Root Finding Methods

1. using numpy, sympy, and SageMath
   - eigenvalues of the companion matrix
   - sympy and SageMath

2. Wrapping Programs
   - OpenXM: Open message eXchange for Mathematics

3. MPSolve
   - an adaptive multiprecision polynomial rootfinder
   - files in Python and `os.system`
   - a Python module to wrap MPSolve

MCS 507 Lecture 6
Mathematical, Statistical and Scientific Software
Jan Verschelde, 1 September 2023
1. using numpy, sympy, and SageMath
   - eigenvalues of the companion matrix
   - sympy and SageMath

2. Wrapping Programs
   - OpenXM: Open message eXchange for Mathematics

3. MPSolve
   - an adaptive multiprecision polynomial rootfinder
   - files in Python and os.system
   - a Python module to wrap MPSolve
Root finding is related to rewriting rules:

\[ x^3 + 2x - 1 = 0 \implies x^3 = -2x + 1 = [0 \ -2 \ +1] \begin{bmatrix} x^2 \\ x \\ 1 \end{bmatrix} \]

and \( x^3 = x \cdot x^2, \ x^2 = x \cdot x, \ x = x \cdot 1 \), or

\[ \begin{bmatrix} x^3 \\ x^2 \\ x \end{bmatrix} = x \cdot \begin{bmatrix} x^2 \\ x \\ 1 \end{bmatrix} \]

So we have:

\[ x \cdot \begin{bmatrix} x^2 \\ x \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & -2 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x^2 \\ x \\ 1 \end{bmatrix} \]
computing eigenvalues

```python
>>> import numpy as np
>>> A = np.matrix([[0,-2,1],[1,0,0],[0,1,0]])
>>> t = np.linalg.eig(A)
>>> t[0]
array([-0.22669883+1.46771151j,
       -0.22669883-1.46771151j,
        0.45339765+0.j])
```

$t[0]$ contains the eigenvectors of $A$

The `roots` commands of `pylab` and `numpy` compute eigenvalues of companion matrices.
Root Finding Methods

1. using numpy, sympy, and SageMath
   - eigenvalues of the companion matrix
   - sympy and SageMath

2. Wrapping Programs
   - OpenXM: Open message eXchange for Mathematics

3. MPSolve
   - an adaptive multiprecision polynomial rootfinder
   - files in Python and os.system
   - a Python module to wrap MPSolve
>>> from sympy import var, roots
>>> x = var('x')
>>> p = x**3 + 2*x - 1
>>> roots(p)
{-2/(3*(1/2 + sqrt(177)/18)**(1/3)) + (1/2 + sqrt(177)/18)**(1/3): 1,
 -2/(3*(-1/2 + sqrt(3)*I/2)*(1/2 + sqrt(177)/18)**(1/3)) + (-1/2 + sqrt(3)*I/2)*(1/2 + sqrt(177)/18)**(1/3): 1,
 (-1/2 - sqrt(3)*I/2)*(1/2 + sqrt(177)/18)**(1/3) - 2/(3*(-1/2 - sqrt(3)*I/2)*(1/2 + sqrt(177)/18)**(1/3)): 1}
sympy session continued

The outcome of \texttt{roots} depends on the coefficient type:

```python
>>> q = x**3 + 2.0*x - 1.0
>>> roots(q)
{-0.226698825758202 - 1.46771150871022*I: 1,
  0.453397651516404: 1,
  -0.226698825758202 + 1.46771150871022*I: 1}
```

The type of what \texttt{roots} returns is a dictionary, its keys are the roots, its values the multiplicities.
solving for $x$

```python
>>> y = var('y')
>>> p = x**2 - y
>>> roots(p, x)
{y**(1/2): 1, -y**(1/2): 1}

>>> p = x**3 - 2*y + 1
>>> roots(p, x)
{(2*y - 1)**(1/3): 1, -(2*y - 1)**(1/3)/2 + 3**(1/2)*(2*y - 1)**(1/3)/2: 1,
 -(2*y - 1)**(1/3)/2 - 3**(1/2)*(2*y - 1)**(1/3)/2: 1}
```
using SageMath

