Computing with rational functions

Tags: RKFUN, rational functions
Name: Mario Berljafa and Stefan Güttel
File: example_rkfun.m
Date: 27/05/2015

Contents

1 Introduction **1**

2 Evaluating an rkfun **2**

3 Plotting **2**

4 Pole- and root-finding **4**

5 Basic arithmetic operations and differentiation **4**

6 Multiple precision computations **6**

7 Conversion to partial fraction form **7**

8 Conversion to quotient and continued fraction form **8**

9 References **9**

1 Introduction

This toolbox comes with an implementation of a class called `rkfun`, which is the fundamental data type to represent and work with rational functions. Objects of this class are produced by the `rkfit` function (described in [2,3]), which in its simplest use case attemts to find a rational function r such that

$$\|F\boldsymbol{b} - r(A)\boldsymbol{b}\|_2 \to \min,$$

where A, F are square matrices and $\boldsymbol{b}$ is a vector of compatible sizes. For example, let us consider $A = \text{tridiag}(-1, 2, -1)$, $\boldsymbol{b} = [1, 0, \ldots, 0]^T$, and $F = (A-2I)^2(A^2-I)^{-1}(A-4I)^{-1}$. We know that $r(z) = (z-2)^2(z^2-1)^{-1}(z-4)^{-1}$ is a minimizer for the above problem. Let us try to find it via RKFIT:

```
N = 100;
A = gallery('tridiag',N);
I = speye(N);
```

```
F = (A - 2*I)^2*inv(A^2 - I)*inv(A - 4*I);
b = eye(N,1);
xi = inf*ones(1,5); % initial poles at infinity
maxit = 5; tol = 1e-10;
[xi,ratfun,misfit] = rkfit(F,A,b,xi,maxit,tol,'real');
```

As we started with $m = 5$ initial poles (at infinity), RKFIT will search for a rational function r of type $(5, 5)$. When more than 1 iteration is performed and the tolerance `tol` is not chosen too small, RKFIT will try to reduce the type of the rational function while still maintaining a relative misfit below the tolerance, i.e., $\|F\boldsymbol{b} - r(A)\boldsymbol{b}\|_2 \leq \text{tol}\|F\boldsymbol{b}\|_2$. Indeed, a reduction has taken place and the type has been reduced to $(2, 3)$, as can be seen from the following output:

```
ratfun

ratfun =
    RKFUN object of type (2,3).
    Real Hessenberg pencil (H,K) of size 4-by-3.
    coeffs = [ -0.0869, -0.394, 1.33, 1.5 ]
```

We can now perform various operations on the `ratfun` object, all implemented as methods of the class `rkfun`. To see a list of all methods just type `help rkfun`. We will now discuss some methods in more detail.

2 Evaluating an `rkfun`

We can easily evaluate $r(z)$ at any point (or many points) in the complex plane. The following command will evaluate $r(2)$ and $r(3 + i)$ simultaneously:

```
format long
ratfun([2 ; 3+1i])

ans =
 -0.000000000000652 + 0.000000000000000i
  0.011764705881620 - 0.152941176470453i
```

We can also evaluate r as a matrix function, i.e., computing $r(M)B$ for matrices M and B, by using two input arguments. For example, by setting $B = I$ we effectively compute the full matrix function $r(M)$:

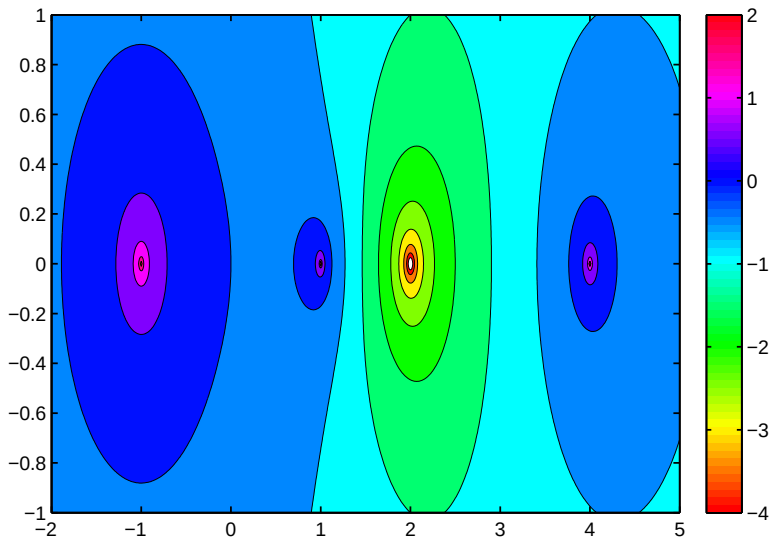
```
M = [ 3 ,  1 ; 0 , 3 ];
B = eye(2);
R = ratfun(M,B)

R =
  -0.125000000000608   -0.281249999999790
                   0   -0.125000000000608
```

