05: The Sandpit (An Arena for Optimization)

Chapter Goal: To understand the practical application of the Jacobian/Gradient in **Optimization**—the process of finding the maximum or minimum value of a function.

---

1. Background: Using the Gradient/Jacobian

- **Recap:** We now know that the Jacobian (or Gradient for a scalar-output function) is a vector that points in the direction of the **steepest ascent**.
- **Goal:** To use this knowledge for **Optimization**, the process of finding the input values that result in the maximum (highest) or minimum (lowest) output of a function.

---

2. Two Ways to Find Extreme Points (Peaks/Valleys)

Method #1: Analytical (If We Have the Full Map)

1. Calculate the Jacobian/Gradient of the function `f`.
2. **Set the Gradient to Zero:** $\nabla f = [0 \quad 0 \quad \ldots]$.
3. **Solve the Equation:** Find all points `(x, y, ...)` that make the Gradient zero.

- **Why this works:** Exactly at the peak of a mountain or the bottom of a valley, the ground is perfectly flat. The slope in all directions is zero.
- **Problems:**
  - For complex functions, solving $\nabla f = 0$ can be extremely difficult algebraically.
  - This method will find **ALL** flat points, including small peaks (**local maxima**), small valleys (**local minima**), and **saddle points**. We won't know which one is the true highest or lowest point globally.

Method #2: Numerical / Iterative (If We Are "Blind")

- This is a much more common situation in Machine Learning.
- We don't have a complete map of our function's "landscape". Calculating the function's value at even a single point might be very "expensive" (e.g., takes a week on a supercomputer).

---

3. Analogies for "Blind" Optimization

Analogy A: Climbing a Mountain at Night

- **The Landscape:** Our function `f(x,y)`.
- **The Darkness:** We cannot see the entire landscape.
- **The Flashlight (Jacobian/Gradient):** At any point we are standing, we can turn on our "flashlight". This illuminates a "signpost" arrow on the ground that says "Peak This Way".
- **The Strategy (Gradient ASCENT):** Follow the direction of the arrow, step by step.
- **The "Local Maxima" Problem:** If you follow these arrows, you might end up on top of a small hill, not the tallest mountain. Once you reach the top of that small hill, all the arrows around you will point towards you, and you'll get "stuck," thinking you've reached the highest point.

Analogy B (Better): Finding the Deepest Point in a "Sandpit"

- **The Sandpit:** Our function's landscape. The sand covers the underlying shape.
- **Goal:** Find the deepest point (Global Minimum).
- **Tool:** A long stick to measure the depth.
- **The Process:**
  - You cannot see the shape of the sandpit floor.
  - You can only **probe** the depth at a few points `(x,y)` by sticking the stick in.
  - You cannot drag the stick through the sand. You must pull it out and probe again at a new location.
- **Connection to ML:** This is a better metaphor. "Probing with the stick" is like **evaluating our Loss function** for a specific set of parameters. The cost is the same, no matter how "far" the next point you choose is (unlike "walking" on a mountain).

---

4. Key Message

- Real-world optimization is often like finding the deepest point in this sandpit: we are "blind" and each measurement (function evaluation) is "expensive".
- Algorithms like Gradient Descent are clever strategies for choosing where to "probe with the stick" next, based on the local information (the gradient) that we have, so that we can get to the bottom of the valley as efficiently as possible.
- The "Sandpit" lab in this course is designed to let you "feel" the challenges of optimization (like the problem of local minima) directly.

---

5. A Concrete Simulation: Gradient Descent in Action

Let's simulate "playing in the sandpit" with a concrete numerical example. This will make the process of optimization (Gradient Descent) feel more real.

Our "Sandpit" Setup

- **The Shape of the Sandpit (Our Loss Function):**

  $$f(x, y) = x^2 + y^2$$

  This is a very simple function shaped like a perfect bowl.
- **Our Goal:** Find the deepest point of this sandpit. We already know the answer is `(0, 0)`, but we will pretend we don't and let our algorithm find it.

Step 1: Preparing the "Compass" (Calculating the Gradient)

First, we need our "compass" that tells us the direction of ascent. We calculate the Gradient of `f(x,y)`.

- $\frac{\partial f}{\partial x} = 2x$
- $\frac{\partial f}{\partial y} = 2y$
- **Gradient Blueprint:** $\nabla f(x, y) = \begin{bmatrix} 2x \\ 2y \end{bmatrix}$

Iterative Simulation: Finding the Bottom of the Valley

Let's begin our adventure.

Iteration #0: Start at a Random Point

- **Initial Position:** We drop our "hiker" at a random point, let's say $P_0 = (3, 4)$.
- **"Probing the depth":** How deep is it here?
  - $f(3, 4) = 3^2 + 4^2 = 9 + 16 = 25$. The "badness" score is 25.

Iteration #1: The First Step

1. **Read the Compass at P_0:**
   - We calculate the Gradient at `(3, 4)`.
   - $\nabla f(3, 4) = [2 \cdot 3, 2 \cdot 4] = [6, 8]$.
   - **Meaning:** From `(3,4)`, the direction of steepest **ascent** is in the direction `[6, 8]`.
2. **Determine the Downhill Direction:**
   - We want to go to the valley, so we take the **opposite** direction.
   - **Step Direction:** $-\nabla f = [-6, -8]$.
3. **Determine the Step Size (Learning Rate):**
   - Let's choose a small step size (α), for example, $\alpha = 0.1$.
4. **Take the Step:**
   - `Step = α * (-∇f) = 0.1 * [-6, -8] = [-0.6, -0.8]`.
   - `New Position (P₁) = Old Position (P₀) + Step`

- $P_1 = (3, 4) + (-0.6, -0.8) = (2.4, 3.2)$.

