HOMEWORK 1

LIAM DUNGAN

Problem 1. What are the approximate absolute and relative errors in approximating π by each of the following quantities?

a.) 3
b.) 3.14
c.) 22/7

You can use either single or double precision for your computations. Please state your choice.

Solution. Using single precision
(a) abs err: 0.1415927 rel err: 4.5070369percent
(b) abs err: 0.0015927 rel err: 0.0506985perent
(c) abs err: 0.0012644 rel err: 0.0402472percent

```
#Problem 1
#Using single precision

pi32 = np.float32(math.pi)

absErr = np.float32(abs(pi32-3.))
print("a    abs err: %.7f   rel err: %.7fpercent" % (absErr,(absErr/pi32)*100))

absErr = np.float32(abs(pi32-3.14))
print("b    abs err: %.7f   rel err: %.7fperent" % (absErr,(absErr/pi32)*100))

absErr = np.float32(abs(pi32-(22./7)))
print("c    abs err: %.7f   rel err: %.7fpercent" % (absErr,(absErr/pi32)*100))
```

Problem 2. In either single or double precision, is the machine epsilon the smallest number ε that can be stored on the computer, such that $1+\varepsilon\neq 1$? Justify your answer.

Solution. Yes, for single and double precision, the machine epsilon is the smallest number that can be stored on the computer in the given format, such that $1+\varepsilon\neq 1$. In other words, ε is the difference between 1 and the next larger, storable number. For single precison, $\varepsilon = 2^{-24}$ which is in fact the smallest storable number for that format.

i.e. in binary

$$0.00000000000000000000000001$$

This is because for single precision binary numbers are stored using 23 binary digits and for double precision 52 binary digits.

Problem 3. Write a program to compute the absolute and relative errors in Stirling's approximation

$$n!\approx\sqrt{2\pi n}(n/e)^n$$

Date: Oct. 3, 2017.

for n=1,2,. . . ,10. Does the absolute error grow or shrink as n increases? Does the relative error grow or shrink as n increases? Is the result affected when using double precision instead of single precision?

Solution. Both the results and code are shown in the screenshots below. Note the relative error is expressed in percentage form. You will see that as n increases from 1:10, **the absolute error increases while the relative error decreases**. The following results were obtained using defualt double precision float, however, when performed using single precision (np.float32) the results are not drastically affected.

```
n       Stirling        Factorial       Abs Err                 Rel Err
1       0.922137008896  1               0.0778629911042                 8.44375514192
2       1.91900435149   2               0.080995648511          4.22071208167
3       5.83620959135   6               0.163790408654          2.80645179188
4       23.5061751329   24              0.493824867107          2.10083037463
5       118.019167958   120             1.98083204241           1.67839858278
6       710.078184642   720             9.92181535782           1.39728491487
7       4980.39583161   5040            59.6041683875           1.19677572632
8       39902.3954527   40320           417.604547343           1.04656510619
9       359536.872842   362880          3343.12715805           0.929842642182
10      3598695.61874   3628800                 30104.381259            0.83653591324
```

```python
#problem 3
def stirling(temp):
    return math.sqrt(2*math.pi*temp)*(temp/math.e)**temp

print "n","\t", "Stirling","\tFactorial","\tAbs Err","\t\tRel Err"
n = 1
for x in range (1,11):
        n *= x
        absErr = abs(stirling(x)-n)
        relErr = (absErr/stirling(x))*100
        print x,"\t", stirling(x), "\t", n, "\t\t", absErr,"\t\t", relErr
```

Problem 4. Let $x \in \mathbb{R}^n$ be an n-dimensional vector. Show that $\|x\|_2$ and $\|x\|_\infty$ are equivalent.

