Solutions to Exercises from Chapter 07

Contents

The `comphy` package is loaded once at the beginning so to make all its functions available to this exercises session.

```
library(comphy)
```

1 Exercises on differentiation

1.1 Exercise 01

Consider the simple function $f(x) = x + e^x + sin(x)$. Calculate the first derivative at $x = 0$ using forward, backward and centred differences. Use $h = 0.1$ and $h = 0.01$. Verify that the numerical errors are $\mathrm{O}(h)$ and $\mathrm{O}(h^2)$.

SOLUTION

The first derivative can ba calculated analytically and it is given by

$$f'(x) = 1 + e^x + \cos(x).$$

At $x = 0$ we have $f'(0) = 3$, exactly.

The `comphy` function to calculate fotward, backward, and centred difference derivatives is `deriv_reg()`. The necessary input are values at a regular grid and the corresponding values tabulated for the function. So, we will need to generate the regular grid first, using the R function `seq`. We can do that via the parameter `by`. As the derivative must be calculated at $x = 0$, we should create the grid so to contain that value. Also, as only one value is required, we don't need to use a large grid (a grid with many values); in this case, a 5-values grid will suffice. The grid will contain points

$$-0.2, -0.1, 0, 0.1, 0.2$$

when $h = 0.1$, and

$$-0.02, -0.01, 0, 0.01, 0.02$$

when $h = 0.01$. We can also use the parameter `length.out=5` in `seq`, to decide the grid size.

```
# Create the grid around x=0 in two ways, adding the final point or without it
# Adding the final point
x1 <- seq(-0.2,0.2,by=0.1) # Grid when h=0.1
x2 <- seq(-0.02,0.02,by=0.01) # Grid when h=0.01
print(x1)
#> [1] -0.2 -0.1  0.0  0.1  0.2
print(x2)
#> [1] -0.02 -0.01  0.00  0.01  0.02
# Without final point
x1 <- seq(-0.2,by=0.1,length.out=5)
x2 <- seq(-0.02,by=0.01,length.out=5)
print(x1)
#> [1] -0.2 -0.1  0.0  0.1  0.2
print(x2)
#> [1] -0.02 -0.01  0.00  0.01  0.02
```

Once the grid is created, the function's tabulated points are also easily created, using the grid's variable as the x in the analytic expression of the function.

```
# Create tabulated points
f1 <- x1 + exp(x1) + sin(x1)
f2 <- x2 + exp(x2) + sin(x2)
```

For this specific exercise, the derivative is required only at $x = 0$. The variable `x0` needed for the `comphy` function to work must include one or more exact grid points of `x`. Here the only point needed is $x = 0$, which happens to be the third grid point of both grid `x1` and `x2`. The derivatives are then easily calculated with `deriv_reg()`, and subsequently printed for display, using formatting to appreciate the accuracy of the numbers calculated.

```
# Only one point is needed for the derivative.
# This must be, exactly, one of the grid points, or the function stops
# h=0.1
x0 <- x1[3]
f1derC <- deriv_reg(x0,x1,f1) # Default scheme is with centred difference
f1derF <- deriv_reg(x0,x1,f1,scheme="f") # Forward difference
f1derB <- deriv_reg(x0,x1,f1,scheme="b") # Backward difference
sprintf("%10.7f, %10.7f, %10.7f",f1derC,f1derF,f1derB)
#> [1] " 3.0000017,  3.0500433,  2.9499600"
# h=0.01
x0 <- x2[3]
f1derC <- deriv_reg(x0,x2,f2) # Default scheme is with centred difference
f1derF <- deriv_reg(x0,x2,f2,scheme="f") # Forward difference
f1derB <- deriv_reg(x0,x2,f2,scheme="b") # Backward difference
sprintf("%10.7f, %10.7f, %10.7f",f1derC,f1derF,f1derB)
#> [1] " 3.0000000,  3.0050000,  2.9950000"
```

The errors are indeed $\mathrm{O}(h)$ or better for $h = 0.1$ and $h = 0.01$ when forward and backward differences are used. When $h = 0.1$ we have

$$f'_{\text{true}}(0) - f'_F(0) = 3 - 3.0500433 = -0.0500433, \qquad f'_{\text{true}}(0) - f'_B(0) = 3 - 2.9499600 = 0.0500400$$

and, when $h = 0.01$, we have

$$f'_{\text{true}}(0) - f'_F(0) = 3 - 3.0050000 = -0.0050000, \qquad f'_{\text{true}}(0) - f'_B(0) = 3 - 2.9950000 = 0.0050000.$$

The accuracy is higher when centred differences are used. For $h = 0.1$ we should have $\mathrm{O}(h^2) = \mathrm{O}(0.01)$. Indeed:

$$f'_{\text{true}}(0) - f'_C(0) = 3 - 3.0000017 = -0.000017,$$

which means that the accuracy is better in this case. And for $h = 0.01$ we should have an accuracy equal to $\mathrm{O}(0.0001)$ which, to the significant figured shown, is certainly true.

1.2 Exercise 02

Use the function $f(x) = e^x$ around $x = 1$, and values of h between 0.001 and 0.1 to show that the error goes like $\mathrm{O}(h^2)$.

SOLUTION

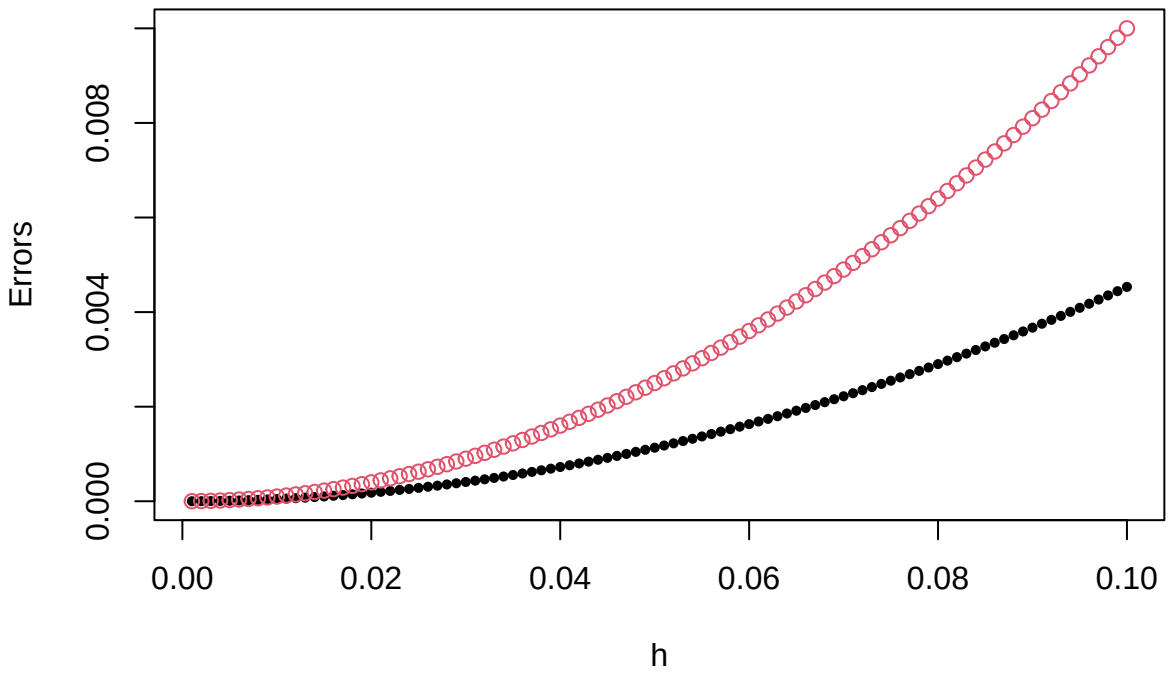
The grid can, as in the previous exercise, consist of just 5 grid points centred at $x = 1$. Then we apply `deriv_reg()` for as many values as h as it is necessary. In the exercise it is not explained how many values of h to use, but we can very simply create a regular grid of h between 0.001 and 0.1, with step 0.001.

