HOMEWORK 7
Elena Gómez
February 28, 2023

PROBLEM 0: Homework checklist
I worked alone.

PROBLEM 1:
(Adapted from Nocedal and Wright 3.1, typos are my own) Backtracking line search is a simple method that attempts to balance sufficient decrease with a long step length by testing a sequence of points:
$\alpha = 1, 1/2, 1/4, \ldots$
and accepting the first point that satisfies sufficient decrease (which is the condition that $L(\alpha) \leq c_1 \alpha L'(0)$ for the line search function $L(\alpha)$ we discussed in class).

1. Implement a backtracking line search routine to select the step length and add this to our simple gradient descent code from the lectures. (Note, you are free to implement gradient descent using your own code too, adapting our code is only suggested for convenience.)

The following function *gradient_descendent_1* is the grandient descendent code with the changes neccesary for the backtracking line search. In this case, the descendent direction is $p = -g$.

In [185]:

```
using Printf, LinearAlgebra

function gradient_descent_1(fgh,x0;
    maxiter=10000,tol=1.0e-8,quiet=false,histx=[],hista=[],gamma=0.01)

    x = copy(x0)
    n = length(x)

    hist = zeros(2,maxiter)
    savehistx = eltype(histx) == Vector{Float64} ? true : false
    savehista = eltype(hista) == Float64 ? true : false
    alpha_list=[]

    f = Inf
    normg = Inf
    lastiter = 0
    g = Vector{Float64}()
    h = Matrix{Float64}(undef, 0, 0)


    if !quiet
        @printf("  %6s  %9s %9s %9s\n", "iter",
            "val", "normg", "fdiff");
    end

    for iter=1:maxiter
        if savehistx
            push!(histx, x)
        end

        if iter>1
            p=-g; #direction

            alpha=1.0;
            while(fgh(x+alpha*p)[1] > f + gamma*alpha*p'*g)
                alpha/=2;
            end

            x=x+alpha*p;

            push!(alpha_list,alpha);

            if savehista
                push!(hista,alpha);
            end
        end

        flast = f
        f,g,h = fgh(x)
        normg = norm(g,Inf)

        fdiff = flast - f

        if !quiet
            @printf("  %6i  %9.2e %9.2e %9.2e\n",
                iter, f, normg, fdiff)
        end

        hist[:,iter] = [f; normg]
```

```
        lastiter = iter

        if normg <= tol
            break
        end
        if !isfinite(normg)
            break
        end
    end

    if lastiter < maxiter
        hist = hist[:,1:lastiter]
    end

    if normg > tol
        @warn "Did not converge"
    end

    return x,f,g,hist,alpha_list,histx
end
```

Out[185]:

```
gradient_descent_1 (generic function with 1 method)
```

The following code are auxiliar functions that we also use, as well as, the code we will use to plot the contour plots.

In [186]:

```julia
using Optim, OptimTestProblems
OptimTestProblems.UnconstrainedProblems.examples["Rosenbrock"]




using Plots
ezcontour(x, y, f) = begin
    X = repeat(x', length(y), 1)
    Y = repeat(y, 1, length(x))
    # Evaluate each f(x, y)
    Z = map((x,y) -> f([x,y]), X, Y)
    plot(x, y, Z, st=:contour)
end





# These codes turn f and g into one function ...
function opt_combine(x, f, g!,h!)
    g = Array{Float64,1}(undef,length(x))
    h= Matrix{Float64}(undef, length(x), length(x));
    g!(g,x)
    h!(h,x)
    return (f(x), g,h)
end

function opt_problem(p::OptimTestProblems.UnconstrainedProblems.OptimizationProblem)
    return x -> opt_combine(x, p.f, p.g!,p.h!)
end

# this just makes it easy to use
opt_problem(s::AbstractString) = opt_problem(
    OptimTestProblems.UnconstrainedProblems.examples[s])

```

Out[186]:

```
opt_problem (generic function with 2 methods)
```

2. Prepare a plot of the step lengths (values of α_k that were accepted) when we run gradient descent with this line search on the Rosenbrock function staring from the point $(1.2, 1.2)$ and also the point $(-1.2, 1)$.

Firstly, we compute the gradient descendent starting from the point $(1.2, 1.2)$

In [187]:

```
opt_fun = MultivariateProblems.UnconstrainedProblems.examples["Rosenbrock"];
fgh = opt_problem(opt_fun);
histx = Vector{Vector{Float64}}();
hista = Vector{Float64}();


x,fx,gx,hist,alpha_list,histx1 = gradient_descent_1(fgh, [1.2,1.2];
maxiter=16000, tol=1.0e-8,quiet=false,
histx=histx, hista=hista, gamma=0.01);

```

```
    28   1.19e-02  2.05e-01  3.12e-06
    29   1.18e-02  7.47e-02  3.68e-05
    30   1.18e-02  2.77e-01  1.33e-05
    31   1.18e-02  8.56e-02  5.64e-05
    32   1.18e-02  1.61e-01  1.62e-05
    33   1.18e-02  2.70e-01  3.90e-06
    34   1.17e-02  9.76e-02  3.47e-05
    35   1.17e-02  3.37e-01  1.64e-05
    36   1.16e-02  1.02e-01  5.26e-05
    37   1.16e-02  2.06e-01  1.72e-05
    38   1.16e-02  1.86e-01  4.77e-06
    39   1.16e-02  7.09e-02  3.23e-05
    40   1.16e-02  2.48e-01  2.06e-05
    41   1.15e-02  8.02e-02  4.79e-05
    42   1.15e-02  1.52e-01  1.86e-05
    43   1.15e-02  2.44e-01  6.02e-06
    44   1.15e-02  9.48e-02  2.91e-05
    45   1.14e-02  3.01e-01  2.54e-05
    46   1.14e-02  9.85e-02  4.24e-05
    47   1.14e-02  3.71e-01  3.81e-06
```

