®ecursos

1. Introducción al contenido:

El objetivo de esta recopilación de recursos consiste en facilitar el aprendizaje de la asignatura "Fundamentos de programación", del grado Biotecnología que se cursa en la Escuela Técnica Superior de Ingeniería Agronómica, Alimentaria y de Biosistemas de la Universidad Politécnica de Madrid. Estos recursos se centran en la parte de la materia relativa al lenguaje de programación R, que se usa en las clases prácticas de la asignatura. Los recursos se han elaborado a partir de los trabajos realizados en dichas clases prácticas y los documentos de apuntes, ejercicios resueltos y guiones de las prácticas 3 y 4 proporcionados por el profesor Arturo Hidalgo López. El porqué de estos recursos se explica en el poco conocimiento que suelen tener los alumnos que empiezan esta carrera sobre la programación; de esta forma, con resúmenes aclaratorios, se facilita la comprensión y se agiliza el aprendizaje. Se recomienda leer y trabajar estos recursos previamente a las prácticas, prestando atención a los comandos y a los errores. Tras la práctica, su uso será aclaratorio y facilitará la corrección de errores antes del examen de prácticas.

*Nota: Para facilitar la comprensión de las funciones y operadores de R, estas se escribiran en **negrita** y las aclaraciones que hagamos dentro de estas las haremos de color verde, de modo que para usar correctamente la función/operador solo habrá que copiar la parte escrita en **negrita** y usar como ejemplo o ayuda la parte escrita en verde. Ante la confusión que ocasionan las comillas en R, vamos a usar comillas dobles en todo el documento: ""*

2. Primer recurso → Conceptos básicos

En este apartado de conceptos básicos aprenderemos funciones de R y operaciones que hay que tener muy claras para la realización de las primeras prácticas de programación, por lo que es ideal para preparar la primera toma de contacto con R. Para realizar este recurso hemos obtenido la información de los apuntes colgados en el Moodle de la asignatura.

1. Funciones básicas:

- **rnorm(1)**: genera un dato al azar muestreado de una distribución normal de media 0 y varianza 1
- **runif (X)**: genera "X" valores aleatorios en el intervalo [0,1]
- **ls()**: genera una lista con las variables almacenadas
- **ls.str()**: genera una lista con las variables almacenadas y la información incluida en ellas
- **ls.str(pat= "O")**: función idéntica pero sólo mostrando los valores que contengan la "O"[1]

[1] Si antes de la "O" añadimos un "^" se mostrarán solo los valores que comiencen con la "O"

- **rm()**: se borran los datos de la memoria

2. Funciones numéricas y algebraicas útiles:

2.1. Operaciones básicas:

- "**+**" = Suma
- "**-**" = Resta
- "*****" = Multiplicación
- "****" = División

2.2. Operadores aritméticos:

- "**<**" = Menor
- "**>**" = Mayor
- "**<=**" = Menor o igual
- "**>=**" Mayor o igual
- "**!=**" Distinto
- "**==**" = Igualdad lógica

2.3. Funciones matemáticas:

2.3.1. Funciones logarítmicas

- "**log(**x**)**" = logaritmo neperiano
- "**log10(**x**)**" = logaritmo en base 10
- "**log2(**x**)**" = logaritmo en base 2
- "**exp(**x**)**" = función exponencial

2.3.2. Funciones trigonométricas

- "**sin(**x**)**" = seno
- "**cos(**x**)**" = coseno
- "**tan(**x**)**" = tangente trigonométrica
- "**asin(**x**)**" = arco seno
- "**acos(**x**)**" = arco coseno

® "**atan(**x**)**" = arco tangente

2.3.3. Otras funciones

® "**abs(**x**)**" = valor absoluto

® "**sqrt(**x**)**" = raíz cuadrada

® "**factorial(**x**)**" = factorial

® "**choose(**n,x**)**" = binomio de Newton n sobre x.

3. Segundo recurso → Práctica 1: vectores y listas

En este recurso se resume el contenido de la primera práctica del curso. Esta primera práctica la hicimos online el 15 de septiembre. El objetivo de la práctica es una introducción a R, y el profesor (Arturo Hidalgo López) empezó explicando algunos conceptos básicos que hemos incluido en el primer recurso. Para demostrar que los contenidos impartidos se habían comprendido, se realizó un ejercicio que utilizaba las estrategias de generar vectores y listar las variables utilizadas. El ejercicio consistía en lo siguiente:

1. Desarrollo de la práctica

1) Se crean los vectores "pais1", "pais2", "poblacion1", "poblacion2", los dos primeros alfanuméricos.

```
> pais1 = "Italia"
> pais2 = "Alemania"
> poblacion1 = 50
> poblacion2 = 80
```

