Resumen:

En esta práctica haréis dos programas. El primero lo completaréis y se trata de aplicar lo que se está viendo en clase, la **interpolación de Lagrange**. Se utilizarán muchas de las estructuras explicadas en las anteriores prácticas.

Cuando quede ya menos de la mitad de la clase habréis terminado el programa y pasareis a hacer un programa para calcular una integral por medio de la **fórmula de Simpson**. Este quedará sin terminar, pero solo debes seguir los pasos de Arturo y esperar a que se realice el algoritmo en clase, tras ello podrás hace el programa completo sin problema

Consejos:

1. Antes de venir a esta práctica, revisa lo que habéis hecho en la anterior, te ayudará a ir más rápido en esta pues llevarás fijados los conceptos de sintaxis.
2. Ten presente los algoritmos explicados en clase, esta sesión es la aplicación práctica de esos mismos algoritmos.
3. Todos pasamos por lo mismo que tú. NO te desesperes porque el programa no te funciona y te da muchos errores, es normal. Relájate y lee los errores que has tenido.
4. Es posible que el programa no de errores, pero no haga nada. Revisa los paréntesis.
5. El programa que estáis haciendo no es fácil, debes prestarle atención para evitar errores 'tontos'.

Ejercicio de Interpolación de Lagrange:

Se conoce la producción de bioetanol que se obtiene en una planta de biocombustibles en función del tiempo. Los valores de la concentración (g/l) están almacenados en el vector: B (10,20,35.33,35.5) y los instantes de tiempo (horas) en el vector s (1,3.5,5,10).

SE PIDE:

Realizar un script llamado Bioetanol para estimar, mediante interpolación de Lagrange, la concentración de bioetanol en los instantes t = 2 y t = 6.5. Para ello, se empleará la siguiente expresión del polinomio interpolador de Lagrange.

$$p = \sum_{i=1}^{n} B_i L_i$$

PASOS A SEGUIR

1. Programar la función Polbase para obtener las funciones de base la interpolación de Lagrange, que están dadas por la expresión:

$$L_i = \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{t - s_j}{s_i - s_j}, (i = 1, \ldots, n)$$

Bien, para definir Polbase, una función, debemos hacer lo siguiente:

Nos fijamos en la fórmula, observamos que variables tenemos A LA DERECHA DEL IGUAL: n, t y el vector s que depende de i y de j por lo que haremos 2 bucles. Además, bajo el productorio se puede observar que la j no puede ser igual a i; por lo tanto, esto indica que debemos aplicar un condicional. Este condicional estará dentro de los bucles anidados y será lo que defina la función. A LA IZQUIERDA DEL IGUAL, observamos que vamos a definir L_i, el cual es un vector y nuestra solución, por ello para que R sepa cual es el resultado de la función escribimos return(L).

```
Polbase = function(n,t,s){
    L = 0
    for (i in 1:n){
        L[i] = 1
        for (j in 1:n){
            if (i != j){
                L[i]=L[i]*(t-s[j])/(s[i]-s[j])
            }
        }
    }
return(L)
}
```

Como debes saber, un productorio se programa así: Resultado=Resultado*(Lo que hay en la fórmula). Inicializamos la variable resultado A 1 PORQUE ES UN PRODUCTORIO. Aplicamos los dos bucles, el condicional y ponemos la fórmula como explique antes.

Tras haber definido ya Polbase y solo después, definiremos el polinomio interpolador.

2. Programar la función PolInterp para obtener el polinomio interpolador en el punto x. Dicha función recibirá como argumentos de entrada:

   - un vector B (que contiene los valores de la función que se interpola),
   - el vector L (que contiene los polinomios de base de Lagrange evaluados en el punto x)
   - la variable n (número de puntos del soporte de interpolación.

$$p = \sum_{i=1}^{n} B_i L_i$$

Utilizaremos la fórmula de sumatorio indicada. Se observa que la función varía respecto a los valores del vector L, el cual está definido por Polbase, el vector B que es un dato y n que también es un dato. Por tanto, esas son las variables de la función.

