MULTIER – Multiprecision Euler's Constant ***e*** and its roots

Abstract

MULTIER is a very simple BASIC program written in 1990 for the HP-71B pocket computer which can quickly compute up to 35,500 digits of Euler's constant e and its roots (square root, cubic root, etc.). Five sample runs are included.

Keywords: *Euler constant e, roots of e, multiprecision, Hewlett-Packard, HP-71B, pocket computer, BASIC.*

1. Introduction

MULTIER is a very simple 5-line *BASIC* program *(269 bytes)* that I wrote in 1990 for the *Hewlett-Packard HP-71B* pocket computer, to quickly compute *up to 35,500 digits* of *Euler*'s constant *e (=2.71828+)* and its r^{th} roots (square root, cubic root, etc.) using multiprecision integer arithmetic. It computes $e^{1/r}$ using the formula:

$$e^{1/r} = \sum_{n=0}^{\infty} \frac{1}{n!\,r^n} = 1 + \frac{1}{1!\,r} + \frac{1}{2!\,r^2} + \frac{1}{3!\,r^3} + \cdots$$

The program considers the multiprecision ongoing result as subdivided in *8-digit blocks* stored as elements of array *E*. Each term is computed from the preceding one, dividing each block by the appropriate divisor and propagating the reminders, then adding it to the running total and finally releasing carries.

The number of blocks needed for the requested number of digits *N* is computed first, and one extra *guard block* is added (so the program actually computes *N+8* digits) in order to guarantee that *all printed digits are correct* (the guard block is *not* printed). Also, the first *non-zero* block is tracked to avoid unnecessary computation.

When computing ***e*** these are the *number of terms* added up, *RAM* used and *running times* for *N* decimal digits:

# Digits	***10***	***100***	***200***	***400***	***1000***	***2000***	***3000***	***5000***
# Terms	26	78	126	216	454	812	1147	1779
RAM *(bytes, approx.)*	159	335	527	917	2,120	4,123	6,126	10,124
Time *(seconds)*	0.05	0.46	1.29	3.96	20.47	72.88	154.28	399.39

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

As each element in the array can hold integers up to 12 digits long, using 8-digit blocks allows for up to 4-digit divisors, so a maximum of approximately *35,500* decimal digits can be computed, which would require using some *9,963* terms and the computation would then need about *71 Kb* of available *RAM*. See **Notes** below.

2. Program Listing

```
1  DESTROY ALL @ INPUT N,R @ K=8 @ L=10^K @ M=CEIL(N/K)+1 @ DIM E(M),T(M+1) @ T(1)=L @ A=1
2  Z=Z+1 @ D=Z*R @ A=A+NOT T(A) @ FOR I=A TO M
3  T(I+1)=T(I+1)+L*MOD(T(I),D) @ T(I)=T(I) DIV D @ E(I)=E(I)+RES @ NEXT I @ IF A<=M THEN 2
4  FOR I=M TO 2 STEP -1 @ E(I-1)=E(I-1)+E(I) DIV L @ E(I)=MOD(E(I),L) @ NEXT I @ FIX 0
5  PRINT IP(EXP(1/R)) @ FIX K @ FOR I=1 TO M-1 @ PRINT STR$(E(I)/L)[3]; @ NEXT I @ PRINT @ STD
```

Note: `Z` is implicitly initialized to 0 upon using it for the first time because `DESTROY ALL` deletes all variables.

3. Usage Instructions

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

4. Examples

The following examples can be used to check that the program was correctly entered and to understand its usage.
All of them assume that the printing width has been set to 48 `(PWIDTH 48)`.

4.1 Example 1

Compute 240 decimal digits of e (= $e^{1/1}$.)

RUN → ? 240, **1** **END LINE** → *(after 1.63 seconds)*

```
2.  718281828459045235360287471352662497757247093699  959574966967627724076630353547594571382178525166
    427427466391932003059921817413596629043572900334  295260595630738132328627943490763233829880753195
    251019011573834187930702154089149934884167509244
```

so we've got (all digits shown are correct):

e = 2.71828182 84590452 35360287 47135266 24977572 47093699 95957496 69676277 24076630 35354759
45713821 78525166 42742746 63919320 03059921 81741359 66290435 72900334 29526059 56307381
32328627 94349076 32338298 80753195 25101901 15738341 87930702 15408914 99348841 67509244

4.2 Example 2

Compute 480 decimal digits of $\sqrt{e}$ (= $e^{1/2}$.)

