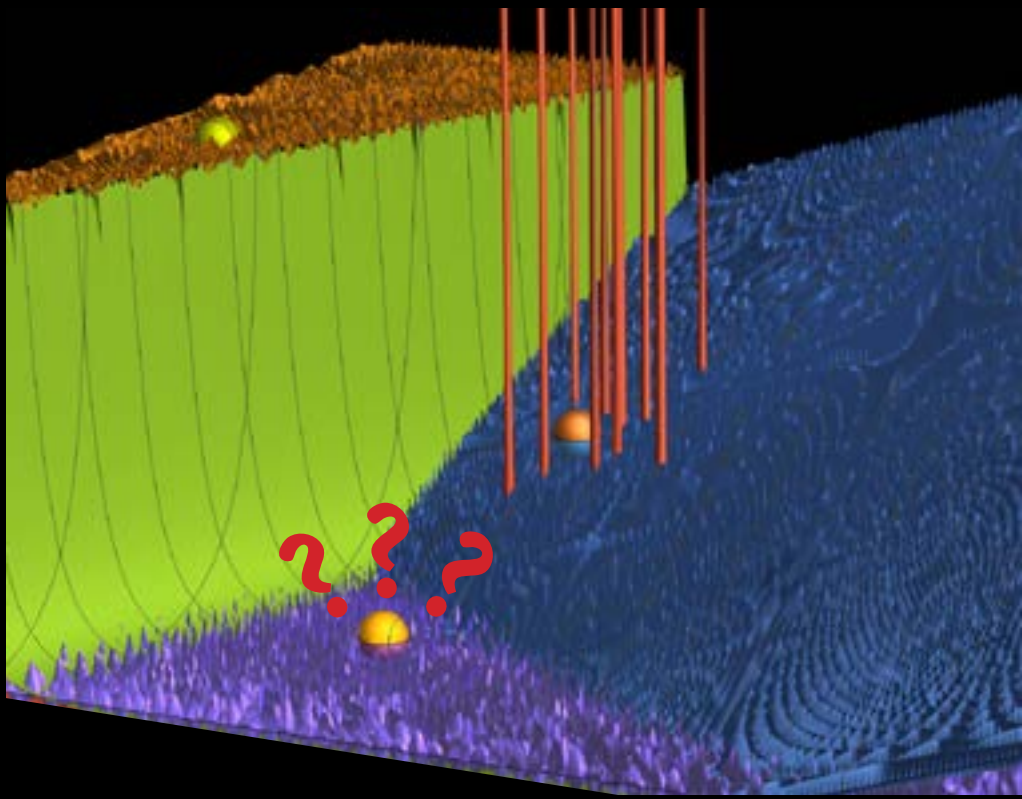
7C4

Swarm Explorer for the matching algorithm of the University of Maryland's Electron Ring -



7C4's story

"To The Swarm, 7C4 and neighbors from iteration 1, reporting minimum of 8.076326--"

"To 7C4, Swarm reports less than 7.041 for iteration 1. Standby for Swarm Throw."

"To The Swarm, 7C4 and neighbors copy."

I placed my hand against the cliff, staring back up at where we'd come. One hundred units down, and surely somewhere below us here we would find a minimum. My neighbors all looked away, wondering which way The Swarm would will them to fly now that we were out of the running.

"To 7C4 and neighbors, we have your flight plan in 3... 2..."

We looked at one another, tensed and waiting to be thrown.

"1."

I flew backwards, right back up and over the cliff we had been so excited to get tossed down on the previous iteration. The sky was as it always is, blank and black. As I tumbled through space I could see hundreds of those cliffs, close to the same size, scattered out to the horizon. What were these structures? Some were hundreds of units long. Some were plateau's, others were ridges and off in the distance, I could just make out an entire forest of razor-thin fingers stretching towards the heavens.

