Structural Renaissance:
a modern first course in engineering analysis and design

Thomas R Bewley

2025-04-12

Dedicated to Zachary & Nadia,
and in loving memory of
Morticia, Pareese, Morena, & Checkers, sunpups extraördinaire.

Contents

Preface

Apple Computer once advertised with the succinct slogan

Think Different.

Ralph Waldo Emerson once said

The mind, once stretched by a new idea, never returns to its original dimensions.

And, Clifford Stoll once said

Data is not information,
Information is not knowledge,
Knowledge is not understanding,
Understanding is not wisdom.

These three mantras form the foundational perspectives of this text. At the highest level, guided by study of the specific technical subjects presented (outlined below), this short text is really about **restructuring how one learns and thinks**, as explored further in §1; thus the double meaning of the word *Structure* in the title.

The technical aspects of this text are designed to augment courses traditionally titled *Structural Analysis*, *Statics*, *Mechanics & Materials*, and/or *Mechanical Design*, which are often encountered in freshmen/sophomore years[1] at major US colleges, universities, and military academies, while students are still refining their choice of major. The text succinctly introduces[2] and inter-relates a range of technical topics from a modern perspective, including Matrix/Vector Math (§2), Material Types and Properties (§3), Stresses in Loaded Beams (§4), and analysis of three essential types of Structures (Frames in §5, Trusses in §6, and Tensegrities in §7).

Structural Renaissance (*SR*) is the first in a series, to be followed by *Renaissance Robotics* (*RR*) and *Numerical Renaissance* (*NR*). The codes being developed for all three are designed to run in both Matlab and Octave (§A); these codes are free[3] and open source, as part of a GitHub codebase dubbed the *Renaissance Repository*. Please help us improve this effort by submitting bug fixes, broken links, and suggestions/typos[4] via this GitHub page.

Note finally that the zeitgeist of modern advances in science and engineering today is at the **intersection** of traditional disciplines; an interdisciplinary perspective is thus foundational to the *SR*/*RR*/*NR* series of texts[5].

---

[1] This text touches only briefly on second-order differential equations (when studying the problem of buckling, and observing the resulting sinusoidal deformations before failure of slender members under compressive load), and otherwise does *not* leverage integral or differential calculus; the material presented should thus also be readily accessible by motivated high school students.

[2] Introduces, but does *not* comprehensively cover, as it is designed to be digested in a single quarter, while motivating further study.

[3] All codes in the Renaissance Repository are copyright by the author, and are distributed under the BSD 3-Clause License.

[4] The two dots over the second vowel in common words like in naïve, Noël, and reëlect is called a *diaeresis*, which may be placed over a vowel to indicate that it is sounded in a separate syllable in situations that might otherwise be ambiguous. For example, adding "co" to "operative" gives a word which might easily be mispronounced if some form of diacritic is not used. One could suggest using a hyphen, but then adding a second prefix (as is often done in scientific writing) becomes problematic: both nonco-operative and non-co-operative are downright silly, but noncoöperative works fine. This series of texts, like the *New Yorker*, thus adopts a style that makes extensive use of diaereses. This approach is hopefully well received by students named Anaïs, Brontë, Chloë, Eloïse, Gaëlle, Joëlle, Maëlle, Zoë, Ismaël, Joël, Laocoön, Loïc, Maël, Noël, Raphaël, etc; to all others, please forgive this idiösyncrasy. :)

[5] The **Renaissance** was a creative and interdisciplinary scientific, artistic, and cultural movement, spanning from the 14th to the 17th centuries, which prioritized individual interests over those of the state, and was characterized by its emphasis on reason and critical thinking. This word thus forms a focal point, or a "North Star" of sorts, for the present series of texts.

It is perhaps useful to begin this (freshman/sophomore-level) study with a pointed question (not a joke...):

What is the difference between a Maker, a Mechanic, an AI, and an Engineer?

One possible answer that hits many of the essential points (see if you can state it better?) is:

An engineer does analysis-based deliberate design

to creatively, efficiently, safely, & aesthetically do useful new things while minimizing negative impact. That is:
• a Maker explores, inspirationally, what might be assembled and made functional,
• a Mechanic builds, largely applying his/her training in effective techniques for closely related problems,
• an AI matches patterns to tweak solutions (not necessarily correct or efficient) generated previously, while
• an Engineer **analyzes** and **designs** to generate new capabilities and efficiencies not previously attainable.

There are only 168 hours in a week, and 52 weeks in a year; how will you make maximum creative impact with your time? In high school, we are conditioned (with a range of rewards and punishments, notably including grades) that education largely involves training, repetition, short-term recall, and pattern matching, without particular emphasis on the "stretching" of our brains to accommodate, and to creatively apply, new ideas. Though our brief experiences as Makers and Mechanics are inspiring, training ourself to be memorizers of data and matchers of patterns (especially in the age of constant connectivity and AI) is a fool's errand.

For the purpose of seeing the world in new ways, identifying new opportunities, and analyzing / designing new things as an engineer, the ability to synthesize and comprehensively summarize a diverse range of knowledge across a broad range of disciplines is paramount, and requires a paradigm shift in how we learn, how we think, and how we do; this shift (see §1) is a central aim of the present series of texts, starting here.

To make certain we start off on the same page, we conclude this short preface with three overarching notions:

1. A key technical goal of this text is to help the reader to **recognize and exploit linear relationships**, which are ubiquitous in engineering systems.

2. The approach we take to the subjects considered is to **divide and conquer**. That is, once we see how to recognize sets of linear algebraic equations, how to standardize them into the form $A\mathbf{x} = \mathbf{b}$, and how to solve them (leading to zero, one, or infinite solutions $\mathbf{x}$), we will codify (that is, automate) this solution process so that we **don't solve linear equations by hand** anymore, instead shifting our focus to larger problems, including:
- how various materials fail under load,
- how individual structural members made from such materials should be designed (leveraging linear analysis tools) to minimize mass and cost while preventing failure under load, and
- how multiple structural members may be used together in three distinct design paradigms (frames, trusses, and tensegrities, again leveraging linear analyses) to efficiently and robustly bear loads in engineering systems.

3. By intent, this text is concise, to the point of being terse. It can not be successfully be read diagonally (that is, skimmed); the key topics it covers simply take time to digest, and such topics will at many points in this text be stated only once. Don't blink (especially in §2). Don't pattern match. Instead, by synthesizing the concepts you are learning, turn your creative energies to the much more rewarding problems of **analysis** and **design**. :)

If after all this you are still dialed in, plan your time accordingly (print this text, mark it up, ...). And, Welcome! I look forward to assisting you on this exciting journey in these consequential times.

Tom Bewley
Colorado Springs

Chapter 1

The Art of Teaching & Learning

1.1 Data, Information, Knowledge, Understanding, and Wisdom

The English language is replete with words capable of nuanced expression. Paradoxically, modern use of this rich language to discuss the Art of Teaching and Learning is inconsistent at best[1]. The purpose of this section is simply to clarify the language to be used in the remainder of this chapter, not to impose the various semantic distinctions made here upon others. We start by refining the language used in a working interpretation of the "Stoll Spectrum" highlighted in the Preface:

Data - Facts, largely devoid of context. Often can be stated with a few words, numbers, dates, etc.
Information - Data with context, as well as algorithms and methods (both mental and physical).
Knowledge - Information with perspective, including the ability to discern both good methods, as well as reliable/pertinent Information (in this age of rampant disinformation [11]), from those which are less so.
Understanding - Knowledge with judgement. Not just "what", but "why", and how other things relate.
Wisdom - Understanding with foresight. Ability to apply a balanced interpretation of things seen previously to new (yet, different) situations. "Seeing both sides", but still able to act decisively. Recognizing that the consequences of decisions are often far reaching, and usually not "zero-sum".

Many subjects (π, the U.S. Civil War, ...) can be framed at all five levels in this Spectrum; give it a try!

1.2 Education: Trainers, Instructors, Teachers, and Guides

In this work, I will use the word "educator" (and, "student") in the general sense, as people who strive to impart (and, to acquire) mastery of a given subject. Distinguishing the five levels articulated above is valuable when considering what educators (and, students) strive to accomplish in various focused learning environments, such as classrooms, homeschools, field trips, internships, apprenticeships, etc. An ancient proverb from Africa states

It takes a village to raise a child.

Education across all levels in the Stoll Spectrum is important. To borrow/modify a memorable quote by Yoda:

Data leads to Information, Information leads Knowledge,
Knowledge leads to Understanding, Understanding leads to Wisdom.

Without the lower levels in this succession, one does not have the foundation upon which to build to the higher levels. Ultimately, as suggested by Kant's *A Critique of Pure Reason* [7], Understanding and Wisdom must, in

[1] Many Conventional Texts on teaching (see, e.g., [2]) appear to miss the semantic distinctions of the type used herein, and the inevitable conclusions to which they lead.

fact, be based on Data and Information. When new **Disruptive Data & Information**[2] is obtained, or new **Disruptive Technology**[3] is introduced, our understanding of things, and our approaches to solving problems, must sometimes be modified fundamentally. That is, time spent experimenting and educating and learning at the lower levels in this Spectrum is foundational, but without a deliberate effort/plan to reason towards the higher levels, humanity in general, and students in particular, might never reach their full potential.

Furthering the Spectrum of semantic distinctions of §1.1, we also define the following types of educators:

Trainers enforce the memorization of Data, via a carrot (gold stars) or stick (demerits) approach,
Instructors model and drill to imprint Information, including physical Abilities and Skills, as well as standard approaches to Project Management (establishing requirements, budgets, schedules, etc.),
Teachers conduct discussions and exercises resulting in the structured acquisition of Knowledge, as well as the development of the critical thinking skills related to Logic and Reason, and
Guides ask deliberate, more open questions, leading students to refine an advanced Understanding of things, and building upon that to help students develop their own approaches to Creative Ideation [10].

Most educators are, at once, working on at least two of the above four levels. In contrast, Wisdom is a trait that can only flourish internally, over time; it is generally not a trait that can be taught by a **Guru**. Amongst other online resources, web searches ($\sim$10 seconds using Alexa/Siri devices), Wikipedia articles ($\sim$5 to 30 minute reads), and Operations Manuals (aka Dash Ones), provide terabytes upon terabytes of all sorts of Data and Information, some (but not all) of which has been vetted to some degree, and most of which (in small quantities) can indeed be referenced and internalized, at least for a time, by a sufficiently-motivated student without much assistance from an educator. Thus, the primary goals of Trainers and Instructors, as defined above, are simply to select the Data and Information to be learnt, to model the desired behavior, and to provide said motivation[4].

Since so much Data and Information is readily available online, and can reasonably be expected to remain so, the committing of Data and Information to (fallible) human memory, and the performing of computations using (fallible) human brains, is rapidly being supplanted by

a) the knowledge of how & where to access, reliably, significantly more such useful Data and Information, and
b) the formalization of both simple and complex algorithms as bug-free, easy-to-use computer codes.

Transferring your mental effort from your fallible human brain to (a) and (b) is a significant paradigm shift as you refine your engineering expertise. Note in particular that modern smartphones are at least 25x more powerful than the Cray-2, the most powerful computer on the planet in the late 1980s, and run at just a few watts. Calculators, like slide rules (and, the abacus that preceded them), simply no longer belong in a modern scientist

[2]Examples of Disruptive Data & Information include: (a) Galileo's development of the telescope, the data from which led to verification of the heliocentric theories of Copernicus, later refined by Kepler and Newton, in sharp contradiction to the dominant "wisdom" of the time (geocentricity, as suggested by the Bible and enforced by the doctrine of the Catholic Church), (b) Eddington's 1919 experimental observation of gravitational lensing (that is, the deflection of light rays by the mass of the sun, shifting the apparent position of stars near the sun during a solar eclipse), and (c) Hulse and Taylor's indirect verification of gravitational waves, by measuring (in 1974, and the several years that followed) how the orbital period of a binary pulsar changed over time, together with the 2015 direct measurements of gravitational waves by the Caltech/MIT Laser Interferometer Gravitational-wave Observatory, which measured truly minute undulations in spacetime caused by gravitational waves. Note that (b) and (c) validated parts of Einstein's Theory of General Relativity, which refined Newton's Law of Universal Gravitation, which replaced religious edict.

[3]Examples of Disruptive Technology include: (a) the development of transistors, calculators, and computers, which radically changed how calculations are performed, (b) the development of the internet and smartphones, and the persistent connectivity which they provide, which radically changed how Data and Information are stored and recalled, and (c) the development of Artificial Intelligence, which radically changed how large amounts of Data are processed and interpreted. These developments reduced the need for (fallible) humans to calculate, to store/recall, and to process Data, while at the same time empowering (creative) humans to address successively larger questions and problems.

[4]Think of the training "wax on, wax off", by Mr. Miyagi in the movie Karate Kid. Repetitive training of this sort is indeed useful to develop strength and to condition physical and mental reflexes (that is, to train Muscle Memory), but on its own does not contribute directly to Knowledge and Understanding.

or engineer's arsenal of tools; rather, "smart devices" that can easily be programmed (including laptops, tablets, and smartphones), with debugged and well-commented codes (including specifying units used, preferably SI), and (when possible) connected to the internet, have rendered them obsolete. So also with certain types of outdated Data-focused and Information-focused teaching and assessment methods[5], as explored in §1.4.

1.3 Career paths: Operations, Development, and Creative Research

Furthering the use of semantic distinctions from §1.1 and §1.2 to clarify our discussion, we define the following three broad types of career paths:

Operations leverages learnt Data and Information (see §1.1) to accomplish specific (oft, repetitive) tasks; in many settings, such tasks are **Dirty, Dull, & Dangerous**, and are thus natural targets for automation.
Development is the combination of existing relevant Knowledge, Solutions, and Technologies, to generate modified Solutions that meet somewhat different requirements than existing Solutions.
Research is the substantial creative extension of existing Knowledge, Solutions, and Technologies, to develop revolutionary new Solutions, with capabilities that were previously only dreamt of[6].

For career paths in Operations, all that is expected of a student's formal education is Training and Instruction (see §1.2). For example, at military academies and technical schools, students are trained how to shoot guns, how to fly planes, how to service motorcycles, how to MIG & TIG weld, etc. Individuals Trained and Instructed with such Data and Information are invaluable in their chosen craft, and to the corresponding military service, without developing any substantially original ideas related to the Art of Shooting, the Art of Flying, the Art of Motorcycle Maintenance, etc. At least, for now.

However, guns, planes, motorcycles, welding equipment, etc, are all changing rapidly with the advent of new automation technologies: guns and welding equipment are increasingly being mounted to robots, fighter jets are increasingly being remotely piloted, and/or accompanied by inexpensive Collaborative Combat Aircraft (CCAs) as force multipliers, motorcycles are increasingly being made electric, etc. Positioning oneself not only to Operate the dominant technologies of today, but also to help Develop and Research the technologies that will replace them in the not-too-distant future, is, ultimately, significantly more valuable than simply Instruction and Training on today's technologies.

The perspectives associated with Knowledge, and the judgement associated with Understanding, are thus inherently more valuable than Operations based just on Data and Information in the long run, as they facilitate:

- the Development of new solutions that build incrementally on existing solutions, and
- the Research (that is, the Creative Ideation) of "out-of-the-box" new solutions that will change the world.

Knowledge and Understanding are also progressively more delicate objectives for an educator to communicate and inspire [13, 5, 9], and for a student to develop. In contrast to the imprinting of short-term Muscle Memory with rote memorization of Data and Information[7], the Teaching and Guiding related to obtaining deeper Knowledge and Understanding of topics[8] that can be tied directly to various personal experiences and interests

[5] Examples of assessment methods that focus on the lower end of the Stoll Spectrum include multiple choice and fill-in-the-blanks, which focus on learnt Data, and performing such-and-such computations by hand, for algorithms which in practice should rightly be performed on a computer (like, most!). Assessment methods that focus on the upper end of this Spectrum are much more difficult to develop, and inherently more subjective to grade, and focus, for example, on (a) the interpretation of the results of such computations, (b) the consideration and development of algorithms that students in the class had not previously seen, etc.

[6] Examples (amongst many!) since 1900 include the development of: the radio, the Wright Flyer, the assembly line, penicillin, the Salk vaccine, the ballpoint pen, plastic, synthetic rubber, atomic & hydrogen bombs, fission-based nuclear power, solar & wind power, Sputnik, the SR-71, computers, the internet, smartphones, genetic engineering, the soft landing of Falcon 9 first-stage boosters, …

[7] For example, how to **sketch** a Bode plot by hand, if one doesn't have a computer available.

[8] For example, how to **use** a Bode plot (drawn by a computer) to **understand** the frequency response of an open-loop system

of students (e.g., how the dimmer switch in the classroom works, or how a cruise control in a car works, or how a One-Wheel skateboard can be kept from falling over, or how an autonomous robotic system can map an area and move within it without collisions, …) has more lasting impact, as such examples fundamentally retructure students' perspectives. This is well summarized by the Ralph Waldo Emerson quote in the Preface.

Note also that Guiding amounts to **Leadership** in learning environments. Effective Leadership in such environments requires a light touch. The essence of this notion is best captured in a quote by Lao Tzu:

A leader is best when people barely know he exists;
when his work is done, his aim fulfilled, they will say: we did it ourselves.

Finally, for Guiding to be effective, a Socratic method is ultimately needed, in which pointed questions are asked by the Guide, and the students find themselves regaining (see §1.4) their (lifelong) **individual responsibility** to build Knowledge and Understanding themselves, both within the classroom and beyond. [This shift of responsibility needs to be clearly articulated; far too often, it goes unsaid.] To accomplish this, Guides need to meet, connect, respond, and engage with the students, wherever they are in the maturity of their thinking. In short, to be effective as a Guide, one additional mantra is essential:

Be agile.

1.4 On the pitfalls of Assessment, and the restoration of Curiosity

John F Kennedy once inspired a nation with the immortal words

We choose to go to the Moon in this decade, and do the other things,
not because they are easy, but because they are hard.

In fact, young children are endowed with an innate curiosity and internal drive that naturally rise to such grand challenges. This is readily seen, e.g., in Montessori classrooms, in which, during quiet focused independent work periods, students will, without being prompted by a Guide, often challenge themselves with, e.g., massive by-hand calculations (long division, square roots, etc). In such settings, students fluidly progress from initial sensorial interactions with a wide range of inspiring physical materials[9], to (as they develop and internalize the appropriate mathematical abstractions) readily generalizable understandings of the key concepts that characterize them. Quite unfortunately, in most Conventional School systems, this innate curiosity and internal (virtually unlimited) drive to explore, characterize, and understand the world around us, and to build efficiently within it, is rapidly and exhaustively driven out of most young students. This represents a failure, in the highest degree, of meeting our objectives as educators, and demands our close scrutiny.

The blame for this failure rests squarely on educators who supposedly should know better, and the Conventional School systems within which most of us work. We are rewarded and punished (with awards, promotions, and funding, or the lack thereof) by these systems, and we pass this demoralizing framework of rewards and punishments (aka "Skinnerian conditioning"; see [8]) on to our students. The rewards and punishments that educators receive are often based on the outcomes of standardized tests by our students, which primarily measure the Data and Information[10] learnt (notably, not the time scale with which it will be retained), as well as

$G(s)$ and, when corresponding feedback $D(s)$ is used to fundamentally **change** the unfavorable dynamics of $G(s)$, to characterize accurately many important aspects of the step response of the corresponding closed-loop system $T(s) = G(s)\,D(s)/[1+G(s)\,D(s)]$. And, ultimately, how to **design** a controller $D(s)$ appropriately to robustly achieve the desired **system objectives**.

[9] For example, the Binomial and Trinomial Cubes are wooden cubes orthogonally cut into, respectively, $2^3 = 8$ and $3^3 = 27$ pieces of various sizes, which (ultimately) illustrate, tactilely, the 8 terms of $(x+y)^3$ and the 27 terms of $(x+y+z)^3$, as well as the 4 terms of $(x+y)^2$ and the 9 terms of $(x+y+z)^2$, making initial investigations of algebra significantly more concrete.

[10] In the language of this chapter, Data and Information are the quantities learnt that are most easily measured in an assessment (see footnote 5), though are most likely not the primary objectives of an advanced course.

student surveys, which measure the short-term satisfaction that students feel from the class[11].

This leads many educators to keep students in their "comfort zones", rather than guiding them into their "growth zones" (see [6] and [3]), and to "teach to the test" (i.e., the assessments that supposedly measure what has been accomplished in the classroom). And, it leads many students to "study / learn for the test" (i.e., to program their short-term memory accordingly, to strive for an "A" in the class, not to master a longer-term understanding of its underlying principles). This reduces many educators to act primarily as Trainers and Instructors (see §1.2), conveying predominantly premasticated Data and Information, and it reduces many students to act as short-term reservoirs for such premasticated content, as it is largely that which will be tested.

Evidence that, as educators in Conventional School systems, we have largely failed our central job, to inspire students towards Creative Ideation informed by advanced Knowledge and Understanding, is evident every time a student asks one of the following common questions:

1) What do I need to know to pass this class?
2) Will this be on the exam?
3) How can I maximize my GPA at this school, so I can get into my next school / job? [4]

The narrowing perspective represented by such questions is a direct result of the Conventional School systems to which our students have been subjected, which consistently focus them on getting an "A". That is, the primary blame for this limiting perspective does not actually fall to the students themselves, and it is highly detrimental towards the larger objective of gaining a generalizable understanding of the **Key Mathematical Tools** in college-level engineering, notably including:

Linear Algebra; Analysis of Structural Systems; Fourier, Laplace, & Z Transforms;
3D Dynamics; Programming; Numerical Methods; Linear Circuits; and Control Theory.

Zen and the Art of Motorcycle Maintenance [12] suggests

When you want to hurry something, that means you no longer care about it and want to get on to other things.

As educators, we need to strive to broaden the initially narrowing perspectives reflected by the above three questions. Our task is not to *answer* these limiting questions, which exacerbate the **Target Fixation** of students on getting an "A", but to *change the conversation*. Too often, we take shortcuts that validate such narrowing perspectives by, e.g., communicating correspondingly narrow **Learning Objectives** for each lecture or class. Such limiting Learning Objectives, in the process of making the course "easier" on the students (by limiting the scope of the Data and Information, and matchable **patterns**, that they "need" to pass the exams), serve to invalidate the importance of acquiring a broader Understanding of the subjects being studied.

It is critical that educators organize well the subjects being studied, presenting their essence with an elegantly clarifying notation (discussed further below), making relationships visual where possible, and tying the subjects presented together, and to specific applications (see §1.3) that motivate the students.

However, educators are often rewarded in student surveys, much more simply, by the degree to which they have premasticated the material presented, via identifying, through narrow Learning Objectives, specific Data and Information, and matchable patterns, that will be needed to pass the exams, thus making the course "easier", rather than challenging students to learn, more individually, how to chew and internalize big important subjects, one bite at a time. This challenge itself has significant value, which often goes unrecognized; in particular, it prepares students to digest the (even more difficult) Key Mathematical Tools (see above) that they will inevitably encounter next (either in a formal learning environment, or on their own).

