

The Com π tition (Δ xe Edition)

In Spring 2025, my Numerical Methods professor ran a com π tition. Here’s the description:

Welcome to the Com π tition! You have been enlisted to approximate π . Your goal is to

1. Find as many distinct ways as possible to approximate π .
2. Find the **best** way to approximate π .

What follows is a ([very] belated) one-year anniversary remastered re-release of my submission to the contest, including exclusive, never-seen-before bonus content. Enjoy!

An exercise in pedantry

To begin, let’s take a stab at task (1). First, observe that the term “approximate” isn’t formally defined. It seems clear that an approximation of π should be somewhat close to the true value of π , but it is not clear *how* close, so we’ll assume that an approximation must be correct to at least one decimal place. Given this definition, we present the following infinite family of algorithms $\mathcal{L} = \{\text{LOOPHOLE}_r \mid r \in \mathbb{Q} \cap [0, 1)\}$, each providing an approximation of π to at least one decimal place:

```
1: function LOOPHOLEr
2:   return 3 + r
3: end function
```

We also observe that the assignment does not formally specify what it means for two algorithms to be distinct from each other. We’ll make the reasonable assumption that the most liberal definition of distinctness was implied; namely, that two algorithms are distinct exactly when their source code differs. We point out that by this definition, for any distinct rational numbers r_1, r_2 , the two algorithms LOOPHOLE_{r_1} and LOOPHOLE_{r_2} are distinct, and hence the family $\mathcal{L}$ contains a countably infinite number of distinct algorithms. Considering any possible algorithm is a string, a finite sequence of characters, there are only a countably infinite number of algorithms anyway, so this is the largest a family of distinct algorithms could actually be. In conclusion, $\mathcal{L}$ is a set of distinct π -approximation (a π ximation?) algorithms, and it is maximally sized, and so I am guaranteed to win or tie in task (1) of the com π tition.

For task (2), we once again leverage ambiguity. The word “best” could mean very many things, but the first definition that comes the mind is that the “best” is the speaker’s subjective favorite. This is a fairly common use of the word best, so it’s not implausible that it was the intended meaning here. So, what is the “best” a π ximation algorithm? At risk of giving almost all of my infinite algorithms a complex, I do have to say I’m personally quite fond of $\text{LOOPHOLE}_{16/113}$. That settles it, then. It seems I have found the “best” a π ximation, so I am guaranteed to win or tie in task (2) of the com π tition as well.

This concludes my submission for the Com π tition. For the cash prize I was promised, my Venmo handle is @aydencoughlin. Thank you for reading and have a wonderful day.

Okay, fine... in the spirit of the competition, I will establish a more charitable definition of an approximation, closer to what was probably intended: **an algorithm is an approximation if it can approximate π to any precision (ignoring hardware limitations) by sufficiently increasing some input of the algorithm.** This definition disqualifies the family $\mathcal{L}$ as well as some other cheating algorithms like MEMORIZATION and HALLUCINATION, shown below, as they break down for sufficiently large N .

```

1: function MEMORIZATION( $N$ )
2:    $\pi \leftarrow$  get first  $N$  digits of  $\pi$  from OEIS A000796
3:   return  $\pi$ 
4: end function

1: function HALLUCINATION( $N$ )
2:    $\pi \leftarrow$  ask ChatGPT: "What are the first  $N$  digits of pi? Only include the digits. Do not include any other text. "
3:   return  $\pi$ 
4: end function

```

From here forward, I'll only propose approximation algorithms following this definition.

