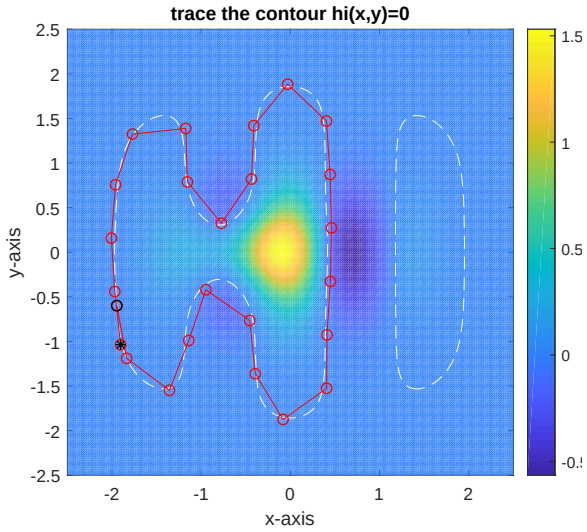
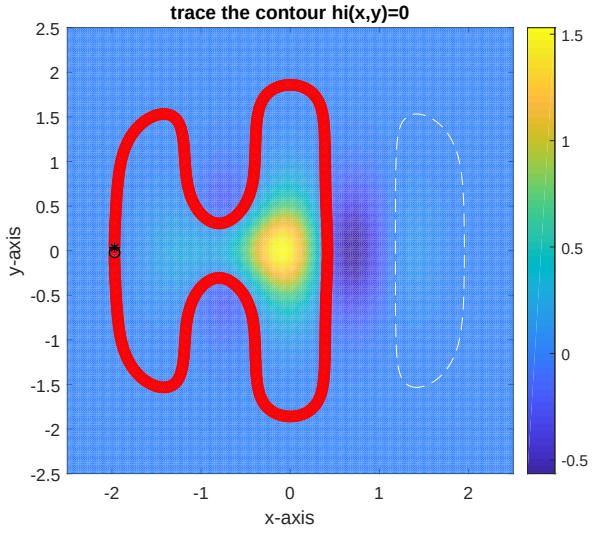
MACM 316 — Computing Assignment #3

David Pham // dhpham@sfu.ca // 301318482

Reproducing the Rough Sketch



(a) Traced by BiS ($\Delta s = 0.6,\ \delta = \frac{3\pi}{4}$)
`Nsteps` $= 24$



(b) Visually smooth curve traced by BiS
($\Delta s = 0.02,\ \delta = \frac{3\pi}{4}$)
`Nsteps` $= 728$

How δ and Δs Affect Convergence

Both methods required a $\Delta s = 0.02$ to produce a visually smooth curve. However, the BiS method can work with both large and small values of Δs, whereas SM requires a relatively small Δs value. With regards to δ, the BiS method requires values of δ where 2δ gets close enough to 2π, such that the interval $[\theta - \delta, \theta + \delta]$ represents something close to a closed circle. If δ is too small, then the values searched may be too narrow: maybe only a half-circle in the direction of θ_n is searched and no sign change is found, since the root lies on the other half of the circle. This becomes less of an issue if Δs becomes smaller alongside δ. For example, $\Delta s = 0.6,\ \delta = \pi/2$ makes BiS fail as it rounds the bottom-left bend of the contour, but BiS using $\Delta s = 0.0001,\ \delta = \pi/2$ is able to trace the contour.

The opposite is true for SM: values of $\Delta s \geq 0.1$ practically don't converge at all, and smaller values of δ are better. SM can handle values like $\delta = \pi/10^{10}$, whereas the smallest δ that BiS can handle is $\frac{\pi}{3}$ for $\Delta s = 0.0001$. For any smaller values of δ, BiS fails no matter how small Δs gets. However, for larger δs, it becomes increasingly difficult for SM to converge.

Cost

With my implementation of BiS, larger δ values lead to higher cost. Since the sign of θ may change multiple times over $[\theta - \delta, \theta + \delta]$, I evaluate HI at each point in `[`$\theta - \delta$`:0.1:`$\theta + \delta$`]`. This, in addition with the `avg_evals = 31` it takes for BiS to reach convergence, makes BiS much more costly than SM to run.

