The 2008 Questions of the Month

May 2008 (first Question of the Month ever)

Question:

Who coined the name "Finite Element Method" and when?

Answer:

The terms "finite element" and "finite element method" are thought to have been coined around 1956 by Raymond W. Clough, a professor at the University of California at Berkeley. The first publication to actually use the term "finite element" was Clough's paper "The Finite Element Method in Plane Stress Analysis," which was presented at the Proceedings of the 2nd annual ASCE Conference on Electronic Computation in 1960. However, Clough apparently used this term in his lectures already since 1956. (By the way, the question on coining the term FEM and the question on the invention of FEM are two different questions! The latter will be reserved for a future Question of the Month...)

ANSWERED CORRECTLY: Elad Priel, Rami Ben-Zvi, Shmuel Vigdergauz, Slava Krylov, Yoav Ophir, Ran Ganel, Alex Yakhot, Nicolae Gluck, Amiel Herszage.

June 2008

Question:

The acronym CFL: What does it mean in the context of CM (just one or two lines please), who discovered or invented it and when?

Answer:

The "CFL Condition" is a condition needed on the maximum value of the ratio of time step to spatial cell/element size (Delta t / Delta x) to guarantee the numerical stability of the given scheme. The acronym CFL stands for Courant, Friedrichs and Lewy, three well-known applied mathematicians who discovered this stability condition in their celebrated paper in 1928.

By the way, the famous applied mathematician Peter Lax (who is still very active today) has been heard to "protest" that the "young generation" thought that CFL stood for Courant-Friedrichs-Lax.

ANSWERED CORRECTLY: Anne Weill, Michel Becovier, Matania Ben-Artzi, Amiel Herszage, Rami Ben-Zvi, Yoav Ophir, Ido Gur.

July 2008

Question:

This time it's a special challenge. In the context of CM, everybody knows that a problem in 3D is almost always more difficult to solve than its counterpart in 2D. Give an example for a problem which, in a certain respect, is more complicated to solve in 2D than in 3D, and explain why.

Answer:

I received 3 different (and correct) answers from 4 readers. I outline all the answers below.

A. [Answered: Eli Turkel, Slava Krylov; and this solution is what I had in mind when I asked the question.] Wave Propagation. In a sense, 2D wave propagation is more complicated than 3D wave propagation. This manifests itself in the computation of wave propagation problems. In 3D, a wave pulse emanating from a source will propagate and leave the region where it started without leaving any trace. On the other hand, in 2D a wave will leave endless ripples after it. Consider, for example, a stone thrown into a still lake. It will create waves on the surface of the water (which are 2D waves), and if you watch these waves carefully you will see many small ripples that continue to be present long after the head front left this local region. Mathematically, this has to do with the more complicated behavior of the Green's function of the wave operator in 2D compared to 3D. This 2D behavior has implications in various wave-related numerical schemes; Absorbing Boundary Conditions is one such type of schemes.

B. [Answered: Matania Ben-Artzi.] Potential Problems in Unbounded Domains. This answer is somewhat related to the previous one. The fundamental solution of the Laplacian blows up (like log r) in 2D, and decays in 3D. Hence, any potential problem, where behavior at infinity should be taken into account, poses specific problems in 2-D, which manifest themselves in numerical schemes. Example: Potential (i.e., irrotational) flow in an exterior domain.

C. [Answered: Amiel Herszage.] Elastic Shells. Problems of thin elastic shells may be regarded as 2D in the sense that once certain assumptions are introduced on the shell behavior, and through-the-thickness integration is performed, the problem is completely characterized by the solution on the mid-surface of the shell. The 3D counterpart would be a problem in a very thick "shell" of the same shape. Computationally, the treatment of the 3D problem is more straight-forward. With the Finite Element method, one can solve the 3D elasticity equations using 3D "brick elements" (also called "solid elements" in some commercial codes) which are quite robust. The thin-shell problem requires shell elements which are always "special"; in fact there is no consensus on a single shell element which is best in all cases, and one has to be very careful in evaluating the numerical results, especially when large deformation is involved.

August 2008

Question:

Another grand challenge!

We all know that a numerical process should better converge. If a numerical process does not converge, then we usually think of it as useless. Nevertheless, give an example for a numerical process which does not converge if we proceed with it further and further, but it is nevertheless useful. Practically, we stop somewhere within the process and we get a "good result".

Answer:

Only four people sent me answers this time. In fact, it is reassuring to know that there is general agreement that non-converging methods are by and large not useful... (Someone told me half jokingly that this time my question was not "educational". I guess he is right.) Having said this, here are the four correct answers:

METHODS BASED ON ASYMPTOTIC EXPANSIONS. [This was suggested in general terms by Orna Agmon Ben-Yehuda and for a specific application - field theory in physics - by Asher Yahalom.] Certain numerical methods are based on a so-called asymptotic expansion. Such an expansion arises if the problem may be associated with a small parameter (epsilon), and is written (in theory) as an infinite series. Sometimes this series converges, but in the more interesting cases (called singular perturbation problems) the series does not converge. If we base a numerical method on such a non-converging asymptotic series, the method would not converge if we take ever more terms in the series! However, what saves the day is the fact that in practice the approximate solution initially approaches the exact solution as we take more and more terms, and only from a certain term, hopefully very remote, the solution starts to "run away" from the exact solution. The Khokhma is, of course, to stop the process before this happens, and the book of Bender and Orszag gives a nice stopping criterion for this.

