Económicas, UBA. Actuario. Análisis Numérico.

Cuatrimestre 1, 2021. RECUPERATORIO del Primer Examen Parcial.
Para aprobar, debe sumar 50 puntos.

Remplace este texto por su Apellido y Nombre, y su Numero de Registro

02/julio/2021

Contents


```
## Warning: package 'flextable' was built under R version 4.1.1

## Warning: package 'tidyverse' was built under R version 4.1.1

## Warning: package 'tibble' was built under R version 4.1.1

## Warning: package 'tidyr' was built under R version 4.1.1

## Warning: package 'readr' was built under R version 4.1.1

## Warning: package 'purrr' was built under R version 4.1.1

## Warning: package 'dplyr' was built under R version 4.1.1

## Warning: package 'stringr' was built under R version 4.1.1

## Warning: package 'forcats' was built under R version 4.1.1
```

1 Resolución de Ecuaciones: Secante. (24 puntos)

Considere la siguiente ecuación: $2cos(x) = e^{-10/x}$.

1.1 Corregir algoritmo

CORREGIR el algoritmo "Secante" en el siguiente bloque de código.

COMENTAR los cambios que realizó (use "#" al final de cada línea modificada).

[***Notar que, para el resto del ejercicio, no debe utilizar su propio algoritmo, sino que debe usar el algoritmo dado y corregido, sin agregar ninguna línea de código adicional.***]

Respuesta:

```
# Edite las líneas que considere erróneas
# Comente al final de cada línea editada
Secante <- function(f,p0,p1,TOL,N){
  i <- 2
  q0 <- f(p0)
  q1 <- f(p1)
  while (i == N){
    p = p1 - q1*(q1-q0)/(p1-p0)
    if (abs(q-q1) < TOL){
      return(p)
    }
    i = i + 1
    p0 = p
    q0 = q1
    p1 = p0
    q1 = f(p0)
  }
  return(paste('El metodo fallo luego de ', n, ' iteraciones'))
}
```

1.2 Graficar función

Plantee la ecuación de la forma $f(x) = 0$ y grafique la función en el intervalo $[0; 20]$ de manera tal que pueda identificar todas las soluciones de la ecuación en el intervalo.

Respuesta:

```
# Ingrese en este bloque todo el código necesario para resolver el ejercicio 1.2
```

1.3 Hallar raíces

Utilizando el algoritmo del punto 1.1, halle todas las raíces identificadas en el punto 1.2.

[***OBSERVACIÓN***: Debe usar el algoritmo dado en 1.1 (corregido por usted). No puede usar su propio algoritmo.]

Respuesta:

```
# Ingrese en este bloque todo el código necesario para resolver el ejercicio 1.3
```

1.4 Iteraciones

Tome el algoritmo del punto 1.1 (copie y pegue) y agregue las líneas de código que considere necesarias para poder visualizar (imprimir) cada iteración del algoritmo. Una vez editado el algoritmo, imprima 7 iteraciones del algoritmo iniciando en $x_0 = 16.25$ y $x_1 = 17.25$. ¿A cuál de las raíces convergeria el algortimo en este caso?

Respuesta:

```
# Ingrese en este bloque todo el código necesario para resolver el ejercicio 1.4
```

2 Resolución de Ecuaciones: Punto Fijo. (24 puntos)

Para este ejercicio, considere la función $h(x) = x^2 \times cos(x)$.

2.1 Teoría

Describa el método de punto fijo que se utiliza para hallar raíces de funciones.

Respuesta (escriba a continuación):

2.2 Hallar raíces

Halle todas las raíces de $h(x)$ en el intervalo $[-5, 5]$ utilizando el método de ***Punto Fijo***.

[***OBSERVACIÓN***: Recuerde hallar en primer lugar la(s) función(es) $g(x)$ y chequee el cumplimiento de las condiciones de existencia.]

