```
import Pkg
Pkg.activate(@__DIR__)
Pkg.instantiate()
using LinearAlgebra, Plots
import ForwardDiff as FD
using Printf
using JLD2
```

```
[32m[1m  Activating[22m[39m environment at
`/home/sman/Work/CMU/Courses/OCRL/OCRL2024/HW/HW1_S24/Project.toml`
```

Q2 (30 pts): Augmented Lagrangian Quadratic Program Solver

Part (A): QP Solver (10 pts)

Here we are going to use the augmented lagrangian method described here in a video, with the corresponding pdf here to solve the following problem:

$$\begin{aligned} \min_{x} \quad & \frac{1}{2} x^T Q x + q^T x \\ \text{s.t.} \quad & Ax - b = 0 \\ & Gx - h \leq 0 \end{aligned}$$

where the cost function is described by $Q \in \mathbb{R}^{n \times n}$, $q \in \mathbb{R}^n$, an equality constraint is described by $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$, and an inequality constraint is described by $G \in \mathbb{R}^{p \times n}$ and $h \in \mathbb{R}^p$.

By introducing a dual variable $\lambda \in \mathbb{R}^m$ for the equality constraint, and $\mu \in \mathbb{R}^p$ for the inequality constraint, we have the following KKT conditions for optimality:

$$\begin{aligned} Qx + q + A^T\lambda + G^T\mu &= 0 && \text{stationarity} \\ Ax - b &= 0 && \text{primal feasibility} \\ Gx - h &\leq 0 && \text{primal feasibility} \\ \mu &\geq 0 && \text{dual feasibility} \\ \mu \odot (Gx - h) &= 0 && \text{complementarity} \end{aligned}$$

