Snapshots of modern mathematics
from Oberwolfach

№ 21/2019

Is it possible to predict the far future before the near future is known accurately?

Martin J. Gander

It has always been the dream of mankind to predict the future. If the future is governed by laws of physics, like in the case of the weather, one can try to make a model, solve the associated equations, and thus predict the future. However, to make accurate predictions can require extremely large amounts of computation. If we need seven days to compute a prediction for the weather tomorrow and the day after tomorrow, the prediction arrives too late and is thus not a prediction any more. Although it may seem improbable, with the advent of powerful computers with many parallel processors, it is possible to compute a prediction for tomorrow and the day after tomorrow simultaneously. We describe a mathematical algorithm which is designed to achieve this.

1 Introduction

Many time-dependent phenomena in nature, science, and engineering can be described by systems of time-dependent *differential equations*. We will see a concrete example below, but generally speaking, a differential equation is an equation that describes how a rate of change (the derivative) in one (or more) variable is related to that or other variables. They arise naturally in many applications; typical examples are weather and climate prediction, the computation of orbital trajectories of spacecraft, the traveling of wireless signals, the blood flow in the human body, flooding and more general catastrophes in nature, and there are many more. These equations are often too complex to permit exact solutions, and numerical approximations are used to better understand and predict such phenomena. Methods to obtain numerical approximations start from an initial condition, for example the current status of the weather, given by the current pressure, temperature, and wind direction, and then compute a prediction proceeding step by step into the future, computing the evolution of the pressure, the temperature, and the wind directions over the coming minutes, hours, and days. If these computations take longer than the actual weather evolution, the prediction becomes useless, so weather prediction, and several of the other applications mentioned above, are real-time problems that have to be solved rapidly.

Until 2004, the processors we use in our computers doubled their computing speed every 18 month, a fact known as Moore's law[1]. However, since 2004, computers are only getting faster by using more and more processors; the speed of each processor is not increasing any more because our current processor technology has reached its physical limits. We therefore now have multicore processing units in our computers and smart phones, and this trend is even more accentuated in today's supercomputers. Supercomputers worldwide are ranked every 6 month on the TOP500 webpage (https://www.top500.org/). In November 2018, the fastest computer was an IBM at Oak Ridge National Laboratory in the USA, with 2,397,824 processing cores, and the third fastest, a supercomputer in China that was the fastest in the world previously, has 10,649,600 processing cores. Naturally the question arises of how to use so many processors to simulate time-dependent phenomena, and in particular real-time phenomena like the weather, as fast as possible. One idea is to compute such phenomena by using some processors to compute the near future and simultaneously other processors to compute the far future, so that one gets the entire solution in time at once.

This idea seems unlikely to work. Consider, for instance, a space craft sent from the earth into an orbit around the moon. How should one compute the trajectory around the moon before knowing exactly where the orbit will start? The purpose of this snapshot is to explain one of the many methods that have been invented over the past five decades to do precisely such a "time parallel

[1] Moore's law is named after Gordon Moore, the co-founder of Fairchild Semiconductor and CEO of Intel, who described in a 4 page manuscript [6] a doubling every year in the number of components per integrated circuit.

time integration". For a complete review of all such mathematical techniques up to 2015, see [1].

To explain this time parallel method, it is best to work with a simple model problem, described by the system of ordinary differential equations

$$\frac{du(t)}{dt} = f(u(t)), \quad t \in (0,T), \quad u(0) = u_0, \tag{1}$$

where the function f is a given function describing the physical phenomenon we are interested in, and $[0,T]$ is a given time interval. For example, if we are interested in cellular growth, and there is unlimited food supply, such as when an embryo begins to form, each cell partitions itself into two new cells, which means that we obtain from one cell two, then from two four, and from four eight, and so on. Each new generation of cells is twice the previous one. The change in the cell population is proportional to its size, and this can be modeled by choosing in (1) for the right hand side $f(u(t)) := au(t)$, where a models the growth rate. Now if initially we have u_0 cells, the function $u(t) = e^{at}u_0$ is a solution of (1), as one can see by checking first that $u(0) = e^0 u_0 = u_0$, that is, that the solution matches the initial condition, and then by computing the derivative in time of the solution:

