

Τεχνικές Βελτιστοποίησης

Ελαχιστοποίηση με χρήση παραγώγων

Θεόδωρος Κατζάλης

AEM: 9282

katzalis@auth.gr

17/11/2021

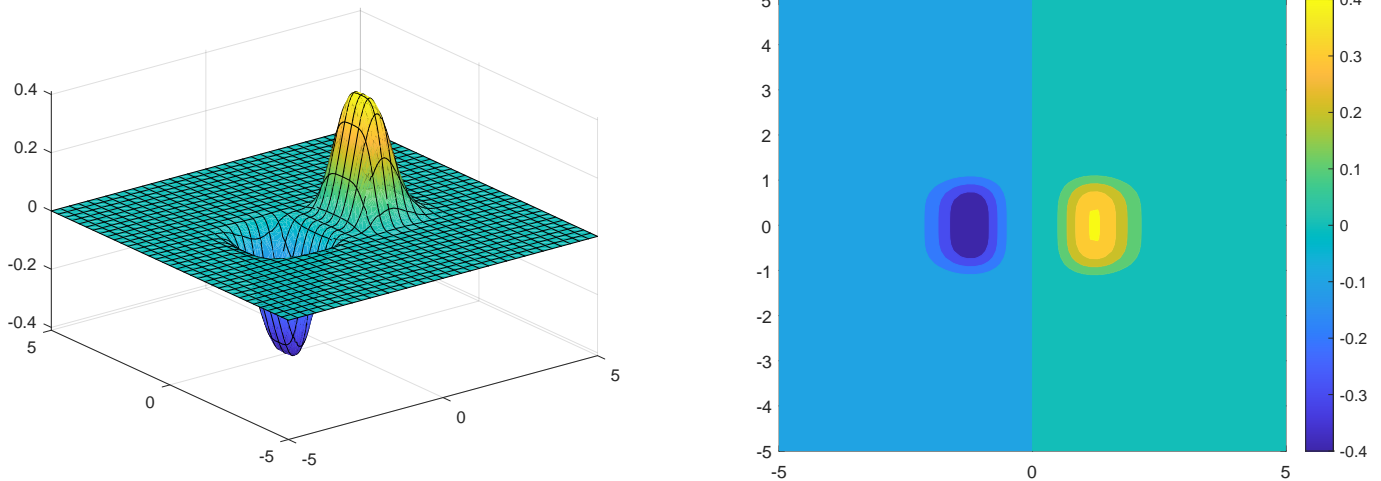
Περιεχόμενα

Θέμα 1 - Εισαγωγή	2
Θέμα 2 - Μέθοδος Μέγιστης Καθόδου (Steepest Descent)	2
Θέμα 3 - Μέθοδος Newton	4
Θέμα 4 - Μέθοδος Levenberg-Marquardt	5
Matlab κώδικας	6

Θέμα 1 - Εισαγωγή

Σκοπός αυτής της εργασίας είναι η μελέτη διάφορων μεθόδων ελαχιστοποίησης μιας δοσμένης συνάρτησης πολλών μεταβλητών $f : \mathbb{R}^n \rightarrow \mathbb{R}$ χωρίς περιορισμούς. Οι μέθοδοι που ακολουθούν είναι επαναληπτικοί. Για την αντικειμενική συνάρτηση έχουμε:

$$f(x, y) = x^3 e^{-x^2 - y^4} \quad (1)$$



Εικόνα 1: Γραφική αναπαράσταση της f

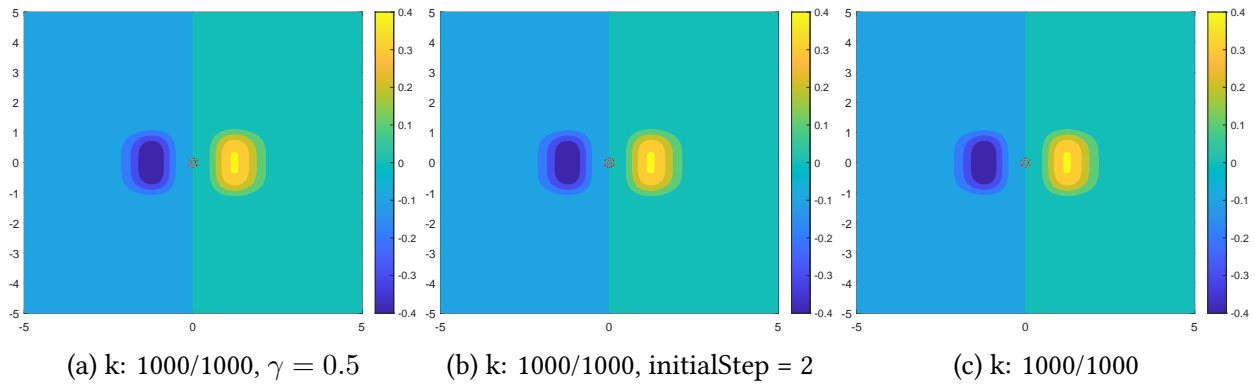
Τα αρχικά σημεία των αλγορίθμων είναι $(0,0)$, $(-1,-1)$ και $(1,1)$.