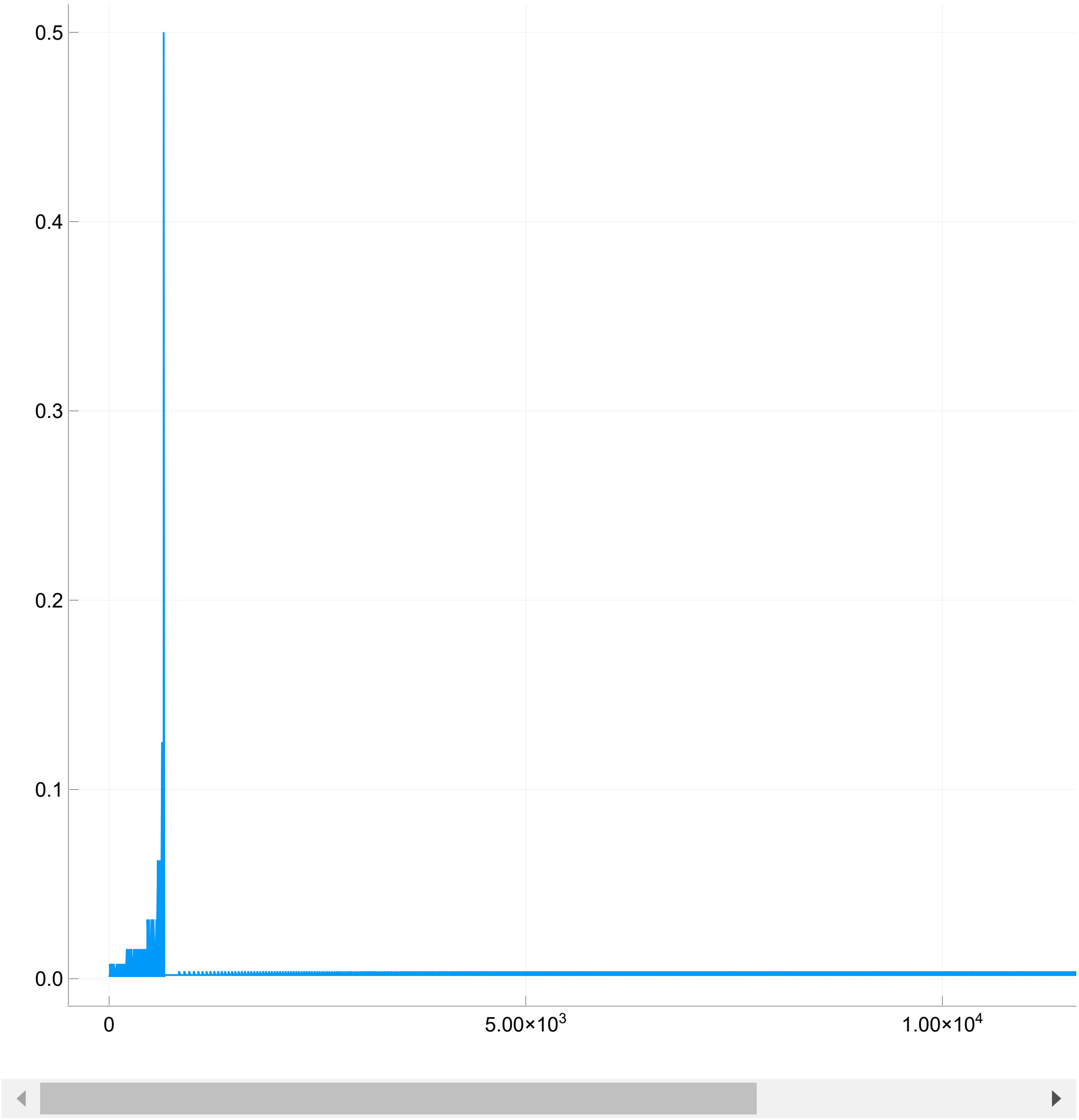
This method does not converge

Now, let's plot the values of α

In [188]:

```
1 plot(alpha_list)
```

Out[188]:



As we can see, this method takes too many iterations. In the plot above, we can see the values of α

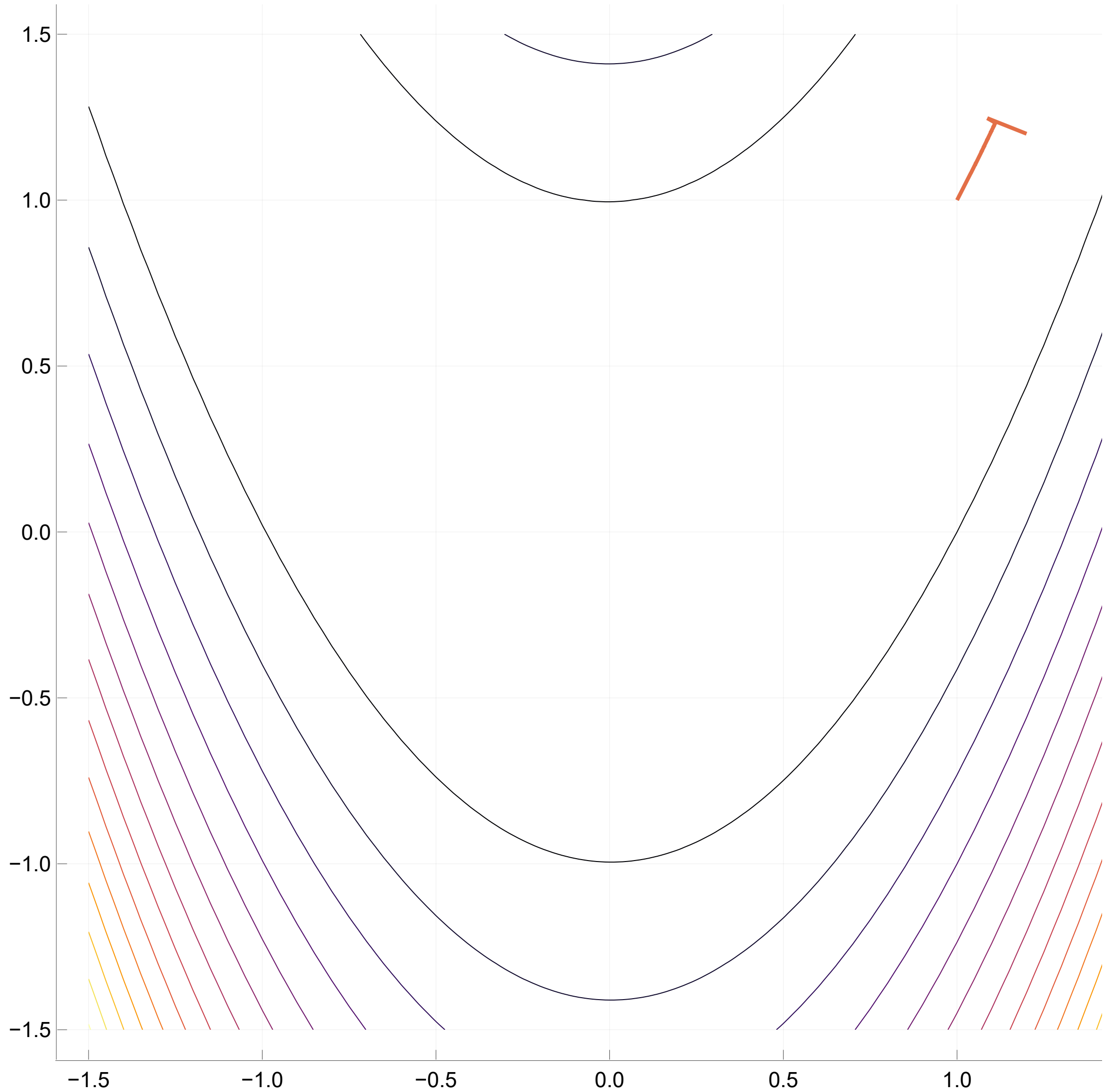
Let's see the contour plot

In [190]:

```
ezcontour(-1.5:0.05:1.5, -1.5:0.05:1.5,
MultivariateProblems.UnconstrainedProblems.examples["Rosenbrock"].f)
#histx1 = histx1[1:3]
plot!(map(first,histx1),map(x->x[2], histx1), linewidth=2)

