Lesson 30: Fibonacci numbers

```
> restart;
```

Fourier series again

Last time I said I didn't know a nice formula for these integrals, which were coefficients in the Fourier series of $\frac{1}{2+\cos(x)}$.

```
> J:= 1/Pi * int(cos(k*t)/(2+cos(t)),t=0..2*Pi) assuming
  k::posint;
```

$$J := \frac{\int_0^{2\pi} \frac{\cos(k\,t)}{2+\cos(t)}\,\mathrm{d}t}{\pi} \qquad (1.1)$$

Well, now I do (and I should have remembered it, because it's a standard Math 300 calculation). The answer should be $\frac{2\left(\sqrt{3}-2\right)^k}{\sqrt{3}}$ for any positive integer k (that's why I assumed k is a **posint**, short for positive integer, rather than just **integer**: that formula wouldn't work for negative k).

```
> seq(simplify(J-2/sqrt(3)*(sqrt(3)-2)^k),k=0..10);
```

$$0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 \qquad (1.2)$$

It's interesting that Maple can do the calculation for any particular k but can't do it in general.

Efficient matrix powers

Last time we saw a way to calculate Fibonacci numbers using linear algebra.

```
> M := <<0,1>|<1,1>>;
  fib3:= proc(n) option remember;
    (M^(n-1))[2,2]
  end proc;
```

$$M := \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

$$fib3 := \textbf{proc}(n)\ \textbf{option}\ remember;\ (M\wedge(n-1))[2,2]\ \textbf{end proc} \qquad (2.1)$$

Doing it this way hides the details of the calculation in the inner workings of Maple's linear algebra machinery (which may be a good thing, because that machinery is usually quite efficient and fast). Still, it's worthwhile seeing what we can learn from thinking about how these things could be done.

What's an efficient way to calculate the n'th power of a matrix?
The most straightforward way is to multiply that many copies of the matrix together, but that's far from efficient: to get M^n you would need $n-1$ matrix multiplications. A better idea is called "repeated squaring". Thus to calculate M^{64}, we would first calculate M^2, then square that to get M^4, then square that to get M^8, then M^{16}, M^{32}, and M^{64}. It only required 6 squarings. To calculate M^{199}, we write 199 as a sum of powers of 2 (this is the base-2 representation of 199):

$199 = 1 + 2 + 4 + 64 + 128$ so $M^{199} = M M^2 M^4 M^{64} M^{128}$. To calculate M^n this way needs at most $2 \log_2(n)$ matrix multiplications.
A convenient way to program it is the following:

```
> mpow:= proc(n)
    option remember;
    if type(n,even)
    then mpow(n/2)^2
    elif type(n,odd)
    then mpow((n-1)/2)^2 . M
    end if;
  end proc:
  mpow(0) := <<1,0>|<0,1>>:
  mpow(1) := M:
```

```
> mpow(20) = M^20;
```

$$\begin{bmatrix} 4181 & 6765 \\ 6765 & 10946 \end{bmatrix} = \begin{bmatrix} 4181 & 6765 \\ 6765 & 10946 \end{bmatrix}$$

```
> mpow(n);
```

(the symbolic variable n is neither even nor odd, so nothing is returned).

```
> fib4:= n -> mpow(n-1)[2,2];
```

$$fib4 := n \rightarrow mpow(n-1)_{2,2}$$

```
> fib4(300); fib4(300)-fib3(300);
```

```
222232244629420445529739893461909967206666939096499764990979600
                                0
```

```
> ti:= time():
  fib3(10^7):
  time()-ti;
  fib3(10^7);
```

```
                    6.801
        ...Integer too large for display...                    (2.2)
```

```
> ti:= time():
  fib4(10^7):
  time()-ti;
```

```
                    6.864                                      (2.3)
```

```
> length(fib4(10^7));
```

```
                  2089877                                      (2.4)
```

Fibonacci identities

So far our Fibonacci functions work only for positive integers. I'd like something that works also for symbolic expressions, e.g. I'd like to be able to get F_{2n+3} as an expression in terms of F_n and

F_{n-1}. We can get this by a change in **mpow**. Note that $M^n = \begin{bmatrix} F_{n-1} & F_n \\ F_n & F_{n+1} \end{bmatrix}$

```
> M^4;
```

$$\begin{bmatrix} 2 & 3 \\ 3 & 5 \end{bmatrix}$$

```
> mpow := proc(n)
    option remember;
    if type(n, posint)
    then if type(n,even)
         then mpow(n/2)^2
         else M . mpow((n-1)/2)^2
         end if
    elif type(n, negint)
         then mpow(-n)^(-1)
    elif type(n,`+`)
         then mpow(op(1,n)) . mpow(n-op(1,n))
    elif type(n,`*`)
         then mpow(n/op(1,n))^op(1,n)
    else <<F[n-1],F[n]>|<F[n],F[n]+F[n-1]>>
    end if
  end proc:
  mpow(0) := <<1,0>|<0,1>>:
  mpow(1) := M:
```

To describe what this does in words:
 If **n** is a positive integer, **type(n, posint)** is true (**posint** means "positive integer") and **mpow** will do what the previous version did.
 If **n** is a negative integer, **type(n, negint)** is true (**negint** means "negative integer"), and **mpow** will use $M^{-n} = \left(M^n\right)^{-1}$.
 If **n** is a sum of terms, say $a + b$, **op(1,n)** will be the first one and **n - op(1,n)** will be the rest. Maple will use $M^{a+b} = M^a M^b$.
 If **n** is a product of terms, say $a\, b$, **op(1,n)** will be the first one and **n/op(1,n)** will be the rest. Maple will use $M^{a\,b} = \left(M^b\right)^a$. I'll do it this way instead of $\left(M^a\right)^b$ because integers usually come first in a product, and I'd rather do
$\left(M^n\right)^2$ than $\left(M^2\right)^n$.

