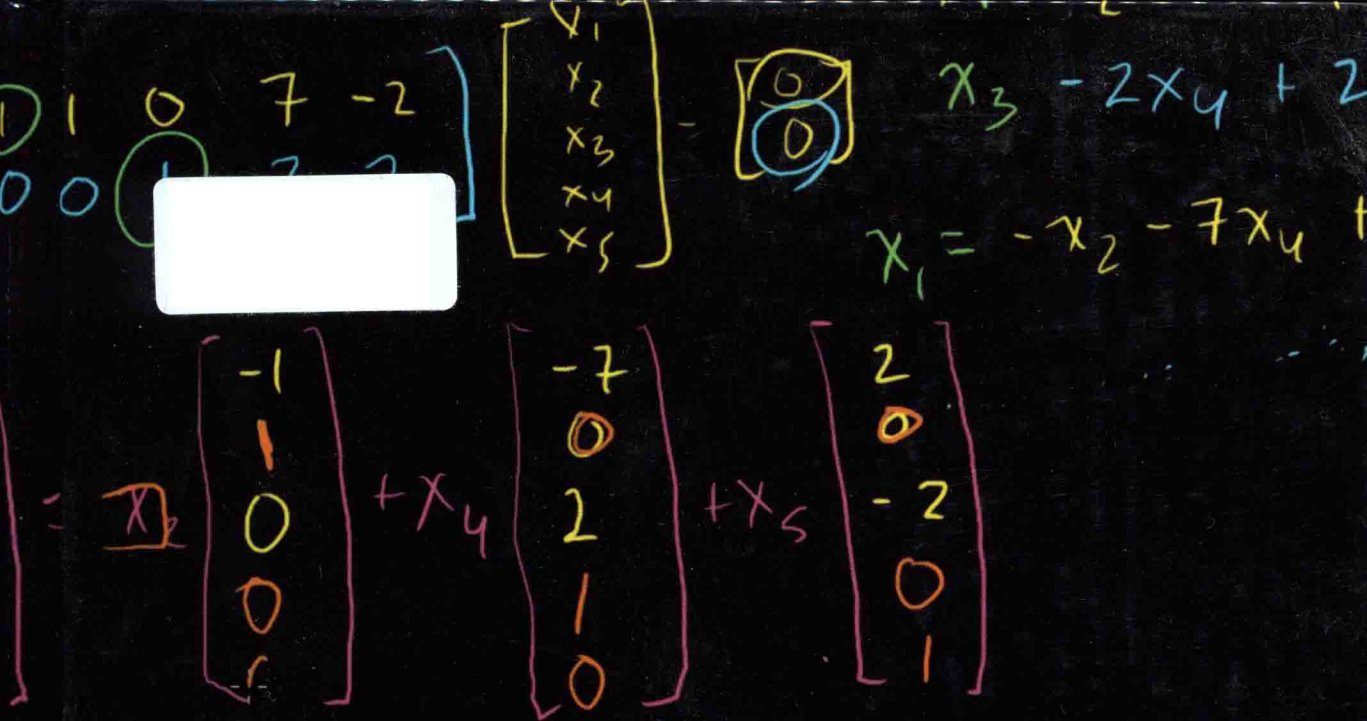

Handbook of Numerical Linear Algebra and Optimisation

Michael Stevens
Editor

Handbook of Numerical Linear Algebra and Optimisation

Michael Stevens
Editor

AURIS REFERENCE LTD.
London, UK

Handbook of Numerical Linear Algebra and Optimisation

Published by
Auris Reference Ltd., UK
www.aurisreference.com

ISBN: 978-1-78154-478-5

Editor: Michael Stevens

Printed in UK

10 9 8 7 6 5 4 3 2 1

Cover Design: Cover Lab

Handbook of
Numerical Linear Algebra and Optimisation

Preface

Linear algebra is central to both pure and applied mathematics. For instance, abstract algebra arises by relaxing the axioms of a vector space, leading to a number of generalizations. Functional analysis studies the infinite-dimensional version of the theory of vector spaces. Combined with calculus, linear algebra facilitates the solution of linear systems of differential equations. Techniques from linear algebra are also used in analytic geometry, engineering, physics, natural sciences, computer science, computer animation, and the social sciences. Because linear algebra is such a well-developed theory, nonlinear mathematical models are sometimes approximated by linear ones.

