Chapter 14

Exponential Function

The function e^x.

The exponential function is denoted mathematically by e^t and in MATLAB by `exp(t)`. This function is the solution to the world's simplest, and perhaps most important, differential equation

$$\dot{y} = ky$$

This differential equation is the basis for any mathematical model describing the time evolution of a quantity with a rate of production that is proportional to the quantity itself. Such models include populations, investments, feedback, and radioactive decay. We are using t for the independent variable, y for the dependent variable, k for the proportionality constant, and

$$\dot{y} = \frac{dy}{dt}$$

for the derivative with respect to t. We are looking for a function that is proportional to its own derivative.

Let's start by examining the function

$$y = 2^t$$

We know what 2^t means if t is an integer, 2^t is the t-th power of 2.

$$2^{-1} = 1/2, \quad 2^0 = 1, \quad 2^1 = 1, \quad 2^2 = 4, ...$$

We also know what 2^t means if $t = p/q$ is a rational number, the ratio of two integers, $2^{p/q}$ is the q-th root of the p-th power of 2.
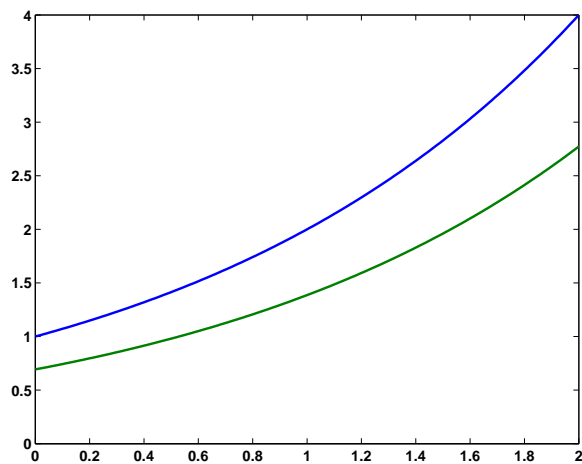
$$2^{1/2} = \sqrt{2} = 1.4142...,$$

Figure 14.1. *The blue curve is the graph of $y = 2^t$. The green curve is the graph of the derivative, $\dot{y} = dy/dt$.*

Figure 14.2. *The ratio, $\dot{y}/y$.*
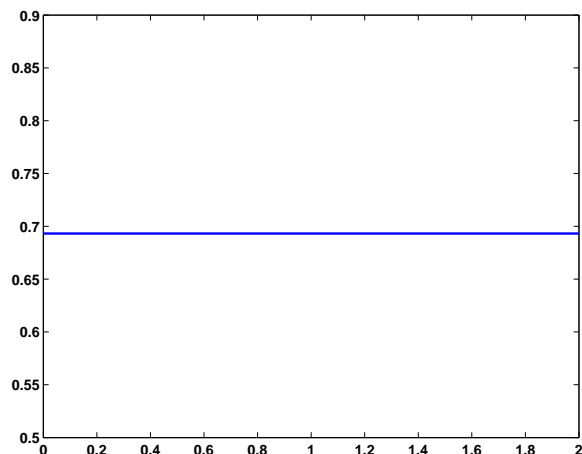
$$
\begin{aligned}
2^{5/3} &= \sqrt[3]{2^5} = 3.1748..., \\
2^{355/113} &= \sqrt[113]{2^{355}} = 8.8250...
\end{aligned}
$$

In principal, for floating point arithmetic, this is all we need to know. All floating point numbers are ratios of two integers. We do not have to be concerned yet about the definition of 2^t for irrational t. If MATLAB can compute powers and roots, we can plot the graph of 2^t, the blue curve in figure 14.1

What is the derivative of 2^t? Maybe you have never considered this question, or don't remember the answer. (Be careful, it is *not* $t2^{t-1}$.) We can plot the graph of the approximate derivative, using a step size of something like 0.0001. The

following code produces figure 14.1, the graphs of both $y = 2^t$ and its approximate derivative, $\dot{y}$.

```
t = 0:.01:2;
h = .00001;
y = 2.^t;
ydot = (2.^(t+h) - 2.^t)/h;
plot(t,[y; ydot])
```

The graph of the derivative has the same shape as the graph of the original function. Let's look at their ratio, $\dot{y}(t)/y(t)$.

```
plot(t,ydot./y)
```

We see that the ratio of the derivative to the function, shown in figure 14.2, has a constant value, $\dot{y}/y = 0.6932...$, that does not depend upon t .

Now, if you are following along with a live MATLAB, repeat the preceeding calculations with $y = 3^t$ instead of $y = 2^t$. You should find that the ratio is again independent of t. This time $\dot{y}/y = 1.0986...$. Better yet, experiment with expgui.

