Contents

hpa : high precision approximation

Find rational approximations and other matches to floating point values.

The name “high precision approximation” is a deliberate apparent oxymoron - it only works well with high internal precision even though the intention is approximation…

Rationale

To obtain insight into underlying solutions and hints to rôle of symmetry

This function attempts to find close approximations using simple fractions or measures using common roots such as sqrt(2)

Summary

A long time ago I wrote a simple program to find close rational approximations to numbers and to show how closely the fractions approximate the values in question. For example to find successively better approximations to Pi : 3/1… 22/7, 179/57… 355/113 etc.

This last fraction (355/113) is an extremely close match to Pi, the next (slightly better) approximation is 52163/16604.

I decided that it may be useful to apply this exploration to answers from optimisation methods to try to uncover hidden representations that would otherwise be difficult to understand.

For example, if a value is shown to be 7.853981633974 the function can discover that it is actually extremely close to 5/2 * Pi which may be useful to help understand the problem.

Optimisation methods often give exact solutions to optimisation problems but these may be disguised rational answers such as 3/7 or sqrt(2)/2

The function may have other "guessing" applications

The project could look for sample problems with exact solutions which can be better interpreted using hpa()

These problems may have analytic solutions which are relatively easy to derive but when the results are expressed numerically the solutions are less obvious

Magic tricks

I like mathematical games of the "think of a number" variety where the 'magician' invites someone to think of a number then must apply various apparently unrelated numerical operations to their hidden number then answer a question about the result. Miraculously the magician is able to reveal the original number.

One good trick involves repeating a three digit number twice which the subject may not realise is equivalent to multiplying the original number by 1001. The subject may also not know that 1001 is 7 x 11 x 13. By a process of disguised division operations the original number can be revealed.

This approximation program (hpa) can be used to uncover hidden numbers in a similar way but without any manipulation or questioning. For example, the ratio 2589 / 7741 can be 'uncovered':

```
python hpa.py -x 2 -n 10000 -v 0.334452913060328

top 2 best approximations to 0.334452913060328

 approximation             error                value

 2589 / 7741  ratio        0.000000000000000    0.334452913060328
 857 / 8103  * sqrt(10)    0.000000000029182    0.334452913089510
```

but only when the internal numerator/denominator limit is raised ("-n 10000") from the default 1000. Otherwise the ratio can't be discovered:

```
top 2 best approximations to 0.334452913060328

 approximation             error                value

 120 / 773  * cbrt(10)     0.000000078923923    0.334452991984251
 155 / 964  * cbrt(9)      0.000000398737437    0.334453311797765
```

However, if there are common factors the revealed answer may be different. For example, dividing 123 by 789 produces 0.155893536121673 but hpa shows this to be "approximated" by 41/263 because 123 and 789 share the factor 3.

Even harder to discover, hpa can find hidden roots and 'special' numbers like Pi and e.

For example, 7 / 11 * the cube root of 3 can be "untangled" from the value 0.917795181104714 which is many orders of magnitude closer to (7 / 11) * cbrt(3) than the next best approximation: (453 / 844) * cbrt(5).

The main disadvantage of this party trick is that it requires very precise arithmetic which is likely to be tiresome and error-prone in a party atmosphere.

The original calculation is apparent as the precision is reduced digit by digit:

```
0.917795181104714
0.91779518110471
0.9177951811047
0.917795181104
0.91779518110
0.9177951811
0.917795181
0.91779518
```

but the original representation becomes the third most likely after removing one more digit:

```
top 10 best approximations to 0.9177951

 approximation             error                value

 308 / 723  * cbrt(10)     0.000000037662269    0.917795137662269
 453 / 844  * cbrt(5)      0.000000046735241    0.917795146735241
 7 / 11  * cbrt(3)         0.000000081104714    0.917795181104714
 244 / 553  * cbrt(9)     -0.000000429430986    0.917794670569014
 178 / 371  * cbrt(7)     -0.000000624168503    0.917794475831497
 159 / 245  * sqrt(2)      0.000000640478866    0.917795740478866
 159 / 490  * sqrt(8)      0.000000640478866    0.917795740478866
 551 / 953  * cbrt(4)     -0.000000787686802    0.917794312313198
 448 / 615  * cbrt(2)      0.000001046915290    0.917796146915290
 882 / 961  ratio         -0.000001135379813    0.917793964620187
```

Nevertheless, I think it is surprising that hpa() works as well as it does.

Settings display

hpa offers two ways to show the settings: -s (settings) and -S (verbose settings)

The first shows the current settings in abbreviated form on one line before the "top" line. The second form shows the same information but in a longer form.

