Homework 2

Due: Monday, 2/2/26, 23:59:59

Problem 1 (double points) — Gauss-Jordan elimination

Write a Python function `gauss_jordan` using the basic arrays defined by NumPy to perform Gauss-Jordan elimination on a square matrix. The routine should take in the matrix and return its inverse. It should have an optional parameter that controls whether it uses partial pivoting, so that you can compare the numerical stability of the method when it uses pivoting and when it doesn't. Be sure to check that your program is working by multiplying the inverse it produces by the original matrix and comparing to the identity matrix.

Suggestions

1. Matrix multiplication in NumPy uses the `@` symbol: `A @ B`
2. As you check your routine, it can be useful to have NumPy show array values as fractions, rather than the default floating point numbers. To display fractions, use the following code:

   ```
   from fractions import Fraction
   np.set_printoptions(formatter={'float': lambda x: str(Fraction(x).limit_denominator())});
   ```

   If you want to revert to floating-point output, call `np.set_printoptions(precision=5)`.

3. You may need to ensure that the data type of the matrix passed to your routine is floating (not integer). You can use a call such as

   ```
   m = np.asarray(m, dtype=np.float64)
   ```

Problem 2 (double points) — Numerical stability

Use your `gauss_jordan` function to investigate the numerical instability of Gauss-Jordan elimination when you do and you don't use pivoting. For test cases, use matrices of pseudorandom numbers generated by a call such as

```
from numpy.random import default_rng
rng = default_rng()
rng.uniform(<lower_bound>, <upper_bound>, size=(<dim>, <dim>))
# For example
rng.uniform(-5, 5, size=(5,5))
array([[ 4.23959642,  3.60732447, -0.16149884,  3.6900891 ,  2.65585838],
       [ 1.97695378, -3.09057092, -4.10553997, -4.78785912,  1.44810866],
       [-4.27088306, -4.36821066, -2.14689027, -0.29851331, -4.93396243],
       [-1.99115728, -1.63366415, -3.04232138, -1.73824459,  3.46747273],
       [-4.71288883, -1.94345835, -1.96308442, -1.07039782,  2.43547367]])
```

One way to explore the numerical error of the inverse is to multiply the matrix by its inverse, subtract the identity matrix, and then report some statistic on the elements of the remaining matrix (which would be all zeros if there were no error). Be sure to explain your choice for **figure of merit** (although it is really a figure of *demerit*!).

Prepare a plot showing your figure of merit as a function of matrix dimension when you use pivoting and when you don't and summarize your conclusion. (Your plot should make very clear why you need to use pivoting!)

Suggestions

1. Errors typically depend on the matrix dimension N to some power. Therefore, it is appropriate to plot on logarithmic axes and to space values of N suitably. Think carefully about how to do this.
2. The smallest value of N for your plot might be something like 256.
3. For each value of N that you choose, you should perform multiple trials (at least 5) and compute the mean and standard deviation of the mean of the values of your chosen figure of merit. Note that `scipy.stats` has the function `sem` (standard error of the mean) that you can import with the statement `from scipy.stats import sem`. Or, you can compute it yourself from the standard deviation.
4. As N gets large, it can take quite some time to run your `gauss_jordan` routine. You might wish to have an indicator to show you how the run is progressing. A simple one to use is `tqdm`, which you can install using pip. Once installed, import with the statement `from tqdm.notebook import tqdm`. To use tqdm in a for loop, just surround the iterator with `tqdm()`. For example,

   ```
   for n in tqdm(range(nreps), f'Testing inversion for N = {dim}'):
   ```

5. To plot using error bars in matplotlib, use something like `ax.errorbar(x, y, yerr=err, fmt='o', capsize=3)`, where `ax` is an existing axes object from `plt.subplots()`.

In [ ]: