Chapter 3

Calendars and Clocks

Computations involving time, dates, biorhythms and Easter.

Calendars are interesting mathematical objects. The Gregorian calendar was first proposed in 1582. It has been gradually adopted by various countries and churches over the four centuries since then. The British Empire, including the colonies in North America, adopted it in 1752. Turkey did not adopt it until 1923. The Gregorian calendar is now the most widely used calendar in the world, but by no means the only one.

In the Gregorian calendar, a year y is a *leap year* if and only if y is divisible by 4 and not divisible by 100, or is divisible by 400. In MATLAB the following expression must be `true`.

```
mod(y,4) == 0 && mod(y,100) ~= 0 || mod(y,400) == 0
```

For example, 2000 was a leap year, but 2100 will not be a leap year. This rule implies that the Gregorian calendar repeats itself every 400 years. In that 400-year period, there are 97 leap years, 4800 months, 20871 weeks, and 146097 days. The average number of days in a Gregorian calendar year is $365 + \frac{97}{400} = 365.2425$.

The MATLAB function `clock` returns a six-element vector `c` with elements

```
c(1) = year
c(2) = month
c(3) = day
c(4) = hour
c(5) = minute
c(6) = seconds
```

Copyright © 2008 Cleve Moler
MATLAB® is a registered trademark of The MathWorks, Inc.™
April 6, 2008

The first five elements are integers, while the sixth element has a fractional part that is accurate to milliseconds. The best way to print a `clock` vector is to use `fprintf` or `sprintf` with a specified *format string* that has both integer and floating point fields.

```
f = '%6d %6d %6d %6d %6d %9.3f\n'
```

I am writting this on May 22, 2007, at about 7:30pm, so

```
c = clock;
fprintf(f,c);
```

produces

```
2007      5     22     19     31    54.015
```

In other words,

```
year = 2007
month = 5
day = 22
hour = 19
minute = 31
seconds = 54.015
```

The MATLAB functions `datenum`, `datevec`, `datestr`, and `weekday` use `clock` and facts about the Gregorian calendar to facilitate computations involving calendar dates. Dates are represented by their *serial date number*, which is the number of days since the theoretical time and day over 20 centuries ago when `clock` would have been six zeroes. We can't pin that down to an actual date because different calendars would have been in use at that time.

The function `datenum` returns the date number for any clock vector. For example, my current date number

```
datenum(c)
```

is

```
733184.848
```

This indicates that the current time is most of the way through day number `733184`. I get the same result from

```
datenum(now)
```

The `datenum` function also works with a given year, month and day, or a date specified as a string. For example both

```
datenum(2007,5,22)
```

and

```
datenum('May 22, 2007')
```

return

```
733184
```

The same result is obtained from

```
fix(now)
```

Computing the difference between two date numbers gives an elapsed time measured in days. How many days are left between today and the first day of next year?

```
datenum('Jan 1, 2008') - datenum(fix(now))
ans =
   224
```

The `weekday` function computes the day of the week, as both an integer between 1 and 7 and a string. For example both

```
[d,w] = weekday(datenum(2007,5,22))
```

and

```
[d,w] = weekday(now)
```

return

```
d =
   3
w =
  Tue
```

So today is the third day of the week, a Tuesday.

Friday the 13th is unlucky, but is it unlikely? What is the probability that the 13th day of any month falls on a Friday? The quick answer is 1/7, but that is not quite right. This code counts the number of times that Friday occurs on the various weekdays in a 400 year calendar cycle and produces figure 3.1.

```
c = zeros(1,7);
for y = 1:400
   for m = 1:12
      d = datenum([y,m,13]);
      w = weekday(d);
      c(w) = c(w) + 1;
   end
end
c
bar(c)
axis([0 8 680 690])
```

