Numerical Methods in Economics

Emmanuel S. Tsyawo

10/16/2021

Contents

Numerical methods are a key part of computations used in Economics, Mathematics, Statistics among other quantitative fields. Numerical methods to be considered in this section include solving systems of linear & non-linear equations, derivatives, integrals, finding the zeroes of a function, and matrix decompositions.

The following packages are going to be needed: `expm`,`Matrix`,`rootSolve`, `numDeriv`, and `pracma`. Ensure they are installed.

Matrix Operations

Systems of linear equations: matrix solve

The equations can be formulated as $AX = B$ where A is $m \times m$, and B is $m \times 1$. The goal is to obtain a vector X which solves the system of equations.

Example 1

```
options(digits=3) #round results to three decimal places
set.seed(3) #set seed for reproducibility
A = matrix(runif(16), nrow = 4) #generate matrix A
A
```

```
##       [,1]  [,2]  [,3]  [,4]
## [1,] 0.168 0.602 0.578 0.534
## [2,] 0.808 0.604 0.631 0.557
## [3,] 0.385 0.125 0.512 0.868
## [4,] 0.328 0.295 0.505 0.830
```

```
set.seed(3)
B = runif(4)
B
```

```
## [1] 0.168 0.808 0.385 0.328
```

Solution

```
solve(A,B) #solve for the unkowns x in Ax=B
```

```
## [1]  1.00e+00 -3.97e-16  7.82e-16 -3.82e-16
```

```
A%*%solve(A,B) # Should recover B
```

```
##       [,1]
## [1,] 0.168
## [2,] 0.808
## [3,] 0.385
## [4,] 0.328
```

```
B #compare
```

```
## [1] 0.168 0.808 0.385 0.328
```

Example 2 Solve for the vector $X = (x_1, x_2, x_3)$ which solves the following system of equations.

$$
\begin{aligned}
2x_1 + 2x_2 - x_3 &= 2 \\
x_1 - 3x_2 + x_3 &= 0 \\
3x_1 + 4x_2 - x_3 &= 1
\end{aligned}
$$

Solution A first step is to convert the system into matrix notation $AX = B$ and then use the `solve(A,B)` function.

```r
(A=matrix(c(2,1,3,2,-3,4,-1,1,-1),ncol = 3))
```

```
##      [,1] [,2] [,3]
## [1,]    2    2   -1
## [2,]    1   -3    1
## [3,]    3    4   -1
```

```r
(B=c(2,0,1))
```

```
## [1] 2 0 1
```

```r
(X=solve(A,B))# the solution is given by X
```

```
## [1]  0.429 -0.714 -2.571
```

```r
MASS::fractions(X) #display results as fractions using the fractions function in the R package MASS
```

```
## [1]   3/7  -5/7 -18/7
```

Matrix decompositions

These have applicability in econometrics, statistics, etc for matrix operations.

Eigendecomposition

This decomposition has the form $A = VDV^{-1}$ where A is a $m \times m$ square matrix, D is a diagonal matrix with the eigen-values of A and the columns of V of contain the eigen-vectors.

```r
options(digits=3) #set the number of decimal places to display
M = matrix(c(2,-1,0,-1,2,-1,0,-1,2), nrow=3, byrow=TRUE)
eigen(M)
```

```
## eigen() decomposition
## $values
## [1] 3.414 2.000 0.586
##
## $vectors
##        [,1]      [,2]  [,3]
## [1,] -0.500 -7.07e-01 0.500
## [2,]  0.707  1.10e-15 0.707
## [3,] -0.500  7.07e-01 0.500
```

Singular value decomposition (SVD).

The decomposition has the form $A = UDV$, where D is a non-negative diagonal matrix. The SVD is a generalisation of the eigendecomposition to an $m \times n$ matrix.

```r
set.seed(13)
A = matrix(rnorm(30), nrow=6)
svd(A)
```

```
## $d
## [1] 3.603 3.218 2.030 1.488 0.813
##
## $u
##        [,1]    [,2]    [,3]   [,4]   [,5]
## [1,] -0.217 -0.4632  0.4614  0.164  0.675
## [2,] -0.154 -0.5416  0.0168 -0.528 -0.444
```

```
## [3,]  0.538 -0.1533  0.5983 -0.290 -0.124
## [4,]  0.574 -0.5585 -0.5013  0.319  0.070
## [5,]  0.547  0.3937  0.0449 -0.261  0.285
## [6,]  0.104  0.0404  0.4190  0.664 -0.496
##
## $v
##        [,1]    [,2]   [,3]   [,4]  [,5]
## [1,]  0.459 -0.0047  0.712 -0.159 0.507
## [2,] -0.115 -0.5192 -0.028  0.758 0.377
## [3,]  0.279  0.7350 -0.355  0.352 0.363
## [4,]  0.333 -0.4023 -0.604 -0.448 0.402
## [5,] -0.766  0.1684  0.039 -0.275 0.554
```

