Snapshots of modern mathematics
from Oberwolfach

№ 11/2017

Mathematics plays a key role in scientific computing

Chi-Wang Shu

I attended a very interesting workshop at the research center MFO in Oberwolfach on "Recent Developments in the Numerics of Nonlinear Hyperbolic Conservation Laws". The title sounds a bit technical, but in plain language we could say: The theme is to survey recent research concerning how mathematics is used to study numerical algorithms involving a special class of equations. These equations arise from computer simulations to solve application problems including those in aerospace engineering, automobile design, and electromagnetic waves in communications as examples. This topic belongs to the general research area called "scientific computing".

1 Introduction

Scientific computing is a research field which deals with applying mathematical tools to design efficient algorithms that are used in computer simulations. As such, it is closely related to computers and the very subject would not exist without the appearance of modern computers. However, the emphasis of scientific computing is very different from that of computer science. Scientific computing emphasizes the role of mathematics to ensure accuracy, stability, and efficiency of numerical algorithms. Computations by computers are, by their nature, limited by the physical resources of the computer (memory, processing power), and so many times they are approximations of the real systems and so contain approximation errors. The goal is to design algorithms which approximate the system as closely as possible in an efficient way.

The efficiency of an algorithm is measured as the amount of operations required depending on the size of the input: the cost of the computation. An exact amount is not always an easily-presented function of the input and will also depend on the implementation of an algorithm, the configuration of the input data, the desired level of accuracy, and many other factors. Instead, we will describe the efficiency of a computation in terms of the *order of cost*. Assume we measure the size of the input to be some number n, and $f(n)$ is some function of n. We will write that the order of the cost for the algorithm is $O(f(n))$ if the amount of operations required by the algorithm for an input of size n is less than $f(n)$ multiplied by some constant which is independent of n.

The role of scientific computing is demonstrated in the following three examples. To fully describe these mathematical methods requires preparation well beyond the scope of this text, so many details will be ignored and only the main idea will be presented. More details can be found in the bibliography.

2 Fast Fourier Transform

One very well-developed and useful mathematical tool is *Fourier analysis*.[1] This is a method (or, rather, a collection of methods) to represent a function of time (a signal) as a combination of periodic functions such as sine and cosine. This presentation is called the *Fourier transform* of the original function. The Fourier transform is now a function of frequencies: it measures the amplitude and phase (as periodic trigonometric functions) of each frequency contributing to the original function. The original function can be reconstructed by a reverse operation.

Besides the obvious property of allowing insight into the periodic components of a function, Fourier analysis simplifies computations in many cases, especially those involving derivatives and integrals. Therefore, Fourier analysis has wide-ranging applications: the analysis and manipulation of sound, images, seismic waves, radio waves, stock-price fluctuation, differential equations, encryption, and many more.

To illustrate this procedure, let us use an example. Assume we want to digitally record sound for a duration of one second. We use a digital microphone which samples the input sound N times a second, to attain the

[1] Fourier analysis is named after the French mathematician Joseph Fourier (1768–1830) who introduced this method in his work on heat transfer.

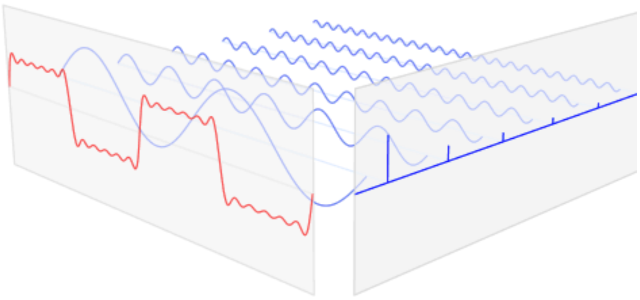

Figure 1: A function of amplitude over time (left face) transformed into a function of amplitude over frequency (right face).

samples $x_0, \ldots, x_{N-1}$, with x_n being the amplitude of the sound wave at time n/N. We should note that since we are dealing with discrete samples, as opposed to a continuous function, we focus throughout this section on a variant of Fourier analysis called *discrete Fourier transform*.