If we take any value a and look at $y = a^t$, we find that, numerically at least, the ratio $\dot{y}/y$ is constant. In other words, $\dot{y}$ is proportional to y. If $a = 2$, the proportionality constant is less than one. If $a = 3$, the proportionality constant is greater than one. Can we find an a so that $\dot{y}/y$ is actually equal to one? If so, we have found a function that is equal to its own derivative.

The approximate derivative of the function $y(t) = a^t$ is

$$\dot{y}(t) = \frac{a^{t+h} - a^t}{h}$$

This can be factored and written

$$\dot{y}(t) = \frac{a^h - 1}{h} a^t$$

So the ratio of the derivative to the function is

$$\frac{\dot{y}(t)}{y(t)} = \frac{a^h - 1}{h}$$

The ratio depends upon h, but not upon t. If we want the ratio to be equal to 1, we need to find a so that

$$\frac{a^h - 1}{h} = 1$$

Solving this equation for a, we find

$$a = (1 + h)^{1/h}$$

The approximate derivative becomes more accurate as h goes to zero, so we are interested in the value of

$$(1 + h)^{1/h}$$

as h approaches zero. This involves taking numbers very close to 1 and raising them to very large powers. The surprising fact is that this limiting process defines a number that turns out to be one of the most important quantities in mathematics

$$e = \lim_{h \to 0} (1 + h)^{1/h}$$

Here is the beginning and end of a table of values generated by repeatedly cutting h in half.

```
format long
format compact
h = 1;
while h > 2*eps
    h = h/2;
    e = (1 + h)^(1/h);
    disp([h e])
end
```

```
0.500000000000000    2.250000000000000
0.250000000000000    2.441406250000000
0.125000000000000    2.565784513950348
0.062500000000000    2.637928497366600
0.031250000000000    2.676990129378183
0.015625000000000    2.697344952565099
         . . .                 . . .
0.000000000000014    2.718281828459026
0.000000000000007    2.718281828459036
0.000000000000004    2.718281828459040
0.000000000000002    2.718281828459043
0.000000000000001    2.718281828459044
0.000000000000000    2.718281828459045
```

The last line of output involves a value of h that is not zero, but is so small that it prints as a string of zeros. We are actually computing

$$(1 + 2^{-51})^{2^{51}}$$

which is

$$(1 + \frac{1}{2251799813685248})^{2251799813685248}$$

The result gives us the numerical value of e correct to 16 significant decimal digits. It's easy to remember the repeating pattern of the first 10 significant digits.

$$e = 2.718281828...$$

Let's derive a more useful representation of the exponential function. Start by putting t back in the picture.

$$
\begin{aligned}
e^t &= \left(\lim_{h \to 0} (1 + h)^{1/h} \right)^t \\
&= \lim_{h \to 0} (1 + h)^{t/h}
\end{aligned}
$$

Here is the *Binomial Theorem.*

$$(a+b)^n = a^n + na^{n-1}b + \frac{n(n-1)}{2!}a^{n-2}b^2 + \frac{n(n-1)(n-2)}{3!}a^{n-3}b^3 + \ldots$$

If n is an integer, this terminates after $n+1$ terms with b^n. But if n is not an integer, the expansion is an infinite series. Apply the binonimial theorem with $a = 1$, $b = h$ and $n = t/h$.

$$\begin{aligned}(1+h)^{t/h} &= 1 + (t/h)h + \frac{(t/h)(t/h-1)}{2!}h^2 + \frac{(t/h)(t/h-1)(t/h-2)}{3!}h^3 + \ldots \\ &= 1 + t + \frac{t(t-h)}{2!} + \frac{t(t-h)(t-2h)}{3!} + \ldots\end{aligned}$$

Now let h go to zero. We get the power series for the exponential function.

$$e^t = 1 + t + \frac{t^2}{2!} + \frac{t^3}{3!} + \ldots + \frac{t^n}{n!} + \ldots$$

This series is a rigorous mathematical definition that applies to any t, positive or negative, rational or irrational, real or complex. The $n+1$-st term is $t^n/n!$. As n increases, the t^n in the numerator is eventually overwhelmed by the $n!$ in the numerator, so the terms go to zero fast enough that the infinite series converges.

It is almost possible to use this power series for actual computation of e^t. Here is an experimental MATLAB program.

```
function s = expex(t)
% EXPEX  Experimental version of EXP(T)
s = 1;
term = 1;
n = 0;
r = 0;
while r ~= s
   r = s;
   n = n + 1;
   term = (t/n)*term;
   s = s + term;
end
```

Notice that there are no powers or factorials. Each term is obtained from the previous one using the fact that

$$\frac{t^n}{n!} = \frac{t}{n}\frac{t^{n-1}}{(n-1)!}$$

The potentially infinite loop is terminated when `r == s`, that is when the floating point values of two successive partial sums are equal.

