[VA] SRC#006- Pi Day 2020 Special: A New Fast Way to Compute Pi

03-14-2020, 09:56 PM **Post: #1**

Valentin Albillo
Senior Member

Posts: 484
Joined: Feb 2015
Warning Level: 0%

[VA] SRC#006- Pi Day 2020 Special: A New Fast Way to Compute Pi

Hi all, welcome to my ***SRC#006- Pi Day 2020 Special***:

To commemorate **Pi Day 2020** I'll give and comment here a new, extremely fast way to compute *Pi* to arbitrary precision, millions or billions of digits if desired, but first a little background of some of my recent posts on the subject matter.

In the past I've posted here a couple of novel ways to compute **Pi**, quite unexpected but *very slow*. The first one, statistical in nature, was *(HP-71B BASIC code)*:

```
1   DESTROY ALL @ RANDOMIZE 1 @ FOR K=1 TO 5 @ N=10^K @ S=0
2   FOR I=1 TO N @ IF NOT MOD(IROUND(RND/RND),2) THEN S=S+1
3   NEXT I @ P=S/N @ STD @ DISP N, @ FIX 3 @ DISP 5-P*4 @ NEXT K

>RUN

10          3.000
100         3.360
1000        3.076
10000       3.125
100000      3.142
```

where the even/odd nature of the random-valued expression `IROUND(RND/RND)` is the key, no circles in sight at all despite what many believe to be always the case.

The second one (posted originally here) was also incredibly simple and this was my implementation for the *HP-71B* as a user-defined function:

```
1   DEF FNP(N)
2       T = N
3       FOR I = N-1 TO 2 STEP -1
4             T = CEIL(T/I)*I
5       NEXT I
6       FNP = N*N/T
7   END DEF
```

These are the function's values for *N = 10, 100, ..., 1000000:*

```
DESTROY ALL @ FIX 5

FNP(10)          -> 2.94118
FNP(100)         -> 3.09215
FNP(1000)        -> 3.13903
FNP(10000)       -> 3.14133
FNP(100000)      -> 3.14153
FNP(1000000)     -> 3.14159
```

The limit for *N -> Inf* is exactly ***Pi***, of course.

As stated above, these methods are quite unexpected, very rarely seen (if at all), and interesting from a purely academic point of view but utterly impractical for computing *Pi* as they are unbearably slow, producing just 5-6 digits of *Pi* after millions of iterations. Even producing as few as 10 digits is probably out of the question.

On the other hand, the method I'll introduce here is equally simple, perhaps even simpler in a way, but capable of producing millions, billions and even trillions of digits of *Pi* as fast or faster than other well-known methods, assuming of course the necessary hardware resources (fast processors, enough RAM) and multiprecision software are available.

For starters, here is the **HP42S** version, just 7 steps (12 bytes) of RPN code, particularized to carry out exactly 3 iterations of the algorithm and requiring no user input at all (but make sure to set `RAD` mode).

```
01    3
02  ENTER
03  LBL 00
04    SIN
05    RCL+ ST L
06    DSE ST Y
07  GTO 00
```

When run in **Free42**, which allows for 34-digit precision, it will instantly produce this 34-digit value of Pi:

```
[R/S] -> 3.14159265359
[SHOW]-> 3.1415926535897932384626433383279502
```

which is the correct value of *Pi* when truncated to 34 digits. Should **Free42** support more than 34 digits, say 100, 1000 or a million, this same code (trivially modified to perform more than 3 iterations) would produce the value of *Pi* to the maximum precision supported.

The algorithm used is extremely simple, and it consists of the following steps:

Step 1: Assign to **x** some approximation to *Pi*, say **x = 3** will do.

Step 2: Assign **x += sin(x)** . This new value will have at least 3 times as many correct digits of *Pi* as the previous value did.

Step 3: Repeat *Step 2* above until the desired precision is achieved (the number of iterations needed is easily computed in advance, see below).

Now you may be thinking something like: *"Hey, you're using a trigonometric function, namely sin(x), to help compute the value of Pi and that's circular [pun intended]. Come to that, you might simply compute Pi = 4*arctan(1), say."*

My reply is: first of all, *arctan* is an *inverse* function, namely the inverse of *tan(x)* which, as the inverses of *sin(x)* and *cos(x)* do, can produce *Pi* or rational fractions of *Pi* (say *Pi/2*, *Pi/4*, *Pi/3*, etc) for rational or algebraic values of their arguments. But *sin(x)* is *not* an inverse function like those and won't produce *Pi* or any fractions of *Pi* for any real rational or algebraic values of its argument. In other words, there's no rational or algebraic value *x* for which *sin(x) = Pi/4* or *Pi/6* or *2/3*Pi*, etc.

Secondly, what matters here is not *which* functions we do use but how *fast* can we implement them and how cleverly we can use them to compute *Pi* as fast as possible, so to be able to use this algorithm to compute *Pi* to millions or billions of digits, apart from suitable hardware we only need a fast *sin(x) custom* implementation, which only needs to be able to compute *sin(x)* for x in the very short range [3 .. Pi], say. *Pi* itself isn't needed in the implementation as no argument reduction is ever performed, which a general implementation of *sin(x)* would require. Further, our custom implementation can use a number of arithmetic tricks to reduce the argument to a suitably small value which might greatly speed the computation, say using the double-angle (half-angle) formulas three times in succession to reduce the [3 .. Pi] argument range to the still smaller [3/8 .. Pi/8] range.

The question is then: how fast can *sin(x)* be computed ? In the paper *"Fast Multiple-Precision Evaluation of Elementary Functions"* by Richard P. Brent (Australian National University, Canberra, Australia) it is shown that *sin(x)* (and some other elementary functions as well) can be evaluated with relative error $O(2^{-n})$, in $O(M(n)\log(n))$ operations as n tends to infinity, for any floating-point argument *x* in a suitable finite interval, where *M(n)* is the number of single-precision operations required to multiply *n*-bit integers.