If we want to remove frequencies which contribute very little, or are beyond the perception of the human ear – in order to reduce the size of the data file – then we can perform the surprisingly simple substitution

$$y_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi ikn}{N}}, \tag{1}$$

to get N complex numbers $y_0, \ldots, y_{N-1}$, which can be considered as samples of the Fourier transform of the function describing the sound. Noting that

$$e^{-2\pi ikn/N} = \cos\Big(-2\pi k\frac{n}{N}\Big) + i\cdot\sin\Big(-2\pi k\frac{n}{N}\Big)$$

we can affirm that each y_k is, indeed, a combination of two periodic functions with period N.

After we perform the summation in Equation (1) for each $k = 0, \ldots, N-1$, and have the sound samples in terms of frequencies, we can remove the desired frequencies. There is a similar explicit formula for the backward computation of x_n in Equation (1) when the y_k's are given. Clearly, the cost (number of operations) of this computation is $O(N)$ for each k, hence the total cost is $O(N^2)$ for all the k's.

When N is very large (for example, $N = 5{,}000$ or more for certain signal analysis and image processing problems), there are two problems: (1) The computational cost to perform the Fourier transform is very high; (2) The accumulation of round-off errors is significant, so the computed results are not very accurate.

Here is where the scientific computing comes into play. The *fast Fourier transform* (FFT) is a mathematical reformulation of Equation (1) which takes advantage of the periodic nature of the functions to regroup terms such that they can be repeatedly used, and so one can obtain the values of y_k *for all* $k = 0, 1, \ldots, N-1$ in $O(N \cdot \log N)$ cost. Let us see how the most famous algorithm for the FFT – the *Cooley–Tukey Algorithm* – does that.

We start by dividing the right-hand sum in Equation (1) into two sums (assume that N is even): first we sum the even-indexed terms $n = 2m$ (for $m = 0, \ldots, N/2-1$) and then the odd-indexed terms $n = 2m+1$. Now, for each y_k with $k < N/2$ we have two sums that very much resemble two transforms of $N/2$ samples (maybe multiplied by a constant). This is interesting, but doesn't seem to solve the problem: we still have the same amount of computations, and we don't know how to compute the y_k's for $k \geq N/2$. Here the periodic nature of the functions comes into play: it appears that the terms for every $k < N/2$ differ from the terms for $k + N/2$ (a period apart) only by a predetermined coefficient.[2] Thus we can start by computing only transforms with half the samples, and reuse the results for the y_k's with $k \geq N/2$. If we repeat this process, then we can obtain the complete answer with the promised cost of computation $O(N \cdot \log N)$.

From the above description, we see that the FFT is more efficient if N is a product of small primes, for example, when $N = 2^n$. In this case, we compute the transform of only very few terms and reuse it to get the transform of double the terms, and then double that, and so forth. Even for N as large as 5,000, $\log N$ is very small. The savings in computational cost is thus very significant — a job which required thousands of minutes can now be completed in a few minutes. Also, the round-off error accumulation is significantly reduced.

We remark that FFT and the straightforward computation of performing the summing of Equation (1) for each y_k, give mathematically identical results. There is no approximation involved here. We made the exact same computation in a different order and more efficiently. Mathematical methods help us in this case to get the same result much faster and more accurately. The original research paper on FFT [3] by Cooley and Tukey, was motivated by the desire to quickly analyze seismic signals, in order to determine whether the USSR was conducting nuclear tests. It was published in the American Mathematical Society (AMS) journal "Mathematics of Computation" and is one of the most often downloaded papers of that journal.

[2] Try it for yourself: start with Equation (1) and follow the described procedure. Perform the calculations and see if you can really use only half of the terms. What coefficients are added?

