homotopy

May 20, 2020

1 a jupyter notebook for homotopy continuation for polynomials in one variable

The aim of this notebook is to provide some suggestions on how to explore the zeros numerically and limitations of numerical computations.

[1]:
```
-- always include this first
restart;
```

```
--loading configuration for package "FourTiTwo" from file
/home/hegland/.Macaulay2/init-FourTiTwo.m2
--loading configuration for package "Topcom" from file
/home/hegland/.Macaulay2/init-Topcom.m2
```

[1]:
```
-- Macaulay2 does include a package but it needs to be loaded
loadPackage "NumericalAlgebraicGeometry";
```

```
--loading configuration for package "NumericalAlgebraicGeometry" from file
/home/hegland/.Macaulay2/init-NumericalAlgebraicGeometry.m2
--loading configuration for package "PHCpack" from file
/home/hegland/.Macaulay2/init-PHCpack.m2
--loading configuration for package "Bertini" from file
/home/hegland/.Macaulay2/init-Bertini.m2
```

[2]:
```
-- univariate complex polynomials
R = CC[z];
```

[9]:
```
-- example 1: solving z^4-1=0

F1 = {z^4 - 1} -- the equation to solve
zs1 = solveSystem F1
xsol = sort(apply(zs1, zi -> (coordinates zi)#0))
<< "list of all solutions:   " << xsol << endl
zr1 = realPoints zs1
xreal = sort(apply(zr1, zi -> (coordinates zi)#0))
<< "list of all real solutions:   " << xreal << endl;
```

```
list of all solutions:   {-1, -ii, ii, 1}
list of all real solutions:   {-1, 1}
```

- the solution is a list of points (indexed from 0 … 3 here)
- each point is a hash table
- if the point is regular only the coordinate list is printed
- we can have a look at the whole hash table with peek

[8]:
```
peek zs1#0
```

```
o8 = Point{Coordinates => {-1}        }
           H => GateHomotopy{…12…}
           LastIncrement => .15
           LastT => 1
           NumberOfSteps => 9
           SolutionStatus => Regular
```

[69]:
```
-- example 2: simple polynomial
F2 = {(z-5)^3*(z+2)^2}

zs2 = solveSystem F2
<< "solutions: " << zs2 << endl;
```

```
solutions: {1 : (5), 1 : (5), 1 : (-2), 1 : (5), 1 : (-2)}
```

[66]:
```
-- example 2: print the coordinates and status of sol. pts
xsol = apply(zs2, zi -> (coordinates zi)#0)
<< "solutions: " << xsol << endl;
<< apply(zs2, zi-> status zi) << endl;
```

```
solutions: {5, 5, -2, 5, -2}

{Singular, Singular, Singular, Singular, Singular}
```

[15]:
```
-- use top-level Macaulay2 homotopy continuation routines
--  (instead of built-in ones, could also use Bertini)
zs = solveSystem F2, Software=>"M2"
```

```
o15 = ({{-1}, {1}, 1 : (1), 1 : (-1), 1 : (1), 1 : (-1)}, Software => M2)

o15 : Sequence
```

[235]:
```
-- example 3: z*(z-1)*...*(z-n) = 0
n = 5
F3 = {product(n+1, i -> z-i)}
zs = solveSystem(F3)
<< apply(zs, zi -> status zi) << endl;
```

```
<< apply(zs, zi -> (coordinates zi)#0) << endl;
```

```
{Regular, Origin, Regular, Regular, Regular, Regular}

{3, -7.4195e-19-5.52702e-19*ii, 5, 1, 2, 4}
```

- here we get a rounding error for solution z=0
- this example is well-known in numerical analysis
- the problem is that in floating point arithmetic the coefficients of the polynomial do not represent the zeros well, see the case for n=10
- as a consequence, the equation det(A-lambda I)=0 does often not recover the eigenvalues of A well. Also the companion matrix (which just contains the coefficients) of a polynomial while having the same eigenvalues in principle does not allow the computation of these eigenvalues accurately in many cases.

  Thus one uses algorithms based on the original matrix A.

[97]:
```
-- example 3 with n=10 (try also n=20 and n=30)
n = 10
F3 = {product(n, i -> z-i)}
zs = solveSystem(F3)
<< apply(zs, zi -> status zi) << endl;
xsol = apply(zs, zi -> (coordinates zi)#0)
<< "solutions " << xsol << endl;
<< "number of sols "<< (length(xsol)) << endl;
<< "degree of polynomial "<< (degree(F3#0))#0 << endl;
<< "coeffs " << apply(listForm(F3#0), yi->yi#1) << endl;
<< "exponents " << apply(listForm(F3#0), yi->(yi#0)#0) << endl;
```

```
{Regular, Regular, Regular, Regular, Regular, Regular, Regular, Regular, Origin}

solutions {2, 5, 9, 8, 3, 1, 6, 4, 5.94139e-12+2.11731e-12*ii}

number of sols 9

degree of polynomial 10

coeffs {1, -45, 870, -9450, 63273, -269325, 723680, -1172700, 1026580, -362880}

exponents {10, 9, 8, 7, 6, 5, 4, 3, 2, 1}
```