The first essential step in broadening this initially narrowing perspective is thus:

Trust.

[11] Many student surveys reflect the students' immediate feeling of mastery of the Data & Information, and matchable patterns, presented in a course, as measured by their ability, in the short term, to recall this Data & Information, and to follow such patterns, on exams; they usually do *not* focus on the degree to which the educator has inspired them, in the long term, to *Think Different*.

By this, it is meant that care needs to be taken, with appropriate new mechanisms in place, such that:

• The students build trust in the educator, that the several foundational pieces being provided in class will coherently build towards a supporting framework (one might say, a "scaffold") upon which the students will ultimately be able to accomplish great and original things in engineering.
• The educator builds trust in the students, that their interest and perspectives are sufficiently broadened to focus on the "how" and the "why" of the subjects covered by the course, not just the narrow question of "what is the minimum Data and Information, and set of patterns, that will get me through the exam".
• The administration builds trust in the entire class (both the educator and the students), that they are working together to move beyond the Data and Information captured in most standardized tests, to generate longer-term generalizable Knowledge and Understanding of the essential course subjects.

As laid out in §1.2, in the present age of ready access to Data and Information online, and ready access to smart devices that can easily be programmed to execute both simple and complex algorithms as bug-free, easy-to-use (and, easy-to-verify) computer codes, the real-world significance of memorized Data and Information is substantially diminished, and the importance of Knowledge and Understanding, to accomplish substantially new things with Data and Information, and to build new Algorithms (solving bigger problems) upon existing, debugged Algorithms (solving smaller problems), is heightened.

As educators, our assessment methods, and to a large degree our teaching methods, which are often closely linked, have (for the most part) not yet undergone this same fundamental paradigm shift. Gottfried Leibniz, who lived long before machines[12] that could do substantial computations were invented, once said

It is unworthy of excellent men to lose hours like slaves in the labor of calculation
which could be relegated to anyone else if machines were used.

This quote gets directly at the question of how students can proceed to progressively deeper and deeper levels in their understanding, given the remarkable Technologies available today. It is not by repeatedly showing that students can do what machines should rightly be used for at that level. Rather students need to develop an Understanding of algorithms well enough to: (a) direct machines to do these (repetitive) tasks for them, (b) confirm that the answers produced by these machines are correct (trust, but verify), (c) know how and why such algorithms may fail, as well how and when they may be substantially accelerated, and, subsequently, (d) use the capabilities provided by these machines and algorithms to solve bigger problems.

Stated differently, as most educators and parents would now agree, it is important for primary students to first do arithmetic, until they thoroughly understand the process of doing arithmetic; once that is mastered, they should really not do arithmetic any longer. Rather, they should then upgrade to using calculators to do arithmetic, so they can begin to focus their mental efforts on bigger problems.

To proceed far in one's engineering studies, and fully realize one's potential, it is necessary to continue this trend to successively higher and higher levels of abstractions: it is similarly important for middle-school students to do algebra (first single-variable, then multi-variable), until they thoroughly understand the process of doing algebra; once that is mastered, they should not do algebra any longer. Rather, they should then use symbolic tools on computers to do algebra[13], as required frequently in engineering, so they can focus their mental efforts on even bigger problems.

Continuing this trend: it is similarly vital for high-school students to do differential and integral calculus and infinite sums and partial fraction expansions, until they thoroughly understand the process of doing these

[12] Indeed, the word "computer" in English, today, describes machines that do computations. Before such machines (first mechanical, then electromechanical, then solid state) were developed, "computers" were in fact humans, who performed often tedious computations, by hand, for others.

[13] Once the use of computers is adopted to do algebra (and, to execute other algorithms that are easily automated), one's calculator should be passed along to a younger sibling, or a neighbor's kid. As discussed in the last paragraph of §1.2, calculators are difficult to program, and the use of them is highly prone to typos; they thus have no place in a modern college student's backpack.

things; once they are mastered, they should no longer do these things by hand. Rather, they should then use computational tools to compute integrals and infinite sums (or, where appropriate, use tables of Laplace and Z transforms) as well as partial fraction expansions, so they can begin to focus their efforts on still bigger problems (e.g., in designing effective single-input-single-output transfer-function-based controllers).

Continuing even further: it is important for college students to do linear algebra, until they thoroughly understand the problem of linear equations ($A\mathbf{x} = \mathbf{b}$) and how to solve them ($\mathbf{x} = A^{-1}\mathbf{b}$ or $\mathbf{x} = A\backslash\mathbf{b}$ or $\mathbf{x} = A^{+}\mathbf{b}$, where $A^{+} = \underline{V\Sigma}^{-1}\underline{U}^{H}$), and how to use the related fundamental matrix decompositions ($A = LU$, $A = QR$, $A = S\Lambda S^{-1}$, $A = UTU^{H}$, $A = U\Sigma V^{H} = \underline{U\Sigma V}^{H}$); once these relationships are well understood, students should first check, then trust, computer programs that perform these useful linear algebraic computations, so they can then confidently use such linear algebraic tools (there are many) when solving even bigger problems (e.g., in designing effective multiple-input-multiple-output state-space controllers).

And so on. That is, once a core underlying subject (arithmetic, algebra, integral calculus, infinite sums, partial fraction expansions, linear algebra, …) is well understood (but, *not a moment before!*), machines should then be employed (once the correctness of the results generated by the machine is verified) to solve that class of problems in the future, so students can move to successively bigger problems.

Two additional relevant quotes by Gottfried Leibniz[14] relate to the central importance of **notation**:

It is worth noting that the notation facilitates discovery.
This, in a most wonderful way, reduces the mind's labour.

In symbols one observes an advantage in discovery which is greatest when they express the exact nature of a thing briefly and, as it were, picture it; then indeed the labor of thought is wonderfully diminished.

At the outset of their studies, the following advice for engineering students helps them to focus appropriately:

- Mathematical abstractions (negative, irrational, & imaginary numbers; sines & cosines; polynomial roots; frequency; latitude & longitude; yaw/pitch/roll; potential & kinetic energy; etc) empower clear engineering thinking, and help us understand and solve important new problems; be prepared to embrace several new ones (quaternions; filters; Fourier, Laplace, & Z transforms; eigen decompositions; etc) as the needs for them arise.
- Think beyond the numbers; understand new relationships with symbols.
- Once you master a new relationship, immortalize this new understanding with a clear, generalizable code.

1.5 Synopsis

To encourage students to think for themselves, the following advice is warranted:

Don't follow instructions[15].

It is useful for educators to provide personal real-life examples of when they followed less-than-clever published instructions themselves, and "disaster" ensued. Students resonate well with such stories, of learning from one's own mistakes. Students also engage intently with stories of various more famous mistakes in engineering (for example: the Tacoma Narrows Bridge, Francis Scott Key Bridge, Challenger, Mars Climate Orbiter, United Airlines 232, Air France 447, AeroVironment Helios, …), and the role that engineers play in properly designing and analyzing, and loudly calling out all possible failure modes, of such systems, so that such disasters do not occur. To encourage students to transition to **constructive interrogations**, rather than questions reflecting a **narrowing perspective** (see §1.4), a Guide can illustrate by example [1], and students quickly pick it up:

[14] Of course, the following quote from the time he lived *about* Gottfried Leibniz helps put his contributions in perspective: *It is rare to find learned men who are clean, do not stink, and have a sense of humor.* I am not clear whether this anonymous quote implies that Leibniz never bathed, or that he was humorless (if you know, please contact me). Given his enlightened contributions to both calculus as well as the binary number system that underlies all modern computers, amongst other things, I suspect the former…

[15] That is, when those instructions are unlawful, dangerous, or less than clever.

• How can we understand this problem differently?
• How can we change this system's dynamics favorably? That is, how can we make this system: lighter, stronger, faster, cheaper, more reliable, more efficient, more stable, more agile, ...
• How can I help you to benefit most from your remaining time at this school/college/university/academy?

Maria Montessori suggested that, to teach primary students effectively, one must *Follow the Child*. Recalling the comment from §1.3, about tying examples in class to the personal experiences and interests of students in their everyday world, this general notion can be applied across all levels of education: one should follow the students' interests to capture their attention, but then adroitly guide the students with leading questions that they might not at first anticipate (regarding automation, agility, efficiency, failure, robustness, fault tolerance, material selection, durability, manufacturing cost, operating cost, environmental impact, etc).

It is important to remain cognizant of the fact that, at the more advanced levels (in college and beyond, in the "real world"), technology plays an increasingly important role in the everyday work of engineers (doing arithmetic, algebra, integrals, sums, partial fraction expansions, linear algebra, root locus plots, Bode plots, etc, on smart devices, thus disengaging these lower-level math problems from the work that engineers need to do). Our classroom assessment methods must begin to de-emphasize both Data & Information, as well as computations that are better left to computers, to best mimic such "real engineering scenarios".

Returning to §1.1, note also that one does not proceed linearly from Data, to Information, to Knowledge, to Understanding (and, eventually, to Wisdom) as one gets older. Quite the contrary, we repeatedly Spiral over all of these levels in different subjects. Maria Montessori's teachings focus on developing student-connected Guides to lead Agile teams of primary children. Educators in freshman-level college classes and beyond would do well to take a cue from that simpler time, before the innate curiosity and internal drive of students were driven out by an imperfect educational system, punishing and rewarding with unilluminating Data-focused and "human computer" educational assessments, and to explore every connection that can be made with students to restore their natural human mindset towards characterizing and understanding the world around us, and creatively designing advanced engineering systems to robustly perform useful tasks within it.

References

[1] Adams, James (2019) Conceptual Blockbusting: A Guide to Better Ideas 1-7

[2] Ambrose, Susan, et al (2010) How Learning Works: 7 Research-Based Principles for Smart Teaching 1-1

[3] Boaler, Jo (2022) Limitless Mind: Learn, Lead, and Live Without Barriers 1-5

[4] Bruni, Frank (2016) Where You Go Is Not Who You'll Be: An Antidote to the College Admissions Mania 1-5

[5] Catmull, Ed (2023) Creativity Inc: Overcoming the Unseen Forces That Stand in the Way of True Inspiration 1-3

[6] Godin, Seth (2007) The Dip: A Little Book That Teaches You When to Quit (and When to Stick) 1-5

[7] Kant (1787) A Critique of Pure Reason 1-1

[8] Kohn, Alfie (2018) Punished By Rewards: The Trouble with Gold Stars, Incentive Plans, A's, Praise, and Other Bribes 1-4

[9] March, James (2020) The Technology of Foolishness 1-3

[10] McKim, Robert (1980) Experiences in Visual Thinking 1-2

[11] Nichols, Tom (2024) The Death of Expertise: The Campaign against Established Knowledge and Why It Matters 1-1

[12] Pirsig, Robert (2006) Zen and the Art of Motorcycle Maintenance: An Inquiry into Values 1-5

[13] Robinson, Ken (2011) Out of Our Minds: Learning to Be Creative; see also his TED talk. 1-3

Chapter 2

Matrix/Vector Math

2.1 Introduction to linear algebraic equations and their solution(s)

Solution of the following system of 3 linear equations (i.e., finding $\{x, y, z\}$) is trivial, as they are decoupled:

$$\begin{aligned} 2\,x \qquad\qquad &= 3 \\ 3\,y \qquad &= 9 \\ 4\,z &= 8 \end{aligned} \quad\Leftrightarrow\quad \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 4 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 3 \\ 9 \\ 8 \end{pmatrix} \quad\Rightarrow\quad \begin{aligned} x &= 3/2 \\ y &= 3 \\ z &= 2 \end{aligned} \tag{2.1}$$

The matrix/vector form of these equations, often denoted $A\mathbf{x} = \mathbf{b}$, is given at center; A in this case is **diagonal**.

Now consider the solution (i.e., finding $\{x_1, x_2, x_3\}$) of the following system of linear equations:

$$\begin{aligned} 2\,x_1 + 3\,x_2 + 4\,x_3 &= 25 \\ 2\,x_2 + 5\,x_3 &= 16 \\ 4\,x_3 &= 8 \end{aligned} \quad\Leftrightarrow\quad \begin{pmatrix} 2 & 3 & 4 \\ 0 & 2 & 5 \\ 0 & 0 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 25 \\ 16 \\ 8 \end{pmatrix} \quad\Rightarrow\quad \begin{aligned} x_1 &= 4 \\ x_2 &= 3 \\ x_3 &= 2 \end{aligned} \tag{2.2}$$

Note the standard notation used in this example, in which the unknowns $\{x_1, x_2, x_3\}$ in the $A\mathbf{x} = \mathbf{b}$ form are numbered as the $n = 3$ elements of the vector of unknowns, $\mathbf{x}$. The matrix A in this case is **triangular**, with zeros in the lower-triangular elements. This 3×3 system is almost as easy to solve as (2.1). The process, called **backsubstitution**, proceeds as follows: solve the last eqn for x_3, substitute the result for x_3 into the previous eqn and solve for x_2, and substitute the results for $\{x_2, x_3\}$ into the first eqn and solve for x_1. Larger $(n \times n)$ systems, with symbolic, real (floating point), or complex elements, follow the same solution process, starting from the n'th element and working back up to the first, and the process is trivial to automate (at least, if the diagonal elements of A are nonzero, a challenge called **singularity** which is addressed further below).

Now consider a problem of the same size, but in which A has no readily exploitable sparsity structure:

$$\begin{aligned} 1\,x_1 + 2\,x_2 + 3\,x_3 &= 10 \\ 4\,x_1 + 5\,x_2 + 6\,x_3 &= 28 \\ 7\,x_1 + 8\,x_2 \qquad\quad &= 37 \end{aligned} \quad\Leftrightarrow\quad \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 10 \\ 28 \\ 37 \end{pmatrix} \quad\Leftrightarrow\quad \left[\begin{array}{ccc|c} 1 & 2 & 3 & 10 \\ 4 & 5 & 6 & 28 \\ 7 & 8 & 0 & 37 \end{array}\right] \tag{2.3a}$$

Note the **augmented matrix notation** introduced in this example above right; by definition, this representation is equivalent to the two forms at left, summarizing all of the essential elements defining this system of equations, but consuming less space. The matrix A in this case is **full**. The (disorderly) "high-school" solution approach, of "plugging one equation into another" ad nauseam, is error-prone. Better is the (orderly) "row-wise" solution process called **Gaussian Elimination**, which we demonstrate by example below.

The goal of the Gaussian Elimination procedure, after a series of simple row combinations, is to transform the set of equations to an (equivalent, but modified) upper triangular matrix form, which as seen in (2.2) is then simple to solve with backsubstitution.

To proceed in the (2.3a) example above, simply

- add -4 times the 1st equation to the 2nd equation, replacing the 2nd equation in the set with the result:

$$\begin{array}{r} -4\cdot\{1\,x_1+2\,x_2+3\,x_3=10\} \\ +\{4\,x_1+5\,x_2+6\,x_3=28\} \\ \hline \{0\,x_1-3\,x_2-6\,x_3=-12\} \end{array} \quad \text{and thus} \quad \left[\begin{array}{ccc|c} 1 & 2 & 3 & 10 \\ 4 & 5 & 6 & 28 \\ 7 & 8 & 0 & 37 \end{array}\right] \Rightarrow \left[\begin{array}{ccc|c} 1 & 2 & 3 & 10 \\ 0 & -3 & -6 & -12 \\ 7 & 8 & 0 & 37 \end{array}\right] \tag{2.3b}$$

The resulting set of equations, as shown above right with the modified values in red, is **completely equivalent** to the original set of equations, as the modified second equation is simply the original second equation with something true (specifically, -4 times the first equation) added. To continue,

- add -7 times the 1st equation to the 3rd equation, replacing the 3rd equation in the set with the result, and
- add -2 times the new 2nd equation to the new 3rd equation, replacing the 3rd equation with the result:

$$\left[\begin{array}{ccc|c} 1 & 2 & 3 & 10 \\ 0 & -3 & -6 & -12 \\ 7 & 8 & 0 & 37 \end{array}\right] \Rightarrow \left[\begin{array}{ccc|c} 1 & 2 & 3 & 10 \\ 0 & -3 & -6 & -12 \\ 0 & -6 & -21 & -33 \end{array}\right] \Rightarrow \left[\begin{array}{ccc|c} 1 & 2 & 3 & 10 \\ 0 & -3 & -6 & -12 \\ 0 & 0 & -9 & -9 \end{array}\right] \Rightarrow \begin{array}{l} x_1=3 \\ x_2=2 \\ x_3=1 \end{array} \tag{2.3c}$$

As the last augemented matrix form above has a triangular A, we just use backsubstitution to solve, as in (2.2). The solution of larger full ($n \times n$) systems, with symbolic, real (floating point), or complex elements, follow the same Gaussian Elimination process, which is again straightforward to automate.

There are **two remaining challenges**. The first is that, during the Gaussian Elimination process, which modifies A to a triangular form while still representing a relationship between the unknowns that is equivalent to the original set of eqns, there may be a **zero element** in an inconvenient place. In particular, when attempting at a certain step in this process to add a multiple α of row i to row $j > i$, in order drive the i'th element of row j to zero, there might already be a zero in the i'th element of row i, and thus no value of α will do the necessary job. The solution is simply to first swap row i with one of the rows below it which is nonzero in the i'th element. This is again best illustrated by example, the notation of which should by now be self evident (the challenge in this particular example, and the row swapping solution implemented, is at the outset, when $i = 1$ and $j = 2$):

$$\begin{array}{r} 4\,x_2+5\,x_3=6 \\ 2\,x_1+1\,x_2\qquad\;\;=0 \\ 8\,x_1+8\,x_2+7\,x_3=8 \end{array} \Leftrightarrow \left[\begin{array}{ccc|c} 0 & 4 & 5 & 6 \\ 2 & 1 & 0 & 0 \\ 8 & 8 & 7 & 8 \end{array}\right] \Leftrightarrow \left[\begin{array}{ccc|c} 2 & 1 & 0 & 0 \\ 0 & 4 & 5 & 6 \\ 8 & 8 & 7 & 8 \end{array}\right] \Leftrightarrow \left[\begin{array}{ccc|c} 2 & 1 & 0 & 0 \\ 0 & 4 & 5 & 6 \\ 0 & 4 & 7 & 8 \end{array}\right] \Leftrightarrow \left[\begin{array}{ccc|c} 2 & 1 & 0 & 0 \\ 0 & 4 & 5 & 6 \\ 0 & 0 & 2 & 2 \end{array}\right] \tag{2.4}$$

and thus $x_1 = -1/8$, $x_2 = 1/4$, $x_3 = 1$. This row swapping process is referred to as **partial pivoting**.

BTW, you might be wondering at this point how much of this process you should be able to do by hand. The practical answer may surprise you, if you haven't yet read §1: **none of it!** You will always have a computer available when you do engineering. Solutions to (2.1) to (2.4), and larger problems with similar structure, are easily generated using carefully-written numerical codes. For instance, copy/paste the following lines in Matlab:

```
>> A=[2 0 0; 0 3 0; 0 0 4], b=[ 3; 9; 8], x=A\b, A*x % problem (2.1)
>> A=[2 3 4; 0 2 5; 0 0 4], b=[25; 16; 8], x=A\b, A*x % problem (2.2)
>> A=[1 2 3; 4 5 6; 7 8 0], b=[10; 28; 37], x=A\b, A*x % problem (2.3)
>> A=[0 4 5; 2 1 0; 8 8 7], b=[ 6; 0; 8], x=A\b, A*x % problem (2.4)
```

To see exactly how fast your computer can solve a very large problem of this sort, try the following in Matlab:

```
>> A=rand(10000,10000); b=rand(10000,1); tic, x=A\b; toc, norm(A*x-b)
```

For example, on a 2024 Mac M3 laptop, this 10000×10000 problem is solved, **without the possibility of human error**, in about 1.8 seconds (as measured by the tic and toc commands shown), with a total error norm of about 10^{-9}. You would never be able to solve a problem even a fraction of that size by hand without error.

On the previous two pages, you learned (by example) how to write a set of n linear equations in n unknowns as $A\mathbf{x} = \mathbf{b}$, and how the full Gaussian Elimination process with partial pivoting, together with backsubstition, enables you to solve for the unknowns $\{x_1, x_2, \ldots, x_n\}$ quite easily, using a computer[1], even for large n and complicated (but, known) entries in A and $\mathbf{b}$, as long as A is nonsingular. We noted that, in engineering practice, you should never compute that algorithm by hand, which is a process that, though well structured, is still prone to human error, even for relatively small n. Instead, simply set up A and $\mathbf{b}$ on a computer, and calculate `A\b`.

The second remaining challenge is the possible **singularity** of A. This is illustrated via example below.

$$\begin{aligned} 1\,x_1 + 2\,x_2 + 3\,x_3 &= 11 \\ 4\,x_1 + 5\,x_2 + 6\,x_3 &= 29 \\ 7\,x_1 + 8\,x_2 + 9\,x_3 &= 47 \end{aligned} \quad\Leftrightarrow\quad \left[\begin{array}{ccc|c} 1 & 2 & 3 & 11 \\ 4 & 5 & 6 & 29 \\ 7 & 8 & 9 & 47 \end{array}\right] \quad\Leftrightarrow \ldots \Leftrightarrow\quad \left[\begin{array}{ccc|c} 1 & 2 & 3 & 11 \\ 0 & -3 & -6 & -15 \\ 0 & 0 & 0 & 0 \end{array}\right] \quad\Leftrightarrow\quad \left[\begin{array}{ccc|c} 1 & 2 & 3 & 11 \\ 0 & 1 & 2 & 5 \\ 0 & 0 & 0 & 0 \end{array}\right] \qquad (2.5a)$$

You should by now be able to fill in the intermediate steps yourself (though, again, in actual engineering practice you will always use a computer). Note first that, in the form at right, the second row was simply divided by its nonzero diagonal element, somewhat simplifying the (equivalent) final form. Note also the trivial final row. The implication of this is that x_3 may be selected *arbitrarily*. It then follows from the second line that $x_2 = 5 - 2x_3$, and from the first line that $x_1 = 11 - 2x_2 - 3x_3 = 1 + x_3$. Thus, $\{x_1, x_2, x_3\} = \{1, 5, 0\}$ and $\{2, 3, 1\}$ are two (of the **infinite**) possible solutions

Another system of equations, with the same matrix A but slightly different RHS, dubbed $\tilde{\mathbf{b}}$, is

$$\begin{aligned} 1\,x_1 + 2\,x_2 + 3\,x_3 &= 11 \\ 4\,x_1 + 5\,x_2 + 6\,x_3 &= 29 \\ 7\,x_1 + 8\,x_2 + 9\,x_3 &= 48 \end{aligned} \quad\Leftrightarrow\quad \left[\begin{array}{ccc|c} 1 & 2 & 3 & 11 \\ 4 & 5 & 6 & 29 \\ 7 & 8 & 9 & 48 \end{array}\right] \quad\Leftrightarrow \ldots \Leftrightarrow\quad \left[\begin{array}{ccc|c} 1 & 2 & 3 & 11 \\ 0 & -3 & -6 & -15 \\ 0 & 0 & 0 & 1 \end{array}\right] \quad\Leftrightarrow\quad \left[\begin{array}{ccc|c} 1 & 2 & 3 & 11 \\ 0 & 1 & 2 & 5 \\ 0 & 0 & 0 & 1 \end{array}\right] \qquad (2.5b)$$

Looking at the last row now, it is seen that there are **zero** sets of $\{x_1, x_2, x_3\}$ that solve this set of equations.

