1
Origins

1.1 Phase Retrieval

Like many new lines of research, the subject of this book was not the product of a systematic, well-argued plan of investigation. It came about as a complete accident.

In May 2001, a workshop on "New approaches to the phase problem for nonperiodic objects" [103] was held at the Lawrence Berkeley National Laboratory. I was invited for the flimsiest of reasons, in that I had some background in very famous nonperiodic objects called quasicrystals, discovered by Dan Shechtman in 1982.

For some background, we first turn to a 1986 meeting of the American Crystallographic Association (ACA), held at McMaster University. A representative sample of the quasicrystal data [97] that was hotly debated at the meeting is shown in Figure 1.1. The bright spots are a record of electrons (produced by an electron microscope) reflected into special directions by planes of atoms in the quasicrystal. Many scientists, including Linus Pauling, were bothered by the fact that the angles of the planes could not be reconciled, mathematically, with a material that was periodic [93]. My ACA presentation, where I argued that the spots could be explained by a generalization of periodicity that has six axes instead of the usual three (but still in three dimensions) [34], was overshadowed by Pauling's, in which he claimed the precision of the spot measurements was insufficient to rule out a periodic arrangement, albeit with a very large repeat distance [88]. In any case, the six-axis generalization of periodicity had nothing to do with the "nonperiodic objects" of the Berkeley workshop 15 years later. The objects of interest there had no periodicity whatsoever.

There was a recognition event at the 1986 ACA meeting that proved to be remarkably prescient. In the previous year, the crystallographers Herbert

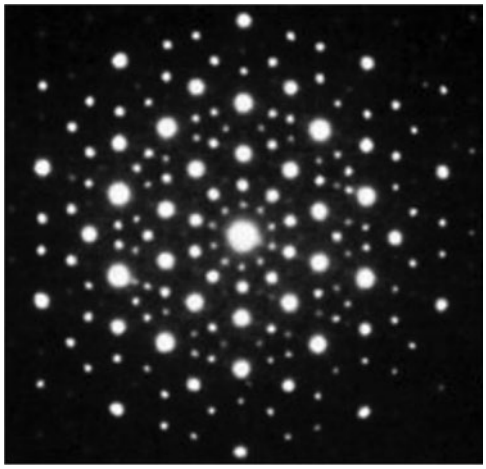

Figure 1.1 An early electron diffraction image of the AlMn quasicrystal [97]. The 10-fold arrangement of the spots challenged the orthodox model of crystalline order. Reprinted from Shechtman et al. (1984) with permission from © 1984 American Physical Society.

Hauptman and Jerome Karle had been awarded the Nobel Prize in Chemistry for having solved "the phase problem." The substance of this accomplishment can be explained even for the data shown in Figure 1.1, which after all only represents a break from orthodox periodicity.

From the physics of the electron reflection process, called diffraction, one can show that the spot intensities are directly interpretable as the coefficients of a Fourier series. By summing the series, called Fourier synthesis, one can reconstruct the microscopic "sources" of the electron reflection, a function in three dimensions commonly referred to as *contrast*. Hauptman and Karle were working with X-ray data where the contrast is the electron density in the material, while it is the electrostatic potential produced by those electrons (and nuclei as well) that the electron microscope data reveals. In either case, by doing the Fourier synthesis, one can recover not only the geometry of the reflecting planes, but the makeup of those planes by atoms.

But the Fourier synthesis dream runs up against a major obstacle. The Fourier coefficients are complex numbers, and the spot intensities only provide the magnitudes of those numbers. Fourier synthesis cannot be attempted without the phase angles of those coefficients, which go unmeasured. This is the phase problem.

Early (pre-1950) crystallographers were undeterred by the phase problem. Progress was made through *modeling* the contrast (arrangement of atoms), as in the iconic photo of Crick and Watson posing with their model of the

double helix. Using strong constraints coming from chemistry, only a limited number of atomic arrangements had to be considered, and getting just the magnitudes of the Fourier coefficients to work out was usually enough to nail down the structure. At the time of the 1986 ACA meeting, the modeling approach was the leading method being applied to determine the atomic structures of quasicrystals.

Hauptman and Karle [60] are credited with solving the phase problem because they found a method that avoids modeling, making atomic structure discovery available to even chemically challenged crystallographers. Their method comprises both a principle and an algorithm. The principle is surprisingly simple. Consider a thought experiment where *random* phases are applied to the Fourier coefficients. In all except a fantastically lucky combination of phases, the resulting contrast (by Fourier synthesis) is a complete mess. Chemistry does not come up because the contrast does not even resemble atoms!

The simplicity of the principle behind the phase problem solution – conjuring up phases to get something that at least resembles atoms – is in sharp contrast with the complexity of exploiting that principle. The unknown phases of the Fourier coefficients are like the dials of an enormous combination lock. Only one combination produces a sensible contrast and unlocks the atomic structure. But finding that combination is hard when there are many dials/phases. The phase problem cannot be declared solved unless there is also a practical algorithm that can unlock the phases encoded in the Fourier magnitudes.

In 1986, I did not think of the phase problem in these terms. The code-breaking angle might have inspired me to dive deeper into the subject, but at the time my interests were still aligned with physics. The secret of quasicrystals, like the secret of life, had to be simple and elegant and not involve difficult computations! Still, I recall being in awe walking through the poster gallery at the ACA meeting, each one chronicling a phase problem success on a complex biomolecule. Unlike the simplicity I was counting on to make the quasicrystal phases amenable to modeling, the giant virus structures on display all owed their existence to an algorithm.

The 2001 Berkeley workshop was organized by the electron microscopist John Spence and was meant to launch a revolution in microscopy. Spence had invited a diverse group of X-ray scientists, electron microscopists, engineers, mathematicians, and physicists, some of whom (myself included) knew almost nothing about the phase problem or any form of cutting-edge microscopy. Those in the latter group were quickly brought up to speed, most notably with regard to the "abcdef" experiment [76].