 If **n** is none of those (e.g. if it is a name), Maple will return the matrix $\begin{bmatrix} F_{n-1} & F_n \\ F_n & F_{n-1} + F_n \end{bmatrix}$
.

```
> mpow(2*n);
```

$$\begin{bmatrix} F_{n-1}^2 + F_n^2 & F_{n-1} F_n + F_n (F_n + F_{n-1}) \\ F_{n-1} F_n + F_n (F_n + F_{n-1}) & F_n^2 + (F_n + F_{n-1})^2 \end{bmatrix} \quad (3.1)$$

```
> mpow(-1)= M^(-1);
```

$$\begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \quad (3.2)$$

```
> fib4(n);
```

$$F_n$$

```
> fib4(n+3);
```

$$3 F_n + 2 F_{n-1}$$

```
> fib4(n+m);
```

$$F_n F_{m-1} + (F_n + F_{n-1}) F_m$$

This could also be written as $F_{n+m} = F_n F_{m-1} + F_{n+1} F_m$. It's an important and useful identity for Fibonacci numbers.

Where does this come from?

```
> mpow(n), mpow(m), mpow(m+n);
```

$$\begin{bmatrix} F_{n-1} & F_n \\ F_n & F_n + F_{n-1} \end{bmatrix}, \begin{bmatrix} F_{m-1} & F_m \\ F_m & F_m + F_{m-1} \end{bmatrix},$$

$$\begin{bmatrix} F_{n-1} F_{m-1} + F_n F_m & F_{n-1} F_m + F_n (F_m + F_{m-1}) \\ F_n F_{m-1} + (F_n + F_{n-1}) F_m & F_n F_m + (F_n + F_{n-1}) (F_m + F_{m-1}) \end{bmatrix}$$

Saving and reading

We've made some complicated definitions that might be useful in other worksheets. You could use copy-and-paste from this worksheet, but there's a much more convenient way. We can save the definitions (and values of whatever variables we want) to a text file, and then read this in to Maple whenever we want to use them. (I'm commenting out this command with # so it won't actually execute - see below)

```
> # save mpow, fib4, M, "fibonacci.txt";
```

That will create a file named "**fibonacci.txt**" on your computer containing Maple statements defining **mpow** and **fib4** and **M**. It will put it in the "current folder". You can see which one that is with the **currentdir** command.

```
> currentdir();
                         "d:\test"                          (4.1)
```

You can also use currentdir to change to a different folder.

```
> currentdir("d:/m210");
                         "d:\test"                          (4.2)
> currentdir();
                         "d:\m210"                          (4.3)
```

If you prefer a different folder, you can put in the path together with the file name, e.g.

```
> # save mpow, fib4, M, "d:/m210/fibonacci.txt";
```

Note that the forward slash "/" is used instead of the backslash "\".
Then in any Maple session where you want to use these definitions:

```
> read "d:/m210/fibonacci.txt";
```

Unfortunately there's a bug that causes an incorrect definition of **mpow** to be saved.
The problem was in the line

```
else mpow((n-1)/2)^2 . M
```

where **save** left out the space between the `2` and the `.`, which caused Maple to interpret `2.` as a number when **read** read that line.

```
> FF:= x -> x^2 . M;
```

$$FF := x \to (x^2).M \tag{4.4}$$

```
> save FF, "d:/test/FF.txt";
```

```
> read "d:/test/FF.txt";
Error, on line 1, syntax error, missing operator or `;`:
FF := x -> x^2.M;
              ^
Error, while reading ``d:/test/FF.txt``
```

I've put a corrected copy of "fibonacci.txt" (containing these definitions and a few more from the next couple of lessons) on our web page, so you can download it.

Some Fibonacci puzzles

$F_{n+m} = F_n F_{m-1} + F_{n+1} F_m$ was only one of many interesting identities involving the Fibonacci numbers.
There is a whole journal, the Fibonacci Quarterly, devoted to Fibonacci numbers and related recurrence relations. Near the back of each issue is a problem section, containing problems at various levels of difficulty contributed by readers. Many of the problems in the elementary section can be solved quite readily using **mpow** and **fib**. Here are a few examples.

(1) Let H_n be any solution (in integers) of the recurrence $H_n = H_{n-1} + H_{n-2}$. Show that
$7\,H_n \equiv H_{n+15}$ **mod** 10.
This means that $7\,H_n$ and H_{n+15} have the same remainder when divided by 10, i.e. the same last digit in the decimal representation. Maple has a **mod** function, which returns the least nonnegative integer with the same remainder. For example:

```
> 37 mod 5;
```

$$2 \tag{5.1}$$

```
> -37 mod 5;
```

$$3 \tag{5.2}$$

It will also work on rational expressions or equations, in which case it operates on the coefficients.

```
> 37 * x^23 + 42 mod 5;
```

$$2\,x^{23} + 2 \tag{5.3}$$

```
> ((23 * x+17)/(12*x+7) = 23) mod 5;
```

$$\frac{3\,x+2}{2\,x+2} = 3 \tag{5.4}$$

```
> 23/(5*x+15) mod 5;
Error, the modular inverse does not exist
> 15/5 mod 5;
                              3                                   (5.5)
```

Maple commands introduced in this lesson:

type(..., odd)
type(..., even)
type(..., posint)
type(..., negint)
save
read
mod