The study of linear algebra first emerged from the study of determinants, which were used to solve systems of linear equations. Determinants were used by Leibniz in 1693, and subsequently, Gabriel Cramer devised Cramer's Rule for solving linear systems in 1750. Later, Gauss further developed the theory of solving linear systems by using Gaussian elimination, which was initially listed as an advancement in geodesy. The main structures of linear algebra are vector spaces. Similarly as in the theory of other algebraic structures, linear algebra studies mappings between vector spaces that preserve the vector-space structure. When a bijective linear mapping exists between two vector spaces, we say that the two spaces are isomorphic. Because an isomorphism preserves linear structure, two isomorphic vector spaces are "essentially the same" from the linear algebra point of view. One essential question in linear algebra is whether a mapping is an isomorphism or not, and this question can be answered by checking if the determinant is nonzero. If a mapping is not an isomorphism, linear algebra is interested in finding its range and the set of elements that get mapped to zero, called the kernel of the mapping. Again in analogue with theories of other algebraic objects, linear algebra is interested in subsets of vector spaces that are vector spaces themselves; these subsets are called linear subspaces. For instance, the range and kernel of a linear mapping are both subspaces, and are thus often called the range space and the nullspace; these are important examples of subspaces. Elements of a general vector

space V may be objects of any nature, for example, functions, polynomials, vectors, or matrices. Linear algebra is concerned with properties common to all vector spaces. Since linear algebra is a successful theory, its methods have been developed and generalized in other parts of mathematics. In module theory, one replaces the field of scalars by a ring. The concepts of linear independence, span, basis, and dimension still make sense. Nevertheless, many theorems from linear algebra become false in module theory. Numerical linear algebra is the study of algorithms for performing linear algebra computations, most notably matrix operations, on computers. It is often a fundamental part of engineering and computational science problems, such as image and signal processing, telecommunication, computational finance, materials science simulations, structural biology, data mining, bioinformatics, fluid dynamics, and many other areas. Such software relies heavily on the development, analysis, and implementation of state-of-the-art algorithms for solving various numerical linear algebra problems, in large part because of the role of matrices in finite difference and finite element methods. Common problems in numerical linear algebra include computing the following: LU decomposition, QR decomposition, singular value decomposition, eigenvalues. In the simplest case, an optimization problem consists of maximizing or minimizing a real function by systematically choosing input values from within an allowed set and computing the value of the function. The generalization of optimization theory and techniques to other formulations comprises a large area of applied mathematics. More generally, optimization includes finding "best available" values of some objective function given a defined domain, including a variety of different types of objective functions and different types of domains.

More generally, if the objective function is not a quadratic function, then many optimization methods use other methods to ensure that some subsequence of iterations converges to an optimal solution. The first and still popular method for ensuring convergence relies on line searches, which optimize a function along one dimension. A second and increasingly popular method for ensuring convergence uses trust regions. Both line searches and trust regions are used in modern methods of non-differentiable optimization. Usually a global optimizer is much slower than advanced local optimizers, so often an efficient global optimizer can be constructed by starting the local optimizer from different starting points.

Based on courses taught to advanced undergraduate students, this book offers a broad introduction to the methods of numerical linear algebra and optimization.

—Editor

Contents

Chapter 1

Numerical Linear Algebra

Numerical linear algebra is the study of algorithms for performing linear algebra computations, most notably matrix operations, on computers. It is often a fundamental part of engineering and computational science problems, such as image and signal processing, telecommunication, computational finance, materials science simulations, structural biology, data mining, bioinformatics, fluid dynamics, and many other areas. Such software relies heavily on the development, analysis, and implementation of state-of-the-art algorithms for solving various numerical linear algebra problems, in large part because of the role of matrices in finite difference and finite element methods.

Common problems in numerical linear algebra include computing the following: LU decomposition, QR decomposition, singular value decomposition, eigenvalues.

Domain Decomposition Methods (CH)

Abstract Additive Schwarz Method

In mathematics, the abstract additive Schwarz method, named after Hermann Schwarz, is an abstract version of the additive Schwarz method, formulated only in terms of linear algebra without reference to domains, subdomains, etc. Many if not all domain decomposition methods can be cast as abstract additive Schwarz method, which is often the first and most convenient approach to their analysis.

Additive Schwarz Method

In mathematics, the additive Schwarz method, named after Hermann Schwarz, solves a boundary value problem for a partial

differential equation approximately by splitting it into boundary value problems on smaller domains and adding the results.

