

```

1 # -----
2 # Author: Philip Coyle
3 # Date Created: 06/10/2020
4 # ps2.jl
5 # -----
6
7 using Optim, Plots, Interpolations, Statistics
8 dir = "/Users/philipcoyle/Documents/School/University_of_Wisconsin/SecondYear/Summer_2020/CodingBootcamp/ProblemSets/PS2/"
9
10 ## Question 1
11 function Himmelblau(in::Array{Float64,1})
12     x = in[1]
13     y = in[2]
14
15     out = (x^2 + y - 11)^2 + (x + y^2 - 7)^2
16     return out
17 end
18
19 function g(G, in::Array{Float64,1})
20     x = in[1]
21     y = in[2]
22
23     G[1] = (4*x)*(x^2 + y - 11) + 2*(x + y^2 - 7)
24     G[2] = 2*(x^2 + y - 11) + (4*y)*(x + y^2 - 7)
25 end
26 function h(H, in::Array{Float64,1})
27     x = in[1]
28     y = in[2]
29
30     H[1,1] = 12*x^2 + 4*y - 44 + 2
31     H[1,2] = 4*x + 4*y
32     H[2,1] = 4*x + 4*y
33     H[2,2] = 2 + 4x + 12y - 28
34 end
35
36 # Part A
37 n_grid = 101
38 x_grid = range(-4, 4, length = n_grid)
39 y_grid = x_grid
40
41 # Preallocate space for function
42 z_grid = zeros(n_grid, n_grid)
43 for i in 1:n_grid
44     for j = 1:n_grid
45         vec_in = [x_grid[i], y_grid[j]]
46         z_grid[i,j] = Himmelblau(vec_in)
47     end
48 end
49
50 # Part B (Trying different guesses: argmin after convergence)
51 n_grid = 101
52
53 for i in 1:n_grid
54     for j = 1:n_grid
55         guess = [x_grid[i], y_grid[j]]
56         opt_newton = optimize(Himmelblau, g, h, guess)
57         if opt_newton.minimizer[1] < 0
58             minimizer[i,j] = 3
59             if opt_newton.minimizer[2] < 0
60                 minimizer[i,j] = 4
61             end
62         else
63             minimizer[i,j] = 2
64             if opt_newton.minimizer[2] > 0
65                 minimizer[i,j] = 1
66             end
67         end
68     end
69 end
70
71 # Part C (Trying different guesses: Number of iterations)
72 num_iter_newton = zeros(n_grid, n_grid)
73 num_iter_nm = zeros(n_grid, n_grid)
74 for i in 1:n_grid
75     for j = 1:n_grid
76         guess = [x_grid[i], y_grid[j]]
77
78         opt_newton = optimize(Himmelblau, g, h, guess)
79         opt_nm = optimize(Himmelblau, guess, NelderMead())
80
81         num_iter_newton[i,j] = opt_newton.iterations
82         num_iter_nm[i,j] = opt_nm.iterations
83     end
84 end
85
86 # Plot
87 p1 = surface(x_grid, y_grid, z_grid, camera = (50, 50), title="Himmelblau Function");
88 p2 = contourf(x_grid, y_grid, log.(z_grid), title="Contour Plot");

```

```

89 p3 = contourf(x_grid, y_grid, minimizer, title="Convergence Destination \n 1 ≡ [3, 2]; 2 ≡ [3.6, -1.8]; 3 ≡ [-2.8, 3.1]; 4 ≡ [-3.8, -3.3]");
90 p4 = contourf(x_grid, y_grid, log.(num_iter_newton), title="(Log) Numer of ITERS to Converge \n Newton's Method");
91 p5 = contourf(x_grid, y_grid, log.(num_iter_nm), title="(Log) Numer of ITERS to Converge \n Nelder Mead");
92
93 l = @layout [a{0.5w} b{0.5w}; c{0.33h}; d{0.5w} e{0.5w}]
94 plot(p1, p2, p3, p4, p5, layout = l, size = (800,900))
95 savefig(dir * "Q1.pdf")
96
97 ## Question 2
98 function Ackley(in::Array{Float64,1})
99     x = in[1]
100     y = in[2]
101
102     out = -20*exp(-0.2*(0.5*(x^2 + y^2))^(0.5)) - exp(0.5*(cos(2*pi*x) + cos(2*pi*y))) + exp(1) + 20
103     return out
104 end
105
106 # Part A
107 n_grid = 101
108 x_grid = range(-4, 4, length = n_grid)
109 y_grid = x_grid
110
111 # Preallocate space for functuon
112 z_grid = zeros(n_grid, n_grid)
113 for i in 1:n_grid
