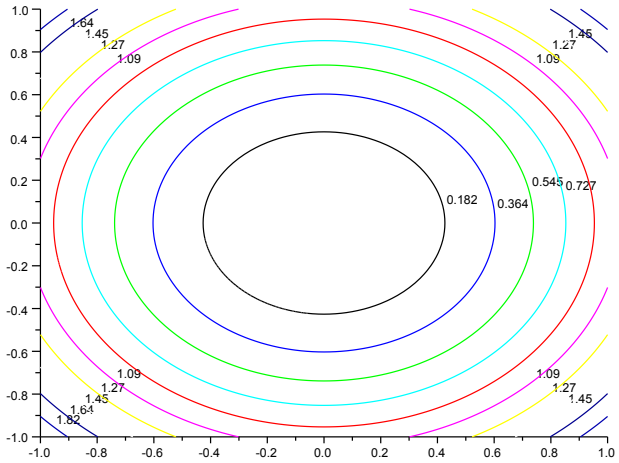

Figure 36: Contour plot of the function $f(x_1, x_2) = x_1^2 + x_2^2$.

In the following Scilab session, we use a simple form of the `contour` function, where the function `myquadratic` is passed as an input argument. The `myquadratic` function takes two input arguments x_1 and x_2 and returns $f(x_1, x_2) = x_1^2 + x_2^2$. The `linspace` function is used to generate vectors of data so that the function is analyzed in the range $[-1, 1]^2$.

```
function f = myquadratic2arg ( x1 , x2 )
  f = x1**2 + x2**2;
endfunction
xdata = linspace ( -1 , 1 , 100 );
ydata = linspace ( -1 , 1 , 100 );
contour ( xdata , ydata , myquadratic2arg , 10)
```

This produces the contour plot presented in figure 36.

In practice, it may happen that our function has the header `z = myfunction ( x )`, where the input variable `x` is a row vector. The problem is that there is only one single input argument, instead of the two arguments required by the `contour` function. There are two possibilities to solve this little problem:

- provide the data to the `contour` function by making two nested loops,
- provide the data to the `contour` function by using `feval`,
- define a new function which calls the first one.

These three solutions are presented in this section. The first goal is to let the reader choose the method which best fits the situation. The second goal is to show that performances issues can be avoided if a consistent use of the functions provided by Scilab is done.

In the following Scilab naive session, we define the quadratic function `myquadratic1arg`, which takes one vector as its single input argument. Then we perform two nested

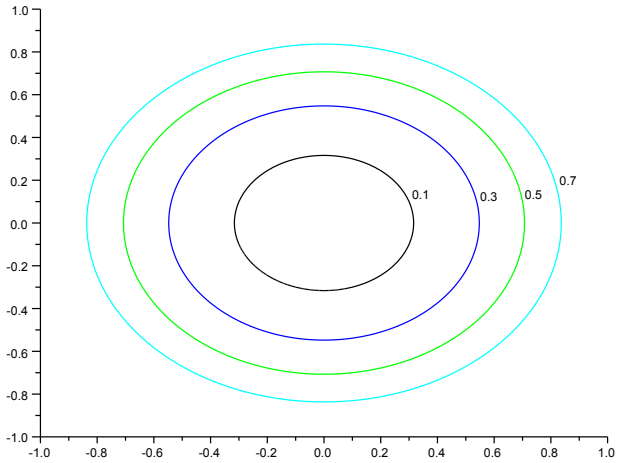

Figure 37: Contour plot of the function $f(x_1, x_2) = x_1^2 + x_2^2$ – The levels are explicitly configured.

loops to compute the `zdata` matrix, which contains the z values. The z values are computed for all the combinations of points $(x(i), y(j)) \in \mathbb{R}^2$, for $i = 1, 2, \ldots, n_x$ and $j = 1, 2, \ldots, n_y$, where n_x and n_y are the number of points in the x and y coordinates. In the end, we call the `contour` function, with the list of required levels (instead of the previous number of levels). This allows to get exactly the levels we want, instead of letting Scilab compute the levels automatically.

```
function f = myquadratic1arg ( x )
  f = x(1)**2 + x(2)**2;
endfunction
xdata = linspace ( -1 , 1 , 100 );
ydata = linspace ( -1 , 1 , 100 );
// Caution ! Two nested loop, this is bad.
for i = 1:length(xdata)
    for j = 1:length(ydata)
      x = [xdata(i) ydata(j)].';
      zdata ( i , j ) = myquadratic1arg ( x );
    end
end
contour ( xdata , ydata , zdata , [0.1 0.3 0.5 0.7])
```

The contour plot is presented in figure 37.

