Copyright (C) 2018, 2012, 2016 by Pearson Education Inc. All Rights Reserved, please visit www.pearsoned.com/permissions/.

CP.2.7.1.a, Sauer3

Implement Newton's Method with appropriate starting points to find all solutions. Check with EX.2.7.3 to make sure your answers are correct.

$$\text{(a)}\begin{cases} u^2+v^2 &= 1 \\ (u-1)^2+v^2 &= 1 \end{cases}$$

Hint from Sauer from his solution as in EX.2.7.3:

a. The curves are circles with radius 1 centered at $(u,v)=(0,0)$ and $(1,0)$, respectively. Solving the first equation for v^2 and substituing into the second yields $(u-1)^2+1-u^2=1$ or $-2u+1=0$, so $u=\frac{1}{2}$. The two solutions are

$$\begin{pmatrix} u_1 \\ v_1 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ \frac{\sqrt{3}}{2} \end{pmatrix}, \quad \text{and} \quad \begin{pmatrix} u_2 \\ v_2 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ -\frac{\sqrt{3}}{2} \end{pmatrix}.$$

Special Instructions:

a. Please first compute the two solutions using either Sauer's exact solutions (see Hint above) or `scipy.optimize.fsolve` or `scipy.optimize.root`. This will be useful to compute the forward error.

b. Start Newton's method from a relative distance (as measured by the infinity norm) of at least 0.1 from the solution.

c. At each iteration of Newton's method, you must print:

   (a) k, the iteration number

   (b) the absolute backward error at iteration number k defined by

   $$\|F(x_k)\|_\infty$$

   where x_k is the current iterate.

   (c) the relative forward error at iteration number k defined by

   $$\|x_k - x\|_\infty / \|x\|_\infty$$

   where x_k is the current iterate and x is the solution as computed by `scipy.optimize.fsolve` or `scipy.optimize.root`.

You can also print the current iterate x_k if you want.

Copyright (c) 2021, Julien Langou. All rights reserved, please visit https://creativecommons.org/licenses/by/4.0/.

