SIAM Review
Vol. 41, No. 3, pp. 577–593

© 1999 Society for Industrial and Applied Mathematics

Derivation of Numerical Methods Using Computer Algebra*

Walter Gander†
Dominik Gruntz‡

Abstract. The use of computer algebra systems in a course on scientific computation is demonstrated. Various examples, such as the derivation of Newton's iteration formula, the secant method, Newton–Cotes and Gaussian integration formulas, as well as Runge–Kutta formulas, are presented. For the derivations, the computer algebra system Maple is used.

Key words. numerical methods, computer algebra, Maple, quadrature formulas, nonlinear equations, finite elements, Rayleigh–Ritz, Galerkin method, Runge–Kutta method

AMS subject classifications. 65-01, 68Q40, 65D32, 65H05, 65L60, 65L06

PII. S003614459935093X

1. Introduction.

At ETH Zürich we have redesigned our courses on numerical analysis. We not only teach numerical algorithms but also introduce the students to computer algebra and make heavy use of computer algebra systems in both lectures and assignments. Computer algebra is used to generate numerical algorithms, to compute discretization errors, to derive convergence rates, to simplify proofs, to run examples, and to generate plots.

We claim that it is easier for students to follow a derivation carried out with the help of a computer algebra system than one done by hand. Computer algebra systems take over the hard manual work, such as solving systems of equations. Students need not be concerned with all the details (and all the small glitches) of a manual derivation and can understand and keep an overview of the general steps of the derivation. A computer-supported derivation is also more convincing than a presentation of the bare results without any reasoning.

Moreover, using computer algebra systems, rather complex numerical formulas can be derived—far more complex than what can be done in class by hand. For example, all useful Newton–Cotes rules can be computed without difficulty, in contrast to hand derivations, which usually end with Simpson's rule.

We will demonstrate these claims with examples taken from our introductory courses in scientific computing. We start with formulas for solving nonlinear equations (Newton's formula, the secant method, and a new formula based on inverse

*Received by the editors July 7, 1998; accepted for publication (in revised form) January 21, 1999; published electronically July 27, 1999.
http://www.siam.org/journals/sirev/41-3/35093.html

†Institute of Scientific Computing, ETH-Zentrum, CH-8092 Zürich, Switzerland (gander@inf.ethz.ch).
‡Fachhochschule Aargau, Klosterzelgstrasse, CH-5210 Windisch, Switzerland (gruntz@fh-aargau.ch).

interpolation) and show, in section 5, how Newton–Cotes rules can be derived. In the second part of the paper we show how computer algebra can be used to support more complex derivations such as Gauss quadrature rules (section 7), Runge–Kutta formulas (section 8), and the methods of Ritz and Galerkin (section 9).

In our examples we use Maple V Release 5, but the computations could also be reproduced with, e.g., Mathematica, MuPad, or any other computer algebra system.

2. Newton's Formula. One of the first formulas that students learn is Newton's iteration for solving a nonlinear equation $f(x) = 0$. Given an approximation x_k for the root s, a better approximation, $x_{k+1} = F(x_k)$, can be obtained using the iteration function

```
> F := x -> x - f(x) / D(f)(x);
```

$$F := x \to x - \frac{f(x)}{\mathrm{D}(f)(x)}. \tag{2.1}$$

This iteration converges quadratically to a *simple* root s of $f(x)$. This can be proven by computing the first derivative of $F(x)$ at $x = s$, which is zero.

```
> dF := D(F)(x);
```

$$dF := \frac{f(x)\,\mathrm{D}^{(2)}(f)(x)}{\mathrm{D}(f)(x)^2}, \tag{2.2}$$

```
> subs(f(s) = 0, D(F)(s));
```

$$0,$$

```
> subs(f(s) = 0, D(D(F))(s));
```

$$\frac{\mathrm{D}^{(2)}(f)(s)}{\mathrm{D}(f)(s)}.$$

Arbitrary precision floating point arithmetic, which is provided by most computer algebra systems, can be used to demonstrate what quadratic convergence means "in real life." As an example, we compute the square root of 9 by using Newton's iteration to solve equation $x^2 - 9 = 0$ starting with $x_0 = 1$. As expected, the number of correct digits doubles with each iteration.

```
> f := x -> x^2 - 9:
> Digits := 70:
> xk := 1.0: to 8 do xk := F(xk); lprint(xk); od:

5.0000000000000000000000000000000000000000000000000000000000000000000000
3.4000000000000000000000000000000000000000000000000000000000000000000000
3.0235294117647058823529411764705882352941176470588235294117647058823529
3.0000915541313801785305561913481345845731288624399176012817578393224999
3.0000000013969838622484784252588819771210853888469481349100703687663350
3.0000000000000000003252606517456513302198682555233168260482212782050130
3.0000000000000000000000000000000000000017632415262334312619531048058334
3.0000000000000000000000000000000000000000000000000000000000000000000000.
```

The convergence result is valid only for simple roots, because the first derivative of f appears in the denominator of (2.2); i.e., the result is valid only if $f'(s) \neq 0$. The behavior of Newton's iteration for an equation with a multiple root is the next topic we discuss. Let us assume that $f(x)$ has a zero of multiplicity n at $x = s$. We therefore define $f(x)$ to be

```
> f := x -> (x-s)^n * g(x);
```

$$f := x \to (x - s)^n\, g(x),$$

where $g(s) \neq 0$. Again we inspect the first derivative of $F(x)$. If $F'(s) \neq 0$, then the iteration converges only linearly.

```
> dF;
```

$$(x-s)^n\, g(x)\left(\frac{(x-s)^n\, n^2\, g(x)}{(x-s)^2} - \frac{(x-s)^n\, n\, g(x)}{(x-s)^2} + 2\,\frac{(x-s)^n\, n\, \mathrm{D}(g)(x)}{x-s} + (x-s)^n\,(\mathrm{D}^{(2)})(g)(x)\right) \Big/ \left(\frac{(x-s)^n\, n\, g(x)}{x-s} + (x-s)^n\,\mathrm{D}(g)(x)\right)^2.$$

Taking the limit of the above expression for $x \to s$ we obtain

```
> limit(%, x=s);
```

$$\frac{n-1}{n}.$$

We have just proven that Newton's iteration converges linearly with factor $(n-1)/n$ if $f(x)$ has a zero of multiplicity n. Thus, e.g., convergence is linear with factor $1/2$ for a double root.