$$\frac{du(t)}{dt} = \frac{d}{dt}(e^{at}u_0) = ae^{at}u_0 = au(t).$$

To obtain the cell division solution example, which doubles the amount of cells every integer time instant, we can start with $u_0 = 1$, which gives $u(0) = 1$, and then choose $a = \ln 2$, which gives $u(1) = e^{\ln 2}u_0 = 2 \cdot 1 = 2$, $u(2) = e^{\ln 2 \cdot 2}u_0 = 2^2 \cdot 1 = 4$, $u(3) = e^{\ln 2 \cdot 3}u_0 = 2^3 \cdot 1 = 8$, and so on. For other phenomena, like the computation of the trajectory of a space craft, or weather prediction, the right hand side f is more complicated, and it is in general not possible to determine the solution $u(t)$ as a "closed form" function of time.

Leonhard Euler (1707–1783) already invented a method to obtain numerical approximations to the solution of (1), which is now called the Forward Euler method: one partitions the time interval $(0,T)$ into M small time steps $\Delta t = T/M$, and then computes at the time points $t_m := m\Delta t$, where m is an integer smaller than M, approximate values $u_m \approx u(t_m)$ of the solution by approximating the time derivative in (1) by a finite difference,

$$\frac{du(t_m)}{dt} \approx \frac{u(t_{m+1}) - u(t_m)}{\Delta t}, \tag{2}$$

which is quite natural, since if one lets Δt go to zero, this limit actually defines the derivative. Replacing the derivative in the differential equation (1) by the approximation (2), we obtain at the discrete time points the relation

$$\frac{u_{m+1} - u_m}{\Delta t} = f(u_m). \tag{3}$$

Starting with the initial condition u_0, we can compute successively the approximation u_{m+1} at the next time point t_{m+1} from the approximation u_m at the time point t_m, by writing the Forward Euler method solved for u_{m+1},

$$u_{m+1} = u_m + \Delta t f(u_m). \tag{4}$$

This clearly shows that solving problems of the form (1) is a sequential process, to compute u_{m+1} one needs to know u_m, and it is difficult to imagine how one could compute a good approximation of u_{m+1} before knowing u_m. Also more sophisticated numerical methods than the Forward Euler method have this sequential nature, since the underlying evolution problem (1) itself has this property: one needs to know $u(t)$ to compute a solution $u(t_1)$ at a later time $t_1 > t$.

2 The Parareal Algorithm

We now describe a time parallel time integration method which tries to break the sequential nature of the evolution problem (1) and its approximation (4), namely the parareal algorithm. The parareal algorithm was invented by Lions, Maday and Turinici in [5], and the authors give explicitly the reason for their invention:

> "Elle a pour principale motivation les problèmes en temps réel, d'où la terminologie proposée de pararéel.[2]"

The purpose of the method is thus encoded in the name the authors gave to their algorithm: parareal for parallel computing for real time applications. To define the parareal algorithm for solving problems of the form (1), we need two ingredients:

1. A "coarse" solver $G(T_2, T_1, u_1)$. This solver should approximate the solution $u(T_2)$ of the differential equation in (1) for a given starting value u_1 at time T_1 in an inexpensive way that can be quite inaccurate, hence the name[3]. One could for example compute one (or a few) Forward Euler steps in the time interval (T_1, T_2) to do this.
2. A "fine" solver $F(T_2, T_1, u_1)$. This solver should also approximate the solution $u(T_2)$ of the differential equation in (1) for a given starting value u_1 at time T_1, and it can and should be computationally expensive, because it should give a high accuracy approximation. One could for example compute

[2] The main motivation is to solve problems in real time, which explains the name parareal we propose for the method.