Thus, **the main advantages of this algorithm** can be summarized like this:

1) It's extremely **simple**, each iteration requires the computation of just *one sin(x)* function and *one in-place addition*, with no other operations or variables involved. Futher, the starting value for the very first iteration is simply the constant **3**, no need to compute many-million-digit values for the starting parameters as other methods require.

2) It's extremely **fast**, the *sin(x)* can be performed in $O(M(n)\log(n))$ operations, which is about as fast as it gets, and it can be optimized to work for just a very short range of x.

3) It converges **cubically**, i.e., each iteration provides a value of *Pi* which has at least *3 times* as many correct digits as the previous iteration, so the number of digits obtained grows *exponentially* and thus computing millions, billions or trillions of digits requires very few iterations, namely:

13 iterations provide ~ 2 million digits
19 iterations provide ~ 1.4 billion (1e9) digits
25 iterations provide ~ 1 trillion (1e12) digits

4) It is *self-correcting*, if you're going to compute a million digits of *Pi* you don't need to use one-million-digit operations from the very first iteration. You can specify the level of precision increasingly for each iteration, namely 5 digit, 13-digit, 35-digit, 102-digit, 303-digit and 905-digit precision for iterations 1..6, respectively, which obviously increases speed enormously.

How does it compare with the best-known methods ? In the paper *"The Life of Pi: From Archimedes to Eniac and Beyond"* of Jonathan M. Borwein, Frsc, Faa, we find this quartic method described:

2. More about Complexity Reduction. To illustrate the stunning complexity reduction in the elliptic algorithms for Pi, let us write a *complete set of algebraic equations* approximating π to well over a trillion digits.

The number π is transcendental and the number $1/a_{20}$ computed next is algebraic nonetheless they coincide for over 1.5 trillion places.

Set $a_0 = 6 - 4\sqrt{2}$, $y_0 = \sqrt{2} - 1$ and then solve the following system:

$$y_1 = \frac{1 - \sqrt[4]{1 - y_0^4}}{1 + \sqrt[4]{1 - y_0^4}}, a_1 = a_0 (1 + y_1)^4 - 2^3 y_1 \left(1 + y_1 + y_1^2\right)$$
$$y_2 = \frac{1 - \sqrt[4]{1 - y_1^4}}{1 + \sqrt[4]{1 - y_1^4}}, a_2 = a_1 (1 + y_2)^4 - 2^5 y_2 \left(1 + y_2 + y_2^2\right)$$
$$y_3 = \frac{1 - \sqrt[4]{1 - y_2^4}}{1 + \sqrt[4]{1 - y_2^4}}, a_3 = a_2 (1 + y_3)^4 - 2^7 y_3 \left(1 + y_3 + y_3^2\right)$$
$$y_4 = \frac{1 - \sqrt[4]{1 - y_3^4}}{1 + \sqrt[4]{1 - y_3^4}}, a_4 = a_3 (1 + y_4)^4 - 2^9 y_4 \left(1 + y_4 + y_4^2\right)$$
$$y_5 = \frac{1 - \sqrt[4]{1 - y_4^4}}{1 + \sqrt[4]{1 - y_4^4}}, a_5 = a_4 (1 + y_5)^4 - 2^{11} y_5 \left(1 + y_5 + y_5^2\right)$$
$$y_6 = \frac{1 - \sqrt[4]{1 - y_5^4}}{1 + \sqrt[4]{1 - y_5^4}}, a_6 = a_5 (1 + y_6)^4 - 2^{13} y_6 \left(1 + y_6 + y_6^2\right)$$
$$y_7 = \frac{1 - \sqrt[4]{1 - y_6^4}}{1 + \sqrt[4]{1 - y_6^4}}, a_7 = a_6 (1 + y_7)^4 - 2^{15} y_7 \left(1 + y_7 + y_7^2\right)$$
$$y_8 = \frac{1 - \sqrt[4]{1 - y_7^4}}{1 + \sqrt[4]{1 - y_7^4}}, a_8 = a_7 (1 + y_8)^4 - 2^{17} y_8 \left(1 + y_8 + y_8^2\right)$$
$$y_9 = \frac{1 - \sqrt[4]{1 - y_8^4}}{1 + \sqrt[4]{1 - y_8^4}}, a_9 = a_8 (1 + y_9)^4 - 2^{19} y_9 \left(1 + y_9 + y_9^2\right)$$
$$y_{10} = \frac{1 - \sqrt[4]{1 - y_9^4}}{1 + \sqrt[4]{1 - y_9^4}}, a_{10} = a_9 (1 + y_{10})^4 - 2^{21} y_{10} \left(1 + y_{10} + y_{10}^2\right)$$