The previous script perfectly works. Still, it is not efficient because it uses two nested loops and this is to avoid in Scilab for performance reasons. Another issue is that we had to store the `zdata` matrix, which might consume a lot of memory space when the number of points is large. This is why this method is to avoid, since it is a bad use of the features provided by Scilab.

In the following script, we use the `feval` function, which evaluates a function on a grid of values and returns the computed data. The generated grid is made of all the combinations of points $(x(i), y(j)) \in \mathbb{R}^2$. We assume here that there

is no possibility to modify the function `myquadratic1arg`, which takes one input argument. Therefore, we create an intermediate function `myquadratic3`, which takes 2 input arguments. Once done, we pass the `myquadratic3` argument to the `feval` function and generate the `zdata` matrix.

```
function f = myquadratic1arg ( x )
  f = x(1)**2 + x(2)**2;
endfunction
function f = myquadratic3 ( x1 , x2 )
  f = myquadratic1arg ( [x1 x2] )
endfunction
xdata = linspace ( -1 , 1 , 100 );
ydata = linspace ( -1 , 1 , 100 );
zdata = feval ( xdata , ydata , myquadratic3 );
contour ( xdata , ydata , zdata , [0.1 0.3 0.5 0.7])
```

The previous script produces, of course, exactly the same plot as previously. This method is also to avoid when possible, since it requires to store the `zdata` matrix, which has size 100×100.

Finally, there is a third way of creating the plot. In the following Scilab session, we use the same intermediate function `myquadratic3` as previously, but we pass it directly to the `contour` function.

```
function f = myquadratic1arg ( x )
  f = x(1)**2 + x(2)**2;
endfunction
function f = myquadratic3 ( x1 , x2 )
  f = myquadratic1arg ( [x1 x2] )
endfunction
xdata = linspace ( -1 , 1 , 100 );
ydata = linspace ( -1 , 1 , 100 );
contour ( xdata , ydata , myquadratic3 , [0.1 0.3 0.5 0.7])
```

The previous script produces, of course, exactly the same plot as previously. The major advantage is that we did not produce the `zdata` matrix.

We have briefly outlined how to produce simple 2D plots. We are now interested in the configuration of the plot, so that the titles, axis and legends corresponds to our data.

7.4 Titles, axes and legends

In this section, we present the Scilab graphics features which allow to configure the title, axes and legends of an x-y plot.

In the following example, we define a quadratic function and plot it with the `plot` function.

```
function f = myquadratic ( x )
  f = x.^2
endfunction
xdata = linspace ( 1 , 10 , 50 );
ydata = myquadratic ( xdata );
plot ( xdata , ydata )
```

We have now the plot which is presented in figure 35.

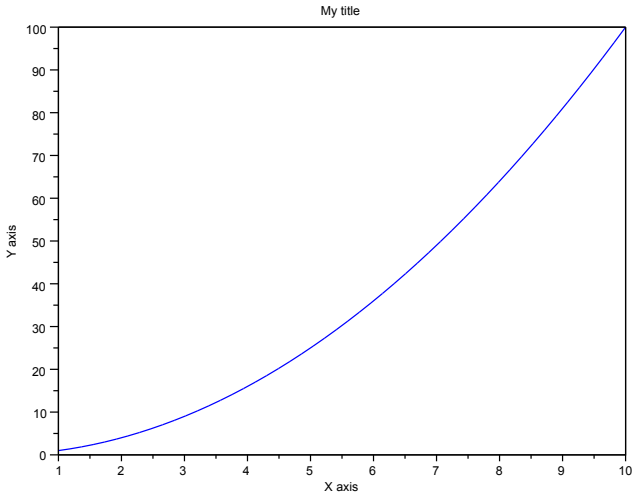

Figure 38: The x-y plot of a quadratic function – This is the same plot as in figure 35, with title and x-y axes configured.

Scilab graphics system is based on *graphics handles*. The graphics handles provide an object-oriented access to the fields of a graphics entity. The graphics layout is decomposed into sub-objects such as the line associated with the curve, the x and y axes, the title, the legends, and so forth. Each object can be in turn decomposed into other objects if required. Each graphics object is associated with a collection of properties, such as the width or color of the line of the curve, for example. These properties can be queried and configured simply by getting or setting their values, like any other Scilab variables. Managing handles is easy and very efficient.