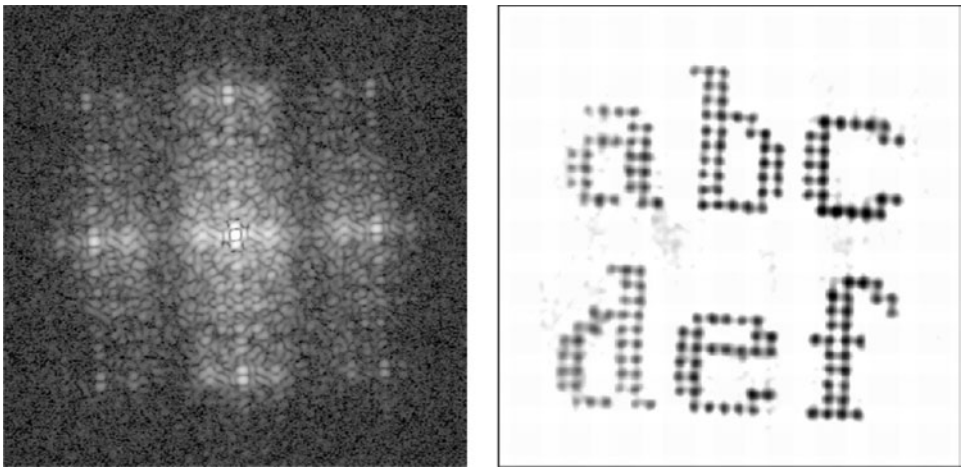

Figure 1.2 First demonstration of lensless microscopy with X-rays, by Miao et al. [76]. The diffraction intensities on the left were given to an algorithm that reconstructed the contrast shown on the right. Reprinted from Miao et al. (2000), with permission from Copyright © 2000, AIP Publishing.

In 1999, a group at SUNY Stony Brook published a proof-of-concept paper [76] demonstrating the algorithmic reconstruction of a nonperiodic contrast from just the magnitudes of its Fourier transform. The data, shown on the left in Figure 1.2, were obtained by shining a coherent (laser-like) beam of X-rays on a fabricated specimen of gold dots deposited on a thin membrane. The image on the right is their reconstruction but is practically indistinguishable from the ground truth, which was obtained using the higher resolving power of an electron microscope. In any case, no conventional, lens-based microscope using light (even X-ray light) was capable of doing what these researchers had managed to do with the help of an algorithm.

The intensity of the diffracted X-rays in the "abcdef" experiment was not concentrated in spots, as in the quasicrystal data. Instead of a Fourier series with unknown phases for the coefficients, in this data the measured and continuously distributed Fourier magnitude is accompanied by an unknown and similarly continuous phase *function*. Later I learned that this feature made the phase problem much easier than it was for periodic contrasts. At the workshop I also learned to think more optimistically about the phase problem, because the participants never used the word "problem" and instead chose to call the algorithmic task "phase retrieval."

Probably the most far-reaching implication of algorithmic phase retrieval of nonperiodic objects I learned about at the workshop was the possibility of

reconstructing the shapes of complex biomolecules (proteins, viruses, etc.) without having to coax them into forming crystals first [81]. Most biomolecules are terrible at forming crystals, but structural biologists persist because only a crystalline specimen, comprising 10^{16} identical copies of the same object, can produce measurable diffraction data. A star-wars caliber X-ray source, with the power to form a diffraction image of a single biomolecule, would eliminate the need to grow crystals. That was in part the rationale behind the Linac Coherent Light Source, an X-ray laser under construction at SLAC. If this feat of engineering succeeded, determining the shapes of the molecules of life might become as easy as "abcdef."

My engagement at the workshop ratcheted up by several levels when the talks turned to the algorithms for phase retrieval. One algorithm stood out: Jim Fienup's *hybrid input-output* (HIO) algorithm [42]. HIO was wildly successful but also mysterious. It belonged to the class of algorithms that acted iteratively and deterministically on estimates of the contrast. Starting with an initial guess, which typically was random, a sequence of transformations was applied until the contrast stopped changing, hopefully after it had settled on the true contrast. My general impression was that the transformations would alternate between modifying the contrast and modifying its Fourier transform, of which only the phase function was free to change because the magnitude was fixed by the data. As far as I could tell, the transformations were doing reasonable things: Fixing what was obviously wrong in the contrast or restoring the known magnitude to the Fourier transform. In this respect HIO was not exceptional. The secret of its success seemed to lie in the precise way the transformations were combined.

A more complete theoretical understanding of HIO was clearly a worthwhile goal. Only HIO had been able to reconstruct "abcdef" in the final and computational stage of the new microscopy. If anyone understood HIO at the time of the workshop, it could only have been Jim Fienup. For the rest of us, and here I mostly refer to myself and three mathematicians [10] at the workshop, his "hybrid" construction looked ad hoc. It was a puzzle why the algorithm worked at all.

When I returned to Ithaca, I set myself the challenge of understanding HIO. This book, written 22 years later, is the outcome of that investigation. However, already within months of the workshop, the three mathematicians Bauschke, Combettes, and Luke [10], came up with a formal and more general statement of what HIO was doing that matched what I had come up with [35]. And as often happens in mathematics, Bauschke and coworkers realized the HIO "formula" had been written down nearly half a century earlier, by

mathematicians Douglas and Rachford [30], in a completely different context: numerical methods for solving partial differential equations.

1.2 Nonconvexity

The context of the Douglas–Rachford formula plays a fundamental role in the nature of the algorithm. In symbols the formula reads

$$x \mapsto x + P_B(R_A(x)) - P_A(x)\ . \tag{1.1}$$

Here x is the thing being reconstructed, like the "abcdef" contrast or the solution of a partial differential equation. The building blocks of the algorithm are called *projections*, and P_A "projects to set A," while P_B "projects to set B." The third operation is called a *reflection*, and

$$R_A(x) = 2P_A(x) - x$$

"reflects in set A." We do not need to get into the motivation and roles for these operations here – that is the purpose of this book – to make a point about the Douglas–Rachford formula. This is that the formula makes sense for arbitrary sets A and B even though the context of the formula is normally limited to the case of *convex* sets. A set is convex if all the points between any pair of points in the set are also in the set. We will see that the "convexity context" of formula (1.1) plays an oversized role in the history of the subject.

When the Douglas–Rachford formula is used with convex sets, it is as though the algorithm never has to make decisions. This comes about because $P_A(x)$, a point in A that is closest to x, is always unique when A is convex. A simple example shows how this changes when A is nonconvex. Suppose x is just a single unknown (an image of one pixel) and A is the set of two elements $\{-1, +1\}$. Now if $x < 0$ then $P_A(x) = -1$, while $x > 0$ implies $P_A(x) = +1$. In both cases, the result of the projection is unique. However, when $x = 0$, the algorithm has to make a decision because both -1 and $+1$ have the same distance to $x = 0$. If $x = 0$ never comes up in the course of the Douglas–Rachford iterations, then one might argue that the algorithm is still not having to make decisions. But this is a very naive view if it turns out that the x values the algorithm encounters have a nonzero density near $x = 0$. In that scenario, some fraction of the $P_A(x)$ values will depend very sensitively on x. Though still formally deterministic (no actual decisions), the algorithm is better characterized as sometimes making *random* decisions. Thinking of the Douglas–Rachford formula as defining a dynamical system, nonconvexity of the sets A and B is a potential source of *chaos*.

