# MACM 316 — Computing Assignment #4

David Pham / dhpham@sfu.ca / 301318482

## Efficiency

I wanted to see how the efficiency of NM and `fzero` changed as the convergence tolerance $\epsilon = 10^{-x}$ got smaller, where $x = 3, 4, \ldots, 15$. So for each value of $\epsilon$, I computed and plotted the mean number of function evaluations required to find a root. This plot can be seen in Fig. 1. In terms of function evaluations, when both methods are using the same convergence tolerance, `fzero` is always more efficient than NM. However, *two* points bracketing the root needs to be found for `fzero` while NM only requires *one* point near the root. As well, NM requires an evaluation of the function's derivative—something that's often more convoluted than the original function. Depending on how well we can solve these problems, and depending on the form of the function, the efficiency of both methods become hard to truly judge. However, just from eyeballing it, it looks like the number of func. evals/root grows at a logarithmic rate, for both NM and `fzero`. In terms of error convergence, NM looks close to exponential When compared to BiS and SM, NM and `fzero` are incredibly efficient when enough about the function is known.

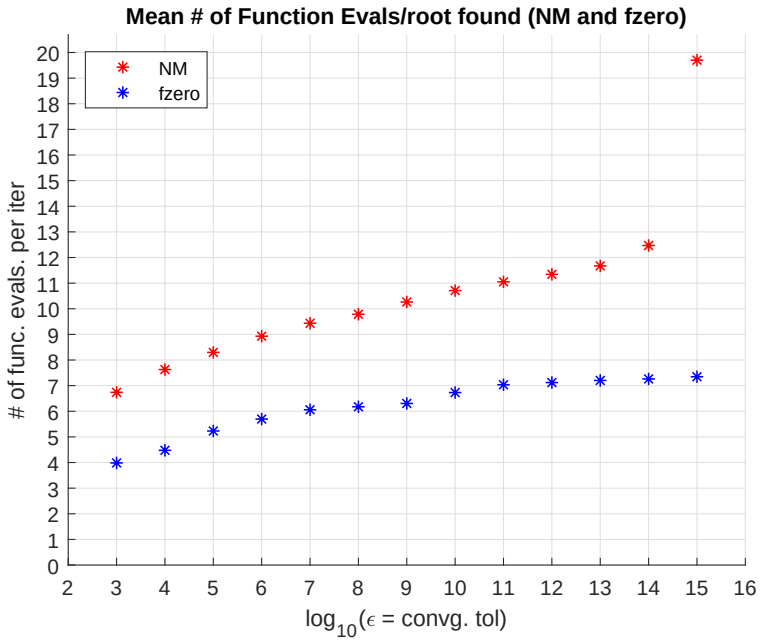


Figure 1: Average number of function evaluations per root found

## Robustness

In terms of robustness, `fzero` does alright while NM seems extremely volatile. As mentioned before, `fzero` requires you to give two points bracketting the root. However, `fzero`'s major weakness is that, when the function $f$ is evaluated at these points, *their signs must differ*. That is, `fzero` can't find the root of a function $f(x) = x^2$. While NM requires only *one* point *sufficiently close* to the root, it's incredibly hard to analyze just *how close* one must get without a great deal of knowledge about the function. As well, since NM requires a division by the derivative $f'$, the derivative must be known, and $f' \neq 0$ close to the root if quadratic convergence is desired.
Originally I had used $|p_n - p_{n-1}| < \epsilon$ as the only stopping condition, but in order to get NM converging at $\epsilon < 10^{-13}$, I had to introduce two more stopping conditions. First, I had to set some max number of iterations $N_{max}$ before I'd stop NM due to divergence. Secondly, I introduced a "seond-chance" stopping-condition $|p_n - p_{n-1}|/|p_n| < \epsilon$ for when $N_{iter} \geq N_{max}$.

