AM205 Final Project
Muscle Redundancy Optimization: Comparing QP, SQP, and Newton–Lagrange

Darius Sattari

December 9, 2025

Page 0

1. Textbook Chapter

This work is rooted primarily in **Chapter 6: Optimization** from Heath [2]. The muscle redundancy problem is formulated as a constrained optimization problem, analyzed through KKT conditions, Newton–Lagrange updates, and sequential quadratic programming.

2. Contributions of the Author

This project was completed individually. All numerical experiments, solver implementation, data generation, and writing were conducted by myself. MATLAB scripts and solvers were written from scratch unless otherwise specified that I used native MATLAB packages.

3. Use of AI Tools

Large language models (GPT 4, Claude Opus 4.5) were used for assistance with organization, clarity, structuring sections, general LaTeX formatting, and generating LaTeX summary tables of results from the results printed to the MATLAB terminal. All mathematical derivations, coding implementation, experiments, and analysis were constructed and performed by the author, with the exception of seeking debugging assistance for MATLAB syntax and advice on generating visualization plots. The core KKT solver logic and active-set algorithm structure were derived and implemented on my own with the help of cited resources. AI was not used to blindly generate equations or code logic.

4. Other Sources

No external collaborators contributed to the development of this project. Discussions were limited to course material and cited published research, which are referenced formally in the bibliography.

Abstract

This project investigates the classical muscle redundancy problem in elbow flexion, where multiple muscles can generate the same joint torque, leading to an underdetermined force-sharing problem. The muscle forces $f \in \mathbb{R}^n$ are computed by solving a constrained optimization problem in a system with moment arm r, torque τ_d, and objective function $J(f)$:

$$\min_f J(f) \quad \text{s.t.} \quad r^\top f = \tau_d, \quad 0 \le f_i \le f_{\max,i},$$

with three objective choices reflecting different neuromuscular control hypotheses: a quadratic effort cost, a cubic effort cost, and a minimax cost on peak muscle force. Three numerical methods are compared using MATLAB's quadratic programming solver (`quadprog`), sequential quadratic programming via `fmincon` (SQP), and a custom Newton–Lagrange KKT solver implemented from scratch.

Four experiments are conducted to evaluate convergence from different initial system state guesses, scaling with the number of muscles, sensitivity to nonlinear objectives, and robustness under ill-conditioned moment arms. For smooth quadratic and cubic costs, the Newton–Lagrange method typically converges in a single iteration and matches the QP and SQP solutions to high accuracy, while SQP requires more iterations but handles all objective types, including the non-smooth minimax formulation. Overall, the results show that Newton–Lagrange is highly efficient and scalable for smooth objectives, whereas SQP provides greater modeling flexibility when non-smooth or more complex neuromuscular objectives are of interest.

1 Introduction

In the study of mathematics, *redundancy* refers to the existence of an infinite number of possible solutions. This is no different in the field of biomechanics and neuroscience. When the Central Nervous System (CNS) is attempting to control the movements of the musculoskeletal system, there are an infinite number of muscle and neural activation patterns [5]. When you perform a bicep curl at the gym, there are theoretically infinite numbers of *solutions* your CNS can converge on to achieve this task [3].

When we have multiple muscles, or arrays of muscle fibers, that can generate the same joint torque, optimization becomes a pertinent question in biomechanics. If a task needs to be completed in repetition, for something as general as walking or as something as specific as chopping wood with an axe, the CNS will modify neural activation patterns to prevent fatigue and reduce the possibility of overuse [7, 6].

The focus of this paper will be the *elbow problem*. This is where muscles must be cued to perform an elbow flexion (think the motion of a bicep curl) at specified torque. The main decision variable for the CNS is the vector of muscle forces denoted by the vector f:

$$f = [f_1, f_2, ..., f_n]^T$$

To perform a given movement requiring a desired torque τ_d, this torque must be matched, giving the constraint, including a moment arm r representing the leverage distance:

$$r^\top f = \tau_d.$$

When $n > 1$, this system is *under-determined*: infinitely many force distributions f satisfy the torque constraint, with a set moment arm r reflecting redundancy in neuromuscular control [3]. To select a unique and physiologically meaningful solution, we express the problem as a constrained optimization:

$$\begin{aligned} \min_{f} \quad & J(f) \\ \text{s.t.} \quad & r^\top f = \tau_d, \\ & 0 \le f_i \le f_{max,i}. \end{aligned}$$

The choice of objective function $J(f)$ encodes assumptions about neural control strategies. In this paper, three perspectives are explored:

$$J_{\text{quad}}(f) = \sum_i f_i^2, \qquad J_{\text{cubic}}(f) = \sum_i f_i^3, \qquad J_{\text{minimax}}(f) = \max_i f_i,$$

each producing distinct recruitment patterns. This enables comparison of the different solver performances [1, 4].