Newton's iteration also has a nice geometrical interpretation. Starting with the approximation x_k, the next value x_{k+1} of the iteration is the intersection of the tangent to $f(x)$ at $[x_k, f(x_k)]$ with the x-axis. This property can also be proven with Maple. We set up an equation $p(x) = a\,x + b$ for the tangent line. $p(x)$ must interpolate $[x_k, f(x_k)]$ and must have the same derivative as $f(x)$ at $x = x_k$. With these two conditions the parameters a and b of the tangent $p(x)$ are defined.

```
> f := 'f':
> p := x -> a*x + b:
> solve({p(x[k]) = f(x[k]), D(p)(x[k]) = D(f)(x[k])}, {a, b});
```

$$\{b = -\mathrm{D}(f)(x_k)\, x_k + f(x_k),\ a = \mathrm{D}(f)(x_k)\}.$$

We have claimed that the intersection of the tangent $p(x)$ with the x-axis is the next Newton approximation. If the equation $p(x) = 0$ is solved, the iteration function (2.1) is obtained, proving that the geometrical interpretation is correct.

```
> assign(%);
> x[k+1] = expand(solve(p(t) = 0, t));
```

$$x_{k+1} = x_k - \frac{f(x_k)}{\mathrm{D}(f)(x_k)}.$$

3. Secant Method. Another method we present to our students is the secant method. Compared to Newton's iteration, the secant method has the advantage that no derivatives are needed. The derivative that appears in Newton's formula is approximated by a finite difference. The interesting aspect we demonstrate with the help of Maple is the convergence rate of this method. The iteration function is given in (3.1).

```
> F := (u, v) -> u - f(u) * (u-v) /  (f(u) - f(v));
```

$$F := (u, v) \to u - \frac{f(u)\,(u-v)}{f(u) - f(v)},$$

```
> x[k+1] = F(x[k], x[k-1]);
```

$$x_{k+1} = x_k - \frac{f(x_k)\,(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})}. \tag{3.1}$$

Using (3.1) we obtain for the error $e_{k+1} = x_{k+1} - s$ the recurrence

$$e_{k+1} = F(s + e_k, s + e_{k-1}) - s.$$

The right-hand side can be expanded into a multivariate Taylor series at $e_k = 0$ and $e_{k+1} = 0$. We assume that s is a simple root ($f'(s) \neq 0$) and also that $f''(s) \neq 0$ holds. We set $f(s) = 0$ and compute the first term of the Taylor series expansion:

```
> f(s) := 0:
> e2 = normal(readlib(mtaylor)(F(s + e1, s + e0) - s, [e0, e{1], 4));
```

$$e2 = \frac{1}{2}\,\frac{e0\,(\mathrm{D}^{(2)})(f)(s)\,e1}{\mathrm{D}(f)(s)}.$$

If we divide this leading coefficient by e_0 and e_1, we see that the limit of the quotient $e_2/(e_0\,e_1)$ is a constant different from zero. We assume that the convergence coefficient is p, substitute $e_2 = K\,{e_1}^p$ and $e_1 = K\,{e_0}^p$, and divide by the constant K^p.

```
> %/e1/e0;
```

$$\frac{e2}{e1\,e0} = \frac{1}{2}\,\frac{(\mathrm{D}^{(2)})(f)(s)}{\mathrm{D}(f)(s)},$$

```
> simplify(subs(e2 = K*e1^p, e1 = K*e0^p, %/K^p), assume=positive);
```

$$e0^{(p^2-p-1)} = \frac{1}{2}\,\frac{K^{-p}\,(\mathrm{D}^{(2)})(f)(s)}{\mathrm{D}(f)(s)}.$$

This equation is valid for all errors e_0. Since the right-hand side is constant, the left-hand side also must be independent of e_0. This is the case only if the exponent of e_0 is zero. This condition is an equation for p, whose solution gives the well-known convergence factor $p = (1 + \sqrt{5})/2$ for the secant method.

```
> solve(ln(lhs(%)), p);
```

$$\frac{1}{2}\,\sqrt{5} + \frac{1}{2},\ \frac{1}{2} - \frac{1}{2}\,\sqrt{5}.$$

4. A New Iteration Formula. Having considered the Newton and the secant methods to compute roots of a nonlinear equation, we now want to show how a new iteration method can be derived and analyzed. The new method is a combination of the Newton and the secant methods. It uses function values and first derivatives at two points. These four data define a degree-3 (Hermite) interpolation polynomial. A zero of this polynomial can be taken as a next approximation of the root. Unfortunately, the explicit expression for this zero is rather complex, so we propose to use *inverse* interpolation for the given data. We then need only to evaluate the resulting polynomial at $y = 0$ to obtain the new approximation for the root. In Maple, this is done with the following commands:

```
> p := x -> a*x^3 + b*x^2 + c*x + d:
> solve({p(-f(x[0]))=x[0], D(p)(-f(x[0]))=-1/D(f)(x[0]),
>        p(-f(x[1]))=x[1], D(p)(-f(x[1]))=-1/D(f)(x[1])},
>    {a,b,c,d}):
> assign(%);
> p(0);
```

$$\Big(-f(x_1)^3\,\mathrm{D}(f)(x_1)\,f(x_0) + f(x_1)^3\,x_0\,\mathrm{D}(f)(x_0)\,\mathrm{D}(f)(x_1) + f(x_1)^2\,\mathrm{D}(f)(x_1)\,f(x_0)^2 - f(x_0)^3\,x_1\,\mathrm{D}(f)(x_0)\,\mathrm{D}(f)(x_1) - f(x_1)^2\,\mathrm{D}(f)(x_0)\,f(x_0)^2 - 3\,f(x_1)^2\,x_0\,\mathrm{D}(f)(x_0)\,\mathrm{D}(f)(x_1)\,f(x_0) + f(x_1)\,f(x_0)^3\,\mathrm{D}(f)(x_0) + 3\,f(x_1)\,x_1\,\mathrm{D}(f)(x_0)\,\mathrm{D}(f)(x_1)\,f(x_0)^2\Big) \Big/ \Big(\mathrm{D}(f)(x_0)\,\mathrm{D}(f)(x_1)\,(f(x_1)-f(x_0))\,(f(x_1)^2 - 2\,f(x_1)\,f(x_0) + f(x_0)^2)\Big).$$

The resulting expression still is not very simple. However, if the evaluation of f and f' is very expensive, it still may pay off since the convergence rate is 2.73, as we will see.