1.3 Chaos and Ergodicity

The analysis of algorithms that have the potential of behaving chaotically is hopeless. That is the reason the Douglas–Rachford formula has traditionally been used only when the sets A and B are convex. With this restriction, and when a solution exists, one can prove the iterations converge to a fixed point and a solution to the problem is at hand [4]. But a simpler and more obvious algorithm, studied by John von Neumann and called alternating projections [8],

$$x \mapsto P_B(P_A(x)),$$

also has these good properties when the sets are convex. Fienup tried this in his phase retrieval experiments and rejected it because it always got stuck on nonsolutions. One of the sets in phase retrieval, the constraint on the Fourier magnitudes, is nonconvex.

After I had managed to express HIO as the general formula (1.1) and was in a position to analyze the algorithm geometrically, it seemed to me what was special about this algorithm was its ability to extricate itself from the traps that plagued alternating projections. Because there are many traps in a hard problem, the resulting dynamics will be very chaotic. Biased by my physics background, I liked the idea of exploiting chaos. Understandably, this perspective was not shared by the mathematics community. Already at the Berkeley workshop, a proposal was made to move HIO into safe territory by "convexifying" the Fourier magnitude constraint. Fienup obliged, changed one line in his code, and probably had already anticipated the result: HIO no longer worked.

In physics, it is fair to study things phenomenologically: make observations, form hypotheses, perform experiments, and so on. This applies even when the subject has been reduced to mathematical formulas. Some formulas are just too hard to yield to mathematical analysis! Over the past two decades, I have taken the phenomenological approach in studies of HIO/Douglas–Rachford. An example of this kind of study, singled out for its kinship to phase retrieval, is the problem of *bit retrieval* [37]. My first experiments were performed at Simon Fraser University, in a sabbatical hosted by Jon Borwein, an early champion of experimental mathematics.

In bit retrieval, one tries to reconstruct the contrast of a one-dimensional crystal from the magnitudes of its Fourier series coefficients (diffraction data). As the name suggests, "retrieval" is possible because the contrast is known to have only two values, say -1 and $+1$, at each pixel. In the same spirit as phase retrieval for physical crystals, where Fourier synthesis with the

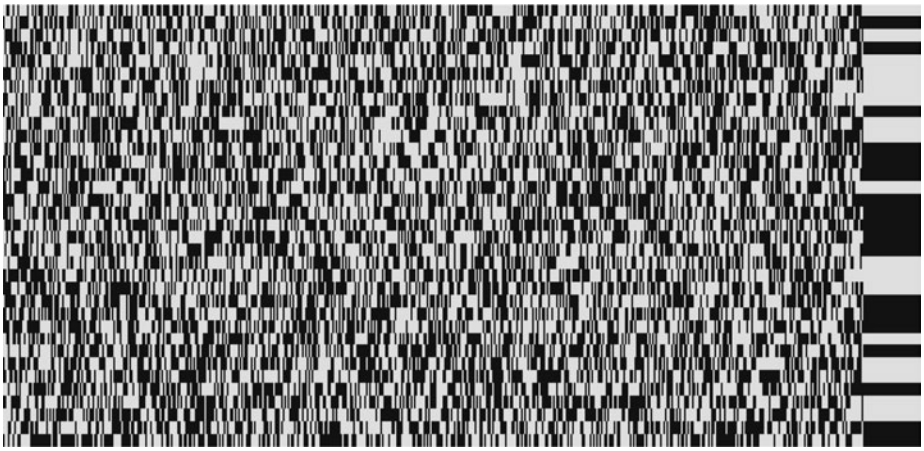

Figure 1.3 Chaotic dynamics in bit retrieval [37], the reconstruction of a two-valued sequence from its Fourier magnitudes. Each row shows the time evolution of one bit in the sequence. After about 34 000 time steps (Douglas–Rachford iterations), the random initial pattern on the left arrives at the solution fixed point on the right.

wrong phases produces a nonatomic mess, the wrong phases in bit retrieval produce a contrast where the pixels deviate from the two special values.

The time evolution of the contrast in bit retrieval, by the Douglas–Rachford algorithm, can be rendered in the same way that is popular for one-dimensional cellular automata. Figure 1.3 shows the retrieval, left to right, of a sequence of 35 bits starting from a random contrast. The juxtaposition of chaos with stability, at the fixed point on the right, is striking.

It is important that formula (1.1) is able to recognize and converge on solutions when it finds them. The local convergence of Douglas–Rachford, near solutions where the sets A and B may be approximated as convex, is responsible for that. But just as important is the behavior prior to arrival at the fixed point, where the dynamics can only be characterized as chaotic. The two contrast values (± 1) are being sampled much like we think of the positions of molecules in a fluid. All configurations in the fluid, subject only to the conservation of energy, arise eventually if one is patient and waits long enough. Thanks to chaos introduced by the projections (when a contrast value is close to zero), the sampling of candidate contrasts in bit retrieval is similarly exhaustive.

The bit retrieval experiments also showed that the dynamics of formula (1.1) was very good at doing something else. Though there is no analog

of energy, the stream of contrasts being generated were all *near*-solutions: Their Fourier magnitudes were close to the known values, and the contrast values themselves were close to either -1 or $+1$. Empirically I knew a very targeted search was taking place because solutions were being found with far, far fewer iterations than the $2^{35} \approx 3 \times 10^{10}$ possible bit sequences!

Might chaotic search be a competitive alternative to the systematic approach to search we are taught in computer science? The time spent in a successful chaotic search has an analog in physics called the *Poincaré recurrence time*. Technically this is the time needed by a finite system of particles to return to its initial configuration within some specified precision. Because water molecule configurations are not special in any obvious way, we expect all of them to be visited over the course of one recurrence time. If Kurt Vonnegut's ice-nine [111] did in fact exist as a (catastrophically) stable solid configuration of water molecules, it would be found within about one recurrence time! Fortunately, the recurrence time for even microscopic physical systems is astronomical, and the possibility of an ice-nine fixed point – should one exist – poses no danger to humankind!