3 Fast Multipole Method

Another classical, but more recent, example for the crucial role of mathematics in the design of important algorithms, is the so-called “fast multipole method”. This fast method has the same spirit as the fast Fourier transform, and it aims at certain types of dense systems of interactions. For example, we may have N particles located at the points x_n, $n = 0, 1, \ldots, N-1$ in the three-dimensional Euclidean space $\mathbb{R}^3$. Each particle interacts with each other and the interaction strength could be inversely proportional to their distance $|x_i - x_j|$ or its square (for example, as it is for gravity). Thus, to compute the movement (due to gravitational field of the other particles) of each particle, we would need to sum up the interaction of this particle with every other particle, resulting in a calculation of order $O(N)$. The total cost of computing the movement of all N particles is thus $O(N^2)$, which is very large as we have discussed before.

This time, the structure of the problem is less apparent than that of the Fourier sum to group terms together to save computational cost. However, with a clever choice of grouping techniques for the particles, Greengard and Rokhlin [4] were able to obtain the *fast multipole method*, which improves the computational cost to $O(N \cdot \log N)$.

The name is derived from the computation at the heart of this method: the multipole expansion. Having the group of particles in some area of the whole space, a *multipole expansion* is a way to express their gravitational influence on a sufficiently distant point as an infinite sum. One benefit of the multipole expansion is that only few of the first terms (“poles”) in this infinite sum should be computed in order to get a quite accurate outcome. Some obstacles still remain, though: for each of the N particles we still need to compute the expansion of the other $N-1$ particles (with computational cost $O(N^2)$) and we cannot guarantee that each particle is distant enough from the rest for the expansion to work.

The fast multipole method solves this by grouping the particles into clusters that can be treated separately, and efficiently computing the interactions between them. First, all N particles are imagined to be contained in a box. This box is divided into eight smaller boxes by dividing the large box in half along each axis. Each of these smaller boxes is also divided, until we reach boxes which are small enough – this size is determined by the accuracy we wish to achieve. At each of these smallest boxes we compute the finite sum for the multipole expansion of the particles in that box (the finite number of poles depends on the desired accuracy). The cost of this process is $O(N)$ as each particle contributes to one expansion. We then move these expansions to the parent box and sum them, thus getting the expansion of all the particles in the parent box.[3] These expansions, in turn, are moved to the upper level of boxes, and so on until we have an expansion for each box at each level. These sums will be used to compute the interactions between the particles.

Starting from the upper levels (the bigger boxes), for each box we add the expansions calculated for boxes outside of its parent box (distant enough particles). These sums are moved down to the box's sub-boxes where we need to add the expansions for boxes which are distant at this level but not at the previous. This continues until, at the lowest level, we have expansions that represent the influence of all the particles distant from the smallest box. This is done while reusing the expansions calculated before. All that is left now to do is, for each particle, to compute the interaction with the other particles in its smallest box and its neighbors (not distant enough – there are only a few of these) in a direct fashion, and add it to the interactions with the distant ones obtained by the former expansions.

This technique is now widely used in solving various integral equations and interaction systems. The discovery of the fast multipole method earned Greengard and Rokhlin the shared AMS 2001 Leroy P. Steele Prize for a Seminal Contribution to Research.

4 Multigrid

We now give another example demonstrating the power of mathematics to obtain efficient algorithms, the so-called multigrid method for solving linear systems.

In principle, we know how to solve a linear system

$$Ax = b,$$

where A is an $N \times N$ matrix. We can use, for example, the well-known Cramer's rule. However, a direct implementation of Cramer's rule involves computing the determinants of N matrices of size $N \times N$ each – an operation cost on the order of $O(N!)$ (N factorial). This is of course not feasible for large N, for example, $N = 1$ million. Such a computation is routine in today's computer simulations: it corresponds to a discretization of three dimensional partial differential equations on a mesh of $100 \times 100 \times 100$ grid points; in this case $N!$ is a number with more than 5 million digits.

[3] This is actually not simple, as the original form of the multipole expansion assumes all particles are centered about the point of origin.

Fortunately, we also have the method of Gauss elimination, another well-known technique. For example, we can solve the linear system

$$\begin{aligned} 2x_1 + x_2 &= 4 \\ x_1 - x_2 &= -1 \end{aligned}$$

in the following way: If we multiply the second equation by -2, and add the first equation to it, the system becomes

$$\begin{aligned} 2x_1 + x_2 &= 4 \\ 3x_2 &= 6 \end{aligned}$$

The system is now in a so-called upper-triangular form, which is easy to solve explicitly by *back substitution*, to obtain first $x_2 = 6/3 = 2$ and then substituting this into $x_1 = (4 - x_2)/2 = 1$.

It is easy to verify that this procedure has a cost of at most $O(N^3)$ to reach the solution (and is even smaller for banded matrices[4] with many zero entries), which is much lower than the $O(N!)$ cost of applying Cramer's rule. We can also show that, with suitable "pivoting", that is, suitable interchanges of rows and columns of A (which do not change the solution), Gauss elimination can always be performed in a stable fashion whenever A is non-singular, resulting in accurate solutions in the presence of computer round-off errors.

However, in many situations, even $O(N^3)$ is an excessive cost. Storage might also be a problem to store the full matrix A, when often, in applications, there are many zeros in its entries.

Using mathematical techniques, scientists found an efficient class of algorithms to solve a certain class of linear systems with only the cost of $O(N)$. This so-called "multigrid method" [1] is clearly optimal, as one would need $O(N)$ cost simply to write down the solution! This method combines different techniques in separate steps to simplify the problem by looking at it at different scales.

First, the solution is approximated using methods that are similar to a sophisticated version of Newton's method to find roots of functions. A series of values is constructed iteratively – in which each is attained from the previous – and each is closer to the real solution. Only very few of the iterations are performed, though, as the purpose of this stage is to make the errors smaller for the next stages, known as *smoothing*.

The next stage is the heart of the method: the problem is sampled to a "coarser grid". Think of it as looking at a picture from afar and instead of details seeing only patches of the significant colors and features. This is done

[4] *Banded matrices* have non-zero entries only on a "band" along their diagonal.

by multiplying A by an appropriate matrix which reduces the dimension. For example, if at first we had N equations in N variables, now we have $N/2$ equations and variables – a so-called coarser grid. The equations and variables that are chosen for this stage are the ones that, due to their coefficients or powers, contribute the most to the final solution – as their solution will be used to solve the others. This simpler problem can now be solved directly or approximated quite easily.

Now, the solution (or its approximation) is moved back to the "fine grid". This is done by methods of *interpolation* which is like guessing the missing colors by averaging the colors of the surrounding patches. Back at this stage more smoothing is done. Quite surprisingly, this is enough. Sometimes, more than just one grid is used, but very few iterations of the processes involved are enough to solve the problem or approximate the solution to a reasonable error for the practical needs at hand.

Because of his seminal contribution to the design of multigrid methods, Brandt was awarded the 2005 Society for Industrial and Applied Mathematics/Association for Computing Machinery (SIAM/ACM) Prize in Computational Science and Engineering. The fact that both SIAM (a society of applied mathematicians) and ACM (a society of computer scientists) award this joint prize to Brandt indicates that both applied mathematicians and computer scientists view highly the important role of mathematics in scientific computing.

Since the appearance of electronic computers, there has been the well-known Moore's law, stating that computer power (speed and also memory) will increase exponentially, doubling every few years. Perhaps less well-known is a similar "Moore's law" describing the improvement of algorithm efficiency, obtained by using mathematics on algorithm design and analysis. For example, the improvement in computational cost from $O(N^2)$ (Gauss elimination for banded matrices in two dimensions) to $O(N)$ (multigrid method) for solving linear systems, is very large: with $N = 10^4$ (a two-dimensional grid of 100×100), the improvement is already a factor of 10^4. This saving becomes even bigger in three and higher dimensions.

