Informatics in Education, 2024, Vol. 23, No. 4, 783–790


DOI: https://doi.org/10.15388/infedu.2024.27

# Pascal and Recursion

Walter GANDER
*Department of Computer Science, ETH Zurich, Switzerland*
*e-mail: gander@inf.ethz.ch*

*This historical note is dedicated to the memory of Niklaus Wirth.*

**Abstract.** In this paper, we show the impact of Pascal on numerical software. Using recursion, a new and elegant algorithm for quadrature could be developed.

**Key words:** Pascal, recursion, adaptive quadrature.

## 1. Introduction

In 1968, Heinz Rutishauser founded a specialist group for computer science in the Department of Mathematics at ETH Zurich. Niklaus Wirth, aged 34, just came back from Stanford and joined this new group together with Peter Läuchli, a local professor of the Seminar of Applied Mathematics.

In 1964, ETH bought a CDC 1604 computer and students who were interested had the opportunity to learn programming in Subset Algol 60. I did that with great pleasure and was quite fluent in Algol in 1968 when I received my diploma in mathematics. Heinz Rutishauser hired me to work as an assistant and graduate student in his new group.

Unfortunately, Heinz Rutishauser passed away already in 1970, two years after the foundation of the group. Niklaus Wirth became the new leader of the computer science group. He developed the first version of his new programming language Pascal. Frankly, I did not like it, because many features which I appreciated in Algol were missing in Pascal. For example the block structure was gone, there were no dynamical arrays, no functions or procedures were allowed as parameters of a procedure, no external code procedures, no automatic conversions from integers to reals, no goto-statements, etc. This made the translation of programs from Algol to Pascal difficult.

## 2. Pascal the Teaching Language

From 1969 to 1989, Urs Hochstrasser was the director of the Swiss Federal Office for Science and Education. He was a graduate student of Eduard Stiefel, who rented the Z4 computer from Konrad Zuse after World War II. It was installed at ETH and operational from 1950 to 1955. Urs Hochstrasser used the Z4 for his PhD studies, and therefore he

could foresee how important computer science would become. In 1983, he encouraged high school teachers to introduce computer science into the curriculum. This was the time when the first personal computers appeared. UCSD Pascal became very popular on the Apple II. In 1984, I spent a sabbatical in Stanford with the idea to write a book on numerical algorithms which could be used at secondary schools. During this time, I got a teaching assignment for the summer to teach Pascal to high school teachers on the IBM PC. One of the teachers told me about Turbo Pascal, a much better implementation than the original IBM Pascal.

Indeed Turbo Pascal from Borland turned out to be a wonderful small and effective system: about 30KB of memory were sufficient for the whole system containing an editor and a fast compiler. I became a fan of it. My book "Computermathematik" (Gander, 1992) and the second book (Gander, 1986) with the solutions to all exercises both use Turbo Pascal for the programs.

Turbo Pascal became the teaching language for programming all over the world, and even now it is still used as "Free Pascal," e.g., in Hong Kong.

## 3. Recursion

*"Recursion is a particularly powerful means in mathematical definitions."* Wirth (1976)

Algol 60 and the original Fortran were both programming languages in which recursion was not available. So in the numerical software as, e.g., published in the Springer Handbook Series (Wilkinson and Reinsch, 1971) and in LINPACK (Dongarra *et al.*, 1979), recursion was not used. Numerical analysts were therefore not familiar with this feature of a programming language.

Recursion was also for me a new feature. I discovered it only when I started to seriously use Turbo Pascal. I was not used to *think recursively* when programming. Like *computational thinking* one has to get aware of this possibility and it is important to train this mode of thinking.

Niklaus Wirth on the other hand was very familiar with recursion. It is certainly a useful technique when building a compiler. He designed Pascal as a language in which recursion is natural and available for the programmer. Wirth used recursive programming also often in examples in his books.

In the book *Algorithms + Data Structures = Programs* (Wirth, 1976), he writes:

> *"The power of recursion evidently lies in the possibility of defining an infinite set of objects by a finite statement. In the same manner, an infinite number of computations can be described by a finite recursive program, even if this program contains no explicit repetitions."*

This book contains a whole chapter on recursion. Many nice examples like Hilbert curves or the eight queens problem are implemented as recursive programs and they have become classical and well known algorithms all around the world.