LU decomposition

The LU decomposition factors a square matrix $A = LU$ into a lower triangular matrix L and an upper triangular matrix U.

```r
require(Matrix)
```

```
## Loading required package: Matrix
```

```r
options(digits=3)
mm = exp(-as.matrix(dist(1:5)))
mm
```

```
##        1      2     3      4      5
## 1 1.0000 0.3679 0.135 0.0498 0.0183
## 2 0.3679 1.0000 0.368 0.1353 0.0498
## 3 0.1353 0.3679 1.000 0.3679 0.1353
## 4 0.0498 0.1353 0.368 1.0000 0.3679
## 5 0.0183 0.0498 0.135 0.3679 1.0000
```

```r
lum = lu(mm) #take the LU decomposition and save as object lum
str(lum) #view its structure
```

```
## Formal class 'denseLU' [package "Matrix"] with 4 slots
##   ..@ x       : num [1:25] 1 0.3679 0.1353 0.0498 0.0183 ...
##   ..@ perm    : int [1:5] 1 2 3 4 5
##   ..@ Dimnames:List of 2
##   .. ..$ : chr [1:5] "1" "2" "3" "4" ...
##   .. ..$ : chr [1:5] "1" "2" "3" "4" ...
##   ..@ Dim     : int [1:2] 5 5
```

```r
elu = expand(lum) #expand to view elements of object lum
elu
```

```
## $L
## 5 x 5 Matrix of class "dtrMatrix" (unitriangular)
##      [,1]   [,2]   [,3]   [,4]   [,5]
## [1,] 1.0000      .      .      .      .
## [2,] 0.3679 1.0000      .      .      .
## [3,] 0.1353 0.3679 1.0000      .      .
## [4,] 0.0498 0.1353 0.3679 1.0000      .
## [5,] 0.0183 0.0498 0.1353 0.3679 1.0000
##
## $U
## 5 x 5 Matrix of class "dtrMatrix"
```

```
##       [,1]   [,2]   [,3]   [,4]   [,5]
## [1,] 1.0000 0.3679 0.1353 0.0498 0.0183
## [2,]      . 0.8647 0.3181 0.1170 0.0430
## [3,]      .      . 0.8647 0.3181 0.1170
## [4,]      .      .      . 0.8647 0.3181
## [5,]      .      .      .      . 0.8647
##
## $P
## 5 x 5 sparse Matrix of class "pMatrix"
##
## [1,] | . . . .
## [2,] . | . . .
## [3,] . . | . .
## [4,] . . . | .
## [5,] . . . . |
```

```r
elu$L%*%elu$U==mm #verify equality
```

```
## 5 x 5 Matrix of class "lgeMatrix"
##      [,1] [,2] [,3] [,4] [,5]
## [1,] TRUE TRUE TRUE TRUE TRUE
## [2,] TRUE TRUE TRUE TRUE TRUE
## [3,] TRUE TRUE TRUE TRUE TRUE
## [4,] TRUE TRUE TRUE TRUE TRUE
## [5,] TRUE TRUE TRUE TRUE TRUE
```

Choleski decomposition

This is a special case of the LU decomposition for real, symmetric, positive-definite matrices.

```r
M = matrix(c(2,-1,0,-1,2,-1,0,-1,2), nrow=3, byrow=TRUE)
M.U=chol(M) #obtains an upper diagonal matrix
t(M.U)%*%M.U #L is the transpose of U
```

```
##      [,1] [,2] [,3]
## [1,]    2   -1    0
## [2,]   -1    2   -1
## [3,]    0   -1    2
```

```r
M #compare for equality
```

```
##      [,1] [,2] [,3]
## [1,]    2   -1    0
## [2,]   -1    2   -1
## [3,]    0   -1    2
```

Matrix Square Root decomposition

The decomposition is $A = A^{1/2}A^{1/2}$ where $A^{1/2}$ is a symmetric matrix. Note that $A^{1/2}$ is neither lower triangular nor upper triangular.

```r
require(expm)
```

```
## Loading required package: expm

##
## Attaching package: 'expm'
```

```
## The following object is masked from 'package:Matrix':
##
##     expm
```

```
m <- diag(2)
sqrtm(m) == m # TRUE
```

```
##      [,1] [,2]
## [1,] TRUE TRUE
## [2,] TRUE TRUE
```

```
ms = 0.5^as.matrix(dist(1:4)) #generate a positive definite matrix
ms
```

```
##       1    2    3     4
## 1 1.000 0.50 0.25 0.125
## 2 0.500 1.00 0.50 0.250
## 3 0.250 0.50 1.00 0.500
## 4 0.125 0.25 0.50 1.000
```

```
ms.5 = sqrtm(ms) #compute the matrix square root
ms.5
```

```
##        [,1]   [,2]   [,3]   [,4]
## [1,] 0.9629 0.2497 0.0946 0.0404
## [2,] 0.2497 0.9326 0.2427 0.0946
## [3,] 0.0946 0.2427 0.9326 0.2497
## [4,] 0.0404 0.0946 0.2497 0.9629
```

```
ms
```

```
##       1    2    3     4
## 1 1.000 0.50 0.25 0.125
## 2 0.500 1.00 0.50 0.250
## 3 0.250 0.50 1.00 0.500
## 4 0.125 0.25 0.50 1.000
```

```
ms.5%*%ms.5 #compare for equality
```

```
##       [,1] [,2] [,3]  [,4]
## [1,] 1.000 0.50 0.25 0.125
## [2,] 0.500 1.00 0.50 0.250
## [3,] 0.250 0.50 1.00 0.500
## [4,] 0.125 0.25 0.50 1.000
```

Systems of Non-linear Equations

Sometimes, a given system of equations may be non-linear in the unknowns. Typical algorithms used for the system of linear equations no longer apply. The `multiroot` function in the `rootSolve` is useful in solving sytems of non-linear equations in R.

Example 1

Consider the following system:

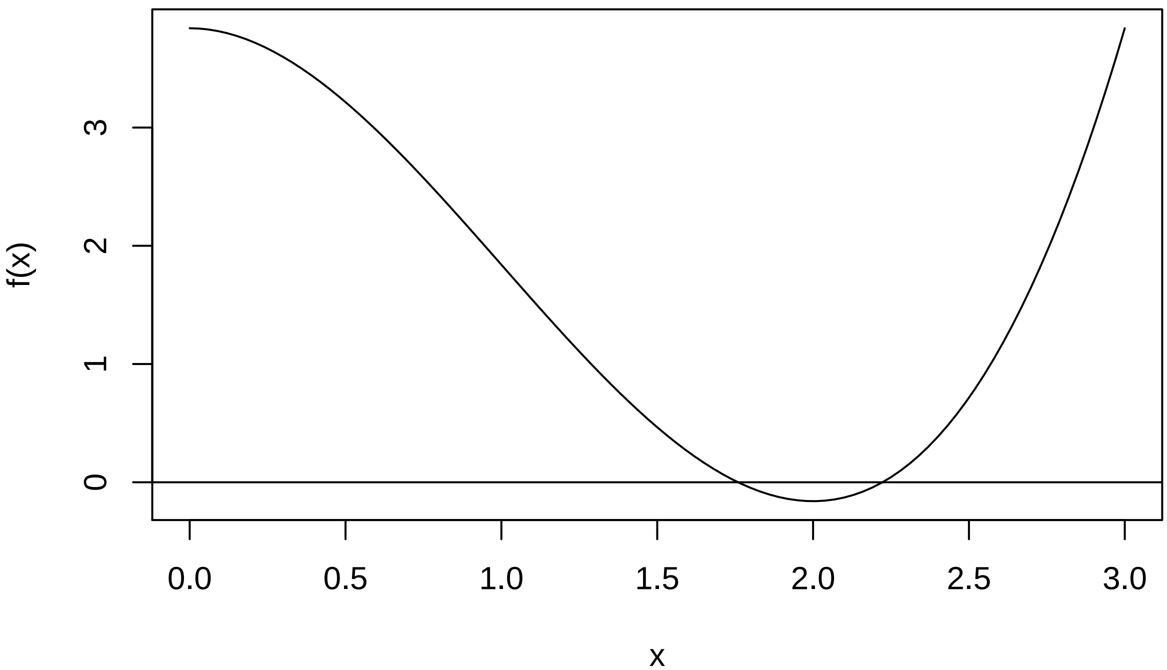
$$\begin{aligned} s^3 - 3s^2 + 4\rho &= 0 \\ \rho - 0.96 &= 0 \end{aligned}$$

