math 1MP assignment 2

Ben Bolker

22:30 01 February 2017

Due Wednesday 8 February at midnight (the end of the day), in the Dropbox on Avenue to Learn.

- Your assignment *must* be submitted as a module (text file) called `yourmacid_hw2.py`, e.g. mine would be `bolker_hw2.py`.
- All of your functions *must* have docstrings.

1. Write an implementation of *Eratosthenes' sieve*, an efficient way to find all the primes in a specified range. It must be defined as `def eratos(n):`, where `n` will be an integer
   - **initialization:**
     - create a list of boolean values of length `n+1` (we want to check primes up to a value of `n` inclusive; we won't do anything with positions 0 and 1, but it's easier to index this way), which are all initially `True`
     - initialize a counter to 2
     - initialize an empty list for the results
   - start a `while` loop that runs as long as the the counter defined above is less than or equal to `n`
     - add the current value of the counter to the results list
     - set the values of the boolean list to `False` for all multiples of the current counter value <= `n`; you can do this with a `while` or a `for` loop. For example if `i`=3 and `n`=10 you should set elements 6 and 9 of the list to `False`
     - search forward for the next possibly-prime value:
       * first increment the counter by 1
       * while the counter is less than or equal to `n` and the corresponding element of the boolean list is `False`, increment the counter by 1.
   - return the results list

Tests (should all be `True`):

```
print(eratos(1) == [])
print(eratos(2) == [2])
print(eratos(5) == [2,3,5])
print(len(eratos(1000)) == 168)
print(len(eratos(100000)) == 9592)
```

2. Write a function `iterate` with the following arguments:

- `f`: a function
- `start`: a numeric starting value
- `tol`: a numerical tolerance (default value `1e-6`)
- `itmax`: a maximum number of iterations (default value 1000)

Starting from the initial value, your function should keep repeating calls to the function (e.g. `y=f(y)` as in the `repeat_fun` function in the class notes) until the absolute value of `f(y)-y` is less than `tol` *or* the number of iterations is equal to `itmax`. For example, if `start=1.01`, `f` is `math.sqrt`, and `tol=1e-4`, the sequence would look like this:

```
i y                   f(y)                f(y)-y
0 1.01                1.004987562112089   -0.005012437887911059
1 1.004987562112089   1.0024906793143211  -0.00249688279776783
2 1.0024906793143211  1.0012445651859097  -0.0012461141284114685
3 1.0012445651859097  1.0006220890955335  -0.0006224760903761339
4 1.0006220890955335  1.0003109961884522  -0.0003110929070813176
```

```
5 1.0003109961884522 1.0001554860062771 -0.00015551018217507817
6 1.0001554860062771 1.0000777399813863 -7.77460248908568e-05
```

On step 6, the absolute value of the difference is less than the tolerance (`1e-4`), so the function returns `[6, 1.0000777399813863]`.

Tests (should all be `True`):

```
def approx_equal(x,y,tol=1e-8):
    """helper function: test whether all elements of
       x and y are equal within tolerance
    """
    if len(x) != len(y):
       return(False)
    for i in range(len(x)):
       if (abs(x[i]-y[i])>tol):
           return(False)
    return(True)

def disc_logist(x):
    """discrete logistic function"""
    return(1.5*x*(1-x))
print(approx_equal(disc_logist,0.5),[15, 0.33333433255312184])
print(approx_equal(iterate(disc_logist,0.5,tol=1e-8),[22, 0.33333334113969143]))
def half(x):
    """just what it says"""
    return(x/2)
print(approx_equal(iterate(half,1000),[29, 9.313225746154785e-07]))
import math
print(approx_equal(iterate(math.sqrt,1.01,tol=1e-4),[6, 1.0000777399813863]))
print(approx_equal(iterate(math.cos,0),[34, 0.7390855263619245]))
print(approx_equal(iterate(math.cos,0,tol=1e-8),[46, 0.7390851366465718]))
print(approx_equal(iterate(math.cos,0,itmax=5),[5, 0.7013687736227565]))
```

3. Write a function `read_evens` that takes a single argument, the name of a file in the current directory. The file will contain a single integer on each line. Your function should return the number of *even* integers (remember that when you read a line it is a string (`str`) that you can convert to an integer with `int()`).

Tests: put the file `even_nums.txt` in the same directory as your Python file (in your project directory, if you're using PyCharm). Then `read_evens("even_nums.txt")` should return 3.