MÉTODO DE DIFERENCIAS FINITAS PARA PROBLEMAS LINEALES.

Valentina Bedoya
Johan Carrillo
Daniel Estrada
Lina Montoya

Física Computacional II - 2020-2

Método de D.F.

Contenido

- Método de diferencias finitas para una ODE de segundo orden con condiciones de frontera.
    - Discretización de la ED.
    - Solución de sistema lineal tridiagonal.
    - Algoritmo
- Estructura del código.
- Implementación del código
    - ejemplo de testeo y análisis de convergencia.
    - Estudio del método sobre un problema físico.

Método de D.F.

Discretización de la ED (BCP)

$$y'' = p(x)y' + q(x)y + r(x)$$

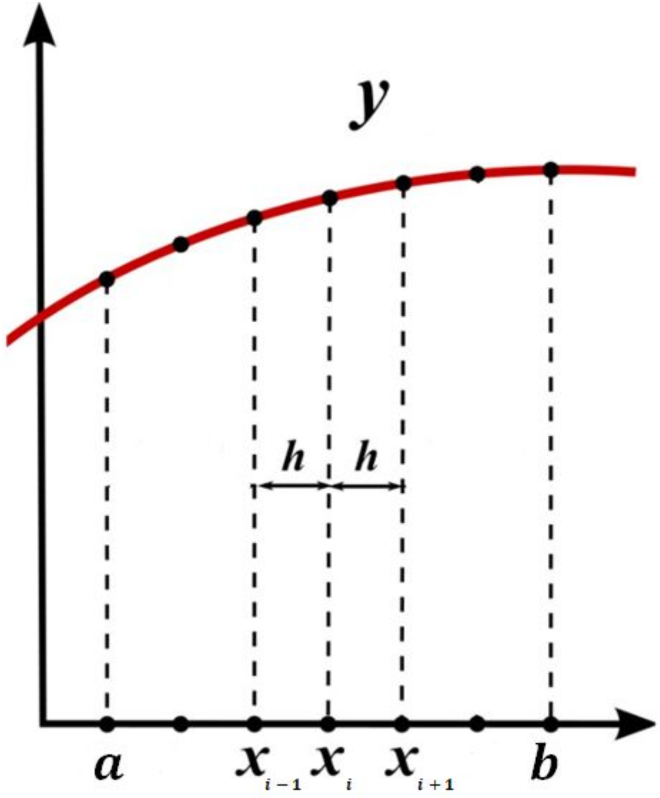
$$a \leq x \leq b$$

$$y(a) = \alpha \quad y(b) = \beta$$

$$h = (b-a)/(N+1)$$

$$x_i = a+ih \qquad i = 0, 1, \ldots, N+1$$

Se expande en Taylor y se evalúa en x_{i-1} y en x_{i+1}



Método de D.F.

Discretización de la ED (BCP)

$$y(x_{i+1}) = y(x_i + h) = y(x_i) + hy'(x_i) + \frac{h^2}{2}y''(x_i) + \frac{h^3}{6}y'''(x_i) + \frac{h^4}{24}y^{(4)}(\xi_i^+)$$
$$y(x_{i-1}) = y(x_i - h) = y(x_i) - hy'(x_i) + \frac{h^2}{2}y''(x_i) - \frac{h^3}{6}y'''(x_i) + \frac{h^4}{24}y^{(4)}(\xi_i^-)$$

Sumando y despejando $\boldsymbol{y''}$:

$$\boxed{y''(x_i) = \frac{1}{h^2}[y(x_{i+1}) - 2y(x_i) + y(x_{i-1})] - \frac{h^2}{12}y^{(4)}(\xi_i)}$$

Restando y despejando $\boldsymbol{y'}$:

fórmula de diferencia centrada

$$\boxed{y'(x_i) = \frac{1}{2h}[y(x_{i+1}) - y(x_{i-1})] - \frac{h^2}{6}y'''(\eta_i)}$$

Método de D.F.