Monte Carlo Methods

One particularly illustrative way of approximating π is the Monte Carlo circle method. We randomly sample points from inside of a square and determine which points lie inside of a circle inscribed in that square, which can fortunately be done quickly using the distance formula. We can use this count to approximate the area of the circle, and if we choose the radius of the circle to be one, its area should be exactly $\pi r^2 = \pi$. The more points we sample, the better this approximation gets.

```

1: function MONTECARLOCIRCLEAREA( $N$ )
2:    $i \leftarrow 0$ 
3:   for  $\_$  in  $1 \dots N$  do
4:      $(x, y) \leftarrow$  random point in  $[-1, 1]^2$ 
5:     if  $\text{dist}((0, 0), (x, y)) < 1$  then
6:       increment  $i$ 
7:     end if
8:   end for
9:   return  $2^2 \times i/N$ 
10: end function

```

But the *area* of a circle isn't the only place π shows up. We can also try the same idea to approximate the circumference. It's not immediately clear how we can generalize the method, since a circle's edge is one dimensional, but one idea is to count how many points lie a distance that is in the interval $[1 - \epsilon/2, 1 + \epsilon/2]$ from the center of the circle. If we divide the number of points by ϵ , we should expect to get a decent approximation of the circumference, improving in quality as $\epsilon \rightarrow 0$. We can then work backwards from $C = 2\pi r$ to approximate π .

```

1: function MONTECARLOCIRCLECIRCUMFERENCE( $N, \epsilon$ )
2:    $a \leftarrow 0$ 
3:   for  $\_$  in  $1 \dots N$  do
4:      $(x, y) \leftarrow$  random point in  $[-1, 1]^2$ 
5:     if  $1 - \epsilon/2 \leq \text{dist}((0, 0), (x, y)) \leq 1 + \epsilon/2$  then
6:       increment  $a$ 
7:     end if
8:   end for
9:   return  $2^2 \times a/(2N\epsilon)$ 
10: end function

```

We can also generalize the method to higher dimensions. The volume of a sphere with radius 1 is $4/3 \cdot \pi$, so three-quarters of a Monte Carlo approximation of the volume should give us an approximation.

```

1: function MONTECARLOSPHEREVOLUME(N)
2:    $i \leftarrow 0$ 
3:   for  $\_$  in  $1 \dots N$  do
4:      $(x, y, z) \leftarrow$  random point in  $[-1, 1]^3$ 
5:     if  $\text{dist}((0, 0, 0), (x, y, z)) \leq 1$  then
6:       increment  $i$ 
7:     end if
8:   end for
9:   return  $2^3 \times i/N \times 3/4$ 
10: end function

```

Similarly, the surface area of a sphere is $4\pi r^2$; we could use a similar method of sampling to approximate the surface area.

```

1: function MONTECARLOSPHERESURFACEAREA(N,  $\epsilon$ )
2:    $a \leftarrow 0$ 
3:   for  $\_$  in  $1 \dots N$  do
4:      $(x, y, z) \leftarrow$  random point in  $[-1, 1]^3$ 
5:     if  $1 \leq \text{dist}((0, 0, 0), (x, y, z)) \leq 1 + \epsilon$  then
6:       increment  $a$ 
7:     end if
8:   end for
9:   return  $2^3 \times a/(4N\epsilon)$ 
10: end function

```

It's pretty easy to see how you could do this for higher dimension spheres, assuming you knew how to determine the hypervolumes and hyperareas of them. But why stop at spheres? Two other three-dimensional shapes which contain π in both their volumes and surface area computations are the π pe and the π ke. π pes (colloquially known as “cylinders”) have volume $\pi r^2 h$. If we sample from the bounding cube of a π pe, we can determine whether a point lies in the π pe by just checking whether its first two coordinates lie in a circle when projected to a two-dimensional plane, and use this fact to approximate the volume of the π pe.

```

1: function MONTECARLOPIPEVOLUME(N)
2:    $i \leftarrow 0$ 
3:   for  $\_$  in  $1 \dots N$  do
4:      $(x, y, z) \leftarrow$  random point in  $[-1, 1]^3$ 
5:     if  $\text{dist}((0, 0), (x, y)) \leq 1$  then
6:       increment  $i$ 
7:     end if
8:   end for
9:   return  $2^3 \times i/N$ 
10: end function