```

Out[190]:



Now, we compute the function starting from the point $(-1.2, 1)$

In [191]:

```
x,fx,gx,hist,alpha_list,histx1 = gradient_descent_1(fgh, [-1.2,1];
maxiter=16000, tol=1.0e-8,quiet=false,
histx=histx, hista=hista, gamma=0.01);
```

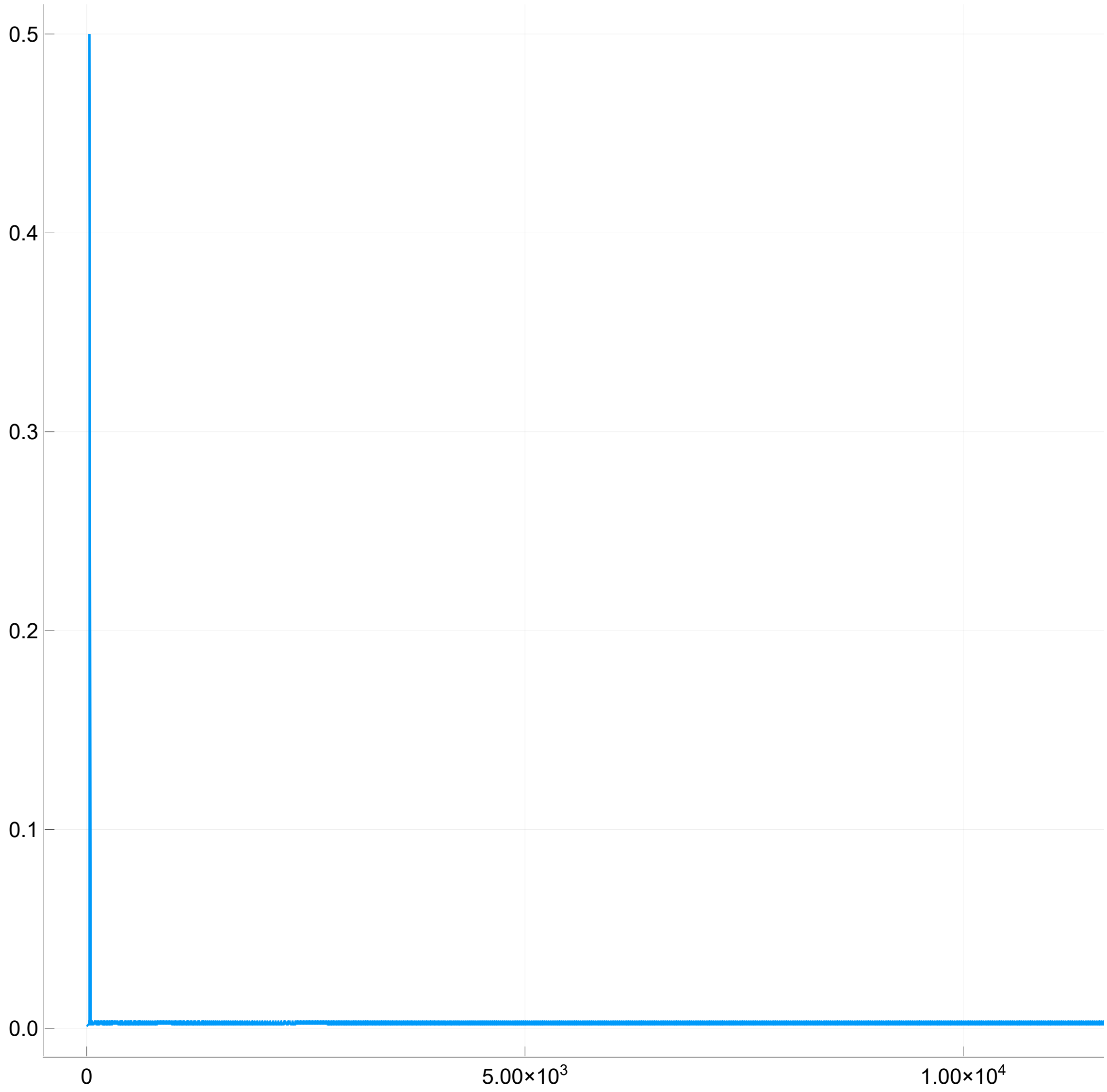
```
103   2.85e-03  8.60e-02  7.43e-06
104   2.85e-03  8.64e-02  6.47e-06
105   2.84e-03  7.04e-02  5.86e-06
106   2.84e-03  6.30e-02  5.47e-06
107   2.83e-03  9.83e-02  3.33e-06
108   2.82e-03  1.05e-01  7.53e-06
109   2.82e-03  7.82e-02  6.52e-06
110   2.81e-03  7.49e-02  5.87e-06
111   2.81e-03  6.55e-02  5.46e-06
112   2.80e-03  1.30e-01  2.77e-06
113   2.80e-03  8.84e-02  7.67e-06
114   2.79e-03  9.06e-02  6.58e-06
115   2.78e-03  7.19e-02  5.90e-06
116   2.78e-03  6.58e-02  5.46e-06
117   2.78e-03  1.02e-01  2.12e-06
118   2.77e-03  1.11e-01  7.84e-06
119   2.76e-03  8.05e-02  6.68e-06
120   2.76e-03  7.90e-02  5.94e-06
121   2.75e-03  6.68e-02  5.48e-06
122   2.75e-03  1.39e-01  1.35e-06
```

Now, let's plot the values of α

In [192]:

```
1 plot(alpha_list)
```

Out[192]:



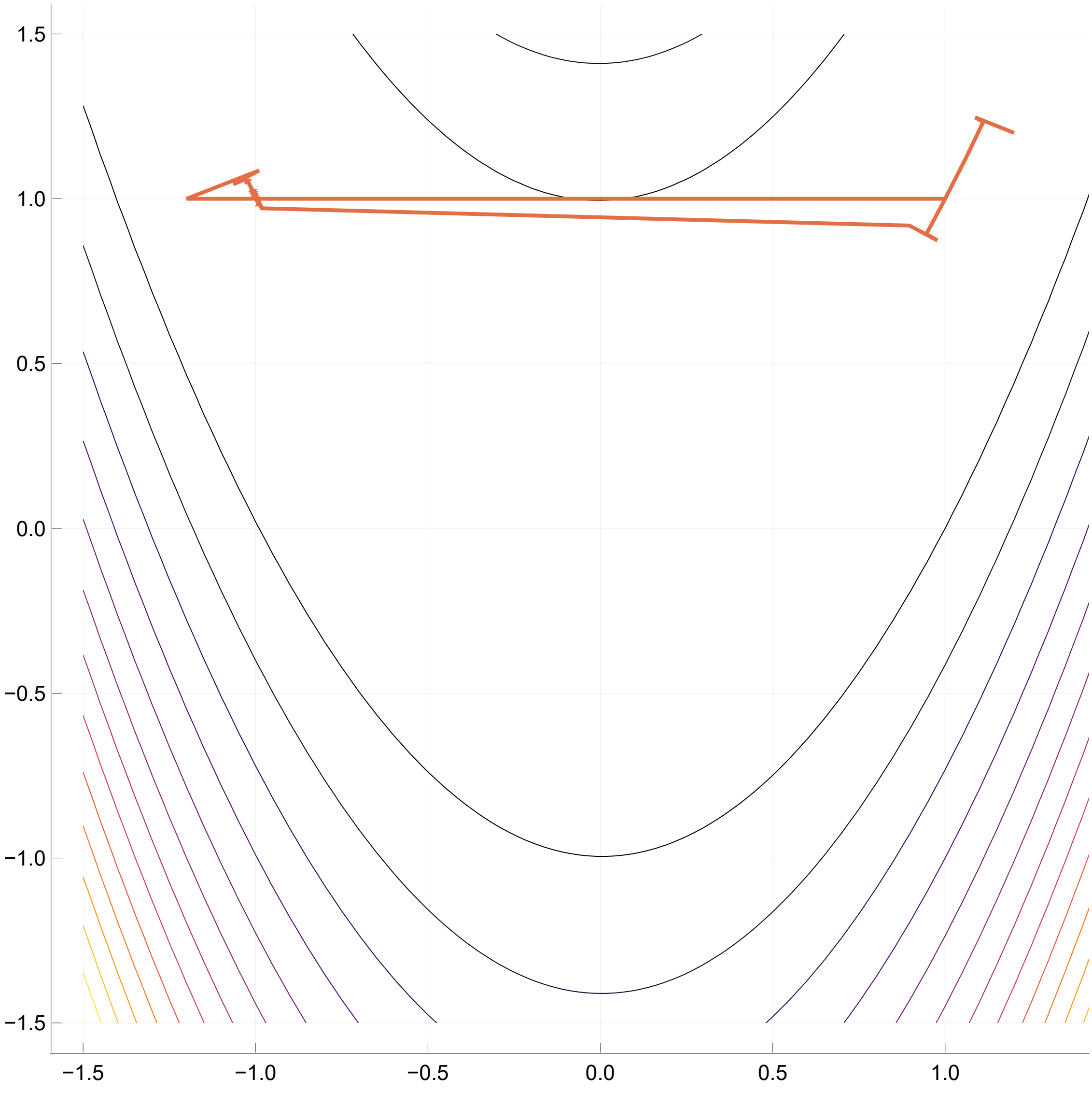
Again, the method takes too many iterations. In the plot above we can see the values of α.

Let's see the contour plot

In [194]:

```
ezcontour(-1.5:0.05:1.5, -1.5:0.05:1.5,
MultivariateProblems.UnconstrainedProblems.examples["Rosenbrock"].f)
#histx1 = histx1[1:3]
plot!(map(first,histx1),map(x->x[2], histx1), linewidth=2)
```

Out[194]:



3. Implement Newton's method using line search as well. (Don't worry about singularity in the Hessian or any of the real issues that come up with Newton – just use the search direction $H(x_k)p = -g_k$.

Let's compute the Newton's method using the linear search. The following function *newton* is the code with the changes neccesary for the backtracking line search. The descendent direction now is $H(x_k)p = -g_k$.

In [195]:

```
using Printf, LinearAlgebra

function newton(fgh,x0;
    maxiter=10000,tol=1.0e-8,quiet=false,histx=[],hista=[],gamma=0.01)

    x = copy(x0)
    n = length(x)

    hist = zeros(2,maxiter)
    savehistx = eltype(histx) == Vector{Float64} ? true : false
    savehista = eltype(hista) == Float64 ? true : false
    alpha_list=[]


    f = Inf
    normg = Inf
    lastiter = 0
    g = Vector{Float64}()
    h = Matrix{Float64}(undef, 0, 0)


    if !quiet
        @printf("  %6s  %9s %9s %9s\n", "iter",
            "val", "normg", "fdiff");
    end

    for iter=1:maxiter
        if savehistx
            push!(histx, x)
        end

        if iter>1
            p=-h\g; #direction

            alpha=1.0;
            while(fgh(x+alpha*p)[1] > f + gamma*alpha*p'*g)
                alpha/=2;
            end

            x=x+alpha*p;

            push!(alpha_list,alpha)

            if savehista
                push!(hista,alpha);
            end
        end

        flast = f
        f,g,h = fgh(x)
        normg = norm(g,Inf)

        fdiff = flast - f

        if !quiet
            @printf("  %6i  %9.2e %9.2e %9.2e\n",
                iter, f, normg, fdiff)
        end

```

```
        hist[:,iter] = [f; normg]
        lastiter = iter

        if normg <= tol
            break
        end
        if !isfinite(normg)
            break
        end
    end

    if lastiter < maxiter
        hist = hist[:,1:lastiter]
    end

    if normg > tol
        @warn "Did not converge"
    end

    return x,f,g,hist,alpha_list,histx
end
```

Out[195]:

```
newton (generic function with 1 method)
```

4. Prepare a plot of the step lengths as in part 2.

Firstly, we compute the function starting from the point $(1.2, 1.2)$

In [196]:

```
x,fx,gx,hist,alpha_list3,histx3= newton(fgh, [1.2,1.2];
maxiter=16000, tol=1.0e-8,quiet=false,
histx=histx, hista=hista, gamma=0.01);
```

```
    iter         val       normg       fdiff
       1    5.80e+00    1.16e+02         Inf
       2    3.84e-02    4.00e-01    5.76e+00
       3    1.88e-02    4.39e+00    1.96e-02
       4    4.29e-03    6.15e-01    1.45e-02
       5    9.03e-04    1.14e+00    3.39e-03
       6    1.85e-05    3.25e-02    8.85e-04
       7    3.40e-08    7.19e-03    1.85e-05
       8    3.23e-14    1.36e-06    3.40e-08
       9    1.09e-25    1.29e-11    3.23e-14
```