Discretización de la ED (BCP)

Para $i = 1, \dots, N$ la ED es:

$$y''(x_i) = p(x_i)y'(x_i) + q(x_i)y(x_i) + r(x_i)$$

Entonces, remplazando lo de antes, haciendo el cambio $\omega_i = y(x_i)$ y agrupando términos se llega a:

$$\boxed{-\left(1 + \frac{h}{2}p(x_i)\right) w_{i-1} + \left(2 + h^2 q(x_i)\right) w_i - \left(1 - \frac{h}{2}p(x_i)\right) w_{i+1} = -h^2 r(x_i)}$$

Sistema de N ecuaciones con N incógnitas, se puede agrupar de forma matricial y resolver Con algún método.

Método de D.F.

Discretización de la ED (BCP)

Para i

Error de truncación O(h^2)

$$\frac{h^2}{12}\left[2p(x_i)y'''(\eta_i) - y^{(4)}(\xi_i)\right]$$

Entonce ... $_i)$ y

agrupan

$$\boxed{-\left(1+\frac{h}{2}p(x_i)\right)w_{i-1} + \left(2+h^2q(x_i)\right)w_i - \left(1-\frac{h}{2}p(x_i)\right)w_{i+1} = -h^2r(x_i)}$$

Sistema de N ecuaciones con N incógnitas, se puede agrupar de forma matricial y resolver Con algún método.

Crout Factorization for Tridiagonal Linear Systems

Método de D.F.

Se cuenta con un sistema de ecuaciones de la siguiente forma:

$$
\begin{aligned}
E_1 &: \quad a_{11}x_1 + a_{12}x_2 &&= a_{1,n+1},\\
E_2 &: \quad a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &&= a_{2,n+1},\\
&\ \ \vdots \qquad\qquad\qquad \vdots && \ \ \vdots\\
E_{n-1} &: \quad a_{n-1,n-2}x_{n-2} + a_{n-1,n-1}x_{n-1} + a_{n-1,n}x_n &&= a_{n-1,n+1},\\
E_n &: \quad a_{n,n-1}x_{n-1} + a_{nn}x_n &&= a_{n,n+1},
\end{aligned}
$$

Crout Factorization for Tridiagonal Linear Systems

Se procede con la matriz de coeficientes:

$$A = \begin{bmatrix} a_{11} & a_{12} & 0 & \cdots & \cdots & 0 \\ a_{21} & a_{22} & a_{23} & \ddots & & \vdots \\ 0 & a_{32} & a_{33} & a_{34} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & \ddots & a_{n-1,n} \\ 0 & \cdots & \cdots & 0 & a_{n,n-1} & a_{nn} \end{bmatrix}$$

La matriz de coeficientes se factoriza de la forma A = LU

$$L = \begin{bmatrix} l_{11} & 0 & \cdots & \cdots & 0 \\ l_{21} & l_{22} & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & l_{n,n-1} & l_{nn} \end{bmatrix} \quad \text{and} \quad U = \begin{bmatrix} 1 & u_{12} & 0 & \cdots & 0 \\ 0 & 1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & u_{n-1,n} \\ 0 & \cdots & \cdots & 0 & 1 \end{bmatrix}$$

Crout Factorization for Tridiagonal Linear Systems

Todo con la intención de encontrar más fácilmente el vector solución al sistema:

$$\mathbf{LU} = \mathbf{A} \Longrightarrow \mathbf{U} = \mathbf{L}^{-1}\mathbf{A}$$

$$\mathbf{Ax} = \mathbf{b}$$

$$\mathbf{Lz} = \mathbf{b} \Longrightarrow \mathbf{z} = \mathbf{L}^{-1}\mathbf{b}$$

$$\mathbf{Ux}? \Longrightarrow \mathbf{Ux} = \mathbf{L}^{-1}\mathbf{Ax} = \mathbf{L}^{-1}\mathbf{b} = \mathbf{z}$$

Finalmente:

$$\mathbf{Ux} = \mathbf{z}$$

$$A = \begin{bmatrix} 2 + h^2 q(x_1) & -1 + \frac{h}{2}p(x_1) & 0 & \cdots & \cdots & 0 \\ -1 - \frac{h}{2}p(x_2) & 2 + h^2 q(x_2) & -1 + \frac{h}{2}p(x_2) & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & \ddots & -1 + \frac{h}{2}p(x_{N-1}) \\ 0 & \cdots & \cdots & 0 & -1 - \frac{h}{2}p(x_N) & 2 + h^2 q(x_N) \end{bmatrix},$$