114     for j = 1:n_grid
115         vec_in = [x_grid[i], y_grid[j]]
116         z_grid[i,j] = Ackley(vec_in)
117     end
118 end
119
120 # Plot
121 p1 = surface(x_grid, y_grid, z_grid, camera = (50, 50), title="Ackley Function");
122 p2 = contourf(x_grid, y_grid, z_grid, title="Contour Plot");
123 plot(p1, p2, layout=(1,2),size = (600,250))
124 savefig(dir * "Q2_a.pdf")
125
126 # Part B (trying different guesses)
127 minimizer = zeros(n_grid, n_grid,2)
128 num_iter = zeros(n_grid, n_grid,2)
129 for i = 1:n_grid
130     for j = 1:n_grid
131         guess = [x_grid[i], y_grid[j]]
132         opt_nm = optimize(Ackley, guess, NelderMead())
133         minimizer[i,j,1] = opt_nm.minimum
134         num_iter[i,j,1] = opt_nm.iterations
135
136         opt_lbfgs = optimize(Ackley, guess, LBFGS())
137         minimizer[i,j,2] = opt_lbfgs.minimum
138         num_iter[i,j,2] = opt_lbfgs.iterations
139     end
140 end
141
142 # Plot
143 p11 = contourf(x_grid, y_grid, log.(minimizer[:, :,1]), title="Nelder Mead \n (Log) Convergence Destination ");
144 p12 = contourf(x_grid, y_grid, log.(minimizer[:, :,2]), title="LBFGS \n (Log) Convergence Destination");
145 p21 = contourf(x_grid, y_grid, log.(num_iter[:, :,1]), title="(Log) Numer of ITERS to Converge");
146 p22 = contourf(x_grid, y_grid, log.(num_iter[:, :,2]), title="(Log) Numer of ITERS to Converge");
147 plot(p11, p12, p21, p22, layout=(2,2),size = (1200,950))
148 savefig(dir * "Q2_b.pdf")
149
150 ## Question 3
151 function Rastrigin(x::Array{Float64,1}; A = 10)
152     sum = 0
153     for i = 1:length(x)
154         sum = sum + x[i]^2 - A*cos(2*pi*x[i])
155     end
156
157     n = length(x)
158     out = A*n + sum
159
160     return out
161 end
162
163 # Part A
164 n_grid = 101
165 x_grid = range(-5.12, 5.12, length = n_grid)
166
167 # Preallocate space for functuon
168 z_grid_ld = zeros(n_grid,1)
169 for i in 1:n_grid
170     z_grid_ld[i] = Rastrigin([x_grid[i]])
171 end
172
173 plot(x_grid, z_grid_ld, title = "Rastrigin Function")
174 savefig(dir * "Q3_a.pdf")
175
176 # Part B

```

```

177 n_grid = 101
178 x_grid = range(-5.12, 5.12, length = n_grid)
179
180 # Preallocate space for function
181 z_grid_2d = zeros(n_grid,n_grid)
182 for i in 1:n_grid
183     for j = 1:n_grid
184         x_in = [x_grid[i], x_grid[j]]
185         z_grid_2d[i,j] = Rastrigin(x_in)
186     end
187 end
188
189 p1 = surface(x_grid,x_grid,z_grid_2d, camera = (50, 50), title="Rastrigin Function");
190 p2 = contourf(x_grid,x_grid,log.(z_grid_2d .+ 1), title="Contour Plot");
191 plot(p1, p2, layout=(1,2),size = (600,250))
192 savefig(dir * "Q3_b.pdf")
193
194 # Part C (trying different guesses)
195 minimizer = zeros(n_grid, n_grid,2)
196 num_iter = zeros(n_grid, n_grid,2)
197 for i = 1:n_grid
198     for j = 1:n_grid
199         guess = [x_grid[i], x_grid[j]]
200         opt_nm = optimize(Rastrigin, guess, NelderMead())
201         minimizer[i,j,1] = opt_nm.minimum
202         num_iter[i,j,1] = opt_nm.iterations
203
204         opt_lbfgs = optimize(Rastrigin, guess, LBFGS())
205         minimizer[i,j,2] = opt_lbfgs.minimum
206         num_iter[i,j,2] = opt_lbfgs.iterations
207     end
208 end
209
210 # Plot
211 p11 = contourf(x_grid, x_grid, log.(minimizer[:, :, 1]), title="Nelder Mead \n (Log) Convergence Destination ");