For example, the short form:

```
hpa.py -e -p -i -b -r -n 100 -2 8 -3 7 -s

 settings: nd 100 sm 8 cm 7 recip e pi phi tau

 top 10 best approximations to 3.141592653589793

 approximation                        error              value

 (1 / 1)  * Pi                        0.000000000000000  3.141592653589793
 (1 / 2)  * Tau                       0.000000000000000  3.141592653589793
 (79 / 43)  * cbrt(5)                -0.000008937602373  3.141583715987420
 (61 / 12)  * recip Phi               0.000080122555505  3.141672776145298
 (95 / 24)  * recip cbrt(2)           0.000138595097269  3.141731248687062
 (95 / 48)  * cbrt(4)                 0.000138595097269  3.141731248687062
 (59 / 46)  * sqrt(6)                 0.000144190414717  3.141736844004511
 (19 / 16)  * sqrt(7)                 0.000237028299408  3.141829681889202
 (78 / 43)  * sqrt(3)                 0.000266950837473  3.141859604427266
 (89 / 77)  * e                       0.000317511771960  3.141910165361753
```

the long form:

```
hpa.py -e -p -i -b -r -n 100 -2 8 -3 7 -S

 numerator/denominator limit          : 100
 square root max                      : 8
 cube root max                        : 7
 matching reciprocals                 : True
 matching e                           : True
 matching pi                          : True
 matching phi                         : True
 matching tau                         : True

 top 10 best approximations to 3.141592653589793

 approximation                        error              value
```

```
(1 / 1)  * Pi                      0.000000000000000  3.141592653589793
(1 / 2)  * Tau                     0.000000000000000  3.141592653589793
(79 / 43)  * cbrt(5)              -0.000008937602373  3.141583715987420
(61 / 12)  * recip Phi             0.000080122555505  3.141672776145298
(95 / 24)  * recip cbrt(2)         0.000138595097269  3.141731248687062
(95 / 48)  * cbrt(4)               0.000138595097269  3.141731248687062
(59 / 46)  * sqrt(6)               0.000144190414717  3.141736844004511
(19 / 16)  * sqrt(7)               0.000237028299408  3.141829681889202
(78 / 43)  * sqrt(3)               0.000266950837473  3.141859604427266
(89 / 77)  * e                     0.000317511771960  3.141910165361753
```

Future extensions

-

Trigonometric functions

For example to discover sin, cos etc. values from common angles such as multiples of 30 °

-

Improve in-code documentation

-

Remove mathematically equivalent results

Sometimes two or more results are equivalent (but may differ in value slightly due to rounding)

For example `(16 / 1)  * recip cbrt(3)` `0.000000000000000` `11.093780389610156` `(16 / 3)  * cbrt(9)` `-0.000000000000002` `11.093780389610155`

Usage

```
usage: hpa.py [-h] [-p] [-i] [-b] [-e] [-r] [-s] [-S] [-n NDMX] [-2 SQRT]
              [-3 CBRT] [-x TOP] [-t THR] [-v VAL]
              [value]

hpa : high precision approximation

positional arguments:
  value                 value to approximate
```

```
optional arguments:
  -h, --help            show this help message and exit
  -p, --pi              enable Pi matching
  -i, --phi             enable Phi matching
  -b, --tau             enable Tau matching
  -e, --e               enable e matching
  -r, --r               enable reciprocal matching
  -s, --s               show settings
  -S, --S               show verbose settings
  -n NDMX, --ndmx NDMX  numerator and denominator limit
  -2 SQRT, --sqrt SQRT  square root max integer value
  -3 CBRT, --cbrt CBRT  cube root max integer value
  -x TOP, --top TOP     number of approximations
  -t THR, --thr THR     sensitivity threshold value
  -v VAL, --val VAL     value to approximate
```

Examples

python hpa.py

Note that no numerator/denominator values greater than 1000 are considered by default

Pi is the default internal value for the approximation

```
top 10 best approximations to 3.141592653589793
```

approximation		error	value
(831 / 506)	* cbrt(7)	-0.000000138009052	3.141592515580741
(473 / 239)	* cbrt(4)	0.000000223317982	3.141592876907775
(355 / 113)	ratio	0.000000266764189	3.141592920353983
(830 / 699)	* sqrt(7)	0.000000462552711	3.141593116142504
(632 / 569)	* sqrt(8)	-0.000000486912127	3.141592166677666
(907 / 622)	* cbrt(10)	0.000002625926153	3.141595279515947
(805 / 533)	* cbrt(9)	-0.000002639411964	3.141590014177829
(501 / 230)	* cbrt(3)	0.000003149558083	3.141595803147876
(463 / 361)	* sqrt(6)	-0.000003315892809	3.141589337696984
(811 / 683)	* sqrt(7)	-0.000005079251019	3.141587574338774

python hpa.py -x 5 -n 10000 -p -b

-x sets number of approximations

-n sets maximum numerator or denominator value (default = 1000)

