[6.50] The Savage Benchmark

In March 2001 a discussion on the comp.sys.hp48 news group examined the Savage benchmark and the performance of various HP and TI calculators with that benchmark. In this tip I describe that benchmark, and summarize some test results provided by HP and TI users. More importantly, I offer some opinion as to why benchmarks are fairly silly, particularly one as simple-minded as the Savage benchmark.

A benchmark is a program intended to measure some attribute of a computer. The attribute may be accuracy, execution time, code compactness or efficiency, or cost. Most generally, benchmark users want to compare the performance of two or more computers, with the net result being an evaluation as to which is the 'best' computer in terms of what the benchmark measures.

For perspective, Eric S. Raymond offers this definition in *The Jargon Lexicon*

> *benchmark n.*
>
> *[techspeak] An inaccurate measure of computer performance. "In the computer industry, there are three kinds of lies: lies, damn lies, and benchmarks." Well-known ones include Whetstone, Dhrystone, Rhealstone (see h), the Gabriel LISP benchmarks (see gabriel), the SPECmark suite, and LINPACK. See also machoflops, MIPS, smoke and mirrors.*
>
> *(http://www.tuxedo.org/~esr/jargon/html/entry/benchmark.html)*

There are at least two fundamental objections to benchmarks with regards to applying them to calculators. The first is that a simple benchmark by its very nature measures a narrow aspect of the calculator's performance. The second objection is not the fault of the benchmark, but the benchmark results can be taken well out of context to infer that one calculator is 'better' than another in some vague, broad sense.

I do not claim that some HP calculator is 'better' than some TI calculator, or vice versa, because such a claim cannot be made on the basis of a single benchmark. Instead, I examine this benchmark because it is instructive and interesting as a specific example of a benchmark.

The Savage benchmark is a simple iterative floating-point calculation:

1. Set a = 1
2. Set a = tan(atan(exp(ln(sqrt(a*a)))))+1
3. Repeat step 2 *n* times

where

tan() is the tangent function
atan() is the arc-tangent function
exp() is the natural-logarithm-base *e* exponentiation function *e*^x
ln() is the natural logarithm function
sqrt() is the square root function
n is the number of loop iterations

and the calculation is performed with the angle mode in radians.

The two results for this benchmark are relative accuracy and execution time. The relative accuracy is (a-(n+1))/(n+1), where *a* is the final value of *a*. Note that relative accuracy can be interpreted as the number of correct significant digits. If the relative accuracy is 1E-9, then we have nearly nine accurate significant digits. In general, the execution time is specified as time/n, or the time for each iteration. If the same number of iterations is used for both calculators, then the total execution time can be reported and compared.

If all the functions in the benchmark calculation were performed with no loss of precision, the relative accuracy would be zero, since *a* reduces to n+1.

Benchmarks can be grouped in two classes, 'synthetic' and 'live'. A live benchmark attempts to simulate a realistic mix of the all the operations the user would perform with the calculator. A synthetic benchmark attempts to measure a much more narrow aspect of performance. The Savage benchmark is clearly a synthetic benchmark, because it only measures a few functions out of the hundreds available on the calculator. Further, it is particularly 'synthetic' because we would never go to the trouble to actually calculate this expression as a real problem: we know that the final result is n+1.

For some reason, we have decided that n = 2499 is appropriate for benchmarking our calculators. There is nothing magic about this number. As I will show later, the relative accuracy is strongly dependent on *n*, even over a relatively narrow range. This makes the relative accuracy after *n* iterations a poor measure of performance. Faster computers typically use n = 100,000 or 250,000. If the computer (or calculator) does not have built-in timer facilities, then the timing must be done by hand. In this case, a large value for n reduces relative uncertainty from human reaction time in starting and stopping the stopwatch.

This table shows the relative error and execution time for a variety of calculators and languages. I cannot vouch for the accuracy, since most results were culled from posts on the HP and TI discussion groups.