Overview

Partial differential equations (PDEs) are used in all sciences to model phenomena. For the purpose of exposition, we give an example physical problem and the accompanying boundary value problem (BVP). Even if the reader is unfamiliar with the notation, the purpose is merely to show what a BVP looks like when written down. (Model problem) The heat distribution in a square metal plate such that the left edge is kept at 1 degree, and the other edges are kept at 0 degree, after letting it sit for a long period of time satisfies the following boundary value problem:

$$f_{xx}(x,y) + f_{yy}(x,y) = 0$$

$$f(0,y) = 1;\ f(x,0) = f(x,1) = f(1,y) = 0$$

where f is the unknown function, f_{xx} and f_{yy} denote the second partial derivatives with respect to x and y, respectively.

Here, the domain is the square $[0,1] \times [0,1]$.

This particular problem can be solved exactly on paper, so there is no need for a computer. However, this is an exceptional case, and most BVPs cannot be solved exactly. The only possibility is to use a computer to find an approximate solution.

Solving on a Computer

A typical way of doing this is to *sample* f at regular intervals in the square $[0,1] \times [0,1]$. For instance, we could take 8 samples in the x direction at $x = 0.1, 0.2, \ldots, 0.8$ and 0.9, and 8 samples in the y direction at similar coordinates. We would then have 64 samples of the square, at places like (0.2,0.8) and (0.6,0.6). The goal of the computer program would be to calculate the value of f at those 64 points, which seems easier than finding an abstract function of the square. There are some difficulties, for instance it is not possible to calculate $f_{xx}(0.5,0.5)$ knowing f at only 64 points in the square. To overcome this, one uses some sort of numerical approximation of the derivatives. We ignore these difficulties and concentrate on another aspect of the problem.

Solving Linear Problems

Whichever method we choose to solve this problem, we will need to solve a large linear system of equations. The reader may recall linear systems of equations from high school, they look like this:

$2a + 5b = 12$ (*)

$6a - 3b = -3$

This is a system of 2 equations in 2 unknowns (a and b). If we solve the BVP above in the manner suggested, we will need to solve a system of 64 equations in 64 unknowns. This is not a hard problem for modern computers, but if we use a larger number of samples, even modern computers cannot solve the BVP very efficiently.

Domain Decomposition

Which brings us to domain decomposition methods. If we split the domain $[0,1] \times [0,1]$ into two *subdomains* $[0,0.5] \times [0,1]$ and $[0.5,1] \times [0,1]$, each has only half of the sample points. So we can try to solve a version of our model problem on each subdomain, but this time each subdomain has only 32 sample points. Finally, given the solutions on each subdomain, we can attempt to reconcile them to obtain a solution of the original problem on $[0,1] \times [0,1]$.

Size of the Problems

In terms of the linear systems, we're trying to split the system of 64 equations in 64 unknowns into two systems of 32 equations in 32 unknowns. This would be a clear gain, for the following reason. Looking back at system (*), we see that there are 6 important pieces of information. They are the coefficients of a and b (2,5 on the first line and 6,–3 on the second line), and the right hand side (which we write as 12,–3). On the other hand, if we take two "systems" of 1 equation in 1 unknown, it might look like this:

System 1: $3a = 15$

System 2: $6b = -4$

We see that this system has only 4 important pieces of information. This means that a computer program will have an easier time solving two 1×1 systems than solving a single 2×2 system, because the pair of 1×1 systems are simpler than the single 2×2 system. While the 64×64 and 32×32 systems are too large to illustrate here, we could say by analogy that the 64×64 system has 4160 pieces of information, while the 32×32 systems each have 1056, or roughly a quarter of the 64×64 system.

Domain Decomposition Algorithm

Unfortunately, for technical reason it is usually not possible to split our grid of 64 points (a 64×64 system of linear equations) into two grids of 32 points (two 32×32 systems of linear equations) and

obtain an answer to the 64×64 system. Instead, the following algorithm is what actually happens:

1) Begin with an approximate solution of the 64×64 system.
2) From the 64×64 system, create two 32×32 systems to improve the approximate solution.
3) Solve the two 32×32 systems.
4) Put the two 32×32 solutions "together" to improve the approximate solution to the 64×64 system.
5) If the solution isn't very good yet, repeat from 2.

There are two ways in which this can be better than solving the base 64×64 system. First, if the number of repetitions of the algorithm is small, solving two 32×32 systems may be more efficient than solving a 64×64 system.