Respuesta:

```
# Ingrese en este bloque todo el código necesario para resolver el ejercicio 2.2
```

2.3 Iteraciones

Considere el punto anterior. Realice 6 iteraciones del algoritmo de Punto Fijo, utilizando en $x_0 = -0.5$. ¿A cuál de las raíces convergeria el algortimo en este caso?

Respuesta:

```
# Ingrese en este bloque todo el código necesario para resolver el ejercicio 2.3
```

2.4 Graficar $g(x)$ y marcar sus puntos fijos

Grafique la función o las funciones $g(x)$ (que utilizó para hallar las raíces de $h(x)$) e identifique los ***puntos fijos*** hallados en el punto 2.2. En el gráfico (o en cada gráfico), marque las coordenadas de cada punto fijo $g(x) = x$ con un punto de color rojo.

Respuesta:

```
# Ingrese en este bloque todo el código necesario para resolver el ejercicio 2.4
```

3 Factorización de Matrices. (12 puntos)

3.1 Factorización de Cholesky

Realice la factorización de Cholesky de la siguiente matriz:

$$A = \begin{bmatrix} 1 & -0.0126 & 0.012 \\ -0.0126 & 1 & 0.0206 \\ 0.012 & 0.0206 & 1 \end{bmatrix}$$

Respuesta:

```
# Ingrese en este bloque todo el código necesario para resolver el ejercicio 3.1
```

3.2 Factorización LU

Realice la factorización de LU de la siguiente matriz:

$$B = \begin{bmatrix} 17 & 15 & 30 & 20 \\ 22 & 23 & 27 & 8 \\ 13 & 19 & 22 & 20 \\ 14 & 18 & 14 & 27 \end{bmatrix}$$

Respuesta:

```
# Ingrese en este bloque todo el código necesario para resolver el ejercicio 3.2
```

4 Interpolación (40 Puntos)

Considere la siguente tabla de datos:

x	y
0.0	0.0000
0.1	0.3205
0.2	0.4175
0.3	0.4916
0.4	0.5555
0.5	0.6143
0.6	0.6708
0.7	0.7276
0.8	0.7877
0.9	0.8574
1.0	1.0000

4.1 Polinomio de Newton: $P_N(x)$

Escriba el Polinomio interpolante de Newton, $P_n(x)$, que pasa por todos los puntos dados.

Respuesta:

```
 # Ingrese en este bloque todo el código necesario para resolver el ejercicio 4.1

# Metodo de diferencias divididas -----------------------------------------
DiferenciasDivididas <- function(x, y){
  n <- length(x)

  #Hago un vector vacio para llenar el df
  empty_vec <- rep(0, times = n)

  df <- data.frame(x, y)

  for (i in 1:(n-1)) {

    df[glue::glue("Q",i)] <- empty_vec

    for (j in (i+1):n) {

      df[j, (i+2)] <- ( df[j,(i+1)] - df[(j-1),(i+1)])/(x[j]-x[j-i])
    }
  }

  return(df)
}

# Polinomio interpolante de Newton -----------------------------------------
```

```r
PolinomioInterpolanteNewton <- function(x, y){
  df <- DiferenciasDivididas(x = x, y = y)

  #Saco la primer columna del df
  df[,1] <- NULL

  n <- ncol(df)

  polinomio <- df[1,1]

  for (i in 2:n) {
    polinomio <- polinomio + glue::glue(" + ", df[i,i])
    for (j in 1:(i-1)) {
      polinomio <- polinomio + glue::glue(" * ( x - ", x[j], " )")
    }
  }
  return(polinomio)
}

PolinomioInterpolanteNewton(x = x, y = y)
```

```
## 0 + 3.205 * ( x - 0 ) + -11.175 * ( x - 0 ) * ( x - 0.1 ) + 33.4333333333333 * ( x - 0 ) * ( x - 0.1
```

