2018 NSF – CBMS
Computational Methods in Optimal Control
Jackson State University, Jackson, MS

Mini-Tutorial: Computational Aspects of Numerical Optimization

Vladislav Bukshtynov
vbukshtynov@fit.edu

Department of Mathematical Sciences
Florida Institute of Technology
Melbourne, FL

July 27, 2018

Some Objectives

1. Solving Optimal Control Problem Computationally
2. "Optimize–then–Discretize" vs. "Discretize–then–Optimize"
3. Application to ODE-based Optimal Control Problem

Main Notations for Generalized Optimal Control (Optimization) Problem

$$\begin{aligned} &\underset{\mathbf{u}}{\text{minimize/maximize}} && \mathcal{J}(\mathbf{x};\mathbf{u}) \\ &\text{subject to} && \mathbf{g}(\mathbf{x};\mathbf{u}) = \mathbf{0} \\ & && \mathbf{h}(\mathbf{x};\mathbf{u}) \leq \mathbf{0} \end{aligned}$$

- state variables $\mathbf{x} \in \mathbb{R}^m$
- optimization (control, decision, design) variables $\mathbf{u} \in \mathbb{R}^n$
- objective (cost) function(al) $\mathcal{J}(\mathbf{x};\mathbf{u}) : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^k$
- constraints, governing equations or equations of states (EOS)
  - (systems of) linear/nonlinear equalities/inequalities
  - (systems of) ODE(s)/PDE(s)
  - other requirements, e.g. functional space for $\mathbf{u}$
- $\mathcal{J}, \mathbf{g}, \mathbf{h}$ are vector functions in general
- feasible region (control set/space) $\mathcal{U}$ defined by constraints
- solution (local/global) $\mathbf{u}^* = \underset{\mathbf{u}\in\mathcal{U}}{\text{argmin}}\ \mathcal{J}(\mathbf{x};\mathbf{u})$
- infeasible solution $\mathbf{u} \notin \mathcal{U}$

Iterative Optimization Algorithm

$$\min_{\mathbf{u}\in\mathbb{R}^n} \quad \mathcal{J}(\mathbf{x};\mathbf{u})$$

$$\text{s. t.} \quad \mathbf{u}\in\mathcal{U}$$

Generate a sequence of suboptimal solutions (iterates)

$$\mathbf{u}^0,\ \mathbf{u}^1,\ \mathbf{u}^2,\ \ldots,\ \mathbf{u}^k,\ \mathbf{u}^{k+1},\ \ldots$$

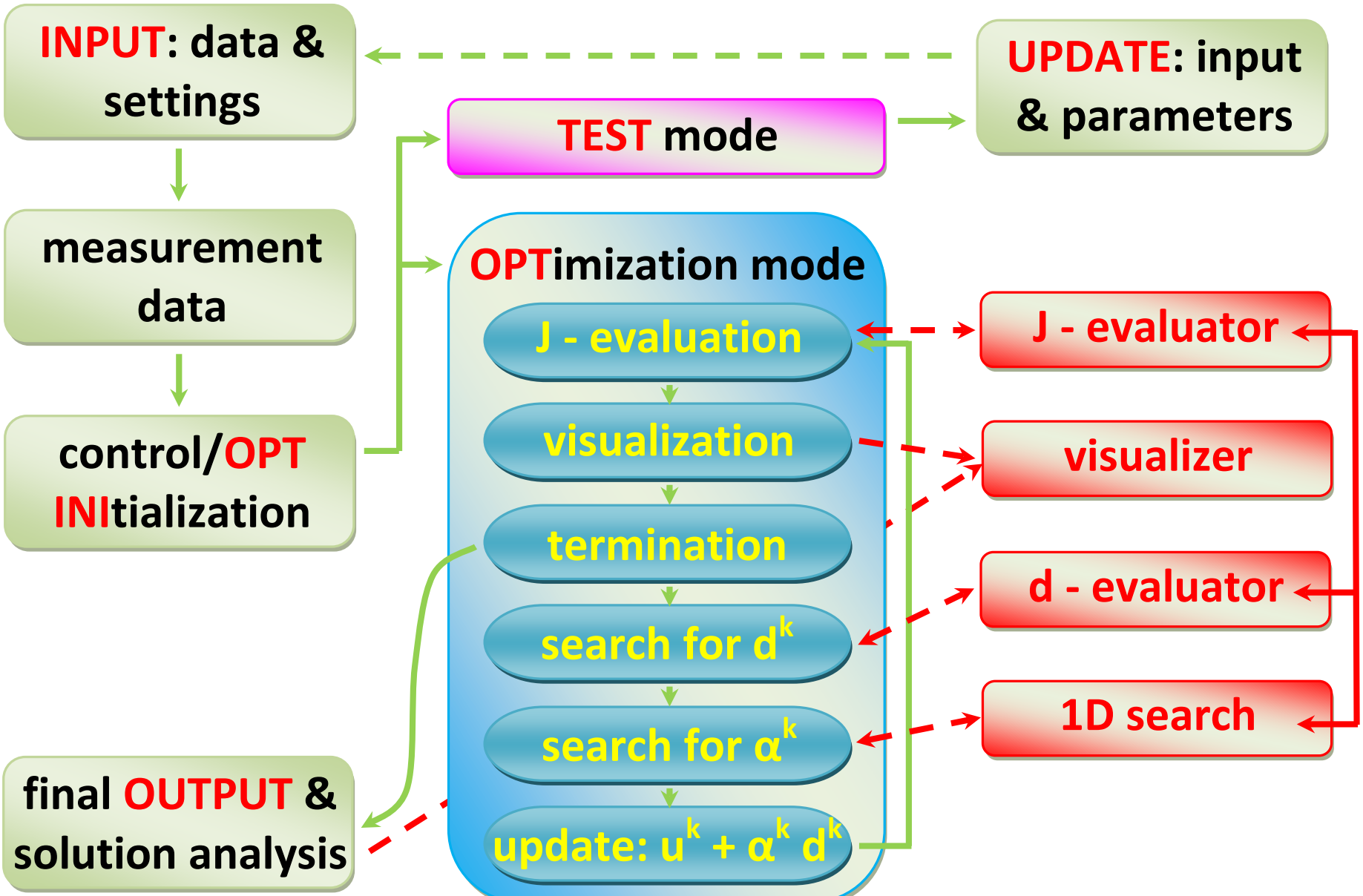
1. Choose initial guess $\mathbf{u}^0$ (and algorithm settings)
2. For $k = 1, 2, \ldots$ check (computational) optimality of $\mathbf{u}^k$, e.g.
   - $\nabla\mathcal{J}(\mathbf{x}^k;\mathbf{u}^k) \cong \mathbf{0}$ (local optimum condition)
   - $\mathbf{u}^k$ reduces $\mathcal{J}(\mathbf{x};\mathbf{u})$ up to a necessary level
   - other termination conditions (backup slide **B1**)
3. If OK (optimal) $\rightarrow$ **STOP**
4. Find a search direction (update) $\mathbf{d}^k$ to improve a solution $\mathbf{u}^k$

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \alpha^k\mathbf{d}^k$$

5. Go to 2

- search direction $\mathbf{d}^k$ improves the solution in some sense, e.g. $\mathbf{d}^k = -\nabla\mathcal{J}(\mathbf{x}^k;\mathbf{u}^k)$
- fundamental strategies (gradient-based): Line Search and Trust Region
- step size α^k is determined in assumption $\mathcal{J}(\mathbf{u}^{k+1}) < \mathcal{J}(\mathbf{u}^k)$ (backup slide **B2**)

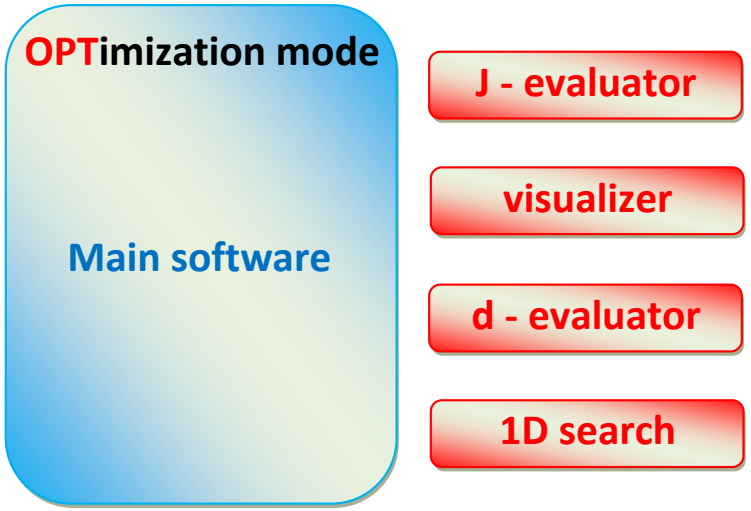
Computational Elements of the Generalized Optimization Framework



Choice of Proper Software

Main (core) software:

- ideally is self-contained: specialized software to solve a particular problem
- difficult to apply to specific needs or modify
- best idea: used as a communication and data processing framework



$\mathcal{J}$-evaluator:

- to evaluate objective function(s) $\mathcal{J}(\mathbf{x};\mathbf{u})$
- may require to solve (systems of) (non)linear equation(s), ODE(s), PDE(s)

d-evaluator:

- to find search direction(s) $\mathbf{d}$
- may require to solve (systems of) (non)linear equation(s), ODE(s), PDE(s)
- may require to communicate effectively with $\mathcal{J}$-evaluator

Choice of Proper Software (cont'd)

1D search:

- to find optimal stepsize α
- depends on the nature of the problem (linearity, convexity, constraints, etc.)
- may require to communicate effectively with $\mathcal{J}$-evaluator

Visualizer:

- to perform analysis of input data (a priori) and obtained solutions (a posteriori)
- to control the progress of optimization algorithm
- ideally should not slow down or interrupt main optimization process via fast and easy access to stored intermediate data

Solver for (systems of) (non)linear equation(s), ODE(s), PDE(s):

- very problem dependent
- trade-off: fast vs. accurate

Examples of core software platforms:

- `MATLAB` + access to parallel computing, math, statistics and optimization toolboxes
- `C/C++`-based scientific environments with added libraries for linear algebra, solving PDEs, optimization, etc., e.g. `FreeFem++`
- separate solvers available in common formats: `MATLAB`, `C/C++`, `Fortran`, etc.

Visualization and Analysis of Obtained Solution

Data to be visualized (depending on problem):

1. Optimization progress: objective function
   - measurement data
   - separate parts of objective (how closely data is fitted)
   - entire objective vs. iteration number k (to check monotonicity)
2. Optimization progress: optimization/control variables
   - optimal or "true" solution (used to generate measurements, then forgotten)
   - current solution
   - some measures how close they are (monotonicity may not be expected!)
3. Other optimization attributes:
   - gradients
   - state variables (if different from control variables)
   - dynamic parameters (search direction stepsize, weighting coefficients, etc.)
   - checking other optimization techniques (regularization, preconditioning, etc.)

Before your big project starts, think how to:

- save intermediate data instead of graphical images
- keep data in easily convertible formats, e.g. `dat` or `txt` files with plain numbers
- convert your data into high resolution images or send to external software

Testing (General Debugging) and Dealing with Problems

$\mathcal{J}$-evaluator

- Test-case #1: run for $\mathbf{u} = \mathbf{u}^*$ to check $\mathcal{J} = \mathcal{J}^*$
- Test-case #2: if $\mathcal{J} \neq \mathcal{J}^*$, check you are able to control $|\mathcal{J} - \mathcal{J}^*| \to 0$ by tuning solver parameters (refining mesh, applying higher-order schemes, etc.)
- Test-case #3: run trustful and commonly used benchmark models and compare your outcomes with published results

d-evaluator (method-dependent)

- Test-case for gradient-based method: run "kappa-test" to check your gradient is accurate and consistent with its FD approximation (see slides 34-36 for details)

main OPT-part

- Test every component separately
- Test communication within the entire framework (variables, dimensionality, names, solution files, etc.)
- Tuning Test: for the same problem change one parameter/technique at a time
- Robustness Test: for fixed set of parameters/techniques run algorithm for the same problem varying initial data; then explore the limits and repeat tuning
- Applicability Test: apply algorithm to problems at different scales (small, moderate, large)

1. Solving Optimal Control Problem Computationally

2. “Optimize–then–Discretize” vs. “Discretize–then–Optimize”

3. Application to ODE-based Optimal Control Problem

Practice Example: Fitting Data by ODE-constrained Optimization

Problem: $f(t)$-parameter identification in BVP-2 by fitting (continuous) data, continuous formulation

$$\min_{f(t)} \mathcal{J}(y, t; f) = \frac{1}{2} \int_0^T (y - \tilde{y})^2 \, dt$$

$$\text{s. t. } a_1 y'' + a_2 y' + a_3 y = f(t), \quad y(0) = y_0, \quad y(T) = y_T$$

where data $(\tilde{x}, \tilde{y})$ are available continuously over interval $[0, T]$, and constants a_1, a_2, a_3 are given.

Approach:

- solve Problem using adjoint-based gradient method (Lagrange multiplier approach)
- derive gradients by "optimize–then–discretize" approach – optimality conditions and optimization algorithm are based on continuous form of the problem

Alternative approach:

- derive gradients by "discretize–then–optimize" approach
- solve Problem using regular gradient-based method by approximating objective gradient using finite difference (FD) schemes – expensive in case of fine discretization in t

Practice Example: "Optimize–then–Discretize" – Deriving Gradient

Objective function

$$\mathcal{J}(y, t; f) = \frac{1}{2} \int_0^T (y - \tilde{y})^2 \, dt$$

Forward problem (governing equation, equation of state) by BVP-2

$$\begin{aligned} & a_1 y'' + a_2 y' + a_3 y = f(t), \\ & y(0) = y_0, \quad y(T) = y_T \end{aligned} \qquad \text{(FP)}$$