- **Result of Iteration #1:** Our hiker is now at $P_1 = (2.4, 3.2)$.
- **Check the new depth:** $f(2.4, 3.2) = (2.4)^2 + (3.2)^2 = 5.76 + 10.24 = 16$.
- Look! The depth has dropped dramatically from 25 to 16. Our strategy is working!

Iteration #2: The Second Step

1. **Read the Compass at P_1:**
   - We are now standing at `(2.4, 3.2)`.
   - $\nabla f(2.4, 3.2) = [2 \cdot 2.4, 2 \cdot 3.2] = [4.8, 6.4]$.
   - *Notice: The gradient vector is getting shorter! (Its length is 8, shorter than the previous length of 10). This is because the slope is getting gentler.*
2. **Determine the Downhill Direction:**
   - **Step Direction:** $-\nabla f = [-4.8, -6.4]$.
3. **Take the Step (with the same learning rate):**
   - `Step = 0.1 * [-4.8, -6.4] = [-0.48, -0.64]`.
   - `New Position (P₂) = Old Position (P₁) + Step`
   - $P_2 = (2.4, 3.2) + (-0.48, -0.64) = (1.92, 2.56)$.

- **Result of Iteration #2:** Our hiker is now at $P_2 = (1.92, 2.56)$.
- **Check the new depth:** $f(1.92, 2.56) \approx 3.68 + 6.55 = 10.23$.
- The depth dropped again from 16 to 10.23.

...And So On...

This process is repeated over and over.

- At each step, the hiker re-calculates the Gradient.
- The Gradient will get shorter and shorter.
- The steps will get smaller and smaller.
- The position will get closer and closer to the lowest point, `(0, 0)`.

When to Stop?

- After many iterations, say at Iteration #50, the hiker might be at a position like $P_{50} = (0.001, 0.002)$.
- The gradient here will be tiny: $\nabla f \approx [0.002, 0.004]$.
- The length of this gradient is very close to zero.
- The algorithm will say, "Okay, this is close enough to the bottom of the valley. Let's stop."

This is a step-by-step simulation of how **Gradient Descent** works.

7. Insights from "The Sandpit" Lab: What We Learned - Part 1

The "Sandpit" lab is more than just a game; it's a practical simulation of the challenges faced in real-world optimization. Here are the key insights from playing it.

Part 1: The Power of the Jacobian (Gradient)

> **My Observation #1:** *"My first attempts were clear. I could see where the arrows were pointing, and following them led me to the phone (the minimum)."*

This is the ideal scenario. The **Jacobian** (∇f) gives us a "superpower": a compass that points directly uphill. By simply taking a step in the **opposite direction** ($-\nabla f$), we are guaranteed to go downhill in the steepest possible direction. This is Gradient Descent in its purest form.

The Sandpit

Jacobian

The supervisor of a building site has dropped their mobile phone into a pit with an uneven floor, and it has rolled to the lowest point. To make matters worse, the pit has subsequently been filled with sand such that the phone is covered and cannot be seen. To find where in the pit their phone is, the supervisor has crafted a *'dip-stick'* with a head designed to measure the slope of the floor if it is poked straight down through the sand.

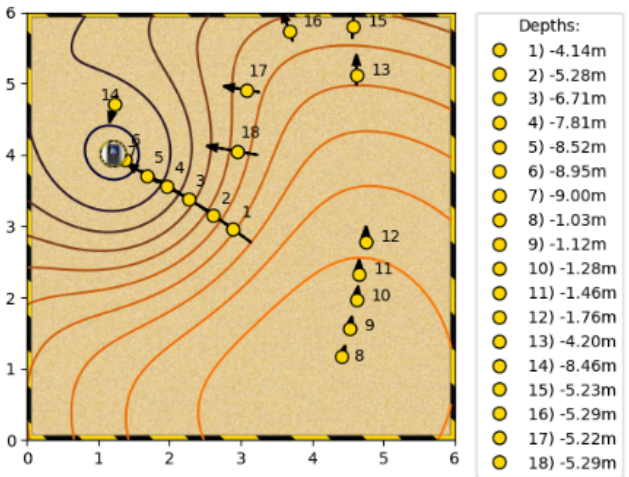
By clicking on any point in the sandpit, and thereby measuring the negative of the Jacobian, $-\mathbf{J} = -\nabla f(\mathbf{x})$, at that point, try and find the supervisor's phone. Try to do this with as few dips as possible - the supervisor has calls to make!

There is no grading for this exercise, when you are finished, close this tab to return to the course.

Run the following cell to start the example.

In [11]:
```
# Click into this cell and press [Shift-Enter] to start.
%run "readonly/sandpit-exercises.ipynb"
sandpit_intro()
```

```
/opt/conda/lib/python3.6/site-packages/IPython/core/magics/pylab.py:160: UserWarning: pylab import has clobbered these variable
s: ['imread']
`%matplotlib` prevents importing * from pylab and numpy
  "\n`%matplotlib` prevents importing * from pylab and numpy"
```



Congratulations!

Well done, you found the phone.