Θέμα 2 - Μέθοδος Μέγιστης Καθόδου (Steepest Descent)

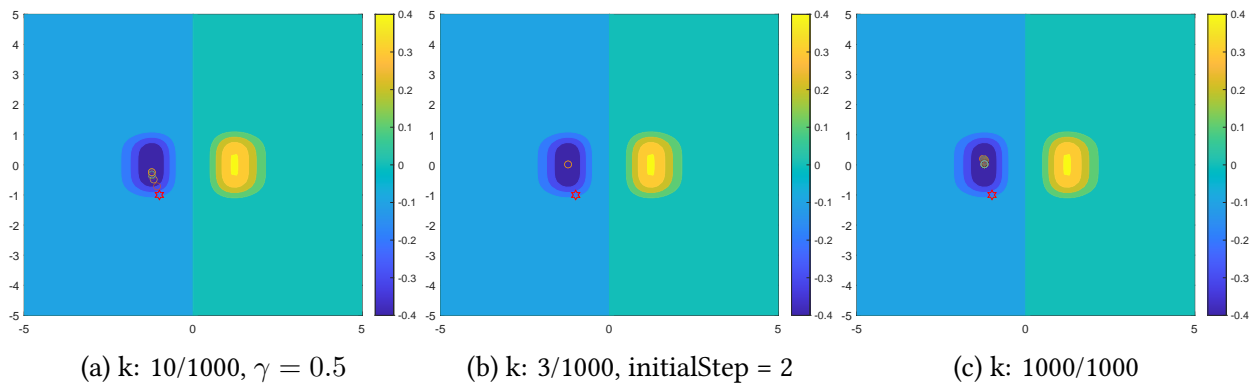
Η συγκεκριμένη μέθοδος περιγράφεται από την σχέση: $x_{k+1} = x_k - \gamma_k \nabla f(x_k)$ και τερματίζεται όταν $|\nabla f(x_k)| < \epsilon$, όπου ϵ σταθερά της επιλογής μας. Στην συγκεκριμένη περίπτωση έχουμε $\epsilon = 10^{-6}$. Για να αποφύγουμε ατέρμονους βρόγχους σε περίπτωση που ο αλγόριθμος αποτύχει να εντοπίσει το επιθυμητό ελάχιστο σημείο, έχουμε μια επιπλέον συνθήκη για τον επιτρεπόμενο μέγιστο αριθμό επαναλήψεων (1000).

Η επιλογή της τιμής γ , καθορίζει το πόσο σύντομα ή μεγάλα θα είναι τα άλματα μεταξύ επαναλήψεων. Μικρή τιμή του γ σημαίνει πιο προσεκτική εξέλιξη των βημάτων άρα μπορούν να αποφευχθούν τυχόν αστάθειες και ταλαντώσεις γύρω από το επιθυμητό σημείο. Ωστόσο, προκαλεί την αύξηση της πολυπλοκότητας του αλγορίθμου εξαιτίας των περισσότερων αριθμών επαναλήψεων. Από την άλλη, μεγαλύτερο γ , προκαλεί μικρότερες επαναλήψεις αλλά υπάρχει η πιθανότητα αστάθειας.