where $\circ$ is element-wise multiplication.

```
# TODO: read below
# NOTE: DO NOT USE A WHILE LOOP ANYWHERE
"""
The data for the QP is stored in `qp` the following way:
    @load joinpath(@__DIR__, "qp_data.jld2") qp

which is a NamedTuple, where
    Q, q, A, b, G, h = qp.Q, qp.q, qp.A, qp.b, qp.G, qp.h

contains all of the problem data you will need for the QP.

Your job is to make the following function

    x, λ, μ = solve_qp(qp; verbose = true, max_iters = 100, tol = 1e-8)

You can use (or not use) any of the additional functions:

as long as solve_qp works.
"""
function cost(qp::NamedTuple, x::Vector)::Real
    0.5*x'*qp.Q*x + dot(qp.q,x)
end

function c_eq(qp::NamedTuple, x::Vector)::Vector
    qp.A*x - qp.b
end

function h_ineq(qp::NamedTuple, x::Vector)::Vector
    qp.G*x - qp.h
end

function lagrangian(qp::NamedTuple, x::Vector, λ::Vector, μ::Vector)::Real
    cost(qp,x) + λ'*c_eq(qp,x) + μ'*h_ineq(qp,x)
end

function mask_matrix(qp::NamedTuple, x::Vector, μ::Vector, ρ::Real)::Matrix
    M = h_ineq(qp,x)
    Iᵨ = 1.0*I(length(M))
    for ii = 1:length(M)
        if M[ii]<0 && μ[ii]==0
            Iᵨ[ii,ii]=0
        else
            Iᵨ[ii,ii]=ρ
        end
    end
    return Iᵨ
end
```

```
function augmented_lagrangian(qp::NamedTuple, x::Vector, λ::Vector, μ::Vector,
ρ::Real)::Real
    lagrangian(qp,x,λ,μ) + ρ/2*c_eq(qp,x)'*c_eq(qp,x) +
1/2*h_ineq(qp,x)'*mask_matrix(qp,x,μ,ρ)*h_ineq(qp,x)
end

function logging(qp::NamedTuple, main_iter::Int, AL_gradient::Vector, x::Vector,
λ::Vector, μ::Vector, ρ::Real)
    # TODO: stationarity norm
    stationarity_norm = norm(FD.gradient(_x -> lagrangian(qp,_x,λ,μ),x))
    @printf("%3d  % 7.2e  % 7.2e  % 7.2e  % 7.2e  % 7.2e  %5.0e\n",
          main_iter, stationarity_norm, norm(AL_gradient), maximum(h_ineq(qp,x)),
          norm(c_eq(qp,x),Inf), abs(dot(μ,h_ineq(qp,x))), ρ)
end

function solve_qp(qp; verbose = true, max_iters = 100, tol = 1e-8)
    x = zeros(length(qp.q))
    λ = zeros(length(qp.b))
    μ = zeros(length(qp.h))
    ρ = 1
    φ = 2
    if verbose
        @printf "iter    |∇Lₓ|      |∇ALₓ|     max(h)     |c|        compl     ρ\n"
        @printf "-----------------------------------------------------------------\n"
    end

    kkt(_x) = FD.gradient(__x -> augmented_lagrangian(qp,__x,λ,μ,ρ), _x)

    # TODO:
    for main_iter = 1:max_iters
        if verbose
            logging(qp, main_iter, kkt(x), x, λ, μ, ρ)
        end

        # NOTE: when you do your dual update for μ, you should compute
        # your element-wise maximum with `max.(a,b)`, not `max(a,b)`
        # TODO: convergence criteria based on tol
        if norm(kkt(x)) < tol
            return x, λ, μ
        end

        Δx = -(FD.jacobian(_x -> kkt(_x),x)) \ kkt(x)
        x = x + Δx
        λ += ρ*c_eq(qp,x)
        μ = max.(0,μ+ρ*h_ineq(qp,x))
        ρ *= φ
```

```
    end
    error("qp solver did not converge")
end

let
    # example solving qp
    @load joinpath(@__DIR__, "qp_data.jld2") qp
    x, λ, μ = solve_qp(qp; verbose = true, tol = 1e-8)
end
```

```
iter   |∇Lₓ|      |∇ALₓ|     max(h)     |c|        compl     ρ
-----------------------------------------------------------------
  1   2.98e+01   5.60e+01   4.38e+00   6.49e+00   0.00e+00  1e+00
  2   4.83e+00   1.83e+01   1.55e+00   1.31e+00   2.64e+00  2e+00
  3   7.00e-01   8.70e+00   4.97e-02   6.01e-01   3.12e-01  4e+00
  4   2.39e-01   2.24e+00   3.78e-02   8.34e-02   4.04e-02  8e+00
  5   1.76e+00   5.20e+00   7.09e-02   5.52e-03   3.69e-02  2e+01
  6   4.51e-14   3.32e+00   1.56e-03   2.71e-03   5.22e-06  3e+01
  7   4.39e-14   9.80e-02  -2.16e-04   3.36e-04   2.46e-04  6e+01
  8   2.17e-13   4.77e-03  -5.77e-06   1.25e-05   6.39e-06  1e+02
  9   3.29e-13   1.42e-04  -8.10e-08   1.94e-07   8.92e-08  3e+02
 10   5.50e-13   2.18e-06  -6.05e-10   1.48e-09   6.65e-10  5e+02
 11   2.49e-12   1.70e-08  -2.31e-12   5.70e-12   2.55e-12  1e+03
 12   2.71e-12   6.09e-11  -4.44e-15   1.11e-14   5.07e-15  2e+03
```

```
([-0.326230805713393, 0.24943797997175676, -0.43226766440522546, -1.4172246971242008, -1.3994527400875794,
0.6099582408523462, -0.07312202122168004, 1.3031477522000228, 0.5389034791065959, -0.7225813651685241],
[-0.12835195123488985, -2.8376241672114153, -0.8320804499660779], [0.03635294263949618, 0.0, 0.0,
1.0594444951137387, 0.0])
```

QP Solver test

```
# 10 points
using Test
@testset "qp solver" begin
    @load joinpath(@__DIR__, "qp_data.jld2") qp
    x, λ, μ = solve_qp(qp; verbose = true, max_iters = 100, tol = 1e-6)

    @load joinpath(@__DIR__, "qp_solutions.jld2") qp_solutions
    @test norm(x - qp_solutions.x,Inf)<1e-3;
    @test norm(λ - qp_solutions.λ,Inf)<1e-3;
    @test norm(μ - qp_solutions.μ,Inf)<1e-3;
end
```

```
iter   |∇Lₓ|      |∇ALₓ|     max(h)     |c|        compl     ρ
---------------------------------------------------------------
  1   2.98e+01   5.60e+01   4.38e+00   6.49e+00   0.00e+00  1e+00
  2   4.83e+00   1.83e+01   1.55e+00   1.31e+00   2.64e+00  2e+00
  3   7.00e-01   8.70e+00   4.97e-02   6.01e-01   3.12e-01  4e+00
  4   2.39e-01   2.24e+00   3.78e-02   8.34e-02   4.04e-02  8e+00
  5   1.76e+00   5.20e+00   7.09e-02   5.52e-03   3.69e-02  2e+01
  6   4.51e-14   3.32e+00   1.56e-03   2.71e-03   5.22e-06  3e+01
  7   4.39e-14   9.80e-02  -2.16e-04   3.36e-04   2.46e-04  6e+01
  8   2.17e-13   4.77e-03  -5.77e-06   1.25e-05   6.39e-06  1e+02
  9   3.29e-13   1.42e-04  -8.10e-08   1.94e-07   8.92e-08  3e+02
 10   5.50e-13   2.18e-06  -6.05e-10   1.48e-09   6.65e-10  5e+02
 11   2.49e-12   1.70e-08  -2.31e-12   5.70e-12   2.55e-12  1e+03
[0m[1mTest Summary: | [22m[32m[1mPass  [22m[39m[36m[1mTotal[22m[39m
qp solver     | [32m   3  [39m[36m    3[39m



Test.DefaultTestSet("qp solver", Any[], 3, false, false)
```

Simulating a Falling Brick with QPs

In this question we'll be simulating a brick falling and sliding on ice in 2D. You will show that this problem can be formulated as a QP, which you will solve using an Augmented Lagrangian method.

The Dynamics

The dynamics of the brick can be written in continuous time as

$$M\dot{v} + Mg = J^T\mu \text{ where } M = mI_{2\times 2},\ g = \begin{bmatrix} 0 \\ 9.81 \end{bmatrix},\ J = \begin{bmatrix} 0 & 1 \end{bmatrix}$$

and $\mu \in \mathbb{R}$ is the normal force. The velocity $v \in \mathbb{R}^2$ and position $q \in \mathbb{R}^2$ are composed of the horizontal and vertical components.

We can discretize the dynamics with backward Euler:

$$\begin{bmatrix} v_{k+1} \\ q_{k+1} \end{bmatrix} = \begin{bmatrix} v_k \\ q_k \end{bmatrix} + \Delta t \cdot \begin{bmatrix} \frac{1}{m} J^T \mu_{k+1} - g \\ v_{k+1} \end{bmatrix}$$

We also have the following contact constraints:

$$\begin{aligned} Jq_{k+1} &\geq 0 && \text{(don't fall through the ice)} \\ \mu_{k+1} &\geq 0 && \text{(normal forces only push, not pull)} \\ \mu_{k+1} Jq_{k+1} &= 0 && \text{(no force at a distance)} \end{aligned}$$

Part (B): QP formulation for Falling Brick (5 pts)

Show that these discrete-time dynamics are equivalent to the following QP by writing down the KKT conditions.

$$\begin{aligned} \text{minimize}_{v_{k+1}} \quad & \frac{1}{2} v_{k+1}^T M v_{k+1} + [M(\Delta t \cdot g - v_k)]^T v_{k+1} \\ \text{subject to} \quad & -J(q_k + \Delta t \cdot v_{k+1}) \leq 0 \end{aligned}$$

TASK: Write down the KKT conditions for the optimization problem above, and show that it's equivalent to the dynamics problem stated previously. Use LaTeX markdown.

PUT ANSWER HERE:

KKT:

$$\begin{aligned} M\, v_{k+1} + M(\Delta t\, g - v_k) - \Delta t\, J^T\, \mu_{k+1} &= 0 && \text{(stationarity)} \\ M\, v_{k+1} - M\, v_k &= \Delta t[J^T \mu_{k+1} - M\, g] \\ v_{k+1} &= v_k + \Delta t \left[\frac{1}{m} J^T\, \mu_{k+1} - g\right] \\ q_{k+1} &= q_k + \Delta t\, v_{k+1} && \text{(velocity kinematics)} \\ -J(q_k + \Delta t\, v_{k+1}) &\leq 0 && \text{(primal feasibility)} \\ J\, q_{k+1} &\geq 0 \\ \mu_{k+1} &\geq 0 && \text{(dual feasibility)} \\ \mu_{k+1} \odot -J(q_k + \Delta t \cdot v_{k+1}) &= 0 && \text{(complementarity)} \\ \mu_{k+1}\, J\, q_{k+1} &= 0 \end{aligned}$$

