MATM063: Principles of Data Science, Python LAB

Worksheet 2 (Week 2)

MATM063: Python basics

Werner Bauer

Department of Mathematics

Key Learning: Matrix operations, Solving matrix problems, Pandas basics

1 SOLVING ITERATION PROBLEMS

Sequences can be generated by an iteration of the form $x_{n+1} = f(x_n)$ for a given initial value x_0. For example, for the iteration

$$x_{n+1} = \frac{1}{2}x_n + 1,$$

taking $x_0 = 4$ gives

$$x_1 = \frac{1}{2}x_0 + 1 = 3, x_2 = \frac{1}{2}x_1 + 1 = 2.5, x_3 = \frac{1}{2}x_2 + 1 = 2.25\ldots$$

We can generate this sequence using a `for` loop:

```
import numpy

x = numpy.zeros(10)
x[0] = 4 # define the first term as 4

for n in range(0,len(x)-1): # n goes from 0 to len(x)-1 (so n+1 goes from 1 to len(x))
      x[n+1] = 0.5*x[n] + 1 # calculates and stores the next term

print(x) # prints the result
# Output:
[4.         3.         2.5        2.25       2.125
 2.0625     2.03125    2.015625   2.0078125 2.00390625]
```

Calculating more then 10 terms, e.g. 50, one can verify that the sequence converges to 2.

NOTE: using a `while` rather than a `for` loop would allow us to stop the iteration as soon as a certain tolerance is reached (see Worksheet 1, Section 7, for more details).

2 SOLVING EQUATIONS NUMERICALLY

We can use `scipy.optimize.fsolve(f,a)` to solve the roots of $f(x) = 0$ near an initial guess of the root which is a. To find a zero of a function g in an interval $[a, b]$ where the sign of $g(a)$ and $g(b)$ is different, we can use `scipy.optimize.brentq(g,a,b)` (see python docu.). Both work for functions with one variable while best results are obtained for an inital guess that is close to the solution $f(x) = 0$ (maybe plot the graph to identify these guesses). Example: let's solve

$$x^2 + y^2 = 1$$
$$y - x = 0.$$

To use `fsolve`, we must first define a vector function of multiple variables, such that the equations consist of the function equated to zero. Thus to solve the preceding equations we define two functions

$$f_0(x_0, x_1) = x_0^2 + x_1^2 - 1$$
$$f_1(x_0, x_1) = x_1 - x_0,$$

where we have replaced x with x_0 and y with x_1, so that we are now looking for a solution to the equations $f_0(x_0, x_1) = 0$ and $f_1(x_0, x_1) = 0$. To solve these with `fsolve` we define a function f with components f_0 and f_1 and may then apply `fsolve` as before:

```
import numpy
import scipy

def f(x):
```

```
    f0 =  x[0]**2 + x[1]**2 - 1
    f1 = x[1] - x[0]
    return numpy.array([f0,f1]) # returns a vector outputting f0 and f1

print(scipy.optimize.fsolve(f,numpy.array([1,1])))
# prints a solution of the equations with initial guess x[0] = 1, x[1] = 1

print(scipy.optimize.fsolve(f,numpy.array([-1,-1]))) # note there is another solution
```

Note: the function f takes a list or `numpy` array `x` of two numbers as input and returns f_0 and f_1 applied to these numbers. In this case we must guess an initial value for both $x[0]$ and $x[1]$, which can be input into `fsolve()` as a `numpy.array` or a list.

3 SOME NUMPY (MATRIX) OPERATIONS

Here is a very short list of operations to be performed on (numpy) matrices (if suitable) provided by numpy: given an $n \times n$ matrix A and an n-vector x:

```
numpy.sum(x) # sum over elements of vector x
numpy.transpose(A) # transpose of A
numpy.trace(A) # calculates the trace of A
numpy.det(A) # determinant of A
numpy.mean(A) # mean over the entries of A
numpy.linalg.inv(A) # calculate inverse of A
numpy.linalg.matrix_power(A,n) # calculates the n-th power of A
[E,V] = numpy.linalg.eig(A) # assigns eigenvalues to E and eigenvectors to V
A@x # matrix multiplication of A and x; returns 1D array

v = numpy.array([1,2,3,4,5])
u = numpy.array([-4**3,-3,-2,-1,0])
print(numpy.concatenate((u,v))) # Output:
[-64  -3  -2  -1   0   1   2   3   4   5]

numpy.dot(v,u) # scalar product of u and v # Output:
-80
```

4 SOLVING MATRIX VECTOR EQUATIONS

Python can solve systems of linear equations. The solution of the linear system $A\mathbf{x} = \mathbf{b}$ is $\mathbf{x} = A^{-1}\mathbf{b}$ whenever A is invertible. This can be used to solve small systems of linear equations.

```
A = numpy.array([[1,2,3],[2,3,1],[3,1,2]]) # define A
b = numpy.array([[2],[1],[9]]) # define a column vector b
x = numpy.linalg.inv(A) @ b
print(x)
[[ 3.]
[-2.]
[ 1.]] # output

print(A@x ) # check Ax = b
print(b)
```

Finding the inverse of a large matrix is usually very computationally expensive and there are more efficient methods of solving large systems of linear equations such as Gaussian elimination (as studied in Linear Algebra). One such method is implemented in `numpy.linalg.solve()`.

```
numpy.linalg.solve(A,b) # solves Ax = b for x
print(numpy.linalg.solve(A,b)) # prints the output
[[ 3.]
[-2.]
[ 1.]] #output
```

This also works if we implement b as a 1D array, that is if `b = numpy.array([2,1,9])`.