Solution A first step is to write a vector-valued function of the system.

```r
rho=0.96 #already given in the second equation
f = function(s){
  s^3 - 3*s^2 +4*rho
}
f(0) #test the function
```

```
## [1] 3.84
```

```r
curve(f,0,3); abline(h=0) #plot the curve
```



Thus we search for roots between 1.5 and 2.5. (See figure in plot)

```r
require(rootSolve)
```

```
## Loading required package: rootSolve
```

```r
options(digits=3)
multiroot(f, c(1.5,2.5))
```

```
## $root
## [1] 1.76 2.22
##
## $f.root
## [1] 1.79e-09 6.45e-07
##
## $iter
## [1] 5
##
## $estim.precis
## [1] 3.24e-07
```

Example 2 Solve for $X = (x_1, x_2)$ in the following system of non-linear equations.

$$10x_1 + 3x_2^2 - 3 = 0$$
$$x_1^2 - \exp(x_2) - 2 = 0$$

Solution First write the vector-valued function, then solve the system.

```
require(rootSolve)
model = function(x) c(10*x[1]+3*x[2]^2-3,x[1]^2 -exp(x[2]) -2)
(ss = multiroot(model,c(1,1)))
```

```
## $root
## [1] -1.45 -2.41
##
## $f.root
## [1]  5.12e-12 -6.08e-14
##
## $iter
## [1] 10
##
## $estim.precis
## [1] 2.59e-12
```

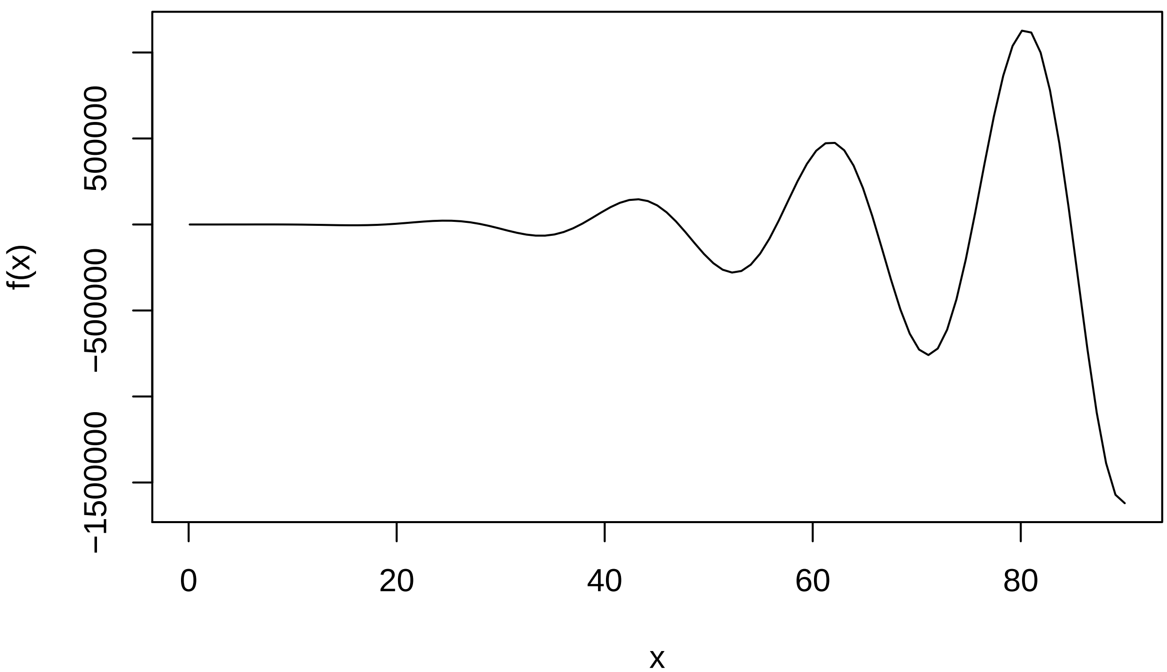
Numerical Differentiation

Numerical differentiation is particularly useful when taking analytical derivatives are difficult or tedious. They are also a good check for analytical derivatives.

Definition-based numerical derivatives

Consider the following function $f(x) = x^3 \sin(x/3) \log(\sqrt{x})$. Let us begin by coding this function.

```
f = function(x) x^3 * sin(x/3) * log(sqrt(x))
curve(f,1/10,90) #plot the function
```



Using the definition of derivatives $f'(x_o) = \lim_{h\to 0} \frac{f(x_o+h)-f(x_o)}{h}$. Choose a small number fixed number, say $\epsilon = 10^{-5}$, then the numerical derivative is as function of function f, x_o, and *epsilon* is

```r
df = function(f,x0,eps) (f(x0+eps)-f(x0))/eps
```

NB: `df()` is a function which takes another function `f()` as input.