The recurrence time of chaotic Douglas–Rachford dynamics is often comparable and sometimes much shorter than the time taken by a systematic branching search. Ice-nines are found routinely. The most detailed study is on the bit retrieval problem and includes estimates of the run/recurrence time and how this depends on a quantifiable hardness measure. The 35-bit instance shown in Figure 1.3 is the hardest for that size because all of its Fourier magnitudes are equal [37].[1] In 2002, an easier, average-case instance with 160 bits was solved in one week. This may still be the record for consummated Poincaré recurrence times.

There is really just one obstacle for chaotic search to be more widely adopted, and it is not the inherent randomness in the run time. Some initial points are luckier than others because the dynamics stumbles on a solution fixed point earlier. The distribution of run times is accurately described by the curve for radioactive decay, consistent with a loss of memory of the past (due to chaos) and there being a time-independent exposure to fixed points. But systematic branching searches are just as unpredictable, with some decision orderings luckier than others.

The unpredictability of the time required to arrive at a fixed point is not so much a concern as is the possibility that the dynamics simply fails to explore a significant fraction of the solution candidates. In physics, the property of visiting all configurations (subject to energy conservation) is called *ergodicity*. It is easy to see how sensitivity to initial conditions, as a

result of chaotic dynamics, helps ergodicity. But that still falls short of a guarantee.

Fortunately, there is an important use-case that makes me optimistic: thermodynamics. Boltzmann faced a very similar challenge when he tried to derive the macroscopic equilibrium properties of gases from the complex dynamics of the constituent particles. The assumption that all configurations consistent with energy conservation were visited with exactly the same frequency gave the correct answers but seemed impossible to prove. Because the phenomenological model – thermodynamics – worked so well, Boltzmann formalized his assumption as the *ergodic hypothesis*. Today no one doubts the truth of the ergodic hypothesis for systems such as gases and fluids, even though the only established results are for simple "billiard" models [101].

The ergodic hypothesis is false for some physical systems, a fact we should be thankful for when contemplating our place in the solar system (and its repetitive and predictable dynamics). By analogy, we should not expect all applications of formula (1.1) to be as ergodic as bit retrieval. Over the years, I've encountered applications where instead of arriving at fixed points, whose existence is not in doubt, the dynamics often gets trapped in the analog of a solar system. Sometimes this can be resolved by defining the sets A and B differently or modifying the metric used to define the distance. In any case, the existence of bad actors is not relevant for the many applications in which ergodicity appears to be upheld.

1.4 Convergence and Logic

When I first started giving talks about (scandalously) applying Douglas–Rachford (DR) to nonconvex problems, I would invariably be asked, "... but does it converge?" I was completely comfortable knowing that DR was doing a very good job at something just as important – a thorough search – and that the mathematically provable convergence that happened in the last few iterations was simply what terminated the search. Though I could point to my experiments on bit retrieval, as a demonstration of the new kind of convergence, I got the sense that the people asking about convergence were looking for something more. Around 2005 a way to satisfy even these skeptics came from an unexpected source: sudoku.

Solving a problem by *logic* is usually construed as the process of increasing knowledge by the application of inference. Knowledge about the problem can only grow, culminating when the culprit is confidently unmasked (Colonel Mustard). Sudoku is a minimalist exercise in inference, something that surely is part of its appeal. Players learn (or discover on their own) a set of inference

rules whose application increases the set of filled-in numbers. A popular rule is to look for two instances of the same number, say 5, in two 3×3 blocks that fall on a shared set of three rows or columns. The unknown location of a third 5 is now constrained not just by a block, but also by the row or column not already used by the two other 5's. Beginners and experts differ mostly in the number of inference rules they have amassed and the complexity of applying them. The size and complexity of the rule toolbox needed to solve a puzzle may also be the basis of the mysterious difficulty ratings.

Computer programs for solving sudoku fall into two groups. Either the program goes through a list of human-engineered inference rules (naked-quad, X-wing, ...), or more laboriously, systematically tries out all ways of completing the puzzle that are not in direct violation of the rules (with the promise that one of the many hypotheses is bound to work out). But these are really just extremes on a spectrum, the elaborate inference rules being just a highly creative alternative to "brute" search. The DR algorithm, if it could be applied to sudoku, would be doing something else entirely.

The first challenge in solving sudoku with DR was formulating the puzzle in terms of two easy constraints, A and B. This turned out to be not very hard at all[2]:

A **:** In each of the 3×3 blocks, numbers and cells are paired one-to-one.

B **:** Each number appears nine times, pairing rows with columns, one-to-one.

Filling in numbers consistent with just A or just B is easy, and provided the projections (nearest patterns) P_A and P_B are also easy to compute, the DR algorithm will do the rest. Though projections aren't covered until Chapter 4, the information in Figures 1.4 and 1.5 will help you understand what DR is doing. Both show the algorithm solving, in six iterations, a New York Times puzzle with difficulty rating "easy."

Figure 1.4 shows just the progress of the projection $p_A = P_A(x)$, so not the actual point x being updated. All the 3×3 blocks in each iteration are in compliance with rule A. The "given" numbers or "clues" are marked by the gray cells and never change. Figure 1.5 shows the progress of the projection $p_B = P_B(R_A(x))$. Think of these as nine overlapping number patterns, all having the same kind of one-to-one pairing of rows with columns as shown highlighted for the number 5. After six iterations (bottom-right panel) the two number patterns, p_A and p_B, are the same and the puzzle is solved.

