4.4 Computing π, $\ln 2$ and e

The approximations $\pi \approx 3.1415927$, $\ln 2 \approx 0.69314718$, $e \approx 2.7182818$ can be obtained by numerical methods applied to the following initial value problems:

$$(1) \qquad y' = \frac{4}{1+x^2}, \quad y(0) = 0, \quad \pi = y(1),$$

$$(2) \qquad y' = \frac{1}{1+x}, \quad y(0) = 0, \quad \ln 2 = y(1),$$

$$(3) \qquad y' = y, \quad y(0) = 1, \quad e = y(1).$$

Equations (1)–(3) *define* the constants π, $\ln 2$ and e through the corresponding initial value problems.

The third problem (3) requires a numerical method like RK4, while the other two can be solved using Simpson's quadrature rule. It is a fact that RK4 reduces to Simpson's rule for $y' = F(x)$, therefore, for simplicity, RK4 can be used for all three problems, ignoring speed issues. It will be seen that the choice of the DE-solver algorithm (e.g., RK4) affects computational accuracy.

Computing $\pi = \int_0^1 4(1+x^2)^{-1}dx$

The easiest method is Simpson's rule. It can be implemented in virtually every computing environment. The code below works in popular `matlab`-compatible numerical laboratories. It modifies easily to other computing platforms, such as `maple` and `mathematica`. To obtain the answer for $\pi = 3.14159265358979323855$ correct to 12 digits, execute the code on the right in Table 10, below the definition of f.

Table 10. Numerical integration of $\int_0^1 4(1+x^2)^{-1}dx$.
Simpson's rule is applied, using `matlab`-compatible code. About 50 subdivisions are required.

```
function ans = simp(x0,x1,n,f)
h=(x1-x0)/n; ans=0;
for i=1:n;
ans1=f(x0)+4*f(x0+h/2)+f(x0+h);
ans=ans+(h/6)*ans1;
x0=x0+h;
end
```

```
function y = f(x)
y = 4/(1+x*x);

ans=simp(0,1,50,f)
```

It is convenient in some laboratories to display answers with `printf` or `fprintf`, in order to show 12 digits. For example, `scilab` prints 3.1415927 by default, but 3.141592653589800 using `printf`.

The results checked in `maple` give $\pi \approx 3.14159265358979323855$, accurate to 20 digits, regardless of the actual `maple` numerical integration algorithm chosen (three were possible). The checks are invoked by `evalf(X,20)` where `X` is replaced by `int(4/(1+x*x),x=0..1)`.

The results for an approximation to π using numerical solvers for differential equations varied considerably from one algorithm to another, although all were accurate to 5 rounded digits. A summary for `odepack` routines appears in Table 11, obtained from the `scilab` interface. A selection of routines supported by `maple` appear in Table 12. Default settings were used with no special attempt to increase accuracy.

The `Gear` routines refer to those in the 1971 textbook [?]. The Livermore stiff solver `lsode` can be found in reference [?]. The Runge-Kutta routine of order 7-8 called `dverk78` appears in the 1991 reference of Enright [?]. The multistep routines of Adams-Moulton and Adams-Bashforth are described in standard numerical analysis texts, such as [?]. Taylor series methods are described in [?]. The Fehlberg variant of RK4 is given in [?].

Table 11. Differential equation numeric solver results for `odepack` routines, applied to the problem $y' = 4/(1+x^2)$, $y(0) = 0$.

Exact value of π	3.14159265358979323851	20 digits
Runge-Kutta 4	3.1415926535910	10 digits
Adams-Moulton lsode	3.1415932355842	6 digits
Stiff Solver lsode	3.1415931587318	5 digits
Runge-Kutta-Fehlberg 45	3.1416249508084	4 digits

Table 12. Differential equation numeric solver results for some `maple`-supported routines, applied to the problem $y' = 4/(1+x^2)$, $y(0) = 0$.

Exact value of π	3.14159265358979323851	20 digits
Classical RK4	3.141592653589790	15 digits
Gear	3.141592653688446	11 digits
Dverk78	3.141592653607044	11 digits
Taylor Series	3.141592654	10 digits
Runge-Kutta-Fehlberg 45	3.141592674191119	8 digits
Multistep Gear	3.141591703761340	7 digits
Lsode stiff solver	3.141591733742521	6 digits

Computing $\ln 2 = \int_0^1 dx/(1+x)$

Like the problem of computing π, the formula for $\ln 2$ arises from the method of quadrature applied to $y' = 1/(1+x)$, $y(0) = 0$. The solution is $y(x) = \int_0^x dt/(1+t)$. Application of Simpson's rule with 150 points gives $\ln 2 \approx 0.693147180563800$, which agrees with the exact value $\ln 2 = 0.69314718055994530942$ through 12 digits.