Set values and take the derivative.

```r
df(f,x0=1,eps = 1e-5)
```

```
## [1] 0.164
```

With a positive ϵ, this is a **forward derivative**.

```r
df(f,x0=1,eps = -1e-5)
```

```
## [1] 0.164
```

With a negative ϵ, this is a **backward derivative**. Notice that both forward and backward derivatives yield the same value.

A third approach is to average the forward and backward derivatives. This delivers the central difference formula

```r
dfc = function(f,x0,eps) {(f(x0+eps)-f(x0-eps))/(2*eps)}
dfc(f,x0=1,eps =1e-05)
```

```
## [1] 0.164
```

Numerical differentiation using the `numDeriv` package

The `numDeriv` package provides stable and powerful numerical tools for taking numerical derivatives.

Derivatives with univariate functions.

Let us use the `grad()` function to take the same derivative as in the preceding sub-section.

```r
require(numDeriv)
```

```
## Loading required package: numDeriv
```

```
##
## Attaching package: 'numDeriv'
```

```
## The following object is masked from 'package:rootSolve':
##
##     hessian
```

```r
options(digits=16)

# try different methods
grad(f, 1, method = "simple")
```

```
## [1] 0.1636540038633105
```

```r
grad(f, 1, method = "Richardson")
```

```
## [1] 0.1635973483989158
```

```r
grad(f, 1, method = "complex")
```

```
## [1] 0.1635973483980761
```

Gradients

Example 1 Let us consider a real-data example where we compute the gradient of the log-likelihood function of the normal regression model at c(rep(1,4)) from *Session 1.1*. First load the data set if it is not already loaded.

```
dat<- read.csv("dat.csv",header = T,sep = " ")
```

For ease of reference, the log-likelihood function is repeated here.

```
llike_lnorm<- function(pars,Y,X){ #pars is the vector [beta,sigma]
  N = length(Y) #obtain number of observations
  X = as.matrix(cbind(1,X)) # include 1's for the intercept term
  k=ncol(X) # number of parameters to estimate
  np = length(pars) #obtain number of parameters (including sigma)
  if(np!=(k+1)){stop("Not enough parameters.")} #ensure the number of parameters is correct
  beta = matrix(pars[-np],ncol = 1) #obtain column vector of slope parameters beta
  sig2 = pars[np] #sigma^2
  U<- Y - X%*%beta #compute U
  ll = -(N/2)*log(2*pi*sig2) -sum(U^2)/(2*sig2)#obtain log joint likelihood
  return(ll)
}
```

Test the function at the given input value.

```
llike_lnorm(pars = rep(1,5),Y=dat$nonwife,X=dat[c("age","education","experience")])
```

```
## [1] -928647.242200775
```

Compute the gradient at input value. Recall to supply other inputs viz. `Y` and `X`.

```
grad(func = llike_lnorm,rep(1,5),Y=dat$nonwife,X=dat[c("age","education","experience")])
```

```
## [1]   -34883.8903270280 -1543446.0693288015  -426238.4568879842
## [4]  -449243.3520572379   927578.7814838189
```

Example 2 Consider the function $f(X) = 2x_1 + 3x_2^2 - \sin(x_3)$. First code the function.

```
f = function(x){2*x[1] + 3*x[2]^2 - sin(x[3])}
round(grad(f,c(1,1,0)),3) # gradient of f at c(1,1,0)
```

```
## [1]  2  6 -1
```

The Jacobian

Consider a vector valued function $f : \mathbb{R}^m \mapsto \mathbb{R}^n$, $f(x)$. How do we compute the $m \times n$ Jacobian

$$\mathbf{J}_f(x) = \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_1} & \cdots & \frac{\partial f_1(x)}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m(x)}{\partial x_1} & \cdots & \frac{\partial f_m(x)}{\partial x_n} \end{bmatrix}?$$

Example $f(x) = [x_1^2 + 2x_2^2 - 3,\ \cos(\pi x_1/2) - 5x_2^3]$. Compute the Jacobian of the system of equations

```
require(numDeriv)
f = function(x) c(x[1]^2 + 2*x[2]^2 - 3,
                  cos(pi*x[1]/2) -5*x[2]^3)
jacobian(f, c(2,1))
```

```
##                     [,1]                [,2]
```

```
## [1,] 3.999999999961194   3.999999999982517
## [2,] 0.000000000000000 -14.999999999949509
```

The Hessian

The hessian matrix may be thought of as the jacobian of the gradient of the function.