- for sufficiently difficult (!) polynomials, not all solutions are found

[9]:
```
-- but for this polynomial it seems to work ...
-- (this is the polynomial used for the startsystem)

length(solveSystem({z^(11)-1}))  -- just printing nu. of solutions
```

```
o9 = 11
```

[119]:
```
-- example 4 (a nicer real polynomial with singular (double) points)
n = 21
x  = sort(apply(n, j -> cos(2*pi*j/n))) -- real parts of unit roots
F4 = {product(n, i -> z-x#i)}
zs = solveSystem(F4)
xsol = sort(apply(zs, zi -> (coordinates zi)#0))
<< apply(zs, zi -> status zi) << endl;
<< "error " << norm(xsol-x)/norm(xsol) << endl;
<< "residual "<<
    norm(F4#0-product(xsol,xi -> z-xi))/norm(F4#0)<<endl;
<< "degree of polynomial "<< (degree(F4#0))#0 << endl;
<< "number of sols "<< (length(zs)) << endl;
<< "coeffs " << apply(listForm(F4#0), yi->yi#1) << endl;
```

```
{Regular, Singular, Singular, Singular, Singular, Singular, Singular, Singular,
Singular, Singular, Singular, Singular, Singular, Singular, Singular, Singular,
Singular, Singular, Singular, Singular, Singular}

error .0439288

residual .02255

degree of polynomial 21

number of sols 21

coeffs {1, 4.44089e-16, -5.25, -2.66454e-15, 11.8125, -4.08562e-14, -14.875,
-3.01092e-13, 11.4844, -2.62013e-13, -5.59863, -3.73035e-14, 1.71069,
3.33067e-16, -.314209, 1.76942e-16, .031723, -6.72205e-18, -.00146866,
3.38813e-19, .0000200272, -9.53674e-7}
```

- one gets relatively big rounding errors, in particular large imaginary components (which should be zero)
  - errors get larger around degree 20
  - errors have nonzero imaginary parts around degree 30

[48]:
```
-- check the last solution and compare with the real part of
-- the corresponding start solution -- do this for different n
<< peek zs#(n-1) << endl;
<< "start sol " << x#(n-1) << endl;
```

```
Point{Coordinates => {-.809017} }
      LastT => 1
      Multiplicity => 2
      SolutionStatus => Singular
```

```
start sol 1
```

[287]:
```
-- example 5 (random solutions)
n = 6    -- works until around n=7 ...
x = sort(apply(n, j -> random(1.0)))
F5 = {product(x, xi -> z-xi)}
zs = solveSystem(F5)
xsol = sort(apply(zs, zi->(coordinates zi)#0))
<< apply(zs, zi -> status zi) << endl;
<< "error " << norm(xsol - x)/norm(x) << endl;
<< "residual "<<
    norm(F5#0-product(xsol,xi -> z-xi))/norm(F5#0)<<endl;
<< "degree of polynomial "<< (degree(F5#0))#0 << endl;
<< "number of sols "<< (length(zs)) << endl;
<< "coeffs " << apply(listForm(F5#0), yi->yi#1) << endl;
```

```
{Regular, Regular, Regular, Regular, Regular, Regular}

error 7.07863e-13

residual 8.92216e-13

degree of polynomial 6

number of sols 6

coeffs {1, -3.58808, 5.18941, -3.86308, 1.55599, -.32045, .0263452}
```

1.0.1 polynomials with random coefficients

- solving polynomials with random coefficients is relatively unproblematic
- all the coefficients are of the same size

[296]:
```
-- example 6 (random coefficients)
n = 40
c = apply(n+1, j -> random(1.0))
F6 = {sum(n+1, i -> c#i * z^i)}
zs = solveSystem(F6)
xsol = sort(apply(zs, zi->(coordinates zi)#0))
-- residual is difference between original and recovered polynomial
<< "residual "<< norm(F6#0 - c#n * product(xsol, xi->z-xi))/norm(F6#0) << endl;
<< "degree of polynomial "<< (degree(F6#0))#0 << endl;
<< "number of sols "<< (length(zs)) << endl;
<< "coeffs " << apply(listForm(F6#0), yi->yi#1) << endl;
```

```
residual 3.91112e-8
```

```
degree of polynomial 40

number of sols 40

coeffs {.423109, .451412, .689069, .115223, .996308, .470021, .922593, .643133,
.796686, .981886, .0467291, .452046, .678419, .88067, .323219, .450221, .511634,
.936532, .116715, .0101581, .279284, .857634, .880767, .763691, .58265, .379274,
.980667, .727799, .766732, .354455, .581848, .727099, .893648, .695019, .495008,
.398728, .735938, .847004, .984405, .540424, .595444}
```