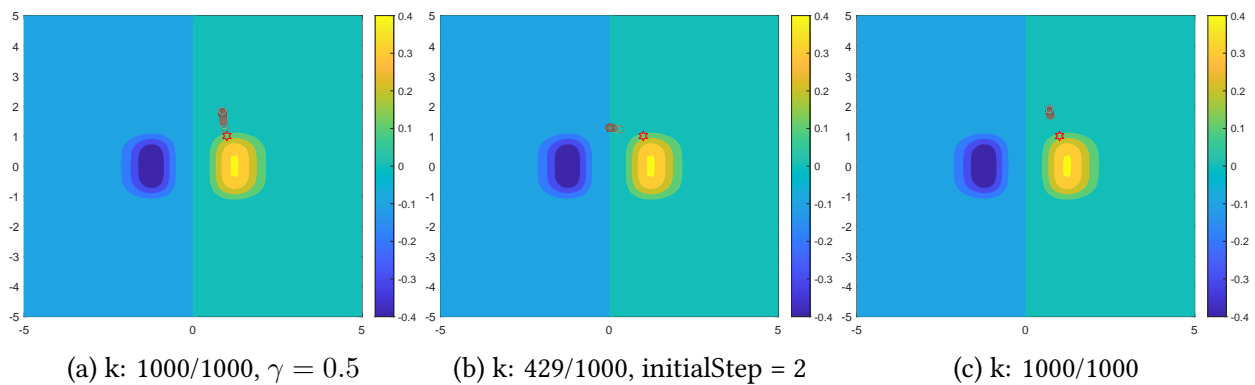
Στη συνέχεια παραθέτουμε τρία διαφορετικά διαγράμματα για κάθε αρχικό σημείο (το αρχικό σημείο αναφοράς είναι το κόκκινο εξάγωνο). Αρχικά θα μελετήσουμε 3 διαφορετικές επιλογές του γ , α) σταθερό, β) τέτοιο ώστε ελαχιστοποιεί την τιμή $f(x_k + \gamma_k * (-\nabla f(x_k)))$ και γ) με τον κανόνα Armijo:



Εικόνα 2: (0,0)



Εικόνα 3: (-1,-1)

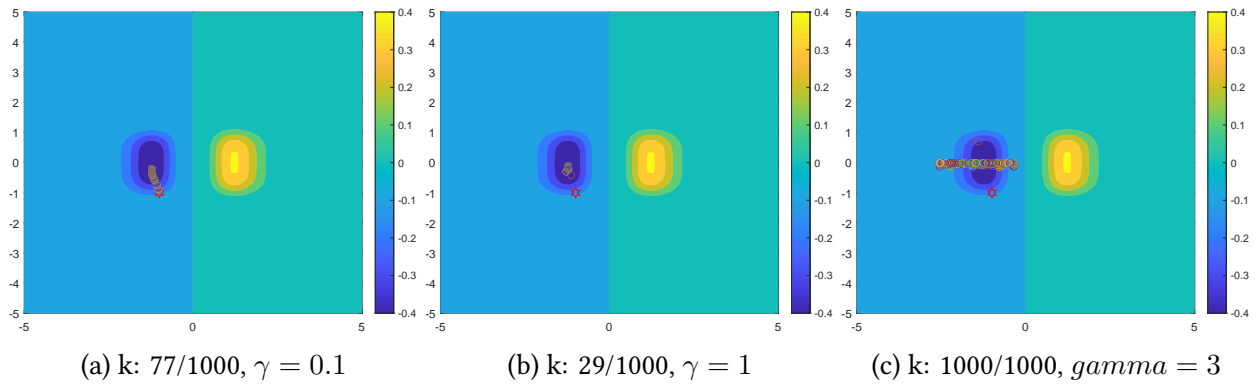


Εικόνα 4: (1,1)

Παρατηρούμε ότι η επιλογή του αρχικού σημείου παίζει καθοριστικό ρόλο για την επιτυχία του αλγορίθμου. Αυτό είναι φυσιολογικό αφού η εξέλιξη των σημείων καθορίζεται απο το διάνυσμα $\nabla f(x)$ και την περιοχή στην οποία υπολογίζεται. Το σημείο (-1,-1) είναι το μόνο το οποίο θέτει τις κατάλληλες προδιαγραφές έτσι ώστε το διάνυσμα $\nabla f(x)$ να εντοπίσει το ολικό ελάχιστο.