5 Algorithms for nonlinear hyperbolic conservation laws

Now let us come back to the theme of the recent workshop at the MFO in Oberwolfach. The discussion in this section involves more advanced mathematics and hence is given in a more sketchy fashion, but the general ideas are the same as in the previous section, namely trying to use mathematical tools to help design and analyze efficient and reliable algorithms for applications.

The specific class of problems in this workshop was the class of nonlinear hyperbolic conservation laws. These are partial differential equations (PDEs) [5] describing the physical phenomena of conservation, such as conservation of mass, conservation of momentum, conservation of energy, conservation of the total number of cars in traffic flows, and more; See, for example, [7]. These PDEs have the inconvenient property that their solutions may become discontinuous, even if the initial and the boundary conditions are smooth. For example, the pressure and density of the airflow at some distance in front of a fast flying aircraft could be discontinuous (changing abruptly, the so-called shocks). This poses a lot of difficulty in the design of efficient and reliable computer algorithms for solving such PDEs. However, tremendous progress has been made over the past decades, and several highly efficient and reliable computer algorithms have been designed and used in applications. As an example of applications, the majority of the design of Boeing 777 was performed based on computer simulations with minimal traditional wind-tunnel tests, thus saving a lot of money and time.

One could get a glimpse of such algorithms with two examples, the weighted essentially non-oscillatory finite difference and finite volume schemes [6], and the discontinuous Galerkin finite element methods [2, 5]. Many of the talks in this workshop involved these two types of algorithms. One indication of the importance of the influence of mathematics in helping the design of these algorithms and the resulting applications, is that the author of this snapshot was also awarded the SIAM/ACM Prize in Computational Science and Engineering, in 2007, for his contribution to the design and analysis of these and other efficient algorithms in applications. It can be anticipated that mathematics will continue to play an important role in scientific computing in future years.

[5] For more about partial differential equations, especially in relation to conservation laws, see Snapshot 7/2015 *Darcy's law and groundwater flow modelling* by Ben Schweizer.

Image credits

Fig. 1 Extracted frame from a public domain animated gif created by Lucas V. Barbosa which can be found at: https://commons.wikimedia.org/wiki/File:Fourier_series_and_transform.gif.

References

[1] A. Brandt, *Guide to multigrid development*, in *Multigrid Methods*, Springer-Verlag, 1982, pp. 220–312.

[2] B. Cockburn and C.-W. Shu, *Runge-Kutta Discontinuous Galerkin methods for convection-dominated problems*, Journal of Scientific Computing **16** (2001), 173–261.

[3] J. W. Cooley and J. W. Tukey, *An algorithm for the machine calculation of complex Fourier series*, Mathematics of Computation **19** (1965), 297–301.

[4] L. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, Journal of Computational Physics **73** (1987), 325–348.

[5] J. Hesthaven and T. Warburton, *Nodal Discontinuous Galerkin Methods*, Springer, New York, 2008.

[6] C.-W. Shu, *High order weighted essentially non-oscillatory schemes for convection dominated problems*, SIAM Review **51** (2009), 82–126.

[7] J. Smoller, *Shock Waves and Reaction-Diffusion Equations*, second edition, Springer-Verlag, New York, 1994.

Chi-Wang Shu *is the Theodore B. Stowell University Professor of Applied Mathematics at Brown University.*
shu@dam.brown.edu

Mathematical subjects
Numerics and Scientific Computing

Connections to other fields
Chemistry and Earth Science, Computer Science, Engineering and Technology, Physics

DOI
10.14760/SNAP-2017-011-EN

Junior Editor
Daniel Kronberg
junior-editors@mfo.de

Senior Editor
Carla Cederbaum
senior-editor@mfo.de

Mathematisches Forschungsinstitut
Oberwolfach gGmbH
Schwarzwaldstr. 9 – 11
77709 Oberwolfach
Germany

Director
Gerhard Huisken