[3] The "G" here stands for "grossier", which is French for "coarse".

a large number of Forward Euler steps in the time interval (T_1, T_2) to do this.

The parareal algorithm is based on a decomposition of the time interval $(0, T)$ of interest into N smaller time intervals of size ΔT, $T_n := n\Delta T$. It starts by computing an initial approximation $U_n^0 \approx u(T_n)$ to the solution using the coarse time integrator,

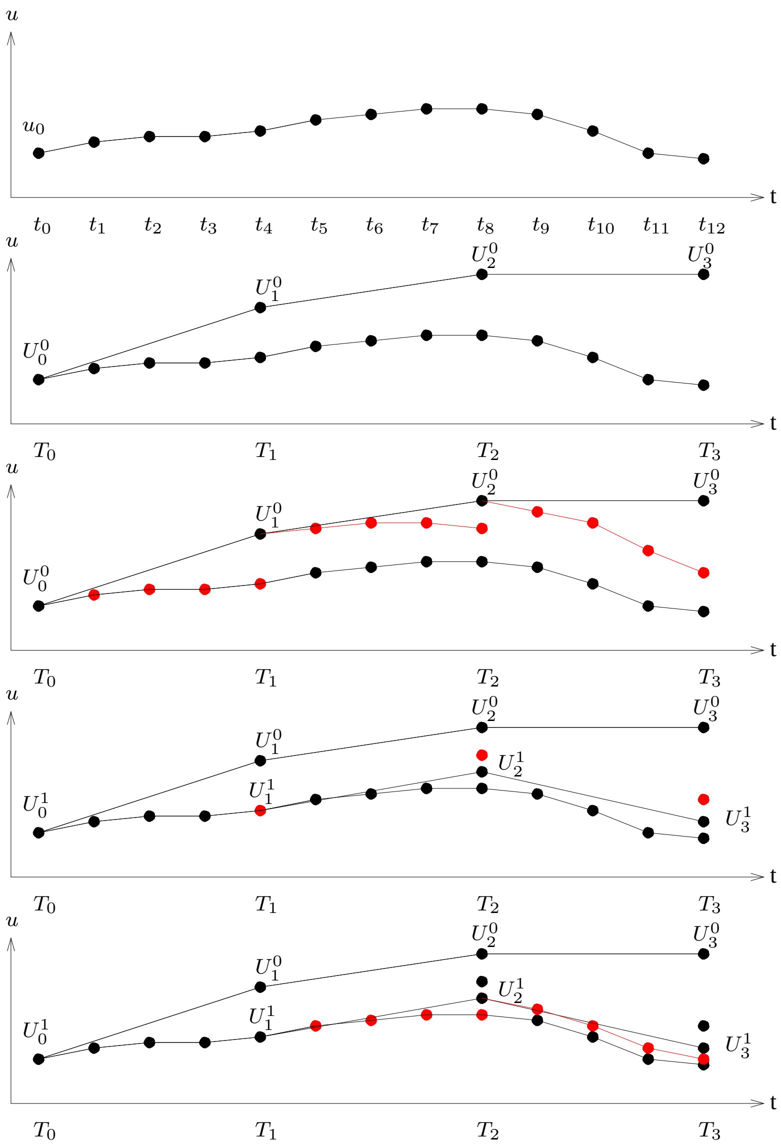
$$U_{n+1}^0 = G(T_{n+1}, T_n, U_n^0), \quad U_0^0 = u_0, \quad \text{for} \quad n = 0, \ldots, N-1. \tag{5}$$

It then corrects this approximation by iteration, computing for iteration index $k = 0, 1, 2, \ldots$

$$U_{n+1}^{k+1} = F(T_{n+1}, T_n, U_n^k) + G(T_{n+1}, T_n, U_n^{k+1}) - G(T_{n+1}, T_n, U_n^k). \tag{6}$$