0 + 3.205 * (x - 0) + -11.175 * (x - 0) * (x - 0.1) + 33.4333333333333 * (x - 0) * (x - 0.1) * (x - 0.2) + -78.2916666666666 * (x - 0) * (x - 0.1) * (x - 0.2) * (x - 0.3) + 150.25 * (x - 0) * (x - 0.1) * (x - 0.2) * (x - 0.3) * (x - 0.4) + -243.055555555555 * (x - 0) * (x - 0.1) * (x - 0.2) * (x - 0.3) * (x - 0.4) * (x - 0.5) + 340.873015873015 * (x - 0) * (x - 0.1) * (x - 0.2) * (x - 0.3) * (x - 0.4) * (x - 0.5) * (x - 0.6) + -421.875 * (x - 0) * (x - 0.1) * (x - 0.2) * (x - 0.3) * (x - 0.4) * (x - 0.5) * (x - 0.6) * (x - 0.7) + 474.537037037044 * (x - 0) * (x - 0.1) * (x - 0.2) * (x - 0.3) * (x - 0.4) * (x - 0.5) * (x - 0.6) * (x - 0.7) * (x - 0.8) + -357.14285714289 * (x - 0) * (x - 0.1) * (x - 0.2) * (x - 0.3) * (x - 0.4) * (x - 0.5) * (x - 0.6) * (x - 0.7) * (x - 0.8) * (x - 0.9)

4.2 Interpolar con $P_N(x)$

Calcule $P_N(0.5213)$.

Respuesta:

```r
# Ingrese en este bloque todo el código necesario para resolver el ejercicio 4.2

eval(expression(0 + 3.205 * ( x - 0 ) + -11.175 * ( x - 0 ) * ( x - 0.1 ) + 33.4333333333333 * ( x - 0 
```

```
## [1] 0.6264739
```

4.3 Cubic Splines: $S_i(x)$

Escriba los trazadores cúbicos, $S_i(x); i = 1, ..., n$ que pasan por todos los puntos dados.Indique claramente qué polinomio S_i se debe usar en cada intervalo de x.