2) A continuación, se crea una lista de todas las variables introducidas con el comando "**ls()**".

```
> ls()
[1] "pais1"      "pais2"      "poblacion1" "poblacion2"
```

3) Se listan aquellas que contengan la letra p y la letra b con el comando "**ls(pat= " ")**".

```
> ls(pat ="p")
[1] "pais1"      "pais2"      "poblacion1" "poblacion2"
> ls(pat="b")
[1] "poblacion1" "poblacion2"
```

4) Se listan las variables que EMPIEZAN por la "p" y la "b" con el comando "**ls(pat= "^ ")**".

```
> ls(pat="^b")
character(0)
> ls(pat="^p")
[1] "pais1"      "pais2"      "poblacion1" "poblacion2"
```

5) Mediante el comando **“ls.str()”** se crea una lista de todas las variables así como los valores o datos almacenados. El programa especifica si la variable es alfanumérica o no.

```
> ls.str()
pais1 :  chr "Italia"
pais2 :  chr "Alemania"
poblacion1 :  num 50
poblacion2 :  num 80
```

6) Por último, se eliminan las variables pais1 y población1 con el comando **“rm()”** y se listan las variables para comprobar que se han eliminado con **“ls()”**.

```
> rm(pais1,poblacion1)
> ls()
[1] "pais2"       "poblacion2"
```

2. Errores más habituales:

Debido a que el nivel de dificultad es muy bajo, no se observan demasiados posibles errores a la hora de desarrollar la práctica. Algunos de los más habituales son:

® No introducir las variables alfanuméricas entre comillas, impidiendo que el programa pueda detectar que está leyendo variables con letras.

```
> pais1 = Italia
Error: objeto 'Italia' no encontrado
```

® Introducir los comandos de forma errónea, añadiendo o quitando letras a estos:

```
> lss(pat="^b")
Error in lss(pat = "^b") : no se pudo encontrar la función "lss"
```

4. Tercer recurso ⟶ Práctica 2 : vectores y matrices

El siguiente recurso es un resumen de la segunda práctica, remarcando consejos y aclaraciones para hacer más sencilla su comprensión. Además, se analizan los errores más comunes detectados en las prácticas, así como posibles dudas. Dicha práctica fue impartida por el profesor Arturo Hidalgo durante los días 20, 21, 22 de septiembre, y su objetivo es una introducción a programar vectores, matrices y matrices creadas por vectores y definir distintas operaciones con vectores y matrices.

1. Comandos relevantes:

® **vector =c ()**: permite crear el vector.

® **#**: todo lo escrito después de la almohadilla no se lee y por tanto no se programa.

® **vector1 * vector2** : multiplica componente a componente, generando un vector sin sentido matemático.

® **vector1 %*% vector2** : producto escalar de los vectores.

- ® **matriz =matrix (c (), ncol=, nrow=)**: este comando crea una matriz con los componentes de c y con las columnas (ncol) y las filas (nrow) que se especifiquen.
- ® **matriz 2 = t(matriz)**: se realiza la traspuesta de matriz entre paréntesis.
- ® **det(matriz)**: genera su determinante.
- ® **solve (A, b)**: resuelve el sistema A*x =b.
- ® **solve(matriz)**: realiza la inversa de la matriz.
- ® **matriz = cbind (vector 1, vector 2, ..., vector n)**: genera una matriz donde las columnas son los vectores.
- ® **matriz = rbind (vector 1, vector 2, ..., vector n)**: genera una matriz donde las filas son los vectores.
- ® **eigen = matriz**: valores y vectores propios.

2. Desarrollo de la práctica:

1) Se definen los vectores “u” y “v”:

```
> u=c(-10,exp (1))
> v=c(3,25)
```

2) Se crean las matrices “A” y “B”, 2 x 2:

```
> A=matrix(c(5/3,10**2,exp(2),0.5),nrow=2,ncol=2)
> B=matrix(c(-6,-3/8,12,1),nrow=2, ncol=2)
```

3) Se operan los vectores u y v haciendo su suma, su resta y su producto escalar y se construye una matriz “C”, cuyas columnas sean los vectores “u” y “v” , y una matriz “R” donde sus filas sean los vectores “u” y “v”.

```
> C=cbind(u,v)
> C
             u  v
[1,] -10.000000  3
[2,]   2.718282 25
> R=rbind(u,v)
> R
   [,1]      [,2]
u  -10  2.718282
v    3 25.000000
```

4) Se resuelve el sistema A*x =b, introduciendo el vector b de dos componentes. Se almacena la solución en un vector.

```
> b=c(2,1)
> solve(A,b)
[1] 0.00865641 0.26871804
```

5) Por último, se almacena la traspuesta de “B” en una matriz y se realiza el producto con la matriz “A”.

```
> BT=t(B)
> BT
     [,1]   [,2]
[1,]   -6 -0.375
[2,]   12  1.000
> A%*%BT
           [,1]       [,2]
[1,]   78.66867   6.764056
[2,] -594.00000 -37.000000
```

El ejercicio anterior está sacado del script del guión de prácticas y por tanto, los resultados están contrastados con los obtenidos por el profesor Arturo Hidalgo.

3. Errores más frecuentes:

Durante la sesión de laboratorio surgieron algunas dificultades para realizar los ejercicios. Las más frecuentes fueron errores de notación y no cerrar los paréntesis al finalizar un comando, muy común a la hora de generar matrices o vectores por lo que el programa da error. Destacan los siguientes:

- ® En este caso ha seguido leyendo y no ha cerrado la matriz B, por tanto, no la reconoce.

```
Error: unexpected symbol in:
"B=matrix(c(-6,-3/8,12,1),nrow=2, ncol=2
u"
```

- ® Otro error muy común es la mala colocación de paréntesis en operaciones aritméticas y obtención de resultados erróneos. Para ello cabe remarcar el orden de las operaciones

    1. Potencias.
    2. Multiplicaciones y divisiones.
    3. Sumas y restas.
    4. Si hay operaciones del mismo rango, se realizan de izquierda a derecha.
    5. Los paréntesis presentan el primer orden de prioridad.

- ® Puede suceder también que se escriban mal los comandos, dando lugar a error, ya que R no puede reconocerlo.

- ® Ante cualquier error es recomendable ejecutar el programa y observar lo que indica. Todos los errores se detectan y leyendo las aclaraciones del propio programa se puede saber dónde está el fallo.

5. Cuarto recurso → Práctica 3 : Representación gráfica de funciones. Bucles

Este recurso es un resumen de la tercera práctica del curso. No estará solo resuelta, sino que también marcaremos algunas anotaciones y consejos con el fin de dejar claros algunos conceptos básicos. Además, se analizan los errores más comunes detectados en esta práctica en concreto, así como posibles dudas. Dicha práctica fue impartida por el profesor Arturo Hidalgo durante los días 4, 5 y 6 de octubre, y su objetivo es una introducción a la representación de funciones en R así como a programar bucles simples como anidados.

Antes de empezar con este recurso, recomendamos tener claros los conceptos anteriores sobre vectores, matrices y sus correspondientes operaciones, además de conceptos básicos sobre algoritmia. De esta forma, se hará más fácil la comprensión de estos ejercicios.

1. Comandos relevantes:

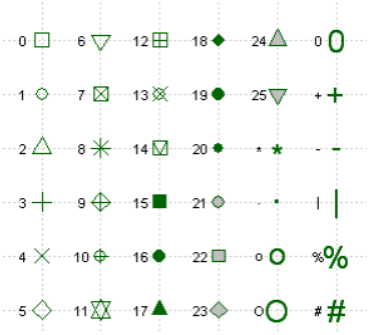
1.1. Representación gráfica de funciones

Para representar una función será esencial poner plot(a,b) aunque a esto podremos añadirle más "detalles" conforme queramos la gráfica. Ej. **plot(a,b,type= "b",col= "blue",xlab= "tiempo", ylab= "concentración",pch=2,main= "gráfico 1")**

- **plot(a,b)** : genera una gráfica siendo a y b vectores (deben estar definidos previamente y pueden tener cualquier nombre). La primera posición será para el eje x y la segunda para el eje y.

Es importante entender que se emplean vectores como ejes porque un vector en r es una forma de almacenar información, es decir, de almacenar los puntos que constituyen un eje.

- **type= "?"**: genera una línea que une los puntos. Dependiendo de la letra cambiará el tipo de línea.
  - **p** solo se verán los puntos
  - **l** solo se verán las líneas que unen los puntos
  - **b** se verán tanto líneas como puntos
  - **c** se verán puntos vacíos unidos por líneas
  - **s** se verá en forma de escalera
  - **h** se verá en forma de histograma (líneas verticales)
  - **n** no se verá nada, ni puntos ni líneas
- **col= ""** : cambiará el color de la gráfica al que nosotros escribamos. Tiene que ir en inglés ('blue' 'green' 'purple'...)
- **xlab= ""** : nombre del eje x
- **ylab= ""** : nombre del eje y



- **pch=** : cambia el tipo de símbolo para los puntos según la siguiente imagen.
- **main= “”** : Poner título al gráfico
- **par(new= “true”)** : superpone las gráficas. Hay que ponerlo cada vez que queramos superponer una nueva. Se escribe encima de plot(,)
- **legend(x= “top” , c(“función 1”, “función 2”, “función 3”) , fill=c(“green”, “blue”, “red”)** : Representa una leyenda en la gráfica.
    - **x= “”** representa el lugar donde se sitúa la leyenda (top, bottom, topleft…)
    - **c(“”)** : representa los nombres en la leyenda, se almacena como un vector
    - **fill= c(“”)** : representa un cuadrado con el color asignado al lado de cada nombre

  Como bien hemos mencionado, el único comando obligatorio para representar la gráfica es el primero **plot(a,b)** mientras que el resto son opcionales, ya que tienen como único fin modificar la gráfica.

1.2. Bucles

- **for (i in vinic:vfin) {**

  Proceso de cálculo

  }

Este es el comando general donde vinic y vfin significan valores de inicio y fin del bucle, la manera que tienen los bucles de funcionar es ir recorriendo valores desde un valor inicial hasta uno final y por esto es que se escribe de esta forma.

Es importante entender que este es el mecanismo detrás de los bucles para hacer el ejercicio que harás en clase de suma y multiplicación de vectores. Como estas dos operaciones se hacen componente a componente, el bucle irá realizando la operación, llamada “proceso de cálculo” para cada valor de la variable i, que en este caso es la **posición** del vector.

1.3. Bucles anidados

- **for (i in vinic:vfin) {**

  for (j in vinic2:vfin2) {

  for (k in vinic3:vfin3) {

  Proceso de cálculo

  }

  }

  }

Siendo i, k y j tres variables distintas, este tipo de estructuras sirven cuando se quieren emplear varias variables que entre sí se relacionan en uno o varios procesos de cálculo como verás en clase a la hora de hacer esta práctica. En el caso concreto de esta práctica realizarás o habrás ya realizado operaciones con matrices, estas consisten de filas y columnas, por lo que deben necesariamente asignarse dos variables para definir las dos dimensiones de la matriz y trabajar con ellas.

1.4. Funciones en R

- **Nombre de la función <- function(argumento1, argumento2,…){** expresión de la función **}**

Definir funciones en R es relativamente sencillo pues tiene las funciones básicas integradas, como puedes ver por la estructura del comando es una simple asignación, a la izquierda elegirás arbitrariamente el nombre que desees darle a la función, a la derecha, *function* es el comando, por lo que debe escribirse así siempre, los argumentos son los valores de x que deberá tomar la función, es decir el vector que se haya empleado para generar el eje de abscisas. La expresión de la función es lo que ya conoces que es una función, una expresión matemática dependiente de una variable x, por ejemplo, sen(x).

Una vez entiendes que compone una función (los valores en x y sus correspondientes imágenes) este apartado es muy sencillo, a la hora de representar la función sólamente hay que emplear los comandos del apartado 1.1

2. Desarrollo de la práctica:

2.1. Representación gráfica de funciones

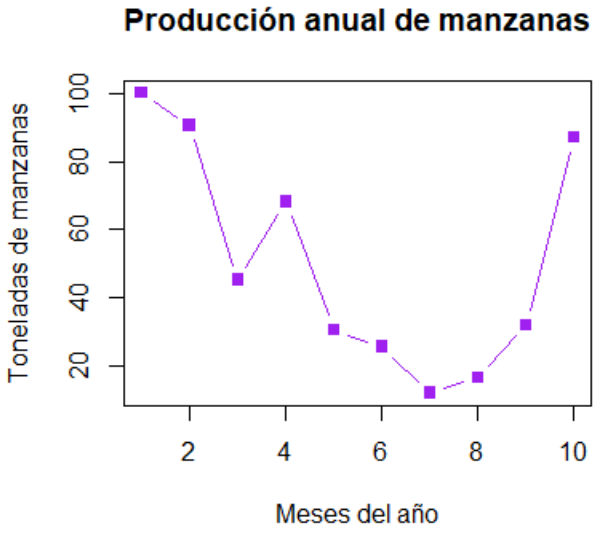
1	2	3	4	5	6	7	8	9	10
100	90.50	45.23	68.14	30.60	25.55	12.01	16.48	32.00	87.10

