Un método de cálculo Numérico para Integrales Definidas.

Arístides Arellán & Adrián Nieves

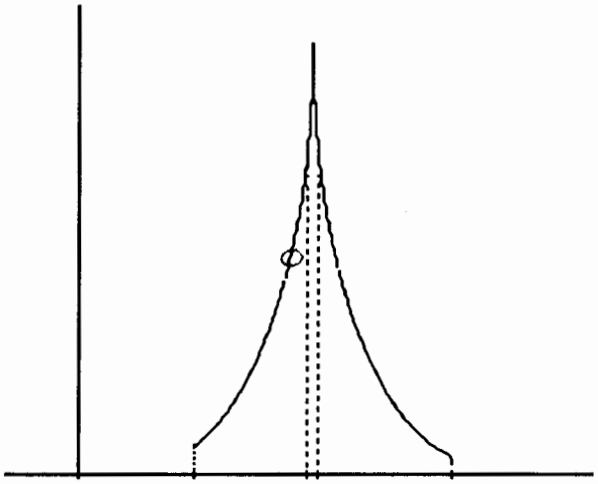
Síntesis. El trabajo presenta un método numérico para el cálculo de integrales definidas usando el *Método Adative Cuadratura*, la teoría de los polinomios de interpolación y la asignación dinámica de memoria, desarrolado con el compilador Turbo Pascal 4.0

INTRODUCCION.

Los métodos básicos para calcular una integral definida $\int_a^b f(x)dx$, donde f es una función continua definida en [a,b], eligen una conveniente partición de [a,b], formada por puntos equidistantes, para luego recurrir a las conocidas fórmulas del Trapecio, Simpson, Gauss, etc, y posteriormente desarrollar el algoritmo del cómputo en base a estos elementos.

Para las llamadas funciones "suaves" este método en general presenta una buena aproximación de la integral. Sin embargo, es conocido que este primer esquema presenta la siguiente dificultad :

Consideremos una función con un comportamiento similar al siguiente gráfico (por ejemplo, $f(x) = 5-| 5-x^3|$)



Con los métodos antes señalados, se obtiene una aproximación satisfactoria de la integral en los subintervalos [a, α] y [β,b], en donde la función tiene, como puede observarse, un comportamiento "suave"; mientras que irregularidades del tipo que presenta la gráfica de la función en el subintervalo [α,β], se "sienten" en el cálculo de la estimación de la integral.

El presente trabajo tiene como finalidad dar una generalización a estos métodos básicos, usando Polinomios de Interpolación y la asignación dinámica de memoria del computador, lo cual, desde nuestro punto de vista, resuelve, en gran parte, la problemática antes señalada.

EL METODO PRESENTADO (Método Adativo de Cuadratura).

La escogencia de una partición uniforme de [a,b], (es decir, una partición formada por puntos equidistantes), es lo que origina el fallo de la estimación de la integral con los métodos básicos, porque trata las estimaciones en los respectivos subintervalos de igual manera, tanto en los subintervalos donde el comportamiento de la funcion es "suave" como en aquellos en donde es irregular. El problema no se resuelve tomando una partición más fina por dos razones, la primera porque en los subintervalos donde el comportamiento de la funcion es "suave" se obtiene una buena estimación con pocos puntos, y la segunda porque mientras más puntos tenga la partición el tiempo de ejecución del programa es mayor.

El método aquí presentado, llamado **Método Adativo de Cuadratura**, no considera, como en el caso anterior, una partición uniforme, en su lugar, presenta un algoritmo que genera particiones más fina sólo en los subintervalos donde el comportamiento de la función es irregular. Como en general no se tiene conocimiento del comportamiento de la función en [a,b], desconocemos las veces en la que necesitaremos refinar estos subintervalos-"irregulares". Es por esto, que no resulta conveniente almacenar la información de la estimación en variables estáticas (alojadas en el inicio del programa) en lugar de ello, resulta más eficiente el uso de variables de almacenamiento dinámico (generadas durante la ejecución del programa de acuerdo a las necesidades del problema que se resuelve)

Al considerar dos puntos, el algoritmo genera como polinomio de interpolación rectas (generalización del "Trapecio"), y con tres puntos, se generan parábolas (generalización de "Simpson").

Paralelamente, el programa presenta varias rutinas de usos frecuente en la programación del ambiente Turbo Pascal : StringXY, LeerRealXY, LeerEnteroXY, WriteXY, y algunas rutinas que utilizan los servicios de la ROM-BIOS, de las máquinas IBM-PC o compatibles.