In the code here presented, a loop over the values of h is carried out and values of the centred difference derivative are stored.

```
## Exercise on the derivative of f(x)=exp(x) at x=1

# Grid of values for h
hh <- seq(0.001,0.1,by=0.001)

# Loop over all values of h.
# At each loop we need to redefine the grid and tabulate f(x)
DfT <- exp(1) # True derivative
DfC <- c()
for (h in hh) {
  x <- seq(1-2*h,by=h,length.out=5)
  f <- exp(x)
  tmp <- deriv_reg(x[3],x,f)
  DfC <- c(DfC,tmp)
}
```

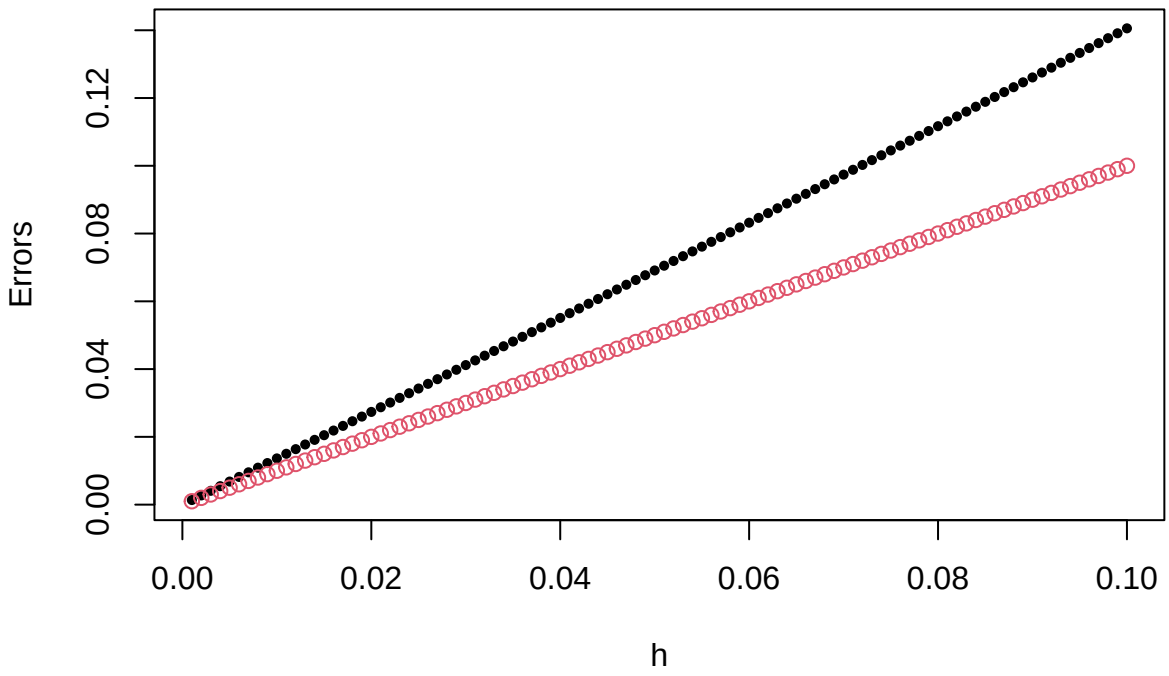
The errors for all values of h can be plotted and compared to a quadratic curve (because of the dependency on h^2). As we are comparing with h^2, which is a non negative function, we will use the absolute value of the errors.

```
# Errors (the true derivative is exp(1))
Deltas <- abs(DfT - DfC)

# Function h^2 compared graphically to the numerical derivative
ff <- hh^2
ylim <- range(ff,Deltas)
plot(hh,Deltas,type="b",pch=16,cex=0.7,ylim=ylim,xlab="h",ylab="Errors")
points(hh,ff,type="b",col=2)
```



The two curves do not overlap but have a similar shape, the quadratic shape. When it is said that the error are $O(h^2)$, this must be intended as *order of magnitude*. A similar exercise done using forward difference derivatives, shown in the following code snippet, displays indeed a linear trend because of errors there being $O(h)$.

```
# Loop over all values of h.
# At each loop we need to redefine the grid and tabulate f(x)
DfF <- c()
for (h in hh) {
  x <- seq(1-2*h,by=h,length.out=5)
  f <- exp(x)
  tmp <- deriv_reg(x[3],x,f,scheme="f")
  DfF <- c(DfF,tmp)
}

# Errors (the true derivative is exp(1))
Deltas <- abs(DfT - DfF)

# Function h^2 compared graphically to the numerical derivative
ff <- hh
ylim <- range(ff,Deltas)
plot(hh,Deltas,type="b",pch=16,cex=0.7,ylim=ylim,xlab="h",ylab="Errors")
points(hh,ff,type="b",col=2)
```



