000000
−0.000025
−0.000050
−0.000075
−0.000100
−0.000125
−0.000150
−0.000175
−0.000200


Heatmap of the difference between the approximate and the exact solutions
Position
0.000000
0.500000
1.000000
Time
0.000000 0.100000 0.200000 0.300000 0.400000 0.500000 0.600000 0.700000 0.800000 0.900000 1.000000
0.0000
−0.0002
−0.0004
−0.0006
−0.0008
−0.0010
−0.0012
−0.0014
−0.0016

Ejemplo 2. Presión en un tubo

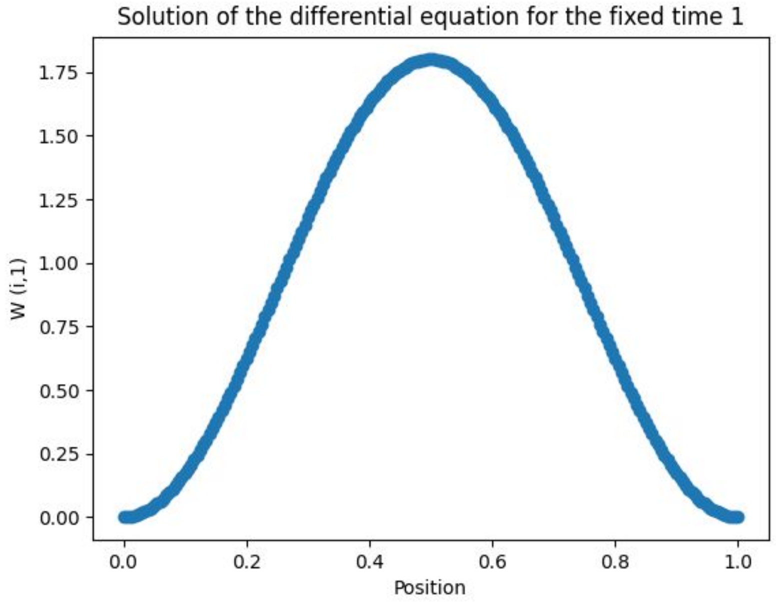
Ondas de presión en un tubo

```
const float p0 = 0.9;

float f(float x){ return p0 *std::cos(2*x*pi) - 1);}
float g(float x){ return 0;}
```

$$p(x,0) = p_0 \cos(2\pi x)$$

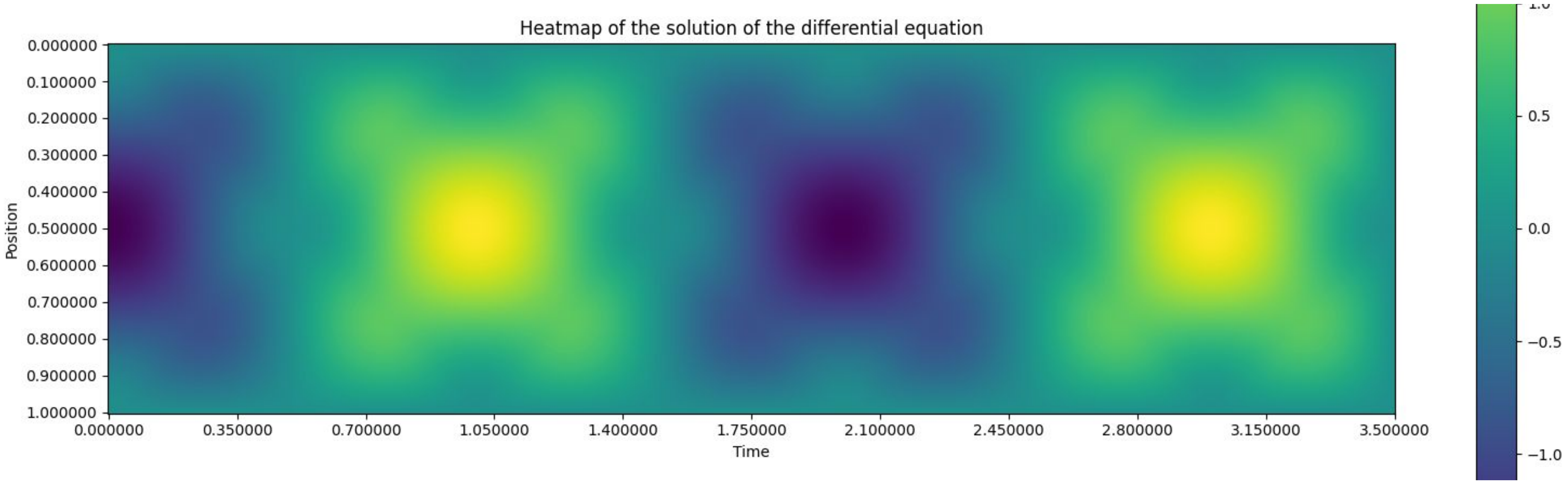
$$p(0,t) = p(l,t) = p_0$$

$$\frac{\partial p}{\partial t}(x,0) = 0 \qquad \frac{\partial^2 p}{\partial x^2} = \frac{1}{c^2}\frac{\partial^2 p}{\partial t^2}$$



Ejemplo 2. Presión en un tubo

$$m = 100 \quad n = 200$$



makefile

```makefile
PYTHONFLAGS = -I/usr/include/python3.9 -lpython3.9
EXAMPLE_FILE = plot

SRC_DIR := src
OBJ_DIR := obj
BIN_DIR := bin

EXAMPLES_DIR := examples
OBJ_EXAMPLES_DIR := $(OBJ_DIR)/$(EXAMPLES_DIR)
EXAMPLE := $(OBJ_EXAMPLES_DIR)/$(EXAMPLE_FILE).o

EXE := $(BIN_DIR)/$(EXAMPLE_FILE)

SRC := $(wildcard $(SRC_DIR)/*.cpp)
OBJ := $(SRC:$(SRC_DIR)/%.cpp=$(OBJ_DIR)/%.o)

CXXFLAGS := -Iinclude/ $(PYTHONFLAGS) -MMD -MP
CFLAGS := -Wall
LDFLAGS := -Llib/
LDLIBS :=

.PHONY: all clean
```

makefile

```makefile
all: $(EXE)

$(EXE): $(OBJ) $(EXAMPLE) | $(BIN_DIR)
	$(CXX) -o $@ $^ $(PYTHONFLAGS)

$(OBJ_DIR)/%.o: $(SRC_DIR)/%.cpp | $(OBJ_DIR)
	$(CXX) $(CXXFLAGS) -c $< -o $@

$(EXAMPLE): $(EXAMPLES_DIR)/$(EXAMPLE_FILE).cpp | $(OBJ_EXAMPLES_DIR)
	$(CXX) $(CXXFLAGS) -c $< -o $@

$(OBJ_EXAMPLES_DIR)/%.o: $(EXAMPLES_DIR)/%.cpp | $(OBJ_EXAMPLES_DIR)
	$(CXX) $(CXXFLAGS) -c $< -o $@

$(BIN_DIR) $(OBJ_DIR) $(OBJ_EXAMPLES_DIR):
	mkdir -p $@

clean:
	@$(RM) -rv $(OBJ_DIR) $(BIN_DIR)

-include $(OBJ:.o=.d)
```