Όσον αφορά το σημείο (-1,-1) θα συγκρίνουμε τις επιλογές του γ . Για την επιλογή του σταθερού σημείου γ παρατηρούμε ότι υπάρχει μια αργή αλλά σταδιακή εξέλιξη των τιμών. Για την ελαχιστοποίηση του $f(x_k + \gamma * (-\nabla f(x)))$ παρατηρούμε μια πιο αποτελεσματική προσέγγιση του ελάχιστης τιμής όπως και με τον κανόνα αρμijo, ο οποίος κάνει ένα μεγάλο άλμα αρχικά και μετά πολύ μικρά βήματα γύρω απο το επιθυμητό σημείο της ελάχιστης τιμής.

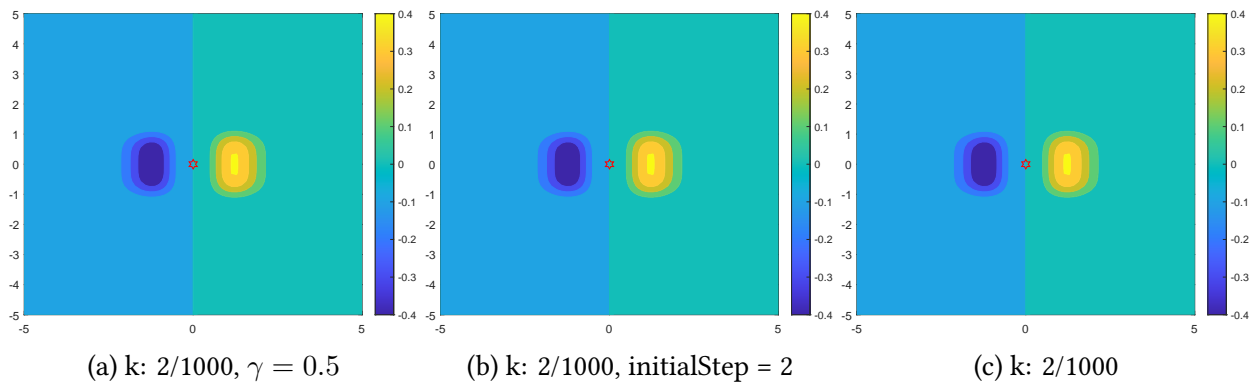
Στη συνέχεια για το σημείο (-1,-1) και για σταθερό γ , μπορούμε να δούμε πως διαφορετικές τιμές του γ επηρεάζουν τον αλγόριθμο:



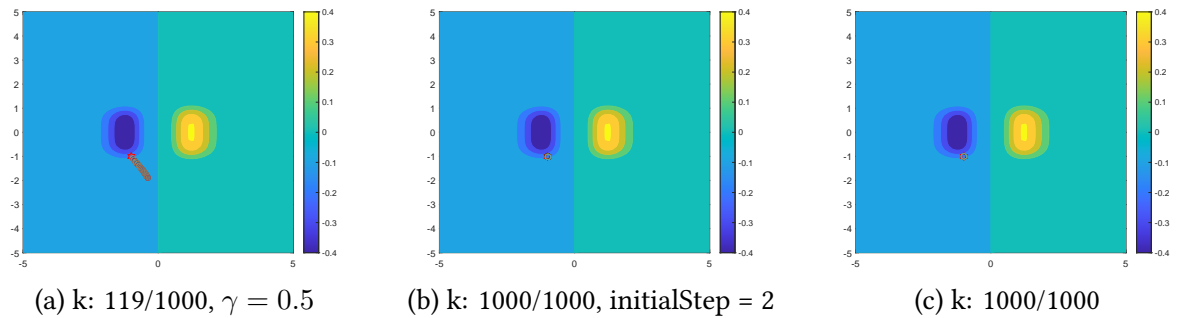
Εικόνα 5: (1,1)

Θέμα 3 - Μέθοδος Newton

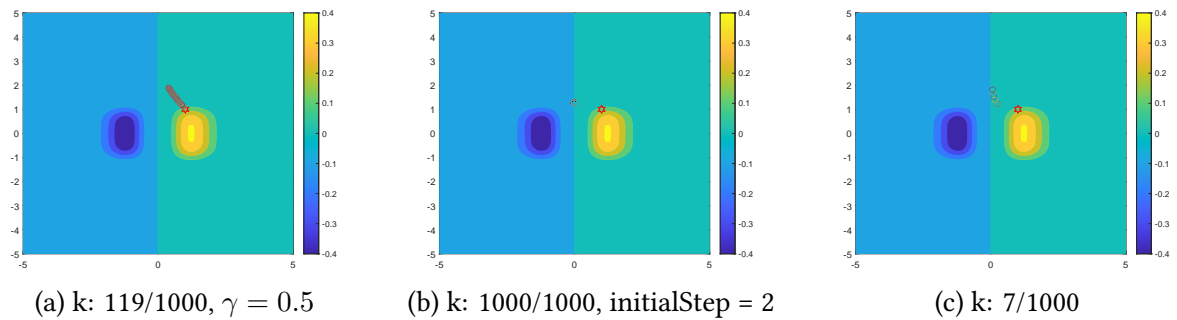
Για την μέθοδο Newton έχουμε:



Εικόνα 6: (0,0)



Εικόνα 7: (-1,-1)

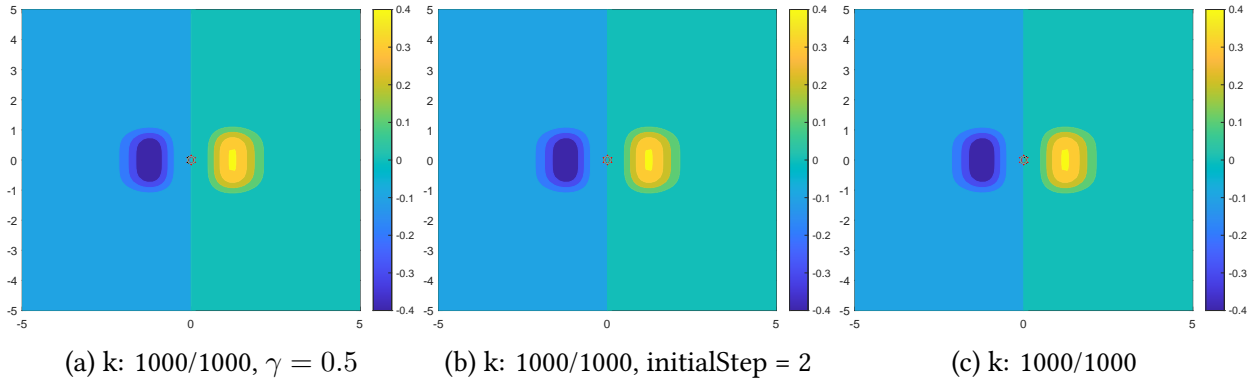


Εικόνα 8: (1,1)

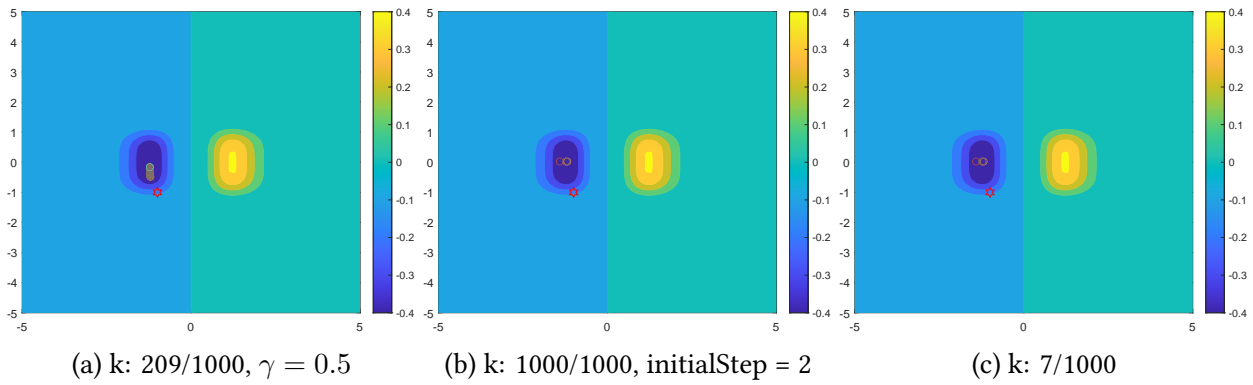
Για την μέθοδο Newton παρατηρούμε ότι αποτυγχάνει να συγκλίνει προς την επιθυμητή τιμή.

