

Improvements to `tic` and `toc` Functions for Measuring Absolute Elapsed Time Performance in MATLAB

By [Martin Knapp-Cordes](#) and Bill McKeeman

Suppose you are developing an application in MATLAB—an image processing program, for example, or financial data import/export. Testing has shown that the application gives the correct results, but the elapsed time for all or parts of the application seems too long. How do you make the application run faster?

First, you must figure out where the time is being spent and locate the parts of the application that take the most time. To do this, you can use the MATLAB Profiler, which automatically instruments your code and measures the frequency and relative elapsed time within and across functions (see the [MATLAB User's Guide](#) for guidelines on profiling for improved performance). To further analyze critical sections of these slower parts, you can create a simple test program and measure the absolute elapsed time of those sections.

Second, you must obtain a precise measurement of absolute elapsed time. MATLAB provides two sets of functions for measuring absolute elapsed time: `clock` and `etime`, and `tic` and `toc`. The `clock` and `etime` functions use dates to track elapsed time, and are useful if you need accuracy to only .01 second. Originally, the `tic` and `toc` functions relied on `clock` for the time and had the same accuracy. This article describes the decoupling of `tic` and `toc` from `clock` and the improvements that give `tic` and `toc` the highest accuracy and most predictable behavior.

What Makes a Good Instrumentation Tool?

A good instrumentation tool:

- Measures something useful
- Is easy to use
- Has low overhead
- Does not skew your results
- Does not interfere with scaling your problem
- Provides accurate results

What Is Time and How Is It Measured?

You expect time as measured on your computer to be uniform, in standard units like seconds, and accurate enough to be used to measure short computations. What limits the uniformity and accuracy of time measurement on your computer? To answer this question we must first understand what time is and how it is measured on a computer.

We perceive and witness time as a succession of transient *nows*. We attach a measurement to each now. By international agreement, the difference between two nows, or the *elapsed time*, is measured in seconds. The device that measures each now we call a *clock*. Real clocks depend on counting the repetition of a uniform and stable physical process, such as the swings of a pendulum, or the vibrations of a crystal, or the periods of the radiation from an excited atom. Each repetition is called a *tick*. The number of ticks per second, or the *frequency*, is measured in hertz.

The best clocks have a high frequency and very stable repetitions, or *low drift*. The current best clocks use the atomic frequency of radiation from an excited Cesium atom ($\sim 9.2 \times 10^9$ hertz), which has defined the second since 1967 and is the basis of *International Atomic Time* (TAI). Neither people nor computers use TAI time directly; they use local time or corrected *Coordinated Universal Time* (UTC), which is defined by international agreement based on the mean solar day with corrections. UTC time is corrected TAI time with constant second duration. All units larger than a second are variable. Leap seconds are added periodically to account for the slowing of the earth's rotation.

A computer system is constructed with its own internal clocks; the number and variety depend on your particular hardware. The hardware architecture probably includes several CPUs, or *cores*, all running at different speeds. Your operating system may run on your hardware or depend on hardware virtualization. Your operating system provides various time services that rely on access to hardware clocks. Some return values are synchronized with external clocks; others are not. For example, to return results in TAI Seconds requires synchronization with TAI sources. The Network Time Protocol (NTP) can be used to do this via the Internet, but maintaining consistently high accuracy is difficult because of server delay variability [1]. Special client/server software is required to smooth out the variability, and OS providers have not agreed on a solution. As a result, those internal clocks on your computer system can appear to have an effective frequency of only 10^2 hertz (10 milliseconds).

Measuring Elapsed Time Using `tic` and `toc`

Wall clock time is the best way to measure elapsed time performance. It is simple to understand and, with careful use, can show consistent changes in performance on a particular hardware or software platform. Increasing the accuracy of the underlying clock enables meaningful measurements using shorter computations and opens up opportunities to assign a performance number to a computation using fewer trials. This can mean better-performing test suites.

Before R2006b, the MATLAB `tic` and `toc` functions were connected to `clock`. Each `tic` command would get the current time as a local time date vector generated by `clock` and save it. The `toc` command would call `clock` again to get the current time and subtract it from the saved local time date vector to get the elapsed time. `clock` calls a time service using API provided by the operating system: the `GetLocalTime` function on Microsoft Windows and the POSIX.1-2001 (IEEE Portable Operating System Interface for Computing Environments) `gettimeofday` and `localtime` functions on Linux and Mac OS.

