Neurodynamic Optimization: Models and Applications

Jun Wang

`jwang@mae.cuhk.edu.hk`

Department of Mechanical & Automation Engineering
The Chinese University of Hong Kong
Shatin, New Territories, Hong Kong
http://www.mae.cuhk.edu.hk/ ̃jwang

Outline

- Introduction

Outline

- Introduction
- Problem Formulation

Outline

- Introduction
- Problem Formulation
- Dynamic Optimization

Outline

- Introduction
- Problem Formulation
- Dynamic Optimization
- Exiting Approaches

Outline

- Introduction
- Problem Formulation
- Dynamic Optimization
- Exiting Approaches
- Neurodynamic Models

Outline

- Introduction
- Problem Formulation
- Dynamic Optimization
- Exiting Approaches
- Neurodynamic Models
- Design Procedure

Outline

- Introduction
- Problem Formulation
- Dynamic Optimization
- Exiting Approaches
- Neurodynamic Models
- Design Procedure
- Winners Take All

Outline

- Introduction
- Problem Formulation
- Dynamic Optimization
- Exiting Approaches
- Neurodynamic Models
- Design Procedure
- Winners Take All
- Linear Assignment

Outline

- Introduction
- Problem Formulation
- Dynamic Optimization
- Exiting Approaches
- Neurodynamic Models
- Design Procedure
- Winners Take All
- Linear Assignment
- Machine Learning

Outline

- Introduction
- Problem Formulation
- Dynamic Optimization
- Exiting Approaches
- Neurodynamic Models
- Design Procedure
- Winners Take All
- Linear Assignment
- Machine Learning
- Robot Control

Outline

- Introduction
- Problem Formulation
- Dynamic Optimization
- Exiting Approaches
- Neurodynamic Models
- Design Procedure
- Winners Take All
- Linear Assignment
- Machine Learning
- Robot Control
- Concluding Remarks

Outline

- Introduction
- Problem Formulation
- Dynamic Optimization
- Exiting Approaches
- Neurodynamic Models
- Design Procedure
- Winners Take All
- Linear Assignment
- Machine Learning
- Robot Control
- Concluding Remarks
- Future Works

Introduction

Optimization is ubiquitous in nature and society.

Introduction

Optimization is ubiquitous in nature and society.

Optimization arises in a wide variety of scientific problems.

Introduction

Optimization is ubiquitous in nature and society.

Optimization arises in a wide variety of scientific problems.

Optimization is an important tool for design, planning, control, operation, and management of engineering systems.

Problem Formulation

Consider a general optimization problem:

$$\begin{aligned} \mathrm{OP}_1: \text{ Minimize } \quad & f(x) \\ \text{subject to } \quad & c(x) \le 0, \\ & d(x) = 0, \end{aligned}$$

where $x \in \Re^n$ is the vector of decision variables, $f(x)$ is an objective function, $c(x) = [c_1(x), \ldots, c_m(x)]^T$ is a vector-valued function, and
$d(x) = [d_1(x), \ldots, d_p(x)]^T$ a vector-valued function.

Problem Formulation

Consider a general optimization problem:

$$\begin{aligned} \mathrm{OP}_1 : \ \text{Minimize} \quad & f(x) \\ \text{subject to} \quad & c(x) \leq 0, \\ & d(x) = 0, \end{aligned}$$

where $x \in \Re^n$ is the vector of decision variables, $f(x)$ is an objective function, $c(x) = [c_1(x), \ldots, c_m(x)]^T$ is a vector-valued function, and $d(x) = [d_1(x), \ldots, d_p(x)]^T$ a vector-valued function. If $f(x)$ and $c(x)$ are convex and $d(x)$ is affine, then OP is a convex programming problem CP. Otherwise, it is a nonconvex program.

Quadratic Programs

$$
\begin{aligned}
\mathrm{QP}_1: \ \text{minimize} \quad & \frac{1}{2}x^T Q x + q^T x \\
\text{subject to} \quad & Ax = b, \\
& l \leq Cx \leq h,
\end{aligned}
$$

where $Q \in \Re^{n\times n}$, $q \in \Re^n, A \in \Re^{m\times n}$,
$b \in \Re^m, C \in \Re^{n\times n}, l \in \Re^n, h \in \Re^n$.

Quadratic Programs

$$\begin{aligned} \mathrm{QP}_1: \ \text{minimize} \quad & \frac{1}{2}x^TQx + q^Tx \\ \text{subject to} \quad & Ax = b, \\ & l \leq Cx \leq h, \end{aligned}$$

where $Q \in \Re^{n\times n}$, $q \in \Re^n$, $A \in \Re^{m\times n}$, $b \in \Re^m, C \in \Re^{n\times n}, l \in \Re^n, h \in \Re^n$.
When $l = 0, h = \infty, C = I$, QP_1 becomes a standard QP:

$$\begin{aligned} \mathrm{QP}_2: \ \text{minimize} \quad & \frac{1}{2}x^TQx + q^Tx \\ \text{subject to} \quad & Ax = b, x \geq 0 \end{aligned}$$

Linear Programs

When $Q = 0$, and $C = I$, QP_1 becomes a linear program with bound constraints:

$$\begin{aligned} \mathrm{LP}_1: \ \text{minimize} \quad & q^T x \\ \text{subject to} \quad & Ax = b, \\ & l \leq x \leq h \end{aligned}$$

Linear Programs

When $Q = 0$, and $C = I$, QP_1 becomes a linear program with bound constraints:

$$\begin{aligned} \text{LP}_1: \ \text{minimize} \quad & q^T x \\ \text{subject to} \quad & Ax = b, \\ & l \le x \le h \end{aligned}$$

In addition, when $l = 0$, and $h = +\infty$, LP_1 becomes a standard linear program:

$$\begin{aligned} \text{LP}_2: \ \text{minimize} \quad & q^T x \\ \text{subject to} \quad & Ax = b, \\ & x \ge 0 \end{aligned}$$

Dynamic Optimization

In many applications (e.g., online pattern recognition, robot motion control, and onboard signal processing), real-time solutions to optimization problems are necessary or desirable.

Dynamic Optimization

In many applications (e.g., online pattern recognition, robot motion control, and onboard signal processing), real-time solutions to optimization problems are necessary or desirable.
For such applications, classical optimization techniques may not be competent due to the problem dimensionality and stringent requirement on computational time.

Dynamic Optimization

- In many applications (e.g., online pattern recognition, robot motion control, and onboard signal processing), real-time solutions to optimization problems are necessary or desirable.
- For such applications, classical optimization techniques may not be competent due to the problem dimensionality and stringent requirement on computational time.
- It is computationally challenging when optimization procedures have to be performed in real time to optimize the performance of dynamical systems.

Dynamic Optimization

- In many applications (e.g., online pattern recognition, robot motion control, and onboard signal processing), real-time solutions to optimization problems are necessary or desirable.
- For such applications, classical optimization techniques may not be competent due to the problem dimensionality and stringent requirement on computational time.
- It is computationally challenging when optimization procedures have to be performed in real time to optimize the performance of dynamical systems.
- One very promising approach to dynamic optimization is to apply artificial neural networks.

Neurodynamic Optimization

Because of the inherent nature of parallel and distributed information processing in neural networks, the convergence rate of the solution process is not decreasing as the size of the problem increases.

Neurodynamic Optimization

Because of the inherent nature of parallel and distributed information processing in neural networks, the convergence rate of the solution process is not decreasing as the size of the problem increases.

Neural networks can be implemented physically in designated hardware such as ASICs where optimization is carried out in a truly parallel and distributed manner.

Neurodynamic Optimization

Because of the inherent nature of parallel and distributed information processing in neural networks, the convergence rate of the solution process is not decreasing as the size of the problem increases.

Neural networks can be implemented physically in designated hardware such as ASICs where optimization is carried out in a truly parallel and distributed manner.

This feature is particularly desirable for dynamic optimization in decentralized decision-making situations.

Existing Approaches

In their seminal work, Tank and Hopfield (1985, 1986) applied the Hopfield networks for solving a linear program and the traveling salesman problem.

Existing Approaches

In their seminal work, Tank and Hopfield (1985, 1986) applied the Hopfield networks for solving a linear program and the traveling salesman problem.

Kennedy and Chua (1988) developed a neural network for nonlinear programming, which contains finite penalty parameters and thus its equilibrium points correspond to approximate optimal solutions only.

Existing Approaches

In their seminal work, Tank and Hopfield (1985, 1986) applied the Hopfield networks for solving a linear program and the traveling salesman problem.

Kennedy and Chua (1988) developed a neural network for nonlinear programming, which contains finite penalty parameters and thus its equilibrium points correspond to approximate optimal solutions only.

The two-phase optimization networks by Maa and Shanblatt (1992).

Existing Approaches

In their seminal work, Tank and Hopfield (1985, 1986) applied the Hopfield networks for solving a linear program and the traveling salesman problem.

Kennedy and Chua (1988) developed a neural network for nonlinear programming, which contains finite penalty parameters and thus its equilibrium points correspond to approximate optimal solutions only.

The two-phase optimization networks by Maa and Shanblatt (1992).

The Lagrangian networks for quadratic programming by Zhang and Constantinides (1992) and Zhang, et al. (1992).

Existing Approaches (cont'd)

A recurrent neural network for quadratic optimization with bounded variables only by Bouzerdoum and Pattison (1993).

Existing Approaches (cont'd)

A recurrent neural network for quadratic optimization with bounded variables only by Bouzerdoum and Pattison (1993).

The deterministic annealing network for linear and convex programming by Wang (1993, 1994).

Existing Approaches (cont'd)

A recurrent neural network for quadratic optimization with bounded variables only by Bouzerdoum and Pattison (1993).

The deterministic annealing network for linear and convex programming by Wang (1993, 1994).

The primal-dual networks for linear and quadratic programming by Xia (1996, 1997).

Existing Approaches (cont'd)

A recurrent neural network for quadratic optimization with bounded variables only by Bouzerdoum and Pattison (1993).

The deterministic annealing network for linear and convex programming by Wang (1993, 1994).

The primal-dual networks for linear and quadratic programming by Xia (1996, 1997).

The projection networks for solving projection equations, constrained optimization, etc by Xia and Wang (1998, 2002, 2004) and Liang and Wang (2000).

Existing Approaches (cont'd)

The dual networks for quadratic programming by Xia and Wang (2001), Zhang and Wang (2002).

Existing Approaches (cont'd)

The dual networks for quadratic programming by Xia and Wang (2001), Zhang and Wang (2002).

A two-layer network for convex programming subject to nonlinear inequality constraints by Xia and Wang (2004).

Existing Approaches (cont'd)

The dual networks for quadratic programming by Xia and Wang (2001), Zhang and Wang (2002).

A two-layer network for convex programming subject to nonlinear inequality constraints by Xia and Wang (2004).

A simplified dual network for quadratic programming by Liu and Wang (2006)

Existing Approaches (cont'd)

- The dual networks for quadratic programming by Xia and Wang (2001), Zhang and Wang (2002).
- A two-layer network for convex programming subject to nonlinear inequality constraints by Xia and Wang (2004).
- A simplified dual network for quadratic programming by Liu and Wang (2006)
- Two one-layer networks with discontinuous activation functions for linear and quadratic programming by Liu and Wang (2007).

General Design Procedure

A design procedure begins with a given objective function and constraint(s).

The next step involves the derivation of a neurodynamic equation which prescribes the motion of the activation states of the neural network.

General Design Procedure

A design procedure begins with a given objective function and constraint(s).

The next step involves the derivation of a neurodynamic equation which prescribes the motion of the activation states of the neural network.

The derivation of a neurodynamic equation is crucial for success of the neural network approach to optimization.

General Design Procedure

A design procedure begins with a given objective function and constraint(s).

The next step involves the derivation of a neurodynamic equation which prescribes the motion of the activation states of the neural network.

The derivation of a neurodynamic equation is crucial for success of the neural network approach to optimization.

A properly derived neurodynamic equation can ensure that the state of neural network reaches an equilibrium and the equilibrium satisfies the constraints and optimizes the objective function.

General Design Procedure (cont’d)

In general, there are two approaches to design neurodynamic equations.

General Design Procedure (cont'd)

In general, there are two approaches to design neurodynamic equations.

The first approach is based on an defined energy function.

General Design Procedure (cont'd)

In general, there are two approaches to design neurodynamic equations.

The first approach is based on an defined energy function.

The second approach is based on the existing optimality conditions.

General Design Procedure

The first approach starts with the formulation of an energy function based on a given objective function and constraints

General Design Procedure

The first approach starts with the formulation of an energy function based on a given objective function and constraints

It plays an important role in neurodynamic optimization.

General Design Procedure

The first approach starts with the formulation of an energy function based on a given objective function and constraints

It plays an important role in neurodynamic optimization.

Ideally, the minimum of a formulated energy function corresponds to the optimal solution of the original optimization problem.

General Design Procedure

The first approach starts with the formulation of an energy function based on a given objective function and constraints

It plays an important role in neurodynamic optimization.

Ideally, the minimum of a formulated energy function corresponds to the optimal solution of the original optimization problem.

For constrained optimization, the minimum of the energy function has to satisfy a set of constraints.

General Design Procedure (cont'd)

The majority of the existing approaches formulates an energy function by incorporating objective function and constraints through functional transformation and numerical weighting.

General Design Procedure (cont'd)

The majority of the existing approaches formulates an energy function by incorporating objective function and constraints through functional transformation and numerical weighting.

Functional transformation is usually used to convert constraints to a penalty function to penalize the violation of constraints; e.g., $p(x) = \sum_{i=1}^{m} \{[-c_i(x)]^+\}^2 + \sum_{j=1}^{p} [d_j(x)]^2$, where $[y]^+ = \max\{0, y\}$.

General Design Procedure (cont'd)

The majority of the existing approaches formulates an energy function by incorporating objective function and constraints through functional transformation and numerical weighting.

Functional transformation is usually used to convert constraints to a penalty function to penalize the violation of constraints; e.g., $p(x) = \sum_{i=1}^{m} \{[-c_i(x)]^+\}^2 + \sum_{j=1}^{p} [d_j(x)]^2$, where $[y]^+ = \max\{0, y\}$.

Numerical weighting is often used to balance constraint satisfaction and objective optimization; e.g., $E(x) = f(x) + wp(x)$ where w is a positive weight.

General Design Procedure (cont'd)

The majority of the existing approaches formulates an energy function by incorporating objective function and constraints through functional transformation and numerical weighting.

Functional transformation is usually used to convert constraints to a penalty function to penalize the violation of constraints; e.g., $p(x) = \sum_{i=1}^{m}\{[-c_i(x)]^+\}^2 + \sum_{j=1}^{p}[d_j(x)]^2$, where $[y]^+ = \max\{0, y\}$.

Numerical weighting is often used to balance constraint satisfaction and objective optimization; e.g., $E(x) = f(x) + wp(x)$ where w is a positive weight.

General Design Procedure (cont'd)

Neurodynamic equations are usually derived as the negative gradient of the energy function:

$$\frac{dx(t)}{dt} \propto -\nabla E(x(t)).$$

General Design Procedure (cont'd)

Neurodynamic equations are usually derived as the negative gradient of the energy function:

$$\frac{dx(t)}{dt} \propto -\nabla E(x(t)).$$

If the enegery function is bounded blow, the stability of the neurodynamics can be ensured.

General Design Procedure (cont'd)

Second approach: Neurodynamic equations of some recent neural networks for optimization are derived based on optimality conditions (e.g., Karush-Kuhn-Tucker condition) and projection equations.

General Design Procedure (cont'd)

Second approach: Neurodynamic equations of some recent neural networks for optimization are derived based on optimality conditions (e.g., Karush-Kuhn-Tucker condition) and projection equations.

Stability analysis is needed explicitly to ensure the that resulting neural network is stable.

General Design Procedure (cont'd)

Second approach: Neurodynamic equations of some recent neural networks for optimization are derived based on optimality conditions (e.g., Karush-Kuhn-Tucker condition) and projection equations.

Stability analysis is needed explicitly to ensure the that resulting neural network is stable.

All equilibria of a stable neural network satisfy the optimality condition.

General Design Procedure (cont'd)

Second approach: Neurodynamic equations of some recent neural networks for optimization are derived based on optimality conditions (e.g., Karush-Kuhn-Tucker condition) and projection equations.

Stability analysis is needed explicitly to ensure the that resulting neural network is stable.

All equilibria of a stable neural network satisfy the optimality condition.

If the problem is a convex program, an equilibrium point represents an optimal solution.

General Design Procedure (cont'd)

The next step is to determine the architecture of the neural network in terms of the neurons and connections based on the derived dynamical equation.

General Design Procedure (cont'd)

The next step is to determine the architecture of the neural network in terms of the neurons and connections based on the derived dynamical equation.

An activation function models important characteristics of a neuron.

General Design Procedure (cont'd)

The next step is to determine the architecture of the neural network in terms of the neurons and connections based on the derived dynamical equation.

An activation function models important characteristics of a neuron.

The range of an activation function usually prescribes the the state space of the neural network.

General Design Procedure (cont'd)

The next step is to determine the architecture of the neural network in terms of the neurons and connections based on the derived dynamical equation.

An activation function models important characteristics of a neuron.

The range of an activation function usually prescribes the the state space of the neural network.

The activation function depends on the feasible region delimited by the constraints.

General Design Procedure (cont'd)

The next step is to determine the architecture of the neural network in terms of the neurons and connections based on the derived dynamical equation.

An activation function models important characteristics of a neuron.

The range of an activation function usually prescribes the the state space of the neural network.

The activation function depends on the feasible region delimited by the constraints.

Specifically, it is necessary for the state space to include the feasible region.

General Design Procedure (cont'd)

Any explicit bounds on decision variables can be realized by properly selecting the range of activation functions.

General Design Procedure (cont'd)

Any explicit bounds on decision variables can be realized by properly selecting the range of activation functions.

If the gradient-based method is adopted in deriving the dynamical equation, then the convex energy function results in an increasing activation function.

General Design Procedure (cont'd)

Any explicit bounds on decision variables can be realized by properly selecting the range of activation functions.

If the gradient-based method is adopted in deriving the dynamical equation, then the convex energy function results in an increasing activation function.

Precisely, if the steepest descent method is used, the activation function is equal to the derivative of the energy function.

General Design Procedure (cont'd)

Any explicit bounds on decision variables can be realized by properly selecting the range of activation functions.

If the gradient-based method is adopted in deriving the dynamical equation, then the convex energy function results in an increasing activation function.

Precisely, if the steepest descent method is used, the activation function is equal to the derivative of the energy function.

The last step is usually devoted to simulation to test the performance of the neural network numerically or physically.

Kennedy-Chua Network

The Kennedy-Chua network for solving OP[a]:

$$\epsilon\frac{dx}{dt} = -\nabla f(x) - s\cdot h(c(x))^T\nabla c(x) - s\cdot d(x)^T\nabla d(x),$$

where $\epsilon > 0$ is a scaling parameter, $x \in \Re^n$ is the state vector, $s > 0$ is a penalty parameter, $h(r) = (h(r_1), ..., h(r_n))^T$, and $h(r_i) = \max\{0, r_i\}$.

[a] M. P. Kennedy and L. O. Chua, "Neural networks for nonlinear programming," *IEEE Transactions on Circuits and Systems*, vol. 35, no. 5, pp. 554–562, May 1988.

Kennedy-Chua Network

The Kennedy-Chua network for solving OP[a]:

$$\epsilon \frac{dx}{dt} = -\nabla f(x) - s \cdot h(c(x))^T \nabla c(x) - s \cdot d(x)^T \nabla d(x),$$

where $\epsilon > 0$ is a scaling parameter, $x \in \Re^n$ is the state vector, $s > 0$ is a penalty parameter, $h(r) = (h(r_1), ..., h(r_n))^T$, and $h(r_i) = \max\{0, r_i\}$.

With a finite penalty parameter s, the network is globally convergent to a near-optimal solution to an OP even though CP.

[a] M. P. Kennedy and L. O. Chua, "Neural networks for nonlinear programming," *IEEE Transactions on Circuits and Systems*, vol. 35, no. 5, pp. 554–562, May 1988.

Deterministic Annealing Network

The deterministic annealing network for solving OP[a]:

$$\epsilon \frac{dx}{dt} = -T(t)\nabla f(x) - h(c(x))^T \nabla c(x) - d(x)^T \nabla d(x),$$

where $\epsilon > 0$ is a scaling parameter, $x \in \Re^n$ is the state vector, $T(t) \geq 0$ is a temperature parameter, $h(r) = (h(r_1), ..., h(r_n))^T$, and $h(r_i) = \max\{0, r_i\}$.

[a] J. Wang, "A deterministic annealing neural network for convex programming," *Neural Networks*, vol. 7, no. 4, pp. 629-641, 1994.

Deterministic Annealing Network

The deterministic annealing network for solving OP[a]:

$$\epsilon \frac{dx}{dt} = -T(t)\nabla f(x) - h(c(x))^T \nabla c(x) - d(x)^T \nabla d(x),$$

where $\epsilon > 0$ is a scaling parameter, $x \in \Re^n$ is the state vector, $T(t) \geq 0$ is a temperature parameter, $h(r) = (h(r_1), ..., h(r_n))^T$, and $h(r_i) = \max\{0, r_i\}$. If $\lim_{t\to\infty} T(t) = 0$, then the network is globally convergent to a feasible near-optimal solution to CP.

[a] J. Wang, "A deterministic annealing neural network for convex programming," *Neural Networks*, vol. 7, no. 4, pp. 629-641, 1994.

Deterministic Annealing Network

The deterministic annealing network for solving OP[a]:

$$\epsilon\frac{dx}{dt} = -T(t)\nabla f(x) - h(c(x))^T\nabla c(x) - d(x)^T\nabla d(x),$$

where $\epsilon > 0$ is a scaling parameter, $x \in \Re^n$ is the state vector, $T(t) \geq 0$ is a temperature parameter, $h(r) = (h(r_1), ..., h(r_n))^T$, and $h(r_i) = \max\{0, r_i\}$. If $\lim_{t\to\infty} T(t) = 0$, then the network is globally convergent to a feasible near-optimal solution to CP. If $T(t)$ decreases gradually to 0, then the network is globally convergent to an optimal solution to CP.

[a] J. Wang, "A deterministic annealing neural network for convex programming," *Neural Networks*, vol. 7, no. 4, pp. 629-641, 1994.

Primal-Dual Network

The primal-dual network for solving LP_2[a]:

$$\epsilon\frac{dx}{dt} = -(q^T x - b^T y)q - A^T(Ax - b) + x^+,$$
$$\epsilon\frac{dy}{dt} = -(q^T x - b^T y)b,$$

where $\epsilon > 0$ is a scaling parameter, $x \in \Re^n$ is the primal state vector, $y \in \Re^m$ is the dual (hidden) state vector, $x^+ = (x_1^+), ..., x_n^+)^T$, and $x_i^+ = \max\{0, x_i\}$.

[a] Y. Xia, "A new neural network for solving linear and quadratic programming problems," *IEEE Transactions on Neural Networks*, vol. 7, no. 6, 1544-1548, 1996.

Primal-Dual Network

The primal-dual network for solving LP_2[a]:

$$\epsilon \frac{dx}{dt} = -(q^T x - b^T y)q - A^T(Ax - b) + x^+,$$
$$\epsilon \frac{dy}{dt} = -(q^T x - b^T y)b,$$

where $\epsilon > 0$ is a scaling parameter, $x \in \Re^n$ is the primal state vector, $y \in \Re^m$ is the dual (hidden) state vector, $x^+ = (x_1^+), ..., x_n^+)^T$, and $x_i^+ = \max\{0, x_i\}$. The network is globally convergent to an optimal solution to LP_1.

[a] Y. Xia, "A new neural network for solving linear and quadratic programming problems," *IEEE Transactions on Neural Networks*, vol. 7, no. 6, 1544-1548, 1996.

Lagrangian Network for QP

If $C = 0$ in QP$_1$:

$$\epsilon \frac{d}{dt} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -Qx(t) - A^T y(t) - q, \\ Ax - b \end{pmatrix}.$$

where $\epsilon > 0$, $x \in \Re^n, y \in \Re^m$.

It is globally exponentially convergent to the optimal solution[a].

---

[a] J. Wang, Q. Hu, and D. Jiang, "A Lagrangian network for kinematic control of redundant robot manipulators," *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 1123-1132, 1999.

Projection Network

A recurrent neural network called the projection network was developed for optimization with bound constraints only[a] [b]

$$\epsilon \frac{dx}{dt} = -x + g(x - \nabla f(x)).$$

[a] Y.S. Xia and J. Wang, "On the stability of globally projected dynamic systems," *J. of Optimization Theory and Applications*, vol. 106, no. 1, pp. 129-150, 2000.

[b] Y.S. Xia, H. Leung, and J. Wang, "A projection neural network and its application to constrained optimization problems," *IEEE Trans. Circuits and Systems I*, vol. 49, no. 4, pp. 447-458, 2002.

Convex Program

Consider a convex programming problem without equality constraints:

$$\begin{aligned} \mathrm{CP}_2: \quad & \text{minimize} \;\; f(x) \\ & \text{subject to} \;\; c(x) \leq 0, x \geq 0 \end{aligned}$$

where $f(x)$ and $c(x) = (c_1(x), ..., c_m(x))^T$ are convex, $m \leq n$.

Equivalent Reformulation

The Karush-Kuhn-Tucker (KKT) conditions for CP:

$$y \geq 0, \quad c(x) \leq 0, x \geq 0$$
$$\nabla f(x) + \nabla c(x) y \geq 0, \quad y^T c(x) = 0$$

According to the projection method, the KKT condition is equivalent to:

$$\begin{cases} h(x - \alpha(\nabla f(x) + \nabla c(x) y)) - x = 0 \\ h(y + \alpha c(x)) - y = 0, \end{cases} \tag{1}$$

where $h(r) = (h(r_1), ..., h(r_n))^T$,
$h(r_i) = \max\{0, r_i\}$, and α is any positive constant.

Two-layer network

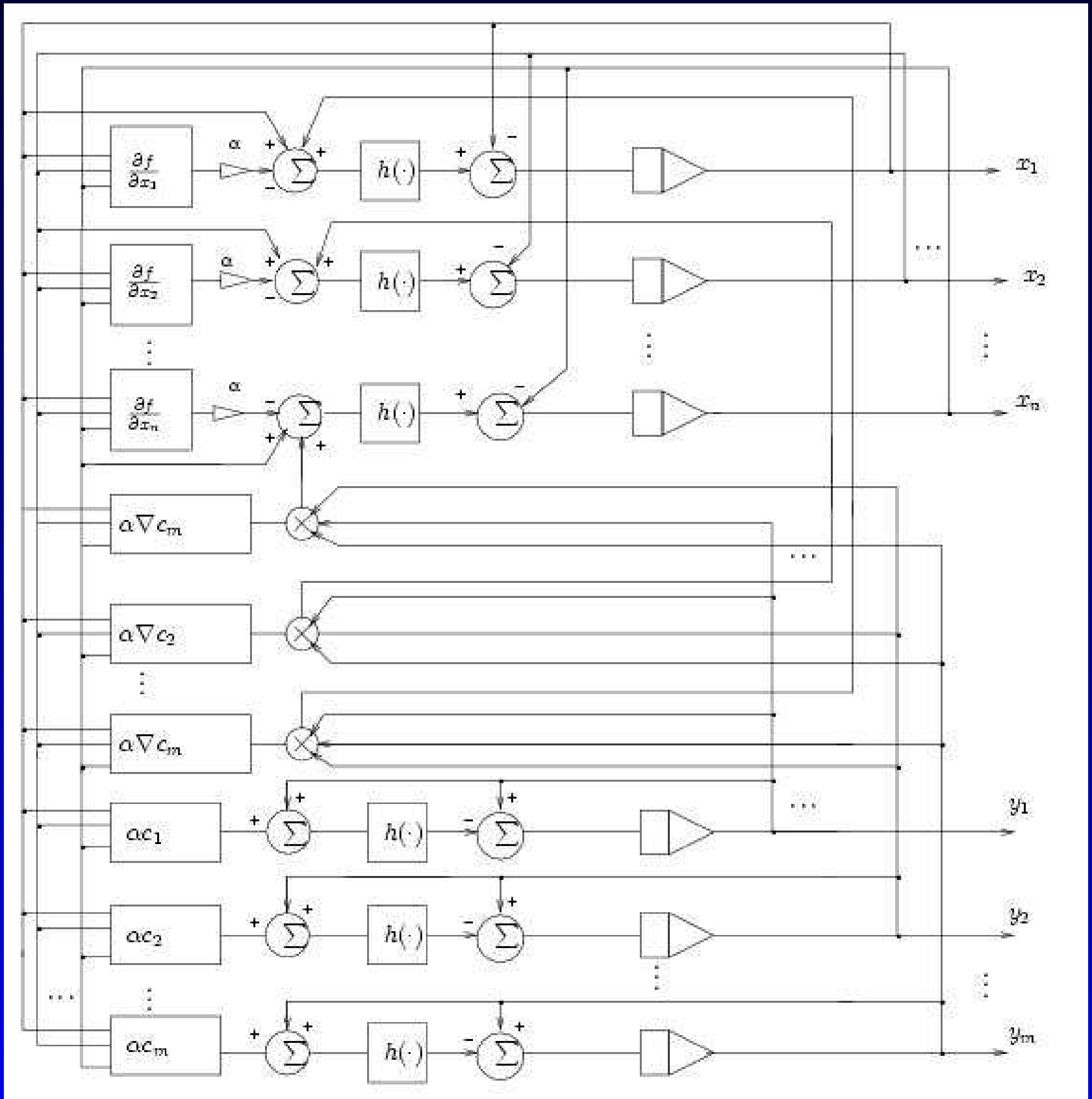
Based on an equivalent formulation, a two-layer neural network was developed for OP [a] is then given by

$$\epsilon \frac{d}{dt} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -x + g(x - (\nabla f(x) + \nabla c(x) y)) \\ -y + h(y + c(x)) \end{pmatrix},$$

where $x \in \Re^n$ and $y \in \Re^m$.

[a] Y.S. Xia and J. Wang, "A recurrent neural network for nonlinear convex optimization subject to nonlinear inequality constraints," *IEEE Trans. Circuits and Systems I*, vol. 51, no. 7, pp. 1385-1394, 2004.

Model Architecture



Convergence Results

For any $x(t_0)$ and $y(t_0)$, $x(t)$ and $y(t)$ are continuous and unique. $u(t) \geq 0$ if $u(t_0) \geq 0$. The equilibrium point solves CP_2.

If $\nabla^2 f(x) + \sum_{i=1}^{n} y_i \nabla^2 c_i(x)$ is positive definite on $\Re_+^{n+m}$, then the two-layer neural network is globally convergent to the KKT point (x^*, y^*), where x^* is the optimal solution to CP_2.

Two-layer Neural Network for QP

If $C = I$ in QP_1, let $\alpha = 1$ in the two-layer neural network for CP:

$$\epsilon \frac{d}{dt} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -x + g((I - Q)x + A^T y - q) \\ -Ax + b \end{pmatrix}.$$

where $\epsilon > 0$, $x \in \Re^n, y \in \Re^m$,
$g(x) = [g(x_1), ..., g(x_n)]^T$

$$g(x_i) = \begin{cases} l_i & x_i < l_i \\ u_i & l_i \le x_i \le h_i \\ h_i & x_i > h_i. \end{cases}$$

It is globally asymptotically convergent to the optimal solution.