Θέμα 4 - Μέθοδος Levenberg-Marquardt

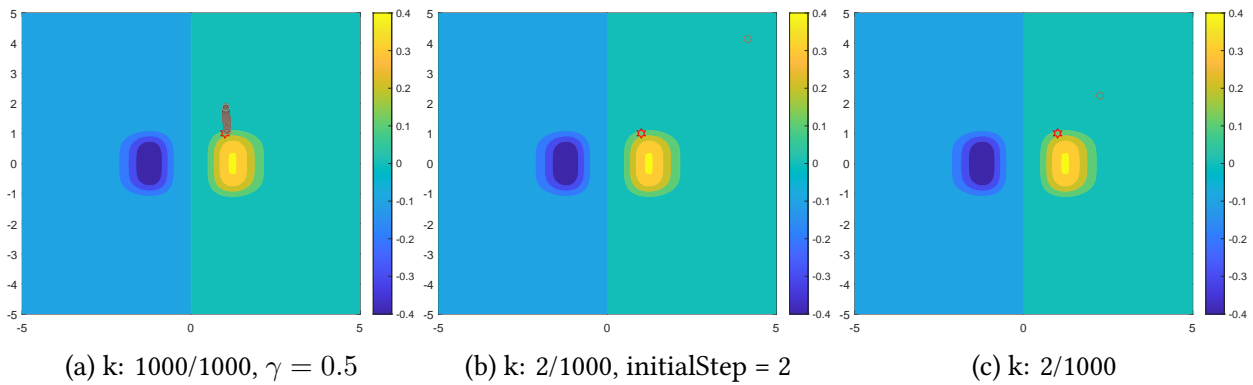
Για την μέθοδο Levenberg-Marquardt έχουμε:



Εικόνα 9: (0,0)



Εικόνα 10: (-1,-1)



Εικόνα 11: (1,1)

Για την μέθοδο Levenberg-Marquardt παρατηρούμε ότι για αρχικό σημείο (-1,-1) ο αλγόριθμος συγκλίνει προς την επιθυμητή τιμή και μάλιστα με τον κανόνα armijo αυτό συμβαίνει σε μόλις 7 βήματα.