Lagrangian (augmented objective function)

$$\mathcal{L}(y, t; f, \psi) = \mathcal{J}(y, t; f) + \left\langle a_1 y'' + a_2 y' + a_3 y - f(t),\ \psi \right\rangle_\chi$$

- state variable $y(t)$
- control variable $f(t)$
- adjoint variable (Lagrange multiplier) $\psi(t)$
- inner product defined in χ-space $\langle \cdot,\ \cdot \rangle_\chi$, e.g., if $\chi = L_2$

$$\langle f_1(t),\ f_2(t) \rangle_{\chi = L_2} = \int_0^T f_1 \cdot f_2 \, dt$$

Thus, we have

$$\mathcal{L}(y, t; f, \psi) = \frac{1}{2} \int_0^T (y - \tilde{y})^2 \, dt + \int_0^T \left[a_1 y'' + a_2 y' + a_3 y - f(t) \right] \psi \, dt$$

Practice Example: "Optimize–then–Discretize" – Deriving Gradient (cont'd)

Riesz Representation Theorem: 1-order variation for Lagrangian $\mathcal{L}(y, t; f, \psi)$ can be expressed (only linear terms)

$$\delta\mathcal{L}(y, t; f, \psi) = \langle \nabla_f \mathcal{L},\ \delta f \rangle_\chi + \langle \nabla_y \mathcal{L},\ \delta y \rangle_\chi + \langle \nabla_\psi \mathcal{L},\ \delta\psi \rangle_\chi + \dots$$

- smaller (than linear) terms "..." (neglect)
- variations (perturbations) for control, state and adjoint variables $\delta f, \delta y, \delta\psi$ (arbitrary chosen functions)
- gradients w. r. t. control, state and adjoint variables $\nabla_f \mathcal{L}, \nabla_y \mathcal{L}, \nabla_\psi \mathcal{L}$
- Fréchet differential $\langle \nabla_f \mathcal{L},\ \delta f \rangle_\chi$ will give expression for (Fréchet) gradient (will be reduced, hopefully to 0, during optimization)
- setting $\langle \nabla_y \mathcal{L},\ \delta y \rangle_\chi$ to 0 will give adjoint equation(s)
- setting $\langle \nabla_\psi \mathcal{L},\ \delta\psi \rangle_\chi$ to 0 is natural as it represents our forward problem (FP)

Practice Example: "Optimize–then–Discretize" – Deriving Gradient (cont'd)

Goal: Derive 1-order variation for Lagrangian

$$\mathcal{L}(y,t;f,\psi) = \frac{1}{2}\int_0^T (y-\tilde{y})^2\,dt + \int_0^T \left[a_1 y'' + a_2 y' + a_3 y - f(t)\right]\psi\,dt$$

consistent with Riesz Representation Theorem and set it to 0 (KKT conditions)

$$\delta\mathcal{L}(y,t;f,\psi) = \int_0^T (y-\tilde{y})\,\delta y\,dt + \int_0^T \left[a_1(\delta y)'' + a_2(\delta y)' + a_3\delta y - \delta f\right]\psi\,dt + \int_0^T \left[a_1 y'' + a_2 y' + a_3 y - f(t)\right]\delta\psi\,dt$$

- second term is not consistent: integrate by parts
- third term: $\int_0^T \left[a_1 y'' + a_2 y' + a_3 y - f(t)\right]\delta\psi\,dt = 0$ due to (FP)

$$= \int_0^T (y-\tilde{y})\,\delta y\,dt + \left[a_1\psi(\delta y)'\right]_0^T - \int_0^T (a_1\psi)'(\delta y)'\,dt$$

$$+ \left[a_2\psi\delta y\right]_0^T - \int_0^T (a_2\psi)'\delta y\,dt + \int_0^T a_3\psi\delta y\,dt - \int_0^T \psi\,\delta f\,dt$$

Practice Example: "Optimize–then–Discretize" – Deriving Gradient (cont'd)

- third term $\int_0^T (a_1\psi)'(\delta y)'\,dt$ is still not consistent: integrate by parts

$$\delta\mathcal{L}(y,t;f,\psi) = \int_0^T (y-\tilde{y})\,\delta y\,dt + \left[a_1\psi(\delta y)'\right]_0^T - \left[(a_1\psi)'\delta y\right]_0^T + \int_0^T (a_1\psi)''\delta y\,dt$$
$$+ \left[a_2\psi\delta y\right]_0^T - \int_0^T (a_2\psi)'\delta y\,dt + \int_0^T a_3\psi\delta y\,dt - \int_0^T \psi\,\delta f\,dt$$

- boundary terms $\left[(a_1\psi)'\delta y\right]_0^T$ and $\left[a_2\psi\delta y\right]_0^T$ are both zeros due to perturbation system – variational form of (FP)

$$a_1(\delta y)'' + a_2(\delta y)' + a_3\delta y = \delta f,$$
$$\delta y(0) = 0, \quad \delta y(T) = 0$$

