Instituto Tecnológico de Costa Rica
Área Académica de Ingeniería en Computadores

Análisis Numérico para Ingeniería - CE3102
Tarea 1

Manual de Usuario
Paquete Octave: ***FunTras***

Profesor: Juan Pablo Soto Quirós
Estudiantes:
Adrián Trejos Salazar
Fabián Crawford Barquero
Irene Muñoz Castro
Luis Pedro Morales Rodríguez
Steven Badilla Soto

I Semestre – 2022

Introducción

El paquete computacional descrito en el presente manual de usuario, corresponde a una colección de métodos (programados en Octave) que aproximan una serie de funciones trascendentes de variable real. Una función trascendente es aquella que no satisface una ecuación polinomial [1]. En este tipo de funciones, la variable independiente figura como exponente, o como índice de la raíz, o se halla afectada por el signo de logaritmo o por cualquiera de los signos que emplea la trigonometría.

Las funciones trascendentes se pueden dividir en elementales y superiores. Las elementales son aquellas que pueden ser expresadas mediante una cantidad finita de operaciones de suma, resta, multiplicación, división, radicación, potenciación a exponentes constantes reales y logaritmación. Mientras que las superiores no pueden cumplir con esta condición [2].

Con este paquete se buscó crear una herramienta para obtener la aproximación numérica de un conjunto de funciones trascendentes elementales de variable real utilizando únicamente las operaciones de suma (+), resta (-), multiplicación (*) y potencia de exponente entero positivo (^). Específicamente, se hizo el desarrollo en GNU Octave para las siguientes funciones:

Tabla 1. Métodos implementados en el paquete FunTras

Función f(x)	***Comando en GNU Octave***	***Función f(x)***	***Comando en GNU Octave***
x^{-1}	`div_t(x)`	e^x	`exp_t(x)`
$\sin(x)$	`sin_t(x)`	$\cos(x)$	`cos_t(x)`
$\tan(x)$	`tan_t(x)`	$\ln(x)$	`ln_t(x)`
$\log_a(x)$	`log_t(x,a)`	a^x	`power_t(x,a)`
$\sinh(x)$	`sinh_t(x)`	$\cosh(x)$	`cosh_t(x)`
$\tanh(x)$	`tanh_t(x)`	$\sqrt{x}$	`sqrt_t(x)`
$\sqrt[a]{x}$	`root_t(x,a)`	$\sin^{-1}(x)$	`asin_t(x)`
$\cos^{-1}(x)$	`acos_t(x)`	$\tan^{-1}(x)$	`atan_t(x)`

Guía de Instalación

Requisitos:

1. Se debe tener instalada una versión estable de GNU Octave (se recomienda la última versión: 6.4.0), cuyos requisitos de sistema y forma de descarga se encuentran en este enlace.
2. Se debe tener instalado el paquete de Octave: symbolic , en su versión 2.9.0 o superior.

Instalación de FunTras:

1. La descarga del paquete FunTras, se puede hacer directamente desde este enlace o a partir del repositorio de GitHub donde se puede analizar su código fuente.
2. Una vez descargado el archivo *.tar,* se debe abrir una terminal de Octave en ese mismo directorio y ejecutar el siguiente comando:

```
pkg install FunTras.tar.gz
```

3. Posteriormente, se puede verificar que la instalación haya sido correcta, ejecutando el siguiente comando para listar todos los paquetes instalados y verificando que aparezca *funtras.*

```
pkg list
```

```
>> pkg list
Package Name  | Version | Installation directory
--------------+---------+-----------------------
    funtras   |   1.0.0 | /home/luispedro/snap/octave/245/octave/funtras-1.0.0
   symbolic   |   2.9.0 | /home/luispedro/snap/octave/245/octave/symbolic-2.9.0
```

Uso de FunTras

1. Para poder empezar a utilizar las funciones del paquete, este debe ser cargado primero mediante el siguiente comando:

```
pkg load funtras
```

```
>> pkg load funtras
>> sin_t(1)
Symbolic pkg v2.9.0: Python communication link active, SymPy v1.5.1.
ans = 0.8415
>> |
```

Funciones implementadas

En esta sección, se especifican los métodos iterativos implementados para generar las funciones que se especifican en la tabla 1, además de los parámetros, restricciones, casos de uso y otras particularidades de cada función. Para su implementación, cada método iterativo utiliza una tolerancia de 10^{-8}, además de una cantidad máxima de iteraciones de 2500.