This work directly intersects with the optimization concepts presented in Chapter 6 of *Heath* [2]. This muscle redundancy problem is framed using KKT, Newton-Lagrange, and constrained optimization methods. In this sense, the elbow torque problem serves as a real-world instance of the mathematical theory introduced in AM205 topic areas such as translating KKT conditions, Newton updates, and constraint satisfaction into a working computational optimizer [2].

The ultimate goal of this report is to build a Newton-Lagrange solver from scratch to solve the muscle redundancy problem. And to then compare it with MATLAB's built-in KKT and constrained optimization solvers in terms of convergence, scalability, robustness, and objective behavior. The optimizations methods explored in this paper are Quadratic Programming, Sequential Quadratic Programming, Newton-Lagrange, and a Minimax Objective Function.

2 Optimization Methods

This section discusses the theory and tailored application of the optimization methods discussed in this paper. Please refer to **Appendix A** for an in depth look into the source code used to leverage these techniques.

2.1 Quadratic Programming (`quadprog`)

QP is leveraged as a baseline method in this paper. The quadratic objective used is

$$J(f) = \sum f_i^2 = f^T f,$$

such that the KKT system is linear and convex, satisfying Heath's definition of a QP problem [2]. This allows QP to solve the force-sharing problem in just one step. This makes QP ideal for comparisons in timing accuracy, however it is limited to this objective function. Yet, this objective does present a physiologically meaningful interpretation. Attempting to minimize squared force models spreading load across muscle rather than optimizing a single muscle. The corresponds to the concept of minimizing effort and metabolic cost, which is a common CNS driven theory.

2.2 Sequential Quadratic Programming (`fmincon`)

SQP is employed because it can handle objective functions beyond the quadratic case presented in QP, such that, it iteratively constructs and solves a series of quadratic subproblems, updating f [2]. This will enable the minimization of the minimax and cubic objective functions:

$$\text{CUBIC: } J = \sum_i f_i^3, \qquad \text{MINIMAX: } J = \max_i f_i$$

where in the Cubic function curvature varies with force magnitude. And in the minimal objective, muscle loading is equalized rather than minimizing the total squared or cubed effort [2].

2.3 Newton–Lagrange (`solve_newton_lagrange`)

The Newton–Lagrange method is implemented as a direct solver for the KKT optimality conditions associated with smooth objective functions. Rather than relying on black-box iterations, this method explicitly constructs the saddle-point system formed by the gradient of the Lagrangian and the torque constraint. Heath also states the gradient of the Lagrangian must satisfy equivalence to zero [2]. Thus, the first-order optimality conditions yield the coupled nonlinear system:

$$\nabla_f J(f) + \lambda r = 0, \qquad r^\top f - \tau_d = 0. \qquad [2]$$

To solve this, we apply Newton's method. At each iteration k, we solve the following symmetric linear system to find the update steps Δf and $\Delta\lambda$:

$$\begin{bmatrix} \nabla^2 J(f_k) & r \\ r^\top & 0 \end{bmatrix} \begin{bmatrix} \Delta f \\ \Delta\lambda \end{bmatrix} = - \begin{bmatrix} \nabla J(f_k) + \lambda_k r \\ r^\top f_k - \tau_d \end{bmatrix}. \qquad [2]$$

This system explicitly links the Hessian of the objective $\nabla^2 J$ (curvature) with the geometric constraints r. For the quadratic objective $J(f) = f^\top f$, the Hessian is constant ($\nabla^2 J = 2I$), reducing the problem to a single linear system solve [2]. However, this formulation is not suitable for the minimax case due to the nonsmooth kink in $J = \max_i f_i$, which prevents forming the necessary Hessian [2].

While the other methods utilized readily available MATLAB packages, this Newton-Lagrange solver was specifically constructed from scratch for this study to demonstrate the direct linear algebra approach. Please refer to **Appendix A.1** for the source code.

3 Experiments & Results

Four numerical experiments are presented evaluating solver performance under changes in initialization, dimensionality, objective function, and problem conditioning. All results were generated from the MATLAB framework described in Section 2, and corresponding figures and tables are included for reference. Timing values are reported in milliseconds, iteration counts represent full optimization steps, and Newton function-evaluations measure gradient evaluations required to form the KKT system. Refer to **Appendix A.3** to find the driving scripts behind these experiments.