$$y_{11} = \frac{1 - \sqrt[4]{1 - y_{10}^4}}{1 + \sqrt[4]{1 - y_{10}^4}}, a_{11} = a_{10} (1 + y_{11})^4 - 2^{23} y_{11} \left(1 + y_{11} + y_{11}^2\right)$$
$$y_{12} = \frac{1 - \sqrt[4]{1 - y_{11}^4}}{1 + \sqrt[4]{1 - y_{11}^4}}, a_{12} = a_{11} (1 + y_{12})^4 - 2^{25} y_{12} \left(1 + y_{12} + y_{12}^2\right)$$
$$y_{13} = \frac{1 - \sqrt[4]{1 - y_{12}^4}}{1 + \sqrt[4]{1 - y_{12}^4}}, a_{13} = a_{12} (1 + y_{13})^4 - 2^{27} y_{13} \left(1 + y_{13} + y_{13}^2\right)$$
$$y_{14} = \frac{1 - \sqrt[4]{1 - y_{13}^4}}{1 + \sqrt[4]{1 - y_{13}^4}}, a_{14} = a_{13} (1 + y_{14})^4 - 2^{29} y_{14} \left(1 + y_{14} + y_{14}^2\right)$$
$$y_{15} = \frac{1 - \sqrt[4]{1 - y_{14}^4}}{1 + \sqrt[4]{1 - y_{14}^4}}, a_{15} = a_{14} (1 + y_{15})^4 - 2^{31} y_{15} \left(1 + y_{15} + y_{15}^2\right)$$
$$y_{16} = \frac{1 - \sqrt[4]{1 - y_{15}^4}}{1 + \sqrt[4]{1 - y_{15}^4}}, a_{16} = a_{15} (1 + y_{16})^4 - 2^{33} y_{16} \left(1 + y_{16} + y_{16}^2\right)$$
$$y_{17} = \frac{1 - \sqrt[4]{1 - y_{16}^4}}{1 + \sqrt[4]{1 - y_{16}^4}}, a_{17} = a_{16} (1 + y_{17})^4 - 2^{35} y_{17} \left(1 + y_{17} + y_{17}^2\right)$$
$$y_{18} = \frac{1 - \sqrt[4]{1 - y_{17}^4}}{1 + \sqrt[4]{1 - y_{17}^4}}, a_{18} = a_{17} (1 + y_{18})^4 - 2^{37} y_{18} \left(1 + y_{18} + y_{18}^2\right)$$
$$y_{19} = \frac{1 - \sqrt[4]{1 - y_{18}^4}}{1 + \sqrt[4]{1 - y_{18}^4}}, a_{19} = a_{18} (1 + y_{19})^4 - 2^{39} y_{19} \left(1 + y_{19} + y_{19}^2\right)$$
$$y_{20} = \frac{1 - \sqrt[4]{1 - y_{19}^4}}{1 + \sqrt[4]{1 - y_{19}^4}}, \mathbf{a_{20}} = a_{19} (1 + y_{20})^4 - 2^{41} y_{20} \left(1 + y_{20} + y_{20}^2\right).$$

This quartic algorithm, with the Salamin–Brent scheme, was first used by Bailey in 1986 [15] and was used repeatedly by Yasumasa Kanada, see Figure 12, in Tokyo in computations of π over 15 years or so, culminating in a 200 billion decimal digit computation in 1999. As recorded in Figure 11, it has been used twice very recently by Takahashi. Only thirty five years earlier in 1963, Dan Shanks — a very knowledgeable participant — was

Being quartic, this method requires less iterations to achieve a given number of digits, namely 20 iterations to provide a trillion (1e12) digits vs. 25 iterations for the cubic method described here, 25 iterations to provide a quadrillion (1 e15) digits vs. 31, and 30 iterations to provide a quintillion (1e18 digits) vs. 37. As can be seen, there's no large increase in the number of iterations for practically feasible computations even as large as a quintillion digits so the speed advantage of being a quartic method instead of a cubic one is not crucial.

Also, the method here requires just 1 *sin* and 1 addition per iteration and just one variable, *x*. On the other hand, Borwein's method requires about 2 raising-to-the-4th power operations, 3 multiplications, 1 squaring, 3 subtractions, 4 additions, 1 division, 1 fourth-root, and at least storing 3 intermediate values, all of it per iteration. Further *all* operations must be performed at *full* precision from the very first one, plus computing the two irrational initial values to full precision as well before even starting.

This means that, in practice and for up to at least a quadrillion/quintillion digits, the method described here could be significantly *faster* if a suitably optimized, fast custom version of *sin(x)* is used. Now let's see the method in action by conducting an *interactive* session using a multiprecision environment. We'll use the environment's native built-in *sin(x)* as we aren't interested here in speed but in showing the accuracy obtained for an increasing number of iterations:

```
> X=3
```

3 *(starting value, **1** correct digit)*

```
> X+=sin(X)
```

3.141120008... *(1st iteration, **4** correct digits)*

```
> X+=sin(X)
```

3.1415926535721955... *(2nd iteration, **11** correct digits)*

```
> X+=sin(X)
```

3.1415926535897932384626433832795019759... *(3rd iteration, **33** correct digits)*

```
> X+=sin(X)
```

3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706785726... *(4th iteration, **100** correct digits)*

```
> X+=sin(X)
```

3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706798214808651328230664709384460955058223172535940812848111745028410270193852110555964462294895493038196442881097566593344612847564823378678316527120190914564856692346034861045432664821339360726024914127340000...
*(5th iteration, **301** correct digits)*

```
> X+=sin(X)
```

3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706798214808651328230664709384460955058223172535940812848111745028410270193852110555964462294895493038196442881097566593344612847564823378678316527120190914564856692346034861045432664821339360726024914127372458700660631558817488152092096282925409171536436789259036001133053054882046652138414695194151609433057270365759591953092186117381932611793105118548074462379962749567351885752724891227938183011949129833673362440656643086021394946395224737190702179860943027705392171762931767523846748184676694051320005681271452635608277857713427577896091736371787214684409012249534301465495853710507922796892589235420199561121290219608640344181598136297747713099605187072113499999983729780499510597317328160963185950244594553469083026425223082533446850352619311881710100031378387528865875332083814206171776691473035925539 72... *(6th iteration, **903** correct digits)*

and so on and so forth.

Happy *Pi* Day !

V.

Find All My HP-related Materials here: **Valentin Albillo's HP Collection**

PM WWW FIND EDIT QUOTE REPORT

03-15-2020, 01:29 AM **Post: #2**

cdmackay
Senior Member

Posts: 482
Joined: Sep 2018

RE: [VA] SRC#006- Pi Day 2020 Special: A New Fast Way to Compute Pi

wonderful! thank you Valentin.

