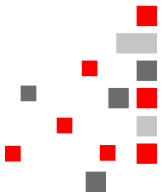
universidad de león

Ajuste de datos con R

Métodos Numéricos y Estadísticos
Grado en Ingeniería Informática / Mecánica

Curso 2020-2021

Eva María Mazcuñán Navarro



Contenidos

Página

Introducción

En esta práctica estudiaremos cómo ajustar un modelo a un conjunto de datos experimentales, mediante la técnica de mínimos cuadrados, usando R.

Mientras que en los problemas que hemos a mano hasta ahora hemos considerado únicamente modelos lineales de uno o dos parámetros, en esta práctica consideraremos modelos tanto lineales como no lineales, y con un número arbitrario de parámetros.

Seguiremos trabajando con una variable dependiente y y una única variable dependiente x, aunque las técnicas que presentaremos se generalizan sin dificultad al caso de varias variables independientes.

1. Requisitos previos

Antes de comenzar esta práctica, necesitas:

- Tener R y RStudio instalados en tu equipo. Ver Instalación de R y RStudio
- Haber estudiado la práctica Primeros pasos con R y RStudio

Para utilizar las funciones que aparecerán a lo largo de la prácticam empezamos cargando el paquete tidyverse:

```
library("tidyverse")
```

2. Modelos lineales

Como hemos visto en teoría, dada una variable dependiente y y una variable independiente x, un modelo lineal de parámetros β_1, β_2, ... β_p tiene la forma

$$y = \beta_1 f_1(x) + \beta_2 f_2(x) + \cdots + \beta_p f_p(x),$$

siendo f_1, f_2, ..., f_p funciones conocidas.

En este capítulo veremos cómo ajustar este tipo de modelos a las observaciones recogidas en una hoja de datos, usando la función `lm()` (**l**inear **m**odel) de R.

Veremos también cómo representar gráficamente un ajuste usando la función `geom_smooth()` del paquete `ggplot2`.

2.1. Planteamiento del problema: El prisma de vidrio

Newton demostró con el prisma que la luz blanca es una mezcla de varios colores y que la refracción depende del color (longitud de onda).

En un experimento, se eligieron diferentes longitudes de onda λ y se trazó el camino seguido por el rayo de luz que atraviesa el prisma, midiendo el ángulo de desviación para a partir del mismo calcular el índice de refracción n del vidrio para el color seleccionado. Los datos obtenidos se recogen en el archivo `cauchy.csv`(click para descargar), que contiene las variables:

- `lambda`: longitud de onda λ, medida en nm.
- `n`: índice de refracción.

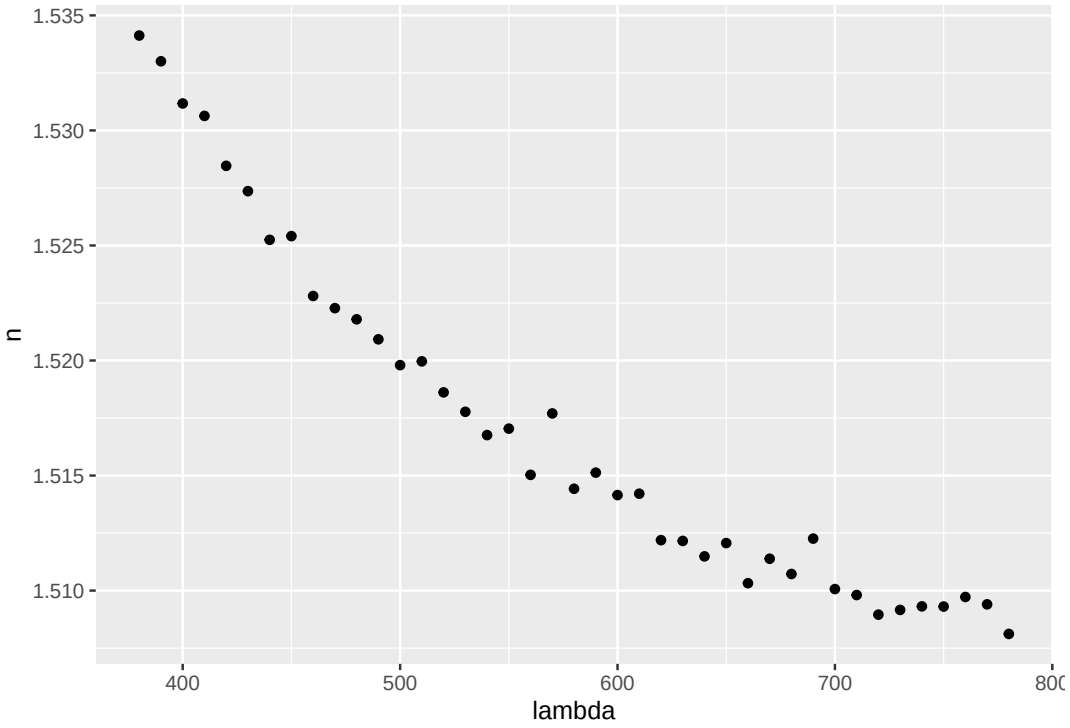
Descarga el fichero `cauchy.csv` y guárdalo en una carpeta de nombre `data` dentro de tu directorio de trabajo.