```

Same story for π kes (also occasionally called “cones”). We sample a point, find the two-dimensional cross section at its last coordinate, a circle, and check if the other two coordinates lie in that circle. Again working backwards from the volume $\pi r^2 h/3$, we get an approximation for π .

```

1: function MONTECARLOPIKEVOLUME(N)
2:    $i \leftarrow 0$ 
3:   for  $\_$  in  $1 \dots N$  do
4:      $(x, y, z) \leftarrow$  random point in  $[-1, 1] \times [-1, 1] \times [0, 1]$ 

```

```

5:      if dist((0, 0), (x, y)) ≤ z2 then
6:          increment i
7:      end if
8:  end for
9:  return 22 × i / N × 3
10: end function

```

You could do the same thing we did with circles and spheres to approximate the surface areas of both of these fellas. Maybe you could even try hyperboloids or tetrahedra or star polyhedra or WHATEVER. What about sampling in four-dimensional space and approximating the surface volume of a four dimensional polytope??? The possibilities are endless.

A Culinary Approach

We mentioned earlier that we would ignore the issues created by limited precision due to hardware constraints, but I wanted to revisit this important practical concern in this section. Barring more advanced representations, most computers can only store a real numbers up to a precision of around 16 decimal digits using double-precision floating-point numbers. Therefore, we could only actually report π to 16 decimal digits in a function that approximates it, even if we could in principle compute more digits. The most straightforward way to circumvent this issue is to ditch the computer. It is only holding us back.

In real life, the Planck length is the bottleneck to how precisely things can be measured. The Planck length is about 1.62×10^{-35} meters; hence, if measuring objects on the order of about a meter, we can get about 35 decimal digits of precision, over double what we can in a computer using double-precision floating-point numbers. The physical world also grants us many luxuries not available in a computer. The following algorithm uses this to our advantage. In addition to our precision parameter N , this algorithm takes in as input some useful objects available in the real world: an oven O , a refrigerator F , a cupboard C , and a cabinet T . As a precondition, these containers are assumed to contain common items in them.

```

1: function BAKE $\pi$ E(N, O, F, C, T)
2:   // In this algorithm, I'm proud to be presenting my family apple pie recipe,
3:   // passed down over many generations. I first learned about this recipe from
4:   // my grandmother when I was just 3 years old. At the time, I didn't fully
5:   // appreciate just how much heart went into this recipe, but dozens of pies
6:   // later, I wish I'd thanked grammy more for sharing this recipe with me.
7:   // If she saw me now, I know she'd be proud of the lengths I've taken to preserve
8:   // this recipe and its intricacies so that I too may one day pass it down to my
9:   // grandchildren in this time-honored tradition.
10:  A ← Set of apples from C
11:  for a in A do
12:    Peel, core, and slice a
13:  end for
14:  B ← Mixing bowl from T
15:  Insert granulated sugar, light brown sugar, flour, cinnamon,
16:    nutmeg, lemon zest, and lemon juice and lime juice (3:1) into B
17:  // There's a lot to love about this specific recipe that sets it apart from the
18:  // myriad lookalikes you'll find on the internet or in cooking books. For one,
19:  // notice the replacement of one quarter of the lemon juice with lime juice. This
20:  // swap is absolutely essential, and it blows my mind just how much this improves
21:  // the taste of the dish! Looking back, grammy truly was a culinary genius; after
22:  // hundreds of hours of experimentation, this combo still results in a
23:  // pie with the perfect proportion of bitter, sweet, and acidic.
24:  Insert all of A into B; gently toss
25:  Preheat O to 400° F

