

Solving 2D Heat Equation Numerically using Python



G. Nervadof · [Follow](#)

Published in Level Up Coding · 6 min read · Oct 13, 2020



591



14



When I was in college studying physics a few years ago, I remember there was a task to solve heat equation analytically for some simple problems. In the next semester we learned about numerical methods to solve some partial differential equations (PDEs) in general. It's really interesting to see how we could solve them numerically and visualize the solutions as a heat map, and it's really cool (pun intended). I also remember, in the previous semester we learned C programming language, so it was natural for us to solve PDEs numerically using C although some students were struggling with C and not with solving the PDE itself. If I had known how to code in Python back then, I would've used it instead of C (I am not saying C is bad though). Here, I am going to show how we can solve 2D heat equation numerically and see how easy it is to “translate” the equations into Python code.

Before we do the Python code, let's talk about the heat equation and finite-difference method. Heat equation is basically a partial differential equation, it is

$$\frac{\partial u}{\partial t} - \alpha \nabla u = 0$$

If we want to solve it in 2D (Cartesian), we can write the heat equation above like this

$$\frac{\partial u}{\partial t} - \alpha \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) = 0$$

where u is the quantity that we want to know, t is for temporal variable, x and y are for spatial variables, and α is diffusivity constant. So basically we want to find the solution u everywhere in x and y , and over time t .

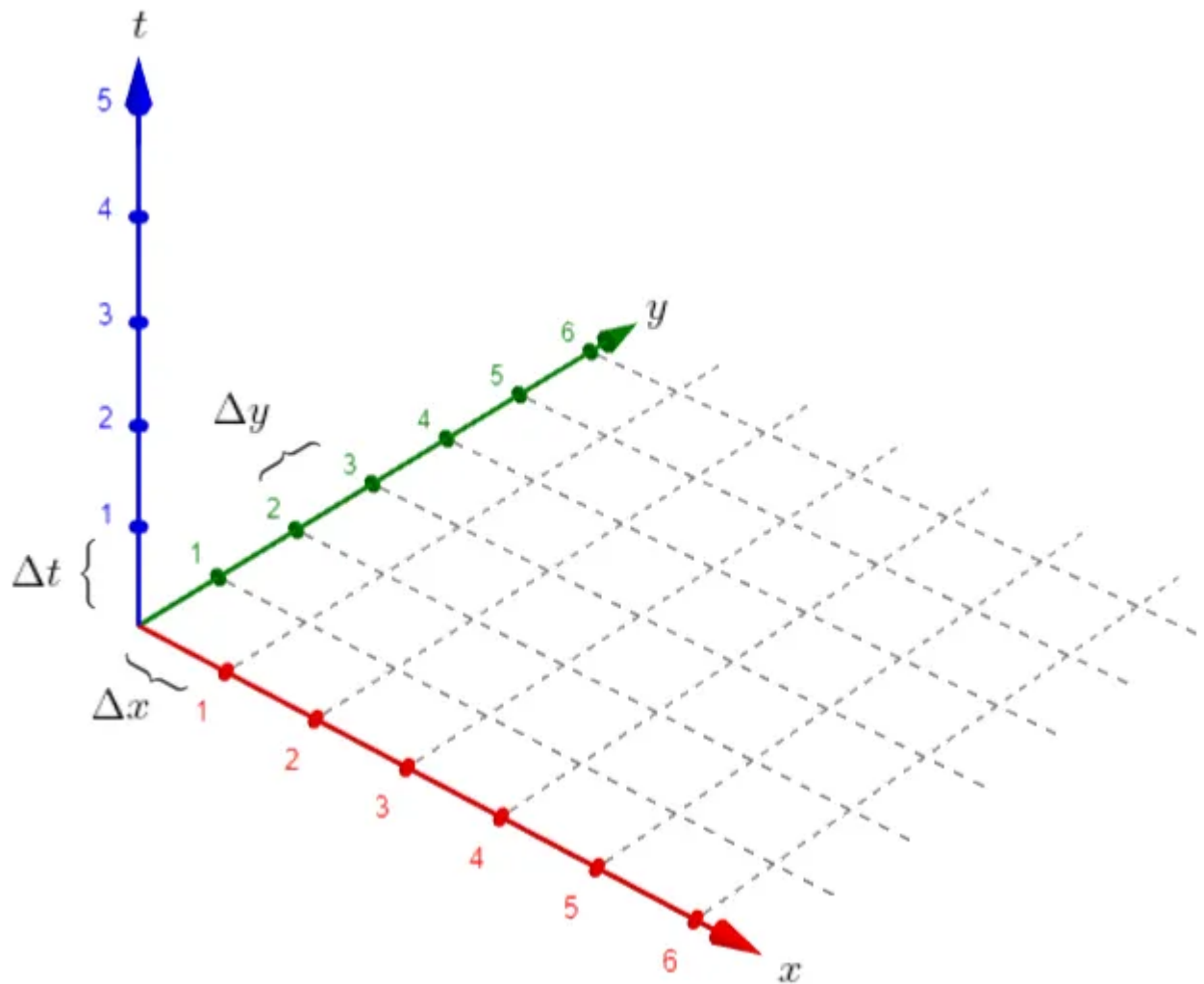
Now let's see the finite-difference method (FDM) in a nutshell. Finite-difference method is a numerical method for solving differential equations by approximating derivative with finite differences. Remember that the definition of derivative is

