Log of Questions of the Month of 2012

****
The January Question of the Month
****

This Question was designed mainly by Jonathan Tal. It requires light programing skills.

Consider the following loop taken from a pseudo-code that implements some algorithm:

==========

Do m = 0, N*(N-2)-1

    j=INT(m/(N-2))

    i=m-j*(N-2)

    k=i+3+j*N

v(k) = v(k-1) * const1 + v(k-2) * const2

Enddo

==========

Assume that the number N is very large, say N^2 is a few millions [last time I wrote that N was a few million, but Jonathan Tal corrected me that it was too high], and therefore this loop entails "heavy" computing. The function INT(...) takes the integer value of a number. For example, INT(3.9)=3 .

The question is: Does this loop lend itself to parallel computing, namely to the use of a computer with many processors that can work in parallel? If the answer is yes, please explain how "the work" done in this loop can be divided among the various processors. If the answer is no, please explain why not. In that case, think about a way to write this piece of code differently so as to make it amenable to parallelization.

(Hint: if the algorithm is too abstract, try N=5.)

******
Answer
******

The loop does not lend itself directly to parallel computing (at least at first sight) since there are dependencies between the entries of the vector v. Since every v(k) depends on v(k-1) and v(k-2) we cannot let different processors compute different groups of these entries, because each processor would need information from the other processors.

However, not all is lost. To understand what is going on exactly in this loop, let's look at the case N=5. For N=5, the loop's parameter m goes from 0 to 14. The corresponding values of j are:

j = 0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4.

And the corresponding values of i are:

i = 0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2.

The corresponding values of k are:

k = 3, 4, 5, 8, 9, 10, 13, 14, 15, 18, 19, 20, 23, 24, 25.

Since every v(k) depends only on the two predecessors v(k-1) and v(k-2), we see that there is a separation into groups here, after all. For example, v(8), v(9) and v(10) do not depend on v(3), v(4) and v(5), but depend only on v(6) and v(7) that do not appear at all in this loop! (Their values were read or calculated somewhere earlier in the program.) Therefore, the processor calculating v(8), v(9) and v(10) can be different than the processor calculating v(3), v(4) and v(5) - there is no dependency between these two groups.

More generally, groups of size N-2 are created in this loop, and these groups can be assigned to different processors and can be computed independently of each other. A clear way to implement this is to notice that the given loop is equivalent to the following one:

==========

Do j = 0, N-1

Do i = 0, N-3

v(i,j) = v(i-1, j) * const1 + v(i-2, j) * const2

Enddo

==========

Here we replaced the vector v(k) by a matrix v(i,j). Clearly, we see that while v(i,j) depends on previous i's it does not depend on previous j's. Therefore, we can assign different values of j to different processors.

Correct answers were received from:

Ran Ganel, Rachel Gordon.

****
The February Question of the Month
****

Sometimes, when solving a two-dimensional or a three-dimensional problem, the solution varies rapidly in one direction, and slowly in another direction. An example is a viscous flow problem with a thin boundary layer. Inside the boundary layer the solution changes rapidly in the direction normal to the layer (the layer-thickness direction) and much more slowly in the direction tangent to the layer. Another example is that of elastic shells. The solution changes rapidly in the thickness direction and much more slowly in the direction tangent to the shell surface.

These are considered difficult problems. If we attempt to solve numerically the full equations (Navier Stokes or 3D elasticity) this requires either extremely fine meshes or some special numerical techniques. For example, shell problems are usually solved by finite element methods using "shell elements", not using 3D elasticity elements.

However, one may wonder why not to use a grid/mesh that is fine enough in the thickness direction and coarse enough in the tangential direction. If the solution is known to vary slowly in the tangential direction, there doesn't seem to be a reason to take a fine grid/mesh in that direction. With a grid/mesh that is fine only in the thickness direction, the computational effort may not be high. But the fact is that usually this is not done. Why not?

****
Answer
****

The truth is that the procedure proposed above _can_ be applied in various cases if one is sufficiently careful. Indeed, in solving viscous flow problems with boundary layers, when the grid/mesh is rectangular and aligned with the directions of the boundary layer, this is a procedure often used. Here is a quote from ET's answer, which also points to some possible difficulties:

"In fact when solving the compressible Navier-Stokes this is exactly what is done. In any case this leads to high aspect ratio cells. For time dependent problems with an explicit scheme there is a stability bound on the time step. This will be dictated by the short side of the cell. Hence, it will take many time steps for a wave to propagate in the direction of the long cells leading to an inefficient scheme. Many implicit schemes, though formally stable at large CFL time steps, nevertheless display similar
problems. Essentially what are are good properties for the scheme in one direction is poor in the other direction.

For steady state problems [when solving via time-stepping until steady state is reached] similar difficulties arise. The scheme will have fast convergence properties in one direction but not the other. If everything is sufficiently neat then one can use multigrid with either semi-coarsening or an explicit smoother in one direction and an implicit smoother in the other. In practice this does not work because the problems do not usually usually split neatly into two directions, i.e., at corners. These problems get worse in three dimensions."

If the problem is solved implicitly in time, or if this is a steady-state flow problem or a static elastic problem which is solved via an "elliptic solver" (namely not by time-stepping to reach steady state but directly in space alone, e.g., by Newton iterations), then one should also be careful about the conditioning of the matrix appearing in the algebraic system of equations that results from the discretization. The cells/elements which have a very large aspect ratio (length divided by width), that in the context of the finite element method are sometimes called "needle elements", usually lead to ill-conditioned matrices. It is advisable to use a good preconditioner when solving such systems.