Respuesta:

```r
# Ingrese en este bloque todo el código necesario para resolver el ejercicio 4.3

SplineNatural <- function(x, y){
  n <- length(x)

  # Paso 1
```

```r
h <- rep(NA, times = (n-1))
for (i in 1:(n-1)) {
  h[i] <- x[i+1] - x[i]
}; rm(i)

# Paso 2
alfa <- rep(NA, times = (n-2))
for (i in 2:(n-1)) {
  alfa[i] <- (3/h[i]) * (y[i+1] - y[i]) - (3/h[i-1]) * (y[i] - y[i-1])
}

# Paso 3
mu <- rep(NA, times = n)
zeta <- rep(NA, times = n)
l <- rep(NA, times = n)

mu[1] <- 0
zeta[1] <- 0
l[1] <- 1


# Paso 4
for (i in 2:(n-1)) {
  l[i] <- 2 * (x[i+1] - x[i-1]) - h[i-1] * mu[i-1]
  mu[i] <- h[i]/l[i]
  zeta[i] <- (alfa[i] - h[i-1] * zeta[i-1])/l[i]
}

# Paso 5
l[n] <- 1
zeta[n] <- 0
c <- rep(NA, times = n)
c[n] <- 0

# Paso 6
b <- rep(NA, times = (n-1))
d <- rep(NA, times = (n-1))
for (j in (n-1):1) {
  c[j] <- zeta[j] - mu[j] * c[j+1]
  b[j] <- (y[j+1] - y[j]) / h[j] - h[j] * (c[j+1] + 2 * c[j])/3
  d[j] <- (c[j+1] - c[j]) / (3*h[j])
}

#Paso 7
resultados <- matrix(rep(NA, 4*(n-1)), nrow = (n-1), ncol = 4, byrow = F)
for (k in 1:(n-1)) {
  resultados[k, 1] <- y[k]
  resultados[k, 2] <- b[k]
  resultados[k, 3] <- c[k]
  resultados[k, 4] <- d[k]
}

#Construyo el polinomio
```

```r
  polinomios <- rep(NA, times = nrow(resultados))
  for (i in 1:nrow(resultados)) {
    polinomios[i] <- glue::glue(resultados[i,1])
    for(j in 2:ncol(resultados)){
      polinomios[i] <- polinomios[i] + glue::glue(" + ", resultados[i,j], " * (x - ", x[i], ")^", (j-1)
    }
  }

  return(polinomios)
}

trazadores <- SplineNatural(x = x, y = y)

trazadores
```

```
##  [1] "0 + 3.78915168257157 * (x - 0)^1 + 0 * (x - 0)^2 + -58.4151682571572 * (x - 0)^3"
##  [2] "0.3205 + 2.03669663485686 * (x - 0.1)^1 + -17.5245504771472 * (x - 0.1)^2 + 68.575841285786 *
##  [3] "0.4175 + 0.589061778001004 * (x - 0.2)^1 + 3.04820190858865 * (x - 0.2)^2 + -15.2881968859869
##  [4] "0.4916 + 0.740056253139126 * (x - 0.3)^1 + -1.53825715720743 * (x - 0.3)^2 + 5.27694625816169
##  [5] "0.5555 + 0.590713209442491 * (x - 0.4)^1 + 0.0448267202410806 * (x - 0.4)^2 + -0.71958814665997
##  [6] "0.6143 + 0.578090909090908 * (x - 0.5)^1 + -0.171049723756911 * (x - 0.5)^2 + 0.40140632847822
##  [7] "0.6708 + 0.555923154193873 * (x - 0.6)^1 + -0.0506278252134439 * (x - 0.6)^2 + 1.7139628327472
##  [8] "0.7276 + 0.597216474133601 * (x - 0.7)^1 + 0.463561024610728 * (x - 0.7)^2 + -4.25725765946747
##  [9] "0.7877 + 0.562210949271723 * (x - 0.8)^1 + -0.813616273229512 * (x - 0.8)^2 + 21.6150678051229
## [10] "0.8574 + 1.04793972877951 * (x - 0.9)^1 + 5.67090406830737 * (x - 0.9)^2 + -18.9030135610246 *
```

[0.0; 0.1] 0 + 3.78915168257157 * (x - 0)^1 + 0 * (x - 0)^2 + -58.4151682571572 * (x - 0)^3
[0.1; 0.2] 0.3205 + 2.03669663485686 * (x - 0.1)^1 + -17.5245504771472 * (x - 0.1)^2 + 68.575841285786 * (x - 0.1)^3
[0.2; 0.3] 0.4175 + 0.589061778001004 * (x - 0.2)^1 + 3.04820190858865 * (x - 0.2)^2 + -15.2881968859869 * (x - 0.2)^3
[0.3; 0.4] 0.4916 + 0.740056253139126 * (x - 0.3)^1 + -1.53825715720743 * (x - 0.3)^2 + 5.27694625816169 * (x - 0.3)^3
[0.4; 0.5] 0.5555 + 0.590713209442491 * (x - 0.4)^1 + 0.0448267202410806 * (x - 0.4)^2 + -0.719588146659973 * (x - 0.4)^3
[0.5; 0.6] 0.6143 + 0.578090909090908 * (x - 0.5)^1 + -0.171049723756911 * (x - 0.5)^2 + 0.401406328478224 * (x - 0.5)^3
[0.6; 0.7] 0.6708 + 0.555923154193873 * (x - 0.6)^1 + -0.0506278252134439 * (x - 0.6)^2 + 1.71396283274724 * (x - 0.6)^3
[0.7; 0.8] 0.7276 + 0.597216474133601 * (x - 0.7)^1 + 0.463561024610728 * (x - 0.7)^2 + -4.25725765946747 * (x - 0.7)^3
[0.8; 0.9] 0.7877 + 0.562210949271723 * (x - 0.8)^1 + -0.813616273229512 * (x - 0.8)^2 + 21.6150678051229 * (x - 0.8)^3
[0.9; 1.0] 0.8574 + 1.04793972877951 * (x - 0.9)^1 + 5.67090406830737 * (x - 0.9)^2 + -18.9030135610246 * (x - 0.9)^3

4.4 Interpolar con $S_i(x)$

Usando los trazadores cúbicos, interpole los datos para el valor $x = 0.5213$.

