Raíces de ecuaciones
Métodos cerrados
Lección 05

Dr. Pablo Alvarado Moya

CE3102 Análisis Numérico para Ingeniería
Área de Ingeniería en Computadores
Tecnológico de Costa Rica

I Semestre 2018

Contenido

Funciones algebraicas y trascendentes

- Sea $p_n(x)$ un polinomio de orden n:

$$p_n(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$$

- Función $y = f(x)$ es **algebraica** si

$$p_n(x) y^n + p_{n-1}(x) y^{n-1} + \ldots + p_1(x) y + p_0(x) = 0$$

- Función es **transcendente** si **no** es algebraica
  - funciones trigonométricas
  - exponenciales
  - logarítmicas
  - etc.

Tipos de problemas

Dos tipos de problemas numéricos:

- Determinar raíces **reales** de ecuaciones algebraicas y trascendentes.
  - Parten de una posición inicial, y
  - Buscan **una** raíz
- Determinar **todas** las raíces reales y complejas de polinomios

Métodos gráficos

Solución de $y = f(x) = 0$ ó $f(x) = g(x)$

- Se grafica la función y se determina el valor de la variable visualmente.
- Método es aproximado, pero conceptualmente seguro.
- Permite detectar posibles errores de métodos numéricos.
- Permite detectar valores iniciales para métodos numéricos.

Ejemplo: graficación (1)

Ejemplo

Encuentre las raíces de

$$5\cos(\pi t) = \frac{1}{2}e^{-t}$$

Ejemplo: graficación (2)

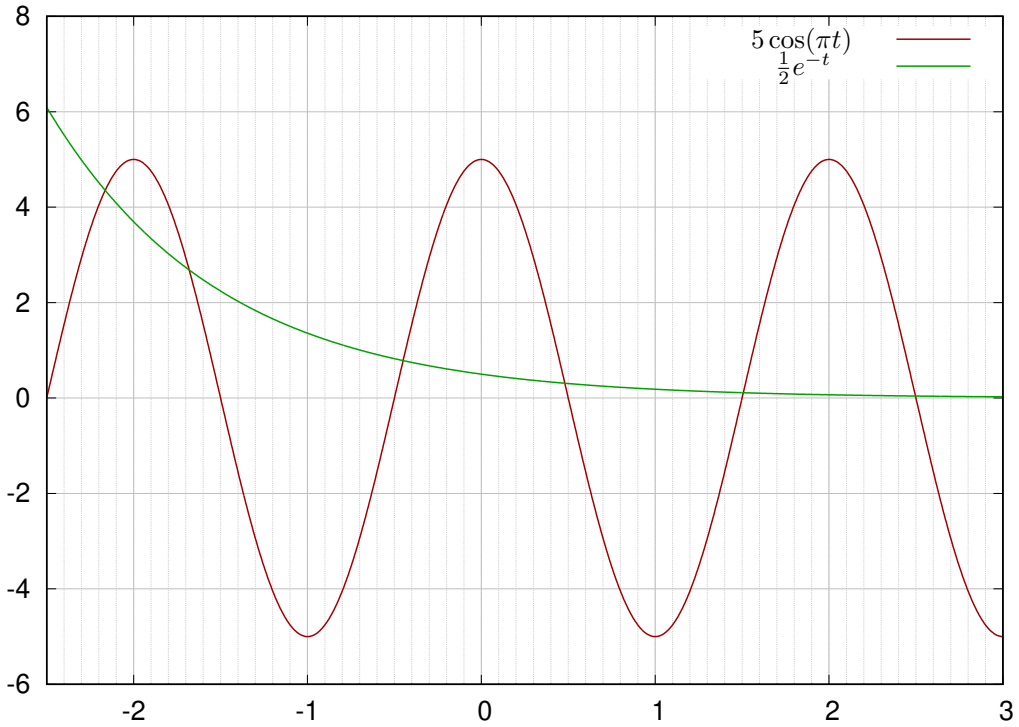
Solución:

Por ejemplo, utilizando GNUPlot:

```
set xrange [-2.5:3]
set samples 1000

set style line 100 lt 1 lc rgb "gray" lw 0.75
set style line 101 lt 1 lc rgb "gray" lw 0.25 dt 2

set mxtics 10
set grid xtics mxtics ytics ls 100, ls 101

plot 5*cos(pi*x) with lines lt -1 lc rgb "#A00000", \
exp(-x)/2 with lines lt -1 lc rgb "#00A000"
```

se obtiene

Ejemplo: graficación (3)



Métodos cerrados

- Punto de partida: Una función cambia de signo en la vecindad de una raíz.

Métodos cerrados

- Punto de partida: Una función cambia de signo en la vecindad de una raíz.
  - Solo se cumple para raíces de multiplicidad impar.