But most basic plot configurations can be done by simple function calls and, in this section, we will focus in these basic features.

In the following script, we use the `title` function in order to configure the title of our plot.

```
title ( "My title" );
```

We may want to configure the axes of our plot as well. For this purpose, we use the `xtitle` function in the following script.

```
xtitle ( "My title" , "X axis" , "Y axis" );
```

Figure 38 presents the produced plot.

It may happen that we want to compare two sets of data in the same 2D plot, that is, one set of x data and two sets of y data. In the following script, we define the two functions $f(x) = x^2$ and $f(x) = 2x^2$ and plot the data on the same x-y plot. We additionally use the "`+-`" and "`o-`" options of the `plot` function, so that we can distinguish the two curves $f(x) = x^2$ and $f(x) = 2x^2$.

```
function f = myquadratic ( x )
  f = x^2
endfunction
function f = myquadratic2 ( x )
```

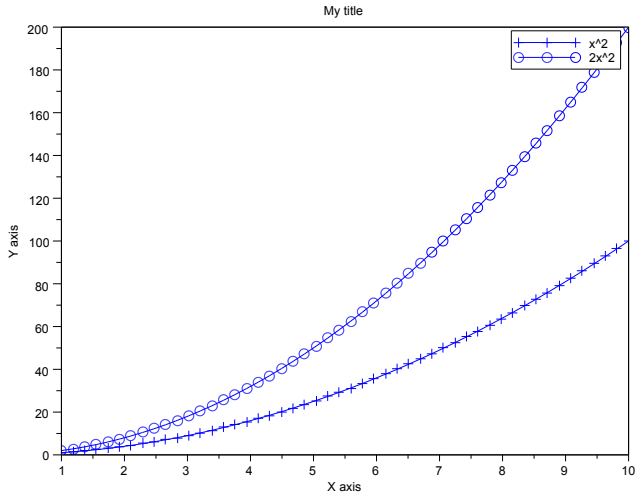

Figure 39: The x-y plot of two quadratic functions – We have configured the legend so that we can distinguish the two functions $f(x) = x^2$ and $f(x) = 2x^2$.

```
  f = 2 * x^2
endfunction
xdata = linspace ( 1 , 10 , 50 );
ydata = myquadratic ( xdata );
plot ( xdata , ydata , "+-" )
ydata2 = myquadratic2 ( xdata );
plot ( xdata , ydata2 , "o-" )
xtitle ( "My title" , "X axis" , "Y axis" );
```

Moreover, we must configure a legend so that we can know what curve is associated with $f(x) = x^2$ and what curve is associated with $f(x) = 2x^2$. For this purpose, we use the `legend` function in order to print the legend associated with each curve.

```
legend ( "x^2" , "2x^2" );
```

Figure 39 presents the produced x-y plot.

We now know how to create a graphics plot and how to configure it. If the plot is sufficiently interesting, it may be worth to put it into a report. To do so, we can export the plot into a file, which is the subject of the next section.

7.5 Export

In this section, we present ways of exporting plots into files, either interactively or automatically with Scilab functions.

Scilab can export any graphics into the vectorial and bitmap formats presented in figure 40. Once a plot is produced, we can export its content into a file, by using interactively the *File > Export to...* menu of the graphics window. We can then set the name of the file and its type.

Vectorial	
`xs2png`	export into PNG
`xs2pdf`	export into PDF
`xs2svg`	export into SVG
`xs2eps`	export into Encapsulated Postscript
`xs2ps`	export into Postscript
`xs2emf`	export into EMF (only for Windows)
Bitmap	
`xs2fig`	export into FIG
`xs2gif`	export into GIF
`xs2jpg`	export into JPG
`xs2bmp`	export into BMP
`xs2ppm`	export into PPM

Figure 40: Export functions.

We can alternatively use the `xs2*` functions, presented in figure 40. All these functions are based on the same calling sequence:

```
xs2png ( window_number , filename )
```

where `window_number` is the number of the graphics window and `filename` is the name of the file to export. For example, the following session exports the plot which is in the graphics window number 0, which is the default graphics window, into the file `foo.png`.

```
xs2png ( 0 , "foo.png" )
```

If we want to produce higher quality documents, the vectorial formats are to be prefered. For example, LaTeX documents may use Scilab plots exported into PDF files to improve their readability, whatever the size of the document.