Importamos los datos con `read_csv()` y los guardamos en un objeto de nombre `cauchy`:

```
cauchy <- read_csv("data/cauchy.csv")
```

Visualizamos los datos dibujando la nube de puntos (λ, n) con `geom_point()`:

```
ggplot(
    data = cauchy,
    mapping = aes(x = lambda, y = n)
) +
    geom_point()
```



2.2. Ajuste con `lm()`

Tomaremos como modelo la fórmula de Cauchy para los índices de refracción n en la región visible del espectro de longitud de onda λ:

$$n(\lambda) = \beta_1 + \frac{\beta_2}{\lambda^2} + \frac{\beta_3}{\lambda^4}$$

donde β_1, β_2 y β_3 son los parámetros a ajustar.

Como se indicó antes, la función de R para ajustar modelos lineales es `lm()`.

```
fit_cauchy <- lm(
        data = cauchy,
        formula = n ~ I(1/lambda^2) + I(1/lambda^4)
    )
```

En el código anterior, se utiliza la función `lm()` para ajustar el modelo propuesto a las observaciones de nuestra hoja de datos. Hemos usado los argumentos `data`, para especificar la hoja de datos con las observaciones, y `formula`, para indicar la fórmula del modelo (enseguida explicaremos cómo construir esta fórmula).

El resultado se almacena en un objeto de nombre `fit_cauchy`.

Si imprimimos el objeto `fit_cauchy` veremos los coeficientes del ajuste:

```
fit_cauchy
```

```
##
## Call:
## lm(formula = n ~ I(1/lambda^2) + I(1/lambda^4), data = cauchy)
##
## Coefficients:
##   (Intercept)  I(1/lambda^2)  I(1/lambda^4)
##           1.5         4908.3      7078041.7
```

Pero la instrucción `summary(fit_cauchy)` revela que el objeto contiene mucha más información de la que muestra su simple impresión:

```
summary(fit_cauchy)
```

```
##
## Call:
## lm(formula = n ~ I(1/lambda^2) + I(1/lambda^4), data = cauchy)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -1.294e-03 -4.672e-04  1.711e-05  2.982e-04  2.216e-03
##
## Coefficients:
##                Estimate Std. Error  t value Pr(>|t|)
## (Intercept)   1.500e+00  7.676e-04 1954.496  < 2e-16 ***
## I(1/lambda^2) 4.908e+03  4.342e+02   11.303 1.02e-13 ***
## I(1/lambda^4) 7.078e+06  5.376e+07    0.132    0.896
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0007213 on 38 degrees of freedom
## Multiple R-squared:  0.9912, Adjusted R-squared:  0.9907
## F-statistic:  2131 on 2 and 38 DF,  p-value: < 2.2e-16
```

Ésta es la razón por la que hemos creado el objeto `fit_cauchy` para almacenar el resultado de la función `lm()`: lo usaremos en los siguientes apartados para extraer información del ajuste realizado.