5 PANDAS BASICS

For our course, we will use Pandas to work with datasets. Here is the documentation `https://pandas.pydata.org/` for Pandas. We import the external package with e.g.:

```
import pandas as pd
```

In the following, you find a list of some basic pandas commands:

```
# Series and Data frame objects:
data1 = pd.Series(['a', 'b', 'c'], index=[1, 3, 5])
data2 = pd.Series(['d', 'e', 'f'], index=[2, 4, 6])
df   = pd.DataFrame({'d1':data1, 'd2':data2})
print(df)
# Output:
    d1   d2
1    a  NaN
2  NaN    d
3    b  NaN
4  NaN    e
5    c  NaN
6  NaN    f

# Column wide operations: add new column 'd3' with ration of d1/d2
df['d3'] = df['d1'] / df['d2']

# delete a column, e.g. 'd1' by:
df.pop('d1')
```

6 ACCESSING DATAFRAME ENTRIES

Examples on how to access elements DataFrames:

```
df.loc[1:2,:] # explicit indexing (last element included)
# Output:
    d1   d2
1    a  NaN
2  NaN    d

df.iloc[1:2,:] # impl. idx (last element not included)
# Output:
    d1 d2
2  NaN  d

df['d1'].loc[2] # value in column 'd1' and expl. idx 2
# Output:
nan
```

7 FINDING AND FILLING NULL VALUES

Pandas provides functions (isnull() and notnull() dropna() fillna()) to find and modify null values. Null values are a problem when e.g. training a machine learning model. Hence we have to preparte the data set by detecting these null values and then either delete the rows or columns or by filling them with suitable values. Here are a short list of pandas commands that help us to do so:

```
data = pd.DataFrame({'A':[1,2,np.nan], 'B':[5,np.nan,np.nan], 'C':[1,2,3]})
data.isnull() # detects NaN or None values
data.notnull() # detects values that are NOT Nan or None
data.dropna() # to drop null values

data.fillna(0) # fill null values with zero
data.fillna(method = 'ffill') # use filling-method 'forward-fill'
data['A'].fillna(value=data['A'].mean()) # filling missing values with column mean
```

NOTE: use `inplace = True` option to permanently modify data!

8 EXERCISES

Question 1: Defining matrices and performing operations

1. Define the matrix

$$A = \begin{pmatrix} 1 & 4 & 2 \\ 3 & 5 & 7 \\ 4 & 3 & 2 \end{pmatrix}.$$

   Find the mean of each column of this matrix using `numpy.mean()`. Which column has the largest mean?

2. Define a four by four matrix A and find its transpose, trace, determinant and inverse.

Question 2: Solving Matrix-vector systems

Find the solution of the following systems of linear equations. Check your answer for each by evaluating $A\mathbf{x} - \mathbf{b}$, where **x** is the unknowns and **b** is the right hand side.

1. System 1:

$$3x + 4y = 5$$
$$-x + 2y = 3$$

2. System 2:

$$4x - 5y + 2z = 7$$
$$8x + 5y - 4z = 10$$
$$3x - 2y + z = 5$$

Question 3: Pandas: treating `NaN` and `None` values

Assume, we have the data frame `df` as given below:

```
df = pd.DataFrame([[np.nan, 2, None, 0],
                   [3, 4, np.nan, 1],
                   [None, np.nan, np.nan, np.nan],
                   [np.nan, 3, np.nan, 4]],
                  columns=list("ABCD"))
```

Then, illustrate how to fill all missing values (for 1-3 without altering df):

1. with 0s;
2. using the forward and backward method;
3. filling all missing values per column with the column mean;
4. replace in `df` all missing values in columns A, B, C, D with 1,2,3,4, respectively (here, make sure to actually modify `df`).

Please provide the corresponding outputs.

Question 4: Pandas: working with Series and DataFrames

A fruit shop sells in 1 day on average (in kg) (20 bananas, 35 apples, 11 pears, 5 pineapples) for a price per kilo of (1.9, 3.5, 4.2, 6.7), respectively.

1. Create to data series named `nmb_sold` and `price_kg` and create a data frame where the keys share the names of these data series.
2. Add a column named `income` that presents the money the shop earned through these sellings for each fruit type.
3. Calculate the total income earned per day by selling fruits and output the following line: The total daily income by selling fruits is ??? Pounds.
4. Add another two rows to the table with average sellings of: Tomatos, 15 kg, and Potatos, 28 kg (price per kg: 2.9 and 1.4, respectively). Create first a new series, then a dataframe, and then add this to the existing dataframe.
5. For this extended data frame: determine also here the income per vegetable and recalculate and output the total daily income for selling both fruits and vegetables.