I flew through the 6FF and we exchanged the looks of two professionals caught up in someone's race. As 6FF hurtled into the forest, I felt The Swarm Throw loose its grip and let me coast towards the ground, rising up to meet me. I crashed, bounced, skidded and tumbled over the uneven surface until I found myself rattling down sides of one of Alice's Rabbit Holes. I came to rest and found my surroundings to be 3.2075483. Finally!

The radio crackled to life with my neighbor's chatter: "Over one hundred... Same here... Well I'm better than last time... Seventeen..." I responded with my own minimum, "3.2 plus here." The cackling died out, then one last neighbor: "3.2558?" The radio was again silent. "I'm at 3.20 plus, who's tagged to call?"

"That's me, 7D8. What's your code?"

"To 7D8, 7C4 reports 3.207583."

"To 7C4, copy your 3.207583."

The minutes passed as 7D8 called The Swarm to report our minimum.

"We've got the new minimum neighbors, prepare for bombardment and shaking."

First my neighbors shook and settled down into the ground even further. I could hear them rattling against the walls as they found ever smaller nooks and crannies moving down, down down toward zero. Next came a hundred more neighbors screaming, skidding and skipping across the plateau above of and crashing down into our holes.

As the creaking slowed to silence, 7D8 barked over our radios: "Previous minimum here was 3.208 minus, report if you're lower." A few moments of shuffling followed as my neighbors took stock of their surroundings. Finally a couple of voices spoke up: "3.203 plus... 3.20278 plus." Another moment of quiet: "I'm over seven-eight."

Once again 7D8 filled the radio, "Taking the seven-eight, what's your code?"

Mathematics behind 7C4

And so the saga continued as the particle swarm sought the minimum of their landscape...

The above story is a flight of fancy from the author thinking about the Particle Swarm Optimization algorithm used in his work, which centered around finding a minimum difference between an injected beam of electrons and electrons in a ring, but we›ll get back to that in a moment.

First, finding a minimum of a function is a mathematical problem that arises in many fields of science and business. Companies want to minimize their costs for production, particles like to stay in the lowest possible energy levels, and pilots want their planes to keep a minimum difference between their ideal course and any turbulence the experience along their trip. All of these problems, and many more, fall under the mathematical problem of finding the minimum of a function.

One problem that arises in finding a minimum is if there are several minimums of a function, such as the function in Fig 1. We can look at the entire plot and see the minimum with the magenta dot is lower, but computers cannot see an entire plot like we can. They have to explore it one tiny step at a time.

One standard technique for finding a minimum is to start somewhere and look at the slope there. Next, take a small movement down the slope and look at the slope there. Continue this process until the slope is zero (or close to zero, computers rarely find it to actually be zero, but 0.000000000000001 might be small enough, depending on the question at hand). This is how the points in Fig. 3 arrive at the minimums.

This process works in a number of situations, and it can work in our test case of Fig. 1, but it might also fail. Depending on the function, the minimum found can vary