Un sumatorio es inicializado igualando a 0 (*elemento neutro de la suma*). Tras ello abrimos un bucle for y aplicamos lo mismo que en el productorio, pero con una suma.

```
polinterp = function(B,L,n){
        p = 0
        for (i in 1:n){
                p = p + B[i]*L[i]
        }
return(p)
}
```

Una vez definidas las dos funciones, haz funcionar el programa, debería darte 0 errores y si no es así, pincha aquí. (ENLAZAR CON EL ARCHIVO DE ERRORES)

Tras la definición de ambas funciones, introducimos los datos. Solo debes copiar y pegar los datos y añadir un = y una c en el caso de que se trate de vectores (*los vectores en el el enunciado son B y s*), no los copies uno a uno o tendrás más fallos.

```
#INTRODUCCIÓN DE DATOS

B=c(10,20,35.33,35.5)

s=c(1,3.5,5,10)

n=length(s)

t=6.5

LB=0

LB=polbase(n,t,s)

I=polinterp(B,LB,n)

LB

I
```

Mientras que B, s y t son datos del enunciado, n será la longitud de uno de los vectores del enuncia por lo que definimos n de la siguiente manera: **n=length(s).** Como las funciones ya están definidas, debes ejecutarlas y almacenar el valor solución en una dirección de memoria.

Para ello, como sabes desde la práctica 1 (ENLAZAR), elige un nombre donde almacenar el valor de Polbase y otro para almacenar el polinomio interpolador. Yo elegí: LB e I. (ENLAZAR A ARCHIVO DE ERRORES)

4. Obtener 1001 abscisas equidistantes en el intervalo [s[0],s[n]] y almacenarlas en un vector x. (Calcularemos el valor del polinomio interpolador en cada abscisa para dibujar).

5. Llamamos 1001 veces a las funciones Polbase y PolInterp. La salida de Polinterp se guardará en un vector f[k], k=1,...,1001.

Para hacer el punto número 4: utilizaremos la secuencia 'seq'. Nos piden 1001 divisiones así que utilizaremos el comando 'length.out' para definir esta longitud.

```
x = seq(s[1],s[n],length.out = 1001)
```

Llamamos 1001 veces a las dos funciones, es decir, utilizamos un bucle que se repita 1001 veces. Además, los valores del polinomio interpolador se guardarán en f[k]. Esta K indica que lo que varía es la k en el bucle.

```
f=0
for (k in 1:1001){
 t=x[k]
 LB=0
 LB=polbase(n,t,s)
 f[k]=polinterp(B,LB,n)
}
```

Dentro del bucle tenemos que llamar a la función para valor de x. Como se ha explicado en clase de algoritmia, se calculan los polinomios de base para luego formar el polinomio interpolador.

Para el primer valor de x, que es uno de los valores de t, se calcula el polinomio de base y luego el polinomio de base en función de ese valor. Por eso se inicializa LB a 0, pues en cada vuelta del bucle LB toma un valor distinto. Queda todo almacenado en f[k] pues la función polinomio interpolador está igualada a esta sintaxis.

DIBUJAMOS

plot(s,B,xlim=c(s[1],s[n]),ylim=c(0,60))

par(new='TRUE')

plot(x,f,xlim=c(s[1],s[n]),ylim=c(0,60),xlab="",ylab="",col='blue',type='l')

Literalmente tienes que copiar esto del enunciado. En otras prácticas habrás aprendido lo que significa cada una de las cosas. En este caso el enunciado te da lo que tienes que escribir para obtener la representación que se pide.

Fórmula de Simpson

Se considera una sustancia cuya densidad viene dada por d(x) siendo x la coordenada espacial. La masa total en cierto intervalo [A,B] está dada por

$$M_{AB} = \int_A^B d(x)dx$$

Se desea realizar un programa en R para resolver dicha integral mediante una fórmula de Simpson compuesta, cuya expresión viene dada por:

$$\int_A^B d(x)dx \approx \frac{h}{6}\left(d(A) + 2\sum_{i=2}^{n-1} d(T_i) + 4\sum_{i=1}^{n-1} d\left(\frac{T_i + T_{i+1}}{2}\right) + d(B)\right)$$

Donde se ha realizado una subdivisión del intervalo [A,B] en (n-1) subintervalos iguales, es decir, considerando n puntos Ti, y siendo h la longitud de cada subintervalo.