div_t

El código fuente de este método se encuentra en el siguiente enlace: div_t

Esta función recibe como parámetro un número ***x=a*** y calcula el inverso de dicho número:

$$f(a) = a^{-1}.$$

Para aproximar esta función, se utilizó la iteración:

$$x_{k+1} = x_k(2 - a \cdot x_k).$$

Donde x_0 es el valor inicial dado por:

$$x_0 = \begin{cases} \text{eps}^{15} & \text{si } 80! < a \leq 100! \\ \text{eps}^{11} & \text{si } 60! < a \leq 80! \\ \text{eps}^{8} & \text{si } 40! < a \leq 60! \\ \text{eps}^{4} & \text{si } 20! < a \leq 40! \\ \text{eps}^{2} & \text{si } 0! < a \leq 20! \end{cases}.$$

eps es una variable ya definida en GNU Octave y representa la precisión relativa de punto flotante (2.2204×10^{-16}). Para esta función, el criterio de parada implementado fue el siguiente:

$$|(x_{k+1} - x_k)| < tol|x_{k+1}|.$$

Restricción: el número que recibe la función no puede ser 0.

Ejemplos:

```
>> div_t(8)
ans = 0.1250
>> div_t(-4)
ans = -0.2500
```
.

exp_t

El código fuente de este método se encuentra en el siguiente enlace: exp_t

Esta función recibe como parámetro un número ***x=a*** y calcula la imagen de dicho número aplicando la siguiente función:

$$f(a) = e^a.$$

Para aproximar dicha función se utilizó el polinomio:

$$S_k(a) = \sum_{n=0}^{k} \frac{a^n}{n!}.$$

Esta función utiliza el siguiente criterio de parada:

$$|S_{k+1}(a) - S_k(a)| < tol.$$

Ejemplos:

```
>> exp_t(2)
ans = 7.3891
>> exp_t(-2)
ans = 0.1353
>> exp_t(0)
ans = 1
```

sin_t

El código fuente de este método se encuentra en el siguiente enlace: sin_t

Esta función recibe como parámetro un número ***x=a*** y calcula la imagen de dicho número aplicando la siguiente función:

$$f(a) = \sin(a)\,..$$

Para aproximar dicha función se utilizó el polinomio:

$$S_k(a) = \sum_{n=0}^{k} (-1)^n \frac{a^{2n+1}}{(2n+1)!}.$$

Esta función utiliza el siguiente criterio de parada:

$$|S_{k+1}(a) - S_k(a)| < tol.$$

Ejemplos:

```
>> sin_t(3)
ans = 0.1411
>> sin_t(0)
ans = 0
>> sin_t(pi_t()/2)
ans = 1.0000
```

cos_t

El código fuente de este método se encuentra en el siguiente enlace: cos_t

Esta función recibe como parámetro un número ***x=a*** y calcula la imagen de dicho número aplicando la siguiente función:

$$f(a) = \cos(a)\ .$$

Para aproximar dicha función se utilizó el polinomio:

$$S_k(a) = \sum_{n=0}^{k} (-1)^n \frac{a^{2n}}{(2n)!}\ .$$

Esta función utiliza el siguiente criterio de parada:

$$|S_{k+1}(a) - S_k(a)| < tol\ .$$

Ejemplos:

```
>> cos_t(-2)
ans = -0.4161
>> cos_t(0)
ans = 1
>> cos_t(pi_t()/2)
ans = -6.5134e-11
```

tan_t

El código fuente de este método se encuentra en el siguiente enlace: tan_t

Esta función recibe como parámetro un número ***x=a*** y calcula la imagen de dicho número aplicando la siguiente función (utilizando los métodos `sin_t` y `cos_t`):

$$f(a) = \frac{\sin(a)}{\cos(a)}\ .$$

Restricción: el parámetro recibido no puede ser un múltiplo de $\pm\frac{\pi}{2}$ porque la función se indefine.

Ejemplos:

```
>> tan_t(4)
ans = 1.1578
>> tan_t(pi_t())
ans = -1.0348e-11
>> tan_t(pi_t()/2)
error: x must be different than a multiple of +-pi/2
error: called from
    tan_t at line 10 column 5
```

ln_t

El código fuente de este método se encuentra en el siguiente enlace: ln_t

Esta función recibe como parámetro un número ***x=a*** y calcula la imagen de dicho número aplicando la siguiente función:

$$f(a) = \ln(a) .$$

Para aproximar dicha función se utilizó el polinomio:

$$S_k(a) = \frac{2(a-1)}{a+1} \sum_{n=0}^{k} \frac{1}{2n+1} \left(\frac{a-1}{a+1} \right)^{2n} .$$

Esta función utiliza el siguiente criterio de parada:

$$|S_{k+1}(a) - S_k(a)| < tol .$$

Restricciones:

- El dominio de ***x*** debe ser]0, ∞]