The situation is much worse if the cells/elements are not rectangular. For example, suppose you consider a curved thin solid (shell) and you use quad straight-edged finite elements (FEs). They cannot be all rectangular because of the curvature of the boundary. Therefore there is no exact separation between the two directions of the element - the "fast direction" in the thickness direction and the "slow direction" tangent to the surface. In other words, when you go in any direction inside the element, you always see a "component" of the fast solution. In this case there is a true accuracy problem, not just ill-conditioning. The standard general FE error estimate states that the error behaves like Ch^a where h is the "size" of the element. This element size h is determined by the largest side of the element. So the overall error will be determined by the coarse direction, and this will ruin all the accuracy of the computation achieved by the fine

discretization in the thickness direction. This is the reason why "needle elements" are not used for shell-like solids, but instead special shell FEs are used.

Correct answers were received from:

Orna Agmon Ben-Yehuda, Idit Avrahami, Rafi Haftka, Amiel Herszage, Eli Turkel, Micha Wolfshtein, Zvi Zaphir.

****
Comment on the February Question of the Month
****

The February Question dealt with situations where the cells/elements in the grid/mesh have a very large aspect ratio. Such "needle cells/elements" are sometimes used, for example, to solve flow problems involving thin boundary layers. The numerical difficulties associated with such grids were briefly discussed.

It should be mentioned that there is an active effort going on to develop numerical methods that overcome these difficulties. For example, Eli Turkel (a member of our community) is working with his group to develop a "preconditioner" that overcomes the difficulty and converges fast even with high aspect ratio grids. "The main purpose of any preconditioning is to find an operator which is close to the inverse but at the same time is cheap to evaluate. The hope is that the preconditioner (approximate inverse) times the original operator has nice properties. In particular it removes much of the stiffness due to the high aspect ratio cells." In the particular case considered by E.T.'s group they advance the Navier-Stokes equations (in order to find the steady state solution) by an explicit Runge-Kutta scheme embedded in a multigrid scheme. Their total algorithm converges about 4 times faster in terms of CPU time than the standard scheme.

Oren Livne and Achi Brandt (also members of our community, the latter being the inventor of Multigrid) also work on developing a numerical method with a similar goal. Their scheme involves algebraic multigrid, which serves here as a very effective preconditioner. "The characteristic directions can be automatically revealed and incorporated into the solver using a bootstrap approach even when not aligned with coordinate lines." In this manner, their scheme overcomes the alignment restriction that I have mentioned last time. Moreover, they use overlapping patches of fine grids within the multigrid hierarchy to obtain a more accurate discretization. This they do adaptively using the tau-correction criterion.

If you are interested in these works and would like to get more details or preprints, please contact Eli Turkel (turkel@post.tau.ac.il) and/or Eli Livne (livne@uchicago.edu) directly. It should be mentioned that a few other research groups are working on the same type of problems in various places in the world.

****
The March Question of the Month
****

Viscoelasticity deals with materials whose behavior is both elastic and viscous. Such materials (which sometimes behave more like solids and sometimes more like fluids) have "memory", namely their response to loading at the current time depends not only on the current loading and on their state in the current instant, but on their entire history (theoretically their history since they came out of the factory). Computationally this is very bad news, since it means that one must store the entire history of the solution on the computer, and this history keeps accumulating and is entirely involved in computing the current solution. For example, suppose you need to do a certain viscoelastic analysis with a 3D body that has 40,000 degrees of freedom, and you need to perform 10,000 time steps. Even a purely elastic analysis would be quite heavy with such parameters, but a viscoelastic analysis would be terrible. As time proceeds the computation would become more and more expensive, and the cost would grow very fast (quadratically). In addition the program would need to store a huge amount of data; the entire history for each of the 40,000 degrees of freedom. Are there any remedies for this difficulty?

****
Answer
****

The difficulty is in the fact that during the time-stepping we cannot seem to be able to "accumulate the information" and entirely forget the old information. There are two main ways to deal with this memory problem.

Here is the first approach. The behavior of viscoelastic materials can be described by a function called the "relaxation function", which is a kernel appearing in the integral that relates the strains to the stresses. (The relaxation function is analogous to Young's modulus in elasticity). There are various types of viscoelastic materials, with different relaxation functions. There is an important and wide class of viscoelastic materials that have a relaxation function of a special form: a sum of exponentials. This is called a Prony series. In this case, it is possible to manipulate the viscoelastic time-stepping process in such a way that allows accumulation of the information in a special "internal variable", and thus avoids the need to remember old information. In this way, only information from the last step is needed for the time-stepping process. RBZ wrote a nice Note on this procedure in Computers & Structures, 1990.

And what do we do if the relaxation function cannot be written as a sum of exponentials? There is a second approach, according to which we do use the old information but in an approximate and efficient way. We do this by replacing the old information by a much more compact representation. For example, instead of storing all the data at 5000 time-steps, we can represent the data by a high-order polynomial (say of degree 10) and keep only the coefficients of this polynomial. We typically do this with the old enough information, whereas we do store and use

the more recent information (say, the last 20 time steps) as it is. Instead of a polynomial we can use a Fourier series, or any other appropriate basis, as OL and ET suggested.

Correct answers were received from:
Rami Ben-Zvi, Oren Livne, Eli Turkel.

****
The April Question of the Month
****

It's not so nice to call someone by an insulting name, even if this someone is a numerical algorithm. Yet, there is a class of algorithms that are called "greedy" (חמדניים).
What are "greedy algorithms"?
Does their greediness pay (משתלמת)?
Can you give an example for such an algorithm, in the context of numerical methods?