A zero on the main diagonal may appear earlier during the Gaussian Elimination process, before getting to the final row. A minor tweak on this process converts the augmented matrix $[A|\mathbf{b}]$ (that is, the matrix A with the column $\mathbf{b}$ tacked on to the right) to what is called **row echelon form**, where (a) all rows consisting only of zeros are at the bottom, and (b) the "leading" nonzero entry of each non-zero row, dubbed the "pivot", is a "1", and is to the right of the pivot in the row immediately above it. We can further simplify a matrix in row echelon form by subtracting the appropriate multiples of each nonzero row from the rows *above* it, resulting in what is known as a **reduced row echelon form** where, additionally, (c) each pivot is the only nonzero entry in its column. Given this, it is straightforward to parameterize *all* solutions of any singular $A\mathbf{x} = \mathbf{b}$ problem: simply select the elements of $\mathbf{x}$ corresponding to each nonpivot column of the modified $[A|\mathbf{b}]$ (in reduced row echelon form) arbitrarily, then solve the nonzero rows of the modified $[A|\mathbf{b}]$ for the elements of $\mathbf{x}$ corresponding to the pivot columns of A. However, this method of parameterizing all solutions of an $A\mathbf{x} = \mathbf{b}$ problem is not nearly as illuminating as that discussed next, so you can/should mostly ignore this paragraph (apologies…).

To summarize §2.1, we have seen that, in general, there may be **zero**, **one**, or **infinite** vectors $\mathbf{x}$ that solve a given $A\mathbf{x} = \mathbf{b}$ problem. Gaussian elimination (`x=A\b`) is the go-to tool to determine $\mathbf{x}$ in the case that there is one solution, and may be extended to address the cases with zero or infinite solutions. However, to understand better the latter two cases, a different viewpoint is suggested, as laid out in §2.3.

[1]The process of automating Gaussian Elimination with partial pivoting in computer code is developed from scratch in §2 of *NR*, and is **beyond the scope** of the present introduction, as Matlab's built-in \ routine works just fine. In addition to generating the $\mathbf{x}$ that solves $A\mathbf{x} = \mathbf{b}$ for a given $\{A, \mathbf{b}\}$, assuming A is square and nonsingular, the full algorithm generates three matrices $\{P, L, U\}$ such that $A = P^T LU$, where U is upper triangular, L is unit lower triangular (that is, lower triangular, with ones on the main diagonal), and P is a permutation matrix (that is, an identity matrix with several row swaps applied); this is referred to as a **fundamental matrix decomposition**, and is sometimes useful. If partial pivoting is not used, the P matrix in this decomposition is just the identity, and may be dropped. To demonstrate, try the following in Matlab: `>> format short, A=randi(10,10), [L,U,P]=lu(A), norm(A-P'*L*U)`

2.2 Vector spaces, subspaces, and orthogonal complements

A brief review of vector calculus is now useful. A **vector** is a directed line segment; it has both a magnitude and a direction. An nD real or complex vector (in 2D, 3D, 4D, ...) has $n = 2$, $n = 3$, or $n > 3$ real or complex elements; the set of all real or complex nD vectors is referred to as $\mathbb{R}^n$ or $\mathbb{C}^n$, respectively. Vectors can have numeric or symbolic elements. A matrix can be thought of as a collection of vectors in its columns.

With the appropriate addition and scalar multiplication operations defined, a **vector space** is the set of all vectors that may be reached by **linear combination** of 0, 1, 2, 3, or more **spanning** vectors (i.e., the space is said to be **spanned** by those vectors), where the scalars (denoted α_i below) in the linear combinations are taken as the same type (e.g., real or complex) as the elements of the vectors in the vector space itself; for example,

$$\text{if } \mathsf{C} = \text{span}\left\{\begin{matrix} 1 & 2 \\ 4 & 5 \\ 7 & 8 \end{matrix}\right\}, \text{ then all } \mathbf{y} \in \mathsf{C} \text{ may be written as } \mathbf{y} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + \alpha_1 \begin{pmatrix} 1 \\ 4 \\ 7 \end{pmatrix} + \alpha_2 \begin{pmatrix} 2 \\ 5 \\ 8 \end{pmatrix} \tag{2.6a}$$

where α_i parameters are real and arbitrary. A vector space is **closed**; that is, any linear combination of vectors in a vector space is also in that vector space. All vector spaces contain at least the **zero vector** (a vector with all elements equal to zero). The set of all real or complex nD vectors, defined above as $\mathbb{R}^n$ or $\mathbb{C}^n$, are vector spaces.

A **subspace** is a subset of a vector space that itself is also a vector space. Noting (2.6a), the expression $\mathsf{C} \subset \mathbb{R}^3$ may be used to indicate that the vector space C is a subspace of $\mathbb{R}^3$; geometrically, it may be visualized as the plane in 3D that contains the origin and the two column vectors in the curly brackets in (2.6a).

Any set of r vectors forming a **basis** that spans a vector space is **independent** (that is, none in the basis may be expressed as a linear combination of the others). An **orthogonal basis** is a basis such that all basis vectors are **mutually orthogonal** (that is, such that the dot product between any two vectors forming the basis is zero). An **orthonormal basis** is an orthogonal basis such that the length of each basis vector is 1.

The **orthogonal complement** $\mathsf{C}^\perp$ of a vector space C is the set of all vectors $\mathbf{y}$ such that each $\mathbf{y}_{\mathsf{C}^\perp} \in \mathsf{C}^\perp$ is orthogonal to all $\mathbf{y}_\mathsf{C} \in \mathsf{C}$, and thus each $\mathbf{y}_\mathsf{C} \in \mathsf{C}$ is orthogonal to all $\mathbf{y}_{\mathsf{C}^\perp} \in \mathsf{C}^\perp$. If C and $\mathsf{C}^\perp$ contain vectors in $\mathbb{R}^m$ (or $\mathbb{C}^m$), than any $\mathbf{y} \in \mathbb{R}^m$ (or $\mathbb{C}^m$) may be uniquely **orthogonally decomposed** such that $\mathbf{y} = \mathbf{y}_\mathsf{C} + \mathbf{y}_{\mathsf{C}^\perp}$ for some $\mathbf{y}_\mathsf{C} \in \mathsf{C}$ and some $\mathbf{y}_{\mathsf{C}^\perp} \in \mathsf{C}^\perp$.

For the example given in (2.6a), an orthogonal basis and an orthonormal basis of C, and a basis of $\mathsf{C}^\perp$, are

$$\mathsf{C} = \text{span}\left\{\begin{matrix} 1 & 3 \\ 4 & 1 \\ 7 & -1 \end{matrix}\right\} = \text{span}\left\{\begin{matrix} 1/\sqrt{66} & 3/\sqrt{11} \\ 4/\sqrt{66} & 1/\sqrt{11} \\ 7/\sqrt{66} & -1/\sqrt{11} \end{matrix}\right\}, \qquad \mathsf{C}^\perp = \text{span}\left\{\begin{matrix} 1 \\ -2 \\ 1 \end{matrix}\right\}. \tag{2.6b}$$

Note (confirm yourself!) that the vectors that span C are perpendicular to the vector that spans $\mathsf{C}^\perp$. Also:

$$\begin{pmatrix} 3 \\ 6 \\ 9 \end{pmatrix} = -1 \begin{pmatrix} 1 \\ 4 \\ 7 \end{pmatrix} + 2 \begin{pmatrix} 2 \\ 5 \\ 8 \end{pmatrix} = \frac{15}{11} \begin{pmatrix} 1 \\ 4 \\ 7 \end{pmatrix} + \frac{6}{11} \begin{pmatrix} 3 \\ 1 \\ -1 \end{pmatrix} \quad \text{and} \quad \mathbf{b} = \begin{pmatrix} 11 \\ 29 \\ 47 \end{pmatrix} = \frac{76}{11} \begin{pmatrix} 1 \\ 4 \\ 7 \end{pmatrix} + \frac{15}{11} \begin{pmatrix} 3 \\ 1 \\ -1 \end{pmatrix} \tag{2.6c}$$

(confirm yourself). The relations above show that, in (2.5a), the third column of A and the vector $\mathbf{b}$ are spanned by the first two columns of A (and we thus say that the **rank** of A is 2). In contrast,

$$\tilde{\mathbf{b}} = \begin{pmatrix} 11 \\ 29 \\ 48 \end{pmatrix} = \mathbf{y}_\mathsf{C} + \mathbf{y}_{\mathsf{C}^\perp} \quad \text{where} \quad \mathbf{y}_\mathsf{C} = \frac{463}{66} \begin{pmatrix} 1 \\ 4 \\ 7 \end{pmatrix} + \frac{14}{11} \begin{pmatrix} 3 \\ 1 \\ -1 \end{pmatrix} \quad \text{and} \quad \mathbf{y}_{\mathsf{C}^\perp} = \frac{1}{6} \begin{pmatrix} 1 \\ -2 \\ 1 \end{pmatrix} \tag{2.6d}$$

(confirm yourself). That is, in (2.5b), the vector $\tilde{\mathbf{b}}$ can *not* be written simply as linear combination of the columns of A, but also contains a component from $\mathsf{C}^\perp$. In §2.3, we illustrate graphically the subspaces that arise from such $A\mathbf{x} = \mathbf{b}$ problems, and summarize how they relate to the solvability (and, to the uniqueness of solutions) of $A\mathbf{x} = \mathbf{b}$, and identify a simple way to calculate their orthogonal bases.

2.3 The four fundamental subspaces defined by A

At this point, for any rectangular[2] $A_{m\times n}$, it is useful to review more concretely what matrix/vector multiplication actually achieves. Viewed one way (with a **row-wise** interpretation of A), the i'th element of $\mathbf{b}$ is simply the **dot product** of row i of A with the vector $\mathbf{x}$; viewed the other way (with a **column-wise** interpretation of A), the vector $\mathbf{b}$ is seen to be a **linear combination** of the columns of A with the elements of $\mathbf{x}$ as weights:

$$A\mathbf{x}=\mathbf{b} \;\Leftrightarrow\; \sum_{j=1}^{n} a_{ij}x_j = b_i \text{ for } i=1,\dots,m \;\Leftrightarrow\; \begin{pmatrix} a_{11}\\ \vdots \\ a_{m1}\end{pmatrix} x_1 + \begin{pmatrix} a_{12}\\ \vdots \\ a_{m2}\end{pmatrix} x_2 + \dots + \begin{pmatrix} a_{1n}\\ \vdots \\ a_{mn}\end{pmatrix} x_n = \begin{pmatrix} b_1\\ \vdots \\ b_m\end{pmatrix} \tag{2.7a}$$

Note that a_{ij} refers to the element in the i'th row and j'th column of A. For the 3×3 example in (2.5a):

$$A\mathbf{x}=\mathbf{b} \quad\Leftrightarrow\quad \begin{matrix} 1\cdot x_1 + 2\cdot x_2 + 3\cdot x_3 = 11\\ 4\cdot x_1 + 5\cdot x_2 + 6\cdot x_3 = 29\\ 7\cdot x_1 + 8\cdot x_2 + 9\cdot x_3 = 47\end{matrix} \quad\Leftrightarrow\quad \begin{pmatrix}1\\4\\7\end{pmatrix} x_1 + \begin{pmatrix}2\\5\\8\end{pmatrix} x_2 + \begin{pmatrix}3\\6\\9\end{pmatrix} x_3 = \begin{pmatrix}11\\29\\47\end{pmatrix} \tag{2.7b}$$

Following the column-wise interpretation of A, it is seen that the vector $\mathbf{b}$ must be a linear combination of the columns of A (that is, $\mathbf{b}$ must be in the **column space** of A, denoted $\mathbf{b}\in\mathsf{C}$) for there to be a vector $\mathbf{x}$ that solves $A\mathbf{x}=\mathbf{b}$. If some other $\tilde{\mathbf{b}}$ has any nonzero component in the orthogonal complement of C, called the **left nullspace** and denoted $\mathsf{L}=\mathsf{C}^\perp$, (that is, if $\tilde{\mathbf{b}}\notin\mathsf{C}$) then there is no vector $\mathbf{x}$ that solves $A\mathbf{x}=\tilde{\mathbf{b}}$.

We similarly call the **row space** of A, denoted R, as the subspace containing all vectors that can be reached by linear combination of the columns of A^T (that is, by the **rows** of A). The orthogonal complement of R is called the **nullspace** and is denoted $\mathsf{N}=\mathsf{R}^\perp$. Any vector $\mathbf{x}_\mathsf{N}\in\mathsf{N}$ is orthogonal to all rows of A, so $A\mathbf{x}_\mathsf{N}=0$. The significance of this to the $A\mathbf{x}=\mathbf{b}$ problem is that, for any $\mathbf{x}_\mathsf{N}\in\mathsf{N}$, we have $A\mathbf{x}_\mathsf{N}=0$, and thus, if $\mathbf{x}_1$ solves $A\mathbf{x}_1=\mathbf{b}$ for some $\mathbf{b}$, then $\mathbf{x}_2=\mathbf{x}_1+\mathbf{x}_\mathsf{N}$ also solves; that is, $A\mathbf{x}_2=A(\mathbf{x}_1+\mathbf{x}_\mathsf{N})=A\mathbf{x}_1+A\mathbf{x}_\mathsf{N}=\mathbf{b}+0$.

Note similarly that any vector $\mathbf{y}_L\in\mathsf{L}$ is orthogonal to all columns of A, so $A^T\mathbf{y}_L=0$.

For the 3×3 example in 2.7b, the subspaces C and $\mathsf{L}=\mathsf{C}^\perp$ are defined in (2.6a) and (2.6b), and the subspaces R and $\mathsf{N}=\mathsf{R}^\perp$ are defined analogously (but based on A^T instead of A), leading to

$$\mathsf{C}=\text{span}\left\{\begin{matrix}1&2\\4&5\\7&8\end{matrix}\right\},\quad \mathsf{L}=\text{span}\left\{\begin{matrix}1\\-2\\1\end{matrix}\right\},\quad \mathsf{R}=\text{span}\left\{\begin{matrix}1&4\\2&5\\3&6\end{matrix}\right\}=\text{span}\left\{\begin{matrix}1&4\\2&1\\3&-2\end{matrix}\right\},\quad \mathsf{N}=\text{span}\left\{\begin{matrix}1\\-2\\1\end{matrix}\right\}, \tag{2.8}$$

noting (confirm yourself) that the vectors that span R are perpendicular to the vector that spans N, and that the third column of A^T (that is, the third row of A) is spanned by the first two columns of A^T (that is, the first two rows of A); in other words, only 2 of the 3 rows of A are linearly independent [consistent with the fact, seen in (2.6c), that only 2 of the 3 columns of A are linearly independent; again, the **rank** of A is said to be 2].

The entire discussion above is summarized compactly in the Strang Cartoons shown in Figures 2.1-2.3; orthogonal bases of $\{\mathsf{C},\mathsf{L},\mathsf{R},\mathsf{N}\}$ for several more small example matrices are listed in Figure 2.4.

The best way to determine the rank of a matrix, in addition to generating orthonormal bases of $\{\mathsf{C},\mathsf{L},\mathsf{R},\mathsf{N}\}$, is to use the singular value decomposition[3] (SVD), which may be written $A=U\Sigma V^H=\underline{U}\,\underline{\Sigma}\,\underline{V}^H$, where $U=\begin{bmatrix}\underline{U} & \overline{U}\end{bmatrix}$, $\Sigma=\begin{bmatrix}\underline{\Sigma} & 0\\ 0 & 0\end{bmatrix}$, $V=\begin{bmatrix}\underline{V} & \overline{V}\end{bmatrix}$, and where $\underline{\Sigma}=\underline{\Sigma}_{r\times r}$ is diagonal, with positive entries in decreasing order along the main diagonal, and where $\{\underline{U},\overline{U},\underline{V},\overline{V}\}$ form orthonormal bases for $\{\mathsf{C},\mathsf{L},\mathsf{R},\mathsf{N}\}$, respectively.

[2]The notation $A_{m\times n}$ is used to indicate that the matrix A has m rows and n columns, where m may or may not equal n.

[3]The computation of the SVD is developed from scratch in §4 of *NR*, and is **beyond the scope** of the present introduction, as Matlab's built-in svd routine works just fine. To demonstrate, try the following: `>> A=[2 -2 -4;-1 3 4;1 -2 -3], [U,S,V]=svd(A)`. For integer matrices, the RR_Subspaces command (click for link) generates integer orthogonal (but, not normalized) bases of the four fundamental subspaces, as reported in Figure 2.4, which are somewhat simpler.

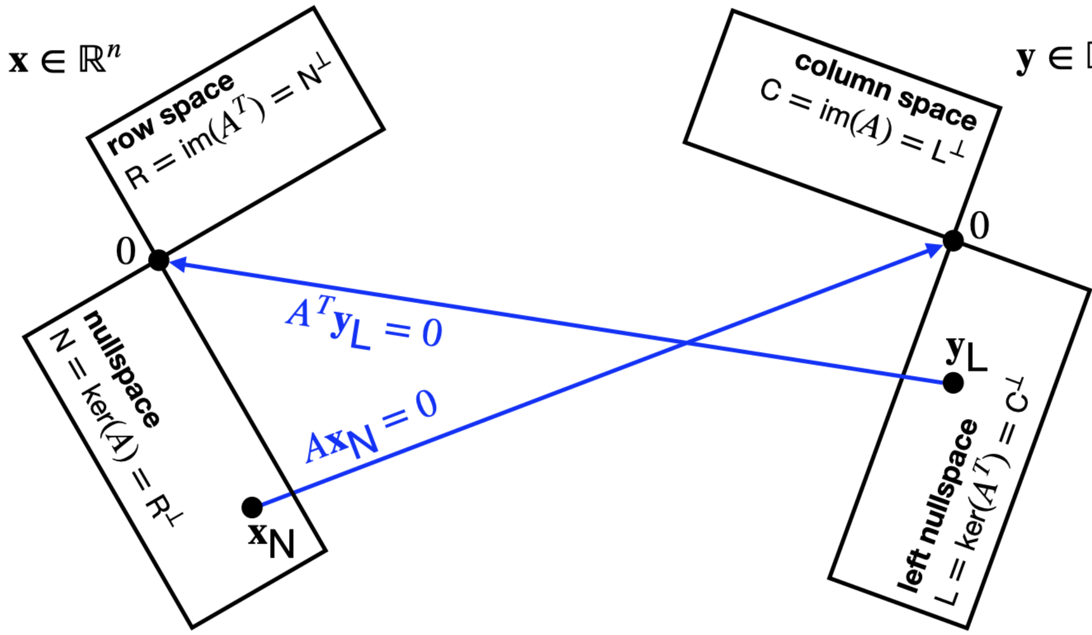


Figure 2.1: Version 1 of the Strang cartoon [4] for any square or rectangular matrix $A_{m\times n}$, depicting stylistically the vector space $\mathbb{R}^n$ on the left half (where $\mathbf{x} \in \mathbb{R}^n$), and the vector space $\mathbb{R}^m$ on the right half (where $\mathbf{y} \in \mathbb{R}^m$). The **nullspace** N is all $\mathbf{x}$ such that $A\mathbf{x} = 0$, and the **left nullspace** L is all $\mathbf{y}$ such that $A^T\mathbf{y} = 0$. Note also that: the **row space** $\mathsf{R} = \mathsf{N}^\perp$ is the orthogonal complement of the nullspace (each $\mathbf{x}_\mathsf{R}$ is orthogonal to all $\mathbf{x}_\mathsf{N}$), and the **column space** $\mathsf{C} = \mathsf{L}^\perp$ is the orthogonal complement of the left nullspace (each $\mathbf{y}_\mathsf{C}$ is orthogonal to all $\mathbf{y}_\mathsf{L}$). If N has more than the 0 element, then $A\mathbf{x} = \mathbf{b}$ is **underdetermined**.

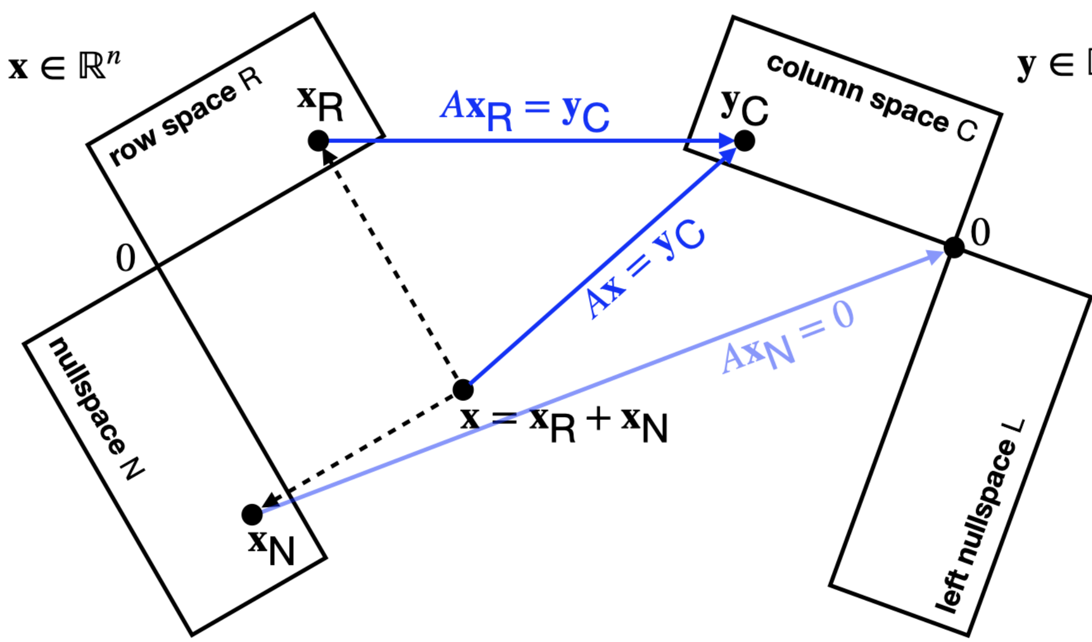


Figure 2.2: Version 2 of the Strang cartoon, illustrating the mapping from $\mathbf{x}$ to $\mathbf{y}$ due to A (that is, $A\mathbf{x} = \mathbf{y}$). Any $\mathbf{x}_\mathsf{R} \in \mathsf{R}$ maps **uniquely** to some $\mathbf{y}_\mathsf{C} \in \mathsf{C}$ via $A\mathbf{x}_\mathsf{R} = \mathbf{y}_\mathsf{C}$. More generally, any $\mathbf{x} \in \mathbb{R}^n$ may be **orthogonally decomposed** such that $\mathbf{x} = \mathbf{x}_\mathsf{R} + \mathbf{x}_\mathsf{N}$; thus, for any $\mathbf{x} \in \mathbb{R}^n$, it follows immediately that $A\mathbf{x} = A(\mathbf{x}_\mathsf{R} + \mathbf{x}_\mathsf{N}) = \mathbf{y}_\mathsf{C}$. Note that the column space C includes everywhere reachable via linear combinations of the columns of A. If L has more than the 0 element, then $A\mathbf{x} = \mathbf{b}$ is **potentially inconsistent**.