Part 2: The Challenge of "Getting Lost" (The Weakness of Gradient Descent)

> **My Observation #2:** *"At step 8, the arrow pointed upwards. Why? Is it because it's always perpendicular to the contour line? The arrows then kept pointing up, and I got lost until step 16. So in the real world, does this mean we have to 'pull out the stick' and start over?"*

Your analysis is **perfectly correct**, and you have discovered one of the fundamental challenges of Gradient Descent.

- **Why did the arrow point "up"?**
  - **Yes, precisely.** The Gradient is **always perpendicular** to the local contour line. In that specific part of the "landscape" (around step 8), the valley floor curves in such a way that the steepest way down *locally* is to move "up" on the map.
  - **Analogy:** Imagine you are in a winding canyon. To go deeper into the canyon, you might have to walk North for a while, even if the main exit is to the South. The Gradient only has **local information**; it doesn't see the big picture.
- **Does this mean we have to "pull out the stick and start over"?**
  - **Yes, exactly!** This is the core of your insight. The simulation shows that a simple Gradient Descent can get "lost" or take a very inefficient path in a complex landscape.
  - **This is a real weakness of Gradient Descent.** It can be very slow in "long, narrow valleys" (often called ravines) and can oscillate back and forth.
  - **Real-world solutions:** Yes, there are many solutions to this! More advanced optimization algorithms are designed to handle this:
    - **Momentum:** Gives the "hiker" some inertia, so they don't just follow the current gradient but also remember their previous direction. This helps smooth out the oscillations.
    - **Adaptive Learning Rates (e.g., Adam Optimizer):** Automatically adjusts the step size (α) to be smaller on steep parts and larger on flat parts.
    - **Random Restarts:** Yes, one simple strategy is to run the algorithm multiple times from different random starting points and pick the best result.

The Sandpit

Jacobian

The supervisor of a building site has dropped their mobile phone into a pit with an uneven floor, and it has rolled to the lowest point. To make matters worse, the pit has subsequently been filled with sand such that the phone is covered and cannot be seen. To find where in the pit their phone is, the supervisor has crafted a *'dip-stick'* with a head designed to measure the slope of the floor if it is poked straight down through the sand.

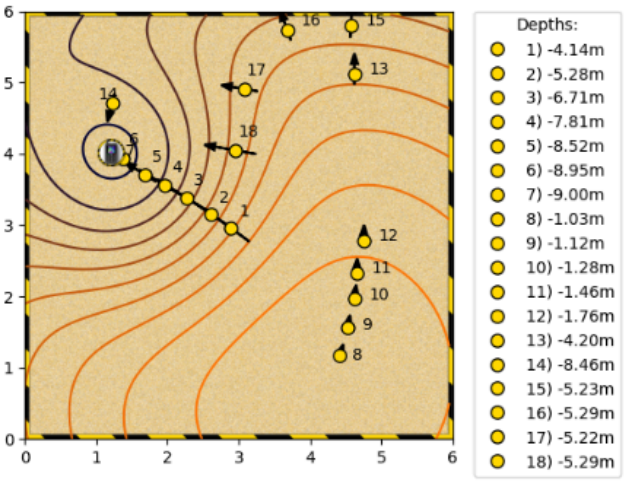
By clicking on any point in the sandpit, and thereby measuring the negative of the Jacobian, $-\mathbf{J} = -\nabla f(\mathbf{x})$, at that point, try and find the supervisor's phone. Try to do this with as few dips as possible - the supervisor has calls to make!

There is no grading for this exercise, when you are finished, close this tab to return to the course.

Run the following cell to start the example.

In [11]:
```
# Click into this cell and press [Shift-Enter] to start.
%run "readonly/sandpit-exercises.ipynb"
sandpit_intro()
```

```
/opt/conda/lib/python3.6/site-packages/IPython/core/magics/pylab.py:160: UserWarning: pylab import has clobbered these variable
s: ['imread']
`%matplotlib` prevents importing * from pylab and numpy
  "\n`%matplotlib` prevents importing * from pylab and numpy"
```



Congratulations!

Well done, you found the phone.

Part 3: The Blindness of the Computer (Why the Jacobian is Essential)

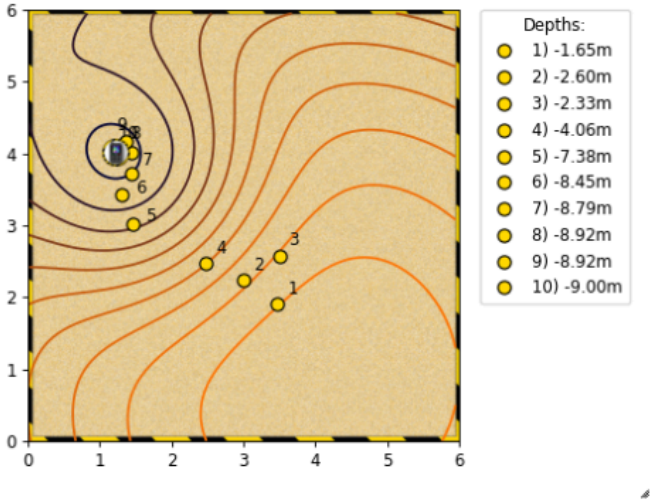
Depth Only

Before trying the dip-stick that measures the Jacobian, the supervisor had contructed a dip-stick that can only measure the depth on each dip.