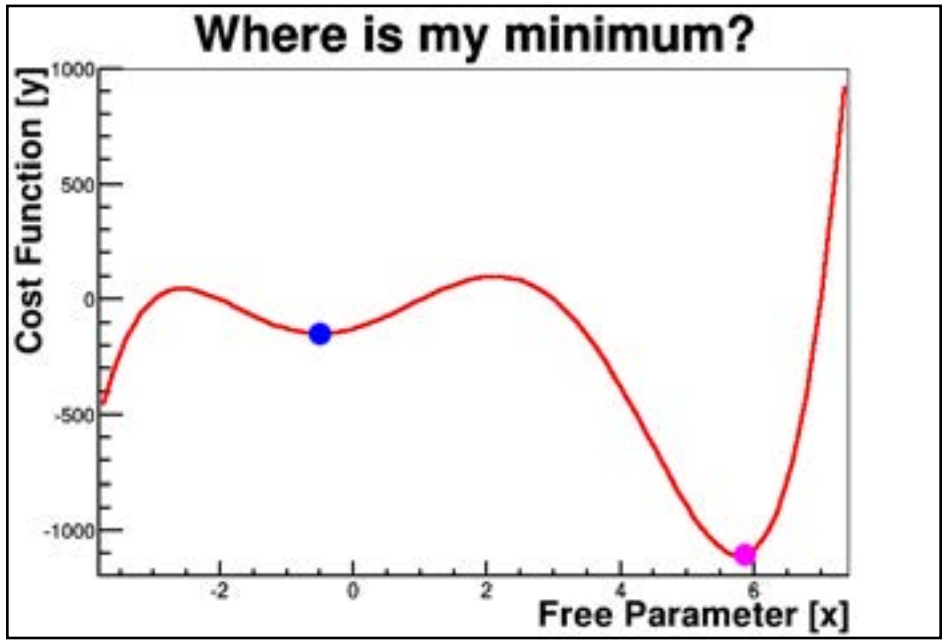


Fig. 1: This function has two minimums: the blue point and the magenta point.

-Drawn with ROOT

on the starting position. As you can guess, our test case depends on where we start as we look for the minimum.

One way to get around this problem is to start in many places, as we did with our three starting positions in Fig. 3. However, this requires us to know something about the shape of the function, so we can start in the correct place. Simple enough for our sample function, but what if we have more than one variable we need to alter, so our line becomes a surface? (Think of a sheet draped over a living room furniture to build a fort, and finding the lowest point on the sheet.) From there, maybe we have three free parameters and we wouldn't be able to draw the function as a picture. See Fig. 6 on the back cover for an example of two free parameters.

In my project, I had six free parameters, so getting a look at the six dimensional surface was out of the question.

Another approach was to use many random test points and take the one that landed on the lowest point of the graph. This has the advantage of checking many areas of the function, but it is also requires us the evaluate the function at every point. That's fine for a simple function, but some functions can take serious time to compute. In Fig. 4 I used 50 random points and took the best point. That's 50 tests, and only one free parameter.

For my project, using only 20 points per free parameter required sixty four million tests.

Considering I could run my test function three times per second, it would take me 215 days to run those points--far too long!

Clearly, I need a smarter way to look for a solution: The Particle Swarm!

The algorithm that influenced the beginning story uses a combination of many test points (a swarm of points) and moving around the function to find a better minimum.

The actual algorithm also employs a few more ideas, but the basic idea is there from this outline.

So what was this project? Teaching an electron beam to land on its feet, instead of its head.

At the University of Maryland Electron Ring, an electron gun shoots a bunch of electrons down a straight injection tube that is connected to a ring almost twenty feet in diameter. In order to keep the beam together and steer it around the ring the beam needs to be matched to the magnetic fields in the ring. The injection tube's job is make sure the electron beam enters the ring ready to follow the path of a matched beam around ring, i.e. landing on its feet in the ring. The injection tube does this by varying its own magnetic fields.

A simplified model of the beam is shown on the back cover. The surface represents the edge of the beam of electrons. First the electrons are focused in the X direction, thus the beam gets taller in the Y direction. Then the beam is focused in the Y direction, and it becomes wider in the X direction. This focusing is done by devices called quadrupoles in the ring and they are fixed in place.

So if the beam entered the ring already focused in the X direction, the first quadrupole would focus it even more. Then two things would happen: the electrons would be too close together in X direction and would push each other apart. Secondly the electrons would be too spread out in the Y direction and we would lose the electrons.

In practice, we don't need to look at the whole beam as in Fig. 5, we only need to know how wide is the beam in X, and how tall is the beam in Y. To this end we plotted both the width of X and the height of Y against the distance the beam had traveled, along with the solution to the ring in Fig. 2 below. The black lines are zero in the injection line, then represent the correct X and Y in the ring on the right side of the graph.