- now we have fully consistent form

$$= \int_0^T (-\psi)\,\delta f\,dt \quad + \left[a_1\psi(\delta y)'\right]_0^T + \int_0^T \left[(y-\tilde{y}) + (a_1\psi)'' - (a_2\psi)' + a_3\psi\right]\delta y\,dt$$
$$= \langle \nabla_f\mathcal{L},\ \delta f\rangle_{L_2} \quad + \langle \nabla_y\mathcal{L},\ \delta y\rangle_{L_2}$$
$$= \int_0^T \nabla_f\mathcal{L}\,\delta f\,dt \quad + \int_0^T \nabla_y\mathcal{L}\,\delta y\,dt$$

Practice Example: "Optimize–then–Discretize" – Solving Problem

Adjoint-based gradient derived in L_2 functional space

$$\nabla_f \mathcal{L} = -\psi(t)$$

Adjoint ODE problem to be solved to find $\psi(t)$

$$\begin{aligned} &(a_1\psi)'' - (a_2\psi)' + a_3\psi = y - \tilde{y}, \\ &\psi(0) = 0, \quad \psi(T) = 0 \end{aligned} \qquad \text{(AP)}$$

Discretizing:

- time: $t_0 = 0,\ t_1 = t_0 + \Delta t,\ t_2 = t_0 + 2\Delta t,\ \ldots;\quad \Delta t = \dfrac{T - t_0}{n}$
- states, controls and adjoints:

$$\mathbf{y} = \begin{bmatrix} y_0 \\ y_1 \\ \cdots \\ y_n \end{bmatrix} = \begin{bmatrix} y(t_0) \\ y(t_1) \\ \cdots \\ y(T) \end{bmatrix}, \quad \mathbf{f} = \begin{bmatrix} f_0 \\ f_1 \\ \cdots \\ f_n \end{bmatrix}, \quad \boldsymbol{\psi} = \begin{bmatrix} \psi_0 \\ \psi_1 \\ \cdots \\ \psi_n \end{bmatrix}$$

- objective: $\mathcal{J}(\mathbf{f}) = \frac{1}{2}\sum_{i=1}^{n-1} \left(y_i(\mathbf{f}) - \tilde{y}_i\right)^2 \Delta t$

Q: in case measurements $\{\tilde{y}_j\}_{j=1}^{M}$ are pointwise (not continuous, $M \neq n+1$), how does this affect derivation of our gradient? Hint: $\mathcal{J}(\mathbf{f}) = \frac{1}{2}\sum_{j=1}^{M} \left(y_j(\mathbf{f}) - \tilde{y}_j\right)^2$

Practice Example: "Optimize–then–Discretize" – Solving Problem (cont'd)

Solution: computations for iterative reconstructions

1. Discretize time; initialize vectors for states, controls and adjoints
2. Obtain and store measurement data $(\tilde{t}_i, \tilde{y}_i)$
3. Choose initial guess $\mathbf{f}^0$ for control
4. Solve (discretized) forward problem (numerically) to find $\mathbf{y}^k$

$$\begin{aligned} a_1 y'' + a_2 y' + a_3 y &= f(t), \\ y(0) = y_0, \quad y(T) &= y_T \end{aligned} \qquad \text{(FP)}$$

5. Evaluate objective: $\mathcal{J}(\mathbf{f}^k) = \frac{1}{2} \sum_{i=1}^{n-1} \left(y_i(\mathbf{f}^k) - \tilde{y}_i \right)^2 \Delta t$
6. For $k = 1, 2, \ldots$ check optimality of $\mathbf{f}^k$, if OK (optimal) $\Rightarrow$ **STOP**
7. Solve (discretized) adjoint problem (numerically) to find $\boldsymbol{\psi}^k$

$$\begin{aligned} (a_1 \psi)'' - (a_2 \psi)' + a_3 \psi &= y - \tilde{y}, \\ \psi(0) = 0, \quad \psi(T) &= 0 \end{aligned} \qquad \text{(AP)}$$

8. Evaluate gradient: $\nabla_{\mathbf{f}} \mathcal{L}(\mathbf{y}^k; \mathbf{f}^k, \boldsymbol{\psi}^k) = -\boldsymbol{\psi}^k$
9. Improve solution by finding optimal stepsize α^k

$$\mathbf{f}^{k+1} = \mathbf{f}^k - \alpha^k \nabla_{\mathbf{f}} \mathcal{L}(\mathbf{y}^k; \mathbf{f}^k, \boldsymbol{\psi}^k)$$

10. $k \leftarrow k + 1$ and go to 4

Practice Example: "Discretize–then–Optimize" – Deriving Gradient

Discretizing main objects:

- time: $t_0 = 0,\ t_1 = t_0 + \Delta t,\ t_2 = t_0 + 2\Delta t,\ \ldots; \quad \Delta t = \dfrac{T - t_0}{n}$
- states, controls and adjoints:

$$\mathbf{y} = \begin{bmatrix} y_0 \\ y_1 \\ \ldots \\ y_n \end{bmatrix} = \begin{bmatrix} y(t_0) \\ y(t_1) \\ \ldots \\ y(T) \end{bmatrix}, \quad \mathbf{f} = \begin{bmatrix} f_0 \\ f_1 \\ \ldots \\ f_n \end{bmatrix}, \quad \boldsymbol{\psi} = \begin{bmatrix} \psi_0 \\ \psi_1 \\ \ldots \\ \psi_n \end{bmatrix}$$

- objective:

$$\mathcal{J}(\mathbf{f}) = \frac{1}{2} \sum_{i=1}^{n-1} (y_i - \tilde{y}_i)^2 \, \Delta t$$

Discretizing forward problem:

- continuous formulation (residual form)

$$\begin{aligned} & a_1 y'' + a_2 y' + a_3 y - f(t) = 0, \\ & y(0) = y_0, \quad y(T) = y_T \end{aligned} \qquad \text{(FP)}$$

- discretized form (system of $n-1$ equations)

$$\mathbf{g}(y_1, y_2, \ldots, y_{n-1};\ f_0, f_1, \ldots, f_n) = \mathbf{0} \qquad \text{(FPd)}$$

Practice Example: "Discretize–then–Optimize" – Deriving Gradient (cont'd)

Lagrangian (augmented objective function)

$$\mathcal{L}(y_1, y_2, \ldots, y_{n-1};\ f_0, f_1, \ldots, f_n,\ \psi_1, \psi_2, \ldots, \psi_{n-1})$$
$$= \frac{1}{2}\sum_{i=1}^{n-1}(y_i - \tilde{y}_i)^2\,\Delta t + \sum_{i=1}^{n-1}\psi_i \cdot \mathbf{g}_i(y_1, y_2, \ldots, y_{n-1};\ f_0, f_1, \ldots, f_n)$$

KKT optimality conditions: $\frac{\partial \mathcal{L}}{\partial y_i} = 0, \quad \frac{\partial \mathcal{L}}{\partial f_i} = 0, \quad \frac{\partial \mathcal{L}}{\partial \psi_i} = 0$

- equations of states: linear system of $n-1$ equations, $i = 1, 2, \ldots, n-1$

$$\frac{\partial \mathcal{L}}{\partial \psi_i} = g_i(y_1, y_2, \ldots, y_{n-1};\ f_0, f_1, \ldots, f_n) = 0 \qquad \text{(FPd)}$$

- adjoint equations: linear system of $n-1$ equations, $i = 1, 2, \ldots, n-1$

$$\frac{\partial \mathcal{L}}{\partial y_i} = (y_i - \tilde{y}_i)\Delta t + \sum_{j=1}^{n-1}\psi_j \frac{\partial g_j}{\partial y_i}(y_1, y_2, \ldots, y_{n-1};\ f_0, f_1, \ldots, f_n) = 0 \qquad \text{(APd)}$$

- gradient evaluation: $n+1$ components

$$\frac{\partial \mathcal{L}}{\partial f_i} = \sum_{j=1}^{n-1}\psi_j \frac{\partial g_j}{\partial f_i}(y_1, y_2, \ldots, y_{n-1};\ f_0, f_1, \ldots, f_n), \quad i = 0, 1, \ldots, n$$

Main complication: every gradient evaluation requires computing $(n-1)^2$ derivatives $\frac{\partial g_j}{\partial y_i}$ and $(n-1)(n+1)$ derivatives $\frac{\partial g_j}{\partial f_i}$ $\Rightarrow$ consider automatic differentiation (AD)

Practice Example: "Discretize–then–Optimize" – Solving Problem

Solution: computations for iterative reconstructions

1. Discretize time; initialize vectors for states, controls and adjoints
2. Obtain and store measurement data $(\tilde{t}_i, \tilde{y}_i)$ and choose initial guess $\mathbf{f}^0$ for control
3. Solve (discretized) forward problem (numerically) to find $\mathbf{y}^k$

$$g_i(y_1, y_2, \ldots, y_{n-1};\ f_0, f_1, \ldots, f_n) = 0, \quad i = 1, 2, \ldots, n-1 \qquad \text{(FPd)}$$