Without information about the Jacobian, try and find the supervisor's phone. This should certainly be more difficult!

In [12]:
```
# Click into this cell and press [Shift-Enter] to continue.
%run "readonly/sandpit-exercises.ipynb"
sandpit_depth_only()
```

```
/opt/conda/lib/python3.6/site-packages/IPython/core/magics/pylab.py:160: UserWarning: pylab import has clobbered these variable
s: ['imread']
`%matplotlib` prevents importing * from pylab and numpy
  "\n`%matplotlib` prevents importing * from pylab and numpy"
```



Congratulations!

Well done, you found the phone.

As you can see, it's much more difficult to find the bottom of the sandpit when you can only sample the depth. Knowing the Jacobian makes decide where to try next.

> **My Observation #3:** *"If I didn't have the Jacobian (like in the second part of the lab), it would be incredibly hard. As a human, I can guess that 'down' is in a certain direction. But a computer doesn't have that ability, right? It would have to just randomly stick probes everywhere."*

You are 100% correct. This is the most important takeaway.

- **The Human Advantage:** We can see the entire contour map. Our brain can perform a **global analysis** and guess the general direction of the minimum.
- **The Computer's Reality:** A computer is "blind". It **cannot** see the whole map. All it can do is:
  1. Stand at one single point `(x,y)`.
  2. Calculate the value of the function `f(x,y)` at that point (the depth).
  3. Calculate the **Gradient** $\nabla f(x,y)$ at that *exact same point*.
- **Without the Jacobian/Gradient:**
  - If the computer only knows the depth at its current location, it has **zero information** about which direction is downhill.
  - Its only options would be:
    1. **Grid Search:** Try every single point on the map. Incredibly slow and impossible for high dimensions.

2. **Random Search:** Randomly probe points until it gets lucky. Very inefficient.

Conclusion:
The **Gradient** is the computer's "eyes". It's the **only piece of information** that allows a blind algorithm to make an **intelligent, informed decision** about where to step next. Without it, optimization in high-dimensional spaces would be practically impossible.

