Método de Rayleigh-Ritz

Andrés Santiago Duque Escobar
Valentina Roquemen Echeverry

Física Computacional II - 2020-2

TABLA DE CONTENIDOS

1. Introducción
   Ecuación diferencial y sus condiciones
2. Método de Rayleigh-Ritz
   Un problema variacional
3. Base lineal a tramos
4. Base cúbica spline
5. Ejemplo
6. Aspectos importantes

Link al repositorio: https://github.com/vlt-ro/Parcial3_Rayleigh-Ritz

Introducción

Problema: Solución de la ecuación diferencial con condiciones de frontera.

$$-\frac{d}{dx}\left(p(x)\frac{dy}{dx}\right) + q(x)y = f(x)\ , \ \text{para} \quad 0 \leqslant x \leqslant 1$$

Condiciones de frontera: $y(0) = y(1) = 0$

Introducción

Condiciones para las funciones q(x), f(x) y p(x):

- $p \in C^1[0,1]$
- $q, f \in C[0,1]$
- $q(x) \geqslant 0 \quad \forall x \in [0,1]$
- $\exists \delta > 0 : p(x) \geqslant \delta \quad \forall x \in [0,1]$

Se garantiza que el problema tiene solución única.

Método de Rayleigh-Ritz: Problema variacional

Teorema: Sea p(x), q(x) y f(x) funciones con las condiciones ya expuestas.

La función $y(x) \in C^2[0,1]$ tiene solución única a la ecuación diferencial:

$$-\frac{d}{dx}(p(x)\frac{dy}{dx}) + q(x)y = f(x) \text{ , para } 0 \leqslant x \leqslant 1$$

si y sólo si $y(x)$ es la única función en $C^2[0,1]$ que minimiza la integral

$$I[u] = \int_0^1 dx \, \{p(x)[u'(x)]^2 + q(x)[u(x)]^2 - 2f(x)u(x)\}$$

$u(x) \in C^2[0,1]$

Método de Rayleigh-Ritz: Problema variacional

- Se minimiza la integral sobre un conjunto de n funciones linealmente independientes que cumplen que:

$$\phi_i(1) = \phi_i(0) = 0 \qquad\qquad i = 1, 2, ..., n$$

- Se define $\phi(x) = \sum_{i=1}^{n} c_i \phi_i(x)$ como una solución aproximada a la función y(x).

Método de Rayleigh-Ritz: Problema variacional

Objetivo: Encontrar los coeficientes c_i que minimizan la integral $I[\sum_{i=1}^{n} c_i\phi_i(x)]$.

- Este mínimo ocurre cuando $\frac{\partial I}{\partial c_j} = 0$ para cada $j = 1, ..., n$.

- De la diferenciación se obtiene que:

$$0 = \sum_{i=1}^{n} [\int_0^1 \{p(x)\phi_i'(x)\phi_j'(x) + q(x)\phi_i(x)\phi_j(x)\} \, dx] c_i - \int_0^1 f(x)\phi_j(x) dx$$

Método de Rayleigh-Ritz: Problema variacional

Se construye un sistema lineal de ecuaciones nxn de la forma:

A**c**=**b**

- A es una matriz simétrica con entradas:

$$a_{ij} = \int_0^1 [p(x)\phi_i'(x)\phi_j'(x) + q(x)\phi_i(x)\phi_j(x)]dx$$

- **b** se define como:

$$b_i = \int_0^1 f(x)\phi_i(x)dx$$

- **c** es el vector que queremos encontrar.

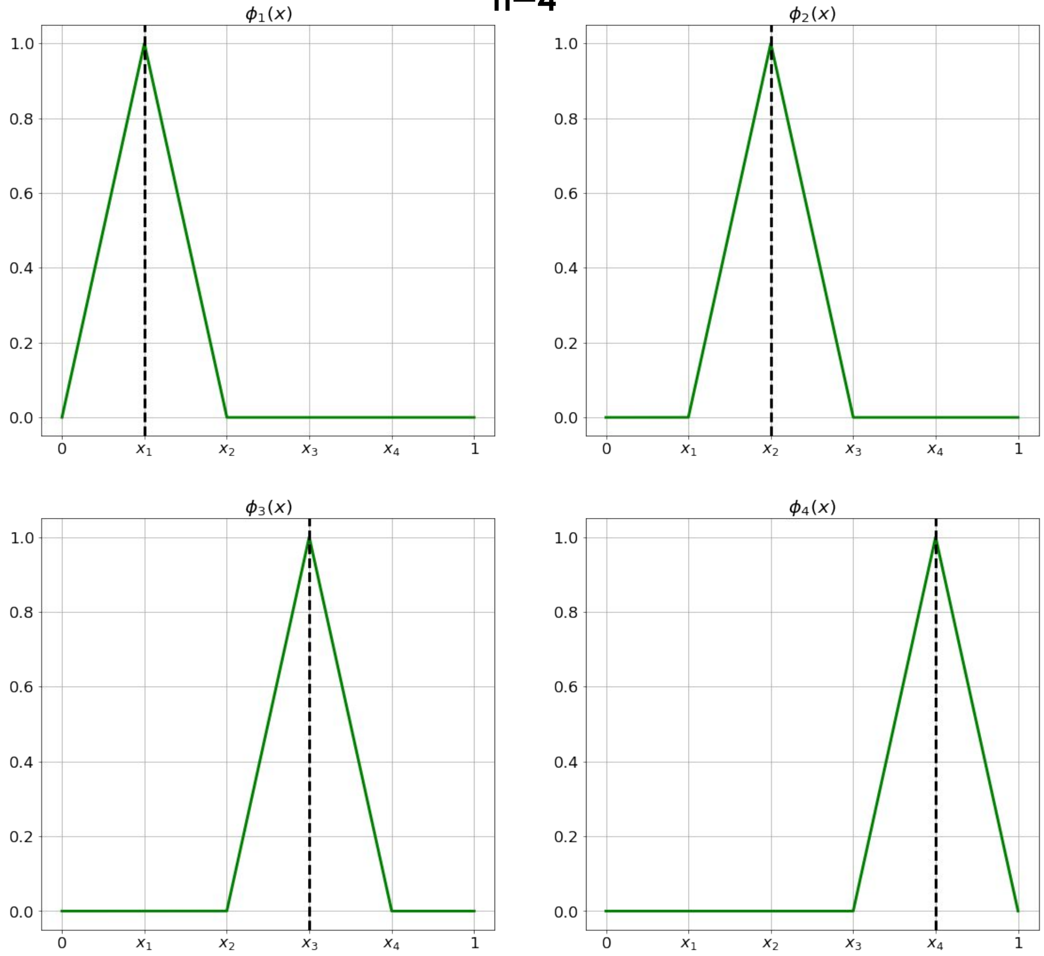
Entonces, ¿qué hacemos ahora?