Model	Language	Execution time, sec	Relative accuracy	Reported by:
HP-15C		~45 min	n/a	Mike Morrow
HP-20S		369	-2.054E-7	Mike Morrow
HP-28S		255	-2.054E-7	Mike Morrow
HP-32Sii		451	-2.054E-7	Mike Morrow
HP-41CX		~45 min	n/a	Mike Morrow
HP-42S		602	-2.054E-7	Mike Morrow
HP-48SX	UserRPL?	199	-2.054E-7	Mike Morrow
HP-48GX	UserRPL?	112	-2.054E-7	Mike Morrow
HP-48GX	Saturn ASM, display on, 15-digit	70	-3.572E-9	Jonathon Busby
HP-48GX	Saturn ASM, display off, 15-digit	62	-3.572E-9	Jonathon Busby
HP-49G	UserRPL	120	-2.054E-7	Ralf Fritzsch
HP-49G	SysRPL, 12-digit, Display on	96	-2.054E-7	Ralf Fritzsch
HP-49G	SysRPL, 12-digit, Display off	87	-2.054E-7	Thomas Rast
HP-49G	SysRPL, 15-digit	138	-3.572E-9	Ralf Fritzsch
Psion XPII	OPL Basic	47 min	-1.022E-5	Doug Burkett
TI-85		368	-3.08E-9	Mike Morrow
TI-86	TI Basic	433	-3.081E-9	Ralf Fritzsch
TI-86	Z80 assembler	328	-3.081E-9	Ralf Fritzsch
TI-89 HW1	C (GCC compiler), AMS 2.05	138	1.011E-9	Doug Burkett
TI-89 HW1	TI Basic, AMS 2.05	272	-3.081E-9	Doug Burkett
TI-92+	TI Basic (HW1? AMS?)	255	-3.081E-9	Jacek Marchel
TI-92+	68K assembler	72	-3.081E-9	Jacek Marchel
TI-92+ HW2	C (GCC compiler) AMS 2.05	96	1.011E-9	Doug Burkett
TI-92+ HW2	TI Basic, AMS 2.05	187	-3.081E-9	Doug Burkett

The best relative accuracy is returned with the TI-89 / TI-92 Plus, with the GCC C compiler program. The best execution time is returned by the HP-48GX with an ASM program and the display turned off.

So what can we conclude from these results? Very little, actually. We can conclude, for this very synthetic problem, that the relative accuracies range from less than 7 to nearly 9 significant digits. We see that newer hardware is more accurate and has faster execution times. We see that the choice of programming language has a strong effect on the execution time. None of this is news; there are no surprises here. Instead, what cannot be concluded is more significant:

- We cannot conclude that the HP-48GX is, in general, faster at numeric computation, because we have only tested a tiny subset of all the functions.
- We cannot conclude that the TI-89 / TI-92 Plus is, in general, more accurate, because again, only a tiny subset of the functions has been tested.
- We cannot even conclude that one calculator is more accurate than any other, even for these functions, because we have not actually tested function evaluation accuracy. Instead, we have tested the combined accuracy of three functions (tangent, e^x and square root) and their inverses. It is possible (though unlikely) that errors in one function evaluation are compensated by opposite errors in the inverse function.

To use the HP-49G and the TI-92 Plus as an example, all we have shown is that either calculator may or may not be faster and more accurate than the other.

There is much data missing from the table above, and this also precludes a fair comparison. We don't know, in each case, which versions of operating systems and compilers were used. We don't know which hardware versions were used. Running changes in hardware and software design may mean that clock rates and algorithms may have been changed. For a truly meaningful benchmark, all these variables must be specified.

In regards to the execution time, we have not really measured the function evaluation time, anyway. Instead, we have measured the combined execution time of the function evaluation and the loop overhead. For the TI-89 / TI-92 Plus TI Basic programs, this distinction is significant because the loop overhead is significant: 46 seconds for the TI-89 HW1, and 31 seconds for the TI-92 Plus HW2. This amounts to about 18 mS/iteration for the TI-89, and 12 mS/iteration for the TI-92 Plus.

This is the TI Basic code used for the TI-89 / TI-92 Plus benchmarks:

```
savage2()
Func
local a,n
1→a
for n,1,2499
 tan(tan⁻¹(e^(ln(√(a*a)))))+1→a
endfor
(a-2500)/2500
EndFunc
```

This is the GCC C code used for the TI-89/92+ benchmarks:

```
// C Source File
// Created 3/22/2001; 12:21:38 PM

#define RETURN_VALUE     // Redirect Return Value

// #define OPTIMIZE_ROM_CALLS // Use ROM Call Optimization

#define SAVE_SCREEN           // Save/Restore LCD Contents
```

```
#include <tigcclib.h>         // Include All Header Files

short _ti89;                  // Produce .89Z File
short _ti92plus;              // Produce .9XZ File


// Main Function
void _main(void)
{
float a=1.0;
int k;
a=1.0;
for(k=1;k<2500;k++) a=tan(atan(exp(log(sqrt(a*a)))))+1.0;
push_Float ((a-2500.0)/2500.0);
}
```

The C version was written by Artraze (http://www.angelfire.com/pa3/cartethus/).