Part ©: Brick Simulation (5 pts)

```
function brick_simulation_qp(q, v; mass = 1.0, Δt = 0.01)

    # TODO: fill in the QP problem data for a simulation step
    # fill in Q, q, G, h, but leave A, b the same
    # this is because there are no equality constraints in this qp

    g = [0,9.81]
    J = [0 1]

    qp = (
        Q = Matrix(mass*I(2)),
        q = Matrix(mass*I(2))*(Δt*g-v),
        A = zeros(0,2), # don't edit this
        b = zeros(0),   # don't edit this
        G = -Δt*J,
        h = J*q
    )

    return qp
end
```

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

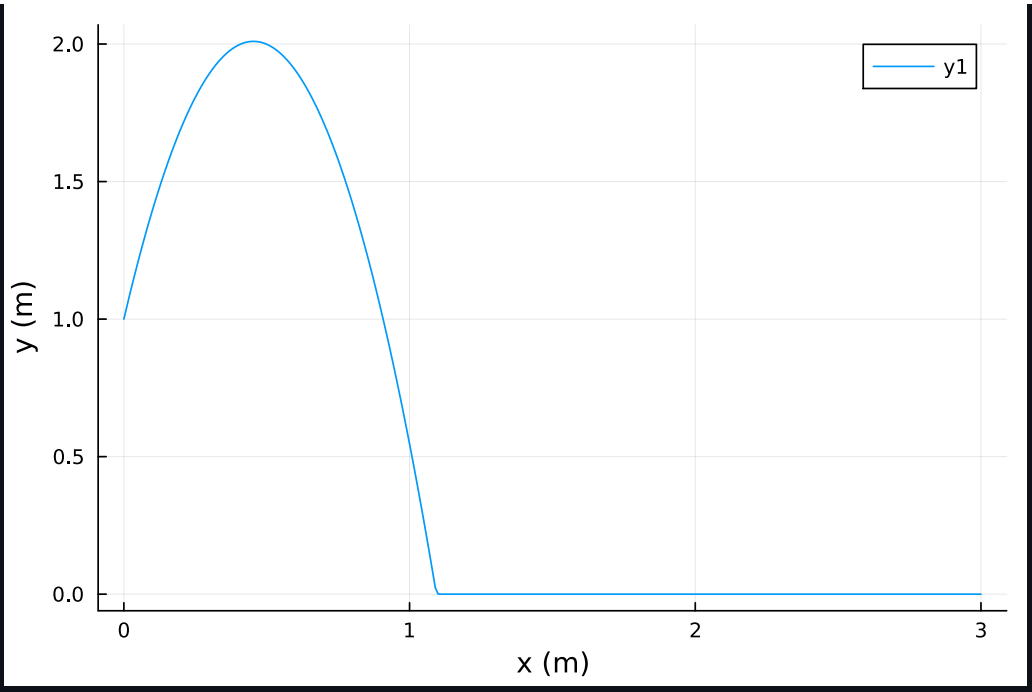
```
@testset "brick qp" begin

    q = [1,3.0]
    v = [2,-3.0]

    qp = brick_simulation_qp(q,v)
    @show typeof(qp.Q)
    # check all the types to make sure they're right
    qp.Q::Matrix{Float64}
    qp.q::Vector{Float64}
    qp.A::Matrix{Float64}
    qp.b::Vector{Float64}
    qp.G::Matrix{Float64}
    qp.h::Vector{Float64}

    @test size(qp.Q) == (2,2)
    @test size(qp.q) == (2,)
    @test size(qp.A) == (0,2)
    @test size(qp.b) == (0,)
    @test size(qp.G) == (1,2)
    @test size(qp.h) == (1,)

    @test abs(tr(qp.Q) - 2) < 1e-10
    @test norm(qp.q - [-2.0, 3.0981]) < 1e-10
    @test norm(qp.G - [0 -.01]) < 1e-10
    @test abs(qp.h[1] -3) < 1e-10

end
```

```
typeof(qp.Q) = Matrix{Float64}
[0m[1mTest Summary: | [22m[32m[1mPass  [22m[39m[36m[1mTotal[22m[39m
brick qp      | [32m  10  [39m[36m   10[39m



Test.DefaultTestSet("brick qp", Any[], 10, false, false)
```

```
include(joinpath(@__DIR__, "animate_brick.jl"))
let

    dt = 0.01
    T = 3.0

    t_vec = 0:dt:T
    N = length(t_vec)

    qs = [zeros(2) for i = 1:N]
    vs = [zeros(2) for i = 1:N]

    qs[1] = [0, 1.0]
    vs[1] = [1, 4.5]

    # TODO: simulate the brick by forming and solving a qp
    # at each timestep. Your QP should solve for vs[k+1], and
    # you should use this to update qs[k+1]
    for ii = 2:N
        qp = brick_simulation_qp(qs[ii-1],vs[ii-1];Δt=dt)
        vs[ii], λ, μ = solve_qp(qp; verbose = true, max_iters = 100, tol = 1e-6)
        qs[ii] = qs[ii-1]+dt*vs[ii]
    end

    xs = [q[1] for q in qs]
    ys = [q[2] for q in qs]

    @show @test abs(maximum(ys)-2)<1e-1
    @show @test minimum(ys) > -1e-2
    @show @test abs(xs[end] - 3) < 1e-2

    xdot = diff(xs)/dt
    @show @test maximum(xdot) < 1.0001
    @show @test minimum(xdot) > 0.9999
    @show @test ys[110] > 1e-2