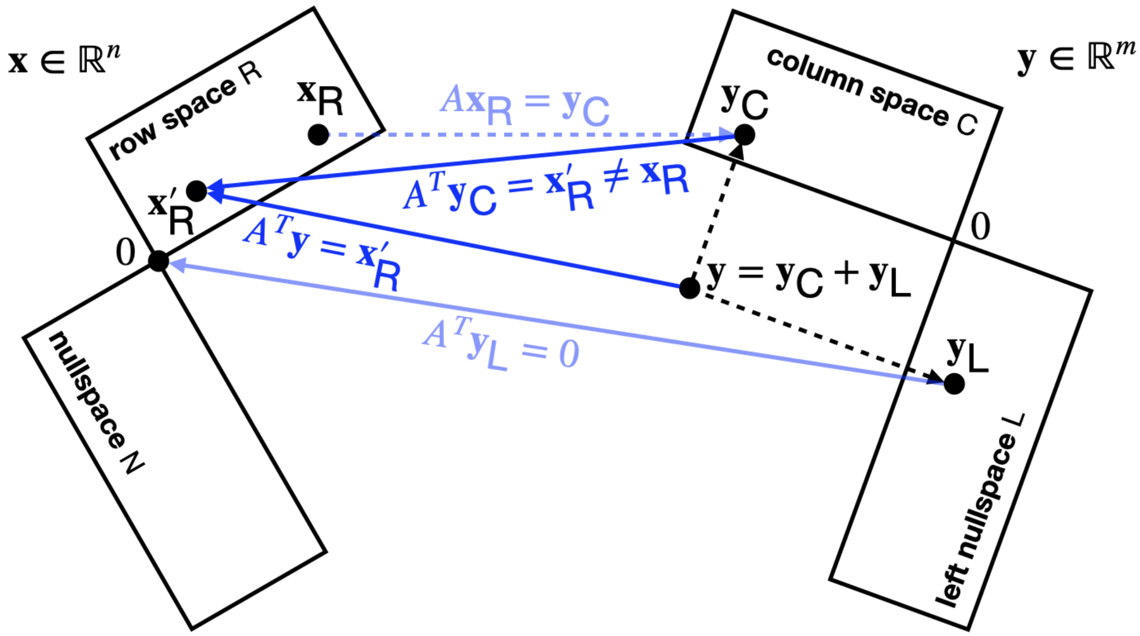


Figure 2.3: Version 3 of the Strang cartoon, illustrating the mapping from $\mathbf{y}$ back to $\mathbf{x}$ due to A^T. Any $\mathbf{y}_\mathsf{C} \in \mathsf{C}$ maps uniquely to some $\mathbf{x}'_\mathsf{R} \in \mathsf{R}$ via $A^T\mathbf{y}_\mathsf{C} = \mathbf{x}'_\mathsf{R}$; however, in general, $\mathbf{x}'_\mathsf{R}$ is **not** the same as the $\mathbf{x}_\mathsf{R}$ from which $\mathbf{y}_\mathsf{C}$ originated in Figure 2.2. More generally, any $\mathbf{y} \in \mathbb{R}^m$ may be orthogonally decomposed such that $\mathbf{y} = \mathbf{y}_\mathsf{C} + \mathbf{y}_\mathsf{L}$. In the special case that $A^T = A^{-1}$ (that is, if $A^T A = I$, and thus A is said to be **orthogonal**), then $\mathbf{x}'_\mathsf{R} = \mathbf{x}_\mathsf{R}$. Note that the row space R includes everywhere reachable via linear combinations of the columns of A^T (or, A^{-1}). If A is invertible, then $\mathbf{x} = A^{-1}\mathbf{y}$ for any $\mathbf{y}$; if N and/or L have more than just the zero element, A is not invertible.

matrix A	$\mathsf{C} = \mathrm{im}(A)$	$\mathsf{L} = \mathsf{C}^\perp$	$\mathsf{R} = \mathrm{im}(A^T)$	$\mathsf{N} = \mathsf{R}^\perp$	(pseudo)-inverse
$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{pmatrix}$	$\mathbb{R}^3$	$\{0\}$	$\mathbb{R}^3$	$\{0\}$	$\frac{1}{9}\begin{pmatrix} -16 & 8 & -1 \\ 14 & -7 & 2 \\ -1 & 2 & -1 \end{pmatrix}$
$\begin{pmatrix} 2 & -2 & -4 \\ -1 & 3 & 4 \\ 1 & -2 & -3 \end{pmatrix}$	$\operatorname{span}\left\{\begin{matrix} 2 & 2 \\ -1 & 3 \\ 1 & -1 \end{matrix}\right\}$	$\operatorname{span}\left\{\begin{matrix} 1 \\ -2 \\ -4 \end{matrix}\right\}$	$\operatorname{span}\left\{\begin{matrix} 1 & 1 \\ -1 & 1 \\ -2 & 0 \end{matrix}\right\}$	$\operatorname{span}\left\{\begin{matrix} -1 \\ 1 \\ -1 \end{matrix}\right\}$	$\frac{1}{63}\begin{pmatrix} 38 & 29 & -5 \\ 28 & 28 & -7 \\ -10 & -1 & -2 \end{pmatrix}$
$\begin{pmatrix} 2 & 2 & -2 \\ 5 & 1 & -3 \\ 1 & 5 & -3 \end{pmatrix}$	$\operatorname{span}\left\{\begin{matrix} 2 & 4 \\ 5 & -5 \\ 1 & 17 \end{matrix}\right\}$	$\operatorname{span}\left\{\begin{matrix} 3 \\ -1 \\ -1 \end{matrix}\right\}$	$\operatorname{span}\left\{\begin{matrix} 1 & -1 \\ 1 & 1 \\ -1 & 0 \end{matrix}\right\}$	$\operatorname{span}\left\{\begin{matrix} 1 \\ 1 \\ 2 \end{matrix}\right\}$	$\frac{1}{264}\begin{pmatrix} 8 & 45 & -21 \\ 8 & -21 & 45 \\ -8 & -12 & -12 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 0 & 0 \end{pmatrix}$	$\operatorname{span}\left\{\begin{matrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{matrix}\right\}$	$\operatorname{span}\left\{\begin{matrix} 0 \\ 0 \\ 1 \end{matrix}\right\}$	$\mathbb{R}^2$	$\{0\}$	$\frac{1}{2}\begin{pmatrix} -4 & 2 & 0 \\ 3 & -1 & 0 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 0 \\ 3 & 4 & 0 \end{pmatrix}$	$\mathbb{R}^2$	$\{0\}$	$\operatorname{span}\left\{\begin{matrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{matrix}\right\}$	$\operatorname{span}\left\{\begin{matrix} 0 \\ 0 \\ 1 \end{matrix}\right\}$	$\frac{1}{2}\begin{pmatrix} -4 & 2 \\ 3 & -1 \\ 0 & 0 \end{pmatrix}$

Figure 2.4: Orthogonal bases of $\{\mathsf{C}, \mathsf{L}, \mathsf{R}, \mathsf{N}\}$, and the inverse (if it exists, as in the first row, with rank 3) or the pseudoinverse (if the inverse does not exist, as in the remaining rows, with rank 2) of a few small matrices. The A matrix in the fifth row is underdetermined, with ∞ solutions to $A\mathbf{x} = \mathbf{b}$ (x_3 can be anything). The A matrix in the fourth row is potentially inconsistent, with 0 or 1 solutions to $A\mathbf{x} = \mathbf{b}$ depending on b_3. The A matrices in the second and third rows are both underdetermined and potentially inconsistent, with 0 or ∞ solutions to $A\mathbf{x} = \mathbf{b}$ depending on $\mathbf{b}$. The A matrix in the first row is nonsingular, with 1 unique solution $\mathbf{x} = A\backslash\mathbf{b}$.

There is one remaining (profound) piece of the puzzle: how to compute the "best" solution to $A\mathbf{x} = \mathbf{y}$ when the inverse A^{-1} doesn't exist, because either

a) there may be no solutions at all (i.e., when A is potentially inconsistent, with nonzero elements in L),
a) multiple solutions may exist (i.e., when A is underdetermined, with nonzero elements in N), or
c) both (a) and (b).

The answer [2] to this question is just to compute $\mathbf{x} = A^{+}\mathbf{y}$ where A^{+} is the **Moore-Penrose pseudoinverse**, which follows easily from the SVD discussed above such that $A^{+} = \underline{V}\Sigma^{-1}\underline{U}^{H}$. As illustrated graphically in Figure 2.3, denoting $\mathbf{y} = \mathbf{y}_{\mathsf{C}} + \mathbf{y}_{\mathsf{L}}$, the result of this simple computation is to map (uniquely) the component $\mathbf{y}_{\mathsf{C}}$ of $\mathbf{y}$, which is in the column space, back to $\mathbf{x}_{\mathsf{R}}$ in the row space (where it originated via the forward mapping $\mathbf{y}_{\mathsf{C}} = A\mathbf{x}_{\mathsf{R}}$), and just to map the component $\mathbf{y}_{\mathsf{L}}$ of $\mathbf{y}$, which is in the left nullspace, back to the origin in $\mathbf{x}$, as it is impossible to arrive at any $\mathbf{y}$ with a component in the left nullspace via the forward mapping.

2.4 Full solution of any $A\mathbf{x} = \mathbf{b}$ problem

We are now ready to put it all together.

In §2.1, we showed how **Gaussian Elimination** (a relatively simple "row-wise" approach) could be used to solve $A\mathbf{x} = \mathbf{b}$ for linear systems of equations for which the corresponding A matrix is square $(n \times n)$ and nonsingular; that is, such that each row can not be written as a linear combination of the other rows, and thus each column can not be written as a linear combination of the other columns, and thus $r = \text{rank}(A) = n$. It was mentioned that a straightforward generalization of the Gaussian Elimination approach can be used to establish that there are no solutions, and/or to parameterize the set all solutions, to the $A\mathbf{x} = \mathbf{b}$ problem in the case of a singular or nonsquare A.

However, these two later cases are much better handled by the ("column-wise") approach discussed in §2.3, the heart of which is the **Singular Value Decomposition** (SVD). Stable algorithms [5] and codes [in Matlab, using `[U,S,V]=svd(A)`] for computing the SVD are readily available, and:

• computing the SVD is the best available way to compute the **rank** r of A (in Matlab, with `r=rank(A)`),
• the SVD determines directly **orthonormal bases** for {C, L, R, N} (summarized in the Strang cartoons), and
• the **Moore-Penrose pseudoinverse** A^{+} follows directly from the SVD (in Matlab, with `Aplus=pinv(A)`).

To sum it all up, **any** linear system of algebraic equations, once rewritten as $A\mathbf{x} = \mathbf{b}$, may be solved on a computer (that is, quickly, and without the possibility of human error) by following the flowchart in Figure 2.6, which simply combines all the various results developed earlier in this chapter in a convenient order.

References

[1] Golub, G, and Van Loan, C (2013) Matrix Computations, 4th Edition. 2-8, 6-2

[2] Penrose, R (1955) A generalized inverse for matrices *Proceedings of the Cambridge Philosophical Society* **51**, 406-413. 2-8

[3] Strang, G (2006) Linear Algebra and Its Applications, 4th Edition. 2-6, 6-2

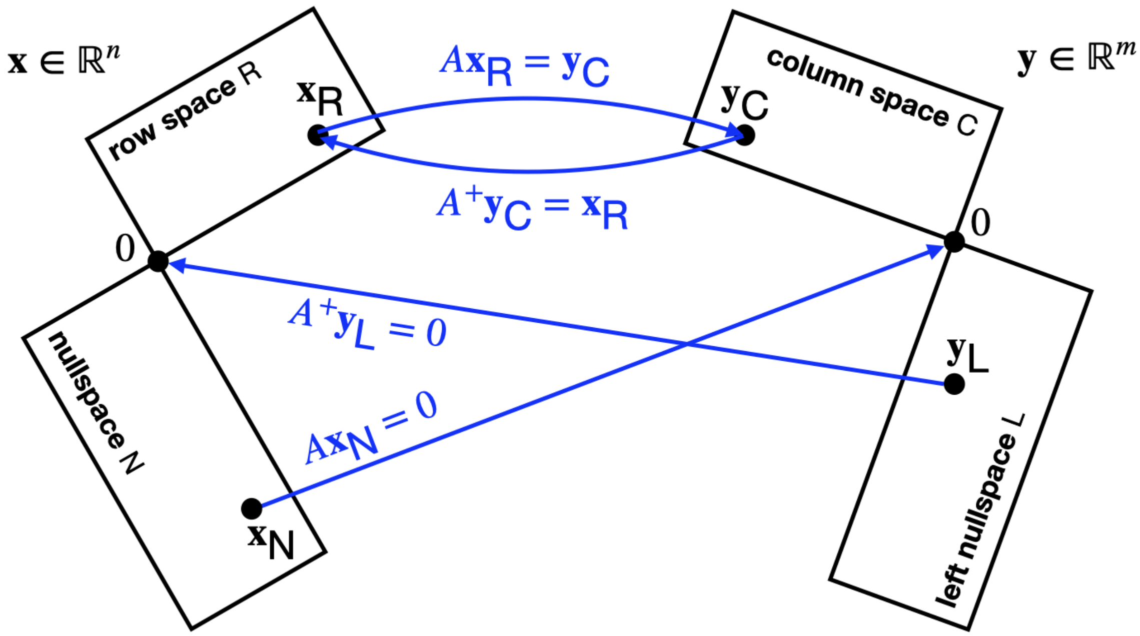


Figure 2.5: Version 4 of the Strang cartoon, illustrating the forward and best available inverse mapping from $\mathbf{x}$ to $\mathbf{y}$ (due to A) and $\mathbf{y}$ back to $\mathbf{x}$ (due to A^+). Note that A^+ is the "best" available substitute for A^{-1} when A^{-1} does not exist, in that it minimizes both $\|\boldsymbol{\epsilon}\|$, and (secondarily) $\|\mathbf{x}\|$, in the solution of $A\mathbf{x} = \mathbf{b} + \boldsymbol{\epsilon}$. Note also that both R and C are spanned by r vectors, where r is the rank of A (in Matlab, `r=rank(A)`), N is spanned by $(n-r)$ vectors, and L is spanned by $(m-r)$ vectors.

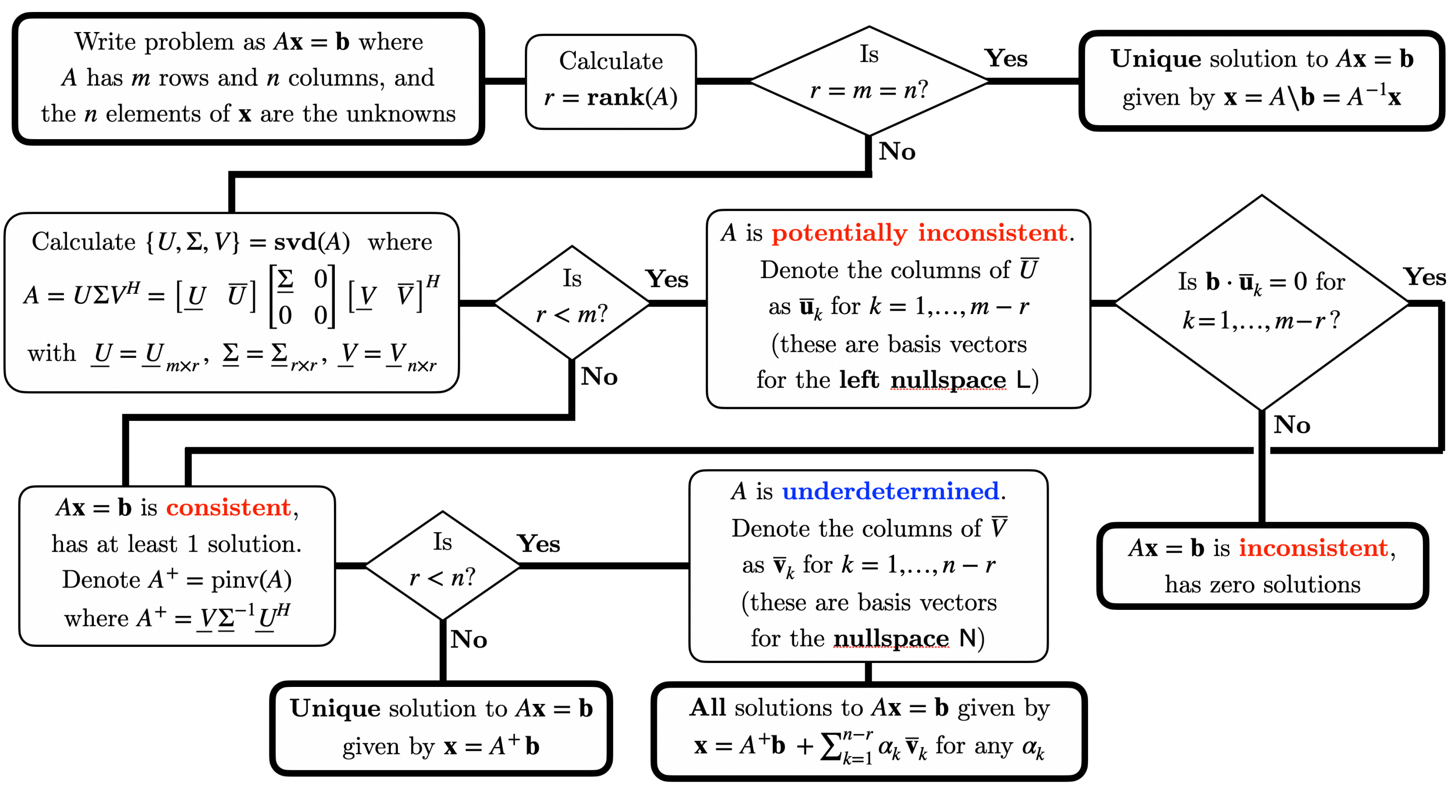


Figure 2.6: Complete flowchart for solving $A\mathbf{x} = \mathbf{b}$ for any $A_{m\times n}$, with $m < n$, $m = n$, or $m > n$.

Chapter 3

Structural Material Types & Properties

We aspire in this text to analyze and, ultimately, design robust structural systems that are, to varying degrees: strong, lightweight, inexpensive, corrosion resistant, long lasting (minimizing life cycle fatigue, dampening vibrations), thermally stable (minimizing buckling due to thermal expansion, brittleness at low temperatures, creep at high temperatures), deployable, controllable, fault tolerant, maintainable, and forgiving to overstress (bending before breaking in emergency landings and earthquakes), with minimal environmental impact.

It's a long list of needs, and involves a lot more than just computing internal member forces when the structural system under consideration is subjected to various nominal and emergency loading conditions, and selecting the structural members accordingly (though that is certainly an essential part of it). Structural systems are made from individual structural members, and in turn individual structural members are made from materials. Thus, any study of the analysis and design of structural systems must start with a survey of the available materials with which the structural members may be built, and how these materials react to load.

3.1 Structural materials: a brief survey

The 4 classes of structural materials are **metals** (with metallic bonds), **ceramics** (with covalent and/or ionic bonds), **polymers** (with covalent bonds of hydrocarbons), and **composites** thereof. In short:

• **Metals** are primarily composed of metallic elements (the 6 most common being **iron**, **aluminum**, **copper**, **titanium**, **magnesium**, and **cobalt**) mixed with lesser amounts of other elements (aka **alloyants**) to form alloys that improve the several properties summarized above, as highlighted in Figure 3.1. Examples:

- **Steel** (aka **plain carbon steel**) is the structural material against which all others are compared. Steel is iron plus 0.05% to 2% carbon by weight (increased carbon generally makes it harder). Steel is high strength and low cost; different manners of processing and heat treating lead to a wide range of different properties (see §3.2).
- **Alloy steel**, which is steel with any of a large number of different alloyants (see Figure 3.1), achieve different balances of strength, hardness, toughness, wear resistance, corrosion resistance, and temperature stability.
- **Stainless steel** is an alloy steel with $\sim$11% chromium and other alloyants. When exposed to oxygen, a passive film of **chromium oxide** forms that protects the material from rust (iron oxide) and other types of corrosion.
- **Cast iron**, which is iron plus >2% carbon and 1% to 3% silicon, is harder and much more brittle than steel. Cast iron is commonly used in cookware due to its health benefits; most other uses of cast iron in engineering applications in the past have been replaced by other materials on this list with better mechanical properties.
- **Aluminum alloys**, with copper, magnesium, manganese, silicon, tin, nickel, zinc, and/or other alloyants, are lightweight, relatively inexpensive structural materials common in many modern engineering applications.