For SM, as both Δs and δ get smaller, SM begins to converge faster. At $\Delta s = 0.1,\ \delta = \pi/2$, `avg_evals = 8.34`. For a smaller $\Delta s = 0.001$ and the same δ as before, `avg_evals = 5.99`. With a smaller $\delta = \pi/10^{10}$ and $\Delta s = 0.1$, `avg_evals = 5.86`. With both Δ and δ small at $\Delta s = 0.001,\ \delta = \pi/10^{10}$, `avg_evals = 3.98`. Thus at its fastest, SM is only using two extra function evaluations, aside from the two required evaluations.

```
%
%  CA3_SM.m -- dhp -- 29 jan 2019
%

clear

%  hi function

%  define domain in x and y
xx = -2.5:0.025:2.5;  yy = -2.5:0.025:2.5;

%  define a 'mesh' to plot on
[xg,yg] = meshgrid(xx,yy);

%  set root-finding tolerance (for initial pt & loop)
tol = 1e-10;
fzero_opt = optimset('TolX',tol);

%  root-finding loop control parameters
%ds = 0.6;
ds = 0.1;
%  useful future variables
itmax = 24;
%delta = pi/2;
delta = pi/2;

%  define the function HI(x,y)
hi = @(x,y) exp(-3*((x + 0.5).^2 + 2*y.^2)) + exp(-x.^2 -
 2*y.^2).*cos(4*x) - 1e-3;

%  define the function HI on the circle (radius = ds)
hi_th = @(th,xn,yn) hi(xn + ds*cos(th),yn + ds*sin(th));

%  find point on the "H" with y=0
%  initial guess for a point very NEAR contour
xi = -1.97; yi = 0;

%  START:  FIND INITIAL POINT on contour (you can use fzero here)
%  root-find angle to point ON contour
th = 0;
th = fzero(@(th) hi_th(th,xi,yi),th,fzero_opt);
%
%  END  :  FIND INITIAL POINT on contour

%  compute first point ON contour
xn = xi + ds*cos(th);
yn = yi + ds*sin(th);

%  make array of contour points
Nsteps = 24000;
zero_contour = zeros(Nsteps+1,2);
zero_contour(1,:) = [xn yn];
```

```
total_evals = 0;

%  loop for the contour
for kk = 1:Nsteps
 %  START:  theta root-finding here (you cannot use fzero here!!)
 %
 %  Secant Method for next point

    %  0)  set two initial guesses
    th0 = th-delta;  hi_th0 = hi_th(th0,xn,yn);
    thn = th+delta;
    Nevals = 1;

    check = thn-th0;

    %  root-finding loop
    while( abs(check) > tol )
        if Nevals > itmax
            fprintf('no convergence:\n')
            fprintf('\tds = %f, delta = %f\n', ds, delta)
            break
        end
        %  1)  function evaluation at thn
        hi_thn = hi_th(thn, xn,yn);
        Nevals = Nevals + 1;

        %  2)  secant update
        thS = thn - (hi_thn * (thn - th0)/(hi_thn - hi_th0));

        %  3)  prepare next iteration
        th0 = thn;  hi_th0 = hi_thn;
        thn = thS;

        check = thn - th0;
    end

    total_evals = total_evals + Nevals;

    %
    %  END:  theta root-finding here
    if Nevals > itmax
        break
    end

    % Compute next point on contour
 xn = xn + ds*cos(thn);
 yn = yn + ds*sin(thn);

    %  update new points & angle
 zero_contour(kk+1,:) = [xn yn];
 th = thn;
end
```

```
avg_evals = total_evals / kk

%  colour contourplot of HI function
figure(2);  clf
pcolor(xx,yy,hi(xg,yg));  colorbar
shading interp;  hold on
contour(xx,yy,hi(xg,yg),[0 0],'w--')
axis equal;  axis image

title('trace the contour hi(x,y)=0')
xlabel('x-axis')
ylabel('y-axis')

%  plot the zero-contour, 1st & last point
plot(zero_contour(:,1),zero_contour(:,2),'ro-')
plot(zero_contour(1,1),zero_contour(1,2),'ko')
plot(zero_contour(end,1),zero_contour(end,2),'k*')
```