## 4. Adaptive Quadrature

The idea of adaptive quadrature is to adapt the integration step-size automatically to the course of the function. By using different step-sizes, the goal is to have about the same discretization error for each interval. The possibility to use recursion in PASCAL opened the way for an elegant implementation for me.

Let $[a, b]$ be the interval of integration and $f$ a real function. We wish to numerically compute

$$I = \int_a^b f(x)\,\mathrm{d}x.$$

Integrating $f$ using two different methods we obtain two approximations $I_1$ and $I_2$. If the relative difference of $I_1$ and $I_2$ is smaller than some prescribed tolerance, one accepts, e.g., $I_1$ as the integral value.

Otherwise the interval $[a, b]$ is divided in two parts $[a, m]$ and $[m, b]$, where $m = (a + b)/2$, and the two integrals are computed independently:

$$I = \int_a^m f(x)\mathrm{d}x + \int_m^b f(x)\mathrm{d}x.$$

Recursion now comes naturally when we compute again two approximations for each integral and, if necessary, continue to subdivide the smaller intervals.

Every recursion must terminate. Wirth (1976) writes

> *"Like repetitive statements, recursive procedures introduce the possibility of non-terminating computations, and thereby also the necessity of considering the problem of* termination*. A fundamental requirement is evidently that the recursive calls of a procedure $P$ are subjected to a condition $B$, which at some time becomes non-satisfied."*

In our case we could try to test the relative difference of the two approximations $I_1$ and $I_2$ and stop the recursion when it is small. However, it is easy to show that this criterion may be too stringent, the difference of the approximations must only be small compared with the value of the whole integral. How can we do this without knowing the integral? Well, we do not need the exact value of the integral, a rough estimate is sufficient.

Thus we consider as approximation $\mathtt{is} = (|(b-a)/6 \cdot (f(a)+4f(m)+f(b))| + b - a)/2$. This is the average of a Simpson approximation plus the length of the interval. It is important that `is` be nonzero, which is why we add $b - a$.

The termination criterion would then be

$$|I_1 - I_2| < \mathtt{tol} \cdot \mathtt{is}$$

with some tolerance `tol`.

More elegant is a machine-independent criterion: terminate recursion if numerically the difference is too small to change `is`, that is, if

$$\texttt{is} + I_1 - I_2 = \texttt{is}.$$

In the following implementation we use for the approximation $I_1$ Simpson's rule for the step-size $h = (b-a)/2$, that is,

$$I_1 = \frac{b-a}{6}\left(f(a) + 4f(m) + f(b)\right), \quad m = \frac{b-a}{2},$$

and for $I_2$ we use Simpson's rule with half the step-size. To do so we need to compute a new function value between the values $\texttt{fa} = f(a)$ and $\texttt{fm} = f(m)$ and another between `fm` and $\texttt{fb} = f(b)$.

Next we would like to avoid the recomputing of function values. Therefore we pass the three function values `fa`, `fm`, `fb` as parameters to the next recursion step.

Finally, when we have computed both Simpson Values, we can use one Romberg step and perform one extrapolation step to improve the approximation. We thus replace the value of $I_1$ by

$$I_1 := \frac{16I_2 - I_1}{15}.$$

This leads to the following PASCAL function:

```
1  FUNCTION adapt(a,b,fa,fm,fb,is: real): real ;
2  VAR h,m,i1,i2,fml,fmr: real ;
3      (* global function f : real *)
4  BEGIN
5    m := (a+b)/2 ; h := (b-a)/4 ;
6    fml := f(a+h) ; fmr := f(b-h) ;
7    i1 := h/1.5*(fa+4*fm+fb) ;
8    i2 := h/3*(fa+4*(fml+fmr)+2*fm+fb) ;
9    i1 := (16*i2-i1)/15 ;
10   IF is + (i1 - i2) = is THEN
11   BEGIN
12     adapt := i1 ;
13     (* print here a, b-a, and i1 for didactical reasons *)
14   END
15   ELSE
16     adapt := adapt(a,m,fa,fml,fm,is) +
17              adapt(m,b,fm,fmr,fb,is) ;
18 END;
```