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{N-1} \\ w_N \end{bmatrix}, \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} -h^2 r(x_1) + \left(1 + \frac{h}{2}p(x_1)\right) w_0 \\ -h^2 r(x_2) \\ \vdots \\ -h^2 r(x_{N-1}) \\ -h^2 r(x_N) + \left(1 - \frac{h}{2}p(x_N)\right) w_{N+1} \end{bmatrix}.$$

Parte I:

$$\begin{array}{cc} B & D \\ \begin{bmatrix} 2+h^2q(x_1) & -1+\frac{h}{2}p(x_1) & 0 & \cdots & 0 \\ -1-\frac{h}{2}p(x_2) & 2+h^2q(x_2) & -1+\frac{h}{2}p(x_2) & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -1+\frac{h}{2}p(x_{N-1}) \\ 0 & \cdots & 0 & -1-\frac{h}{2}p(x_N) & 2+h^2q(x_N) \end{bmatrix} & \begin{bmatrix} -h^2r(x_1)+\left(1+\frac{h}{2}p(x_1)\right)w_0 \\ -h^2r(x_2) \\ \vdots \\ -h^2r(x_{N-1}) \\ -h^2r(x_N)+\left(1-\frac{h}{2}p(x_N)\right)w_{N+1} \end{bmatrix} \\ C \quad A & \end{array}$$

Método de D.F.

Algoritmo

Parte II:

Set $l_1 = a_1$;

$u_1 = b_1/a_1$;

$z_1 = d_1/l_1$.

For $i = 2, \ldots, N-1$ set $l_i = a_i - c_i u_{i-1}$;

$u_i = b_i/l_i$;

$z_i = (d_i - c_i z_{i-1})/l_i$.

Set $l_N = a_N - c_N u_{N-1}$;

$z_N = (d_N - c_N z_{N-1})/l_N$.

Set $w_0 = \alpha$;

$w_{N+1} = \beta$.

$w_N = z_N$.

For $i = N-1, \ldots, 1$ set $w_i = z_i - u_i w_{i+1}$.

Método de D.F.

Estructura del código

FiniteDiff

```
Attributes (private):
int n=0;
double h;
double a = 0.0;
double b = 1.0;
double alpha;
double beta;

double (*p)(double);
double (*q)(double);
double (*r)(double);

vector< double > A;
vector< double > C;
vector< double > D;
vector< double > B;
```

$N, \quad h, \quad a \leq x \leq b\,, \quad y(a) = \alpha, \quad y(b) = \beta$

$$y'' = \underline{p(x)}y' + \underline{q(x)}y + \underline{r(x)}$$

$$< A > = 2 + h^2 q(x_i), \qquad i = 1, \dots, N$$

$$< B > = -1 + \frac{h}{2} p(x_i), \qquad i = 1, \dots, N-1$$

$$< C > = -1 + \frac{h}{2} p(x_i), \qquad i = 2, \dots, N$$

$$< D > = -h^2 r(x_i), \qquad i = 2, \dots, N$$

$$D_1 = -h^2 r(x_1) + \left(1 + \frac{h}{2} p(x_1)\right)\alpha,$$

$$D_N = -h^2 r(x_N) + \left(1 - \frac{h}{2} p(x_N)\right)\beta$$

FiniteDiff

```
methods (public):
void setInterval(double, double);
void setBoundaryCond(double, double);
void setP(double (*)(double));
void setQ(double (*)(double));
void setR(double (*)(double));
void solve(double *, double *);

(private):
void setH(void);
void calcule_EqSys(double *);
void croutFactorization(double *, double *);
```

```
void FiniteDiff::setH(){
    // Se define el tamano del paso
    h = double ((b - a) / (n + 1));
}

void FiniteDiff::setInterval(double xmin, double xmax){
    // intervalo de integracion
    a = xmin;
    b = xmax;
    setH();
}

void FiniteDiff::setBoundaryCond(double ymin, double ymax){
    // condiciones de frontera
    alpha = ymin;
    beta = ymax;
}

void FiniteDiff::setP(double (*function)(double)){
    // funcion que acompana la primera derivada en la ED
    p = function;
}

void FiniteDiff::setQ(double (*function)(double)){
    // funcion que acompana la derivada de orden cero en la ED
    q = function;
}

void FiniteDiff::setR(double (*function)(double)){
    // funcion que no va a acompanada de y en la ED
    r = function;
}
```

Método de D.F.

