Bengt
05/27/24

A test on evaluating hypergeometric functions by directly solving numerically their governing ODEs

This handout is a follow-up to expressing my concern (in an e-mail exchange with Cécile) that approximating hypergeometric functions by simply solving numerically their governing ODEs might work very well and thus be a serious competitor to the fractional derivative approach. We ought to explore this possibility before publishing anything about the latter (since we likely would end up being asked about this). Below is a first numerical test related to this.

Present test problem:

Since each hypergeometric function satisfies a linear ODE and the numerical approach below immediately generalizes to any such case, we consider here (without much loss of generality) the same ${}_2F_1$ case as I looked at in some of my previous handouts: ${}_2F_1(a,b;c;z)$ in 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 $y(z)={}_2F_1(a,b;c;z)$ is

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

This ODE has two singular points, at $z=0$ and $z=1$ (as well as at $z=\infty$). Being of second order, it has two linearly independent solutions. The hypergeometric function ${}_2F_1(a,b;c;z)$ is the solution that is analytic at the singular point $z=0$ and there takes the value 1. Since initiating a numerical solution at an ODE's singularity point seems unnecessarily risky, I have in the numerical test below instead started the ODE solver at $z=1/2$, at which we easily (for example by the known Taylor expansion at the origin) obtain

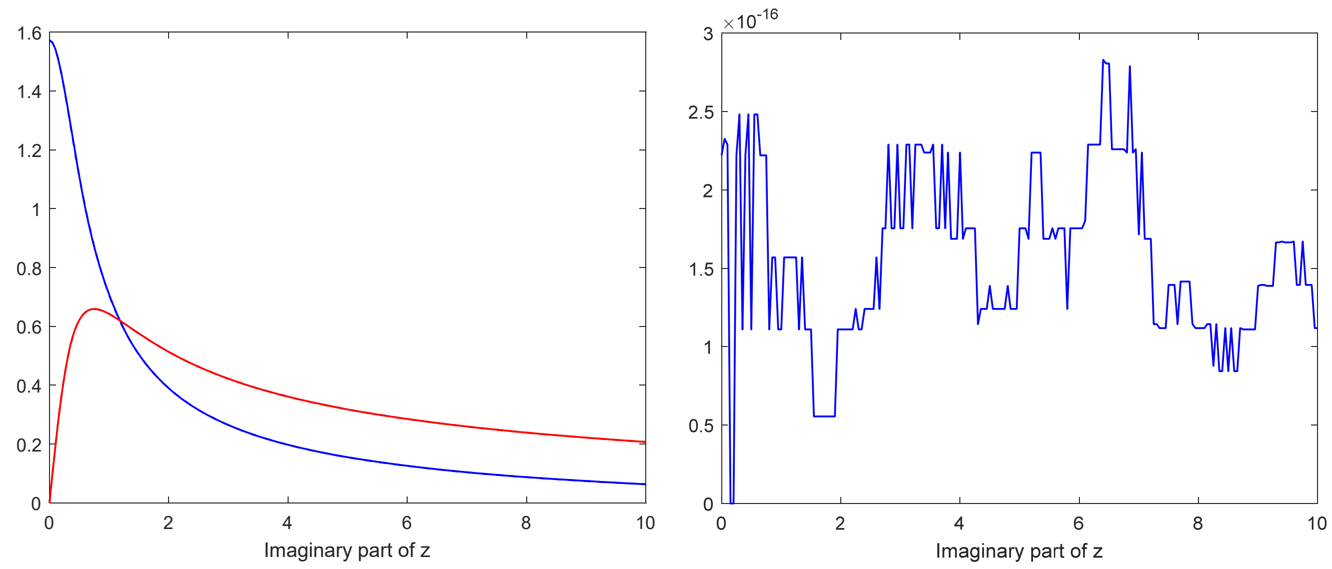
$$y(1/2)=1.572478431493706,\quad y'(1/2)=2.015953977463255\,. \tag{2}$$

Numerical implementation:

The code in the Appendix implements the Taylor series method for ODEs, as this is described in Section 3.1.3 of my book manuscript [F24]. We note from the description that:

i. The accuracy that is achieved improves very rapidly with the order of the expansions (cf., the book's Figure 3.2), and
ii. No symbolic algebra is needed. The necessary recursions for series multiplications and divisions can be expressed as single-line MATLAB anonymous functions.

