FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 1

[75.12] Análisis Numérico I
Segundo cuatrimestre de 2018

Alumno	Padrón	Mail
del Mazo, Federico	100029	delmazofederico@gmail.com
Kristal, Juan Ignacio	99779	kristaljuanignacio@gmail.com

Curso 07:

- Dr Daniel Fabian Rodriguez
- Valeria Machiunas
- Federico Balzarotti
- Michael Portocarrero

ANÁLISIS NUMÉRICO I - 75.12 – 95.04

Curso: Rodríguez- Balzarotti - Machiunas- Portocarrero

2º cuatrimestre de 2018

TRABAJO PRÁCTICO DE MÁQUINA Nº 1

Desarrollo del práctico:

1) Programar un algoritmo para estimar la unidad de máquina (μ), en simple y en doble precisión.
2) Implementar el método de trapecios compuestos para evaluar la integral:

a) $$F(\alpha,\beta)=\int_{1}^{240}\frac{\sin(Px)+\beta x^2}{\alpha\ x}dx$$

donde P = (N° de padrón de integrante 1 + N° de padrón de integrante 2) /50

o bien P = Nº de padrón / 25

α=0.17 y β=0.41

de tal forma que el módulo del error absoluto de truncamiento sea menor que 10^{-5}.

Informar qué valor de n (cantidad de trapecios) se ha utilizado, justificando la elección.

Considere P exacto, y α y β bien redondeados.

3) Fijado dicho valor de n, luego:
   I. Calcular la condición del problema mediante la técnica de perturbaciones experimentales.
   II. Estimar experimentalmente el término de estabilidad.
   III. Utilizando los resultados anteriores que sean necesarios, y suponiendo nulo el error inherente, acotar el error total.
   IV. Repetir III suponiendo que el error inherente relativo está acotado por $0.5\ 10^{-4}$
   V. Indicar la fuente más importante de error en los dos casos anteriores.

La entrega del presente trabajo práctico deberá realizarse de acuerdo al reglamento del curso, en la fecha informada en clase.

Índice

1. Introducción

El trabajo práctico tiene como objetivo el cálculo y acotamiento de errores de la siguiente integral:

$$F(\alpha, \beta) = \int_1^{240} \frac{\sin(Px) + \beta x^2}{\alpha x} dx \tag{1}$$

Siendo:

- $P = \frac{\sum_{padrones}}{50} = \frac{99779+100029}{50} = 3996{,}16$ exacto
- $\alpha = 0{,}17$ bien redondeando
- $\beta = 0{,}41$ bien redondeando

Específicamente:

- Se estimará el valor de la unidad de maquina μ.
- Se evaluará la integral con el método de trapecios compuestos teniendo con objetivo en mente que el error absoluto de truncamiento sea menor a 1×10^{-5}.
- Se calculará computacionalmente la cantidad de trapecios utilizada en el método descrito anteriormente.
- Se calculará la condición del problema mediante perturbaciones experimentales.
- Se estimará experimentalmente el término de estabilidad.
- Se acotará el error total.

2. Desarrollo

2.1. Estimación de μ

Para los cálculos del μ se utilizó el algoritmo del ejemplo 6.4 del libro de Hernan Gonzales [1].

2.2. Función y sus derivadas

Siempre teniendo en cuenta los valores de P, α, β previamente utilizados, definimos la funcion $f(x)$ como:

$$f(x) = \frac{\sin(Px) + \beta x^2}{\alpha x} \tag{2}$$

Graficamos la función para saber un poco más de ella en la figura 1

De esta función calculamos sus derivadas y las graficamos en las figuras 2 3, para utilizar en cálculos posteriores.

En la figura 4 se puede observar graficamente la cota superior de la funcion derivada segunda y por lo tanto es utilizada como cota para calcular el error de truncamiento

$$f'(x) = \frac{P\cos(Px)}{\alpha x} - \frac{\sin(Px)}{\alpha x^2} + \frac{\beta}{\alpha} \tag{3}$$

$$f''(x) = -\frac{2P\cos(Px)}{\alpha x^2} + \frac{2\sin(Px)}{\alpha x^3} - \frac{P^2\sin(Px)}{\alpha x} \tag{4}$$

2.3. Método de trapecios compuestos

Sabemos que el error de truncamiento producido por el método de trapecios compuestos es:

$$\epsilon_t = -\frac{(b-a)^3}{12n^2} * f''(\xi) \tag{5}$$

Donde b, a son los limites de integración y n es la cantidad de trapecios. Como lo que queremos es acotar el error de truncamiento, debemos evaluar a la segunda derivada en su imagen máxima, es decir $\xi = 1$.

Por lo tanto, y ahora con el error de truncamiento al que queremos llegar, despejamos la cantidad de trapecios:

$$n = \sqrt{\left| -\frac{(b-a)^3 * f''(1)}{12\epsilon_t} \right|} \tag{6}$$

Es con este n que se puede finalmente implementar la función de el método de los trapecios compuestos.

2.3.1. Truncamiento de n

Al despejar por el método de los trapecios compuestos el n necesario para tener el error deseado se notó que este valor era de tal magnitud y orden que computacionalmente carecería de sentido usarlo para cada cálculo. Es por esto que se decidió hacer un truncamiento de este valor, para poder tratarlo como es debido y en un lógico margen de tiempo. De todas formas, solo anecdóticamente, se incluye una corrida del programa con el n original.

El criterio para truncar n es el de ver como escala el cálculo de la integral respecto del valor, y luego decidir un punto de corte tratable arbitrariamente (en nuestro caso, 5 minutos). Se puede ver en el gráfico 5 que esta es una función lineal, lo cual tiene sentido ya que lo único que adiciona computacionalmente es el ciclo definido `for` de la función, haciendolo $\mathcal{O}(n)$, siendo n el mismo n con el que venimos tratando, redundantemente.

2.4. Condición del problema y término de estabilidad

Para realizar las perturbaciones lo haremos sobre alpha y beta respectivamente, introduciendo un error en un ciclo de 16 iteraciones. Una vez obtenidos distintos resultados y teniendo en cuenta el primer valor obtenido, calculamos la nueva condición del problema, una por iteración. Finalmente, de todos los CP obtenidos, tomamos el mayor, tanto para alpha como para beta y es ese nuestro resultado final.