Estructura del código

FiniteDiff

```
methods (public):
void setInterval(double, double);
void setBoundaryCond(double, double);
void setP(double (*)(double));
void setQ(double (*)(double));
void setR(double (*)(double));
void solve(double *, double *);

(private):
void setH(void);
void calcule_EqSys(double *);
void croutFactorization(double *, double *);
```

```
void FiniteDiff::calcule_EqSys(double *x){
    // funcion que crea los elementos de matriz linealizando el problema
    x[0] = a;

    // ------ step 1 ----------------
    x[1] = a + h;
    A.push_back( 2  + h*h * q(x[1]));  // A[1]
    B.push_back( -1  + 0.5 * h * p(x[1]));  // B[1]
    D.push_back( -1 * h*h * r(x[1]) + (1 + 0.5 * h * p(x[1])) * alpha );  // D[1]

    // ------ step 2 ----------------
    for (int i = 2; i <= n-1 ; i++){
        x[i] = a + i * h;
        A.push_back( 2  + h*h * q(x[i]));  // empieza en A[2]
        B.push_back( -1  + 0.5 * h * p(x[i]));  // empieza en B[2]
        C.push_back( -1  - 0.5 * h * p(x[i]));  // empieza en C[1] - termina en C[n-2]
        D.push_back( - h*h * r(x[i]));  // empieza en D[2] - termina en D[n-1]
    }

    // ------ step 3 ----------------
    x[n] = b - h;
    A.push_back( 2  + h*h * q(x[n]));  // A[n]
    C.push_back( -1  - 0.5 * h * p(x[n]));  // C[n-1]
    D.push_back( - h*h * r(x[n]) + (1 - 0.5 * h * p(x[n])) * beta );  // D[n]
    x[n+1] = b;
}
```

Estructura del código

Método de D.F.

FiniteDiff

```
methods (public):
void setInterval(double, double);
void setBoundaryCond(double, double);
void setP(double (*)(double));
void setQ(double (*)(double));
void setR(double (*)(double));
void solve(double *, double *);

(private):
void setH(void);
void calcule_EqSys(double *);
void croutFactorization(double *, doub
```

```
void FiniteDiff::croutFactorization(double *x, double *w){
    // resuelve un sistema lineal tridiagonal

    int i;
    double L[n+1];
    double U[n+1];
    double Z[n+1];
    // ------ step 4 ----------------
    L[1] = A.at(1);
    U[1] = B.at(1) / A.at(1);
    Z[1] = D.at(1) / L[1];

    // ------ step 5 ----------------
    for (i = 2; i <= n-1 ; i++){
        L[i] =  A.at(i) - C.at(i-1) * U[i-1];
        U[i] =  B.at(i) / L[i];
        Z[i] = ( D.at(i) - C.at(i-1) * Z[i-1]) / L[i] ;
    }

    // ------ step 6 ----------------
    L[n] = A.at(n) - C.at(n-1) * U[n-1];
    Z[n] = ( D.at(n) - C.at(n-1) * Z[n-1]) / L[n];

    // ------ step 7 ----------------
    w[0] = alpha;
    w[n] = Z[n];
    w[n+1] = beta;

    // ------ step 8 ----------------
    for (i = n-1; i >= 1 ; i--){
        w[i] = Z[i] - U[i] * w[i+1];
    }
}
```

Método de D.F.

Ejemplo de testeo

$$y'' + 8y' + 16y = 4\,sin(4x)$$

$$y'' = \underbrace{-8}_{p(x)}\, y' \underbrace{-16}_{q(x)}\, y + \underbrace{4\,sin(4x)}_{r(x)}$$