3 Plotting

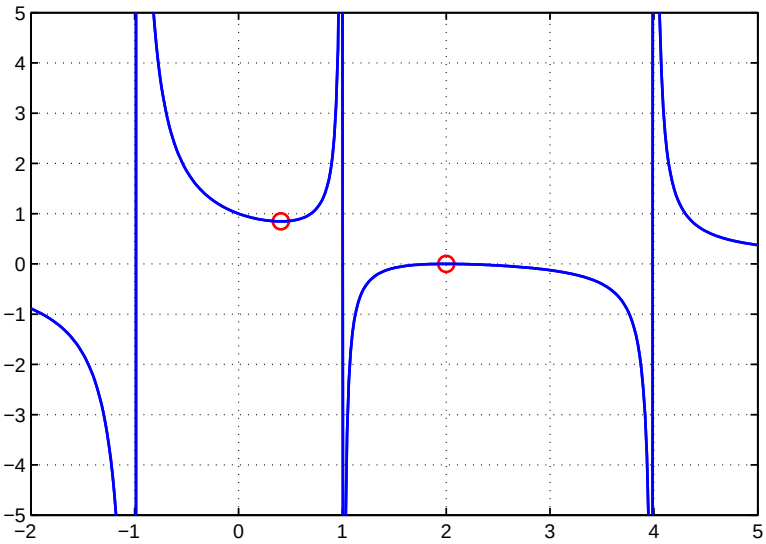
As r can be evaluated at any point in the complex plane, it is straightforward to produce plots of this function. For example, here is a contour plot of $\log_{10}|r|$ over the complex region $[-2,5]\times[-1,1]i$:

```
figure(1)
[X,Y] = meshgrid(-2:.01:5,-1:.01:1); Z = X + 1i*Y;
R = ratfun(Z);
contourf(X,Y,log10(abs(R)),-4:.5:2)
colormap hsv, colorbar
```



Another command `ezplot` can be used to get a quick idea of how `ratfun` looks over an interval on the real axis, in this case, $[-2,5]$:

```
figure(2)
ezplot(ratfun,[-2,5])
ylim([-5,5])
grid on
```



4 Pole- and root-finding

From the above plot we guess that r has poles at $x = \pm 1$ and $x = 4$ and a root at $x = 2$, which is to be expected from the definition of r. The two commands `poles` and `roots` do exactly what their names suggest:

```
pls = poles(ratfun)
rts = roots(ratfun)

pls =
  -0.999999999987786
   1.000000000000003
   3.999999999999999
rts =
  1.999999999997977 + 0.000001978069056i
  1.999999999997977 - 0.000001978069056i
```

As expected from a type (2,3) rational function, there are two roots and three poles. Note that the pole at $x = -1$ is identified with slightly less accuracy than the poles at $x = 1$ and $x = 4$. This is because the point $x = -1$ is outside the spectral interval of A and hence sampled less accurately. Also the double root at $x = 2$ is identified up to an accuracy of $\approx 10^{-7}$ only. This is not surprising as the function is flat nearby multiple roots. However, the backward error of the roots is small:

```
ratfun(rts)
```

```
ans =
   1.0e-16 *
 -0.277555756156289 - 0.000000926442286i
 -0.277555756156289 + 0.000000926442286i
```

5 Basic arithmetic operations and differentiation

We have implemented some very basic operations on `rkfun`'s, namely, the multiplication by a scalar and the addition of a scalar. The result of such operations is again an `rkfun` object. For example, the following command computes points z where $2r(z) = \pi$:

```
z = roots(2*ratfun - pi)
ratfun(z)

z =
  -0.405042315578132
   0.857694376021649
ans =
   1.570796326794891
   1.570796326794892
```

