UNIVERSIDAD DE CONCEPCION
FACULTAD DE CIENCIAS
FISICAS Y MATEMATICAS
DEPARTAMENTO DE INGENIERIA MATEMATICA

Cálculo Numérico (521230)

Laboratorio 6
Integración numérica

El objetivo de este laboratorio es aprender técnicas de integración numérica, así como el uso correcto de funciones como parámetros.

En muchas ocasiones resulta sumamente útil poder pasar una función como parámetro. Un caso típico es cuando se quiere escribir un programa para calcular la integral de una función cualquiera mediante una regla de integración numérica particular. Así, por ejemplo, si se escribe un programa MATLAB llamado `trap` que calcula la aproximación de la integral de una función dada en un intervalo genérico $[a, b]$ por la regla de los trapecios con N subintervalos, uno querría poder llamar a ese programa con sentencias como las siguientes:

```
>> trap(f,0,1,10) % Calcula la integral de f(x) en [0,1] con 10 subintervalos.
>> trap(g,-1,1,16) % Calcula la integral de g(x) en [-1,1] con 16 subintervalos.
>> trap('sin',0,pi,8) % Calcula la integral de sen(x) en [0,pi] con 8 subintervalos.
```

Para poder hacer esto, el programa `trap` debe recibir la función pasada como parámetro ($f(x)$, $g(x)$ o $\operatorname{sen} x$, respectivamente) en una variable alfabética (es decir una variable cuyos valores son nombres y no números) y, cada vez que haga falta evaluar esa función, debe utilizarse el comando `feval`, cuya sintaxis es como en el ejemplo siguiente:

```
>> function int=trap(funct,a,b,N)
>> ...
>> y=feval(funct,a)
```

En este ejemplo, `funct` es la variable alfabética que contiene el nombre de la función que se quiere evaluar, `a` es el valor de la variable donde se quiere evaluar la función e `y` devuelve el valor calculado.

Por otra parte, debe haber un programa *function* (`f.m` en el primer ejemplo o `g.m` en el segundo) en el que se defina la función que se quiere integrar. Esto no es necesario cuando se trata de una de las funciones de biblioteca de MATLAB, como en el caso de $\operatorname{sen} x$ en el tercer ejemplo.

Alternativamente, cuando una función $f(x)$ es muy simple y se quiere evitar tener que hacer un programa `f.m` para evaluarla, puede utilizarse el comando `inline` como en el siguiente ejemplo:

```
>> f = inline('exp(-x.^2)'); % Define la funci\'on f(x)=exp(-x^2).
>> trap(f,0,1,20) % Calcula la integral de f(x) en [0,1] con 20 subintervalos.
```

1. (a) Transcriba en un archivo `trap.m` el siguiente programa que calcula la aproximación de la integral de una función dada en un intervalo genérico $[a, b]$ por la regla de los trapecios con N subintervalos:

```
function int=trap(funct,a,b,N)

h=(b-a)/N;
x=a+h*(1:N-1);
int=h*((feval(funct,a)+feval(funct,b))/2+sum(feval(funct,x)));
```

(b) Testee el programa anterior con las siguientes integrales y $N = 10, 20, 40, 80$:

$$\int_0^3 x^2\,dx\,; \qquad \int_{-1}^1 \mathrm{e}^{-x^2}\,dx\,; \qquad \int_1^2 \log x\,dx \qquad \text{y} \qquad \int_0^1 \sqrt{x}\,dx\,.$$

Para ello ingrese cuando sea posible el integrando como una función de biblioteca y, cuando no lo sea, utilice el comando `inline`. Imprima los resultados en formato largo (`format long`).

(c) Haga un programa semejante para la regla de Simpson y testeelo con las mismas funciones.

2. El fin de este ejercicio es verificar experimentalmente que el error del método de los trapecios aplicado a un integrando con derivadas acotadas en el intervalo de integración es de orden $\mathcal{O}(h^2)$.

   (a) Haga un programa que calcule la integral $\int_0^1 \mathrm{e}^{-x}\,dx$ por la regla de los trapecios con $N = 10, 20, 30, \ldots, 100$ subintervalos y almacene los errores respectivos.
   Sugerencia: para calcular el error utilice el valor verdadero de la integral, que en este caso puede calcularse exactamente.

   (b) Grafique en escala logarítmica estos errores versus N y la función $f(N) = (1/N)^2$.
   Sugerencia: para graficar en escala logarítmica utilice el comando `loglog` en lugar de `plot`; la sintaxis de ambos comandos es la misma.

   (c) Explique por qué el gráfico anterior muestra que el error de la regla de los trapecios es en este caso $\mathcal{O}(h^2)$.