Solution. Two vector norms $\| x \|_a$ and $\| x \|_b$ are called equivalent if there exist real numbers c,d>0, such that c‖x‖ a≤ ‖x‖b ≤ ‖x‖a.

$$\| x \|_\infty = max_{i=1}^{n} \mid x_i \mid \leq \sqrt{\sum_{i=1}^{n} x_i^2} = \| x \|_2$$

$$\| x \|_\infty \leq \| x \|_2 \leq n \| x \|_\infty$$

Consider the vector v = (1,2,3,4,5)

$$\| x \|_\infty = 5 \leq \| x \|_2 \approx 7.4 \leq 5 * \| x \|_\infty = 25$$

$$5 \leq 7.4 \leq 25$$

It follows from the definition the two norms are equivalent, as proven above.

Problem 5. Consider the image blurring example discussed in class, and suppose we denote the matrix of grayscale pixel values as I. Modify the Python script $_{blur.py}$ to use the following operation instead:

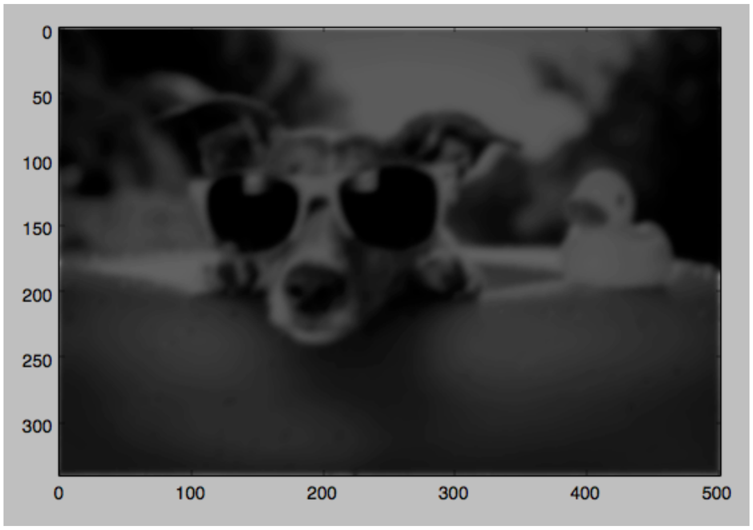
$$*See Assignment*$$

Compute the blurred image after 20 iterations of this modified scheme.

Solution. The modified code and resulting image is shown below. Also note the code is available in 5.py.

```python
# blur.py
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from scipy.sparse import lil_matrix

# read image file
fname = 'chill.jpg'
image = Image.open(fname).convert("L")
arr = np.asarray(image)
arr.setflags(write = 1)

# initialize blurring matrix
m = arr.shape[0]
n = arr.shape[1]
dofs = m*n
A = lil_matrix((dofs,dofs))
A.setdiag(np.ones(dofs))
for i in range(1,m-1):
    for j in range(1,n-1):
        A[n*i+j,n*i+j] = 1./2.
        A[n*i+j,n*(i-1)+j] = 1./16.
        A[n*i+j,n*(i+1)+(j-1)] = 1./16.
        A[n*i+j,n*(i-1)+(j+1)] = 1./16.
        A[n*i+j,n*i+(j-1)] = 1./16.

        A[n*i+j,n*i+(j+1)] = 1./16.
        A[n*i+j,n*(i+1)+j] = 1./16.
        A[n*i+j,n*(i+1)+(j-1)] = 1./16.
        A[n*i+j,n*(i+1)+(j+1)] = 1./16.
A = A.tocsr()
```

```python
# Blurring function - converts image to a vector, multiplies by
# the blurring matrix, and copies the result back into the image
def blur():
    x = np.zeros(shape=(dofs,1))
    for i in range(0,m):
        for j in range(0,n):
            x[n*i+j] = arr[i,j]

    y = A.dot(x)
    for i in range(0,m):
        for j in range(0,n):
            arr[i,j] = y[n*i+j]

# Execute the blurring function 20 times
for i in range(0,20):
    blur()

# Display the blurred image
plt.imshow(arr,cmap='gray')
plt.show()
```


0
50
100
150
200
250
300
0
100
200
300
400
500