Implementación.

Este programa se realizó usando el compilador Turbo Pascal 4.0, de la Borland; tambien es ejecutable desde la versión 5.0.

Para la estimación de la integral en los subintervalos, se apróxima la función en el subintervalo generando polinomios de interpolacion de 2,3,4 o 5 puntos[1] (información solicitada durante la ejecución del programa) y el programa obtiene explícitamente el polinomio interpolador con el procedimiento "PolinimioInterpolador"; posteriormente el procedimiento "IntegralPolinomio" evalúa la integral de este polinomio en los extremos del respectivo subintervalo.

El polinomio interpolador, usado para la aproximación de la función se genera en el programa de la siguiente manera : Sabemos que para los puntos $x_0, x_1, x_2, \ldots, x_n$ el polinomio interpolador viene dado por :

$$P(x) = \sum_{i=0}^{n} p_i(x) * f(x_i),$$

$$\text{donde } p_i(x) = \frac{(x-x_0)(x-x_1)\ldots(x-x_{i-1})(x-x_{i+1})\ldots(x-x_n)}{(x_i-x_0)(x_i-x_1)\ldots(x_i-x_{i-1})(x_i-x_{i+1})\ldots(x_i-x_n)}$$

- Los polinomios base p_i, se generan con el procedimiento "PolBase", que usa recursivamente el procedimiento "MulPol".

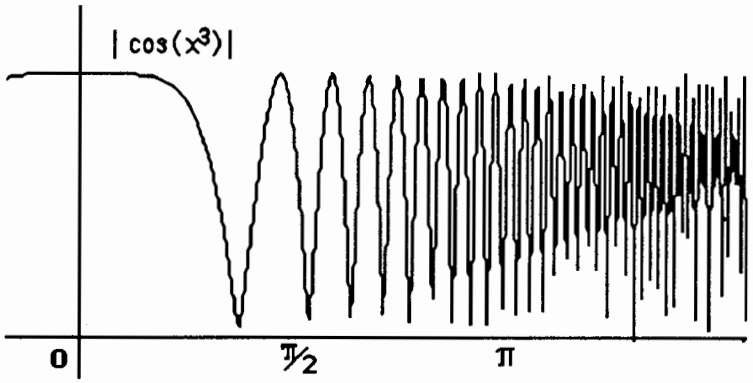
- "MulPol", calcula el producto de un polinomio Q(x) por (x-a) y regresa los coeficientes de este producto en el arreglo P.

- El polinomio interpolador, se obtiene en el procedimiento "PolinomioInterpolador" que regresa sus coeficientes en el arreglo solucion.

[1]Sólo consideramos estos puntos para obtener máxima eficiencia.

Aplicaciones.

1. Calcular la Integral : $\int_0^{\pi} |\cos(x^3)|\, dx$



El programa con los siguientes datos presenta los valores :

Tolerancia = 1e-7	Nº Puntos = 2	Resultado = 2.22346266
Tolerancia = 1e-7	Nº Puntos = 3	Resultado = 2.22350869
Tolerancia = 1e-6	Nº Puntos = 2	Resultado = 2.2233065
Tolerancia = 1e-6	Nº Puntos = 3	Resultado = 2.2235094
Tolerancia = 1e-5	Nº Puntos = 2	Resultado = 2.2232679
Tolerancia = 1e-5	Nº Puntos = 3	Resultado = 2.223494

Nota : El cálculo de esta integral usando los programas de integración del SoftWare "Numerical Methods ToolBox", de la Borland sólo dá resultado cuando se solicita una precisión de dos dígitos.

Bibliografía.

1. Burden, Richard L., and J. Douglas Faires : "Numerical Analysis", 3rd Ed., Boston : Prindle, Weber & Schmidt, 1985.
2. Kandall E. Atkinson : "An Introducction to Numerical Analysis", John Wiley & Sons, Inc.