2.3. Fórmulas en R

Al usar la función `lm()`, hemos indicado el siguiente valor para el argumento `formula` :

```
n ~ I(1/lambda^2) + I(1/lambda^4)
```

La expresión anterior es un objeto de R de tipo `formula`, que se corresponde con la fórmula del módelo

$$n = \beta_1 + \frac{\beta_2}{\lambda^2} + \frac{\beta_3}{\lambda^4}.$$

En la siguiente tabla se muestran algunos ejemplos más de fórmulas en R correspondientes a diferentes modelos:

Modelo	`formula`
$y = \beta_1 + \beta_2 x$	`y ~ x`
$y = \beta_1 x$	`y ~ 0 + x`
$y = \beta_1 + \beta_2 x + \beta_3 x^2$	`y ~ x + I(x^2)`
$y = \beta_1 \cos(x) + \beta_2 \sin(x)$	`y ~ 0 + I(cos(x)) + I(sin(x))`

Para especificar la fórmula de un modelo en R hay que tener en cuenta las siguientes reglas:

- Cada sumando en una fórmula de R, indica la función que multiplica a un parámetro em la fórmula del modelo. Así, el sumando `x` en una fórmula de R será un sumando de la forma $\beta_i x$ en la fórmula matemática del modelo.

- R añade siempre una constante como primer sumando de la fórmula del modelo (β_1 en los ejemplos anteriores). Para evitar la inclusión automática de esa constante, hay que escribir el sumando `0`. Así, la fórmula `y ~ x` se corresponde con $y = \beta_1 + \beta_2 x$; y si se quiere omitir la constante `\beta_1`, se ha de escribir `y ~ 0 + x`.

- Las funciones que son una transformación de la variable independiente se han de escribir incluidas en la función `I()`. Por ejemplo, un término de la forma $\beta_i x^2$ en la fórmula de un modelo se escribe en R como `I(x^2)`.

2.4. Coeficientes

Para obtener los coeficientes del ajuste usamos las función `coefficients()`:

```
coefficients(fit_cauchy)
```

```
##   (Intercept) I(1/lambda^2) I(1/lambda^4)
##  1.500310e+00  4.908268e+03  7.078042e+06
```

La salida nos informa de que los coeficientes que solucionan el problema de mínimos cuadrados (ecuaciones normales de Gauss) son:

$$\hat{\beta}_1 = 1.50031,$$

$$\hat{\beta}_2 = 4908.268,$$

y

$$\hat{\beta}_3 = 7078042.$$

Nota: Dependiendo de la configuración, los valores de la salida pueden aparecer en notación científica, simbolizando `e+00`, `e+03` y `e+06` que hay que multiplicar por $10^0 = 1$, $10^3 = 1000$ y $10^6 = 1000000$ respectivamente.

Por tanto el ajuste buscado es

$$n = 1.50031 + \frac{4908.268}{\lambda^2} + \frac{7078042}{\lambda^4}.$$

2.5. Valores ajustados

Los valores ajustados (o esperados) para la variable dependiente se obtienen con la función `fitted()`:

```
fitted(fit_cauchy)
```

```
##        1        2        3        4        5        6        7        8
## 1.534640 1.532886 1.531263 1.529759 1.528362 1.527062 1.525851 1.524721
##        9       10       11       12       13       14       15       16
## 1.523664 1.522674 1.521746 1.520875 1.520056 1.519285 1.518558 1.517873
##       17       18       19       20       21       22       23       24
## 1.517225 1.516613 1.516033 1.515484 1.514963 1.514468 1.513998 1.513552
##       25       26       27       28       29       30       31       32
## 1.513126 1.512721 1.512335 1.511967 1.511615 1.511279 1.510958 1.510650
##       33       34       35       36       37       38       39       40
## 1.510356 1.510074 1.509804 1.509545 1.509297 1.509058 1.508829 1.508608
##       41
## 1.508396
```