1.3 Exercise 03

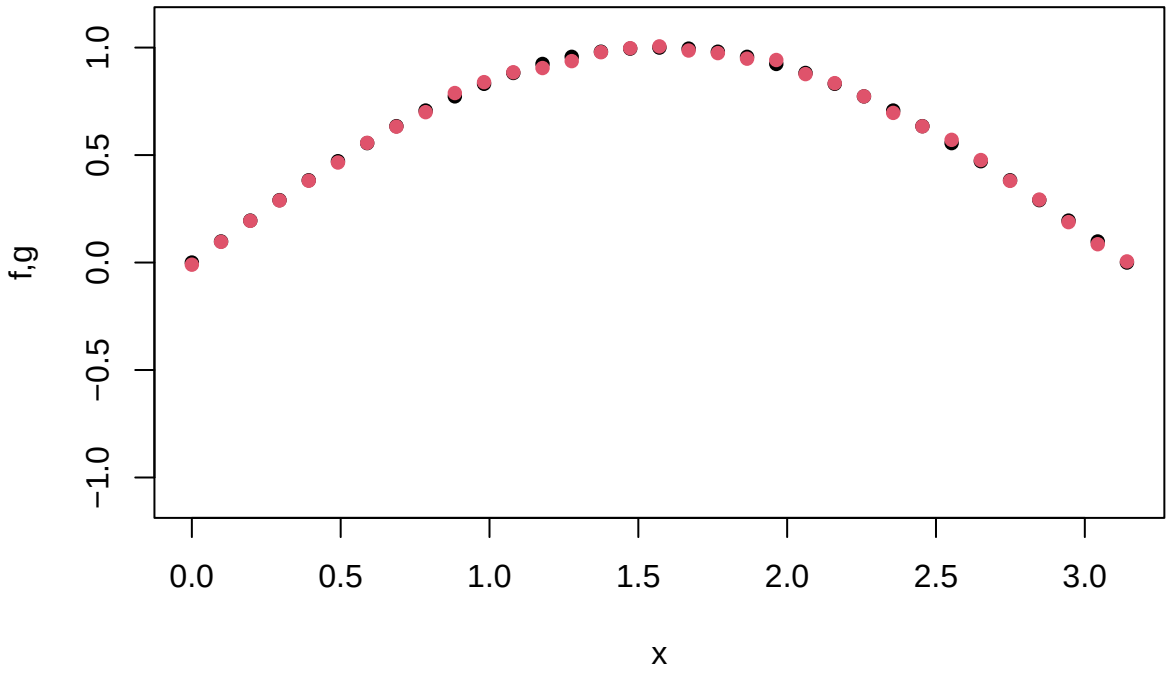
Experimental or measurement errors on sampled points of a function can amplify the errors of a numerical derivative. In this exercise, create a function, $f(x) = \sin(x)$ at 33 regularly spaced points between 0 and π (h is then roughly 0.1). Then create a function $g(x) = f(x) + \epsilon$, where ϵ is a normal random variable extracted from a distribution with mean 0 and standard deviation 0.01. Plot the two functions together to appreciate their difference. Next, calculate the centred difference derivative at all points (clearly excluding 0 and π) for both $f(x)$ and $g(x)$. Plot both derivatives and appreciate how the differences for them are amplified, compared to the differences of the functions. Finally, find the maximum error for both derivatives with respect to the true derivative, $f'(x) = \cos(x)$.

SOLUTION

First we create the tabulated values for $f(x)$ and $g(x)$ and plot them to visually show that they are visually quite close.

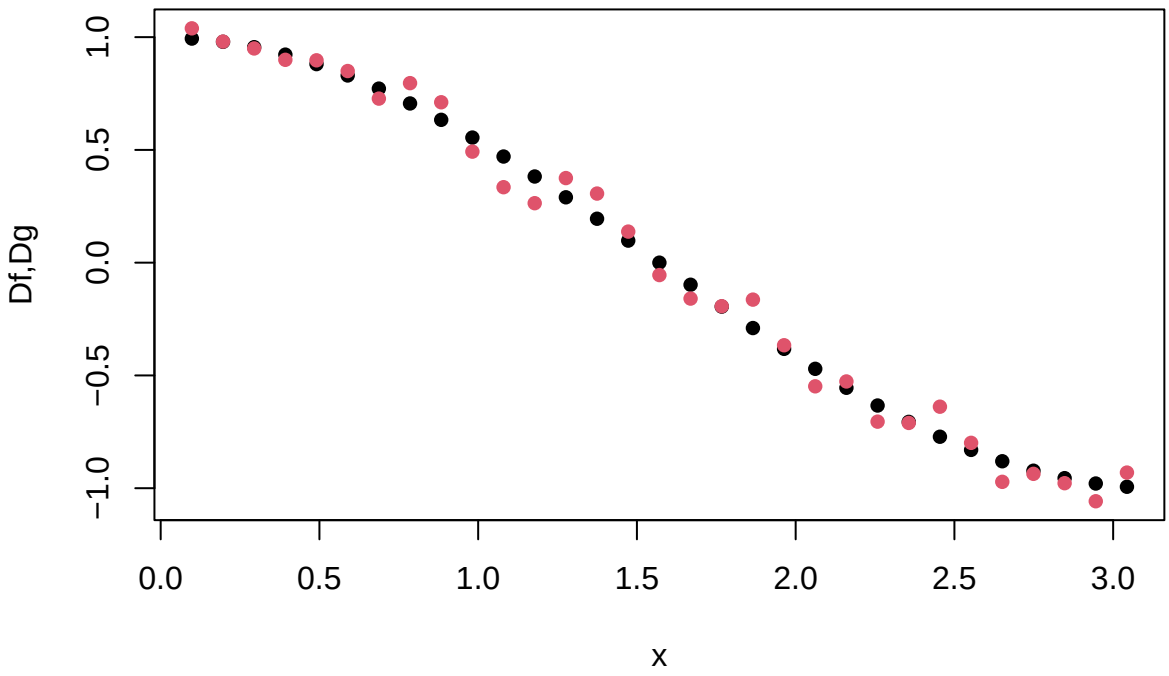
```
# x grid
x <- seq(0,pi,length.out=33)

# Simulated errors. The seed is added for reproducibility
set.seed(167)
eps <- rnorm(n=33,mean=0,sd=0.01)

# Tabulated values for f and g
f <- sin(x)
g <- sin(x) + eps
```

```
# Plots
plot(x,f,pch=16,ylim=c(-1.1,1.1),xlab="x",ylab="f,g")
points(x,g,pch=16,col=2)
```



The plots of the function with and without errors are very similar, as the standard deviation is small compared to the scale of the plot, $\sigma = 0.01$. Next, we calculate the centred difference derivative for both $f(x)$ and $g(x)$. The plots show that they are substantially different.

```
# Centred difference derivative of f(x)
Df <- deriv_reg(x0=x[2:32],x=x,f=f,scheme="c")

# Centred difference derivative of g(x)
Dg <- deriv_reg(x0=x[2:32],x=x,f=g,scheme="c")

# Plots
ylim=range(Df,Dg)
plot(x[2:32],Df,pch=16,ylim=ylim,xlab="x",ylab="Df,Dg")
points(x[2:32],Dg,pch=16,col=2)
```



The two plots are visibly different. It would be nice to calculate the errors of both numerical derivatives by comparison with the correct derivative, $f'(x) = \cos(x)$. This is done in the next code snippet, where the absolute value of the maximum error for both $f'(x)$ and $g'(x)$ is calculated.

```
# Correct first derivative at the grid points of interest
Dtrue <- cos(x[2:32])

# Max error for numerical derivative of f
Emaxf <- max(abs(Df - Dtrue))
print(Emaxf)
#> [1] 0.001597876

# Max error for numerical derivative of g
Emaxg <- max(abs(Dg - Dtrue))
print(Emaxg)
#> [1] 0.136778
```

As we can see, the maximum error for $g'(x)$ is much larger than the numerical error due to the numerical calculation of the derivative: $0.137 \gg 0.002$. This must be taken into account when calculating derivatives of empirical functions.