Métodos cerrados

- Punto de partida: Una función cambia de signo en la vecindad de una raíz.
  - Solo se cumple para raíces de multiplicidad impar.
  - Raíces de multiplicidad pares quedan excluidas de estos métodos.

Métodos cerrados

- Punto de partida: Una función cambia de signo en la vecindad de una raíz.
  - Solo se cumple para raíces de multiplicidad impar.
  - Raíces de multiplicidad pares quedan excluidas de estos métodos.

Ejemplo

$$f(x) = k(x - x_0)(x - x_1)(x - x_2)^2(x - x_3)^3$$

Métodos cerrados

- Punto de partida: Una función cambia de signo en la vecindad de una raíz.
  - Solo se cumple para raíces de multiplicidad impar.
  - Raíces de multiplicidad pares quedan excluidas de estos métodos.

Ejemplo

$$f(x) = k(x - x_0)(x - x_1)(x - x_2)^2(x - x_3)^3$$

- Algoritmos cerrados buscan **una** raíz.

Métodos cerrados

- Punto de partida: Una función cambia de signo en la vecindad de una raíz.
  - Solo se cumple para raíces de multiplicidad impar.
  - Raíces de multiplicidad pares quedan excluidas de estos métodos.

Ejemplo

$$f(x) = k(x - x_0)(x - x_1)(x - x_2)^2(x - x_3)^3$$

- Algoritmos cerrados buscan **una** raíz.
- Métodos cerrados parten de un intervalo que *encierra* la raíz.

Métodos cerrados

- Punto de partida: Una función cambia de signo en la vecindad de una raíz.
  - Solo se cumple para raíces de multiplicidad impar.
  - Raíces de multiplicidad pares quedan excluidas de estos métodos.

Ejemplo

$$f(x) = k(x - x_0)(x - x_1)(x - x_2)^2(x - x_3)^3$$

- Algoritmos cerrados buscan **una** raíz.
- Métodos cerrados parten de un intervalo que *encierra* la raíz.
- Intervalo se reduce iterativamente para acorralar la raíz:
  $\Rightarrow$ son métodos **convergentes**

Métodos cerrados

- Punto de partida: Una función cambia de signo en la vecindad de una raíz.
  - Solo se cumple para raíces de multiplicidad impar.
  - Raíces de multiplicidad pares quedan excluidas de estos métodos.

Ejemplo

$$f(x) = k(x - x_0)(x - x_1)(x - x_2)^2(x - x_3)^3$$

- Algoritmos cerrados buscan **una** raíz.
- Métodos cerrados parten de un intervalo que *encierra* la raíz.
- Intervalo se reduce iterativamente para acorralar la raíz:
  $\Rightarrow$ son métodos **convergentes**
- Estos métodos asumen conocimiento del intervalo inicial:
  $\Rightarrow$ otros algoritmos deben utilizarse para inicializarlos.

Dos métodos:

1. Método de bisección
2. Método de interpolación lineal

Método de bisección

- Se busca raíz x_r tal que $f(x_r) = 0$.
- Parte de suposición que $x_r \in [x_l, x_u]$
- Si intervalo suficientemente pequeño entonces signos de $f(x_l)$ y $f(x_u)$ difieren:

$$f(x_l)f(x_u) < 0$$

- Algoritmo de bisección consiste en partir en cada iteración el intervalo en dos
- Condición de parada se realiza cuando el error aproximado

$$\epsilon_a = \left| \frac{x_r^{(i)} - x_r^{(i-1)}}{x_r^{(i)}} \right| 100\,\%$$

es menor a un umbral

Reducción del error en la bisección

Puesto que

$$x_r^{(i)} - x_r^{(i-1)} = \frac{x_u - x_l}{2} \qquad x_r^{(i)} = \frac{x_u + x_l}{2}$$

entonces el error aproximado se puede expresar como

$$\epsilon_a = \left| \frac{x_u - x_l}{x_u + x_l} \right| 100\,\%$$

En cada iteración el error aproximado se reduce a la mitad y

$$E_a^{(n)} = \frac{\Delta x^{(0)}}{2^n} = \frac{x_u^{(0)} - x_l^{(0)}}{2^n}$$

y para un error deseado E_d se despeja el número de iteraciones

$$n = \log_2 \left(\frac{\Delta x^{(0)}}{E_d} \right)$$

Ejemplo: método de bisección (1)

Para encontrar las raíces de

$$5\cos(\pi t) = \frac{1}{2}e^{-t}$$

se utiliza el método de bisección resolviendo

$$5\cos(\pi t) - \frac{1}{2}e^{-t} = 0$$

Ejemplo: método de bisección (2)