8 Notes and references

There are a number of topics which have not been presented in this document. We hope that the current document is a good starting point for using Scilab so that learning about these specific topics should not be a problem. We have already mentioned a number of other sources of documentation for this purpose at the begining of this document.

French readers may be interested by [5], where a good introduction is given about how to create and interface to an existing library, how to use Scilab to compute the solution of an Ordinary Differential Equation, how to use Scicos and many other subjects. The same content is presented in English in [3]. English readers should be interested by [4], which gives a deeper overview of Scicos. These books are of great interest, but are rather obsolete since they were written mainly for older version of Scilab.

Further reading may be obtained from the Scilab web cite [6], in the documentation section.

9 Acknowledgments

I would like to thank Claude Gomez, Vincent Couvert, Allan Cornet and Serge Steer who let me share their comments about this document. I am also grateful to Julie Paul and Sylvestre Ledru who helped me during the writing of this document. Thanks are also expressed to Artem Glebov for proofreading this document.

10 Answers to exercises

10.1 Answers for section 1.7

Answer to Exercise 1.1 (*Installing Scilab*) Let us install the current version of Scilab on your system: at the time where this document is written, this is Scilab v5.2. Installing Scilab is very easy, since binaries are provided. Figure 41 present the early steps required to install Scilab v5.1.1 under Windows. □

Answer to Exercise 1.2 (*Inline help:* `derivative`) The `derivative` function allows to compute the numerical derivative of a function. The purpose of this exercise is to find the corresponding help page, by various means. We open the help browser from the console, in the *? > Help Browser* menu. We now select the search pane on the left, and type `derivative`, then press the <Enter> key. All the pages containing the word *derivative* are displayed. The first one is the help page we are searching for. We can also use the help provided on Scilab web site

http://www.scilab.org/product/man

We use the find tool of our favorite web browser and search for the word *derivative*. We successively find the functions: `diff`, `bsplin3val`, `derivative`, `derivat` and `dlgamma`. The searched page is the following:

http://www.scilab.org/product/man/derivative.html

We finally use the console to find the help.

```
help derivative
```

Figure 42 presents the help page for the `derivative`. □

10.2 Answers for section 2.6

Answer to Exercise 2.1 (*The console*) Type the following statement in the console.

```
atoms
```

Now type on the <Tab> key. What happens? We see that all functions which name begins with the letters "atoms" are displayed, as presented in figure 43. Now type the "I" letter, and type again on <Tab>. What happens ? We see that all functions which name begins with the letters "atomsI" are displayed, as presented in figure 44. □

Answer to Exercise 2.2 (*Using exec*) The `SCI` variable contains the name of the directory of the current Scilab installation. The statement `SCI+"/modules"` creates a string which is the concatenation of the Scilab directory name and the "/modules" string, as shown in the following session.

```
-->SCI+"/modules"
 ans  =
 C:/PROGRA~1/SCILAB~1.0-B/modules
```

Therefore, when we perform the `ls(SCI+"/modules")` statement, Scilab displays the list of files in the *modules* subdirectory of Scilab.

```
-->ls(SCI+"/modules")
 ans  =

!xpad                    !
!                        !
!xcos                    !
!                        !
!windows_tools           !
!                        !
[...]
```