Es ahora que, teniendo en cuenta los valores de la integral con precisión simple y precisión doble (utilizando el valor de n no truncado, para poder apreciar de manera mas exagerada la inestabilidad

del algoritmo) podemos calcular el valor del término de estabilidad. Esto lo hacemos con la siguiente ecuación:

$$Te = \frac{integral\_doble - integral\_simple}{integral\_doble * \mu_s} \tag{7}$$

Finalmente, se grafica en la figura 6 y la figura 7 el cálculo del Cp respecto de alpha y beta en cada iteración.

2.5. Error total

Sabiendo que el error total es la sumatoria de los errores inherentes, de redondeo y de truncamiento querémos ahora acotar el error total.

$$Et = Ei + Er + Etr \tag{8}$$

$$Et = Cp * r + Te * \mu_s + Etr \tag{9}$$

Este cálculo lo haremos con dos casos en particular:

- Error inherente nulo
- Error inherente acotado

2.6. Fuentes de error

Se puede observar en ambos casos que la mayor fuente de error es el error de redondeo, y esto tiene sentido de acuerdo al método empleado. Al efectuar el metodo de trapecios compuestos se requieren computar muchisimas sumas y multiplicaciones, cada una con sus respectivos errores de redondeo los cuales son variantes en base al error de maquina particular utilizado. Considerando el alto valor de n calculado, la cantidad de sumas es extremadamente grande y por lo tanto el error toma un valor incluso mayor al de truncamiento para dicho n. Todo esto considerando al n original pedido para acotar al error de truncamiento.

3. Resultados

3.1. Estimación de μ

Con el algoritmo utilizado se llego al resultado 1×10^{-8} para el μ de precisión simple y 1×10^{-16} para el μ de precisión doble.