The constraints A and B, as geometrical sets, are as nonconvex as possible: sets of isolated points. In spite of this, there is a kind of convergence not so different from the mathematically rigorous convergence when both sets are

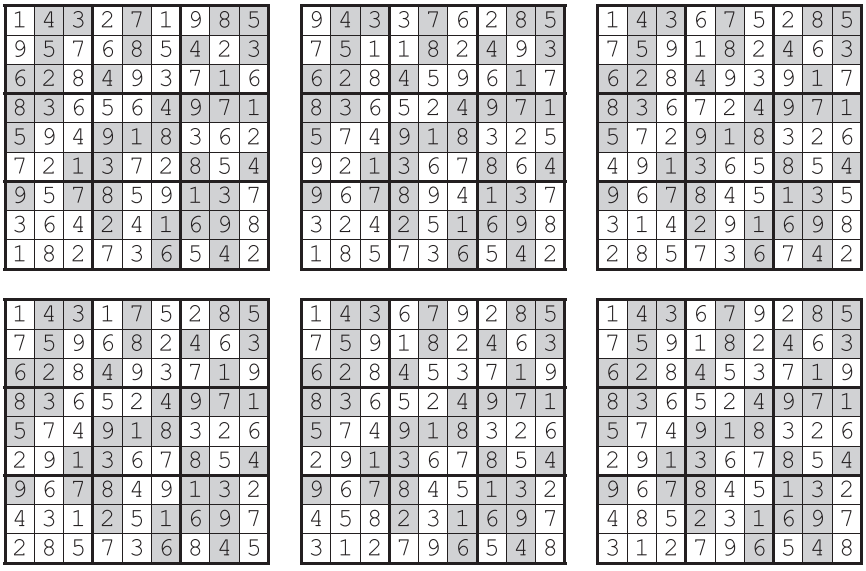

Figure 1.4 A sudoku puzzle being solved in six Douglas–Rachford iterations (top-left to bottom-right). These do not show the updating of the point x, just consistency of the projection $P_A(x)$ with sudoku rule A. The gray cells hold the given numbers (and never change).

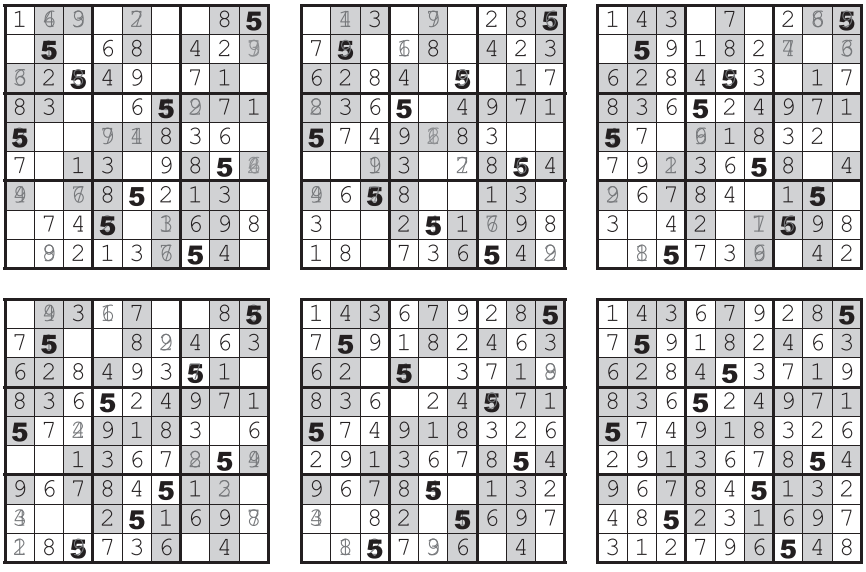

Figure 1.5 Same as Figure 1.4 but for rule B. All numbers have the relationship displayed by the nine highlighted 5's, a one-to-one pairing of rows with columns.

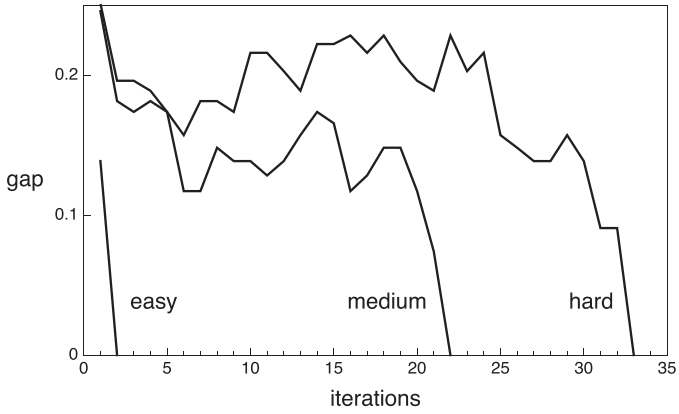

Figure 1.6 Typical behavior of the gap (distance between constraints A and B) when solving the three difficulty levels of New York Times sudoku puzzle.

convex. To convey this, we consider the *gap*, or the current distance between the points p_A and p_B. When the gap is zero, the two points are the same point, and this point is a solution to the puzzle (because both constraints are satisfied). The generalization of convergence in this very nonconvex application of DR is the behavior of the gap.

Figure 1.6 shows plots of the gap for three New York Times sudoku puzzles. In these runs, the clue data were implemented differently,[3] with the result that "easy" puzzles are always solved in just two iterations. In "medium" puzzles, it's harder to argue something analogous to inference is going on, because the gap is not always decreasing. A mostly decreasing gap for "hard" puzzles does not begin until after the algorithm has done some exploring (here about 20 iterations). This progression aligns with our expectations, but is also necessary if a widely believed technical property of sudoku is true.

In the study of computational complexity, sudoku belongs to a class of problems where (it's believed) long exhaustive searches can never be completely avoided. To realize the extremes of hardness, puzzles must be "diabolically" designed or generalized to 16×16 grids and higher. Puzzles for human consumption are neither so large nor diabolical to obviate a logical route to the solution. Interestingly, these puzzles also have the property that the Douglas–Rachford gap is mostly decreasing.

The sudoku episode of the nonconvex Douglas–Rachford story was important in three respects. First, it attracted far more interest than any accomplishment in phase retrieval could have achieved! Second, and strictly as

sudoku difficulty	gentle	moderate	tough	diabolical	extreme
average DR solution-time	3.8	6.8	9.3	12.9	15.6

Table 1.1 Growth in solution time with a logic-based difficulty scale [105].

empirical evidence, I could now point to the behavior of the gap as a generalization of convergence for even highly nonconvex problems. Third, and for me the most inspiring, was the discovery that DR seemed to be capable of generating its own brand of logic. Though my sudoku solver was built using just the elementary rules of the puzzle, upon execution, its progress toward the solution was apparently as systematic (a shrinking gap) as a solution by standard logic. And if the solution process for easy puzzles/problems had the hallmarks of logic, then DR was likely also doing something clever on the harder counterparts. A model to explain the cleverness is proposed in Section 5.5.6 where we study the "flow" of the algorithm and define the *solution time*, closely related to the number of iterations. Table 1.1 lists solution times for five classes of puzzles graded by the depth of logic required to solve them. These results were obtained by sampling Andrew Stuart's puzzle collection [105]. All puzzles require some amount of logic, but the number and complexity of inference rules needed to solve them grows substantially between "gentle" and "extreme."