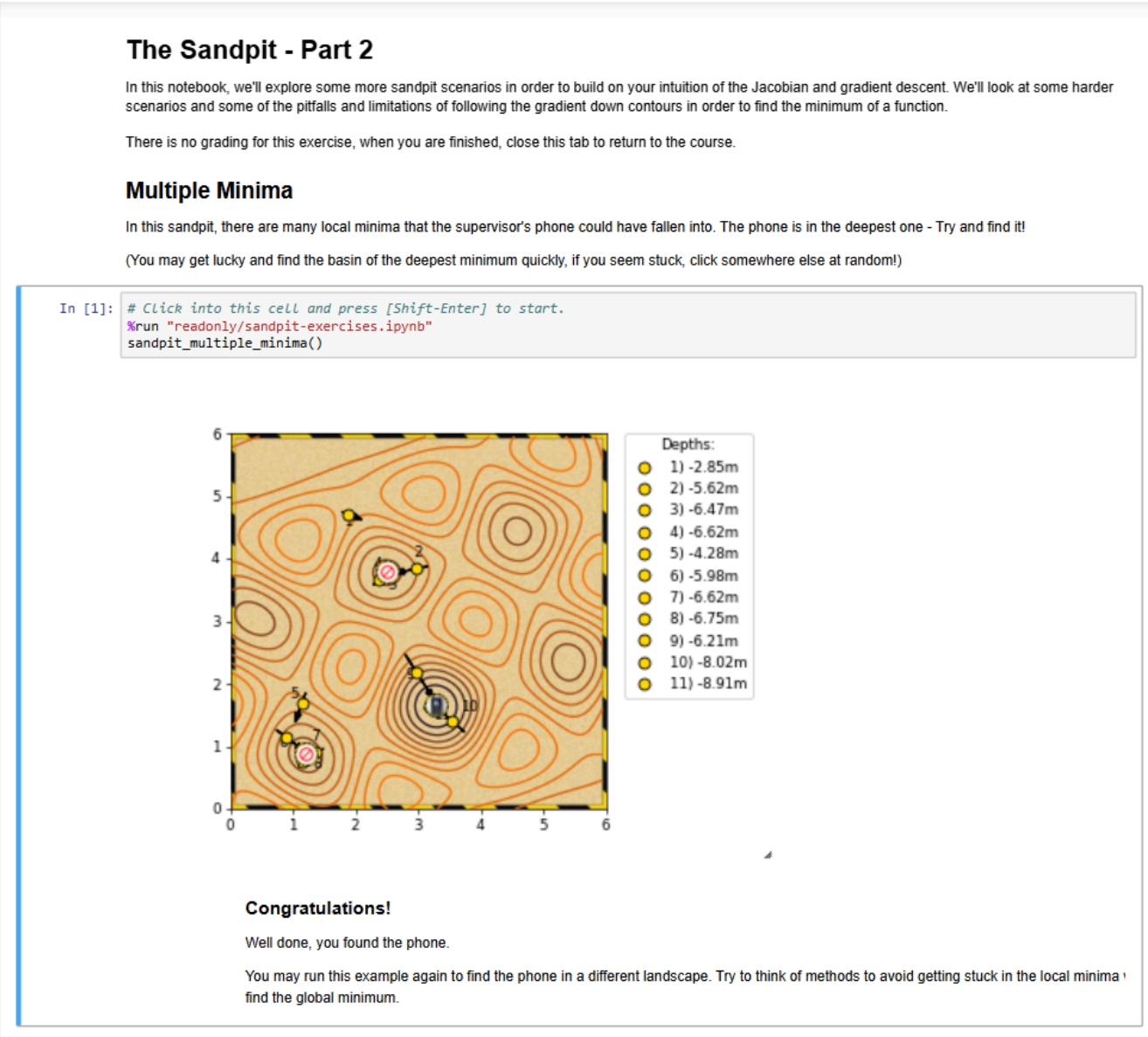
---

8. Insights from "The Sandpit" Lab - Part 2

"The Sandpit - Part 2" is the next level of our optimization game. While Part 1 taught us the basics of Gradient Descent, Part 2 confronts us with three more difficult, "real-world" scenarios that highlight the weaknesses and challenges of this algorithm.

Let's break down each scenario.

Scenario 1: Multiple Minima



- **The Problem Description:**

> *"In this sandpit, there are many local minima... The phone is in the deepest one - Try and find it!"*

- **What's Different?**
  - This is the classic **Local Minima vs. Global Minimum** problem!
  - There are several "pools" or "valleys" (areas surrounded by closed contours). Points 1, 2, 3, 4, and 5 are all at the bottom of these valleys.
  - **BUT**, only one valley is the absolute deepest (the **Global Minimum**), where the phone is.
  - The other valleys are "traps" (the **Local Minima**).
- **The Challenge For You:**
  - A "naive" Gradient Descent algorithm is highly dependent on your starting point.
  - If you start near point 2, your algorithm will happily descend the slope and stop at the bottom of valley 2, thinking it's done, even though the phone is in valley 1.
- **Strategies to Try:**
  - **Multiple Restarts:** Try starting from several different random points. Click near point 4, follow the gradient. Fail? Okay, try again from near point 5. Follow it again. Fail? Try from near point 3. Ah, this one seems to be heading into a deeper valley!

Scenario 2: Noisy Functions

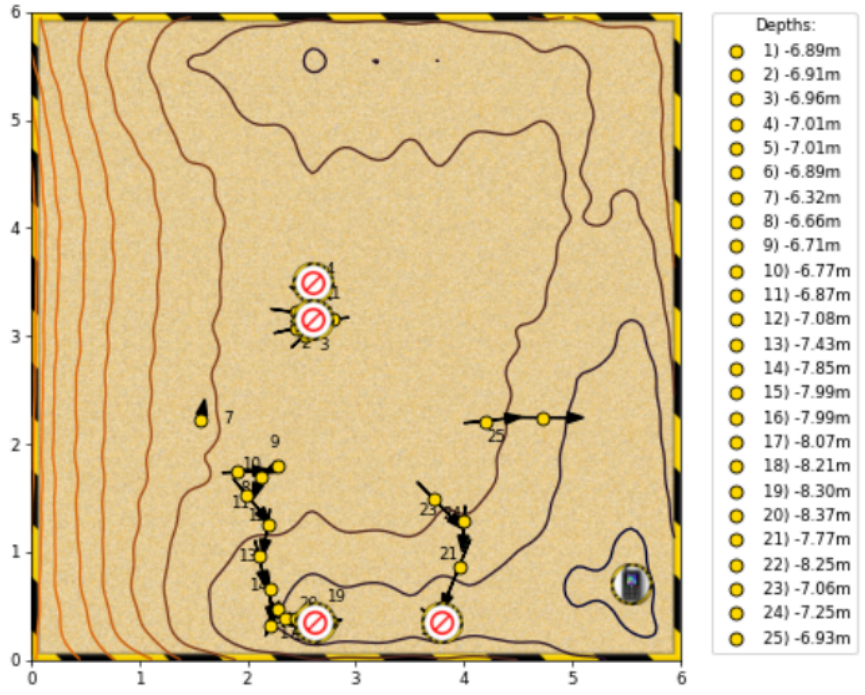
Noisy Functions ¶

Before the sand was loaded into this next pit, the pit floor was covered with rocks. This means when the supervisor tries to measure the slope, the surface they measure is rough and can change quickly.

Try to find the phone in this situation. You may notice that it is frustratingly harder to do so, as the roughness may generate more local minima.

In [2]:
```
# Click into this cell and press [Shift-Enter] to continue.
%run "readonly/sandpit-exercises.ipynb"
sandpit_rocks()
```

```
/opt/conda/lib/python3.6/site-packages/IPython/core/magics/pylab.py:160: UserWarning: pylab import has clobbered these variable
s: ['imread']
`%matplotlib` prevents importing * from pylab and numpy
  "\n`%matplotlib` prevents importing * from pylab and numpy"
```



Try again. You've taken too many tries to find the phone. Reload the sandpit and try again.

- **The Problem Description:**

> *"Before the sand was loaded... the pit floor was covered with rocks. This means when the supervisor tries to measure the slope, the surface they measure is rough and can change quickly."*

- **What's Different?**
  - The landscape is no longer smooth. It's full of "pebbles" and "tiny potholes".
  - Mathematically, this means the function is **"noisy"**. There are a huge number of tiny local minima everywhere.
- **The Consequence:** The Gradient becomes unreliable.
  - At one point, the gradient might point "south".
  - But if you move just a tiny bit, you might "step on a pebble" and the gradient suddenly points "north-west".
  - The path becomes very erratic and it's extremely easy to get stuck in one of the very shallow "pebble valleys".