sage: x = polygen(ZZ)
sage: p = (x-1)*(x^3 + 2*x - 1)
sage: p
x^4 - x^3 + 2*x^2 - 3*x + 1
sage: p.roots()
[(1, 1)]
sage: p.roots(ring=RR)
[(0.453397651516404, 1), (1.00000000000000, 1)]
sage: p.roots(ring=RealField(100))
[(0.45339765151640376764474653900, 1),
(1.00000000000000000000000000000, 1)]
Computing all complex roots:

```
sage: p.roots(ring=CC)
[(0.453397651516404, 1),
 (1.00000000000000, 1),
 (-0.226698825758202 - 1.46771150871022*I, 1),
 (-0.226698825758202 + 1.46771150871022*I, 1)]
sage: p.roots(CC,multiplicities=False)
[0.453397651516404, 1.00000000000000,
 -0.226698825758202 - 1.46771150871022*I,
 -0.226698825758202 + 1.46771150871022*I]
```

Instead of `CC` we can take `ComplexField(n)` where `n` is the number of bits.
Root Finding Methods

1. using numpy, sympy, and SageMath
   - eigenvalues of the companion matrix
   - sympy and SageMath

2. Wrapping Programs
   - OpenXM: Open message eXchange for Mathematics

3. MPSolve
   - an adaptive multiprecision polynomial rootfinder
   - files in Python and os.system
   - a Python module to wrap MPSolve
OpenXM: Open message eXchange for Mathematics

How do mathematical programs communicate with each other?

- OpenXM stands for Open message eXchange for Mathematics. It proposes an infrastructure for mathematical communication.
  
  URL: http://www.openxm.org.

- OpenMath is an extensible standard for representing the semantics of mathematical objects.
  
  URL: http://www.openmath.org

- This lecture: wrap executable versions to extend Python.
Root Finding Methods

1. using numpy, sympy, and SageMath
   - eigenvalues of the companion matrix
   - sympy and SageMath

2. Wrapping Programs
   - OpenXM: Open message eXchange for Mathematics

3. MPSolve
   - an adaptive multiprecision polynomial rootfinder
   - files in Python and os.system
   - a Python module to wrap MPSolve

**MPSolve-2.2** is available for download on the web, free *only* for academic use, as of May 2001.

The current **MPSolve-3.2.1**, released on 12 June 2020, is on github, licensed under GPL version 3.

Applies simultaneous iteration, shrinking disks in the complex plane enclosing the roots.

GMP for adaptive use of multiprecision arithmetic.
using MPSolve

For $x^3 + 2x - 1$, the input file is

\[
p(x) = x^3 + 2x - 1
\]

dri
0
3
-1
2
0
1

If saved in input and the executable is mpsolve:

\$
\text{mpsolve input > output}
\$

With > output we redirect the output from screen to a file output.

The file output contains approximations for the roots:

\((-0.226698825758e0, 0.146771150871e1)\)
\((-0.226698825758e0, -0.146771150871e1)\)
\((0.4533976515164e0, 0.e0)\)
calling in Python

Our goal is to use \texttt{mpsolve} in Python:

```python
g>>> from call_mpsolve import mpsolve
g>>> L = mpsolve('x**3 + 2*x - 1')
g>>> for e in L: print e
g...
(-0.22669882575799999, 1.4677115087099999)
(-0.22669882575799999, -1.4677115087099999)
(0.45339765151639999, 0.0)
g```
Root Finding Methods

1. using numpy, sympy, and SageMath
   - eigenvalues of the companion matrix
   - sympy and SageMath

2. Wrapping Programs
   - OpenXM: Open message eXchange for Mathematics

3. MPSolve
   - an adaptive multiprecision polynomial rootfinder
   - files in Python and os.system
   - a Python module to wrap MPSolve
Our `call_mpsolve` is a module encapsulating

```
$ mpsolve input > output
```

The solver proceeds in three steps:

1. Writes a polynomial in a string, e.g. "\(x^3+2x-1\)" to a file in the input format of MPSolve.
2. Executes `mpsolve input > output`.
3. Extracts a list of tuples of the output file.
working with files

We write strings to files:

```python
>>> f = open('/tmp/myfile','w')
>>> f.write('hello there')
>>> f.close()
```

The session continued:

```python
>>> g = open('/tmp/myfile','r')
>>> g.readlines()
['hello there']
>>> g.close()
```
system calls

```python
>>> import os
>>> os.system('ls -lt /tmp/myfile')
-rw-r--r-- 1 jan wheel 11 Sep 20 22:45 /tmp/myfile
0
```

A return value 0 means success.

We can remove files:

```python
>>> os.remove('/tmp/myfile')
>>> os.system('ls -lt /tmp/myfile')
ls: /tmp/myfile: No such file or directory
256
```
Root Finding Methods

1. using numpy, sympy, and SageMath
   - eigenvalues of the companion matrix
   - sympy and SageMath

2. Wrapping Programs
   - OpenXM: Open message eXchange for Mathematics

3. MPSolve
   - an adaptive multiprecision polynomial rootfinder
   - files in Python and `os.system`
   - a Python module to wrap MPSolve
from sympy import var, Poly, degree

def mpsolve_integer_input(strpol):
    #
    # Given in strpol is a string representing
    a polynomial in x with integer coefficients.
    On return is a string which can be
    written as an input file for mpsolve.
    #
    x = var('x')
    pol = eval(strpol)
    qpol = Poly(pol)
    coeffs = qpol.all_coeffs()
    coeffs.reverse()
    result = '! ' + strpol + '
' + 'dri' + '
' + '0
' + str(degree(qpol)) + '
'
    for cff in coeffs:
        result = result + str(cff) + '
'
    return result
def random_number(ndgts):
    """
    Returns a random number of ndgts digits long.
    """
    import random
    return random.randint(10**(ndgts-1), 10**ndgts)

def write_input_file(ipt):
    """
    Generates a random file name and writes the string ipt to file.
    Returns the name of the file.
    """
    name = 'in' + str(random_number(8))
    infile = open(name, 'w')
    infile.write(ipt)
    infile.close()
    return name
```python
def get_mpsolve_location():
    ""
    Searches the command line for the location of the program, otherwise prompts the user.
    """
    import sys
    args = sys.argv
    if len(args) == 1:
        prog = raw_input\('give absolute path name for mpsolve : \')
    else:
        prog = args[1]
    return prog
```
def extract_roots(name):
    """
    Give the name of the output file, 
    returns the roots in a list.
    """
    outfile = open(name, 'r')
    lines = outfile.readlines()
    outfile.close()
    return [eval(e) for e in lines]
def mpsolve(pol, program='~/tmp/mpsolve'):
    """
    Given in the string pol a polynomial in x, returns a list of roots.
    """
    input_data = mpsolve_integer_input(pol)
    input_file = write_input_file(input_data)
    output_file = 'out' + str(random_number(8))
    cmd = program + ' ' + input_file + ' > ' + output_file
    os.system(cmd)
    roots = extract_roots(output_file)
    os.remove(input_file)
    os.remove(output_file)
    return roots
Summary + Exercises

We looked at root finding in the numerical and symbolic sense. We sketched a module to export MPSolve, using system calls.

Exercises:

1. Consider \( p = (x - 1)^{20} \). Make a plot of the roots. Describe what you observe.

2. Use SageMath to compute all roots of \( p = (x - 1)^{20} \) and \( q = (x - 1)^{20} + 10^{-10} \). What do you observe?

3. Compute the roots of \( p = (x - c)^d + \epsilon \) in SageMath for values of \( c \) from 1 to 0, for increasing \( d \) and decreasing \( \epsilon \), making sure the working precision is high enough so \( 1.0 + \epsilon \neq 1.0 \). Relate the location of the roots of \( p \) to values of \( c, d, \) and \( \epsilon \).
Consider two Python interpreters $A$ and $B$:
1. $A$ has no capability for numerical eigenvalue computations;
2. $B$ has access to `numpy` to compute eigenvalues.

Write a script which prints a form for the user to enter the coefficients of a matrix.
After the user submits the form, the script prints all eigenvalues of the matrix into the browser window.
In submitting your answer, provide screen shots to demonstrate that your script works.