    @show @test abs(ys[111]) < 1e-2
    @show @test abs(ys[112]) < 1e-2

    display(plot(xs, ys, ylabel = "y (m)", xlabel = "x (m)"))

    animate_brick(qs)
end
```

```
iter   |∇L_x|      |∇AL_x|     max(h)     |c|        compl     ρ
-----------------------------------------------------------------
  1   4.51e+00   4.51e+00  -1.00e+00   0.00e+00   0.00e+00  1e+00
  2   0.00e+00   0.00e+00  -1.04e+00   0.00e+00   0.00e+00  2e+00
iter   |∇L_x|      |∇AL_x|     max(h)     |c|        compl     ρ
-----------------------------------------------------------------
  1   4.42e+00   4.42e+00  -1.04e+00   0.00e+00   0.00e+00  1e+00
  2   0.00e+00   0.00e+00  -1.09e+00   0.00e+00   0.00e+00  2e+00
iter   |∇L_x|      |∇AL_x|     max(h)     |c|        compl     ρ
-----------------------------------------------------------------
  1   4.32e+00   4.32e+00  -1.09e+00   0.00e+00   0.00e+00  1e+00
  2   0.00e+00   0.00e+00  -1.13e+00   0.00e+00   0.00e+00  2e+00
iter   |∇L_x|      |∇AL_x|     max(h)     |c|        compl     ρ
-----------------------------------------------------------------
  1   4.23e+00   4.23e+00  -1.13e+00   0.00e+00   0.00e+00  1e+00
  2   0.00e+00   0.00e+00  -1.17e+00   0.00e+00   0.00e+00  2e+00
iter   |∇L_x|      |∇AL_x|     max(h)     |c|        compl     ρ
-----------------------------------------------------------------
  1   4.13e+00   4.13e+00  -1.17e+00   0.00e+00   0.00e+00  1e+00
  2   0.00e+00   0.00e+00  -1.21e+00   0.00e+00   0.00e+00  2e+00
iter   |∇L_x|      |∇AL_x|     max(h)     |c|        compl     ρ
-----------------------------------------------------------------
  1   4.04e+00   4.04e+00  -1.21e+00   0.00e+00   0.00e+00  1e+00
  2   0.00e+00   0.00e+00  -1.25e+00   0.00e+00   0.00e+00  2e+00
iter   |∇L_x|      |∇AL_x|     max(h)     |c|        compl     ρ
-----------------------------------------------------------------
  1   3.94e+00   3.94e+00  -1.25e+00   0.00e+00   0.00e+00  1e+00
  2   0.00e+00   0.00e+00  -1.29e+00   0.00e+00   0.00e+00  2e+00
iter   |∇L_x|      |∇AL_x|     max(h)     |c|        compl     ρ
-----------------------------------------------------------------
  1   3.85e+00   3.85e+00  -1.29e+00   0.00e+00   0.00e+00  1e+00
  2   0.00e+00   0.00e+00  -1.32e+00   0.00e+00   0.00e+00  2e+00
iter   |∇L_x|      |∇AL_x|     max(h)     |c|        compl     ρ
-----------------------------------------------------------------