1.5 Deconstruction

So far this narrative has betrayed my background as a physicist. Though a computer scientist may not view search as a dynamical system, when subject to chaos and limited by ergodicity, this perspective is certainly one that most physicists are comfortable with. On the other hand, one of the main objectives of this book would not be met if physicists were not also taken into unfamiliar territory.

Search algorithms are often characterized as "physical" if they mimic the dynamics of a physical system. I will give two examples: disk packing and neural network training.

Figure 1.7 shows a set of 14 disks being packed by an algorithm that simulates the process of compression. The enclosure, a circle, starts out large and shrinks opportunistically whenever allowed by the positions of the disks. In between compressions the disks make small random steps called Brownian motion. The idea is that through Brownian motion the disks can rearrange into configurations with more wiggle room and a more efficient use of space.

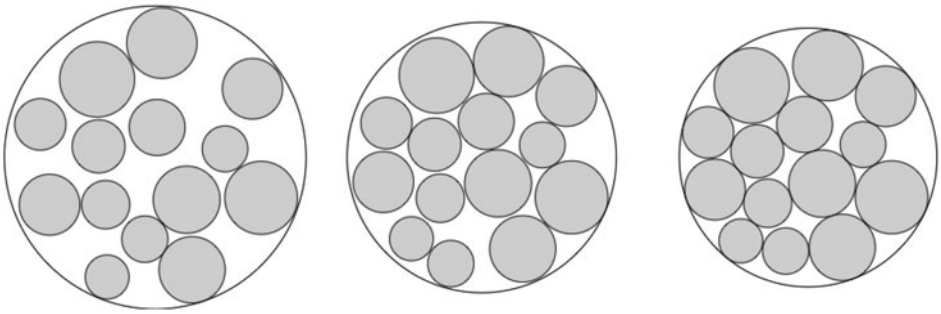

Figure 1.7 The compression of a system of 14 disks by a shrinking circular enclosure comes to a halt (frame on the right) when the disks become jammed.

Eventually there is no wiggle room left. When this happens, one says the configuration of disks is "jammed"; no further compression is possible.

The second example concerns the application that is pervasive in nearly all of modern technology: artificial neural networks. The core algorithm for the training of networks was set forth in "Learning internal representations by error propagation" by Rumelhart, Hinton, and Williams[4] (RHW) [91, 92] whose citation count increased even as you were reading this sentence. This algorithm is physics based in the sense that the discrepancy between the network's target output and its actual output is quantified by an energy function ("loss"), and the network's parameters ("weights") are adjusted to decrease the discrepancy/loss in much the same way a ball rolls down a hilly landscape so as to reduce its gravitational energy.

Though most of the RHW citations are for the back-propagation formula,[5] this paper also features several simple and revealing applications. The network for one of these, shown in Figure 1.8, is called an autoencoder. Here the network is tasked with exactly reproducing n input patterns in its outputs, both expressed on n nodes. What makes this challenging is a bottleneck of only $\log_2 n$ nodes that the information is forced to pass through. By choice of the nonlinear functions that generate the values on the bottleneck nodes, which smoothly interpolate between 0 and 1, the autoencoder is biased toward transmitting the information through the bottleneck as a binary code. Though the authors were thrilled that their error-propagation-trained autoencoder worked (outputs exactly matched inputs), they had to concede that often the codes appearing in the bottleneck were not strictly binary, because they included "intermediate values." It was a victory for self-taught data

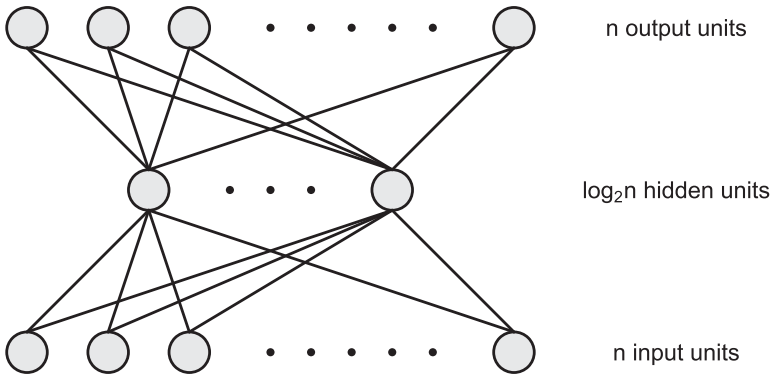

Figure 1.8 Autoencoder network from "Learning internal representations by error propagation" [91] used in the study of the binary encoding problem.

compression, but a mixed result for interpretability of the data representation.

Figure 1.9 shows the distribution of code values for the $n = 16$ autoencoder, using the same "sigmoid" activation functions used by RHW, but with a state-of-the-art training algorithm [62] that uses "momentum" in addition to energy to guide its course through the loss/energy landscape. Though the endpoint of training, a point with zero loss, is found faster than the method used by RHW, the results are no better: Most of the codes are nonbinary because of an admixture of intermediate values.

Were I to remain true to my physics training, I would view the jammed, nonoptimally packed disk configuration, and the not-quite-binary autoencoder as inevitable outcomes when systems cross some threshold of complexity. There is even a subdiscipline of physics, for disordered and glassy

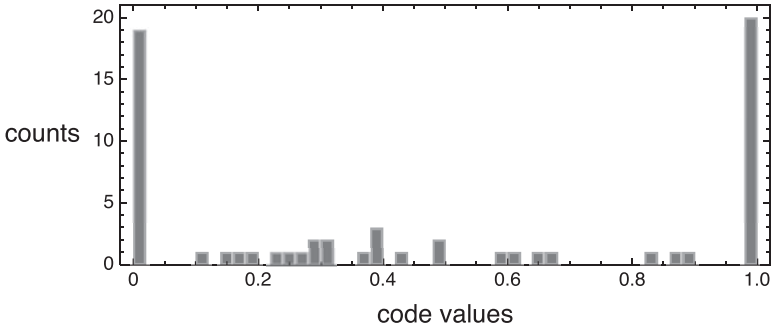

Figure 1.9 Distribution of the 16×4 code values found by the gradient-descent-trained autoencoder.

systems, where both of these applications have been welcomed with open arms [1, 59]. The point of this book, however, is to offer you an alternative.