There are two important observations: the first one is that since at iteration step k all approximate values U_n^k are known, one can compute all the fine solutions $F(T_{n+1}, T_n, U_n^k)$ in parallel using N processors, that is, one computes simultaneously on all time intervals, both in the near and far future, a new approximate fine solution. Let us reiterate that the fine solver in each time step ΔT will use a large number of smaller time intervals, say of size $\Delta T/1000$, to compute the new approximation (whereas the coarse solver will perhaps use ΔT itself, or a small number of subdivisions). The second observation is that once one has all these fine solutions, to obtain the new approximation U_{n+1}^{k+1}, one needs to know U_n^{k+1} for the first coarse evaluation $G(T_{n+1}, T_n, U_n^{k+1})$ on the right in (6) (the second one is known from the previous iteration). So this second step is sequential again from T_n to T_{n+1}, but it only involves the coarse and inexpensive G propagator, and is thus much faster than solving the problem with the fine propagator.

We show in Figure 1 an illustration of how the parareal algorithm approximates a numerical solution in one spatial dimension. We see an interesting property of the parareal algorithm: in the first iteration, the approximation of the parareal algorithm becomes identical to the fine solution on the first coarse time interval $(0, T_1)$, since the fine solver F starts at the correct initial condition. In the second iteration, the approximation of the parareal algorithm becomes identical also to the fine solution on the second coarse time interval (T_1, T_2), since the first iteration provided already the fine accuracy on the first coarse time interval $(0, T_1)$, and by induction, this process continues. So the parareal algorithm will converge to the fine accuracy at the latest after $k = N$ iterations. This is however too late, since then we will have performed N times, once in each iteration, a fine solution in parallel using N processors on all coarse time intervals (T_n, T_{n+1}), and it would have taken the same time to do the fine solution sequentially starting with the first time interval $(0, T_1)$, then the



u
u_0
t
t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10} t_{11} t_{12}
u
U_0^0 U_1^0 U_2^0 U_3^0
t
T_0 T_1 T_2 T_3
u
U_0^0 U_1^0 U_2^0 U_3^0
t
T_0 T_1 T_2 T_3
u
U_0^1 U_1^0 U_1^1 U_2^0 U_2^1 U_3^0 U_3^1
t
T_0 T_1 T_2 T_3
u
U_0^1 U_1^0 U_1^1 U_2^0 U_2^1 U_3^0 U_3^1
t
T_0 T_1 T_2 T_3

Figure 1: Illustration of how the parareal algorithm approximates the solution of a one dimensional evolution problem. Top: approximate solution to be computed. Second line: initial coarse approximation computed by parareal. Third line: fine corrections computed in parallel in red. Fourth line: completion of the first parareal correction step using the coarse approximation sequentially. Last line: second fine corrections computed in parallel in red.

second time interval (T_1, T_2), and so on, until reaching the last time interval (T_{N-1}, T_N), and this using only one processor, so no extra speed would have been obtained. In order to be useful, the parareal algorithm must therefore converge to the fine accuracy for a number of iterations k much smaller than N. We illustrate that indeed this can happen in the next section.

3 Numerical experiment: an Arenstorf orbit

Arenstorf orbits are stable orbits of a space craft to travel between the earth and the moon, named after the mathematician Richard F. Arenstorf (1929–2014). This is a "three-body problem", where the three bodies are the earth, the moon and the spacecraft, and for these problems a general analytic solution cannot be found. Arenstorf worked for Nasa during the Apollo program and found such a stable orbit in the shape of a figure eight, which was the basis for the lunar landing program. He also found an emergency rescue orbit which was used to bring Apollo 13 back home after the incident that forced them to abort the moon landing. Assuming that the trajectory lies in a plane, the equations of motion for the coordinates $x(t)$ and $y(t)$ of the spacecraft are (see [4])