$$y(0) = 0\,,\ y\left(\frac{\pi}{8}\right) = 0$$

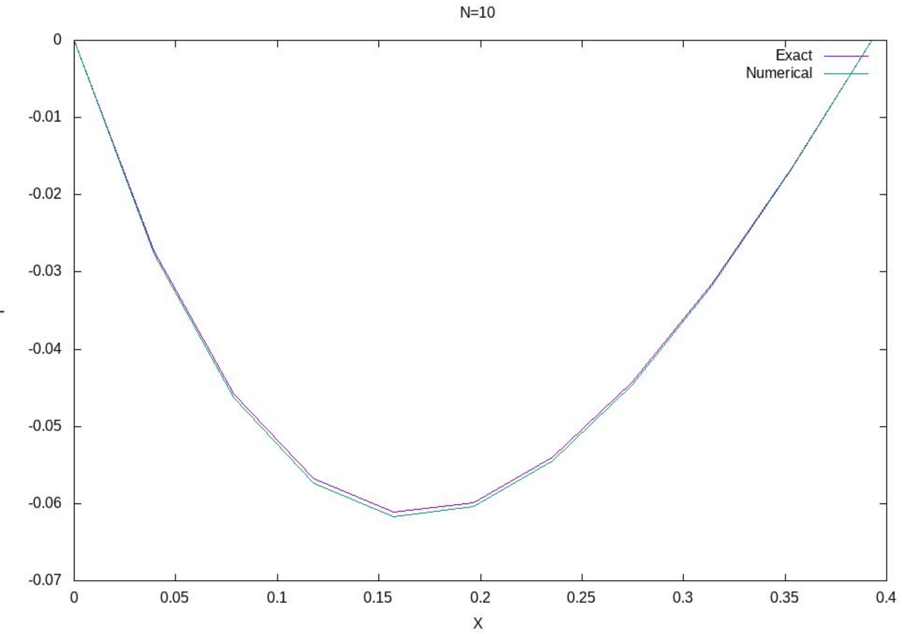
$$y(x) = \frac{1}{8}\,e^{-4x} - \frac{x}{\pi}\,e^{-4x} - \frac{1}{8}\,cos(4 * x)$$

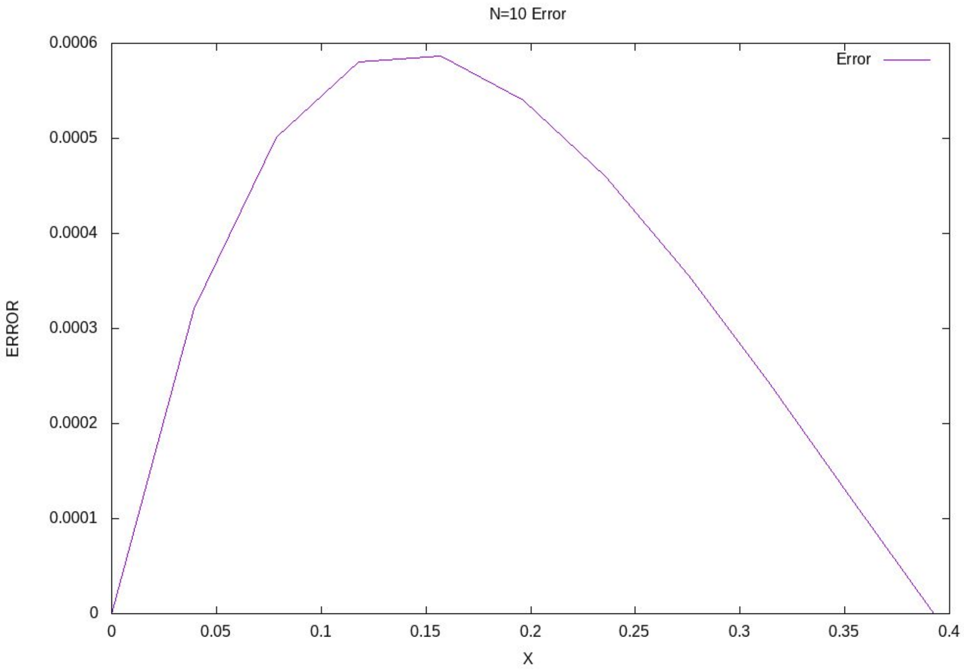
```
int N = 100;
double xmin = 0, xmax = 0.5 * M_PI_4;
double ymin = 0, ymax = 0;
FiniteDiff FDmethod(N);
double x[N+1], y_fdSol[N+1];
```

```
FDmethod.setInterval(xmin, xmax);
FDmethod.setBoundaryCond(ymin,ymax);
FDmethod.setP(p);
FDmethod.setQ(q);
FDmethod.setR(r);

FDmethod.solve(x, y_fdSol);
```