Example 1 Compute the hessian of f at $[1, 1, 0]$.

```
f = function(x){2*x[1] + 3*x[2]^2 - sin(x[3])}
hessian(f,c(1,1,0))
```

```
##                        [,1]                  [,2]                  [,3]
## [1,]  0.000000000000000e+00 -4.101521077503342e-12  0.000000000000000e+00
## [2,] -4.101521077503342e-12  6.000000000000890e+00 -4.081342357941147e-13
## [3,]  0.000000000000000e+00 -4.081342357941147e-13  0.000000000000000e+00
```

Example 2 Back to the likelihood example.

```
options(digits = 5) #set number of digits to display
hessian(func = llike_lnorm,rep(1,5),Y=dat$nonwife,X=dat[c("age","education","experience")])
```

```
##        [,1]     [,2]    [,3]    [,4]     [,5]
## [1,]   -753   -32031   -9252   -8005    34884
## [2,] -32031 -1411535 -391896 -356877  1543446
## [3,]  -9252  -391896 -117588  -99273   426238
## [4,]  -8005  -356877  -99273 -134063   449243
## [5,]  34884  1543446  426238  449243 -1855534
```

Higher-order derivatives

For higher-order derivatives, the `fderiv` function in the `pracma` package is well suited.

```
require(pracma)
```

```
## Loading required package: pracma
```

```
##
## Attaching package: 'pracma'
```

```
## The following objects are masked from 'package:numDeriv':
##
##     grad, hessian, jacobian
```

```
## The following objects are masked from 'package:rootSolve':
##
##     gradient, hessian
```

```
## The following objects are masked from 'package:expm':
##
##     expm, logm, sqrtm
```

```
## The following objects are masked from 'package:Matrix':
##
##     expm, lu, tril, triu
```

```
f = function(x) x^3 * sin(x/3) * log(sqrt(x))
x = 1:4
fderiv(f,x) # 1st derivative at 4 points
```

```
## [1]  0.1636  4.5348 18.9378 43.5914
```

```r
fderiv(f,x,n=2,h=1e-5) # 2nd derivative at 4 points
```

```
## [1]  1.1330  8.6999 20.2076 27.5697
```

Numerical Integration

While some integrals can be difficult to take, others simply do not have analytical expressions. This is where numerical integration comes in.

Finite integrals

Example 1 Consider the function $f(x) = \cos(x)\exp(-x)$ and the integral $q = \int_0^\pi f(x)dx$. Just as in the case of numerical derivatives, we need to write the function first.

```r
f = function(x) exp(-x) * cos(x)
```

Now let us take the integral

```r
( q = integrate(f, 0, pi) )
```

```
## 0.52161 with absolute error < 7.6e-15
```

Example 2 The integrand function needs to be vectorized, otherwise one will get an error message, e.g., with the following nonnegative function:

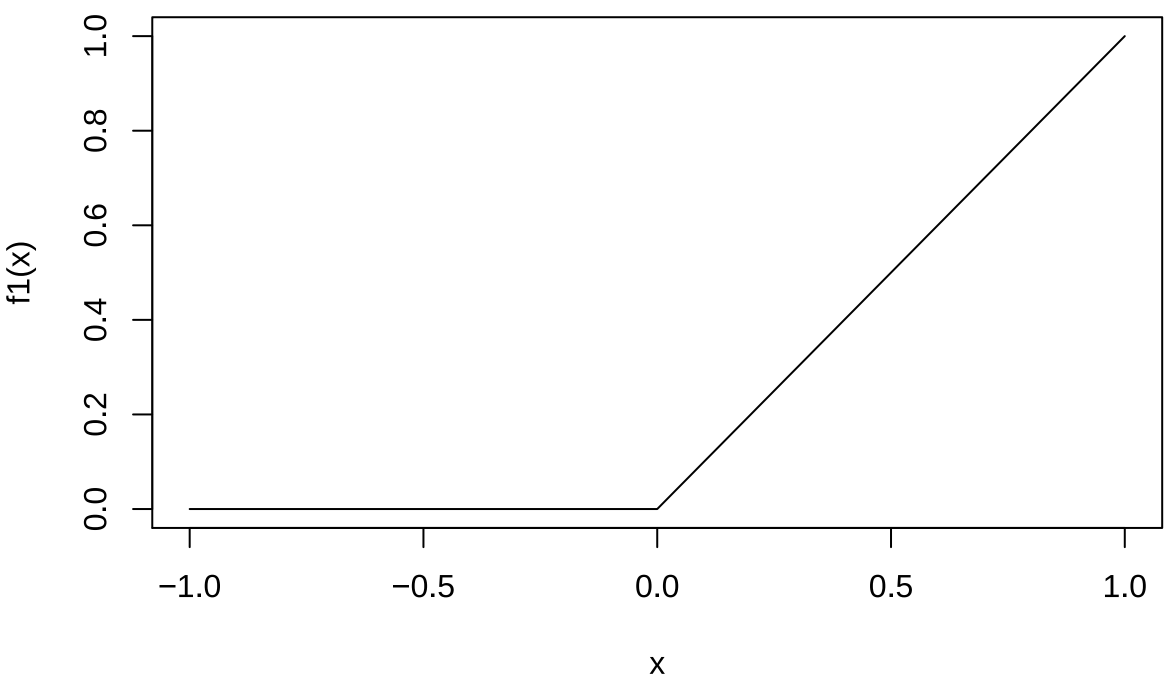
```r
f1 = function(x){ max(0, x)}
#integrate(f1, -1, 1) # why?
```