Cambridge, UK
41CL/DM41X 12/15C/16C DM15/16 71B 17B/BII/bII+ 28S 42S/DM42 48GX 50g 35s 30b/WP34S Prime G2 & Casios, Rockwell 18R :)

PM FIND QUOTE REPORT

03-15-2020, 12:54 PM **Post: #3**

Posts: 372

J-F Garnier
Senior Member
Joined: Dec 2013

RE: [VA] SRC#006- Pi Day 2020 Special: A New Fast Way to Compute Pi

Hi Valentin,

It's a pleasure to read you again in the forum!

Your iterative algorithm is very clever, and is (as always with your contributions) the opportunity to think and learn new things.

So basically, your iterative algorithm solves X=X+SIN(X) around X=3, that is SIN(X)=0 and the solution is of course X=PI.
Now, let me discuss your claim that it is an efficient way to calculate PI:

You anticipated the objections
*> Now you may be thinking something like: "Hey, you're using a trigonometric function, namely sin(x), to help compute the value of Pi and that's circular [pun intended]. Come to that, you might simply compute Pi = 4*arctan(1), say."*
and you justified your claim with:
>we only need a fast sin(x) custom implementation, which only needs to be able to compute sin(x) for x in the very short range [3 .. Pi], say. Pi itself isn't needed in the implementation as no argument reduction is ever performed, which a general implementation of sin(x) would require.

In the algorithms used by HP (and probably by others, but I know better the HP ones), there is an explicit argument reduction that used a stored value of PI. In the Saturn implementation, the PI value used for argument reduction is stored on 31 digits, as you known, and this is that permits to calculate SIN(3.14159265358) accurately as 9.79323846264E-12 (the next 12 digits of PI).

The fundamental question is: is it possible to efficiently calculate the SIN(X) function around X=PI, actually closer and closer to PI at each iteration, without using argument reduction (that would require PI itself).
You mentioned a reference to an article that states that sin(x) can be performed in O(M(n)log(n)) operations, but doesn't it assume explicitly or implicitly that the argument is reduced to a small range such as [0..PI/4], or if not that the PI value is known with the same n-digits precision?

So here is the challenge I would like to propose:
write an efficient program that calculates, for instance, the 2nd iteration starting from X=3+SIN(3)=3.141120008... and returns the result accurate to let's say 11 digits as: 3.1415926535(7)
without using the PI value itself (so don't use trig operations).
Here is my attempt on the 71B, based on the sin expansion series with a few refinements: the terms are stored in an array, then summed with 15-digit accuracy using the Math ROM DOT operation, starting from the smallest terms.
Even with this refinements, I had to sum terms of the sin(x) expansion up to x^21 (11 terms).

```
10 !
20 OPTION BASE 1
30 N=100
40 DIM A(N),B(N)
60 X=3 @ X=X+SIN(X)
70 S=0
80 X1=1
90 X2=X*X
100 MAT A=ZER
110 K=N
120 FOR I=1 TO 21 STEP 2
130 A(K)=X1/FACT(I)
140 K=K-1
150 X1=-X1*X2
160 NEXT I
170 MAT B=(X)
180 S=DOT(A,B)
190 DISP X;S;SIN(X)
200 DISP X+S

> RUN
3.14112000806 4.7264551803E-4 4.72645512196E-4
3.14159265358
```

J-F

Bernd Grubert
Member

Posts: 80
Joined: Dec 2013

RE: [VA] SRC#006- Pi Day 2020 Special: A New Fast Way to Compute Pi

Hello Valentin,
Thanks for the interesting way to calculate pi.

My HP 50G says, it could be interpreted as an application of the newton iteration method to the function:

f(x) = c * sqrt((cos(x) + 1) / (cos(x) - 1))

This is the solution of y(x) / y'(x) = -sin(x).
The Newton iteration is then:

x_n+1 = x_n - y(x_n) / y'(x_n) = x_n + sin(x_n)

with x_0 = 3.

Is that correct?

May be there are functions with an even faster convergence.

Kind regards
Bernd

Edit: Corrections and additional explanations.

EMAIL PM FIND QUOTE REPORT

03-15-2020, 01:35 PM **Post: #5**

EdS2
Member

Posts: 171
Joined: Apr 2014

RE: [VA] SRC#006- Pi Day 2020 Special: A New Fast Way to Compute Pi

Very nice! I like the idea that sin() isn't actually too computationally expensive.

(For the rnd/rnd method I found I had to adjust the program - is there a typo in there? I adjusted it to print 1+4*s/n)

EMAIL PM FIND QUOTE REPORT

03-15-2020, 04:29 PM **Post: #6**

Albert Chan
Senior Member

Posts: 852
Joined: Jul 2018

RE: [VA] SRC#006- Pi Day 2020 Special: A New Fast Way to Compute Pi

> **Bernd Grubert Wrote:** ➡ (03-15-2020 12:55 PM)
> My HP 50G says, it could be interpreted as an application of the newton iteration method to the function:
>
> f(x) = c * sqrt((cos(x) + 1) / (cos(x) - 1))
>
> This is the solution of y(x) / y'(x) = -sin(x).
> The Newton iteration is then:
>
> x_n+1 = x_n - y(x_n) / y'(x_n) = x_n + sin(x_n)
>
> with x_0 = 3.
>
> Is that correct?
>
> May be there are functions with an even faster convergence.

This is not how iteration X+=SIN(X) is derived.

With guess X=3, X+ASIN(SIN(X)) = X+ASIN(SIN(PI-X)) = X+(PI-X) = PI

ASIN(ε) = ε + ε^3/6 + 3ε^5/40 + ... = ε + O(ε^3)

Thus, with guess X=3, X+=SIN(X), converge to Pi cubically.

