Lab 03 — Scripts and Control Constructs

Math3101/5305   Session 1, 2017

March 10, 2017

If a computation requires more than just a few steps it is usually simplest to type the commands into a *source file*, also known as a *script*, and get Julia to execute them by giving the name of the file as the argument of the `include` function. The file extension `.jl` is used to indicate a Julia source file.

1. Open the script `sine.jl` using your preferred editor. The script defines a function that approximates $\sin x$ by summing the first N terms of its Taylor expansion:

$$\sin x \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots + (-1)^{N-1}\,\frac{x^{2N-1}}{(2N-1)!}.$$

The kth term in this sum is

$$a_k = (-1)^{k-1}\,\frac{x^{2k-1}}{(2k-1)!},$$

and since

$$\frac{a_{k+1}}{a_k} = \frac{(-1)^k x^{2k+1}}{(2k+1)!}\,\frac{(2k-1)!}{(-1)^{k-1}x^{2k-1}} = -\frac{x^2}{(2k+1)(2k)},$$

the terms can be computed efficiently using the recursion

$$a_1 = x \quad \text{and} \quad a_{k+1} = \frac{-x^2}{(2k+1)(2k)}\,a_k \quad \text{for } k \geq 1.$$

The script uses a *for-loop* to accumulate the sum in the variable s. Table 1 shows the sequence of values of `a` and `s` for the case $N = 4$ as the loop counter `k` takes the successive values 1, 2 and 3. The syntax `s += a` is a shorthand for `s = s + a`.

k	a	s
1	a_2	$a_1 + a_2$
2	a_3	$a_1 + a_2 + a_3$
3	a_4	$a_1 + a_2 + a_3 + a_4$

Table 1: The steps of the for-loop in the case $N = 4$.

(a) Start Julia from the same folder as the file `sine.jl`, and then run the script by typing the commands

```
julia> x, N = 0.4, 4
julia> include("sine.jl")
```

(b) How many terms N are needed for the Taylor approximation to yield $\sin x$ correct to double precision if $x = 0.4$? If $x = 1.4$?

(c) Is it possible to achieve full double precision accuracy if $x = 20.0$?

(d) Try running the script when x is complex.

(e) Write a script `cosine.jl` that carries out the analogous computations for the approximation

$$\cos x \approx 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots + (-1)^{N-1} \frac{x^{2N-2}}{(2N-2)!}.$$

2. The binomial coefficients satisfy the recurrence

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \quad \text{for } 1 \le k \le n-1,$$

with

$$\binom{n}{0} = \binom{n}{n} = 1.$$

The binomial coefficients can be read from *Pascal's triangle*:

$$\begin{array}{llllll}
1 \\
1 & 1 \\
1 & 2 & 1 \\
1 & 3 & 3 & 1 \\
1 & 4 & 6 & 4 & 1 \\
1 & 5 & 10 & 10 & 5 & 1 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots
\end{array}$$

Complete the script `pascal.jl` so that it prints out the first N rows of Pascal's triangle.

3.* Write an alternative version `pascal2.jl` of the previous script that uses only a one-dimensional array of length N. (Hint: use a for-loop that counts down rather than up.)

5. The source file `quadratic.jl` defines a function `quadroots` that returns the roots of a quadratic equation (with real coefficients). Observe what happens if you type

```
julia> include("quadratic.jl")
help?> quadroots
```

Write a script `test_quadratic.jl` designed to test the function. Think about the range of cases to be checked.

6. The *sieve of Erathosthenes* is a classical algorithm for finding all prime numbers up to a given number N:

1. Create a list of all whole numbers from 2 to N.
2. Strike out from the list all proper multiples of 2, that is, strike out 4, 6, 8, ...
3. The next number j that has not been struck out is prime. Strike out all proper multiples of j, that is, strike out $2j$, $3j$, $4j$, ...
4. Repeat step 3 until the next remaining number j is greater than $\sqrt{N}$.

Write a Julia script `sieve.jl` that implements this algorithm and prints out the list of prime numbers. (Hint: create an array `p` of such that `p[j]` is `true` iff `j` is prime.)

7. Write a source file `rpascal.jl` containing a recursive function `binom` that returns the value of the binomial coefficient $\binom{n}{k}$ if $0 \le k \le n$.