2.6. Residuos

Para obtener los residuos del ajuste usamos la función `residuals()`:

```
residuals(fit_cauchy)
```

```
##             1             2             3             4             5
## -5.158258e-04  1.192086e-04 -9.157603e-05  8.724810e-04  9.998852e-05
##             6             7             8             9            10
##  2.981569e-04 -6.063847e-04  6.853349e-04 -8.594240e-04 -3.992586e-04
##            11            12            13            14            15
##  4.507231e-05  4.592263e-05 -2.575917e-04  6.782515e-04  5.616847e-05
##            16            17            18            19            20
## -1.021129e-04 -4.671765e-04  4.270844e-04 -1.002945e-03  2.215907e-03
##            21            22            23            24            25
## -5.376294e-04  6.594862e-04  1.524706e-04  6.591191e-04 -9.333365e-04
##            26            27            28            29            30
## -5.619564e-04 -8.481528e-04  9.871010e-05 -1.294146e-03  1.065956e-04
##            31            32            33            34            35
## -2.355407e-04  1.609116e-03 -2.899098e-04 -2.667854e-04 -8.491117e-04
##            36            37            38            39            40
## -3.842938e-04  1.710620e-05  2.483173e-04  8.932864e-04  7.950677e-04
##            41
## -2.796932e-04
```

Para calcular el error cuadrático del ajuste (RSS) usamos

```
sum(residuals(fit_cauchy)^2)
```

```
## [1] 1.976785e-05
```

En el resumen del ajuste podemos leer

```
Residual standard error: 0.0007213 on 38 degrees of freedom
```

El error estandard residual, que suele denotarse σ, se obtiene con la función `sigma()`. La relación entre esta cantidad σ y el error cuadrático RSS es:

$$\sigma = \sqrt{RSS/38}$$

o equivalentemente

$$RSS = 38\sigma^2.$$

El valor 38, se llama grados de libertad de los residuos y se obtiene de restar, a las $n = 41$ observaciones, los $p = 3$ parámetros del modelo. Se obtiene con la función `df.residual()`.

```
sigma(fit_cauchy)
df.residual(fit_cauchy)*sigma(fit_cauchy)^2 # RSS
```

```
## [1] 0.0007212534
## [1] 1.976785e-05
```

2.7. Predicciones

La siguiente tabla recoge las longitudes de onda correspondientes a algunos colores del arcoiris:

Color	λ
Rojo	640
Amarillo	589
Verde	509
Azul	486
Violeta	434

Queremos calcular los índices de refracción que predice nuestro ajuste para los colores anteriores. Para hacerlo usamos la función `predict()`:

```
# hoja de datos con los nuevos valores de lambda
new_lambda <- tibble(
        lambda = c(640, 589, 509, 486, 434)
    )
# predicciones
predict(
    fit_cauchy,
    new_lambda
)
```

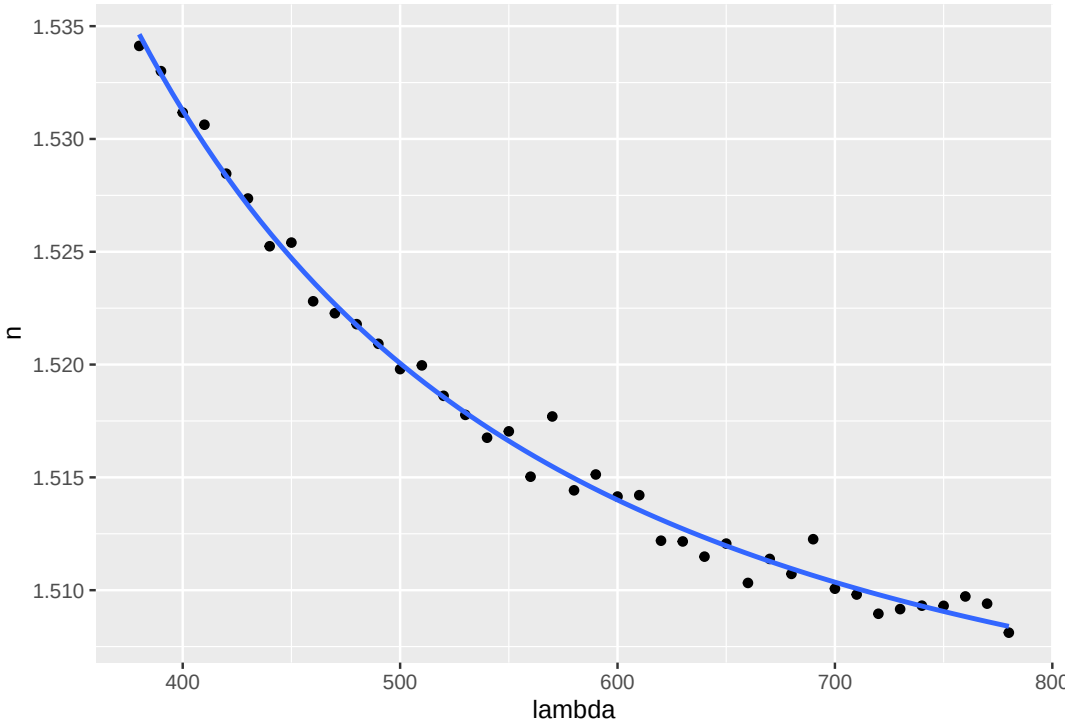
```
##        1        2        3        4        5
## 1.512335 1.514517 1.519360 1.521217 1.526568
```