Elegir una base

- $\boxed{\text{Base lineal a tramos.}}$
- Base cúbica spline.

Base lineal a tramos

n=4



Base lineal a tramos

Sea: $0 = x_0 < x_1 < \ldots < x_n < x_{n+1} = 1$

$h_i = x_{i+1} - x_i$ para cada $i = 1, 2, \ldots, n$

Definimos:

$$\phi_i(x) = \begin{cases} 0, & 0 \leqslant x \leqslant x_{i-1} \\ \frac{1}{h_{i-1}}(x - x_{i-1}), & x_{i-1} < x \leqslant x_i \\ \frac{1}{h_i}(x_{i+1} - x), & x_i < x \leqslant x_{i+1} \\ 0, & x_{i+1} < x \leqslant 1 \end{cases}$$

Base lineal a tramos

Note que: $\phi_i(x)$ y $\phi_i'(x)$ son diferentes de 0 solo en (x_{i-1}, x_{i+1})

Consecuencia: $\phi_i(x)\phi_j(x) \equiv 0$ y $\phi_i'(x)\phi_j'(x) \equiv 0$

excepto cuando j es $i-1, i$ o $i+1$.

$$A = \begin{bmatrix} a_{11} & a_{12} & 0 & . & . & . & & 0 \\ a_{21} & a_{22} & a_{23} & 0 & & & & . \\ 0 & a_{32} & a_{33} & a_{34} & 0 & & & . \\ . & 0 & a_{43} & a_{44} & . & . & & . \\ . & & 0 & . & . & . & . & \\ . & & & . & . & . & . & . \\ & & & & . & . & . & . \\ 0 & . & . & . & & . & . & a_{nn} \end{bmatrix}$$

Factorización Crout para sistemas lineales tridiagonales.

Base lineal a tramos

There are six types of integrals to be evaluated:

$$Q_{1,i} = \left(\frac{1}{h_i}\right)^2 \int_{x_i}^{x_{i+1}} (x_{i+1} - x)(x - x_i)q(x)\,dx, \quad \text{for each } i = 1, 2, \ldots, n-1,$$

$$Q_{2,i} = \left(\frac{1}{h_{i-1}}\right)^2 \int_{x_{i-1}}^{x_i} (x - x_{i-1})^2 q(x)\,dx, \quad \text{for each } i = 1, 2, \ldots, n,$$

$$Q_{3,i} = \left(\frac{1}{h_i}\right)^2 \int_{x_i}^{x_{i+1}} (x_{i+1} - x)^2 q(x)\,dx, \quad \text{for each } i = 1, 2, \ldots, n,$$

$$Q_{4,i} = \left(\frac{1}{h_{i-1}}\right)^2 \int_{x_{i-1}}^{x_i} p(x)\,dx, \quad \text{for each } i = 1, 2, \ldots, n+1,$$

$$Q_{5,i} = \frac{1}{h_{i-1}} \int_{x_{i-1}}^{x_i} (x - x_{i-1}) f(x)\,dx, \quad \text{for each } i = 1, 2, \ldots, n,$$

and

$$Q_{6,i} = \frac{1}{h_i} \int_{x_i}^{x_{i+1}} (x_{i+1} - x) f(x)\,dx, \quad \text{for each } i = 1, 2, \ldots, n.$$

Ejemplo:

$$b_i = \int_0^1 f(x)\phi_i(x)\,dx$$

$$= \frac{1}{h_{i-1}} \int_{x_{i-1}}^{x_i} (x - x_{i-1}) f(x)\,dx + \frac{1}{h_i} \int_{x_i}^{x_{i+1}} (x_{i+1} - x) f(x)\,dx,$$

$$= Q_{5,i} + Q_{6,i}, \quad \text{for each } i = 1, 2, \ldots, n.$$

Número de integrales a resolver = 6n-1

Base lineal a tramos

Algoritmo 11.5 (Burden, 2010)

To approximate the solution to the boundary-value problem

$$-\frac{d}{dx}\left(p(x)\frac{dy}{dx}\right) + q(x)y = f(x), \quad \text{for } 0 \le x \le 1, \text{ with } y(0) = 0 \text{ and } y(1) = 0$$

with the piecewise linear function

$$\phi(x) = \sum_{i=1}^{n} c_i\phi_i(x) :$$

INPUT integer $n \ge 1$; points $x_0 = 0 < x_1 < \cdots < x_n < x_{n+1} = 1$.

OUTPUT coefficients $c_1, \ldots, c_n$.