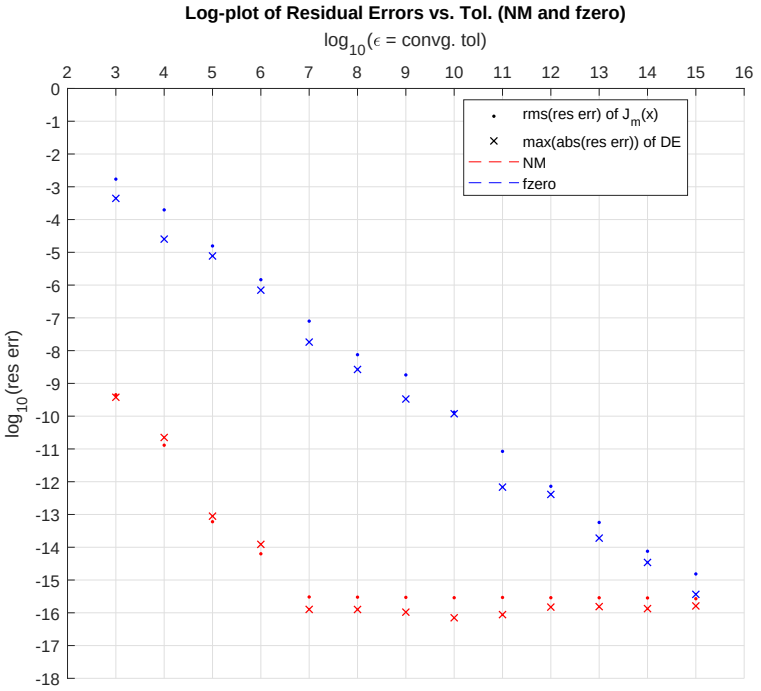


Figure 2: Residual errors of NM and `fzero`

## Accuracy

Both NM and `fzero` had no issues with accuracy, given that NM was given a good initial point and `fzero` was given a sign-change interval. Given the right conditions, NM converged quadratically, as expected, such that any $\epsilon leq 10^{-7}$ gave a residual error $\epsilon_{res} = log_{10}[rms(J_m(z_{m,k})] \approx -16$. This is an error in the last digit of a double-precision floating point number, so the error is as insignificant as possible after $x > 6$. With `fzero`, I was able to find $\epsilon \geq 10^{-17}$, though I didn't plot the data from this. Seemingly, `fzero` can get as precise as you want it to, and it stays efficient the entire way.