****
Answer
****

I will start from a quote from OL's answer:
"Greedy algorithms are a strategy to arrive at a solution of a problem by making the move that provides the largest profit from the current state at every step. For instance, energy minimization can be solved by a point-by-point minimization, i.e., scanning variables in sequence and minimizing the energy as a function of each one, fixing all others at their latest values. The general difficulty with the greedy strategy is that it is may be trapped in a local minimum." I will add that sometimes greedy algorithms converge very slowly to the optimum.

Greedy algorithms appear in various areas of computer science. In the context of computational methods, the example that I had in mind (also suggested by OABY) is the Steepest Descent algorithm, to find the minimum of a function of many variables. Finding the minimum of a multi-variable function is encountered often in computational mechanics. Here is an illustration for the idea behind this algorithm (for a function of two variables), from "every day life".

Suppose you are a hiker in the wilderness. You hiked all day, and planned to get back to your camp before the sun sets, but alas, you didn't calculate the timing right, and you found yourself, late in a very dark night, standing in a high place, while your camp is deep down below, in the lowest level of the topography. So you should climb down until you reach the "minimum", which is your camp. The trouble is that you cannot see anything, although you can feel the slope of the ground with your feet. Let's assume that the topography is smooth (no sharp rocks, etc.), and that your speed of climbing is constant and does not depend on the slope. Your goal is to reach the

camp in the smallest amount of time. This is in fact the problem of finding the minimum of a function (that describes the topography), z(x,y), when your starting point ("initial guess") is given, (x0,y0).

The Steepest Descent algorithm says the following: Go by small steps of constant length. In each step you take, choose the direction in which the slope of z(x,y) is the largest. This seems like a reasonable thing to do. And it is a greedy algorithm, because in each step you go in the direction that will give you the best "profit", namely will bring you down the most. But it turns out that this is far from being an efficient algorithm. Typically it converges to the minimum very slowly, in a zig-zag manner. There are much better algorithms that one can use to find the minimum, e.g., conjugate gradients. Well, that's the punishment for being greedy!

Correct answers were obtained from:
Orna Agmon Ben Yehuda, Rafi Haftka, Oren Livne, Michael Medvinsky.

****
The May Question of the Month
****

During a coffee break in a conference, you heard two people discussing if a certain problem should/can be solved with DNS, LES or RANS. (a) What do these "words" mean? Write a line or two of explanation on each of them. (b) What are the main factors that determine if a certain problem should/can be solved with DNS, LES or RANS? (The answer to (b) may be in principle very long, but here it is ok just to scratch the surface and give a very short and crude answer.)

****
Answer
****

I received four excellent and detailed answers (very similar to each other, of course). The answer below is a mixture of all of them.

(a)

DNS, LES and RANS are all computational approaches used in solving fluid flow problems, namely they are approaches employed in Computational Fluid Dynamics (CFD). These approaches are associated with different level of accuracy and computational effort - DNS (= Direct Numerical Simulation) is in general the most accurate but also the most computational intensive, and RANS (= Reynolds Averaging Navier Stokes) is in general the least accurate but is also the least expensive computationally. LES (= Large Eddy Simulation) is generally in between them in terms of accuracy and computational effort. You get what you pay for!

RANS is used for turbulent flows with a "turbulence model" (e.g. k-epsilon, k-omega, Reynolds-stress, etc.). A turbulence model is based on the assumption that while the flow field itself is unknown, it has measurable statistical properties. The turbulence model is applied to all but the very largest scale of motion of the mean flow.

LES is based on separating the large scales from the small (unresolved, or sub-grid) scales. The small scaled are "modeled" as in RANS, but the large scales are treated directly. Thus, there is some modeling involved but less than in RANS.

DNS does not involve any "modeling"; it solves the Navier Equations in brute force.

(b)

Choosing one of the three approaches for a given problem depends on the type of the problem and on our computational resources, and the amount of computational effort (--> computing time) that we can afford.

In DNS, results are in principle reliable (if the discretization is good enough) but the computational cost is usually prohibitive, and the memory required is astronomical, which makes this option impractical. Therefore the use of DNS is restricted to fundamental ("academic") studies at low Re numbers and with simple geometries. It cannot be used today to solve realistic physical or engineering problems.

The use of LES makes sense if the small scales lend themselves to turbulence-modeling, e.g., they are homogeneous and isotropic. The results are usually reliable, although to a smaller degree than DNS, especially near solid walls. The computational effort may be acceptable if the computer used is strong enough. Still, time-dependent LES frequently requires weeks of computer time, even on a
multi-processor workstation. Hence, it is mainly used in simple configurations and in more academic settings. Nevertheless, application of LES to more realistic configurations is gaining force in recent years, especially with the use of super-computers with parallel processing.

RANS, because of its affordability, is the standard workhorse of turbulent flow calculations. The calculation can be 2D or 3D and since a relatively coarse grid can be used fairly complex geometries and multi-physics problems can be solved. RANS is certainly less reliable than the other approaches, and therefore requires very careful validation by comparison of experimental data. (Of course, all three approaches require validation, but RANS results should be especially taken with great cautiousness.)

Correct answers were received from:

Steven Frankel, Amiel Herszage, Eli Turkel, Micha Wolfshtein.

****
The June Question of the Month
****

Consider a computational scheme that uses a grid/mesh to solve a problem in a given domain, say in 2D. There are two main approaches for receiving solutions with improved accuracy in computational schemes: either we refine the grid/mesh or we use a higher-order scheme.