With the stopping criterion `is + (i1 - i2) = is`, the integral is computed to machine precision. If we wish to obtain a result with less precision, then we can define a

tolerance `tol` and simply enlarge the rough estimate by replacing

$$\texttt{is} := \texttt{is} * \texttt{tol}/\varepsilon,$$

where $\varepsilon$ denotes the machine precision.

## 5. Examples

We give two examples for concrete functions and intervals.

1. Integrate $f(x) = \sqrt{x}$ over the interval $[0, 1]$.

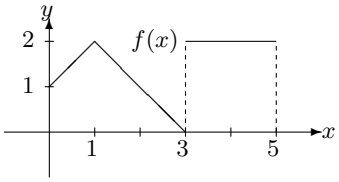
$$I = \int_0^1 \sqrt{x}\, \mathrm{d}x = \frac{2}{3} = 0.666666666666\ldots.$$

We compute the integral with $\texttt{tol} = 1e{-}5$ and get the result `0.66665999490706` with only 38 function evaluations. By printing the quantities `a`, `b-a`, and `i1` after the assignment `adapt:=i1`, we obtain the following table:

| a | b − a | i1 |
|---|---|---|
| 0.0000000 | 0.0078125 | 0.00045420327593 |
| 0.0078125 | 0.0078125 | 0.00084172670019 |
| 0.0156250 | 0.0156250 | 0.00238076263043 |
| 0.0312500 | 0.0312500 | 0.00673381360150 |
| 0.0625000 | 0.0625000 | 0.01904610104346 |
| 0.1250000 | 0.1250000 | 0.05387050881198 |
| 0.2500000 | 0.2500000 | 0.15236880834770 |
| 0.5000000 | 0.5000000 | 0.43096407049588 |

We needed only 8 sub-intervals. The smallest, the first one, has length $0.0078125$ and the length of the last interval is $0.5$. Adaptive quadrature is remarkably efficient for this example.

2. Non continuous function $f(x)$ integrated on $[0, 5]$:




We obtain the following table for `a`, `b-a`, and `i1`:

| a | b − a | i1 |
|---|---|---|
| 0.000000000000 | 0.625000000000 | 0.820312500000 |
| 0.625000000000 | 0.312500000000 | 0.556640625000 |
| 0.937500000000 | 0.039062500000 | 0.076446533203 |
| 0.976562500000 | 0.019531250000 | 0.038795471191 |
| 0.996093750000 | 0.019531250000 | 0.038936191134 |
| 1.015625000000 | 0.078125000000 | 0.151977539062 |
| 1.093750000000 | 0.156250000000 | 0.285644531250 |
| 1.250000000000 | 1.250000000000 | 1.406250000000 |
| 2.500000000000 | 0.312500000000 | 0.107421875000 |
| 2.812500000000 | 0.156250000000 | 0.017089843750 |
| 2.968750000000 | 0.019531250000 | 0.000419616699 |
| 2.988281250000 | 0.009765625000 | 0.000066757202 |
| 2.998046875000 | 0.001220703125 | 0.000001639127 |
| 2.999267578125 | 0.000610351562 | 0.000000260770 |
| 2.999877929687 | 0.000076293945 | 0.000000006402 |
| 2.999954223632 | 0.000076293945 | 0.000066122492 |
| 3.000030517578 | 0.000152587890 | 0.000305175781 |
| 3.000183105468 | 0.000305175781 | 0.000610351562 |
| 3.000488281250 | 0.002441406250 | 0.004882812500 |
| 3.002929687500 | 0.004882812500 | 0.009765625000 |
| 3.007812500000 | 0.039062500000 | 0.078125000000 |
| 3.046875000000 | 0.078125000000 | 0.156250000000 |
| 3.125000000000 | 0.625000000000 | 1.250000000000 |
| 3.750000000000 | 1.250000000000 | 2.500000000000 |

The exact value of the integral is 7.5. With $\texttt{tol} = 1e{-}6$, we obtain 7.500008477131 for the integral, and we needed 105 function evaluations. Note that the step-size is small around $x = 1$ (peak) and around $x = 3$ (step).