1.4 Exercise 04

Find the numerical first derivative of $f(x) = x^2 - 3\cos(2x) + e^x$ at $x_0 = -\pi/6, 0, \pi/6$, when the function is tabulated at the 7 points $x = -\pi/2, -\pi/3, -\pi/4, 0, \pi/4, \pi/3, \pi/2$. Compare the values found with the values of the exact, analytic derivative.

SOLUTION

The tabulated values must be initially created as vectors `x` and `f`. Then the values at which the first derivative needs to be calculated will be stored in another vector, `x0`. The numerical derivative is calculated straight away with `deriv_irr`.

```
# Tabulated values of function
x <- c(-pi/2,-pi/3,-pi/4,0,pi/4,pi/3,pi/2)
f <- x^3-3*cos(2*x)+exp(x)

# Values for the derivative
x0 <- c(-pi/6,0,pi/6)

# Numerical derivative
f1 <- deriv_irr(x0,x,f)
print(f1)
#> [1] -3.784234  1.000351  7.709251
```

The analytic expression of the derivative is $f'(x) = 3x^2 + 6\sin(2x) + e^x$. It is therefore possible to compute the exact numerical values of the derivative at the points assigned, and compare them with the corresponding approximated values.

```
# Exact analytic derivative at the x0 values
f1exact <- 3*x0^2+6*sin(2*x0)+exp(x0)

# Comparison
print(f1)
#> [1] -3.784234  1.000351  7.709251
print(f1exact)
#> [1] -3.781301  1.000000  7.706711
```

1.5 Exercise 05

Considering the case in Exercise 04, try and give an estimate of the errors associated with the numerical derivatives calculated at the 7 grid points.

SOLUTION

Let us reproduce in memory all values needed.

```
# Reproduce data
x <- c(-pi/2,-pi/3,-pi/4,0,pi/4,pi/3,pi/2)
f <- x^3-3*cos(2*x)+exp(x)

# The values for the derivative are the same grid points
f1 <- deriv_irr(x,x,f)

# Exact values
f1exact <- 3*x^2+6*sin(2*x)+exp(x)

# Comparison and absolute error
print(f1)
#> [1]  7.330501 -1.532406 -3.706936  1.000351 10.056791 11.313332 12.486818
print(f1exact)
#> [1]  7.610083 -1.555364 -3.693511  1.000000 10.043831 11.335674 12.212681
aerr <- abs(f1-f1exact)
print(aerr)
#> [1] 0.2795818009 0.0229586023 0.0134245381 0.0003509314 0.0129600509
#> [6] 0.0223429085 0.2741370151
```

It is interesting to observe, as expected, that the smallest errors occur in the central region of the grid because all differences of the type $x - x_i$ tend to be smaller than when x is close to the extreme of the interval.

The error is given by the formula

$$\Delta f'(x_i) = \frac{f^{(7)}(\xi)}{(n+1)!}(x_i - x_1)\cdots(x_i - x_{i-1})(x_i - x_{i+1})\cdots(x_i - x_7).$$

We don't know how to evaluate $f^{(7)}(\xi)/7!$ exactly. But we can calculate $f^{(7)}$ analytically and try and work out its minimum and maximum in the range $[-\pi/2, \pi/2]$. It turns out that

$$f^{(7)}(x)/7! = (e^x - 384\sin(2x))/5040.$$

Now, the largest value of e^x in the interval $[-\pi/2, \pi/2]$ is approximately 4.8105 and $\sin(2x)$ varies between -1 and $+1$. Therefore the largest value (in magnitude) for $f^{(7)}(\xi)/7!$ is 0.077145. This is what we need to give upper bounds for the error, as calculated in the following chunk.

```
# Maximum of f^(7)(xi)/7!
k <- (exp(pi/2)+384)/5040
print(k)
#> [1] 0.07714494

# Largest values for the error, depending on x_i used
idx <- 1:7
err <- c()
for (i in 1:7) {
  err <- c(err,k*abs(prod((x[i]-x)[idx[-i]])))
}

# Display and compare (TRUE or FALSE)
print(err)
#> [1] 0.96570713 0.11127490 0.08449937 0.12876095 0.08449937 0.11127490 0.96570713
print(aerr)
#> [1] 0.2795818009 0.0229586023 0.0134245381 0.0003509314 0.0129600509
#> [6] 0.0223429085 0.2741370151
aerr <= err
#> [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

The actual errors are always less or equal than the largest expected margin. It is important to underlie, though, that it is not always possible to provide a sensible range of values to the quantity $f^{(n+1)}(\xi)/(n+1)!$.

1.6 Exercise 06

Using reasoning similar to the one used to derive the three-point formula for the centred difference derivative, derive a three-point formula for the second derivative. Apply the formula found to calculate numerically the second derivative of the function $f(x) = x^2$ on the regular grid from 0 to 1, using $h = 0.01$.

SOLUTION

The starting point are the Taylor expansions in powers of h and $-h$:

$$f(x_i + h) = f(x_i) + f'(x_i)h + \frac{f''(x_i)}{2}h^2 + \frac{f'''(x_i)}{6}h^3 + \frac{f^{iv}(x_i)}{24}h^4 + \mathrm{O}(h^5)$$

$$f(x_i - h) = f(x_i) - f'(x_i)h + \frac{f''(x_i)}{2}h^2 - \frac{f'''(x_i)}{6}h^3 + \frac{f^{iv}(x_i)}{24}h^4 - \mathrm{O}(h^5).$$

Adding these formula yields:

$$f(x_i + h) + f(x_i - h) = 2f(x_i) + f^{''}(x_i)h^2 + \frac{f^{iv}(x_i)}{12}h^4 + \mathrm{O}(h^6).$$

The second derivative can be obtained by reversing this formula and it is found it is given by knowledge of the three points, $x_i - h, x_i, x_i + h$:

$$f^{''}(x_i) = \frac{f(x_i + h) + f(x_i - h) - 2f(x_i)}{h^2} + \mathrm{O}(h^2).$$

Let us now apply the formula to calculate the second derivative of $f(x) = x^2$ between 0 and 1. We should find a constant quantity as $f^{''}(x) = 2$. In the following section of code, given to the presence of $x_i - h$, x_i, $x_i + h$, we will not be able to calculate f^{ii} for the first and last point of the grid.