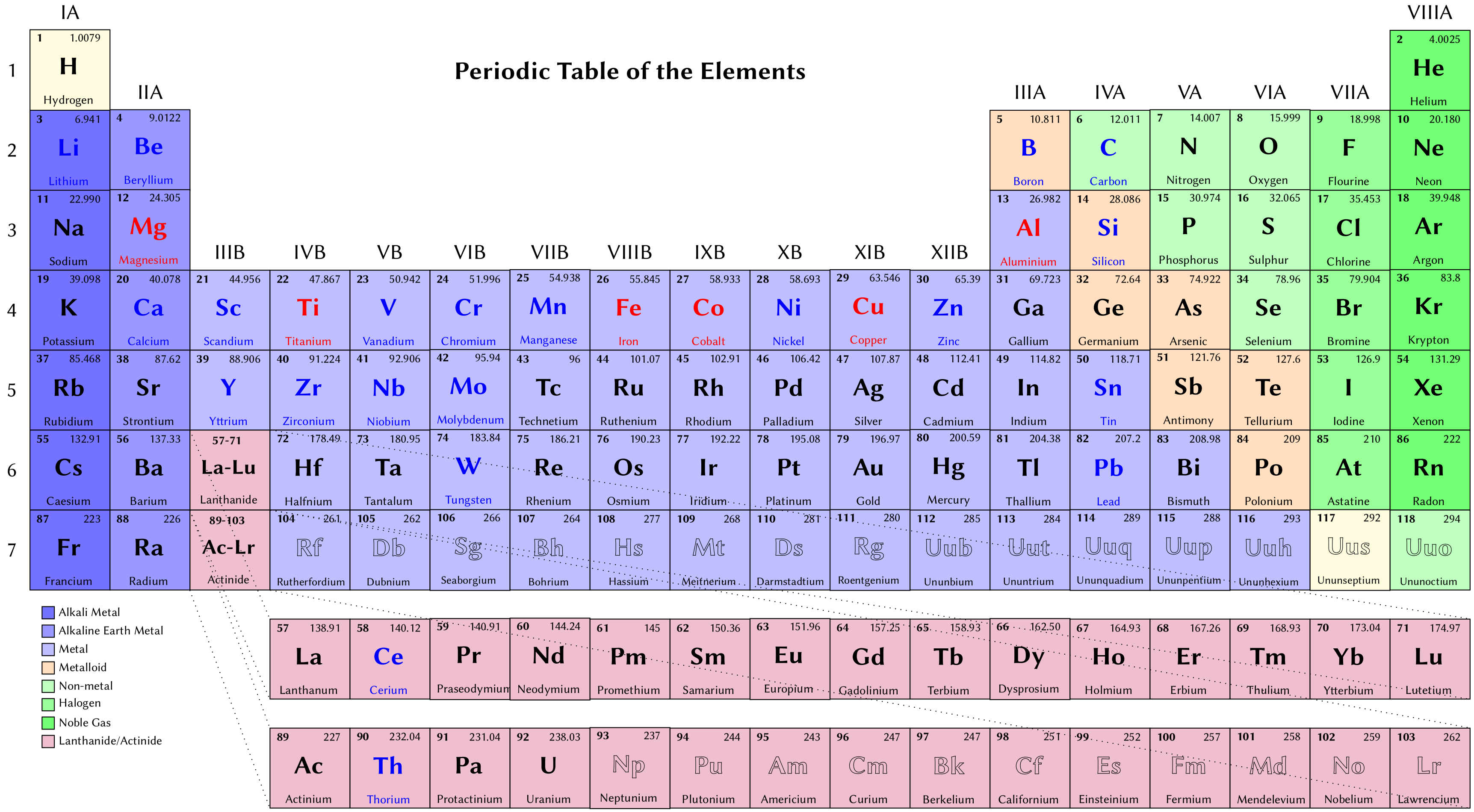


Figure 3.1: Periodic Table of the Elements, for providing context related to structural metals. The 6 most common structural metals are **iron** alloys (notably, **steel**, which adds up to 2% **carbon** to improve strength, and **stainless steel**, which additionally adds $\sim 11\%$ **chromium** to prevent rust/corrosion in the presence of oxygen), as well as **aluminum**, **copper**, **titanium**, **magnesium**, and **cobalt**.
Common additional alloyants of steel include **manganese**, **nickel**, **molybdenum**, **vanadium**, **silicon**, and **boron**.
Common alloyants of aluminum include **copper**, **magnesium**, **manganese**, **silicon**, **tin**, **nickel**, **zinc**, **lithium**, and **zirconium**.
Common alloyants of titanium include **aluminum**, **vanadium**, **tin**, **molybdenum**, and **silicon**.
Common alloyants of magnesium include **aluminum**, **manganese**, **zinc**, **zirconium**, **silicon**, and **yttrium**.
Common alloys of copper include **bronze**, with $\sim 12\%$ **tin**, **brass**, with $\sim 30\%$ **zinc**, and **gun metal**, with $\sim 9\%$ **tin** and $\sim 3\%$ **zinc**.
Common alloys of cobalt typically include 25% to 32% **chromium**, with **molybdenum**, **manganese**, **silicon**, **iron**, **nickel**, and/or **carbon**.
Other useful alloyants are also marked in blue. Note: in the pdf, everything in color in this caption is hyperlinked to further info online.

- **Bronze** is a **copper alloy** with ∼12% tin and other alloyants, providing relatively high ductility and strength.
- **Brass** is a copper alloy with ∼ 30% zinc, offering high corrosion resistance and malleability, and low friction.
- **Gun metal** (aka **red brass**) is a copper alloy with ∼ 9% tin and ∼ 3% zinc, which casts and machines well, and is resistant to corrosion by steam and salt water.
- **Titanium alloys**, with ∼ 6% aluminum, ∼ 4% vanadium, and other alloyants, offer very high toughness, corrosion resistance, and strength-to-weight ratio valuable in aerospace applications, but they are expensive.
- **Magnesium alloys**, with aluminium, manganese, zinc, zirconium, silicon, yttrium, and other alloyants, offer the very best strength-to-weight ratio available, but are also expensive.
- **Cobalt-chrome** is mostly cobalt, 25% to 32% chromium, and small amounts of molybdenum, manganese, silicon, iron, nickel, carbon, and other alloyants, which provides good resistance to wear and corrosion.

Metal materials may formed into shape with some combination of 3 general techniques; they may be:

wrought [shaped via deformation, like extrusion/drawing (pushing/pulling through a die), rolling (squeezing between pairs of roller), upsetting (squeezing in a machine press, decreasing length & increasing area), die stamping, cutting, raising/sinking/planishing (hammering with a backing), roll forming/bending, and forging[1]],

machined [shaped via **subtractive manufacturing**, the 3 important modes of which are **drilling** (moving a rotating bit in its longitudinal direction against a stationary workpiece, to cut a circular hole), **milling** (moving a rotating bit in its lateral directions against a stationary workpiece, cutting with the edge of the bit), and **turning** with a lathe (rotating the workpiece, slowly moving a cutting bit up against it; cf. drilling and milling)], or

cast [shaped via pouring molten metal into a mold].

Various types of metals, with different alloyants, respond to these distinct methods of forming quite differently. Some metals (notably stainless steel, aluminum, titanium, and cobalt-chrome) may be formed with **additive manufacturing** (aka 3D printing) via a Direct Metal Laser Sintering (DMLS) processes.

• **Ceramics** are made of inorganic, non-metallic compounds like oxides, nitrides, or carbides. Ceramics are typically hard, brittle, electrically insulating, and highly resistant to corrosion and extreme temperatures. Examples include porcelain, glass, and diamond. An important use of brittle ceramics in structural systems is as sacrificial shear pins acting as **mechanical safegaurds**, designed to fail at a predictable level of stress before other parts in the system are damaged. Common uses include the connection between a lawnmower blade and the lawnmower engine, or between a pushback tractor and a large aircraft at an airport gate. In such applications, the shear pin acts akin to an electrical fuse designed to fail at a predictable level of current, protecting the rest of the system. Shear pins can also be used as **conditional operators**, preventing a device (like a fire extinguisher, or the explosive charge in an air-to-air missile) from operating until a specified force (on the handle, or from the launch of the missile) breaks the pin, thus allowing the device to function.

• **Polymers** are organic compounds (containing carbon-hydrogen and carbon-carbon covalent bonds) that are lightweight, often relatively flexible, and can easily be formed into complex shapes. Examples include nylon, rubber, natural fibers (wood, . . .), and plastics (ABS, PLA, PETG, TPU, PVA, Polypropylene, ...). Nylon and plastics are often used in 3D printing, the three major types of which are FDM (Fused Deposition Molding), SLA (Stereolithography), and SLS (Selective Laser Sintering). PMMA (polymethyl methacrylate) is a hard plastic commonly known as acrylic and marketed under brand names like plexiglass, lucite, or perspex. PVC (polyvinyl chloride) and CPVC (chlorinated PVC) are commonly used for pipes. Some polymers, like kevlar (an aromatic polyamide characterized by long rigid crystalline polymer chains), are extremely strong.

[1] Forged metal is a special case of wrought metal that has been worked at elevated temperatures using localized compressive forces, typically with a hammer & anvil or a closed die.

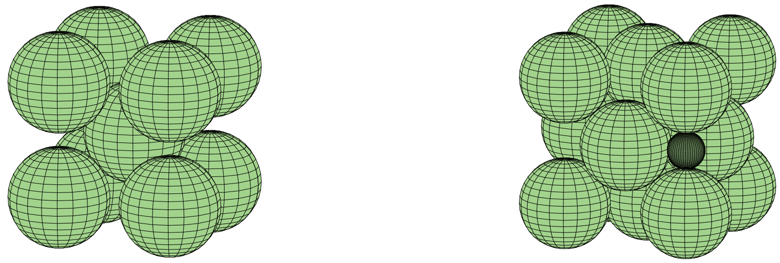


Figure 3.2: The 2 primary crystal structures of low-carbon and medium-carbon steel, shown as sphere packings: (left) Body Centered Cubic (**BCC**) aka ferrite, and (right) Face Centered Cubic (**FCC**) aka austenite. Iron atoms are depicted in green, and a small interstitial carbon atom is depicted, at right, in black.

• **Structural composites** generally incorporate a **matrix**, which is the structural material (polymer, metal or ceramic) that binds together the other components, with embedded **reinforcement**, which provides additional strength and stiffness (gravel, steel bars, carbon or glass or natural fibers, ...), and include the following:
- **concrete** consists of an inexpensive construction aggregate (e.g., sand, gravel), which acts as the reinforcement, bonded together with a fluid cement that, once set, forms the matrix;
- **reinforced concrete** consists of concrete (itself a composite) with embedded steel reinforcement bars;
- **fiberglass** is formed by taking flexible cloths made of thin strands of glass, which act as reinforcement when laid in layers over a mold (e.g., of a boat hull) and coated with an epoxy resin that, once set, forms the matrix;
- **plywood** is composed of thin layers of wood reinforcement that are stacked and glued;
- a **wood-plastic composite** (WPC) is a blend of wood fibers, acting as reinforcement, set in a plastic matrix;
- a **carbon-fiber reinforced polymer** (CFRP) incorporates a set of **carbon fibers** set in a hard plastic matrix;
- a **metal matrix composite** (MMC) consists of a metal matrix with reinforcing elements embedded within;
- a **ceramic matrix composite** (CMC) consists of a ceramic matrix with reinforcing elements embedded within;
- a **polymer matrix composite** (PMC) consists of a polymer matrix with reinforcing elements within.

Structural composites are often much stronger in some directions (those that have been reinforced) than the others, which must be accounted for properly during design and construction of a structure that uses them.

Steel is, by far, the most important structural material for land-based and heavy seagoing applications. For land-based structures, reinforced concrete and wood are common inexpensive alternatives. For lightweight seagoing applications, WPC, fiberglass, and CFRPs are common. For aerospace applications, lightweight aluminum, titanium, and magnesium alloys are common, with CFRPs playing an increasing role.

3.2 Crystal structure of structural metals

This section surveys the crystal structure and plastic deformation of the metals commonly used in structural applications; for a more in depth discussion, see Callister & Rethwisch [1]. We must look over many different length scales in such a study; we start (in §3.2.1) from the smallest length scales, and work our way up.

3.2.1 FCC & BCC: the crystal structures of low-carbon steel, aluminum, and copper

Plain carbon steel is characterized by its percent carbon content by weight, according to the following ranges: low carbon: 0.05% - 0.3%, medium carbon: 0.3% - 0.6%, high carbon: 0.6% - 1.2%, ultra-high carbon: 1.2% - 2.0%.

In the packing of atoms in a metallic structure, individual atoms may be visualized as spheres, each with an effective radius from the center of the nucleus to the effective edge of the outermost electron shell. Defining an angstrom as $1\,\text{Å} = 10^{-10}$ m, the effective radius of an iron atom is about $1.27\,\text{Å}$. As depicted in Figure 3.2, the 2 primary structures that low- and medium-carbon steel can form can be visualized as sphere packings, namely:

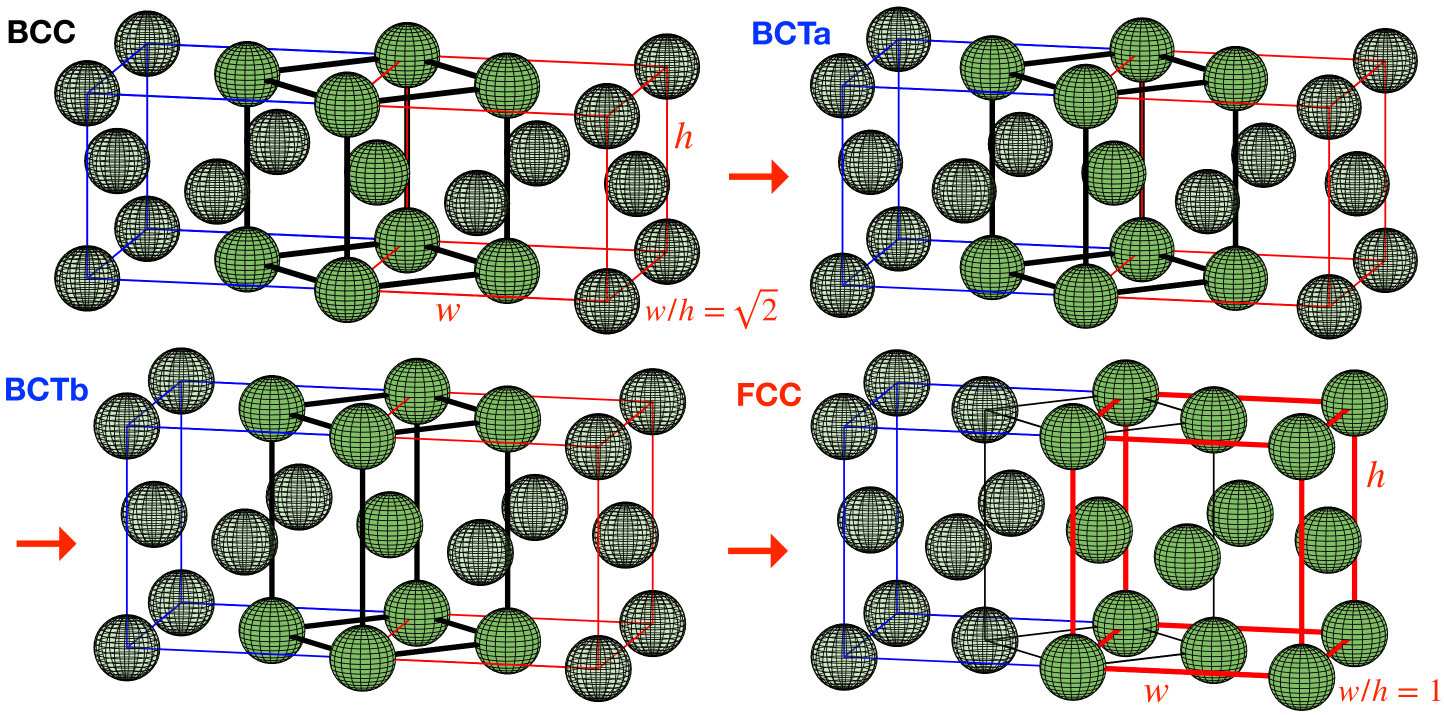


Figure 3.3: The Body Centered Tetragonal (BCT, aka martensite) metastable intermediate states between (top left) BCC and (bottom right) FCC, with the two intermediate states shown having $1 < w/h < \sqrt{2}$. The same 23 iron atoms are shown in all 4 frames, the dimensions in the different directions is just changed slightly. The spheres are depicted here at a smaller size than in Figure 3.2 simply to clarify the visualization.

• **Body Centered Cubic** (**BCC**, aka **ferrite**), with each unit cell (2 iron atoms) being 2.9 Å per side, and
• **Face Centered Cubic** (**FCC**, aka **austenite**), with each unit cell (4 iron atoms) being 3.6 Å per side.
In particular, BCC and FCC are called **lattice packings**, as they look identical when any sphere is shifted to the origin. Defining the **coordination number** of a sphere packing as the number of nearest neighboring spheres to each individual sphere, and the defining the **packing density** of a sphere packing as the proportion of each unit cell that is inside one of the spheres, it follows that
- BCC has a coordination number of 8 and a packing density of 0.68, whereas
- FCC has a coordination number of 12 and a packing density of 0.74.

Note that ferrite (the BCC form) can absorb up to **0.02% carbon** by weight, whereas austenite (the FCC form) can absorb up to **0.83% carbon** by weight. The carbon atoms in both cases sit in **interstitial** locations, between the larger green spheres in the BCC or FCC configurations depicted in Figure 3.2. In both cases, the carbon atoms don't quite fit in the available voids, thus requiring a slight deformation of the crystal structure to accommodate them. This deformation is slightly reduced in the case of FCC (as compared to BCC), even though its packing density is somewhat higher, and thus the FCC form can accommodate more carbon. As discussed in §3.2.4, the presence of such interstitial atoms, and the deformation to the crystal structure that they cause, inhibit the slip of one plane of the crystal with respect to the next, thus making the steel harder.

There is a related family of metastable crystal structures between BCC and FCC, as depicted in Figure 3.3. Starting with BCC, if the crystal is just stretched a bit in one direction while it is simultaneously compressed a bit in the other two directions, the crystal structure can be smoothly transformed from BCC to FCC and back. This is called a **martensitic transformation**; the family of intermediate metastable crystal structures is called

• **Body Centered Tetragonal** (**BCT**, aka **martensite**).

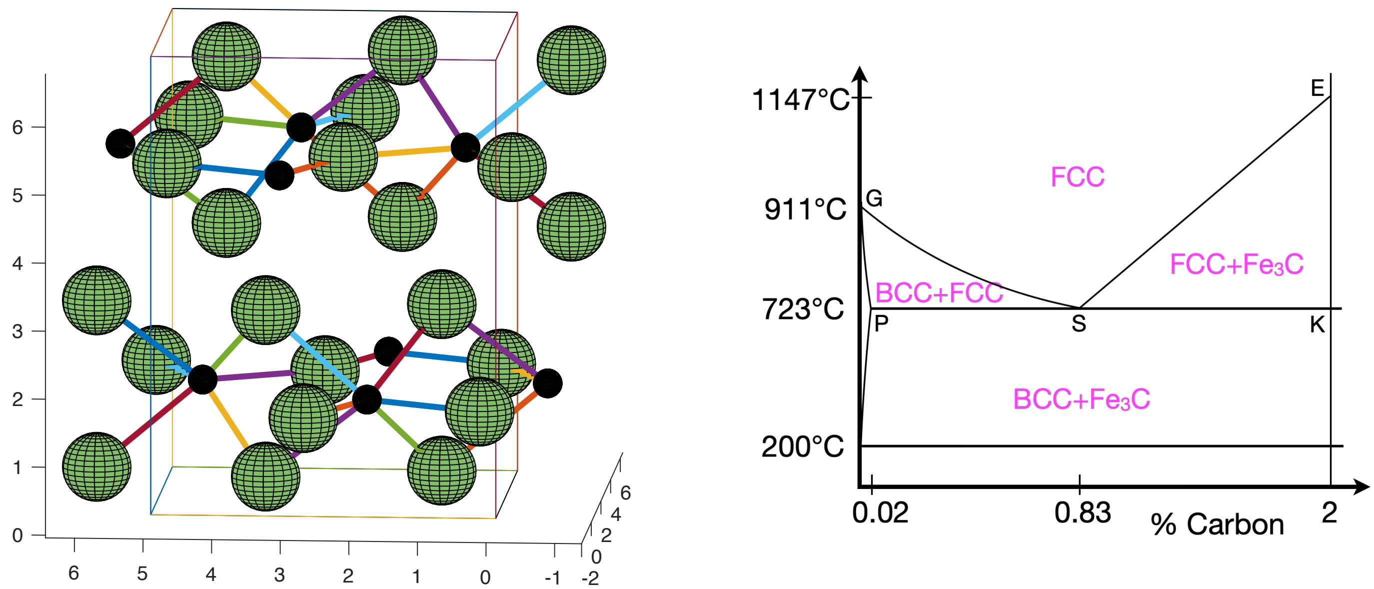


Figure 3.4: (a) The structure of **iron carbide** (Fe_3C aka **cementite**), which is a hard, brittle ceramic in which the atoms are bound by covalent (shown as lines) and ionic bonds (cf. the metallic bonds binding ferrite and austenite). (b) The **phase diagram** for carbon steel, illustrating the phases present at thermodynamic equilibrium. Steel with essentially no carbon (to the left of point P) is ferrite (BCC) for T $< 911°$C, and austenite (FCC) for T $> 911°$C. The distribution of these two crystal phases together with iron carbide (Fe_3C) at thermal equilibrium is shown for up to 2% carbon. Note that there is no appreciable diffusion of carbon below $200°$C.

3.2.2 Iron carbide (Fe_3C aka cementite) and its role in high-carbon steel

High carbon and ultra-high carbon steel have more carbon than can be absorbed into the BCC and FCC crystal structures, and thus a third (hard, brittle) structure of iron + carbon comes into play, namely:

- **iron carbide** (Fe_3C, aka **cementite**) with each unit cell (12 iron + 4 carbon atoms) being $4.5\,\text{Å} \times 5\,\text{Å} \times 6.7\,\text{Å}$.

The unit cell of iron carbide (see Figure 3.4a) is known as **orthorhombic Primitive oP16**, and is **6.67% carbon** by weight. Given the classifications stated earlier, it is a **ceramic**, bound with covalent and ionic bonds.

Note that high-carbon and ultra-high-carbon steel generally have an overall percentage of carbon which is somewhere between the maximum that austenite can absorb (0.83%) and that which iron carbide has (6.67%). Steel is in fact composed of tiny, crystalline regions called **grains**, each of which may be ferrite, austenite, martensite, or iron carbide, as depicted in the phase diagram shown in Figure 3.4b. High-carbon and ultra-high-carbon steel are thus a metal matrix composite (MMC); more precisely, a **metal matrix ceramic composite** (**MMCC**). [Other notable MMCCs include silicon carbide, alumina, and boron carbide.] The size and shape of the grains in this composition, and the distribution of the different crystal structures, significantly impact the properties of the material (see §3.2.5). In such composites, the ceramic phase provides hardness and resistance to wear (but, often, at the cost of significant brittleness), while the metal matrix provides ductility and toughness.

3.2.3 HCP: the crystal structure of titanium, magnesium, and cobalt

Note that §3.2.1 and §3.2.2 highlighted the crystal structure of plain carbon steel (iron+carbon) in particular. Aluminum and Copper are similar, with FCC being the dominant crystal structure at room temperature.

Titanium, magnesium, and cobalt, on the other hand, form a different crystal structure:

- **Hexagonal Close Packed** (**HCP**), with the same coordination and packing density as FCC.

?

Figure 3.5: ?

The difference between FCC and HCP is subtle, but profound. FCC may be visualized as stacks of 2D hexagonal packings. The first layer is called A. The second layer is called B, and sits in the valleys between the circles of the first layer (there ABABAB... versus ABCABC...

3.2.4 Shearing of crystals: distribution and movement of defects and interstitials

Within each ferrite, austenite, or martensite crystal grain (that is, on a length scale about 10^{-8} m to 10^{-7} m), **defects** in the crystal structure (that is, voids and dislocations; see Figure 3.5) and the distribution of interstitial atoms play a substantial role in determining the strength of a material.

HCP metals have only 3 slip systems with one slip plane and 3 slip directions while BCC metals have at least 12 slip systems with 6 slip planes and 2 slip directions up to 24 slip planes and FCC metals have 12 slip systems with 4 slip planes and 3 slip directions [2]

However, as the material is subsequently worked, the distribution and movement of these interstitials and defects play a critical role in the plastic deformation of the material.

As mentioned previously, interstitial carbon

The effects of various alloyants

3.2.5 Microstructure (grain size/shape, grain crystal phases and their distribution)

working the material affects the microstructure

10^{-7} to 10^{-4} m

pearlite

Steel with *very* low carbon (to the left of point P in Figure 3.4) is BCC (Ferrite) at T < 911 C, and FCC (Austenite) at T > 911 C. The distribution of these two crystal phases, together with a third phase (Fe3C), for up to 2% carbon, is also shown. Note that there is no appreciable diffusion of carbon below 200 C.