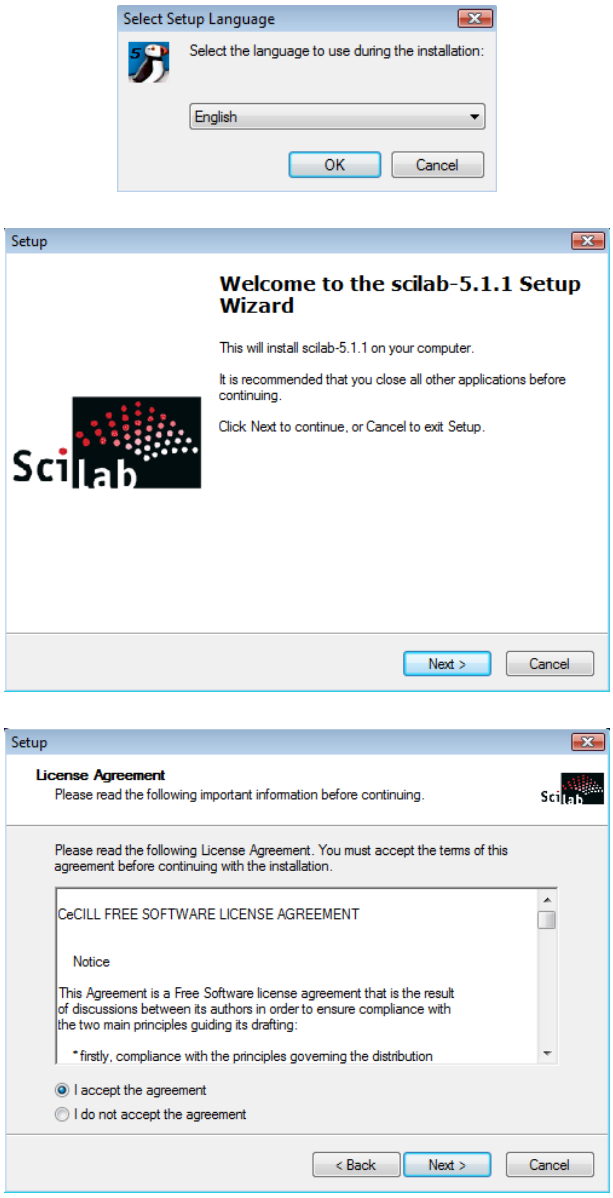

Figure 41: First steps during the installation of Scilab v5.1.1 under Windows.

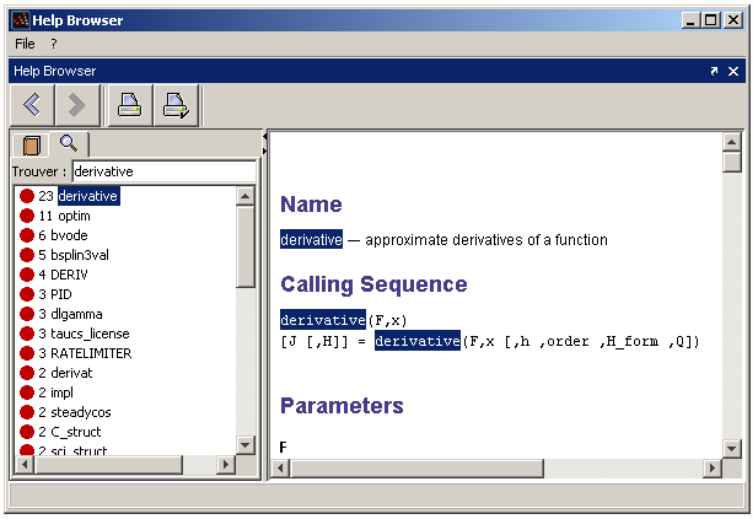

Figure 42: Help page for the `derivative` function.

Figure 43: Using the completion to browse the ATOMS functions.

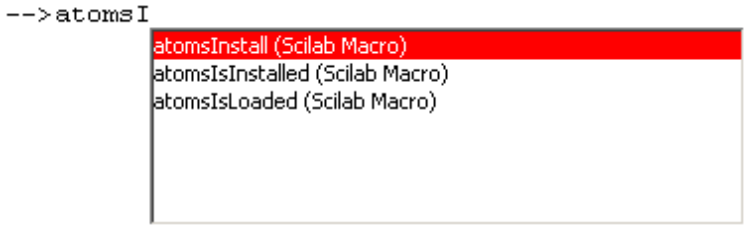

Figure 44: Using the completion to browse the ATOMS functions.

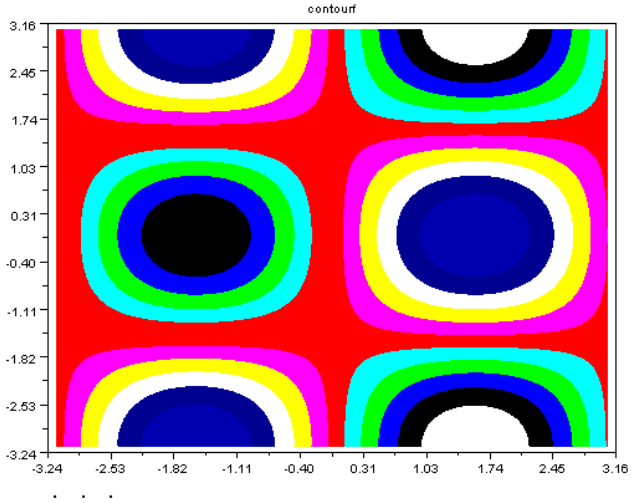

Figure 45: The demo `contourf.dem.sce`.

