PISPIGOT – Producing Digits of π one at a time

Abstract

PISPIGOT is a proof-of-concept BASIC program written in 1996 for the HP-71B pocket computer to produce an arbitrary number of digits of π one at a time using a so-called spigot algorithm.

Keywords: *spigot algorithm, Pi, π, one digit at a time, multiprecision, HP-71B, pocket computer, BASIC, Hewlett-Packard*

1. Introduction

PISPIGOT is a very short *(6 lines, 285 bytes) BASIC* program that I wrote in 1996 for the *HP-71B* pocket computer as a proof-of-concept example of implementing a *spigot algorithm* to produce an arbitrary number of digits of π one at a time. It will also run in other *HP BASIC* models with minimal changes (see ***Note 1*** below).

The algorithm produces N digits of π (up to many thousands, limited only by available memory and time) one by one using just integer arithmetic on reasonably small integer values. None of the previous digits are needed once computed and no floating-point operations are needed at all whether single-precision or high-precision.

On the other hand, the number N of digits needs to be specified in advance (so it's not possible to add more digits to the previously produced ones without having to restart the whole computation) and an array with some 10N/3 integer elements must be dimensioned to generate N digits, which is significantly more memory than needed by other multiprecision algorithms and also *much* slower. Thus, this algorithm and the resulting program featured here aren't intended to be competitive and are best considered as a proof-of-concept example.

The spigot algorithm is explained in detail in the reference given below. Basically, we start from the series:

$$\frac{\pi}{2} = \sum_{i=0}^{\infty} \frac{i!}{(2i+1)!!} = 1 + \frac{1}{3} + \frac{1.2}{3.5} + \frac{1.2.3}{3.5.7} + \ \ldots = 1 + \frac{1}{3}\left(1 + \frac{2}{5}\left(1 + \frac{3}{7}\left(1 + \frac{4}{9}\,(1 + \cdots)\right)\right)\right)$$

which can be considered a *mixed-radix base* (1/3, 2/5, 3/7, 4/9, ...) representation for π/2, so the digits of π itself in this base would all be **2**. The workings of the algorithm can be seen in the table below, taken from the reference. The array is initialized with the digits of π in the mixed-radix base (all 2) and then the operations described in the leftmost column are performed, starting from the rightmost column and going left, column by column and row by row, from top to bottom, producing one digit of π per row (**3**, **1**, **4**, **1**). We needed [10***4**/3] = 13 columns (array elements) to get **4** digits.

	Digits of π	1/3	2/5	3/7	4/9	5/11	6/13	7/15	8/17	9/19	10/21	11/23	12/25
Initialize		2	2	2	2	2	2	2	2	2	2	2	2
× 10		20	20	20	20	20	20	20	20	20	20	20	20
Carry	3	+10	+12	+12	+12	+10	+12	+7	+8	+9	+0	+0	+0
		30	32	32	32	30	32	27	28	29	20	20	20
Remainders		0	2	2	4	3	10	1	13	12	1	20	20
× 10		0	20	20	40	30	100	10	130	120	10	200	200
Carry	1	+13	+20	+33	+40	+65	+48	+98	+88	+72	+150	+132	+96
		13	40	53	80	95	148	108	218	192	160	332	296
Remainders		3	1	3	3	5	5	4	8	5	8	17	20
× 10		30	10	30	30	50	50	40	80	50	80	170	200
Carry	4	+11	+24	+30	+40	+40	+42	+63	+64	+90	+120	+88	+0
		41	34	60	70	90	92	103	144	140	200	258	200
Remainders		1	1	0	0	0	4	12	9	4	10	6	16
× 10		10	10	0	0	0	40	120	90	40	100	60	160
Carry	1	+4	+2	+9	+24	+55	+84	+63	+48	+72	+60	+66	+0
		14	12	9	24	55	124	183	138	112	160	126	160

When computing π with this program these are the *5 last digits*, *RAM* used and *running times* for *N* digits:

# Digits	***10***	***20***	***40***	***50***	***100***	***200***	***500***	***800***	***1000***
5 last digits	92653	32384	84197	93750*	17067	30381	19491	63185	20198
***~RAM** (bytes)*	280	380	580	680	1,180	2,180	5,180	8,180	10,180
***Time** (seconds)*	0.19	0.75	3.01	4.67	18.73	75.14	472	1,212	1,896

*(all running times in this paper are for the **go71b** emulator running at 128x on a mid-range Android **Samsung** tablet)*

The amount of *RAM* needed is approximately **10*N+180** bytes where **N** is the number of digits to compute so, for example, having 64 Kb of *RAM* available would allow the computation of up to ~ 6,500 digits.