For the convergence analysis, we expand

$$e_{k+1} = F(s + e_k, s + e_{k-1}) - s$$

into a multivariate Taylor series at $e_k = 0$ and $e_{k+1} = 0$, as we have done for the secant method.

```
> F := unapply(%, x[0], x[1]):
> f(s) := 0:
> e2 = readlib(mtaylor)(F(s+e0, s+e1)-s, [e0,e1], 8):
> eq := normal(%);
```

$$eq := e2 = \frac{1}{24}\,e1^2\,e0^2\Big(\mathrm{D}(f)(s)^2\,(\mathrm{D}^{(4)})(f)(s) + 15\,(\mathrm{D}^{(2)})(f)(s)^3 - 10\,(\mathrm{D}^{(2)})(f)(s)\,(\mathrm{D}^{(3)})(f)(s)\,\mathrm{D}(f)(s)\Big) \Big/ \mathrm{D}(f)(s)^3.$$

As before with the secant and Newton methods we consider only simple roots; i.e., we assume that $D(f)(s) \neq 0$. If this condition holds, then the above equation tells us that in the limit

```
> e2/e0^2/e1^2 = const;
```

$$\frac{e2}{e0^2\,e1^2} = \text{const}.$$

Let us again introduce the convergence coefficient p and make the following substitutions:

```
> subs(e2=K*e1^p, e1=K*e0^p, %);
```

$$\frac{(K\,e0^p)^p}{K\,e0^2\,(e0^p)^2} = \text{const},$$

```
> simplify(%, assume=positive);
```

$$K^{(-1+p)}\,e0^{(p^2-2-2\,p)} = \text{const}.$$

This equation must hold for all errors e_0. Since K, p, and const are all constant, the exponent of e_0 must be zero.

```
> solve(p^2 - 2*p - 2 = 0, p);
```

$$1+\sqrt{3},\ 1-\sqrt{3},$$

```
> evalf([%]);
```

$$[2.732050808,\ -.732050808].$$

Thus, the convergence factor is $p = 1 + \sqrt{3}$ and we have superquadratic convergence.

Let us use Maple to demonstrate the above convergence rate with an example. We use our algorithm to compute the zero of the function $f(x) = e^x + x$ starting with $x_0 = 0$ and $x_1 = 1$. For every iteration we print the number of correct digits (first column) and its ratio to the number of correct digits in the previous step (second column). This ratio should converge to the convergence rate $p = 2.73$. We see that this is the case.

```
> f := x -> exp(x) + x;
```

$$f := x \to e^x + x,$$

```
> solve(f(x)=0, x);
```

$$-\mathrm{LambertW}(1),$$

```
> Digits := 500:
> x0 := 0.0: x1 := 1.0:
> for i to 6 do x2 := evalf(F(x0,x1));
>    d2 := evalf(log[10](abs(x2+LambertW(1)))):
>    if i = 1 then lprint(evalf(d2,20))
>    else lprint(evalf(d2,20), evalf(d2/d1,20))
>    fi;
>    x0 := x1: x1 := x2: d1 := d2:
> od:
```

```
-2.7349672475721192576
-7.8214175487946676893   2.8597847216407742880
-23.118600850923949542   2.9558070140989569391
-63.885801375143936026   2.7633939349141321183
-176.01456902838525572   2.7551437915728687417
-481.80650538330786806   2.7373103717659224742.
```

5. Newton–Cotes Rules. Another example of rules that can be derived easily with a computer algebra system is quadrature rules for approximating a definite integral. A well-known example is Simpson's rule,

$$\int_a^b f(x)\,dx \approx (b-a)\left(\frac{1}{6}f(a) + \frac{2}{3}f\left(\frac{a+b}{2}\right) + \frac{1}{6}f(b)\right),$$

which uses three equidistant function values at the endpoints and in the middle of the integration interval. The formula is obtained by interpolating the three function values and by computing the integral of the interpolating polynomial of degree 2. Let us first define a polynomial of degree 2. We then state the interpolation conditions and solve the resulting linear system for the coefficients of the polynomial. Finally, we integrate the polynomial and simplify the result.

```
> p := x-> a0 + a1*x + a2*x^2:
> solve( {p(a)=f(a), p((a+b)/2)=f((a+b)/2), p(b)=f(b)}, {a0,a1,a2}):
> assign(%);
> factor(int(p(x), x=a..b));
```

$$-\frac{1}{6}\left(f(a) + 4\,f\left(\frac{1}{2}\,a + \frac{1}{2}\,b\right) + f(b)\right)(a-b).$$

What is the error of this integration rule? Discretization errors can be computed very simply by appropriate series expansions. Let $h = (b-a)/2$; then we obtain

```
> En := subs(b = a+2*h, int(f(x), x=a..b) - %);
```

$$En := \int_a^{a+2\,h} f(x)\,dx - \frac{1}{3}\left(f(a) + 4\,f(a+h) + f(a+2\,h)\right)h,$$

```
> series(En, h);
```

$$-\frac{1}{90}\,(\mathrm{D}^{(4)})(f)(a)\,h^5 + \mathrm{O}(h^6).$$

This shows that the error is proportional to h^5. For the composite Simpson rule (n intervals of length $2h$, $b-a=2nh$), the error therefore is $\frac{b-a}{180}h^4 f^{(4)}(\xi)$ for some ξ in (a,b).

Instead of computing the interpolation polynomial as above, we can use the Maple function `interp` to interpolate a polynomial through the points $(0,y_0),(h,y_1)$, and $(2h,y_2)$. By integration we obtain the same result as above.

```
> factor(int(interp([0,h,2*h],[y[0],y[1],y[2]],z),z=0..2*h)*(b-a)/(2*h));
```

$$\frac{1}{6}\,(b-a)\,(y_0+4\,y_1+y_2).$$

More generally, we obtain *Newton–Cotes Rules* for $\int_a^b f(x)\,dx$ by interpolating $n+1$ function values with given equidistant nodes and by integrating the degree-n interpolation polynomial. The following procedure generates such an $(n+1)$-point normalized Newton–Cotes rule.

```
> Cotes := n -> factor((b-a)/(n*h) *
>    int(interp([seq(i*h, i=0..n)], [seq(y[i], i=0..n)], z), z=0..n*h)):
```

With this procedure we can, e.g., construct the trapezoidal rule ($n=1$), the Milne rule ($n=4$), or the Weddle rule ($n=6$):

```
> Cotes(1);
```

$$\frac{1}{2}\,(b-a)\,(y_0+y_1),$$

```
> Cotes(4);
```

$$\frac{1}{90}\,(b-a)\,(7\,y_0+32\,y_1+12\,y_2+32\,y_3+7\,y_4),$$

```
> Cotes(6);
```

$$\frac{1}{840}\,(b-a)\,(41\,y_0+216\,y_1+27\,y_2+272\,y_3+27\,y_4+216\,y_5+41\,y_6).$$

For $n=8$ we obtain the following equidistant nine-point rule which is used by the MATLAB function `quad8`.

```
> Cotes(8);
```

$$\frac{1}{28350}\,(b-a)\,(989\,y_0+5888\,y_1-928\,y_2+10496\,y_3-4540\,y_4+10496\,y_5 \\ -928\,y_6+5888\,y_7+989\,y_8).$$