More robust numerical integration algorithms produce the exact answer for $\ln 2$, within the limitations of machine representation of numbers.

Differential equation methods, as in the case of computing π, have results accurate to at least 5 digits, as is shown in Tables 13 and 14. Lower order methods such as classical Euler will produce results accurate to three digits or less.

Table 13. Differential equation numeric solver results for `odepack` routines, applied to the problem $y' = 1/(1+x)$, $y(0) = 0$.

Exact value of $\ln 2$	0.69314718055994530942	20 digits
Adams-Moulton lsode	0.69314720834637	7 digits
Stiff Solver lsode	0.69314702723982	6 digits
Runge-Kutta 4	0.69314718056011	11 digits
Runge-Kutta-Fehlberg 45	0.69314973055488	5 digits

Table 14. Differential equation numeric solver results for `maple`-supported routines, applied to the problem $y' = 1/(1+x)$, $y(0) = 0$.

Exact value of $\ln 2$	0.69314718055994530942	20 digits
Classical Euler	0.6943987430550621	2 digits
Classical Heun	0.6931487430550620	5 digits
Classical RK4	0.6931471805611659	11 digits
Gear	0.6931471805646605	11 digits
Gear Poly-extr	0.6931471805664855	11 digits
Dverk78	0.6931471805696615	11 digits
Adams-Bashforth	0.6931471793736268	8 digits
Adams-Bashforth-Moulton	0.6931471806484283	10 digits
Taylor Series	0.6931471806	10 digits
Runge-Kutta-Fehlberg 45	0.6931481489496502	5 digits
Lsode stiff solver	0.6931470754312113	7 digits
Rosenbrock stiff solver	0.6931473787603164	6 digits

Computing e from $y' = y$, $y(0) = 1$

The initial attack on the problem uses classical RK4 with $f(x, y) = y$. After 300 steps, classical RK4 finds the correct answer for e to 12 digits: $e \approx 2.71828182846$. In Table 15, the details appear of how to accomplish the calculation using `matlab`-compatible code. Corresponding `maple` code appears in Table 16 and in Table 17. Additional code for `octave` and `scilab` appear in Tables 18 and 19.

Table 15. Numerical solution of $y' = y$, $y(0) = 1$.
Classical RK4 with 300 subdivisions using `matlab`-compatible code.

```
function [x,y]=rk4(x0,y0,x1,n,f)
x=x0;y=y0;h=(x1-x0)/n;
for i=1:n;
 k1=h*f(x,y);
 k2=h*f(x+h/2,y+k1/2);
 k3=h*f(x+h/2,y+k2/2);
 k4=h*f(x+h,y+k3);
 y=y+(k1+2*k2+2*k3+k4)/6;
 x=x+h;
end
```

```
function yp = ff(x,y)
 yp= y;


[x,y]=rk4(0,1,1,300,ff)
```

Table 16. Numerical solution of $y' = y$, $y(0) = 1$ by `maple` internal classical RK4 code.

```
de:=diff(y(x),x)=y(x):
ic:=y(0)=1:
Y:=dsolve({de,ic},y(x),
          type=numeric,method=classical[rk4]):
Y(1);
```

Table 17. Numerical solution of $y' = y$, $y(0) = 1$ by classical RK4 with 300 subdivisions using `maple`-compatible code.

```
rk4 := proc(x0,y0,x1,n,f)
local x,y,k1,k2,k3,k4,h,i:
x=x0:  y=y0:  h=(x1-x0)/n:
for i from 1 to n do
 k1:=h*f(x,y):k2:=h*f(x+h/2,y+k1/2):
 k3:=h*f(x+h/2,y+k2/2):k4:=h*f(x+h,y+k3):
 y:=evalf(y+(k1+2*k2+2*k3+k4)/6,Digits+4):
 x:=x+h:
od:
RETURN(y):
end:

f:=(x,y)->y;
rk4(0,1,1,300,f);
```

A `matlab` *m*-file `"rk4.m"` is loaded into `scilab`-4.0 by `getf("rk4.m")`. Most `scilab` code is loaded by using default file extension `.sci`, e.g., `rk4scilab.sci` is a `scilab` file name. This code must obey `scilab` rules. An example appears below in Table 18.