```

```

26: // This is by far my least part of the recipe... waiting for half an hour
27: // for the oven to heat up to the proper temperature? That's why
28: // my grammy told me about RapiBake™, the latest innovation in baking technology.
29: // RapiBake™ is a portable oven that takes baking to the next level by providing
30: // state-of-the-art heating technology. The RapiBake™ SmartOven Plus features
31: // a uranium-based nuclear reactor (enrichment tools not included) which can be used
32: // to go from zero-to-400 in just two minutes: that's 80% shorter than the competition!
33: // Axe the wait and get a flexible, portable oven – it's a win-win for the whole family!1
34: With RapiBake™ SmartOven Plus: Enable QuickHeat to 400° F
35:  $D_b, D_t \leftarrow$  Pie crust discs from  $F$ 
36: Roll both  $D_b, D_t$  into 12" circles with thickness 0.125"
37: // This step above is where the money is at! Make sure to get this right, as any
38: // eccentricity has a large effect on the final precision of the output. I strongly
39: // recommend using a standard geometer's compass with a knife attached to the end
40: // to make sure you make it completely circular.
41:  $L \leftarrow$  Pie plate from  $T$ 
42: Place  $D_b$  into  $L$ 
43: Empty the contents of  $B$  into  $L$ , on top of  $D_b$ ; spread apple slices evenly
44: Place  $D_t$  onto  $L$ 
45:  $k \leftarrow$  Sharp knife from  $C$ 
46: // Speaking of knives... with  $\pi$  day festivities coming up, I wanted to shout
47: // out Knovia, a local business I've been getting gifts from for years. Knovia sells
48: // personalized knives – the unique part is that you can design a knife built from revolving
49: // any curve 15-30 degrees around the  $x$ -axis. Last year, I got my friend her very own
50: // Weierstrass function knife, and she absolutely loved it! Have a few friends that would
51: // adore a personalized function knife? You're in luck! Using my code AYDENMC at checkout
52: // on knoviaknives.com, you can get three knives for the price of  $\pi/2$ .
53: Use  $k$  to cut vents into  $D_t$  and trim excess dough
54:  $H \leftarrow$  Baking sheet from  $T$ 
55: Place  $L$  onto  $H$ 
56: Place  $H$  into  $O$ 
57: Wait 25 minutes
58: Set  $O$  to 375° F
59: Wait 30-35 minutes
60: Remove  $H$  from  $O$ 
61: With RapiBake™ SmartOven Plus: Dispose of radioactive waste
62:  $\pi e \leftarrow$  Contents of  $L$ 
63:  $r \leftarrow$  radius of  $\pi e$ , to  $N$  decimal places
64:  $C \leftarrow$  circumference of  $\pi e$ , to  $N$  decimal places
65: return  $C/(2r)$ 
66: end function

```

There are many ways this algorithm can be modified to yield different efficiency and efficacy outcomes in various contexts. The choice of apple will have the largest effect on the taste of the πe ; my grammy personally championed Honeycrisps.

¹Surprisingly, this product does *not* contain chemicals known to the state of California to cause cancer.

Function Things

While a circle itself has no functional representation, as it does not pass the vertical line test, a *semicircle* can be represented as a function quite easily. We can take the implicit circle equation and rewrite it, solving for y :

$$\begin{aligned}x^2 + y^2 &= 1 \\y^2 &= 1 - x^2 \\y &= \pm\sqrt{1 - x^2}\end{aligned}$$

Arbitrarily choosing the positive branch of the plus-minus, we get the function $f(x) = \sqrt{1 - x^2}$. We can now do function things to this function. One function thing we can do is integration. We know from an earlier section that the area of a circle is a terribly useful quantity for computing π . If we know the radius of the circle, we are typically one step away from π . The fact that we are dealing with a semicircle puts us only one multiplication away further. Another useful function thing is arc length. The arc length of this function corresponds to half of the circumference of the circle, which is again very easy to extract π from.

However, the function f itself is fairly difficult to analyze. We will not have much luck attempting to integrate or compute its arc length directly, but we can approximate it with, for instance, a linear spline, which is much easier to analyze. To get a linear spline approximation of this function, we can just sample equally spaced values of x .

```
1: function SEMICIRCLEAREA(N)
2:    $X \leftarrow \{n/N \mid n \in \{-N, \dots, N\}\}$ 
3:    $p \leftarrow \{(x, \sqrt{1 - x^2}) \mid x \in X\}$ 
4:    $s \leftarrow$  linear spline built from  $p$ 
5:    $A \leftarrow 0$ 
6:   for  $s_i \in s$  do
7:      $A \leftarrow A + \int_{X_i}^{X_{i+1}} s_i \, dx$  using power rule
8:   end for
9:   return  $2A$ 
10: end function

1: function SEMICIRCLEPERIMETER(N)
2:    $X \leftarrow \{n/N \mid n \in \{-N, \dots, N\}\}$ 
3:    $p \leftarrow \{(x, \sqrt{1 - x^2}) \mid x \in X\}$ 
4:    $s \leftarrow$  linear spline built from  $p$ 
5:    $C \leftarrow 0$ 
6:   for  $s_i \in s$  do
7:      $C \leftarrow C +$  arc length of  $s_i$  over  $[X_i, X_{i+1}]$ 
8:   end for
9:   return  $C$ 
10: end function
```