We can improve convergence rate with better estimation for ASIN
Say, with Pade[2,2] of ASIN(ε) = ε/(1-ε*ε/6)

> X=3
> Y=SIN(X)
> X+Y
3.14112000806
> X+Y/(1-Y*Y/6)
3.14158996537

EMAIL PM FIND QUOTE REPORT

03-15-2020, 04:42 PM **Post: #7**

Albert Chan
Senior Member

Posts: 852
Joined: Jul 2018

RE: [VA] SRC#006- Pi Day 2020 Special: A New Fast Way to Compute Pi

J-F Garnier Wrote: ⇨ (03-15-2020 12:54 PM)

So here is the challenge I would like to propose:
write an efficient program that calculates, for instance, the 2nd iteration starting from X=3+SIN(3)=3.141120008...
and returns the result accurate to let's say 11 digits as: 3.1415926535(7)
without using the PI value itself (so don't use trig operations).

Below use multiple angle formula (4 times), $\sin(5x) = 16\sin^5 x - 20\sin^3 x + 5\sin x$

Code:

```
10 FNM(X)=X*(5-X*X*(4+16*(1-X)*(1+X)))
20 FNT(X)=X*(1-X*X/6*(1-X*X/20))
30 FNS(X)=FNM(FNM(FNM(FNM(FNT(X/625)))))
40 FNP(X)=X+FNS(X)
```

>FNP(3)
3.14112000806
>FNP(RES)
3.14159265358

EMAIL PM FIND QUOTE REPORT

03-15-2020, 04:54 PM **Post: #8**

toml_12953
Senior Member

Posts: 1,346
Joined: Dec 2013

RE: [VA] SRC#006- Pi Day 2020 Special: A New Fast Way to Compute Pi

Valentin Albillo Wrote: ⇨ (03-14-2020 09:56 PM)

Hi all, welcome to my ***SRC#006- Pi Day 2020 Special***:

Is there a way to calculate individual digits of PI without the use of an array? I've seen methods that use arrays but they limit you to the maximum size of an array in a particular language.

Tom L
...other than that, Mrs. Lincoln, what did you think of the play?

EMAIL PM FIND QUOTE REPORT

03-15-2020, 05:21 PM (This post was last modified: 03-15-2020 05:24 PM by J-F Garnier.) **Post: #9**

J-F Garnier
Senior Member

Posts: 372
Joined: Dec 2013

RE: [VA] SRC#006- Pi Day 2020 Special: A New Fast Way to Compute Pi

Albert Chan Wrote: ⇨ (03-15-2020 04:42 PM)

Below use multiple angle formula (4 times), $\sin(5x) = 16\sin^5 x - 20\sin^3 x + 5\sin x$

Code:

```
10 FNM(X)=X*(5-X*X*(4+16*(1-X)*(1+X)))
20 FNT(X)=X*(1-X*X/6*(1-X*X/20))
```

```
30 FNS(X)=FNM(FNM(FNM(FNM(FNT(X/625)))))
40 FNP(X)=X+FNS(X)
```

>FNP(3)
3.14112000806
>FNP(RES)
3.14159265358

Nice and compact solution !

I needed to calculate 11 terms in the form u(n)=x^(2n+1)/(2n+1)!
each can be computed from the previous one by u(n) = u(n-1) * (x^2) / (2n*(2n+1)) that is 2 multiplications plus one division (I didn't optimized the factorial in my proposed solution). Total 33 mult/division operations.
Your solution needs 5 multiplications for FNM(X), called 4 times, plus the 4 multiplications and some divisions in FNT(X).

J-F

EMAIL PM WWW FIND QUOTE REPORT

03-15-2020, 08:08 PM **Post: #10**

Albert Chan
Senior Member

Posts: 852
Joined: Jul 2018

RE: [VA] SRC#006- Pi Day 2020 Special: A New Fast Way to Compute Pi

Slightly optimized version, using less operations.

Code:

```
10 FNM(X)=X*(5-X*X*(4+16*(1-X)*(1+X)))
20 FNT(X)=X*(1-X*X*(20-X*X)/120)
30 FNS(X)=FNM(FNM(FNM(FNT(.008*X))))
40 FNP(X)=X+FNS(X)
```

>FNP(3)
3.14112000806
>FNP(RES)
3.14159265358

EMAIL PM FIND QUOTE REPORT

03-16-2020, 06:12 AM **Post: #11**

ttw
Member

Posts: 189
Joined: Jun 2014

RE: [VA] SRC#006- Pi Day 2020 Special: A New Fast Way to Compute Pi

One problem with fast generation of the digits of Pi (or whatever) is that almost all the time is taken in the last step. Going through a quadratically convergent method to a quartically versions doesn't gain as much as one wants. The last step still uses full length arithmetic go get the whole digit string. (Assuming that one uses short approximations in the early stages as suggested above.)

EMAIL PM FIND QUOTE REPORT

03-16-2020, 11:34 AM (This post was last modified: 03-16-2020 11:35 AM by EdS2.) **Post: #12**

EdS2
Member

Posts: 171
Joined: Apr 2014

RE: [VA] SRC#006- Pi Day 2020 Special: A New Fast Way to Compute Pi

> **ttw Wrote:** ⇨ (03-16-2020 06:12 AM)
> One problem with fast generation of the digits of Pi (or whatever) is that almost all the time is taken in the last step...

True for some methods, but there are spigot methods with the opposite property: they produce the digits successively, with the first digits coming slowly and the final digits coming very quickly. There's no full-width calculation other than the propagation of carries.

See http://www.pi314.net/eng/goutte.php and the Gibbons paper too:
https://www.cs.ox.ac.uk/jeremy.gibbons/p...spigot.pdf

Here's a video with links in the description:
https://www.youtube.com/watch?v=3KXGFtDXOc8

EMAIL PM FIND QUOTE REPORT

J-F Garnier
Senior Member

Posts: 372
Joined: Dec 2013

RE: [VA] SRC#006- Pi Day 2020 Special: A New Fast Way to Compute Pi

Albert Chan Wrote: ⇨ (03-15-2020 08:08 PM)

Slightly optimized version, using less operations.