1) Construimos un vector *meses* que contenga los números de 1 a 10 asignados a los meses y uno llamado *manzanas* que contenga la producción de manzanas a partir de los datos.

```
> meses=seq(1,10,1)
> manzanas<-c(100,90.50,45.23,68.14,30.60,25.55,12.01,16.48,32.00,87.10)
```

2) Representamos gráficamente la producción mensual de manzanas. Usando línea continua y color morado. Etiquetamos los ejes como ‘Meses del año’ (abscisas) ‘Toneladas de manzanas’ (ordenadas). Ponemos como título ‘Producción anual de manzanas’ y utilizaremos el símbolo del cuadrado relleno

```
plot(meses,manzanas,type='b',col='purple',xlab='Meses del año',ylab='Toneladas de manzanas',pch=15,main='Producción anual de manzanas')
#eje x, eje y, tipo de linea, color del gráfico, nombre de los dos ejes, símbolo de los puntos, título de la gráfica
```



2.1. Ejercicios con bucles

v=(12,-3,5,18.7) y w=(12,0.25,77,exp(2))

1) Obtener la suma de los dos vectores mediante bucles y comprobarlo

```
> v<-c(12,-3,5,18.7)
> w<-c(12,0.25,77,exp(2))
> u=c(0) #tenemos que definirlo antes, también podemos poner u=0
> for (i in 1:length (v)){
+   u[i]=v[i]+w[i]
+ }
> u
[1] 24.00000 -2.75000 82.00000 26.08906
> v+w
[1] 24.00000 -2.75000 82.00000 26.08906
```

2)Obtener la suma de las componentes del vector v y almacenarlos en SumaC y comprobarlo

```
> SumaC=0
> for(i in 1:length(v)){
+  SumaC=SumaC+v[i]
+ }
> SumaC
[1] 32.7
```

```
> sum(v)
[1] 32.7
```

3)Realizar el producto escalar de ambos vectores mediante bucles y comprobarlo

```
> Pesc=0
> for(pepin in 1:length(v)){
+   Pesc=Pesc+v[pepin]*w[pepin]
+ }
> Pesc
[1] 666.4253
> v%*%w
         [,1]
[1,] 666.4253
```

4) Multiplicar ambos vectores componente a componente

```
> Mcomp=0
> for(i in 1:length(v)){
+   Mcomp[i]=v[i]*w[i]
+ }
> Mcomp
[1] 144.0000  -0.7500 385.0000 138.1753
> v*w
[1] 144.0000  -0.7500 385.0000 138.1753
```

5)Hacer una tabla con los resultados

```
> Titulo=c('Suma','Producto escalar') #filas
> Valor=c(SumaC, Pesc) #columnas
> data.frame (Titulo,Valor) #escribe la tabla con las anteriores filas y columnas
           Titulo    Valor
1            Suma  32.7000
2 Producto escalar 666.4253
```

2.3. Ejercicios con bucles anidados

1) Construir una matriz A1 de manera que los vectores v, w, sean sus filas, y una matriz A2 de manera que los mismos vectores sean sus columnas.

```
> A1=rbind(v,w)
> A1
  [,1]  [,2] [,3]      [,4]
v   12 -3.00    5 18.700000
w   12  0.25   77  7.389056
> A2=cbind(v,w)
> A2
         v         w
[1,] 12.0 12.000000
[2,] -3.0  0.250000
[3,]  5.0 77.000000
[4,] 18.7  7.389056
```

2) Multiplicar, empleando bucles, ambas matrices, obteniendo una matriz C. Antes de operar tenemos que igualarla a 0. Al final comprobaremos el resultado.

```
> C=matrix(c(0),nrow=nrow(A1),ncol=ncol(A2))
```

Si solo ponemos C=0 serían todo filas de 0, por eso igualamos a las columnas y filas de A1 y A2, según las de la matriz resultante.

```
> for(i in 1:nrow(C)){ #es lo mismo que nrow(A1)
+   for(j in 1:ncol(C)){ #es lo mismo que ncol(A2)
+     for(k in 1:ncol(A1)){
+       C[i,j]=C[i,j]+A1[i,k]*A2[k,j]
+     }
+   }
+ }
> C
         [,1]      [,2]
[1,] 527.6900  666.4253
[2,] 666.4253 6127.6607
> A1%*%A2
         v         w
v 527.6900  666.4253
w 666.4253 6127.6607
```