4. Evaluate objective: $\mathcal{J}(\mathbf{f}^k) = \frac{1}{2} \sum_{i=1}^{n-1} (y_i - \tilde{y}_i)^2 \, \Delta t$
5. For $k = 1, 2, \ldots$ check optimality of $\mathbf{f}^k$, if OK (optimal) $\Rightarrow$ **STOP**
6. Evaluate $\frac{\partial g_j}{\partial y_i}$ and solve (discretized) adjoint problem (numerically) to find $\boldsymbol{\psi}^k$

$$\sum_{j=1}^{n-1} \psi_j \frac{\partial g_j}{\partial y_i} = -(y_i - \tilde{y}_i)\Delta t, \quad i = 1, 2, \ldots, n-1 \qquad \text{(APd)}$$

7. Evaluate $\frac{\partial g_j}{\partial f_i}$ and compute $(n+1)$ components of gradient:

$$\nabla_{\mathbf{f}} \mathcal{L}(\mathbf{y}^k; \mathbf{f}^k, \boldsymbol{\psi}^k) = \sum_{j=1}^{n-1} \psi_j \frac{\partial g_j}{\partial f_i}, \quad i = 0, 1, \ldots, n$$

8. Improve solution by finding optimal stepsize α^k

$$\mathbf{f}^{k+1} = \mathbf{f}^k - \alpha^k \nabla_{\mathbf{f}} \mathcal{L}(\mathbf{y}^k; \mathbf{f}^k, \boldsymbol{\psi}^k)$$

9. $k \leftarrow k + 1$ and go to 4

1. Solving Optimal Control Problem Computationally
2. "Optimize–then–Discretize" vs. "Discretize–then–Optimize"
3. Application to ODE-based Optimal Control Problem

Example of ODE-based Model: Lotka–Volterra Equations

Lotka-Volterra (LV) Predator–Prey model is represented by a pair of 1-order nonlinear ODEs

$$\dot{x_1} = (\alpha - \beta\, x_2)\, x_1$$
$$\dot{x_2} = (-\gamma + \delta\, x_1)\, x_2$$

to describe dynamics of biological systems, i.e. change in predator/prey populations

- number of prey (rabbits, hares, etc.) $x_1(t)$
- number of predators (foxes, wolves, etc.) $x_2(t)$
- instantaneous growth rates of two populations $\dot{x_1} = \frac{dx_1}{dt}$, $\dot{x_2} = \frac{dx_2}{dt}$ w.r.t. time t
- parameters describing interaction between two species $\alpha, \beta, \gamma, \delta > 0$

Physical meaning and assumptions:

1. Prey population finds ample food at all times
2. Food supply of predator population depends entirely on the size of prey population
3. Rate of change of population is proportional to its size
4. During the process, environment does not change in favor of one species
5. Predators have limitless appetite

Kolmogorov model: a general framework that models dynamics of ecological systems with predator-prey interactions, competition, disease, and mutualism

Numerical Integration of ODEs Using MATLAB: ode23 vs. ode45

Available MATLAB functions for numerical solution of ODEs:

- ode23 solves nonstiff ODEs employing 2- and 3-order Runge-Kutta formulas for medium accuracy

  Syntaxis (mandatory/optional):

  ```
  [t, y, te, ye, ie] = ode23(odefun, tspan, y0, options)
  ```

  - t, y: each row in solution array y corresponds to a value returned in column vector t
  - odefun: function handle which defines the functions to be integrated
  - tspan: interval of integration, specified as a vector $[t_0\ t_1\ t_2\ \ldots\ t_f]$; at least two elements $[t_0\ t_f]$ should be specified
  - y0: ICs specified as a vector, must be the same length as odefun
  - options: options structure (look at MATLAB tutorial)
  - te, ye, ie: optionally find where (event) functions of (t,y) are zero

- ode45 solves nonstiff ODEs employing 4- and 5-order Runge-Kutta formulas for higher accuracy

  Syntaxis (mandatory/optional):

  ```
  [t, y, te, ye, ie] = ode45(odefun, tspan, y0, options)
  ```

https://www.mathworks.com/help/matlab/ref/ode23.html
https://www.mathworks.com/help/matlab/ref/ode45.html

Problem: Optimization Model Constrained by LV Equations

Problem: $u(t)$-parameter identification in nonlinear IVP-1 system defined by LV-problem by fitting (continuous) data $x_1^*(t)$ and $x_2^*(t)$, $t \in (0, T]$

$$\min_{u(t)} \mathcal{J}(x_1, x_2, t; u) = \beta_1 \int_0^T (x_1 - \tilde{x}_1)^2 \, dt + \beta_2 \int_0^T (x_2 - \tilde{x}_2)^2 \, dt$$

- subject to following Lotka-Volterra Predator–Prey model

$$\dot{x_1} = (\alpha - \beta \, x_2) \, x_1 - u(t) \, x_1$$
$$\dot{x_2} = -(\gamma - \delta \, x_1) \, x_2 - u(t) \, x_2$$
$$x_1(0) = x_1^0, \qquad x_2(0) = x_2^0, \qquad t \in (0, T]$$

- when data $\tilde{x}_1(t)$ and $\tilde{x}_2(t)$ are available continuously over interval $[0, T]$, and
- constants $\alpha, \beta, \gamma, \delta \geq 0$ are given.