Base lineal a tramos

```cpp
class LinearRayleighRitz: public RayleighRitz
{
class LBasis
{
public:
    /**
     * @brief Constructor para inicializar los parametros
     *        de un elemento de la base específico.
     * @param x_im1 Punto x_{i-1}
     * @param x_i   Punto x_i
     * @param x_ip1 Punto x_{i+1}
     */
    LBasis(double x_im1, double x_i,double x_ip1);

    /**
     * @brief Sobrecarga del operator () para usar los obj
     *        esta clase como "functors"
     * @param x Punto en el cual se evaluará la función ba
     * @note Ecuación 11.30
     */
    double operator()(double x);

private:
    /* Puntos que definen la función por partes */
    double x_im1, x_i, x_ip1;
};

    std::vector<LBasis> &getBasis();
```

```cpp
double LBasis::operator()(double x)
{
    double result = 0;

    if( x_im1 < x && x <= x_i )
    {
        double h = x_i - x_im1;
        result = (x - x_im1) / h;
    }
    else if(x_i < x && x < x_ip1)
    {
        double h = x_ip1 - x_i;
        result = (x_ip1 - x) / h;
    }

    return result;
}
```

Base lineal a tramos

```
class LinearRayleighRitz: public RayleighRitz
{
pu
```

```
double LinearRayleighRitz::Q1(int i, std::vector<double>& x, double(*q)(double))
{
    auto f = [&x,q,i](float _x)->float{return q(_x)*(x[i+1] - _x)*(_x - x[i]);};
    auto r = simpson_rule(x[i],x[i+1],10,f);
    return r/(h[i]*h[i]);
}
```

```
     *        con condiciones de frontera y(0)=y(1)=0
     * @return Coeficientes de expansión.
     */
    std::vector<double> solve(double(*p)(double),
                              double(*q)(double),
                              double(*f)(double));

    /**
     * @brief Expande la funcion y(x) en la base.
     * @param x Punto en el cual se evaluara la funcion base
     */
    double eval(double x);

    /**
     * @brief Retorna la base que expande la aproximación de la
     * solución.
     */
    std::vector<LBasis> &getBasis();
```

```
+1} - x_i
```

```
        /*
         * Integrales a resolver (página 700)
         */
        double Q1(int , std::vector<double>& , double(*)(double));
        double Q2(int , std::vector<double>& , double(*)(double));
        double Q3(int , std::vector<double>& , double(*)(double));
        double Q4(int , std::vector<double>& , double(*)(double));
        double Q5(int , std::vector<double>& , double(*)(double));
        double Q6(int , std::vector<double>& , double(*)(double));
};
```

Base lineal a tramos

```cpp
class LinearRayleighRitz: public RayleighRitz
{
public:

    /**
     * @param x Recibe vector con los nodos.
     */
    LinearRayleighRitz(std::vector<double>& x);

    /**
     * @brief Resuelve de forma aproximada la EDO
     *         -D( p(x) D(y) ) + q(x)y = f(x)
     *         con condiciones de frontera y(0)=y(1)=0
     * @return Coeficientes de expansión.
     */
    std::vector<double> solve(double(*p)(double),
                              double(*q)(double),
                              double(*f)(double));

    /**
     * @brief Expande la funcion y(x) en la base.
     * @param x Punto en el cual se evaluara la funcion base
     */
    double eval(double x);

    /**
     * @brief Retorna la base que expande la aproximación de la
     * solución.
     */
    std::vector<LBasis> &getBasis();

```

```cpp

private:
    std::vector<double> h; // Coeficientes h_i = x_{i+1} - x_i
    std::vector<LBasis> phi; // Base lineal a trozos
    std::vector<double> c; //Coeficientes de expansión
    std::vector<double>& x;

    /*
     * Integrales a resolver (página 700)
     */
    double Q1(int , std::vector<double>& , double(*)(double));
    double Q2(int , std::vector<double>& , double(*)(double));
    double Q3(int , std::vector<double>& , double(*)(double));
    double Q4(int , std::vector<double>& , double(*)(double));
    double Q5(int , std::vector<double>& , double(*)(double));
    double Q6(int , std::vector<double>& , double(*)(double));
};
```

Base lineal a tramos

```cpp
std::vector<double> LinearRayleighRitz::solve(double(*p)(double),
                                               double(*q)(double),
                                               double(*f)(double))
{
    int n = x.size() - 2;

    /*******************************************/
    /* Step 1: Cálculo de los coeficientes h_i */
    /*******************************************/
    for(int i=0; i<=n; ++i)
        h.push_back(x[i+1]-x[i]);

    /*************************************************/
    /* Step 2: Definir la base lineal por tramos phi_i */
    /*************************************************/
    for(int i=1; i <= n; ++i)
        phi.push_back(LBasis(x[i-1],x[i], x[i+1]));
```

```cpp
/**************************************/
/* Step 3: Calculo de las integrales */
/**************************************/

/* Aunque es un desperdicio de recursos, se va a
 * definir la posición 0 de los vectores que almacenan
 * el resultado de las integrales, con el fin de seguir
 * el algoritmo y no tener problemas con los índices usados
 * en el libro. La idea es facilitar la lectura del código
 * y la comparación con el libro. */
vector<double> q1(1),q2(1),q3(1),q4(1),q5(1),q6(1);
for(int i=1; i<=n; ++i)
{
    if(i<n)
        q1.push_back(Q1(i,x,q));
    q2.push_back(Q2(i,x,q));
    q3.push_back(Q3(i,x,q));
    q4.push_back(Q4(i,x,p));
    q5.push_back(Q5(i,x,f));
    q6.push_back(Q6(i,x,f));
}
q4.push_back(Q4(n+1,x,p));
```