3.2. Función y sus derivadas

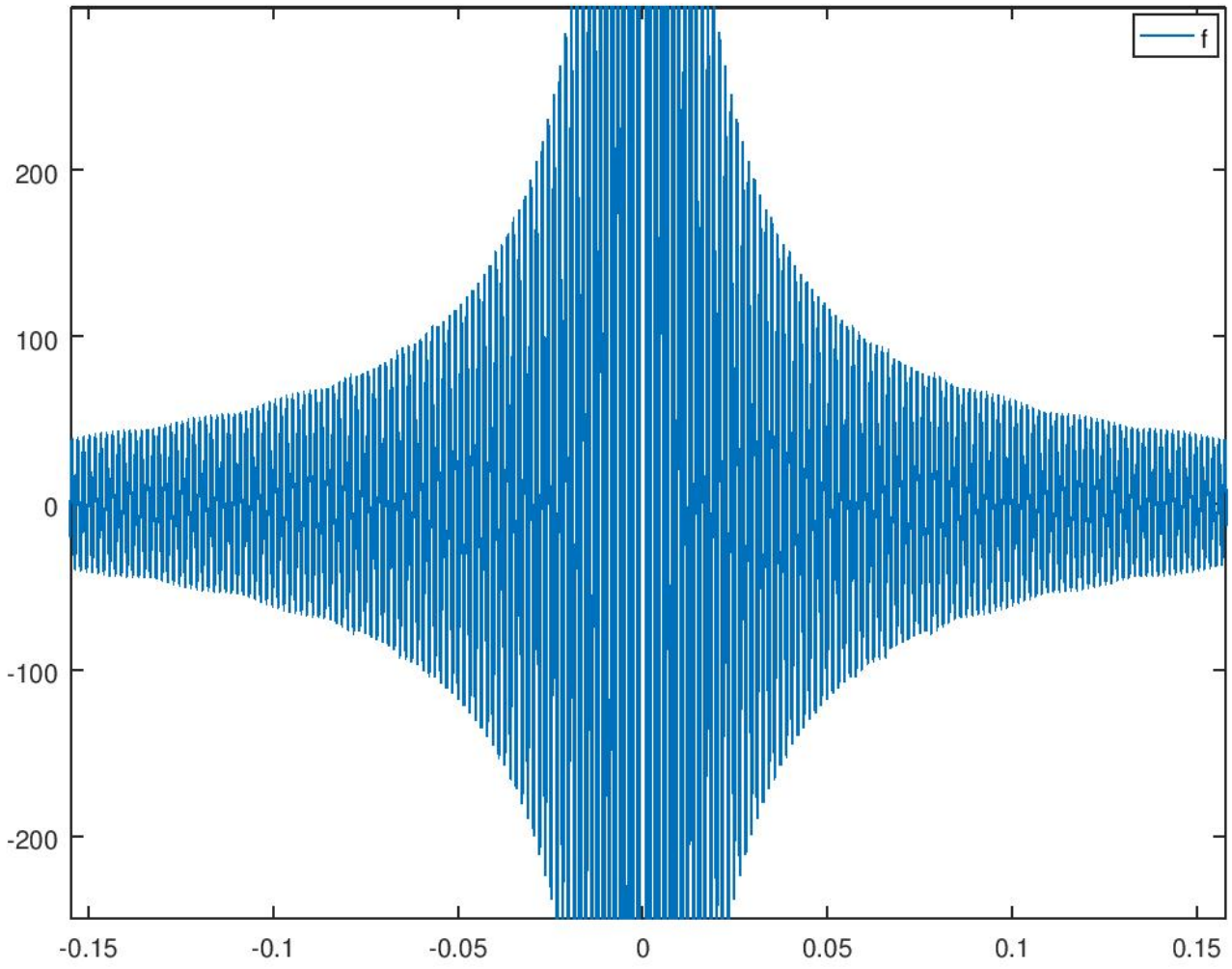

Figura 1: $f(x)$

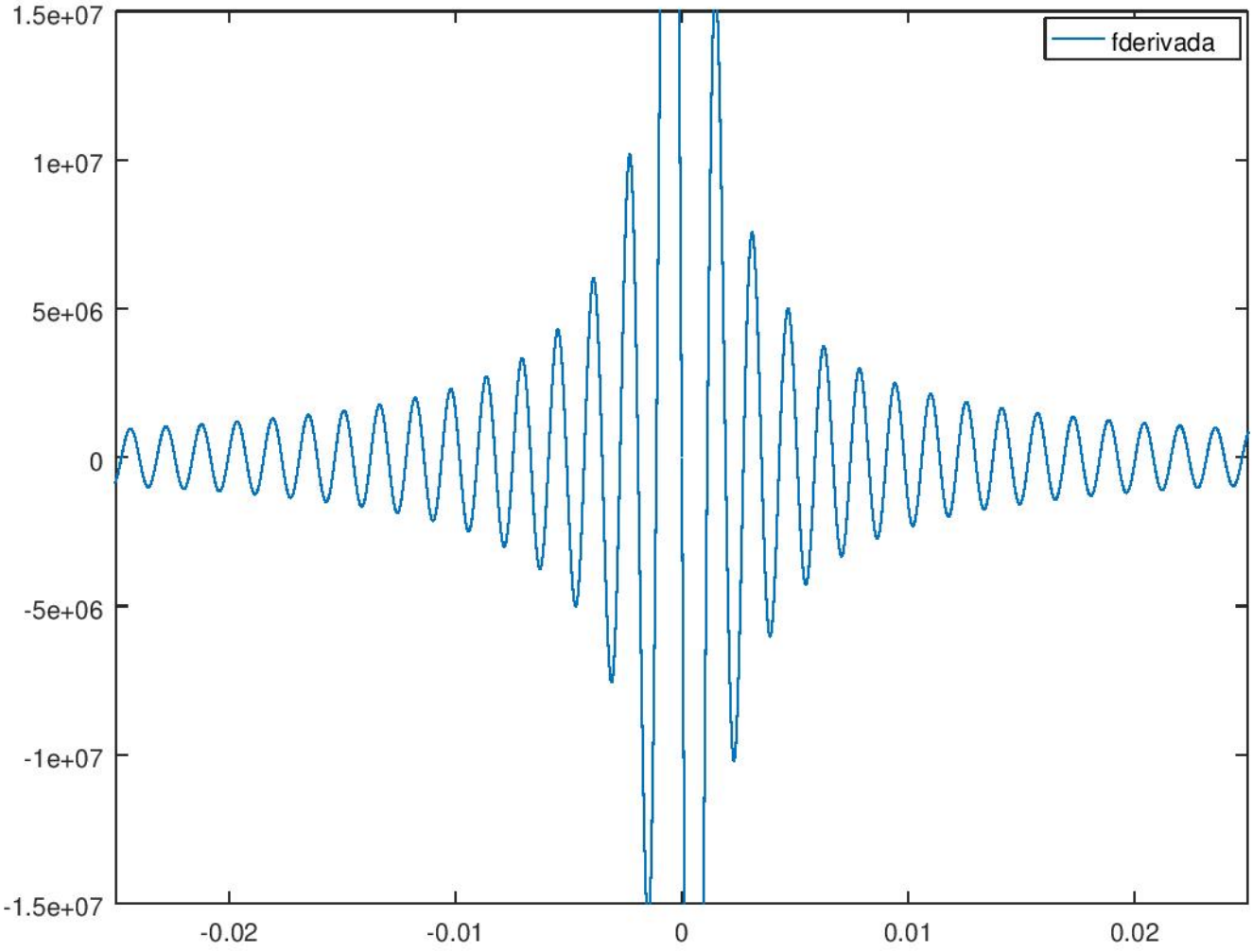

Figura 2: $f'(x)$

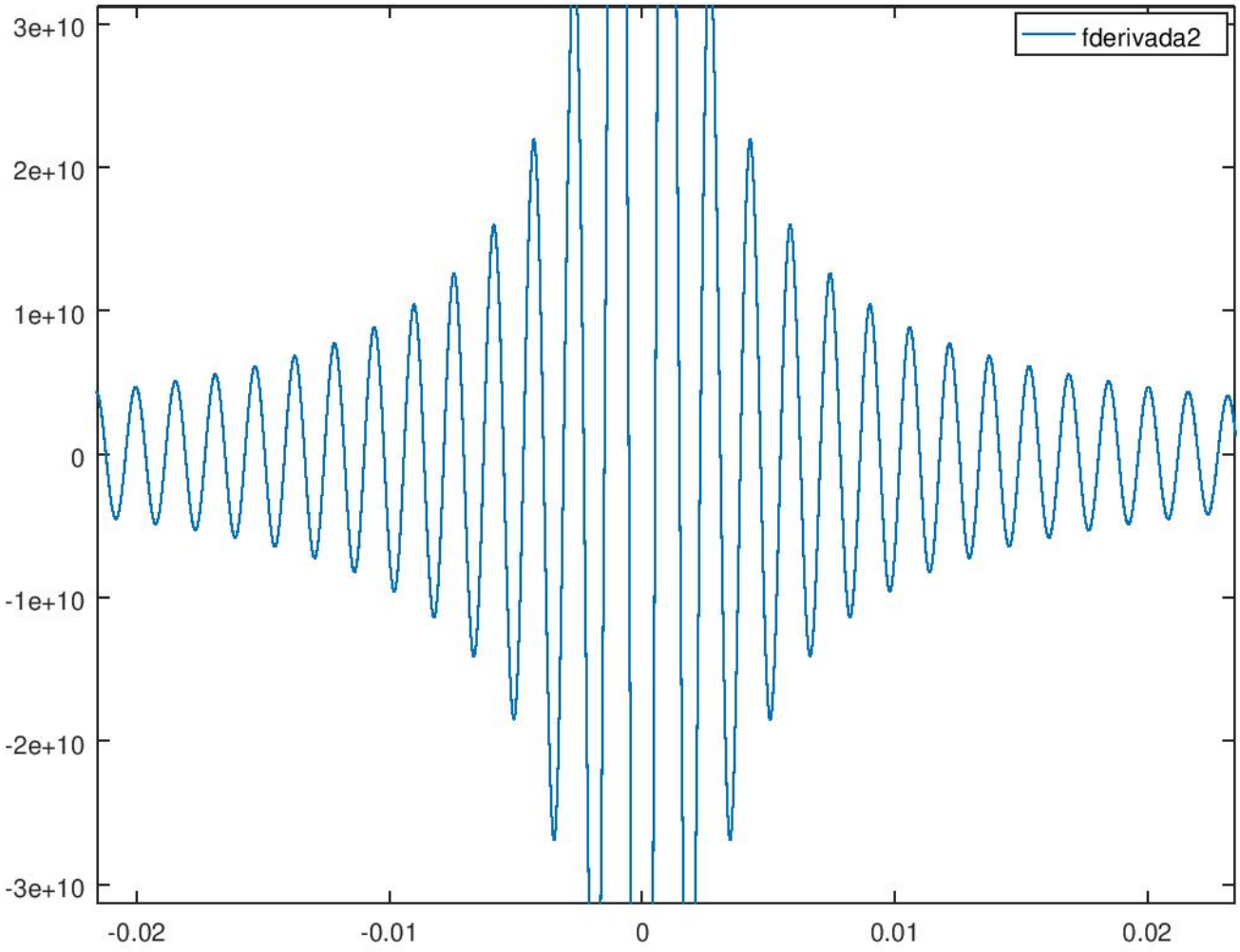