3. Boralnd International : Turbo Pascal, Numerical Methods ToolsBox.

4. Boralnd International : Turbo Pascal 4.0, Compilador.

5. Boralnd International : Turbo Pascal 5.0, Compilador.

6. Peter Norton : "Guia del programador para el IBM-PC", Red Editorial IberoAmericana, Andes.

```
PROGRAM IntegracionNumerica;

{-------Calculo de Integral Definida.  Usando "Método Adative de  Cuadratura"
 -------Para considerar la funcion cos(x), sustituya en la "function f " la tercera linea por :
 ------- f : = cos(x)
 -------Luego ejecute el programa.                                    ----------------------------------------
-------}

USES  CRT, DOS;

CONST
   N : integer = 6;
   TOLERANCIA : Byte = 6;

TYPE
   Arreglo          = ARRAY[-1..50] OF real;
   DatoIntegral     =  record
                         ext1,
                         ext2,
                         Est    : real
                       end;

   Ptr              = ^Nodo;
   Nodo             =  record
                          info : DatoIntegral;
                          next : Ptr
                       end;
   String80         =  String[80];

VAR
   a, b             :  real;
   integral         :  real;
   coeficiente,
   particion,
   solucion         : Arreglo;
   k,
   Dig_Iguales      : Byte;
   ExtInf,
   ExtSup           :  real;
   Nuevaprox,
   est1, est2       :  real;
   Dato             :  DatoIntegral;
   Stack            :  Ptr;


{=================================================================función a
integrar}

function f (x : real) : real;
begin
    f := sin(x)
end;
```

```
{-------------------------------------------------------------------------------
WriteXY}

procedure WriteXY (x, y : Byte;  mensaje : String80);
begin
    GoToXY(x, y);
   ClrEOL;
   write(mensaje)
end;

{-------------------------------------------------------------------------------
AjustarCursor}

procedure AjustarCursor (LineaInicial, LineaFinal : Byte);
var
     Reg : Registers;
begin       {AjustarCursor  }
   with  Reg do
         begin
           AX := $0100;  {intr = $10; servicio AH: = $01 }
           BX := $0;
           CH := LineaInicial;
           CL := LineaFinal;
           Intr($10,  Reg)
         end;
end;          {AjustarCursor }

{-------------------------------------------------------------------------------
Mensaje}

procedure Mensaje (informacion : String80;  Atributo : Integer);
var
   a, b : Byte;
begin   {Mensaje  }
   a := WhereX;      {guarda posicion del cursor }
   b := WhereY;
   AjustarCursor(2, 1);       {quito el cursor de pantalla }
   TextColor(Atributo);
   WriteXY(1,  25,  Informacion);
   Delay(60);
   GoToXY(a, b);                {regresa posición inicial  del cursor}
   AjustarCursor(12,  13);    {repone  cursor}
   TextColor(7);                {repone Video Inicial}
end;   {Mensaje }

{-------------------------------------------------------------------------------
StringXY }

function StringXY (x, y : Byte;  informacion : String80) : String80;
var
  s : String;
  ch : char;
  IOCode : integer;
```

```
begin       {StringXY }
  WriteXY( x, y, Informacion);
  s := '';
  ch := #48;
  Repeat
    If KeyPressed then
      begin
        ch:=ReadKey;
        Mensaje('',7);
        case ch of
            #27 : Halt;
            #13: ;
            #32 ..#126 : begin
                          s:=s+ ch;
                          write(ch)
                         end;
                #8 : if Length(s) > 0 then
                       begin
                         write(#8,#32,#8);
                         Delete(s, Length(s), 1)
                       end
      end {case}
    end;
  Until  (s<>'' ) or (ch = #13 );
  StringXY:= s
end;     {StringXY }

{-------------------------------------------------------------------------------
LeerRealXY}

procedure LeerRealXY (x,y: Byte; informacion : String80; var r : real) ;
var
  s : String80;
  IOCode : Integer;
begin
   Repeat
     s:= StringXY(x, y, informacion);
     Val (s, r, IOCode);     {función del Turbo }
     If IOCode <> 0 then
         Mensaje('Error en la entrada del numero. Repitala.', 236)
   Until (IOCode = 0 )
end;

{-------------------------------------------------------------------------------