    - Notice in the screenshot, you took 25 steps and still hadn't found the solution. This shows how difficult it is to navigate a "rough" landscape.
- **The Challenge For You:**
    - Here, the step size (learning rate) becomes critical. If your steps are too big, you might jump over many pebbles. If they are too small, you'll get stuck immediately.
    - This is a very common problem when working with noisy, real-world data.

> ⚠ **Warning**
>
> Saya masih gak paham, sehingga dibawah ini saya jelaskan dengan bahasa indonesia

Mari kita fokus total pada **Skenario 2: Lanskap yang "Berisik" (Noisy Functions)**. Kita akan lupakan dulu yang lain.

Ini adalah konsep yang sangat penting karena data di dunia nyata hampir **tidak pernah** mulus.

Analogi: Mencari Titik Terendah di Lapangan Berkerikil

Bayangkan kamu berada di sebuah lapangan rumput yang luas. Tujuanmu adalah menemukan titik terendah di lapangan itu.

Kasus Normal (Lanskap Mulus):

- Lapangannya seperti lapangan golf. Tanahnya bergelombang naik-turun, tapi **mulus**.
- Jika kamu meletakkan bola di satu titik, ia akan menggelinding dengan mulus ke arah turunan. Arahnya jelas.
- Ini adalah fungsi `f(x,y)` yang "sopan" dan mulus.

Kasus Skenario 2 (Lanskap "Berisik"):

> **Teks:** "...the pit floor was covered with rocks. This means... the surface they measure is rough and can change quickly."
> *(...dasar lubang ditutupi bebatuan. Ini berarti... permukaan yang mereka ukur kasar dan bisa berubah dengan cepat.)*

- Sekarang, bayangkan seseorang menebarkan **lapisan kerikil yang tebal** di seluruh permukaan lapangan golf itu.
- Secara **global**, bentuk "lembah" dan "bukit" utamanya masih ada.
- Tapi secara **lokal**, permukaannya sekarang **sangat kasar**. Penuh dengan tonjolan-tonjolan kecil dan lubang-lubang kecil di antara kerikil.

Apa Dampaknya pada "Kompas" Gradien?

Inilah masalah utamanya. Ingat, Gradien adalah **alat yang sangat lokal**. Dia hanya "merasakan" tanah **tepat di bawah kakimu**. Dia tidak bisa melihat gambaran besarnya.

Apa yang terjadi saat kamu menghitung Gradien di lanskap berkerikil?

1. **Posisi A:** Kamu kebetulan berdiri di sisi kanan sebuah kerikil kecil.
   - "Tanah" di bawah kakimu miring ke arah **Timur**.
   - Gradienmu akan menunjuk ke **Barat** (arah tanjakan kerikil).
   - Langkah Gradient Descent-mu akan diarahkan ke **Timur**.
2. **Posisi B (Geser 1 cm dari A):** Kamu sekarang berdiri di sisi kiri dari kerikil yang sama.
   - "Tanah" di bawah kakimu sekarang miring ke arah **Barat**.
   - Gradienmu akan menunjuk ke **Timur**.
   - Langkah Gradient Descent-mu akan diarahkan ke **Barat**.

Lihat Masalahnya?
Hanya dengan bergeser 1 cm, arah yang disarankan oleh "kompas"-mu bisa **berbalik 180 derajat!** Gradien menjadi **sangat tidak stabil dan tidak bisa diandalkan**. Dia lebih sibuk memberitahumu cara menuruni "kerikil" daripada cara menuruni "lembah" yang sesungguhnya.

Kenapa Algoritmanya Gagal?

> "Try again. You've taken too many tries to find the phone."
> *(Coba lagi. Kamu sudah mencoba terlalu banyak untuk menemukan teleponnya.)*

Algoritmamu gagal karena ia **terjebak**.

- Dia mengambil satu langkah ke Timur.
- Di posisi baru, gradiennya mungkin menyuruhnya kembali ke Barat.
- Lalu kembali ke Timur, lalu ke Utara, dst.
- Dia akan **"bergetar" atau "terjebak menari-nari"** di dalam salah satu **lubang kecil di antara kerikil-kerikil**.