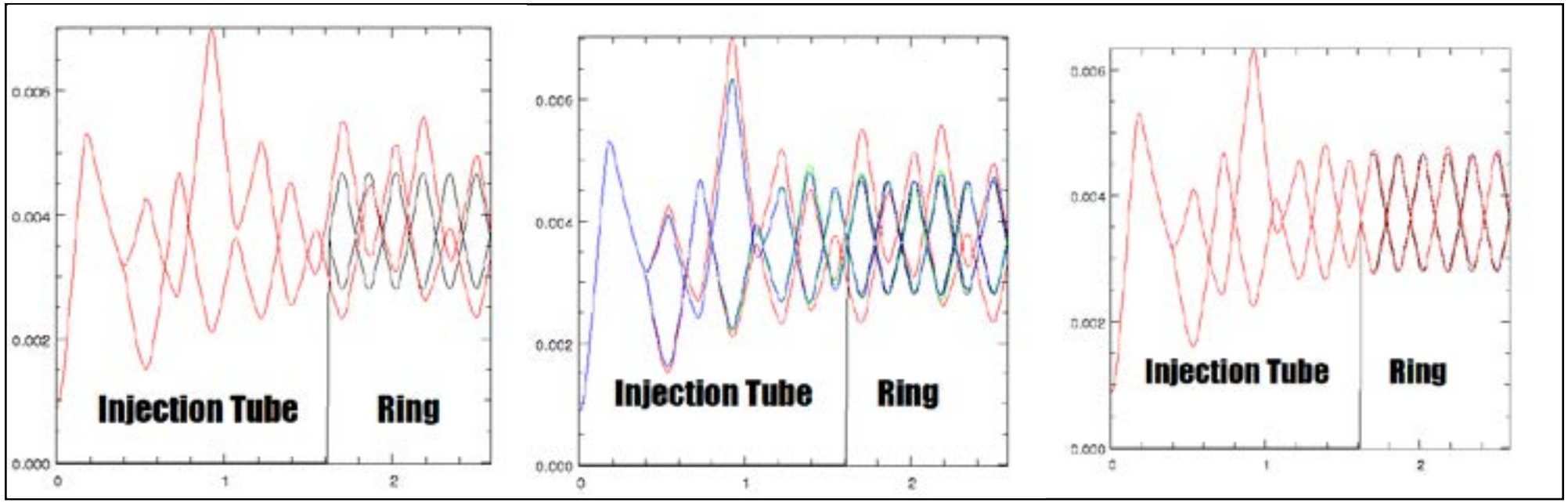


Fig. 2: Three solutions from the injection tube into the ring. The solution starts off far from the ring solution (left). After finding a few closer minimums, the solutions start to get closer (center). Finally, the last solution, the lowest minimum found, as a solution from the injection tube into the ring that is much closer than the original (right).

-Drawn with Python's Gist Package

So in order for the electron beam to land on its feet, we changed the quadrupole magnets until the beam from the injection tube matched the beam in the ring, as seen in the last frame of Fig. 2. This was my test function: how close is the beam from the injection line to the beam in the ring? This question took about a third of a second to answer--a long time for a computer than can add two numbers millions or billions of times per second.

Even after taking all this into account, it still took up to six hours to run a particle swarm, and often those solutions weren't perfect and required other fine tunings with other algorithms.

Afterword

The particles are named in hexadecimal, a way of representing number often used in computers, such as to name colors.

The hero of our story is named after the year I was born. The tagged particle to call is named after the first presidential election I was old enough to vote in and the particle the hero flew through, they don't bump into each other, is the birth year of one my favorite scientists.