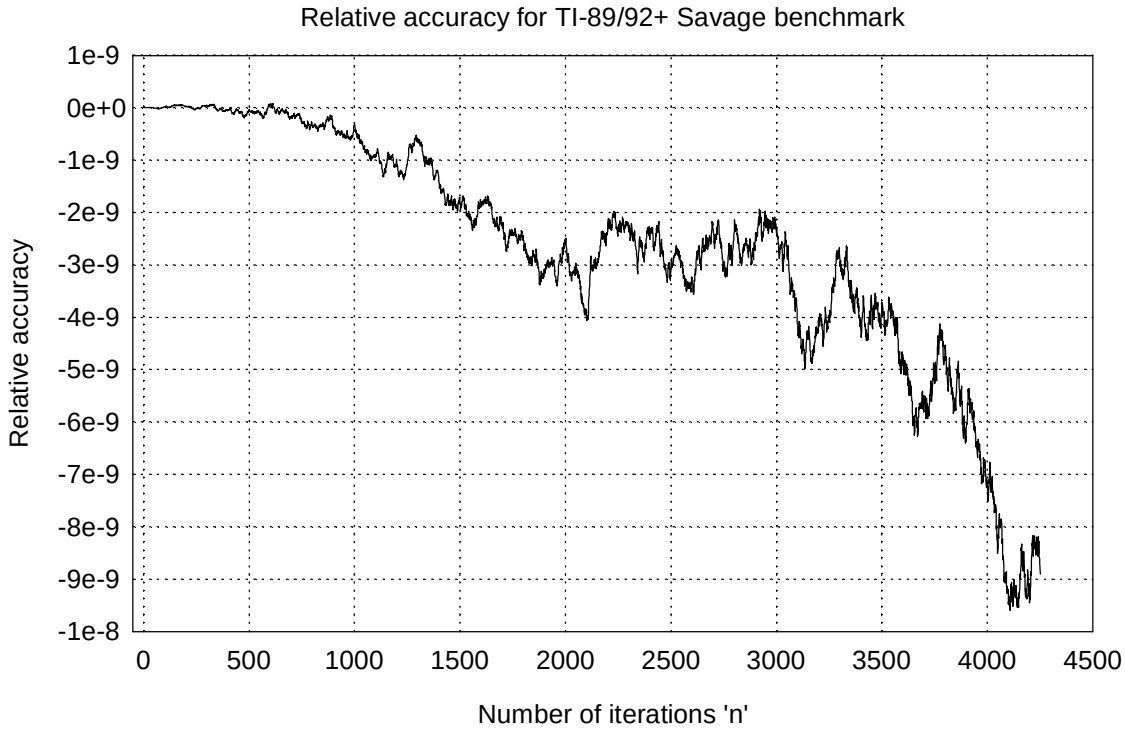
Note that the C version is 641 bytes, and the TI Basic version is only 83 bytes.

There has been some confusion about the number of significant digits used by the TI-89 / TI-92 Plus, which can be cleared up by referring to the TI SDK manual. In usual calculator operation, 12 significant digits are displayed, except for regression results, in which 14 significant digits can be shown. 14 significant digits are used for 'normal' calculator operation, and for TI Basic calculations. However, C programs can use the C 'double' floating point, type, as this quote from section 16.4 describes:

> *"Applications can work with float numbers on the estack or in C floating-point variables. The compiler supports two forms of floating-point values as described in Chapter 2 of the compiler documentation. The calculator implementation uses the standard C type double. The symbols BCD16 and Float are also defined to be double. BCD16 is the recommended type for declaring floating-point variables in applications.*
>
> *"This type uses a 16-digit mantissa and provides more accuracy. Thus, BCD16 variables provide the best results when implementing iterative algorithms that require a great deal of floating-point computation.*
>
> *"push_Float is the routine that converts a C floating-point value into a tagged floating-point value on the expression stack. The 16-digit value is rounded to 14-digits, pushed onto the estack, and then a FLOAT_TAG is pushed on top."*

The C program listed above defines *a* as double, and this is why the TI-89/92+ relative accuracy is slightly better than that of the HP-49G. For the vast majority of applications, this difference is insignificant.

I mentioned earlier that the relative accuracy at a single point is a poor indicator of accuracy. The plot below shows the relative accuracy at each calculator point, up to 4,250 points. This data is from the TI-89 / TI-92 Plus TI Basic program.



As expected, the relative error starts out at zero, then increases with the number of iterations. However, the error does not increase smoothly, and varies considerably over short ranges. For example, the relative error is about -3E-9 at 2500 iterations, but was -2E-9 at about 2200 iterations. Since the actual shape of the relative error curve will depend on the number of significant digits, as well as the algorithms used for the evaluated functions, some other statistical measure would be a better indication of the relative accuracy. Some candidates are the RMS error over some range, or even the mode. Of course, the best description of the relative error is given by the error plot itself.

References:

Byte magazine, 1985, vol 10 no 11
Sky & Telescope magazine, March 1987

Neither of these references discuss the benchmark in much detail.