```
# Step (h)
h <- 0.01

# Grid
x <- seq(0,1,by=h)

# Function
f <- x^2

# Length of grid
n <- length(x)

# Second derivative (three-point formula)
DDf <- rep(NA,length.out=n) # First and last grid points not used
DDf[2:(n-1)] <- (f[1:(n-2)]+f[3:n]-2*f[2:(n-1)])/h^2
print(DDf)
#>   [1] NA  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
#>  [26]  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
#>  [51]  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
#>  [76]  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
#> [101] NA
```

In this specific example errors are not immediately visible, given the simple polynomial but things are in general different with other types of functions.

It is also of some interest if the same result can be found if the formula for the first derivative is applied twice.

```
# First derivative (centred difference)
Dcf <- deriv_reg(x,x,f)
print(Dcf)
#>   [1]   NA 0.02 0.04 0.06 0.08 0.10 0.12 0.14 0.16 0.18 0.20 0.22 0.24 0.26 0.28
#>  [16] 0.30 0.32 0.34 0.36 0.38 0.40 0.42 0.44 0.46 0.48 0.50 0.52 0.54 0.56 0.58
#>  [31] 0.60 0.62 0.64 0.66 0.68 0.70 0.72 0.74 0.76 0.78 0.80 0.82 0.84 0.86 0.88
#>  [46] 0.90 0.92 0.94 0.96 0.98 1.00 1.02 1.04 1.06 1.08 1.10 1.12 1.14 1.16 1.18
#>  [61] 1.20 1.22 1.24 1.26 1.28 1.30 1.32 1.34 1.36 1.38 1.40 1.42 1.44 1.46 1.48
#>  [76] 1.50 1.52 1.54 1.56 1.58 1.60 1.62 1.64 1.66 1.68 1.70 1.72 1.74 1.76 1.78
#>  [91] 1.80 1.82 1.84 1.86 1.88 1.90 1.92 1.94 1.96 1.98   NA

# Second derivative - One more first derivative
DDcf <- deriv_reg(x,x,Dcf)
print(DDcf)
#>   [1] NA NA  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
```

```
#>  [26]  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
#>  [51]  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
#>  [76]  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2 NA
#> [101] NA
```

The result is the same, but now the second derivative could not be computed at four, rather than two points, having used centred difference twice.

2 Exercises on Integration

2.1 Exercise 07

Calculate the following integral,

$$\frac{1}{\sqrt{2\pi}} \int_{-1}^{+1} e^{-x^2/2} \, \mathrm{d}x,$$

numerically using the trapezoid, and Simpson's 1/3 and 3/8 rules. Compare the results obtained with those displayed with the R function `pnorm`.

SOLUTION

The given integral is a definite integral of the Gaussian function. To be more specific, the given function is normalised so that

$$\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} e^{-x^2/2} \, \mathrm{d}x = 1.$$

Therefore, the definite integral will be smaller than 1. The integral

$$\Phi(x) \equiv \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2} \, \mathrm{d}t$$

is calculated in R with the function `pnorm`. For example, the integral between $-\infty$ and 0 must be 1/2 because the integral from $-\infty$ to $+\infty$ is 1. Indeed,

```
print(pnorm(0))
#> [1] 0.5
```

Accordingly, the requested integral can be given as difference of Φ functions,

$$\frac{1}{\sqrt{2\pi}} \int_{-1}^{+1} e^{-x^2/2} \, \mathrm{d}x = \Phi(+1) - \Phi(-1)$$

that is, difference of `pnorm` at $+1$ and -1:

```
print(pnorm(1)-pnorm(-1))
#> [1] 0.6826895
```

Now that we know the correct value,let us calculate it with the requested approximations. The range $[-1, 1]$ can be divided into an even number of intervals which is also a multiple of 3 so that all rules can be applied in full with a same grid; this number is chosen as $n = 198$. Initially the appropriate grid and the tabulated values must be created:

```
# Grid (divisible by 3 and even)
n <- 198
x <- seq(-1,1,length.out=(n+1))

# Tabulated values
f <- exp(-x^2/2)/sqrt(2*pi)
```

then the three algorithms are applied using the function `numint_reg`.

1. **Trapezoid**

```
value <- numint_reg(x,f,scheme="trap")
print(value)
#> [1] 0.6826854
```

2. **Simpson's 1/3**

```
value <- numint_reg(x,f,scheme="sim13")
print(value)
#> [1] 0.6826895
```

3. **Simpson's 3/8**

```
value <- numint_reg(x,f,scheme="sim38")
print(value)
#> [1] 0.6826895
```

We can see that while the accuracy is quite satisfactory even with the trapezoid rule, Simpson's 1/3 reproduces the correct results to seven decimals. Obviously, Simpson's 3/8 does it as well, but as the accuracy of both hmethods is the same ($\mathrm{O}(h^4)$), Simpson's 1/3 is normally the preferred method of choice, given its relative algorithmic simplicity.

2.2 Exercise 08

Consider the following *complete elliptic integral of the first kind*,

$$K(k) \equiv \int_0^{\pi/2} \frac{\mathrm{d}\theta}{1 - k^2 \sin^2(\theta)},$$

where $k \in (-1, 1)$. Calculate $K(0.5)$ numerically using `numint_reg` and compare your result with that calculated using the R package `elliptic`.

SOLUTION

The integrand when $k = 0.5$ is given by the following function:

$$f(\theta) = \frac{1}{\sqrt{1 - 0.25 \sin^2(\theta)}}.$$

The numerical integral must be calculated between 0 and $\pi/2$. We can use a grid with $n = 200$.

