iteration cycles periodically between these values. When implemented on a computer, however, using floating-point arithmetic, this is not the case. As seen in table 5.1, computed results agree with exact results to at least five decimal places until iteration 40, where accumulated error becomes visible in the fifth decimal digit. As the iterations continue, the error grows until, at iteration 55, the computed result takes on the value 1 and remains there for all subsequent iterations. Later in the chapter, we will see why such errors occur.

TABLE 5.1
Computed results from iteration (5.1). After 55 iterations, the computed value is 1 and it remains there for all subsequent iterations.

k	True x_k	Computed x_k
0	0.10000	0.10000
1	0.20000	0.20000
2	0.40000	0.40000
3	0.80000	0.80000
4	0.60000	0.60000
5	0.20000	0.20000
10	0.40000	0.40000
20	0.60000	0.60000
40	0.60000	0.60001
42	0.40000	0.40002
44	0.60000	0.60010
50	0.40000	0.40625
54	0.40000	0.50000
55	0.80000	1.00000

We begin this chapter with two stories that demonstrate the potential cost of overlooking the effects of rounding errors.

5.1 COSTLY DISASTERS CAUSED BY ROUNDING ERRORS

The Intel Pentium Flaw. [37, 55, 76, 78, 82] In the summer of 1994, Intel anticipated the commercial success of its new Pentium chip. The new chip was twice as fast at division as previous Intel chips running at the same clock rate. Concurrently, Professor Thomas R. Nicely, a mathematician at Lynchburg College in Virginia, was computing the sum of the reciprocals of prime numbers using a computer with the new Pentium chip. The computational and theoretical results differed significantly. However, results run on a computer using an older 486 CPU calculated correct results. In time, Nicely tracked the error to the Intel chip. Having contacted Intel and received little response to his initial queries, Nicely posted a general notice on the

Internet asking for others to confirm his findings. The posting (dated October 30, 1994) with subject line *Bug in the Pentium FPU* [78] began:

> It appears that there is a bug in the floating point unit (numeric coprocessor) of many, and perhaps all, Pentium processors.

This email began a furor of activity, so much so that only weeks later on December 13, IBM halted shipment of their Pentium machines, and in late December, Intel agreed to replace all flawed Pentium chips upon request. The company put aside a reserve of \$420 million to cover costs, a major investment for a flaw. With a flurry of Internet activity between November 29 and December 11, Intel had become a laughingstock on the Internet joke circuit, but it wasn't funny to Intel. On Friday, December 16, Intel stock closed at \$59.50, down \$3.25 for the week.

Figure 5.1. A bug in the Pentium chip cost Intel millions of dollars. (Courtesy of CPU Collection Konstantin Lanzet.)

What type of error could the chip make in its arithmetic? The *New York Times* printed the following example of the Pentium bug: Let $A = 4{,}195{,}835.0$ and $B = 3{,}145{,}727.0$, and consider the quantity

$$A - (A/B) * B.$$

In exact arithmetic, of course, this would be 0, but the Pentium computed 256, because the quotient A/B was accurate to only about 5 decimal places. Is this *close enough*? For many applications it probably is, but we will see in later sections that we need to be able to count on computers to do better than this.

While such an example can make one wonder how Intel missed such an error, it should be noted that subsequent analysis confirmed the subtlety of the mistake. Alan Edelman, professor of mathematics at Massachusetts Institute of Technology, writes in his article of 1997 published in *SIAM Review* [37]:

> We also wish to emphasize that, despite the jokes, the bug is far more subtle than many people realize. . . . The bug in the Pentium was an easy mistake to make, and a difficult one to catch.

Ariane 5 Disaster. [4, 45] The Ariane 5, a giant rocket capable of sending a pair of three-ton satellites into orbit with each launch, took 10 years and 7 billion dollars for the European Space Agency to build. Its maiden launch was met with eager anticipation as the rocket was intended to propel Europe far into the lead in the commercial space business.

On June 4, 1996, the unmanned rocket took off cleanly but veered off course and exploded in just under 40 seconds after liftoff. Why? To answer this question, we must step back into the programming of the onboard computers.

During the design of an earlier rocket, programmers decided to implement an additional "feature" that would leave the horizontal positioning function (designed for positioning the rocket on the ground) running after the countdown had started, anticipating the possibility of a delayed takeoff. Since the expected deviation while on the ground was minimal, only a small amount of memory (16 bits) was allocated to the storage of this information. After the launch, however, the horizontal deviation was large enough that the number could not be stored correctly with 16 bits, resulting in an exception error. This error instructed the primary unit to shut down. Then, all functions were transferred to a backup unit, created for redundancy in case the primary unit shut down. Unfortunately, the backup system contained the same bug and shut itself down. Suddenly, the rocket was veering off course causing damage between the solid rocket boosters and the main body of the rocket. Detecting the mechanical failure, the master control systems triggered a self-destruct cycle, as had been programmed in the event of serious mechanical failure in flight. Suddenly, the rocket and its expensive payloads were scattered over about 12 square kilometers east of the launch pad. Millions of dollars would have been saved if the data had simply been saved in a larger variable rather than the 16-bit memory location allocated in the program.

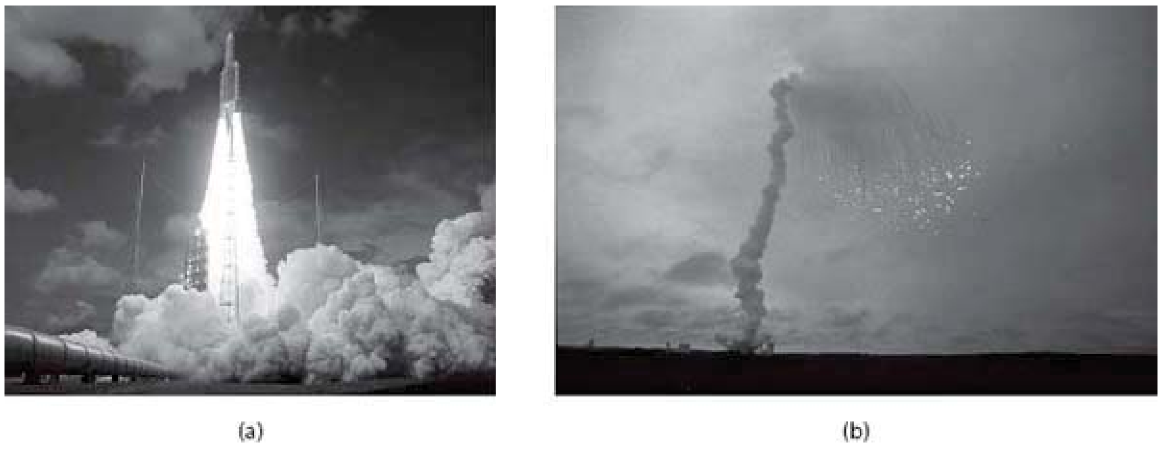

Figure 5.2. The Ariane 5 rocket (a) lifting off and (b) flight 501 self-destructing after a numerical error on June 4, 1996. (Courtesy of ESA/CNES.)