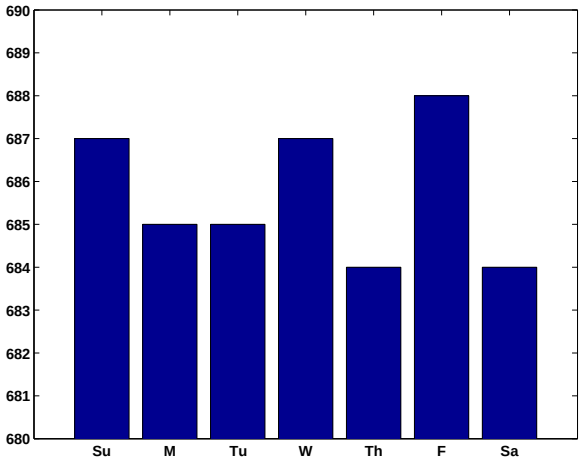

Figure 3.1. *The 13th is more likely to be on Friday than any other day.*

```
set(gca,'xticklabel',{'Su','M','Tu','W','Th','F','Sa'})

c =
   687   685   685   687   684   688   684
```

So the 13th day of a month is more likely to be on a Friday than any other day of the week. The probability is `688/4800 = .143333`. This probability is close to, but slightly larger than, `1/7 = .142857`.

Biorhythms were invented over 100 years ago and entered our popular culture in the 1960s. You can still find many Web sites today that offer to prepare personalized biorhythms, or that sell software to compute them. Biorhythms are based on the notion that three sinusoidal cycles influence our lives. The physical cycle has a period of 23 days, the emotional cycle has a period of 28 days, and the intellectual cycle has a period of 33 days. For any individual, the cycles are initialized at birth.

Figure 3.2 is my biorhythm, which begins on August 17, 1939, plotted for an eight-week period centered around the date this is being written, May 22, 2007. It shows that I must have been in pretty bad shape about a week ago, but that my physical strength can be expected to reach a peak a few days from now, and that my intellectual powers and emotional well-being should reach their peaks within a few hours of each other about a week after that.

A search of the United States Government Patent and Trademark Office database of US patents issued between 1976 and 2007 finds 113 patents that are based upon or mention biorhythms. The Web site is

```
http://patft.uspto.gov
```

The date and graphics functions in MATLAB make the computation and display of biorhythms particularly convenient. The following code segment is part of our program `biorhythm.m` that plots a biorhythm for an eight-week period centered

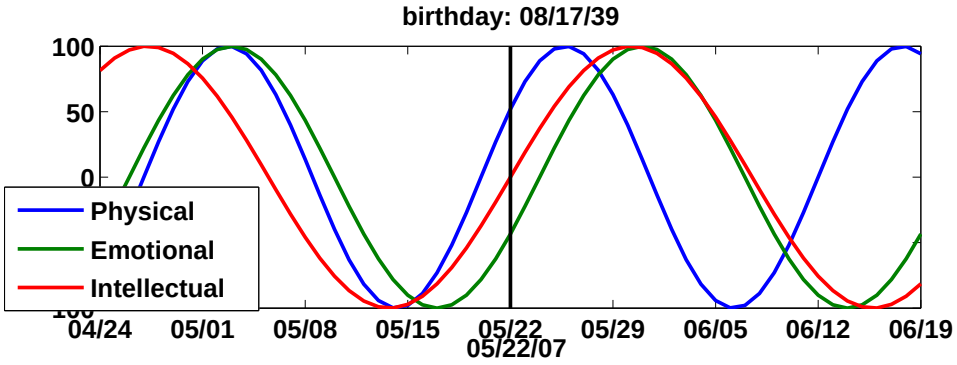

Figure 3.2. *My biorhythm.*

on the current date.

```
t0 = datenum('Aug. 17, 1939')
t1 = fix(now);
t = (t1-28):1:(t1+28);
y = 100*[sin(2*pi*(t-t0)/23)
         sin(2*pi*(t-t0)/28)
         sin(2*pi*(t-t0)/33)];
plot(t,y)
```

You see that the time variable `t` is measured in days and that the trig functions take arguments measured in radians.