Respuesta:

```r
# Ingrese en este bloque todo el código necesario para resolver el ejercicio 4.4

eval(expression(0.6143 + 0.578090909090908 * (x - 0.5)^1 + -0.171049723756911 * (x - 0.5)^2 + 0.4014063
```

```
## [1] 0.6265396
```

4.5 Graficar

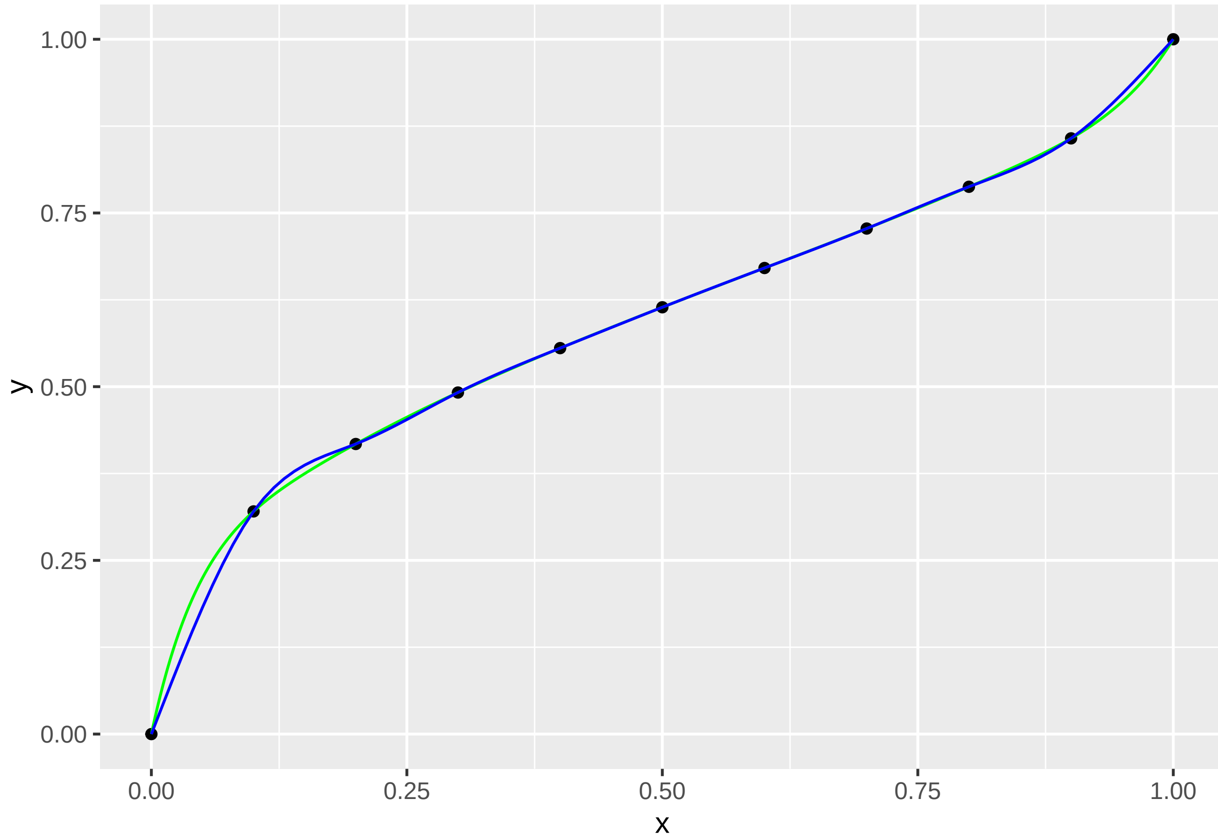
Grafique lo siguiente:

a. Datos dados en la tabla mediante puntos (círculos, rellenos o no).
b. Línea continua de color verde con la función $P_N(x)$ para x en $[0; 1]$.
c. Línea continua de color azul con los trazadores cúbicos para x en $[0; 1]$.