$$\frac{dx}{dt} = x + 2\dot{y} - b\frac{x+a}{D_1} - a\frac{x-b}{D_2}, \quad \frac{dy}{dt} = y - 2\dot{x} - b\frac{y}{D_1} - a\frac{y}{D_2},$$

where D_j, $j = 1, 2$ are the functions of x and y given by

$$D_1 = ((x+a)^2 + y^2)^{\frac{3}{2}}, \; D_2 = ((x-b)^2 + y^2)^{\frac{3}{2}}.$$

If the parameters are $a = 0.012277471$ and $b = 1 - a$, and the initial conditions are chosen to be $x(0) = 0.994$, $\dot{x} = 0$, $y(0) = 0$, $\dot{y}(0) = -2.00158510637908$, then the solution is the nice closed orbit shown in Figure 2 with period $T = 17.06521656015796$, see [4]. Four years before the invention of the parareal algorithm, Saha, Stadel and Tremaine had already developed a time-parallel method to compute planetary orbits of our solar system. Their algorithm has the same structure as the parareal algorithm, except that instead of the coarse integrator G they used a simpler model, namely, the interaction of each planet with the sun only, neglecting the interaction with the other planets, see [7]. We show now how the parareal algorithm can be used to compute the Arenstorf orbit in Figure 2. We use a classical "fourth order Runge-Kutta method", which is essentially similar to Euler's method outlined above, but it uses a weighted average of four increments instead of just one at each step and is thus more accurate. For the time steps we choose $\Delta T = \frac{T}{250}$ for the coarse integrator G, which means we can use 250 processors in parallel, and $\Delta t = \frac{T}{80000}$ for the fine integrator F, such that the fine trajectory has an accuracy of 9.98×10^{-6}, as in

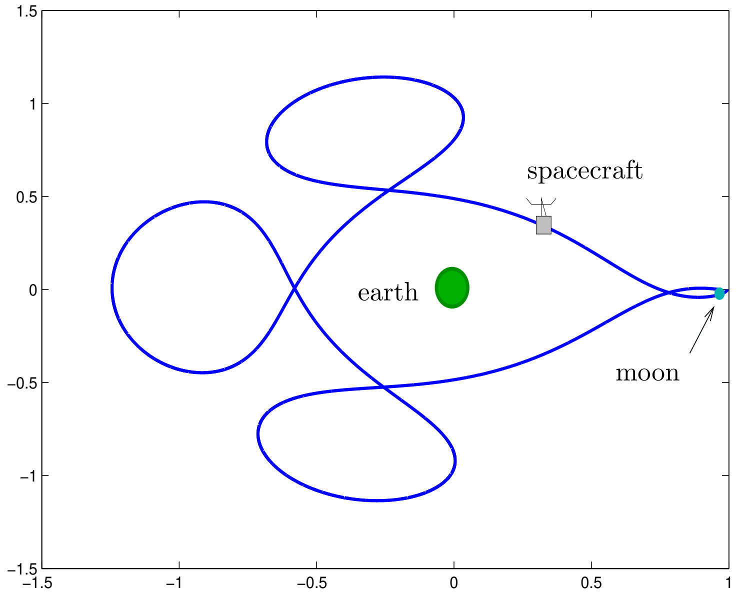


Figure 2: A so called Arenstorf orbit of a spacecraft.