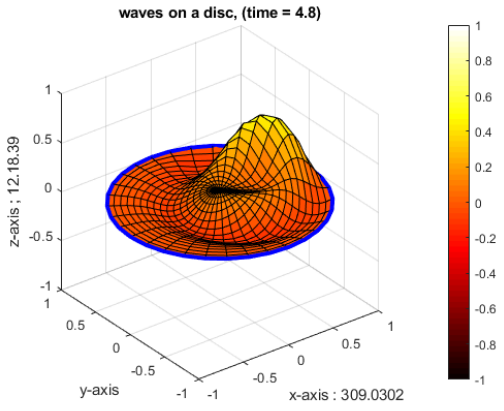


Figure 3: Last frame of gif generated by NM, `tol = 1e-15`

```
%
%  CA4_tols.m -- dhpham -- 12 feb 2019
%
%  Run CA4_nm.m and CA4_fzero.m for every value in 'tols', a vector of
%  convergence tolerances.
%
%  For each tolerance value, record the:
%       1)  mean # of function evals per root found. for each
 procedure,
%       2)  rms of the residual error given by J_m(x), at every root
 found
%       3)  maximum abs. value of the res error when solving the DE in
%           CA4_BesselMovie for 0
%
%   Create a log-plot of the residual errors, and a plot of the mean
%   Nevals.
%

clear

%  vector of convergence tolerances to test
tols = [3:15];

%    mean_Nevals:  mean # of function evals per iter. of procedure
%     bsl_reserr:  take rms of J_m(x) evaluated at all roots found
%  diffeq_reserr:  max abs val of res err; given by CA4_BesselMovie

fzero_data = struct( 'mean_Nevals', zeros(size(tols)), ...
                     'bsl_reserr',  zeros(size(tols)), ...
                     'diffeq_reserr',  zeros(size(tols)) );

nm_data = struct( 'mean_Nevals',    zeros(size(tols)), ...
                  'bsl_reserr',     zeros(size(tols)), ...
                  'diffeq_reserr',     zeros(size(tols)) );

%  Run CA4_nm.m for each value in 'tols'
for toli = tols(1):tols(end)
    %  i-th iteration
    ix = toli - tols(1) + 1;

    %  Set tol for CA4_nm.m
    tolnm = 1 * 10^(-toli);

    %  Run NM
    CA4_nm_submsn;
    %  Store data produced by NM
    nm_data.mean_Nevals(ix) = mean_Nevals;
    nm_data.bsl_reserr(ix) = bsl_reserr;

    Zmk = Amk;          %  Prepare Zmk for CA4_BesselMovie
    nofigs = true;        %  skip creating the gif
    CA4_BesselMovie_submsn;
```

```
    %  Store data produced by CA4_BesselMovie.m
    nm_data.diffeq_reserr(ix) = max(abs(real(test(:,end))/mP));
end

%  Run CA4_fzero.m for each value in 'tols'
for toli = tols(1):tols(end)
    %  i-th iteration
    ix = toli - tols(1) + 1;

    %  Set tol for CA4_fzero.m
    tolfz = 1 * 10^(-toli);

    %  Run fzero
    CA4_fzero_submsn;
    %  Store data produced by NM
    fzero_data.mean_Nevals(ix) = mean_Nevals;
    fzero_data.bsl_reserr(ix) = bsl_reserr;

    Zmk = Amk;          %  Prepare Zmk for CA4_BesselMovie
    nofigs = true;        %  skip creating the gif
    CA4_BesselMovie_submsn;

    %  Store data produced by CA4_BesselMovie.m
    fzero_data.diffeq_reserr(ix) = max(abs(real(test(:,end))/mP));
end



%  Create log-plot figure of error results for NM and fzero
figure(4000); clf

%  Set figure parameters
ax1 = gca;
title('Log-plot of Residual Errors vs. Tol. (NM and fzero)')
hold on;    grid on;

xlabel('log_{10}(\epsilon = convg. tol)')
ax1.XAxisLocation = 'top';
ax1.XLim = [min(tols-1) max(tols+1)];
ax1.XTick = [ax1.XLim(1):ax1.XLim(end)];

ylabel('log_{10}(res err)')
ax1.YLim = [-18 0];
ax1.YTick = [ax1.YLim(1):ax1.YLim(end)];


%  Create legend using dummy varaibles
invis = [NaN NaN];
dum(1,1) = plot(invis,'.k');     lbl(1,1) = "rms(res err) of J_{m}
(x)";
dum(2,1) = plot(invis,'xk');     lbl(2,1) = "max(abs(res err)) of DE";
dum(3,1) = plot(invis,'--r');    lbl(3,1) = "NM";
dum(4,1) = plot(invis,'--b');    lbl(4,1) = "fzero";
```

```
legend(ax1,dum(:,1),lbl(:,1),'AutoUpdate','off','Location','best')


%  plot NM res errs
plot(tols,log10(nm_data.bsl_reserr),'.r')
plot(tols,log10(nm_data.diffeq_reserr),'xr')

%  plot fzero res errs
plot(tols,log10(fzero_data.bsl_reserr),'.b')
plot(tols,log10(fzero_data.diffeq_reserr),'xb')



%  Create figure of mean function evals. at each 'tol' for NM and
 fzero
figure(4001); clf
title('Mean # of Function Evals/root found (NM and fzero)')

ax2 = gca;
legend(ax2,'Location','northwest')
hold on;    grid on;


%  Set axis parameters
xlabel('log_{10}(\epsilon = convg. tol)')
ax2.XAxisLocation = 'bottom';
ax2.XLim = [min(tols-1) max(tols+1)];
ax2.XTick = [ax2.XLim(1):ax2.XLim(end)];

ylabel('# of func. evals. per iter')
ax2.YLim = [0 nm_data.mean_Nevals(end)+1];
ax2.YTick = [ax2.YLim(1):ax2.YLim(end)];


%  Plot mean func. evals to find each root
plot(tols,nm_data.mean_Nevals,'*r', ...
        'DisplayName','NM')
plot(tols,fzero_data.mean_Nevals,'*b', ...
        'DisplayName', 'fzero')
```

