```
%
%  CA4_nm.m -- dhpham -- 12 feb 2019
%
%  MAIN USAGE: Create a 16x16 matrix, each row k containing 16
 positive roots
%  of the m-th Bessel function, where m = 0,1,... ,15, k = 1,2,...,16.
%  Find these roots using Newton's Method.
%
%  INPUT: Use an existing 'tolnm' as the convergence tolerance if it
 exists,
%  otherwise use tol = 1e-6.
%
%  OUTPUT:
%           'Amk':  the 16x16 matrix containing roots to the m Bessel
 funcs.;
%   'mean_Nevals':  the mean # of func. evals per iteration
%    'bsl_reserr':  the residual error of J_m(x) evaluated at each
 root z_{m,k} in Amk.
%

disp('==== USING NM ====')
if (exist('tolnm','var') ~= 1)
 % Default convergence tolerance
    tol=1e-6
else
 %  check if existant tol is scalar, set to default if not
    if (sum(size(tolnm)==[1 1])~=2)
        disp('tol is not correct size! setting default tol = 1e-6')
        tol = 1e-6;
    else
        tol = tolnm;
    end
    tol
end

%  matrix size (kP = # of zeros, mP-1 = max bessel index)
kP = 16;
mP = kP;

%  matrix for zeros
Amk = zeros(mP,kP);

% Count for function evals
Nevals=0;

% Maximum # of NM root-finding iterations
itmax = 24;

%  define Newton's Method for Bessel function
nm = @(x,m) x - (2*besselj(m,x))/(besselj(m-1,x)-besselj(m+1,x));

%  For Bessel functions J_m(x), 0 <= m <= 'mP-1'
```

```
for  mm = 0:mP-1
    if mm < 1
        %  Initial guess for first root z_{0,1}
        %  using the hint given in CA4.pdf where z_{0,1} = 2.5
        zmki = 2.5;
    else
        %  For Bessel functions m > 0, first root z_{m,1} bracketed
 by:
        %  z_{m-1,1} < z_{m,1} < z_{m-1,2}Amk
        %  (zmki = initial guess as midpoint of this bracket)
        zmki = (Amk(mm,1) + Amk(mm,2))/2;
        %zmki = Amk(mm,1);
    end

    %  Find 'kP' zeros, 1 <= 'kk' <= 'kP'
    for  kk = 1:kP

        %  Set initial guess for k-th root, where k>1
        if  kk > 1
            %  The k-th root is approx. at z_{m,k-1} + pi
            zmki = Amk(mm+1,kk-1) + pi;
        end

        %  Find z_{m,k} using Newton's Method
        zn = zmki;
        check = 1;
        iter = 0;   %  Track # of NM iterations so far

        %  Root-finding using NM
        while  abs(check) > tol

            %  Stop condition for divergence
            if  iter > itmax
                if  abs(check)/abs(zn) > tol
                    fprintf("NM failed to find root: \n")
                    mm
                    kk
                    fprintf("Reason: NM doesn't converge after %d
 iterations\n", itmax)
                    break
                else
                    break
                end
            end

            %  Update approximation of root
            Amk(mm+1,kk) = nm(zn,mm);

            %  Track function evals for NM
            Nevals = Nevals + 3;

            %  Exit with message when derivative == 0
            if abs( (Amk(mm+1,kk) ) == Inf)
                fprintf("NM failed to find root: \n")
```

```
                mm
                kk
                fprintf("Reason: Derivative == 0\n")
                return
            end

            %  Update stopping condition
            check = Amk(mm+1,kk) - zn;
            %fprintf("check = %.15f\n", check);

            %  Prepare next iteration
            zn = Amk(mm+1,kk);
            iter = iter + 1;
        end
        %  kk-th root found, go next
    end
end

%
% Output mean Nevals and take residual error as rms
%
mean_Nevals = Nevals / (kP^2)

%  residual error as rms when J_m(x) evaluated at roots in Amk
Jmk = zeros(kP);
for mm = 0:mP-1
    Jmk(mm+1,:) = besselj(mm,Amk(mm+1,:));
end
bsl_reserr = rms(Jmk(:))
```

Published with MATLAB® R2018a