LeerEntero}

procedure LeerEnteroXY (x,y: Byte; informacion : String80; var n : integer) ;
var
   s : String80;
   IOCode : Integer;
begin  {LeerEnteroXY }
   Repeat
     s:= StringXY(x, y, informacion);
```

```
      Val (s, n, IOCode);
      If IOCode <> 0 then
         Mensaje('Error en la entrada del numero. Repitala.', 236)
   Until (IOCode = 0 )
end;     {LeerEnteroXY }

{-----------------------------------------------------------------
Evaluar_PrimitivaPolinomio }

function Evaluar_PrimitivaPolinomio (Coeficiente : Arreglo; x : real) : real;
var
  k         : Byte;
  integral : real;
  potencia      : real;
begin
   integral : = 0;
   potencia : = x;
   for k := 1 to n do
      begin
         integral : = integral + (Solucion[k - 1] / k) * potencia;
         potencia : = potencia * x
      end;
   Evaluar_PrimitivaPolinomio  : = integral
end;
{-------------------------------------------------------------------------------
MultPol }

procedure MulPol (GradoP : byte; P : Arreglo; Alfa : real; var Q : Arreglo);
var
  j    : Byte;
begin   {MultPol }
  for j := GradoP + 1 DownTo  0 do
      Q[j] := P[j - 1] - P[j] * Alfa
end;    {MultPol }

{-------------------------------------------------------------------------------
OverFlow }

procedure OverFlow;
begin   {OverFlow}
   ClrScr;
   WriteXY(5,5,' Error al intentar calcular la aproximacion de la integral con : ');
   WriteXY(5,7,'Tolerancia  =  : ');      write( Tolerancia : 2);
   WriteXY(5,9,'Numero de particiones  =  : ');      write( n: 2);
   WriteXY(5,12,'Intente con valores menores..... ');
   Mensaje('Pulse una tecla para salir.', 7);
   AjustarCursor(2,1);
   Repeat  Until   (KeyPressed);
   AjustarCursor(12,13);
   Halt
end;    {OverFlow}
```

```
{-------------------------------------------------------------------------------
PolBase }

procedure PolBase (i : byte;  var P : Arreglo);
var
  j, g : Byte;
  Denominador : real;
begin      {PolBase}
   FillChar(P, SizeOf(P), 0);
   P[0] : = 1;
   g : = 0;
   Denominador : = 1.0;
   for j : = 0 to n - 1 do
       if j <> i then
          begin
            MultPol (g, P, Particion[j],  P);      {obtencion del Numerador}
            g : = Succ(g);
            Denominador : = Denominador * (Particion[i] - Particion[j]);
          end;
   If (Abs(Denominador ) = 0.0 ) or (Abs(Denominador ) < 1e-35 ) then
      OverFlow;
   for j := 0 to n - 1 do              {Q tendrá los coef del poli Pi(x) }
       P[j] : = P[j] / Denominador
end;   {PolBase}

{-------------------------------------------------------------------------
PolinomioInterpolador}

procedure PolinomioInterpolador(Particion : Arreglo; var Solucion : Arreglo);
var
  j   : byte;
begin  {PolinomioInterpolador}
   FillChar(Solucion, SizeOf(Solucion), 0);
   for j : = 0 to n - 1 do
       begin
          PolBase(j, Coeficiente);
          for k : = 0 to n - 1 do
              if abs(solucion[k] > 1e+30 then
                 OverFlow
              else
                 Solucion[k] : = Solucion[k] + Coeficiente[k] * f (Particion[j])
       end
end;   {PolinomioInterpolador}

{-------------------------------------------------------------------------------
Aproximacion }

function Aproximacion (a,b : real ) : real;
var
   h : real;
begin   {Aproximacion}
   h : = (b-a)/(n-1);
   for k : = 0 to n-1  do  { genera una particion de [a,b] de n-puntos }
```

```
        Particion[k]: = a + h*k;
    PolinomioInterpolador(Particion, Solucion );
    Aproximacion : = Evaluar_PrimitivaPolinomio(Solucion,b) - Evaluar_PrimitivaPolinomio (Solucion,
a)
end;    {Aproximacion}

{-------------------------------------------------------------------------------
-Push}

procedure Push ( Dato: DatoIntegral; var Stack : Ptr);
var
          Nodo : Ptr;
begin   {Push}
   New(Nodo);          {crea un nuevo Nodo }
   Nodo^.Info: = Dato;
   Nodo^.Next: = Stack;
   Stack: = Nodo
end;  {Push}

{-------------------------------------------------------------------------------
-Pop }

procedure Pop  (var  Dato: DatoIntegral; var Stack : Ptr);
var
   Nodo : Ptr;
begin   {Pop  }
   Nodo : = Stack ;  {apunta a la entrada de la pila }
   Stack : = Nodo^.Next ;
   Dato: = Nodo^.info;