Figura 3: $f''(x)$

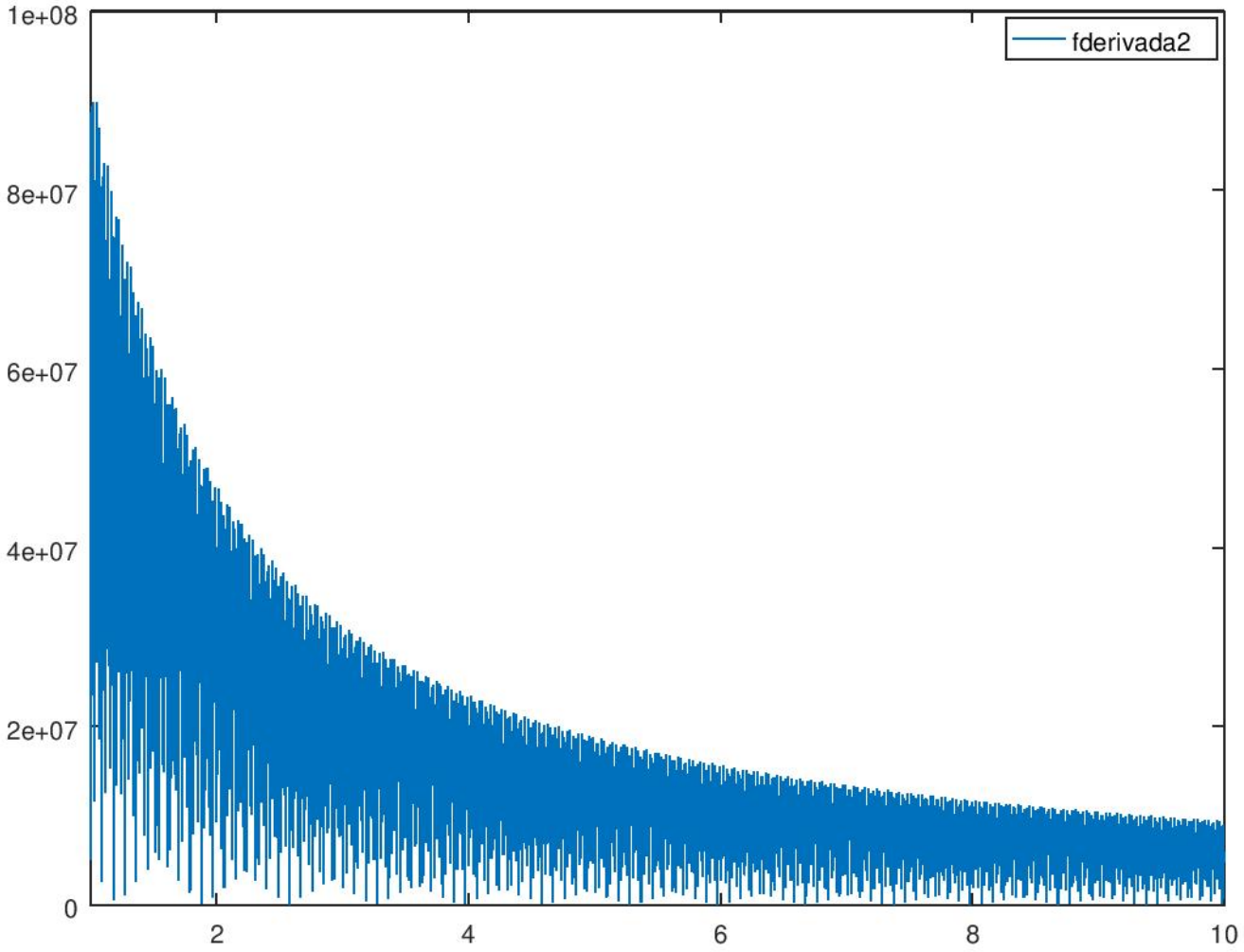

Figura 4: $f''(x)$, con limites 1 a 100 para poder ver como claramente el 1 es el máximo valor de su imagen.