  1   3.75e+00   3.75e+00  -1.32e+00   0.00e+00   0.00e+00  1e+00
  2   0.00e+00   0.00e+00  -1.36e+00   0.00e+00   0.00e+00  2e+00
iter   |∇L_x|      |∇AL_x|     max(h)     |c|        compl     ρ
-----------------------------------------------------------------
  1   3.66e+00   3.66e+00  -1.36e+00   0.00e+00   0.00e+00  1e+00
  2   0.00e+00   0.00e+00  -1.40e+00   0.00e+00   0.00e+00  2e+00
iter   |∇L_x|      |∇AL_x|     max(h)     |c|        compl     ρ
-----------------------------------------------------------------
  1   3.56e+00   3.56e+00  -1.40e+00   0.00e+00   0.00e+00  1e+00
  2   0.00e+00   0.00e+00  -1.43e+00   0.00e+00   0.00e+00  2e+00
iter   |∇L_x|      |∇AL_x|     max(h)     |c|        compl     ρ
-----------------------------------------------------------------
  1   3.47e+00   3.47e+00  -1.43e+00   0.00e+00   0.00e+00  1e+00
  2   0.00e+00   0.00e+00  -1.46e+00   0.00e+00   0.00e+00  2e+00
iter   |∇L_x|      |∇AL_x|     max(h)     |c|        compl     ρ
-----------------------------------------------------------------
  1   3.38e+00   3.38e+00  -1.46e+00   0.00e+00   0.00e+00  1e+00
  2   0.00e+00   0.00e+00  -1.50e+00   0.00e+00   0.00e+00  2e+00
iter   |∇L_x|      |∇AL_x|     max(h)     |c|        compl     ρ
-----------------------------------------------------------------
  1   3.28e+00   3.28e+00  -1.50e+00   0.00e+00   0.00e+00  1e+00
  2   0.00e+00   0.00e+00  -1.53e+00   0.00e+00   0.00e+00  2e+00
iter   |∇L_x|      |∇AL_x|     max(h)     |c|        compl     ρ
```

```
 13   0.00e+00   2.74e-02   6.68e-04   0.00e+00   2.09e-03  4e+03
 14   0.00e+00   3.88e-02   4.74e-04   0.00e+00   2.40e-03  8e+03
 15   0.00e+00   4.27e-02   2.60e-04   0.00e+00   1.88e-03  2e+04
 16   1.39e-17   3.23e-02   9.87e-05   0.00e+00   8.71e-04  3e+04
 17   1.39e-17   1.51e-02   2.31e-05   0.00e+00   2.21e-04  7e+04
 18   0.00e+00   4.01e-03   3.06e-06   0.00e+00   2.99e-05  1e+05
 19   1.39e-17   5.68e-04   2.17e-07   0.00e+00   2.12e-06  3e+05
 20   0.00e+00   4.17e-05   7.96e-09   0.00e+00   7.81e-08  5e+05
 21   0.00e+00   1.56e-06   1.49e-10   0.00e+00   1.46e-09  1e+06
 22   1.39e-17   2.95e-08   1.41e-12   0.00e+00   1.38e-11  2e+06
iter   |∇Lₓ|      |∇ALₓ|     max(h)     |c|        compl     ρ
---------------------------------------------------------------
  1   1.00e+00   1.00e+00   1.41e-12   0.00e+00   0.00e+00  1e+00
  2   6.31e-18   1.96e-05   9.81e-04   0.00e+00   9.62e-07  2e+00