Matlab κώδικας

Για όλα τα παραπάνω παρατίθεται ο κώδικας Matlab.

```
1 %% MINIMIZATION USING DERIVATIVES
2 clear
3 clc
4
5 %% PLOT OBJECTIVE FUNCTION
6
7 f = @(x,y) x^3 * exp(-x^2-y^4);
8 %figure; fsurf(f)
9 %exportgraphics(gcf,'surf.pdf','ContentType','Vector')
10 %figure; fcontour(f,'Fill','On'); colorbar
11 %exportgraphics(gcf,'contour.pdf','ContentType','Vector')
12
13 initialConditions = [0,0 ; -1,-1 ; 1,1];
14 epsilon = 1e-6;
15 maxIterations = 1e3;
16
17 %% STEEPEST DECENT
18 [fig,k] = ...
    steepestDescent(maxIterations,epsilon,initialConditions(3,:),f,'stable',0.5);
19 exportgraphics(gcf,'steepestDescent_(1,1)_stable.pdf','ContentType','Vector')
20 [fig,k] = ...
    steepestDescent(maxIterations,epsilon,initialConditions(3,:),f,'min',0.5);
21 exportgraphics(gcf,'steepestDescent_(1,1)_min.pdf','ContentType','Vector')
22 [fig,k] = ...
    steepestDescent(maxIterations,epsilon,initialConditions(3,:),f,'armijo',0.5);
23 exportgraphics(gcf,'steepestDescent_(1,1)_armijo.pdf','ContentType','Vector')
24
25 %% NEWTON
26 [fig,k] = newton(maxIterations,epsilon,initialConditions(3,:),f,'stable',0.1);
27 exportgraphics(gcf,'newton_(1,1)_stable.pdf','ContentType','Vector')
28 [fig,k] = newton(maxIterations,epsilon,initialConditions(3,:),f,'min',0.1);
29 exportgraphics(gcf,'newton_(1,1)_min.pdf','ContentType','Vector')
30 [fig,k] = newton(maxIterations,epsilon,initialConditions(3,:),f,'armijo',0.1);
31 exportgraphics(gcf,'newton_(1,1)_armijo.pdf','ContentType','Vector')
32
33 %% Levenberg Marquardt
34 [fig,k] = levenMarq(maxIterations,epsilon,initialConditions(3,:),f,'stable',0.1);
35 exportgraphics(gcf,'leven_(1,1)_stable.pdf','ContentType','Vector')
36 [fig,k] = levenMarq(maxIterations,epsilon,initialConditions(3,:),f,'min',0.1);
37 exportgraphics(gcf,'leven_(1,1)_min.pdf','ContentType','Vector')
38 [fig,k] = levenMarq(maxIterations,epsilon,initialConditions(3,:),f,'armijo',0.1);
39 exportgraphics(gcf,'leven_(1,1)_armijo.pdf','ContentType','Vector')
40
41
42 % FUNCTIONS
43 function [fig,k] = ...
    steepestDescent(maxIterations,epsilon,x0,f,gammaFlag,gammaStable)
44     fig = figure; fcontour(f,'Fill','on'); colorbar
45     hold on
46
47     x = sym('x');
48     y = sym('y');
49     fx = matlabFunction(diff(sym(f),x));
50     fy = matlabFunction(diff(sym(f),y));
51
52     xk = x0(1);
53     yk = x0(2);
```

```

54     plot(xk,yk, 'h', 'color', 'red', 'MarkerSize', 8)
55     %text(xk,yk,"x0", 'Color', 'red', 'FontSize', 14)
56     xkNext = @(gamma,xk,yk) xk - gamma * fx(xk,yk);
57     ykNext = @(gamma,xk,yk) yk - gamma * fy(xk,yk);
58     k = 1;
59     while norm(fx(xk,yk),fy(xk,yk)) > epsilon && k<maxIterations
60         if gammaFlag == "stable"
61             gamma = gammaStable;
62         elseif gammaFlag == "min"
63             fMin = @(gamma) f(xkNext(gamma,xk,yk),ykNext(gamma,xk,yk));
64             gamma = fminbnd(fMin,0,5);
65         elseif gammaFlag == "armijo"
66             initStep = 2;
67             [gamma,mk] = ...
                armijo(initStep, f, fx, fy, @(x,y)-fx(x,y), @(x,y)-fy(x,y), xk,yk, xkNext, ykNext)
68         else
69             disp("Invalid gamma flag")
70             exit
71         end
72         %fprintf("%0.2f,%0.2f,gamma=%d\n",xk,yk,gamma)
73         xk = xkNext(gamma,xk,yk);
74         yk = ykNext(gamma,xk,yk);
75         k = k + 1;
76         plot(xk,yk, '—o')
77     end
78     fprintf("Iters: %d/%d\n",k,maxIterations)
79 end
80
81 function [fig,k] = newton(maxIterations,epsilon,x0,f,gammaFlag,gammaStable)
82     fig = figure; fcontour(f, 'Fill', 'on'); colorbar
83     hold on
84
85     x = sym('x');
86     y = sym('y');
87     fx = diff(sym(f),x);
88     fy = diff(sym(f),y);
89     fxx = diff(sym(fx),x);
90     fxy = diff(sym(fx),y);
91     fyx = diff(sym(fy),x);
92     fyy = diff(sym(fy),y);
93     J = [fxx fxy; fyx fyy];
94     dk = -inv(J) * [fx; fy];
95     dkX = matlabFunction(dk(1));
96     dkY = matlabFunction(dk(2));
97     fx = matlabFunction(sym(fx));
98     fy = matlabFunction(sym(fy));
99     xkNext = @(gamma,xk,yk) xk + gamma * dkX(xk,yk);
100    ykNext = @(gamma,xk,yk) yk + gamma * dkY(xk,yk);
101
102
103    xk = x0(1);
104    yk = x0(2);
105    plot(xk,yk, 'h', 'color', 'red', 'MarkerSize', 8)
106    %text(xk,yk,"x0", 'Color', 'red', 'FontSize', 14)
107    k = 1;
108    while norm(fx(xk,yk),fy(xk,yk)) > epsilon && k<maxIterations
109        if gammaFlag == "stable"
110            gamma = gammaStable;
111        elseif gammaFlag == "min"
112            fMin = @(gamma) f(xkNext(gamma,xk,yk),ykNext(gamma,xk,yk));
113            gamma = fminbnd(fMin,0,5);
114        elseif gammaFlag == "armijo"
115            initStep = 2;