3.1 Experiment 1: Convergence vs Initial Guess

This experiment tests the robustness of each solver to different muscular starting points using a two–muscle configuration with a torque target of $\tau_d = 18$ N–m, which is about half the mean max for a healthy adult [6]. Eight initial guesses were chosen, ranging from very low activation to high activation, asymmetric distributions, and values exceeding bound constraints:

- **Uniform Guesses:** Low (50,N), Mid (150,N), and High (250,N) uniform distributions representing generalized co-contraction levels across multiple muscle groups.
- **Asymmetric Guesses:** Scenarios where one muscle group is heavily pre-loaded (200,N) while the other is relaxed (50,N), mimicking imbalances caused by fatigue or favoring.
- **Infeasible Points:** Above bounds and Very low guesses that violate the box constraints $0 \leq f \leq f_{max}$, forcing the solver to project back into the feasible region immediately.

The results of these initial guesses run through the optimization methods are reported below in Table 1 and Figure 1:

Initial Guess	**Time (ms)**			**Iterations**	
	QP	**SQP**	**Newton**	**SQP**	**Newton**
Low uniform	71.08	2722.03	444.83	2	2
Mid uniform	11.95	15.16	18.10	2	2
High uniform	70.79	14.77	31.29	2	2
Asymmetric 1	15.52	44.72	13.51	2	2
Asymmetric 2	5.01	17.22	14.24	2	2
Very low	5.86	10.15	12.44	2	2
Near optimal	5.04	13.94	27.97	2	2
Above bounds	2.06	15.37	3.42	2	2

Table 1: Convergence performance across eight initial guesses. Initial guesses simulate real life edge-case scenarios, such as pre-fatigue, confusion, or incorrect inputs.

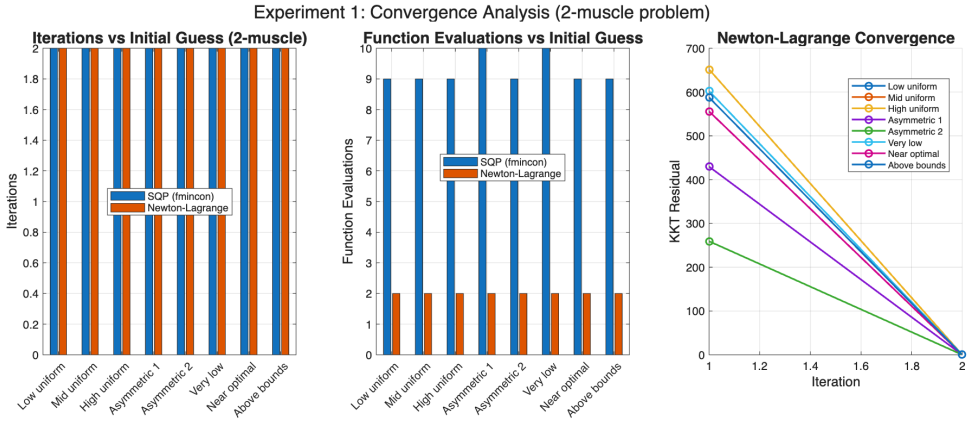

Figure 1: Convergence behavior under different initial guesses. Newton–Lagrange converges in exactly two iterations with only two function evaluations, while SQP requires more evaluations but a similar number of iterations. KKT residuals show Newton–Lagrange reaching machine precision in two steps for all initializations.

As expected, QP is insensitive to initialization and returns the solution directly. Both SQP and Newton–Lagrange converged in exactly two iterations for every starting point, indicating a smooth and well-conditioned low-dimensional landscape. Runtime varied across trials, but iteration counts remained invariant, suggesting low sensitivity to initial conditions.

3.2 Experiment 2: Scaling with Number of Muscles (n)

To evaluate computational scalability, the number of muscles was varied from $n = 2$ to $n = 20$. At each value of n muscles, all three optimization methods were run. Time, Iterations, and Error are all compared to to values from QP as a baseline, as shown in Table 2 and Figure 2:

	QP (Baseline)	SQP			Newton		
n	**Time (ms)**	**Time (ms)**	**Iters**	**Error**	**Time (ms)**	**Iters**	**Error**
2	46.13	58.44	3	5e-08	22.83	1	4e-11
4	3.73	17.66	4	2e-06	0.44	1	5e-11
6	5.35	14.83	4	3e-06	0.50	1	6e-11
8	2.23	9.17	6	2e-05	0.38	1	3e-10
10	1.75	15.48	8	2e-05	0.41	1	2e-09
12	2.55	9.47	8	3e-05	2.02	1	6e-12
14	8.98	16.76	6	4e-05	0.64	1	2e-10
16	3.32	7.62	6	7e-05	26.33	2	2e-09
18	2.57	12.60	7	8e-05	5.95	2	5e-11
20	2.34	10.49	7	8e-05	0.76	1	1e-09

Table 2: Scalability from $n = 2$ to $n = 20$ muscles. The error is based on deviation from QP.