Easter Day is one of the most important events in the Christian calendar. It is also one of the most mathematically elusive. In fact, regularization of the observance of Easter was one of the primary motivations for calendar reform. The informal rule is that Easter Day is the first Sunday after the first full moon after the vernal equinox. But the ecclesiastical full moon and equinox involved in this rule are not always the same as the corresponding astronomonical events, which, after all, depend upon the location of the observer on the earth. Computing the date of Easter is featured in Don Knuth's classic *The Art of Computer Programming* and has consequently become a frequent exercise in programming courses. Our MATLAB version of Knuth's program, `easter.m`, is the subject of several exercises in this chapter.

Exercises

3.1 *Microcentury.* The optimum length of a classroom lecture is one *microcentury*. How long is that?

3.2 $\pi \cdot 10^7$. A good estimate of the number of seconds in a year is $\pi \cdot 10^7$. How accurate is this estimate?

3.3 *First datenum.* The first countries to adopt the Gregorian calendar were Spain, Portugal and much of Italy. They did so on October 15, 1582, of the new calendar. The previous day was October 4, 1582, using the old, Julian, calendar. So October 5 through October 14, 1582, did not exist in these countries. What is the MATLAB serial date number for October 15, 1582?

3.4 *Future datenum's.* Use `datestr` to determine when `datenum` will reach 750,000. When will it reach 1,000,000?

3.5 *Your birthday.* On which day of the week were you born? In a 400-year Gregorian calendar cycle, what is the probability that your birthday occurs on a Saturday? Which weekday is the most likely for your birthday?

3.6 *Ops per century.* Which does more operations, a human computer doing one operation per second for a century, or an electronic computer doing one operation per microsecond for a minute?

3.7 *Julian day.* The Julian Day Number (JDN) is commonly used to date astronomical observations. Find the definition of Julian Day on the Web and explain why

```
JDN = datenum + 1721058.5
```

In particular, why does the conversion include an 0.5 fractional part?

3.8 *Unix time.* The Unix operating system and POSIX operating system standard measure time in seconds since 00:00:00 Universal time on January 1, 1970. There are `86,400` seconds in one day. Consequently, Unix time, `time_t`, can be computed in MATLAB with

```
time_t = 86400*(datenum(y,m,d) - datenum(1970,1,1))
```

Some Unix systems store the time in an 32-bit signed integer register. When will `time_t` exceed 2^{31} and overflow on such systems.

3.9 *Easter.*
(a) The comments in `easter.m` use the terms "golden number", "epact", and "metonic cycle". Find the definitions of these terms on the Web.
(b) Plot a `bar` graph of the dates of Easter during the 21-st century.
(c) How many times during the 21-st century does Easter occur in March and how many in April?
(d) On how many different dates can Easter occur? What is the earliest? What is the latest?

(e) Is the date of Easter a periodic function of the year number?

3.10 `biorhythm`.
(a) Use `biorhythm` to plot your own biorhythm, based on your birthday and centered around the current date.
(b) All three biorhythm cycles start at zero when you were born. How long does it take until all three simultaneously return to that initial condition? How old were you, or will you be, on that date? Plot your biorhythm near that date. You should find the `lcm` function helpful.
(c) Is it possible for all three biorhythm cycles to reach their maximum or minimum at exactly the same time? Why or why not?

3.11 `clockex`. This exercise is about the `clockex` program in our `exm` toolbox.
(a) Why does `clockex` use trig functions?
(b) Make `clockex` run counter-clockwise.
(c) Modify `clockex` to have a digital display of your own design.
(d) Why do the hour and minute hands in `clockex` move nearly continuously while the second hand moves in discrete steps.
(e) What happens if the statement

```
pause(1.0)
```

in `clockex.m` is changed to

```
pause(0.9)
```

or

```
pause(1.1)
```

(f) What happens if the statement

```
k = ceil(c(6))
```

in `clockex.m` is changed to

```
k = round(c(6))
```

(g) Instead of

```
k = ceil(c(6))
```

in `clockex.m`, I considered using

```
k = ceil(c(6)+realmin)
```

Why would that be a good idea?