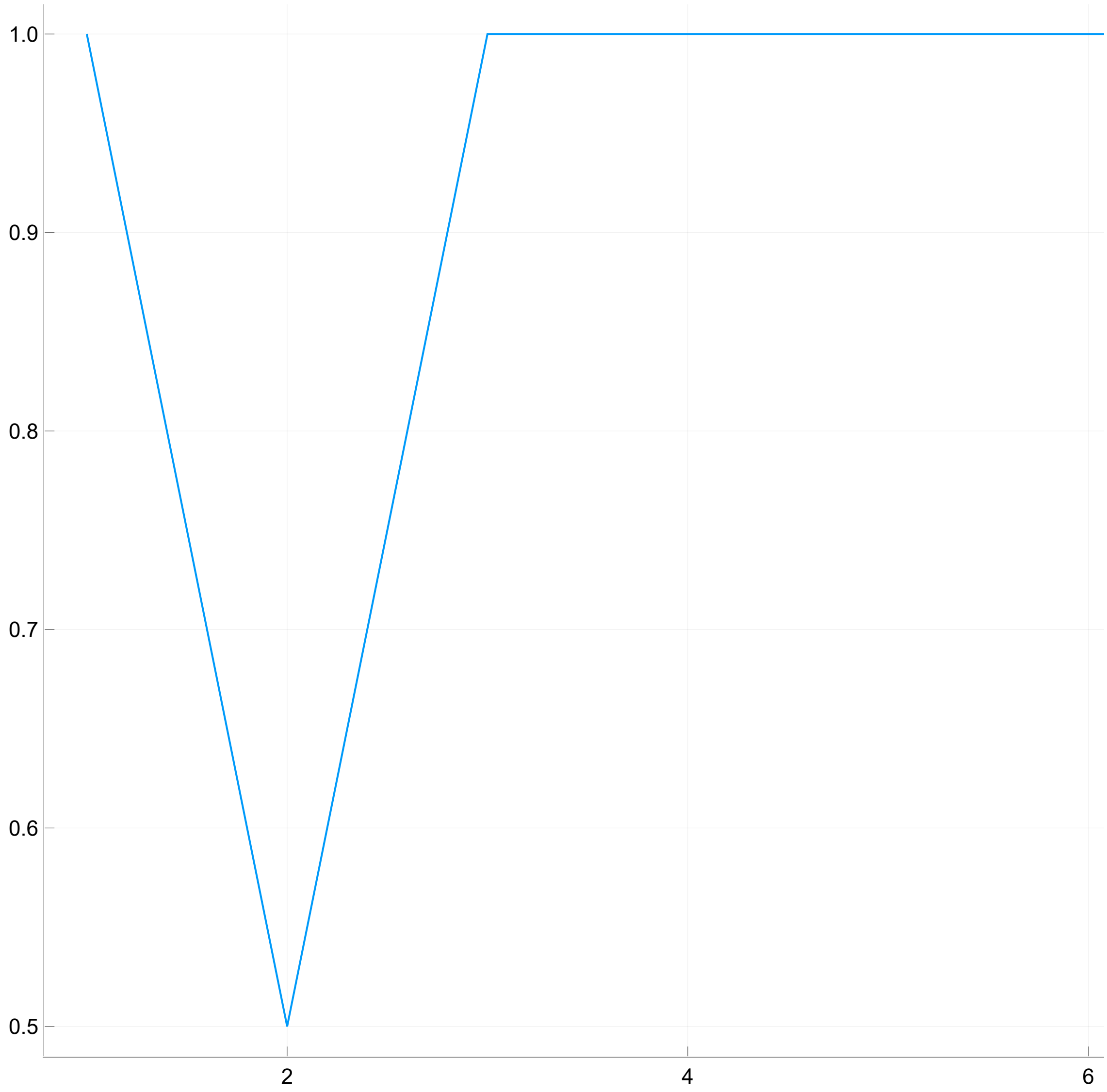
In this case, it converges.

Let's plot the values of α

In [197]:

```
1 plot(alpha_list3)
```

Out[197]:



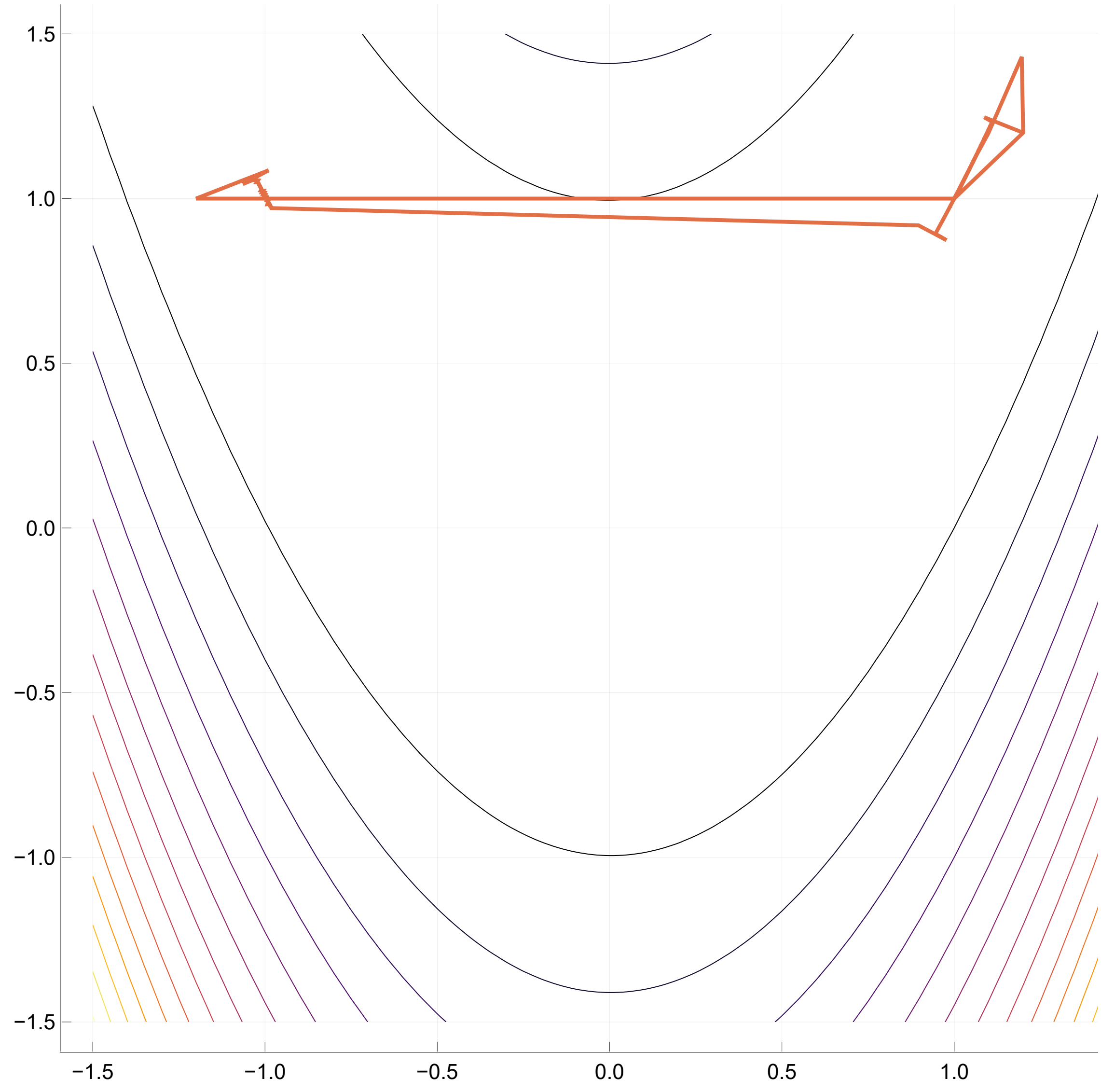
This plot is much more clear that the ones from the gradient descendent.