Published with MATLAB® R2018a

```
%
%  CA3_BiS.m -- dhp -- 29 jan 2019
%

clear

%  hi function

%  define domain in x and y
xx = -2.5:0.025:2.5;  yy = -2.5:0.025:2.5;

%  define a 'mesh' to plot on
[xg,yg] = meshgrid(xx,yy);

%  set root-finding tolerance (for initial pt & loop)
tol = 1e-10;
fzero_opt = optimset('TolX',tol);

%  root-finding loop control parameters
ds = 0.6;
%  useful future variables
%  itmax = 24;
delta = 3*pi/4;

%  define the function HI(x,y)
hi = @(x,y) exp(-3*((x + 0.5).^2 + 2*y.^2)) + exp(-x.^2 -
 2*y.^2).*cos(4*x) - 1e-3;

%  define the function HI on the circle (radius = ds)
hi_th = @(th,xn,yn) hi(xn + ds*cos(th),yn + ds*sin(th));

%  function concerns radius of circle centered at (x0,y0) with radius
 = ds
%  and subtracts distance b/w point (x,y) and (x0,y0) from radius of
 circle
%
circ_rmndr = @(x,y,x0,y0) ds^2 - (x-x0)^2 - (y-y0)^2;

%  find point on the "H" with y=0
%  initial guess for a point very NEAR contour
xi = -1.97; yi = 0;

%  START:  FIND INITIAL POINT on contour (you can use fzero here)
%  root-find angle to point ON contour
th = 0;
th = fzero(@(th) hi_th(th,xi,yi),th,fzero_opt);
%
%  END  :  FIND INITIAL POINT on contour

%  compute first point ON contour
xn = xi + ds*cos(th);
yn = yi + ds*sin(th);
```

```
%  make array of contour points
Nsteps = 24;
zero_contour = zeros(Nsteps+1,2);
zero_contour(1,:) = [xn yn];

Nevals = 0;
%  loop for the contour
for kk = 1:Nsteps
    %  START:  theta root-finding here (you cannot use fzero here!!)
    %
    %  BiS for next angle, using previous angle as initial guess

    %  bracketting list
    th_list = [th-delta:0.1:th+delta];
    hi_list = hi_th(th_list,xn,yn);
    sign_check = sign(hi_list);
    sign_diffs = diff(sign_check);

    %  plot radius of theta root-finding circle
    %  x_th = xn + ds*cos(th_list);
    %  y_th = yn + ds*sin(th_list);
    %  plot(x_th(:),y_th(:),'m--')

    %  check for any exact zeros!!
    ix = find(sign_check==0);
    if ( isempty(ix) ~= 1 )
        thn = th_list(ix);
    else
        %  check for sign-change interval
        ix = find(sign_diffs);
        if ( isempty(ix) == 1 )
            disp('no sign change')
            break
        else
            %  choose theta root-find in correct direction
            fwd_root = false;

            %  BEGIN:  Direction finding
            for ii = ix
                thi = th_list(ii);
                xxi = xn + ds*cos(thi);
                yyi = yn + ds*sin(thi);

                %  see if root lies within radius of previous search
                if( kk > 1 )
                    circ_dist1 =
 circ_rmndr(xxi,yyi,zero_contour(kk-1,1),zero_contour(kk-1,2));
                    if( circ_dist1 > 1e-15 )
                        continue
                    end
                end
                %  see if root lies within radius of second previous
 search
```

```
                if( kk > 2 )
                    circ_dist2 =
circ_rmndr(xxi,yyi,zero_contour(kk-2,1),zero_contour(kk-2,2));
                    if( circ_dist2 > 1e-15 )
                        continue
                    end
                end

                % new root outside radius of past two searches:
progress probably being made
                ix = ii;
                fwd_root = true;
                break
            end

            if( fwd_root == false)
                disp('no further progress possible');
                break;
            else
                thn = th_list(ix);
            end
            %  END:  Direction finding
        end
    end

    %  0)  set sign-change interval
    thL = th_list(ix  );  hi_thL = hi_th(thL,xn,yn);
    thR = th_list(ix+1);  hi_thR = hi_th(thL,xn,yn);
    Nevals = Nevals + 2;

    %  1)  compute first midpoint
    check = (thR-thL)/2;

    thB  = thL + check;

    %  root-finding loop
    while (abs(check)>tol)
        %  2)  function evaluation
        hi_thB = hi_th(thB,xn,yn);
        Nevals = Nevals + 1;

        %  3)  decision
        if (hi_thB==0)
            thL = thB;  hi_thL = hi_thB;
            thR = thB;  hi_thR = hi_thB;
        else
            if (hi_thL*hi_thB > 0)
                thL = thB;  hi_thL = hi_thB;
            else
                thR = thB;  hi_thR = hi_thB;
            end
        end

        %  4)  prepare next iteration
```

```
        check = (thR-thL)/2;

        thB  = thL + check;
    end

    %
    %  END:  theta root-finding here

    % Compute next point on contour
    xn = xn + ds*cos(thn);
    yn = yn + ds*sin(thn);

    %  update new points & angle
    zero_contour(kk+1,:) = [xn yn];
    th = thn;
end

avg_evals = Nevals/Nsteps;

%  colour contourplot of HI function
figure(1);
pcolor(xx,yy,hi(xg,yg));  colorbar
shading interp;  hold on
contour(xx,yy,hi(xg,yg),[0 0],'w--')
axis equal;  axis image

title('trace the contour hi(x,y)=0')
xlabel('x-axis')
ylabel('y-axis')

%  plot the zero-contour, 1st & last point
plot(zero_contour(:,1),zero_contour(:,2),'ro-')
plot(zero_contour(1,1),zero_contour(1,2),'ko')
plot(zero_contour(end,1),zero_contour(end,2),'k*')
```

Published with MATLAB® R2018a