```
# Grid
x <- seq(0,pi/2,length.out=201)

# Function values
f <- 1/(sqrt(1-0.25*(sin(x))^2))

# Integral
nvalue <- numint_reg(x,f)
pval <- sprintf("%10.8f\n",nvalue)
cat(pval)
#> 1.68575035
```

One of the R packages that calculates the complete elliptic integrals of the first kind is *elliptic*. Its function `K.fun` does the job. The parameter `m` of the package is what in the above notation has been indicated as k^2. So:

```r
# Make sure library(elliptic) has loaded the package
require(elliptic)
#> Loading required package: elliptic
#>
#> Attaching package: 'elliptic'
#> The following objects are masked from 'package:stats':
#>
#>     sd, sigma
#> The following object is masked from 'package:base':
#>
#>     is.primitive

# Integral. m=k^2=0.5^2=0.25
nvalue <- K.fun(m=0.25)
pval <- sprintf("%10.8f\n",nvalue)
cat(pval)
#> 1.68575035
```

the numeric values are identical to a high accuracy.

2.3 Exercise 09

Calculate the local error for Simpson's 3/8 rule and verify that the result is $-(3/80)h^5 f^{(4)}(\xi)$.

SOLUTION

The interpolation error, given that for SImpson's 3/8 rule we use x_i, x_{i+1}, x_{i+2}, and x_{i+3}, is

$$\Delta P_3(x) = \frac{f^{(4)}(\xi)}{4!}(x - x_i)(x - x_{i+1})(x - x_{i+2})(x - x_{i+3}).$$

Using the integration variable $s = (x - x_i)/h$, we can then write the local error as the following integral:

$$\text{local error} = \frac{f^{(4)}(\xi)}{4!} h^5 \int_0^3 s(s-1)(s-2)(s-3)\, \mathrm{d}s.$$

The definite integral in the above expression turns out to be $-9/10$. Therefore:

$$\text{local error} = \frac{f^{(4)}(\xi)}{4!} h^5 \left(-\frac{9}{10}\right) = -\frac{3}{80} h^5 f^{(4)}(\xi),$$

as suggested by the exercise.

2.4 Exercise 10

The global error when applying the trapezoid rule is

$$\frac{a-b}{12} f^{(2)}(\xi) h^2.$$

While it is not possible to know the value of $f^{(2)}(\xi)$, the quantity $(a-b)/12$ is constant and, even though $f^{(2)}(\xi)$ varies across the interval (x_1, x_{n+1}), it will be bounded by a finite number, in general comparable with the values that the function takes in the integration interval. Accordingly, the global error should show a square dependency on h.

Use the trapezoid rule to integrate $f(x) = x^2 - 1$ in the interval $[-1, 1]$, for many values of h when $[-1, 1]$ is divided into $n = 20, 21, \ldots, 39, 40$ equal intervals. Plot the global error versus the corresponding values of h.

You should verify visually that the set of points in the plot follows a curved, rather than straight, pattern. Can we ascertain that the curve is a quadratic?

SOLUTION

A plot of the global error vs h should appear as a parabola. To calculate the error we need to know the correct value of the integral. This is readily calculated:

$$I = \int_{-1}^{+1} (x^2 - 1)\, \mathrm{d}x = \left[\frac{x^3}{3} - x\right]_{-1}^{+1} = \frac{1}{3} - 1 + \frac{1}{3} - 1 = -\frac{4}{3} \approx -1.33.$$

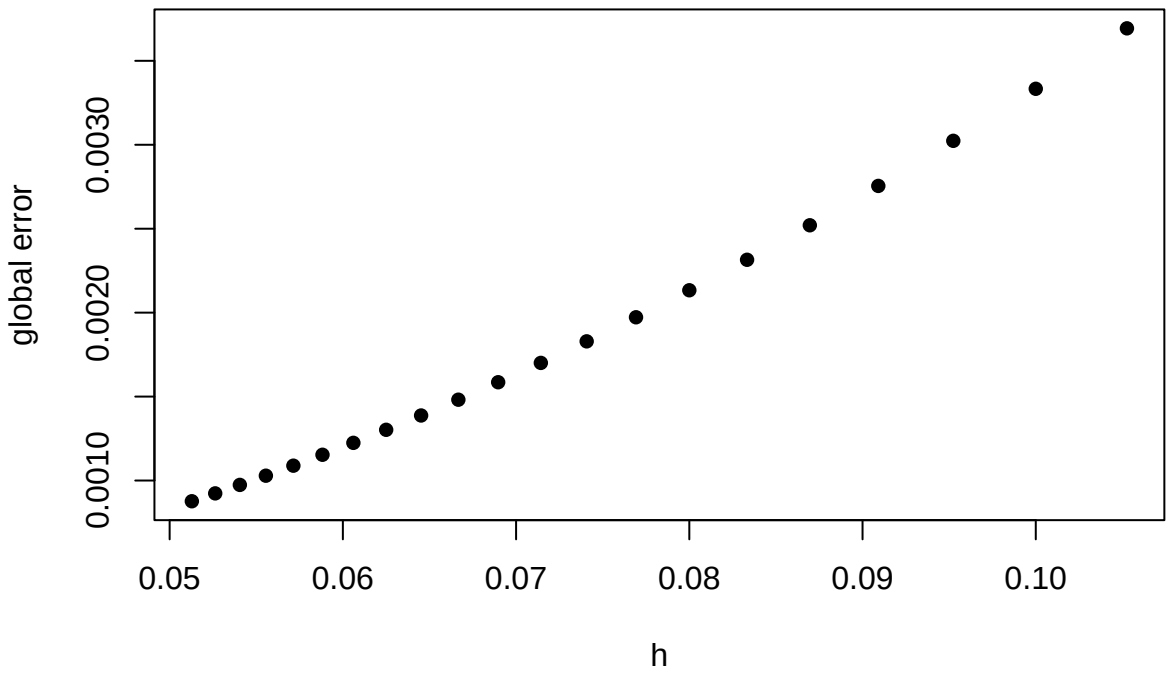
If $J(h)$ indicates the approximated value of the integral using the trapezoid rule with a specific value of h, the global error will be

$$\Delta J(h) = J(h) - I.$$

We need to plot the values of $\Delta J(h)$ versus h.

The values of $J(h)$ can be easily obtained using `numint_reg()`, with `scheme="trap"`. Also, the way to determine the value of h, given the number of regularly-spaced intervals, is through the difference, say, of the first two grid points.

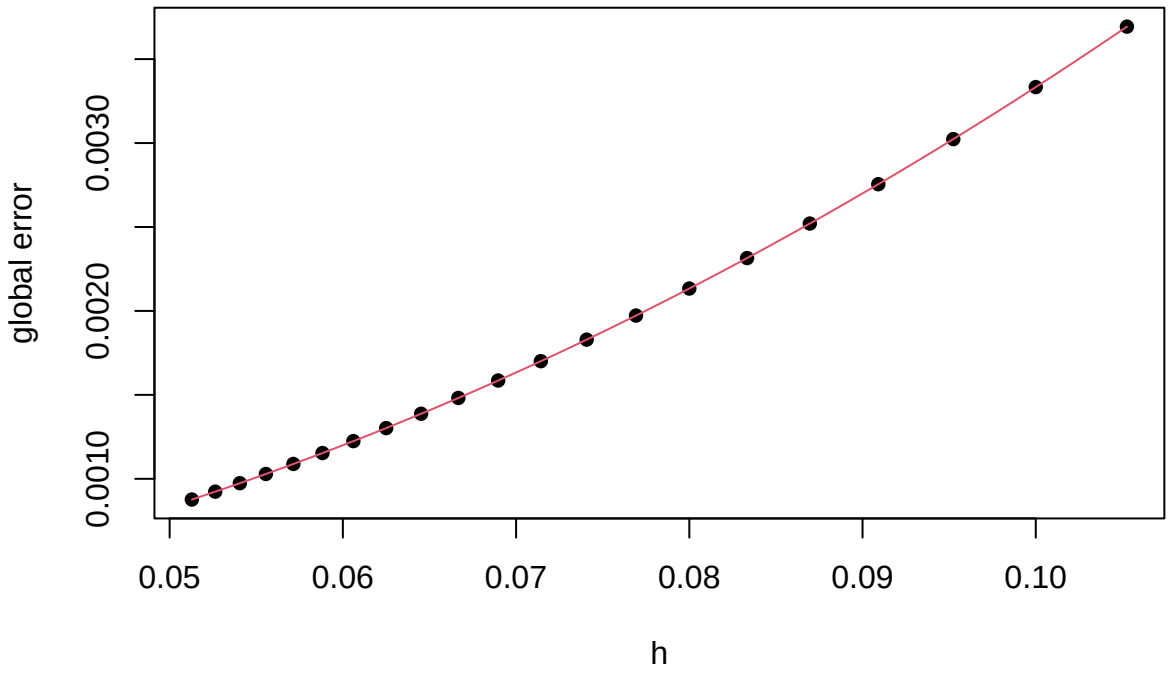
```
# Correct integral
I <- -4/3