Code:

```
10 FNM(X)=X*(5-X*X*(4+16*(1-X)*(1+X)))
20 FNT(X)=X*(1-X*X*(20-X*X)/120)
30 FNS(X)=FNM(FNM(FNM(FNT(.008*X))))
40 FNP(X)=X+FNS(X)
```

>FNP(3)
3.14112000806
>FNP(RES)
3.14159265358

Ok, now let's try the next iteration, on the 35-digits Decimal Free42.
The program is easily ported to RPN, thanks to Albert's structured code:

Code:

```
# FNM(X)=X*(5-X*X*(4+16*(1-X)*(1+X)))
lbl fnm
1  rcl y  -
1  lastx  +  *
16  *
4  +
rcl y  x^2  *
5  x<>y  -
*
rtn
```

3
xeq fnp
3,141120008059...
xeq fnp
3,14159265357...
xeq fnp
3,141592653589[636096612966527996358]

13 correct digits. Not accurate enough.

Let's try with 5 calls to fnm:

Code:

```
lbl fns
625 /
xeq fnt
xeq fnm
xeq fnm
xeq fnm
xeq fnm
rtn
```

3
xeq fnp
xeq fnp
xeq fnp
3,1415926535897932[28408529447963301]

Still not accurate enough.

Let's try to add 1 term in the sin expansion:

Code:

```
# FNT(X)=X*(1-X*X/6*(1-X*X/20*(1-X*X/42)))
lbl fnt
rcl x  x^2
rcl x  42  /
1  x<>y  -
rcl y  *  20  /
1  x<>y  -
*
6  /
1  x<>y  -
*
```

3
xeq fnp
xeq fnp
xeq fnp
3,14159265358979323846264[6911462229]

24 correct digits. Still not accurate enough.

Let's try to add another term in the sin expansion:

Code:

```
# FNT(X)=X*(1-X*X/6*(1-X*X/20*(1-X*X/42*(1-X*X/72) )))
lbl fnt
rcl x  x^2
rcl x  72  /
1  x<>y  -
rcl y  *  42  /
1  x<>y  -
rcl y  *  20  /
1  x<>y  -
*
6  /
```

3
xeq fnp
xeq fnp
xeq fnp
3,14159265358979323846264338327[8692]

30 correct digits. Let's stop here.

Total 9 + 5*5 = 34 mult/div operations, with optimizations (avoiding to calculate x*x several times)

J-F

EMAIL PM WWW FIND QUOTE REPORT

03-17-2020, 09:43 PM **Post: #14**

Valentin Albillo
Senior Member

Posts: 484
Joined: Feb 2015
Warning Level: 0%

RE: [VA] SRC#006- Pi Day 2020 Special: A New Fast Way to Compute Pi

.
Hi, all:

Thanks for your interest in my ***SRC#006***, much appreciated. As for fast computation of *sin(x)*, these papers and link are interesting and relevant:

E. Karatsuba. "Fast evaluation of transcendental functions (1991)"
B. Haible & T. Papanikolaou. "Fast multiprecision evaluation of series of rational numbers"
Binary Splitting

Remember that no general implementation of *sin(x)* is required, just an optimized one for suitably small *x* in a very short interval.

EdS2 Wrote:

For the rnd/rnd method I found I had to adjust the program - **is there a typo in there?**

No, there isn't. I always check the code I post by keying it in anew from the [Preview Post] listing and running it, so that I can check the results before hitting [Post]. I did that once again and it runs fine so no typo. I suggest you check what you keyed in, if there's a typo it certainly is on your side.

Also, I saw you mentioned my fast program here ...

https://stardot.org.uk/forums/viewtopic....2&p=262553

... but you modified the code to start from X_0=1 as the first approximation, so it takes longer to converge. You shouldn't do that just because you feel like it, you should leave my original code alone and start from X_0=3. The fact that it also converges when starting at X_0=1 (albeit taking extra unnecessary iterations) is ancillary, the correct starting value X_0 must be such that Abs(Pi-X_0) < 1, and the simplest such value is X_0 = 3. Arbitrarily chaging it to 1, as you did, does not meet the condition.

In the future, I would appreciate it if, when giving me credit for some code I wrote, you don't modify my code in any way (apart from adapting it to some particular programming language) without clearly stating that you modified my original code and what modifications you made (in this particular case, changing the starting value, which I don't approve of).

Thanks in advance.

Regards.
V.

Find All My HP-related Materials here: **Valentin Albillo's HP Collection**

PM WWW FIND EDIT QUOTE REPORT

03-18-2020, 04:03 AM **Post: #15**

ttw
Member

Posts: 189
Joined: Jun 2014

RE: [VA] SRC#006- Pi Day 2020 Special: A New Fast Way to Compute Pi

Spigot methods are nice but generally give results in a fixed base. I'll check the literature to see if there is a spigot method for continued fractions.

EMAIL PM FIND QUOTE REPORT

03-18-2020, 11:51 AM **Post: #16**

EdS2
Member

Posts: 171
Joined: Apr 2014

RE: [VA] SRC#006- Pi Day 2020 Special: A New Fast Way to Compute Pi

.

> **Valentin Albillo Wrote:** ➡ (03-17-2020 09:43 PM)
> As for fast computation of sin(x), these papers and link are interesting and relevant:

Thanks for the pointers!

> **Quote:**
> I saw you mentioned my fast program here ...
> https://stardot.org.uk/forums/viewtopic....2&p=262553
>
> ... but you modified the code ...
>
> In the future, I would appreciate it if, when giving me credit for some code I wrote, you don't modify my code in any way (apart from adapting it to some particular programming language) without clearly stating that you modified my original code...

Understood, apologies, and I've added a note to that post.

It seems to me that any strictly positive starting value less than pi will converge to pi - is that not so?