$$f'(a) = \lim_{h \rightarrow 0} \frac{f(a+h) - f(a)}{h}$$

In finite-difference method, we approximate it and remove the limit. So, instead of using differential and limit symbol, we use delta symbol which is the finite difference. Note that this is oversimplified, because we have to use Taylor series expansion and derive it from there by assuming some terms to be sufficiently small, but we get the rough idea behind this method.

$$f'(a) \approx \frac{f(a+h) - f(a)}{h}$$

In finite-difference method, we are going to “discretize” the spatial domain and the time interval x , y , and t . We can write it like this



Cartesian coordinate, where x and y axis are for spatial variables, and t for temporal variable (coordinate axes from [GeoGebra](#), edited by author)

$$x_i = i\Delta x$$

$$y_j = j\Delta y$$

$$t_k = k\Delta t$$

As we can see, i , j , and k are the steps for each difference for x , y , and t respectively. What we want is the solution u , which is

$$u(x, y, t) = u_{i,j}^k$$

Note that k is superscript to denote time step for u . We can write the heat equation above using finite-difference method like this

$$\frac{u_{i,j}^{k+1} - u_{i,j}^k}{\Delta t} - \alpha \left(\frac{u_{i+1,j}^k - 2u_{i,j}^k + u_{i-1,j}^k}{\Delta x^2} + \frac{u_{i,j+1}^k - 2u_{i,j}^k + u_{i,j-1}^k}{\Delta y^2} \right) = 0$$

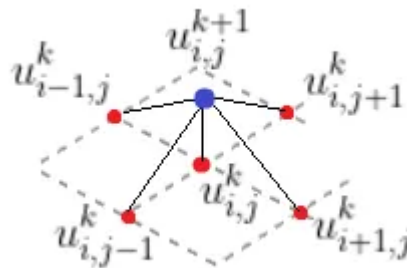
If we arrange the equation above by taking $\Delta x = \Delta y$, we get this final equation

$$u_{i,j}^{k+1} = \gamma(u_{i+1,j}^k + u_{i-1,j}^k + u_{i,j+1}^k + u_{i,j-1}^k - 4u_{i,j}^k) + u_{i,j}^k$$

