MACM 316 - Computing Assignment 3 Report

Colton Blackwell
301451278

Purpose

This report aims to investigate and compare two numerical methods, namely MATLAB's `fzero` and the secant method, to analyze and compute the zero contour of a 2D function HI(x,y). The paper begins by outlining the concept of 2D contours and their transformation into a series of 1D root-finding problems. Following this, the report conducts experiments with various h and δ values to optimize the contour tracing process. Results encompass findings from reproducing a designated figure using the secant method, determining the number of steps required for different parameter combinations, observing convergence failures, and providing potential explanations for these anomalies. The efficiency and robustness of both methods are then evaluated using time measurements and convergence analysis.

The importance of this report lies in its implications for numerical analysis and computational mathematics. Root finding is a fundamental problem in various scientific and engineering applications. Understanding the performance of different root-finding methods, such as the secant method and MATLAB's fzero, can lead to more accurate and reliable solutions in practical scenarios.

Observations

Attempting to recreate the target image using the secant method with h=0.6 and δ=π/2 produces Figure 1, where a line with points diverges from the contour line to negative infinity. Reducing h to 0.1 and setting δ=π/4 uncovers a gradual appearance of more lines, which slowly start to outline the contour (Fig. 2).

When combining these parameters (h = 0.1, delta = pi/4, niter = 145), the algorithm can effectively trace the contour curve by taking small steps (h = 0.1) with a suitable interval size (delta = pi/2) and sufficient iterations (niter = 145) to cover the contour without missing parts or overshooting (Fig. 2). This combination strikes a balance between accuracy, convergence, and computational efficiency, allowing the algorithm to trace all the way around the curve successfully.

Analyzing the computations from both implementations reveals intriguing findings. By utilizing the tic and toc syntax, we can note that the time taken for fzero to trace the bounds was 0.008882 seconds, noticeably slower compared to the secant method's 0.005940 seconds.

Understanding

When the step size h is smaller, such as h = 0.1, the algorithm takes smaller steps along the contour, leading to a more accurate tracing of the contour line HI(x,y) = 0. This finer granularity in step size allows the algorithm to capture the intricate details of the contour, resulting in a successful tracing of the contour line. On the other hand, when this increased to a larger value like h= 0.6 or greater, the algorithm takes larger steps along the contour. This larger step size can cause the algorithm to overshoot or miss crucial points, leading to an inability to trace the line accurately. The relationship between the step size hand the ability to trace the contour line effectively can be understood in terms of the trade-off between accuracy and computational efficiency. Smaller step sizes improve accuracy but may require more computational iterations to trace the contour fully. Conversely, larger step sizes may improve computational efficiency but can compromise the accuracy of the traced contour.

A notable difference between the secant method and fzero is their convergence behavior. The secant method typically exhibits a gradual improvement in accuracy with each iteration. On the other hand, fzero may not always show a similar improvement in accuracy with increasing iterations, which can sometimes lead to disappointing results, especially for complex functions or poorly chosen initial guesses

Convergence errors can occur if the initial guesses θ_0 and θ_n are poorly chosen, leading to divergence, particularly if HI(x, y) has steep gradients or is not well-behaved near the guesses. Additionally, with larger step sizes, the method may overshoot the root, missing contour points and requiring corrections in subsequent iterations. To mitigate these errors, we should dynamically adjust h and $\boldsymbol{\delta}$ based on the function's behavior to balance accuracy and computational efficiency. In the future, preprocessing HI(x, y) to smooth out discontinuities or rapid changes could further enhance the method's stabiliy.