Speaking of functions... in some sense, we can think of π *itself* as a function $\pi : \mathbb{N} \rightarrow \{0, \dots, 9\}$ which takes in n and returns the n -th digit of π ; so, $\pi(1) = 3$, $\pi(2) = 1$, and so on. Functions can be interpolated, as we know, and it is completely safe to extend interpolated function into arbitrary domains, despite what they tell you. We can thus interpolate the function π restricted to, say, the first 100 digits, and then use the polynomial to compute the rest of the digits.

Before we showcase the full algorithm, there is one small wrinkle. This polynomial will not be quite right; for instance, for our interpolating polynomial $\tilde{\pi}$ will guess that the 101st digit of π is 2583720115864892292076046123003, when it is in fact 9. We can fix this by transforming the output so that it lies in the codomain $\{0, \dots, 9\}$:

$$\tilde{\pi}^*(x) = \lfloor \tilde{\pi}(x) \bmod 10 \rfloor$$

Now, the 101st digit of π is predicted to be 3, which still isn't right, but hey, it's closer! And this is an approximation anyway! We expect things to be wrong sometimes!!! Here is the full algorithm:

```

1: function DUBIOUS EXTRAPOLATION(N)
2:    $d \leftarrow \langle 3, 1, 4, 1, 5, \dots \rangle$  // first 100 digits of  $\pi$ 
3:    $s \leftarrow \{(i, d[i]) \mid i \in \mathbb{N}, 1 \leq i \leq 100\}$ 
4:    $\tilde{\pi} \leftarrow$  polynomially interpolate  $s$ 
5:   Let  $\tilde{\pi}^*(x) = \lfloor \tilde{\pi}(x) \bmod 10 \rfloor$ 
6:   return  $\sum_{i=1}^N \tilde{\pi}^*(N) \cdot 10^{1-i}$ 
7: end function

```

But there is really no reason to restrict our uses of $\tilde{\pi}^*$ to natural numbers. Before the transformation, it is just a polynomial, which we can plug any integer into. This allows us to discover some novel, never-seen-before digits of π . For example, the 0th digit of π (the one before the first 3) is $\tilde{\pi}^*(0) = 2$. Who knew? We can continue this backwards as far as we want. The *true* digits of π are therefore

...02268000428826448604802068442823.1415926535896821273526423832...

We can also as easily find rational digits of π , or even the irrational digits of π . The digit between the first 1 and 4 is $\tilde{\pi}^*\left(\frac{1}{2}\right) = 8$. The π th digit of π , apparently, is $\tilde{\pi}^*(\pi) = 0$.

But why stop at real numbers? With a simple modification, we can extend the domain to complex numbers as well. We take the length of the polynomial output instead and apply the standard transformation.

$$\tilde{\pi}_{\mathbb{C}}^*(x) = \lfloor |\tilde{\pi}(x)| \bmod 10 \rfloor$$

Then we see that the i -th digit of π is $\tilde{\pi}_{\mathbb{C}}^*(i) = 8$. Wow! Nature is beautiful.

Verification-Based Approaches

In this section, we devise a series of approximation algorithms that use two facts about the sine function, one familiar and one new:

- $\sin(\pi) = 0$
- For small ϵ , $\sin(\pi + \epsilon) \approx \epsilon$, and this approximation gets better as $\epsilon \rightarrow 0$.