3. El fin de este ejercicio es verificar experimentalmente que el error del método de los trapecios aplicado a un integrando con derivadas no acotadas en el intervalo de integración no es de orden $\mathcal{O}(h^2)$.

   (a) Repita el ejercicio anterior con la integral $\int_0^1 \sqrt{x}\,dx$.

   (b) Para estimar el orden de convergencia del método en este caso, determine por cuadrados mínimos las constantes C y α que mejor ajustan los errores mediante el modelo

   $$\text{error} \approx Ch^\alpha.$$

   (c) Grafique en escala logarítmica los errores versus N y la función $f(N) = (1/N)^\alpha$.

4. MATLAB tiene dos comandos, `quad` y `quad8`, que permiten calcular de manera automática integrales de funciones suaves con error menor que una tolerancia dada. El primero es un método de bajo orden basado en la regla de Simpson, mientras que el segundo es un método de alto orden.

   (a) Utilice el *help* de MATLAB para conocer la sintaxis de estos comandos.

   (b) Calcule mediante `quad` las integrales $\int_{-1}^1 \mathrm{e}^{-x^2}\,dx$ y $\int_0^1 \sqrt{x}\,dx$ del Ej. 1(b), primero con la tolerancia prefijada por el comando y después con error menor que 10^{-6}.

Soluciones propuestas

1. Ej. 1 (b). File Test_Trap.m :

```
format long
f=inline('x.^2');
X=[trap(f,0,3,10) trap(f,0,3,20) trap(f,0,3,40) trap(f,0,3,80)]

g=inline('exp(-x.^2)');
E=[trap(g,-1,1,10) trap(g,-1,1,20) trap(g,-1,1,40) trap(g,-1,1,80)]

L=[trap('log',1,2,10) trap('log',1,2,20) trap('log',1,2,40) ...
   trap('log',1,2,80)]

S=[trap('sqrt',0,1,10) trap('sqrt',0,1,20) trap('sqrt',0,1,40) ...
   trap('sqrt',0,1,80)]
```

Salida:

```
>> Test_Trap

X =
   9.04500000000000   9.01125000000000   9.00281250000000   9.00070312500000

E =
   1.48873667952733   1.49242159226350   1.49334167387975   1.49357162247796

L =
   0.38587793674575   0.38619020963221   0.38626832040247   0.38628785076256

S =
   0.66050934170682   0.66444659142664   0.66587096567353   0.66638264723897
```

2. Ej. 1 (c). File simp.m :

```
function int=simpson(funct,a,b,N)

h=(b-a)/(2*N);
xp=a+(2:2:2*N-2)*h;
xi=a+(1:2:2*N-1)*h;
int=(h/3)*(feval(funct,a)+feval(funct,b)+2*sum(feval(funct,xp))...
           +4*sum(feval(funct,xi)));
```

File Test_Simpson.m :

```
format long

X=[simp(f,0,3,10) simp(f,0,3,20) simp(f,0,3,40) simp(f,0,3,80)]

E=[simp(g,-1,1,10) simp(g,-1,1,20) simp(g,-1,1,40) simp(g,-1,1,80)]

L=[simp('log',1,2,10) simp('log',1,2,20) simp('log',1,2,40) ...
   simp('log',1,2,80)]

S=[simp('sqrt',0,1,10) simp('sqrt',0,1,20) simp('sqrt',0,1,40) ...
   simp('sqrt',0,1,80)]
```

Salida:

```
>> Test_Simpson

X =
    9.00000000000000   9.00000000000000   9.00000000000000   9.00000000000000

E =
    1.49364989650889   1.49364836775183   1.49364827201070   1.49364826602401

L =
    0.38629430059436   0.38629435732589   0.38629436088259   0.38629436110506

S =
    0.66575900799992   0.66634575708916   0.66655320776078   0.66662655287100
```

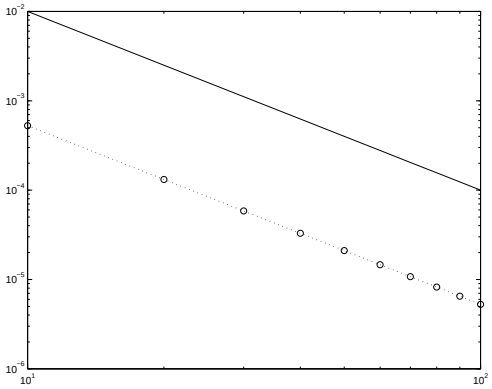
3. Ej. 2 (a,b).