The effect of heat treatment on microstructure

The effect of processing on microstructure

3.2.6 Macrostructure (fillets, chamfers, roughness)

10^{-4} m and up

The most common use of zinc in engineering is the application of a protective zinc coating to steel, a process known as **galvanization**.

3.3 Properties of structural materials

This section outlines the six main classes of properties of materials, **Mechanical**, **Deteriorative**, **Thermal**, **Electrical/Electronic**, **Magnetic**, and **Electromagnetic**. There is much that may be said; as we are focused in this text on building structures, we focus primarily (in §3.3.1) on the several relevant and distinct mechanical properties of materials, many of which were mentioned informally in the introduction to this chapter.

3.3.1 Mechanical properties

Figure of tension, compression, bending, torsion, shear, ...

3.3.1.1 Tensile & compressive strength, stiffness v. flexibility, elasticity v. plasticity

3.3.1.2 Brittle v. ductile failure

3.3.1.3 Shear and torsional strength

3.3.1.4 Fatigue strength and high-temperature creep

3.3.1.5 Resilience, toughness, and low-temperature ductile → brittle transition

Fracture toughness
  Ductile → Brittle transition in cold environments

3.3.1.6 Hardness

3.3.1.7 Malleability, ductility, and machinability

3.3.2 Deteriorative (chemical) properties: corrosion and biocompatibility

3.3.3 Thermal and thermo-mechanical properties

3.3.4 Electrical/Electronic and electro-mechanical properties

piezoelectric effect

3.3.5 Magnetic and magneto-mechanical properties

Note also that ferrite and cementite are **ferromagnetic**, meaning that they is attracted by magnetic fields and can be magnetized to become a permanent magnet, whereas austenite is nonmagnetic.

3.3.6 Electromagnetic (optical/RF) properties

References

[1] Callister Jr, WD, Rethwisch, DG (2018) *Materials Science and Engineering: An Introduction 10th Edition*, Wiley. 3-4

[2] Afolaranmi, GA, Tettey, Meek, RMD, & Grant, MH (2008) Release of Chromium from Orthopaedic Arthroplasties *Open Orthop J.* **2**, 10-18.

[3] Kovochich, M, Monnot, A, et al. (2021) Carcinogenic hazard assessment of cobalt-containing alloys in medical devices: Review of in vivo studies. *Regulatory Toxicology and Pharmocology* **122**, 104910.

Chapter 4

Stresses and Failures in Loaded Beams

4.1 Backstory: dynamics of single 3D solid body

The two most consequential equations in classical mechanics are

$$\mathbf{f} = M\,\mathbf{a} \tag{4.1a}$$
$$\mathbf{m} = I^B\boldsymbol{\alpha} \tag{4.1b}$$

where $\{\mathbf{f}, \mathbf{m}\}$ are the total applied forces and moments, and $\{\mathbf{a}, \boldsymbol{\alpha}\}$ are the linear and angular accelerations.

Note that (4.1b) is simplified using a body-fitted coordinate system, moving with the body, with origin at its center of mass (CM), and coordinate directions $\{\mathbf{e}^1, \mathbf{e}^2, \mathbf{e}^3\}$ aligned along its special ?principle axes? (e.g. axes of symmetry). Taking ?? as the density of the material, ? = mass of body, ? inertial tensor of body, with ?principle moments? ?, ?, ?, and ?. Can accommodate both spinning 3D bodies and aerodynamic loads, both of which are beyond the scope of the present class; these are rich follow-up subjects that you will like study closely in later undergraduate coursework if/when you stick with engineering.

References

[1] Landau, L, & Lifshitz, E (1976) *Mechanics*. Butterworth/Heinemann.

Chapter 5

Frame Structures

Chapter 6

Truss Structures

6.1 Static equilibria of 2D and 3D truss structures

We now summarize an elegant "force" (as opposed to "displacement") method (Timoshenko [1], Likins [2], Skelton [1]) for determining the static equilibria of (pin-jointed) 2D and (ball-jointed) 3D *truss* structures, with
• m *members* $\{\vec{m}_1, \dots, \vec{m}_m\}$, each in either pure tension or pure compression (aka "two-force" members), and
• q *free nodes* $\{\vec{q}_1, \dots, \vec{q}_q\}$ and p *fixed nodes* $\{\vec{p}_1, \dots, \vec{p}_p\}$, collectively called the $n = q + p$ *nodes* $\{\vec{n}_1, \dots, \vec{n}_n\}$.
The n nodal locations $\vec{n}_i$ are each vectors from the origin of the reference frame used in $\mathbb{R}^d$, where $d = 2$ or 3 is the dimension of the problem considered, and are assembled as follows[1]:

$$\begin{aligned} Q &= \begin{bmatrix} \vec{q}_1 & \cdots & \vec{q}_q \end{bmatrix}, \quad P = \begin{bmatrix} \vec{p}_1 & \cdots & \vec{p}_p \end{bmatrix} \\ \Rightarrow \quad N &= \begin{bmatrix} \vec{n}_1 & \cdots & \vec{n}_n \end{bmatrix} = \begin{bmatrix} Q & P \end{bmatrix} = \begin{bmatrix} \vec{q}_1 & \cdots & \vec{q}_q & \vec{p}_1 & \cdots & \vec{p}_p \end{bmatrix}; \end{aligned} \tag{6.1a}$$

Q, which defines the locations of the free nodes, is called the *configuration matrix* of the structure. Each member $\vec{m}_k = \vec{n}_{k,1} - \vec{n}_{k,2}$ connects two nodes, $\vec{n}_{k,1}$ and $\vec{n}_{k,2}$, at least one of which is free, and are assembled as follows:

$$M = \begin{bmatrix} \vec{m}_1 & \cdots & \vec{m}_m \end{bmatrix}. \tag{6.1b}$$

It is useful to define a vector of member lengths $\boldsymbol{\ell}$, and a matrix of normalized member directions D, such that

$$\ell_k = \|\vec{m}_k\|, \quad \vec{d}_k = \vec{m}_k/\ell_k, \quad D = \begin{bmatrix} \vec{d}_1 & \cdots & \vec{d}_m \end{bmatrix}; \tag{6.1c}$$

note that $\|\vec{d}_k\| = 1$ for all k. The connectivity of a structure, relating the n nodes N in (6.1a) to the m members M in (6.1b), is described via its *connectivity matrix* C, defined and partitioned such that

$$M = N\,C^T, \quad C = \begin{bmatrix} C_Q & C_P \end{bmatrix} \quad \Rightarrow \quad M = \begin{bmatrix} Q & P \end{bmatrix} \begin{bmatrix} C_Q^T \\ C_P^T \end{bmatrix}, \tag{6.1d}$$

where, denoting $\mathbf{e}_j$ as the vector in the j'th column of the identity matrix, each column of C^T is given by $(\mathbf{e}_{k,1} - \mathbf{e}_{k,2})$, indicating the two nodes $\vec{n}_{k,1}$ and $\vec{n}_{k,2}$ that member $\vec{m}_k$ of the structure connects, with one entry equal to 1, one entry equal to -1, and all other entries equal to zero.

Consider also external forces $\{\vec{u}_1, \dots, \vec{u}_q\}$, including the net effects of the weights of the members themselves, applied to each of the q free nodes, as well as the reaction forces $\{\vec{v}_1, \dots, \vec{v}_p\}$ at each of the p fixed nodes, and similarly assemble

$$U = \begin{bmatrix} \vec{u}_1 & \cdots & \vec{u}_q \end{bmatrix}, \quad V = \begin{bmatrix} \vec{v}_1 & \cdots & \vec{v}_p \end{bmatrix}, \quad W = \begin{bmatrix} U & V \end{bmatrix}. \tag{6.1e}$$

[1] All vectors in $\mathbb{R}^d$ are denoted with an arrow (e.g., $\vec{q}_i$). All other vectors [see, e.g., (7.4b)] are denoted in bold (e.g., $\mathbf{x}$).

Internally, denote by $(\vec{d}_k\, x_k)$ and $-(\vec{d}_k\, x_k)$ the forces that member $\vec{m}_k$ applies at nodes $\vec{n}_{k,2}$ and $\vec{n}_{k,1}$, respectively, where x_k denotes the tension force (if positive) or compression force (if negative) in member $\vec{m}_k$. Thus, the matrix of directed internal member forces may be written simply as DX, where $X = \text{diag}(\mathbf{x})$. The cumulative force $\vec{f}_k$ at each node $\vec{n}_k$, due to the sum of *all* of the directed internal forces applied by each connected member $\vec{m}_j$, is then given simply by applying the connectivity matrix C such that

$$\begin{bmatrix}\vec{f}_1 & \cdots & \vec{f}_n\end{bmatrix} = F = -DXC, \tag{6.2a}$$

with the minus sign because positive x_k denotes *tension* forces in the member direction $\vec{d}_k$. Note that no trig is involved! *Static equilibrium* is reached simply when sum of the internal forces at each node, F, is in balance with (that is, equal and opposite to) the external forces W applied at each node such that

$$F = -DXC = -W \quad \Rightarrow \quad DXC = \begin{bmatrix}U & V\end{bmatrix}. \tag{6.2b}$$

For any truss structure, as defined in (6.1), with m members M connecting q free nodes Q and p fixed nodes P via the connectivity matrix C, and with external forces U applied at each free node, static equilibrium thus gives the linear system of equations (6.2b) in the m unknown internal forces $\{x_1, \ldots, x_m\}$ and the p unknown reaction forces V. As discussed further in §7.1.1, this linear system of equations may have 0, 1, or ∞ solutions, depending on the setup of the problem. The problem of determining the static equilibrium may be simplified by leveraging the partitioning $C = \begin{bmatrix}C_Q & C_P\end{bmatrix}$, and first solving for the internal forces at static equilibrium via

$$DXC_Q = U. \tag{6.3a}$$

These conditions of static equilibrium, which are *linear* in the unknowns x_k on the main diagonal of X, may easily be rewritten (automatically, in software) in terms of these unknown member forces rearranged as a vector $\mathbf{x}$, and represented in the standard matrix/vector form

$$A\mathbf{x} = \mathbf{b}, \tag{6.3b}$$

and subsequently solved for $\mathbf{x}$, after which the reaction forces may be computed directly via $V = DXC_P$.

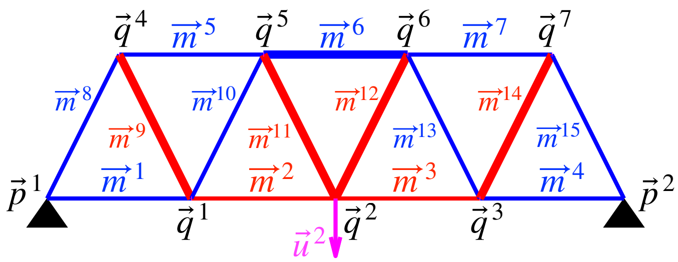
6.1.1 Brief review of the Singular Value Decomposition (SVD)

Consider now an arbitrary $\hat{m} \times \hat{n}$ matrix $\hat{A}$, and define the four fundamental subspaces [4]:
(a) the *column space* of $\hat{A}$ is the space of all $\mathbf{y}$ such that $\hat{A}\,\mathbf{x} = \mathbf{y}$ for some $\mathbf{x}$,
(b) the *row space* of $\hat{A}$ is the space of all $\mathbf{x}$ such that $\hat{A}^H\,\mathbf{y} = \mathbf{x}$ for some $\mathbf{y}$,
(c) the *left nullspace* of $\hat{A}$ is the space of all $\mathbf{y}$ such that $\hat{A}^H\,\mathbf{y} = 0$, and
(d) the *nullspace* of $\hat{A}$ is the space of all $\mathbf{x}$ such that $\hat{A}\,\mathbf{x} = 0$.
The left nullspace is the orthogonal complement of the column space, and the nullspace is the orthogonal complement of the row space. That is, the vectors in the left nullspace and those in the column space are mutually orthogonal, and the vectors in the nullspace and those in the row space are mutually orthogonal; further, any $\mathbf{x}$ may be written as the sum of one vector from the row space and one vector from the nullspace, and any $\mathbf{y}$ may be written as the sum of one vector from the column space and one vector from the left nullspace. The mappings from the row space to the column space via $\hat{A}$, and from the column space back to the row space via $\hat{A}^H$ (or, via $\hat{A}^+$, defined below), are unique.

It is also useful to refer to the components of the block decomposition of the SVD [5], which is defined as

$$\hat{A}_{\hat{m}\times\hat{n}} = \hat{U}_{\hat{m}\times\hat{m}}\,\hat{\Sigma}_{\hat{m}\times\hat{n}}\,\hat{V}^H_{\hat{n}\times\hat{n}} = \begin{bmatrix}\underline{U}_{\hat{m}\times r} & \overline{U}_{\hat{m}\times(\hat{m}-r)}\end{bmatrix} \begin{bmatrix}\underline{\Sigma}_{r\times r} & 0\\ 0 & 0\end{bmatrix} \begin{bmatrix}\underline{V}_{\hat{n}\times r} & \overline{V}_{\hat{n}\times(\hat{n}-r)}\end{bmatrix}^H,$$

where $\underline{\Sigma}$ is diagonal with real, non-negative elements σ_i on the main diagonal, arranged in descending order, U and V are unitary, and r is the *rank* of the matrix A. Note also that $\hat{A} = \underline{U}\underline{\Sigma}\underline{V}$.



$$\mathbf{x}_{1:7} = \begin{pmatrix} -0.250 \\ 0.250 \\ 0.250 \\ -0.250 \\ -0.500 \\ -1.000 \\ -0.500 \end{pmatrix}, \quad \mathbf{x}_{8:15} = \begin{pmatrix} -0.559 \\ 0.559 \\ -0.559 \\ 0.559 \\ 0.559 \\ -0.559 \\ 0.559 \\ -0.559 \end{pmatrix}$$

Figure 6.1: A Warren truss structure with a centrally-applied load, indicating the notation used in the analysis. In this text, red members (positive force) are in tension, blue members (negative force) are in compression.

```
% script RR_Truss_Warren4.m
% Set up a Warren truss with 4 sections, solve for its internal forces, and plot
P=[ 0  1;   % Columns denote (x,y) locations of each of the p=2 fixed nodes (normalized)
    0  0];
Q=[ 2  4  6  1  3  5  7;    % Locations of each of the n=7 free nodes (normalized)
    0  0  0  2  2  2  2]/8;
U=[ 0  0  0  0  0  0  0;    % External forces on the n free nodes of the truss (normalized)
    0 -1  0  0  0  0  0];
CT=[ 1 -1  0  0  0  0  0  0  1  1  0  0  0  0  0;  % Connectivity of the truss
     0  1 -1  0  0  0  0  0  0  0  1  1  0  0  0;  % Note: each of the m=15 columns of C^T
     0  0  1 -1  0  0  0  0  0  0  0  0  1  1  0;  % (that is, each of the m=15 rows of C)
     0  0  0  0 -1  0  0 -1 -1  0  0  0  0  0  0;  % has exactly one entry equal to +1
     0  0  0  0  1 -1  0  0  0 -1 -1  0  0  0  0;  % and one entry equal to -1, indicating
     0  0  0  0  0  1 -1  0  0  0  0 -1 -1  0  0;  % which two nodes that that member
     0  0  0  0  0  0  1  0  0  0  0  0  0 -1 -1;  % connects. It doesn't matter which is
    -1  0  0  0  0  0  0  1  0  0  0  0  0  0  0;  % taken positive and which negative.
     0  0  0  1  0  0  0  0  0  0  0  0  0  0  1]; C=CT';
% Now, convert the D*X*CQ=U problem in (6.3a) to the standard A*x=u form in (6.3b)
[A,b]=RR_Convert_DXCQ_eq_U_to_Ax_eq_b(Q,P,C,U);
% Then, just solve for the tension and compression in the members, and plot.
x=pinv(A)*b                 % This just implements (6.4b), Assumes zero pretension!
RR_Plot_Truss(Q,P,C,U,x);   % Plot truss (red=positive=tension, blue=negative=compression)
```

Figure 6.2: A simple code that calculates the equilibrium forces in a Warren truss (see Figure 7.1). To change the load, modify U; to change the configuration of the truss (see Figure 6.5), modify $\{$P, Q, C$\}$. The (provided) code RR_convert_DXCQ_eq_U_to_Ax_eq_u, which converts the $DXC_Q = U$ problem in (7.4a), which is linear in the diagonal elements of X, to the standard $A\mathbf{x} = \mathbf{b}$ form in (7.4b) using equationsToMatrix in Matlab, is straightforward. The (provided) code RR_Plot_Truss that plots the truss is trivial.

It follows that:
(i) r is both the number of independent rows of $\hat{A}$, and the number of independent columns of $\hat{A}$,
(ii) the r columns of $\underline{U}$ form an orthogonal basis of the column space of $\hat{A}$,
(iii) the r columns of $\underline{V}$ form an orthogonal basis of the row space of $\hat{A}$,
(iv) the $\hat{m} - r$ columns of $\overline{U}$ form an orthogonal basis of the left nullspace of $\hat{A}$, and
(v) the $\hat{n} - r$ columns of $\overline{V}$ form an orthogonal basis of the nullspace of $\hat{A}$.

Using the Moore-Penrose pseudoinverse $\hat{A}^+ = \underline{V}\underline{\Sigma}^{-1}\underline{U}^H$, least-squares problems may be solved as follows:

$$\hat{A}\,\mathbf{x} = \mathbf{b} + \boldsymbol{\epsilon} \tag{6.4a}$$

$$\Rightarrow \quad \mathbf{x} = \hat{A}^+\mathbf{b}, \tag{6.4b}$$

this $\mathbf{x}$ minimizes $\|\boldsymbol{\epsilon}\|$ and, amongst all $\mathbf{x}$ that satisfy (6.4a) for that minimal value of $\|\boldsymbol{\epsilon}\|$, also minimizes $\|\mathbf{x}\|$.

6.1.2 SVD analysis of the conditions of static equilibrium of a truss structure

The linear system of equations (7.4b) governing the member forces $\mathbf{x}$ at static equilibrium of a proposed truss may have 0, 1, or an infinite number of solutions. Stated differently, performing an SVD of the $\hat{m} \times \hat{n}$ matrix A, with $\hat{m} = dq$ and $\hat{n} = m$ in the analysis of §6.1.1, where d is the dimension of the problem considered, q is the number of free nodes, and m is the number of members, the problem in (7.4b) is said to be:

(a) *potentially inconsistent* if $r < \hat{m}$, and thus A has some rows which are linearly dependent on the other rows [in this case, (7.4b) will either have 0 solutions or at least one solution, depending upon whether or not the external force vector $\mathbf{u}$ is spanned by the columns of $\underline{U}$], and/or

(b) *underdetermined* if $r < \hat{n}$, and thus there are fewer independent equations than there are unknowns [in this case, if (7.4b) has a solution $\mathbf{x}$, then $\mathbf{x}$ plus any linear combination of the columns of $\overline{V}$ is also a solution].

The equations of static equilibrium of a truss structure in (7.4b), $A\mathbf{x} = \mathbf{u}$, may thus be:

- potentially inconsistent only ($\hat{m} > r = \hat{n}$), with 0 or 1 solution depending on $\mathbf{u}$,
- underdetermined only ($\hat{n} > r = \hat{m}$), with ∞ solutions,
- potentially inconsistent *and* underdetermined ($\hat{n} > r$, $\hat{m} > r$), with 0 or ∞ solutions depending on $\mathbf{u}$, or
- neither potentially inconsistent nor underdetermined ($\hat{n} = \hat{m} = r$), with exactly 1 solution.

The last condition above is called *static determinance*; notwithstanding its overemphasis in undergraduate-level pedagogical texts, this last condition is the exception, not the norm, in practical applications.

An example application of the above analysis framework to the Warren truss depicted in Figure 7.1 is given in Figure 6.2. If a solution to the problem of static equilibrium in (7.4b) exists (that is, if $r = dq$, or $r < dq$ but $\overline{U}^T \mathbf{b} = 0$), then the solution with zero pretension is given by $\mathbf{x} = \hat{A}^{+}\mathbf{b}$ [see (6.4b)], as implemented in this code; if such a solution exists, then *all* solutions to this static equilibrium problem are given by the value of $\mathbf{x}$ returned by this code plus any linear combination of the columns of $\overline{V}$.

6.1.3 Elimination of infinitesimal modes from a potentially inconsistent structure

If A in (7.4b) is *potentially inconsistent*, with $r < dq$, then the corresponding truss has infinitesimal mechanisms associated with zero deformation energy. Such configurations can be either *unstable* or *soft*. The first case (instability) is, clearly, catastrophic, with small disturbances acting on the structure leading rapidly to failure; visualize, e.g., two opposing members under compression meeting at a node (i.e., a pin joint in 2D, or a ball joint in 3D) where external disturbance forces may be applied, with no additional members attached to stabilize.

The second case (soft or “wobbly” modes), though not catastrophic, is also an undesirable feature for a truss; visualize, e.g., two opposing member under tension meeting at a node where external forces may be applied, again with no additional members attached to stabilize. In this case, there are no finite force distributions in the members that can sustain a range of disturbances on the nodes (specifically, any disturbance forces U generated with components in the directions of the columns of $\overline{U}$) for this free node configuration Q. However, assuming that the members are somewhat *elastic*, a significant *deformation* of the free node configuration vector Q may well lead to a *deformed* configuration that can sustain the problematic disturbance profile. Unfortunately, a different disturbance profile will generally lead to a different deformation of the structure, so this approach leads to “wobbly” structures in the presence of unsteady external loads. Soft modes are thus also undesirable in a truss, as they lead to relatively large deformations of Q in response to small disturbances U.

Fortunately, as discussed further in [2], the condition of potential inconsistency in (7.4b), with $r < dq$ (and, the corresponding presence of unstable or soft modes), can often be removed entirely from a truss with a given configuration of members simply by judiciously adding more members, under tension, attached to the problematical nodes, thereby increasing r if the new members are well positioned.

?

Figure 6.3: Some other 2D bridge trusses

?