We already know the first fact, and used it in our rootfinding methods. The second fact is less obvious, but it can be justified using the Maclaurin series expansion of $\sin(\pi + x)$ around $x = 0$:

$$\begin{aligned}
 \sin(\pi + x) &= \frac{\sin^{(0)}(\pi + 0)x^0}{0!} + \frac{\sin^{(1)}(\pi + 0)x^1}{1!} + o(x^2) \\
 &= 0 + \cos(\pi)x + o(x^2) \\
 &= x + o(x^2)
 \end{aligned}$$

This second fact seems rather uninteresting, but it is actually quite useful. It tells us that, for $x \approx \pi$, computing $\sin(x) = \sin(\pi + \epsilon) \approx \epsilon$ actually gives us a very close approximation of how far away x is from π . That is, if we have a *guess* of π , we can verify how good it is simply by computing its sine. Therefore, we can use this technique as a subroutine to determine whether some value is within a specified precision of π .

At this point, it may be obvious what the next step is: if we can determine whether a guess is good enough, why not just try some? BOGOFIND is the most direct application of this idea:

```

1: function BOGOFIND(N)
2:   repeat
3:      $\pi? \leftarrow$  uniformly random from  $[3, 4]$ 
4:   until  $|\sin(\pi?)| \leq 10^{-N+1}$ 
5:   return  $\pi?$ 
6: end function

```

As with many functions in the Bogo family, this unfortunately takes $O(\infty)$ in the worst case. We can do slightly better by trying a provably finite number of options:

```

1: function DETERMINISTICBOGOFIND(N)
2:   for  $d$  in  $0 \dots 10^N$  do
3:      $\pi? \leftarrow 3 + d \cdot 10^{-N}$ 
4:     if  $|\sin(\pi?)| \leq 10^{-N+1}$  then
5:       return  $\pi?$ 
6:     end if
7:   end for
8: end function

```

This is still suboptimal, taking $\Theta(10^N)$ time in the worst case. We can do better than that.

Most of the inefficiency of the above two algorithms is that they need to check multiple different guesses – a much more efficient approach would guess one number and just always be right the first time. To do exactly that, we take inspiration here from some prior work in utilizing the many-worlds interpretation of quantum mechanics for sorting.

```

1: function LUCKYGUESS(N)
2:    $x \leftarrow 3$ 
3:   for  $i$  in  $1 \dots N$  do
4:      $r \leftarrow$  quantum uniformly random selection from  $\{0, \dots, 9\}$ 
5:      $x \leftarrow x + r10^{-i}$ 
6:   end for
7:   Observe  $x$ 
8:   if  $|\sin(x)| \leq 10^{-N+1}$  then
9:     return  $x$ 
10:  else
11:    Destroy the universe
12:  end if
13: end function

```

After running, any remaining universes will contain an approximation accurate to $N \pm 1$ decimal places. This approach is much more efficient: this algorithm has the asymptotically optimal time complexity and space complexity of $\Theta(N)$. However, it does result in the destruction of $\Omega(10^N)$ universes, making it morally prohibitive in practice.

The Best Way To Compute π

Finally, we close with the objectively best way to compute π .

```
1: function COSMICGLORY(N)
2:   Allocate  $N$   $N$ -bit integers  $I[1 \dots N]$ 
3:   Sleep for 1 trillion years           // this is important
4:    $m \leftarrow$  New hash map
5:   for  $k$  in  $0 \dots N$  do
6:      $m[k] = 0$ 
7:   end for
8:   for  $i$  in  $I$  do
9:     Increment  $m \left[ \sum_{j=0}^{N-1} (i \gg j) \& 1 \right]$  by  $1/N$ 
10:  end for
11:  return  $(m[\sum_{k \in m} k/N \cdot m[k]])^{-2} / (2(\sum_{k \in m} k/N \cdot (m[k] - \sum_{k \in m} k/N \cdot m[k])^2)^2)$ 
12: end function
```

So... what's going on here?