RUN → ? 480, **2** **END LINE** → *(after 5.02 seconds, all digits shown are correct)*

```
1.  648721270700128146848650787814163571653776100710  148011575079311640661021194215608632776520056366
    643002866637756307797004671166975219609159840971  452490059796929422659098403914719948464659489244
    896868905336418465720841066656859800088924981211  712287375214972195511971609034091115619799869839
    960642655091754574626304483075194758258782625439  931955712690076545322881476100957739788486181443
    265208203424170104718338591510630125661475533808  252026061400972891959084050148915029440695633113
```

4.3 Example 3

Compute 720 decimal digits of $\sqrt[3]{e}$ (= $e^{1/3}$.)

RUN → ? 720, **3** **END LINE** → *(after 9.72 seconds, all digits shown are correct)*

```
1.  395612425086089528628125319602586837597906515199  406982617516706031739015645951846969788817295830
    224135211184410418862096122123292777892939306349  394171747892493746212272685603480921799586962260
    969315833843005487778329856005531093679992145466  169898215623308599070119486045938278074673553410
    482582976369968617125771037541850047115745588493  189886485567833332235948981060734517579610566308
    794281144836223693155678604825977972824564504924  278799108317614554115186027922603625325343723408
    384377554349428676020539520485484013489497095790  213253765785890450306588622157751049439259932348
    924065935947532531330902919351236811148733639451  407173764583076109722921936291821685303197119106
    424157626059883580144262745696104334665465929493
```

4.4 Example 4

Compute 960 decimal digits of $\sqrt[5]{e}$ (= $e^{1/5}$.)

RUN → ? 960, **5** **END LINE** → *(after 15.13 seconds, all digits shown are correct)*

```
1.  22140275816016983392107199463967417030758094152050364127342509859920623308363781624228874401337247396902783756582071222657872335735698322420203091783847852468196963515090275879656423105784038067892980765878816657563573480602908117582872101992234299467757368481108634774781001772362473042116988340017941610960856955425714831378573372553893956298389961669480424133901501204485310137032866319130117745025529439090023275431137278565353847987589636920249682466927599785832165897017057220001748308578946383893972707329587358930504280675093721475993946440377260064756932917131359219760411691155771829441526008670646923360497991133179039993290092715239375622033454256824621475647492540244895714360091259413962978212740240853358499138869393556227022916329858349964688820804868034062665683528001375385988225523420645716904784338118261829962977004700386154037959557326902836851761601888943184174990821709528271060857084170456895072537358520316462031169522523228160920380
```

4.5 Example 5

Compute 1,200 decimal digits of e^2 (= $e^{1/0.5}$.)

RUN → ? 1200, **0.5** **END LINE** → *(after 33.01 seconds, all digits shown are correct)*

```
7.  38905609893065022723042746057500781318031557055184732408712782252257379607905776338431248507912179477375316126547886612388460369278127337447839221339807777490012289560741075370239133094755068208656818202696478682084042209822552348757424625414146799281293318880707633010193378997407299869600953033075153208188236846947930299135587714456831239232727646025883399964612128492852096789051388246639871228137268610647356263729295182227842948434586135287693866985752001549960148075071971293369418851997228882636255971941095866191479871504328397693264610235116312389990010513783406764498663892685615821864215577248492011193531621171951731747269796829345199850541848631971356859470229125573983561105149793681450277644807642985104182117055944191787683471285276497809713462504140235242158740938668254271570392645296404550628778001311092650138483345302646363141560471888117657942786348599076704527119372958723995987073310814961253109770593530099050329681075421090877626308572485003827872276144866745056498738587715751056243438943967139442950926006678296181965286063965971609339583335312827375276761571680732195169019642072457884477550696614543737966757387168282379855757192141990342828672289491780965647272324536045
```

Notes

1. Each block is 8-digit long so as to allow for divisors *up to 4 digits*, i.e. up to *D=9999*, memory permitting. Adding an 8-digit number to a 4-digit remainder times 10^8 gives a 12-digit result at most, so overflow is avoided. If *D* were restricted to 3 digits (up to *999)*, then 9-digit blocks could be used instead and both memory usage and computation time would be reduced but only *~2,500* digits would be achievable. Conversely, using 7-digit blocks and 5-digit divisors would increase memory usage and running time but theoretically up to *455,000* digits would be achievable, subject to available memory and a slight modification to release carries correctly.

References

M. Spiegel et al. *Mathematical Handbook of Formulas and Tables (Schaum's Outlines, Mcgraw-Hill)*

Copyrights