Let's see the contour plot.

In [198]:

```
ezcontour(-1.5:0.05:1.5, -1.5:0.05:1.5,
MultivariateProblems.UnconstrainedProblems.examples["Rosenbrock"].f)
#histx1 = histx1[1:3]
plot!(map(first,histx3),map(x->x[2], histx3), linewidth=2)
```

Out[198]:



Now, we compute the function starting from the point $(-1.2, 1)$

In [199]:

```
x,fx,gx,hist,alpha_list4,histx4= newton(fgh, [-1.2,1];
maxiter=16000, tol=1.0e-8,quiet=false,
histx=histx, hista=hista, gamma=0.01);
```

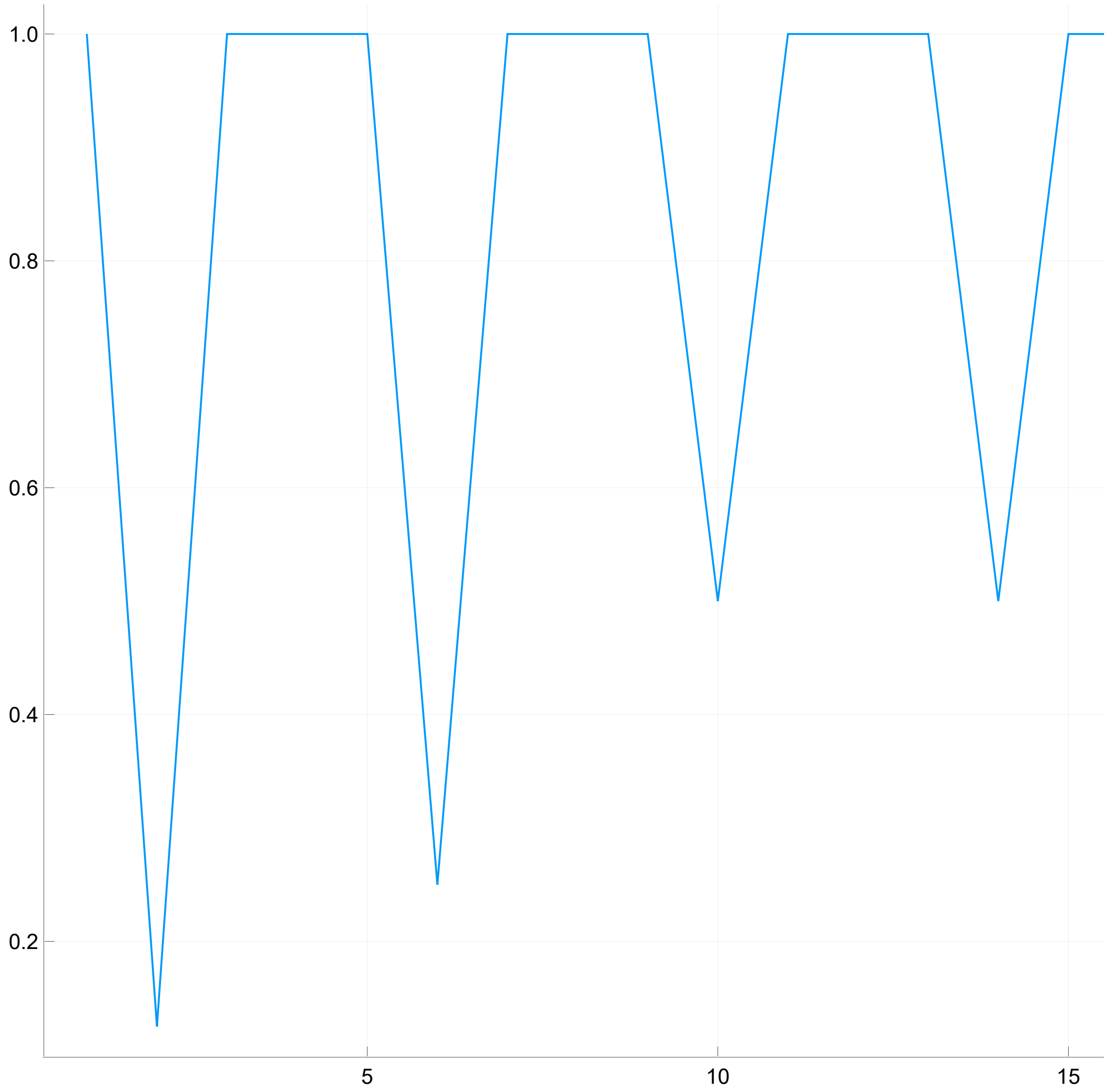
```
iter        val      normg      fdiff
   1   2.42e+01   2.16e+02        Inf
   2   4.73e+00   4.64e+00   1.95e+01
   3   4.09e+00   2.60e+01   6.44e-01
   4   3.23e+00   1.06e+01   8.59e-01
   5   3.21e+00   2.21e+01   1.48e-02
   6   1.94e+00   3.49e+00   1.27e+00
   7   1.60e+00   7.42e+00   3.42e-01
   8   1.18e+00   4.13e+00   4.22e-01
   9   9.22e-01   8.63e+00   2.56e-01
  10   5.97e-01   1.59e+00   3.25e-01
  11   4.53e-01   5.21e+00   1.45e-01
  12   2.81e-01   1.99e+00   1.72e-01
  13   2.11e-01   7.20e+00   6.94e-02
  14   8.90e-02   4.84e-01   1.22e-01
  15   5.15e-02   3.22e+00   3.75e-02
  16   2.00e-02   1.00e+00   3.15e-02
  17   7.17e-03   2.21e+00   1.28e-02
  18   1.07e-03   1.96e-01   6.10e-03
  19   7.78e-05   3.10e-01   9.92e-04
  20   2.82e-07   3.23e-03   7.75e-05
  21   8.52e-12   1.06e-04   2.82e-07
  22   3.74e-21   3.73e-10   8.52e-12
```