The alternative is not a more powerful, physics-inspired algorithm. It goes deeper than that. A good name for the main idea is *deconstruction*. Before taking on a new problem, the first step is to deconstruct it into sets A and B. Phase retrieval was important in the development of the subject simply because A and B were fairly obvious. For most other problems, deconstruction into A and B is a new skill.

Let's deconstruct the disk packing problem. My former student Simon Gravel came up with a great name for this style of deconstruction: *divide and concur* [48]. Each of the 14 disks to be packed is "divided" into 13 versions, called "replicas." For example, disk 3-7 is the replica of disk 3 that "watches out for disk 7," while disk 7-3 is the replica of disk 7 that "watches out for disk 3." Set A corresponds to all such replica pairs satisfying the packing constraint (not intersecting). This is a very easy constraint to satisfy because the constraint on each replica pair is independent of the constraints on all the other replica pairs. If replicas 3-7 and 7-3 do not intersect, nothing needs to be done. Otherwise, to project to set A, the replicas are moved the minimum distance, which amounts to making them tangent. Set B implements "concur," or the constraint that replicas 3-1, 3-2, 3-4, ... , 3-14 are actually the same disk! The projection here is to replace their centers by the mean of their centers.

The packing problem can be deconstructed even further in a way that takes advantage of the nonuniform disk sizes. Replica 3-7 not only has a center (to be determined), but is given a variable radius. To keep replicas 3-7 and 7-3 from intersecting, the projection may shrink their radii in addition to moving their centers. Whichever combination of those actions involves the smallest sum-of-squares change is the one taken by P_A. The variable radii are addressed in concur, or P_B. After all the replicas of disk 1, disk 2, and so on are given a shared radius by averaging, the resulting set of 14 radii are minimally modified to match the radii of the original packing problem.

This is quite an elaborate deconstruction, but very much worth the effort. Three configurations encountered in the Douglas–Rachford (DR) dynamics are shown in Figure 1.10. An animation would show disks moving according to some strange laws of physics that allowed intersections and penetration of the enclosure. Even stranger, disks of different sizes occasionally appear to swap positions by teleportation (when they swap radii). But the exotic dynamics are not at all exotic in the deconstructed world. The figure shows only the dynamics of the 14 concurred centers and radii and cannot convey what all the replicas are doing!

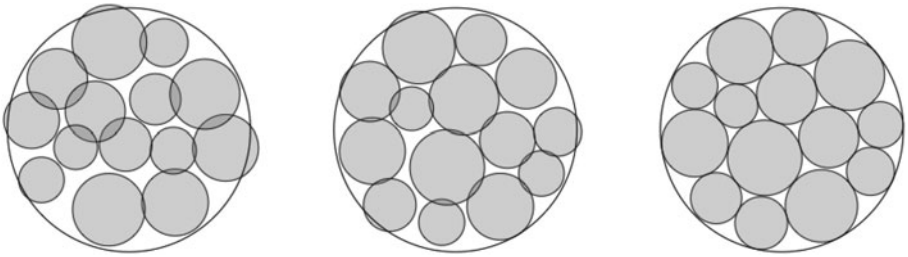

Figure 1.10 Douglas–Rachford iterations acting on the constraints A and B of the deconstructed packing problem (with a fixed circular enclosure) terminate when a true packing is found (right frame).

The 14 disk-radii in this packing puzzle were designed so there is a unique solution for the given enclosure.[6] The DR algorithm finds this solution in about the same time taken by the compression algorithm to find one of the many jammed nonsolutions (enclosures larger than optimal by several percent).

Let's now turn to the deconstruction of the neural network training problem. In constraint A each neuron is treated in isolation, independent of all the other neurons in the network. Every neuron has a vector of inputs, x, a single output, y, and a vector of weight parameters w. In the binary encoding/decoding network, or any other network where we want the neurons to learn dichotomies, the outputs may have only two values, say $y = 1$ or $y = -1$. The choice of output is determined from the value of the dot product $w \cdot x$. Projecting to the "neuron constraint" is actually quite easy and involves no transcendental functions (as in the sigmoid step-like activation function). At the end of projection P_A, all the neuron outputs are either 1 or -1, exactly. P_A also minimally modifies the inputs and weights for each neuron. Interestingly, x and w experience the same degree of change because they appear symmetrically in the constraint through their dot product. If you are impatient, the details of the neuron projection are spelled out in Section 7.7. Finally, the network's input nodes are special and for them P_A simply sets the y variables to the data values.

The connectivity of the network and the constraint coming from the output-side of the data is addressed in constraint B. Each neuron's output y should concur with the inputs x of the receiving neurons. A key part of the deconstruction is to realize that the input variables live on the edges of the network, not the nodes. That's because one neuron's output y gets sent to

multiple neurons, and the replicas of that y, called x, align with the edges that connect the output node with the various input nodes. The complete concur projection therefore involves, at each neuron's output, a single y and the x values on all the edges from that output. Like the neuron constraint projection, P_B is very simple (replacing numbers by their average) and a local computation because independent sets of variables are involved. At the output nodes, where there are no outgoing x variables, P_B simply sets the y variables to the data values.

Neural networks are trained on batches of data and this continues to be the case in the deconstructed problem. In the $n = 16$ autoencoder, there are only 16 data and it is feasible to have the batches be the entire data. This is how the gradient-descent results shown in Figure 1.9 were obtained. In the deconstructed problem, the entire network of x, y, and w variables is replicated 16 times. For the most part, the variables in the different network replicas are independent, and each receives a different set of data that constrain the input and output nodes. But P_B now has an additional task: The weights w should concur across all the network replicas.

If this deconstruction also strikes you as elaborate, it just means you can use some training in the craft of deconstruction. Chapter 2 is a warm-up with simple examples; the more elaborate examples in Chapter 7 convey the scope of deconstruction. But already now you can see that the deconstructed neural-network-training problem is quite nice: many simple and local constraints that are easily satisfied (projected to). Most of the programming work is in the implementation of the two projections, and is comparable in complexity to back-propagation code. A single DR iteration, comprising one P_A and one P_B and little else, involves about as much computation as a single gradient step.

How can the progress of training be monitored when there is no loss function? In this and any application of DR, we can do exactly what we did for sudoku: Measure progress by the behavior of the gap, the currently achieved distance between a pair of points in sets A and B. Points $p_A \in A$ and $p_B \in B$ now correspond to an assignment of values to all the replicated x, y, and w variables. If you take a moment to review the deconstruction and what it means when $p_A = p_B$, you will realize that the problem has been solved: A concurring set of replicated variables that satisfy all the constraints at the neurons and the input/output nodes.

Figure 1.11 shows the progress of the gap for the $n = 32$ binary autoencoder problem. Unlike the loss in gradient descent, the gap does not decrease in every iteration. When it plummets toward zero, the training is complete. There is no point in plotting a distribution analogous to Figure 1.9 because

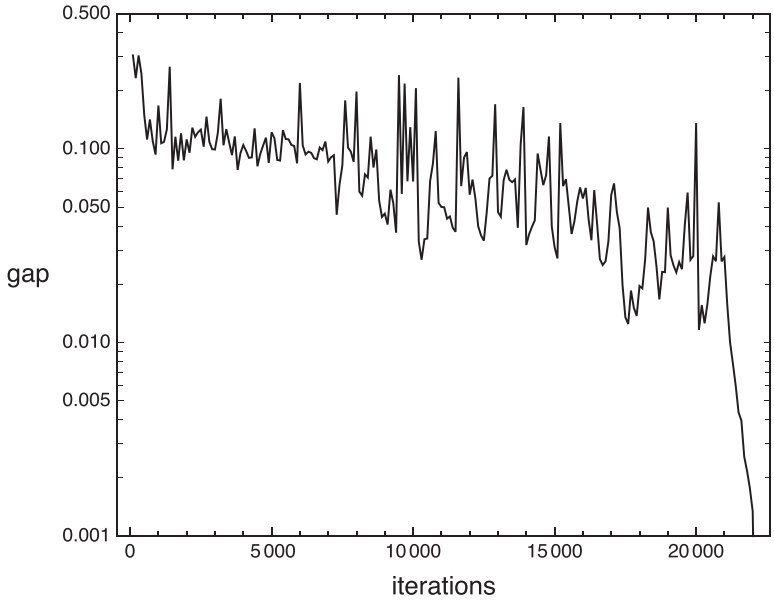

Figure 1.11 Progress of the gap in the training of the deconstructed autoencoder by DR iterations.

zero-gap means all the concur projections (P_B) of the neuron outputs match the binary values dictated by the neuron constraints (P_A). Finally, if you review the DR formula (1.1), you will see that a gap of size zero means the dynamics has arrived at a fixed point.

I could also mention that training the deconstructed network using DR is significantly faster than gradient descent training, but that misses the more important point that even after reaching zero loss the gradient-descent trained autoencoder has not learned to encode all the data into binary codes. This is a serious shortcoming when the top decoder-half of the trained autoencoder is to be used as a generative model. A generative model is supposed to generate the data when a simple code is sampled at the decoder's inputs. This fails when the codes generated in the autoencoder training have the distribution of Figure 1.9. Another benefit of the interpretability of the codes generated in the deconstructed problem is that the learned weights are interpretable as well, thereby eliminating, in this instance at least, the black-box nature of the data representation. To achieve this to the fullest extent, a norm constraint on the weights and a margin separating the neuron outputs must be imposed as well. These are minor tweaks in the definitions of A and B and covered in Section 7.7.

There is no evidence that either training method – by gradient descent on

loss or DR iterations on the deconstructed problem – is relevant for *natural* neural networks. The method used by biology is probably none-of-the-above. On the other hand, the method that makes the most sense from an engineering perspective can and should be debated. But the gradient method is embedded so deeply in our technology, it is unrealistic to imagine it being supplanted any time soon. We can draw a parallel with the long reign of the Roman Empire. Similar to the many amenities provided by the main purveyors of machine learning, the Romans kept the roads paved and water in the viaducts flowing. However, they also imposed – and here my analogy is inspired by the binary encoding exercise – a rather poor system for representing numbers! Though 2023 may prove to be a pivotal year in the deployment of artificial intelligence, the core technology may in introspect be remembered as vintage MMXXIII.

The Roman system of representing numbers held sway for hundreds of years. If this depresses you, then you might find comfort in a story where "One small village of indomitable Gauls still holds out against the [Roman] invaders" [47]. The villagers owed their ability to resist assimilation, in part, to a magic potion that conferred superhuman powers. While I cannot promise you superhuman powers, and there is nothing magical about the skills you will learn, the new method of solving problems will at times seem miraculous.

Exercises

1.1 Though a whole chapter is devoted to the Douglas–Rachford formula (1.1) and an important generalization, now[7] is a good time for you to establish the very direct relationship between fixed points and solutions. If x^* is a fixed point, then

$$x^* = x^* + P_B(R_A(x^*)) - P_A(x^*) \; .$$

(a) Use this equation to write down an explicit expression for a solution $x_{\text{sol}} \in A \cap B$. Your expression will *involve* x^* but will not be as simple as $x_{\text{sol}} = x^*$.

(b) As a first example of the relationship between fixed points and solutions, consider a problem in the plane (two variables). For A take an infinite line and for B a single point on that line. The solution x_{sol} is obvious, but what about the fixed points x^*?

1.2 In the deconstruction of the disk packing problem, the radii were allowed to vary, and the projection P_B was tasked with restoring their true values (those of the original puzzle). Suppose the puzzle has three disks

whose radii comprise the set $R = \{1, 2, 3\}$. Let (r_1, r_2, r_3) be the vector of radii that P_B will act on. In the complete problem P_B does a lot more, but in this exercise we just focus on the restoration of the disks' radii. For example, it might turn out that

$$P_B(r_1, r_2, r_3) = (2, 1, 3) \ ,$$

or some other permutation of the numbers in R. You don't know which is the correct one until you are given the numbers r_1, r_2, and r_3. The projection $(2, 1, 3)$ is correct if the Euclidean distance, or its square,

$$(r_1 - 2)^2 + (r_2 - 1)^2 + (r_3 - 3)^2 \ ,$$

is the smallest possible for the particular (r_1, r_2, r_3).

(a) Work out the projection

$$P_B(\ 0.5\ ,\ 2.1\ ,\ -0.5\ ) \ .$$

While it's true that a negative radius makes no sense, it's important that projections can handle arbitrary inputs. Notice that the reflection R_A in the Douglas–Rachford formula can turn positive numbers negative.

(b) With three disks, there are only $3! = 6$ cases to consider. Can you find an efficient algorithm that avoids trying all $n!$ cases (!) when the packing has n disks?