Con $\epsilon_s = 0,01\,\%$

i	x_l	x_u	x_r	ϵ_a
1	0	1	0,5	100
2	0	0,5	0,25	100
3	0,25	0,5	0,375	33,333333333333
4	0,375	0,5	0,4375	14,285714285714
5	0,4375	0,5	0,46875	6,6666666666667
6	0,46875	0,5	0,484375	3,2258064516129
7	0,46875	0,484375	0,4765625	1,6393442622951
8	0,4765625	0,484375	0,48046875	0,8130081300813
9	0,4765625	0,48046875	0,478515625	0,40816326530612
10	0,478515625	0,48046875	0,4794921875	0,20366598778004
11	0,4794921875	0,48046875	0,47998046875	0,10172939979654
12	0,47998046875	0,48046875	0,480224609375	0,050838840874428
13	0,480224609375	0,48046875	0,4803466796875	0,025412960609911
14	0,480224609375	0,4803466796875	0,48028564453125	0,012708095056551
15	0,48028564453125	0,4803466796875	0,480316162109375	0,0063536438147277

Código en C++ de algoritmo de bisección (1)

```
template <typename T>
T biseccion(T (*f)(const T), // puntero a función
            T xl,            // límite inferior de intervalo
            T xu,            // límite superior de intervalo
            const T es=sqrt(std::numeric_limits<T>::epsilon()),
            const int maxi=std::numeric_limits<T>::digits) {

  T xr=xl;       // hay que iniciar con algo válido
  T fl=f(xl); // sombra para ahorrar evaluaciones de f()

  T ea=T();      // error aproximado

  for (int i=maxi;i>0;--i){
    T xrold(xr);      // lo necesitamos para el cálculo del error
    xr=(xl+xu)/T(2); // nueva estimacion de raiz, centrada
    T fr=f(xr);       // sombra de f en el centro

    // Evite una divisón por cero
    if (std::abs(xr) > std::numeric_limits<T>::epsilon()) {
      ea=std::abs((xr-xrold)/xr)*T(100); // nuevo error aprox.
```

Código en C++ de algoritmo de bisección (2)

```
    }

    T cond=fl*fr; // esto es negativo si extremo inferior y el
                  // nuevo centro tienen signos diferentes
    if (cond < T(0)) {
      xu=xr;      // es negativo: siga con lado izquierdo
    } else if (cond > T(0)) {
      xl=xr;      // es positivo: siga con lado derecho
      fl=fr;
    } else {
      ea=T(0);    // No hay error! => Algún borde es cero
      xr=(std::abs(fl) < std::numeric_limits<T>::epsilon())
         ? xl : xr; // fl==0
    }

    if (ea < es) return xr; // si se alcanzó precisión, termine
}

// Retorne un NaN si no encontró ninguna raíz
// en el número de iteraciones especificado
```

Código en C++ de algoritmo de bisección (3)

```
    return std::numeric_limits<T>::quiet_NaN();
}
```

Código en C++ de algoritmo de bisección (4)

Ejemplo de uso:

```
// Una función para buscarle raíces
double test(const double x) {
  static const double pi = 3.14159265358979323846264338327950288;
  return 0.5*std::exp(-x)-5.0*std::cos(pi*x);
}

// Programa principal
int main(int argc,char* argv[]) {
  double xl = 0.0; // límite inferior del intervalo
  double xu = 1.0; // límite superior del intervalo

  std::cout << "Raíz en [" << xl << "," << xu << "]= "
            << biseccion(test,xl,xu) << std::endl;
}
```

Otros conceptos de error

- Definición anterior del error aproximado:

$$\epsilon_a = \left| \frac{x_r^{(i)} - x_r^{(i-1)}}{x_r^{(i)}} \right| 100\,\%$$

es altamente riesgosa si la raíz es cero o cerca de cero.