In [6] we find one-sided formulas that can also be generated with Maple in the same way. For example, given four equidistant function values (three intervals), find an approximation for the integral over the third interval:

```
> factor(int(interp([0,h,2*h,3*h],[y[0],y[1],y[2],y[3]], t), t=2*h..3*h));
```

$$\frac{1}{24}\,h\,(9\,y_3+19\,y_2-5\,y_1+y_0).$$

6. Approximating Derivatives. When replacing derivatives by finite differences, one uses relations like, e.g.,

$$y''(x) \approx \frac{y(x-h) - 2y(x) + y(x+h)}{h^2}. \tag{6.1}$$

They are obtained by computing derivatives of the corresponding interpolation polynomial. Relation (6.1) is obtained by the Maple statement

```
> diff(interp([x-h, x, x+h], [y(x-h), y(x), y(x+h)], z), z$2);
```

$$\frac{y(x+h) - 2\,y(x) + y(x-h)}{h^2}.$$

Again, we can determine the discretization error with the help of a series expansion.

```
> % - D(D(y))(x);
```

$$\frac{y(x+h) - 2\,y(x) + y(x-h)}{h^2} - (\mathrm{D}^{(2)})(y)(x),$$

```
> series(%, h);
```

$$\frac{1}{12}\,(\mathrm{D}^{(4)})(y)(x)\,h^2 + \mathrm{O}(h^4).$$

Similarly, with the statements

```
> diff(interp([x, x+h, x+2*h], [y(x), y(x+h), y(x+2*h)], z), z):
> normal(subs(z=x, %));
```

$$-\frac{1}{2}\,\frac{3\,y(x) + y(x+2\,h) - 4\,y(x+h)}{h},$$

an approximation for $y'(x)$ is obtained. The discretization error of this approximation is also of order h^2.

```
> series(% - D(y)(x), h, 4);
```

$$-\frac{1}{3}\,(\mathrm{D}^{(3)})(y)(x)\,h^2 + \mathrm{O}(h^3).$$

7. Gauss Quadrature. The idea of Gauss quadrature is to find nodes x_i and weights w_i so that the quadrature rule

$$\int_{-1}^{1} f(x)\,dx \approx \sum_{i=1}^{n} w_i\,f(x_i) \tag{7.1}$$

is exact for polynomials of degree as high as possible. For $n = 3$ we have to determine the six unknowns w_1, w_2, w_3, x_1, x_2, and x_3. We demand exact values for the integrals of the monomials x^j for $j = 0, \ldots, 5$ and obtain six (nonlinear) equations:

$$w_1 x_1^j + w_2 x_2^j + w_3 x_3^j = \int_{-1}^{1} x^j\,dx, \tag{7.2}$$

```
> eqns := {seq(w[1]*x[1]^k+w[2]*x[2]^k+w[3]*x[3]^k = int(x^k, x=-1..1),
>              k=0..5)};
```

$$\begin{aligned} eqns := \Big\{ & w_1\,x_1 + w_2\,x_2 + w_3\,x_3 = 0,\ w_1\,{x_1}^2 + w_2\,{x_2}^2 + w_3\,{x_3}^2 = \frac{2}{3}, \\ & w_1\,{x_1}^5 + w_2\,{x_2}^5 + w_3\,{x_3}^5 = 0,\ w_1\,{x_1}^4 + w_2\,{x_2}^4 + w_3\,{x_3}^4 = \frac{2}{5}, \\ & w_1\,{x_1}^3 + w_2\,{x_2}^3 + w_3\,{x_3}^3 = 0,\ w_1 + w_2 + w_3 = 2 \Big\}. \end{aligned}$$

We can solve this system with Maple:

```
> sols := solve(eqns, indets(eqns, name)):
> convert(sols[1], radical);
```

$$\left\{w_3 = \frac{5}{9},\ w_2 = \frac{5}{9},\ w_1 = \frac{8}{9},\ x_2 = -\frac{1}{5}\sqrt{3}\sqrt{5},\ x_3 = \frac{1}{5}\sqrt{3}\sqrt{5},\ x_1 = 0\right\}.$$

This brute force approach will not work for all values of n. For larger n the system of nonlinear equations becomes too complicated for Maple. One has to add some more sophisticated theory to compute the rules. It is our goal to find nodes and weights to get an exact rule for polynomials of degree up to $2n-1$:

$$\int_{-1}^{1} P_{2n-1}(x)\,dx = \sum_{i=1}^{n} w_i P_{2n-1}(x_i). \tag{7.3}$$

We can argue as follows. Consider the decomposition of P_{2n-1} obtained by dividing by some polynomial $Q_n(x)$ of degree n:

$$P_{2n-1}(x) = H_{n-1}(x)Q_n(x) + R_{n-1}(x).$$

Then

$$\int_{-1}^{1} P_{2n-1}(x)\,dx = \int_{-1}^{1} H_{n-1}(x)Q_n(x)\,dx + \int_{-1}^{1} R_{n-1}(x)\,dx.$$

Applying rule (7.3) on both sides and subtracting yields the following expression for the error:

$$\begin{aligned}\text{error} := \int_{-1}^{1} H_{n-1}(x)Q_n(x)\,dx - \sum_{i=1}^{n} w_i H_{n-1}(x_i)Q_n(x_i)\\ + \int_{-1}^{1} R_{n-1}(x)\,dx - \sum_{i=1}^{n} w_i R_{n-1}(x_i).\end{aligned}$$

Now it is easy to see that we can make the error zero by the following choices. First take $Q_n(x)$ as the orthogonal polynomial on the interval $[-1,1]$ corresponding to the scalar product

$$(f,g) = \int_{-1}^{1} f(x)g(x)\,dx.$$

By this choice and by the definition of an orthogonal polynomial, the first term in the error vanishes: $\int_{-1}^{1} H_{n-1}(x)Q_n(x)\,dx = 0$. Q_n is a *Legendre* polynomial available in Maple as `orthopoly[P](n,x)`.

Second, choose as the nodes the (real) zeros of Q_n. Then the second term in the error will also vanish: $\sum_{i=1}^{n} w_i H_{n-1}(x_i)Q_n(x_i) = 0$.

Finally, compute the weights according to Newton–Cotes by integrating the interpolation polynomial for R_{n-1}, which is of course again R_{n-1} by the uniqueness of the interpolation polynomial. Thus

$$\int_{-1}^{1} R_{n-1}(x)\,dx = \sum_{i=1}^{n} w_i R_{n-1}(x_i),$$

and the last two error terms cancel.