For example, suppose you are trying to solve a certain problem using a 2nd-order Finite Difference (FD) method with a coarse grid. You get results and you are not so satisfied. You would like to obtain a more accurate solution. Then you have two options: either to refine the grid, say by dividing the distance between all the grid points by a factor of 2, or to use the original grid, but replacing the 2nd-order FD scheme by, say, a 4th-order FD scheme.

If Finite Elements (FE) are closer to your heart, imagine you are using a coarse mesh with bilinear rectangular elements, and you want a more accurate solution; you can either refine the mesh, say by dividing each rectangle into 4 smaller rectangles, or you can use biparabolic elements instead of the bilinear elements. In most commercial FE codes you would be able to make either of these choices. This is related to what is called the h-version and the p-version of the FE method. The h-version is based on converging to the exact solution by refining the mesh, while the p-version is based on using a fixed mesh but with shape functions of a higher degree of polynomial.

Now, it is well known that in many cases, using a higher-order scheme reduces the error much more dramatically than refining the mesh. For example, in the FE example given above, if we refine the mesh by dividing each rectangular element into smaller elements, the error would be reduced algebraically, while if we fix the mesh and increase the degree of the polynomial of the shape functions, the error would be reduced exponentially. The computational cost would be about the same in these two cases, since it is mostly determined by the number of unknowns. Whether we replace each element by 4 smaller elements or we replace the bilinear elements by biparabolic elements, we increase the number of unknowns by a factor of 4.

(The next paragraph is based on a comment by Michael Medvinsky. Many thanks!)
With FDs, the details are a little different. Changing a FD scheme to a higher-order FD scheme with the same grid does not change the number of unknowns - only the stencil becomes wider and involves more grid points. Thus, there is an increase in the number of unknowns appearing in each equation of the resulting linear system, but not in the overall number of unknowns. The computational cost increases because (a) there are more terms to compute in the system's matrix, and (b) the sparseness of the matrix decreases. A simple calculation shows that refining the grid and increasing the order of accuracy by one involve roughly the same amount of computational effort (at least in order of magnitude).

If this was always the case, then nobody would ever refine grids/meshes; we would always increase the order of the scheme and thus get exponential convergence to the exact solution; and

all commercial codes would allow high-order methods. This is certainly not the case - in fact the standard way of obtaining solutions of improved accuracy is to refine the grid/mesh. Why is that?

****
Answer
****

Probably the main reason that the p-version (or convergence via increasing the order of the scheme) is not always superior to the h-version (or convergence via grid refinement) is that the exponential convergence of the former is true only if the exact solution is sufficiently smooth. For non-smooth solutions, increasing p would improve the accuracy only up to a certain point, and beyond it one would need to switch to the h-version to get good convergence. A combination of both approaches, the h-p approach, has been shown to be very effective, and in fact optimal if designed properly.

Non-smoothness of the exact solution, that is mentioned above, may be caused by various reasons, such as some sort of singularity (e.g., a reentrant corner in the domain, a concentrated load, a discontinuity in the derivatives of the solution, etc.).

This reason was included in all the answers that I have received. In addition:

OL and OABY add that h-refinement is considerably simpler to program than p-refinement.

MM adds that in Finite Difference schemes, increasing the order causes the stencil to be larger, and thus complicates the treatment near boundaries.

OABY adds: "In parallel distributed memory codes, higher order schemes in FD require passing more information more often between computation nodes, thus increasing communication overhead nd making the whole thing less worthwhile. On the other hand, refining the mesh means that (in 2D, for example, with n zones or cells and c computational nodes) you would compute 4n/c instead of n/c computations per computational node, but you would only be transferring twice as much information about boundaries, so parallelization would even be more efficient."

RH and RG add that complex geometries may be difficult to describe well with large elements used in the p-version.

RG adds the following three reasons not mentioned above for preferring the h-version over the p-version:
1) Sparser and better conditioned systems.
2) Better verification of the maximum principle if there is one.
3) Easier treatment of nonlinearities (simpler treatment of numerical integration).

Correct answers were received from:

Orna Agmon Ben-Yehuda, Travis Fisher (answer communicated by Steve Frankel), Roland Glowinski, Rafi Haftka, Oren Livne, Michael Medvinsky, Zvi Zaphir.

****
The July Question of the Month
****

This time, a little programming thinking is required.

Roshka, our old friend from the computational engineering company BATALA (Benchmarks And Theoretical Analysis for Low-tech Applications), needed to calculate the Frobenius norm S of an NxN matrix A, namely

S = sqrt(SUM (A_ij)^2)

where the sum is over i and j, each going from 1 to N. Roshka programmed this in a simple way, doing two nested loops to calculate the double sum, thus doing N^2 operations. Now, suppose that the matrix A that Roshka works with has the following expression:

A_ij = v_i w_j

where v and w are two N-dimensional vectors. Suppose N is a huge number, say 10 million. What would you advise to Roshka?

****
Answer
****

This time I received many answers (I think it is an all-time record for the number of answers), and they are all correct. Some readers wrote down a very nice derivation based on tensor algebra, and some added some detailed comments. But here I will just write the answer in the most elementary and shortest way possible:

S^2 = SUM (A_ij)^2 = SUM_i SUM_j (v_i w_j)^2 = [SUM_i (v_i)^2] [SUM_j (w_j)^2]

and therefore

S = sqrt(SUM_i (v_i)^2) sqrt(SUM_j (w_j)^2)

which can also be written as

S= ||v|| ||w|| .

Thus, the Frobenius norm of the matrix A is equal in this case to the product of the norms (lengths) of the two vectors. This calculation requires O(2N) operations and not O(N^2) operations.