There are "only" two things wrong with this program – its speed and its accuracy. The terms in the series increase as long as $|t/n| \geq 1$, then decrease after n reaches the point where $|t/n| < 1$. So if $|t|$ is not too large, say $|t| < 2$, everything is OK; only a few terms are required and the sum is computed accurately. But

larger values of t require more terms and the program requires more time. This is not a very serious defect if t is real and positive. The series converges so rapidly that the extra time is hardly noticeable.

However, if t is real and negative the computed result may be inaccurate. The terms alternate in sign and cancel each other in the sum to produce a small value for e^t. Take, for example, $t = -20$. The true value of e^{-20} is roughly $2 \cdot 10^{-9}$. Unfortunately, the largest terms in the series are $(-20)^{19}/19!$ and $(-20)^{20}/20!$, which are opposite in sign and both of size $4 \cdot 10^7$. There is 16 orders of magnitude difference between the size of the largest terms and the size of the final sum. With only 16 digits of accuracy, we lose everything. The computed value obtained from `expex(-20)` is completely wrong.

For real, negative t it is possible to get an accurate result from the power series by using the fact that

$$e^t = \frac{1}{e^{-t}}$$

For complex t, there is no such easy fix for the accuracy difficulties of the power series.

In contrast to its more famous cousin, π, the actual numerical value of e is not very important. It's the exponential function

$$e^t$$

that's important. In fact, MATLAB doesn't have the value of e built in. Nevertheless, we can use

```
e = expex(1)
```

to compute an approximate value for e. Only seventeen terms are required to get floating point accuracy.

```
e = 2.718281828459045
```

If we revise the code to compute the numerator and denominator separately using integer arithmetic, we find a rational approximation to e with 17! in the denominator.

$$e = \frac{966858672404690}{355687428096000}$$

After computing `e`, you could then use `e^t`, but `exp(t)` is preferable.

The *logarithm* is the *inverse* function of the exponential. If

$$x = e^t$$

then

$$\log_e(x) = t$$

The function $\log_e(x)$ is known as the *natural* logarithm and is often denoted by $\ln x$. More generally, if

$$x = a^t$$

then

$$\log_a(x) = t$$

The function $\log_{10}(x)$ is known as the *common* logarithm. MATLAB uses `log(x)`, `log10(x)`, and `log2(x)` for $\log_e(x)$, $\log_{10}(x)$, and $\log_2(x)$.

The term *exponential growth* is often used informally to describe any kind of rapid growth. Mathematically, the term refers to any time evolution, $y(t)$, described by the differential equation

$$\dot{y} = ky$$

The solution to this equation is determined for all t by specifying the value of y at one particular t, usually $t = 0$.

$$y(0) = y_0$$

Then

$$y(t) = y_0 e^{kt}$$

Suppose, at time $t = 0$, we have a million *E. coli* bacteria in a test tube under ideal laboratory conditions. Twenty minutes later each bacterium has fissioned to produce another one. So at $t = 20$, the population is two million. Every 20 minutes the population doubles. At $t = 40$, it's four million. At $t = 60$, it's eight million. And so on. The population, measured in millions of cells, $y(t)$, is

$$y(t) = 2^{t/20}$$

Let $k = \ln 2/20 = .0347$. Then, with t measured in minutes and the population $y(t)$ measured in millions, we have

$$\dot{y} = ky, \quad y(0) = 1$$

Consequently

$$y(t) = e^{kt}$$

This is exponential growth, but it cannot go on forever. Eventually, the growth rate is affected by the size of the container. Initially at least, the size of the population is modelled by the exponential function.

Suppose, at time $t = 0$, you invest \$1000 in a savings account that pays 5% interest, compounded yearly. A year later, at $t = 1$, the bank adds 5% of \$1000 to your account, giving you $y(1) = 1050$. Another year later you get 5% of 1050, which is 52.50, giving $y(2) = 1102.50$. If $y(0) = 1000$ is your initial investment, $r = 0.05$ is the yearly interest rate, t is measured in years, and h is the step size for the compound interest calculation, we have

$$y(t + h) = y(t) + rhy(t)$$

What if the interest is compounded monthly instead of yearly? At the end of the each month, you get .05/12 times your current balance added to your account. The

same equation applies, but now with $h = 1/12$ instead of $h = 1$. Rewrite the equation as

$$\frac{y(t+h) - y(t)}{h} = ry(t)$$

and let h tend to zero. We get

$$\dot{y}(t) = ry(t)$$

This defines interest compounded *continuously*. The evolution of your investment is described by

$$y(t) = y(0)e^{rt}$$

Here is a MATLAB program that tabulates the growth of $1000 invested at 5% over a 20 year period , with interest compounded yearly, monthly, and continuously.

```
format bank
r = 0.05;
y0 = 1000;
for t = 0:20
   y1 = (1+r)^t*y0;
   y2 = (1+r/12)^(12*t)*y0;
   y3 = exp(r*t)*y0;
   disp([t y1 y2 y3])
end
```