212 p12 = contourf(x_grid, x_grid, log.(minimizer[:, :, 2]), title="LBFGS \n (Log) Convergence Destination");
213 p21 = contourf(x_grid, x_grid, log.(num_iter[:, :, 1]), title="(Log) Number of Iters to Converge");
214 p22 = contourf(x_grid, x_grid, log.(num_iter[:, :, 2]), title="(Log) Number of Iters to Converge");
215 plot(p11, p12, p21, p22, layout=(2,2),size = (1200,950))
216 savefig(dir * "Q3_c.pdf")
217
218 ## Question 4
219 function lin_interp(f::Function, a::Float64, b::Float64, n::Int64, val::Float64)
220     if val < a || val > b
221         msg = "x must be between a and b"
222         throw(msg)
223     end
224
225     if n < 1
226         msg = "n must be a positive integer"
227         throw(msg)
228     end
229
230     grid = range(a, b, length = n)
231     f_grid = f.(grid)
232     f_interp = interpolate(f_grid,BSpline{Linear}())
233
234     # Get Index
235     inx_upper = findfirst(x->x>val, grid)
236     inx_lower = inx_upper - 1
237     val_upper, val_lower = grid[inx_upper], grid[inx_lower]
238     inx = inx_lower + (val - val_lower) / (val_upper - val_lower)
239
240     out = f_interp(inx)
241     return out
242 end
243
244 func(x) = log(x)
245 a = 1.
246 b = 10.
247 x = 1.25
248 n = 5
249
250 lin_interp(func, a, b, n, x)
251
252 ## Question 5
253 function get_inx(val::Float64, grid::Array{Float64,1})
254     if val >= grid[length(grid)]
255         inx = length(grid)
256     elseif val <= grid[1]
257         inx = 1
258     else
259         inx_upper = findfirst(x->x>val, grid)
260         inx_lower = inx_upper - 1
261         val_upper, val_lower = grid[inx_upper], grid[inx_lower]
262         inx = inx_lower + (val - val_lower) / (val_upper - val_lower)
263     end
264 end

```

```

265
266 function approx_error(f::Function, grid::Array{Float64,1}, grid_fine::Array{Float64,1})
267     f_grid = f.(grid)
268     f_interp = interpolate(f_grid,BSpline{Linear}())
269     f_apx = zeros(length(grid_fine),1)
270
271     for (i, grid_i) = enumerate(grid_fine)
272         grid_inx = get_inx(grid_i, grid)
273         f_apx[i] = f_interp(grid_inx)
274     end
275
276     f_out = abs.(f_apx .- f.(grid_fine))
277     f_out_avg = mean(f_out)
278
279     return f_out, f_out_avg
280 end
281
282
283
284 function opt_grid(grid::Array{Float64,1})
285     grid[1] = 0
286     grid[end] = 100
287
288     f(x) = log(x+1)
289     grid_fine = collect(0:0.1:100)
290
291     ~, out = approx_error(f, grid, grid_fine)
292
293     return out
294 end
295
296
297 x_grid = collect(range(0,100, length = 11))
298 x_grid_fine = collect(0:0.1:100)
299 f(x) = log(x+1)
300
301 # Part A
302 error, avg_error = approx_error(f, x_grid, x_grid_fine)
303 plot(x_grid, f.(x_grid), label = "Interpolated Function", color=:blue,lw = 2)
304 plot!(x_grid_fine, f.(x_grid_fine), label = "True Function", color=:black,lw = 2)
305 plot!(x_grid_fine, error, label = "Approximation Error", color=:red,lw = 2)
306 savefig(dir * "Q5_LinearGrid.pdf")
307
308
309 # Part C
310 opt = optimize(opt_grid, x_grid)
311 error, avg_error = approx_error(f, opt.minimizer, x_grid_fine)
312 plot(opt.minimizer, f.(opt.minimizer), label = "Interpolated Function", color=:blue,lw = 2)
313 plot!(x_grid_fine, f.(x_grid_fine), label = "True Function", color=:black,lw = 2)
314 plot!(x_grid_fine, error, label = "Approximation Error", color=:red,lw = 2)
315 savefig(dir * "Q5_OptimizedGrid.pdf")

```