-p enables Pi matching

-b enables Tau matching (2 * Pi)

```
top 5 best approximations to 3.141592653589793

 approximation                        error              value

 (1 / 1)  * Pi                        0.000000000000000  3.141592653589793
 (1 / 2)  * Tau                       0.000000000000000  3.141592653589793
 (8153 / 5803)  * sqrt(5)             0.000000008921974  3.141592662511767
 (9941 / 8372)  * sqrt(7)             0.000000010444261  3.141592664034054
 (5211 / 4063)  * sqrt(6)             0.000000024146643  3.141592677736436
```

python hpa.py -v 7.853981633974 -p

-v value is very close to 5/2 * Pi

```
top 10 best approximations to 7.853981633974

 approximation                        error              value

 (5 / 2)  * Pi                        0.000000000000483  7.853981633974483
 (501 / 92)  * cbrt(3)                0.000007873895692  7.853989507869692
 (932 / 227)  * cbrt(7)              -0.000008672106746  7.853972961867253
 (849 / 286)  * sqrt(7)               0.000014390829628  7.853996024803628
 (689 / 189)  * cbrt(10)              0.000015728099904  7.853997362073904
 (389 / 90)  * cbrt(6)                0.000017372822692  7.853999006796692
 (757 / 153)  * cbrt(4)               0.000022263672582  7.854003897646582
 (395 / 86)  * cbrt(5)               -0.000022344005450  7.853959289968549
 (425 / 121)  * sqrt(5)              -0.000023861763994  7.853957772210006
 (263 / 58)  * sqrt(3)               -0.000027109997884  7.853954523976116
```

python hpa.py 1.414213562373095

```
top 10 best approximations to 1.414213562373095

 approximation                        error              value

 (1 / 1)  * sqrt(2)                   0.000000000000000  1.414213562373095
 (1 / 2)  * sqrt(8)                   0.000000000000000  1.414213562373095
 (363 / 553)  * cbrt(10)             -0.000000556077301  1.414213006295794
 (463 / 681)  * cbrt(9)              -0.000000919094047  1.414212643279048
 (454 / 463)  * cbrt(3)               0.000000919094644  1.414214481467739
 (416 / 503)  * cbrt(5)               0.000001137065286  1.414214699438381
 (456 / 721)  * sqrt(5)              -0.000001360237306  1.414212202135789
 (571 / 989)  * sqrt(6)               0.000001445846515  1.414215008219610
```

```
(305 / 682)  * sqrt(10)             0.000001520253526  1.414215082626621
(397 / 537)  * cbrt(7)              0.000001855803140  1.414215418176235
```

python hpa.py -v 0.917795181104714 -x 15

This strange value happens to be 7/11 times the cube root of 3

```
top 15 best approximations to 0.917795181104714

approximation                       error              value

(7 / 11)  * cbrt(3)                 0.000000000000000  0.917795181104714
(453 / 844)  * cbrt(5)             -0.000000034369473  0.917795146735241
(308 / 723)  * cbrt(10)            -0.000000043442445  0.917795137662269
(244 / 553)  * cbrt(9)             -0.000000510535700  0.917794670569014
(159 / 245)  * sqrt(2)              0.000000559374152  0.917795740478866
(159 / 490)  * sqrt(8)              0.000000559374152  0.917795740478866
(178 / 371)  * cbrt(7)             -0.000000705273217  0.917794475831497
(551 / 953)  * cbrt(4)             -0.000000868791516  0.917794312313198
(448 / 615)  * cbrt(2)              0.000000965810576  0.917796146915290
(882 / 961)  ratio                 -0.000001216484527  0.917793964620187
(523 / 987)  * sqrt(3)             -0.000001288137619  0.917793892967095
(497 / 984)  * cbrt(6)             -0.000001548342952  0.917793632761762
(355 / 614)  * cbrt(4)              0.000001844056053  0.917797025160767
(815 / 888)  ratio                 -0.000002388311921  0.917792792792793
(162 / 467)  * sqrt(7)              0.000002918236750  0.917798099341464
```