Table 18. Numerical solution of $y' = y$, $y(0) = 1$ by classical RK4 with 300 subdivisions, using `scilab`-4.0 code.

```
function
[x,y]=rk4sci(x0,y0,x1,n,f)
x=x0,y=y0,h=(x1-x0)/n
 for i=1:n
 k1=h*f(x,y)
 k2=h*f(x+h/2,y+k1/2)
 k3=h*f(x+h/2,y+k2/2)
 k4=h*f(x+h,y+k3)
 y=y+(k1+2*k2+2*k3+k4)/6
 x=x+h
 end
endfunction
```

```
function yp = ff(x,y)
 yp= y
endfunction

[x,y]=rk4sci(0,1,1,300,ff)
```

The popularity of `octave` as a free alternative to `matlab` has kept it alive for a number of years. Writing code for `octave` is similar to `matlab` and `scilab`, however readers are advised to look at sample code supplied with `octave` before trying complicated projects. In Table 19 can be seen some essential agreements and differences between the languages. Versions of `scilab` after 4.0 have a `matlab` to `scilab` code translator.

Table 19. Numerical solution of $y' = y$, $y(0) = 1$ by classical RK4 with 300 subdivisions using `octave`-2.1.

```
function
[x,y]=rk4oct(x0,y0,x1,n,f)
x=x0;y=y0;h=(x1-x0)/n;
 for i=1:n
 k1=h*feval(f,x,y);
 k2=h*feval(f,x+h/2,y+k1/2);
 k3=h*feval(f,x+h/2,y+k2/2);
 k4=h*feval(f,x+h,y+k3);
 y=y+(k1+2*k2+2*k3+k4)/6;
 x=x+h;
 endfor
endfunction
```

```
function yp = ff(x,y)
yp= y;
end

[x,y]=rk4oct(0,1,1,300,'ff')
```

Exercises 4.4

Computing π. Compute $\pi = y(1)$ from the initial value problem $y' = 4/(1 + x^2)$, $y(0) = 0$, using the given method.

1. Use the Rectangular integration rule. Determine the number of steps for 5-digit precision.

2. Use the Rectangular integration rule. Determine the number of steps for 8-digit precision.

3. Use the Trapezoidal integration rule. Determine the number of steps for 5-digit precision.

4. Use the Trapezoidal integration

rule. Determine the number of steps for 8-digit precision.

5. Use classical RK4. Determine the number of steps for 5-digit precision.

6. Use classical RK4. Determine the number of steps for 10-digit precision.

7. Use computer algebra system assist for RK4. Report the number of digits of precision using system defaults.

8. Use numerical workbench assist for RK4. Report the number of digits of precision using system defaults.

Computing $\ln(2)$. Compute $\ln(2) = y(1)$ from the initial value problem $y' = 1/(1+x)$, $y(0) = 0$, using the given method.

9. Use the Rectangular integration rule. Determine the number of steps for 5-digit precision.

10. Use the Rectangular integration rule. Determine the number of steps for 8-digit precision.

11. Use the Trapezoidal integration rule. Determine the number of steps for 5-digit precision.

12. Use the Trapezoidal integration rule. Determine the number of steps for 8-digit precision.

13. Use classical RK4. Determine the number of steps for 5-digit precision.

14. Use classical RK4. Determine the number of steps for 10-digit precision.

15. Use computer algebra system assist for RK4. Report the number of digits of precision using system defaults.

16. Use numerical workbench assist for RK4. Report the number of digits of precision using system defaults.

Computing e. Compute $e = y(1)$ from the initial value problem $y' = y$, $y(0) = 1$, using the given computer assist. Report the number of digits of precision using system defaults.

17. Improved Euler method, also known as Heun's method.

18. RK4 method.

19. RKF45 method.

20. Adams-Moulton method.

Stiff Differential Equation. The flame propagation equation $y' = y^2(1-y)$ is known to be **stiff** for initial conditions $y(0) = y_0$ with $y_0 > 0$ and small. Use classical RK4 and then a stiff solver to compute and plot the solution $y(t)$ in each case. Expect 3000 steps with RK4 versus 100 with a stiff solver.

The exact solution of this equation can be expressed in terms of the **Lambert function** $w(u)$, defined by $u = w(x)$ if and only if $ue^u = x$. For example, $y(0) = 0.01$ gives

$$y(t) = \frac{1}{w\left(99e^{99-t}\right) + 1}.$$

See R.M. Corless, G.H. Gonnet, D.E.G. Hare, D.J. Jeffrey, and D.E. Knuth. "On The Lambert W Function," Advances in Computational Mathematics 5 (1996): 329-359.

21. $y(0) = 0.01$

22. $y(0) = 0.005$

23. $y(0) = 0.001$

24. $y(0) = 0.0001$