Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Deutsch
فارسی
한국어
Edit links

Spigot algorithm

From Wikipedia, the free encyclopedia

A **spigot algorithm** is an algorithm for computing the value of a mathematical constant such as π or *e* which generates output digits left to right, with limited intermediate storage.

The name comes from a "spigot", meaning a tap or valve controlling the flow of a liquid.

Interest in such algorithms was spurred in the early days of computational mathematics by extreme constraints on memory, and an algorithm for calculating the digits of *e* appears in a paper by Sale in 1968.[1] The name "Spigot algorithm" appears to have been coined by Stanley Rabinowitz and Stan Wagon,[2] whose algorithm for calculating the digits of π is sometimes referred to as "*the* spigot algorithm for π".

The spigot algorithm of Rabinowitz and Wagon is *bounded*, in the sense that the number of required digits must be specified in advance. Jeremy Gibbons (2004)[3] uses the term "*streaming algorithm*" to mean one which can be run indefinitely, without a prior bound. A further refinement is an algorithm which can compute a single arbitrary digit, without first computing the preceding digits: an example is the Bailey-Borwein-Plouffe formula, a digit extraction algorithm for π which produces hexadecimal digits.

Example [edit]

This example illustrates the working of a spigot algorithm by calculating the binary digits of the natural logarithm of 2 (sequence A068426 in OEIS) using the identity

$$\ln(2) = \sum_{k=1}^{\infty} \frac{1}{k2^k}.$$

To start calculating binary digits from, say, the 8th place we multiply this identity by 2^7(since 7 = 8 - 1):

$$2^7 \ln(2) = 2^7 \sum_{k=1}^{\infty} \frac{1}{k2^k}.$$

We then divide the infinite sum into a "head", in which the exponents of 2 are greater than or equal to zero, and a "tail", in which the exponents of 2 are negative:

$$2^7 \ln(2) = \sum_{k=1}^{7} \frac{2^{7-k}}{k} + \sum_{k=8}^{\infty} \frac{1}{k2^{k-7}}.$$

We are only interested in the fractional part of this value, so we can replace each of the summands in the "head" by

$$\frac{2^{7-k} \mod k}{k}.$$

Calculating each of these terms and adding them to a running total where we again only keep the fractional part, we have:

k	$A = 2^{7-k}$	$B = A \bmod k$	$C = B / k$	Sum of *C* mod 1
1	64	0	0	0
2	32	0	0	0
3	16	1	1/3	1/3
4	8	0	0	1/3
5	4	4	4/5	2/15
6	2	2	1/3	7/15
7	1	1	1/7	64/105

We add a few terms in the "tail", noting that the error introduced by truncating the sum is less than the final term:

k	$D = 1/k2^{k-7}$	Sum of *D*	Maximum error

8	1/16	1/16	1/16
9	1/36	13/144	1/36
10	1/80	37/360	1/80

Adding the "head" and the first few terms of the "tail" together we get:

$$2^7 \ln(2) \mod 1 \approx \frac{64}{105} + \frac{37}{360} = 0.10011100\cdots_2 + 0.00011010\cdots_2 = 0.1011\cdots_2 ,$$

so the 8th to 11th binary digits in the binary expansion of ln(2) are 1, 0, 1, 1. Note that we have not calculated the values of the first seven binary digits – indeed, all information about them has been intentionally discarded by using modular arithmetic in the "head" sum.

The same approach can be used to calculate digits of the binary expansion of ln(2) starting from an arbitrary n^{th} position. The number of terms in the "head" sum increases linearly with n, but the complexity of each term only increases with the logarithm of n if an efficient method of modular exponentiation is used. The precision of calculations and intermediate results and the number of terms taken from the "tail" sum are all independent of n, and only depend on the number of binary digits that are being calculated – single precision arithmetic can be used to calculate around 12 binary digits, regardless of the starting position.

References [edit]

1. ^ Sale, AHJ (1968). "The calculation of *e* to many significant digits". *The Computer Journal* **11** (2): 229–230. doi:10.1093/comjnl/11.2.229. Retrieved 8 May 2013.
2. ^ Rabinowitz, Stanley; Wagon, Stan (1995). "A Spigot Algorithm for the Digits of Pi" (PDF). *American Mathematical Monthly* **102** (3): 195–203. doi:10.2307/2975006. Retrieved 8 May 2013.
3. ^ Gibbons, Jeremy (24 May 2004). "Unbounded Spigot Algorithms for the Digits of Pi" (PDF).

Further reading [edit]

- Arndt, Jorg; Haenel, Christoph, *π unleashed*, Springer Verlag, 2000.
- Weisstein, Eric W., "Spigot algorithm", *MathWorld*.

Categories: Computer arithmetic algorithms