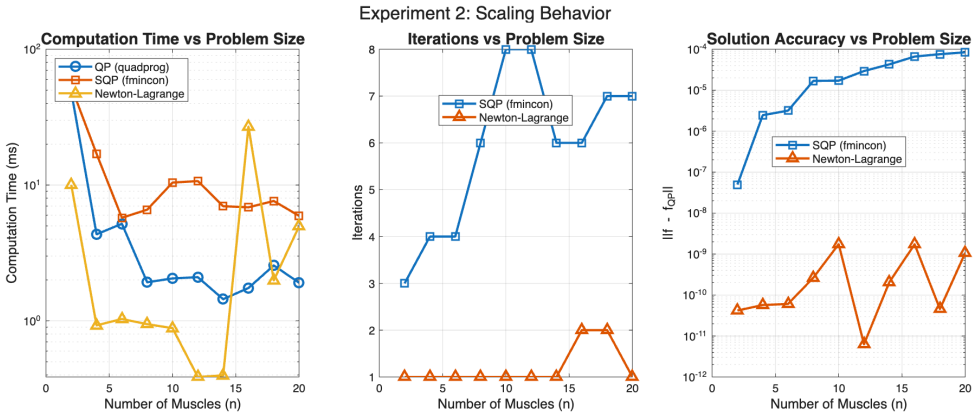

Figure 2: Scaling with problem size. Newton–Lagrange is the fastest method and converges in 1–2 iterations for all n, while SQP requires 6–8 iterations and higher computational overhead. Newton–Lagrange also achieves consistently higher accuracy across all problem sizes.

Newton–Lagrange consistently converged in one step for all problem sizes due to the linearity of the quadratic KKT system. SQP required between three and eight iterations while remaining highly accurate relative to the QP baseline. Timing remained low for all methods, although SQP increased slightly with n due to repeated subproblem solves. This experiment confirms the efficiency of the Newton update when the objective is smooth and quadratic, while SQP exhibits predictable scaling behavior consistent with general nonlinear solvers.

3.3 Experiment 3: Objective Function Comparison

The three objective functions were evaluated on a five–muscle system to assess how solver behavior changes with curvature and smoothness. Five muscles were selected as this is a common simplification typically the Biceps Brachii Long Head, Biceps Brachii Short Head, Brachialis, Brachioradialis, and sometimes the Pronator Teres [3, 7]. The results are displayed in Table 3 and Figure 3:

	QP	SQP			Newton		
Objective	**Time (ms)**	**Time (ms)**	**Iters**	**Feval**	**Time (ms)**	**Iters**	**Feval**
Quadratic	29.11	72.50	6	42	10.23	1	1
Cubic	–	26.84	16	138	47.72	1	1
Minimax	–	111.44	–	–	–	–	–

Table 3: Performance comparison across different objective functions. Dashes (-) denote cases where the solver was not run for that objective. SQP was able to evaluate the minimax objective but could not perform Newton-style update iterations, and therefore no iteration or function-evaluation counts are reported.

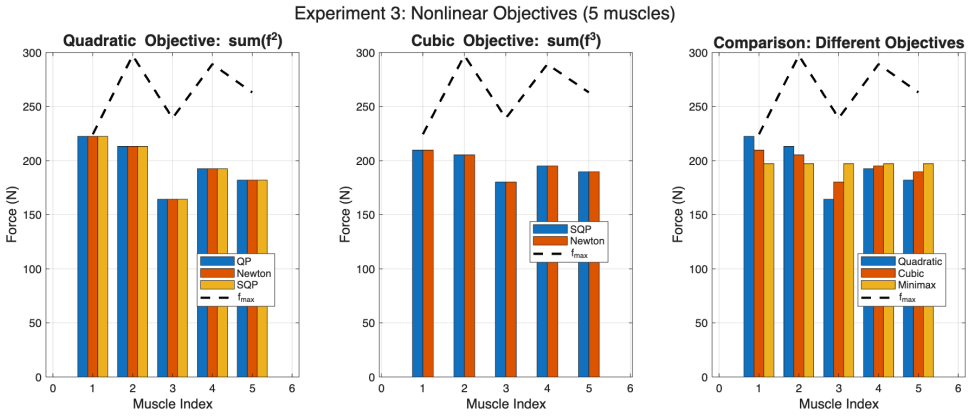

Figure 3: Muscle-force solutions under quadratic, cubic, and minimax objectives. All solvers agree closely for each objective, with nonlinear objectives shifting force toward stronger muscles. The comparison highlights how objective choice alters force distribution patterns.

For both quadratic and cubic objectives, Newton solved the system in a single KKT update, while SQP required six iterations for the quadratic case and sixteen for the cubic due to nonlinear curvature. Only SQP was able to handle the minimax function, equalizing peak force but preventing Newton updates due to non-differentiability, placing an emphasis on smoothness being a requirement.

3.4 Experiment 4: Conditioning Effects

Finally, conditioning was modified by varying moment arm ratios from 1 to 20. In an *ideal* system all moment arms (muscles) have the same length. However, in real anatomy this is not the case and it must be tested if these solvers are robust enough to handle these imbalances. The results are shown in Table 4 and Figure 4 below:

Ratio r_2/r_1	Cond κ(KKT)	QP Time (ms)	SQP Time (ms)	SQP Iters	Newton Time (ms)	Newton Iters
1	2.22e3	5.66	8.12	2	0.52	1
1.5	1.37e3	6.12	12.08	4	0.46	1
2	8.91e2	1.62	4.29	4	0.33	1
3	4.46e2	1.57	6.18	3	0.32	1
5	1.73e2	1.87	11.22	4	0.89	1
7	9.09e1	1.73	9.78	4	0.34	1
10	4.60e1	2.20	11.50	4	0.35	1
15	2.16e1	1.75	16.49	3	5.45	1
20	1.30e1	1.79	7.60	4	0.37	1

Table 4: Solver robustness under ill-conditioned moment arms. Varying moment arm mimics imbalances seen in true anatomy

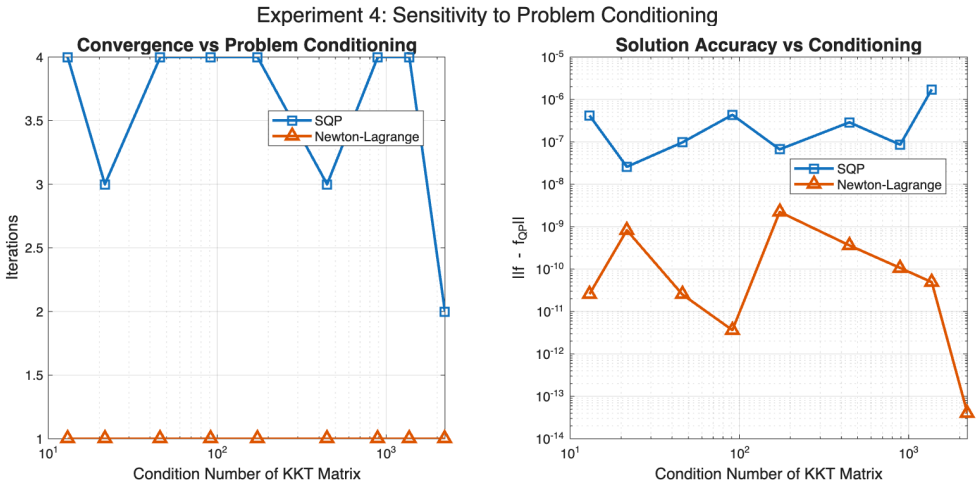

Figure 4: Sensitivity to KKT matrix conditioning. Newton–Lagrange converges in a single iteration and maintains very high accuracy even for poorly conditioned problems. SQP requires more iterations and shows slightly larger error but remains stable.

Condition numbers varied over two orders of magnitude, yet Newton–Lagrange consistently converged in a single step with minimal timing variance. SQP required between two and four iterations, remaining stable but slower due to repeated QP subproblems. This experiment highlights the resilience of Newton updates even under poor geometry, and the dependable robustness of SQP.

4 Discussion

The results across all four experiments highlight how solver choice depends strongly on objective structure and mathematical smoothness. When the cost is quadratic, both QP and Newton–Lagrange converge nearly instantly. SQP, while still accurate, consistently required multiple iterations and higher function evaluation counts. This difference is expected: Newton solves the first–order optimality conditions directly, while SQP refines an approximation through sequential QP solves. Thus, for smooth convex effort minimization, Newton behaves more like a direct solver than an iterative optimizer.

The convergence in exactly two iterations for Newton–Lagrange in Experiment 1 is theoretically significant. Since the objective is quadratic (f^2), the Hessian is constant. In an unconstrained

setting, Newton's method would converge in exactly one step. The second iteration required here implies that the solver needed one step to identify the correct "active set" of constraints (i.e., which muscles are clamped at 0 or $f_{\max}$). Once the active bounds are identified, the problem reduces to a linear system solution, which Newton solves instantly.

The scaling study (Experiment 2) showed that the quadratic problem remains well–conditioned even as the number of muscles increases. Newton again converges immediately because the Hessian is constant and the KKT matrix does not change with iteration. SQP scales reasonably well but grows in cost due to repeated linearizations, suggesting its computational overhead is structural rather than numerical.

It is important to note that while Newton–Lagrange shows $O(1)$ iteration scaling, this hides the underlying linear algebra cost. Each Newton step solves a KKT system of size $(n+1)\times(n+1)$, which generally costs $O(n^3)$ using standard matrix factorization. In contrast, SQP solves a quadratic subproblem at every iteration, which is iteratively expensive. For the system sizes tested ($n \leq 20$), the matrix factorization cost is negligible compared to the overhead of the iterative SQP logic. However, for massive biomechanical models (e.g., full-body simulations with $n > 1000$), the $O(n^3)$ cost of the direct linear solve in Newton's method might eventually become a bottleneck compared to iterative approximation methods. These observations demonstrate that Newton is currently the most efficient method for large smooth systems in this domain, while QP remains a reliable calibrated benchmark.

The nonlinear objective comparison (Experiment 3) reveals the central tradeoff in neuromuscular modeling. Cubic effort amplifies differences between muscles, causing the optimizer to favor uneven recruitment. Both SQP and Newton reach the same solution, but SQP requires significantly more iterations due to nonlinearity of curvature. In contrast, the minimax objective equalizes peak force, a qualitatively different solution reflecting a "load protection" strategy. Newton fails here not because the solution is incorrect, but because the objective is non-smooth and violates the second–order differentiability required to form a Hessian. This illustrates a practical boundary: Newton is fast, but only when the assumed mathematical smoothness holds.

The mathematical failure of Newton's method on the Minimax objective highlights a physiological trade-off. The Minimax objective ($J = \max f_i$) represents a fatigue minimization strategy, ensuring no single muscle is overloaded. However, this safety comes at the cost of mathematical smoothness. The objective function has a discontinuous derivative (a "kink") wherever two muscles share the peak force. Standard gradient-based solvers like Newton–Lagrange rely on curvature information (Hessian) that does not exist at these kinks. SQP succeeds here only because it reformulates the problem using an "epigraph" variable t, effectively lifting the non-smooth problem into a higher-dimensional smooth space ($f_i \leq t$).

Finally, the conditioning study (Experiment 4) demonstrates that neither Newton nor SQP exhibit instability even as moment arm ratios vary by more than an order of magnitude. Newton consistently converged in one step, and SQP remained robust in 2–4 iterations across all cases. This result is valuable because real biomechanics often involve muscles with extreme leverage differences. These experiments indicate that both methods remain stable under such anatomical asymmetry, with Newton providing the best computational efficiency when smoothness assumptions apply.

The surprising stability of the Newton solver despite varying condition numbers can be attributed to the specific structure of the KKT system. The KKT matrix is a "saddle point" system:

$$\begin{bmatrix} H & r \\ r^\top & 0 \end{bmatrix}$$

Even when the moment arms r vary by orders of magnitude (making r ill-conditioned), the strong convexity of the Hessian ($H = 2I$) regularizes the upper-left block. This ensures that the full KKT matrix remains invertible and well-behaved, preventing the numerical instability that might otherwise occur if we were solving the dual problem alone.

5 Conclusion and Future Work

This paper compared three optimization strategies for solving muscle redundancy in the elbow flexion model. For smooth quadratic effort minimization, Newton–Lagrange and QP performed nearly identically in accuracy, with Newton reaching the solution in a single iteration and scaling effortlessly to twenty-muscle systems. SQP, while slower, remained dependable across all trials and was the only method capable of solving the minimax objective, demonstrating superior versatility in non-smooth muscle recruitment modeling.

The most important outcome is that solver selection is not absolute but conditional. When the objective is smooth and twice differentiable, Newton is the fastest and most scalable method tested. However, when modeling fatigue protection, recruitment inequality, or non-smooth physiological constraints, SQP provides the only general-purpose approach. QP serves as a useful reference solver but is restricted to convex quadratic workloads.

It is worth noting the computational scale of this validation. The scaling and conditioning experiments involved thousands of randomized trials (specifically in Experiments 2 and 4) to ensure statistical robustness. This extensive testing verified that the Newton-Lagrange method is not only theoretically sound but numerically stable across a wide range of anatomical parameters and random geometric configurations.

Future extensions of this work may include replacing the minimax objective with a smooth approximation such as a log sum exponent, which would allow Newton to remain applicable while retaining force equalization behavior [1]. Quasi-Newton methods could reduce Hessian construction cost, and warm-started homotopy continuation could improve large-scale convergence [2, 4]. These directions would move the model closer to real neuromotor control, where both smooth effort minimization and load-sharing protection coexist in dynamic balance.

Beyond biological modeling, these findings have direct implications for the control of redundant robotic systems, such as cable-driven manipulators, soft robots, and powered exoskeletons. Just as the CNS must resolve redundancy to distribute loads across muscles, robotic controllers face an identical challenge in allocating actuator forces to achieve a desired end-effector torque. The sub-millisecond convergence times observed in the Newton–Lagrange experiments suggest that this direct solver is well-suited for high-frequency real-time control loops required in active prosthetics. Conversely, the robust performance of SQP on the minimax objective offers a viable strategy for robotic systems where minimizing peak motor current, to prevent overheating or saturation, is more critical than pure computational speed.

References

[1] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, UK, 2004. Used to justify quadratic programming formulation and convexity.

[2] Michael T. Heath. *Scientific Computing*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2018. Chapter 6 used as primary numerical method reference for optimization techniques.

[3] Masaya Hirashima and Tadashi Oya. How Does the Brain Solve Muscle Redundancy? Filling the Gap Between Optimization and Muscle Synergy Hypotheses. *Neuroscience Research*, 104:80–87, 2016. Links optimal control and muscle synergy hypotheses in redundancy resolution.

[4] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, 2006. Reference for SQP, Newton's method, and KKT systems.

[5] Emanuel Todorov. Optimality principles in sensorimotor control. *Nature Neuroscience*, 7(9):907–915, 2004. Foundational work connecting redundancy to optimal control in motor systems.

[6] David A. Winter. *Biomechanics and Motor Control of Human Movement*. John Wiley Sons, Hoboken, NJ, 2009. Standard reference for human joint biomechanics and movement analysis.

[7] Felix E. Zajac. Muscle and Tendon: Properties, Models, Scaling, and Application to Biomechanics and Motor Control. *Critical Reviews in Biomedical Engineering*, 17:359–411, 1989. Classic muscle modeling paper widely used for force and tendon modeling.

A Code Overview and Implementation

The numerical results in this report were generated using a custom object-oriented MATLAB framework. The code was developed in MATLAB R2024b and requires the **Optimization Toolbox** for the baseline `quadprog` and `fmincon` solvers. All timing results reported in this paper were generated on a standard consumer workstation (2019 Apple Macbook Pro, Intel i5 processor, 8 GB RAM) to validate real-time feasibility.

This appendix provides a detailed walkthrough of the three core scripts, highlighting the specific algorithms used to bridge biomechanical modeling with KKT-based optimization. Please note that for reproducibility the seed was set to 42 (`rng(42)`).

A.1 The Optimization Engine: `MuscleOptimizer.m`

The core contribution of this project is the `MuscleOptimizer` class. This script implements the Newton–Lagrange method from scratch.

A.1.1 Core Logic: Direct KKT Solver

The fundamental mathematical operation is implemented in `solve_newton_lagrange`. Unlike iterative approximations, this function explicitly constructs the KKT saddle-point matrix derived in the report.

```
% Newton-Lagrange Loop
for iter = 1:obj.max_iter
    % 1. Evaluate Gradient and Hessian
    [J_val, g, H] = obj.get_objective_functions(objective_type, f);

    % 2. Compute KKT Residuals
    % [grad_L] = [grad_J + lambda * r]
    % [constr] [r'*f - tau_d ]
    grad_L_f = g + lambda * r;
    constraint = r' * f - tau_d;
    residual = [grad_L_f; constraint];

    % 3. Construct the Full KKT System
    % [ H r ] [ delta_f ] = [ -grad_L ]
    % [ r' 0 ] [ delta_lam] [ -constr ]
    KKT_matrix = [H, r; r', 0];
    rhs = -residual;

    % 4. Solve Linear System (Directly)
    delta = KKT_matrix \ rhs;
    delta_f = delta(1:n);
    delta_lambda = delta(n+1);

    % 5. Update Iterates
    f = f + delta_f;
    lambda = lambda + delta_lambda;
end
```

A.1.2 Globalization: Backtracking Line Search

To ensure global convergence, the solver employs a backtracking line search. This prevents the Newton step from overshooting in non-quadratic regions (Experiment 3):

```
function alpha = line_search(obj, f, lambda, df, dlam, J, grad_J, r, tau_d)
    % Backtracking line search for Newton-Lagrange
```

```
    c = 1e-4; rho = 0.5;

    % Current merit (residual norm squared)
    g0 = grad_J(f);
    res0 = [g0 + lambda*r; r'*f - tau_d];
    phi0 = norm(res0)^2;

    alpha = 1.0;
    for k = 1:25
        f_new = f + alpha * df;
        lambda_new = lambda + alpha * dlam;

        g_new = grad_J(f_new);
        res_new = [g_new + lambda_new*r; r'*f_new - tau_d];
        phi_new = norm(res_new)^2;

        if phi_new <= phi0 - c * alpha * phi0 % Sufficient decrease
            break;
        end
        alpha = rho * alpha;
    end
end
```

A.1.3 Handling Bounds: The Active-Set Strategy

To enforce $0 \leq f \leq f_{\max}$, the method `solve_newton_lagrange_activeset` wraps the direct solver. It partitions muscles into "free" and "active" sets. The critical logic involves checking the Lagrange multipliers (μ) to see if a constraint should be released:

```
% Check if active constraints should be released based on multiplier signs
% mu_lb = g + lambda*r. If mu < 0, the constraint is pulling away from the bound.
for i = active_lb(:)'
    if mu_lb(i) < -obj.tolerance
        active_lb = setdiff(active_lb, i); % Release muscle from 0 bound
    end
end
for i = active_ub(:)'
    if mu_ub(i) < -obj.tolerance
        active_ub = setdiff(active_ub, i); % Release muscle from f_max bound
    end
end
```

A.2 The Biomechanical Model: `n_muscles.m`

This class standardizes the problem definition. It ensures that regardless of which solver is used (QP, SQP, or Newton), the physical parameters ($r, f_{\max}, \tau_d$) are identical.

A.2.1 Wrapper for SQP and Minimax

A key feature of this script is the handling of the non-smooth Minimax objective ($J = \max f_i$). Since gradients are undefined at the "kink" where forces match, the code automatically reformulates the problem using an epigraph variable t before passing it to `fmincon`:

```
case 'minimax'
    % Reformulate: min t s.t. f_i <= t for all i
    obj_fun = @(x) x(end); % Minimize the slack variable t
    f0 = [f0; max(f0)]; % Augment state vector: [f; t]

    % Add inequality constraints: f_i - t <= 0
    % This lifts the non-smooth problem into a smooth higher-dimensional space
    A_ineq = [eye(obj.n), -ones(obj.n, 1)];
```

```
    b_ineq = zeros(obj.n, 1);
```

A.3 Driver Script and Experiments: `MuscleProblem.m`

The main driver script orchestrates the four experiments presented in the results section. Below is a walkthrough of the specific code used to generate the data for each experiment.

Experiment 1: Convergence vs. Initial Guess

This section tests robustness by looping through a pre-defined list of starting points, ranging from "Low uniform" to "Above bounds".

```
% Define diverse initial guesses
initial_guesses = {
    [50; 50], 'Low uniform';
    [250; 250], 'High uniform';
    [300; 300], 'Above bounds';
    % ... (8 total guesses)
};

for i = 1:n_guesses
    f0 = initial_guesses{i, 1};

    % Track full history to plot convergence trajectory
    [f_newton, ~, hist] = opt1.solve_newton_lagrange_activeset(f0, 'quadratic');

    % Store iteration count and residual history
    exp1_newton_iters(i) = hist.iterations;
    exp1_newton_residuals{i} = hist.kkt_residual;
end
```

Experiment 2: Scaling Behavior

This loop explicitly measures computational cost as n increases. Crucially, it generates a "hard" problem (where bounds are active) to stress-test the active-set logic.

```
n_values = 2:2:20; % Scale from 2 to 20 muscles

for i = 1:length(n_values)
    n = n_values(i);
    prob = n_muscles(n, 0, 'hard'); % Create constrained problem

    % Measure QP Baseline
    tic; prob.solve_quadratic(); exp2_qp_time(i) = toc;

    % Measure Newton-Lagrange
    tic;
    [f_newton, ~] = opt.solve_newton_lagrange_activeset(f0, 'quadratic');
    exp2_newton_time(i) = toc;

    % Validate Accuracy against QP baseline
    exp2_accuracy_newton(i) = norm(f_newton - f_qp);
end
```

Experiment 3: Nonlinear Objectives

This experiment compares solver behavior across different cost functions. The code iterates through objective types, handling the Minimax exception separately.

```
objectives = {'quadratic', 'cubic'};

for j = 1:length(objectives)
    obj_type = objectives{j};

    % Compare SQP vs Newton on smooth objectives
    [f_sqp, ~] = prob3.solve_fmincon(obj_type, f0);
    [f_newton, ~] = opt3.solve_newton_lagrange_activeset(f0, obj_type);
end

% Special handling for Minimax (Newton cannot solve this)
[f_minimax, ~] = prob3.solve_fmincon('minimax', f0);
% Note: Newton call is skipped here due to non-differentiability
```

Experiment 4: Conditioning Analysis

This final experiment explicitly manipulates the geometry of the muscles. By varying the moment arm ratio, we force the KKT matrix to become ill-conditioned to test numerical stability.

```
ratios = [1, 2, 5, 10, 20];

for i = 1:length(ratios)
    % Create ill-conditioned geometry
    prob = n_muscles.create_ill_conditioned(ratios(i));

    % Explicitly calculate Condition Number of the KKT matrix
    KKT = [2*eye(n), prob.r; prob.r', 0];
    exp4_cond(i) = cond(KKT);

    % Run solver to check for divergence
    [f_newton, ~] = opt.solve_newton_lagrange_activeset(f0, 'quadratic');
end
```