From the integration start point $z=1/2$, we can take steps along any (jagged, variable step length) path in the complex plane (since the Taylor method is a one-step method). In the present test, I have simply stepped straight upwards from $z=1/2$ to $z=1/2+10i$, using 200 steps of size $k=0.05\,i$, with the accuracy order set to $p=20$. The code produces two figures. Figure 1 (a) shows along the vertical line from $z=1/2$ to $z=1/2+10i$ the real and imaginary parts as computed (solid blue and red curves, respectively) and superposed on these (blue and red dashed curves) the analytic result according to MATLAB's hypergeom function. The sets of curves visually completely overlap; hence no dashed curves are visible. Figure 1 (b) shows the magnitude of the difference between using the Taylor method and hpergeom evaluations. This difference is entirely on the machine rounding level.



(a) Numerical values for real and imaginary parts (b) Magnitude of the difference between the calculations

Figure 1. Numerical results, as described in the text.

Reference:

[F24] Fornberg, B., High Accuracy Finite Difference Methods, to appear, Cambridge University Press (2024).

Appendix: Thee following code produces the two subplots of Figure 1.

```
clear; close all;
%   Run the 2F1 test example with the Taylor series method
nt =  201;          % Use nt-1 continuation steps
p  =   20;          % Accuracy order - nr of terms in each Taylor expansion
k  = 0.05i;         % Value of each step; here straight up
yv = zeros(1,nt);   % Vectors to store results for y and v initially and
vv = zeros(1,nt);   % after each step
a = 1/2; b = 1/3; c = 1/4;  % Set the 2F1 parameters
yv(1) = 1.572478431493706;  % Set initial conditions at z = 1/2
vv(1) = 2.015953977463255;  % for y and v = y'

% Functions for series multiplication and division. Given entries 1,2,...,n
% in a and b and for division also entries 1,2,...,n-1 in c (row vectors),
% create the nth entry of the product a*b or quotient a/b, respectively
mult = @(a,b,n)   a(1:n)*b(n:-1:1).';                  % c = a*b
div  = @(a,b,c,n) (a(n)-b(2:n)*c(n-1:-1:1).')/b(1);  % c = a/b

y  = zeros(1,p+1);  % Vectors to hold the Taylor expansions for y and v
v  = zeros(1,p+1);  % during each time step
z1 = zeros(1,p);    % Vector to hold the 2-term expansion of z
z2 = zeros(1,p);    % Vector to hold the 3-term expansion of (1-z)z
t1 = zeros(1,p);    % Vector to hold an intermediate expansion
t2 = zeros(1,p);    % Vector to hold an intermediate expansion
```

```
y(1) = yv(1);       % Set start values for y and v for the first iteration
v(1) = vv(1);

for m = 2:nt             % Loop over the nt-1 steps
    zc = 0.5+(m-2)*k;    % Center for current series expansion
    z1(1:2) = [zc,1];                  % z      /. z-> zc+k
    z2(1:3) = [zc*(1-zc),1-2*zc,-1];     % (1-z)z /. z-> zc+k

    % Carry out one step
    for n = 1:p     % Iterate the Taylor expansions to include p+1 terms
        t1(n) = a*b*y(n) - c*v(n) + (a+b+1)*mult(z1,v,n);
        t2(n) = div(t1,z2,t2,n);
        v(n+1) = t2(n)/n;             % Integrate term to gain one more for v
        y(n+1) = v (n)/n;             % Integrate to gain one more term for y
    end
                % Evaluate the Taylor expansions for y and v at location k
    yv(m) = polyval(y(p+1:-1:1),k);    % Store results for y and v from
    vv(m) = polyval(v(p+1:-1:1),k);    % the last step
    y(1)  = yv(m);                     % Use these values to initiate
    v(1)  = vv(m);                     % the next step
    % Step finished
end             % All steps finished

lw = 1;         % LineWidth in plots
plot(abs(k)*(0:nt-1),real(yv),'b-','LineWidth',lw); hold on
plot(abs(k)*(0:nt-1),imag(yv),'r-','LineWidth',lw);
xlabel('Imaginary part of z')
F = hypergeom([a,b],c,0.5+k*(0:nt-1));
plot(abs(k)*(0:nt-1),real(F),'b--','LineWidth',lw);
plot(abs(k)*(0:nt-1),imag(F),'r--','LineWidth',lw);
figure
plot(abs(k)*(0:nt-1),abs(F-yv),'b-','LineWidth',lw);
xlabel('Imaginary part of z')
```