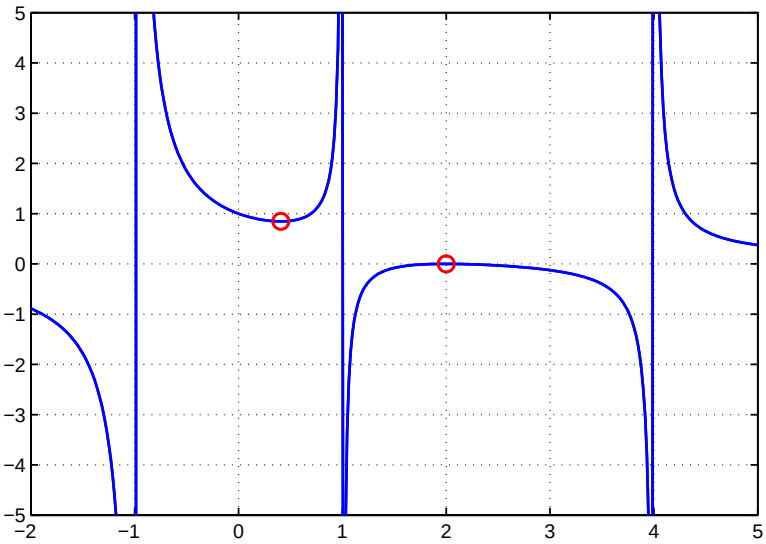
We have currently not implemented the summation and multiplication of two `rkfun`'s, though this is doable in principle. However, we can already differentiate a rational function using the `diff` command. The following will find all local extrema of r by computing the roots of r':

```
extrema = roots(diff(ratfun))

extrema =
  0.407592767463221 + 0.000000000000000i
  2.000000097271605 + 0.000000000000000i
  2.796203354877036 + 2.627131418477983i
  2.796203354877036 - 2.627131418477983i
```

There are two real extrema which we can add to the above plot of r:

```
figure(2), hold on
plot(extrema(1:2),ratfun(extrema(1:2)),'ro')
```



The syntax and "feel" of these computations is inspired by the Chebfun system [4], which represents polynomials via Chebyshev interpolants and allows for many more operations to be performed than our `rkfun` implementation. Here we are representing rational functions through their coefficients in a discrete-orthogonal rational function basis. Working with rational functions poses some challenges not encountered with polynomials. For example, the indefinite integral of a rational function is not necessarily a rational function but may contain logarithmic terms.

6 Multiple precision computations

Objects of class `rkfun` can be converted to MATLAB's Variable Precision Arithmetic (VPA) as follows:

```
vpa(ratfun)

ans =
    RKFUN object of type (2,3).
    Real Hessenberg pencil (H,K) of size 4-by-3.
    Variable precision arithmetic (VPA) activated.
    coeffs = [ -0.0869, -0.394, 1.33, 1.5 ]
```

Alternatively, we can also use the Advanpix Multiple Precision (MP) toolbox [1], which is typically more efficient and reliable than VPA. We recommend the use of this toolbox in particular for high-precision root-finding of `rkfun`'s and conversion to partial fraction form:

```
ratfun = mp(ratfun)
```

```
ratfun =
    RKFUN object of type (2,3).
    Real Hessenberg pencil (H,K) of size 4-by-3.
    Multiple precision arithmetic (ADVANPIX) activated.
    coeffs = [ -0.0869, -0.394, 1.33, 1.5 ]
```

When evaluating a multiple precision ratfun, the result will be returned in multiple precision:

```
ratfun([2 ; 3+1i])


ans =


Columns 1 through 1

        -6.5214076339632695317110660289725 82e-13 +
0i
           0.011764705881619708454245991986474 89 -
0.15294117647045252515457828101731 16i
```

It is important to understand that, although the evaluation of `ratfun` is now done in multiple precision, the computation of `ratfun` using the `rkfit` command has been performed in standard double precision. `rkfit` does not currently support the computation of `rkfun`'s in multiple precision. Here are the roots of `ratfun` computed in multiple precision:

```
roots(mp(ratfun))


ans =


Columns 1 through 1

        1.9999999999979769039352472439737 27 +
1.9780911456204383034773656143661 26e-06i
        1.9999999999979769039352472439737 27 -
1.9780911456204383034773656143661 26e-06i
```

7 Conversion to partial fraction form

It is often convenient to convert a rational function r into its partial fraction form

$$r(z) = \alpha_0 + \frac{\alpha_1}{z - \xi_1} + \cdots + \frac{\alpha_m}{z - \xi_m}.$$