Correct answers were received from:

Idit Avrahami, Rami Ben-Zvi, Michel Bercovier, Michael Bogomolny, Roland Glowinski, Rafi Haftka, Amiel Herszage, Oren Livne, Michael Medvinsky, Ilya Soloveichik, Jonathan Tal, Asher Yahalom, Zvi Zaphir.

****
The August Question of the Month
****

Many numerical methods involve the solution of a linear algebraic system of equations, Ax=b, where A is a given matrix, b is a given vector, and x is an unknown vector. There are two types of solvers for this system: direct solvers and iterative solvers. Direct solvers solve the system by operating on the matrix and vector components, and find the exact solution (up to round-off errors due to the computer used) in a finite number of operations. The prototype direct method is Gauss Elimination. Iterative solvers find an approximate solution iteratively. The solution in iteration j+1 is found from the solution obtained in iteration j, and sometimes also from previous iterations (for example, by a formula of the form x_{j+1} = C x_j + d, where C and d are related to A and b). The solution keeps improving as the iteration process continues. Two of the popular iterative solvers in computational mechanics are Gauss-Siedel and (if A is symmetric) Conjugate Gradients.

The question is: Since direct solvers find the "exact" solution whereas iterative solvers only find an approximate solution, why would anyone choose to use an iterative solver? In other words, what are the advantages of iterative solvers over direct ones?

****
Answer
****

A lot can be said about this question. The answer is not always simple, and you can hear people in conferences argue hotly in favor of direct or iterative methods. In any case, I think it is fair to state that the consensus in CM is that for "small" problems" direct methods are more appropriate and for "large problems" iterative methods are more appropriate (and sometimes are the only

practical way to solve the problem). Where the border passes between "large" and "small" is not always agreed upon. I will comment on this below.

I think that the essential argument is given by the following point. Direct solvers are of the type of methods that give you "everything or nothing", while with iterative solvers you can stop the solution process whenever you like, and the quality of the result that you get depends on the amount of computational effort that you are willing to make. To take an extreme example, suppose you are content with a 0.5% error (because your discretization error is 1% anyway), and suppose a direct method will give you a 0.0001% round-off error and would entail a large computational effort. Wouldn't it be a waste to pay so much for something that you do not need? It makes much more sense to use an iterative method, stop it after your error estimator tells you that you reached a level of about 0.5% error, and just pay less and get less accuracy.

OL provided the following list of advantages of iterative solvers:

Tunable solution accuracy. (Similar to my argument above.)

Generality. Direct solvers often rely on a specific structure or clever ordering of the unknowns (red-black/structured grids). On unstructured grids or graphs, they are plagued more by fill-in.

Extensibility. (I will omit the explanation.)

Speed. Well-designed, iterative methods can achieve O(N) complexity where direct solvers typically scale at best as O(N^(3/2)). The power increases with the dimension.

Lower memory cost. One does not need to store the matrix of coefficients, only local stencils.

Natural parallelization. Iterative solvers require local processing whereas direct solvers create coupling between farther unknowns.

Natural homogenization. (I will omit the explanation.)

OABY adds the following advantage: If the algebraic solver is used within a time-stepping loop (implicit), "the initial guess for the iterative solver can sometimes be taken from the solution in the previous time step, whereas for a direct solution we cannot use a guess - we must solve everything all over again."

Returning to the question of what the border is between "small" problems (for which direct solvers seem to be appropriate) and "large" problems (for which iterative solvers seem to be appropriate), the typical number that one often hears about is of the order of 10,000 unknowns. However, there are CM researchers and practitioners whose opinions on this matter are very different. Some think that iterative methods should _always_ be used, at least for problems with sparse matrices, even for "small" problems. On the other hand, some think that direct solvers should always be preferred, due to their robustness, and iterative methods should be kept to

"huge" problems only. For example, Dr. Vladimir Belsky, who is the Development Director of Solvers in Dassault Systemes SIMULIA Corp., sent me a fascinating and well-written (unpublished) paper that he wrote about the subject, with many examples and graphs, based on experiments with the FE software Abaqus. His analysis - in the context of nonlinear solid mechanics - seems to demonstrate the advantage of direct solvers over iterative ones. Moreover, his group has been using a direct solver to solve problems with more than 250 Million (!) unknowns, without difficulty, on a 256 core compute cluster. If you are interested in more details, you may contact Dr. Belsky directly: Vladimir.BELSKY@3ds.com .

Correct answers were received by:
****
Orna Agmon Ben-Yehuda, Steven Frankel, Leonid Kucherov, Oren Livne, Evgeny Shavelzon, Eli Turkel, Anne Weill.

Comment on the August Question of the Month

Rachel Gordon remarks that there has been some work on Hybrid Solvers which combine the direct and iterative methodology and are based on partitioning the matrix into blocks. One of the researchers who is doing such work is Iain S. Duff. See http://en.wikipedia.org/wiki/Iain_S._Duff about him (but no Hybrid methods there), and see, e.g., the paper: Duff, I. S. (2007), The use of hybrid techniques at CERFACS for the solution of large problems on parallel machines, Proceedings in Applied Mathematics and Mechanics 7 (1), 1140501-1140502.

****
The September Question of the Month
****

The following question is dedicated to the memory of my old Brazilian friend Prof. Leo Franca who passed away last month. Leo was a great researcher who made important contributions to CM, especially in the area of stabilization techniques.