Figure 6.4: Some 2D roof trusses

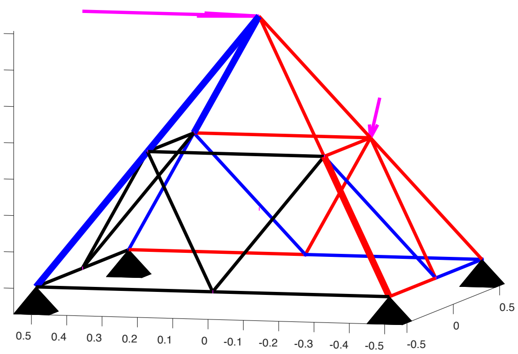


Figure 6.5: Some 3D trusses

References

[1] Timoshenko, S (2017) *Engineering Mechanics*, 5Th edition. McGraw Hill. 6-1

[2] Likins, PW (1973) *Elements of engineering mechanics*. McGraw-Hill. 6-1

[3] Skelton, R.E., & de Oliveira, M.C. (2009) *Tensegrity Systems*. Springer. 6-1, 7-1, 7-2, 7-3

[4] Strang, G (2006) Linear Algebra and Its Applications, 4th Edition. 2-6, 6-2

[5] Golub, G, and Van Loan, C (2013) Matrix Computations, 4th Edition. 2-8, 6-2

[6] Masic, M., Skelton, R.E., & Gill, P.E. (2006) Optimization of tensegrity structures. *International Journal of Solids and Structures* **43** (16), 4687-4703. 6-4, 7-4

Chapter 7

Tensegrity Structures

7.1 Static equilibria of 2D and 3D tensegrity structures

We now make a minor adjustment[1] (see [1]) to the analysis in §6.1, to characterize *tensegrity* structures, with
• b *bars* $\{\vec{b}_1, \ldots, \vec{b}_b\}$, each in pure compression, and s *strings* $\{\vec{s}_1, \ldots, \vec{s}_s\}$, each in pure tension,
• $m = b + s$ total *members* $\{\vec{m}_1, \ldots, \vec{m}_m\}$ (that is, the b bars and s strings, taken together), and
• q *free nodes* $\{\vec{q}_1, \ldots, \vec{q}_q\}$ and p *fixed nodes* $\{\vec{p}_1, \ldots, \vec{p}_p\}$, collectively called the $n = q + p$ *nodes* $\{\vec{n}_1, \ldots, \vec{n}_n\}$.
The n nodal locations $\vec{n}_i$ are each vectors from the origin of the reference frame used in $\mathbb{R}^d$, where $d = 2$ or 3 is the dimension of the problem considered, and are assembled as follows:

$$\begin{aligned} &Q = \begin{bmatrix}\vec{q}_1 & \cdots & \vec{q}_q\end{bmatrix}, \quad P = \begin{bmatrix}\vec{p}_1 & \cdots & \vec{p}_p\end{bmatrix} \\ \Rightarrow \quad &N = \begin{bmatrix}\vec{n}_1 & \cdots & \vec{n}_n\end{bmatrix} = \begin{bmatrix}Q & P\end{bmatrix} = \begin{bmatrix}\vec{q}_1 & \cdots & \vec{q}_q & \vec{p}_1 & \cdots & \vec{p}_p\end{bmatrix}; \end{aligned} \tag{7.1a}$$

Q, which defines the locations of the free nodes, is called the *configuration matrix* of the structure. Each member $\vec{m}_k = \vec{n}_{k,1} - \vec{n}_{k,2}$ connects two nodes, $\vec{n}_{k,1}$ and $\vec{n}_{k,2}$, at least one of which is free, and are organized as follows:

$$B = \begin{bmatrix}\vec{b}_1 & \cdots & \vec{b}_b\end{bmatrix}, \quad S = \begin{bmatrix}\vec{s}_1 & \cdots & \vec{s}_s\end{bmatrix}, \quad M = \begin{bmatrix}\vec{m}_1 & \cdots & \vec{m}_m\end{bmatrix} = \begin{bmatrix}B & S\end{bmatrix} = \begin{bmatrix}\vec{b}_1 & \cdots & \vec{b}_b & \vec{s}_1 & \cdots & \vec{s}_s\end{bmatrix}; \tag{7.1b}$$

it is useful to define a vector of member lengths $\boldsymbol{\ell}$, and a matrix of normalized member directions D, such that

$$\ell_k = \|\vec{m}_k\|, \quad \vec{d}_k = \vec{m}_k/\ell_k, \quad D = \begin{bmatrix}\vec{d}_1 & \cdots & \vec{d}_m\end{bmatrix} = \begin{bmatrix}D^{\mathrm{b}} & D^{\mathrm{s}}\end{bmatrix}, \tag{7.1c}$$

where $\|\vec{d}_k\| = 1$ for all k. Note that the first b columns of M and D correspond to the bars, and the last s columns of M and D correspond to the strings. The connectivity of a structure, relating the n nodes N in (7.1a) to the m members M in (7.1b), is described via its *connectivity matrix* C, defined and partitioned such that

$$M = N\,C^T, \quad C = \begin{bmatrix}C_Q & C_P\end{bmatrix} = \begin{bmatrix}C_B \\ C_S\end{bmatrix} \quad \Rightarrow \quad M = \begin{bmatrix}Q & P\end{bmatrix}\begin{bmatrix}C_Q^T \\ C_P^T\end{bmatrix} = N\,\begin{bmatrix}C_B^T & C_S^T\end{bmatrix} = \begin{bmatrix}B & S\end{bmatrix}, \tag{7.1d}$$

where, denoting $\mathbf{e}_j$ as the vector in the j'th column of the identity matrix, each column[2] of C^T is given by $(\mathbf{e}_{k,1} - \mathbf{e}_{k,2})$, indicating the two nodes $\vec{n}_{k,1}$ and $\vec{n}_{k,2}$ that member $\vec{m}_k$ of the structure connects, with one entry equal to 1, one entry equal to -1, and all other entries equal to zero.

Consider the external forces $\{\vec{u}_1, \ldots, \vec{u}_q\}$, including the net effect of the weights of the members themselves, applied to each of the q free nodes, and the reaction forces $\{\vec{v}_1, \ldots, \vec{v}_p\}$ at each of the p fixed nodes, and assemble

$$U = \begin{bmatrix}\vec{u}_1 & \cdots & \vec{u}_q\end{bmatrix}, \quad V = \begin{bmatrix}\vec{v}_1 & \cdots & \vec{v}_p\end{bmatrix}, \quad W = \begin{bmatrix}U & V\end{bmatrix}. \tag{7.1e}$$

[1] The very minor differences between the intro to §6.1 and intro to §7.1 are highlighted here in red.
[2] By (7.1d), note that the first b columns of C^T define the bars, and the last s columns of C^T define the strings.

Internally, denote[3] by $(\vec{d}_k\, x_k)$ and $-(\vec{d}_k\, x_k)$ the forces that member $\vec{m}_k$ applies at nodes $\vec{n}_{k,2}$ and $\vec{n}_{k,1}$, respectively, where x_k denotes the tension force (if positive, as required in the strings) or compression force (if negative, as we are looking for in the bars) in member $\vec{m}_k$; for further discussion, see §7.1.1. That is, the bars can carry compressive or tensile forces, but strings can only carry tensile forces; i.e., $x^{\text{s}}_j \geq 0$ for $j = 1, \ldots, s$.

The internal member forces may be written

$$\begin{aligned} DX &= \begin{bmatrix} D^{\text{b}} & D^{\text{s}} \end{bmatrix} \begin{bmatrix} X^{\text{b}} & 0 \\ 0 & X^{\text{s}} \end{bmatrix} = \begin{bmatrix} D^{\text{b}}\, X^{\text{b}} & D^{\text{s}}\, X^{\text{s}} \end{bmatrix}, \\ \text{where} \quad X &= \text{diag}(\mathbf{x}), \quad \mathbf{x} = \begin{bmatrix} \mathbf{x}^{\text{b}} \\ \mathbf{x}^{\text{s}} \end{bmatrix}, \quad X^{\text{b}} = \text{diag}(\mathbf{x}^{\text{b}}), \quad X^{\text{s}} = \text{diag}(\mathbf{x}^{\text{s}}). \end{aligned} \tag{7.2}$$

As before, the cumulative force $\vec{f}_k$ at each node $\vec{n}_k$, due to the sum of *all* of the directed internal forces applied by each connected member $\vec{m}_j$, is then given simply by applying the connectivity matrix C such that

$$\begin{bmatrix} \vec{f}_1 & \cdots & \vec{f}_n \end{bmatrix} = F = -DXC, \tag{7.3a}$$

with the minus sign because positive x_k denotes *tension* forces in the member direction $\vec{d}_k$. Static equilibrium is reached simply when sum of the internal forces at each node, F, is in balance with (that is, equal and opposite to) the external forces W applied at each node such that

$$F = -DXC = -W \quad \Rightarrow \quad DXC = \begin{bmatrix} U & V \end{bmatrix}. \tag{7.3b}$$

For any structure as defined in (7.1), with m members M connecting q free nodes Q and p fixed nodes P via the connectivity matrix C, and with external forces U applied at each free node, static equilibrium thus gives the linear system of equations (7.3b) in the m unknown internal forces $\{x_1, \ldots, x_m\}$ and the p unknown reaction forces V. As discussed in §7.1.1, tensegrity system are *pretensionable*, so we are specifically looking for system configurations in which this linear system of equations has ∞ solutions. The problem of determining the static equilibrium may be simplified by leveraging the partitioning $C = \begin{bmatrix} C_Q & C_P \end{bmatrix}$, and first solving for the internal forces at static equilibrium via

$$DXC_Q = U. \tag{7.4a}$$

These conditions of static equilibrium, which are *linear in the unknowns* x_k on the main diagonal of X, may easily be rewritten (automatically, in software) in terms of these unknown member forces rearranged as a vector $\mathbf{x}$, with the external forces in U rearranged as a vector $\mathbf{b}$, and thus this linear system may be represented in the standard matrix/vector form

$$A\mathbf{x} = \mathbf{b}, \tag{7.4b}$$

and subsequently solved for $\mathbf{x}$, after which the reaction forces may be computed directly via $V = DXC_P$.

[3]Note that [1] defines and solves for the *force density* $\sigma_k = x_k/\ell_k$ in each member, rather than solving for the forces x_k themselves (where, again positive σ_k denotes tension and negative σ_k denotes compression). They further denote the force density in string $\vec{s}_j$ by γ_j (with, again, $\gamma_j > 0$ denoting *tension*), and the force density in bar $\vec{b}_i$ by λ_i (with, in contrast, $\lambda_i > 0$ denoting *compression*). Using that (slightly more complicated) notation, the present derivation is expressed by applying the relations

$$D\,X = M\,\Sigma = \begin{bmatrix} B & S \end{bmatrix} \begin{bmatrix} -\Lambda & 0 \\ 0 & \Gamma \end{bmatrix} = \begin{bmatrix} -B\,\Lambda & S\,\Gamma \end{bmatrix} \quad \text{where} \;\; \Sigma = \text{diag}(\sigma_1, \ldots, \sigma_m), \;\; \Lambda = \text{diag}(\lambda_1, \ldots, \lambda_b), \;\; \Gamma = \text{diag}(\gamma_1, \ldots, \gamma_s).$$

?

Figure 7.1: A 3D tensegrity structure with an applied compressive load, indicating the notation used in the analysis. Note that, to the same structural configuration, pretension is applied one way to the solution at left, and the other way to the solution at right, providing "dual" arrangments of the bars (blue) and the strings (red).

7.1.1 SVD analysis of the conditions of static equilibrium of tensegrity structures

If a tensegrity structure is designed and pretensioned correctly, the force in each of the strings will be positive (indicating tension), and the force in each of the bars will be negative (indicating compression). The art of tensegrity systems design, is to determine for what practically useful system configurations (and, with what applied pretensions) that this can be achieved. A few examples are given in this chapter; a much more comprehensive list is provided in [1].

The linear system of equations (7.4b) governing the member forces $\mathbf{x}$ at static equilibrium of a valid tensegrity structure has an infinite number of solutions. Stated differently, performing an SVD of the $\hat{m} \times \hat{n}$ matrix A, with $\hat{m} = dq$ and $\hat{n} = m$ in the analysis of §6.1.1, where d is the dimension of the problem considered, q is the number of free nodes, and m is the number of members, the problem in (7.4b) is said to be:

(a) *potentially inconsistent* if $r < \hat{m}$, and thus A has some rows which are linearly dependent on the other rows [in this case, (7.4b) will either have 0 solutions or at least one solution, depending upon whether or not the external force vector $\mathbf{u}$ is spanned by the columns of $\underline{U}$], and/or

(b) *underdetermined* if $r < \hat{n}$, and thus there are fewer independent equations than there are unknowns [in this case, if (7.4b) has a solution $\mathbf{x}$, then $\mathbf{x}$ plus any linear combination of the columns of $\overline{V}$ is also a solution].

The equations of static equilibrium of a truss structure in (7.4b), $A\mathbf{x} = \mathbf{u}$, may thus be:

- potentially inconsistent only ($\hat{m} > r = \hat{n}$), with 0 or 1 solution depending on $\mathbf{u}$,
- underdetermined only ($\hat{n} > r = \hat{m}$), with ∞ solutions,
- potentially inconsistent *and* underdetermined ($\hat{n} > r$, $\hat{m} > r$), with 0 or ∞ solutions depending on $\mathbf{u}$, or
- neither potentially inconsistent nor underdetermined ($\hat{n} = \hat{m} = r$), with exactly 1 solution.

The last condition above is called *static determinance*; notwithstanding its overemphasis in undergraduate-level pedagogical texts, this last condition is the exception, not the norm, in practical applications.

An example application of the above analysis framework to the Warren truss depicted in Figure 7.1 is given in Figure 6.2. If a solution to the problem of static equilibrium in (7.4b) exists (that is, if $r = dq$, or $r < dq$ but $\overline{U}^T \mathbf{b} = 0$), then the solution with zero pretension is given by $\mathbf{x} = \hat{A}^{+}\mathbf{b}$ [see (6.4b)], as implemented in this code; if such a solution exists, then *all* solutions to this static equilibrium problem are given by the value of $\mathbf{x}$ returned by this code plus any linear combination of the columns of $\overline{V}$.

7.1.2 Elimination of infinitesimal modes from a potentially inconsistent structure

If A in (7.4b) is *potentially inconsistent*, with $r < dq$, then the corresponding truss has infinitesimal mechanisms associated with zero deformation energy. Such configurations can be either *unstable* or *soft*. The first case (instability) is, clearly, catastrophic, with small disturbances acting on the structure leading rapidly to failure; visualize, e.g., two opposing members under compression meeting at a node (i.e., a pin joint in 2D, or a ball joint in 3D) where external disturbance forces may be applied, with no additional members attached to stabilize.

The second case (soft or "wobbly" modes), though not catastrophic, is also an undesirable feature for a truss; visualize, e.g., two opposing member under tension meeting at a node where external forces may be applied, again with no additional members attached to stabilize. In this case, there are no finite force distributions in the members that can sustain a range of disturbances on the nodes (specifically, any disturbance forces U generated with components in the directions of the columns of $\overline{U}$) for this free node configuration Q. However, assuming that the members are somewhat *elastic*, a significant *deformation* of the free node configuration vector Q may well lead to a *deformed* configuration that can sustain the problematic disturbance profile. Unfortunately, a different disturbance profile will generally lead to a different deformation of the structure, so this approach leads to “wobbly” structures in the presence of unsteady external loads. Soft modes are thus also undesirable in a truss, as they lead to relatively large deformations of Q in response to small disturbances U.

Fortunately, as discussed further in [2], the condition of potential inconsistency in (7.4b), with $r < dq$ (and, the corresponding presence of unstable or soft modes), can often be removed entirely from a truss with a given configuration of members simply by judiciously adding more members, under tension, attached to the problematical nodes, thereby increasing r if the new members are well positioned.

References

[1] Skelton, R.E., & de Oliveira, M.C. (2009) *Tensegrity Systems*. Springer. 6-1, 7-1, 7-2, 7-3

[2] Masic, M., Skelton, R.E., & Gill, P.E. (2006) Optimization of tensegrity structures. *International Journal of Solids and Structures* **43** (16), 4687-4703. 6-4, 7-4

Appendix A

Matlab programming

A useful definition of a **small-scale numerical problem** is a calculation that takes longer to code and debug than it does to run. By this definition, many calculations that you will perform in science, engineering, and other disciplines, both as a student and in industry, are indeed small-scale numerical problems.

Matlab (a **portmanteau word**[1] formed from **mat**rix **lab**oratory) is a powerful high-level programming language marketed by MathWorks. Though expensive[2], Matlab has become something of a de facto industry standard for many classes of small-scale numerical problems, in areas such as linear algebra, data analysis & visualization, control design, system identification, and optimization.

GNU Octave is a powerful, free, community-developed alternative to Matlab that is almost entirely compatible with the (well-established and documented) Matlab syntax. You are assured free access to a legal copy of Octave for any computer platform that you might use in the future, so it is a good idea to test all of the major codes that you develop in Matlab syntax in both Matlab and Octave, as we have attempted to do in this series of texts[3], so that you can be assured that you will be able to run them without difficulty in the future.

Both Matlab & Octave[4] provide an interactive, user-friendly environment in which the plotting of simulation results is especially simple. These programming environments are thus quite useful as intuitive testing grounds in which one can experiment with small-scale numerical problems on a laptop or desktop computer. There are two main directions one can go from there:

(1) embedding numerical algorithms efficiently into inexpensive low-power microcontrollers for controlling robotic systems, a class of problems that we focus on in particular throughout *RR*, and
(2) designing numerical algorithms that efficiently scale to much larger numerical problems which tax the capabilities of the largest and most modern computational platforms that you can afford to use, a class of problems that we focus on in *NR*.

In both cases, low-level compiler-based languages [such as C, Fortran, and many others] are strongly preferred, as they give the programmer much more precise control over both the memory usage and the parallelization of the numerical algorithm. Conversion of numerical algorithms from Matlab syntax to the syntax of such lower-level languages is generally straightforward, as discussed further in, e.g., §11.4 of *NR*.

---

[1] A portmanteau word is formed out of parts of other words, which is common in the naming of computer hardware and software. For example, **Fortran** is a portmanteau word formed from **for**mula **tran**slation, **codec** from **co**der/**dec**oder, **voxel** from **vo**lumetric pi**xel**, etc. Such words are often formed informally as new technology is developed, then become established through usage.

[2] Note that special Matlab For Students deals are available on many college and university campuses.

[3] Please contact the author if you encounter errors running any of the algorithms in this series of texts, including *Renaissance Robotics* and *Numerical Renaissance*, and the associated Renaissance Repository, in recent versions of either Matlab or Octave.

[4] A few popular alternatives to Matlab & Octave well suited for both small-scale numerical problems (working with floating-point numbers) as well as symbolic computations (that is, software-based manipulation of mathematical expressions for solving algebra and calculus problems) include Python/NumPy/SciPy/SymPy, R, Julia, Scilab, Maxima, Mathematica/Alpha, and CPL.

A.1 Fundamentals of both Matlab and Octave

Once you get Matlab or Octave up and running, you will likely find that no manual or formal classroom instruction on either language itself is even necessary. Most of the basic constructs available in such languages [primarily, basic arithmetic on floating-point numbers, **for** loops, **if** statements, and **function** calls] can generally be understood easily simply by examining sample codes, such as those developed throughout *RR* and *NR*.

It is helpful to recognize that complex problems are solved efficiently on modern computers simply by sequencing appropriately basic arithmetic, for loops, if statements, function calls, and data storage and retrieval together, using logic which is admittedly sometimes subtle. It is generally the logic itself, and the judgement and reasoning involved in organizing it, that makes The Art of Computer Programming a skill that takes years to master; the syntax of the language best suited for the job (Matlab, C, or something else) is generally something that is quite easy to pick up, or convert to, by examining a handful of well-written example codes.

Convenient built-in command names in Matlab/Octave are all intuitive (**sin** for computing the sine, **eig** for computing eigenvalues/eigenvectors, etc.), and extensive **help** for all commands is readily available in both Matlab and Octave, at the >> prompt in the command window, simply by typing, for example,

```
>> help eig
```

Even more information is available online. These help pages also point you to several related commands, which can be used to learn what you need to know about any given aspect of Matlab or Octave very quickly.

To help get you off to a fast start, we now introduce some of the fundamental constructs used in Matlab and Octave, then explain some of their more subtle features. To begin, Matlab or Octave can function as an ordinary calculator. At the command prompt, try typing[5]

```
>> 1+1
```

Matlab or Octave should reassure you that the universe is still in good order. Note that you can always scroll back to see the preliminary definitions and calculations that led to a particular result using the up arrow on your keyboard. To enter a matrix, type

```
>> A=[1 2 3; 4 5 6; 7 8 0]
```

Note that matrix elements are separated by commas or (where it can be done without ambiguity) spaces, and a semicolon indicates the end of each row of the matrix. Matlab/Octave responds with

```
A =
     1     2     3
     4     5     6
     7     8     0
```

By default, Matlab/Octave operates in a **verbose** mode[6] in which the results generated by any given command will be printed on the screen as soon as they are calculated. Once a code segment is debugged, such a verbose behavior quickly becomes tedious, and slows the computer down. To suppress this behavior, simply type a semicolon after any command that would otherwise dump output to the screen; the use of semicolons after calculations or function calls also allows several commands to be included on a single line, such as

```
>> A=[1 2 3; 4 5 6; 7 8 0]; x=5;
```

To put multiple commands on a single line without suppressing echo mode, separate the commands by commas (try it!). Three periods in a row means that the present command is continued on the following line, as in:

```
>> A=[1 2 3;   ...
      4 5 6;   ...
      7 8 0];
```

[5]To get maximum value from this appendix, we recommend copying/pasting (or, retyping) the commands following the >> prompts in the text into your own Matlab/Octave window, modifying it a bit, and checking that the output generated makes sense.

[6]This verbose behavior can be further augmented by toggling on the **echo** command, which displays the actual statements encountered during execution of a script (see §A.2.1). This command, which apparently evolved from the TRON command of the 1980s vintage BASIC programming language, prints so much information to the screen that its practical utility is actually somewhat limited.

The legibility of your code can be substantially improved by aligning long expressions in a natural fashion using spaces or tabs, as illustrated above. Elements of a matrix can also be arithmetic expressions, such as 3*pi, etc.; when doing this, it often necessary to separate matrix elements by commas to remove any possible ambiguity.

Matlab syntax has control flow statements, such as **for** loops, similar to other programming languages. Note that each **for** must be matched by an **end**. To illustrate, the commands

```
>> for j=1:10, a(j)=j^2; end, a, b=[0:2:10]
```

build row vectors (try it!), whereas the commands

```
>> for j=1:10, c(j,1)=j^2; end, c, d=[0:2:10]'
```

build column vectors. In most cases, you want the latter, not the former. *The most common mistake made in Matlab syntax is to build a row vector when you intend to build a column vector*, as their use in Matlab/Octave is usually not interchangeable; thus, pay especially close attention to this issue if your code is misbehaving.

The format of a **while** statement is similar to that of **for**, but exits at the control of a logical condition:

```
>> m=0; while m<7, m=m+2; end, m
```

An **if** statement may be used as follows[7]:

```
>> n=7; if n>0, sgn=1, elseif n<0, sgn=-1, elseif n==0, sgn=0, else disp('undefined'), end
```

The (related) **switch**/**case** construction allow one to check a single variable against several possible conditions

```
>> switch n, case -1, c='N', case 0, c='Z', case 1, c='P', otherwise, c='?', end
```

A column vector y can be premultiplied by a matrix A and the result stored in a column vector z with, e.g.,

```
>> A=[1 2 3; 4 5 6; 7 8 0], y=[1 2 3]', z=A*y
```

Subsequent multiplication of the vector z by a scalar, like the predefined constant **pi**, may be accomplished with

```
>> w=pi*z
```

A 3×3 matrix A with complex random entries, each with real and imaginary parts uniformly distributed between 0 and 1[8], its conjugate $B = \overline{A}$, its transpose $C = A^T$, and its conjugate transpose $D = A^H = \overline{A^T}$ (see §1 of *NR*), may be generated as follows

```
>> A=rand(3,3)+sqrt(-1)*rand(3,3), B=conj(A), C=A.', D=A'
```

The inverse of a square matrix (that is, the matrix B such that $B\,A = I$, if it exists) may be obtained by typing

```
>> B=inv(A), check=B*A
```

For pedagogical purposes, this inverse command is rewritten in §2 of *NR*. As mentioned there, you should never actually compute a matrix inverse[9] in a **production code** (that is, in a code designed to run at the maximum possible speed, without failure), though it is sometimes convenient to compute a matrix inverse in a **test code** (that is, in a code used for demonstration purposes only, on small-scale numerical problems).

