Lab Class 0

Math3101/5305

Term 2, 2023

1. Write a function

```
b = Pascal(N)
```

that computes the first $N+1$ rows of *Pascal's triangle*, storing the numbers in the lower triangle of an $(N+1)\times(N+1)$ matrix `b`. For example, if $N=5$ then

$$\mathtt{b} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 & 0 \\ 1 & 3 & 3 & 1 & 0 & 0 \\ 1 & 4 & 6 & 4 & 1 & 0 \\ 1 & 5 & 10 & 10 & 5 & 1 \end{bmatrix}.$$

In general,

$$\mathtt{b[n,k]} = \binom{n-1}{k-1} \quad \text{for } 1 \le k \le n \le N+1,$$

with

$$\mathtt{b[n,k]} = 0 \quad \text{for } 1 \le n < k \le N+1.$$

Use the recurrence relation

$$\mathtt{b[n,k]} = \mathtt{b[n-1,k-1]} + \mathtt{b[n-1,k]} \quad \text{for } 2 \le k \le n \le N+1.$$

Why is this approach better than using the formula

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}?$$

Note: Python programmers should put

$$\mathtt{b[n,k]} = \binom{n}{k} \quad \text{for } 0 \le k \le n \le N,$$

since Python arrays are indexed from zero. (You can do the same in Julia if you make `b` an `OffsetMatrix`.)

2. Given $a > 0$ and $b > 0$, we define the sequences a_0, a_1, a_2, ... and b_0, b_1, b_2, ... by first putting $a_0 = a$ and $b_0 = b$, and then computing

$$a_{n+1} = \tfrac{1}{2}(a_n + b_n) \quad \text{and} \quad b_{n+1} = \sqrt{a_n b_n} \quad \text{for } n \geq 0.$$

It can be shown that both sequences converge very rapidly to a common limit, called the *arithmetic–geometric mean* of a and b, denoted by

$$\operatorname{agm}(a, b) = \lim_{n\to\infty} a_n = \lim_{n\to\infty} b_n.$$

Gauss discovered a remarkable connection with the elliptic integral

$$I(a, b) = \int_0^{\pi/2} \frac{d\theta}{\sqrt{a^2 \cos^2\theta + b^2 \sin^2\theta}},$$

namely that $2\,I(a, b) = \pi \operatorname{agm}(a, b)$. One consequence is that the period of oscillation of a simple pendulum of length ℓ moving under a uniform gravitational acceleration g with initial angular displacement α is

$$T = \frac{T_0}{\operatorname{agm}(1, \cos\alpha/2)} \quad \text{where} \quad T_0 = 2\pi\sqrt{\frac{\ell}{g}}.$$

Here, T_0 is the well-known approximation to the period obtained by linearising the ODE for the angular displacement.

(i) Write a function `agm(a,b)` that returns the arithmetic–geometric mean of a and b. Use the stopping criterion

$$|a_n - b_n| < (a_n + b_n)\epsilon,$$

where ϵ is the machine epsilon.

(ii) Hence plot T as a function of α, when $\ell = 3$ and $g = 9.8$.

Note: Python programmers should import `finfo` and `float64` from `numpy`. The machine epsilon is then given by `finfo(float64).eps`.

3. The *tangent numbers* T_n arise in the Taylor expansion of the tangent function,

$$\tan\theta = \sum_{n=0}^{\infty} \frac{T_n \theta^n}{n!} \quad \text{for } |\theta| < \frac{\pi}{2}.$$

Thus,

$$T_n = (d/d\theta)^n \tan\theta|_{\theta=0},$$

and since $(d/d\theta)\tan\theta = 1 + \tan^2\theta$ a simple induction on $n \geq 0$ shows that there exists a sequence of polynomials P_n satisfying

$$(d/d\theta)^n \tan\theta = P_n(\tan\theta) \quad \text{and} \quad P_{n+1}(x) = (1 + x^2)P_n'(x).$$

From $P_0(x) = x$ we see that P_n has degree $n + 1$. Writing $P_n(x) = \sum_{k=0}^{n+1} a_{n,k} x^k$ and setting $a_{nk} = 0$ if $k < 0$ or $k > n + 1$, it follows that

$$a_{n+1,k} = (k - 1)a_{n,k-1} + (k + 1)a_{n,k+1} \quad \text{with} \quad a_{0,0} = 0 \text{ and } a_{0,1} = 1.$$

Using these relations we may compute

$$T_n = P_n(0) = a_{n,0}.$$

In fact, since $\tan\theta$ is an odd function, $t_n = 0$ if n is even, and moreover $a_{n,k} = 0$ whenever $n + k$ is even. Write a function `tangent_numbers(N)` that returns an array containing the first $N + 1$ tangent numbers T_0, T_1, T_2, ..., T_N.

Note: to debug your code, print out the values of $a_{n,k}$ as shown below.

	$k=0$	1	2	3	4	5	6
$n=0$	0	1					
1	1	0	1				
2	0	2	0	2			
3	2	0	8	0	6		
4	0	16	0	40	0	24	
5	16	0	136	0	240	0	120

Matlab programmers will have to store $a_{n,k}$ as `a(n+1,k+1)`.

Warning: the tangent numbers grow very rapidly. In fact, already $T_{25} > 2^{63} - 1$ and so cannot be represented as a standard 64-bit (signed) integer.

4. The *sieve of Eratosthenes* is a classical algorithm for finding all prime numbers less than or equal to a given number N:

1. Create a list of all whole numbers from 1 to N.
2. Strike out from the list the number 1.
3. The next remaining number j is prime. Strike out all proper multiples of j, that is, strike out $2j$, $3j$, $4j$, $\ldots$.
4. Repeat step 3 until the next remaining number j is greater than $\sqrt{N}$ (or equivalently, until $j^2 > N$).

For example, if $N = 20$ the steps of the algorithm look as follows.

$$1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20$$
$$\cancel{1}, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19.20$$
$$\cancel{1}, \mathbf{2}, 3, \cancel{4}, 5, \cancel{6}, 7, \cancel{8}, 9, \cancel{10}, 11, \cancel{12}, 13, \cancel{14}, 15, \cancel{16}, 17, \cancel{18}, 19, \cancel{20}$$
$$\cancel{1}, 2, \mathbf{3}, \cancel{4}, 5, \cancel{6}, 7, \cancel{8}, \cancel{9}, \cancel{10}, 11, \cancel{12}, 13, \cancel{14}, \cancel{15}, \cancel{16}, 17, \cancel{18}, 19, \cancel{20}$$

Here, j is shown in bold.

(i) Write a function

```
isprime = Eratosthenes(N)
```

that returns a boolean array of length N such that `isprime[j]` equals `true` iff the number j is prime ($1 \le j \le N$).

(ii) Hence write a second function `listprimes(N)` that returns an integer array consisting of all prime numbers less than or equal to N.

Note: Python programmers will find it easier to think in terms of a list of numbers from 0 to N. Start by setting `isprime = empty(N+1, dtype=bool8)`, where `empty` and `bool8` need to be imported from `numpy`.