[2]. We show in Figure 3 on the left the initial approximation and the first two iterations of the parareal algorithm computing approximations of the Arenstorf orbit. We see that the initial coarse approximation is completely incorrect and shows an approximate trajectory with the blue circles which spirals outward (note the difference in scale to the figures below). The first parareal correction that needed one parallel fine solve on 250 processors and one sequential coarse solve shows a corrected blue trajectory which is already much closer to the accurate Arenstorf orbit shown in green and computed sequentially with the fine solver. After the second parareal correction, we can not distinguish the parareal approximation any more from the accurate Arenstorf orbit. On the right in Figure 3, we show the error in each coordinate of the initial approximation at the top and the first two iterations of parareal below. We see that after the second parareal iteration, the approximation obtained has an accuracy of about 10^{-4} throughout the time interval of computation. We also show both on the left (with a red circle) and right (with a red vertical bar) how far one could have computed the accurate trajectory using only one processor in the same amount of time, and we see that the parareal algorithm does indeed not need to iterate $k = N = 250$ times to reach the fine accuracy in this example, two iterations suffice! One can thus compute this trajectory with 250 processors 125 times

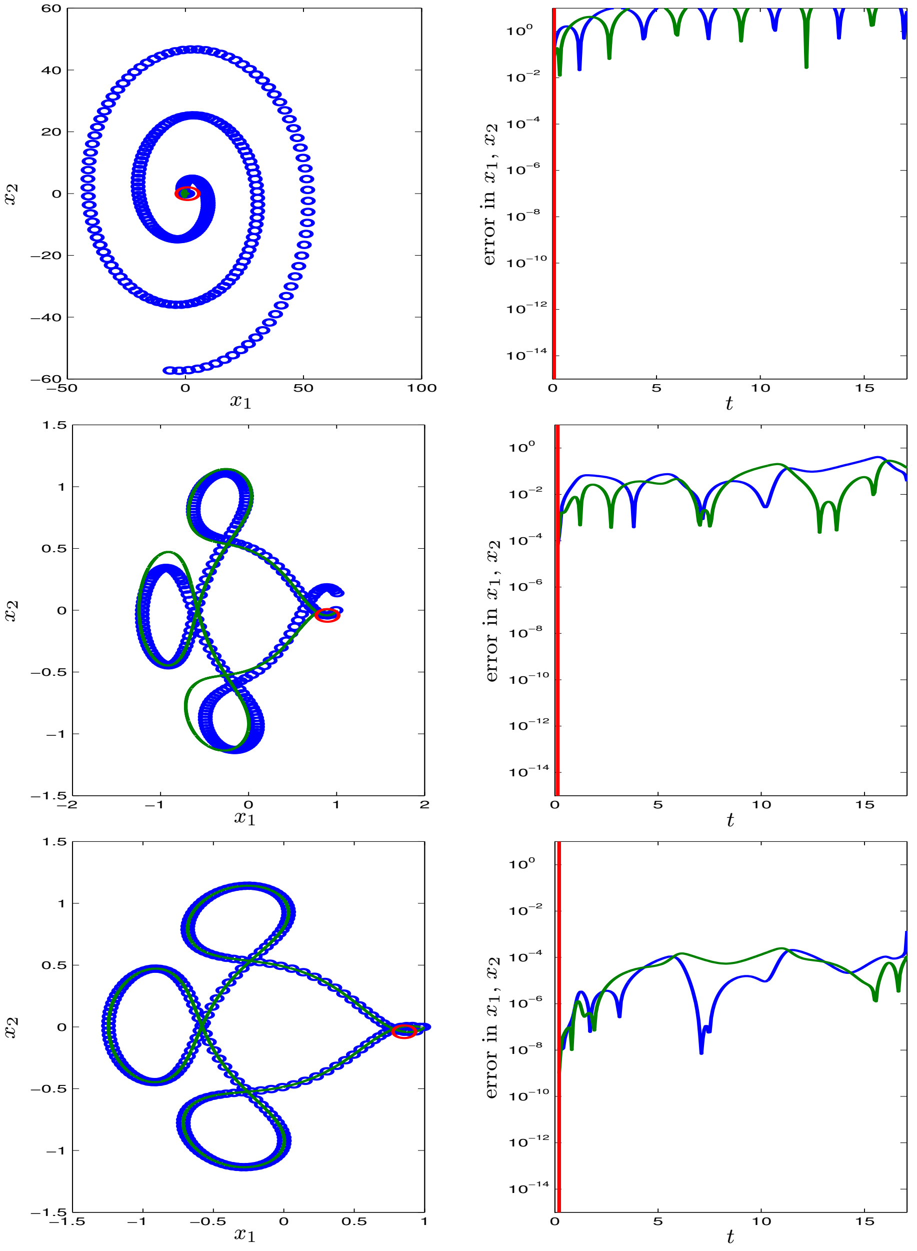


Figure 3: Parareal to compute the Arenstorf orbit of a space craft. Left: initial coarse trajectory (top) and first two parareal iterations below. Right: error in the two coordinates of the initial coarse trajectory (top) and error in the first two parareal iterations below.

faster than with one processor. Computing rapidly in this way the emergency trajectory to come back home would certainly have been of interest to the Apollo 13 mission. This shows that one can indeed predict for this problem the far future before the near future is known accurately. For a detailed convergence analysis of the parareal algorithm see [3, 2].

4 Conclusions

We explained one time parallel time integration method, the parareal algorithm, and we showed that with this algorithm, it is indeed possible to compute an approximation of a time dependent problem using many processors faster by parallelization in time than when using only one processor solving the problem sequentially. This is surprising, because one has at first the feeling it is not possible to predict the far future before the near future is known accurately.