Executing the demonstration script `contourf.dem.sce` produces the graphic presented in figure 45.

The difference between the two statements

```
exec(SCI+"/modules/graphics/demos/2d_3d_plots/contourf.dem.sce")
exec(SCI+"/modules/graphics/demos/2d_3d_plots/contourf.dem.sce");
```

is that the second ends with a semicolon ";". We use this operator so that Scilab does not display the content of the file when the script is executed, which is convenient when the script contains many lines. □

10.3 Answers for section 3.13

Answer to Exercise 3.1 (*Precedence of operators*) The order of the operations when they are applied to a mathematical expression is called the *precedence*. For example, when the expression $2 \times 3 + 4$, it is equivalent to the expression $(2 \times 3) + 4$. The following session shows that the precedence of Scilab operators is the same as the usual mathematical operators.

```
-->2 * 3 + 4
 ans  =
    10.
-->2 + 3 * 4
 ans  =
    14.
-->2 / 3 + 4
 ans  =
    4.6666667
-->2 + 3 / 4
 ans  =
    2.75
```

□

Answer to Exercise 3.2 (*Parentheses*) When the precedence of the operators does not allow to compute the result we want, parentheses can be used to force the order of the operations. In Scilab, we can use the usual round brackets "(" and ")".

```
-->2 * (3 + 4)
 ans  =
    14.
-->(2 + 3) * 4
 ans  =
    20.
-->(2 + 3) / 4
 ans  =
    1.25
-->3 / (2 + 4)
 ans  =
    0.5
```

□

Answer to Exercise 3.3 (*Exponents*) When we want to define constants with exponents, as the constant $1.23456789 \times 10^{10}$, we use the "`d`" letter to define the exponent, as in the following session.

```
-->1.23456789d10
 ans  =
    1.235D+10
```

We can alternatively use the "`e`" letter, as in the following session which computes the constants $1.23456789 \times 10^{10}$ and $1.23456789 \times 10^{-5}$.

```
-->1.23456789e10
 ans  =
    1.235D+10
-->1.23456789e-5
 ans  =
    0.0000123
```

□

Answer to Exercise 3.4 (*Functions*) The `sqrt` behaves exactly as mathematically expected for positive arguments.

```
-->sqrt(4)
 ans  =
    2.
-->sqrt(9)
 ans  =
    3.
```

For negative arguments x, Scilab returns y as the complex solution of the equation $x^2 = y$.

```
-->sqrt(-1)
 ans  =
    i
-->sqrt(-2)
 ans  =
    1.4142136i
```

The `exp` function is the exponential function, where the base e is Euler's constant. The `log` is the natural logarithm, which is the inverse function of the exponential function.

```
-->exp(1)
 ans  =
    2.7182818
-->log(exp(2))
 ans  =
```

```
    2.
-->exp(log(2))
 ans  =
    2.
```

The `log10` function is the base-10 logarithm. Notice that if x is an integer, then $\log_{10}(x)$ is the number of decimal digits of x.

```
-->10^2
 ans  =
    100.
-->log10(10^2)
 ans  =
    2.
-->10^log10(2)
 ans  =
    2.
```

The `sign` function returns the sign of its argument, and returns zero when x is zero.

```
-->sign(2)
 ans  =
    1.
-->sign(-2)
 ans  =
  - 1.
-->sign(0)
 ans  =
    0.
```

□

Answer to Exercise 3.5 (*Trigonometry*) The following session is a sample use of the `cos` and `sin` functions.

```
-->cos(0)
 ans  =
    1.
-->sin(0)
 ans  =
    0.
```

Because of the limited precision of floating point numbers, the result of a trigonometry function (as any other function) is subject to rounding. In the following session, the mathematical identity $\sin(\pi) = 0$ is approximated at best, given the machine precision associated with double variables.

```
-->cos(%pi)
 ans  =
  - 1.
-->sin(%pi)
 ans  =
    1.225D-16
-->cos(%pi/4) - sin(%pi/4)
 ans  =
    1.110D-16
```

□

10.4 Answers for section 4.18

Answer to Exercise 4.1 (*Plus one*) Let us create the vector $(x_1+1, x_2+1, x_3+1, x_4+1)$ with the following x. The following session performs the computation and uses the usual addition operator "+". In this case, the scalar 1 is added to each element of the vector x.

```
-->x = 1:4;
-->y = x + 1
 y  =
    2.    3.    4.    5.
```