Ejemplos:

```
>> ln_t(2)
ans = 0.6931
>> ln_t(1)
ans = 0
>> ln_t(0.5)
ans = -0.6931
```

log_t

El código fuente de este método se encuentra en el siguiente enlace: log_t

Esta función recibe como parámetros un número ***x*** y una base ***a*** :`log_t(x,a)`, luego calcula la imagen de ***x*** aplicando la siguiente función (utilizando el método `ln_t`):

$$\log_a(x) = \frac{\ln(x)}{\ln(a)} .$$

Restricciones:

- El dominio de ***x*** debe ser]0, ∞]
- La base debe pertenecer al conjunto $R^+ - \{1\}$

Ejemplos:

```
>> log_t(5,10)
ans = 0.6990
>> log_t(2,2)
ans = 1
>> log_t(2,1)
error: invalid base
error: called from
    log_t at line 11 column 5
>> log_t(-1,1)
error: x value out of dominium
error: called from
    log_t at line 9 column 5
```

power_t

El código fuente de este método se encuentra en el siguiente enlace: power_t

Esta función recibe como parámetro un número ***x*** y una base ***a*** y calcula la imagen aplicando la siguiente función:

$$a^x = e^x(x \cdot \ln(a)) .$$

Para aproximar dicha función se utilizaron las aproximaciones de las funciones e^x y ln(x).

Esta función utiliza el siguiente criterio de parada en ambas funciones descritas anteriormente, el cuál es:

$$|S_{k+1}(a) - S_k(a)| < tol$$

Restricciones:

- La base debe pertenecer al conjunto R^+.

Ejemplos:

```
>> power_t(-2,2)
ans = 0.2500
>> power_t(-2,9)
ans = 0.012346
>> power_t(2,9)
ans = 81.000
```

sinh_t

El código fuente de este método se encuentra en el siguiente enlace: sinh_t

Esta función recibe como parámetro un número ***x=a*** y calcula la imagen de dicho número aplicando la siguiente función:

$$f(a) = \sinh(a) .$$

Para aproximar dicha función se utilizó el polinomio:

$$S_k(a) = \sum_{n=0}^{k} \frac{a^{2n+1}}{(2n+1)!}$$

Esta función utiliza el siguiente criterio de parada:

$$|S_{k+1}(a) - S_k(a)| < tol$$

Ejemplos:

```
>> sinh_t(-3)
ans = -10.01787492720147
>> sinh_t(0)
ans = 0
>> sinh_t(5)
ans = 74.20321057776711
```

cosh_t

El código fuente de este método se encuentra en el siguiente enlace: cosh_t

Esta función recibe como parámetro un número ***x=a*** y calcula la imagen de dicho número aplicando la siguiente función:

$$f(a) = \cosh(a)\,.$$

Para aproximar dicha función se utilizó el polinomio:

$$S_k(a) = \sum_{n=0}^{k} \frac{a^{2n}}{(2n)!}$$

Esta función utiliza el siguiente criterio de parada:

$$|S_{k+1}(a) - S_k(a)| < tol$$

Ejemplos:

```
>> cosh_t(0)
ans = 1
>> cosh_t(-2)
ans = 3.762195691042252
>> cosh_t(6)
ans = 201.7156361224247
```

tanh_t

El código fuente de este método se encuentra en el siguiente enlace: tanh_t

Esta función recibe como parámetro un número ***x=a*** y calcula la imagen de dicho número aplicando la siguiente función (utilizando los métodos `sinh_t` y `cosh_t`):

$$f(a) = \frac{\sinh(a)}{\cosh(a)}\,.$$

Ejemplos:

```
>> tanh_t(-0.5)
ans = -0.462117157249354
>> tanh_t(0)
ans = 0
>> tanh_t(2)
ans = 0.964027580085263
>> tanh_t(7)
ans = 0.999998336944222
```

sqrt_t

El código fuente de este método se encuentra en el siguiente enlace: sqrt_t

Esta función recibe como parámetro un número ***x=a*** y calcula la imagen aplicando la siguiente función:

$$\sqrt{x} = x^{\frac{1}{2}}.$$

Para aproximar dicha función se utilizaron la aproximación de la función a^x.

Esta función utiliza el siguiente criterio de parada:

$$|S_{k+1}(a) - S_k(a)| < tol .$$

Restricciones:

- La x debe pertenecer al conjunto [0, ∞].

Ejemplos:

```
>> sqrt_t(6)
ans = 2.4495
>> sqrt_t(1)
ans = 1
>> sqrt_t(0)
result = 0
```

root_t

El código fuente de este método se encuentra en el siguiente enlace: root_t

Esta función recibe como parámetros un número ***x*** y una raíz ***a*** : `root_t(x,a)`. Para aproximar la función $f(a) = \sqrt[p]{a}$, se aproxima el cero positivo de la siguiente función, utilizando el método de Newton-Raphson:

$$g(x) = x^p - a .$$

Para esta sucesión se utiliza un valor inicial: $x_0 = \frac{a}{2}$.