So, we can compute a Gauss quadrature rule, e.g., for $n = 12$, with the following Maple statements:

```
> X := sort([fsolve(orthopoly[P](12, x)=0, x)]);
```

$$X := [-.9815606342,\ -.9041172564,\ -.7699026742,\ -.5873179543,\ -.3678314990,\ -.1252334085,\ .1252334085,\ .3678314990,\ .5873179543,\ .7699026742,\ .9041172564,\ .9815606342],$$

```
> interp(X, [seq(y[i], i=1..12)], z):
> Q := int(%, z=-1..1);
```

$$\begin{aligned} Q := {} & .04717506586\,y_1 + .1069394295\,y_2 + .1600776434\,y_3 + .2031689029\,y_4 \\ & + .2334973032\,y_5 + .2491475198\,y_6 + .2491470907\,y_7 + .2334921379\,y_8 \\ & + .2031674530\,y_9 + .1600783833\,y_{10} + .1069391353\,y_{11} + .04717532540\,y_{12}. \end{aligned}$$

We note that numerical errors occur (the weights should be symmetric) because we are computing the rules here in a well-known unstable way. However, Maple offers us more precision by increasing the value of `Digits`. With two runs of the above statements with different precision we are able to obtain the rules correct to the number of decimal digits we want.

The Gauss–Lobatto quadrature rule on $[-1, 1]$ using the endpoints and two intermediate points can be computed with a computer algebra system as follows [1]. Considering the symmetry of the formula, we stipulate

```
> A := f -> int(f(t), t=-1..1) = a*(f(-1) + f(1)) + b*(f(-xi) + f(xi));
```

$$A := f \to \int_{-1}^{1} f(t)\,dt = a\,(f(-1) + f(1)) + b\,(f(-\xi) + f(\xi))$$

and require it to be exact for $f(x) = 1, x^2$, and x^4.

```
> solve({A(x->1), A(x->x^2), A(x->x^4)}, {a,b,xi});
```

$$\left\{a = \frac{1}{6},\ b = \frac{5}{6},\ \xi = \mathrm{RootOf}(5\_Z^2 - 1)\right\}.$$

If we would like to compute a Kronrod extension by adding three more points (by symmetry, one of them will be 0), we stipulate

```
> A := f -> int(f(t), t=-1..1) = a*(f(-1) + f(1)) + b*(f(-xi) + f(xi))
>           + c*(f(-1/sqrt(5)) + f(1/sqrt(5))) + d*f(0);
```

$$\begin{aligned} A := {} & f \to \int_{-1}^{1} f(t)\,dt \\ & = a\,(f(-1) + f(1)) + b\,(f(-\xi) + f(\xi)) + c\left(f\left(-\frac{1}{\sqrt{5}}\right) + f\left(\frac{1}{\sqrt{5}}\right)\right) + d\,f(0) \end{aligned}$$

and require exactness for $f(x) = 1, x^2, x^4, x^6$, and x^8.

```
> seq(A(unapply(x^(2*i),x)), i=0..4);
```

$$2 = 2\,a + 2\,b + 2\,c + d, \quad \frac{2}{3} = 2\,a + 2\,b\,\xi^2 + \frac{2}{5}\,c, \quad \frac{2}{5} = 2\,a + 2\,b\,\xi^4 + \frac{2}{25}\,c,$$
$$\frac{2}{7} = 2\,a + 2\,b\,\xi^6 + \frac{2}{125}\,c, \quad \frac{2}{9} = 2\,a + 2\,b\,\xi^8 + \frac{2}{625}\,c$$

```
> solve({%}, {a,b,c,d,xi});
```

$$\left\{\xi = \mathrm{RootOf}(3\_Z^2 - 2),\ b = \frac{72}{245},\ d = \frac{16}{35},\ c = \frac{125}{294},\ a = \frac{11}{210}\right\}.$$

Thus our rule becomes

$$\int_{-1}^{1} f(t)\,dt = \frac{11}{210}(f(-1)+f(1)) + \frac{72}{245}\left(f\left(-\sqrt{\frac{2}{3}}\right)+f\left(\sqrt{\frac{2}{3}}\right)\right) + \frac{125}{294}\left(f\left(-\frac{1}{\sqrt{5}}\right)+f\left(\frac{1}{\sqrt{5}}\right)\right) + \frac{16}{35}f(0).$$

For a more elaborate treatment of generating Gauss quadrature formulas symbolically, we refer to [7].

8. Generation of Explicit Runge–Kutta Formulas. In this section we show how a computer algebra system can be used to derive explicit Runge–Kutta formulas. Such formulas are used to solve systems of differential equations of first order. The solution of the initial value problem

$$y'(x) = f\big(x, y(x)\big), \qquad y(x_k) = y_k \tag{8.1}$$

can be approximated by a Taylor series around x_k, which is obtained from (8.1) by repeated differentiation and replacing $y'(x)$ by $f\big(x, y(x)\big)$ every time it appears.

$$\begin{aligned} y(x_k+h) &= \sum_{i=0}^{\infty} y^{(i)}(x_k)\frac{h^i}{i!} \\ &= y(x_k) + hf\big(x_k, y(x_k)\big) + \frac{h^2}{2}\left(\frac{\partial}{\partial x} f\big(x, y(x)\big)\bigg|_{x=x_k}\right) + \cdots \\ &= y(x_k) \\ &\quad + \underbrace{h\left(f\big(x_k, y(x_k)\big) + \frac{h}{2}\Big(f_x\big(x_k, y(x_k)\big) + f\big(x_k, y(x_k)\big) f_y\big(x_k, y(x_k)\big)\Big) + \cdots\right)}_{\Phi(x_k, y(x_k), h)}. \end{aligned} \tag{8.2}$$

The idea of the Runge–Kutta methods is to approximate the Taylor series (8.2) up to order m by using only *values* of $f\big(x, y(x)\big)$ and none of its derivatives. The general form of an *s-stage explicit Runge–Kutta method* is

$$\begin{aligned} k_1 &= f(x, y), \\ k_2 &= f(x + c_2\,h, y + h\,a_{2,1}\,k_1), \\ &\vdots \\ k_s &= f\left(x + c_s\,h, y + h\sum_{j=1}^{s-1} a_{s,j}\,k_j\right), \\ \Phi(x, y, h) &= \sum_{i=1}^{s} b_i k_i, \\ y_{k+1} &= y_k + h\,\Phi(x_k, y_k, h), \end{aligned} \tag{8.3}$$

where s is the number of "stages" and $a_{i,j}$, b_i, and c_i are real coefficients.

To derive the coefficients of such a method, the series expansions of (8.2) and (8.3) are equated. This leads to a set of nonlinear equations that must be solved for the parameters $a_{i,j}$, b_i, and c_i.