The first few and last few lines of output are

t	yearly	monthly	continuous
0	1000.00	1000.00	1000.00
1	1050.00	1051.16	1051.27
2	1102.50	1104.94	1105.17
3	1157.63	1161.47	1161.83
4	1215.51	1220.90	1221.40
5	1276.28	1283.36	1284.03
..	....	....	....
16	2182.87	2221.85	2225.54
17	2292.02	2335.52	2339.65
18	2406.62	2455.01	2459.60
19	2526.95	2580.61	2585.71
20	2653.30	2712.64	2718.28

Compound interest actually qualifies as exponential growth, although with modest interest rates, most people would not use that term.

Let's borrow money to buy a car. We'll take out a $20,000 car loan at 10% per year interest, make monthly payments, and plan to pay off the loan in 3 years. What is our monthly payment, p? Each monthly transaction adds interest to our

current balance and subtracts the monthly payment.

$$
\begin{aligned}
y(t+h) &= y(t) + rhy(t) - p \\
&= (1+rh)y(t) - p
\end{aligned}
$$

Apply this repeatedly for two, three, then n months.

$$
\begin{aligned}
y(t+2h) &= (1+rh)y(t+h) - p \\
&= (1+rh)^2 y(t) - ((1+rh)+1)p \\
y(t+3h) &= (1+rh)^3 y(t) - ((1+rh)^2 + (1+rh) + 1)p \\
y(t+nh) &= (1+rh)^n y(0) - ((1+rh)^{n-1} + ... + (1+rh) + 1)p \\
&= (1+rh)^n y(0) - ((1+rh)^n - 1)/(1+rh-1)p
\end{aligned}
$$

Solve for p

$$
p = (1+rh)^n / ((1+rh)^n - 1)rhy_0
$$

Use MATLAB to evaluate this for our car loan.

```
y0 = 20000
r = .10
h = 1/12
n = 36
p = (1+r*h)^n/((1+r*h)^n-1)*r*h*y0
```

We find the monthly payment would be

```
p = 645.34
```

If we didn't have to pay interest on the loan and just made 36 monthly payments, they would be

```
y0/n
```

```
  = 555.56
```

It's hard to think about continuous compounding for a loan because we would have to figure out how to make infinitely many infinitely small payments.

Exercises

14.1 *e cubed.* The value of e^3 is close to 20. How close? What is the percentage error?

14.2 *expgui.*
(a) With `expgui`, the graph of $y = a^x$, the blue line, always intercepts the y-axis at $y = 1$. Where does the graph of dy/dx, the green line, intercept the y-axis?

(b) What happens if you replace `plot` by `semilogy` in `expgui`?

14.3 *Computing e.*
(a) If we try to compute $(1+h)^{1/h}$ for small values of h that are inverse powers of 10, it doesn't work very well. Since inverse powers of 10 cannot be represented exactly as binary floating point numbers, the portion of h that effectively gets added to 1 is different than the value involved in the computation of $1/h$. That's why we used inverse powers of 2 in the computation shown in the text. Try this:

```
format long
format compact
h = 1;
while h > 1.e-15
    h = h/10;
    e = (1 + h)^(1/h);
    disp([h e])
end
```

How close do you get to computing the correct value of e?
(b) Now try this instead:

```
format long
format compact
h = 1;
while h > 1.e-15
    h = h/10;
    e = (1 + h)^(1/(1+h-1));
    disp([h e])
end
```

How well does this work? Why?

14.4 *expex.* Modify `expex` by inserting

```
disp([term s])
```

as the last statement inside the `while` loop. Change the output you see at the command line.

```
format compact
format long
```

Explain what you see when you try `expex(t)` for various real values of `t`.

```
expex(.001)
expex(-.001)
expex(.1)
expex(-.1)
expex(1)
expex(-1)
```

Try some imaginary values of `t`.

```
expex(.1i)
expex(i)
expex(i*pi/3)
expex(i*pi)
expex(2*i*pi)
```

Increase the width of the output window, change the output format and try larger values of `t`.

```
format long e
expex(10)
expex(-10)
expex(10*pi*i)
```

14.5 *Instrument expex.* Investigate both the cost and the accuracy of `expex`. Modify `expex` so that it returns both the sum `s` and the number of terms required `n`. Assess the relative error by comparing the result from `expex(t)` with the result from the built-in function `exp(t)`.

```
relerr = abs((exp(t) - expex(t))/exp(t))
```

Make a table showing that the number of terms required increases and the relative error detioriates for large t, particularly negative t.

14.6 *Rational expex.* Make a version of `expex` that does not do any divisions, but returns two large integers that form the numerator and denominator of a rational approximation to e. Reproduce the result

$$e = \frac{966858672404690}{355687428096000}$$