Base lineal a tramos

```
/*********************************************************************/
/* Step 4 and 5: Calculo de los factores alpha_i, beta_i y b_i */
/*********************************************************************/
// Se define el primer elemento, por la misma razón que
// el valor de las integrales
vector<double> alpha(1), beta(1), b(1);
for(int i=1; i<=n; ++i)
{
    alpha.push_back(q4[i] + q4[i+1] + q2[i] + q3[i]);
    b.push_back(q5[i] + q6[i]);

    if(i < n)
        beta.push_back(q1[i] - q4[i+1]);
}
```

```
/*********************************************************************/
/* Step 6, 7 and 8: Calculo de los factores a_i, zeta_i y z_i */
/*********************************************************************/
vector<double> a(2), zeta(2), z(2);
a[1] = alpha[1];
zeta[1] = beta[1] / alpha[1];
z[1] = b[1] / a[1];

for(int i=2; i<=n; ++i)
{
    a.push_back(alpha[i] - beta[i-1]*zeta[i-1]);
    z.push_back( (b[i] - beta[i-1]*z[i-1])/a[i] );

    if(i<n)
        zeta.push_back(beta[i]/a[i]);
}

/*****************************************************/
/* Step 9 and 10: Calculo de los factores c_i */
/*****************************************************/
/* Coeficientes de la expansión en series*/
vector<double> c(n); //c_i con i=0,...,n-1
c[n-1] = z[n];
for(int i=n-1; i>0; --i)
    c[i-1] = z[i] - zeta[i]*c[i];
this->c = c;
return c;
```

Base lineal a tramos

```cpp
class LinearRayleighRitz: public RayleighRitz
{
public:

    /**
     * @param x Recibe vector con los nodos.
     */
    LinearRayleighRitz(std::vector<double>& x);

    /**
     * @brief Resuelve de forma aproximada la EDO
     *        -D( p(x) D(y) ) + q(x)y = f(x)
     *        con condiciones de frontera y(0)=y(1)=0
     * @return Coeficientes de expansión.
     */
    std::vector<double> solve(double(*p)(double),
                              double(*q)(double),
                              double(*f)(double));

    /**
     * @brief Expande la funcion y(x) en la base.
     * @param x Punto en el cual se evaluara la funcion base
     */
    double eval(double x);

    /**
     * @brief Retorna la base que expande la aproximación de la
     * solución.
     */
    std::vector<LBasis> &getBasis();
```

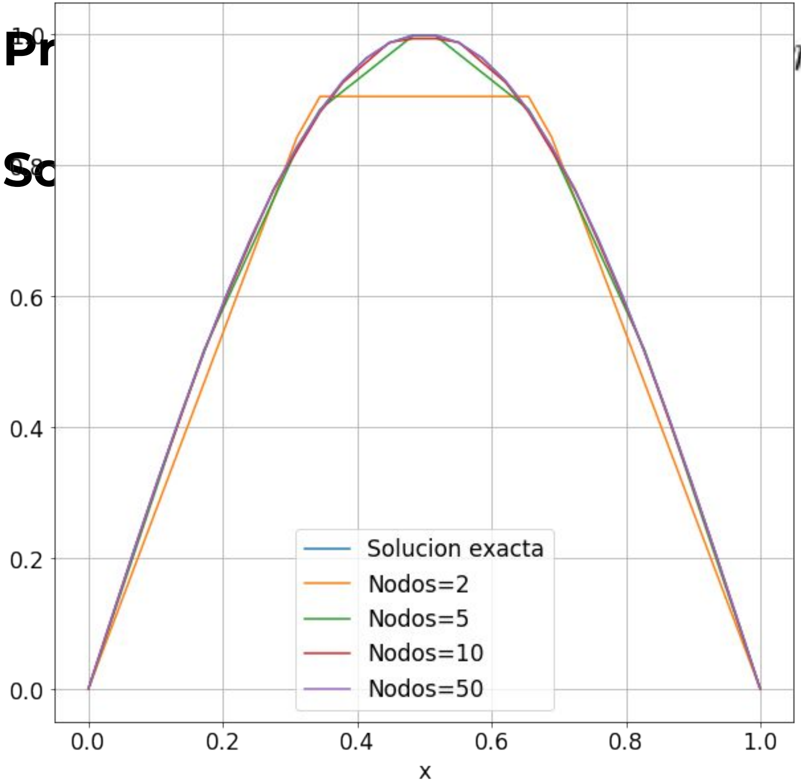
```cpp
double LinearRayleighRitz::eval(double x)
{
    double rs = 0;

    for(size_t i=0; i<c.size(); ++i)
        rs += c[i] * phi[i](x);

    return rs;
}
```