DOMINATION OF ROUND-OFF ERRORS. [This was suggested by Rachel Gordon in the context of iterative solvers. It also occurs in the case of direct solvers.] Imagine solving a linear problem using finite elements or finite differences, and obtaining a linear system of algebraic equations, and then solving it by Gauss Elimination. If you refine the mesh more and more the system (the matrix) will become larger, and you will get a more accurate solution. However, people are not always aware that this is true only up to a certain level of refinement! Interestingly, as we refine the mesh the discretization error decreases, but the round-off error (generated by the fact that the computer can carry a finite number of digits) increases! In other words, the matrix becomes more and more ill-conditioned. When the round-off error dominates, one cannot converge to the exact solution by simply reducing the discretization error further. In most cases one does not reach the level of refinement where round-off errors start to dominate. But if the problem is "stiff" (if you don't

know what this is, never mind) this can happen in practice. The same is true for iterative solvers. [Rachel Gordon gave GMRES as a point in case.] Still, if we stop early enough we may get a useful numerical solution.

INVERSE PROBLEMS. [Rachel Gordon suggested this, pointing into the specific application of tomography.] Inverse problems are notoriously ill-posed, and thus are very sensitive to all sorts of "noise". Therefore their numerical solution is often extremely difficult. Typically, a numerical process for solving an inverse problem will produce an approximate solution that would approach the exact solution, e.g., by taking a finer and finer discretization, but only until the level of noise inherent in the problem is reached. From that moment on the solution cannot "converge" any more. One way to look at this is algebraic. A super-sensitive problem leads to a super sensitive matrix, and such a matrix would give rise to large round-off errors, which brings us back to the previous answer.

NUMERICAL PROCESSES THAT SIMULATE NON-CONVERGING MODELS. [Slava Krylov suggested this.] Some mathematical models that are supposed to represent real behavior of certain physical, social and economical systems, are inherently "non-converging". [Slava Krylov gave Cellular Automata and Logistic Maps as examples.] Of course, a numerical simulation of such a model would also be "non-converging" in the same sense.

September 2008

Question:

This time we consider the mathematical model which is being solved via the numerical method. We are all aware of the fact that if we do not pick the appropriate model relevant to the physical problem at hand, we would not get a good solution no matter how excellent our numerical technique is.

First, please watch the fascinating movie at http://www.youtube.com/watch?v=3mclp9QmCGs showing the famous failure of the Tacoma bridge in 1940.

Now the question. Years later, when computers and NASTRAN (the finite element code) were available, someone tried to simulate the dynamic response of the bridge and obtain the failure numerically. This someone modeled the geometry of the bridge quite precisely, and applied the wind as a given load in a certain fixed direction on the bridge, taking into account the wind conditions that were measured on the day of this storm. This simulation did _not_ predict the failure of the bridge. Why not? What was wrong in the model? (Later the model was improved, and then it did predict the failure.)

Answer:

This was apparently a tough one. Only one reader, Amiel Herszage, sent me a correct answer.

It is tempting at first to think that the failure was caused by simple resonance. Namely, the wind load spectrum included load acting in one of the eigenfrequencies of the bridge (a torsional mode, as we can see in the movie, and indeed the bridge design was weak in torsion), which caused resonance. In theory and without damping, resonance leads to unbounded deformation and stresses, so it seems like a reasonable cause for failure. This simple mechanism was rejected, the main reason being that the damping that the bridge system was estimated to have would have prevented failure from resonance unless the wind load was extremely high, which was not the case on the day of the failure.

The true reason for the failure was aeroelastic flutter (PIRPUR in Hebrew), which is an effect of loss of stability known well in aeronautical engineering. To get a flutter effect, one must take into account the interaction between the structure and the fluid (wind) flow around the structure. There is a coupled mechanism: the wind flow affects the load on the bridge; this load causes the bridge to deform; the deformation of the bridge changes the flow of the wind around the bridge, the wind flow affects the load on the bridge..., and so on in an endless circle. More specifically flutter is caused by the lag of the aerodynamic forces after the "the dynamics of the bridge", in a way which effectively introduced "negative damping", which in turn causes instability.

So in short, the answer to the question is that the person trying to simulate the failure using FE analysis, should have considered a coupled model of the bridge and the fluid flow around it. With a model of the bridge alone (representing the wind as a given load) one cannot get the failure that really occurred.

October 2008

Question:

You have in your working environment an incompressible flow code (it doesn't matter what method it is based on), which is used routinely for problems in hydrodynamics. The code is truly incompressible, namely assumes given constant density and does not include any temperature effects. One day, you suspect that a certain industrial process that you are working on involves Rayleigh-Benard instability, in which temperature plays a crucial role. Rather than buying a new code, you decide to change the existing code in order to be able to simulate the Rayleigh-Benard convection. What is the easiest way to do this, namely the way which would require minimal changes in the existing code?