*Published with MATLAB® R2018a*

```
%
%  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*

```
%
%  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 MATLAB's fzero.
%
%  INPUT: Use an existing 'tolfz' 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 FZERO ====')

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

%  define bessel function
bfunc = @(x,mm) besselj(mm,x);

%  fzero options, no output unless non-convergence
opt1 = optimset('TolX',tol,'Display','notify');
```

```
%  For Bessel functions J_m(x), 0 <= m <= 'mP-1'
for mm = 0:mP-1
    if mm < 1
        %  Initial bracket for first root z_{0,1}:
        %  using the hint given in CA4.pdf where z_{0,1} = 2.5
        ri = [2 3];
    else
        %  For Bessel functions m > 0, first root z_{m,1} bracketed
 by:
        %  z_{m-1,1} < z_{m,1} < z_{m-1,2}
        ri = [Amk(mm,1) Amk(mm,2)];
    end

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

        %  Use "k-1"-th root + pi as left bound of initial bracket
        if kk > 1
            %  left endpoint of interval
            rL = floor(Amk(mm+1,kk-1)+pi);

            %  check for sign change between J_m(rL) and J_m(rL+ii)
            ii = [1 2];
            sign_changes = diff(sign([bfunc(rL,mm) bfunc(rL+ii,mm)]));
            iix = find(sign_changes,1);

            ri = [rL rL+ii(iix)];
        end

        %  Find z_{m,k} using fzero
        [Amk(mm+1,kk),err,exitflag,output] = fzero( @(x)
 bfunc(x,mm),ri,opt1);