These `clock` values suffer from several problems. Special handling must be used at the daylight saving time boundaries. On Microsoft Windows, the UTC time service used by the `GetLocalTime` function has known low accuracy and, under certain conditions, can give negative elapsed times between successive calls to the service. Because other applications of `clock`, such as `date`, did not require higher accuracy, we did not attempt to improve the limited accuracy and non-uniform behavior of `clock`, but instead focused on improving `tic` and `toc`.

Creating More Accurate Time Values

Our goal was to create more accurate time values for each `tic` using units as close as possible to standard seconds. We elected to stay with known interfaces, which compromised the requirement of keeping the units as standard seconds [2]. On Linux and Mac OS, we stopped converting to local time and used the `gettimeofday` function directly. On Microsoft Windows, we opted for a pure local time service, using `QueryPerformanceCounter` and `QueryPerformanceFrequency` functions, with units in approximate standard Seconds.

With R2006b, the `tic` and `toc` functions started using these time services. The output from each service can be used to form a 64-bit counter value that is saved as a `tic` value. When `toc` is called, the service is called again, and a difference is formed with the last `tic` value. That difference is divided by a frequency and returned as the elapsed time. The frequency used for `gettimeofday` is 10^6 hertz. The frequency of the clock used by the `QueryPerformanceCounter` function is returned by the `QueryPerformanceFrequency` function.

Here is a simple way to see the resolution of `tic/toc`:

```
>> tic; toc
Elapsed time is 0.000001 seconds.
>>
```

By comparison, `etime` returns 0, as its resolution is not fine enough to detect the very short time interval.

```
>> t =clock; etime(clock,t)
ans =
    0
>>
```

Accuracy issues such as the one caused by daylight saving time have been removed on all platforms. Handling leap seconds is still an issue on Linux and Mac OS, however. In addition, the counters underlying the services—for example, the counter in `gettimeofday`—may be less than 64 bits wide. By 2038, the number of seconds since the Epoch (January 1, 1970) will exceed its capacity of 32 bits.

Hardware-Dependent Behavior

These time services can show hardware-dependent behavior. For example, on modern Intel architectures there is a timestamp counter (TSC) register on each CPU. This 64-bit register is incremented on each CPU cycle. We considered using a service to drive `tic` and `toc` using the RTSC instruction that reads the counter, giving very low overhead access to a clock with frequency in the gigahertz (10^9) range. Unfortunately, it failed as a general solution. The power management software on laptops often slows down a CPU to preserve battery usage and control chassis temperature. In addition, in multiple CPU configurations the TSC values on different processors may be different, and the RTSC instruction may not read the same processor at the `tic` and the `toc` steps, causing random results.

The time-service APIs hide the platform dependencies. For example, the `QueryPerformanceCounter` function uses the best high-resolution counter configured by the operating system. This means that if the TSC cannot be used, it will use the Advanced Programmable Interrupt Controller (APIC) clock, which has lower resolution. In rare cases, the operating system selected the wrong clock, and `tic` and `toc` appeared to be broken. Operating system configuration or processor driver patches can be applied to make `tic` and `toc` work correctly. MATLAB does not need to be fixed.

In summary, use `tic` and `toc` to measure elapsed time in MATLAB, because the functions have the highest accuracy and most predictable behavior. The basic syntax is

```
tic;
... computation ...
toc
```

where the `tic` and `toc` lines are recognized by MATLAB for minimum overhead.

The `tic` and `toc` syntax was extended in R2008b to allow multiple nested `tic` and `toc` counters. Note that the new syntax that lets you save the counter from `tic` and pass it to `toc`, like

```
REPS = 1000; minTime = Inf; nsum = 10;
tic;
for i=1:REPS
    tstart = tic;
    sum = 0; for j=1:nsum, sum = sum + besselj(j,REPS); end
    telapsed = toc(tstart);
    minTime = min(telapsed,minTime);
end
averageTime = toc/REPS;
```

can result in extra overhead. This issue will be fixed in a future version of MATLAB.

References

- Ridoux, Julien and Darryl Veitch. "Principles of Robust Timing over the Internet," *ACM Queue*, Vol. 8, Issue 4, April 2010.
- Nilsson, John. "[Implement a Continuously Updating High-Resolution Time Provider for Windows](#)," *MSDN Magazine*, March 2004.

Products Used

- [MATLAB®](#)

Learn More

- [Measuring MATLAB Performance](#)
- [Profiling to Improve Performance](#)