In this case, the method also converges.
Let's see the values of α

In [200]:

```
plot(alpha_list4)
```

Out[200]:

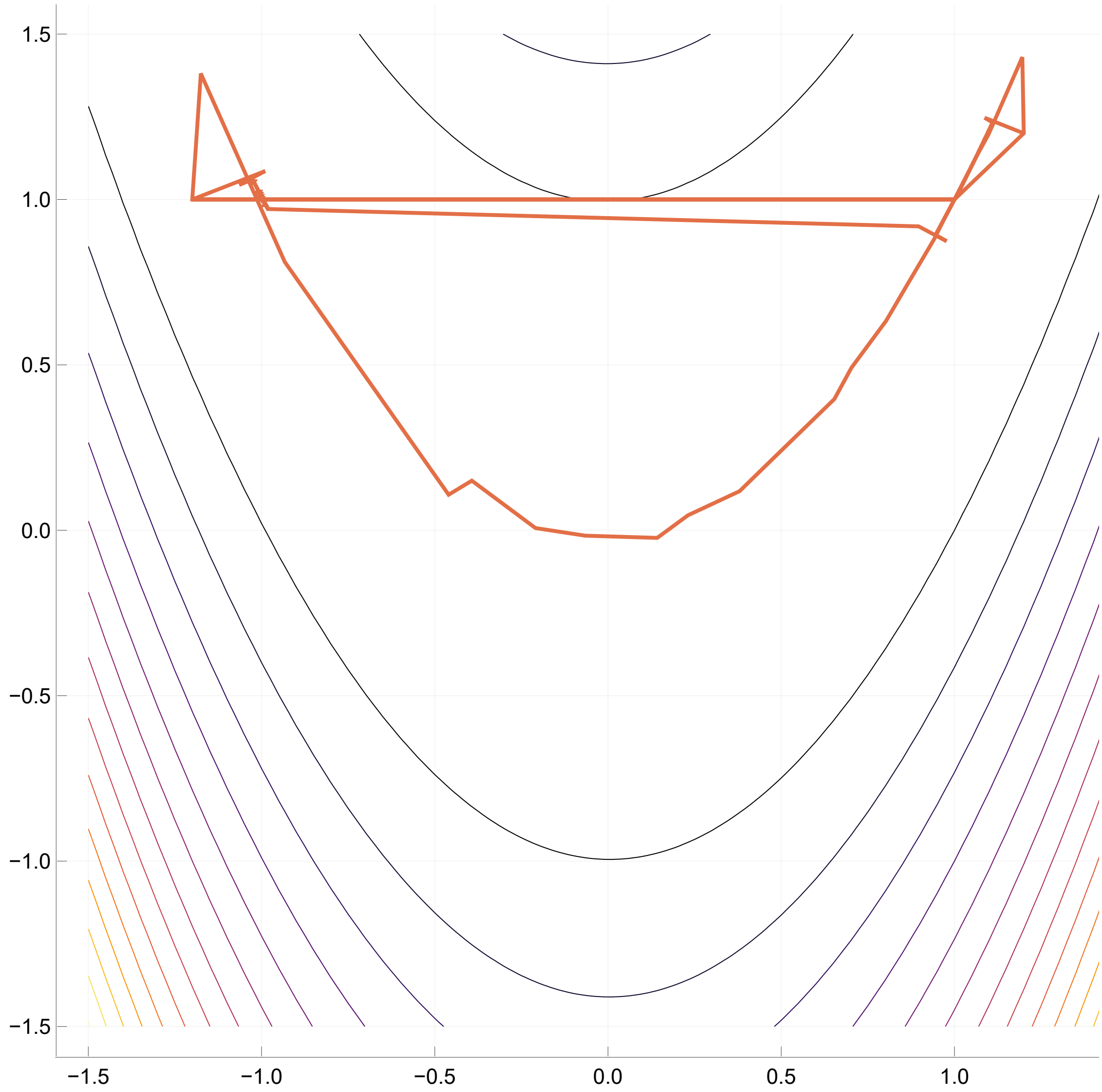


Now,let's see the contour plot

In [201]:

```
ezcontour(-1.5:0.05:1.5, -1.5:0.05:1.5,
MultivariateProblems.UnconstrainedProblems.examples["Rosenbrock"].f)
#histx1 = histx1[1:3]
plot!(map(first,histx4),map(x->x[2], histx4), linewidth=2)
```

Out[201]:



5. Discuss any notable differences or similarities.

The main difference between the Gradient Descendent andthe Newton's Method is the number of iterations. The Newton's method needs less iterations.
Another difference that can be seen from the plots is that the value of α is close to 0 in most of the iterations when using the Gradient descendent. While the value of α is close to 1 in most of the iterations when using the NEwton's Method.

PROBLEM 2

(Exercise 4.2 in Kochenderfer and Wheeler) The first Wolfe condition requires $f(x + \alpha p) \leq f(x) + c_1 \alpha p^T g$. What is the maximum step, length α that satisfies this condition given

that $f(x) = 5 + x_1^2 + x_2^2, x = [-1, 1]^T, p = [1, 0]^T, c_1 = 10^{-4}$

With the information of the statement, we have to find the maximum α that satisfies the first Wolfe condition.

Firstly, let's compute g
$g = \frac{\partial f(x)}{\partial x_1}\imath + \frac{\partial f(x)}{\partial x_2}\jmath = 2x_1\imath + 2x_2\jmath = \begin{bmatrix} 2x_1 & 2x_2 \end{bmatrix}$
As $x = [-1, 1]^T$, $g = \begin{bmatrix} -2 & 2 \end{bmatrix}$

Let's compute $f(x + \alpha p)$
$f(x + \alpha p) = 5 + (x_1 + \alpha p_1)^2 + (x_2 + \alpha p_2)^2$
As $x = [-1 \ \ 1]$ and $p = [1 \ \ 0]$,
$f(x + \alpha p) = 5 + (-1 + \alpha \cdot 1)^2 + (1 + \alpha \cdot 0)^2 = 5 + 1 + \alpha^2 - 2\alpha + 1 = \alpha^2 - 2\alpha + 7$

Let's compute $f(x)$
As $x = [-1 \ \ 1]$,
$f(x) = 5 + (-1)^2 + (1)^2 = 7$

Let's compute $p^T g$
$p^T g = [1 \ \ 0] \cdot \begin{bmatrix} -2 \\ 2 \end{bmatrix} = -2$