3.3. Método de trapecios compuestos

3.3.1. Truncamiento de n

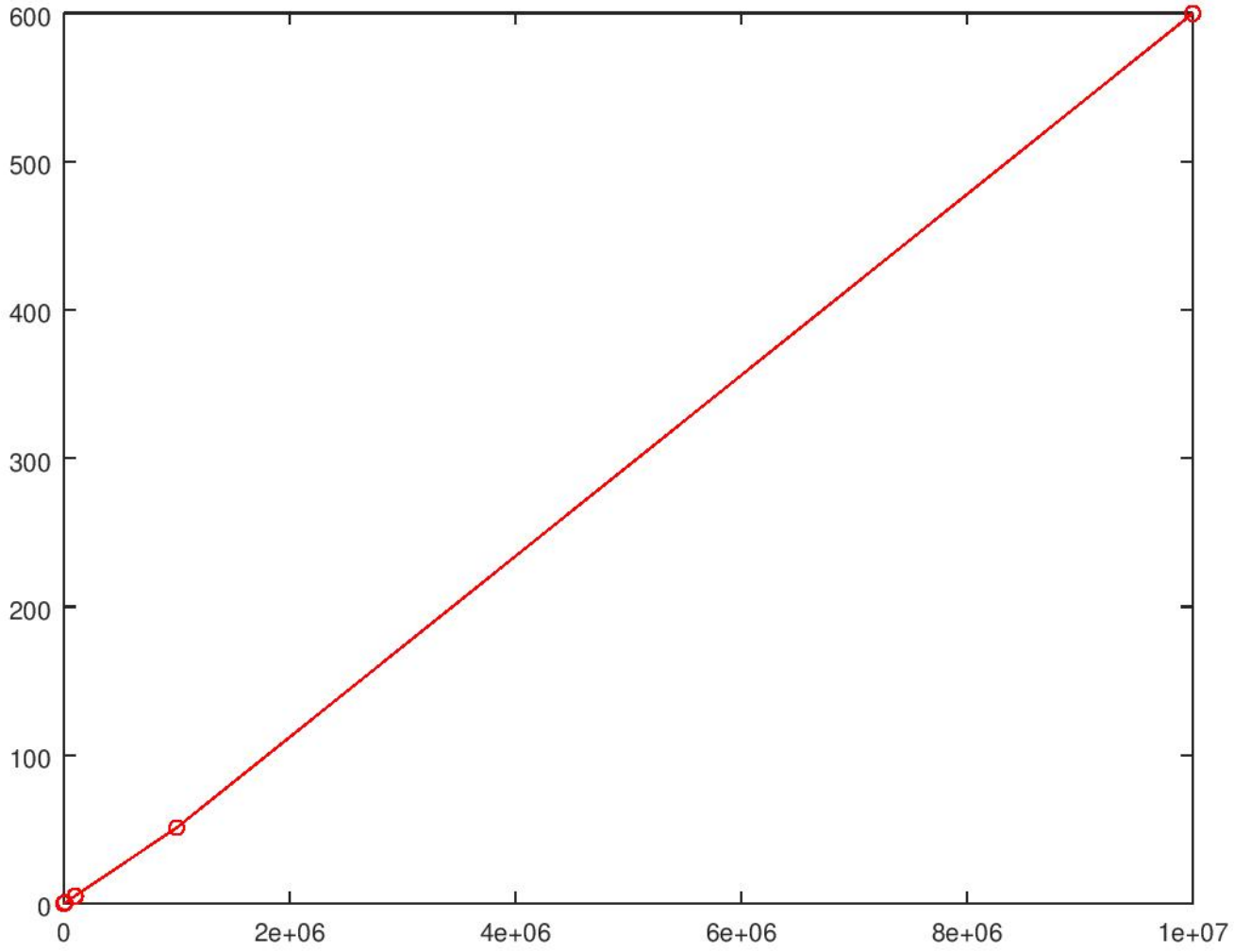

Figura 5: Calculo de la integral para distintos n en funcion del tiempo

3.3.2. Precisión simple

Haciendo los calculos se vió que n es igual a 241403216, y luego se decidió truncar el número n a 5000000. Para el n original la integral dió como resultado 6.9458×10^4, mientras que para n truncado dió 6.9351×10^4.

3.3.3. Precisión doble

Haciendo los calculos se vió que n es igual a 2.4160×10^8, y luego se decidió truncar el número n a 5000000 para que sea un cálculo tratable. Para el n truncado la integral dió 6.9458×10^4.

3.4. Condición del problema y término de estabilidad

Con el método de perturbaciones experimentales llegamos a un cp respecto de alpha igual a 28.571 y respecto de beta igual a 4.9627. El término de estabilidad es igual a 1.1201×10^6.