  3   5.06e-18   3.92e-05   9.81e-04   0.00e+00   2.89e-06  4e+00
  4   8.40e-19   7.84e-05   9.80e-04   0.00e+00   6.73e-06  8e+00
  5   1.71e-18   1.57e-04   9.80e-04   0.00e+00   1.44e-05  2e+01
  6   6.51e-18   3.13e-04   9.78e-04   0.00e+00   2.97e-05  3e+01
  7   9.76e-19   6.24e-04   9.75e-04   0.00e+00   6.00e-05  6e+01
  8   5.42e-18   1.24e-03   9.69e-04   0.00e+00   1.20e-04  1e+02
  9   2.60e-18   2.45e-03   9.56e-04   0.00e+00   2.35e-04  3e+02
 10   6.94e-18   4.77e-03   9.33e-04   0.00e+00   4.52e-04  5e+02
 11   6.94e-18   9.08e-03   8.87e-04   0.00e+00   8.33e-04  1e+03
 12   3.47e-18   1.65e-02   8.05e-04   0.00e+00   1.42e-03  2e+03
 13   0.00e+00   2.74e-02   6.68e-04   0.00e+00   2.09e-03  4e+03
 14   0.00e+00   3.88e-02   4.74e-04   0.00e+00   2.40e-03  8e+03
 15   0.00e+00   4.27e-02   2.60e-04   0.00e+00   1.88e-03  2e+04
 16   1.39e-17   3.23e-02   9.87e-05   0.00e+00   8.71e-04  3e+04
 17   1.39e-17   1.51e-02   2.31e-05   0.00e+00   2.21e-04  7e+04
 18   0.00e+00   4.01e-03   3.06e-06   0.00e+00   2.99e-05  1e+05
 19   1.39e-17   5.68e-04   2.17e-07   0.00e+00   2.12e-06  3e+05
 20   0.00e+00   4.17e-05   7.96e-09   0.00e+00   7.81e-08  5e+05
 21   0.00e+00   1.56e-06   1.49e-10   0.00e+00   1.46e-09  1e+06
 22   1.39e-17   2.95e-08   1.41e-12   0.00e+00   1.38e-11  2e+06
#= /home/sman/Work/CMU/Courses/OCRL/OCRL2024/HW/HW1_S24/Q3.ipynb:28 =# @test(abs(maximum(ys) - 2) < 0.1) =
Test Passed
#= /home/sman/Work/CMU/Courses/OCRL/OCRL2024/HW/HW1_S24/Q3.ipynb:29 =# @test(minimum(ys) > -0.01) = Test
Passed
#= /home/sman/Work/CMU/Courses/OCRL/OCRL2024/HW/HW1_S24/Q3.ipynb:30 =# @test(abs(xs[end] - 3) < 0.01) =
Test Passed
```




```
#= /home/sman/Work/CMU/Courses/OCRL/OCRL2024/HW/HW1_S24/Q3.ipynb:33 =# @test(maximum(xdot) < 1.0001) =
Test Passed
#= /home/sman/Work/CMU/Courses/OCRL/OCRL2024/HW/HW1_S24/Q3.ipynb:34 =# @test(minimum(xdot) > 0.9999) =
Test Passed
#= /home/sman/Work/CMU/Courses/OCRL/OCRL2024/HW/HW1_S24/Q3.ipynb:35 =# @test(ys[110] > 0.01) = Test Passed
#= /home/sman/Work/CMU/Courses/OCRL/OCRL2024/HW/HW1_S24/Q3.ipynb:36 =# @test(abs(ys[111]) < 0.01) = Test
Passed
#= /home/sman/Work/CMU/Courses/OCRL/OCRL2024/HW/HW1_S24/Q3.ipynb:37 =# @test(abs(ys[112]) < 0.01) = Test
Passed


┌ Info: MeshCat server started. You can open the visualizer by visiting the following URL in your browser:
│ http://127.0.0.1:8703
└ @ MeshCat /root/.julia/packages/MeshCat/vWPbP/src/visualizer.jl:73
```