En la función `predict()` indicamos como primer argumento nuestro ajuste `fit_cauchy` y como segundo argumento una hoja de datos, que hemos llamado `new_lambda`, con los nuevos valores para la variable independiente `lambda`. Hemos construido dicha hoja de datos con la función `tibble()` (*tidy table*).

2.8. Gráfico del ajuste

Al comienzo del problema representamos la nube de puntos de nuestras observaciones con la función `geom_point()`. Añadimos ahora el gráfico del ajuste con la función `geom_smooth()`:

```
ggplot(
    data = cauchy,
    mapping = aes(x = lambda, y = n)
) +
    geom_point() +
    geom_smooth(
        method = "lm",
        formula = y ~ I(1/x^2) + I(1/x^4),
        se = FALSE
    )
```



En los argumentos de `geom_smooth()` hemos escrito `method = "lm"` para indicar que el ajuste se realiza con la función `lm()`.

Notar que al especificar la fórmula del modelo en el argumento `formula` no se utilizan los nombres `lambda` y `n` de las variables (como se hizo en la función `lm()`) sino los nombres `x` e `y` de las estéticas asociadas.

El argumento `se = FALSE` inhibe representar los intervalos de confianza (que estudiaremos en el segundo bloque del curso).

3. Modelos no lineales

Modelos como

$$y = \beta_1 e^{\beta_2 x}$$

(modelo exponencial) o como

$$y = \beta_1 x^{\beta_2}$$

(modelo potencial) no tienen una estructura lineal.

Para modelos lineales, el problema de encontrar los valores de los parámetros que minimizan el error cuadrático del ajuste se traduce en resolver el sistema de ecuaciones normales de Gauss, que es un sistema lineal.
Pero en el caso de modelos no lineales como los anteriores, la situación se complica porque ahora la solución viene dada por un sistema de ecuaciones no lineal.

En este capítulo veremos cómo ajustar este tipo de modelos a una colección de observaciones, usando la función `nls()` (**n**onlinea **l**east **s**quares) de R.

3.1. Planteamiento del problema: Ley de enfriamiento de Newton

La ley de enfriamiento de Newton establece que cuando un sólido a temperatura inicial T_0 se deja enfriar en un ambiente de temperatura T_a, su temperatura en función del tiempo t viene dada por

$$T(t) = T_a + (T_0 - T_a)e^{-kt}$$

donde k es una constante que depende de la forma del sólido y del calor específico del material que lo componga.

En un experimento se calentó una barra de hierro hasta una temperatura inicial de T_0 y después se dejó enfriar en un ambiente a temperatura T_a. Se midió la temperatura de la barra a intervalos de tiempo de 1 minuto, durante una hora. Los datos obtenidos se recogen en el archivo `newton.csv`(click para descargar), que contiene las variables:

- `time`: tiempo transcurrido, en minutos, desde el instante inicial ($t = 0$).