where

$$\gamma = \alpha \frac{\Delta t}{\Delta x^2}$$

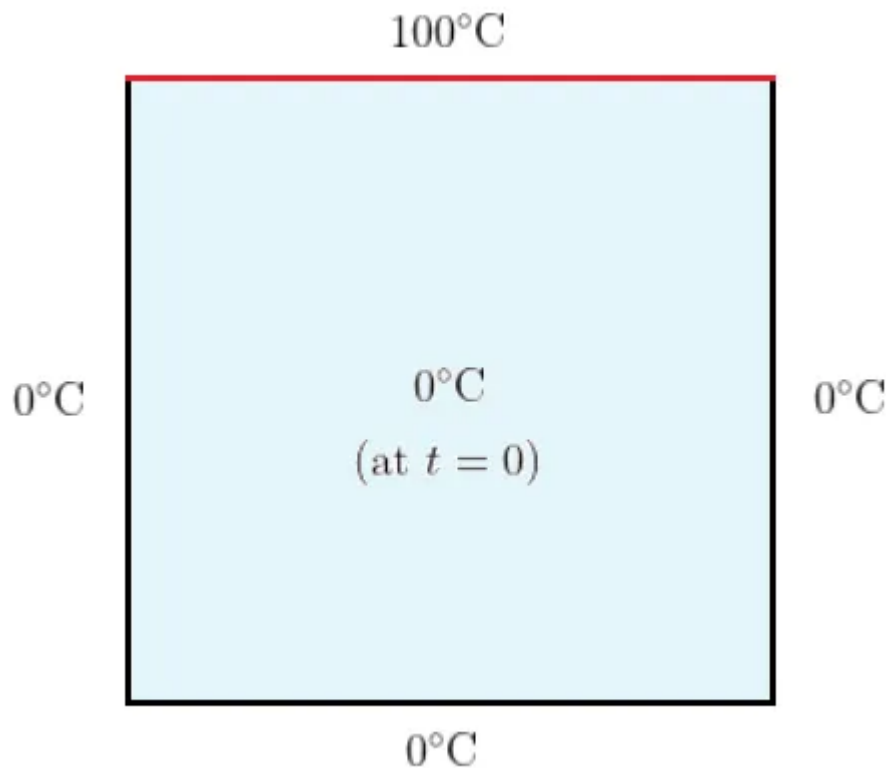
We can use this stencil to remember the equation above (look at subscripts i , j for spatial steps and superscript k for the time step)



We use explicit method to get the solution for the heat equation, so it will be numerically stable whenever

$$\Delta t \leq \frac{\Delta x^2}{4\alpha}$$

Everything is ready. Now we can solve the original heat equation approximated by algebraic equation above, which is computer-friendly. For an exercise problem, let's suppose a thin square plate with the side of 50 unit length. The temperature everywhere inside the plate is originally 0 degree (at $t = 0$), let's see the diagram below (this is not realistic, but it's good for exercise)



Boundary and initial conditions for our exercise (image by author)

For our model, let's take $\Delta x = 1$ and $\alpha = 2.0$. Now we can use Python code to solve this problem numerically to see the temperature everywhere (denoted by i and j) and over time (denoted by k). Let's first import all of the necessary libraries, and then set up the boundary and initial conditions.

```

1  # We use numpy (for array related operations) and matplotlib (for plotting)
2  # because they will help us a lot
3  import numpy as np
4  import matplotlib.pyplot as plt
5  import matplotlib.animation as animation
6  from matplotlib.animation import FuncAnimation
7
8  plate_length = 50
9  max_iter_time = 1000
10
11  alpha = 2.0
12  delta_x = 1
13
14  # Calculated params
15  delta_t = (delta_x ** 2)/(4 * alpha)
16  gamma = (alpha * delta_t) / (delta_x ** 2)
17
18  # Initialize solution: the grid of u(k, i, j)
19  u = np.empty((max_iter_time, plate_length, plate_length))
20
21  # Initial condition everywhere inside the grid
22  u_initial = 0.0
23
24  # Boundary conditions (fixed temperature)
25  u_top = 100.0
26  u_left = 0.0
27  u_bottom = 0.0
28  u_right = 0.0
29
30  # Set the initial condition
31  u.fill(u_initial)
32
33  # Set the boundary conditions
34  u[:, (plate_length-1):, :] = u_top
35  u[:, :, :1] = u_left
36  u[:, :1, 1:] = u_bottom
37  u[:, :, (plate_length-1):] = u_right

```

fdm_2d_heat_equation_conditions.py hosted with ♥ by GitHub

[view raw](#)

We've set up the initial and boundary conditions, let's write the calculation function based on finite-difference method that we've derived above.

```

1  def calculate(u):
2      for k in range(0, max_iter_time-1, 1):
3          for i in range(1, plate_length-1, delta_x):
4              for j in range(1, plate_length-1, delta_x):
5                  u[k + 1, i, j] = gamma * (u[k][i+1][j] + u[k][i-1][j] + u[k][i][j+1] + u[k][i][j-1])
6
7      return u

```

fdm_2d_heat_equation_algebraicalc.py hosted with ♥ by GitHub

[view raw](#)