Part D (5 pts): Solve a QP

Use your QP solver to solve the following optimization problem:

$$
\begin{aligned}
\min_{y\in\mathbb{R}^2, a\in\mathbb{R}, b\in\mathbb{R}} \quad & \frac{1}{2} y^T \begin{bmatrix} 1 & .3 \\ .3 & 1 \end{bmatrix} y + a^2 + 2b^2 + \begin{bmatrix} -2 & 3.4 \end{bmatrix} y + 2a + 4b \\
\text{st} \quad & a + b = 1 \\
& \begin{bmatrix} -1 & 2.3 \end{bmatrix} y + a - 2b = 3 \\
& -0.5 \leq y \leq 1 \\
& -1 \leq a \leq 1 \\
& -1 \leq b \leq 1
\end{aligned}
$$

You should be able to put this into our standard QP form that we used above, and solve.

```
function rand_qp()

    qp = (
        Q = [  1 0.3 0 0;
             0.3   1 0 0;
               0   0 2 0;
               0   0 0 4],
        q = [-2, 3.4, 2, 4],
        A = [ 0   0 1  1;
             -1 2.3 1 -2],
        b = [1, 3],
        G = [ 1  0  0  0;
              0  1  0  0;
             -1  0  0  0;
              0 -1  0  0;
              0  0  1  0;
              0  0 -1  0;
              0  0  0  1;
              0  0  0 -1],
        h = [1, 1, 0.5, 0.5, 1, 1, 1, 1]
    )

    return qp
end
```

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

```
@testset "part D" begin

    x, λ, μ = solve_qp(rand_qp(); verbose = true, max_iters = 100, tol = 1e-6)
    y = x[1:2]
    a = x[3]
    b = x[4]

    @test norm(y - [-0.080823; 0.834424]) < 1e-3
    @test abs(a - 1) < 1e-3
    @test abs(b) < 1e-3
end
```

```
iter   |∇Lₓ|      |∇ALₓ|     max(h)     |c|        compl     ρ
---------------------------------------------------------------
  1   5.96e+00   9.91e+00  -5.00e-01   3.00e+00   0.00e+00  1e+00
  2   2.77e-01   9.55e+00   2.77e-01   1.71e+00   7.69e-02  2e+00
  3   6.01e-01   2.37e+00   3.01e-01   6.65e-01   1.74e-01  4e+00
  4   7.19e-01   4.35e+00   3.12e-01   4.56e-01   5.78e-01  8e+00
  5   3.64e-15   2.09e+00   1.94e-01   1.50e-01   6.61e-01  2e+01
  6   4.97e-15   1.75e+00   7.82e-02   5.97e-02   3.64e-01  3e+01
  7   3.80e-14   8.74e-01   1.96e-02   1.49e-02   1.03e-01  6e+01
  8   2.74e-14   2.50e-01   2.80e-03   2.13e-03   1.53e-02  1e+02
  9   4.66e-14   3.85e-02   2.15e-04   1.64e-04   1.18e-03  3e+02
 10   1.05e-13   3.08e-03   8.62e-06   6.57e-06   4.74e-05  5e+02
 11   3.44e-13   1.26e-04   1.76e-07   1.34e-07   9.67e-07  1e+03
 12   1.03e-12   2.60e-06   1.82e-09   1.38e-09   9.98e-09  2e+03
 13   2.27e-12   2.69e-08   9.42e-12   7.18e-12   5.18e-11  4e+03
␛[0m␛[1mTest Summary: | ␛[22m␛[32m␛[1mPass  ␛[22m␛[39m␛[36m␛[1mTotal␛[22m␛[39m
part D        | ␛[32m   3  ␛[39m␛[36m    3␛[39m



Test.DefaultTestSet("part D", Any[], 3, false, false)
```

Part E (5 pts): One sentence short answer

1. For our Augmented Lagrangian solver, if our initial guess for x is feasible (meaning it satisfies the constraints), will it stay feasible through each iteration?

No, the augmented lagrangian penalizes violation rather than increase cost infinitely near violation

2. Does the Augmented Lagrangian function for this problem always have continuous first derivatives?

Yes, the max(0,c(x))^2 term of the AL has continuous derivative, since the Lagrangian also has a continuous derivative, then AL overall is continuous

3. Is the QP in part D always convex?

Yes, the hessian is PD

```
# check if part D QP is always convex:
eigvals(rand_qp().Q)
```

```
4-element Vector{Float64}:
 0.7
 1.3
 2.0
 4.0
```