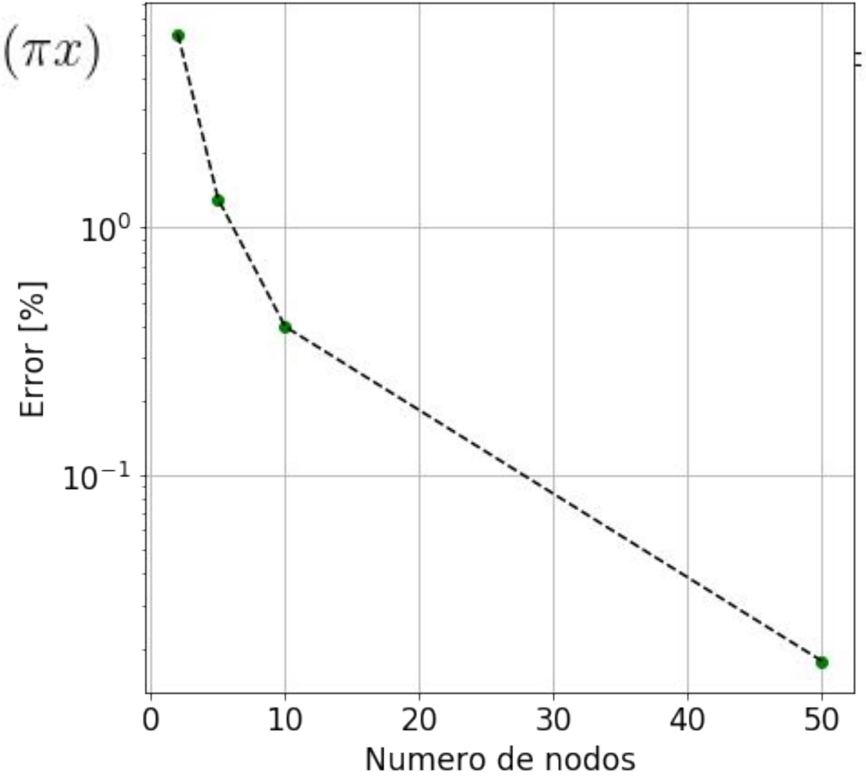
Base lineal a tramos

Pr

So

$n(\pi x)$ $= 0$





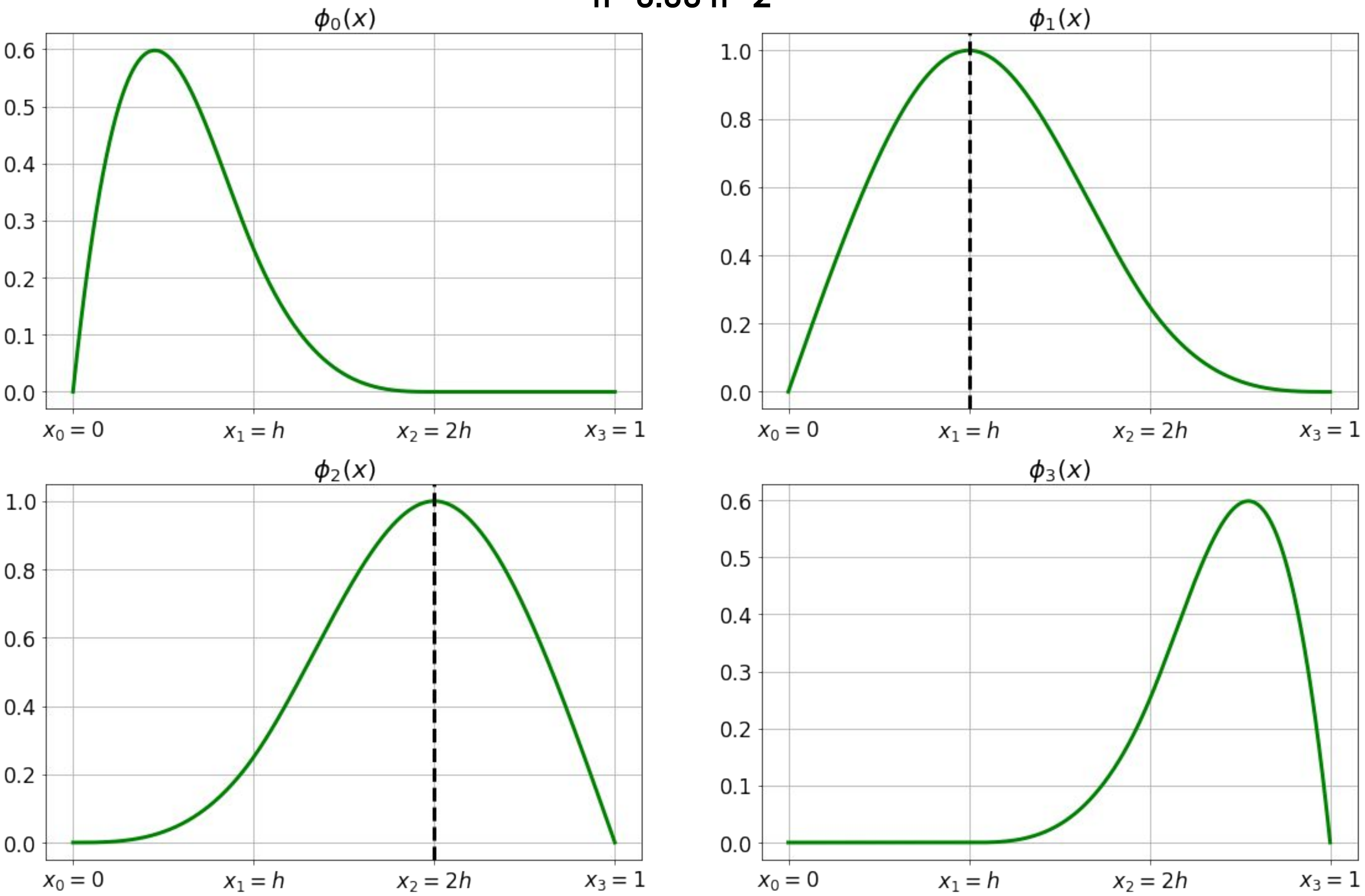
Entonces, ¿qué hacemos ahora?

Elegir una base

- Base lineal a tramos. ✔
- Base cúbica spline.