## 6. Final Remarks

This small, very simple but also effective PASCAL function for adaptive quadrature reflects truly the philosophy of Niklaus Wirth: *Small is beautiful* and the quote that he often cited: *Make it as simple as possible, but not simpler.*

I presented this algorithm the first time in the "Kolloquium über numerische Mathematik" at ETH on December 9, 1982. I demonstrated its performance live on an Apple II computer with UCSD PASCAL. Peter Henrici was very impressed – he had never before used a programming language which enabled recursion.

Later, Walter Gautschi and I expanded this idea of adaptive quadrature. We programmed two recursive quadrature functions in MATLAB. In the abstract of our paper (Gander and Gautschi, 2000), we wrote:

> *"Adaptive quadrature programs being recursive by nature, the choice of a good termination criterion is given particular attention. Two* MATLAB *quadrature programs are presented. The first is an implementation of the well-known adap-*

*tive recursive Simpson's rule; the second is new and is based on a four-point Gauss-Lobatto formula and two successive Kronrod extensions."*

Our quadrature functions outperformed the corresponding MATLAB functions and Cleve Moler wrote:

```
Date: Thu, 27 Apr 2000 17:19:58 -0400 (EDT)
From: Cleve Moler <moler@mathworks.com>
To: gander@inf.ethz.ch, wxg@cs.purdue.edu
Subject: Quadrature routines
Hi, guys --
I am in the process of making your quadrature routines
part of MATLAB 6.
```

For some years we had convinced even Cleve that recursion is useful. The adaptive functions for integration are explained in details in our textbook (Gander *et al.*, 2014).

## 7. Teaching Issues

1. The presented topic, to compute integrals using recursion, is suited to be taught in focus subjects, mathematics or computer science, at secondary schools. Prerequisite in mathematics is the concept of a Riemann sum, realized by the Trapezoidal – or Simpson's rule, to numerically compute a definite integral. Recursion can be explained in computer science using examples from the book by Wirth (1976).
2. Many integrals cannot be solved analytically, that is, they cannot be expressed by elementary functions. A well known example is the error function

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x \mathrm{e}^{-t^2}\,\mathrm{d}t.$$

   This function can be computed numerically, e.g., by using our function `adapt`.
3. New programming languages in general allow recursion. Therefore it is straightforward to translate the PASCAL function `adapt`.

   A main program still has to be written, which defines the function to be integrated together with the necessary initialization.
4. Finally a little challenge: solve the equation

$$f(x) = \int_0^1 \mathrm{e}^{xt^2}\mathrm{d}t - 2 = 0$$

   by computing $f$ and $f'$ with `adapt` and by using Newton's iteration

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

   **Note:** the solution is $x = 1.674824928512617$.

## References

Dongarra, J., Bunch, J.R., Moler, C.B., Stewart, G.W. (1979). *LINPACK Users' Guide*. SIAM.

Gander, W. (1986). *Computermathematik, Lösungen der Aufgaben mit Pascal Programmen*. Birkhäuser Basel. Download (free of charge) from https://people.inf.ethz.ch/gander/.

Gander, W. (1992). *Computermathematik* (2nd ed.). Birkhäuser Basel. Download (free of charge) from https://people.inf.ethz.ch/gander/.

Gander, W., Gautschi, W. (2000). Adaptive Quadrature – Revisited. *BIT*, 40(1), 84–101.

Gander, W., Gander, M.J., Kwok, F. (2014). *Scientific Computing, an Introduction Using Maple and Matlab*.

Wilkinson, J.H., Reinsch, C. (1971). *Linear Algebra*. Springer. Part of the book series *Grundlehren der mathematischen Wissenschaften (GL, volume 186)*.

Wirth, N. (1976). *Algorithms + Data Structures = Programs*. Prentice-Hall, INC.