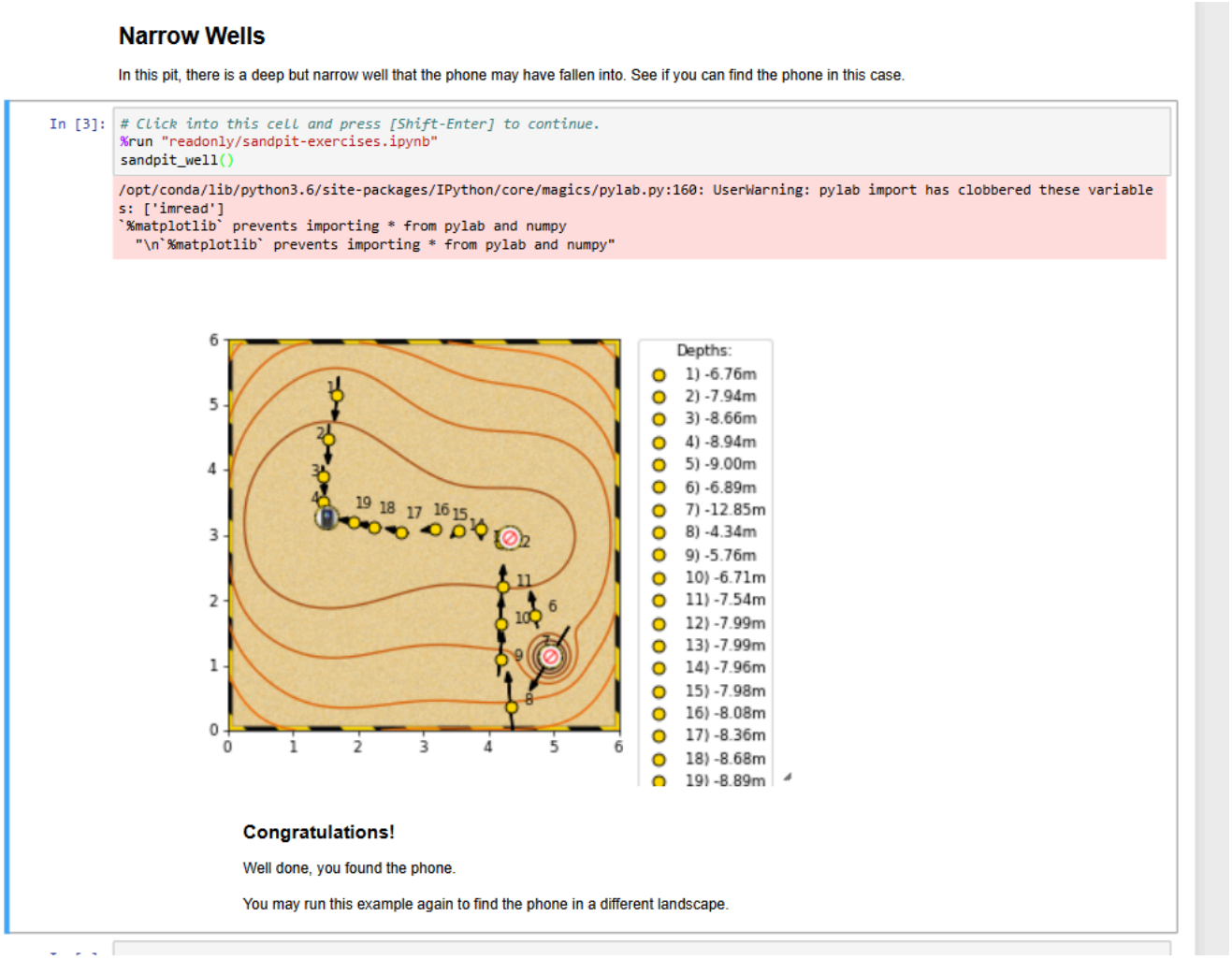
Lubang kecil ini adalah sebuah **Local Minimum**, tapi ia adalah lembah yang sangat dangkal dan tidak berguna. Algoritma akan berhenti di sana, mengira sudah menemukan solusi, padahal ia baru saja terjebak di antara dua butir pasir. Karena ia tidak pernah mencapai lembah yang sebenarnya, jumlah langkahnya akan habis dan kamu akan gagal.

Kesimpulan Intuitif:
Skenario 2 mengajarkan kita bahwa:

> **Gradient Descent sangat sensitif terhadap lanskap yang "kasar" atau "berisik". Informasi gradien lokal menjadi tidak bisa dipercaya untuk memandu kita dalam gambaran besar.**

Di dunia Machine Learning, ini sering terjadi. Data yang "berisik" bisa menciptakan banyak sekali *local minima* yang dangkal. Inilah mengapa teknik-teknik seperti *SGD* (yang "mengguncang" prosesnya) atau *momentum* (yang "menghaluskan" perjalanannya) menjadi sangat penting untuk membantu algoritma "melompati" kerikil-kerikil ini dan terus bergerak menuju lembah yang sesungguhnya.

Scenario 3: Narrow Wells (Valleys)



- **The Problem Description:**

  > *"In this pit, there is a deep but narrow well... See if you can find the phone in this case."*

- **What's Different?**
  - This is the **"Zig-zagging in Valleys"** problem that you identified earlier, but in an extreme form!
  - There is a very narrow and steep "canyon" or "well".

- **The Consequence:**
  - Remember, the Gradient always points **perpendicular** to the contour lines.
  - Inside a narrow canyon, the contour lines are almost parallel.
  - As a result, the `-Gradient` direction will tend to "bounce" back and forth from one wall of the canyon to the other, rather than walking straight down the canyon floor.
  - Look at the screenshot: the path from point 1 to 2, then to 3, 4, 5, etc., is a highly inefficient zig-zag pattern. You need a huge number of steps just to move a small distance downwards.
- **The Challenge For You:**
  - This shows why "naive" Gradient Descent can become incredibly **slow** on certain types of landscapes.
  - More advanced algorithms like **Momentum** or **Adam** are designed specifically to overcome this zig-zagging problem by "smoothing out" the step direction over time.

Conclusion:

Part 2 is not just an exercise. It's an interactive demonstration of three major problems in gradient-based optimization:

1. **Multiple Minima:** Getting stuck in a "good enough" solution, not the "best" one.
2. **Noisy Functions:** The gradient becomes unstable and difficult to follow.
3. **Narrow Valleys:** Convergence becomes extremely slow due to zig-zagging movement.