About the wonderful and unexpected rnd/rnd code:

> **Quote:**
> I suggest you check what you keyed in, if there's a typo it certainly is on your side.

Hmm. I must be missing something. Here's your posted code:

Code:

```
1   DESTROY ALL @ RANDOMIZE 1 @ FOR K=1 TO 5 @ N=10^K @ S=0
2   FOR I=1 TO N @ IF NOT MOD(IROUND(RND/RND),2) THEN S=S+1
3   NEXT I @ P=S/N @ STD @ DISP N, @ FIX 3 @ DISP 5-P*4 @ NEXT K
```

previously posted in this slightly different form:

Code:

```
10 INPUT K @ N=0 @ FOR I=1 TO K @ N=N-MOD(IROUND(RND/RND),2) @ NEXT I @ DISP 1-4*N/K
```

and here's mine, in BBC Basic:

Code:

```
10 FOR K=1 TO 7
20 N%=10^K
30 S%=0
40 FOR I=1 TO N%
50 S%=S%+INT(RND(1)/RND(1)+.5)MOD 2
60 NEXT
70 PRINT N%,1+4*S%/N%
80 NEXT
```

Evidently something is somehow slightly different.

EMAIL PM FIND QUOTE REPORT

03-19-2020, 05:58 AM **Post: #17**

Valentin Albillo
Senior Member

Posts: 484
Joined: Feb 2015
Warning Level: 0%

RE: [VA] SRC#006- Pi Day 2020 Special: A New Fast Way to Compute Pi

.
Hi, **EdS2**:

EdS2 Wrote: ➡ (03-18-2020 11:51 AM)

Thanks for the pointers!

You're welcome. I hope you'll find them interesting.

Quote:

Understood, apologies, and I've added a note to that post.

Thanks. Let bygones be bygones.

Quote:

It seems to me that any strictly positive starting value less than pi will converge to pi - is that not so?

Yes, and not only for such values less than *Pi* but also for values up to (but not including) *2*Pi*. However, as the strictly positive starting value approaches 0, the number of iterations required to approximate *Pi* does increase *(logarithmically)* without limit, as can be easily seen this way: consider that for some *epsilon > 0* we have *eps+sin(eps) ~ eps + eps = 2*eps* as the next value, which means that every iteration the initial *eps* value is approximately *doubling*, i.e., growing *exponentially*, so in a *logarithmic* number of iterations it will reach "macro" size (say 0.1, 0.2 ...) and afterwards it will converge cubically to *Pi*.

Quote:

About the wonderful and unexpected rnd/rnd code: [...] Hmm. I must be missing something. Here's your posted code:

1 DESTROY ALL @ RANDOMIZE 1 @ FOR K=1 TO 5 @ N=10^K @ S=0
2 FOR I=1 TO N @ IF NOT MOD(IROUND(RND/RND),2) THEN S=S+1
3 NEXT I @ P=S/N @ STD @ DISP N, @ FIX 3 @ DISP 5-P*4 @ NEXT K
[...]
and here's mine, in BBC Basic:

10 FOR K=1 TO 7
20 N%=10^K
30 S%=0

```
40 FOR I=1 TO N%
50 S%=S%+INT(RND(1)/RND(1)+.5)MOD 2
60 NEXT
70 PRINT N%,1+4*S%/N%
80 NEXT
```

Evidently something is somehow slightly different.

There are two possibilities. I'm not familar with *BBC BASIC* but it might be the case that integer variables (%) can only hold from -32768 to +32767, so 10^5, 10^6 and 10^7 wouldn't fit in *N%*. However, I assume that's not the case or you'd noticed pretty quickly.

The second possibility is that you're not implementing the **IROUND** function correctly. I see you're implementing **IROUND(x)** as **INT(x+.5)** but that's not how **IROUND** works, which actually depends on the current **OPTION ROUND** setting, namely: **OPTION ROUND NEAR/ZERO/POS/NEG**.

My code assumes the *default* setting, **OPTION ROUND NEAR**, which causes **IROUND(x)** to round *x* to the nearest integer and, in case of a tie, to the even value. Thus we have:

```
IROUND(4.1) = 4 (4 is nearest)
IROUND(4.6) = 5 (5 is nearest)
IROUND(4.5) = 4 (both 4 and 5 are equally near, but 4 is even)
IROUND(5.5) = 6 (both 5 and 6 are equally near, but 6 is even)
```

This may or may not explain the difference but I have a question for you: have you entered my unmodified posted code in a physical or emulated *HP-71B* ? Does it produce the posted results ? If not, could you post the listing and the results you get ?

Regards.
V.

Find All My HP-related Materials here: **Valentin Albillo's HP Collection**

PM WWW FIND EDIT QUOTE REPORT

03-20-2020, 06:08 PM **Post: #18**

Bernd Grubert
Member

Posts: 80
Joined: Dec 2013

RE: [VA] SRC#006- Pi Day 2020 Special: A New Fast Way to Compute Pi

Albert Chan Wrote: ⇨ (03-15-2020 04:29 PM)

This is not how iteration X+=SIN(X) is derived.

With guess X=3, X+ASIN(SIN(X)) = X+ASIN(SIN(PI-X)) = X+(PI-X) = PI

ASIN(ε) = ε + ε^3/6 + 3ε^5/40 + ... = ε + O(ε^3)

Thus, with guess X=3, X+=SIN(X), converge to Pi cubically.

We can improve convergence rate with better estimation for ASIN
Say, with Pade[2,2] of ASIN(ε) = ε/(1-ε*ε/6)

```
> X=3
> Y=SIN(X)
> X+Y
3.14112000806
> X+Y/(1-Y*Y/6)
3.14158996537
```

Thanks for the explanation. I was on the wrong track.