For this derivation we must compute the Taylor series expansions of (8.2) and (8.3). Maple knows how to expand a function with two parameters that both depend on x, but we have to inform Maple that $y'(x)$ is to be replaced by $f\big(x, y(x)\big)$ whenever it appears. We do this by overwriting the derivative of the operator `y`.

```
> D(y) := x -> f(x,y(x)):
> taylor(y(x+h), h=0, 3);
```

$$y(x) + f(x,\, y(x))\, h + \left(\frac{1}{2}\, D_1(f)(x,\, y(x)) + \frac{1}{2}\, D_2(f)(x,\, y(x))\, f(x,\, y(x))\right) h^2 + \mathrm{O}(h^3).$$

In this result, $D_1(f)(x, y(x))$ stands for the derivative of f with respect to the first argument, i.e., $f_x\big(x, y(x)\big)$. In order to make the result more readable, we define some alias substitutions.

```
> alias(F = f(x,y(x)), Fx = D[1](f)(x,y(x)), Fy = D[2](f)(x,y(x)),
>       Fxx = D[1,1](f)(x,y(x)), Fxy = D[1,2](f)(x,y(x)),
>       Fyy = D[2,2](f)(x,y(x))):
```

We are now ready to derive the parameters of a Runge–Kutta formula for $s = 3$, which is of order $m = 3$.

```
> m := 3:
> taylor(y(x+h),h=0,m+1);
```

$$y(x) + F\, h + \left(\frac{1}{2}\, Fx + \frac{1}{2}\, Fy\, F\right) h^2 + \left(\frac{1}{6}\, Fxx + \frac{1}{3}\, F\, Fxy + \frac{1}{6}\, F^2\, Fyy + \frac{1}{6}\, Fy\, Fx + \frac{1}{6}\, Fy^2\, F\right) h^3 + \mathrm{O}(h^4),$$

```
> TaylorPhi := normal((convert(%,polynom) - y(x))/h);
```

$$\mathit{TaylorPhi} := F + \frac{1}{2}\, h\, Fx + \frac{1}{2}\, h\, Fy\, F + \frac{1}{6}\, h^2\, Fxx + \frac{1}{3}\, h^2\, F\, Fxy + \frac{1}{6}\, h^2\, F^2\, Fyy + \frac{1}{6}\, h^2\, Fy\, Fx + \frac{1}{6}\, h^2\, Fy^2\, F.$$

The variable `TaylorPhi` corresponds to Φ in (8.2).

For the Runge–Kutta scheme we get the following Taylor series. Note that we keep the parameters $a_{i,j}$, b_i, and c_i in symbolic form. `RungeKuttaPhi` corresponds to Φ in (8.3).

```
> k1 := taylor(f(x,        y(x)),                          h=0, m):
> k2 := taylor(f(x+c[2]*h, y(x)+h*(a[2,1]*k1)),            h=0, m):
> k3 := taylor(f(x+c[3]*h, y(x)+h*(a[3,1]*k1+a[3,2]*k2)),h=0, m):
> RungeKuttaPhi := convert(series(b[1]*k1+b[2]*k2+b[3]*k3,h,m),polynom);
```

$$\begin{aligned}\mathit{RungeKuttaPhi} :=\ & b_1\, F + b_2\, F + b_3\, F \\ & + (b_2\, (Fx\, c_2 + Fy\, a_{2,1}\, F) + b_3\, (Fx\, c_3 + Fy\, a_{3,1}\, F + Fy\, a_{3,2}\, F))\, h \\ & + \left(b_2\, \left(\frac{1}{2}\, Fxx\, {c_2}^2 + c_2\, Fxy\, a_{2,1}\, F + \frac{1}{2}\, {a_{2,1}}^2\, F^2\, Fyy\right)\right. \\ & + b_3 \left(\frac{1}{2}\, Fxx\, {c_3}^2 + c_3\, Fxy\, a_{3,1}\, F + c_3\, Fxy\, a_{3,2}\, F + \frac{1}{2}\, {a_{3,1}}^2\, F^2\, Fyy\right. \\ & \left.\left. + a_{3,1}\, F^2\, Fyy\, a_{3,2} \ + \frac{1}{2}\, {a_{3,2}}^2\, F^2\, Fyy + Fy\, a_{3,2}\, Fx\, c_2 + Fy^2\, a_{3,2}\, a_{2,1}\, F\right)\right) h^2.\end{aligned}$$

$$
\begin{array}{c|cc}
0 & & \\
\frac{2}{3}\ \frac{3\,b_3\,{c_3}^2-1}{2\,b_3\,c_3-1} & \frac{2}{3}\ \frac{3\,b_3\,{c_3}^2-1}{2\,b_3\,c_3-1} & \\
c_3 & \frac{1-6\,b_3\,c_3+12\,{b_3}^2{c_3}^3}{4\,b_3\,(3\,b_3\,{c_3}^2-1)} & \frac{2\,b_3\,c_3-1}{4\,b_3\,(3\,b_3\,{c_3}^2-1)} \\
\hline
 & \frac{12\,b_3\,{c_3}^2-1+4\,b_3-12\,b_3\,c_3}{12\,b_3\,{c_3}^2-4} & \frac{3}{4}\,\frac{(2b_3\,c_3-1)^2}{1-3b_3\,{c_3}^2} \qquad b_3
\end{array}
$$

Fig. 8.1 *Three-stage Runge–Kutta methods of order* 3 *(first solution).*

$$
\begin{array}{c|cc}
0 & & \\
\frac{2}{9a_{3,2}} & \frac{2}{9a_{3,2}} & \\
\frac{2}{3} & \frac{2}{3}-a_{3,2} \qquad a_{3,2} & \\
\hline
 & \frac{1}{4} \qquad 0 \qquad \frac{3}{4} &
\end{array}
$$

Fig. 8.2 *Three-stage Runge–Kutta methods of order* 3 *(second solution).*

The difference `d` between the two polynomials `TaylorPhi` and `RungeKuttaPhi` should be zero. We consider `d` to be a polynomial in the unknowns `h`, `F`, `Fx`, `Fy`, `Fxx`, etc., and set the coefficients of that polynomial to zero. This gives us a nonlinear system of equations that must be solved.

```
> d := expand(TaylorPhi-RungeKuttaPhi):
> eqns := {coeffs(d, [h,F,Fx,Fy,Fxx,Fxy,Fyy])};
```

$$
\begin{aligned}
eqns := \Big\{ & -b_2\,c_2 - b_3\,c_3 + \frac{1}{2},\ -\frac{1}{2}\,b_3\,{c_3}^2 - \frac{1}{2}\,b_2\,{c_2}^2 + \frac{1}{6}, \\
& -b_3\,a_{3,1} - b_3\,a_{3,2} - b_2\,a_{2,1} + \frac{1}{2}, \\
& \frac{1}{6} - \frac{1}{2}\,b_2\,{a_{2,1}}^2 - b_3\,a_{3,1}\,a_{3,2} - \frac{1}{2}\,b_3\,{a_{3,2}}^2 - \frac{1}{2}\,b_3\,{a_{3,1}}^2, \\
& -b_2\,c_2\,a_{2,1} - b_3\,c_3\,a_{3,2} + \frac{1}{3} - b_3\,c_3\,a_{3,1},\ 1 - b_2 - b_3 - b_1,\ \frac{1}{6} - b_3\,a_{3,2}\,c_2, \\
& \frac{1}{6} - b_3\,a_{3,2}\,a_{2,1} \Big\},
\end{aligned}
$$

```
> solve(eqns, indets(eqns));
```