File Orden_exp.m :

```
f=inline('exp(-x)');
N=10:10:100;
error=[];
for n=N
    integ=trap(f,0,1,n);
    error=[error abs(integ-(1-exp(-1)))];
end

loglog(N,error,'o:',N,(1./N).^2,'-')
```

Salida:



4. Ej. 2 (c). El gráfico en escala logarítmica de los errores versus N es una recta paralela a la de la función $f(N) = (1/N)^2$. La pendiente de esta última es -2, pues

$$\log f(N) = -2 \log N;$$

por lo tanto, tambien los errores satisfacen una relación del tipo

$$\log(\text{error}) = -2 \log N + b,$$

y, por ende, dado que $h = 1/N$,

$$\text{error} = \frac{C}{N^2} = Ch^2, \qquad \text{con } C = 10^b.$$

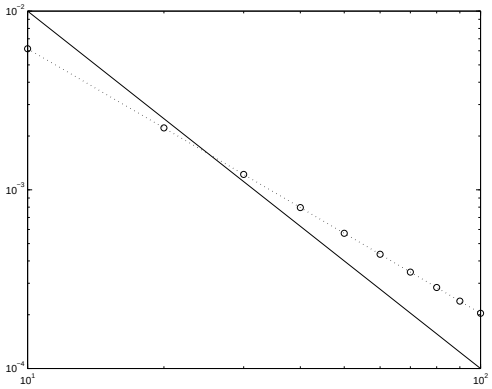
5. Ej. 3 (a).

File Orden_exp.m:

```
g=inline('sqrt(x)');
N=10:10:100;
error=[];
for n=N
    integ=trap(g,0,1,n);
    error=[error abs(integ-2/3)];
end

loglog(N,error,'o:',N,(1./N).^2,'-')
```

Salida:



En este caso la gráfica de los errores es una recta de menor pendiente, lo que indica que el orden de convergencia es menos que cuadrático.

6. Ej. 3 (b,c).

File Orden_exp.m (cont.):

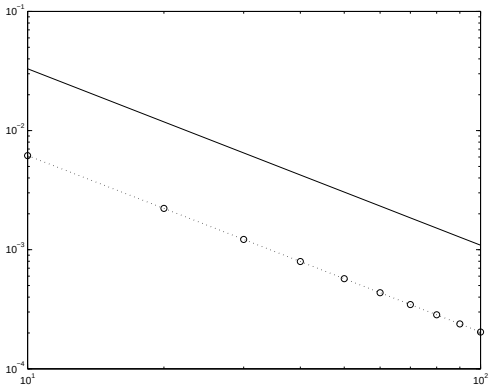
```
m=length(N);
A=[ones(m,1) -log(N)'];
b=log(error)';
z=A\b;
alpha=z(2)

loglog(N,error,'o:',N,(1./N).^alpha,'-')
```

Salida:

```
>> orden

alpha =
    1.4811
```



7. Ej. 4 (b). File Ej7_4.m:

```
format long

g=inline('exp(-x.^2)');
E1=quad(g,-1,1)
E2=quad(g,-1,1,1.e-6)

S1=quad('sqrt',0,1)
S2=quad('sqrt',0,1,1.e-6)
```

Salida:

```
>> Ej7_4

E1 =
   1.49371075958197

E2 =
   1.49364826567254

Warning: Recursion level limit reached in quad. Singularity likely.
> In /usr/local/matlab5/toolbox/matlab/funfun/quad.m (quadstp) at line 102
  In /usr/local/matlab5/toolbox/matlab/funfun/quad.m (quadstp) at line 141
  ...
  In /usr/local/matlab5/toolbox/matlab/funfun/quad.m at line 72
  In /home/rodolfo/TEX/cursos/CNUM/cnum-1.02/Matlab/Ej7_1.m at line 32
Warning: Recursion level limit reached 8 times.
> In /usr/local/matlab5/toolbox/matlab/funfun/quad.m at line 80
  In /home/rodolfo/TEX/cursos/CNUM/cnum-1.02/Matlab/Ej7_1.m at line 32

S1 =
   0.66666250953207

Warning: Recursion level limit reached in quad. Singularity likely.
> In /usr/local/matlab5/toolbox/matlab/funfun/quad.m (quadstp) at line 102
  In /usr/local/matlab5/toolbox/matlab/funfun/quad.m (quadstp) at line 141
  ...
  In /usr/local/matlab5/toolbox/matlab/funfun/quad.m (quadstp) at line 141
  In /usr/local/matlab5/toolbox/matlab/funfun/quad.m at line 72
  In /home/rodolfo/TEX/cursos/CNUM/cnum-1.02/Matlab/Ej7_1.m at line 33
Warning: Recursion level limit reached 40 times.
> In /usr/local/matlab5/toolbox/matlab/funfun/quad.m at line 80
  In /home/rodolfo/TEX/cursos/CNUM/cnum-1.02/Matlab/Ej7_1.m at line 33

S2 =
   0.66666578927560
```