A 5×5 identity matrix may be constructed with

```
>> E=eye(5)
```

Tridiagonal matrices (see §1.2.7 and §2.2.5 of *NR*) may be constructed by, e.g., the following command:

```
>> m=5, x=randn(m-1,1), T=1*diag(ones(m-1,1),-1)-2*diag(ones(m,1),0)+diag(x,1)
```

[7]When in the Matlab command window, using the up and down arrows allows you to scroll back and forth through recently executed commands. For example, after copy/pasting/running the commands shown here, scroll back to it using the up arrow, walk to the left of the line with the left arrow, change n=7 to n=**NaN** or **Inf**, hit enter, and see if the new answer makes sense (see §**??**).

[8]Note that the command A=**randn**(2,3) generates a 2×3 matrix A with real, random, independent, normally-distributed entries, each sampled from a Gaussian probability distribution with zero mean and standard deviation 1 (see §6 of *NR*), and the command A=**randi**(13,2,3) generates a 2×3 matrix with positive integer entries uniformly distributed between 1 and 13.

[9]The algorithm to compute a matrix inverse is computationally very expensive, as discussed in §2 of *NR*, and destroys any known sparsity structure (that is, sets of elements known to be zero) in the original matrix, as discussed in §1 of *NR*.

There are two "matrix division" commands in Matlab/Octave, **mldivide**, also written as \, and **mrdivide**, also written as /; if A is a nonsingular square matrix, then A\B and B/A correspond formally to left and right multiplication of B (which must be of the appropriate size that the product is well defined) by the inverse of A [i.e., **inv**(A)*B and B***inv**(A), respectively]. However, the commands A\B and B/A obtain these answers directly via **Gaussian elimination with pivoting**, as developed from scratch (again, for pedagogical reasons) in §2 of *NR*, while leaving the matrix A intact, *without* computing **inv**(A) along the way (which, as mentioned in Footnote 9 above, is prohibitively expensive for large matrices). Thus, to solve a system A*x=b for the unknown vector x, one may simply type, for example,

```
>> A=[1 2 3; 4 5 6; 7 8 0]; b=[5 8 -7]'; x=A\b
```

which results in

```
x =
    -1
     0
     2
```

To check this result, just type

```
>> A*x
```

which verifies, as expected, that

```
ans =
     5
     8
    -7
```

Starting with the innermost group(s) of operations nested in parentheses and working outward, the usual precedence rules are observed by Matlab/Octave. First, all the exponentials are calculated. Then, all the multiplications and divisions are calculated. Finally, all the additions and subtractions are calculated. In each of these three catagories, the calculation proceeds from left to right through the expression. Thus

```
>> a=5/5*3, b=5/(5*3)
```

gives a=3 and b=0.3333. If in doubt, use parentheses to ensure the order of operations is as you intend.

Matrix sizes must be such that the requested linear algebra operation is well defined (see §1 of *NR*), or an error will be thrown. For example, suppose we have two column vectors x and y and wish to perform the component-wise product of each element of x with the corresponding element of y. Such a component-wise multiplication may be accomplished in Matlab syntax as, for example,

```
>> x=[1:5]';  y=[6:10]';  z=x.*y
```

Note that z=x*y throws an error, since this implies a linear algebra operation that is undefined (that is, a column vector times a column vector). In contrast, a row vector times a column vector is well defined, so z=x'*y is successful, generating the inner product of x and y (try it!).

The period generally distinguishes matrix operations from component-wise operations, for example (try it!)

```
>> A=[1 2; 3 4], B=[5 6; 7 8]
>> C1=A^2,  D1=A*B,  E1=A/B   % Matrix operations!
>> C2=A.^2, D2=A.*B, E2=A./B  % Component-wise operations!
```

Typing **whos** lists all variables you have created up to that point, and typing **clear** removes these variables from your workspace. Typing **clc** clears the command window.

The **format** command toggles the number of significant figures printed to the screen, for example,

```
>> format long, pi, format short, pi
```

Various self-explanatory Matlab/Octave functions include: **factorial**, **abs**, **conj**, **real**, **imag**, **sin**, **cos**, **tan**, **asin**, **acos**, **atan**, **atan2**, **sinh**, **cosh**, **tanh**, **asinh**, **acosh**, **atanh**, **exp**, **log**, **log10**; some predefined constants[10] include **pi**, i, j,

[10] Note that Inf is actually signed in Matlab (see §**??**); try typing, e.g., 10^310 and -10^310.

eps, **Inf**, **NaN**. Your code can actually change such constants[11] (it is particularly common to use i and j as indexing variables); be careful if you do this, and later need to use these constants as originally defined!

Matlab and Octave are distributed with many special additional functions to aid in linear problem-solving, control design, etc. Many of these advanced built-in functions are themselves just prewritten m-files (see below) that can be opened and accessed by the user for examination with, for example, a command such as **open bode**. In most cases, *RR* and *NR* avoid most of these convenient **black-box** functions, instead working up the core of many of them from scratch, to remove the mystery that might otherwise be associated with their use.

Sometimes, Matlab or Octave will suspend the printing of text to the command window, or the drawing of a plot to a figure window, until later computations are finished or a **pause** statement is reached. The **fflush**(stdout) command in Octave, and the **drawnow**('update') command in Matlab, can be used to force this output to be printed or drawn. This is one of the few little (yet, annoying) differences between Matlab and Octave.

All of Matlab's "random" number functions, including the **rand**, **randi**, and **randn** commands mentioned above, draw values from a shared pseudorandom number generator (PRNG). Matlab actually has several deterministic algorithms implemented for pseudorandom number generation with good statistical properties implemented; it uses the Mersenne Twister by default. Matlab's PRNG algorithm has a "seed" that is reset every time Matlab is restarted (thereby generating the exact same sequence of pseudorandom numbers every time Matlab is restarted). This seed can be manually reset to the default initial state with the **rng**('default') command, or can be manually set to a "random" initial state, based on the current time, using the **rng**('shuffle') command.

A.2 Matlab programming procedures: stay organized!

As an alternative to interactive mode, you can also save a series of Matlab/Octave commands in m-files, which are just ASCII (aka plain text) files with descriptive filenames, ending in .m, containing a sequence of commands listed exactly as you would enter them if running interactively. When working on Matlab/Octave problems that take more than one line to express (that is, essentially, all the time, even when using Matlab/Octave as a simple calculator!), *it is imperative to write and run m-files rather than working in interactive mode*. By so doing, it is self evident which calculation follows from which. Further, following this approach, the several commands typically required to perform a given calculation do not need to be retyped when the calculation or simulation needs to be modified and rerun, which is generally much more often than one would care to admit. *Staying organized with different versions of your m-files as a project evolves (even a fairly simple project!) is essential. Create new directories and subdirectories as appropriate for each problem you work on to stay organized, and to keep from accidentally overwriting previously written and debugged codes.*

To execute the commands in a script named[12] **foo**.m, type **foo** at the >> prompt. Any good text editor may be used to edit m-files. A % symbol in such a file indicates that the rest of that line is a comment. Typing **help foo** prints the first set of commented lines of **foo**.m to the screen; **type foo** prints the entire **foo**.m code to the screen. *Comment all m-files clearly, sufficiently, and succinctly*, focusing specifically on its inputs and outputs, so that you can come back to the code later and understand how it works. You can also print nicely to the screen (using, e.g., the **disp** or **fprintf** commands) to update the user on the code's progress as it runs; several codes discussed in this text use a RR_VERBOSE flag to turn such updates on or off. It also helps to use descriptive variable names within any code. There is an important tradeoff between succinctness and readability; this tradeoff should be made deliberately, don't go overboard on one side or the other.

[11] If you're bored, try redefining **pi**=3.2 and see how much it messes things up; be glad this wasn't established by legislative fiat!

[12] Almost all texts describing computer programming, dating back to the 1978 classic by Kernighan and Ritchie, make reference to expository codes named **foo** and **bar**. Following this convention is a small way one can pay respect to those greats in the field of computing who came before us.

A.2.1 The distinction between scripts and functions

There are two distinct types of m-files: **scripts** and **functions**.

A **script** is a set of Matlab/Octave commands that run just as if you had typed in each command interactively. A script has access to all previously-defined variables (that is, if called from the interactive window, it inherits the **base workspace**), and all variables that it defines are available for later inspection in that workspace, which is sometimes useful when debugging. In order to make a test script run the same way every time (repeatability is usually strongly desired in numerical calculations!), it is generally a good idea to put a **clear** command at the beginning of all of your scripts, so they always run "from scratch".

A **function**, on the other hand, is a set of commands that begins with a **function** declaration that defines that function's inputs and outputs; for example,

```
function [output1,output2] = bar(input1,input2,input3)
```

A function so defined may then be **called** (as in a compiler-based programming language) with the command

```
[c,d] = bar(a,b,c)
```

Note that some variables (in the above example, c) may be used as both inputs and outputs.

When a function is running, it can only reference those external variables that are transferred in via the input list with which it was called; in the present example, the function is only "aware" of the external variables {a,b,c}, which this function refers to internally as {input1,input2,input3}. A notable exception to this rule is those variables that are declared as **global** in multiple functions and (usually) the base workspace; in this case, a single copy of the variable so declared is shared, but only to those functions that include its global declaration. Global variables are usually denoted with both all capital letters and especially descriptive variable names (e.g., **global** FUNCTION_EVALUATION_COUNTER), to keep these special variables from being accidentally overwritten in the several different places that they might be used. Global variables should be used only sparingly.

The special variables **nargin** and **nargout** identify the number of input and output arguments, respectively, that are actually used when any given function is called. Input and output arguments are assigned from the left to the right, so any missing arguments in this call are necessarily those at the right end of each list. If some of the input arguments, often tagged as "optional", are omitted in the function call, logic may be implemented in the function to set these variables to certain default values; if this logic is not implemented properly, the function will crash when these variables (if left undefined by the function call) are first referenced. If some of the output arguments are omitted in the function call, logic may be implemented in the function to avoid explicit computation of these omitted output variables in order to reduce execution time.

After a function finishes running, the only variables that are modified in the workspace that called the function, as compared to before the function was run, are those (**nargout**) variables in the function's output list that are paired with corresponding variables in the command that called the function, in addition (possibly) to some of the variables declared as global. In the above example, if called as c2=**bar**(a,b,c1), the function **bar** only modifies[13] the single (that is, **nargout**=1) variable c2, which **bar** refers to internally as {output1}.

Functions are much more easily embedded as smaller parts of larger programs than scripts, as functions make crystal clear, via their input/output argument list, what information used in, and returned by, the called function. In complicated codes, unintentionally assigning a minor variables (like the index i or k) with different meanings in different scripts that call each other can lead to a bug that is nearly impossible to find. The proper use of functions, and the associated passing of only the relevant data back and forth (known as **handshaking**), goes a long way towards preventing such insidious bugs from appearing in your production codes.

On the other hand, short test codes, such as those provided with many of the functions developed in *RR* and *NR*, are often convenient to write as scripts, so that the variables defined by the test script may be checked (for debugging purposes) after the test script is run.

[13] That is, in addition to those variables defined as global, as mentioned above.

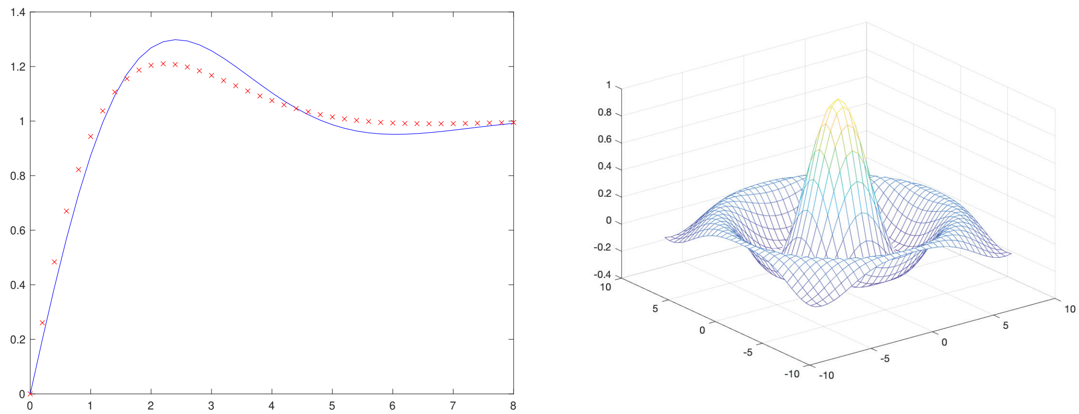


Figure A.1: Sample 2D and 3D plots: (left) the step responses 2nd-order continuous-time (CT) and discrete-time (DT) systems, as developed in *RR*, and (right) $\mathrm{sinc}(r) = \sin(r)/r$ where $r = \sqrt{x^2 + y^2}$.

A.3 Plotting

Both 2D and 3D plots are easy to generate in Matlab and Octave, as shown below:

A sample 2D plot

```
>> t=[0:.2:8];  z=[.5;  .7];
>> s=z;  o=sqrt(1-z.^2);  d=z./o;
>> y=exp(-s*t).*(-cos(o*t)+d.*sin(o*t))+1;
>> plot(t,y(1,:),'b-',t,y(2,:),'rx')
```

A sample 3D plot

```
>> [x,y]=meshgrid(-8:.5:8,-8:.5:8);
>> R=sqrt(x.^2 + y.^2)+eps;
>> Z=sin(R)./R;
>> mesh(x,y,Z)
```

The code segments listed above produce the figures shown in Figure A.1. Linear versus log axes, titles, axis labels, etc, can be controlled with **loglog**, **semilogx**, **semilogy**, **title**, **xlabel**, **ylabel**, and related commands (see the corresponding help pages); **axis**([0 5 -1.1 1.1]) zooms a 2D plot (like that above left) to the region $[0, 5]$ on the horizontal axis and $[-1.1, 1.1]$ on the vertical axis, **axis** equal sets the aspect ratio so that equal tick mark increments on the various axes are equal in size, and **axis** square makes the current axis box square. Try it!

Commands like those above produce plots in figure windows. Once a plot is as you like it, you will often want to save it, so you can email it, include it in a talk, post it on social media (you are, after all, an engineer!), make printouts of it, and/or include it in a paper or textbook you are writing. For scientific writing, LaTeX invariably produces the best results, though you may find that **what-you-see-is-what-you-get** (wysiwyg) word processors like Microsoft Word or Apple Pages are, at least initially, somewhat easier to use[14]. The recommended format to save your figures for such purposes is encapsulated postscript (denoted with a **.eps** suffix). Encapsulated postscript is a robust, platform-independent standard for both vector graphics files (representing lines as lines) and bitmaps (collections of pixels); vector graphics look especially sharp when printed out or zoomed in. All major typesetting programs, including LaTeX, Word, and Pages, can import **.eps** bfiles.

To produce a color **.eps** file, execute the **print** command after setting up the plot as you like it[15]:

```
print -vector -depsc sinc.eps
```

Once you have created an **.eps** file, you may view it with any (free) eps viewer, such as ghostscript and gv. Adobe Illustrator is a good commercial software package for making edits to **.eps** files (changing line types, font sizes, etc.), which is often necessary when preparing scientific documents. The psfrag package is especially powerful for replacing tags (characters) in **.eps** files with mathematical expressions generated by LaTeX, thus seamlessly integratings complex figures and equations into your documents.

[14] That is, until you begin to care about how well the equations are typeset, at which point your best option is to switch to LaTeX.

[15] Above a certain number of elements in a given figure, Matlab automatically switches from the (highest-quality) **-vector** renderer to the (bitmapped) **-image** renderer. Manually forcing the former, as done here, ensures the (higher-quality) vector output. Note that, prior to fall 2021, the **-vector** and **-image** options to the **print** command were called **-painters** and **-opengl**, respectively.

In Matlab, the contents of a figure window may also be saved with the command `saveas(1,'foo.fig')`, and later reopened and edited further with `open foo.fig`. The author does not typically recommend this approach, however, as it substantially limits further modification. Instead, to send a figure via email or to include it in a paper, use the **.eps** format discussed previously, and manually downgrade it to a **.jpeg**, if necessary, to reduce file size. If you want the option (you usually will!) to edit the figure later in Matlab or Octave, your flexibility is maximized by saving, in an m-file, the entire list of commands that generated the figure, thus allowing you to tweak these commands in the future, and regenerate the figure of interest from scratch.

Printouts of the text appearing in the Matlab or Octave command window after a code is run is best achieved simply by copy/pasting this text into the editor of your choosing, then printing (or generating a pdf) from there.

A.4 Source code repositories: Github and its alternatives

Source code repositories (aka repos) are like Google Docs, but for numerical codes. They:

- provide a backup of all of your most important coding work,
- sync codebases between different computers that you might use during the week,
- provide version control, allowing you to revert to a previous version of a code if a new edit breaks things,
- allow developers to fork a codebase from its mainline into a private branch for code development, and
- allow repo owners to merge new code, once debugged, from private branches back to the mainline,
- thus facilitating the simultaneous collaboration of many developers on a large set of interacting codes, while
- enabling repo owners to distribute (and, keep updated) the mainline of a big codebase to a large set of users.

Note that forking is easy, and happens at the push of a button; merging (that is, reconciling possibly conflicting code updates in different branches) is where substantial care is sometimes required. As of this writing, Github is the dominant standard for source code repositories; alternatives to Github include:

GitLab, Bitbucket, GitBucket, Sourceforge, AWS CodeCommit, and Google Cloud Source Repositories.

The many codes developed in both *Renaissance Robotics* and *Numerical Renaissance* are maintained at the Renaissance Repository (https://github.com/tbewley/RR).

You may not realize it now, but as an engineering student interested in robotics and numerical methods, you will both use large codebases and, ultimately, write lots of codes yourself (in Matlab, C, Python, Fortran, and many other languages). To become successful in this endeavor, you should thus become familiar with the proper use of source code repositories early on. Further, one of the most valuable things you can include in your resume, when looking for academic or industry jobs in the fields of robotics and numerical methods, is a link to your Github page. Even if it just has codes from various classes and small projects that you have worked on thus far, a well organized Github page demonstrates clearly to a potential employer your coding style and clarity of thought, and well showcases the major engineering skills that you have developed thus far.

After opening an account at Github[16], it is also convenient to download Github Desktop[16] onto the (Mac or Windows) computers that you plan to use to write code, which provides a convenient graphical interface to:

- clone a repo (like the Renaissance Repository) that you want to use,
- update your local clone of a repo with recent improvements from the mainline version of the repository,
- sync/merge your own daily local code developments back into your own repos online,
- fork someone else's repo into a private branch that you can work on yourself, as a developer, and, ultimately,
- submit a pull request to suggest to a repo owner that they merge your new codes into their mainline.

To get started, (a) open a Github account, (b) download Github Desktop, (c) use Github Desktop to clone the Renaissance Repository to your computer, and (d) create/use your own repos for your classes and projects.

[16]Complete instructions for using Github and Github Desktop are available at the corresponding websites. Choose your Gihub username carefully, because you will most likely want to use this account for the rest of your life!

A.5 Navigating your path

Akin to both unix/Linux/Mac shells and the Windows command prompt, the **cd** command changes directory (that is, the current Matlab working directory, for saving new files), the command **dir** lists the files in that working directory, and the command **pwd** prints the working directory to the screen. If the necessary m-files to run your code are stored in more than the current working directory, which is generally the case if you are staying well organized (please do!), the command **path** can be used to view the set of directories that Matlab will look within to find the additional m-files that it may need, and the command **addpath**(dir1,dir2), where dir1 and dir2 are strings containing complete path names (e.g., dir1='/Users/bewley/classes/MAE144', etc.), may be used to add directories to this path.

In particular, once you have cloned (using Github Desktop, as discussed in the previous section) the Renaissance Repository to your computer, you will want to add all of the directories containing your local copy of the codes it contains to your path. This is best done automatically, when firing up Matlab. A convenient script, RR_path_init.m, is provided to help with this.

To use this path initialization code, you should set up your computer to call it automatically when firing up Matlab. This may be done by appending a call to RR_path_init in the **startup**.m file of your default Matlab **userpath** directory.

In short, fire up Matlab, and type the command **userpath**, which will return the name of the default directory that Matlab starts up in on your machine. Within that directory (important!), edit the file **startup**.m (or, create a new file of this name if one doesn't already exist); this file should contain, at least, the following lines:

```
RRbase='/Users/bewley/RR'; cd(RRbase); RR_path_init
```

Note: replace the directory name in single quotes above with the full path to the location that you have installed the Renaissance Repository on your computer. Note that forward slashes, /, as shown above, are used on Macs, whereas backslashes, \, are used in Windows; on a Windows machine, the full path to this directory might look something like C:\Users\bewley\RR. Note you can also put other commonly needed Matlab initialization commands in your **startup**.m file; in particular, you will probably want to add the paths to your personal Matlab project directories that you use often (see the last sentence of the first paragraph of this section), and perhaps also (at the end of the **startup**.m file) a **cd** to the directory that you plan to be working in most in the foreseeable future. You may also include a host of other actions, such as calling a routine like RR_physical_constants, to define some of the physical constants that you might often need in your line of work.

Many other getting-started functions are available in the Renaissance Repository. Rather than listing such functions here, the reader is asked to fire up Matlab and, e.g., **cd**(strcat(RRbase,'Renaissance_Robotics/chapAA')) in the command window at the >> prompt, then **dir**, then **type** RR_double_factorial, **type** RR_swap, **type** RR_permute, etc, for each command that you find. By studying such sample codes, and running the test commands embedded in their initial comments, I reckon you'll digest them quickly, and find your own way forward from there.

A.6 Advanced prepackaged numerical routines

Matlab and Octave have a ton of very useful advanced prepackaged numerical routines built in, including **inv**, **lu**, **qr**, **cond**, **eig**, **schur**, **svd**, **norm**, **rank**, **pinv**, **tf**, **minreal**, **bode**, **rlocus**, **impulse**, **step**, **c2d**, **lyap**, **dlyap**, **icare**, **idare**, etc. As you progress through the *RR* and *NR* texts, these routines will quickly become useful for you, as you learn *what* they compute. However, the purpose of the *RR* and *NR* texts is actually not simply to catalog these prepackaged routines, but rather to flush out *when* and *where* you might use them, *why* they are the tools of choice for certain problems, and *how* they actually work. With this knowledge, the reader will be able to select and use such routines with much greater understanding and forethought. We thus avoid using almost all such prepackaged routines in these texts, opting instead to rewrite many of them from scratch.