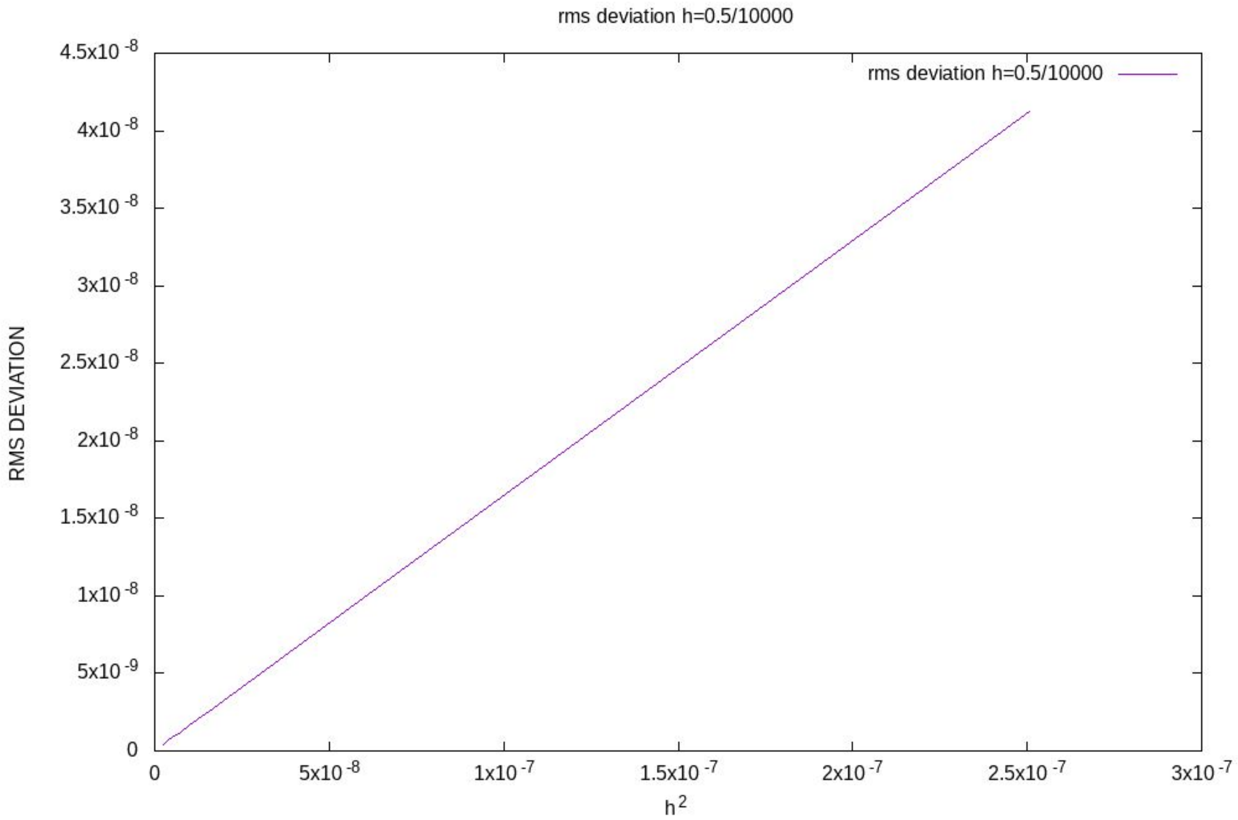
Ejemplo de testeo