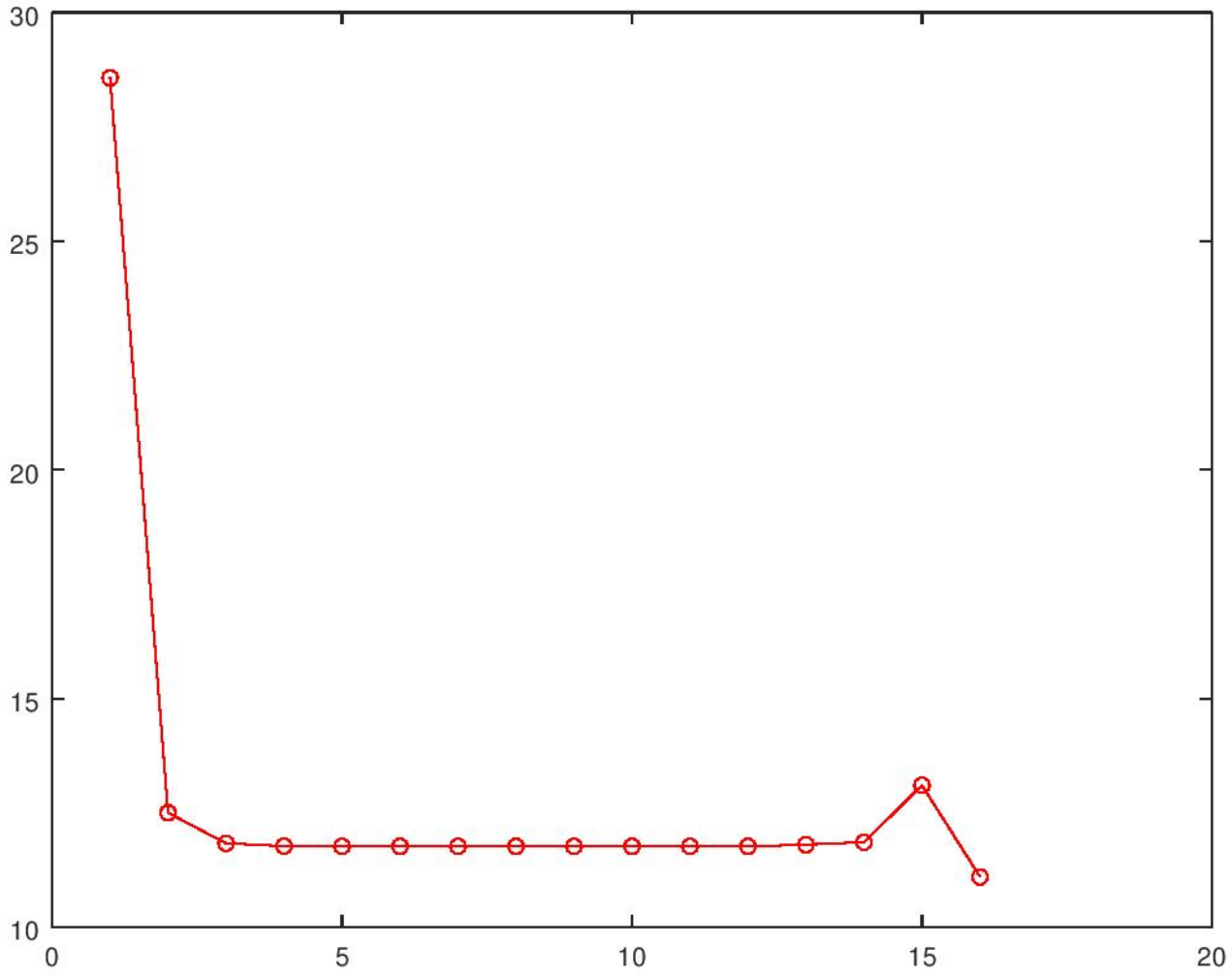

Figura 6: Cp respecto de alpha por iteración

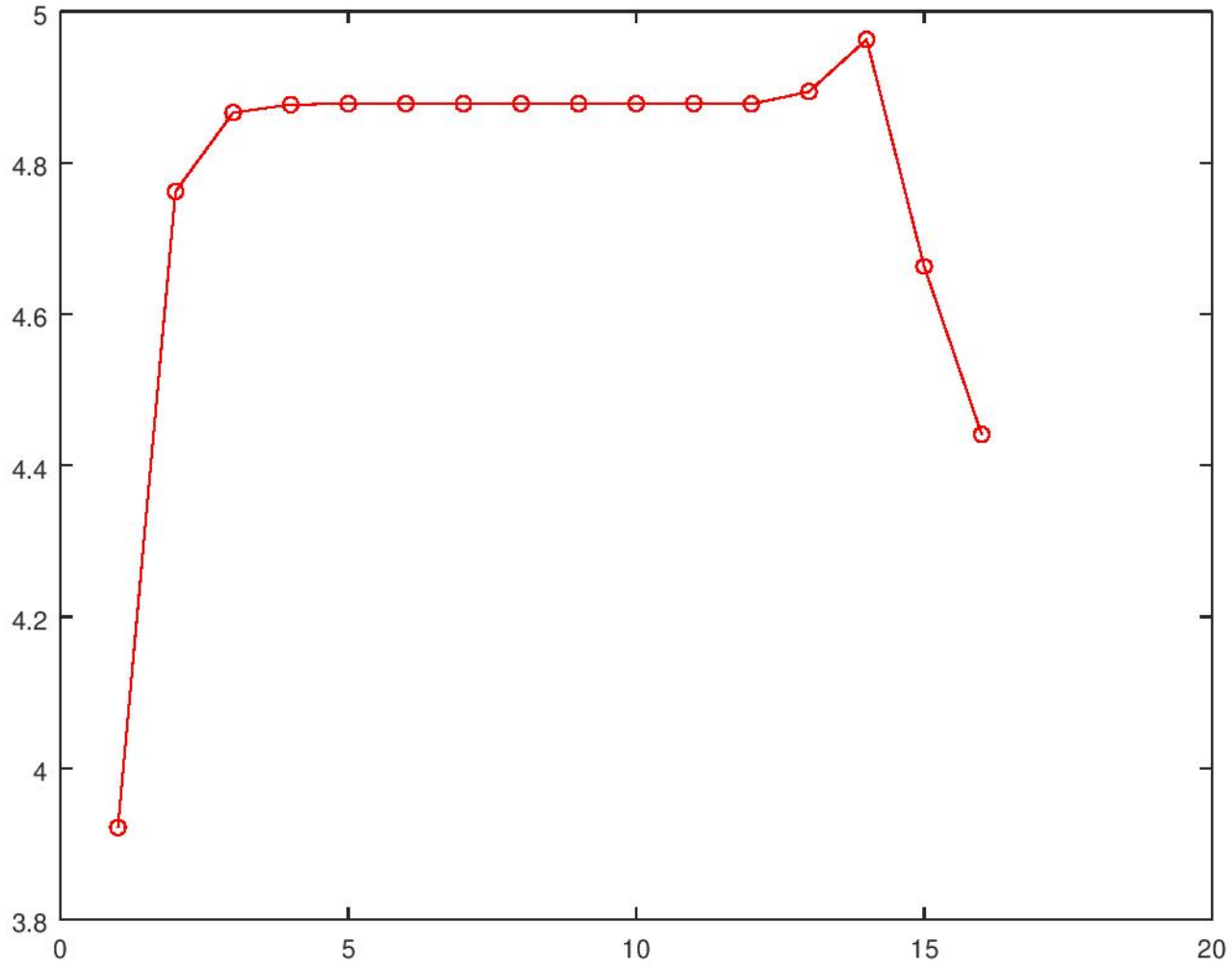

Figura 7: Cp respecto de beta por iteración

3.5. Error total

3.5.1. Error inherente nulo

$$Et = Cp * r + Te * \mu_s + Etr \tag{10}$$

$$Et < 0 + 1{,}1201 * 10^{6} * 1 * 10^{-8} + 10^{-5} \tag{11}$$

$$Et < 0{,}011211 \tag{12}$$

$$Et = Cp * r + Te * \mu_d + Etr \tag{13}$$

$$Et < 0 + 1{,}1201 * 10^{6} * 1 * 10^{-16} + 10^{-5} \tag{14}$$

$$Et < 10^{-5} \tag{15}$$

3.5.2. Error inherente acotado

$$Et = Cp * r + Te * \mu_s + Etr \tag{16}$$

$$Et < 0{,}5 * 10^{-4} + 1{,}1201 * 10^{6} * 1 * 10^{-8} + 10^{-5} \tag{17}$$

$$Et < 0{,}011261 \tag{18}$$

$$Et = Cp * r + Te * \mu_d + Etr \tag{19}$$

$$Et < 0{,}5 * 10^{-4} + 1{,}1201 * 10^{6} * 1 * 10^{-16} + 10^{-5} \tag{20}$$

$$Et < 6 * 10^{-5} \tag{21}$$

4. Conclusiones

Por empezar podemos concluir que el problema presentado esta mas cerca de estar bien condicionado (pues la condicion del problema no es tan lejana a 1) de lo que está de ser estable. Esto se debe al enorme valor que toma el termino de estabilidad debido a la gran cantidad de sumas que se realizan. Dichas sumas llevan atadas varios errores de redondeo que terminan produciendo el error mas importante del algoritmo. Si bien las cotas de error fueron calculadas con un error de maquina single dicho error puede ser acotado con una mejor precision de maquina (double) y llegar a un resultado de menor error. Concluimos que al aumentar la precision el error de redondeo disminuye y por lo tanto la cota de error total tambien lo hace. Distinto es disminuir la cantidad de cuentas a realizar (o dicho de otra forma, disminuir el valor de n) lo cual si bien disminuye el error de redondeo provocaría una mayor cota en el error de truncamiento. Por lo tanto a la hora de realizar calculos numéricos hay que trabajar con la mejor precisión posible siempre que queramos conseguir cotas de error lo mas pequeñas posibles.

Ahora bien, para un nivel de precisión doble el error de redondeo (al menos para el termino de estabilidad calculado) es basicamente despreciable contra el error de truncamiento. Esto sucede ya que al aumentar la precision, como bien dicho antes, se consiguen mejores cotas de error pues se pierde menos información en cada operación. Esto entonces nos lleva a pensar en que sería lógico buscar conseguir un error de truncamiento aun mas pequeño para que el error de redondeo deje de ser despreciable pero el resultado sea aún mas exacto. Pero esto tiene tambien sus, presentadas anteriormente, desventajas en tanto a tiempo de calculo no manejable. Se podría aumentar el numero de trapecios pero a costa de un tiempo de ejecucion mucho mas elevado y menos manejable. Ahora bien, considerando la cota de error inherente relativo, este error pasa a ser aquel de mayor relevancia pues su magnitud es mayor a la del mismo truncamiento. En este caso no tendría sentido

intentar disminuir el error de truncamiento mucho mas pues, si bien no es totalmente despreciable, reducirlo no es rentable considerando la cantidad de tiempo necesario para realizar dichos calculos y tampoco es rentable pues no es siquiera el error mas relevante de todo el calculo.

Tambien podemos observar que, si bien el termino de estabilidad es muy grande, la diferencia en las medidas de la integral con precision simple y doble no estuvieron tan alejadas como esperado. Esperabamos que la diferencia fuera mayor pues los redondeos iban a afectar de forma mas directa al resultado teniendo menos precision a la hora de calcular pero evidentemente no fué así en nuestro caso. Pasamos gran parte del tiempo intentando identificar un error en el calculo de la integral en precision single sin poder lograrlo por lo que concluimos que o bien nuestra hipotesis era falsa o bien hay un error algoritmico en el calculo con dicho nivel de precision (el cual lamentablemente Octave no facilita su trabajo). Una pequeña conclusión a la cual llegamos luego de dicho tiempo fue que, logicamente, el termino de estabilidad depende directamente del algoritmo utilizado pues, a la hora de implementar el metodo de los trapecios, siempre realizamos sumas de numeros de magnitudes similares (pues, algoritmicamente hablando, calculamos el area de cada trapecio y los sumamos, en lugar de sumar todas las alturas de dichos trapecios y luego multiplicarlos por sus diminutas bases) y si bien los resultados no fueron tampoco demasiado diferentes, se pudo observar que usando un algoritmo distinto al que propusimos nosotros llevaba a un termino de estabilidad aún mas elevado y por lo tanto un error más elevado.

A. Anexo I: Código Fuente

El programa utilizado es GNU Octave, versión 4.2.2 y se procuró utilizar sintaxis compatible con Matlab, teniendo como única excepción la función `fplot` que brinda Octave para graficar funciones, despues de consultar con los docentes.

mu.m

```
function mu
  mu_simple
  mu_doble
end

function mu_doble
  mu_doble=1; digitos=1; x=2;
  while (x>1)
    digitos = digitos+1;
    mu_doble = mu_doble/10;
    x = 1+mu_doble;
  endwhile
  mu_doble
end

function mu_simple
  mu_simple=single(1); digitos=single(1); x=single(2);
  while (x>1)
    digitos = digitos+1;
    mu_simple = mu_simple/10;
    x = 1+mu_simple;
  endwhile
  mu_simple
end
```

main.m

```
function integral = main
  padron1 = 100029; padron2 = 99779;
  global P = (padron1 + padron2) / 50;
  global LIM_INF = 1; global LIM_SUP = 240;
  global ALPHA = 0.17; global BETA = 0.41;
  global ERR_MAX = 10e-5;

  n_sin_truncar = calcular_n(ERR_MAX)
  n = 5000
  integral = calcular_area(n)
  cps = calcular_cps(n);
  cpa = cps(1), cpb = cps(2)
  error_redondeo = calcular_err()
end

function a = calcular_area(n)
  global LIM_SUP LIM_INF;
  h = ( LIM_SUP - LIM_INF ) ./ n;
  f_inicio = f(LIM_INF) / 2;
  f_fin = f(LIM_SUP) / 2;
  f_i = 0;
  for i = 1:n-1;
    f_i = f_i + f(LIM_INF + i*h) * h;
  end
    a = ( f_inicio + f_fin ) * h + f_i;
end

function n = calcular_n(error_maximo_truncamiento)
  global LIM_SUP LIM_INF
  num = - ( (LIM_SUP - LIM_INF)^3 ) * fderivada2(1);
  denom = error_maximo_truncamiento * 12;
  n = sqrt(abs(num/denom));
end

##### TERMINO DE ESTABILIDAD Y CONDICION DEL PROBLEMA #####

function err = calcular_err()
  mu_single = 1.0000e-08;
  integral_d = 6.9458e+04;
  integral_s = 7.0236e+04;
  te = (integral_d.-integral_s)./(integral_d.*(mu_single));
  te = abs(te)
```

```
  err = te.*mu_single;
end

function cps = calcular_cps(n)
  cps_a = []; cps_b = [];
  for i = 1:16;
    perturbacion = 1/(10 .^i);
    cps_a = [cps_a, perturbarA(perturbacion,n)];
    cps_b = [cps_b, perturbarB(perturbacion,n)];
   end
   cps_a
   cps_b
   #plot(1:16,cps_a,'o-r')
   #plot(1:16,cps_b,'o-r')
   cpa = max(cps_a);
   cpb = max(cps_b);
   cps = [cpa,cpb];
end

##### PERTURBACIONES #####

function cp = perturbarA(perturbacion,n)
  global ALPHA
  ALPHA += perturbacion;
  valor_perturbado_sup = calcular_area(n);
  ALPHA -= 2 .*perturbacion;
  valor_perturbado_inf = calcular_area(n);
  ALPHA += perturbacion;
  cp = abs((1 .- (valor_perturbado_inf ./ valor_perturbado_sup)) ./ perturbacion);
end

function cp = perturbarB(perturbacion,n)
  global BETA
  BETA += perturbacion;
  valor_perturbado_sup = calcular_area(n);
  BETA -= 2 .*perturbacion;
  valor_perturbado_inf = calcular_area(n);
  BETA += perturbacion;
  cp = abs((1 .- (valor_perturbado_inf ./ valor_perturbado_sup)) ./ perturbacion);
end

##### FUNCION Y SUS DERIVADAS #####
```

```
function y = f(x)
  global P ALPHA BETA

  y = ( sin(x.*P) + BETA * (x.^2) ) ./ (x.*ALPHA);
end

function y = fderivada(x)
  global P ALPHA BETA

  primer_term = (P./(ALPHA.*x)) .* cos(P.*x);
  segundo_term = - ( ( sin(P.*x) ) ./ ( ALPHA .* (x.^2) ) );
  tercer_term = BETA / ALPHA;
  y = abs(primer_term) + abs(segundo_term) + abs(tercer_term);
end

function y = fderivada2(x)
  global P ALPHA BETA

  primer_term = - (2.*P.*cos(P.*x) ) ./ (ALPHA .* (x.^2) );
  segundo_term = 2* sin(P.*x) ./ (ALPHA .* (x.^3) );
  tercer_term = - ( ( (P^2)*sin(P.*x) ) ./ (ALPHA .* x) );
  y = abs(primer_term) + abs(segundo_term) + abs(tercer_term);
end
```