Illustrative Example

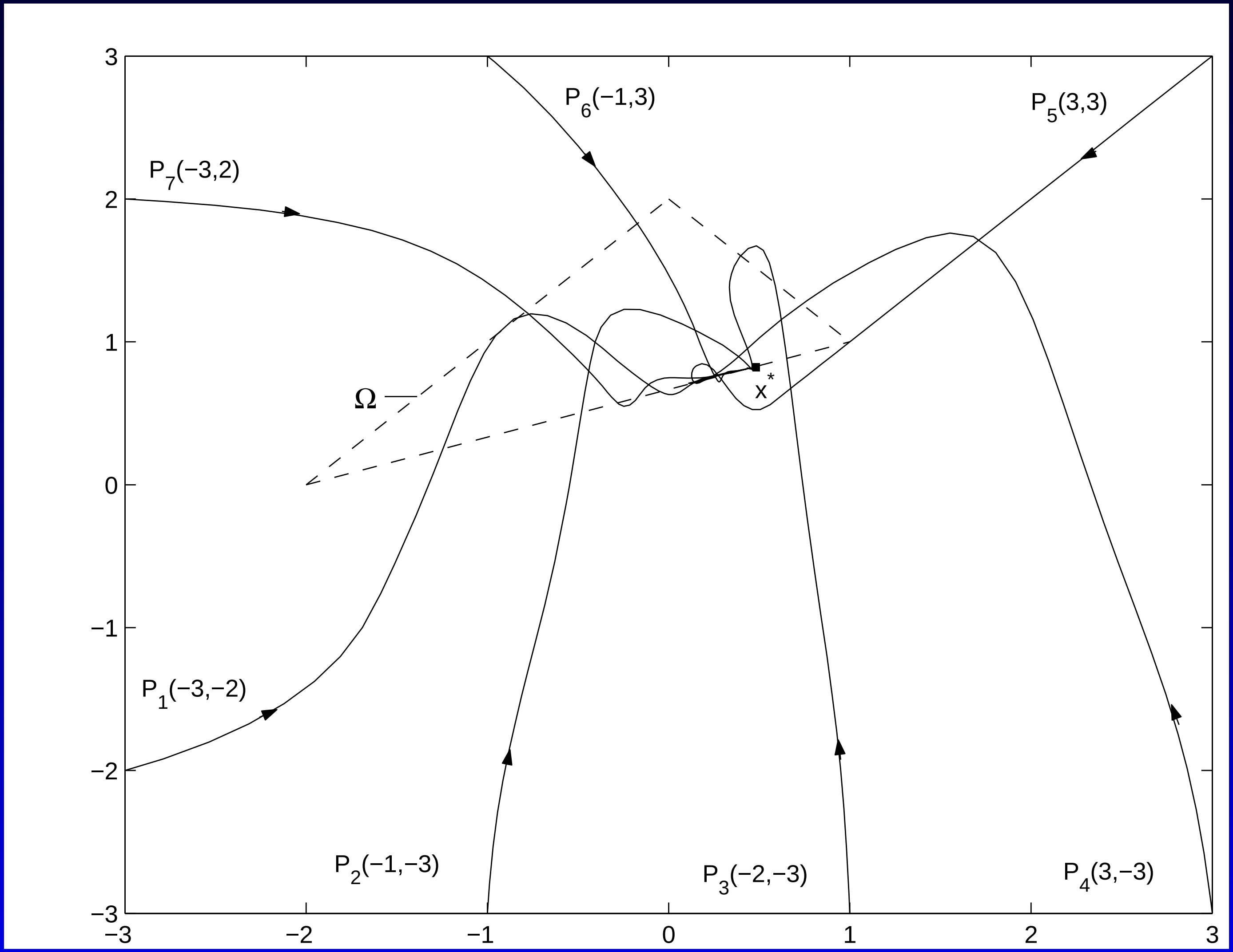
$$\begin{array}{rl} \text{minimize} & \frac{1}{4}x_1^4 + 0.5x_1^2 + \frac{1}{4}x_2^4 + 0.5x_2^2 - 0.9x_1x_2 \\ \text{subject to} & Ax \leq b,\ x \geq 0 \end{array}$$

where

$$A = \begin{pmatrix} 1 & 1 \\ -1 & 1 \\ 1 & -3 \end{pmatrix} \text{ and } b = \begin{pmatrix} 2 \\ 2 \\ -2 \end{pmatrix}.$$

This problem has an optimal solution $x^* = [0.427, 0.809]^T$.

Simulation Results



Illustrative Example

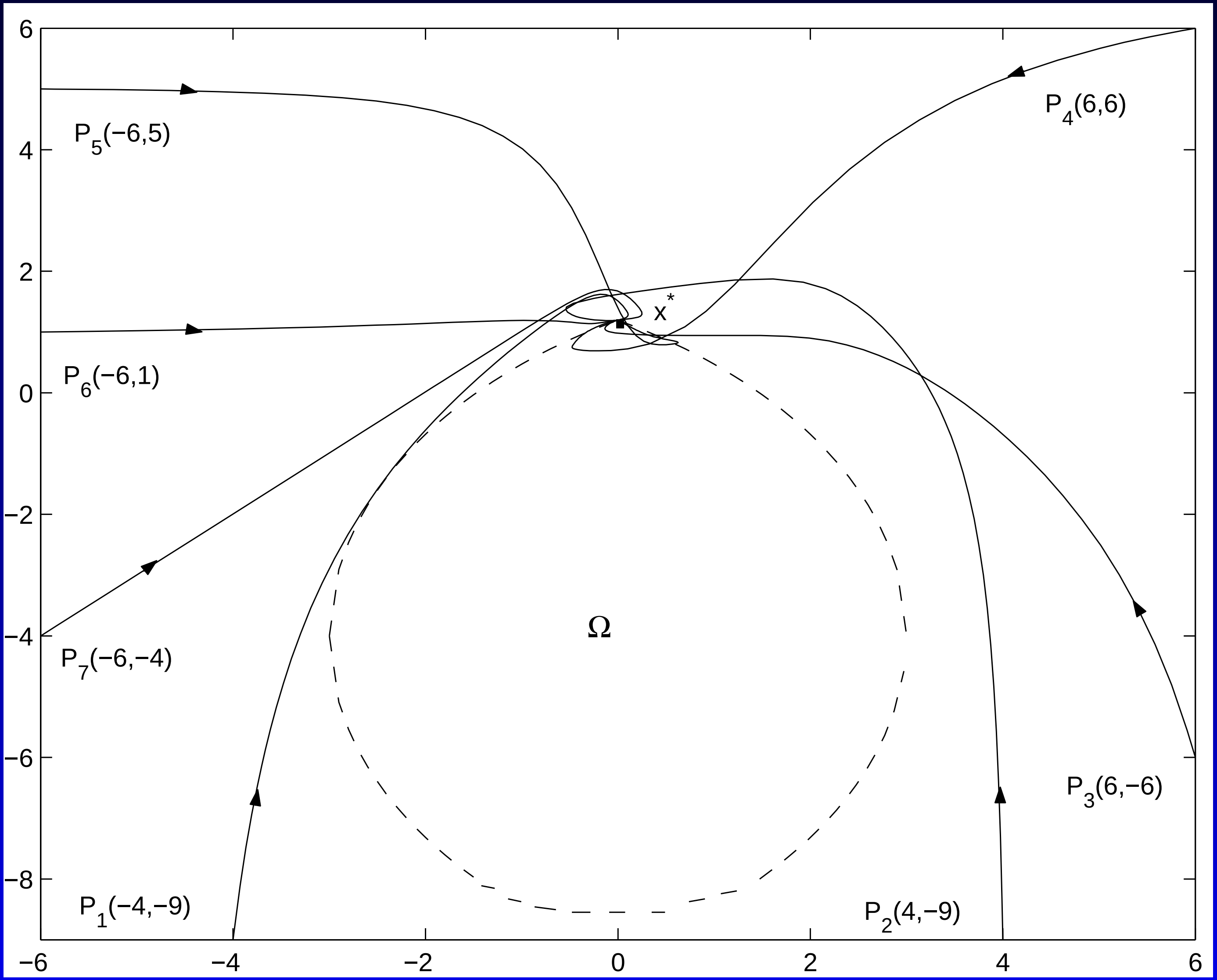
$$\begin{array}{ll} \text{minimize} & x_1^2 + 2x_1x_2 + x_2^2 + (x_1-1)^4 + (x_2-3) \\ \text{subject to} & x \geq 0,\ c_i(x) \leq 0\ (i = 1,2,3), \end{array}$$

where

$$\begin{cases} c_1(x) = x_1^2 + x_2^2 - 64, \\ c_2(x) = (x_1+3)^2 + (x_2+4)^2 - 36, \\ c_3(x) = (x_1-3)^2 + (x_2+4)^2 - 36. \end{cases}$$

This problem has an optimal solution $x^* = (0, 1.96)^T$.

Simulation Results



Illustrative Example

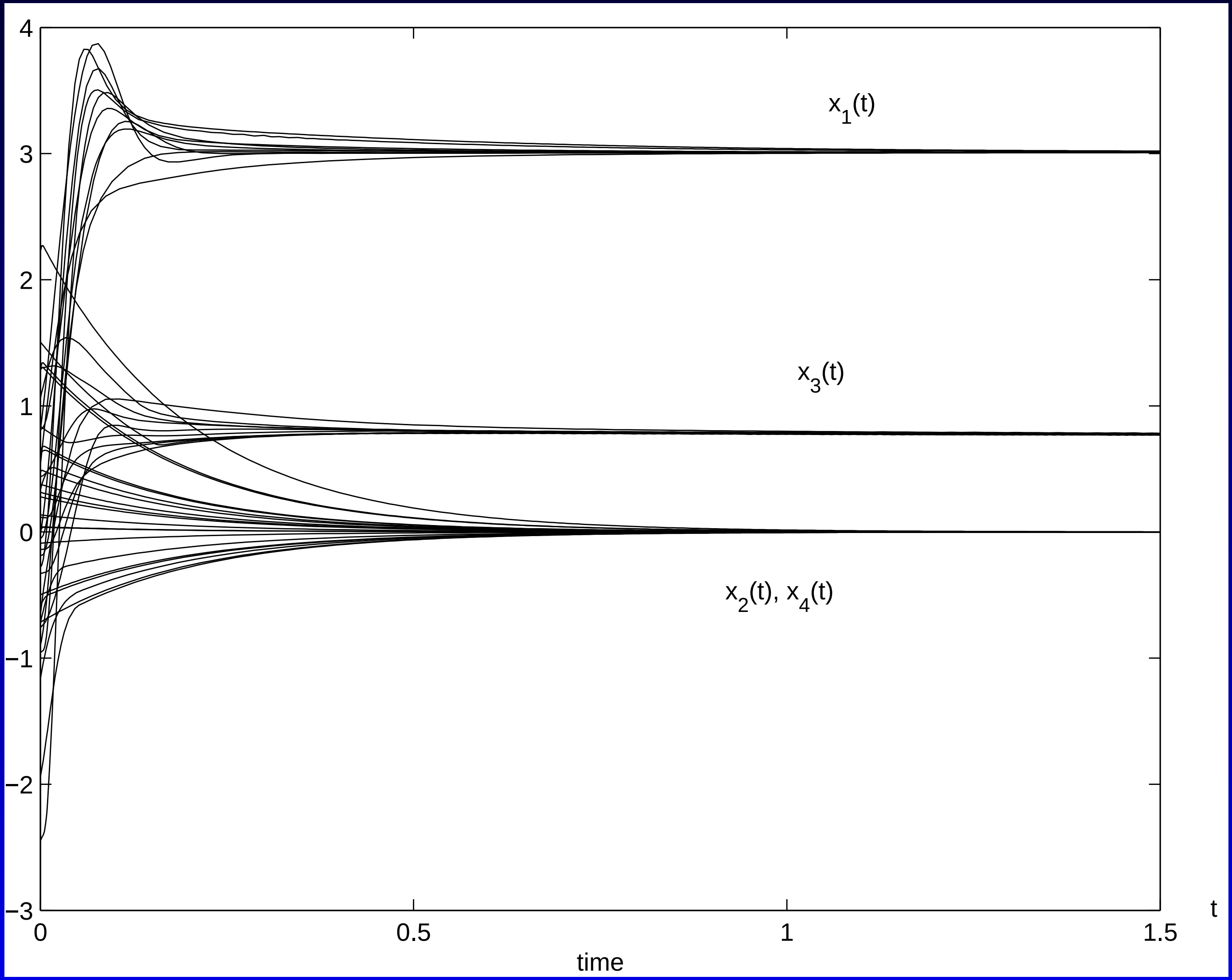
$$\begin{array}{rl} \text{minimize} & (x_1 - x_2)^2 + (x_2 - x_3)^2 + (x_3 - x_4)^4 \\ \text{subject to} & x \geq 0,\ c_i(x) \leq 0\ (i = 1, 2), \end{array}$$

where

$$c_1(x) = x_1^2 + x_2^2 + x_3^2 + x_4^2 - 9$$
$$c_1(x) = (x_1 - 4)^2 + (x_2 + 4)^2 + (x_3 - 1)^2 + (x_4 + 1)$$

This problem has an optimal solution
$x^* = (3.013, 0, 0.766, 0)^T$.

Simulation Results



Dual Network for QP$_2$

For strictly convex QP$_2$, Q is invertible. The dynamic equation of the dual network:

$$\begin{aligned}
\epsilon\frac{dy(t)}{dt} &= -CQ^{-1}C^Ty + g\left(CQ^{-1}C^Ty - y - Cq\right) \\
&\quad +Cq + b, \\
x(t) &= Q^{-1}C^Ty - q,
\end{aligned}$$

where $\epsilon > 0$.
It is also globally exponentially convergent to the optimal solution[a] [b].

---

[a] Y. Xia and J. Wang, “A dual neural network for kinematic control of redundant robot manipulators,” *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 31, no. 1, pp. 147-154, 2001.

[b] Y. Zhang and J. Wang, “A dual neural network for convex quadratic programming subject to linear equality and inequality constraints,” *Physics Letters A*, pp. 271-278, 2002.

Simplified Dual Network for QP_1

For strictly convex QP_1, Q is invertible. The dynamic equation of the simplified dual network [a]:

$$\epsilon\frac{du}{dt} = -Cx + g(Cx - u),$$
$$x = Q^{-1}(A^T y + C^T u - q),$$
$$y = (AQ^{-1}A^T)^{-1}\left[-AQ^{-1}C^T u + AQ^{-1}q + b\right],$$

where $u \in \mathbb{R}^n$ is the state vector, $\epsilon > 0$.
It is proven to be globally asymptotically convergent to the optimal solution.

---

[a] S. Liu and J. Wang, "A simplified dual neural network for quadratic programming with its KWTA application," *IEEE Trans. Neural Networks*, vol. 17, no. 6, pp. 1500-1510, 2006.

Illustrative Example

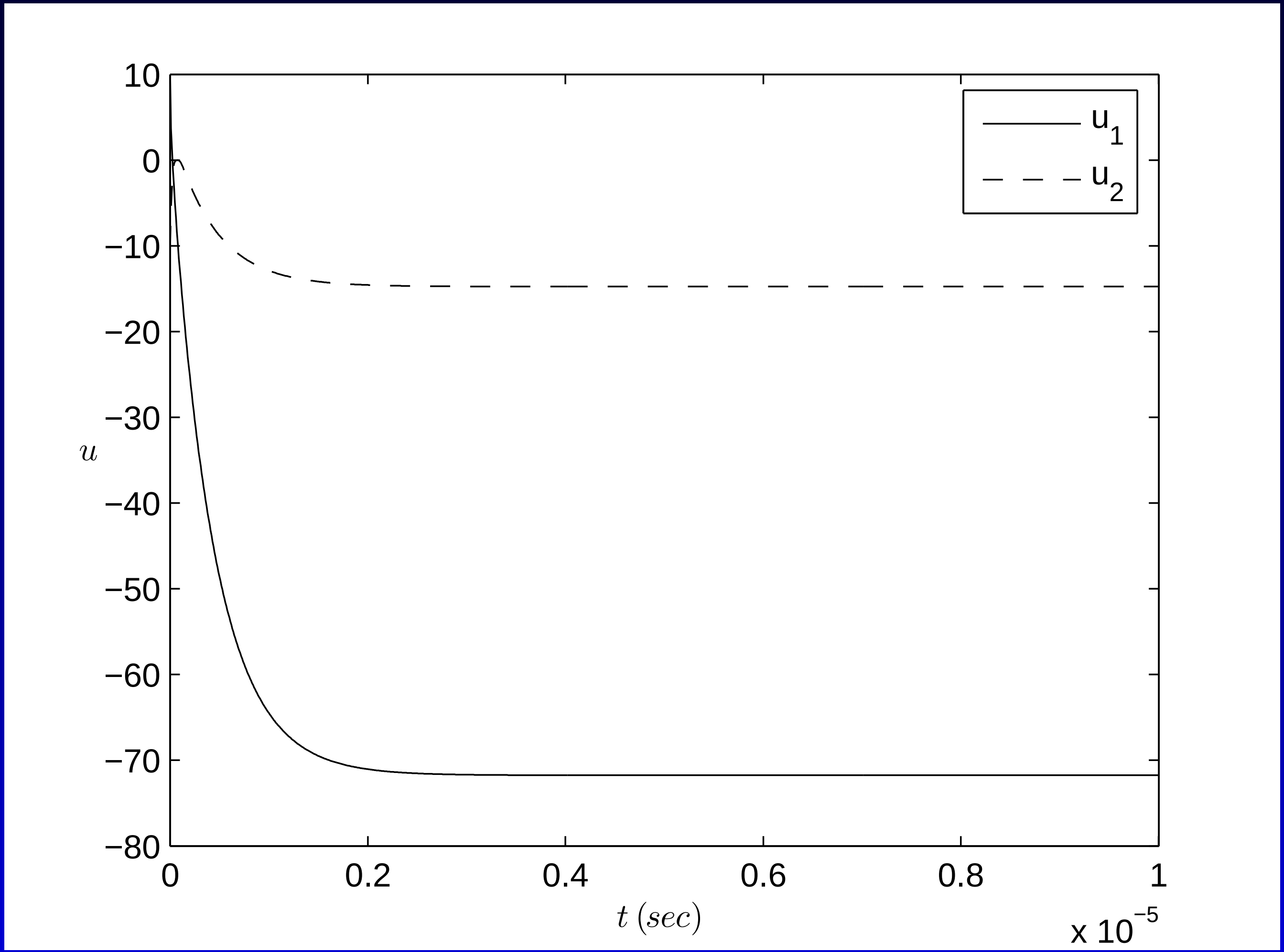
$$\begin{array}{ll} \text{minimize} & 3x_1^2+3x_2^2+4x_3^2+5x_4^2+3x_1x_2+5x_1x_3+ \\ & x_2x_4-11x_1-5x_4 \\ \text{subject to} & 3x_1-3x_2-2x_3+x_4=0, \\ & 4x_1+x_2-x_3-2x_4=0, \\ & -x_1+x_2\le -1, \\ & -2\le 3x_1+x_3\le 4. \end{array}$$

Illustrative Example (cont'd)

$$Q = \begin{bmatrix} 6 & 3 & 5 & 0 \\ 3 & 6 & 0 & 1 \\ 5 & 0 & 8 & 0 \\ 0 & 1 & 0 & 10 \end{bmatrix}, \; q = \begin{bmatrix} -11 \\ 0 \\ 0 \\ -5 \end{bmatrix},$$
$$A = \begin{bmatrix} 3 & -3 & -2 & 1 \\ 4 & 1 & -1 & -2 \end{bmatrix}, \; b = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$
$$C = \begin{bmatrix} -1 & 1 & 0 & 0 \\ 3 & 0 & 1 & 0 \end{bmatrix}, \; l = \begin{bmatrix} -\infty \\ -2 \end{bmatrix}, \; h = \begin{bmatrix} -1 \\ 4 \end{bmatrix}.$$

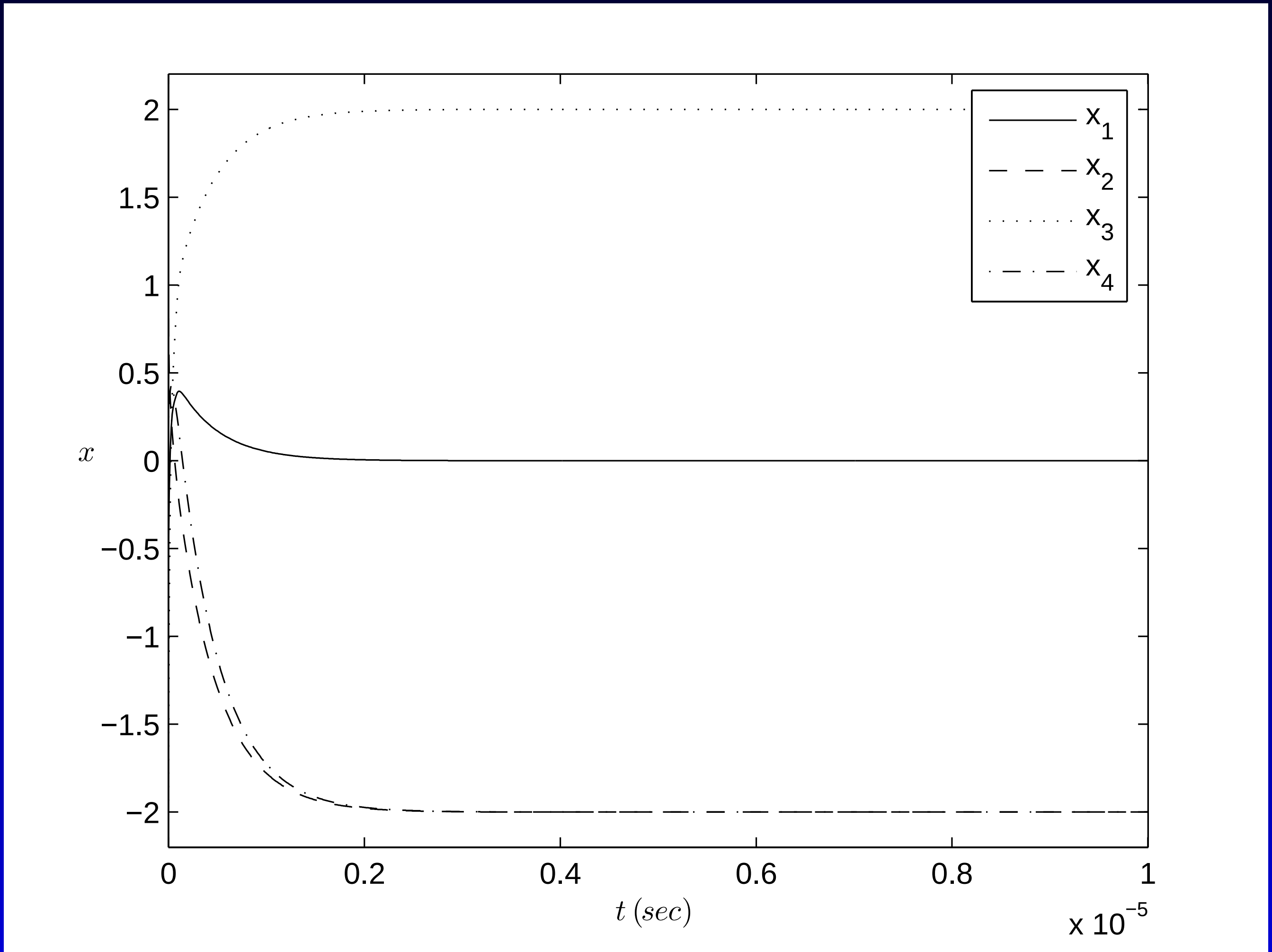
The simplified dual neural network for solving this quadratic programming problem needs only two neurons, whereas the Lagrange neural network needs twelve neurons, the primal-dual neural network needs nine neurons, the dual neural network needs four neurons.

Illustrative Example (cont'd)



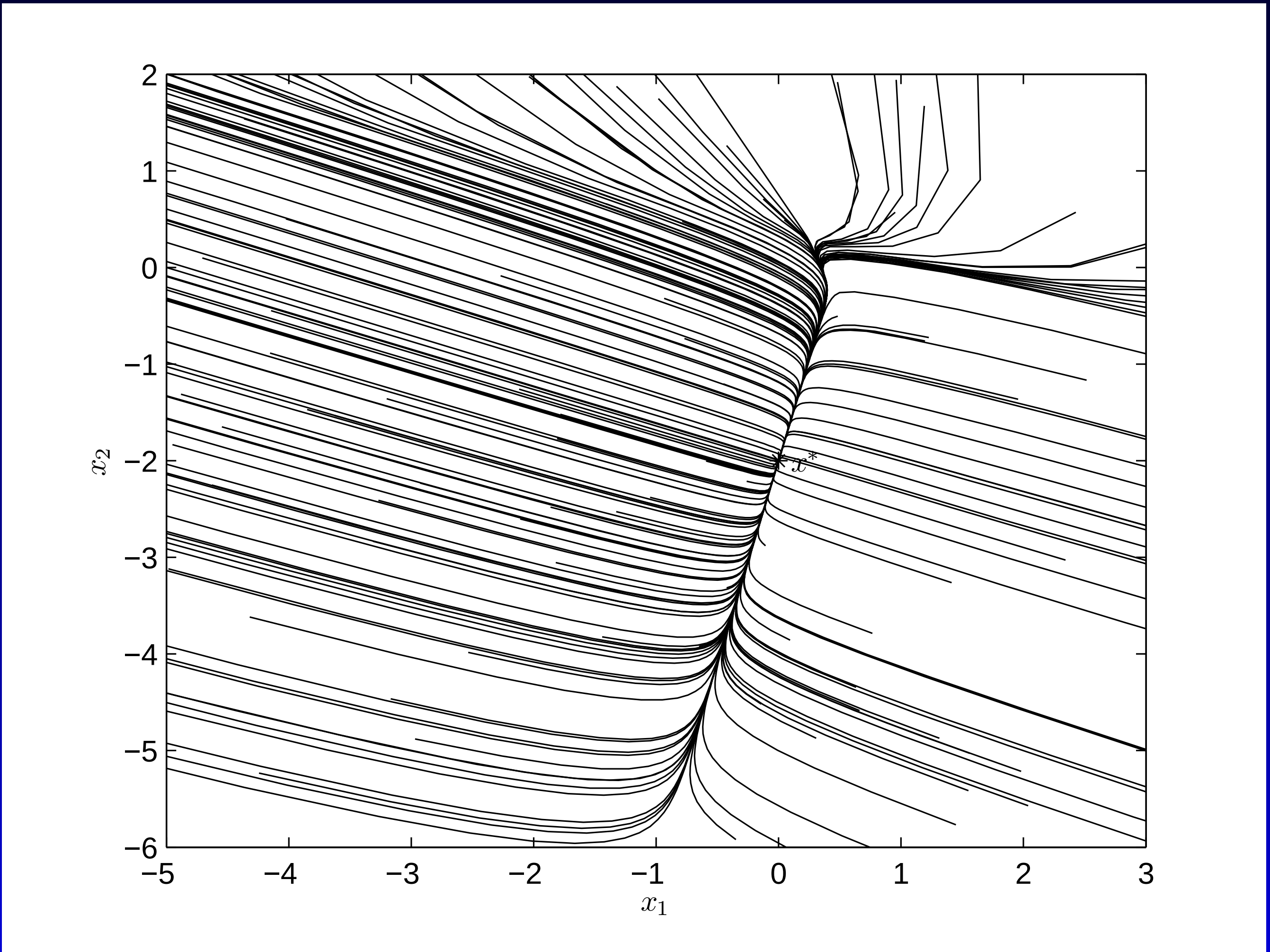
Transient behaviors of the state vector u.

Illustrative Example (cont'd)



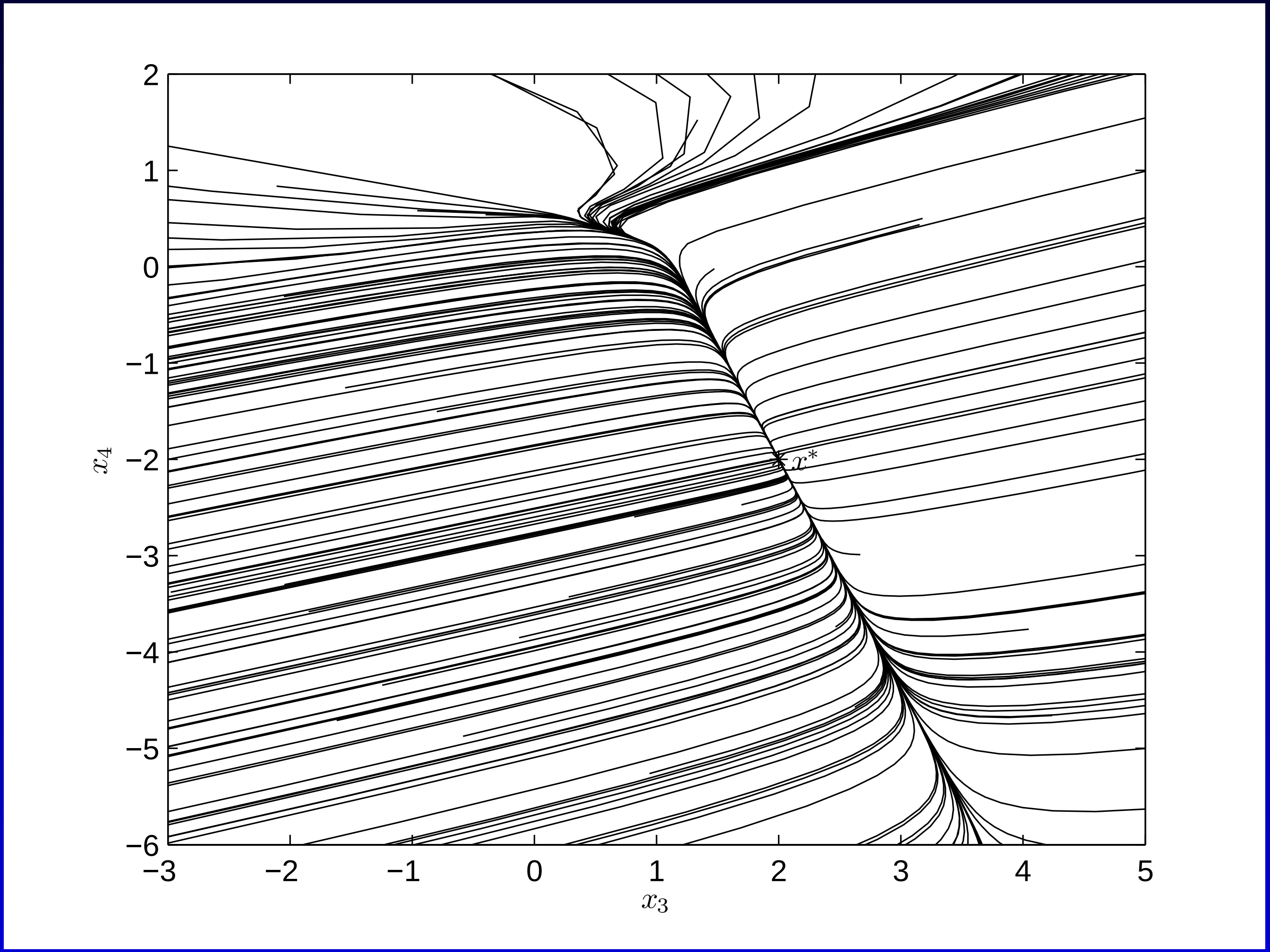
Transient behaviors of the output vector x.

Illustrative Example (cont'd)



Trajectories of x_1 and x_2 from different initial states.

Illustrative Example (cont'd)



Trajectories of x_3 and x_4 from different initial states.

A New Model for LP

A new recurrent neural network model with a discontinuous activation function was recently developed for linear programming LP_1.

A New Model for LP

A new recurrent neural network model with a discontinuous activation function was recently developed for linear programming LP_1.

The dynamic equation of the new model is described as follows:

$$\epsilon \frac{dx}{dt} = -Px - \sigma(I - P)g(x) + s, \qquad (3)$$

where $g(x) = (g_1(x_1), g_2(x_2), \ldots, g_n(x_n))^T$ is the vector-valued activation function, ϵ is a positive scaling constant, σ is a nonnegative gain parameter, $P = A^T(AA^T)^{-1}A$, and $s = -(I - P)q + A^T(AA^T)^{-1}b$.

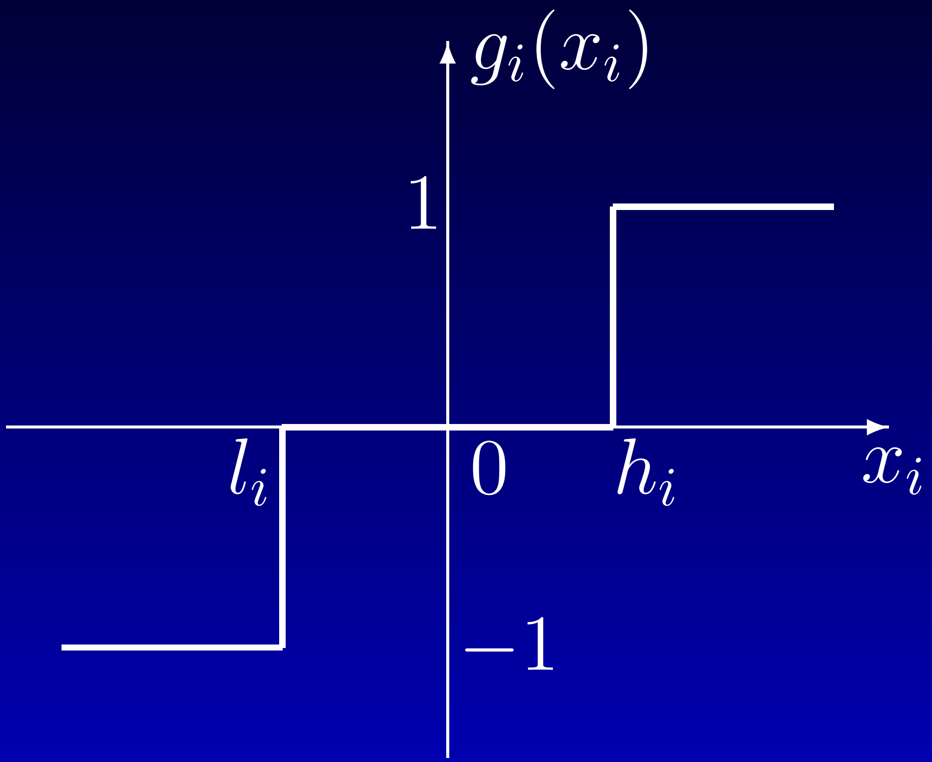
Activation Function

The following activation function is defined[a]: For $i = 1, 2, \ldots, n$;

$$g_i(x_i) = \begin{cases} 1, & \text{if } x_i > h_i, \\ [0, 1], & \text{if } x_i = h_i, \\ 0, & \text{if } x_i \in (l_i, h_i), \\ [-1, 0], & \text{if } x_i = l_i, \\ -1, & \text{if } x_i < l_i. \end{cases}$$

[a] Q. Liu, and J. Wang, "A one-layer recurrent neural network with a discontinuous activation function for linear programming," Neural Computation, in press, 2007.

Activation Function (cont'd)



Convergence Results

The neural network is globally convergent to an optimal solution of LP_1 with $C = I$, if $\bar{\Omega} \subset \Omega$, where $\bar{\Omega}$ is the equilibrium point set and $\Omega = \{x | l \leq x \leq h\}$.
The neural network is globally convergent to an optimal solution of LP_1 with $C = I$, if it has a unique equilibrium point and $\sigma \geq 0$ when $(I - P)c = 0$ or one of the following conditions holds when $(I - P)c \neq 0$:

(i) $\sigma \geq \|(I - P)c\|_p / \min^{+}_{\gamma \in X} \|(I - P)\gamma\|_p$ for $p = 1, 2, \infty$, or

(ii) $\sigma \geq c^T(I - P)c / \min^{+}_{\gamma \in X} \{|c^T(I - P)\gamma|\}$,

where $X = \{-1, 0, 1\}^n$

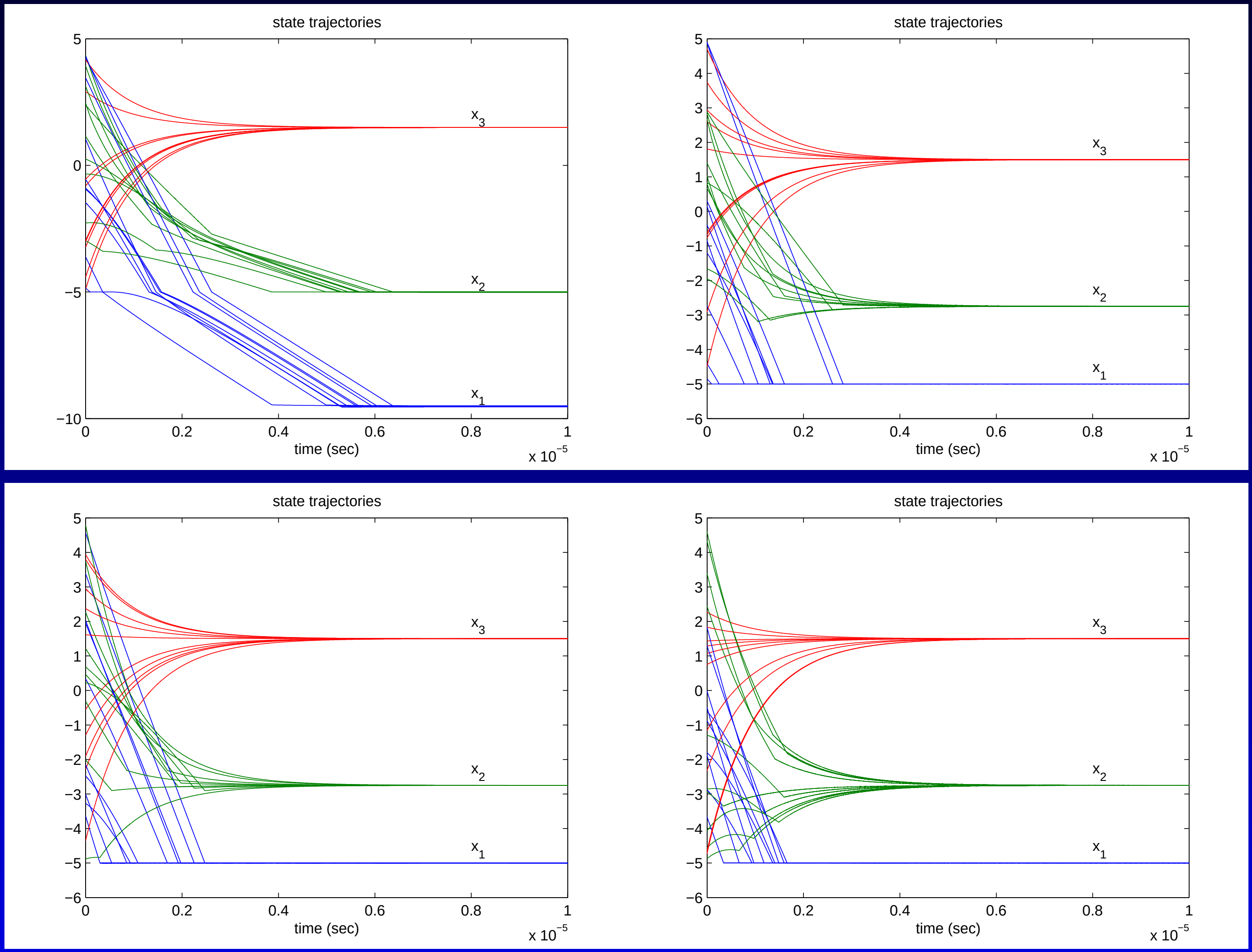
Simulation Results

Consider the following LP problem:

$$\begin{array}{ll} \text{minimize} & 4x_1 + x_2 + 2x_3, \\ \text{subject to} & x_1 - 2x_2 + x_3 = 2, \\ & -x_1 + 2x_2 + x_3 = 1, \\ & -5 \leq x_1, x_2, x_3 \leq 5. \end{array}$$

According to the above condition, the lower bound of σ is 9

Simulation Results (cont'd)



Transient behaviors of the states with four different values of $\sigma \in \{3, 5, 9, 15\}$.

k Winners Take All Operation

The k-winners-take-all (kWTA) operation is to select the k largest inputs out of n inputs ($1 \leq k \leq n$).

k Winners Take All Operation

The k-winners-take-all (kWTA) operation is to select the k largest inputs out of n inputs ($1 \leq k \leq n$).

The kWTA operation has important applications in machine learning, such as k-neighborhood classification, k-means clustering, etc.

k Winners Take All Operation

The k-winners-take-all (kWTA) operation is to select the k largest inputs out of n inputs ($1 \leq k \leq n$).

The kWTA operation has important applications in machine learning, such as k-neighborhood classification, k-means clustering, etc.

As the number of inputs increases and/or the selection process should be operated in real time, parallel algorithms and hardware implementation are desirable.

kWTA Problem Formulations

The kWTA function can be defined as:

$$x_i = f(u_i) = \begin{cases} 1, & \text{if } u_i \in \{k \text{ largest elements of } u\}, \\ 0, & \text{otherwise}, \end{cases}$$

where $u \in \mathbb{R}^n$ and $x \in \mathbb{R}^n$ is the input vector and output vector, respectively.

kWTA Problem Formulations

The kWTA function can be defined as:

$$x_i = f(u_i) = \begin{cases} 1, & \text{if } u_i \in \{k \text{ largest elements of } u\}, \\ 0, & \text{otherwise,} \end{cases}$$

where $u \in \mathbb{R}^n$ and $x \in \mathbb{R}^n$ is the input vector and output vector, respectively.
The kWTA solution can be determined by solving the following linear integer program:

$$\begin{array}{ll} \text{minimize} & -\sum_{i=1}^{n} u_i x_i, \\ \text{subject to} & \sum_{i=1}^{n} x_i = k, \\ & x_i \in \{0, 1\}, \quad i = 1, 2, \ldots, n. \end{array}$$

kWTA Problem Formulations (cont'd)

If the kth and $(k+1)$th largest elements of u are different (denoted as $\bar{u}_k$ and $\bar{u}_{k+1}$ respectively), the kWTA problem is equivalent to the following LP or QP problems:

$$\begin{array}{ll} \text{minimize} & -u^T x \ \text{ or } \ \frac{a}{2}x^T x - u^T x, \\ \text{subject to} & \sum_{i=1}^{n} x_i = k, \\ & 0 \le x_i \le 1, \quad i = 1, 2, \ldots, n, \end{array}$$

where $a \le \bar{u}_k - \bar{u}_{k+1}$ is a positive constant.

The Primal-Dual Network for kWTA

The primal-dual network based on the QP formulation needs $3n+1$ neurons and $6n+2$ connections, and its dynamic equations can be written as:

$$\left\{\begin{array}{rcl}
\frac{dx}{dt} & = & -(1+a)(x-(x+ve+w-ax+u)^+) \\
 & & -(e^Tx-k)e-x-y+e \\
\frac{dy}{dt} & = & -y+(y+w)^+-x-y+e \\
\frac{dv}{dt} & = & -e^T(x-(x+ve+w-ax+u)^+) \\
 & & +e^Tx-k \\
\frac{dw}{dt} & = & -x+(x+ve+w-ax+u)^+ \\
 & & -y+(y+w)^++x+y-e
\end{array}\right.$$

where $x, y, w \in \mathbb{R}^n$, $v \in \mathbb{R}$, $e = (1, 1, \ldots, 1)^T \in \mathbb{R}^n$, $x^+ = (x_1^+, \ldots, x_n^+)^T$, and $x_i^+ = \max\{0, x_i\}$

The Projection Network for kWTA

The projection neural network for kWTA operation based on the QP formulation needs $n+1$ neurons and $2n+2$ connections, which dynamic equations can be written as:

$$\begin{cases} \frac{dx}{dt} &= \lambda\left[-x + f(x - \eta(ax - u - ve))\right] \\ \frac{dv}{dt} &= \lambda(-e^T x + k). \end{cases}$$

where $x \in \mathbb{R}^n$, $v \in \mathbb{R}$, λ and η are positive constants, $f(x) = (f(x_1), \ldots, f(x_n))^T$ and

$$f(x_i) = \begin{cases} 0, & \text{if } x_i < 0, \\ x_i, & \text{if } 0 \le x_i \le 1, \\ 1, & \text{if } x_i > 1. \end{cases}$$

The Simplified Dual Network for kWTA

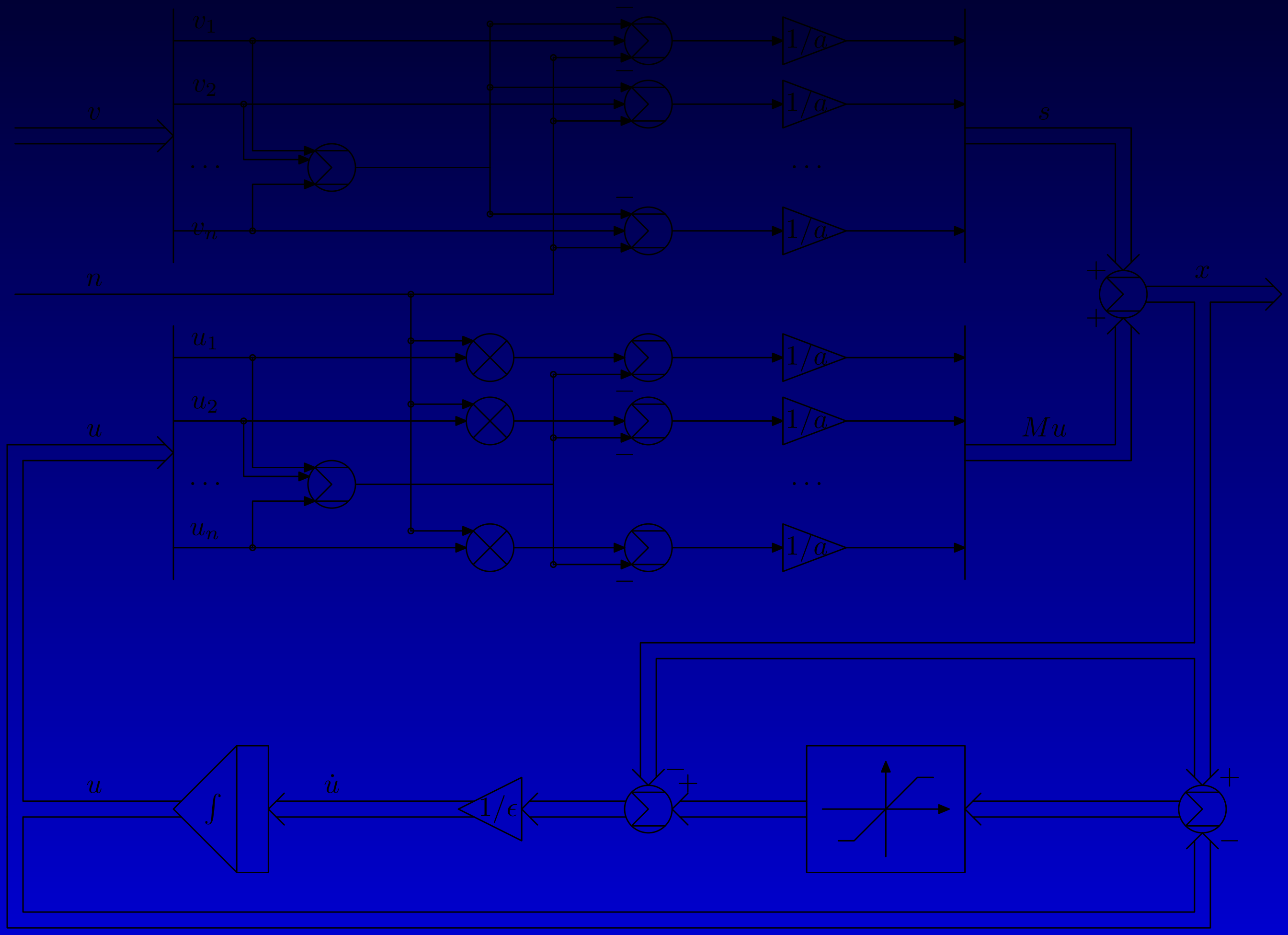
The simplified dual neural network for kWTA operation based on the QP formulation [a] needs n neurons and $3n$ connections, and its dynamic equation can be written as:

$$\begin{cases} \frac{dy}{dt} &= \lambda\left[-My + f((M-I)y - s) - s\right] \\ x &= My + s, \end{cases}$$

where $x, y \in \mathbb{R}^n$, $M = 2(I - ee^T/n)/a$, $s = Mu + ke/n$, I is an identity matrix, λ and f are defined as before.

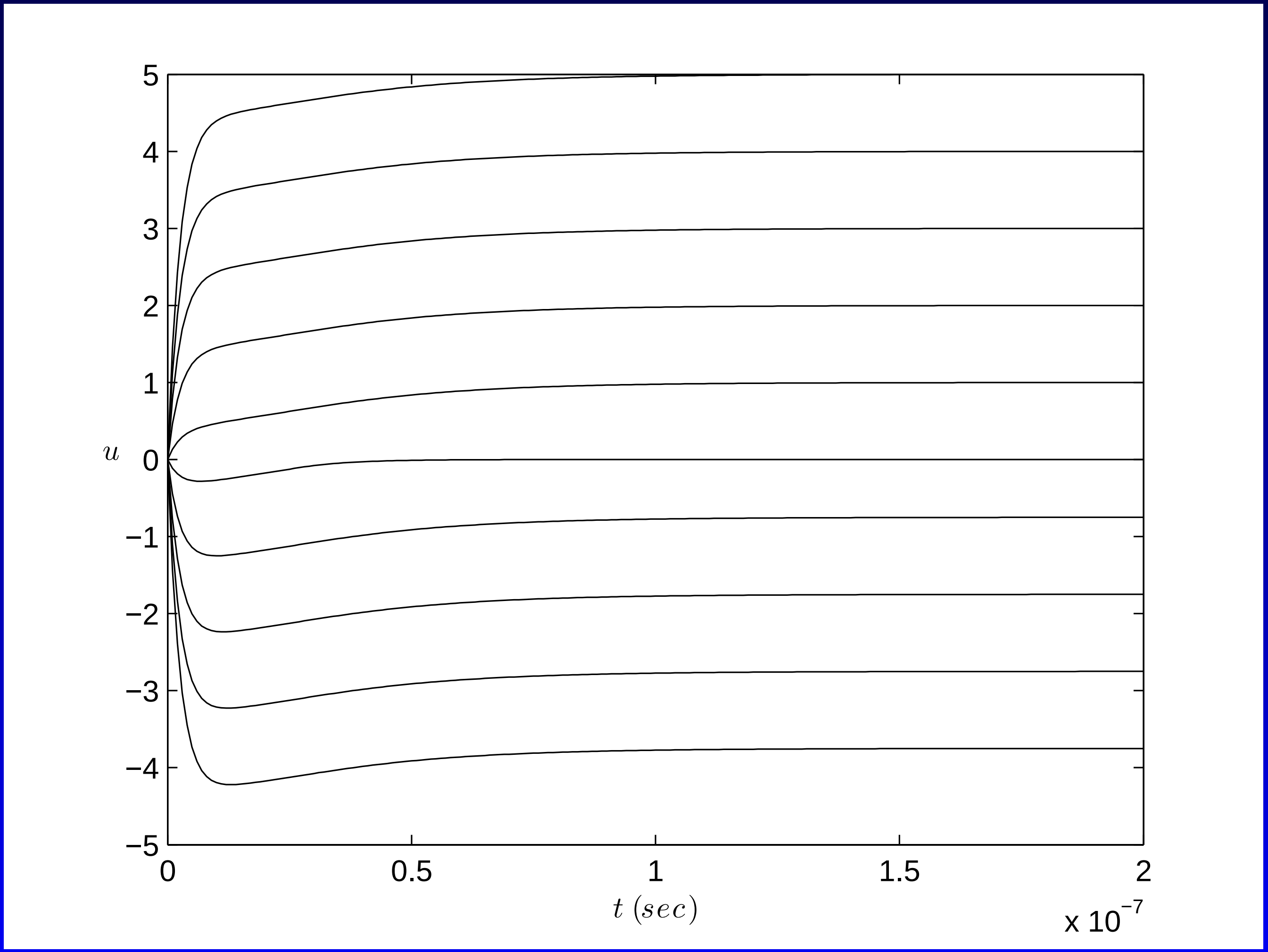
[a] S. Liu and J. Wang, “A simplified dual neural network for quadratic programming with its KWTA application,” *IEEE Trans. Neural Networks*, vol. 17, no. 6, pp. 1500-1510, 2006.

The Simplified Dual Network for kWTA



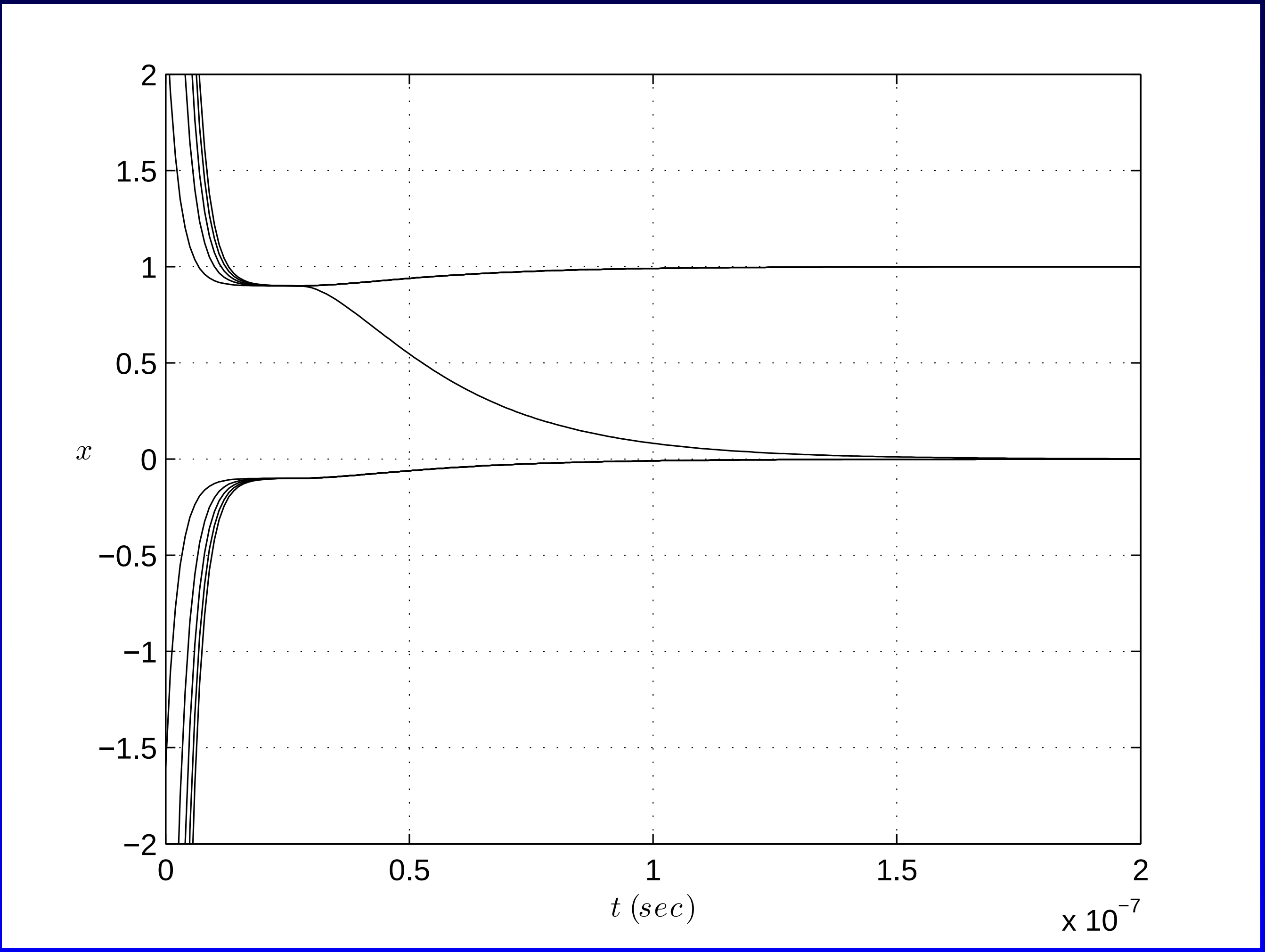
A Static Example

Let the inputs are $v_i = i \ (i = 1, 2, \cdots, n)$, $n = 10, k = 2, \epsilon = 10^{-8}$, and $a = 0.25$.

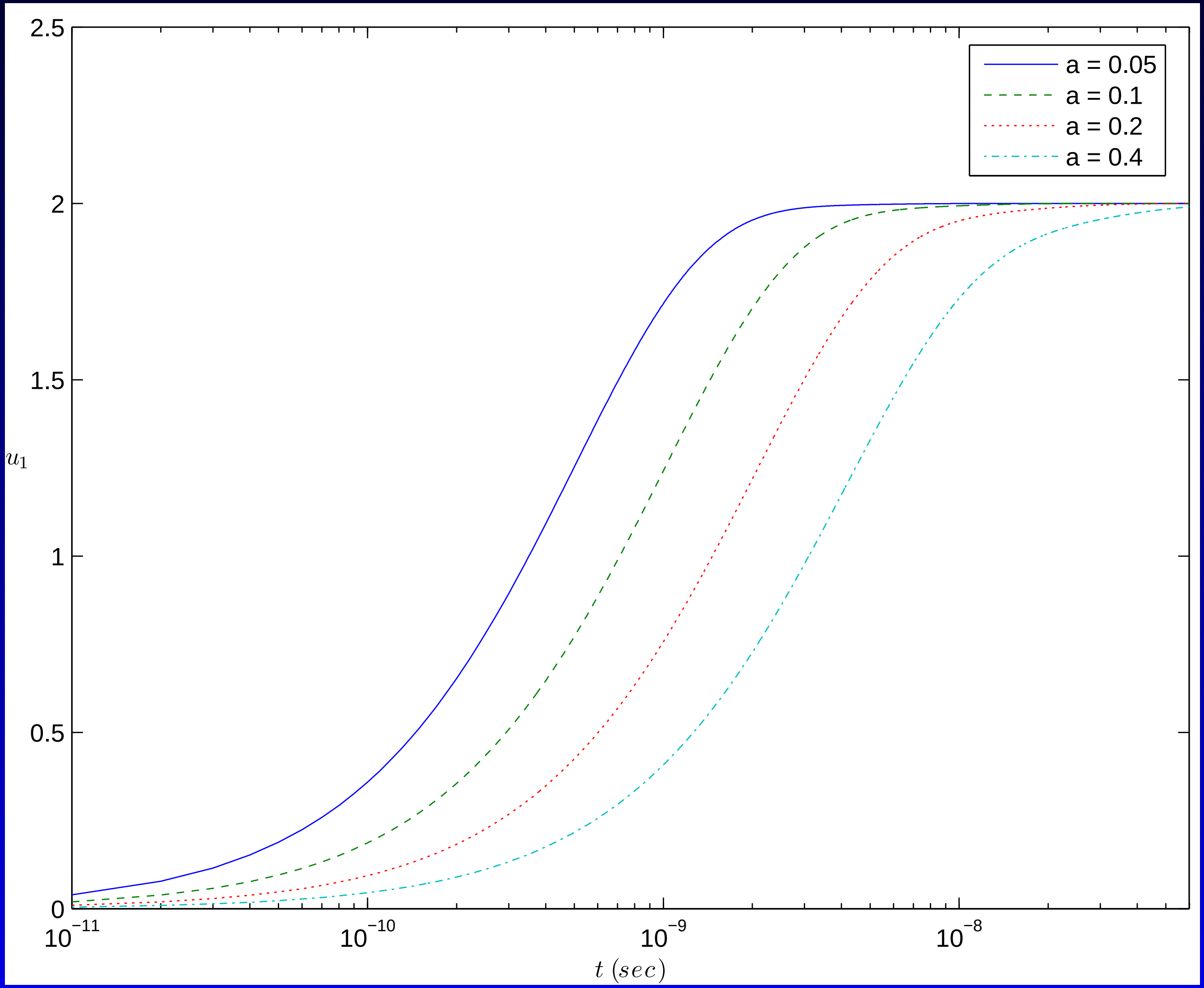


A Static Example (cont'd)

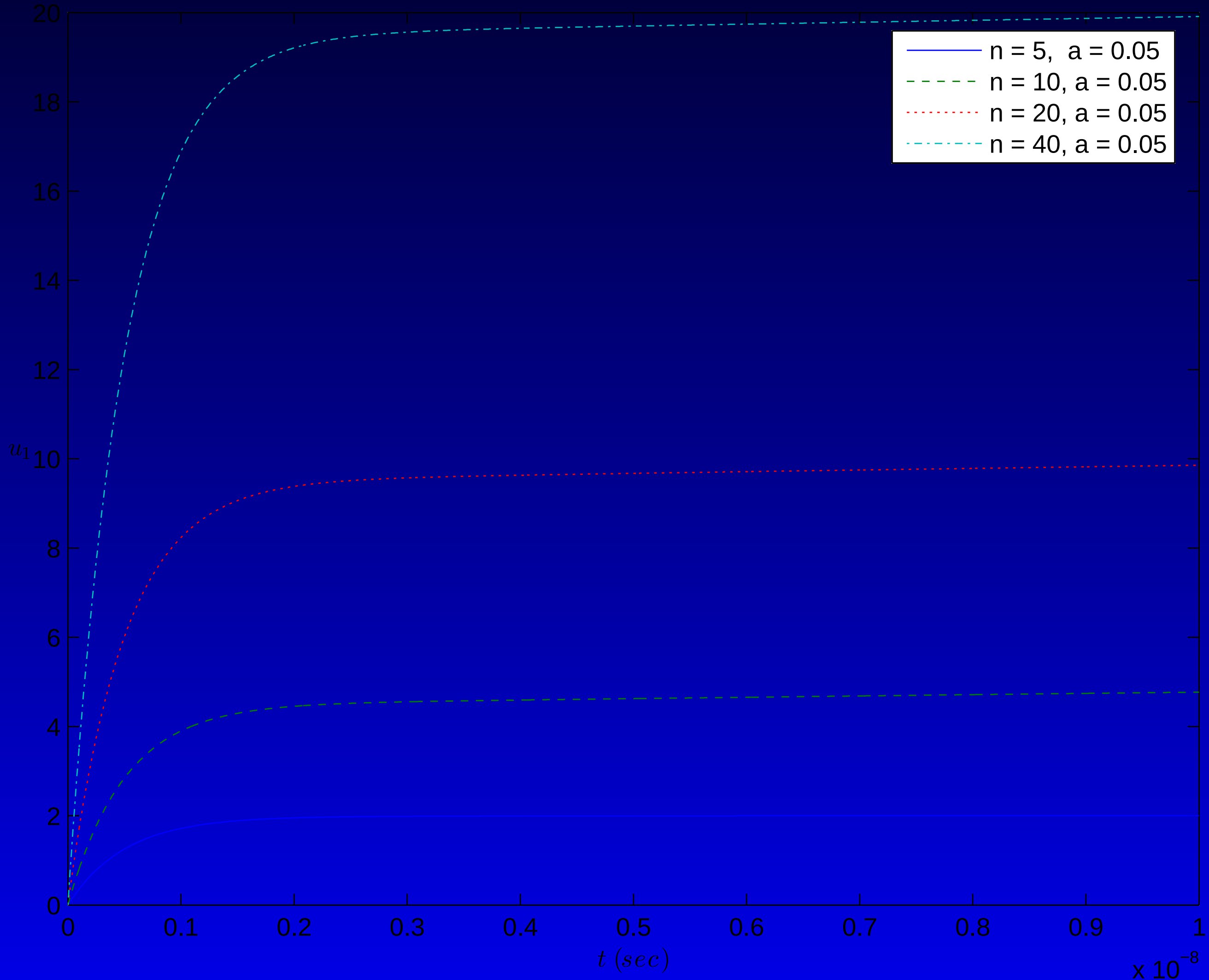
Let the inputs are $v_i = i \ (i = 1, 2, \cdots, n)$, $n = 10, k = 2, \epsilon = 10^{-8}$, and $a = 0.25$.



A Static Example (cont'd)

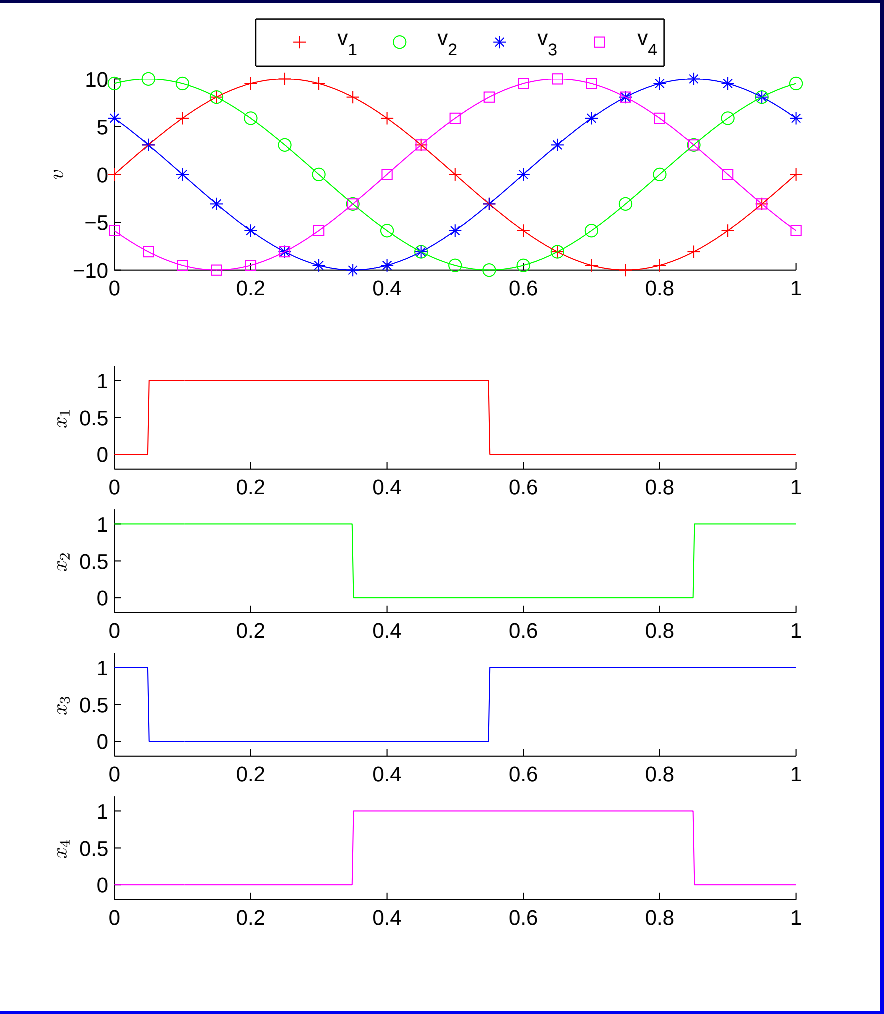


A Static Example (cont’d)



A Dynamic Example

Let inputs be 4 sinusoidal input signals (i.e., $n = 4$) $v_i(t) = 10\sin[2\pi(1000t + 0.2(i-1))]$, and $k = 2$.



A One-layer kWTA Network

The dynamic equation of a new LP-based kWTA network model is described as follows:

$$\epsilon \frac{dx}{dt} = -Px - \sigma(I - P)g(x) + s, \tag{4}$$

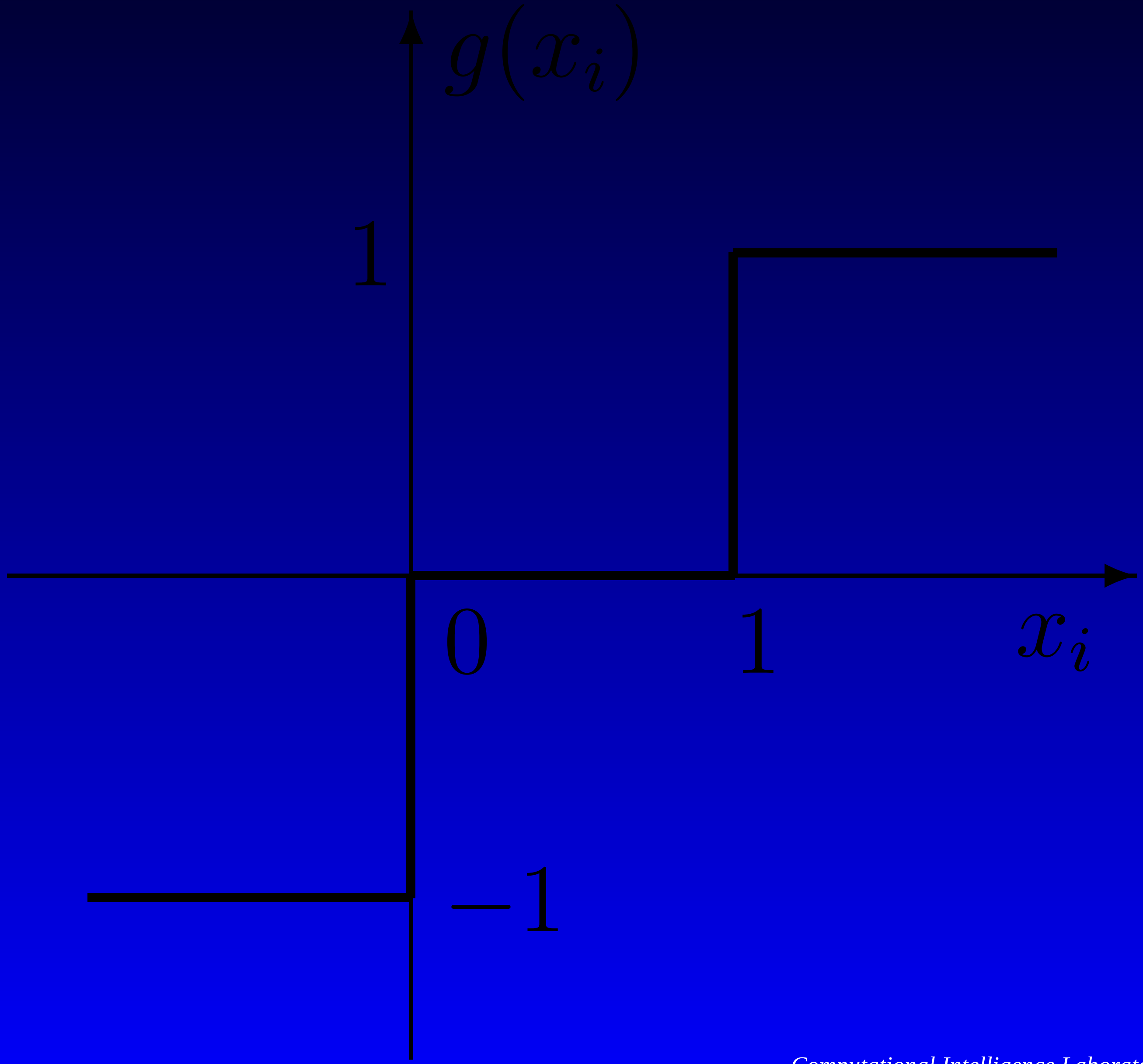
where $P = ee^T/n$, $s = u - Pu + ke/n$, ϵ is a positive scaling constant, σ is a nonnegative gain parameter, and $g(x) = (g(x_1), g(x_2), \ldots, g(x_n))^T$ is a discontinuous vector-valued activation function.

Activation Function

A discontinuous activation function is defined as follows:

$$g(x_i) = \begin{cases} 1, & \text{if } x_i > 1, \\ [0, 1], & \text{if } x_i = 1, \\ 0, & \text{if } 0 < x_i < 1, \\ [-1, 0], & \text{if } x_i = 0, \\ -1, & \text{if } x_i < 0. \end{cases}$$

Activation Function (cont'd)



Convergence Results

- The network (4) can perform the kWTA operation if $\bar{\Omega} \subset \{x \in \mathbb{R}^n : 0 \le x \le 1\}$, where $\bar{\Omega}$ is the set of equilibrium point(s).
- The network (4) can perform the kWTA operation if it has a unique equilibrium point and $\sigma \ge 0$ when $(I - ee^T/n)u = 0$ or one of the following conditions holds when $(I - ee^T/n)u \ne 0$:
  - (i) $\sigma \ge \frac{\sum_{i=1}^n |u_i - \sum_{j=1}^n u_j/n|}{2n-2}$, or
  - (ii) $\sigma \ge n\sqrt{\frac{\sum_{i=1}^n (u_i - \sum_{j=1}^n u_j/n)^2}{n(n-1)}}$, or
  - (iii) $\sigma \ge 2\max_i |u_i - \sum_{j=1}^n u_j/n|$, or,
  - (iv) $\sigma \ge \frac{\sqrt{\sum_{i=1}^n (u_i - \sum_{j=1}^n u_j/n)^2}}{\min^+_{\gamma_i \in \{-1,0,1\}} \left\{ |\sum_{i=1}^n (u_i - \sum_{j=1}^n u_j/n)\gamma_i| \right\}}$.

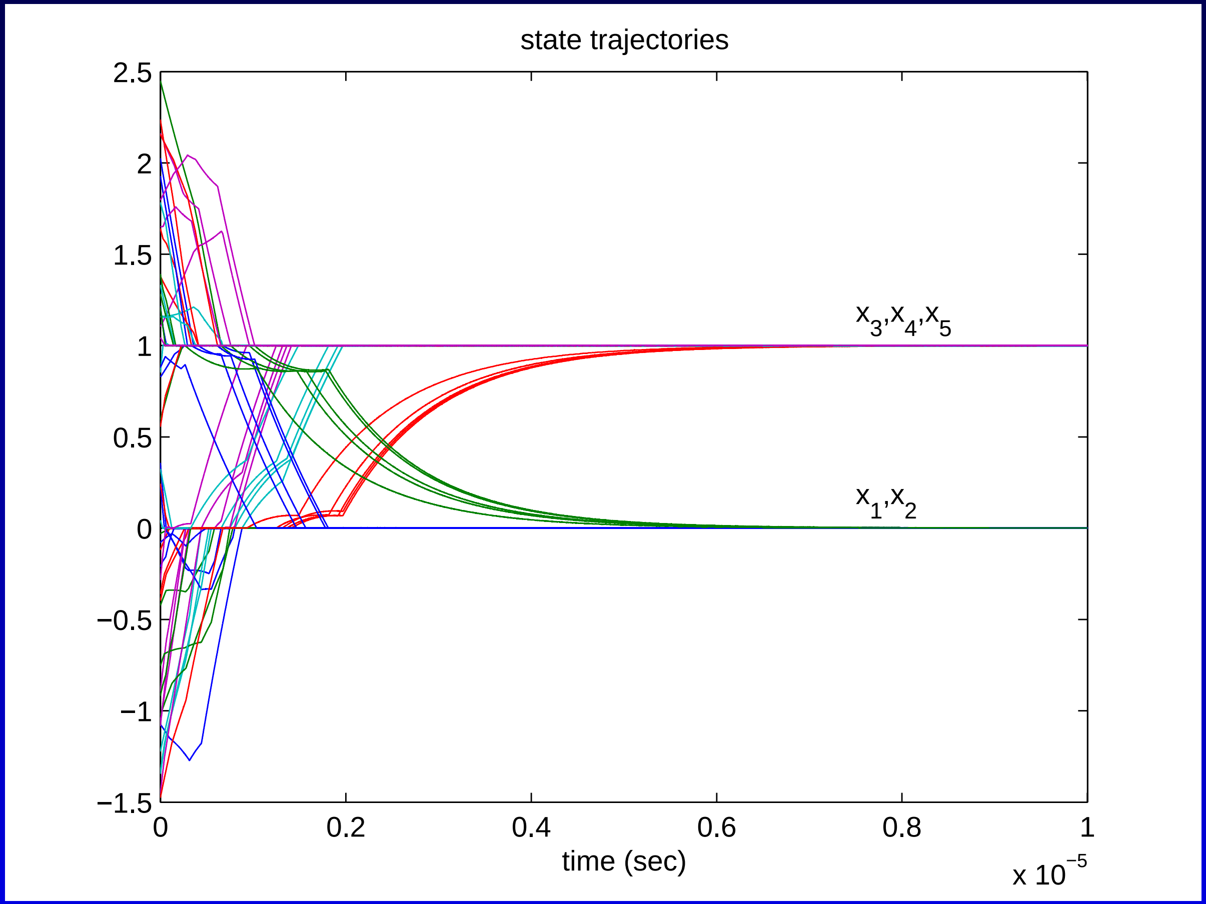
Model Comparisons

model type	Eqn(s).	neurons	conn
Primal-dual neural network	(**??**)	$3n + 1$	$6n$ -
Projection neural network	(**??**)	$n + 1$	$2n$ -
Simplified dual network	(**??**)	n	$3n$
Neural network herein	(4)	n	$2n$
Neural network herein	(**??**)(**??**)	n	$3n$

Table 1: Comparison of related neural networks in terms of model complexity.

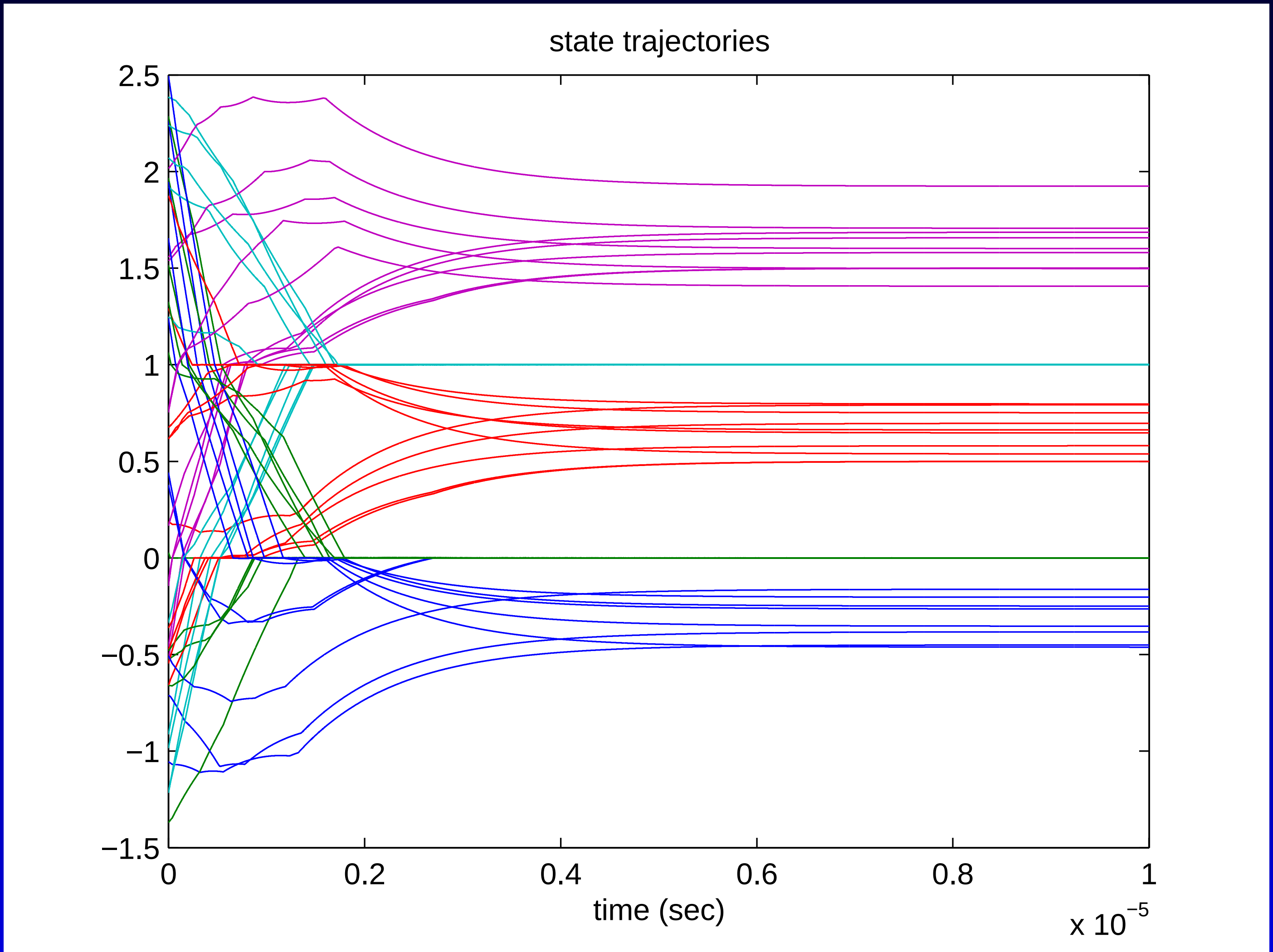
Simulation Results

Consider a kWTA problem with input vector $u_i = i\ (i = 1, 2, \ldots, n)$, $n = 5, k = 3$.



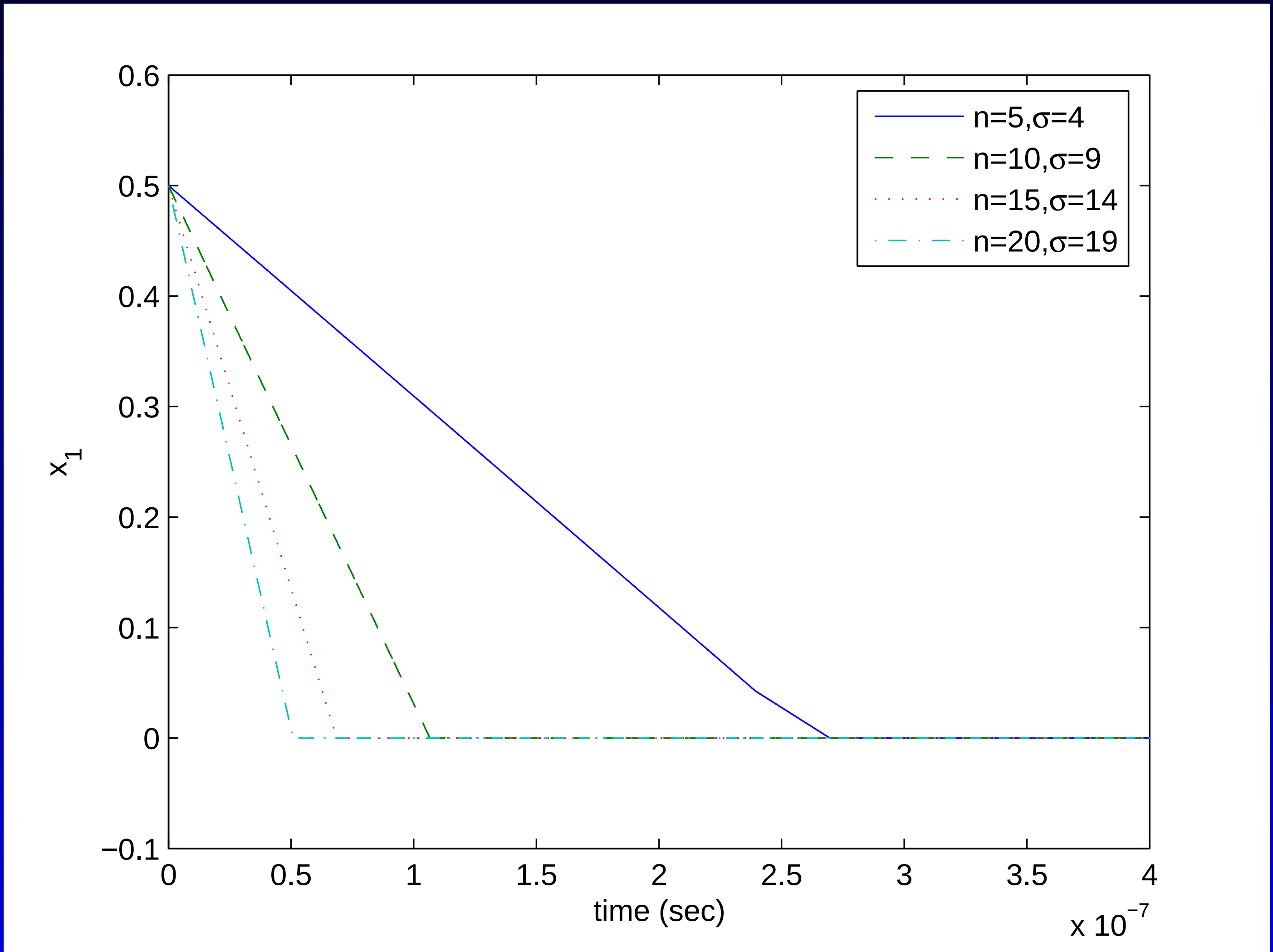
Transient behaviors of the kWTA network $\sigma = 6$.

Convergence Results (cont’d)



Transient behaviors of the kWTA network with $\sigma = 2$.

Convergence Results (cont'd)



Convergence behavior of the kWTA network with respect to different values of n.

Linear Assignment Problem

The linear assignment problem is to find an optimal solution to the following linear integer programming problem:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}, \\
\text{subject to} \quad & \sum_{j=1}^{n} x_{ij} = 1, \quad i = 1, 2, \ldots, n, \\
& \sum_{i=1}^{n} x_{ij} = 1, \quad j = 1, 2, \ldots, n, \\
& x_{ij} \in \{0, 1\}, \quad i, j = 1, 2, \ldots, n.
\end{aligned}
$$

Linear Assignment Problem (cont'd)

If the optimal solution to problem (71) is unique, then it is equivalent to the following linear programming problem:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}, \\
\text{subject to} \quad & \sum_{j=1}^{n} x_{ij} = 1, \quad i = 1, 2, \ldots, n, \\
& \sum_{i=1}^{n} x_{ij} = 1, \quad j = 1, 2, \ldots, n, \\
& 0 \le x_{ij} \le 1, \quad i, j = 1, 2, \ldots, n.
\end{aligned}
$$

Simulation Results

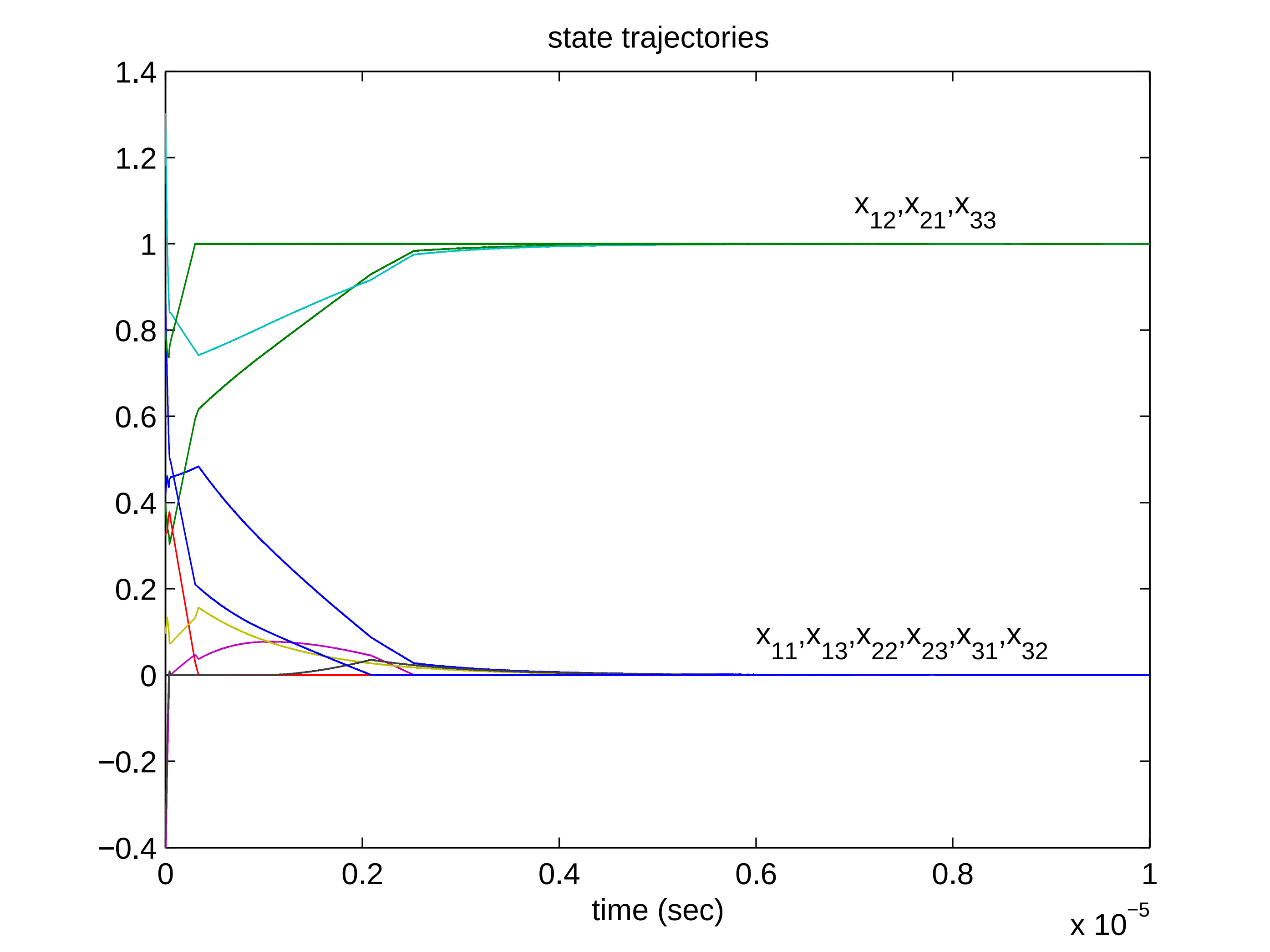
Consider a linear assignment problem with

$$C = \begin{pmatrix} 4 & 2 & 5 \\ 3 & 1.5 & 2 \\ 4 & 2.5 & 1 \end{pmatrix}.$$

A lower bound of σ is 13.

Simulation Results (cont’d)

Let $\epsilon = 10^{-6}$ and $\sigma = 15$.



Support Vector Machine

Consider a set of training examples

$$\{(\underline{x}_1, y_1), (\underline{x}_2, y_2), ..., (\underline{x}_N, y_N)\}$$

where the i-th example $\underline{x}_i \in R^n$ belongs to one of two separate classes labeled by $y_i \in \{-1, 1\}$.

A support vector machine provides an optimal partition with maximum possible margin for pattern classification.

SVM Primal Problem

$$\min \frac{1}{2} w^T w + c \sum_{i=1}^{N} \xi_i$$

$$\text{s.t.} \begin{cases} y_i[w^T \phi(\underline{x}_i) + b] \geq 1 - \xi_i, \ i = 1, \cdots, N \\ \xi_i \geq 0, \quad i = 1, \cdots, N. \end{cases}$$

where $c > 0$ is a regularization parameter for the tradeoff between model complexity and training error, and ξ_i measures the (absolute) difference between $w^T z + b$ and y_i.

SVM Dual Problem

$$\max \quad -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \phi(\underline{x}_i)^T \phi(\underline{x}_j)\alpha_i\alpha_j + \sum_{i=1}^{N}\alpha_i$$

$$\text{s.t.} \begin{cases} \sum_{i=1}^{N} \alpha_i y_i = 0 \\ 0 \le \alpha_i \le c, \quad i = 1, \cdots, N. \end{cases}$$

SVM Dual Problem

For convenient computation here, let $a_i = \alpha_i y_i$. Then the SVM dual problem can be equivalently written as

$$\min \ \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} a_i a_j K(\underline{x}_i, \underline{x}_j) - \sum_{i=1}^{N} a_i y_i$$

$$\text{s.t.} \ \begin{cases} \sum_{i=1}^{N} a_i = 0 \\ c_i^- \leq a_i \leq c_i^+, \quad i = 1, \cdots, N. \end{cases}$$

where $c_i^- = c \cdot sgn(1 - y_i)$ and $c_i^+ = c \cdot sgn(1 + y_i)$ for $i = 1, ..., N$.

SVM Learning Network

$$\epsilon\frac{d}{dt}\begin{pmatrix} a \\ \mu \end{pmatrix} = \begin{pmatrix} -a + h(a - (Qa + e\mu - y)) \\ -e^T a \end{pmatrix}$$

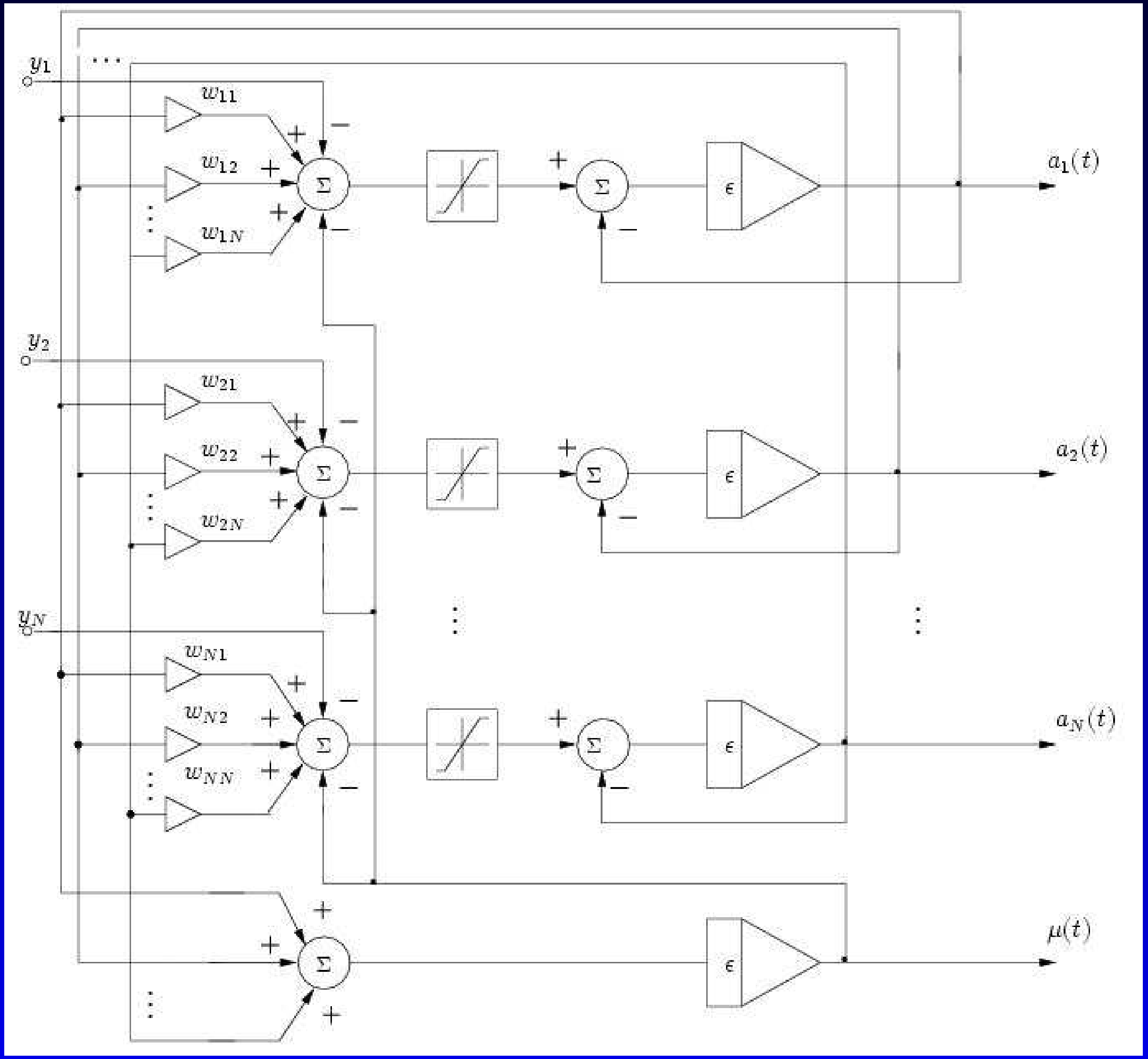
where $\epsilon > 0$, $a \in \Re^N$, and $\mu \in \Re$, $e = (1, \dots, 1)^T$.

$$\begin{aligned} \epsilon\frac{da_i}{dt} &= -a_i + h(\sum_{k=1}^{N} w_{ik}a_k - \mu + y_i), i = 1, ..., N; \\ \epsilon\frac{d\mu}{dt} &= -\sum_{k=1}^{N} a_k, \end{aligned}$$

where $Q = [q_{ij}] = [K(x^{(i)}, y^{(j)})]$, $w_{ik} = \delta_{ik} - q_{ik}$.[a]

[a] Y. Xia and J. Wang, "A one-layer recurrent neural network for support vector machine learn-

Network Architecture



Iris Benchmark Problem

The data of the iris problem are characterized with four attributes (i.e., the petal length and width, setal length and width).

The dataset consists of 150 samples belonging to three classes (i.e., viginica, versilcolor, setosa), each class has 50 samples.

120 samples for training and the remaining 30 for testing.

We use $c = 0.25$ and the polynomial kernel function $K(x, y) = (x^T y + 1)^p$, with $p = 2$ and $p = 4$.

Simulation Results

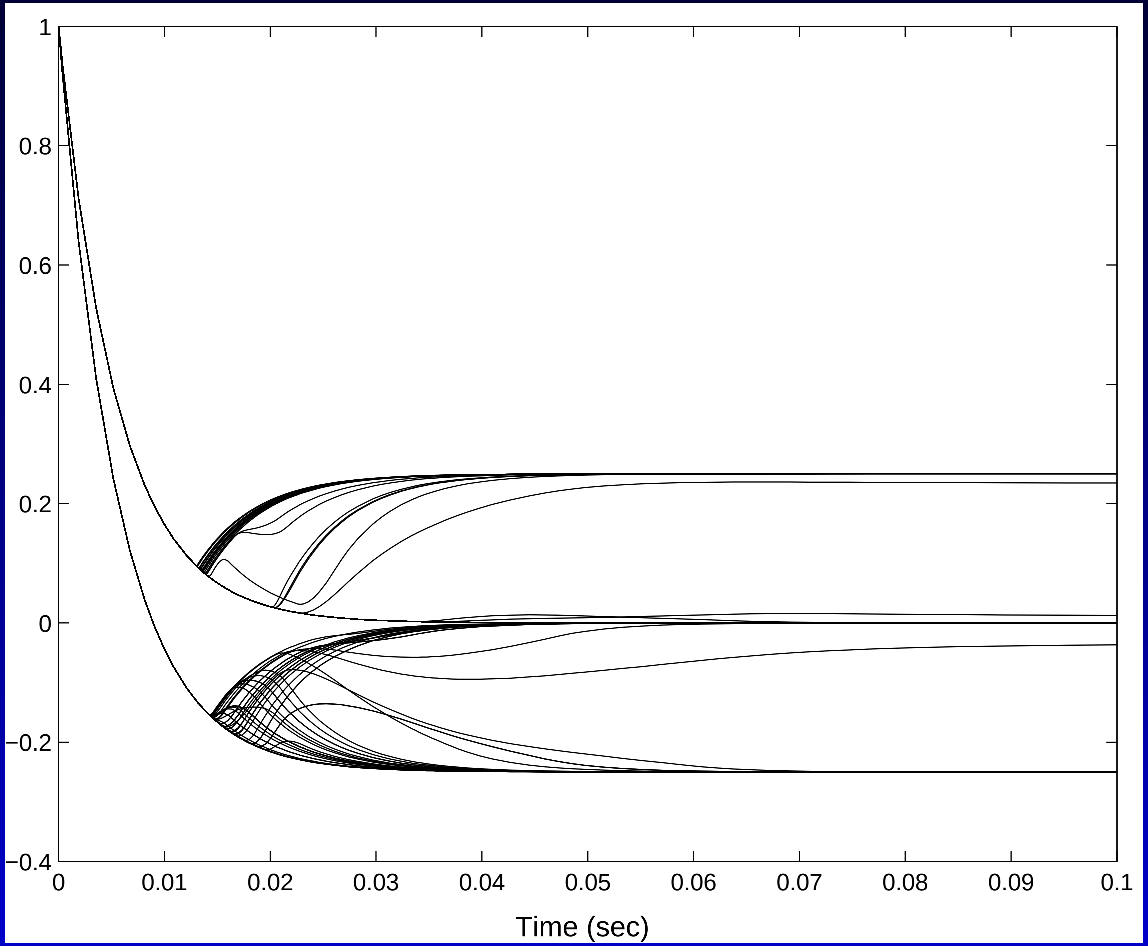


Figure 1: Convergence of the SVM Learning neural network with $\epsilon = 1/150$ and $p = 2$

Simulation Results

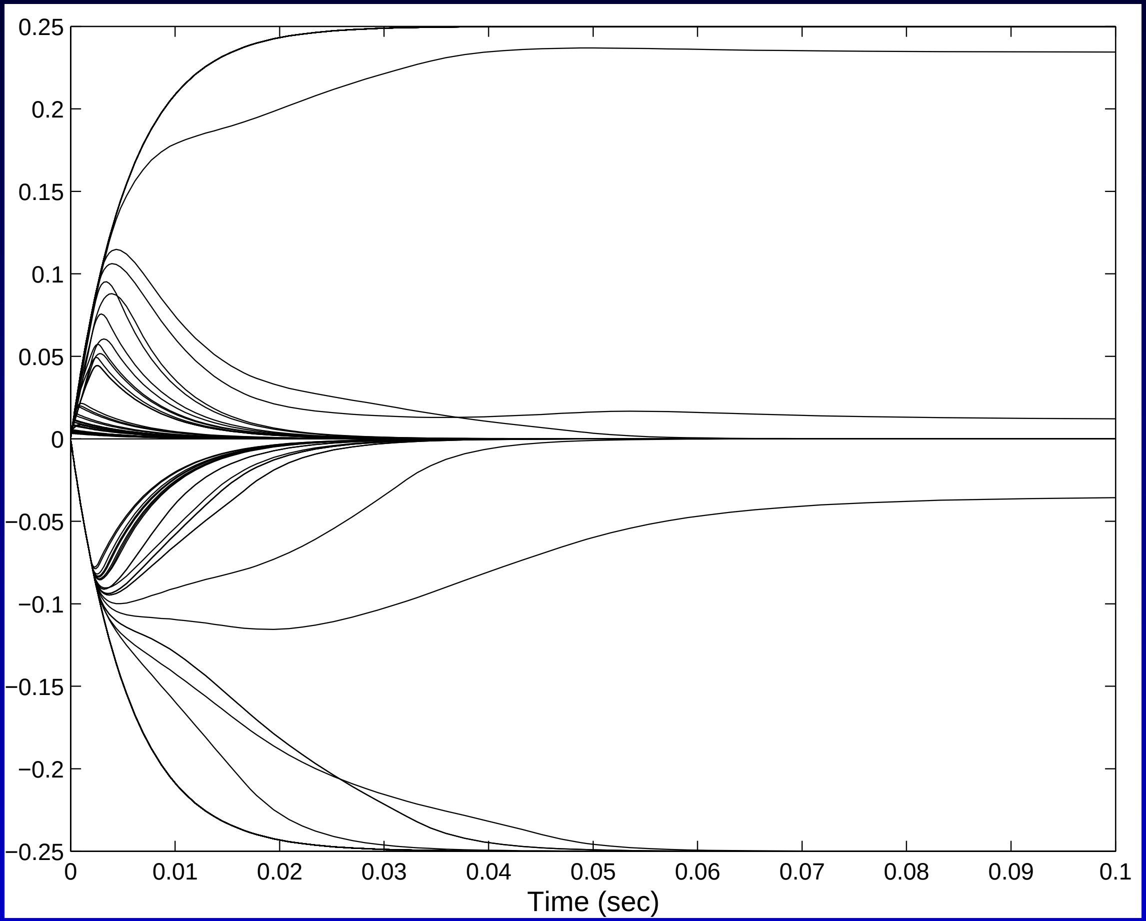


Figure 2: Convergence of the proposed neural network with $\epsilon = 1/150$ and $p = 4$

Simulation Results

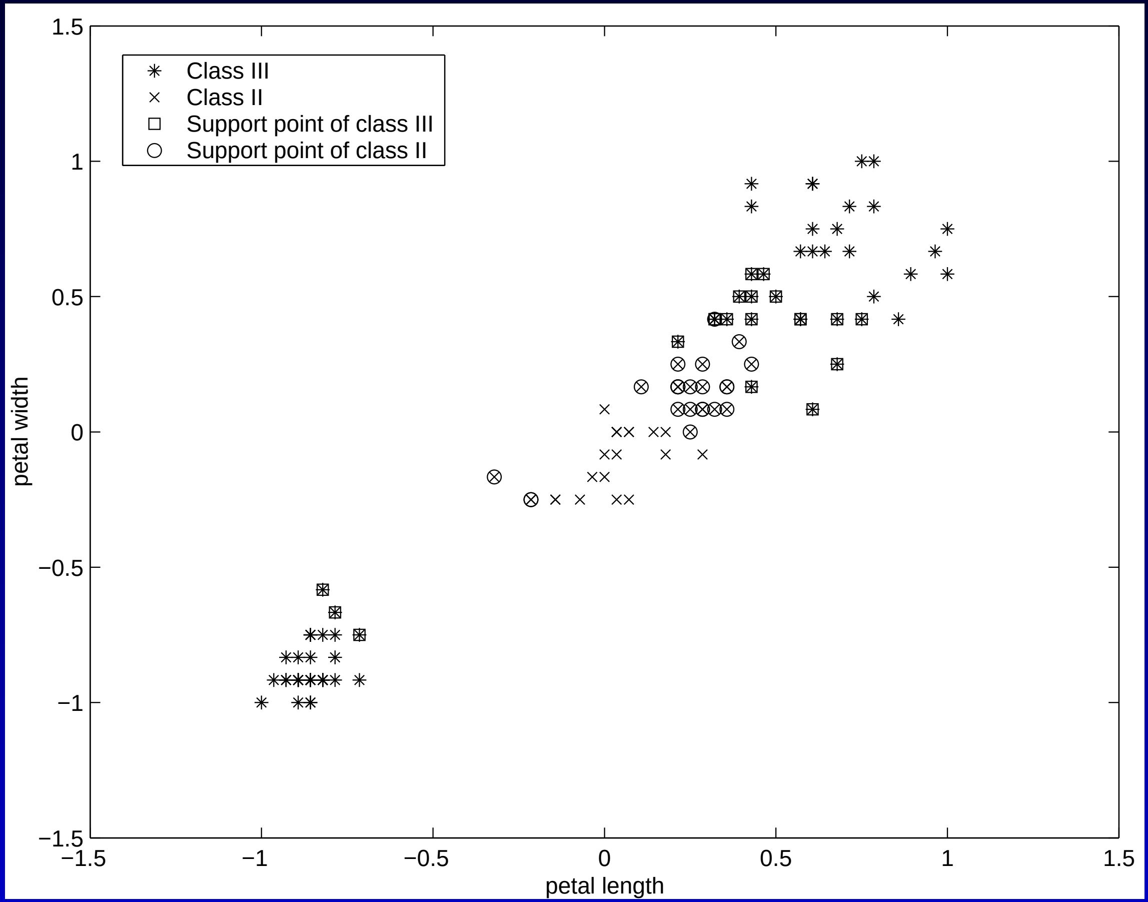


Figure 3: Support vectors of SVC using the proposed neural network with a polynomial kernel $p = 2$

Simulation Results

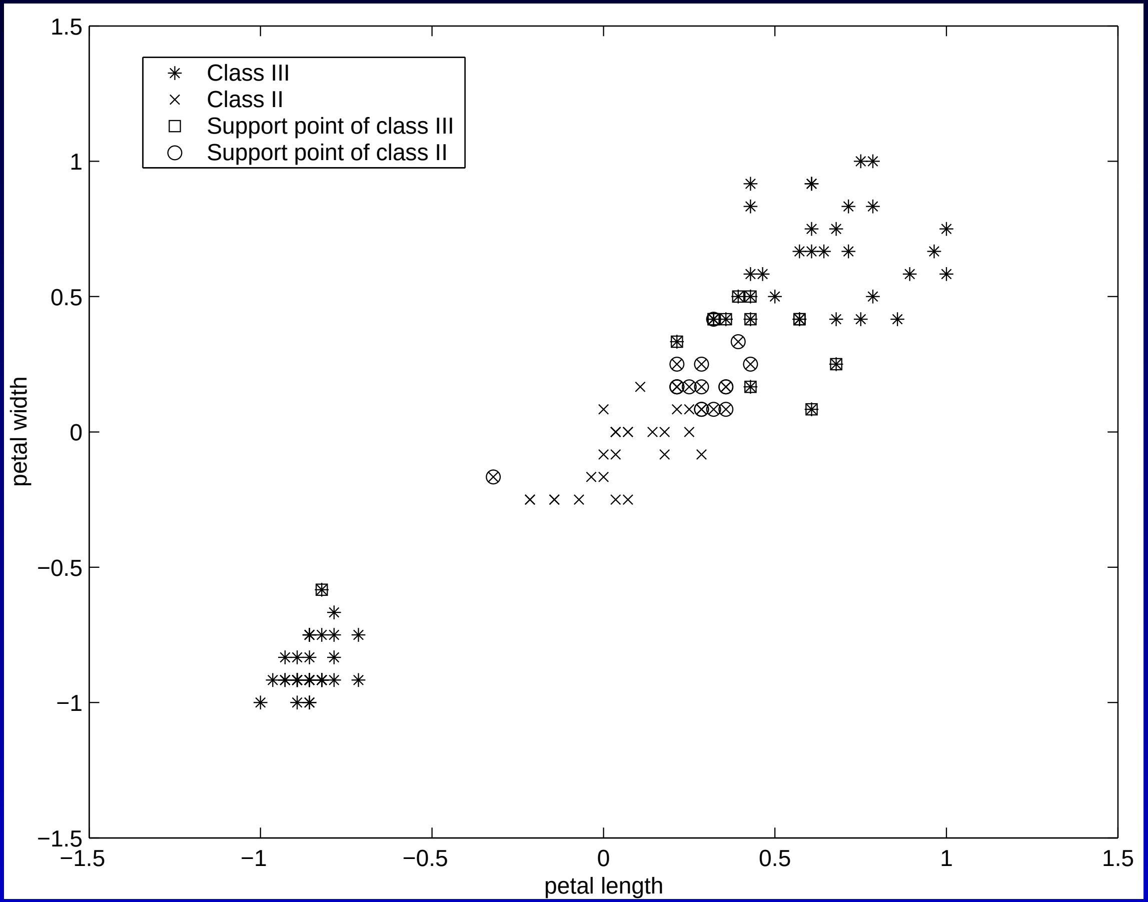


Figure 4: Support vectors of SVC using the proposed neural network with a polynomial kernel and $p = 4$

Adult Benchmark Problem

The UCI adult benchmark task is to predict whether a household has an income greater than $50,000 based on 14 other fields in a census form.

Eight of those fields are categorical, while six are continuous. The six fields are quantized into quintile, which yields a total of 123 sparse binary features.

1605 training samples and 2000 testing samples.

Gaussian RBF kernel with width of 10 and $c = 0.5$.

Let $\epsilon = 0.1$, and the initial point $\mathbf{z_0} \in^{1606}$ with the element being 1.

Adult Benchmark Problem

Method	iterations	SVs	Testing accuracy
SOR	924	635	84.06
SMO	3230	633	84.06
SVM-light	294	634	84.25
NN	567	633	84.15

Table 2: Comparisons of results of the SOR, SMO, SVM-light, and proposed neural network algorithm

Support Vector Regression (SVR)

Consider the regression problem of approximating a set of data

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

with a regression function as

$$\phi(x) = \sum_{i=1}^{N} \alpha_i \Phi_i(x) + \varsigma,$$

where $\Phi_i(x)(i = 1, 2, \ldots, n)$ are the feature functions defined in a high-dimensional space, $\alpha_i(i = 1, 2, \ldots, n)$ and ς are parameters of the model to be estimated.

SVR (cont'd)

By utilizing Huber loss function, the above regression function can be represented as

$$\phi(x) = \sum_{i=1}^{N} \theta_i K(x, x_i) + \varsigma, \qquad (5)$$

where $K(x, y)$ is a kernel function satisfying $K(x, y) = \Phi(x)^T \Phi(x)$.

SVR (cont'd)

θ_i $(i = 1, 2, \ldots, N)$ can be obtained from the following quadratic program:

$$\begin{aligned} \min \quad & \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\theta_i\theta_j K(x_i, x_j) - \sum_{i=1}^{N}\theta_i y_i + \frac{\varepsilon}{2\mu}\sum_{i=1}^{N}\theta_i^2, \\ \text{s.t.} \quad & \sum_{i=1}^{N}\theta_i = 0, \\ & -\mu \le \theta_i \le \mu, \quad i = 1, 2, \ldots, N; \end{aligned}$$

where $\varepsilon > 0$ is an accuracy parameter required for the approximation, $\mu > 0$ is a pre-specified parameter.

SVR (cont'd)

The neural network with a discontinuous activation function for solving the above quadratic program:

$$\begin{aligned} \epsilon \frac{dz}{dt} &= -Pz + [PQ + \frac{\alpha}{N} ee^T] g(z) + q, \\ \theta &= (PQ + \frac{\alpha}{N} ee^T)^{-1} (Pz - q), \end{aligned}$$

where $e = [1, 1, \ldots, 1]^T$,
$P = I - ee^T/N, Q = \{K(x_i, x_j)\}_{N \times N} + \varepsilon I/\mu$,
$q = (I - ee^T/N)y$ with $y = -(y_1, y_2, \ldots, y_n)$, and
$h = -l = \mu e$ in the activation function.

SVR (cont'd)

Moreover, ς can be obtained from

$$\varsigma = -\frac{1}{N}(e^T(Q-I)\theta^* + e^T c - e^T z^*),$$

where z^* is an equilibrium point and θ^* is an output vector corresponding to z^*.

Compared with existing neural networks for SVM learning, the existing neural networks need either two-layer structure and $n+1$ neurons.

In contrast, the neural network herein has one-layer structure and n neurons only.

SVR (cont'd)

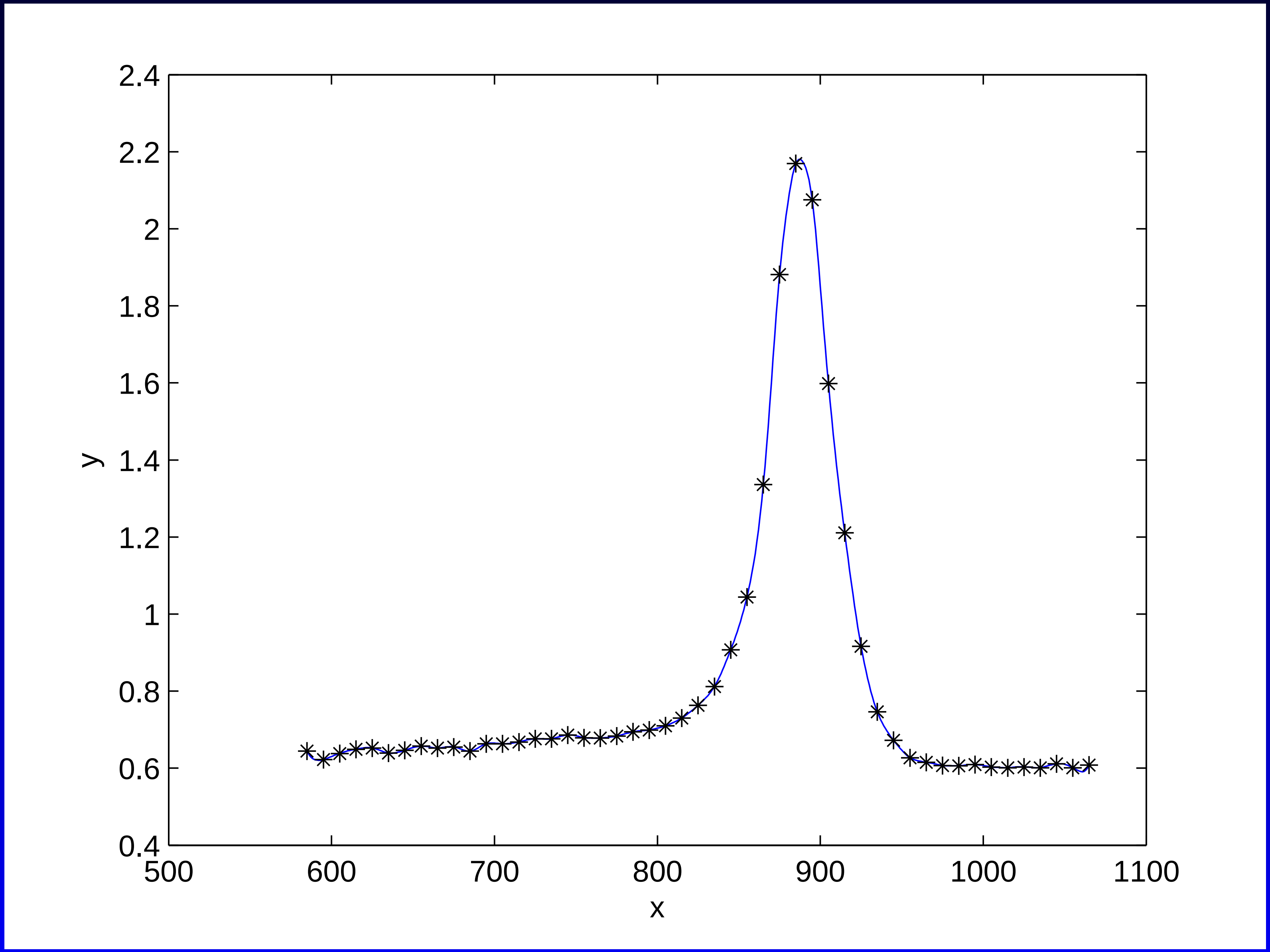
For the SVR learning by using the proposed neural network based on titanium regression data[a]. Let the kernel be a Gaussian function:

$$K(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right)$$

$\varepsilon = 0.01$, $\mu = 100$ and $\sigma = 6$.

[a] P. Dierckx, *Curve and Surface Fitting with Splines*, Clarendon Press, Oxford, 1993.

Regression Result



Inverse Kinematics Problem

Because $\dot{\theta}$ is underdetermined in a kinematically redundant manipulator, one way to determine $\dot{\theta}(t)$ without the need for computing the pseudoinverse is to solve:

$$\begin{aligned}
\text{minimize} \quad & \frac{1}{2}\dot{\theta}(t)^T W \dot{\theta}(t) + c^T \dot{\theta}(t), \\
\text{subject to} \quad & J(\theta(t))\dot{\theta}(t) = \dot{x}_d(t), \\
& \eta^- \leqslant \dot{\theta} \leqslant \eta^+
\end{aligned}$$

where W is a positive-definite weighting matrix, c is an column vector, and $\eta^\pm$ are upper and lower bounds of the joint velocity vector.

Lagrangian Network Dynamics

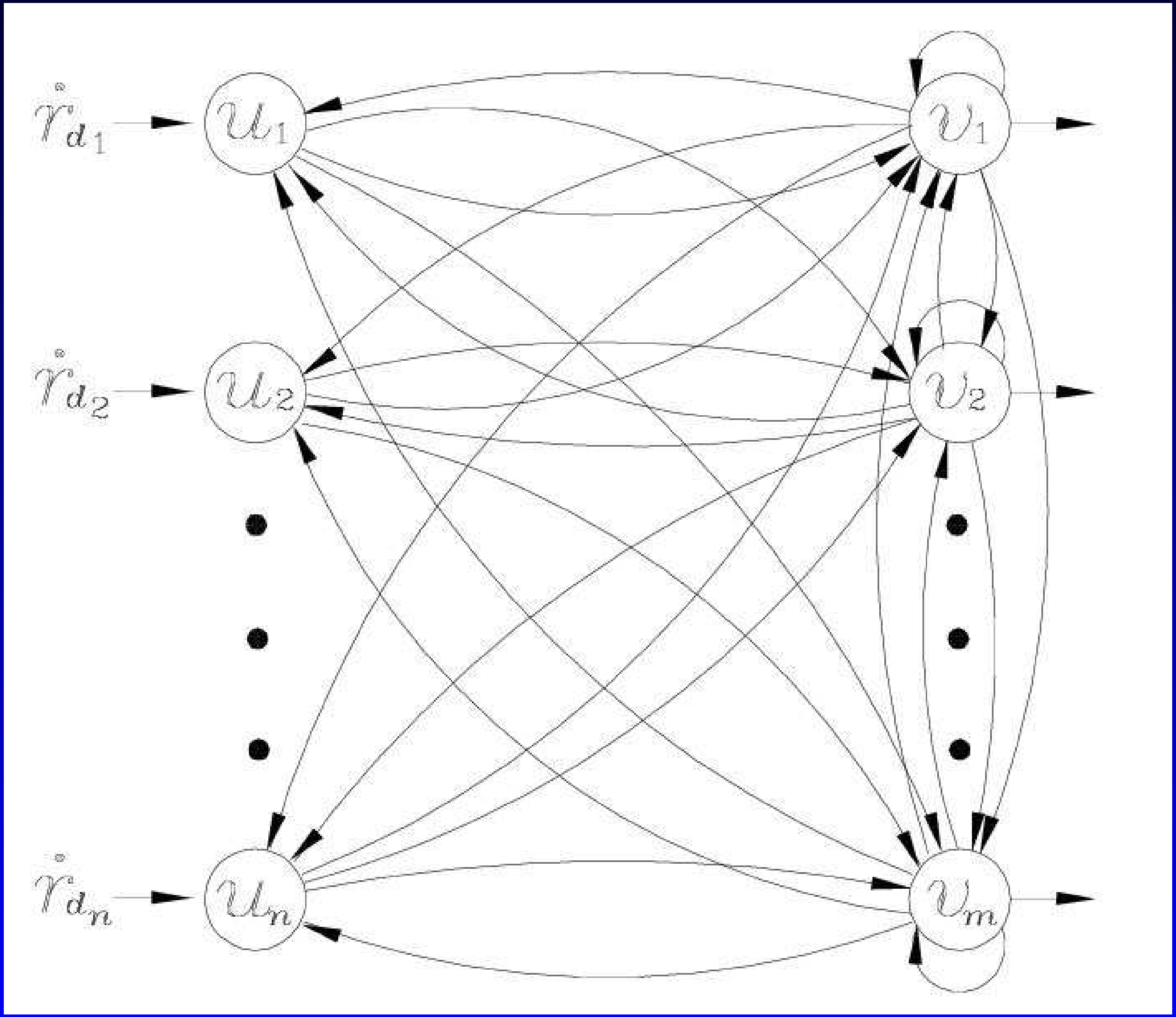
Let the state vectors of output neurons and hidden neurons be denoted by $v(t)$ and $u(t)$, representing estimated $\dot{\theta}(t)$ and estimated $\lambda(t)$, respectively.

The dynamic equation of the two-layer Lagrangian network can be expressed as:

$$\begin{aligned} \epsilon_1 \frac{dv(t)}{dt} &= -Wv(t) - J(\theta(t))^T u(t) - c, \\ \epsilon_2 \frac{du(t)}{dt} &= J(\theta(t))v(t) - \dot{x}_d(t), \end{aligned}$$

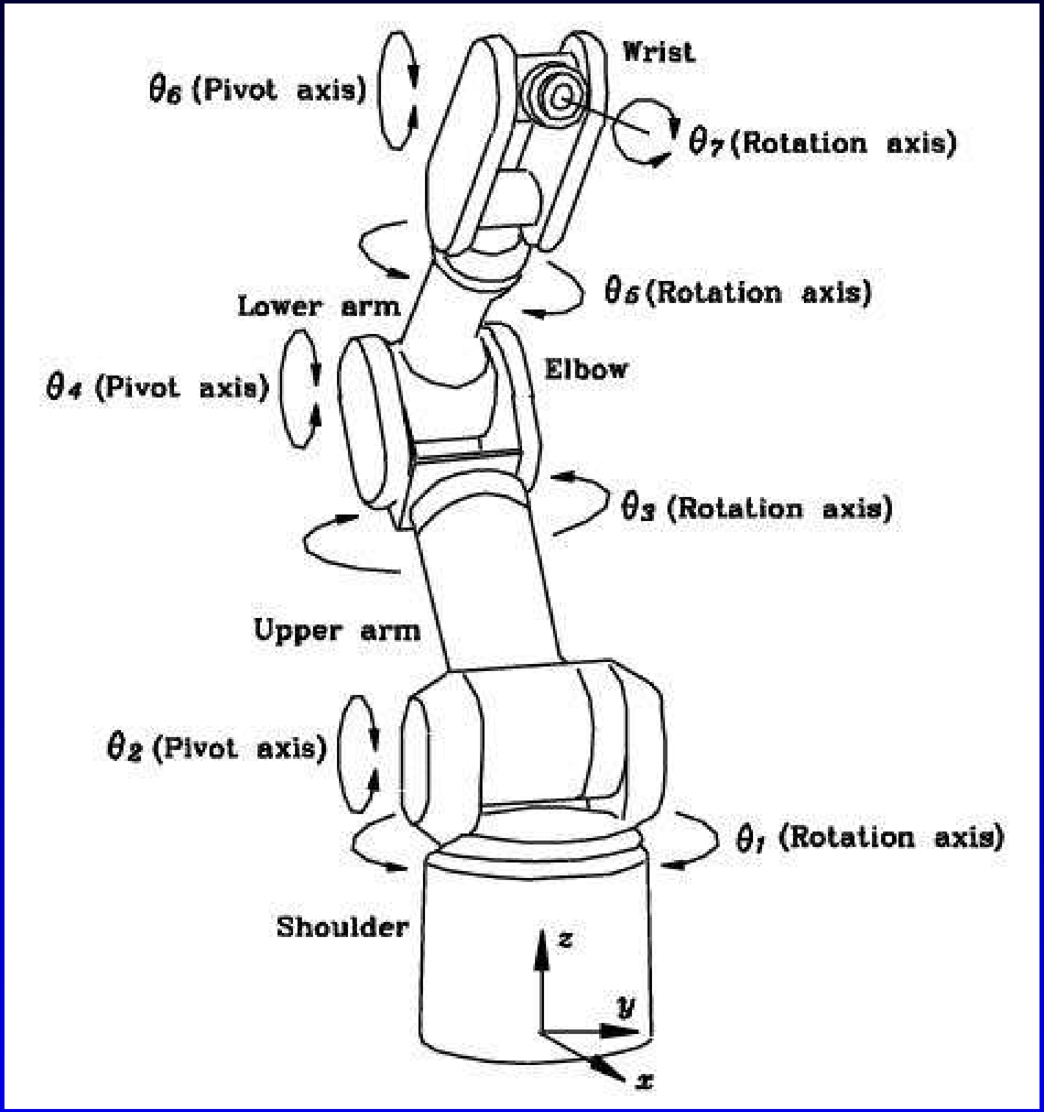
where $\epsilon_1 > 0$ and $\epsilon_2 > 0$.

Lagrangian Network Architecture

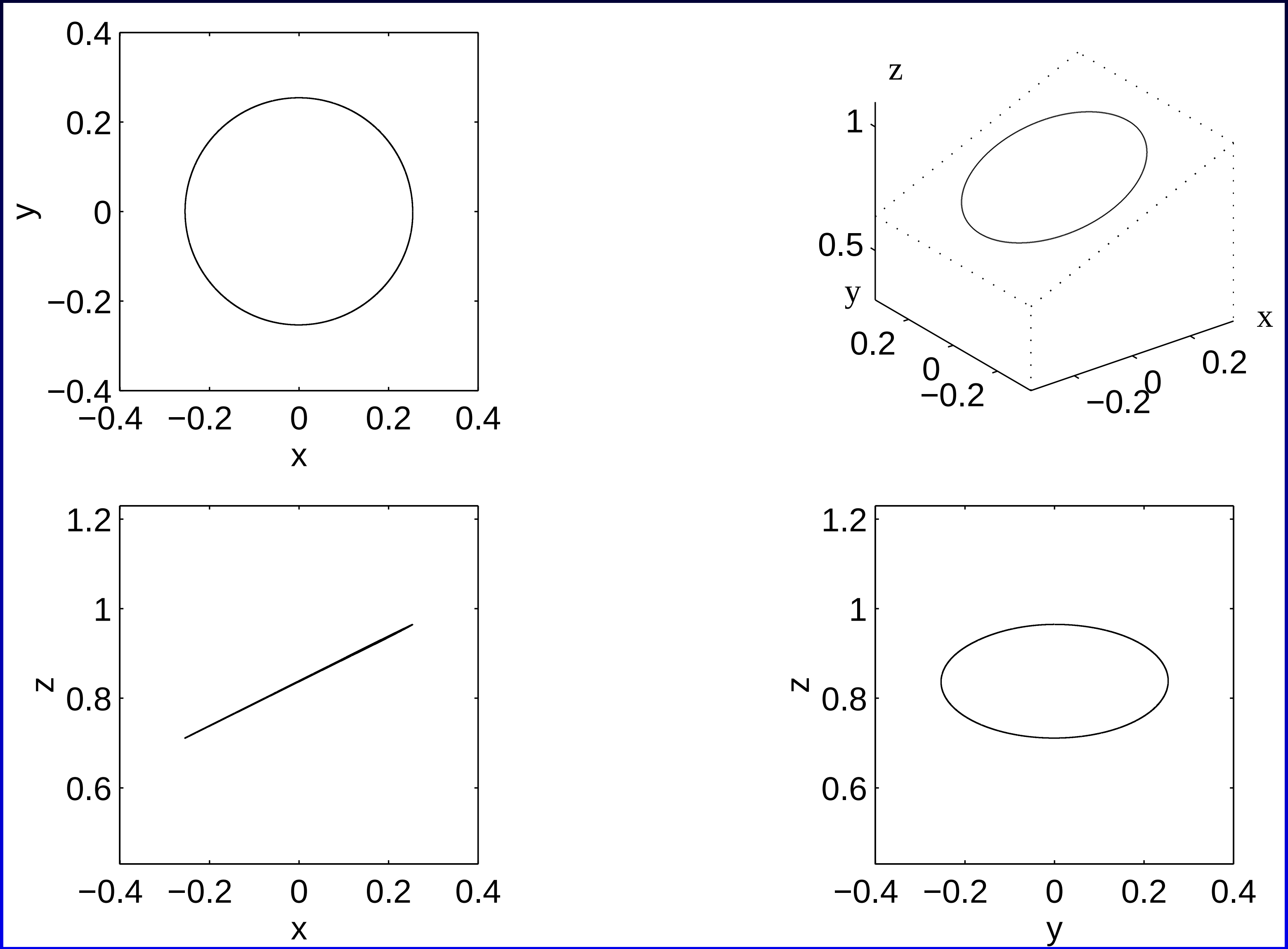


7-DOF PA10 Manipulator

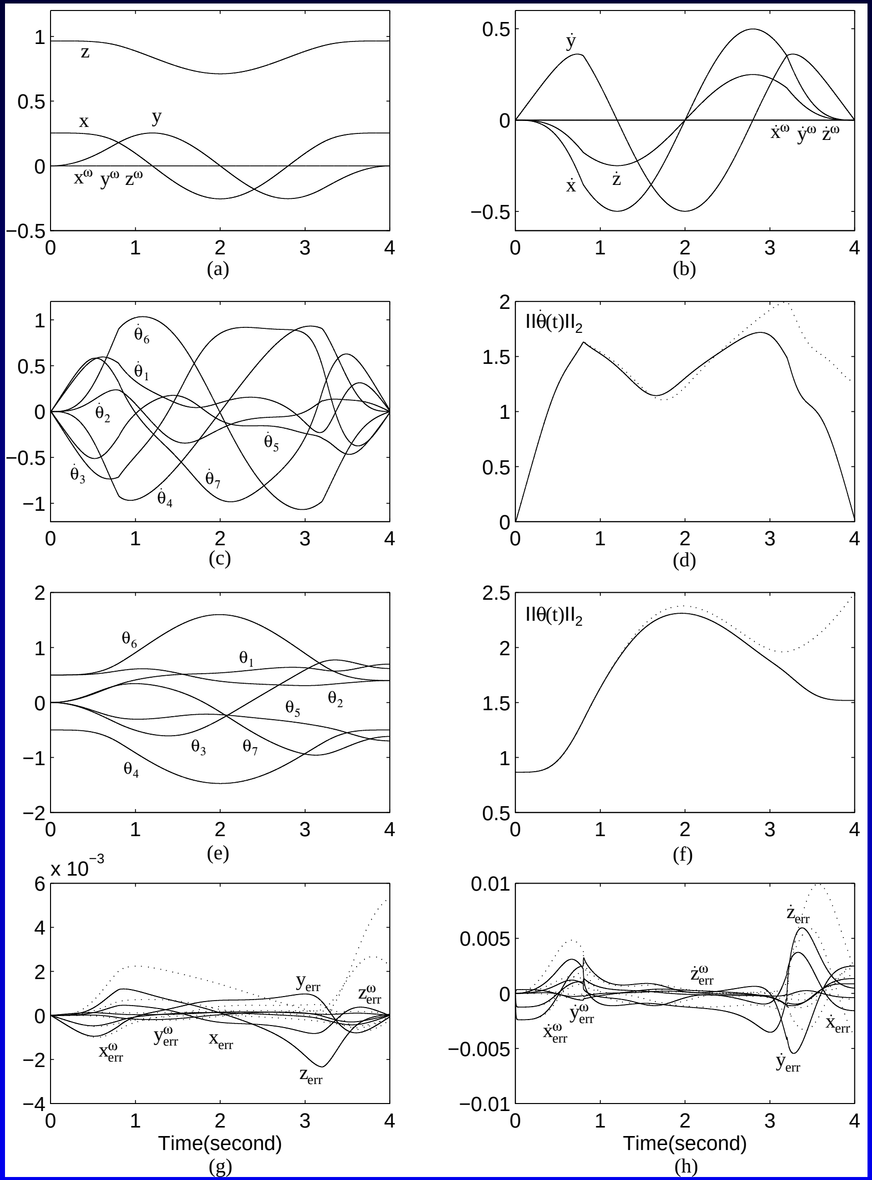
Coordinate system of PA10 manipulator



Circular Motion of the PA10 Manipulator



Simulation Results



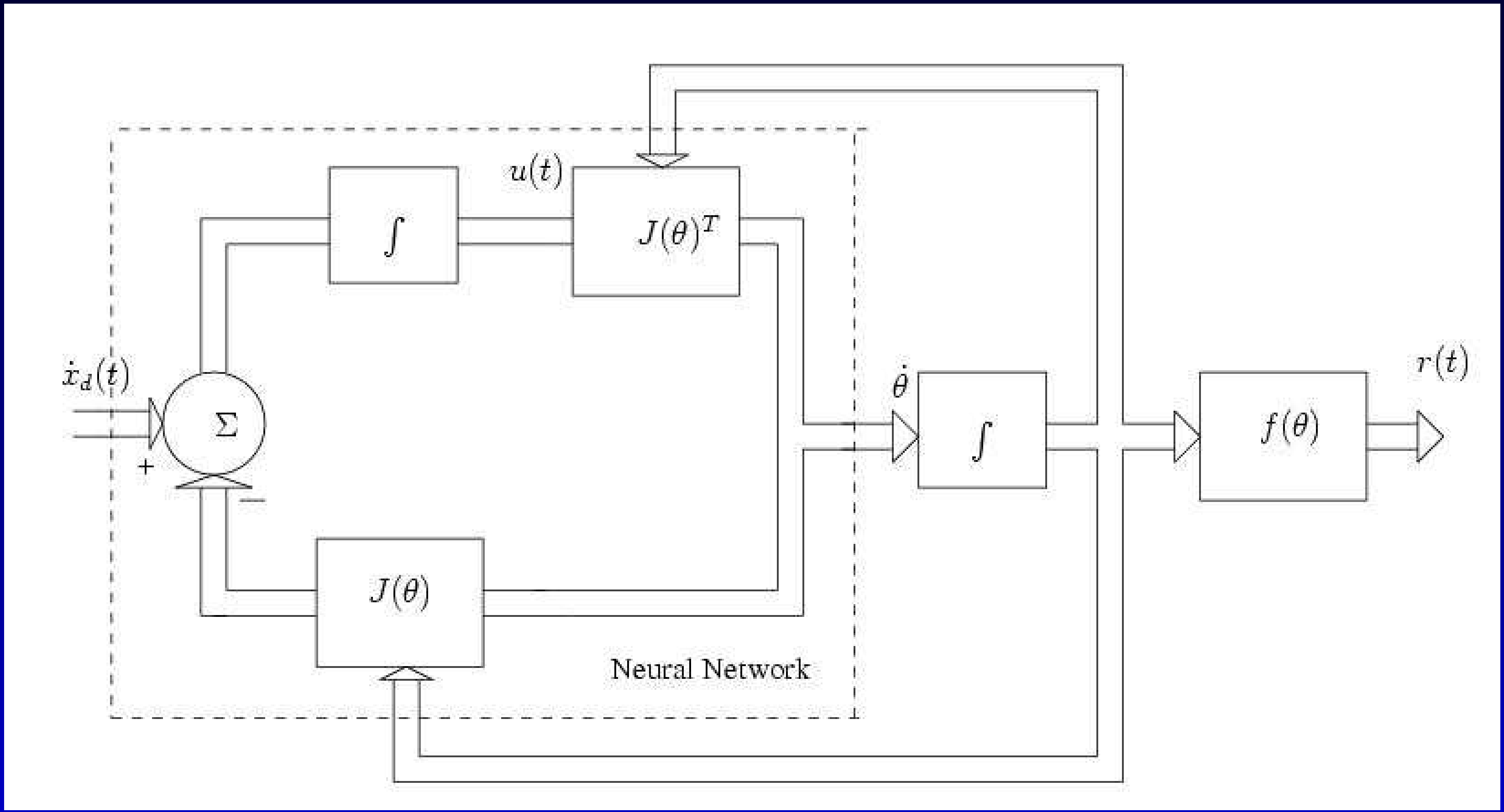
Dual Network Dynamics

To reduce the number of neurons to minimum, next we propose a dual neural network with its dynamic equation and output equation defined as

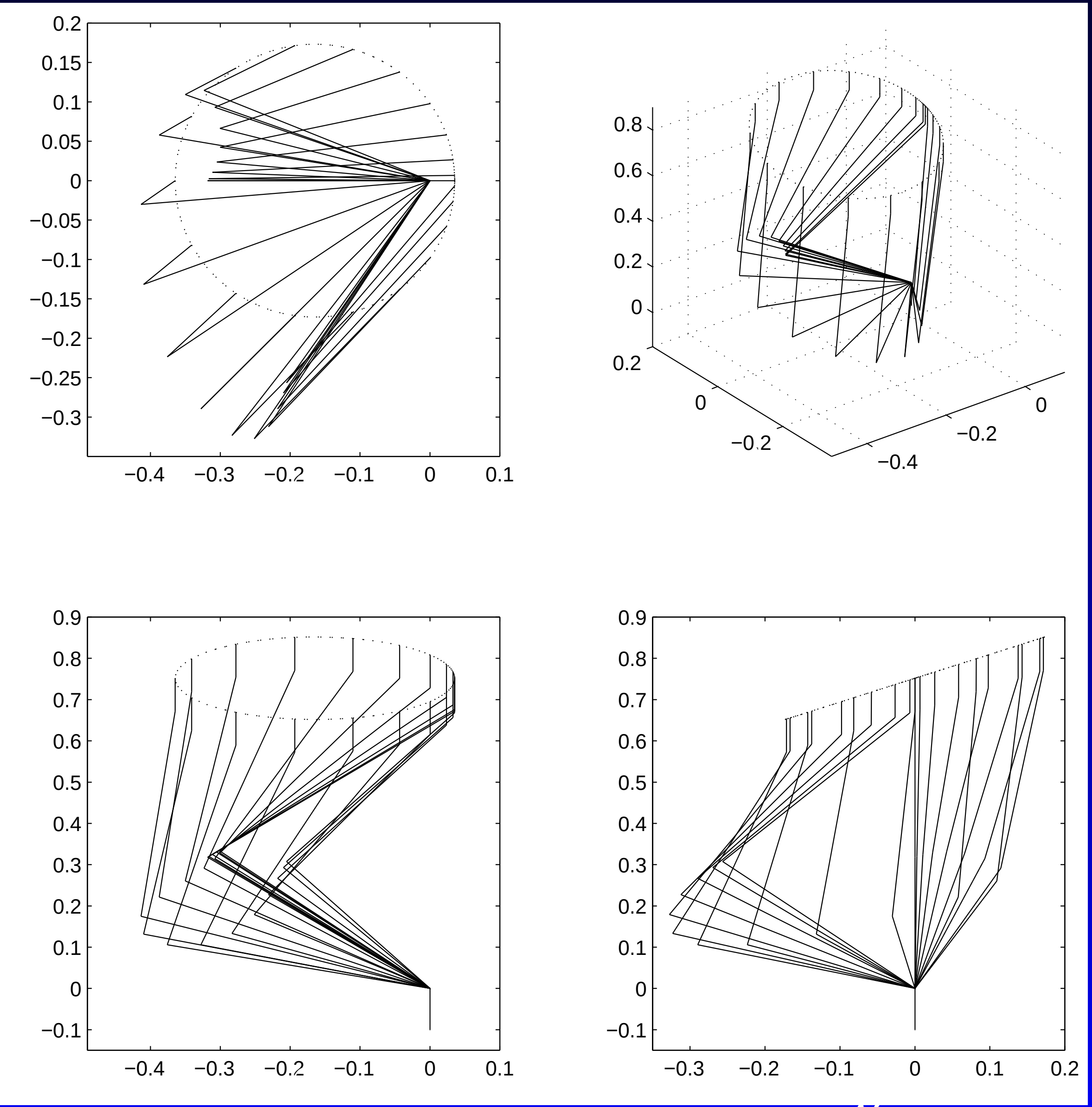
$$
\begin{aligned}
\epsilon\frac{du(t)}{dt} &= -J(\theta(t))W^{-1}J^T(\theta))u + \dot{x}_d(t), \\
v(t) &= J^T(\theta(t))u(t);
\end{aligned}
$$

where u is the dual state variable, v is the output variable.
The Lagrangian network contains $n + m$ neurons. But the dual network contains only m neurons, where n is the number of joints and m is the dimension of the cartesian space (i.e., 6).

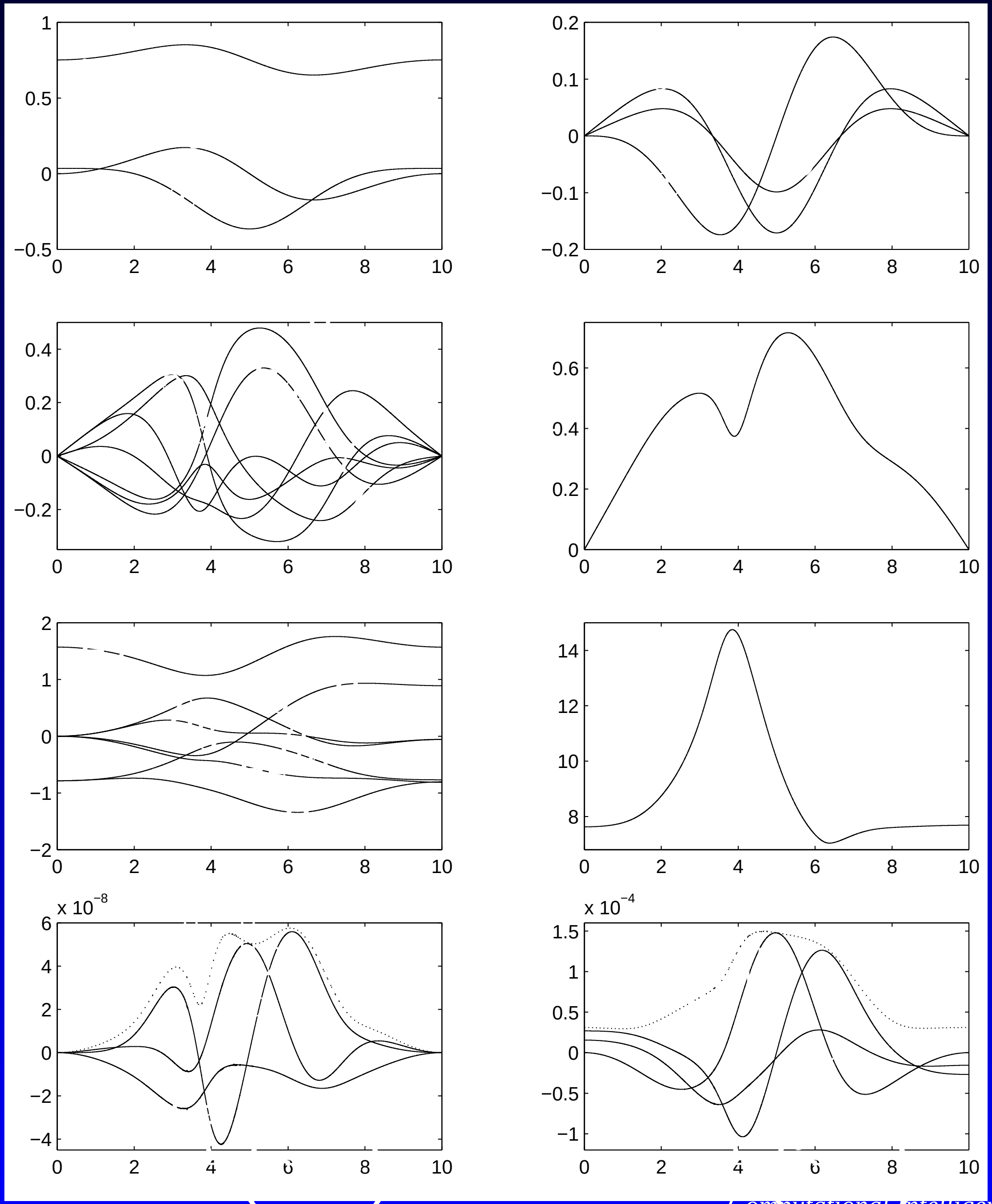
Dual Network



Simulation Results



Simulation Results



Bounded Inverse Kinematics

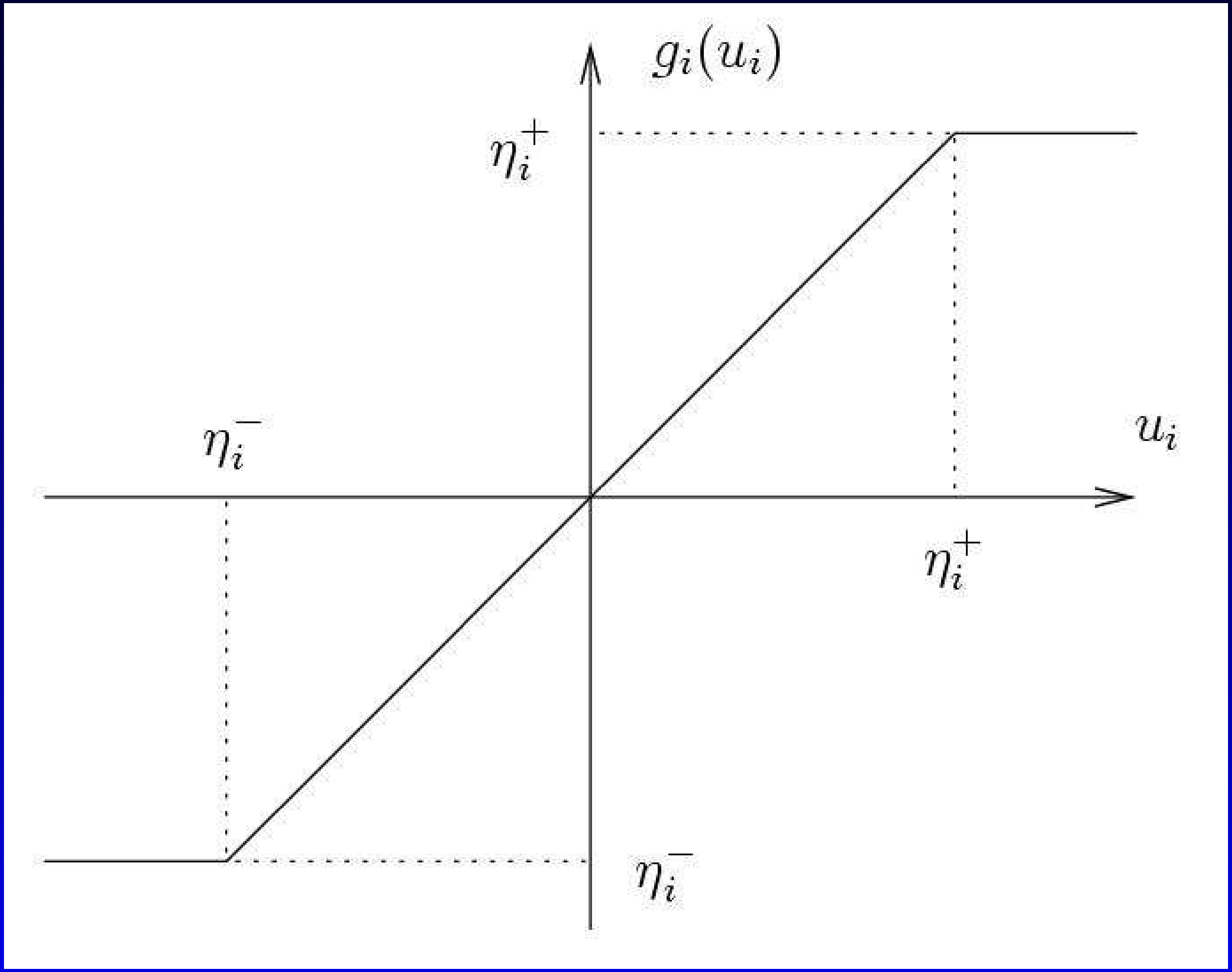
The dual neural network with the following dynamic equation and output equation

$$\begin{aligned} \epsilon_1 \frac{dx}{dt} &= -JW^{-1}J^T x + \dot{x}_d \\ \epsilon_2 \frac{dy}{dt} &= -W^{-1}y + g((W^{-1} - I)y) \\ v &= J^T x + y \end{aligned}$$

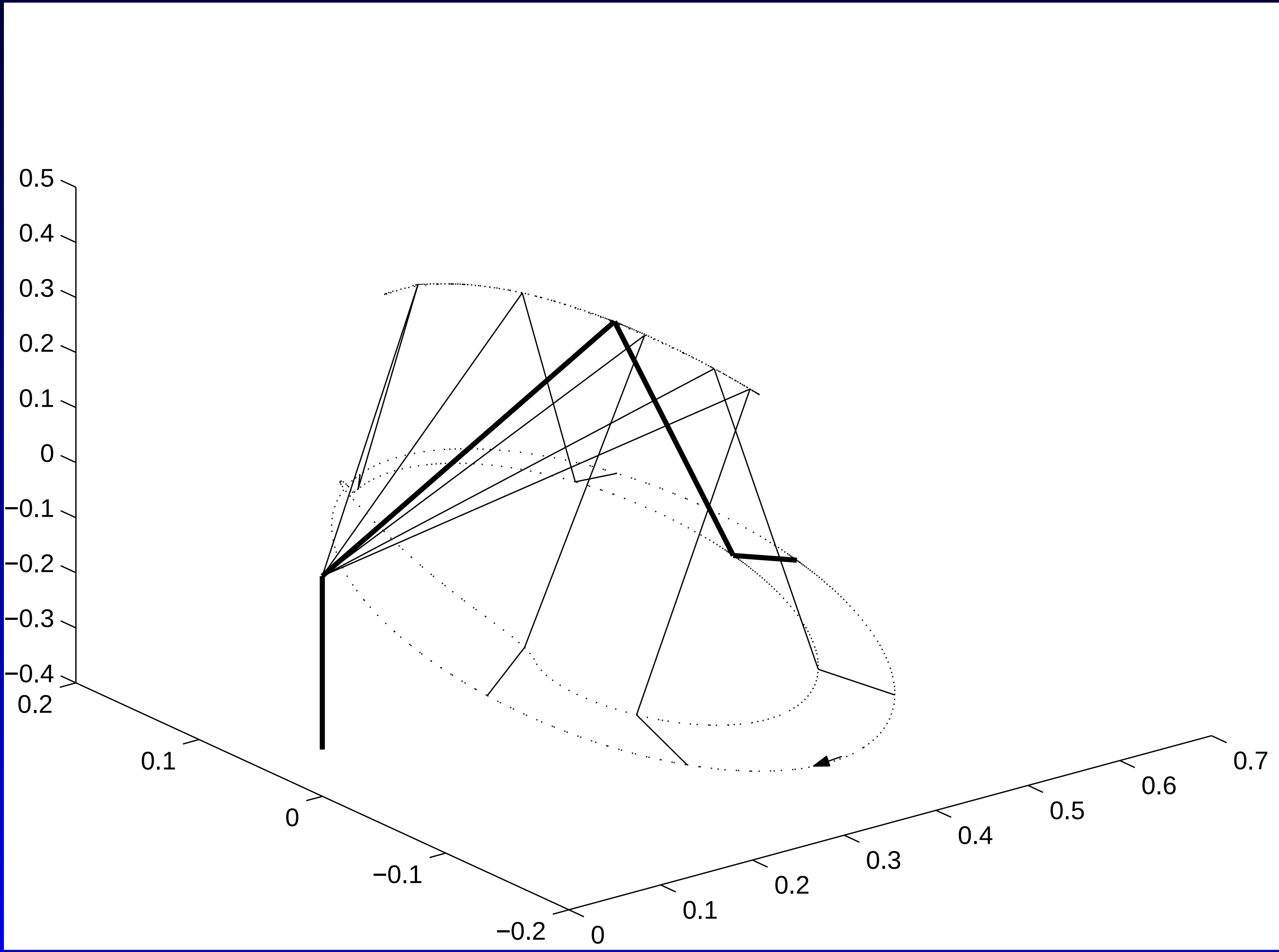
where the piecewise linear activation function

$$g_i(u_i) = \begin{cases} \eta_i^-, & \text{if } u_i < \eta_i^- \\ u_i, & \text{if } \eta_i^- \leqslant u_i \leqslant \eta_i^+ \\ \eta_i^+, & \text{if } u_i > \eta_i^+ \end{cases}$$

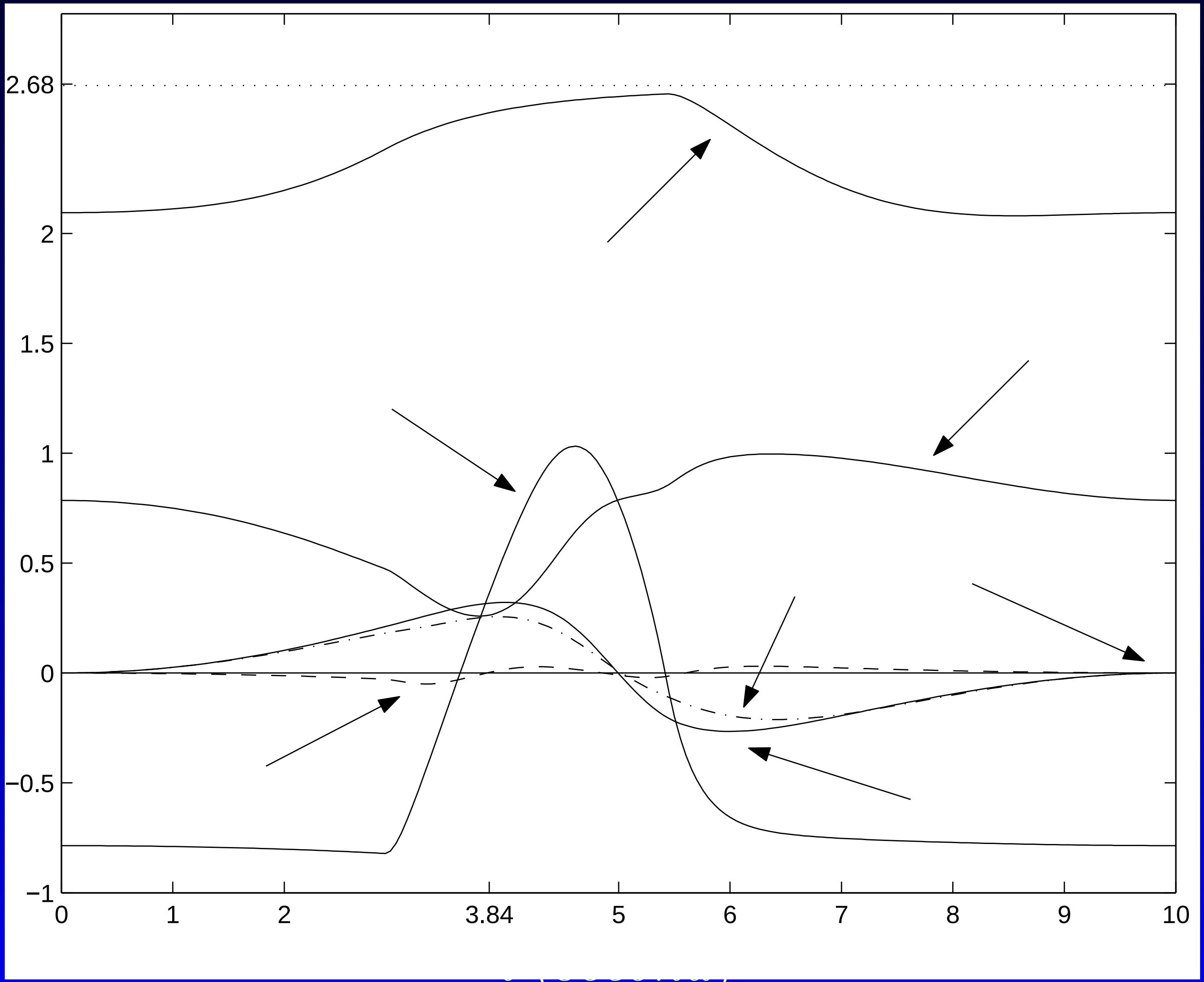
Piecewise Linear Activation Function



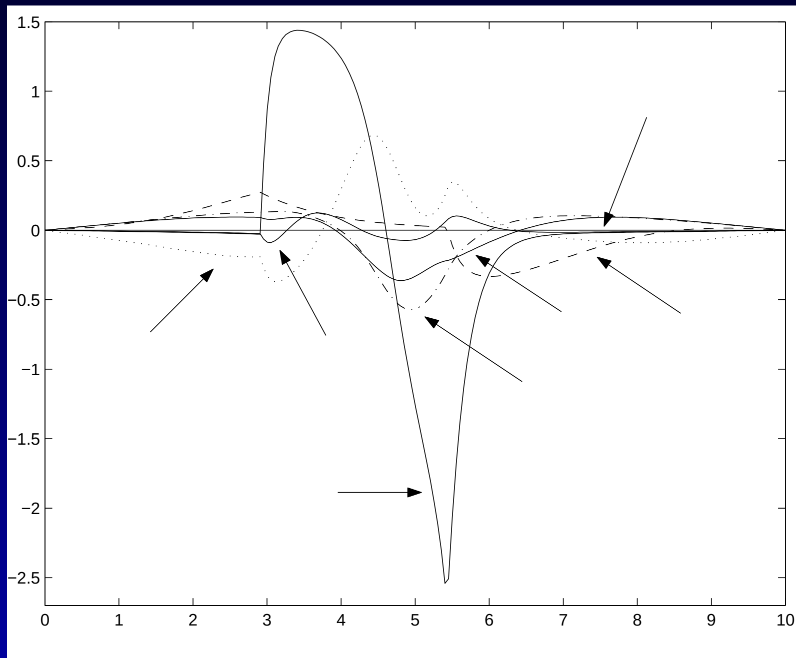
PA10 Drift-free Circular Motion



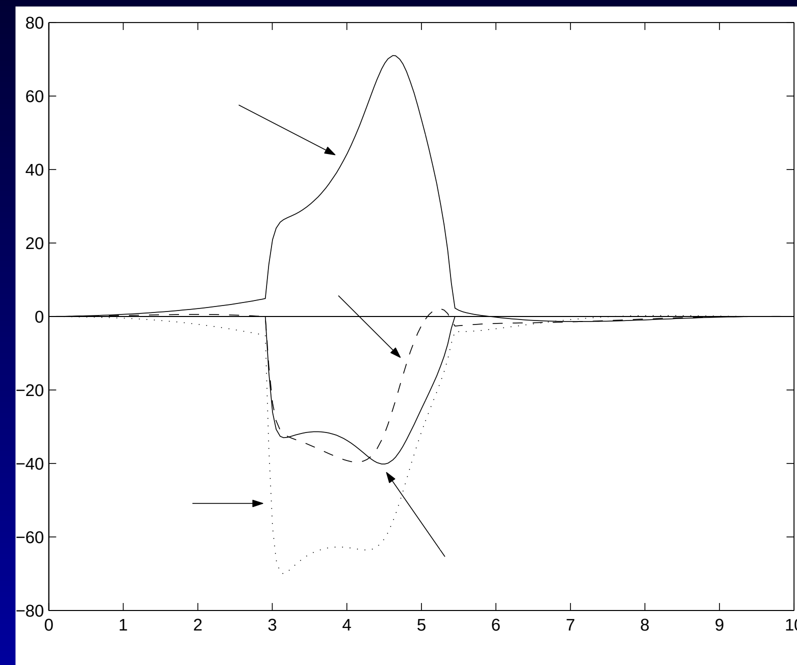
PA10 Joint Variables



Joint Velocities and Dual State Variables



(a) Joint rate variables in rad/sec



(b) Nonzero dual decision variables

Euclidean Norm vs. Infinity Norm

The Euclidean norm (or 2-norm) is widely used often because of its analytical tractability.

Minimizing the 2-norm of the joint velocities does not necessarily minimize the magnitudes of the individual joint velocities.

This is undesirable in situations where the individual joint velocities are of primary interest.

Minimizing the infinity norm of velocity variables can minimize the maximum velocity.

Inverse Kinematics Problem

Minimizing the infinity norm of $\dot{\theta}$ subject to the kinematic constraint:

$$\min_{\dot{\theta}} \left\|\dot{\theta}\right\|_\infty = \min_{\dot{\theta}} \max_{1\le j\le n} \quad |e_j^T \dot{\theta}|,$$

$$\text{s.t.} \;\; J(\theta(t))\dot{\theta}(t) = \dot{x}_d(t),$$

where e_j is the j-th column of the identity matrix.

Inverse Kinematics Problem

Let

$$s = \max_{1 \le j \le n} |e_j^T \dot{\theta}|.$$

The inverse kinamatic problem can be written as

$$\begin{aligned} \min_{\dot{\theta}_n} \quad & s \\ \text{s.t.} \quad & |e_j^T \dot{\theta}| \le s, \quad j = 1, 2, \ldots, n \\ & J(\theta(t))\dot{\theta}(t) = \dot{x}_d(t). \end{aligned}$$

Inverse Kinematics Problem Re-formulation

The inverse kinematics problem can be summarized in a matrix form:

$$
\begin{array}{ll}
\min & s \\
\text{s.t.} & \begin{bmatrix} -I & I_n \\ I & I_n \end{bmatrix} \begin{bmatrix} \dot{\theta} \\ s \end{bmatrix} \geq \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\
& J(\theta)\dot{\theta} = \dot{x}_d(t),
\end{array}
$$

where $I_n = (1, 1, \ldots, 1)^T \in R^n$ and I is the identity matrix.

Primal Inverse Kinematics Problem Formulation

Let $y = (y_1^T, y_2)^T, y_1 = \dot{\theta}, y_2 = s$, then a final form of the problem can be derived as

$$\begin{aligned} \min \quad & c^T y \\ \text{s.t.} \quad & A_1 y \geq 0, \\ & A_2 y = b(t), \end{aligned}$$

where

$$A_1 = \begin{bmatrix} -I & I_n \\ I & I_n \end{bmatrix}, A_2(t) = [J(\theta(t), 0],$$

$$b(t) = \dot{x}_d(t), c^T = [0, 0, \ldots, 1].$$

Dual Inverse Kinematics Problem Formulation

The dual problem of the preceding linear program is defined as follows:

$$\begin{aligned} \max \quad & b^T z_2 \\ \text{s.t.} \quad & A_1^T z_1 + A_2^T z_2 = c, \\ & z_1 \geq 0, \end{aligned}$$

where $z = (z_1^T, z_2^T)^T$ is the dual decision variable.

Energy Function

An energy function to be minimized can be defined based on the primal and dual formulation:

$$
\begin{aligned}
E(y,z) = \quad & \frac{1}{2}(c^T y - bz_2)^2 + \frac{1}{2}\|A_2 y - b\|_2^2 + \\
& \frac{1}{2}\|A_1^T z_1 + A_2^T z_2 - c\|_2^2 + \\
& \frac{1}{4}(A_1 y)^T (A_1 y - |A_1 y|) + \\
& \frac{1}{4} z_1^T (z_1 - |z_1|).
\end{aligned}
$$

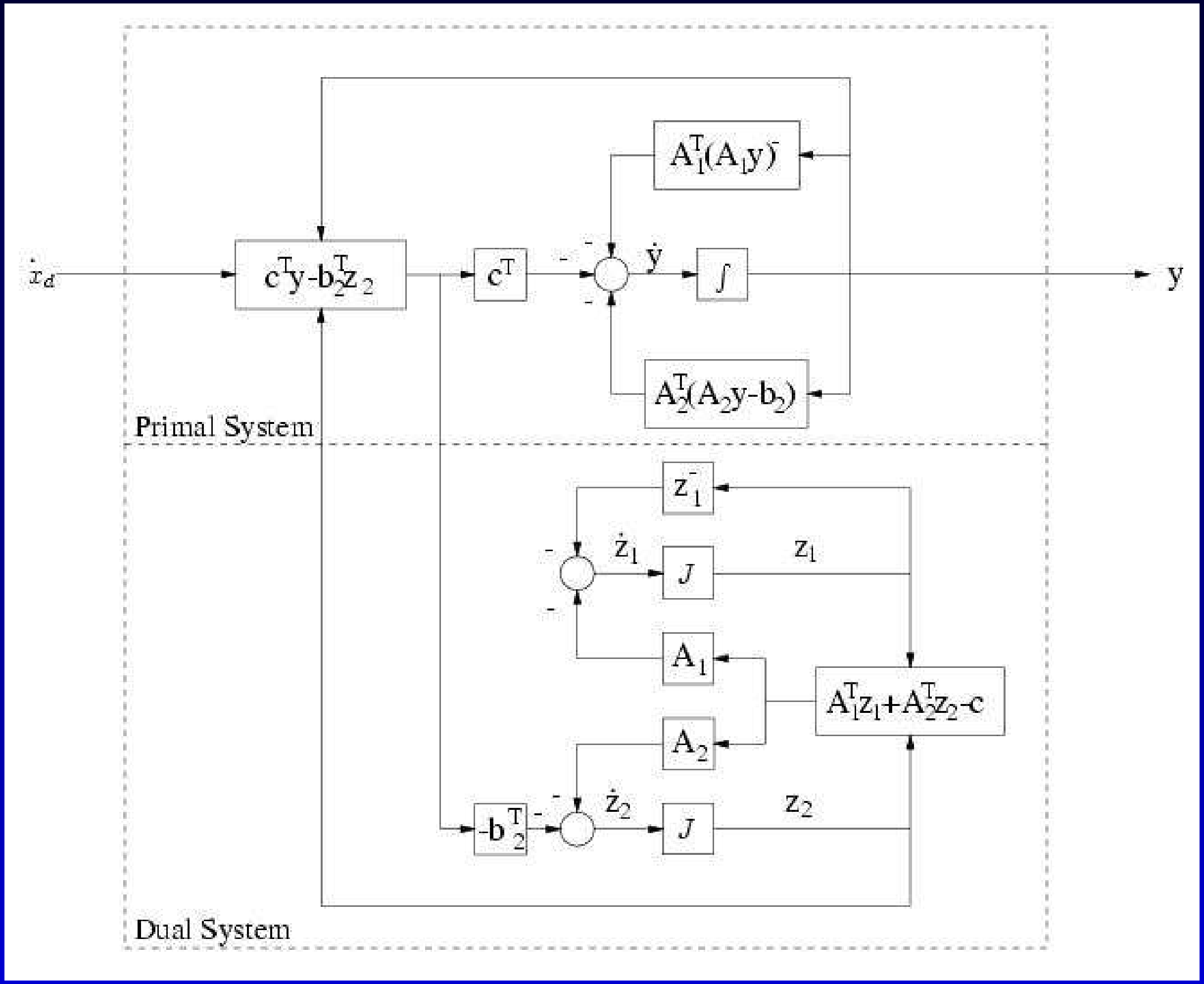
Primal-Dual Network

The dynamic equation of the primal-dual network:

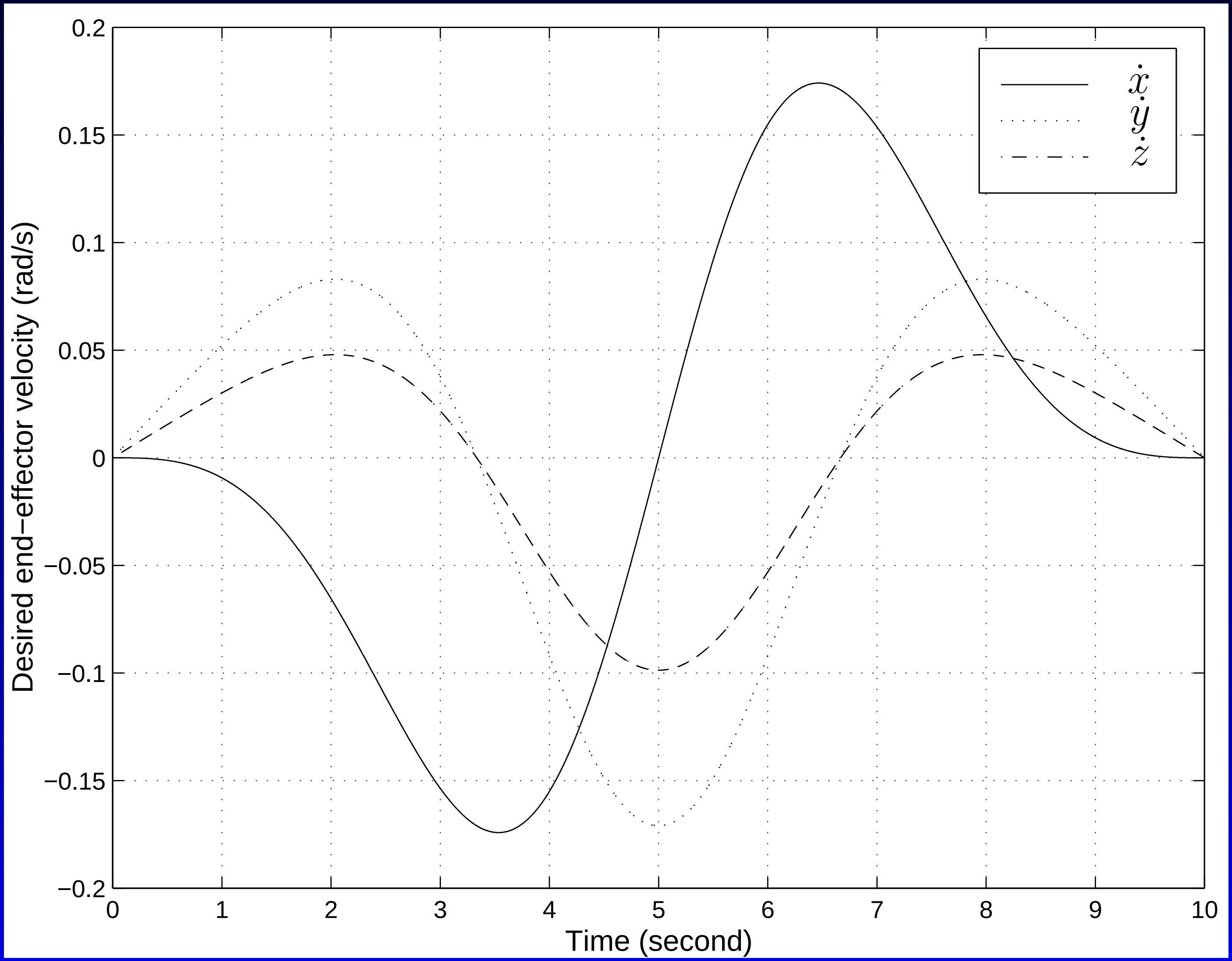
$$
\begin{aligned}
\epsilon_1 \dot{y} &= -c(c^T y - b^T z_2) + A_1^T h(-A_1 y) + \\
&\quad A_2^T (A_2 y - b), \\
\epsilon_2 \dot{z}_1 &= -h(-z_1) + A_1(A_1^T z_1 + A_2^T z_2 - c), \\
\epsilon_3 \dot{z}_2 &= -b(c^T y - b^T z_2) + A_2(A_1^T z_1 + A_2^T z_2 - c),
\end{aligned}
$$

where y, z_1, z_2, are state vectors; $h(x) = \max\{0, x\}$; and ϵ_i are positive scaling constants.

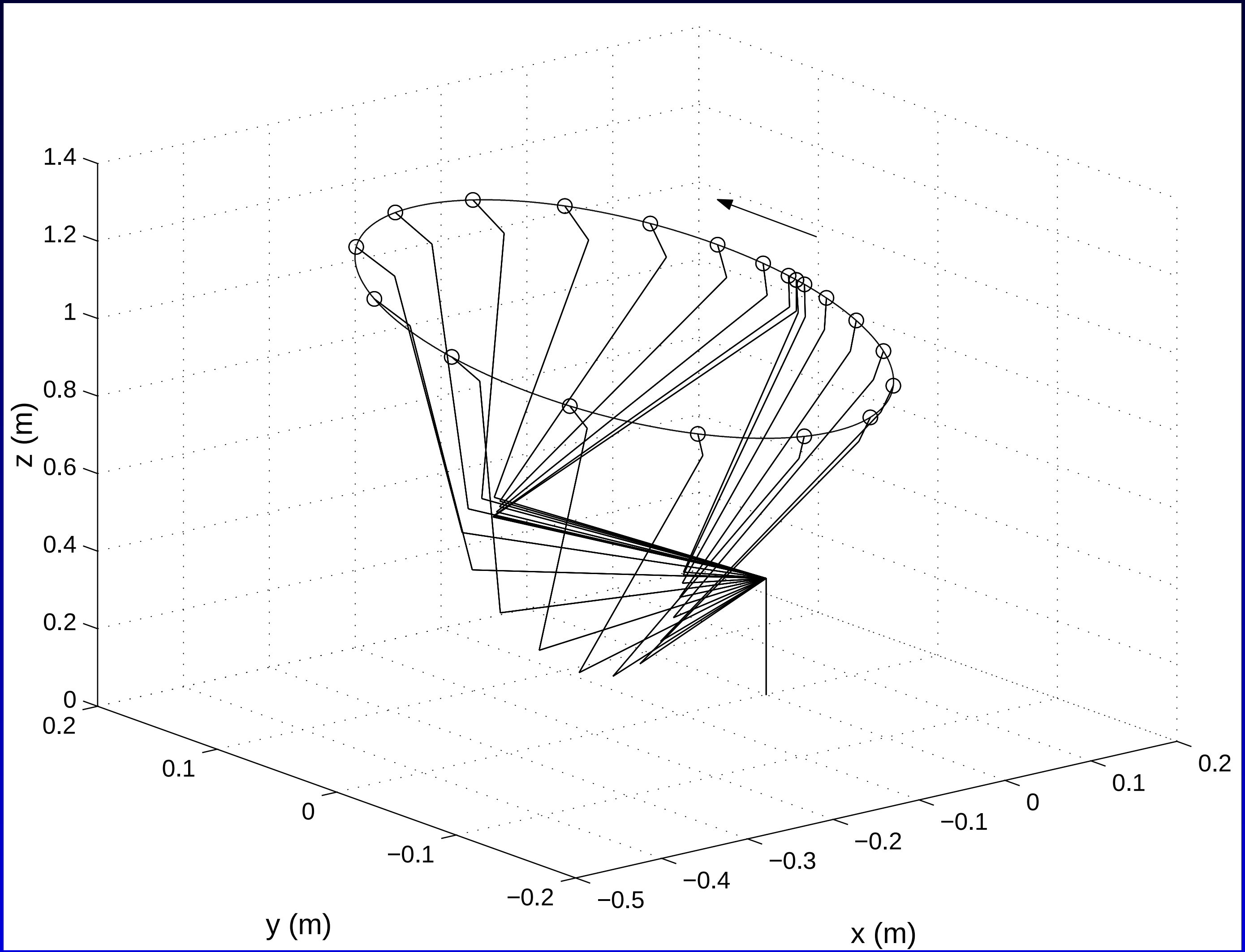
Primal-Dual Network Architecture



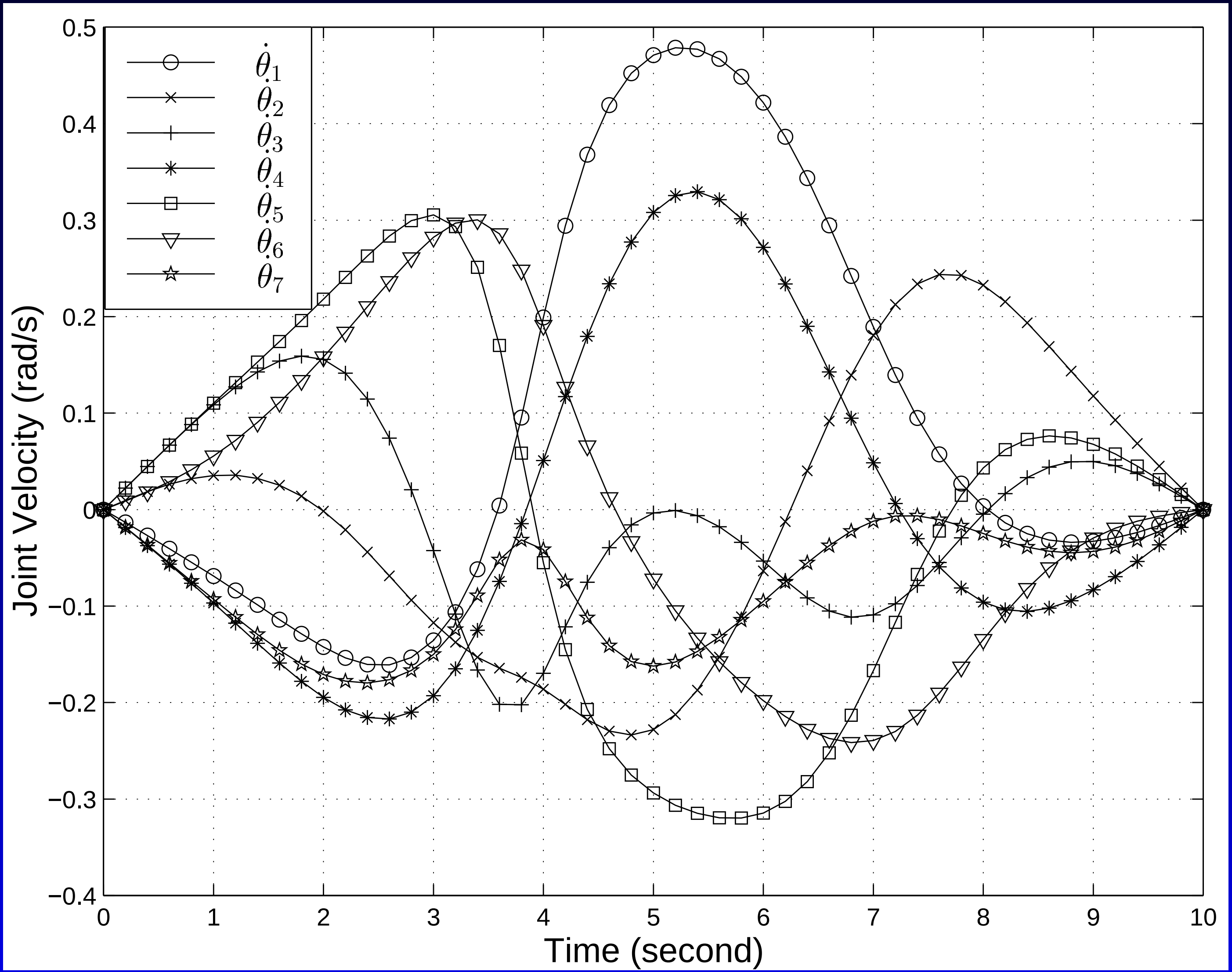
Desired Position of PA10 End-Effector



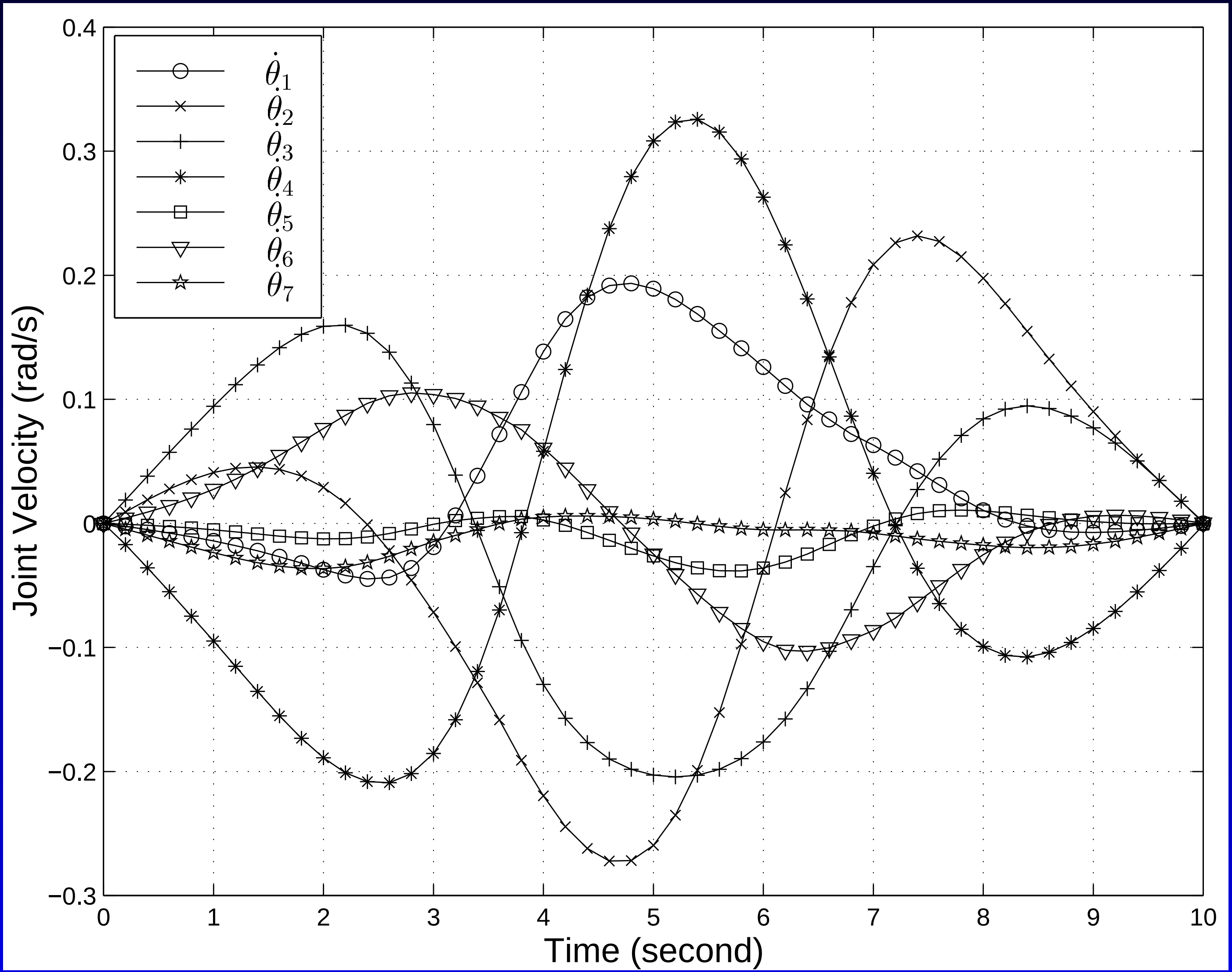
PA10 Circular Motion



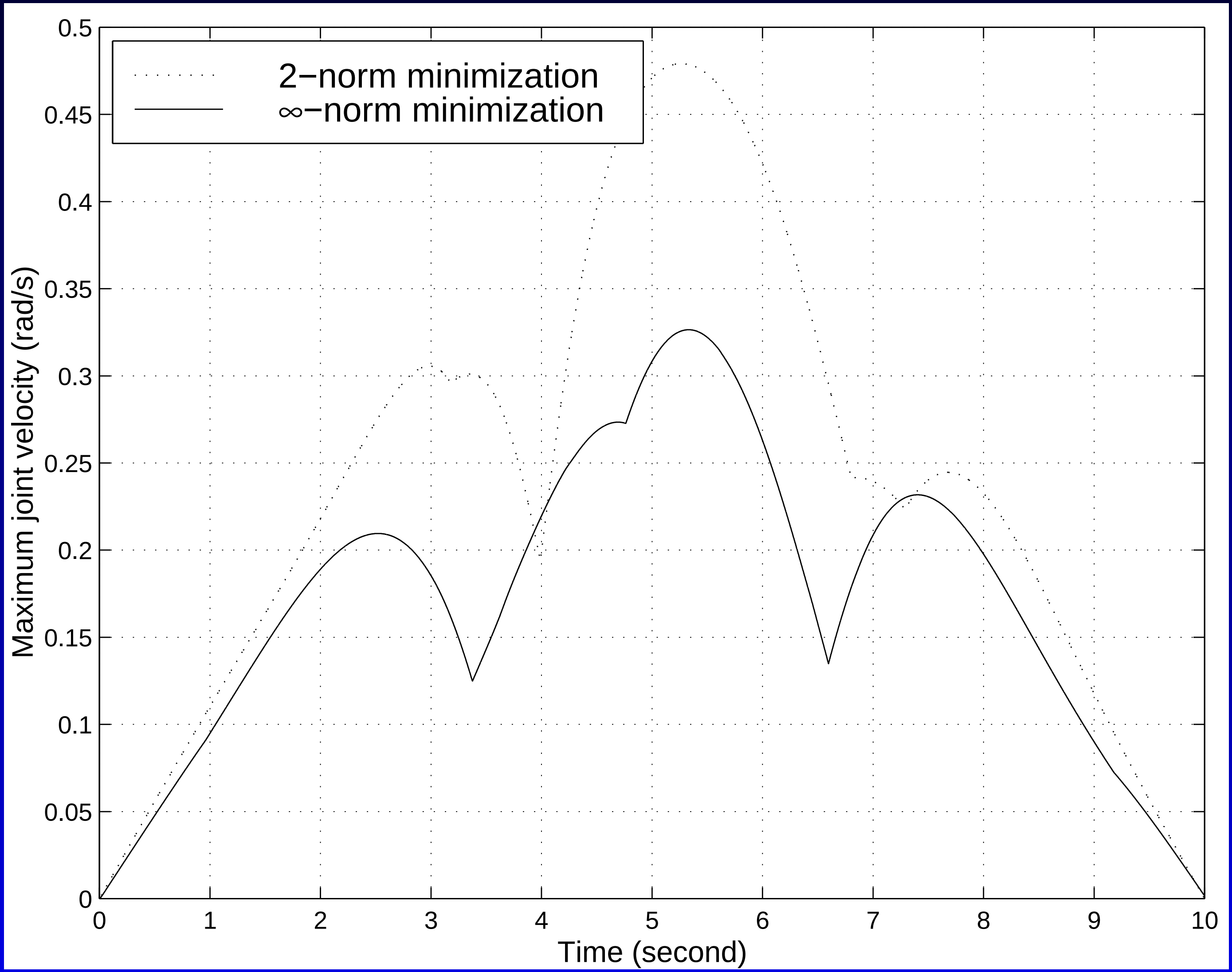
Joint Velocities from the Lagrangian Network



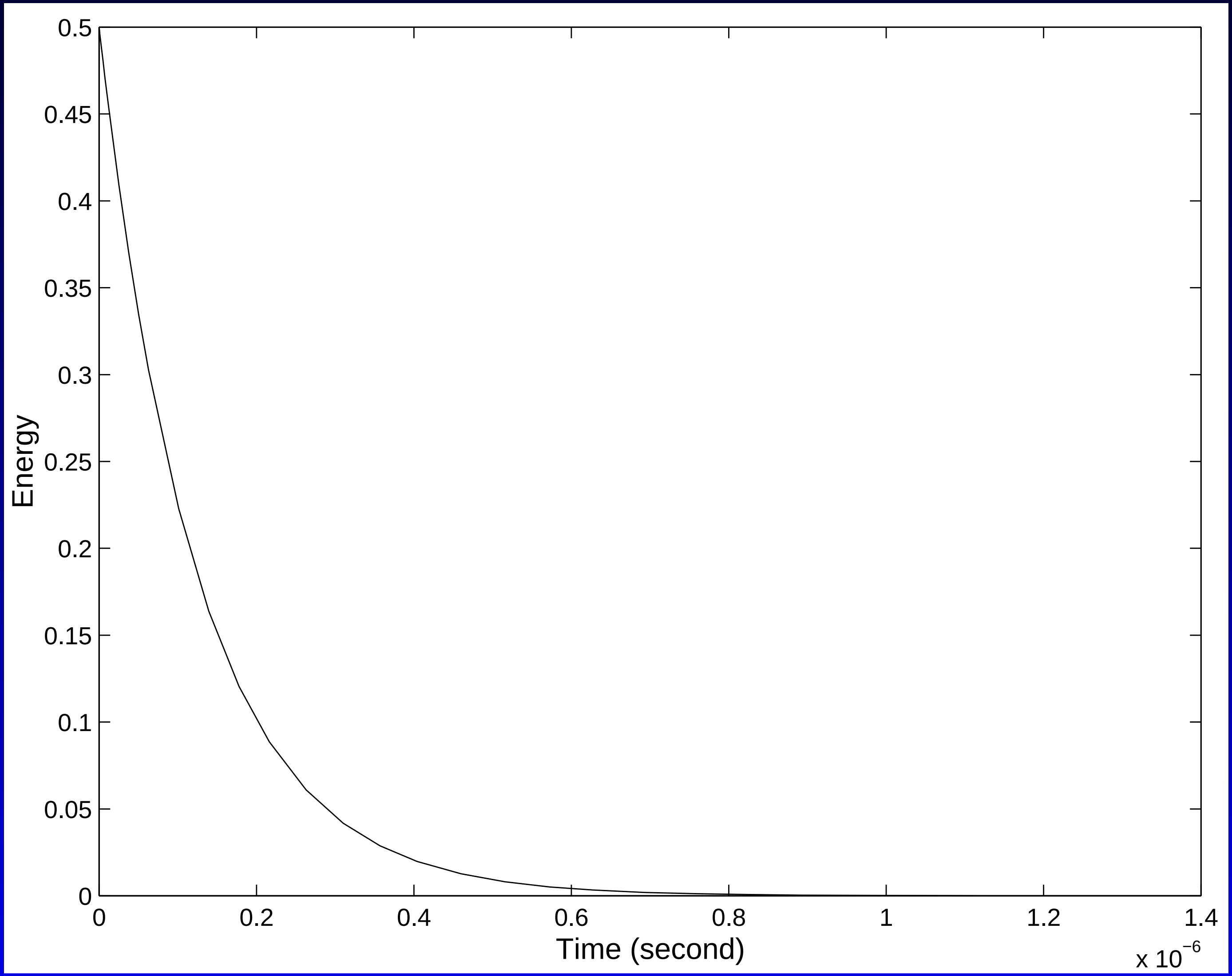
Joint Velocities from the Primal-Dual Network



Infinity Norm of Joint Velocities



Transients of Energy Function



Bi-criteria Kinematic Control

The bi-criteria redundancy resolution scheme subject to joint limits:

$$
\begin{aligned}
\text{minimize} \quad & \frac{1}{2}\left\{\alpha\|\dot{\theta}\|_2^2 + (1-\alpha)\|\dot{\theta}\|_\infty^2\right\} \\
\text{subject to} \quad & J(\theta)\dot{\theta} = \dot{x}_d \\
& \eta^- \leqslant \dot{\theta} \leqslant \eta^+
\end{aligned}
$$

where $\eta^\pm$ denote upper and lower limits of joint velocities respectively.

Problem Reformulation

With e_j denoting the jth column of identity matrix I,

$$\|\dot{\theta}\|_\infty = \max\{|\dot{\theta}_1|, |\dot{\theta}_2|, \cdots, |\dot{\theta}_n|\} = \max_{1 \leqslant j \leqslant n} |e_j^T \dot{\theta}|.$$

With $s(t) := \|\dot{\theta}(t)\|_\infty$, the term $(1-\alpha)\|\dot{\theta}(t)\|_\infty^2/2$ equals

$$\begin{cases} \text{min. } \frac{1-\alpha}{2} s^2(t) \\ \text{s.t. } |e_j^T \dot{\theta}| \leqslant s(t) \end{cases}$$

$$\Longrightarrow \quad \begin{cases} \text{min. } \frac{1-\alpha}{2} s^2(t) \\ \text{s.t. } \begin{bmatrix} I & -1 \\ -I & -1 \end{bmatrix} \begin{bmatrix} \dot{\theta}(t) \\ s(t) \end{bmatrix} \leqslant \begin{bmatrix} 0 \\ 0 \end{bmatrix} \end{cases}$$

Problem Formulation

With $y := [\dot{\theta}^T, s]^T$, the bi-criteria problem becomes:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2} y^T Q y \\ \text{subject to} & Ay \leqslant b \\ & Cy = d \\ & y^- \leqslant y \leqslant y^+ \end{array}$$

where $Q := \begin{bmatrix} \alpha I & \\ & (1-\alpha) \end{bmatrix}, \ A := \begin{bmatrix} I & -1 \\ -I & -1 \end{bmatrix}, \ b := 0 \in R^{2n},$

$C := \begin{bmatrix} J(\theta) & \mathbf{0} \end{bmatrix}, \ d := \dot{x}_d(t), \ y^- := \begin{bmatrix} \eta^- \\ 0 \end{bmatrix}, \ y^+ := \begin{bmatrix} \eta^+ \\ \max\{\eta^{\pm}\} \end{bmatrix}$

Problem Formulation

Treat equality and inequality constraints as bound constraints:

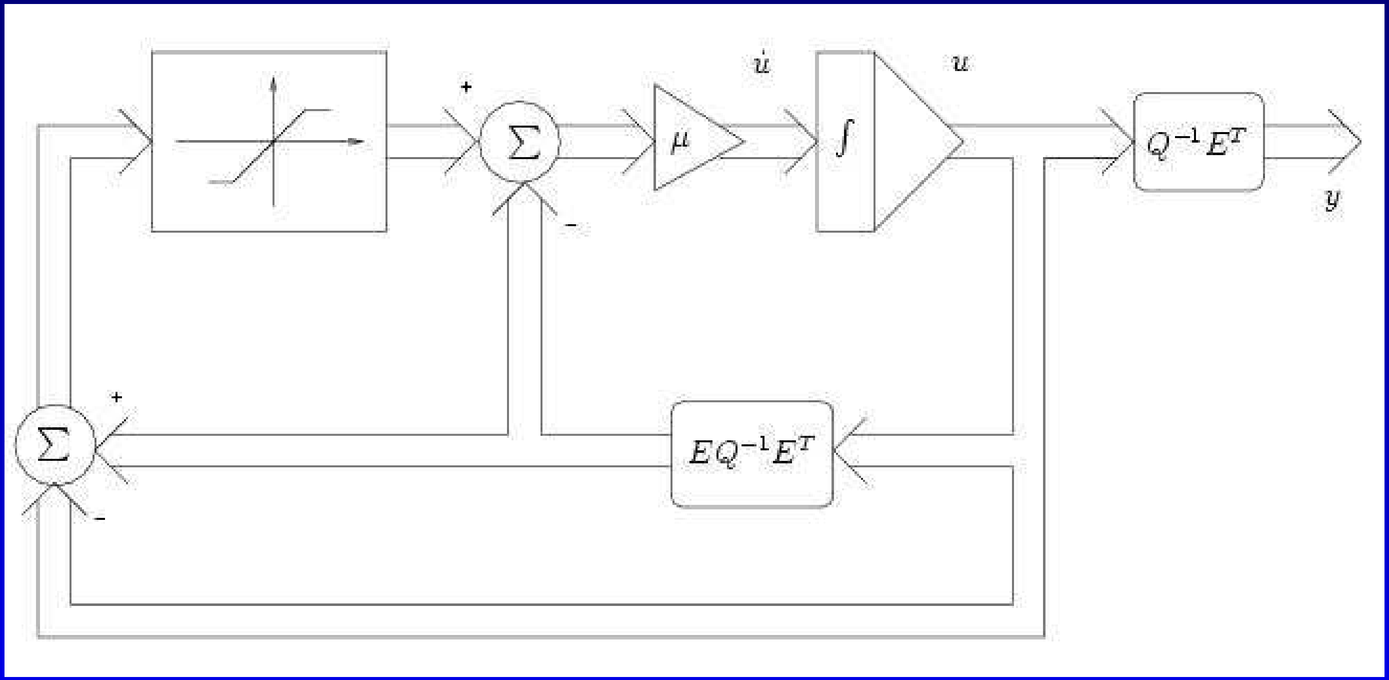
$$\xi^- = \begin{bmatrix} b^- \\ d \\ y^- \end{bmatrix}, \xi^+ = \begin{bmatrix} b \\ d \\ y^+ \end{bmatrix}, E = \begin{bmatrix} A \\ C \\ I \end{bmatrix}$$

with b^- sufficiently negative to represent $-\infty$. Then the bicriteria kinematic control problem can be rewritten as

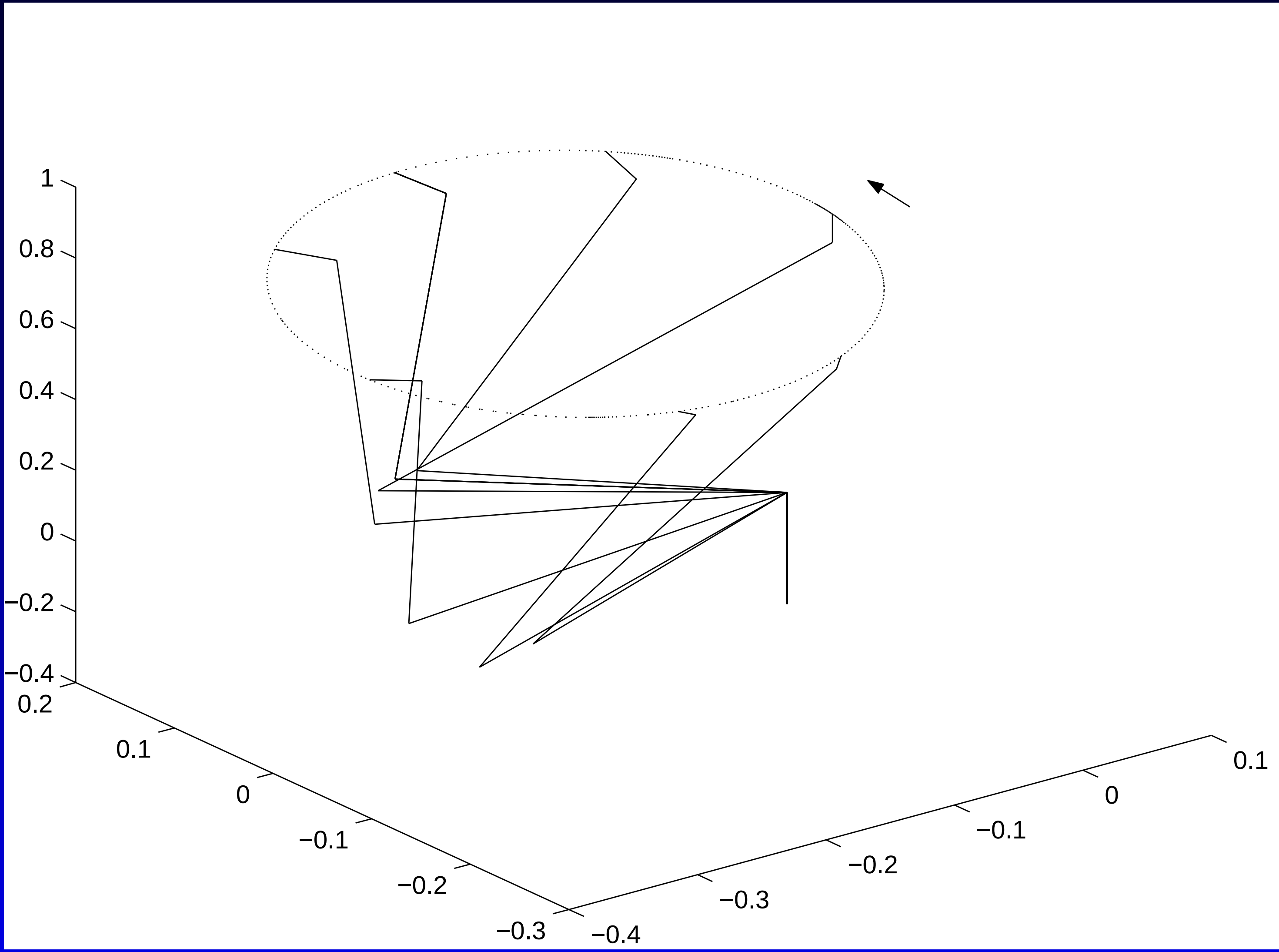
$$\begin{array}{ll} \text{minimize} & \frac{1}{2} y^T Q y \\ \text{subject to} & \xi^- \leqslant Ey \leqslant \xi^+. \end{array}$$

Dual Network Dynamics

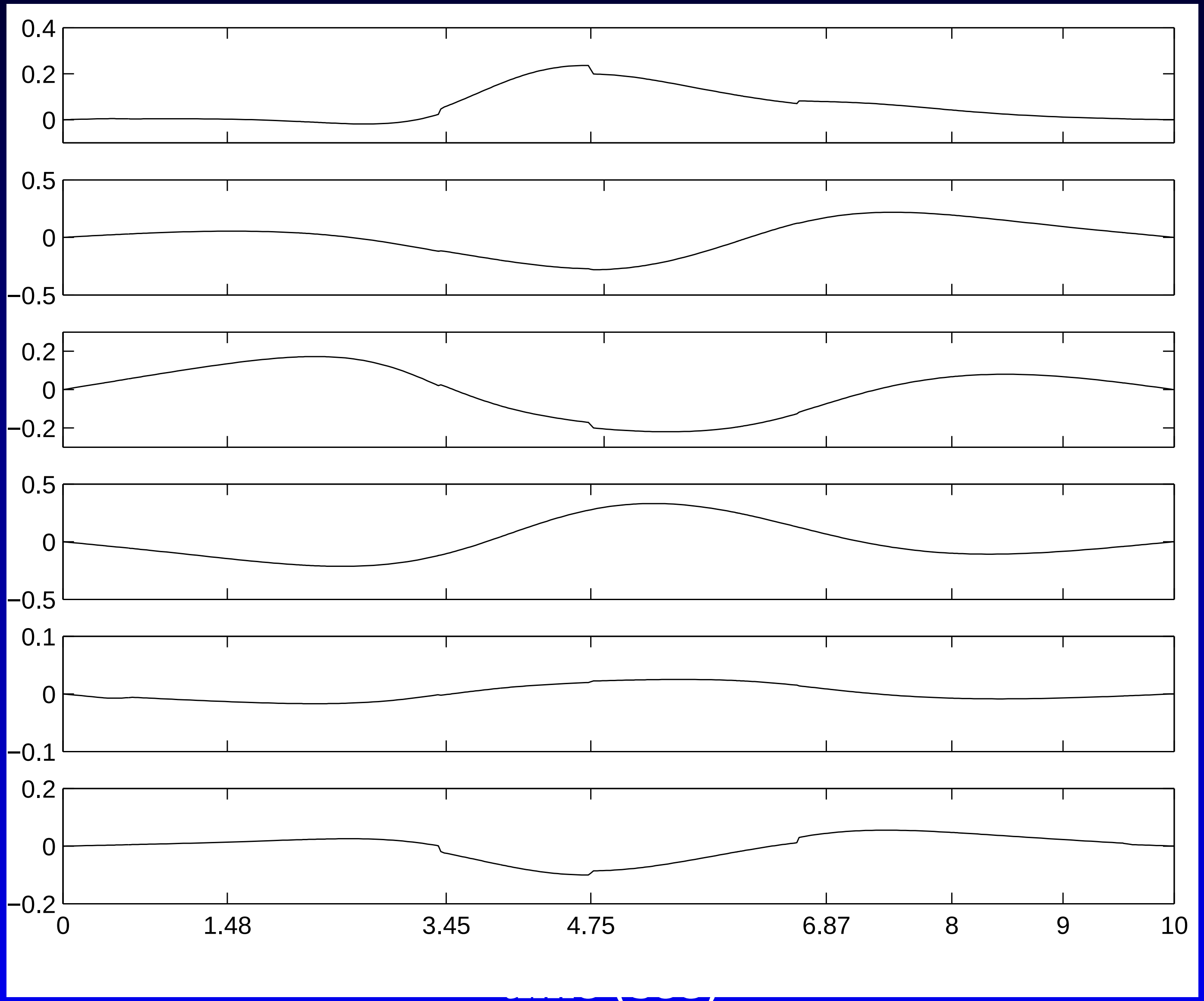
$$\epsilon\frac{du(t)}{dt} = -EQ^{-1}E^Tu(t) + g((EQ^{-1}E^T - I)u(t)),$$
$$y(t) = Q^{-1}E^Tu(t).$$



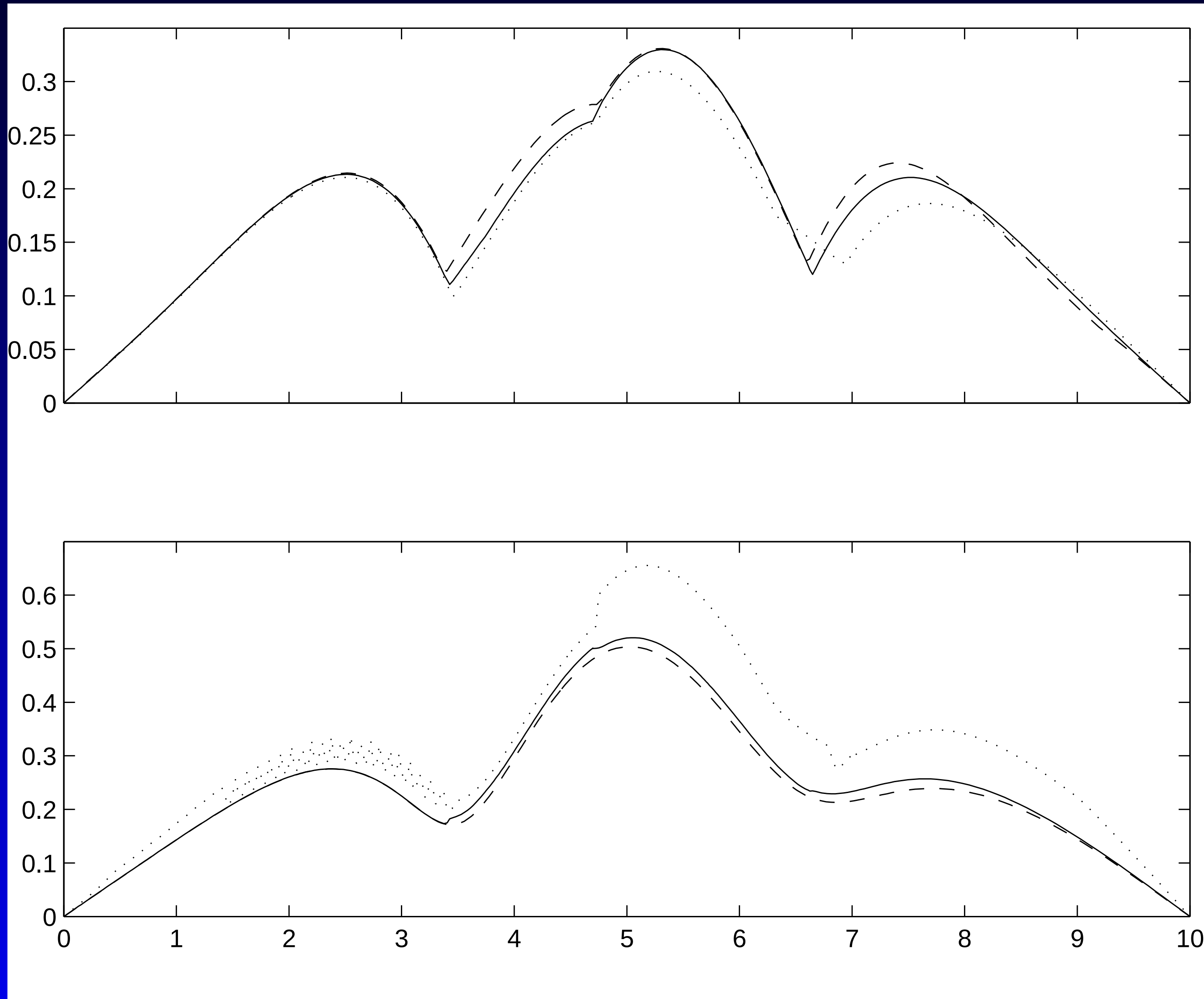
Simulation Results



Joint Velocity



Norm Comparison



Grasping Force Optimization

Consider a multifingered robot hand grasping a single object in a 3-dimensional workspace with m point contacts between the grasped object and the fingers.

The problem of the grasp force optimization is to find a set of contact forces such that the object is held at the desired position and external forces are compensated.

A grasping force x_i is applied by each finger to hold the object without slippage and to balance any external forces.

Grasping Force Optimization

To ensure non-slipping at a contact point, the grasping force x_i should satisfy $x_{i1}^2 + x_{i2}^2 \leq \mu_i x_{i3}^2$, where $\mu_i > 0$ is the friction coefficient at finger i, and x_{i1}, x_{i2}, and x_{i3} are components of contact force x_i in the contact coordinate frame.

Besides the form-closure constraints, to balance any external wrench f_{ext} to maintain a stable grasp, each finger must apply a grasping force $x_i = [x_{i1}, x_{i2}, x_{i3}]$ to the object such that $Gx = -f_{ext}$, where $G \in R^{6\times 3m}$ is the grasp transformation matrix and $x = [x_1, ..., x_m]^T \in R^{3m}$ is the grasping force vector.

Grasping Force Optimization

The optimal grasping force optimization can be formulated as the following quadtatic minimization problem with linear and quadratic constraints:

$$\begin{array}{ll} \text{minimize} & f(x) = \frac{1}{2} x^T Q x \\ \text{subject to} & c_i(x) \leq 0, \; i = 1, ..., m; \\ & Gx = -f_{ext} \end{array}$$

where $q \in R^{3m}$, Q is a $3m \times 3m$ positive definite matrix, and $c_i(x) = \sqrt{x_{i1}^2 + x_{i2}^2} - \mu_i x_{i3}$.

Neurodynamic Optimization of Gasping Force

Based on the problem formulation, we develop the three-layer recurrent neural network for gasping force optimization

$$\epsilon\frac{d}{dt}\left(\begin{array}{c} x \\ y \\ z \end{array}\right) = \left(\begin{array}{c} -Qx - \nabla c(x)y + G^T z \\ -y + h(c(x)+y) \\ -Gx - f_{ext} \end{array}\right),$$

where $x \in R^{3m}$, $y \in R^m$, $z \in R^6$, and $\epsilon > 0$ is a scaling parameter.
The neural network is globally convergent to the KKT point (x^*, y^*, z^*), where x^* is the optimal gasping force.

Neurodynamic Optimization of Gasping Force

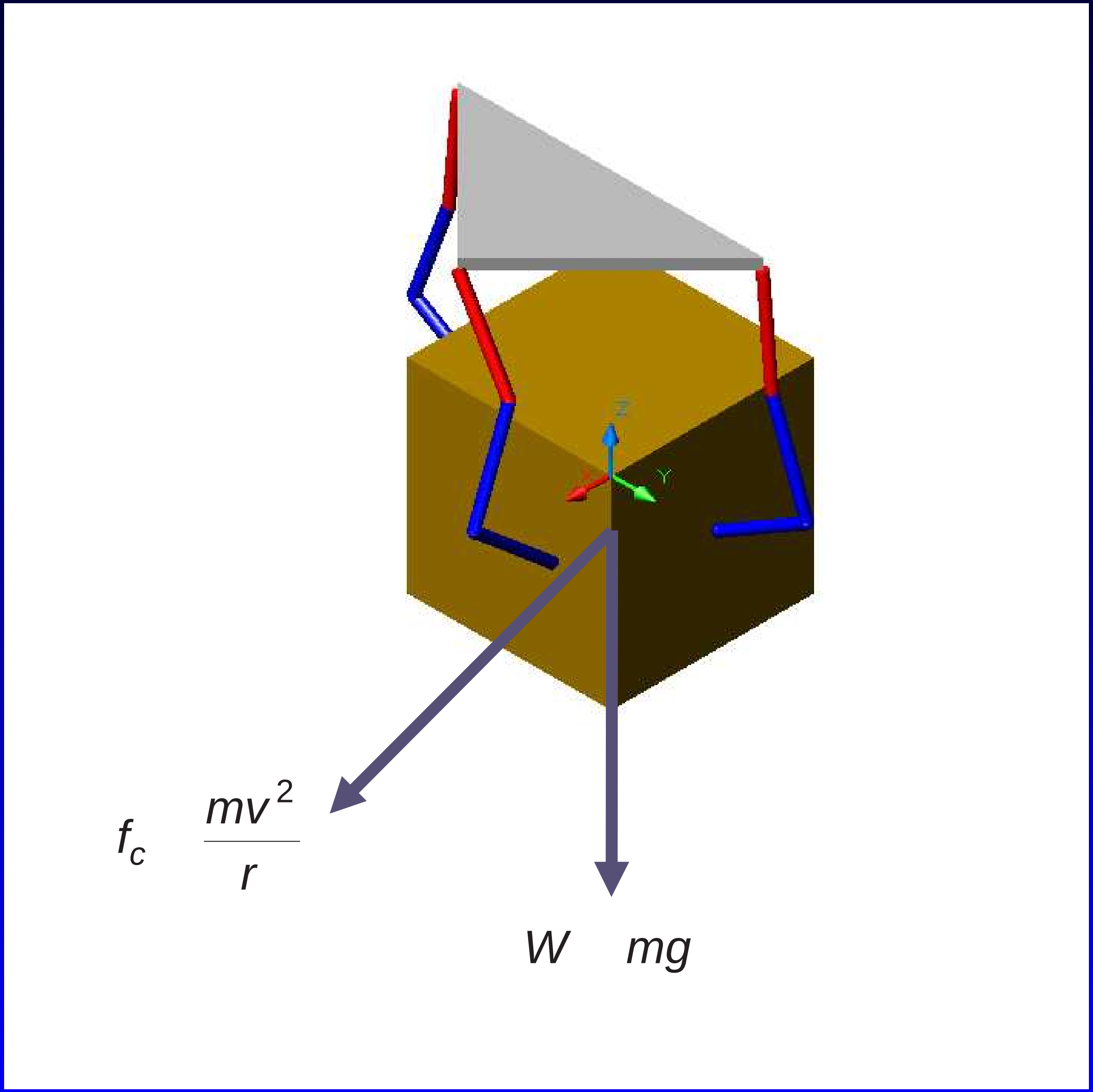
Consider a minimum norm force $f(x) = \frac{1}{2}\|x\|^2$.

A polyhedral object with $M = 0.1$kg is grasped by a three-fingered robotic hand.

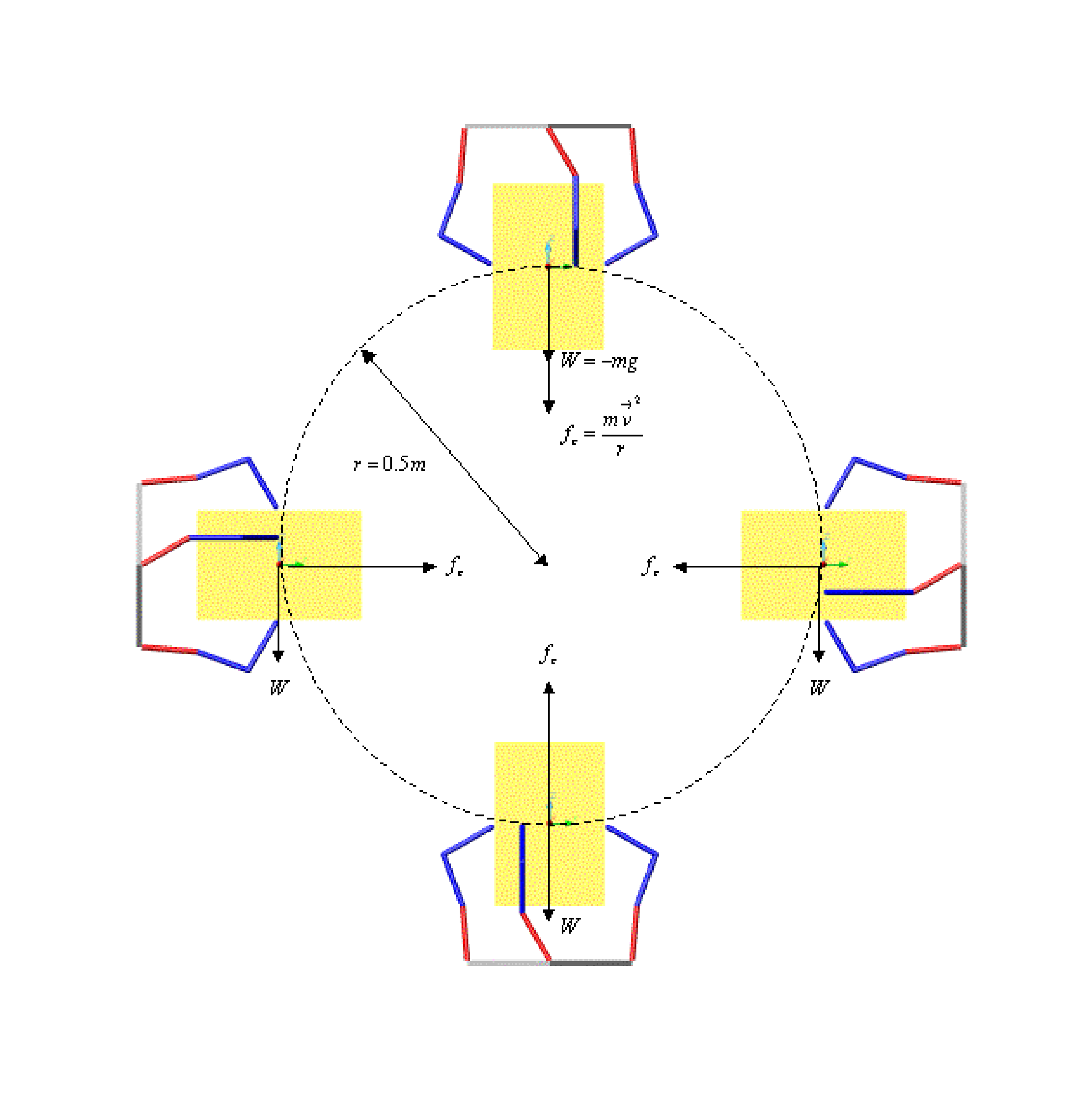
Let the robotic hand move along a circular trajectory of radius $r = 0.5$m with a constant velocity v =0.2m/s.

The time-varying external wrench applied to the center of mass of the object is $f_{ext} = [0, f_c \sin(\theta(t)), -Mg + f_c \cos(\theta(t)), 0, 0, 0]^T$, where $g = 9.8(m/s^2)$, $\theta \in [0, 2\pi]$, and $f_c = Mv^2/r$.

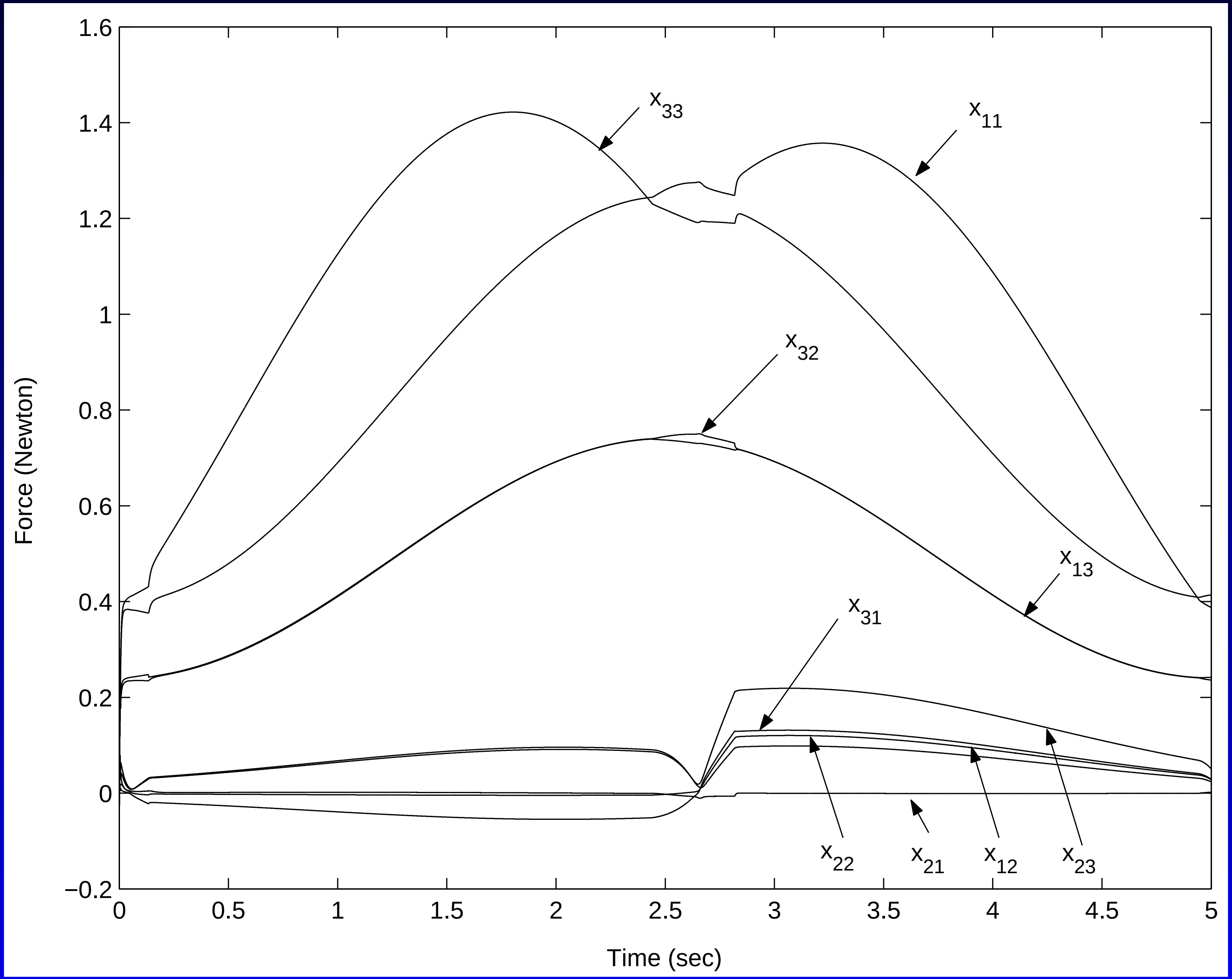
Simulation Results



Simulation Results



Simulation Results



Simulation Results

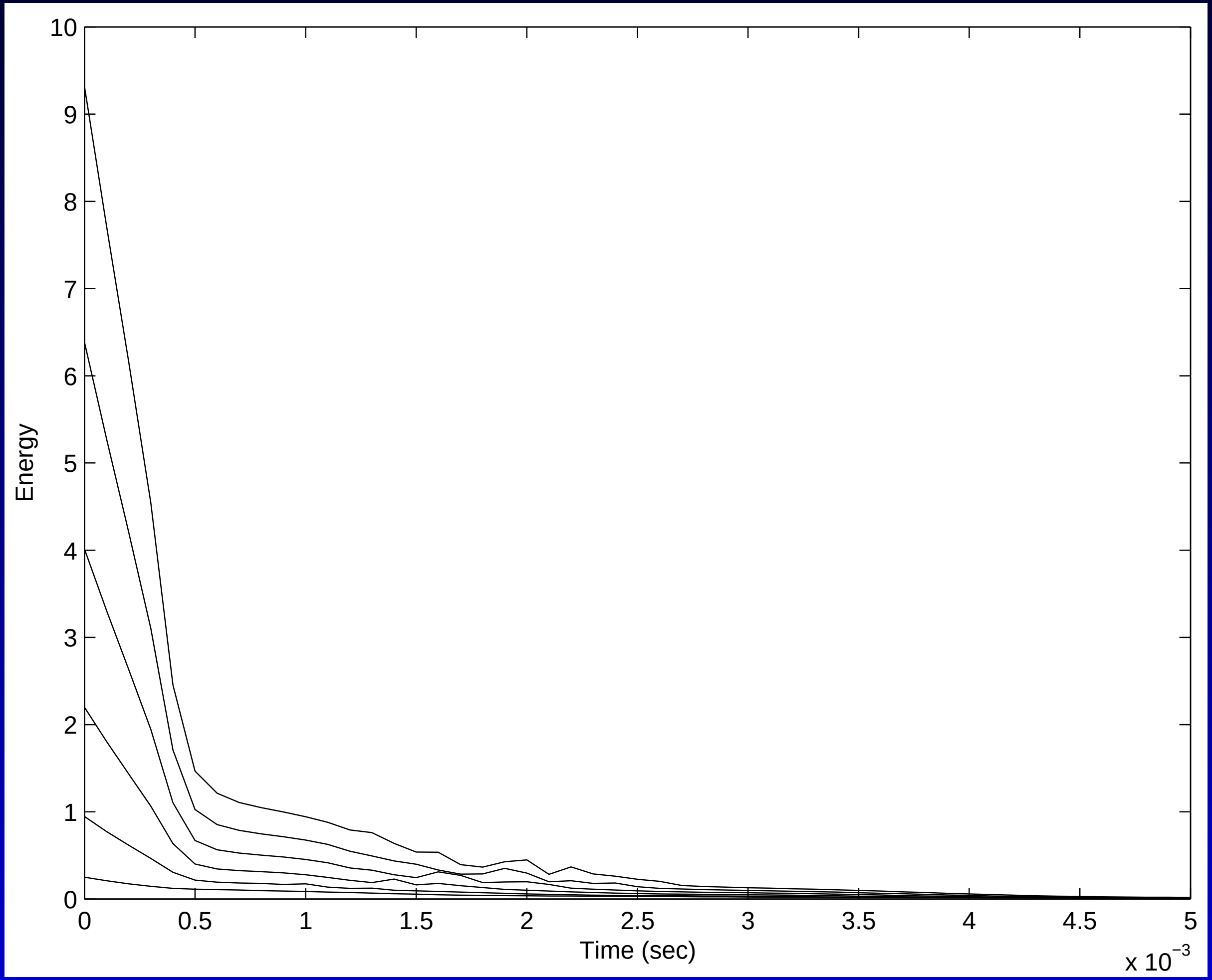


Figure 7: Convergence of the energy function with $\epsilon = 0.0001$

Simulation Results

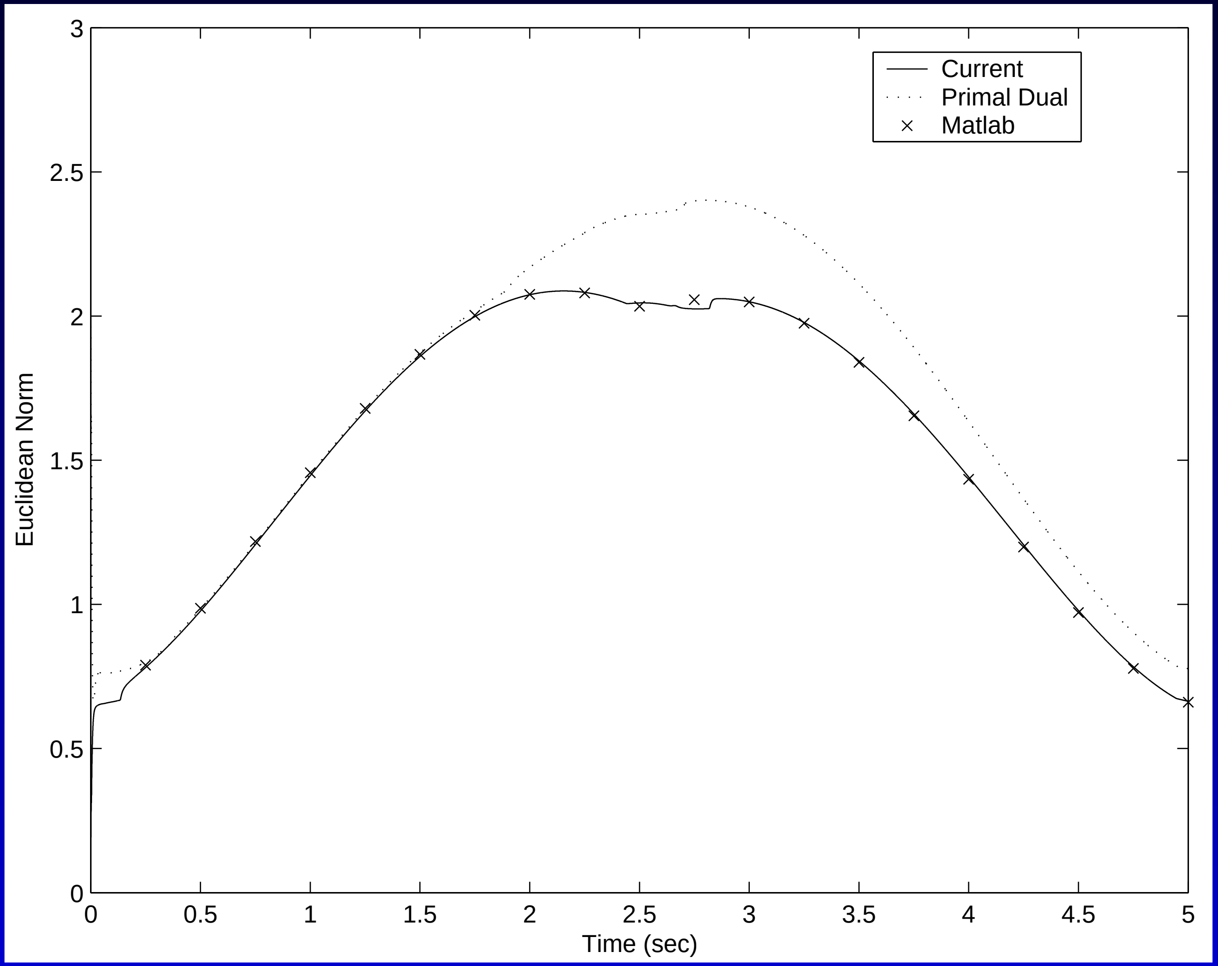


Figure 8: Comparison of Euclidean norm of optimal forces using three different methods

Concluding Remarks

Neurodynamic optimization has been demonstrated to be a powerful alternative approach to many optimization problems.

For convex optimization, recurrent neural networks are available with global convergence to the optimal solution.

Neurodynamic optimization approaches provide parallel distributed computational models more suitable for real-time applications.

Future Works

The existing neurodynamic optimization model can still be improved to reduce their model complexity or increase their convergence rate.

The available neurodynamic optimization model can be applied to more areas such as control, robotics, and signal processing.

Neurodynamic approaches to global optimization and discrete optimization are much more interesting and challenging.

It is more needed to develop neurodynamic models for nonconvex optimization and combinatorial optimization.