```

```

116         [gamma,mk] = armijo( initStep , f , fx , fy , dkX,dkY , xk , yk , xkNext , ykNext );
117     else
118         disp("Invalid gamma flag ")
119         exit
120     end
121     %fprintf("%0.2f,%0.2f ,gamma=%d\n",xk,yk,gamma)
122     xk = xkNext(gamma,xk,yk);
123     yk = ykNext(gamma,xk,yk);
124     k = k + 1;
125     plot(xk,yk, '—o')
126 end
127 fprintf("Iters: %d/%d\n",k,maxIterations)
128 end
129
130 function [fig,k] = levenMarq(maxIterations,epsilon,x0,f,gammaFlag,gammaStable)
131     fig = figure; fcontour(f, 'Fill','on'); colorbar
132     hold on
133
134     x = sym('x');
135     y = sym('y');
136     fx = diff(sym(f),x);
137     fy = diff(sym(f),y);
138     fxx = diff(sym(fx),x);
139     fxy = diff(sym(fx),y);
140     fyx = diff(sym(fy),x);
141     fyy = diff(sym(fy),y);
142     J = [fxx fxy ; fyx fyy];
143     J = matlabFunction(sym(J));
144     fx = matlabFunction(diff(sym(f),x));
145     fy = matlabFunction(diff(sym(f),y));
146
147     xk = x0(1);
148     yk = x0(2);
149     plot(xk,yk, 'h', 'color','red','MarkerSize',8)
150     %text(xk,yk,"x0",'Color','red','FontSize',14)
151     k = 1;
152     while norm(fx(xk,yk),fy(xk,yk)) > epsilon && k<maxIterations
153         hessian = J(xk,yk);
154         eigen = eig(hessian);
155         m = abs(max(eigen))+0.1;
156         d = - inv(hessian+m*eye(2)) * [fx(xk,yk) ; fy(xk,yk)];
157
158         if gammaFlag == "stable"
159             gamma = gammaStable;
160         elseif gammaFlag == "min"
161             fMin = @(gamma) f(xk+gamma*d(1),yk+gamma*d(2));
162             gamma = fminbnd(fMin,0,5);
163         elseif gammaFlag == "armijo"
164             initStep = 2;
165             [gamma,mk] = armijo( initStep , f , fx , fy , J , xk , yk );
166         else
167             disp("Invalid gamma flag ")
168             exit
169         end
170
171         %fprintf("%0.2f,%0.2f ,gamma=%d\n",xk,yk,gamma)
172         xk = xk + gamma * d(1);
173         yk = yk + gamma * d(2);
174         k = k + 1;
175         plot(xk,yk, '—o')
176     end
177     fprintf("Iters: %d/%d\n",k,maxIterations)
178

```



```

179 % TODO: Refactoring. Consistency among armijo functions
180 function [gamma,mk] = armijo(s,f,fx,fy,J,xk,yk)
181     alpha = 1e-1;
182     beta = 0.1;
183     mk = 0;
184     gamma = s*(beta^mk);
185
186     hessian = J(xk,yk);
187     eigen = eig(hessian);
188     m = abs(max(eigen))+0.1;
189     d = - inv(hessian+m*eye(2)) * [fx(xk,yk) ; fy(xk,yk)];
190     xkN = xk + gamma * d(1);
191     ykN = yk + gamma * d(2);
192
193     D = fx(xk,yk)*d(1) + fy(xk,yk)*d(2);
194     while (f(xk,yk) - f(xkN,ykN)) < -alpha*gamma*D
195         mk = mk + 1;
196         gamma = s*(beta^mk);
197
198         hessian = J(xk,yk);
199         eigen = eig(hessian);
200         m = abs(max(eigen))+0.1;
201         d = - inv(hessian+m*eye(2)) * [fx(xk,yk) ; fy(xk,yk)];
202         xkN = xk + gamma * d(1);
203         ykN = yk + gamma * d(2);
204     end
205 end
206 end
207
208 function [gamma,mk] = armijo(s,f,fx,fy,dkX,dkY,xk,yk,xkNext,ykNext)
209     alpha = 1e-1;
210     beta = 0.1;
211     mk = 0;
212     gamma = s*(beta^mk);
213     xkN = xkNext(gamma,xk,yk);
214     ykN = ykNext(gamma,xk,yk);
215     D = fx(xk,yk)*dkX(xk,yk) + fy(xk,yk)*dkY(xk,yk);
216     while (f(xk,yk) - f(xkN,ykN)) < -alpha*gamma*D
217         mk = mk + 1;
218         gamma = s*(beta^mk);
219         xkN = xkNext(gamma,xk,yk);
220         ykN = ykNext(gamma,xk,yk);
221     end
222 end

```