El criterio de parada de la función es:

$$|(x_{k+1} - x_k)| < tol|x_{k+1}| .$$

Restricción: si la raíz es par, no se pueden calcular sobre números negativos

asin_t

El código fuente de este método se encuentra en el siguiente enlace: asin_t

Esta función recibe como parámetro un número ***x=a*** y calcula la imagen de dicho número aplicando la siguiente función:

$$f(x)= \sin^{-1}(x)$$

Para aproximar dicha función se utilizó el polinomio:

$$S_k(a) = \sum_{n=0}^{k} \frac{(2n)!}{4^n (n!)^2 (2n+1)} a^{2n+1}.$$

Esta función utiliza el siguiente criterio de parada:

$$|S_{k+1}(a) - S_k(a)| < tol.$$

Restricciones:

- El dominio de ***x*** debe ser [-1, 1]

Ejemplos:

```
>> asin_t(0.5)
ans = 0.523598774479260
>> asin_t(0)
ans = 0
>> asin_t(-0.8)
ans = -0.927295203740429
```

acos_t

El código fuente de este método se encuentra en el siguiente enlace: acos_t

Esta función recibe como parámetro un número ***x=a*** y calcula la imagen de dicho número aplicando la siguiente función (utilizando los métodos `pi_t` y `asin_t`):

$$\cos^{-1}(x) = \frac{\pi}{2} - \sin^{-1}(x).$$

Restricciones:

- El dominio de ***x*** debe ser [-1, 1]

Ejemplos:

```
>> acos_t(-0.8)
ans = 2.498091530535326
>> acos_t(0)
ans = 1.570796326794897
>> acos_t(0.7)
ans = 0.795398835693962
>> acos_t(2)
error: x value out of dominium, must be part of range [-1,1]
error: called from
    asin_t at line 18 column 5
    acos_t at line 8 column 10
```

atan_t

El código fuente de este método se encuentra en el siguiente enlace: [atan_t]

Esta función recibe como parámetro un número ***x=a*** y calcula la imagen de dicho número aplicando la siguiente función:

$$f(a) = \tan^{-1}(a) \; .$$

Para aproximar dicha función se utilizó el polinomio:

$$S_k(a) = \sum_{n=0}^{k} (-1)^n \frac{a^{2n+1}}{2n+1}$$

Esta función utiliza el siguiente criterio de parada:

$$|S_{k+1}(a) - S_k(a)| < tol$$

Restricciones:

- El dominio de ***x*** debe ser [-1, 1]

Ejemplos:

```
>> atan_t(-0.8)
ans = -0.674740945182285
>> atan_t(0)
ans = 0
>> atan_t(0.6)
ans = 0.540419499190018
```

pi_t

El código fuente de este método se encuentra en el siguiente enlace: pi_t

Esta función aproxima el valor de π utilizando la siguiente fórmula que fue descubierta por los hermanos Chudnovsky [3], quienes se basaron en la fórmula de Ramanujan:

$$\mathrm{S}=\sum_{n=0}^{\infty}(-1)^n\frac{(6n)!(k_2+nk_1)}{(n!)^3(3n)!(8k_4k_5)^n}$$

$$\pi=\frac{k_6\sqrt{k_3}}{S}$$

Donde $k_1 = 545140134$, $k_2 = 13591409$, $k_3 = 640320$, $k_4 = 100100025$, $k_5 = 327843840$, $k_6 = 53360$

Esta función utiliza el siguiente criterio de parada:

$$|S_{k+1}(a) - S_k(a)| < tol$$

Esta es conocida por ser una de las aproximaciones que convergen más rápidamente al valor de π, al obtener 14 dígitos por aproximación.

Ejemplos:

```
>> pi_t()
ans = 3.141592653589794
```

test_funtras

El código fuente de este método se encuentra en el siguiente enlace: test_funtras

La función test_funtras utiliza distintas funciones de la librería fun_tras para retornar el resultado de la siguiente función:

$$\frac{\sqrt[3]{\sin\left(\frac{3}{7}\right)+\ln(2)}}{\sinh(\sqrt{2})}+\tan^{-1}(e^{-1})$$

Cuyo resultado aproximado corresponde a:

```
>> test_funtras()

ans = 0.887378862317666
```

Referencias

[1] E. J. Townsend, Functions of a Complex Variable, BiblioLife, LLC, (2009).

[2] Gómez Gómez, Doralia. Variable Compleja. Escuela de Ingeniería Eléctrica. Universidad de La Habana. p. 96.

[3] D. V. Chudnovsky and G. V. Chudnovsky, Approximations and complex multiplication according to Ramanujan, in Ramanujan Revisited, Academic Press Inc., Boston, (1988), p. 375-396 & p. 468-472.