By "feeling" these problems yourself, you will have a much deeper appreciation for why ML researchers are constantly developing more advanced optimization algorithms.

> ⚠ **Warning**
>
> Saya masih gak paham, sehingga dibawah ini saya jelaskan dengan bahasa indonesia

Skenario 3, "Narrow Wells" (Lembah/Ngarai yang Sempit).

Analogi: Menuruni Ngarai yang Sangat Curam dan Sempit

Bayangkan kamu tidak lagi di lapangan terbuka, tapi kamu berada di dalam sebuah **ngarai** atau **jurang** yang sangat sempit.

- **Bentuk Ngarai:**
  - Dasar ngarai ini **menurun** ke satu arah (misalnya, ke arah "Selatan"). Inilah jalan menuju harta karun.

- **TAPI**, dinding di sisi kiri (Barat) dan kanan (Timur) sangat-sangat **curam**.

(Bayangkan ini adalah penampang ngarai. Arah "turun" yang sebenarnya adalah ke dalam layar, tapi dinding kiri-kanannya sangat curam).

Masalahmu:
Tujuanmu adalah berjalan menyusuri **dasar ngarai** ke arah Selatan. Tapi kamu membawa "kompas gradien" yang "bodoh".

Apa yang Dilihat oleh "Kompas" Gradien?

Ingat, Gradien selalu menunjuk ke arah **TANJAKAN PALING CURAM**.

1. **Kamu Berdiri di Sisi Kiri Ngarai:**
   - Kamu sedikit lebih dekat ke dinding Barat.
   - Kamu bertanya pada kompas, "Dari sini, arah mana yang paling cepat naik?"
   - Kompas akan melihat dinding Timur yang menjulang tinggi di seberang sana. Dinding itu jauh lebih curam daripada tanjakan landai ke arah "Utara" di sepanjang dasar ngarai.
   - **Kompas akan menunjuk lurus ke arah Dinding Timur.**
2. **Arah Langkahmu (`-∇f`):**
   - Karena kamu ingin turun, kamu mengambil arah sebaliknya.
   - Kamu akan berjalan lurus ke arah **Dinding Barat**, menyeberangi dasar ngarai.
3. **Kamu Tiba di Sisi Kanan Ngarai:**
   - Sekarang kamu berada di dekat dinding Timur.
   - Kamu bertanya lagi pada kompas.
   - Sekarang, kompas akan melihat dinding Barat yang curam. Ia akan menunjuk lurus ke arah **Dinding Barat**.
4. **Arah Langkahmu Berikutnya:**
   - Kamu mengambil arah sebaliknya, dan berjalan lurus kembali ke arah **Dinding Timur**.

"Aha!" Moment: Lahirnya Gerakan Zig-Zag

> **Teks:** (Ini adalah observasi dari gambar di lab) Perjalanan dari titik 1 ke 2, lalu ke 3, 4, 5, dst., adalah sebuah **pola zig-zag yang sangat tidak efisien**.

Inilah yang terjadi:

- Alih-alih berjalan dengan tenang menyusuri dasar ngarai ke arah Selatan, kamu malah **"memantul" bolak-balik** dari dinding kiri ke dinding kanan.

- Di setiap pantulan, kamu memang bergerak **sedikit** ke arah Selatan (turun sedikit), tapi sebagian besar energimu terbuang untuk gerakan **horizontal (kiri-kanan)** yang tidak produktif.

(Ini adalah visualisasi dari gerakan zig-zag. Arah yang benar adalah lurus ke bawah, tapi algoritma malah bolak-balik).

Kenapa Algoritmanya Gagal (menjadi tidak efisien)?
Karena Gradien itu "rabun dekat" dan "terobsesi dengan kecuraman".

- Dia melihat dinding ngarai yang super curam dan berpikir, "WOW! Ini tanjakan yang luar biasa! Arah inilah yang paling penting!"
- Dia **mengabaikan** fakta bahwa ada turunan yang landai tapi konsisten di sepanjang dasar ngarai, yang sebenarnya merupakan jalan menuju solusi.

Kesimpulan Intuitif:
Skenario 3 mengajarkan kita bahwa:

> **Gradient Descent "naif" sangat tidak efisien di lanskap yang memiliki "skala" yang sangat berbeda di setiap arah (sangat curam di satu arah, sangat landai di arah lain).**

Di dunia Machine Learning, lanskap seperti ini sangat umum terjadi. Ini disebut *ill-conditioned problem*.

- **Contoh:** Mengubah satu parameter mungkin memberikan dampak yang sangat besar pada Loss (dinding curam), sementara mengubah parameter lain hampir tidak ada dampaknya (dasar landai).

Inilah mengapa algoritma optimisasi modern seperti **Adam** atau **RMSProp** sangat populer. Mereka punya mekanisme internal untuk "melihat" sejarah gradien dan "menyesuaikan" ukuran langkah di setiap arah secara terpisah. Mereka bisa "sadar", "Oh, di arah ini kita terus-terusan bolak-balik. Mari kita perkecil langkah di arah ini. Dan di arah sana, kita tidak membuat kemajuan. Mari kita perbesar langkah di arah itu." Ini membantu melicinkan gerakan zig-zag dan mempercepat perjalanan menuruni ngarai.

Tip

Very recommended to watch this video :
https://www.youtube.com/watch?v=GkB4vW16QHI

Tags: #mml-specialization #multivariate-calculus #optimization #gradient-descent #local-minima