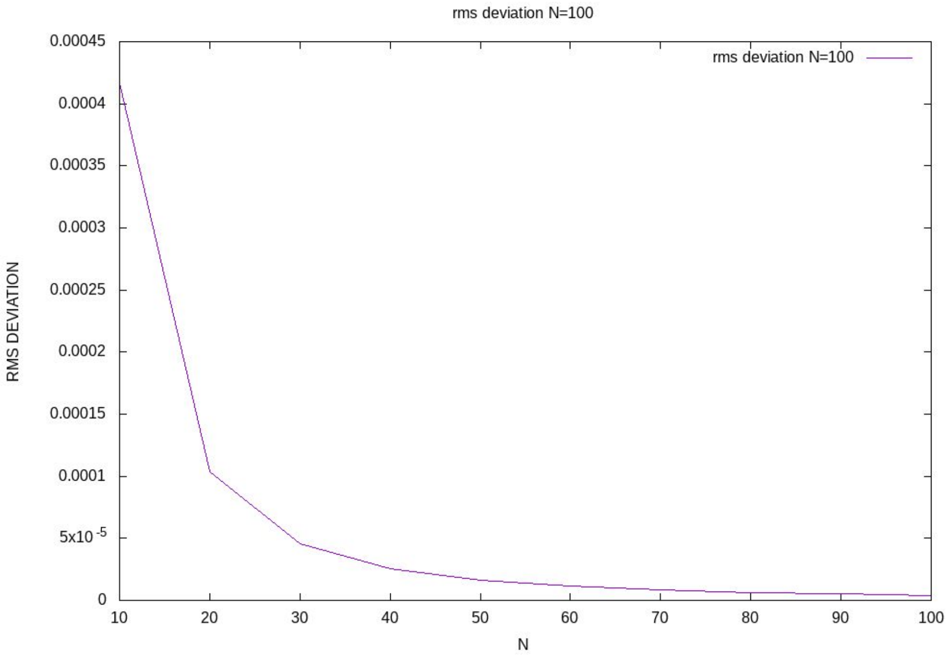


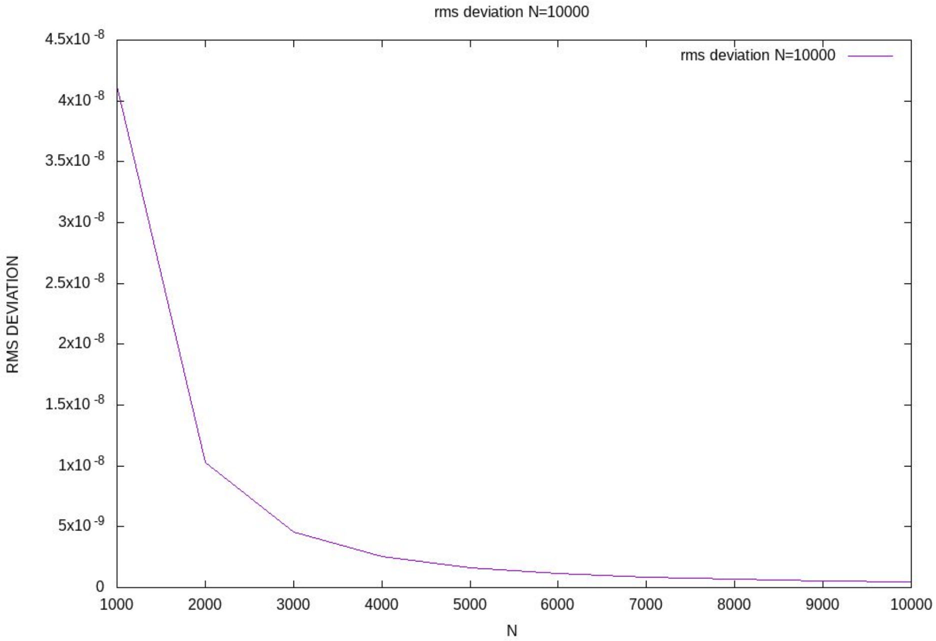
Método de D.F.