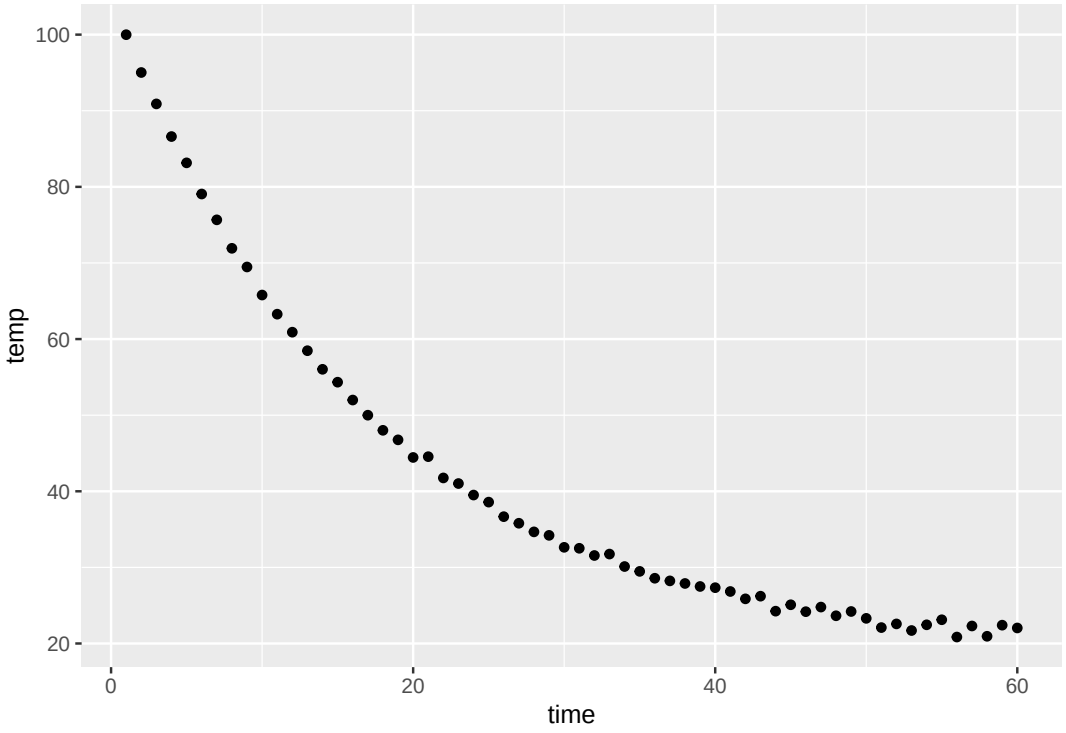
- `temp`: temperatura de la barra de hierro, en ºC.

Como antes, descarga el fichero `newton.csv` y guárdalo en una carpeta de nombre `data` dentro de tu directorio de trabajo. Importamos los datos con `read_csv()` y los guardamos en un objeto de nombre `newton`:

```
newton <- read_csv("data/newton.csv")
```

Visualizamos los datos dibujando la nube de puntos (t, T) con `geom_point()`:

```
ggplot(
    data = newton,
    mapping = aes(x = time, y = temp)
) +
    geom_point()
```



3.2. Ajuste con `nls()`

Antes de utilizar la función `nls()` necesitamos crear una función para la fórmula de la temperatura en nuestro modelo, que tiene como argumentos la variable independiente t, y los tres parámetros T_0, T_a y k:

```
T_model <- function(t, Ta, T0, k){
    Ta + (T0-Ta)*exp(-k*t)
}
```

También necesitamos proporcionar unos valores iniciales para los parámetros. Estos valores iniciales ayudarán en el proceso de optimización del error cuadrático (que pasa por resolver un sistema no lineal). A la vista de los datos tomaremos 20 como valor inicial para T_a y 100 como valor inicial para T_0. Para el valor inicial de k, tomamos 0.05. (En la tarea de evaluación se propondrán los valores iniciales para los correspondientes parámetros).

Con los elementos anteriores, ya podemos utilizar la función `nls()`. Utilizaremos la función `T_model` que acabamos de definir para el valor del argumento `formula`, e indicaremos los valores iniciales de los parámetros en el argumento `start`, en forma de lista con la función `list()`:

```
fit_newton <- nls(
    data = newton,
    formula = temp ~ T_model(time, Ta, T0, k),
    start = list(Ta = 20, T0 = 100,  k = 0.05)
)
```

3.3. Coeficientes

Los coeficientes del ajuste son:

```
coefficients(fit_newton)
```

```
##           Ta           T0            k
##  19.02958737 104.98070986   0.05994108
```