        Nevals = Nevals + output.funcCount;
    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*

```
%
%  BesselMovie.m -- djm -- 03 feb 2019
%
% - this script make an animated gif
%  file that is a solution to the
%  'wave equation' on a disc
% - you do NOT have to understand this
%  script at all
% - to use it, the matrix Zmk with size
%  16x16 must be defined by YOUR code

%  REMOVE THIS "clear"!

%  bessel (integer) index
kP = 16;  dr  = 1/(kP+1);
mP = kP;  dth = pi/mP;

%  INSTRUCTOR SOLUTION CODE
%  bessel_roots

if (exist('Zmk','var')==0)
 %  proxy matrix of bogus values
 Zmk = transpose((1:16)*1.3)*ones(1,16);
else
 %  check if student matrix is 16x16
 if (sum(size(Zmk)==[16 16])~=2)
  disp('matrix is not correct size!')
  return
 end
end

th = (0:2*mP-1)*dth;
rr = (1:kP)*dr;

[rg,tg] = meshgrid(rr,th);
[xg,yg] = pol2cart(tg,rg);

%  make initial condition
pk = 16;
yy = exp(-pk*((xg-0.5).^2 + yg.^2)).*((1 - rg.^4).^2);
yp = -2*pk*yg.*yy ...
     -8*exp(-pk*((xg-0.5).^2 + yg.^2)).*(1 - rg.^4).*(rg.^2).*yg;

%  fourier transforms
yyfft = fft(yy,[],1);
ypfft = fft(yp,[],1);

%  build coefficients

Cmk = zeros(mP,kP);
Dmk = zeros(mP,kP);
```

```
for mm = 0:mP-1;
 for kk = 1:kP
  alpha = Zmk(mm+1,kk);
  Cmk(mm+1,kk) = sum(yyfft(mm+1,:).*besselj(mm,alpha*rr).*rr)*dr;
  Dmk(mm+1,kk) = sum(ypfft(mm+1,:).*besselj(mm,alpha*rr).*rr)*dr;

  bprime = (besselj(mm+1,alpha) - besselj(mm-1,alpha)) / 2;

  if (mm==0)
   Cmk(mm+1,kk) = 1*Cmk(mm+1,kk)/(bprime^2);
   Dmk(mm+1,kk) = 1*Dmk(mm+1,kk)/(bprime^2);
  else
   Cmk(mm+1,kk) = 2*Cmk(mm+1,kk)/(bprime^2);
   Dmk(mm+1,kk) = 2*Dmk(mm+1,kk)/(bprime^2);
  end
 end
end

%  plot coordinates
[rp,tp] = meshgrid([0 rr 1],[th 2*pi]);
[xp,yp] = pol2cart(tp,rp);

%  zero check plot
test = zeros(size(rp));

for mm = 0:mP-1
 for kk = 1:kP
  test = test + Cmk(mm+1,kk)*besselj(mm,Zmk(mm
+1,kk)*rp).*exp(1i*mm*tp);
 end
end

%  Don't create any figures if 'nofigs' variable exists and is true
if(exist('nofigs','var') == 1)
    if nofigs == true
        return
    end
end
figure(3000);  clf
plot(real(test(:,end))/mP,'kx')
title('this plot should show numerical zeros! (what does this mean?)')
xlabel(['\theta-axis (max abs val = '
 num2str(max(abs(real(test(:,end))/mP))) ')'])
ylabel('surface values at r=1 as a function of \theta')

%  initialize for animated gif
h = figure(3001);
axis tight manual % this ensures that getframe() returns a consistent
 size
filename = 'BesselMovie.gif';

%  make animation
figure(3001);  clf
for tt = [(0:48)*0.1]
```

```
    soln = zeros(size(rp));
    for mm = 0:mP-1
        for kk = 1:kP
            next = Cmk(mm+1,kk)*besselj(mm,Zmk(mm
+1,kk)*rp).*exp(1i*mm*tp);
            soln = soln + next*cos(Zmk(mm+1,kk)*tt);
            next = Dmk(mm+1,kk)*besselj(mm,Zmk(mm
+1,kk)*rp).*exp(1i*mm*tp);
            soln = soln + next*sin(Zmk(mm+1,kk)*tt)/Zmk(mm+1,kk);
        end
    end

    surf(xp,yp,real(soln)/mP);  hold on
    plot3(xp(:,end),yp(:,end),real(soln(:,end)/mP),'b','linewidth',3);
  hold off
    title(['waves on a disc, (time = ' num2str(tt) ')'])
    xlabel(['x-axis : ' num2str(sum(Zmk(:,end)/pi))])
    cl = clock;
    zlabel(['z-axis ; ' num2str(cl(3)) '.' num2str(cl(4)) '.'
 num2str(cl(5))])
    ylabel('y-axis')
    axis equal
    axis([-1 1 -1 1 -1 1])
    caxis([-1 1])
    colormap(hot)
    colorbar
    drawnow

    % Capture the plot as an image
    frame = getframe(h);
    im = frame2im(frame);
    [imind,cm] = rgb2ind(im,256);

    % Write to the GIF File
    if (tt==0)
        imwrite(imind,cm,filename,'gif', 'Loopcount',1);
    else
        imwrite(imind,cm,filename,'gif','WriteMode','append');
    end
end
```

*Published with MATLAB® R2018a*