EMAIL PM FIND QUOTE REPORT

03-21-2020, 09:21 PM **Post: #19**

EdS2
Member

Posts: 171
Joined: Apr 2014

RE: [VA] SRC#006- Pi Day 2020 Special: A New Fast Way to Compute Pi

.
Hi Valentin

> **Valentin Albillo Wrote:** ⇒ (03-19-2020 05:58 AM)
>
> > **Quote:**
> > It seems to me that any strictly positive starting value less than pi will converge to pi - is that not so?
>
> Yes[...] However, as the strictly positive starting value approaches 0, the number of iterations required to approximate *Pi* does increase *(logarithmically)* [...]and afterwards it will converge cubically to *Pi*.

Ah yes, good point.

> **Quote:**
>
> > **Quote:**
> > About the wonderful and unexpected rnd/rnd code: [...] Hmm. I must be missing something...
>
> There are two possibilities [...]

Thanks for the thoughts, but I think it's something else.

> **Quote:**
> This may or may not explain the difference but I have a question for you: have you entered my unmodified posted code in a physical or emulated *HP-71B* ? Does it produce the posted results ? If not, could you post the listing and the results you get ?

I hadn't, but now I have! (I even found how to run at full speed.) And indeed, of course, your two programs both produce the hoped-for results. No typo. There remains the question of my understanding, which is still missing a piece.

What's very curious to me is that you have two quite different final expressions in the two versions of the HP-71B rnd/rnd programs that you've posted. In the one-liner, you use the same form as my Basic port - you counted negatively and therefore perform a subtraction instead of an addition, but otherwise the same. In the multi-line version which you posted most recently, the final expression is quite different - and I don't (yet) understand why it should be.

Your earlier program: 1-4*N/K
My Basic port: 1+4*S%/N%
Your recent program: 5-P*4

Cheers
Ed

EMAIL PM FIND QUOTE REPORT

03-23-2020, 01:09 AM **Post: #20**

Valentin Albillo
Senior Member

Posts: 484
Joined: Feb 2015
Warning Level: 0%

RE: [VA] SRC#006- Pi Day 2020 Special: A New Fast Way to Compute Pi

.
Hi, **EdS2**:

> **EdS2 Wrote:** ⇒ (03-21-2020 09:21 PM)
> I hadn't, but now I have! (I even found how to run at full speed.) And indeed, of course, **your two programs both produce the hoped-for results. No typo**.

Of course indeed, I told you as much. 

> **Quote:**
> There remains the question of my understanding, which is still missing a piece.

Let's see ...

> **Quote:**
> What's very curious to me is that you have two quite different final expressions in the two versions of the HP-71B rnd/rnd programs that you've posted.

I didn't remember the exact earlier version, the one-liner I posted many years ago, but as I know the underlying theory I simply coded it again from scratch and came up with the second version, the 3-liner, which of course produces the exact same results. That explains why I posted two different versions.

> **Quote:**
> In the one-liner, you use the same form as my Basic port - you counted negatively and therefore perform a subtraction instead of an addition, but otherwise the same. In the multi-line version which you posted most recently, the final expression is quite different - and I don't (yet) understand why it should be.
>
> Your earlier program: 1-4*N/K
> My Basic port: 1+4*S%/N%
> Your recent program: 5-P*4

It's quite simple, actually. My recent program is this:

```
1   DESTROY ALL @ RANDOMIZE 1 @ FOR K=1 TO 5 @ N=10^K @ S=0
2   FOR I=1 TO N @ IF NOT MOD(IROUND(RND/RND),2) THEN S=S+1
3   NEXT I @ P=S/N @ STD @ DISP N, @ FIX 3 @ DISP 5-P*4 @ NEXT K
```

which is computing the probability that the closest integer to *A/B* is **even**, where *A* and *B* are uniformly distributed random numbers in [0,1), as produced by the **RND** keyword. Each time the **rounded** value is **even** (i.e., *it's 0 modulo* **2**) the number of favorable outcomes (**S**) is *incremented* by one (see line 2). After **N** tries have been sampled, the probability **P** for the **even** case will be the number of favorables outcomes (**S**) divided by the number of tries (**N**), thus we have the estimated probability **P = S/N**.

But I know from theory that in the limit, for N -> Infinity, the exact probability **P = (5-Pi)/4**, so isolating *Pi* we have **Pi = 5-P*4**, which is displayed by the program in line 3 above.

Now, my earlier program, the one-liner, namely:

```
10   INPUT K @ N=0 @ FOR I=1 TO K @ N=N-MOD(IROUND(RND/RND),2) @ NEXT I @ DISP 1-4*N/K
```

is computing the probability that the closest integer to *A/B* is **odd**, where *A* and *B* are uniformly distributed random numbers in [0,1), as produced by the **RND** keyword. Each time the **rounded** value is **odd** (i.e., *isn't 0 modulo* **2**) the number of favorable outcomes (**N**) is *decremented* by one, and after **K** tries have been sampled, the probability for the **odd** case will be the number of favorables outcomes (**-N**) divided by the number of tries (**K**), thus we have the estimated probability P = **-N/K**.

As the probability of the rounded division being either **even** or **odd** is **1** (*certainty*), the probability for the **odd** case is 1 minus the probability for the **even** case, thus it's **P = 1-(5-Pi)/4 = (Pi-1)/4**, so isolating *Pi* we have **Pi** = 1+4*P = 1+4*(-N/K) **= 1-4*N/K**, which is then displayed by the one-line program.

As you can see, the source of your confusion (sorry for it !) was due to the fact that I didn't remember my previous one-liner program posted many years ago and thus I simply created code anew based on the same theory, but as it happened the new code was a three-liner, used different variables for the number of favorable outcomes and the number of tries, and further it was using the probability for the *even* case to compute *Pi* instead of the probability for the *odd* case, as the one-liner did.

I hope this explains the whole affair to you, and thanks for your interest.

Regards.
V.

Find All My HP-related Materials here: **Valentin Albillo's HP Collection**