Ejemplo de testeo



Ejemplo de testeo

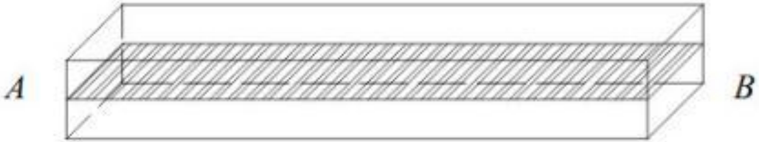


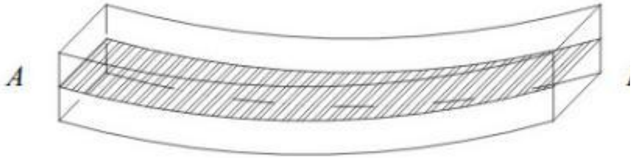


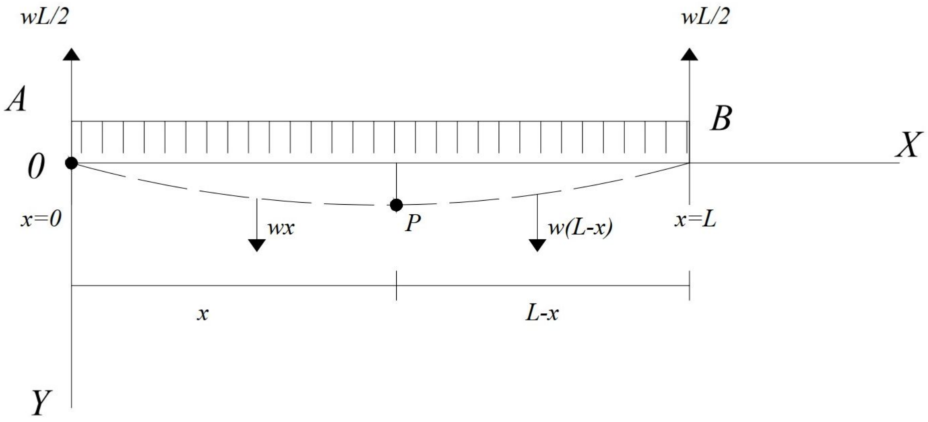
Aplicación física

Método de D.F.

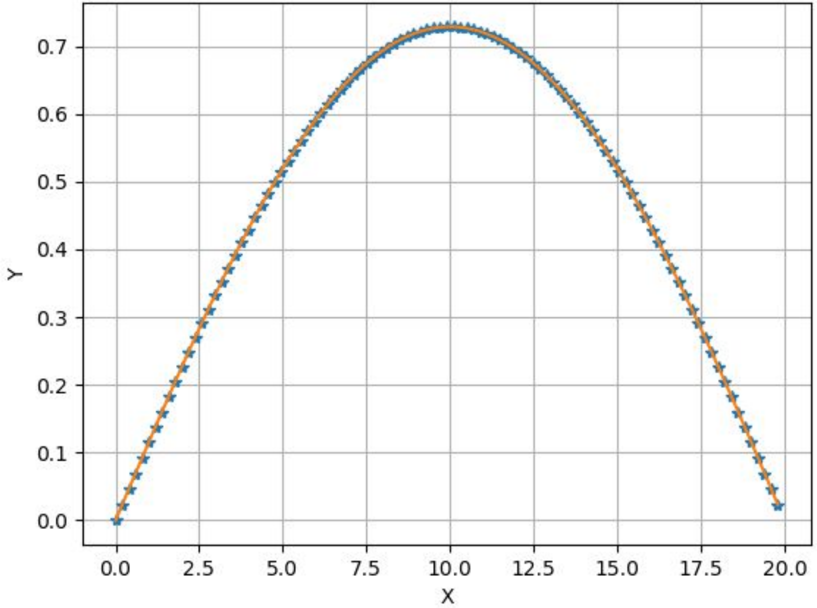
Sistema físico: Deflexión de una viga.

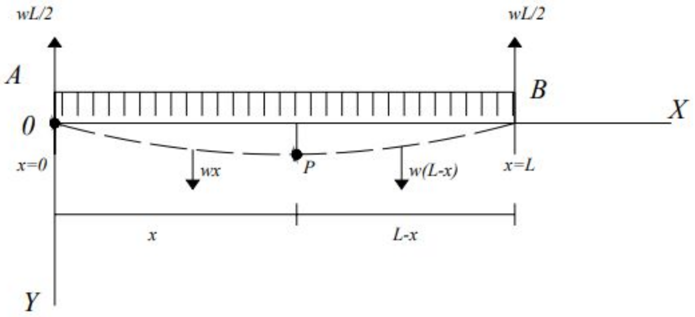






$$EI\frac{y''}{\left[1+(y')^2\right]^{3/2}} = M(x)$$





$$M(x) = w(L-x)\left(\frac{L-x}{2}\right) - (L-x)\frac{wL}{2} = \frac{wx^2}{2} - \frac{wLx}{2}$$

$$y = \frac{w}{24EI}(x^4 - 2Lx^3 + L^3x)$$

Repositorio de trabajo

https://github.com/DanielEstrada971102/Parcial3_FDmethod