Let's prepare the plot function so we can visualize the solution (for each k) as a heat map. We use Matplotlib library, it's easy to use.

```

1  def plotheatmap(u_k, k):
2      # Clear the current plot figure
3      plt.clf()
4      plt.title(f"Temperature at t = {k*delta_t:.3f} unit time")
5      plt.xlabel("x")
6      plt.ylabel("y")
7
8      # This is to plot u_k (u at time-step k)
9      plt.pcolormesh(u_k, cmap=plt.cm.jet, vmin=0, vmax=100)
10     plt.colorbar()
11
12     return plt

```

fdm_2d_heat_equation_plotheatmap.py hosted with ♥ by GitHub

[view raw](#)

One more thing that we need is to animate the result because we want to see the temperature points inside the plate change over time. So let's create the function to animate the solution.


```
1  def animate(k):  
2      plotheatmap(u[k], k)  
3  
4  anim = animation.FuncAnimation(plt.figure(), animate, interval=1, frames=max_iter_time, r  
5  anim.save("heat_equation_solution.gif")
```

fdm_2d_heat_equation_animate.py hosted with ♥ by GitHub

[view raw](#)

Now, we're done! Let's see the complete code below and run it.

```

1  import numpy as np
2  import matplotlib.pyplot as plt
3  import matplotlib.animation as animation
4  from matplotlib.animation import FuncAnimation
5
6  print("2D heat equation solver")
7
8  plate_length = 50
9  max_iter_time = 750
10
11  alpha = 2
12  delta_x = 1
13
14  delta_t = (delta_x ** 2)/(4 * alpha)
15  gamma = (alpha * delta_t) / (delta_x ** 2)
16
17  # Initialize solution: the grid of u(k, i, j)
18  u = np.empty((max_iter_time, plate_length, plate_length))
19
20  # Initial condition everywhere inside the grid
21  u_initial = 0
22
23  # Boundary conditions
24  u_top = 100.0
25  u_left = 0.0
26  u_bottom = 0.0
27  u_right = 0.0
28
29  # Set the initial condition
30  u.fill(u_initial)
31
32  # Set the boundary conditions
33  u[:, (plate_length-1):, :] = u_top
34  u[:, :, :1] = u_left
35  u[:, :1, 1:] = u_bottom
36  u[:, :, (plate_length-1):] = u_right
37
38  def calculate(u):
39      for k in range(0, max_iter_time-1, 1):
40          for i in range(1, plate_length-1, delta_x):
41              for j in range(1, plate_length-1, delta_x):
42                  u[k + 1, i, j] = gamma * (u[k][i+1][j] + u[k][i-1][j] + u[k][i][j+1] + u[k][i][j-1]) + (1 - 4 * gamma) * u[k][i][j]
43
44      return u
45

```

```
46 def plotheatmap(u_k, k):
47     # Clear the current plot figure
48     plt.clf()
49
50     plt.title(f"Temperature at t = {k*delta_t:.3f} unit time")
51     plt.xlabel("x")
52     plt.ylabel("y")
53
54     # This is to plot u_k (u at time-step k)
55     plt.pcolormesh(u_k, cmap=plt.cm.jet, vmin=0, vmax=100)
56     plt.colorbar()
57
58     return plt
59
60 # Do the calculation here
61 u = calculate(u)
62
63 def animate(k):
64     plotheatmap(u[k], k)
65
```