python hpa.py -v 1.61803398875 -i

-i enables Phi matching (golden ratio)

```
top 10 best approximations to 1.61803398875

approximation                       error              value

(1 / 1)  * Phi                     -0.000000000000105  1.618033988749895
(447 / 502)  * cbrt(6)             -0.000000313459230  1.618033675290770
(733 / 976)  * cbrt(10)            -0.000000558633841  1.618033430116159
(455 / 744)  * sqrt(7)             -0.000000592870445  1.618033395879555
(581 / 570)  * cbrt(4)              0.000001118607059  1.618035107357059
(987 / 610)  ratio                 -0.000001201864754  1.618032786885246
(610 / 843)  * sqrt(5)             -0.000001407166522  1.618032581583478
(613 / 928)  * sqrt(6)              0.000001800394491  1.618035789144491
(307 / 600)  * sqrt(10)            -0.000001919297179  1.618032069452821
(516 / 451)  * sqrt(2)              0.000001927401923  1.618035916151923
```

python hpa.py -2 1 -3 1 -x 20

Display ratios only

Options -2 1 and -3 1 set the upper limits for square and cube roots

```
top 20 best approximations to 3.141592653589793

 approximation                        error              value

 (355 / 113)  ratio                    0.000000266764189  3.141592920353983
 (333 / 106)  ratio                   -0.000083219627529  3.141509433962264
 (311 / 99)  ratio                    -0.000178512175652  3.141414141414141
 (289 / 92)  ratio                    -0.000288305763706  3.141304347826087
 (267 / 85)  ratio                    -0.000416183001558  3.141176470588235
 (245 / 78)  ratio                    -0.000567012564152  3.141025641025641
 (223 / 71)  ratio                    -0.000747583167258  3.140845070422535
 (201 / 64)  ratio                    -0.000967653589793  3.140625000000000
 (179 / 57)  ratio                    -0.001241776396811  3.140350877192982
 (22 / 7)  ratio                       0.001264489267350  3.142857142857143
 (19 / 6)  ratio                       0.025074013076873  3.166666666666667
 (16 / 5)  ratio                       0.058407346410207  3.200000000000000
 (13 / 4)  ratio                       0.108407346410207  3.250000000000000
 (3 / 1)  ratio                       -0.141592653589793  3.000000000000000
 (2 / 1)  ratio                       -1.141592653589793  2.000000000000000
 (1 / 1)  ratio                       -2.141592653589793  1.000000000000000
```

python hpa.py -v 1.912931182772389 -x 10

Note that the last two approximations are identical

```
 (537 / 397)  * sqrt(2)               -0.000002510242535  1.912928672529854
 (537 / 794)  * sqrt(8)               -0.000002510242535  1.912928672529854

top 10 best approximations to 1.912931182772389

 approximation                        error              value

 (1 / 1)  * cbrt(7)                    0.000000000000000  1.912931182772389
 (760 / 573)  * cbrt(3)                0.000000184476530  1.912931367248919
 (467 / 772)  * sqrt(10)               0.000001028754338  1.912932211526727
 (579 / 550)  * cbrt(6)                0.000001223136354  1.912932405908743
 (349 / 316)  * sqrt(3)               -0.000001651628281  1.912929531144108
 (249 / 164)  * cbrt(2)               -0.000002271651514  1.912928911120874
 (820 / 733)  * cbrt(5)                0.000002345569891  1.912933528342280
 (725 / 379)  ratio                   -0.000002422877930  1.912928759894459
 (537 / 397)  * sqrt(2)               -0.000002510242535  1.912928672529854
 (537 / 794)  * sqrt(8)               -0.000002510242535  1.912928672529854
```

python hpa.py -r -i -v 0.618033988749855

-r enables reciprocal matching

Reciprocal of Phi

```
top 10 best approximations to 0.618033988749855

 approximation                        error              value

 (1 / 1)  * recip Phi                 0.000000000000040  0.618033988749895
 (502 / 447)  * recip cbrt(6)         0.000000119730795  0.618034108480650
 (253 / 700)  * cbrt(5)               0.000000174834723  0.618034163584578
 (976 / 733)  * recip cbrt(10)        0.000000213379213  0.618034202129068
 (744 / 455)  * recip sqrt(7)         0.000000226456442  0.618034215206297
 (285 / 581)  * cbrt(2)              -0.000000427269582  0.618033561480273
 (570 / 581)  * recip cbrt(4)        -0.000000427269582  0.618033561480273
 (610 / 987)  ratio                   0.000000459071827  0.618034447821682
 (843 / 610)  * recip sqrt(5)         0.000000537490251  0.618034526240106
 (928 / 613)  * recip sqrt(6)        -0.000000687688738  0.618033301061117
```