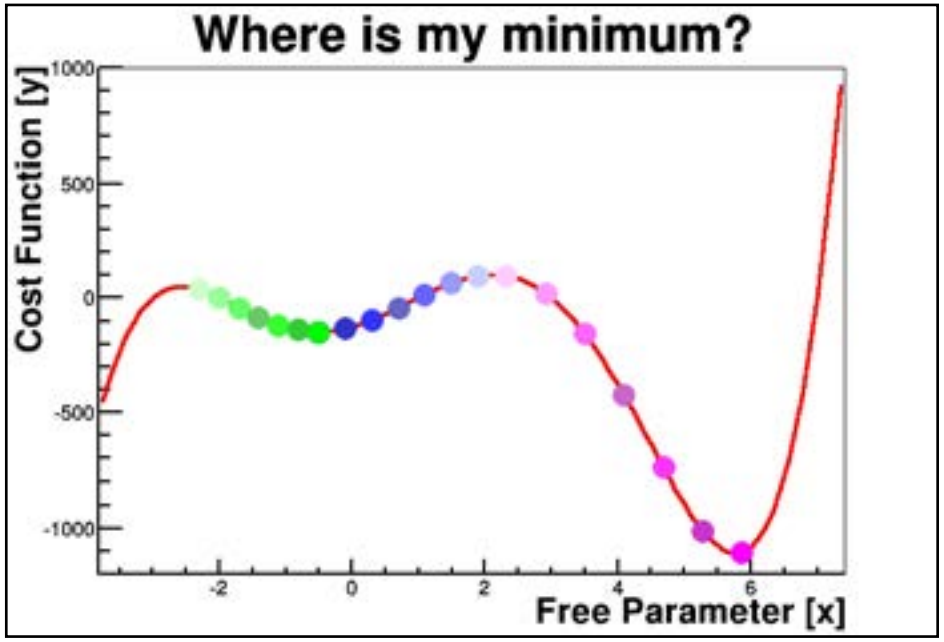


Fig. 3: Depending on where we start, faint points, we will roll towards one of two possible minimums, brighter points.

-Drawn with ROOT

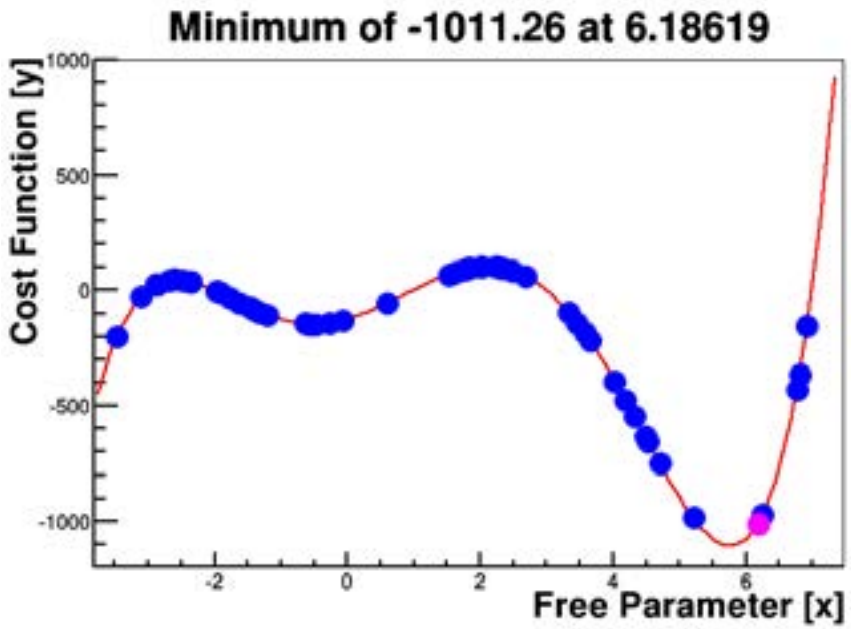


Fig. 4: Using 50 random test points, you'll usually get luck enough to test near the minimum. This technique however, is very copmutationally expensive for large problems.

-Drawn with ROOT

Notes on Images, Scientific Work

The landscape with particles on the cover were drawn with Wolfram's Mathematica program.

The example minimum plots were made with ROOT, CERN's C++ Data Analysis Framework, Figs. 1,3,4.

The beam visualization was drawn with Mathematica, though it takes a general shape from what we know about the beam. It does not reflect any experimental or simulated data, and is used for illustrative purposes only, Fig. 5.

The example beam plot was drawn in gist, after running the simulation in Warp, Fig. 5.