El programa se ejecutará con los siguientes datos: A=0, B=3.05, densidad dada por d(x)=exp(x)sin(x). Como número de puntos para el desarrollo de la fórmula se tomará n=35.

Primero de todo, como repetirá numerosas veces Arturo durante la practica, efinimos las funciones. La densidad viene definida de la siguiente manera: exp(x)*sin(x).

```
d = function(x){
  exp(x)*sin(x)
}
```

La segunda función es un poco más compleja así que intentaré explicarla detenidamente para su entendimiento.

La denominaremos 'simpson'. Su expresión viene dada en el enunciada. Para conseguir que no de un error lo primero que haremos será denominar las variables de la función que dependen de datos, es decir no las podemos calcular. Dichas variables son A, B y n.

Aquello que sí podemos calcular son la h, t y ambos sumatorios. Pues bien procedemos a ello.

```
simpson = function(A,B,n){
  h=(B-A) /(n-1) #h se define como la longitud de cada uno de los segmentos en los que dividimos A y B. Por ello es el vector AB partido del número de trozos menos uno. #

  t = seq (A,B,length.out = n) #t es el vector que posteriormente en los sumatorios. Se trata de una división a partes iguales del vector AB en n puntos. Para ello se utiliza 'seq', secuencia la cual nos da n puntos comprendidos entre A y B.

  s1 = 0
  i=2
  while (i <= (n-1)){
    s1 = s1 + d(t[i])
    i = i+1
  }
  s2=0
  j=1
  while (j <= (n-1)){
    s2 = s2 + d ((t[j]+t[j+1])/2)
    j = j+1
  } #Ambos sumatorios s1 y s2, se inicializan igualando su nombre a 0. En este caso estamos utilizando un bucle while para hacer un sumatorio. En todo caso fue una orden del guión, se podría hacer perfectamente con un bucle for. (véase abajo).
```

El sumatorio llama constantemente a la función densidad por lo que esta debe de estar definida previamente. Como hacemos siempre dentro de la función densidad introducimos la fórmula y tendremos la función casi y terminada. #

La función entera:

```
simpson = function(A,B,n){
 h=(B-A)/(n-1)
 t = seq(A,B,length.out = n)
 s1 = 0
 i=2
 while (i <= (n-1)){
  s1 = s1 + d(t[i])
  i = i+1
 }
 s2=0
 j=1
 while (j <= (n-1)){
  s2 = s2 + d ((t[j]+t[j+1])/2)
  j = j+1
 }
 M=(h/6)*(densidad(A)+2*s1+4*s2+densidad(B))
 return(M)
} #Se aprecia que el único cambio es que damos valor a M copiando la fórmula. #
```

Ahora todo es pan comido. Debes copiar los datos del enunciado y llamar a la funciones como hemos hecho en la práctica anterior.

```
#DATOS
A=0 ; B= 3.05 ; n = 35
S=simpson(A,B,n)
S
```

El resultado debe ser aproximadamente: 11.97907159...

Tras ello se pide la representación gráfica de lo que acabamos de calcular. En este caso el enunciado nos vuelve a mostrar las instrucciones que debemos introducir en R para que salga la función.

```
#REPRESENTACIÓN

par(mfrow=c(2,1))

x=seq(A,B,0.01)

plot(x,densidad(x[1:length(x)]),xlim=c(A,B),ylim=c(0,8),xlab='x',ylab='rho')

par(new='TRUE')

#De nuevo la instrucción para que superponga las representaciones de ambas funciones dadas por plot. #

plot(t,densidad(t[1:length(t)]),xlim=c(A,B),ylim=c(0,8),type='h',col='green',xlab='',ylab
='')
```