Respuesta:

```
# Ingrese en este bloque todo el código necesario para resolver el ejercicio 4.5

x1 <- seq(from = 0, to = 1, by = 0.001)

y_pn <- eval(expression(0 + 3.205 * ( x - 0 ) + -11.175 * ( x - 0 ) * ( x - 0.1 ) + 33.4333333333333 *

x_sp1 <- seq(from = 0, to = 0.1, by = 0.01)
fx_sp1 <- eval(parse(text = trazadores[1]), list(x = x_sp1))

x_sp2 <- seq(from = 0.1, to = 0.2, by = 0.01)
fx_sp2 <- eval(parse(text = trazadores[2]), list(x = x_sp2))

x_sp3 <- seq(from = 0.2, to = 0.3, by = 0.01)
fx_sp3 <- eval(parse(text = trazadores[3]), list(x = x_sp3))

x_sp4 <- seq(from = 0.3, to = 0.4, by = 0.01)
fx_sp4 <- eval(parse(text = trazadores[4]), list(x = x_sp4))

x_sp5 <- seq(from = 0.4, to = 0.5, by = 0.01)
fx_sp5 <- eval(parse(text = trazadores[5]), list(x = x_sp5))

x_sp6 <- seq(from = 0.5, to = 0.6, by = 0.01)
fx_sp6 <- eval(parse(text = trazadores[6]), list(x = x_sp6))

x_sp7 <- seq(from = 0.6, to = 0.7, by = 0.01)
fx_sp7 <- eval(parse(text = trazadores[7]), list(x = x_sp7))

x_sp8 <- seq(from = 0.7, to = 0.8, by = 0.01)
fx_sp8 <- eval(parse(text = trazadores[8]), list(x = x_sp8))

x_sp9 <- seq(from = 0.8, to = 0.9, by = 0.01)
fx_sp9 <- eval(parse(text = trazadores[9]), list(x = x_sp9))

x_sp10 <- seq(from = 0.9, to = 1, by = 0.01)
fx_sp10 <- eval(parse(text = trazadores[10]), list(x = x_sp10))

ggplot() +
  geom_point(aes(x = x, y = y)) +
  geom_line(aes(x = x1, y = y_pn), colour = "green") +
  geom_line(aes(x = x_sp1, y = fx_sp1), colour = "blue") +
  geom_line(aes(x = x_sp2, y = fx_sp2), colour = "blue") +
  geom_line(aes(x = x_sp3, y = fx_sp3), colour = "blue") +
  geom_line(aes(x = x_sp4, y = fx_sp4), colour = "blue") +
  geom_line(aes(x = x_sp5, y = fx_sp5), colour = "blue") +
```

```
geom_line(aes(x = x_sp6, y = fx_sp6), colour = "blue") +
geom_line(aes(x = x_sp7, y = fx_sp7), colour = "blue") +
geom_line(aes(x = x_sp8, y = fx_sp8), colour = "blue") +
geom_line(aes(x = x_sp9, y = fx_sp9), colour = "blue") +
geom_line(aes(x = x_sp10, y = fx_sp10), colour = "blue") +
xlab("x") + ylab("y")
```



4.6 Comentar Resultados

A partir de lo hallado en el punto anterior, comente sobre las diferencias entre los métodos para realizar aproximaciones de la función entre los puntos dados.

Respuesta (escriba sus comentarios a continuación):