Base cúbica spline

h=0.33 n=2



Base cúbica spline

Sea:

$$h = \frac{1}{n+1}$$

Definimos:

$$S(x) = \begin{cases} 0, & x \leqslant -2 \\ \frac{1}{4}(2+x)^3, & -2 < x \leqslant -1 \\ \frac{1}{4}[(2+x)^3 - (1+x)^3], & -1 < x \leqslant 0 \\ \frac{1}{4}[(2-x)^3 - (1-x)^3], & 0 < x \leqslant 1 \\ \frac{1}{4}(2-x)^3, & 1 < x \leqslant 2 \\ 0, & 2 < x \end{cases} \qquad \phi_i(x) = \begin{cases} S(\frac{x}{h}) - 4S(\frac{x+h}{h}), & i = 0 \\ S(\frac{x-h}{h}) - S(\frac{x+h}{h}), & i = 1 \\ S(\frac{x-ih}{h}), & 2 \leqslant i \leqslant n-1 \\ S(\frac{x-nh}{h}) - S(\frac{x+(n+2)h}{h}), & i = n \\ S(\frac{x-(n+1)h}{h}) - 4S(\frac{x+(n+2)h}{h}), & i = n+1 \end{cases}$$

Base cúbica spline

Note que: $\phi_i(x)$ y $\phi_i'(x)$ son diferentes de 0 para $x \in [x_{i-2}, x_{i+2}]$

Consecuencia: Matriz de banda con ancho de banda a lo sumo de 7.

$$A = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} & 0 & \cdots & & & 0 \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} & & & & \vdots \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & & & \vdots \\ a_{30} & a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & & & & & & a_{n-2,n+1} \\ \vdots & & & \ddots & & & & & a_{n-1,n+1} \\ \vdots & & & & \ddots & & & & a_{n,n+1} \\ 0 & \cdots & & & 0 & a_{n+1,n-2} & a_{n+1,n-1} & a_{n+1,n} & a_{n+1,n+1} \end{bmatrix}$$

Reducción de Gauss-Jordan.

Número de integrales a resolver para n>3 = 5n-6

Base cúbica spline

Algoritmo 11.6 (Burden, 2010)

To approximate the solution to the boundary-value problem

$$-\frac{d}{dx}\left(p(x)\frac{dy}{dx}\right) + q(x)y = f(x), \quad \text{for } 0 \leq x \leq 1, \text{ with } y(0) = 0 \text{ and } y(1) = 0$$

with the sum of cubic splines

$$\phi(x) = \sum_{i=0}^{n+1} c_i \phi_i(x) :$$

INPUT integer $n \geq 1$.

OUTPUT coefficients $c_0, \ldots, c_{n+1}$.

Base cúbica spline

```cpp
class csBasis
{
public:

    /**
     * @brief Constructor para inicializar un
     *        elemento de la base
     */
    csBasis(int i, int n, double h);

    /**
     * Sobrecarga del constructor de copia
     */
    csBasis(const csBasis &);

    ~csBasis();

    double operator()(double x);

    double dPhi(double);

private:
    Phi *phi;
    int i;
    int n;
    double h;

    void setMembers(int i, int n, double h);
};
```

```cpp
void csBasis::setMembers(int i, int n, double h)
{
    this->i = i;
    this->n = n;
    this->h = h;

    /* Seleccionar un elemento de la base específico */
    if(i==0)
        phi = new Phi_0(h);
    else if(i==1)
        phi = new Phi_1(h);
    else if(i<n)
        phi = new Phi_i(h,i);
    else if(i==n)
        phi = new Phi_n(h,n);
    else if(i==(n+1))
        phi = new Phi_n_1(h,n);
    else
        throw "Error: Parametros invalidos";
}
```

bspline.h

Base cúbica spline

```
class csRayleighRitz : public RayleighRitz
{
public:
    csRayleighRitz(std::size_t n);

    /**
     * @brief Resuelve de forma aproximada la EDO
     *        -D( p(x) D(y) ) + q(x)y = f(x)
     *        con condiciones de frontera y(0)=y(1)=0
     * @return Coeficientes de expansión.
     */
    vector<double> solve(double(*p)(double),
                         double(*q)(double),
                         double(*f)(double));

    /**
     * @brief Expande la funcion y(x) en la base.
     * @param x Punto en el cual se evaluara la funcion base
     */
    double eval(double);

    /**
     * @brief Retorna la base que expande la aproximación de la
     * solución.
     */
    vector<csBasis>& getBasis();
```

```
private:
    vector<csBasis> phi; //B-Spline basis
    vector<double> c; //Coeficientes de expansión
    std::size_t n;
    double h;

    /**
     * Retorna el valor del j-iesimo nodo
     */
    double x_i(int i);
};
```

Base cúbica spline

```
vector<double> csRayleighRitz::solve(dou
                                     double(*q)(double),
                                     double(*f)(double))
{
    vector<vector<double>> A(n+2,vector<
    vector<double> b(n+2); // Vector b d
    vector<double> c(n+2); // Vector c d


    /********************************
    /*   Step 1: Cálculo del coeficiente
    /********************************
    h = 1/static_cast<double>(1+n);

    // Los pasos 2 y 3 se representan co

    /********************************
    /* Step 4: Definición de los element
    /********************************
    for(unsigned i=0; i<= n+1; ++i)
        phi.push_back( csBasis(i,n,h));
```

```
csBasis::csBasis(int i,int n, double h): phi(nullptr)
{
    setMembers(i,n,h);
}

csBasis::csBasis(const csBasis &old): phi(nullptr)
{
    setMembers(old.i,old.n,old.h);
}

csBasis::~csBasis()
{
    // Liberar memória
    if(phi)
        delete phi;
}
```