Open in app ↗

Sign up

Sign in



Search

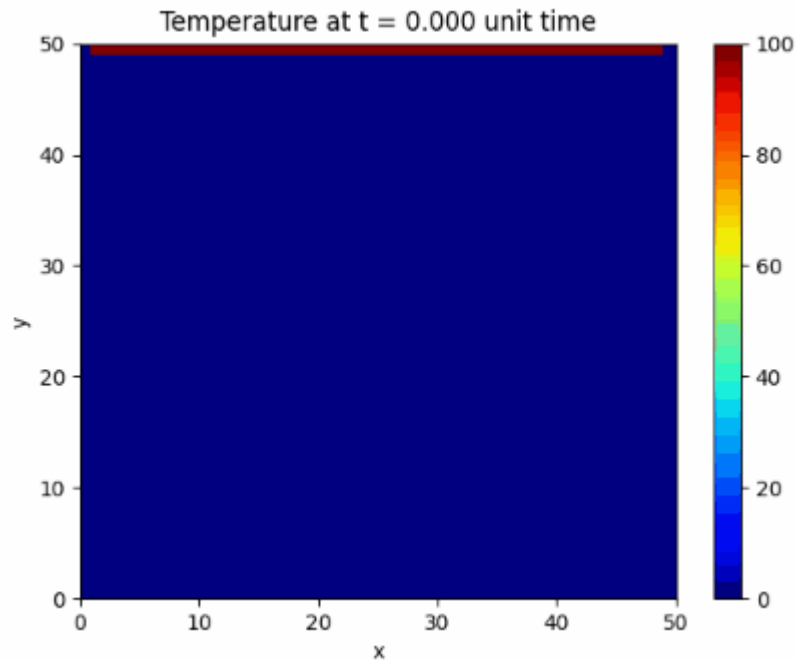
Write



fdm_2d_heat_equation.py hosted with ♥ by GitHub

view raw

That's it! And here's the result



The numeric solution of our simple heat equation exercise

Cool isn't it? By the way, you can try the code above using this Python Online Compiler <https://repl.it/languages/python3>, make sure you change the *max_iter_time* to 50 before you run the code to make the iteration result faster.

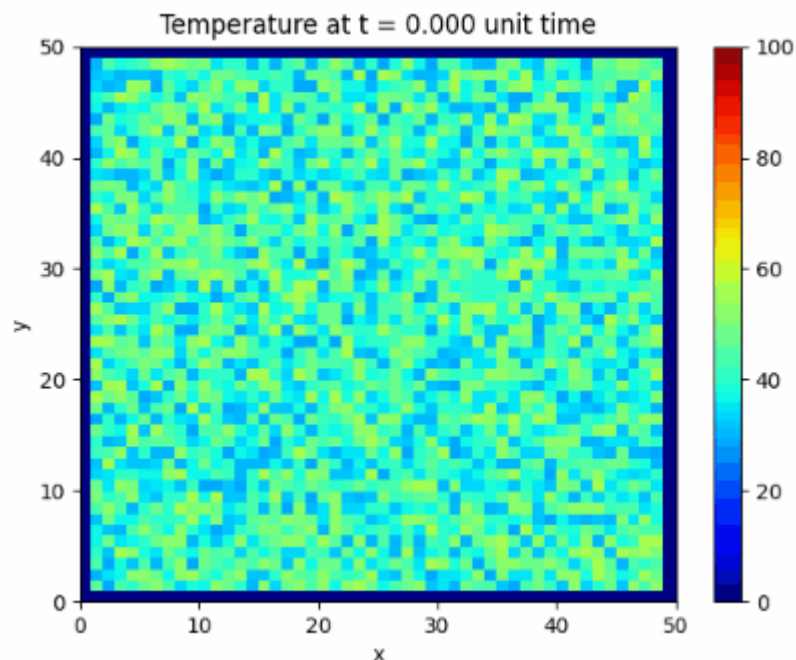
Okay, we have the code now, let's play with something more interesting, let's set all of the boundary conditions to 0, and then randomize the initial condition for the interior grid.

```
1  # Change boundary conditions
2  u_top = 0.0
3  u_left = 0.0
4  u_bottom = 0.0
5  u_right = 0.0
6
7  # Change u_initial (random temperature between 28.5 and 55.5 degree)
8  #u_initial = 0
9  u_initial = np.random.uniform(low=28.5, high=55.5, size=(plate_length,plate_length))
10
11 # Change initial conditions
12 #u.fill(u_initial)
13 u[0,:,:] = u_initial
```

fdm_2d_heat_equation_rand.py hosted with ♥ by GitHub

[view raw](#)

And here's the result



The numeric solution where all boundary conditions are 0 with randomized initial condition inside the grid

Python is relatively easy to learn for beginners compared to other programming languages. I would recommend to use Python for solving computational problems like we've done here, at least for prototyping

because it has really powerful numerical and scientific libraries. The community is also growing bigger and bigger, and that can make things easier to Google when we're stuck with it. Python may not be as fast as C or C++, but using Python we can focus more on the problem solving itself rather than the language, of course we still need to know the Python syntax and its simple array manipulation to some degree, but once we get it, it will be really powerful.