Repetido con explicaciones y pequeños cambios también válidos

```
> A1=rbind(v,w) # Combinando v, w por filas
> A2=cbind (v,w) # Combinando v,w por columnas
> #Hemos construido A1, A2
> #La matriz resultante C tendra tantas filas como A1 y tantas columnnas como A2
> m=nrow(A1) #Por comodidad m es num filas A1
> n=ncol(A2) #n es num col A2
> p=ncol(A1)
> #Inicializamos C con ceros
> C=matrix(c(0), nrow=m, ncol=n)
> #Hacemos la multiplicacion
> for(i in 1:m) { #bucle para filas de C
+   for(j in 1:n) { #bucle para coli,nas de C
+     for(k in 1:p){ #bucle columnas A1
+       C[i,j]=C[i,j]+A1[i,k]*A2[k,j]
+     }
+   }
+ }
> C
         [,1]      [,2]
[1,] 527.6900  666.4253
[2,] 666.4253 6127.6607
> #Comprobacion
> A1%*%A2
         v         w
v 527.6900  666.4253
w 666.4253 6127.6607
```

2.4. Ejercicios de funciones y representación gráfica

$$f(x) = x^2\cos(x^2) \quad g(x) = \sin(x^2)e^{-\frac{x^2}{10}} \quad h(x) = \sin(x^8)e^{-x}$$

1) Definir la función f(x) y obtener el valor f(pi/4)

```
> f<-function (x) {
+   x^2*cos(x^2)
+ }

> f(pi/4)
[1] 0.5031676
```

2) Obtener un vector xx que tome 1001 valores en [0,10]

```
> xx=seq(0,10,0.01) #generamos 1001 valores en [0,10]
> #alternativa: xx=seq(0,10,length=1001)
```

3) Representar gráficamente la función f tomando como abscisas los valores xx. Etiquetar el eje de abscisas con 'Abscisa' y el eje de ordenadas con 'mis funciones f,g,h'. La gráfica va en color verde.

```
> plot(xx,f(xx), xlab='abscisa', ylab='mis funciones f,g,h', col='green')
```

4) Define la función g(x) y h(x)

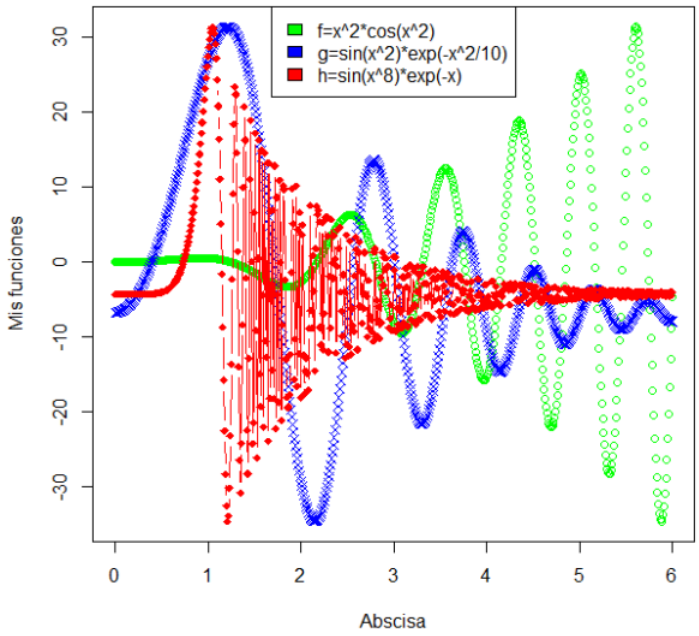
```
> g=function(pepe) {
+   sin(pepe^2)*exp(-pepe^2/10)
+ }
>
> h=function(x) {
+   sin(x^8)*exp(-x)
+ }
```

5) Dibujar la función g en los puntos xx en color azul eliminando los ejes y el número 4 para los símbolos. Idem con la función h en rojo y con 18 para los símbolos. Superponer las 3 gráficas.

```
> par(new='true') #superpone las gráficas
> plot(xx,g(xx), col='blue', xlab='', ylab='', axes=FALSE, pch=4)
>
> par(new='true') #hay que ponerlo cada vez que representemos una
> plot(xx,h(xx), xlab='', ylab='', axes=FALSE, col='red', pch=18)
```

6) Añadir una leyenda

```
> legend(x='top', c('función f', 'función g', 'función h'), fill=c('green', 'blue','red'))
```



6. Quinto recurso → Práctica 4 : Definición de funciones y representación gráfica aplicando condicionales "if", "while"

Este recurso es el resúmen de la cuarta práctica del curso, donde aprenderemos a representar las funciones de forma más definida y aplicando estructuras condicionales. Dicha práctica fue impartida por el profesor Arturo Hidalgo durante los días 18, 19 y 20 de octubre. Es muy recomendable que tengáis claro los conceptos básicos sobre representación de gráficas (práctica 3) antes de empezar con este recurso.