Now vectorize the function

```r
f1=Vectorize(f1)
integrate(f1, -1, 1)
```

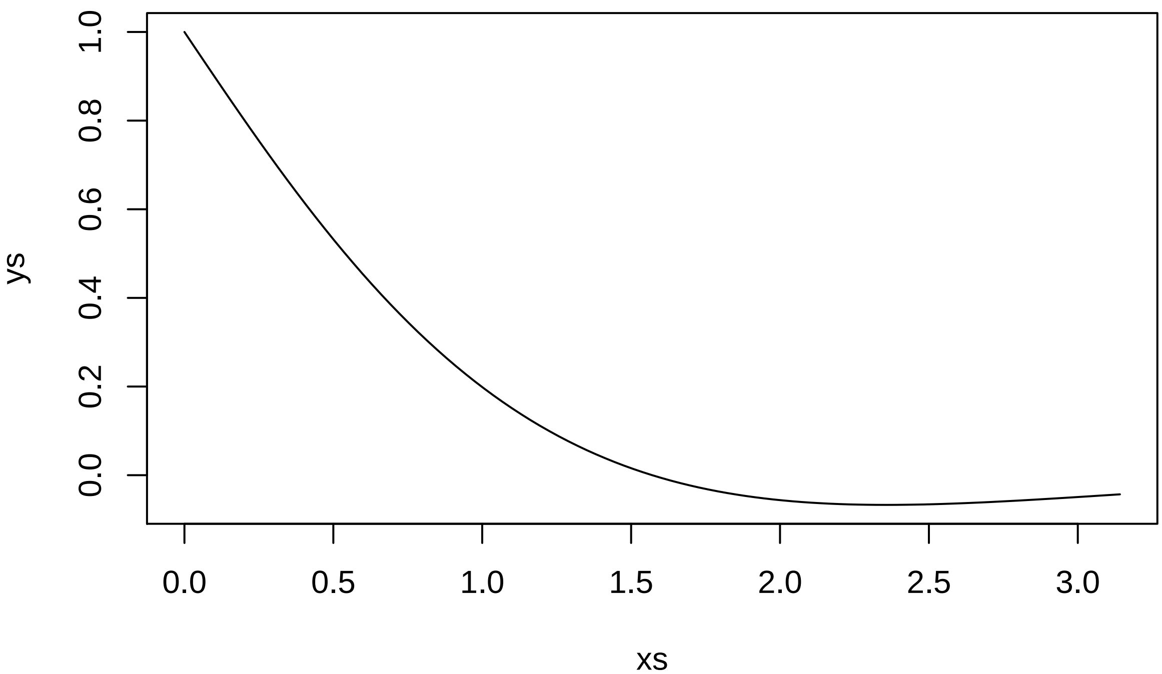
```
## 0.5 with absolute error < 5.6e-15
```

```r
curve(f1,-1,1)
```



Example 3 Discretised functions occur when the function is not explicitly known, but is represented by a number of discrete points. Consider the following example.

```
require(pracma)
f = function(x) exp(-x) * cos(x)
xs = seq(0, pi, length.out = 101) # what does the function seq() do?
ys = f(xs)
plot(xs,ys,type = "l")
```




```
trapz(xs, ys)
```

```
## [1] 0.52169
```

Use the integrate() with the explicit form of the function for comparison.

```
integrate(f,0,pi)
```

```
## 0.52161 with absolute error < 7.6e-15
```