□

Answer to Exercise 4.2 (*Vectorized multiplication*) Let us create the vector $(x_1y_1, x_2y_2, x_3y_3, x_4y_4)$ with the following x and y. The following session performs the computation and uses the elementwise multiplication operation ".*".

```
-->x = 1:4;
-->y = 5:8;
-->z = x .* y
 z  =
    5.    12.    21.    32.
```

□

Answer to Exercise 4.3 (*Vectorized invert*) Let us create the vector $\left(\frac{1}{x_1}, \frac{1}{x_2}, \frac{1}{x_3}, \frac{1}{x_4}\right)$. The following session performs the computation and uses the elementwise division operation "./".

```
-->x = 1:4;
-->y = 1 ./ x
 y  =
    1.    0.5    0.3333333    0.25
```

The following session does not compute what we want, but, instead, compute the solution y of the equation $xy = 1$.

```
-->y = 1 / x // This is not what we want here !
 y  =
    0.0333333
    0.0666667
    0.1
    0.1333333
```

□

Answer to Exercise 4.4 (*Vectorized division*) Let us create the vector $\left(\frac{x_1}{y_1}, \frac{x_2}{y_2}, \frac{x_3}{y_3}, \frac{x_4}{y_4}\right)$. The following session performs the computation and uses the elementwise division operation "./".

```
-->x = 12*(6:9);
-->y = 1:4;
-->z = x ./ y
 z  =
    72.    42.    32.    27.
```

□

Answer to Exercise 4.5 (*Vectorized squaring*) Let us create the vector $(x_1^2, x_2^2, x_3^2, x_4^2)$ with $x = 1, 2, 3, 4$. The following session performs the computation and uses the elementwise squaring operation ".^".

```
-->x = 1:4;
-->y = x.^2
 y  =
    1.    4.    9.    16.
```

□

Answer to Exercise 4.6 (*Vectorized sinus*) Let us create the vector $(sin(x_1), sin(x_2), \ldots, sin(x_{10}))$ with $x \in [0, \pi]$. The following session performs the computation and uses the `linspace` function.

```
-->x = linspace(0,%pi,10);
-->y = sin(x)
 y  =
          column 1 to 6
    0.     0.3420201    0.6427876    0.8660254    0.9848078    0.9848078
          column 7 to 10
    0.8660254    0.6427876    0.3420201    1.225D-16
```

□

Answer to Exercise 4.7 (*Vectorized function*) Let us compute the $y = f(x)$ values of the function f defined by the equation

$$f(x) = \log_{10}(r/10^x + 10^x) \tag{3}$$

with $r = 2.220.10^{-16}$ for $x \in [-16, 0]$. The following session performs the computation and uses the elementwise division "`./`" operator.

```
r = 2.220D-16;
x = linspace (-16,0,10);
y = log10(r./10.^x + 10.^x);
```

This function appears in the computation of the optimal step to be used in a numerical derivative. It shows that the optimal step to use with a forward order one finite difference is equal to `h=sqrt(%eps)`. □

References

[1] Atlas - automatically tuned linear algebra software. http://math-atlas.sourceforge.net.

[2] Cecill and free software. http://www.cecill.info.

[3] C Bunks, J.-P. Chancelier, F. Delebecque, C. Gomez, M. Goursat, R. Nikoukhah, and S. Steer. *Engineering and Scientific Computing With Scilab*. Birkhauser Boston, 1999.

[4] Stephen L. Campbell, Jean-Philippe Chancelier, and Ramine Nikoukhah. *Modeling and Simulation in Scilab/Scicos*. Springer, 2006.

[5] J.-P. Chancelier, F. Delebecque, C. Gomez, M. Goursat, R. Nikoukhah, and S. Steer. *Introduction à Scilab, Deuxième Édition*. Springer, 2007.

[6] The Scilab Consortium. Scilab. http://www.scilab.org.

[7] Intel. Intel math kernel library. http://software.intel.com/en-us/intel-mkl/.

[8] Sylvestre Ledru. Different execution modes of scilab. http://wiki.scilab.org/Different_execution_modes_of_Scilab.

[9] Cleve Moler. Numerical computing with matlab.

[10] Flexdock project. Flexdock project home. https://flexdock.dev.java.net/.

Index