First, we need to talk about cosmic rays.² This is the name given to high-energy particles or clusters of particles traveling close to the speed of light. Despite their name, cosmic rays aren't actually related to electromagnetic radiation, but rather are typically high-energy, high-speed protons or alpha particles. Cosmic rays are thought to come from a variety of sources, including solar flares, supernovas, and potentially even the absorption of matter by black holes. Occasionally, cosmic rays make their way into Earth's atmosphere. When they do, they typically interact with particles in the atmosphere and break apart, "cascading" into more fundamental particles, like protons, neutrons, and electrons, and later into subatomic particles, like pions and muons.

If a cosmic ray has enough energy as it enters the atmosphere, these resulting subatomic particles may still have a sizable amount of energy. You could even imagine, against all odds, one of these subatomic particles making it all the way down to sea level, quietly gliding through the wall of the building you're in right now, through the case of the device this PDF is being read on at this very moment, and bursting as it interacts with a silicon nucleus inside of the device. This burst could have enough energy to interact with a nearby transistor in memory, switching it from representing a 0 to representing a 1. *This* is the main driving force of COSMICGLORY.

These events (and others like them) are typically called *soft errors*. Most of the time, these are not good things (hence the word *error*), so extra physical protections are added to digital devices to mitigate them or redundancies are included in the case they occur. On the other hand, we *rely* on these events to occur in order to guarantee the correctness of this algorithm. We encourage any way possible to increase the number of likely cosmic ray interactions, such as leaving the device on the peak of a tall mountain or even putting it into low-Earth orbit, as long the result can eventually be accessed when the algorithm terminates.

To justify this algorithm's correctness, we first make an assumption about the distribution of cosmic ray bit-flips. In particular, we assume that the occurrence of cosmic ray bit-flips are a Poisson point process. That is, all bit-flips are independent from each other, and so in any given time period of a fixed length, the mean number of bit-flips is the same. This is a standard assumption for physical processes of this nature.^[citation needed]

Given this assumption, we can start to investigate how the algorithm works. Because bit-flips follow a Poisson point process, the total number of bits of one of the N -bit integers that have been flipped as a result of cosmic radiation after 1 trillion years follows a Poisson distribution with mean μ . In fact, it doesn't actually matter what μ is, and so we will not bother trying to determine its value. The only thing that matters is that it does not change over time. The first two loops populate the hash map m with this distribution – essentially, after the loops, m is a discrete probability density function which resembles a Poisson distribution.

²I am not a physicist and hereby release any responsibility of correctly explaining (or even understanding) complicated physics topics. But I tried my best. :)

As $N \rightarrow \infty$, there are more opportunities for bit-swaps, and in fact $\mu \rightarrow \infty$. This is useful to us – as it turns out, as $\mu \rightarrow \infty$, the Poisson distribution of mean μ approaches a binomial distribution of the same mean. However, we are also increasing the size of I – essentially, the number of samples – so this distribution is become less and less discrete-looking and more continuous-looking. In other words, it is approaching a Gaussian distribution. Experienced π -hunters may have figured out by now where we are going with this. If we observe the probability distribution function of the Gaussian distribution, we notice a familiar two-legged Greek character:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

We have this function (or at least an asymptotically close approximation), and so we can simply reverse engineer it to extract π . This is the purpose of the final line of the algorithm. Plugging in $x = \mu$ essentially removes the entire exponentiation expression, and we can reciprocate, square, and divide by $2\sigma^2$ to get π . To save space, I've shoved all of these steps into one line.

And there it is! As it turns out, you don't need circles to compute π . You just need a big pile of RAM, some basic statistics, and *a lot* of patience.

Acknowledgements

Thanks to Nick for coming up with this very entertaining assignment. Thanks to Ram for a useful discussion of many of these methods. Thanks to Lance for making sure my explanation of cosmic rays is at least mostly coherent. Thanks to ~~littlespoonfarm.com~~ grammy for the apple π e recipe. Finally, a special thanks to all of the dying stars in the corners of the vast universe, expelling immeasurably fast particles in their final moments to all of our benefit. You guys are the real MVPs.

Submitted by Ayden Coughlin on 17 de octubre de 2026.