Let's substitute all the values into the first Wolfe condition
$f(x + \alpha p) \leq f(x) + c_1 \alpha p^T g$
$\alpha^2 - 2\alpha + 7 \leq 7 - 2 \cdot 10^{-4}\alpha$
$\alpha^2 - 2\alpha + 2 \cdot 10^{-4}\alpha \leq 0$
If we solve this inequality , we obtain
$\alpha \leq 2(1 - 10^{-4}) \approx 1.9998$
So,the maximum length α that satisfies this condition is: $\alpha = 1.9998$

PROBLEM 3
(Modified from Griva, Sofer, and Nash 11.5.8.) Consider the function.
$f(x) = -q^T x + \sum_j y_j log(C^T x)_j$
where C is a very large matrix and computing $C^T x$ is expensive. Show how we can use the structure of the function to reduce the number of matrix vector products $C^T x$ that would be required for checking the Wolfe conditions when searching in a direction p. (Hint, compute $w = C^T p$ once, and see how to re-use it.)

The Wolfe conditions are
1. $f(x_k + \alpha p_k) \leq f(x_k) + \alpha c_1 p_k^T g_k$
2. $g(x_k + \alpha p_k)^T p \geq c_2 g_k^T p_k$

Let's compute the gradient
$\frac{\partial f}{\partial x_i} = -q_i + \sum_j y_j \frac{c_{ij}}{(C^T x)_j}$

When checking the wolfe conditions, we have to compute $f(x_k + \alpha p_k)$ and $g(x_k + \alpha p_k)$:
$f(x_k + \alpha p_k) = -q^T(x_k + \alpha p_k) + \sum_j y_j log(C^T(x_k + \alpha p_k))_j$
$g(x_k + \alpha p_k) = -q_i + \sum_j y_j \frac{c_{ij}}{(C^T(x_k+\alpha p_k))_j}$

At the begining of each iteration k we can compute:
$w = C^T p$
$z = C^T x_k$
$f(x_k)$
$g(x_k)$
Every time we verify if the wolfe conditions are fulfilled we can reuse them, because $C^T x$ is linear.

When computing $f(x_k + \alpha p_k)$ and $g(x_k + \alpha p_k)$, the term $C^T(x_k + \alpha p_k)$ is the most expensive. And as we have computed some values as explained above. Now, we have:
$C^T(x_k + \alpha p_k) = z + \alpha w$
And this is the sum of 2 vectors

Now, the expressions $f(x_k + \alpha p_k)$ and $g(x_k + \alpha p_k)$ are:
$f(x_k + \alpha p_k) = -q^T x_k - q^T x_k \alpha p_k + \sum_j y_j log(z + \alpha w))_j$
$g(x_k + \alpha p_k) = -q_i + \sum_j y_j \frac{c_{ij}}{z_j+\alpha w_j}$
Because we apply $C^T(x_k + \alpha p_k) = z + \alpha w$

So, by computing $w = C^T p$, we reduce the number of matrix-vector products C^T required for checking the Wolfe conditions

PROBLEM 4
Read section 3.5 in Nocedal and Wright on step-length selection. Then solve problem 3.13 (typos are mine).

1. In the notation we have from class where $L(\alpha)$ is the function projected into the current search direction, then show that the quadratic function that interpolates $L(0)$, $L'(0)$, and $L(\alpha_0)$ is given by (3.57), ie

$$\phi_q(\alpha)_q = \left(\frac{L(\alpha_0)-L(0)-\alpha_0 L'(0)}{\alpha_0^2}\right)\alpha^2 + L'(0)\alpha + \phi(0).$$

Let's rewrite the expresion above as:
$$L_q(\alpha)_q = \left(\frac{L(\alpha_0)-L(0)-\alpha_0 L'(0)}{\alpha_0^2}\right)\alpha^2 + L'(0)\alpha + L(0).$$

We have to show that the quadratic function that interpolates $L(0)$, $L'(0)$, and $L(\alpha_0)$ is given by :
$$L_q(\alpha)_q = \left(\frac{L(\alpha_0)-L(0)-\alpha_0 L'(0)}{\alpha_0^2}\right)\alpha^2 + L'(0)\alpha + L(0).$$

Let's start with the the general form of a quadractic function:
$L_q(\alpha) = a\alpha^2 + b\alpha + c$
Then we have to compute the coefficients a,b and c

$L_q(0) = a(0)^2 + b(0) + c = L(0) \Longrightarrow c = L(0)$
$L_q'(0) = 2a(0) + b = L'(0) \Longrightarrow b = L'(0)$
$L_q(\alpha_0) = a(\alpha_0)^2 + b\alpha_0 + c = L(\alpha_0)$, As $b = L_q'(0)$ and $c = L(0)$, we substitute these values:
$L_q(\alpha_0) = a(\alpha_0)^2 + b\alpha_0 + c = a(\alpha_0)^2 + L_q'(0)\alpha_0 + L(0) = L(\alpha_0)$ And we get:
$a = \frac{L(\alpha_0) - L(0) - L_q'(0)\alpha_0}{(\alpha_0)^2}$

So , substituting the values of a,b and c, we obtain
$L_q(\alpha)_q = \left(\frac{L(\alpha_0) - L(0) - \alpha_0 L'(0)}{\alpha_0^2}\right)\alpha^2 + L'(0)\alpha + L(0).$

So, we obtain the same expression than the one given by (3.57).
Although er use the notation from class, the expression is the same.

2.Suppose that the sufficient decrease condition is not satisfied at α_0. Then show that $\alpha_1 < \frac{\alpha_0}{2(1-c_1)}$

If α_0 does not satisfy the sufficient decrease conditions, then:
$0 < L(\alpha_0) - L(0) - c_1 L'(0)\alpha_0$
$\quad < L(\alpha_0) - L(0) - L'(0)\alpha_0$
Because $L'(0) < 0$ and $c1 < 1$. So, $a > 0$.
As L_q is convex and has a minimizer at:
$\alpha_1 = -\frac{L'(0)\alpha_0^2}{2[L(\alpha_0) - L(0) - L'(0)\alpha_0]}$ (3.58)

Notice that
$0 < (c_1 - 1)L'(0)\alpha_0 = L(0) + c_1 L'(0)\alpha_0 - L(0) - L'(0)\alpha_0 < L(\alpha_0) - L(0) - L'(0)\alpha_0.$
The last inequality follows from the supposition that the sufficient decrease condition is not satisfied at α_0.
Ans by using the relations explained above, we show that:
$\alpha_1 < -\frac{L'(0)\alpha_0^2}{2(c_1 - 1)L'(0)\alpha_0} = \frac{\alpha_0}{2(1-c_1)}$