Answer:

The first thought that one might have is that one has to turn the code into a compressible code. After all, Rayleigh-Benard instability is caused by buoyancy effects, namely hot fluid becomes less dense and thus tends to go up (against gravity) while cold fluid becomes more dense and thus tends to go down. Thus,

density becomes variable and the fluid is not incompressible any more.

However, turning an incompressible code into a fully compressible code is a nightmare... First of all, the velocity and pressure on one hand and the temperature on the other become fully coupled (through the equation of state that connects the energy equation to the momentum equations), and one has to solve for the temperature field simultaneously with the other variables. This involves some major technical changes in the code. Second, the properties of numerical schemes (stability and accuracy) for compressible and incompressible flows are totally different, and thus the "switch" from incompressible to compressible requires a lot of caution and is far from being "automatic".

What may save the day is the so-called Boussinesq approximation, which is valid in many applications (for example, it is heavily used in crystal growth applications). It is based on approximating the change in the density due to the temperature by the first-order term in a Taylor series around the nominal density. In this case the equations to be solved are not modified at all, except for a "buoyancy forcing term" in the right-hand-side of the relevant momentum equation which is a linear function of the temperature. This has the advantage that the density remain a given constant in the governing equations. Moreover, there is no full coupling between the temperature and the other variables. Thus, we can first solve for the temperature distribution (maybe using a separate, thermal code), and then solve separately for the velocity and pressure fields using our slightly modified incompressible code.

Correct answers were obtained from (in alphabetical order):

Simon Brandon, Alex Gelfgat, Amiel Herszage, Stephane Seror, David Sidilkover, Alex Yakhot.

November 2008

Question:

Which of the following three expressions is a well-defined term in the jargon of High-Performance Computing (HPC) and what does it mean? The expressions are:

(1) Amazingly parallel
(2) Hugely parallel
(3) Embarrassingly parallel

Answer:

The only term of the three that has become a standard term in the jargon of High Performance Computing (HPC) is "embarrassingly parallel", that describes an algorithm which can be implemented in a completely parallel way. In the words of Michel Bercovier, "This designates parallel numerical algorithms that have little or no data exchanges and shared memory requirements, and a very

loose synchronisatin requirement. They can thus be run not only on clusters but on Grids in a very effective way. Usually they are also easy to implement. The best example is given by Monte Carlo methods for the solution of stochastic ODEs or PDEs."

Correct answers to this questions were given by (in alphabetical order):

Orna Agmon Ben-Yehuda, Michel Bercovier, Simon Brandon, Mahmood Jabareen, Stephane Seror, Jonathan Tal, Shaul Tayeb, Anne Weill.

December 2008

Question:

Here is a real story. A smart engineer called Tamar (Shem Baduy) had a code that solved numerically a class of problems governed by PDEs. This was a finite element code, but in fact the question is quite general and applies to many other types of discretization methods. The code involved evaluation of some integrals which was done numerically. One day Tamar realized that these integrals can be calculated analytically, so she replaced the routine in the code that performed the numerical integration by a routine that contained the analytic formulas for the integrals. She checked the code by running it for a few benchmark problems, and to her surprise she saw that the errors increased rather than decreased! She checked for a bug, but didn't find any. How can this be explained?

Answer:

First, here is the general answer that I had in mind and that I think relates to the real story of Tamar. Any code based on discretization (finite elements, finite differences, finite volumes, spectral...) generates discretization errors in the solution. If the code involves some numerical integration in it, then _in addition_ to the discretization error it also generates integration error. Now, who says that the two errors add up together and don't partly cancel each other? In other words, who says that the discretization error and integration error have the same sign? (The only law that says this in Murphy's law...) So it may certainly happen that the integration error will cancel part of the discretization error. In this case, the integration error actually "helps", and when we replace the numerical integration by exact analytical integration we would see that the total error increased!

Three readers (PT, RBZ and ZZ) gave a very good extreme example for this situation. In some pathological situations, such as thin beams, plates and shells when using a naive variational formulation and a naive approximation space, the finite element solution exhibits what is called "locking". In this case the discretization error becomes enormous. One remedy for this situation is a technique called selective-reduced integration. Without going into details, selective-reduced integration amounts to deliberately increasing the

integration error so that it cancels most of the locking error. So in this case both the discretization error and integration error are very large, but they almost cancel each other and thus the total error is small. If, god forbid, we replace the selective-reduced integration by an exact analytical integration the locking phenomenon will render the solution useless.

There is another possible correct answer. Since the statement of the question did not say that the analytic integration that Tamar used was _exact_, one might think that the analytic integration was itself approximate (say, asymptotic). In this case it might happen that the error involved in the analytic approximation is larger than that of the numerical integration.

Correct answers were received from the following (in alphabetical order): Rami Ben-Zvi, Amiel Herszage, Pavel Trapper, Asher Yahalom, Zvi Zaphir.