mainsingle.m

```
function integral = mainsingle
  padron1 = single(100029); padron2 = single(99779);
  global P = (padron1 + padron2) / single(50)
  global LIM_INF = single(1); global LIM_SUP = single(240);
  global ALPHA = single(0.17); global BETA = single(0.41);
  global ERR_MAX = single(10e-5);

  n_sin_truncar = calcular_n(ERR_MAX)
  n = single(5000000)
  integral = calcular_area(n)
end

function a = calcular_area(n)
  global LIM_SUP LIM_INF;
  h = single(( LIM_SUP - LIM_INF ) ./ n);
  f_inicio = single(f(LIM_INF) / 2);
  f_fin = single(f(LIM_SUP) / 2);
  f_i = 0;
  for i = 1:n-1;
    f_i = f_i + f(LIM_INF + i*h) * h;
    f_i = single(f_i);
  end
    a = single(( f_inicio + f_fin ) * h + f_i);
end

function n = calcular_n(error_maximo_truncamiento)
  global LIM_SUP LIM_INF

  num = - ( (LIM_SUP - LIM_INF)^3 ) * fderivada2(1);
  num = single(num);
  denom = error_maximo_truncamiento * 12;
  denom = single(denom);
  n = single(sqrt(abs(num/denom)));
end

##### FUNCION Y SUS DERIVADAS #####

function y = f(x)
  x = single(x);
  global P ALPHA BETA

  y = ( single(sin(x.*P)) + BETA * (x.^2) ) ./ (x.*ALPHA);
```

```
  y = single(y);
end

function y = fderivada(x)
  x = single(x);
  global P ALPHA BETA

  primer_term = (P./(ALPHA.*x)) .* single(cos(P.*x));
  segundo_term = - ( ( single(sin(P.*x)) ) ./ ( ALPHA .* (x.^2) ) );
  tercer_term = BETA / ALPHA;
  y = abs(single(primer_term)) + abs(single(segundo_term)) + abs(single(tercer_term)
      ↪ );
  y = single(y);
end

function y = fderivada2(x)
  x = single(x);
  global P ALPHA BETA

  primer_term = - (2.*P.*single(cos(P.*x)) ) ./ (ALPHA .* (x.^2) );
  segundo_term = 2* single(sin(P.*x)) ./ (ALPHA .* (x.^3) );
  tercer_term = - ( ( (P^2)*single(sin(P.*x)) ) ./ (ALPHA .* x) );
  y = abs(single(primer_term)) + abs(single(segundo_term)) + abs(single(tercer_term)
      ↪ );
  y = single(y);
end
```

Generación de graficos

```
fplot(@f, [-0.02 0.02])
fplot(@fderivada, [-0.02 0.02])
fplot(@fderivada2, [-0.02 0.02])
```

Truncamiento y gráfico de n

```
function y = graficar_n()
    x = [1,10,100,1000,10000,100000,1000000,10000000];
    y = [];
    for n = x
        n
        tic;
        integral = calcular_area(n)
        y = [y,toc];
        printf("Tiempo␣=␣%ds\n\n",y(length(y)))
    end
    plot(x,y,'o-r')
end
```

B. Anexo II: Resultados Numéricos

Corrida de `mu.m`

```
>> mu
mu_simple = 1.0000e-08
mu_doble = 1.0000e-16
```

title

```
>> main
n = 2.4160e+08
Elapsed time is 12404.8 seconds.
ans = 6.9458e+04
```

title

```
\begin{lstlisting}[language=Octave,title=Corrida de \texttt{mu.m}]
n_sin_truncar = 2.4160e+08
n = 5000000
integral = 6.9458e+04
cpa = 28.571
cpb = 4.9627
te = 1.1201e+06
```

title

```
>> mainsingle
n_sin_truncar = 241403216
n = 5000000
integral = 6.9351e+04
```

Cálculos hechos para el criterio de truncamiento de n

```
>> graficar_n
n = 1
integral = 6.9497e+04
Tiempo = 0.000295877s

n = 10
integral = 6.9459e+04
Tiempo = 0.000695944s

n = 100
integral = 6.9465e+04
Tiempo = 0.00530505s
```

```
n = 1000
integral = 6.9465e+04
Tiempo = 0.0516629s

n = 10000
integral = 6.9458e+04
Tiempo = 0.509583s

n = 100000
integral = 6.9458e+04
Tiempo = 5.11305s

n = 1000000
integral = 6.9458e+04
Tiempo = 51.1415s

n = 10000000
integral = 6.9458e+04
Tiempo = 599.773s
```

Cps calculados para las distintas iteraciones

```
cps_a =

Columns 1 through 11:

    28.571 12.500 11.834 11.772 11.765 11.765 11.765 11.765 11.765 11.765 11.765

Columns 12 through 16:

    11.765 11.768 11.857 12.212 11.102

cps_b =

Columns 1 through 11:

    3.9212 4.7614 4.8657 4.8763 4.8774 4.8775 4.8775 4.8775 4.8775 4.8775 4.8776

Columns 12 through 16:

4.8770 4.8683 4.7962 4.7740 4.4409
```

Bibliografía

[1] Gonzales, Hernan: *Análisis Numérico, Primer Curso* Buenos Aires: Nueva Librería, 2002.