NumPy and SciPy Worksheet

February 15, 2022

1 NumPy and SciPy

Further problems on using NumPy and SciPy functionalities.

1.1 NumPy

1. **Slicing**

Create a random 5×5 array, A, of integers in the range 1 to 10. (*Hint*: you may find the `randint` function helpful.) Use slicing to extract:

(a) the first two rows of A;

(b) the last two columns of A;

(c) and the 3×3 matrix in the "bottom right corner" of A. Call this new matrix A_{slice}. Now change the $[0, 0]$ entry of A_{slice} to any (different) integer you want, then print both the original matrix A and A_{slice}. What do you notice? Finally, create a new matrix A_{copy} *via* the following command

```
A_copy = np.array(A_slice)
```

and repeat the foregoing exercise of changing one of the values of A_{copy}, printing both A and A_{copy}.

2. **Array indexing**

It can sometimes be useful to use an array of indices to access or change elements. Create a 4×4 array called B with elements of your choosing. Now, create two new 4×1 arrays as follows:

```
row_indcs = np.array([0, 1, 2, 3])
```

```
col_indcs = np.array([0, 2, 3, 1])
```

Knowing what you know about indexing of matrices in NumPy, can you work out how to extract the `[0, 0]`, `[1, 2]`, `[2, 3]`, `and [3, 1]` entries of B? Using the `+=` command, add 1000 to these entries and print the result.

3. **Filtering**

Create a function that takes in an array, `mat`, and two parameters, `a` and `b`, and returns an array whose values are the entries of `mat` that are greater than `a` and less than `b`.

Test your function by creating a *Vandermonde matrix* of size no more than 10×10 and by choosing two parameters a and b. Print the result. Is it what you expect?

(*Hint*: use the NumPy function `np.vander()` to efficiently create the Vandermonde matrix.)

4. **Broadcasting**

Broadcasting is the name used for how NumPy deals with operations with arrays of different sizes. For example, broadcasting can be a useful tool by replacing the occurence of loops with efficient *vectorized* operations. However, broadcasting is subject to some constraints. From the NumPy documentation:

"When operating on two arrays, NumPy compares their shapes *element-wise*. It starts with the trailing dimensions and works its way forward. Two dimensions are *compatible* when

- they are equal, or
- one of them is 1''

Here are some functionining and non-functioning examples for you to try. In each of the following, compute the sum $(A + B)$ and element-wise multiplcation $(A * B)$ of the two arrays requested in each part of the question.

(a) Create a two-dimensional array, A, of dimension 8×3 and a one-dimensional array, B (or just a number).

(b) Create a *three-dimensional array* A of dimension $5 \times 6 \times 2$ and a two-dimensional array B of size 6×2.

Whats happens if

- in (b), A is of size $5 \times 3 \times 2$;
- instead of B, we take the transpose of B and apply the operations?

5. **Finding roots of polynomial in NumPy**

Research the np.roots function in NumPy to find the roots of the following polynomials.

(a) $x^2 - 2x + 1 = 0$

(b) $2x^3 + 6x^2 - 7x + 2 = 0$

(c) A 10-degree polynomial whose coefficients are an array of random integers.

(d) A 1000-degree polynomail whose coefficients are an array of random integers. (Note how fast you can do this!)

1.2 SciPy

6. Using solve_ivp, solve the differential equation $\mathrm{d}y/\mathrm{d}t = y$ subject to $y(0) = 1$ for $0 <$ t < 5. (The exact solution is of course $y = \exp(t)$).

7. **Solving a second order differential equation**

Using `solve_ivp`, solve the *damped harmonic oscillator*

$y'' + by' + \omega^2 y = 0$ subject to $y(0) = 1$ and $y'(0) = 0$ for $0 < t < 10$,

where b is the damping coefficients and ω is the natural frequency.

Hint: by defining $y' = u$, write the second-order differential equation above as a *system* of first-order differential equations for the unknows y and u (similar to the SIR example in the notes).

Recalling the syntax

```
sol = solve_ivp(func, t_span, y0, t_eval = np.linspace(0, tend, num = 101))
```

use the following code snippet to plot the *phase portrait* of the solution for different values of the damping parameter b:

```
plt.plot(sol.y[1,:], sol.y[0,:])

plt.xlabel('y')

plt.ylabel('dy/dt')

plt.title('Damped harmonic oscillator')
```

Do the results correspond to your physical intuition?

```
[ ]:
```