Integration over the entire real line

Example 1 Consider the Gaussian function $f(x) = \exp(-x^2/2)$. We consider the integral $q = \int_{-\infty}^{\infty} f(x)dx$ which is known to equal $\sqrt{2\pi}$. How good is a numerical integral?

```
fgauss = function(t) exp(-t^2/2) # specify a function
( q = integrate(fgauss, -Inf, Inf) )
```

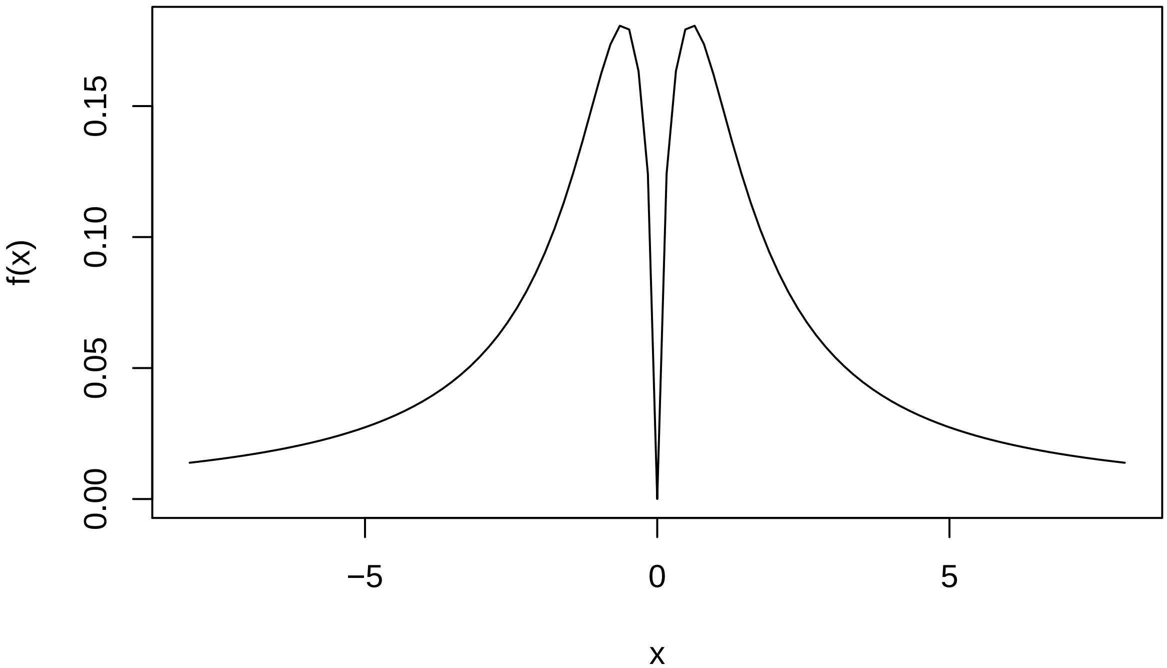
```
## 2.5066 with absolute error < 0.00023
```

```
q$value / sqrt(2*pi) # is our approximation good enough?
```

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

Example 2 Numerical integrals can be useful in computing moments or correlations which might be hard to deal with analytically. Although the first moment of the $Cauchy(0,1)$ does not exist, its fractional moment does exist. Analytically evaluating $\mathbb{E}[|X|^{1/2}] = \int_{-\infty}^{\infty} \frac{|x|^{1/2}}{\pi(1+x^2)}dx$ for $X \sim Cauchy(0,1)$ might seem a difficult task. Let us do so using a numerical integral. First, code the integrand $f(x) = \frac{|x|^{1/2}}{\pi(1+x^2)}$.

```
f=function(x){sqrt(abs(x))/(pi*(1+x^2))}
curve(f,-8,8) #visualise the integrand
```




```
integrate(f,-Inf,Inf)
```

```
## 1.4142 with absolute error < 4.2e-05
```

The numerical integral pretty much coincides with the analytical solution $\mathbb{E}[|X|^{1/2}] = \sqrt{2} \approx 1.414213562373095$.

Monte Carlo and sparse grid integration

Another approach to integration uses Monte Carlo simulation Consider the example $\mathbb{E}[|X|^{1/2}]$ where $X \sim Cauchy(0,1)$. The strong law of large numbers can be used to justify this method.

```
set.seed(21)
X = rcauchy(1e6)
mean(sqrt(abs(X)))
```

```
## [1] 1.4145
```

How good is this approximation?

Exercises