Second, the two 32×32 systems need not be solved on the same computer, so this algorithm can be run in *parallel* to use the power of multiple computers.

In fact, solving two 32×32 systems instead of a 64×64 system on a single computer (without using parallelism) is unlikely to be efficient. However, if we use more than two subdomains, the picture can change. For instance, we could use four 16×16 problems, and there's a chance that solving these will be better than solving a single 64×64 problem even if the domain decomposition algorithm needs to iterate a few times.

A Technical Example

Here we assume that the reader is familiar with partial differential equations.

We will be solving the partial differential equation

$u_{xx} + u_{yy} = f$ (**)

The boundary condition is boundedness at infinity.

We decompose the domain R^2 into two overlapping subdomains $H_1 = (-\infty,1] \times R$ and $H_2 = [0,+\infty) \times R$. In each subdomain, we will be solving a BVP of the form:

$u^{(j)}{}_{xx} + u^{(j)}{}_{yy} = f$ in H_j

$u^{(j)}(x_j,y) = g(y)$

where $x_1 = 1$ and $x_2 = 0$ and taking boundedness at infinity as the other boundary condition. We denote the solution $u^{(j)}$ of the above problem by $S(f,g)$. Note that S is bilinear.

The Schwarz algorithm proceeds as follows:

1. Start with approximate solutions ${u^{(1)}}_0$ and ${u^{(2)}}_0$ of the PDE in subdomains H_1 and H_2 respectively. Initialize k to 1.
2. Calculate ${u^{(j)}}_{k+1} = S(f,{u^{(3-j)}}_k(x_j))$ with $j = 1,2$.
3. Increase k by one and repeat 2 until sufficient precision is achieved.

Balancing Domain Decomposition Method

In numerical analysis, the balancing domain decomposition method (BDD) is an iterative method to find the solution of a symmetric positive definite system of linear algebraic equations arising from the finite element method. In each iteration, it combines the solution of local problems on non-overlapping subdomains with a coarse problem created from the subdomain nullspaces. BDD requires only solution of subdomain problems rather than access to the matrices of those problems, so it is applicable to situations where only the solution operators are available, such as in oil reservoir simulation by mixed finite elements. In its original formulation, BDD performs well only for 2nd order problems, such elasticity in 2D and 3D. For 4th order problems, such as plate bending, it needs to be modified by adding to the coarse problem special basis functions that enforce continuity of the solution at subdomain corners, which makes it however more expensive. The BDDC method uses the same corner basis functions as, but in an additive rather than multiplicative fashion. The dual counterpart to BDD is FETI, which enforces the equality of the solution between the subdomain by Lagrange multipliers. The base versions of BDD and FETI are not mathematically equivalent, though a special version of FETI designed to be robust for hard problems has the same eigenvalues and thus essentially the same performance as BDD.

The operator of the system solved by BDD is the same as obtained by eliminating the unknowns in the interiors of the subdomain, thus reducing the problem to the Schur complement on the subdomain interface. Since the BDD preconditioner involves the solution of Neumann problems on all subdomain, it belongs to class of Neumann–Neumann methods, named so because they solve a Neumann problem on both sides of the interface between subdomains.

In the simplest case, the coarse space of BDD consists of functions constant on each subdomain and averaged on the interfaces. More generally, on each subdomain, the coarse space needs to only contain the nullspace of the problem as a subspace.

BDDC

In numerical analysis, BDDC (balancing domain decomposition by constraints) is a domain decomposition method for solving large symmetric, positive definite systems of linear equations that arise from the finite element method. BDDC is used as a preconditioner to the conjugate gradient method. A specific version of BDDC is characterized by the choice of coarse degrees of freedom, which can be values at the corners of the subdomains, or averages over the edges or the faces of the interface between the subdomains. One application of the BDDC preconditioner then combines the solution of local problems on each subdomains with the solution of a global coarse problem with the coarse degrees of freedom as the unknowns. The local problems on different subdomains are completely independent of each other, so the method is suitable for parallel computing. With a proper choice of the coarse degrees of freedom (corners in 2D, corners plus edges or corners plus faces in 3D) and with regular subdomain shapes, the condition number of the method is bounded when increasing the number of subdomains, and it grows only very slowly with the number of elements per subdomain. Thus the number of iterations is bounded in the same way, and the method scales well with the problem size and the number of subdomains.