Many of us know that a convergent numerical method must be (1) locally accurate, (2) stable. This is what the famous Lax theorem tells us. In time-dependent problems, the lack of stability usually manifests itself in that the solution "blows up" in time, namely at a certain time the solution begins to grow (typically exponentially fast) in an unbounded way. For example, this happens when one uses explicit time-marching with a time-step which is not small enough (thus violating the so-called CFL condition).

However, consider now a steady-state or static problem (in mathematical terms an elliptic problem), where time is not involved at all. Let us assume that the numerical method that we use give us a non-singular system of algebraic equations that can be solved. And suppose the method can be shown to be locally accurate. Can the method be unstable under such circumstances? If it can, how would the instability manifest itself in the solution? The solution cannot "blow up in time", because there is no time in this problem. Give at least one example for an instability in the steady-state or static case, and explain how the instability is "seen".

****
Answer
****

Here are a few examples for an instability appearing in static / steady-state cases.

1. AH and RBZ give an example of a finite difference scheme for steady-state incompressible flow problems. If the difference scheme is not appropriate, the pressure may strongly oscillate from one grid point to another (or from one cell to another in finite volume methods). This is called "checkerboard pattern" (checkerboard = game-board of Damka, which is also a chess-board), because in extreme cases one would get a change of sign between neighboring cells, which reminds one of the black and white cells on a checkerboard. One possible remedy in this case is to use what is called a staggered grid.

2. A similar situation happens in "mixed finite element" methods. If one does not use appropriate shape functions for the various variables involved, an instability may arise in the form of a "checkerboard pattern". There is a condition (called the Babuska-Brezzi condition) for stability, and if the shape functions satisfy this condition then they are safe for use.

3. Another instability phenomenon which may arise in mixed finite element methods, if one does not use appropriate shape functions, is called "locking". In this case, the solution is not oscillatory but simply approaches zero (instead of approaching the exact solution) as one refines the discretization. This can be explained in the following manner. Mixed finite element methods try to enforce some governing equations (say the momentum equations) and also a constraint (say the incompressibility constraint). If the approximation space is not chosen well, the constraint may be enforced too strongly so as to dominate the entire system of equations. Good approximation gives good balance between the governing equations and the constraint.

4. RBZ: "In solid mechanics, the so-called “hourglass modes" are kinematic patterns of single cells/elements deforming rectangles into trapezoid with no stress involved when a constant stress is assumed within the cell/element. These modes are manifested as zigzag global deformation patterns. They are avoided by either higher order scheme, higher integration order within elements or by adding a small suppressing anti-hourglass artificial viscosity term."

5. OL gives an example of a "bad" finite difference scheme (which is an explicit "marching scheme") for Laplace's equation. His nice analysis shows that if (i,j) are the indicators of the grid

points, then an exponential blow-up in the norm U[j] = max_j|u[i,j]| is to be expected as j grows. OL even implemented the scheme and demonstrated the instability numerically.

6. OABY gives an example of a method for mesh smoothing. This in fact involves the solution of Laplace's equation. "In most smoothing methods, we more or less move each vertex to the center of its neighbours, for example according to a rule that says that each edge is a spring, so that the springs dictate the direction and amount of motion." If one takes too large steps in this process, one may get an entangled mesh, which becomes worse and worse with further steps. (Is this an instability? Probably yes, because by definition an instability is a situation where small changes in the data cause large changes in the solution, and this seems to be the situation here.)

7. In a static problem involving a very thin boundary layer (e.g., a string on an elastic foundation with a very stiff elastic constant), if one does not treat the boundary layer appropriately, one would get spurious oscillations in large parts of the domain (far beyond the boundary layer itself). This is essentially a stability problem.

8. If the solution of a steady-state problem involves a discontinuity, or a shock, and it is not appropriately treated, again one would get spurious oscillations in large parts of the domain.

Correct answers were received from:

Orna Agmon Ben-Yehuda, Rami Ben-Zvi, Amiel Herszage, Oren Livne.

****
The October Question of the Month
****

Remember Roshka? He is your employee in the computational engineering company BATALA (Benchmarks And Theoretical Analysis for Low-tech Applications). Here is a dialog between you two.

You: Roshka, any progress with the composite plate problem I gave you last month?

Roshka: I have great difficulty with this problem. It cannot be solved on our computers.

You: I don't think that this problem is so complicated.

Roshka: Well, I have looked at the analysis that Shlomo did a year ago, of a model with a single fiber embedded in a narrow epoxy bar. He wanted to look at the behavior of a single fiber. That model required about 1000 degrees of freedom.

You: I do not see the connection with the analysis that I asked you to do.

Roshka: Don't you see? The model we have now is that of a plate with 40 layers of epoxy, each one having about 5000 fibers oriented in a layer-dependent direction. In view of Shlomo's model, this means that our model would require about 1000 x 40 x 5000 degrees of freedom, namely 2 hundred million degrees of freedom! If you want us to solve this, then BATALA should buy a super-computer. I have always claimed that we needed one of those monsters, and now I have the proof.

You: No need for that, Roshka. Here is what you should do.

What would you instruct Roshka to do?

****
Answer
****

When dealing with a medium which is highly heterogeneous (material properties change very rapidly in the body of interest, in our case fiber-matrix-fiber-matrix-fiber-...), it is not practical to model each and every piece of material - each and every fiber - separately. The common approach to handle such problems is called "homogenization". In this approach the body is regarded as a homogeneous (uniform material) body having effective material properties. These effective properties are calculated as some sort of averages of the material properties of the constituents - the fibers and the epoxy matrix in our case. Once the body is looked at as homogeneous with effective material properties, modeling using a numerical method becomes quite easy, and requires only the number of degrees of freedom that a homogeneous body would require.