$$
\begin{aligned}
& \Big\{ b_3 = b_3,\ a_{3,2} = \frac{1}{4}\,\frac{2\,b_3\,c_3-1}{b_3\,(3\,b_3\,{c_3}^2-1)},\ c_2 = \frac{2}{3}\,\frac{3\,b_3\,{c_3}^2-1}{2\,b_3\,c_3-1}, \\
& b_1 = \frac{1}{4}\,\frac{12\,b_3\,{c_3}^2-1-12\,b_3\,c_3+4\,b_3}{3\,b_3\,{c_3}^2-1},\ b_2 = -\frac{3}{4}\,\frac{4\,{b_3}^2\,{c_3}^2-4\,b_3\,c_3+1}{3\,b_3\,{c_3}^2-1}, \\
& a_{3,1} = \frac{1}{4}\,\frac{-6\,b_3\,c_3+1+12\,{b_3}^2\,{c_3}^3}{b_3\,(3\,b_3\,{c_3}^2-1)},\ a_{2,1} = \frac{2}{3}\,\frac{3\,b_3\,{c_3}^2-1}{2\,b_3\,c_3-1},\ c_3 = c_3 \Big\}, \\
& \Big\{ a_{3,2} = a_{3,2},\ b_3 = \frac{3}{4},\ b_2 = 0,\ a_{3,1} = \frac{2}{3} - a_{3,2},\ a_{2,1} = \frac{2}{9}\,\frac{1}{a_{3,2}}, \\
& c_2 = \frac{2}{9}\,\frac{1}{a_{3,2}},\ b_1 = \frac{1}{4},\ c_3 = \frac{2}{3} \Big\}.
\end{aligned}
$$

For $s = 3$, we found two (parameterized) solutions which can be represented by the coefficient schemes shown in Figures 8.1 and 8.2. Note that in the first solution,

the unknowns c_3 and b_3 are free parameters; i.e., they can take on any value. This is indicated by the entries `c3 = c3` and `b3 = b3` in the solution set. For the second solution, $a_{3,2}$ is a free parameter. From this solution we get Heun's method of third order if we set $a_{3,2} = 2/3$. For further information on this topic, we refer to [4].

9. Methods of Ritz and Galerkin. This section shows an example in which the computer algebra systems take over the hard manual work and the students can understand the general steps of the method. Consider the boundary value problem

$$\mathcal{D}(u(x)) = -u''(x) + 2u(x) = f(x), \quad u(0) = 0, \quad u'(\pi) = 0. \tag{9.1}$$

For $f(x) = \frac{17}{4}\sin(\frac{3}{2}x)$, the solution is

```
> DG := u -> -D(D(u)) + 2*u - f:
> f  := x -> 17/4*sin(3/2*x):
> dsolve({DG(u)(x)=0, u(0)=0, D(u)(Pi)=0}, u(x));
```

$$u(x) = \sin\left(\frac{3}{2}x\right),$$

```
> assign(%):
```

Equation (9.1) is obtained if we want to minimize the functional

$$L = \int_0^\pi u'(x)^2 + 2u(x)^2 - 2f(x)u(x)\,dx. \tag{9.2}$$

In the method of *Ritz* the solution u is approximated by a linear combination of known functions $\phi_j(x)$:

$$y(x) = \sum_{j=1}^{n} c_j \phi_j(x). \tag{9.3}$$

Introducing $y(x)$ in (9.2) we obtain a quadratic form in the unknown coefficients c_j. Minimizing this quadratic form gives us values for c_j and an approximation $y(x)$.

The principle is easy to describe, but to compute a concrete example is rather tedious, even if we choose only $n = 2$, $\phi_1(x) = x - \frac{x^2}{2\pi}$, and $\phi_2(x) = x\exp(-\frac{x}{\pi})$. Both functions satisfy the boundary conditions.

With Maple we can compute and plot the approximation with the following statements:

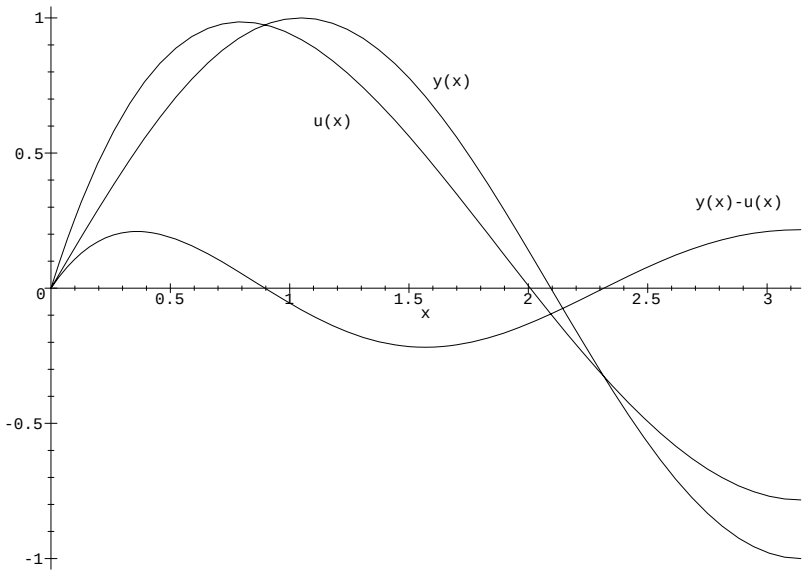
```
> y := x -> c1*(x-x^2/(2*Pi)) + c2*x*exp(-x/Pi):
> L := int(diff(y(x),x)^2+2*y(x)^2-2*f(x)*y(x),x=0..Pi):
> evalf(L);
```

$$9.315538005\,c1^2 + 5.69163633\,c2^2 + 14.5140300\,c1\,c2 - 1.582748541\,c2 - .8016693429\,c1,$$

```
> v := linalg[grad](L, [c1,c2]):
> solve({v[1],v[2]}, {c1,c2}):
> evalf(%);
```

$$\{c2 = 12.52353174,\ c1 = -9.713086137\},$$

```
> assign(%);
> plot({y(x), u(x), y(x)-u(x)}, x=0..Pi, color=black);
```



Not every differential equation minimizes a functional. In the Galerkin method one tries to solve (9.1) also by a linear combination of the form (9.3). The goal is to choose the coefficients c_j such that the residual $r(x) = \mathcal{D}y(x) - f(x)$ becomes small in some sense. For this we have to choose another set of functions $\{\psi_j(x)\}$. The coefficients c_j are now computed in such a way that the *residual is orthogonal to the space spanned by the functions* $\psi_j(x)$:

$$\int_0^\pi r(x)\psi_j(x)\,dx = 0, \quad j = 1, \ldots, n. \tag{9.4}$$

Thus we again obtain a system of linear equations for the coefficients c_j.