   Dispose(Nodo)     {libera el Nodo de la pila }
end;  {Pop  }

{-----------------------------------------------------------------------
NuevaAproximacion }

procedure NuevaAproximacion   (Dato: DatoIntegral; var NuevAprox,   est1, est2 : real);
begin   {NuevaAproximacion   }
   with Dato do
        begin
          est1: = Aproximacion(ext1,  (ext1+ext2)/2  );
          est2: = Aproximacion((ext1+ext2)/2,  ext2  );
          NuevAprox: = est1+est2
        end
end;   {NuevaAproximacion   }

{-----------------------------------------------------------------------
ComparарEstimaciones}

procedure ComparарEstimaciones(est1,est2  : real; var digitos : Byte);
var
   m       :  byte;
   exp10   :  real;
   dif     :  real;
```

```
begin
   m : = 0;
   exp10 : = 1;
   Repeat
     m : = succ(m);
     exp10 : = exp10 * 10;
     dif : = Int(est2 * exp10) -  Int(est1 * exp10)
   Until  (dif<>0)  or  (m>19);
   digitos: = m
end;
{-------------------------------------------------------------------------
EscribirResultado}

procedure EscribirResultado;
begin
  ClrScr;
  WriteXY(5,5,'Extremo inferior : ');      write(a:1:6);
  WriteXY(5,7,'Extremo superior : ');      write(b:1:6);
  WriteXY(5,9,'Tolerancia  : ');           write(Tolerancia:2);
  WriteXY(5,11,'Aproximacion  : ');        write(Integral:9:Tolerancia+1);
  Mensaje('Pulse una tecla para Salir.',7);
  AjustarCursor(12,13);
  Halt
end;
{-------------------------------------------------------------------------
LeerExtremos}

procedure LeerExtremos(var a,b: real);
begin
  Repeat
     LeerRealXY(5,5,'Entre el extremo inferior : ', a );
     LeerRealXY(5,7,'Entre el extremo superior : ', b );
     If (a >= b) then
        begin
          ClrScr;
          Mensaje('El extremo inferior debe ser menor que el extremo superior.',15)
        end
  Until ( a< b)
end;

{-------------------------------------------------------------------------
LeerTolerancia}

procedure  LeerTolerancia (var Tolerancia: Byte);
var
   n : integer;
begin
  Repeat
     WriteXY(5,9,'Para obtener k digitos de precision entre 1e-k ');
     LeerEnterolXY(5,10,'Entre el k de la Tolerancia ( hasta 1e-7) :  1e-', n );
     Tolerancia : = n;
     If not (Tolerancia In [1..7] ) then
        Mensaje('Solo enteros del 1 hasta 7, inclusive.',15)
```

```
  Until ( Tolerancia In [1. . 7] )
end;

{-------------------------------------------------------------------------------
nParticion}

procedure  nParticion (var n: integer );
begin
  Repeat
     LeerEnteroXY(5,12,'Numeros de puntos de la particion  (n entre 2 y 5) : ', n );
     If not (n in [2..5] ) then
        Mensaje('Solo enteros del 2 hasta 5, inclusives.',15 )
  Until ( n In  [2..5] )
end;

{-------------------------------------------------------------------------------
EvaluarIntegral}

procedure  EvaluarIntegral ;
begin  {EvaluarIntegral}
   AjustarCursor(2,1);
   NuevaAproximacion(Dato, NuevAprox, est1, est2 ) ;
  Repeat
     CompararEstimaciones (Dato.Est, NuevAprox, Dig_Iguales );
     If  ( Dig_Iguales >= Tolerancia )  then
         begin  {#1}
           integral : = integral + NuevAprox;
           If  Stack = Nil  then
              begin
                 EscribirResultado;
                 Mensaje('Pulse una tecla para salir. ',15);
                 Halt
              end
           else
              Pop(Dato, Stack)
         end  {#1}
      else
         begin
           with Dato do
                begin
                  extInf:  =ext1;
                  extSup: = ext2;
                  ext2:  =  (ext1+ext2)*0.5;
                  Est: = est1
                end;
           Push(Dato, Stack);
           Dato.Est: =est2;
           Dato.ext1:  =(extInf+extSup)/2;
           Dato.ext2: =extSup
         end;
     NuevaAproximacion (Dato, NuevAprox, est1, est2)
  Until ( False )
end;   {EvaluarIntegral}
```

```
{-------------------------------------------------------------------------
*principal}

BEGIN
   ClrScr;
   Stack : =Nil;
   integral : =0.0;
   LeerExtremos(a,b);
   LeerTolerancia(Tolerancia);
   with Dato do
        begin
           ext1: =a;
           ext2: = b;
           Est: = Aproximacion(ext1, ext2 )
        end;
   nParticion(n);
   EvaluarIntegral
END.
```