The `residue` command of our toolbox performs such a conversion. Currently, this only works when the poles ξ_j of r are distinct and r is not of superdiagonal type (i.e., there is no linear term in r). As the conversion to partial fraction form can be an ill-conditioned transformation, we recommend to use `residue` in conjunction with the multiple precision feature. Here are the poles ξ_j and residues α_j ($j = 1, \ldots, m$), as well as the absolute term α_0, of the function r defined above:

```
[alpha,xi,alpha0,cnd] = residue(mp(ratfun));
double([xi , alpha])
double(alpha0)

ans =
  -0.999999999987786   0.899999999994009
   1.000000000000003  -0.166666666666637
   3.999999999999999   0.266666666666479
ans =
     7.839257895180645e-17
```

The absolute term is close to zero because r is of subdiagonal type. The output `cnd` of `residue` corresponds to the condition number of the transformation to partial fraction form. In this case of a low-order rational function with well separated poles the condition number is actually quite moderate:

```
cnd

cnd =
  56.150570383900494
```

8 Conversion to quotient and continued fraction form

Our toolbox also implements the conversion of a `rkfun` to quotient form $r = p/q$ with two polynomials p and q given in the monomial basis. As with the conversion to partial fraction form, we recommend performing this transformation in multiple precision arithmetic due to potential ill-conditioning. Here we convert r into the p/q form and evaluate it at $x = 2$ using MATLAB's `polyval` command:

```
[p,q] = poly(mp(ratfun));
polyval(p,2)./polyval(q,2)
```

```
ans =

    -6.5214076339632688132193445920766 03e-13 +
1.7045389458197495405689479 0
3587348e-52i
```

A `rkfun` can also be converted into continued fraction form

```
              1
 r(z) = ------------------------------------------------------ + absterm
        hh(1)*z + 1
                  ---------------------------------------------
                  h(1) + 1
                         -------------------------------------
                         hh(2)*z +...+ 1
                                       -------------------------
                                       h(m-1) + 1
                                                ----------------
                                                hh(m)*z + 1/h(m)
```

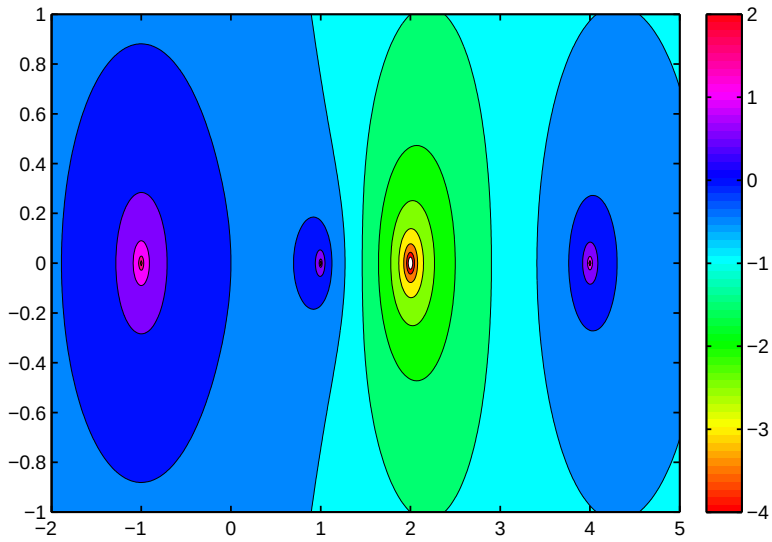
as follows:

```
[h,hh,absterm,cnd] = contfrac(mp(ratfun));
```

That's it for this tutorial. Note that more methods will be added over time and we'd be happy to receive any feedback or bug reports. For more details about the internal representation of `rkfun`, see [3].

The following command merely creates a nice thumbnail.

```
figure(1), hold on, plot(NaN) % create thumbnail
```



9 References

[1] Advanpix LLC., *Multiprecision Computing Toolbox for MATLAB,* ver 3.8.3.8882, Tokyo, Japan, 2015. http://www.advanpix.com/.

[2] M. Berljafa and S. Güttel. *Generalized rational Krylov decompositions with an application to rational approximation,* SIAM. J. Matrix Anal. Appl., 36(2):894–916, 2015.

[3] M. Berljafa and S. Güttel. *The RKFIT algorithm for nonlinear rational approximation,* MIMS EPrint 2015.38 (http://eprints.ma.man.ac.uk/2309/), Manchester Institute for Mathematical Sciences, The University of Manchester, UK, 2015.

[4] T. A. Driscoll, N. Hale, and L. N. Trefethen. *Chebfun Guide,* Pafnuty Publications, Oxford, 2014. http://www.chebfun.org