# Define range of intervals (n)
n <- 20:40

# Vector of h
h <- c()

# Vector of global errors
ge <- c()

# Main loop
for (m in n) {
  # Define x grid and corresponding f's
  x <- seq(-1,1,length.out=m)
  f <- x^2-1

  # Store h
  h <- c(h,x[2]-x[1])

  # Numeric integral
  J <- numint_reg(x,f,scheme="trap")

  # Global error
  ge <- c(ge,J-I)
}

# Plot
plot(h,ge,pch=16,xlab="h",ylab="global error")
```



It indeed appears that the set of points visually curves. But to ascertain that they fit a quadratic curve, we need to carry out a linear regression with h^2.

```
# Linear regression
mdel <- lm(ge ~ I(h^2))
summary(mdel)
#>
#> Call:
#> lm(formula = ge ~ I(h^2))
#>
#> Residuals:
#>        Min         1Q     Median         3Q        Max
#> -1.138e-15 -5.888e-16 -1.601e-16  5.575e-16  1.245e-15
#>
#> Coefficients:
#>              Estimate Std. Error   t value Pr(>|t|)
#> (Intercept) 6.738e-17  3.915e-16 1.720e-01    0.865
#> I(h^2)      3.333e-01  6.531e-14 5.103e+12   <2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 7.328e-16 on 19 degrees of freedom
#> Multiple R-squared:      1,  Adjusted R-squared:      1
#> F-statistic: 2.605e+25 on 1 and 19 DF,  p-value: < 2.2e-16

# Plot data and regression line
xx <- seq(min(h),max(h),length.out=100)
```

```
yy <- predict(mdel,newdata=data.frame(h=xx))
plot(h,ge,pch=16,xlab="h",ylab="global error")
points(xx,yy,type="l",col=2)
```



The regression's statistics are strongly in favour of a quadratic curve, thus conferming the theoretical results.

2.5 Exercise 11

Use 2-point and 3-point Gaussian quadrature to estimate numerically the integral

$$\int_{-2}^{3} x^4 \, \mathrm{d}x.$$

What difference do you observe in going from the 2-point to the 3-point quadrature?

SOLUTION

An n-point Gaussian quadrature can give the exact estimate of definite integrals of polynomials of up to degree $2n-1$. For $n=2$, the exact estimate is for polynomials of degree up to 3, while for $n=3$, the degree goes up to 5. As the integrand in the exercise is a polynomial of degree 4, the 2-point quadrature will not return the exact value but the 3-point quadrature will.

The correct value of the integral is

$$\int_{-2}^{3} x^4 \, \mathrm{d}x = \left[\frac{x^5}{5}\right]_{-2}^{3} = \frac{243}{5} + \frac{32}{5} = \frac{275}{5} = 55.$$

Let us, next, use the 2-point Gaussian quadrature:

```r
# Function
f <- function(x) {ff <- x^4; return(ff)}

# 2-point quadrature
ltmp <- Gquad(f,-2,3,n=2)
print(ltmp$itg)
#> [1] 37.63889
```

The result is not very close to 55, as expected. But with three points, exactness will be reached.

```r
# 3-point quadrature
ltmp <- Gquad(f,-2,3,n=3)
print(ltmp$itg)
#> [1] 55
```

2.6 Exercise 12

Calculate numerically the integral

$$\frac{1}{\sqrt{2\pi}} \int_{-1}^{+1} e^{-x^2/2} \, \mathrm{d}x,$$

using Gaussian quadrature, what order n is necessary for the result to be comparable with the one given by the function `pnorm`?

SOLUTION

The value of the given definite integral can be calculated as difference $\Phi(1) - \Phi(-1)$, where

$$\Phi(x) = \frac{1}{\sqrt{2}} \int_{-\infty}^{x} e^{-t^2/2} \, \mathrm{d}t.$$

Clearly, $\Phi(-\infty) = 0$ and $\Phi(+\infty) = 1$. The function $\Phi(x)$ is calculated numerically with high precision using `pnorm`. For example

```r
# A very large, negative value is close to -infinity
print(pnorm(-1000))
#> [1] 0

# A very large, positive value is close to +infinity
print(pnorm(1000))
#> [1] 1
```

Therefore, an accurate numerical value for the given definite integral is

```r
Tvalue <- pnorm(1)-pnorm(-1)
res <- sprintf("Accurate value: %14.12f\n",Tvalue)
cat(res)
#> Accurate value: 0.682689492137
```

We can now observe the errors when using Gaussian quadrature as differences with the value `Tvalue` found.