2. Program Listing

```
1  DESTROY ALL @ Z$="0000000000" @ T$="9999999999" @ INPUT N @ L=10*N DIV 3 @ INTEGER A(L)
2   FOR I=1 TO L @ A(I)=2 @ NEXT I @ M=0 @ P=0 @ FOR J=1 TO N @ Q=0 @ K=2*L+1
3  FOR I=L TO 1 STEP -1 @ K=K-2 @ X=10*A(I)+Q*I @ A(I)=MOD(X,K) @ Q=X DIV K @ NEXT I
4  A(1)=MOD(Q,10) @ Q=Q DIV 10 @ IF Q=9 THEN M=M+1 @ GOTO 6
5  IF Q#10 THEN PRINT STR$(P);T$[1,M]; @ P=Q @ M=0 ELSE PRINT STR$(P+1);Z$[1,M]; @ P=0 @ M=0
6  NEXT J @ PRINT STR$(P)
```

3. Usage Instructions

See the worked examples below to understand how to use the program.

4. Examples

The following examples can be useful to check that the program is correctly entered and to understand its usage.

4.1 Example

Produce the first ***24 digits*** of π, then the first ***76 digits***.

```
RUN ?   24 END LINE   0 314159265358979323846264          (all digits are Ok, ~1")
RUN ?   76 END LINE   0 31415926535897932384626433832795028841
                        971693993751058209749445923078164062 86      (all Ok, ~11" )
```

4.2 Example

Produce the first ***1,000 digits*** of π.

```
RUN ? 1000 END LINE 0

 31415926535897932384626433832795028841971693993751 05820974944592307816406286208998628034825342117067
 98214808651328230664709384460955058223172535940812 84811174502841027019385211055596446229489549303819
 64428810975665933446128475648233786783165271201909 14564856692346034861045432664821339360726024914127
 37245870066063155881748815209209628292540917153643 67892590360011330530548820466521384146951941511609
 43305727036575959195309218611738193261179310511854 80744623799627495673518857527248912279381830119491
 29833673362440656643086021394946395224737190702179 86094370277053921717629317675238467481846766940513
 20005681271452635608277857713427577896091736371787 21468440901224953430146549585371050792279689258923
 54201995611212902196086403441815981362977477130996 05187072113499999983729780499510597317328160963185
 95024459455346908302642522308253344685035261931188 17101000313783875288658753320838142061717766914730
 35982534904287554687311595628638823537875937519577 81857780532171226806613001927876611195909216420198

                                        (all digits are Ok,  ~31')
```

Notes

1. This program will run in several other *HP BASIC* computers by simply *removing HP71B*-specific statements such as **`DESTROY ALL`**. If the particular *BASIC* version does not allow for dinamic (re)dimensioning, then array **`A`** must be dimensioned to have **`INT(10*N/3)`** elements, where **`N`** is the number of digits to compute. Furthermore, all variables used in the program (besides the **`A`** array) can be declared as *integer* for maximizing speed on some versions.

2. [*] Any digit-producing algorithm for normal numbers (which π probably is) has the (unlikely) problem that the last digit(s) may be incorrect if there's a terminating string of 9s, and even with no 9s the very last digit might be wrong, as when computing N=50 digits, where the last computed digit printed (underlined above) is a **0** but should actually be a **1**.

3. I also wrote a version for the ***SHARP PC-1350/PC-1360*** pocket computers and compatibles, see ***References***.

References

S. Rabinowitz and S. Wagonrancis (1995). *A Spigot Algorithm for the Digits of π*. *The American Mathematical Monthly, Vol. 102, No. 3*

Valentin Albillo (2019) *SHARP Program VA135 - PC-1350 Producing Digits of Pi one at a time*

Copyrights