The way to calculate the effective properties of a composite structure is not trivial, and requires its own mathematical theory. One of the main inventors of this theory is Israeli - Prof. Zvi Hashin from Tel-Aviv University, who won the Israel Prize a couple of years ago on his work related to this.

Of course, one has to be aware that by this sort of analysis one obtains only a global "average" type of response of the structure, and is giving up the micro-structure details of the body from the outset. Thus, no information on the contact between layers, delamination, buckling of fibers, etc., can be obtained from the results of the analysis, since the fibers and layers are not modeled at all. If such micro effects are important to the analyzer, she should do some special, more local, analysis, of the sort that Shlomo in the story has done.

Correct answers were obtained from:

Orna Agmon Ben-Yehuda, Rami Ben-Zvi, Rafi Haftka, Anne Weill, Zvi Zaphir.

****
Comments on the October Question of the Month
****

The October Question was related to the subject of homogenization that is used for the analysis of a composite material. The idea underlying homogenization is to replace the original problem, concerning a highly heterogeneous body (material properties changing rapidly in space), with one which is homogeneous (a single material), with effective material properties. I remarked: "Of course, one has to be aware that by this sort of analysis one obtains only a global "average" type of response of the structure, and is giving up the micro-structure details of the body from the outset... If such micro effects are important to the analyzer, she should do some special, more local, analysis, of the sort that Shlomo in the story has done."

Naturally, the last observation points to the fact that this is a multiscale problem. Homogenization gives the solution on the macro (large) scale, and misses the details of the mezo scale (the layers) and the micro scale (the fibers). One comment that is in place here is that in recent years various sophisticated computational approaches have been proposed to take into account the various scales. We mention here the pioneering work of Tinsley Oden, and the more recent (and ongoing) work of Jacob Fish (from Columbia University) and Rami Haj-Ali (from Tel Aviv University, a full member of IACMM). Roughly speaking, this work adaptively identifies those regions where mezo- and micro-scale effects are important, and enhances the model so as to take into account these effects.

A related but farther-reaching comment was provided by Achi Brandt (main inventor of the Multigrid method) and Oren Livne, who state that it is better to do the homogenization numerically, as part of the computational method, rather than analytically, as suggested in the Roshka story. AB writes: "Actually, BAMG (Bootstrap Algebraic Multigrid) is generally the fastest way for a very general numerical homogenization. With FAS (Full Approximation Scheme), it can be used to homogenize (or “upscale”) nonlinear systems. (Strong nonlinearities, such as phase transitions, can be treated by the more general Systematic Upscaling (SU) methodology). Also, together with using everywhere the coarse-level equations, BAMG/FAS gives the possibility to use locally some of the finer levels wherever/whenever desired for analyzing local properties."

****
The November Question of the Month
****

Consider a linear steady-state heat transfer problem in a fluid domain, that generally involves both heat conduction (Holakha) and heat convection (Hasa'a). In particular, consider the following five cases / problems:

a. Only heat conduction is considered.

b. Only heat convection is considered.

c. Both heat conduction and heat convection are considered, and they are both "strong".

d. Both heat conduction and heat convection are considered, with "strong" conduction and "weak" convection.

e. Both heat conduction and heat convection are considered, with "weak" conduction and "strong" convection.

Which of the five problems is the most difficult one for numerical treatment? Why?

****
Answer
****

The answer is that problem e (both heat conduction and heat convection are considered, with "weak" conduction and "strong" convection) is the most difficult. To explain it very briefly, we note that among the 5 problems only problem e is associated with significant multiscale effects. The small scale that is generated in this problem is quite difficult to resolve with standard numerical methods. Moreover, if we ignore the small scale and try to solve the problem using standard numerical tools and reasonably crude grids/meshes, we would encounter stability difficulties, which manifest themselves as severe numerical artifacts (like large non-physical oscillations) that would pollute the whole solution.

To see this mathematically, let us write down the governing equation discussed in the question. The steady-state linear convection-conduction problem, also called the advection-diffusion problem, is given by the equation

k Laplacian u + b V . Grad u + f(x,y,z) = 0

with appropriate boundary conditions. Here u is the unknown temperature, f is the heat source, k is the conductivity and (k Laplacian u) is the conductivity term, b is the convection coefficient, V is a given velocity vector, and (b V . Grad u) is the convection term. To make things even simpler, we can think of the corresponding 1D problem:

k u" + b V u' + f(x) = 0,

with appropriate boundary conditions.

The question is what the hardest problem to treat numerically is:

a. b=0
b. k=0
c. b and k are both large
d. k is large and b is small
e. k is small and b is large

As we said above, the answer is e. The limit b-->0 is innocent, but the limit k-->0 is singular, and problem e is said to be a singular perturbation problem. A thin boundary layer is formed, in which the solution changes very rapidly. Asymptotic methods exploit this structure to get a nice analytic solution; however for computational treatment (say, in 2D with general geometry) this is a challenge.

RG remarks that even from a purely algebraic standpoint, when using iterative methods to solve the linear system of algebraic equations, the problem e is the most difficult, and requires some special iterative techniques to guarantee fast convergence.

This difficult problem is called "advection-dominated advection-diffusion". The advection-diffusion equation is so notorious for this reason that I have heard people in the CM community sometimes calling it the "defective-confusion equation"!

Correct answers were received from:

Rami Ben-Zvi, Rachel Gordon, Amiel Herszage, Alexander Yakhot.