History

BDDC was introduced by Dohrmann as a simpler primal alternative to the FETI-DP domain decomposition method by Farhat et al. The name of the method was coined by Mandel and Dohrmann, because it can be understood as further development of the BDD (balancing domain decomposition) method. The same method was also proposed independently by Fragakis and Papadrakakis under the name P-FETI-DP, and by Cros, which, however, was not recognised for some time. See for a proof that these are all actually the same method as BDDC. Mandel, Dohrmann, and Tezaur proved that the eigenvalues of BDDC and FETI-DP are identical, except for the eigenvalue equal to one, which may be present in BDDC but not for FETI-DP, and thus their number of iterations is practically the same. Much simpler proofs of this fact were obtained later by Li and Widlund and by Brenner and Sung.

Coarse Space

The coarse space of BDDC consists of energy minimal functions with the given values of the coarse degrees of freedom. This is the same coarse space as used for corners in a version of BDD for plates

and shells. The difference is that in BDDC, the coarse problem is used in an additive fashion, while in BDD, it is used a multiplicatively.

A Mechanical Description

The BDDC method is often used to solve problems from linear elasticity, and it can be perhaps best explained in terms of the deformation of an elastic structure. The elasticity problem is to determine the deformation of a structure subject to prescribed displacements and forces applied to it. After applying the finite element method, we obtain a system of linear algebraic equations, where the unknowns are the displacements at the nodes of the elements and the right-hand side comes from the forces (and from nonzero prescribed displacements on the boundary, but, for simplicity, assume that these are zero).

A preconditioner takes a right hand side and delivers an approximate solution. So, suppose we have an elastic structure divided into nonoverlappling substructures, and, for simplicity, suppose the coarse degrees of freedom are only subdomain corners. Suppose forces applied to the structure are given.

The first step in the BDDC method is the interior correction, which consists of finding the deformation of each subdomain separately given the forces applied to the subdomain except at the interface of the subdomain with its neighbours. Since the interior of each subdomain moves independently and the interface remains at zero deformation, this causes kinks at the interface. The forces on the interface necessary to keep the kinks in balance are added to the forces already given on the interface. The interface forces are then distributed to the subdomain (either equally, or with weights in proportion to the stiffness of the material of the subdomains, so that stiffer subdomains get more force).

The second step, called subdomain correction, is finding the deformation for these interface forces on each subdomain separately subject to the condition of zero displacements on the subdomain corners. Note that the values of the subdomain correction across the interface in general differ.

At the same time as the subdomain correction, the coarse correction is computed, which consists of the displacement at all subdomain corners, interpolated between the corners on each subdomain separately by the condition that the subdomain assumes the same shape as it would with no forces applied to it at all. Then the interface forces, same as for the subdomain correction, are applied to find the values

of the coarse correction at subdomain corners. Thus, the interface forces are averaged and the coarse solution is found by the Galerkin method. Again, the values of the coarse correction on subdomain interfaces is in general discontinuous across the interface.

Finally, the subdomain corrections and the coarse correction are added and the sum is averaged across the subdomain interfaces, with the same weights as were used to distribute the forces to the subdomain earlier. This gives the value of the output of BDDC on the interfaces between the subdomains. The values of the output of BDDC in the interior of the subdomains are then obtained by repeating the interior correction.

In a practical implementation, the right-hand-side and the initial approximation for the iterations are preprocessed so that all forces inside the subdomains are zero. This is done by one application of the interior correction as above. Then the forces inside the subdomains stay zero during the conjugate gradients iterations, and so the first interior correction in each application of BDDC can be omitted.

Coarse Space (Numerical Analysis)

In numerical analysis, coarse problem is an auxiliary system of equations used in an iterative method for the solution of a given larger system of equations. A coarse problem is basically a version of the same problem at a lower resolution, retaining its essential characteristics, but with fewer variables. The purpose of the coarse problem is to propagate information throughout the whole problem globally.

In multigrid methods for partial differential equations, the coarse problem is typically obtained as a discretization of the same equation on a coarser grid (usually, in finite difference methods) or by a Galerkin approximation on a subspace, called a coarse space. In finite element methods, the Galerkin approximation is typically used, with the coarse space generated by larger elements on the same domain. Typically, the coarse problem corresponds to a grid that is twice or three times coarser.

In domain decomposition methods, the construction of a coarse problem follows the same principles as in multigrid methods, but the coarser problem has much fewer unknowns, generally only one or just a few unknowns per subdomain or substructure, and the coarse space can be of a quite different type that the original finite element space, e.g. piecewise constants with averaging in balancing domain