Bengt
07/11/24

The Taylor series method for evaluating the ${}_2F_1$ function, and a closed form recursion relation for its coefficients

Background:

I described in my 05/27/24 handout the Taylor series method applied to the task of numerically evaluating ${}_2F_1(a,b;c;z)$, with the numerical demonstration focused on the case of $a=\frac{1}{2}, b=\frac{1}{3}, c=\frac{1}{4}$, i.e., ${}_2F_1\left(\frac{1}{2},\frac{1}{3};\frac{1}{4};z\right)$. The governing ODE for $y(z)={}_2F_1(a,b;c;z)$ was there written as

$$(1-z)\,z\;y''(z)+(c-(a+b+1)\,z)\,y'(z)-a\,b\,y(z)=0\,. \tag{1}$$

We are interested in the solution that satisfies $y(0)=1$. In view of my 06/23/24 handout, it suffices for all *z* to do this calculation in the complex plane region $|z|\le 1 \cap \mathrm{Re}(z)\le \frac{1}{2}$. This suggests the strategy that is illustrated in Figure 1 below:

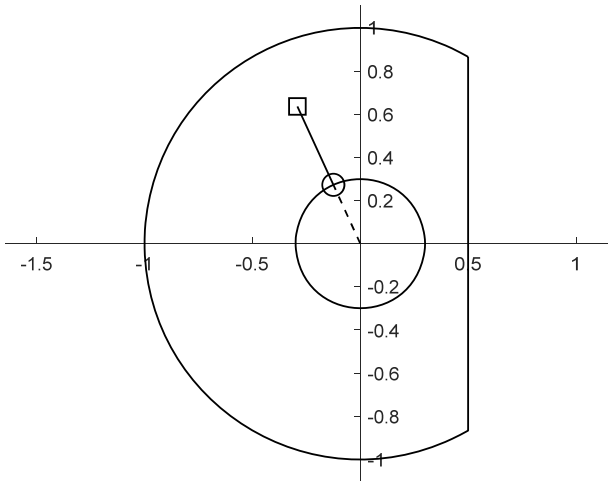

Figure 1: The outer boundary encloses the complex plane region $|z|\le 1 \cap \mathrm{Re}(z)\le \frac{1}{2}$. The inner circle has here radius 0.3.

For an evaluation point inside the central circle, a direct evaluation using the Taylor expansion of ${}_2F_1(a,b;c;z)$ converges rapidly. Else, at a point as suggested by the small square in Figure 1, we evaluate with the Tylor expansion at the point marked by a small circle (at the same angular direction from the origin) and continue out to the evaluation point with high order accurate Taylor ODE stepping.

New observation:

Denoting our Taylor method step size by *k*, inserting

$$y(z_0+k) = \underset{\updownarrow}{c_1} + \underset{\updownarrow}{c_2 k} + c_3 k^2 + c_2 k^3 + c_3 k^4 + \ldots \qquad (3)$$
$$y(z_0)\quad y'(z_0)$$

into

$$(1-(z_0+k))((z_0+k)\frac{d^2y(z_0+k)}{d^2k} + (c-(a+b+1)(z_0+k))\frac{dy(z_0+k)}{dk} - a\,b\,y(z_0+k) = 0 \qquad (4)$$

and equating coefficients for *k* gives 3-term recursion relations between the expansion coefficients in (3). Given values for $y(z_0)$ and $y'(z_0)$, i.e., for c_1 and c_2, we can by these explicitly calculate any number of further Taylor coefficients $c_3, c_4, c_5, \ldots$. Form these, we can then evaluate $y(z_0+k)$ and $y'(z_0+k)$, i.e. obtain the information we need for the next Taylor step (two start values, since the ODE is of second order).

Since the coefficient relations are 3-term formulas, the cost for each coefficient becomes $O(1)$ operations and for accuracy order *p* a total of $O(p)$ operations (rather than $O(p^2)$ as is typical for the Taylor series method). This becomes the case thanks to (1) being a linear ODE, with coefficients that are very low degree polynomials in *z*.

Numerical verification:

Rather than stepping in *z*, we verify simpler the MATLAB code below by considering just the one case of expanding ${}_2F_1\left(\frac{1}{2},\frac{1}{3};\frac{1}{4};z\right)$ around $z = 0.2+0.3i$. Mathematica gives the first 9 Taylor terms as

```
In[] := N[Series[Hypergeometric2F1[1/2, 1/3, 1/4, z], {z, 0.2 + 0.3 i, 9}]]
Out[] = (1.08191 + 0.260872 i) + (0.720801 + 0.465418 i) (z - (0.2 + 0.3 i)) +
  (0.479805 + 0.642976 i) (z - (0.2 + 0.3 i))^2 + (0.226639 + 0.778717 i) (z - (0.2 + 0.3 i))^3 -
  (0.063878 - 0.849232 i) (z - (0.2 + 0.3 i))^4 - (0.384185 - 0.829895 i) (z - (0.2 + 0.3 i))^5 -
  (0.709433 - 0.699947 i) (z - (0.2 + 0.3 i))^6 - (1.00206 - 0.447542 i) (z - (0.2 + 0.3 i))^7 -
  (1.21567 - 0.0747066 i) (z - (0.2 + 0.3 i))^8 - (1.30019 + 0.398368 i) (z - (0.2 + 0.3 i))^9 + O[z - (0.2 + 0.3 i)]^10
```

Picking up from here the values for the first two coefficients, the MATLAB code shown below

```
clear; close all;
a = 1/2; b = 1/3; c = 1/4;          % Set 2F1 parameters for numerical test
z = 0.2+0.3i;
p = 10;                             % Get expansion up through O(k^(p-1))
cf = zeros(1,p);                    % Vector for the coefficients
cf(1) = 1.08191  + 0.260872i;       % Initialize with the first two
cf(2) = 0.720801 + 0.465418i;       % coefficients

d1 = -a*b;  d2 = 1-a-b;     d3 = c-(1+a+b)*z; d4 = d3;
d5 = 2-4*z; d6 = 2*z*(z-1); d7 = d6;          d8 = d6;

for n = 3:p
    cf(n) = (d1*cf(n-2)+d3*cf(n-1))/d6;
    d2 = d2-2; d1 = d1+d2; d4 = d4+d5; d3 = d3+d4; d7 = d7+d8; d6 = d6+d7;
end
```

produces in the vector `cf` the result

```
cf =
  Columns 1 through 2
   1.0819e+00 + 2.6087e-01i   7.2080e-01 + 4.6542e-01i
  Columns 3 through 4
   4.7981e-01 + 6.4298e-01i   2.2664e-01 + 7.7872e-01i
  Columns 5 through 6
  -6.3878e-02 + 8.4923e-01i  -3.8418e-01 + 8.2989e-01i
  Columns 7 through 8
  -7.0943e-01 + 6.9995e-01i  -1.0021e+00 + 4.4754e-01i
  Columns 9 through 10
  -1.2157e+00 + 7.4699e-02i  -1.3002e+00 - 3.9833e-01i
```

Given that the input values for the first two coefficients were provided by Mathematica to only 6 or 7 significant digits, the agreement seems fully satisfactory.

The derivation of the MATLAB code is slightly tricky, so we omit the details here. It is computationally very fast, as each successive Taylor coefficient is obtained by just 10 arithmetic operations.