Level Up Coding

Thanks for being a part of our community! Level Up is transforming tech recruiting. **Find your perfect job at the best companies.**

Level Up — Transforming the Hiring Process

Enabling software engineers to find the perfect role that they love
Finding talent is the most painful part of...

jobs.levelup.dev

Python

Data Visualization

Numerical Analysis

Physics

Follow

158 Followers · Writer for Level Up Coding

Living in the emerald of the equator, believes that science and technology can and should make the world a better place | Unity in diversity

More from G. Nervadof and Level Up Coding

chosen, a domain name cannot be renamed.

Region •

We can host all of your data in any of these regions. Useful if you want to specify where to have your data hosted for any reason (e.g. EU)

New Region **BETA**



Get an extended trial duration during beta (ideal for non-production use cases). [Learn more.](#)



G. Nervadof

Auth0 and ASP.NET Core MVC (.NET 6.0) Tutorial

A tutorial to integrate Auth0 with ASP.NET Core MVC for sign-up, login, logout, email...

16 min read · Apr 7



96



Mohammad Faisal in Level Up Coding

Awesome Terminal Applications

Sharpen the axe before you cut the wood.

🌟 · 4 min read · Nov 12




3.5K



27





 Sanjay Priyadarshi in Level Up Coding

The 4 Rules of Writing Code Only Mature Developers Know About

Steal These

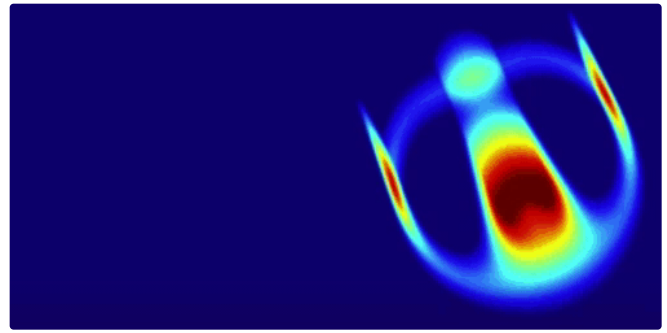
★ · 7 min read · Nov 7




1K



12



 G. Nervadof in Level Up Coding

Playing with Lenia, a Continuous Version of Conway's Game of Life

If you read this article, chances are you already know about Conway's Game of Life....

4 min read · Dec 19, 2021



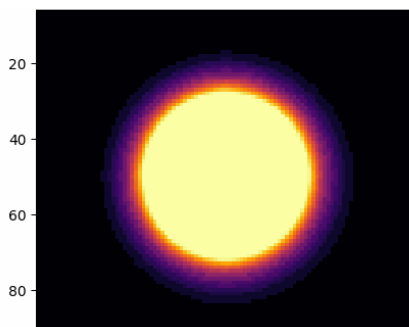
373



See all from G. Nervadof

See all from Level Up Coding

Recommended from Medium



○ Le Nguyen in Towards Data Science

Simulation 101: Conductive Heat Transfer

Conduction, or heat transfer between objects, is something we experience everyday. Here,...

★ · 11 min read · Jul 24



○ Antonio Pappaterra

Python in Action: Simulating Fluid Dynamics and Structural Analysis...

In the ever-evolving landscape of scientific and engineering simulations, Python has...

3 min read · Oct 18

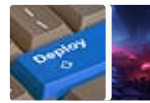


Lists



Coding & Development

11 stories · 313 saves



Predictive Modeling w/ Python

20 stories · 687 saves



Practical Guides to Machine Learning

10 stories · 785 saves

ChatGPT

23 stories · 311 saves

○ DZ in Python in Plain English

Monte Carlo Simulation with Matplotlib Animation

For some mathematical problems, an exact solution can be very difficult to obtain. Instea...

6 min read · Aug 1

○ Philip Mocz in Level Up Coding

Create Your Own Finite Difference Wave Equation Simulation (With...

In today's recreational coding exercise, we will simulate the Wave Equation with the Finite...

★ · 6 min read · Aug 28

 5 



 430 



 Tom Urbain


7 Significant Health Issues Every Off-Planet Astronauts Endure...

Imagine you've just stepped off a plane after a grueling 10-hour flight. Your body feels wear...

11 min read · Dec 5

 51 



 Ryan Kmetz

Satellite Imagery : One Size Doesn't Fit All

Understanding the differences between earth imagery resolutions.

3 min read · Aug 2

 2 



See more recommendations