- Otros autores (Press et al.) sugieren por ello utilizar como error aproximado

$$\epsilon_a = \left| x_r^{(i)} - x_r^{(i-1)} \right|$$

- En este último caso, el umbral de parada suele utilizarse como

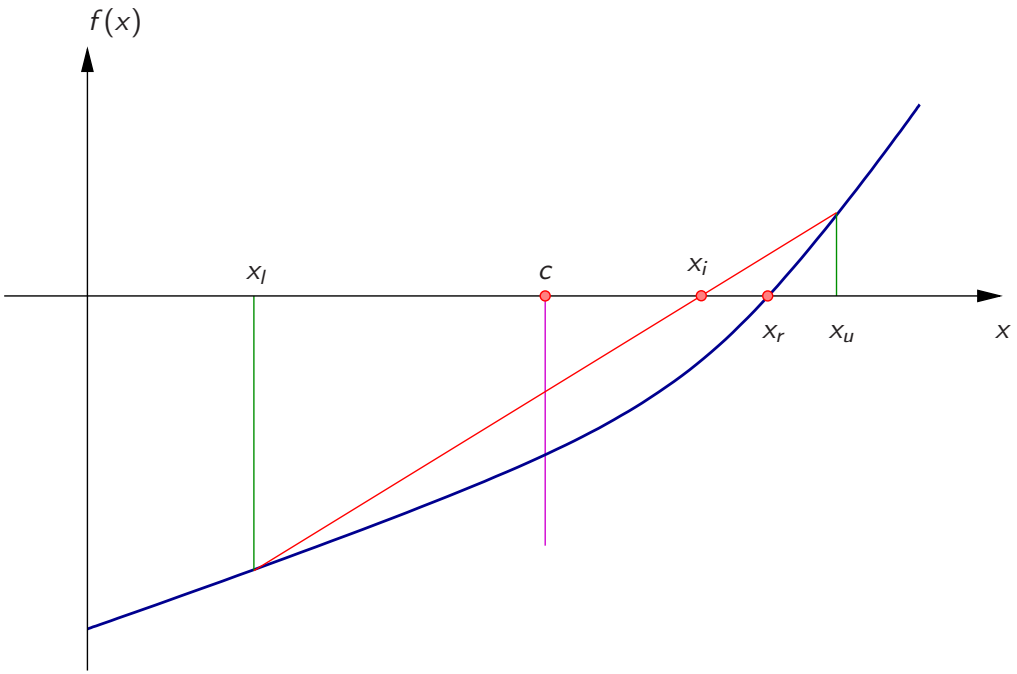
$$\epsilon_s = \frac{x_u - x_l}{2} \mathscr{E}$$

donde $\mathscr{E}$ es el epsilon del formato numérico utilizado.

Método de interpolación lineal

- También denominado método regula falsi o **de la falsa posición**
- Método de bisección ignora cercanía de $f(x_l)$, $f(x_u)$ y $f(x_r)$ a cero
- El método de interpolación lineal, en vez de dividir el intervalo en dos, asume una aproximación lineal de la función para encontrar la raíz.

Cálculo de raíz aproximada



Por triángulos semejantes

$$\frac{f(x_l)}{x_i - x_l} = \frac{f(x_u)}{x_i - x_u}$$

de donde se despeja

$$x_i = x_u - \frac{f(x_u)(x_l - x_u)}{f(x_l) - f(x_u)}$$

Desventajas de la interpolación

- Método parte de suposición que la raíz se encuentra siempre más cercana al extremo menor en magnitud.
- Lo anterior no es siempre cierto (por ejemplo, $f(x) = x^{10} - 1$)
- Método debe modificarse

Modificación a interpolación lineal

Truco para solucionar el problema

Si se cambia el mismo extremo del intervalo (el superior, o el inferior) por más de dos veces consecutivas, entonces simúlese un menor valor de la función en el otro extremo, multiplicándolo por un factor 1/2.

Esto permite que en algún momento el valor simulado de la función se acerque lo suficiente para acorralar mejor a la raíz.

Algoritmo modificado de interpolación lineal (1)

```
template <typename T>
T ilm(T (*f)(const T),
              T xl,
              T xu,
              const T es=std::sqrt(std::numeric_limits<T>::epsilon()),
              const int maxi=std::numeric_limits<T>::digits) {

  T xr=xl; // hay que iniciar con algo válido
  T fl = f(xl);
  T fu = f(xu);

  T ea=T();

  int iu(0),il(0); // contadores para detectar estancamientos



  for (int i=maxi;i>0;--i){
```

Algoritmo modificado de interpolación lineal (2)

```
T xrold(xr); // lo necesitamos para el cálculo del error
xr=xu-fu*(xl-xu)/(fl-fu);
T fr=f(xr);

// Evite una divisón por cero
if (std::abs(xr) > std::numeric_limits<T>::epsilon()) {
  ea = std::abs((xr-xrold)/xr)*T(100);
}

T cond=fl*fr; // cual subintervalo contiene la raíz?
if (cond < T(0)) { // el lado izquierdo tiene la raíz.
  xu=xr;
  fu=fr;
  iu=0;
  il++;
  if (il>=2) {
    fl /= T(2);
  }
} else if (cond > T(0)) { // el lado derecho tiene la raíz.
  xl = xr;
```

Algoritmo modificado de interpolación lineal (3)

```
            fl = fr;
            il=0;
            iu++;
            if (iu>=2) {
              fu /= 2;
            }
        } else {
            ea = T(0); // No hay error! Increíble!
            xr = (fl == T(0)) ? xl : xu;
        }

        if (ea < es) return xr;
    }

    // Retorne un NaN si no encontró ninguna raíz
    return std::numeric_limits<T>::quiet_NaN();
}
```

Resumen

Este documento ha sido elaborado con software libre incluyendo LaTeX, Beamer, GNUPlot, GNU/Octave, XFig, Inkscape, LTI-Lib-2, GNU-Make y Subversion en GNU/Linux