EX.2.7.2.a, Sauer3, solution, Langou

Colab: `https://colab.research.google.com/drive/1PRQ6gpiXodEJBQhQRzifdhheXmzb7ehJ`

```python
from math import sqrt
import numpy as np
import scipy
```

```
from scipy.optimize import fsolve
from scipy.optimize import root
```

```
F = lambda x : np.array( [ x[ 0 ]**2 + x[ 1 ]**2 - 1.,\
                           ( x[ 0 ] - 1. )**2 + x[ 1 ]**2 - 1. ] )
```

```
# Per EX.2.7.3, we have two exact solution x1 and 2 such that
x1 = np.array( [ 1./2.,   sqrt(3.)/2. ] )
x2 = np.array( [ 1./2., - sqrt(3.)/2. ] )

# Let us check that || F(x1) ||_oo and || F(x2) ||_oo are small
print( "|| F(x1) ||_oo =", f"{np.linalg.norm(F(x1),np.infty):.2E}" )
print( "|| F(x2) ||_oo =", f"{np.linalg.norm(F(x2),np.infty):.2E}" )
```

```
|| F(x1) ||_oo = 1.11E-16
|| F(x2) ||_oo = 1.11E-16
```

```
# using scipy.optimize.fsolve to find two approximate solutions

# x_ is the exact solution that we are trying to find, it is either
# x1 or x2, as per EX.2.7.3. x_ is useful to compute the forward error.

# x is the initial guess, we find that
#        starting with [0, 1], scipy.optimize.fsolve converge to x1,
#        starting with [0,-1], scipy.optimize.fsolve converge to x2
# so  x = [0, 1] and x_ = x1 is what we use for the first iteration,
# and x = [0,-1] and x_ = x2 is what we use for the second iteration

for x, x_ in zip( [ np.array( [ 0., 1. ] ), np.array( [ 0., -1. ] ) ],
                  [ x1, x2 ] ):


  x = scipy.optimize.fsolve( F, x )

  print( "\nx = [", f"{x[0]:+20.16f}", "]"\
        "     x* = [", f"{x_[0]:+20.16f}", "]")
  print( "    [", f"{x[1]:+20.16f}", "]"\
         "           [", f"{x_[1]:+20.16f}", "]")
  print( "forward  error: || x - x* ||_oo = ",\
 f"{np.linalg.norm(x-x_,np.infty):7.2e}",\
         "\nbackward error: || F(x) ||_oo    = ",\
 f"{np.linalg.norm(F(x),np.infty):7.2e}")
```

```
x = [  +0.5000000000000000 ]     x* = [  +0.5000000000000000 ]
    [  +0.8660254037844409 ]          [  +0.8660254037844386 ]
forward  error: || x - x* ||_oo =  2.33e-15
backward error: || F(x) ||_oo    =  4.00e-15

x = [  +0.5000000000000000 ]     x* = [  +0.5000000000000000 ]
    [  -0.8660254037844409 ]          [  -0.8660254037844386 ]
forward  error: || x - x* ||_oo =  2.33e-15
backward error: || F(x) ||_oo    =  4.00e-15
```

```
# using scipy.optimize.root to find two approximate solutions
for x, x_ in zip( [ np.array( [ 0., 1. ] ), np.array( [ 0., -1. ] ) ],
                  [ x1, x2 ] ):


  x = scipy.optimize.root( F, x ).x

  print( "\nx = [", f"{x[0]:+20.16f}", "]"\
         "     x* = [", f"{x_[0]:+20.16f}", "]")
  print( "    [", f"{x[1]:+20.16f}", "]"\
         "           [", f"{x_[1]:+20.16f}", "]")
  print( "forward  error: || x - x* ||_oo = ",\
 f"{np.linalg.norm(x-x_,np.infty):7.2e}",\
         "\nbackward error: || F(x) ||_oo   = ",\
 f"{np.linalg.norm(F(x),np.infty):7.2e}")
```

```
x = [  +0.5000000000000000 ]     x* = [  +0.5000000000000000 ]
    [  +0.8660254037844409 ]          [  +0.8660254037844386 ]
forward  error: || x - x* ||_oo =  2.33e-15
backward error: || F(x) ||_oo   =  4.00e-15

x = [  +0.5000000000000000 ]     x* = [  +0.5000000000000000 ]
    [  -0.8660254037844409 ]          [  -0.8660254037844386 ]
forward  error: || x - x* ||_oo =  2.33e-15
backward error: || F(x) ||_oo   =  4.00e-15
```

```
# this is Jacobian of F

DF = lambda x : np.array([[ 2. * x[0], 2. * x[1] ],
                          [ 2. * ( x[0] - 1. ), 2. * x[1] ] ])

for x, x_ in zip( [ np.array( [ 0.1, 1.5 ] ), np.array( [ 0., -1.5 ] ) ],
                  [ x1, x2 ] ):

  print( "**", "[", f"{x_[0]:+20.16f}", f"{x_[1]:+20.16f}", "]",\
         "*******", "*******" )

  print( f"{0:2d}", "[", f"{x[0]:+20.16f}", f"{x[1]:+20.16f}",\
       "]", f"{np.linalg.norm( x - x_, np.infty):4.1e}", \
      f"{np.linalg.norm( F(x), np.infty):4.1e}" )

# this is the Newton's method loop, 6 iterations should be enough
  for i in range(1,6):

#   this is the Newton's method update
    x = x - np.linalg.solve( DF(x), F(x) )

    print( f"{i:2d}", "[", f"{x[0]:+20.16f}", f"{x[1]:+20.16f}",\
           "]", f"{np.linalg.norm( x - x_, np.infty):4.1e}",\
          f"{np.linalg.norm( F(x), np.infty):4.1e}" )

  print( "\nx = [", f"{x[0]:+20.16f}", "]"\
        "      x∗ = [", f"{x_[0]:+20.16f}", "]")
  print( "     [", f"{x[1]:+20.16f}", "]"\
         "            [", f"{x_[1]:+20.16f}", "]")
  print( "forward  error: || x − x∗ ||_oo = ",\
        f"{np.linalg.norm(x−x_,np.infty):7.2e}",\
         "\nbackward error: || F(x) ||_oo   = ",\
        f"{np.linalg.norm(F(x),np.infty):7.2e}")
  print("\n")
```

```
** [  +0.5000000000000000  +0.8660254037844386 ] *******  *******
 0 [  +0.1000000000000000  +1.5000000000000000 ] 6.3e−01 2.1e+00
 1 [  +0.5000000000000001  +1.0533333333333335 ] 1.9e−01 3.6e−01
 2 [  +0.5000000000000001  +0.8826793248945148 ] 1.7e−02 2.9e−02
 3 [  +0.5000000000000000  +0.8661825124197711 ] 1.6e−04 2.7e−04
 4 [  +0.5000000000000000  +0.8660254180326618 ] 1.4e−08 2.5e−08
 5 [  +0.5000000000000000  +0.8660254037844387 ] 1.1e−16 0.0e+00

x = [  +0.5000000000000000 ]     x∗ = [  +0.5000000000000000 ]
    [  +0.8660254037844387 ]          [  +0.8660254037844386 ]
forward  error: || x − x∗ ||_oo =  1.11e−16
backward error: || F(x) ||_oo   =  0.00e+00


** [  +0.5000000000000000  −0.8660254037844386 ] *******  *******
 0 [  +0.0000000000000000  −1.5000000000000000 ] 6.3e−01 2.2e+00
 1 [  +0.5000000000000000  −1.0833333333333333 ] 2.2e−01 4.2e−01
 2 [  +0.5000000000000000  −0.8878205128205128 ] 2.2e−02 3.8e−02
 3 [  +0.5000000000000000  −0.8662929278904008 ] 2.7e−04 4.6e−04
 4 [  +0.5000000000000000  −0.8660254450921447 ] 4.1e−08 7.2e−08
 5 [  +0.5000000000000000  −0.8660254037844396 ] 1.0e−15 1.8e−15

x = [  +0.5000000000000000 ]     x∗ = [  +0.5000000000000000 ]
    [  −0.8660254037844396 ]          [  −0.8660254037844386 ]
forward  error: || x − x∗ ||_oo =  9.99e−16
backward error: || F(x) ||_oo   =  1.78e−15
```