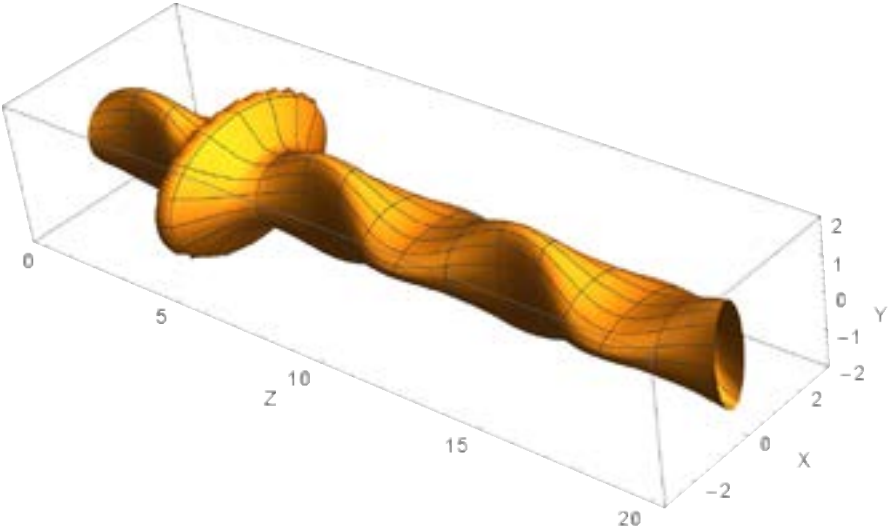


Fig. 5: This is a simplified 3D visualization of the beam. The bulge indicates altered shape in the injection line. The beam is squashed horizontally and vertically, alternating.

-Drawn with Mathematcia

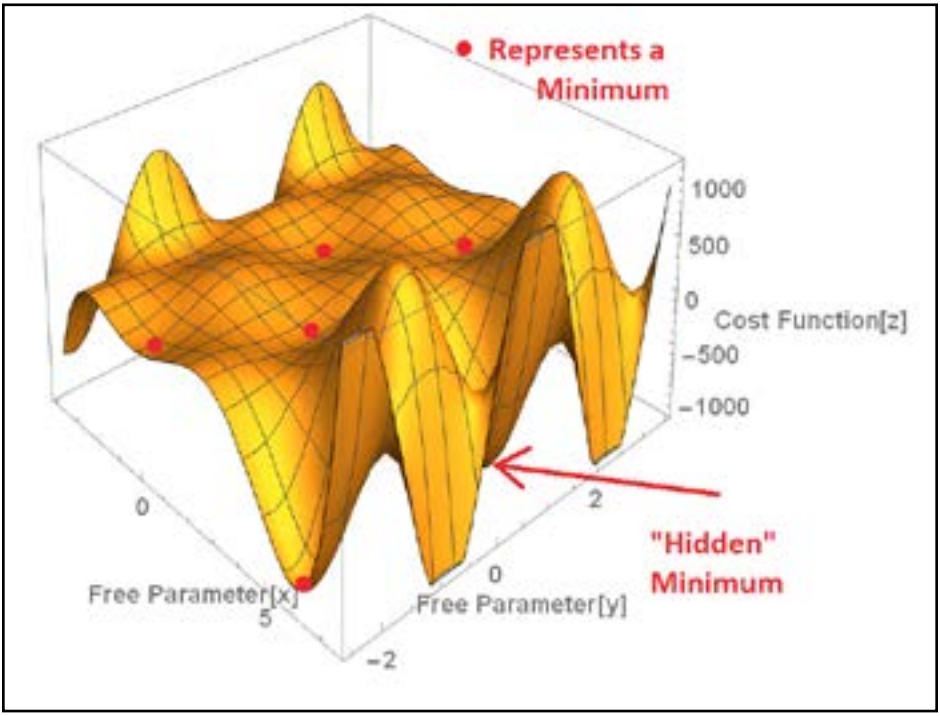


Fig. 6: Here is an example of a test function with two free parameters. Even though we can visualize it, the hidden minimum is not easy to see, and what if the pits were too small to see?

-Drawn with ROOT