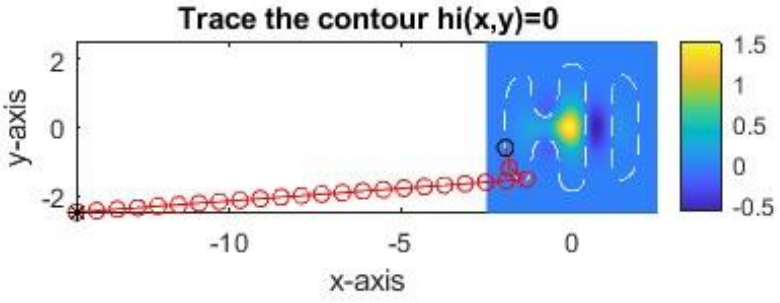

Fig. 1

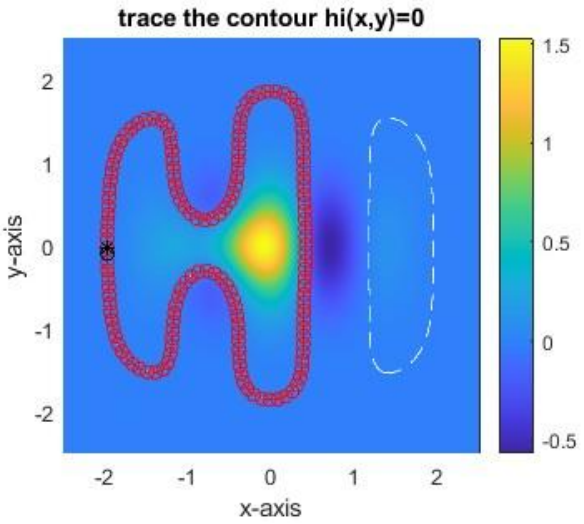

Fig. 2

CA3_demo.m

```
% CA3_demo.m -- djm -- jsm edit
%

% Define the 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-6;
fzero_opt = optimset('TolX',tol);

% Root-finding loop control parameters

h = 0.6;
delta = pi/2;

% Define the function HI(x,y)
hi = @(x,y) exp(-3*((x + 0.5).^2 + 2*y.^2)) + exp(-

% Define the function HI on the circle (radius = h)
hi_th = @(th,xn,yn) hi(xn + h*cos(th),yn + h*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 (using fzero
% 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 + h*cos(th);
yn = yi + h*sin(th);

% Make array of contour points
Nsteps = 24; % Increase the number of steps for mor
zero_contour = zeros(Nsteps+1,2);
zero_contour(1,:) = [xn yn];

% Loop for the contour
tic;
for kk = 1:Nsteps
    % START: theta root-finding here (using secant
    xint = [th - delta, th + delta];
    [thn, niter, xlist] = secant(@(th) hi_th(th,xn,
    % END: theta root-finding here

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

secant.m

```
function [root, niter, xlist] = secant( func, xint, tol )
% SECANT: Secant method for solving a nonlinear equation.
%
%  Sample usage:
%    [root, niter, xlist] = secant( func, xint, tol )
%
%  Input:
%     func  - function to be solved
%     xint  - interval [xleft,xright] bracketing the root
%     tol   - convergence tolerance (OPTIONAL, defaults to 1e-6)
%
%  Output:
%     root  - final estimate of the root
%     niter - number of iterations required to converge
%     xlist - list of iterates, an array of length 'niter'

% First, do some checking on the input parameters:
if nargin < 2
  fprintf(1, 'SECANT: must be called with at least two arguments' );
  error( 'Usage:  [root, niter, xlist] = secant( func, xint, [tol] )' );
end
if length(xint) ~= 2, error( 'Parameter ''xint'' must be a vector of length 2.' ), end
if nargin < 3, tol = 1e-6; end   % default value for 'tol'
% fcnchk(...) allows a string function to be sent as a parameter, and
% coverts it to the correct type to allow evaluation by feval().
func = fcnchk(func);

a = xint(1); b = xint(2);
c = b;
% Always start with f(a) > f(b)
fa = feval( func, a );
fb = feval( func, b );
if( fa < fb )
  ta  = a;   a = b;   b = ta;   % swap a and b
  tfa = fa; fa = fb; fb = tfa;  % swap f(a) and f(b)
end

xlist = [ a; b ];
niter = 0;

while( abs(a-b) > tol )     % absolute error tolerance
  % Compute the point where the secant line joining
  % a and b intersects the x-axis
  c  = b - fb * (a-b) / (fa-fb);
  fc = feval( func, c );
  xlist = [ xlist; c ];     % accumulate list of x-values
  a = b;  fa = fb;
  b = c;  fb = fc;
  niter = niter + 1;
end

root = c;
%END secant.
```

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

    % Colour contour plot 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*');
```