****
The December Question of the Month
****

Suppose we want to build a computational model (for example, a finite element model) of a complicated mechanical system, in an industrial setting. We know that we are going to use this model many times: with different loads, attached to various other sub-systems, with different initial conditions, etc. In fact, this very model is going to serve us for a whole year of computations in our project on this system. We are doing a very careful job, and construct an appropriate model, which would give us good accuracy, for the given system. Alas (in Hebrew: Akh Oya), this great model is very "heavy", namely includes many degrees of freedom (say, 400,000), and a single run of it on our strongest computer, and with our best algebraic solution techniques, takes about 5 hours. Since we wish to run it many times, as mentioned above, and we'd like to have quick results each time, we cannot actually afford to use it.

What to do? Roshka thinks that we should "reduce" the model by simply using a grid/mesh which is cruder. Of course, this would compromise the high accuracy, but Roshka thinks there is no choice. Is there any better option?

****
Answer
****

Let us suppose that the problem is linear. If the problem is static/steady/elliptic, the discretization in space leads to the linear algebraic system Kd=F. If the problem is time-dependent, then let us assume that we solve it via implicit time-stepping, in which case after discretization in space and time we also obtain, in each time step n, the linear algebraic system K d_n=F_n. In these cases, much (if not most) of the computing time is devoted to the solution of this algebraic system. It would be of much relief if we could find a fast way to solve this system. Of course, fast methods like Multigrid (and its variants) come to mind. If Roshka uses a "slow" solver, we should tell him to switch to fast solvers, as suggested by AY.

Another idea is the one proposed by RBZ and MB. The model is meant to be used repeatedly with different data, but the geometry it is based on and the grid/mesh used in space are assumed to be fixed. Thus, the matrix K remains fixed during all the runs. Suppose we solve Kd=F using a direct method like Gauss elimination. Gauss elimination can be described in two steps: (1) the factorization (decomposition) of the matrix K, and (2) the use of this factorization to solve the system (forward reduction / back substitution). Step (1) does not depend on F; only step (2) does. The factorization of K (step (1)) is the time-consuming part of the solution of Kd=F, while step (2) is very cheap. Therefore, a good idea is to factorize K only one time in the beginning of the entire project, and then to keep and use this factorization in all the subsequent runs. [Another way to go about this, which is less efficient but is perhaps more easily understandable, is to compute the inverse K^{-1} once in the beginning of the project, and then in each run to calculate K^{-1} F.] If the model is a subsystem of a larger model, some further procedure should be taken (called sub-structuring).

What I had in mind was actually different. I was thinking about a class of techniques called Model Reduction. The idea underlying these techniques is to replace the original discrete model by another model that has a much smaller number of degrees of freedom (DOF), yet in some important sense retains the accuracy of the original model. Producing the reduced model may be quite expensive, but the idea is that we do this only one time. Once we have the reduced model in our hands, we can use it repeatedly, and it is much more efficient than the original model. Of course, Roshka's suggestion to take a coarser mesh would also produce a "reduced model" but the high accuracy would be lost. A good reduced model does not cause a reduction in accuracy, as measured by some predetermined measure.

NE proposed the use of the model reduction method which is most popular in CM. In this method, the system, which has N DOF, is decomposed to its normal modes (modal decomposition). To this end, we first solve the eigenvalue problem associated with the original problem; the modes are the eigenvector-eigenvalue pairs. Of course, to solve the eigenvalue problem costs significantly more than solving the original problem, but so what - we do this only one time. This modal decomposition enables us to transform the original system to an equivalent "modal system", in which the N DOF are the "Fourier coefficients" of the various modes. Now

comes the crucial part: we _truncate_ our system, leaving only the first M DOF, where M is much smaller than N. This is based on the assumption (which in many cases is very good) that the first M modes are the most important ones, while the higher (N minus M) modes are not physically important and are also not representative of the true system. In this manner we are left with a much smaller system (M DOF instead of N DOF), and this is the system that we use for all our subsequent runs.

There are other, more sophisticated, ways to do model reduction. This is a "hot" subject, as reflected by the many papers in journals and conferences on the subject. (See one conference on this subject in the conference list above.)

Correct answers were received from:

Rami Ben-Zvi, Michael Bogomolny, Nir Emuna, Asher Yahalom.

****
Comments on the December Question of the Month
****

The December question of the month asked how to take a given discrete model and make it much more efficient for repetitive use. In my answer, I related (among other things) to the concept of _reduced models_. The idea underlying these techniques is to replace the original discrete model by another model that has a much smaller number of degrees of freedom (DOF), yet in some important sense retains the accuracy of the original model. As a simple example for a reduced model technique, I mentioned modal decomposition and truncation (which was the answer of Nir Emuna). Here are two comments on model reductions.

1. The Opening Lecture of the last Israel Symposium on Computational Mechanics (ISCM-33), given by Prof. Jan Hesthaven, exactly dealt with reduced modeling. Prof. Hesthaven's fascinating lecture, entitled "Reduced order models you can believe in", showed how to develop reduced methods endowed with rigorous error estimators to certify the accuracy of the model and to give it a true predictive value.

2. Prof. Achi Brandt, the main inventor of the celebrated MultiGrid technique, makes the important comment that a certain version / branch of MultiGrid, called Full Approximation Scheme (FAS), in addition to being a very efficient algebraic solver, actually produces an excellent reduced model. In AB's words: "As a byproduct of the FAS cycle, coarse equations at all scales (levels of the solver) are equipped with fine-to-coarse defect corrections (the so-called tau terms). These coarse equations, at any desired scale, constitute a very effective reduced model. The tau terms ensure that the fine level accuracy is fully preserved at the coarse level." If you are interested in more details, please write to me and I will send you the full comment of AB.