```r
# Function
f <- function(x) {ff <- exp(-x^2/2)/sqrt(2*pi); return(ff)}

# 3-point quadrature
ltmp <- Gquad(f,-1,1,n=3)
res <- sprintf("3-point quadrature\n %14.12f\n",ltmp$itg)
cat(res)
#> 3-point quadrature
```

```
#>  0.682997260714
res <- sprintf("Absolute difference: %14.12f\n",abs(ltmp$itg-Tvalue))
cat(res)
#> Absolute difference: 0.0003077685777

# 4-point quadrature
ltmp <- Gquad(f,-1,1,n=4)
res <- sprintf("4-point quadrature\n %14.12f\n",ltmp$itg)
cat(res)
#> 4-point quadrature
#>  0.682679806144
res <- sprintf("Absolute difference: %14.12f\n",abs(ltmp$itg-Tvalue))
cat(res)
#> Absolute difference: 0.000009685993

# 5-point quadrature (default)
ltmp <- Gquad(f,-1,1)
res <- sprintf("5-point quadrature\n %14.12f\n",ltmp$itg)
cat(res)
#> 5-point quadrature
#>  0.682689735388
res <- sprintf("Absolute difference: %14.12f\n",abs(ltmp$itg-Tvalue))
cat(res)
#> Absolute difference: 0.000000243251

# 6-point quadrature
ltmp <- Gquad(f,-1,1,n=6)
res <- sprintf("6-point quadrature\n %14.12f\n",ltmp$itg)
cat(res)
#> 6-point quadrature
#>  0.682689487053
res <- sprintf("Absolute difference: %14.12f\n",abs(ltmp$itg-Tvalue))
cat(res)
#> Absolute difference: 0.000000005084
```

Already the error is of the order of 10^{-7} for $n = 5$ and it goes down to 10^{-9} for $n = 6$.

2.7 Exercise 13

Adapt the code of function `Gquad` to write an algorithm to calculate numerically 2D integrals over rectangular domains. Use the algorithm developed to calculate

$$\iint_R ye^x \, \mathrm{d}x\mathrm{d}y, \qquad R = [0, 2] \times [0, 3].$$

SOLUTION

The code used in `Gquad` can be extracted simply by typing `Gquad` in an R console. The result yields the initial prototype shown in the following snippet.

```
function(f,a,b,n=5) {
  # The lowest quadrature is linear
  if (n < 1) stop("n must be a positive integer.")

  # n=1 must be treated separately
```

```
  if (n == 1) {
    x <- 0
    w <- 2
    xt <- (b-a)/2*x+(b+a)/2
    wt <- (b-a)/2*w

    # Compute weighted sum
    itg <- sum(wt*f(xt))

    # List containing zeros, weights, and integral
    ltmp <- list(xt=xt,wt=wt,itg=itg)

    return(ltmp)
  }

  # Golub-Welsch algorithm to avoid using a Legendre R package
  i <- 1:(n-1)
  a_diag <- rep(0,length.out=n)          # Diagonal entries (Legendre: all 0)
  b_sub <- i/sqrt(4*i^2-1)               # Sub-diagonal entries
  T <- diag(a_diag)                      # Build Jacobi matrix
  T[cbind(i+1,i)] <- b_sub               # Lower diagonal
  T[cbind(i,i+1)] <- b_sub               # Upper diagonal

  eig <- eigen(T,symmetric=TRUE)
  x <- eig$values                        # Nodes in [-1,1]
  V <- eig$vectors
  w <- 2 * (V[1,])^2                     # Weights

  # Transform from [-1,1] to [a,b]
  xt <- (b-a)/2*x+(b+a)/2
  wt <- (b-a)/2*w

  # Compute weighted sum
  itg <- sum(wt*f(xt))

  # List containing zeros, weights, and integral
  ltmp <- list(xt=xt,wt=wt,itg=itg)

  return(ltmp)
}
```

We can see that, being a function for 1D integration, `Gquad` only includes one set of extremes of integration. In the new function, we will need a set for x, `ax,bx` and a set for y, `ay,by`. Also, there is no immediate reason for the order of the quadrature for x to be the same of the order of the quadrature for y. So, we will use a `nx` and a `ny`.

Calculation of the nodes and weights can be done using the same lines as in the above code, making sure to do one set of calculations for `nx` and one set for `ny`. It is here convenient to arrange zeros, weights and sampled values for $f(x, y)$, in a same matrix form so that the type of final expression, `sum(wt*f(xt))`, does not need changing. This can be easily done using the R function `outer`, which is an *outer product of arrays*, and that includes the option to create arrays also from 2D functions, exactly what we need. Furthermore, to avoid useless complications in coding, connected to the special case $n = 1$, we will only allow values of `nx` and `ny` greater or equal than 2. These considerations lead to the new function `G2quad`, shown below.

```r
G2quad <- function(f,ax=-1,bx=1,ay=-1,by=1,nx=5,ny=5) {
  # The lowest quadrature has 2 points
  if (nx < 2) stop("nx must be a positive integer.")
  if (ny < 2) stop("ny must be a positive integer.")

  ## Zeros and weights for x
  i <- 1:(nx-1)
  a_diag <- rep(0,length.out=nx)
  b_sub <- i/sqrt(4*i^2-1)
  T <- diag(a_diag)
  T[cbind(i+1,i)] <- b_sub
  T[cbind(i,i+1)] <- b_sub
  eig <- eigen(T,symmetric=TRUE)
  x <- eig$values
  V <- eig$vectors
  w <- 2*(V[1,])^2
  xt <- (bx-ax)/2*x+(bx+ax)/2
  wxt <- (bx-ax)/2*w

  ## Zeros and weights for y
  i <- 1:(ny-1)
  a_diag <- rep(0,length.out=ny)
  b_sub <- i/sqrt(4*i^2-1)
  T <- diag(a_diag)
  T[cbind(i+1,i)] <- b_sub
  T[cbind(i,i+1)] <- b_sub
  eig <- eigen(T,symmetric=TRUE)
  y <- eig$values
  U <- eig$vectors
  w <- 2*(U[1,])^2
  yt <- (by-ay)/2*y+(by+ay)/2
  wyt <- (by-ay)/2*w

  # Gaussian quadrature
  fM <- outer(xt,yt,f)
  W <- outer(wxt,wyt)
  itg <- sum(W * fM)

  return(itg)
}

print(class(G2quad))
#> [1] "function"
```

Make sure to examine and scrutinise the above code with care. It is important, when writing R code, to make use as much as possible of structural tools like `outer` in order to exploit parallel operations like the multiplication in `W*fM`.

Let us, next, use the function just implemented to calculate the given integral. The integral can also be easily calculated analytically:

$$\iint_R ye^x \, \mathrm{d}x\mathrm{d}y = \int_0^2 e^x \, \mathrm{d}x \int_0^3 y \, \mathrm{d}y = \frac{9}{2}(e^2 - 1) \approx 28.751.$$

The function is relatively easy to use, once the 2D integrand function is defined.

```r
# Define 2D function
f <- function(x,y) {ff <- y*exp(x); return(ff)}

# 5-point quadrature: 2D integration
itg <- G2quad(f,0,2,0,3)
print(itg)
#> [1] 28.75075
```

The result confirms that the code works. The relative complexity of what coded above should give some ideas of the type of algorithms and the variety of code possible to calculate multiple integrals using the Gaussian quadrature.