Base cúbica spline

```
/****************************************/
/* Step 5-8: Calculo de las componentes*/
/* de la matriz A                      */
/****************************************/
for (int i = 0; i < n+2; ++i)
{
  for (int j = i; j <= std::min(i+3,int(n+1)); ++j)
  {
    double L = std::max(x_i(j-2), 0.);
    double U = std::min(x_i(i+2), 1.);

    A[i][j] =  simpson_rule( L, U, 100,[i,j,p,q,this](float x)->float
                        {return p(x)*this->phi[i].dPhi(x)*this->phi[j].dPhi(x)+
                                q(x)*this->phi[i](x)*this->phi[j](x);});

    if (i!=j) A[j][i] = A[i][j]; // La matriz es simetrica
  }

  if (i>=4) { for (int j = 0; j <= i-4; ++j) A[i][j] = 0;}
  if (i<=n-3) { for (int j = i+4; j <= n+1; ++j) A[i][j] = 0;}
}
```

Base cúbica spline

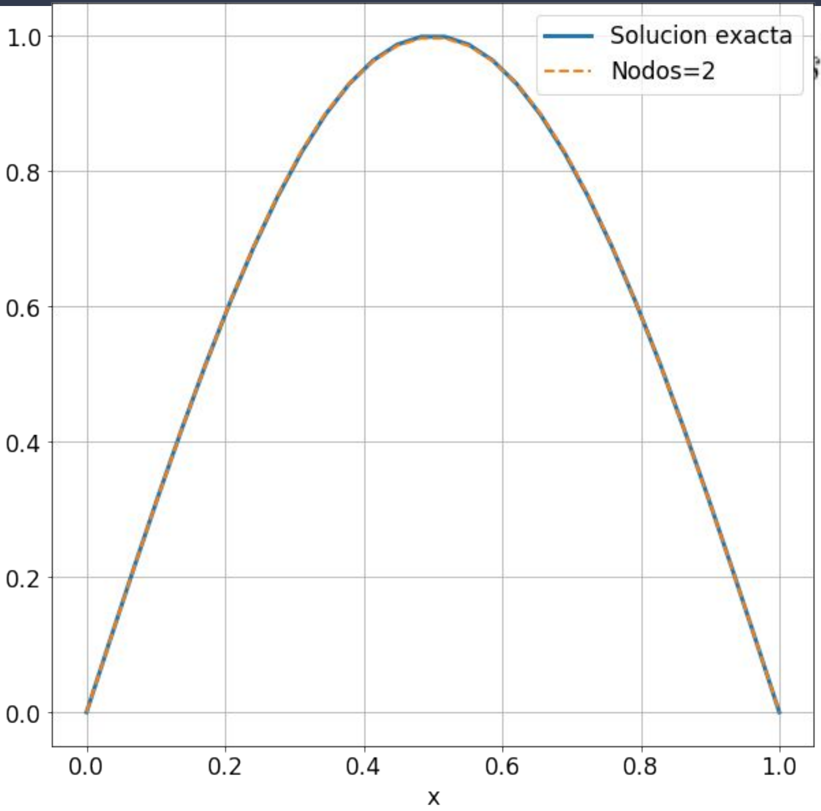
```
    /****************************************/
    /* Step 9: Calculo de las componentes del*/
    /* vector b                             */
    /****************************************/
    for (int i = 0; i < n+2; ++i)
    {
      double L = std::max(x_i(i-2),(double)(0));
      double U = std::min(x_i(i+2),(double)(1));

      b[i] = simpson_rule(L,U,100,[i,f,this](float x)->float {return f(x)*this->phi[i](x);});
    }

    /***************************************/
    /* Step 10: Se encuentra c, resolviendo */
    /* el sistema de ecuaciones            */
    /***************************************/
    gauss_jordan(n+2, A, b, c);

    this->c = c;
    return c;
}
```

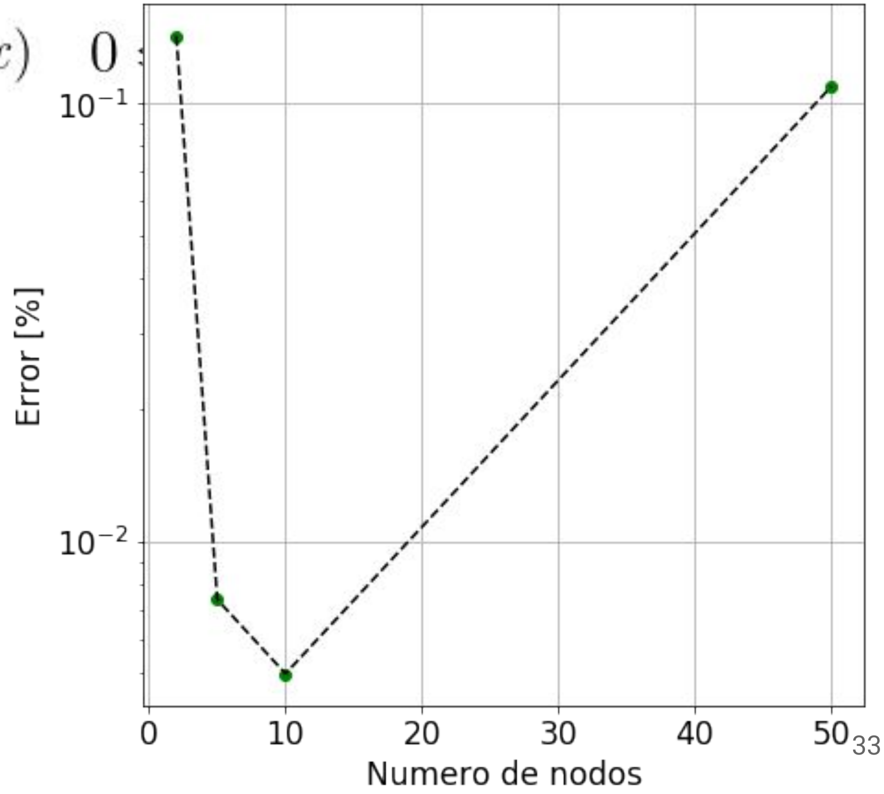
Base cúbica spline

```cpp
class csRayleighRitz : public RayleighRitz
{
public:
    csRayleighRitz(std::size_t n);

    /**
     * @brief Resuelve de forma aproximada la EDO
     *        -D( p(x) D(y) ) + q(x)y = f(x)
     *        con condiciones de frontera y(0)=y(1)=0
     * @return Coeficientes de expansión.
     */
    vector<double> solve(double(*p)(double),
                         double(*q)(double),
                         double(*f)(double));

    /**
     * @brief Expande la funcion y(x) en la base.
     * @param x Punto en el cual se evaluara la funcion base
     */
    double eval(double);

    /**
     * @brief Retorna la base que expande la aproximación de la
     * solución.
     */
    vector<csBasis>& getBasis();
```

```cpp
private:
    vector<csBasis> phi; //B-Spline basis
    vector<double> c; //Coeficientes de expansión
    std::size_t n;
    double h;

    /**
     * Retorna el valor del j-iesimo nodo
     */
    double x_i(int i);
};
```