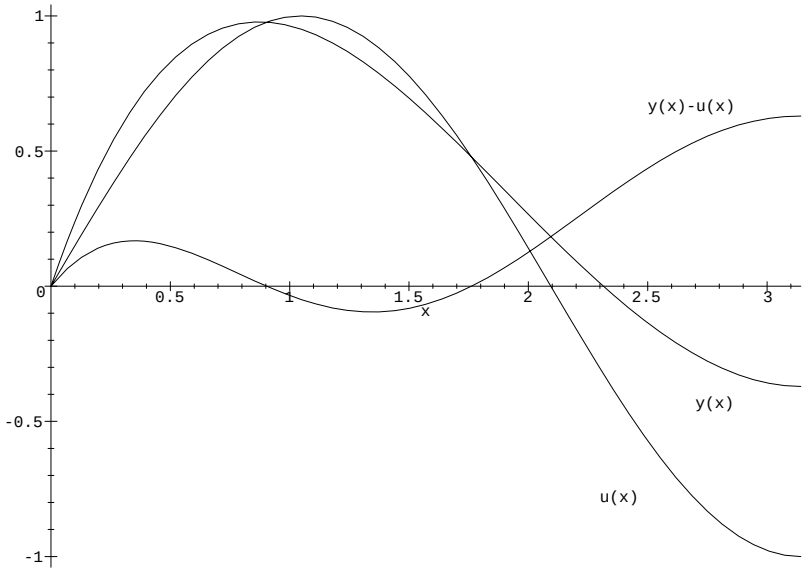
```
> c1 := 'c1': c2 := 'c2':
> psi1 := x -> sin(x):
> psi2 := x -> cos(x):
> eq := {int(DG(y)(t)*psi1(t), t=0..Pi), int(DG(y)(t)*psi2(t), t=0..Pi)}:
> evalf(eq);
```

$$\{-1.038400155\,c2 - 5.100000003 - 2.000000001\,c1,\\ 5.051451973\,c1 - 3.400000002 + 4.146353669\,c2\},$$

```
> solve(eq, {c1, c2}):
> evalf(%);
```

$$\{c1 = -8.098032911,\ c2 = 10.68573206\},$$

```
> assign(%);
> plot({y(x), u(x), y(x)-u(x)}, x=0..Pi, color=black);
```



10. Conclusions. In this paper we have given some examples of how we use computer algebra systems in our scientific computing courses. Many numerical methods and classical proofs can be developed with only a few statements in a computer algebra system.

However, it might be difficult sometimes to find the right ones, so this method is in general still restricted to the reproduction of classical results. The use of a computer algebra system also requires much experience, as it is not always easy to find an elegant way to obtain the result one expects.

We have also observed that it may be particularly complicated to convince a computer algebra system to perform a specific task. As an example, take the convergence analysis in section 3. We came across the expression $(a^p)^p/a^p/a$ and were interested in what the exponent would be if this expression were written as a^b. How is b obtained? Right—by taking the logarithm to the base a. The result, however, does not simplify, even after using the `simplify` command.

```
> b := log[a]((a^p)^p/a^p/a);
```

$$b := \frac{\ln\left(\dfrac{(a^p)^p}{a^p\, a}\right)}{\ln(a)},$$

```
> simplify(b);
```

$$\frac{\ln\left((a^p)^p\, a^{(-p-1)}\right)}{\ln(a)}.$$

What is the problem? Maple does not know as much as we do. Maple cannot simplify this expression, as it assumes a and p to be complex numbers. Obviously, a and p are real and positive in our context, but Maple has to be informed about this fact using the `assume` facility. This can be done directly in the `simplify` command for all the indeterminants that appear in the expression to be simplified or for each unknown with the `assume` command. The symbol ~ signals that assumptions have been made on a variable.

```
> simplify(b, assume=positive);
```

$$p^2 - p - 1,$$

```
> assume(a > 0);   # a is assumed to be real and positive
> assume(p, real); # b is assumed to be real
> simplify(b);
```

$$p^{\sim 2} - p^{\sim} - 1.$$

Consider also the discussion of discretization errors in section 6. We have computed a series expansion of an expression comparable to the following

```
> f(x+h) - f(x) - h*diff(f(x), x);
```

$$f(x+h) - f(x) - h\left(\frac{\partial}{\partial x} f(x)\right).$$

Which leading term do you expect if this expression is expanded into a series? Maple gives you the following answer:

```
> series(%, h, 2);
```

$$\left(\mathrm{D}(f)(x) - \left(\frac{\partial}{\partial x} f(x)\right)\right) h + \mathrm{O}(h^2).$$

This result may be surprising for a student. The leading coefficient is indeed zero, but Maple does not recognize this zero automatically. In general, it is particularly difficult to recognize zeros, but in this example the above result can be simplified using a special option to the command `simplify`.

```
> simplify(%, diff);
```

$$\mathrm{O}(h^2).$$

Computer algebra systems still need to progress further. They are not yet a replacement for paper and pencil. We must also admit that computer algebra systems still have bugs and sometimes produce erroneous results or results that are valid only under some assumptions. It is also very important to demonstrate this fact to students. Students should learn that results cannot be trusted blindly. Whenever a numerical method is derived, the result must be compared with one's expectations.

Nevertheless, as many examples in this article have demonstrated, a computer algebra system is a very powerful tool for use in teaching numerical methods. Further examples of the use of computer algebra systems can be found in, e.g., [2, 3, 5].

REFERENCES

[1] W. Gander and W. Gautschi, *Adaptive Quadrature—Revisited*, Research Report, Computer Science Department, ETH, Zürich, Switzerland, 1998.

[2] W. Gander and D. Gruntz, *The billiard problem*, Internat. J. Math. Ed. Sci. Tech., 23 (1992), pp. 825–830.

[3] W. Gander and J. Hřebíček, eds., *Solving Problems in Scientific Computing Using Maple and Matlab*, 3rd ed., Springer-Verlag, Berlin, 1997.

[4] D. Gruntz, *Symbolic computation of explicit Runge–Kutta formulas*, in Solving Problems in Scientific Computing Using Maple and Matlab, W. Gander and J. Hřebíček, eds., Springer-Verlag, Berlin, 1997, pp. 281–296.

[5] D. Gruntz, *Automatic differentiation and bisection*, MapleTech, Maple Technical Newsletter, 4 (1997), pp. 14–19.

[6] E. Stiefel, *Einführung in die numerische Mathematik*, Teubner, Leipzig, Germany, 1976.

[7] U. von Matt, *Gauss Quadrature*, in Solving Problems in Scientific Computing Using Maple and Matlab, W. Gander and J. Hřebíček, eds., Springer-Verlag, Berlin, 1997, pp. 251–279.