The parareal algorithm is however only one among many different techniques to predict the far future and the near future simultaneously. Among the iterative methods doing this are multiple shooting methods in time of which the parareal algorithm is a variant, space-time domain decomposition methods, and also multigrid type methods, of which again the parareal algorithm is a variant with so called aggressive coarsening. And there are also direct time parallel methods, which do not even iterate, like the ParaExp method, the method based on diagonalization of the time stepping matrix, and Revisionist Integral Deferred Correction (RIDC). An overview of all these methods can be found in [1].

5 For further reading

The snapshot 9/2017 *Computing the long term evolution of the solar system with geometric numerical integrators* by S.F. Vilmart and G. Vilmart (https://publications.mfo.de/handle/mfo/1355) might be of interest to the reader, as it also concerns numerical methods for solving differential equations, but with a view towards long term solutions, rather than obtaining good-quality predictions of a system in as short a time as possible.

References

[1] M. J. Gander, *50 years of time parallel time integration*, Multiple Shooting and Time Domain Decomposition Methods, Springer, 2015, pp. 69–113.

[2] M. J. Gander and E. Hairer, *Nonlinear convergence analysis for the parareal algorithm*, Domain Decomposition Methods in Science and Engineering XVII

(O. B. Widlund and D. E. Keyes, eds.), Lecture Notes in Computational Science and Engineering, vol. 60, Springer, 2008, pp. 45–56.

[3] M. J. Gander and S. Vandewalle, *Analysis of the parareal time-parallel time-integration method*, SIAM Journal on Scientific Computing **29** (2007), no. 2, 556–578.

[4] E. Hairer, S. P. Nørsett, and G. Wanner, *Solving ordinary differential equations I: Nonstiff problems*, Springer-Verlag, Second Revised Edition, 1993.

[5] J.-L. Lions, Y. Maday, and G. Turinici, *Résolution d'EDP par un schéma en temps 'pararéel'*, Comptes Rendus de l'Académie des Sciences-Series I-Mathematics **332** (2001), no. 7, 661–668.

[6] G. E. Moore, *Cramming more components onto integrated circuits*, Electronics **38** (1965), no. 8.

[7] P. Saha, J. Stadel, and S. Tremaine, *A parallel integration method for solar system dynamics*, Astron. J. **114** (1997), 409–415.

Martin J. Gander *is a professor of mathematics at the University of Geneva.*

Mathematical subjects
Numerics and Scientific Computing

Connections to other fields
Computer Science, Engineering and Technology

DOI
10.14760/SNAP-2019-021-EN

ISSN 2626-1995