Tendréis todos los comandos que necesitaréis saber para representar una función o más de una en una misma gráfica, para cambiar el aspectos de la función o la escala numérica en caso de tener más de una función y para aplicar bucles "while" y "if". También dejaremos el ejemplo de la práctica resuelto.

1. Comandos relevantes:

® Para generar un nº "x" de valores entre [a,b] usamos el comando: **=seq (a, b, paso[2])** o **=seq (a, b, length=x)**

Ej: Siendo x= 20, a=0 y b=10 y T el nombre de la lista:

[2] Se define como la variación entre cada uno de los elementos del conjunto [a,b], que es constante y se calcula como: paso=(b-a)/(x-1), siendo x el número de elementos total que queremos obtener

```
> T=seq(0,10,length=20)

> T

 [1]  0.0000000  0.5263158  1.0526316  1.5789474  2.1052632
2.6315789

 [7]  3.1578947  3.6842105  4.2105263  4.7368421  5.2631579
5.7894737

[13]  6.3157895  6.8421053  7.3684211  7.8947368  8.4210526
8.9473684

[19]  9.4736842 10.0000000
```

2. Cómo definir funciones:

- Para nombrar la función que vamos a crear escribimos su nombre, comando de igualdad (=/ <-) y "function (-variable que queramos -)". A continuación se escribe entre claudators la ecuación a la que es igual.

Ej: Siendo "f" el nombre de la función y "x" nuestra variable[3]:

```
f<-function(x){
 x*cos(x^2)
}
```

- Si a continuación escribimos "f", en la consola, nos devuelve el texto de la imagen anterior, ya que la función solamente actúa cuando la llamamos por su nombre y le mandamos los datos de entrada, por ejemplo:

```
> f(pi/4)
[1] 0.6406528
```

- Para dibujar la gráfica de una función usamos el comando **plot(valor de x, valor de y, xlab= "Texto en el eje de las x", ylab= "Texto en el eje de las y", col= "color") par(new= "true")**[4]

Ej: Siendo "xx" el nombre de la lista que va a dar los valores a x, "f(xx)" la función[5] que va a dar los valores a y, "Abcisas" el texto en xlab=, "Mis funciones f, g, h" el texto en ylab= y "green" el texto en col=:

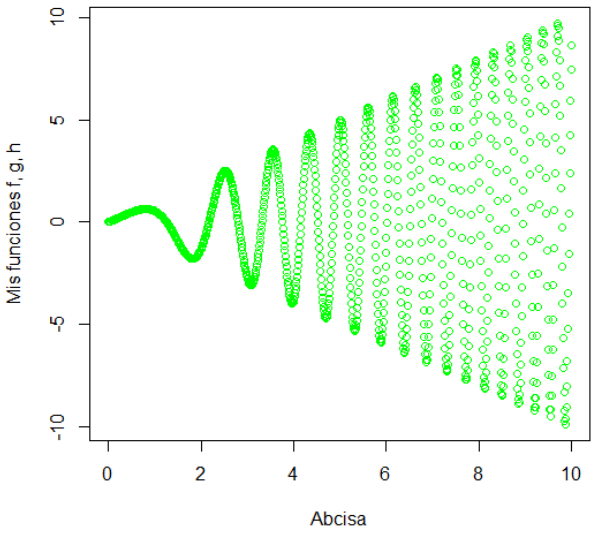
[3] En vez de llamar "x" a la variable se puede poner cualquier nombre como "pepín" o "peponcio"

[4] Este operador sirve para que puedan haber varias funciones superpuestas en la misma gráfica

[5] Tanto la lista como la función deben estar previamente guardadas en el programa

```
plot(xx,f(xx),xlab='Abcisa', ylab='Mis funciones f, g, h',
col='green')

par(new=TRUE)
```

Obtenemos el siguiente gráfico:



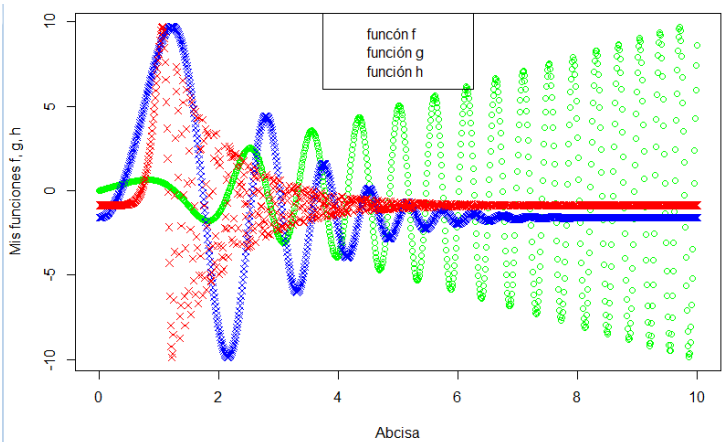
® Para combinar varias funciones en un mismo gráfico volvemos a escribir todo el comando **plot(** anterior pero escribiendo solo unas comillas en el texto de los ejes: **xlab= "", ylab= ""** , y los comandos **axes=FALSE** para que no se solapen distintos textos de los ejes o los propios ejes:

Ej de varias funciones, añadiendo algunos operadores más como **legend(x= "ubicación"[6], c(valor 1, valor 2; ...)** :

```
plot(xx,f(xx),xlab='Abcisa', ylab='Mis funciones f, g, h',
col='green')

par(new=TRUE)

plot(xx,g(xx),xlab='', ylab='', col='blue', axes=FALSE, pch=4)

par(new=TRUE)

plot(xx,h(xx),xlab='', ylab='', col='red', axes=FALSE, pch=4)

legend(x='top',c('funcón f', 'función g', 'función h'))
```

Gráfico:

[6] Las posibles ubicaciones son: "bottomright", "bottom", "bottomleft", "left", "topright", "top", "topleft", "right", "left"


funcón f
función g
función h
Mis funciones f, g, h
Abcisa
10
5
0
-5
-10
0
2
4
6
8
10

3. Aplicación de bucles “while” y “if”

® Una aplicación del bucle “while” es hacer una operación hasta que ocurra la condición, como puede ser que un número “X” sea igual al número de sumandos de la operación, por lo que se acaba el bucle al acabarse los sumandos.

Ej: Siendo “v” el vector (1,3.4,exp(3),log(6)), “length(v)” número de sumandos del vector “v”, “i” el número que va a ir marcando los sumandos que ya se han operado, “sumaV” el sumatorio de las componentes de “v”:

```
v=c(1,3.4,exp(3),log(6))
i=1; sumaV<-0
while (i<=length(v)){
 sumaV = sumaV+v[i]
 i=i+1
}
sumaV
```

“i” va aumentando en una unidad cada vez que se hace la operación de el valor de “sumaV”, por lo que cuando se acaben los sumandos “i” valdrá el número de sumandos, “length(v)”, y se acabará el bucle.

4. Ejercicio del medicamento

N(t) = 10exp(t/20) es la evolución de células infectadas dependiendo del tiempo. Dependiendo de N se pueden dar diferentes situaciones:

- Si N < 5000, enfermo recuperable sin medicación. EN VERDE
- Si N >= 5000 y N <= 10000, medicación moderada. EN NARANJA
- Si N > 20000, medicación agresiva. EN ROJO
- En otro caso se trata de una situación intermedia. EN AZUL

Se hace un estudio de 168h, calculando la evolución de células cada media hora (dt = 0.5).

En el vector “xd” se guardan los valores del tiempo, en el vector “yd” el número de células infectadas y en el vector “color” los colores de cada situación.

REPRESENTAR ESTA GRÁFICA CON R:

1. Introducimos los vectores vacíos y datos que sabemos.

```
N<-function(t){
 10*exp(t/20)
}
i=1; xd=c(0); yd=0; color=0; tfin=168; dt=0.5; t=0
```

2. Con un bucle “while” analizamos las situaciones posibles en los 168h.
    - lo que está al final quiere decir que “i” aumenta 1 cada vez que “t” aumenta 0.5 hasta llegar a 168.
    - lo del medio son las condicionales para asignar el color.

```
while(t<=tfin){
 xd[i]=t; yd[i]= N(t)
 if (N(t)<5000){
  color[i]='green'
 }else if (N(t)>=5000 & N(t)<=10000){
  color[i]='orange'
 }else if (N(t)>=20000){
  color[i]='red'
 }else{
  color[i]='blue'
 }
 t=t+dt; i=i+1
}
```

IMPORTANTE: Seguir la misma estructura que aparece de cerrar una condicional “}” en otra línea y seguir ahí con otra condicional. No es solo más ordenado visualmente sino que es esencial para que funcione.

3. Representamos la gráfica con plot. tipo “h” para el histograma. El vector N cogerá su color correspondiente ya que “yd” y “color” cambian a la vez a medida que avanza el tiempo “xd”. (R no es capaz de representar tilde

```
plot(xd,yd,type='h', col=color, xlab='Tiempo (h)', ylab='Num c�lulas')
```