Redondeando los resultados a dos decimales, las estimaciones obtenidas son

$$\hat{T_a} = 19.03,$$

$$\hat{T_0} = 104.98,$$

y

$$\hat{k} = 0.06,$$

y el ajuste quedaría:

$$T(t) = 19.03 + (104.98 - 19.03)e^{-0.06t}.$$

3.4. Predicciones

Los pronósticos para las temperaturas a la que se encontrará la barra de hierro a los 70, 80 y 90 minutos serían:

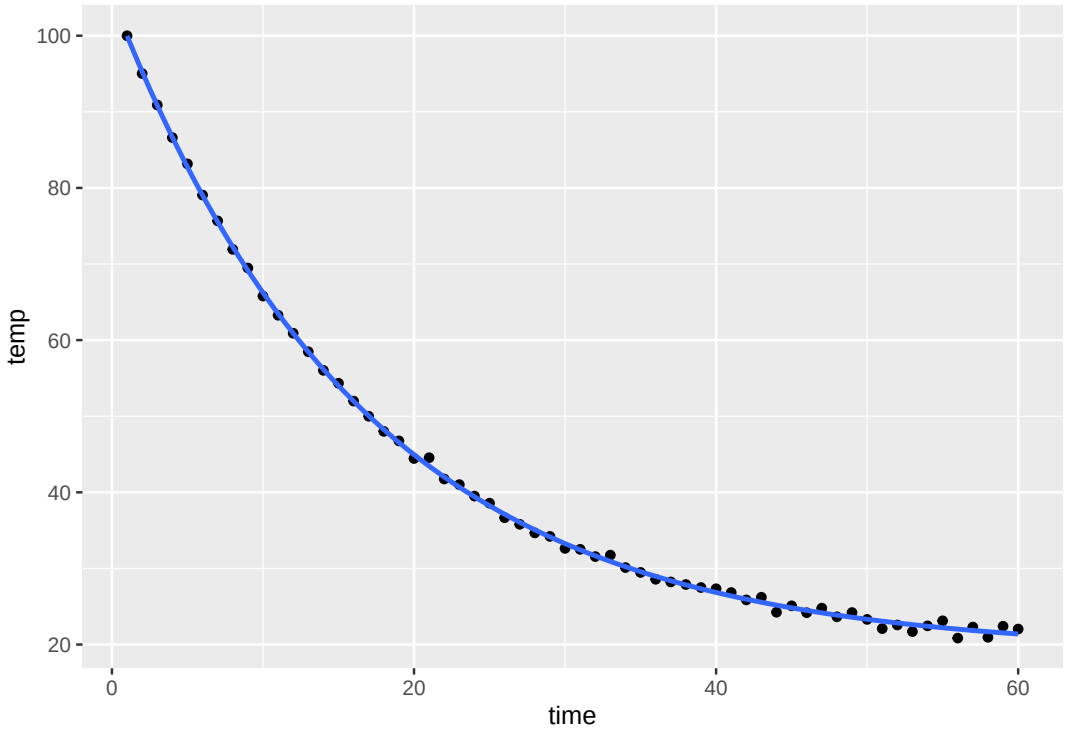
```
predict(
    fit_newton,
    tibble(time = c(70, 80, 90))
)
```

```
## [1] 20.32380 19.74029 19.41986
```

3.5. Gráfico ajuste

Como en el capítulo anterior usamos la función `geom_smooth()` para representar el ajuste:

```
ggplot(
    data = newton,
    mapping = aes(x = time, y = temp)
) +
    geom_point() +
    geom_smooth(
        method = "nls",
        formula = y ~ T_model(x, T0, Ta, k),
        method.args = list(
            start = list(T0 = 100, Ta = 20, k = 0.05)
        ),
        se = FALSE # intervalo de confianza no implementado en predict.nls
    )
```



En los argumentos de `geom_smooth()` hemos escrito ahora `method = "nls"` para indicar que el ajuste se realiza con la función `nls()`.

El argumento `start` que requiere la función `nls()` se especifica en `method.args = list(...)`.

Notar que hemos usado nuestra función `T_model()` para especificar la fórmula del modelo, como al usar la función `nls()`, pero que en este caso no se utilizan los nombres `time` y `temp` de las variables sino los nombres `x` e `y` de las correspondientes estéticas.

El argumento `se = FALSE` es necesario en este caso (si no lo incluyes obtendrás un error) por una cuestión técnica que no vamos a detallar.