Base cúbica spline





Método de Rayleigh-Ritz: Problema variacional

Teorema: Sea p(x), q(x) y f(x) funciones con las condiciones ya expuestas.

La función $y(x) \in C^2[0,1]$ tiene solución única a la ecuación diferencial:

$$-\frac{d}{dx}\left(p(x)\frac{dy}{dx}\right) + q(x)y = f(x)\text{, para } 0 \leqslant x \leqslant 1$$

si y sólo si $y(x)$ es la única función en $C^2[0,1]$ que minimiza la integral

$$I[u] = \int_0^1 dx\,\{p(x)[u'(x)]^2 + q(x)[u(x)]^2 - 2f(x)u(x)\}$$

Ejemplo

Partícula sometida a una fuerza de la forma $F(t) = Acos(\pi t)$

Encontrar x(t), sabiendo que x(0)=x(1)=0.

$$L = \frac{1}{2}m\dot{x}^2 + xF(t) \qquad\qquad S = \int_0^1 L dt$$

Recordemos: $$I[u] = \int_0^1 dx \left\{p(x)[u'(x)]^2 + q(x)[u(x)]^2 - 2f(x)u(x)\right\}$$

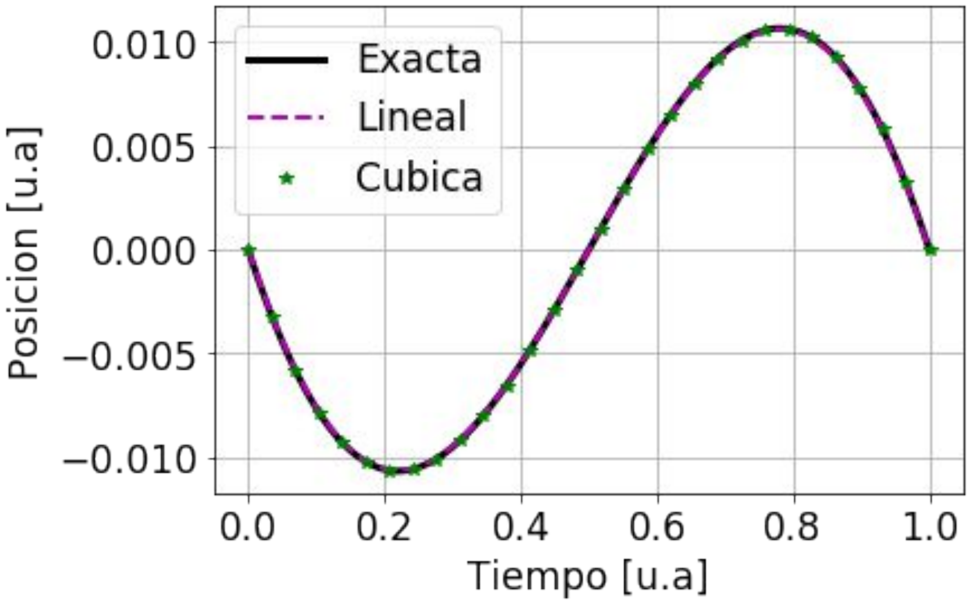
$$q(t) = 0 \qquad p(t) = \frac{m}{2} \qquad f(t) = \frac{-F(t)}{2} \qquad u(x) = x(t)$$

Ejemplo

Solución exacta:

$$x(t) = \frac{-A}{m\pi^2}(cos(\pi t) + 1)$$

A = 0.5 u.a.
m = 1 u.a.



Aspectos importantes

- Problemas en las aproximaciones del libro:

Approximations to the other integrals are derived in a similar manner and are given by

$$Q_{2,i} \approx \frac{h_{i-1}}{12}[3q(x_i) + q(x_{i-1})], \qquad Q_{3,i} \approx \frac{h_i}{12}[3q(x_i) + q(x_{i+1})],$$

$$Q_{4,i} \approx \frac{h_{i-1}}{2}[p(x_i) + p(x_{i-1})], \qquad Q_{5,i} \approx \frac{h_{i-1}}{6}[2f(x_i) + f(x_{i-1})],$$

and

$$Q_{6,i} \approx \frac{h_i}{6}[2f(x_i) + f(x_{i+1})].$$

The matrix A is positive definite (see Exercise 13), so the linear system $A\mathbf{c} = \mathbf{b}$ can be solved by Cholesky's Algorithm 6.[illegible] Gaussian elimination. Algorithm 11.6 details the

Gracias!

¿Alguna pregunta?