Approach:

- solve Problem using adjoint-based gradient method (Lagrange multiplier approach)
- derive gradients by "optimize–then–discretize" approach – optimality conditions and optimization algorithm are based on continuous form of the problem

LV Optimization Model: Deriving Gradient

Objective function

$$\mathcal{J}(x_1, x_2, t; u) = \beta_1 \int_0^T (x_1 - \tilde{x}_1)^2 \, dt + \beta_2 \int_0^T (x_2 - \tilde{x}_2)^2 \, dt$$

defined over control space

$$\mathcal{U} = \{u(t) \in L_2[0, T] : 0 \le u(t) \le 1 \text{ for a.e. } t \in [0, T]\}$$

Forward problem (governing equation, equations of state) by LV-problem

$$\begin{aligned} \dot{x_1} &= (\alpha - \beta\, x_2)\, x_1 - u\, x_1 \\ \dot{x_2} &= -(\gamma - \delta\, x_1)\, x_2 - u\, x_2 \qquad\qquad (\mathrm{FP}) \\ x_1(0) = x_1^0, &\quad x_2(0) = x_2^0, \quad t \in (0, T] \end{aligned}$$

- state variables $x_1(t)$ and $x_2(t)$
- control variable $u(t)$: keeping in mind additional constraint $0 \le u(t) \le 1$
- two adjoint variables (Lagrange multipliers) $\psi_1(t)$ and $\psi_2(t)$ will be assigned for each equation in LV-system (FP)

LV Optimization Model: Deriving Gradient (cont'd)

Lagrangian (augmented objective function) defined in L_2 functional space

$$\mathcal{L}(x_1, x_2, t; u, \psi_1, \psi_2) = \beta_1 \int_0^T (x_1 - \tilde{x}_1)^2 \, dt + \beta_2 \int_0^T (x_2 - \tilde{x}_2)^2 \, dt$$
$$+ \int_0^T [\dot{x}_1 - (\alpha - \beta\, x_2)x_1 + ux_1]\, \psi_1 \, dt + \int_0^T [\dot{x}_2 + (\gamma - \delta \cdot x_1)x_2 + ux_2]\, \psi_2 \, dt$$

As done previously (see Part 12), 1-order variation for this Lagrangian

$$\delta\mathcal{L}(x_1, x_2, t; u, \psi_1, \psi_2) = 2\beta_1 \int_0^T (x_1 - \tilde{x}_1)\delta x_1 \, dt + 2\beta_2 \int_0^T (x_2 - \tilde{x}_2)\delta x_2 \, dt$$
$$+ \int_0^T [\delta\dot{x}_1 + \beta x_1\, \delta x_2 - (\alpha - \beta\, x_2)\delta x_1 + x_1 \delta u + u\delta x_1]\, \psi_1 \, dt$$
$$+ \int_0^T [\dot{x}_1 - (\alpha - \beta\, x_2)x_1 + ux_1]\, \delta\psi_1 \, dt + \int_0^T [\dot{x}_2 + (\gamma - \delta \cdot x_1)x_2 + ux_2]\, \delta\psi_2 \, dt$$
$$+ \int_0^T [\delta\dot{x}_2 - \delta \cdot x_2\, \delta x_1 + (\gamma - \delta \cdot x_1)\delta x_2 + x_2 \delta u + u\delta x_2]\, \psi_2 \, dt$$

should be consistent with Riesz Representation Theorem and set to 0 (KKT conditions)

$$\delta\mathcal{L}(x_1, x_2, t; u, \psi_1, \psi_2) = \int_0^T \nabla_u \mathcal{L}\, \delta u \, dt + \int_0^T \nabla_x \mathcal{L}\, \delta \mathbf{x} \, dt + \int_0^T \nabla_\psi \mathcal{L}\, \delta\psi \, dt$$

LV Optimization Model: Deriving Gradient (cont'd)

Due to (FP) 4th and 5th terms

$$\int_0^T [\dot{x}_1 - (\alpha - \beta\, x_2)x_1 + ux_1]\, \delta\psi_1\, dt + \int_0^T [\dot{x}_2 + (\gamma - \delta \cdot x_1)x_2 + ux_2]\, \delta\psi_2\, dt = 0$$

Some parts of 3rd and 7th terms are not consistent: integrate by parts

$$\int_0^T \psi_1 \delta\dot{x}_1\, dt = [\psi_1 \delta x_1]_0^T - \int_0^T \dot{\psi}_1 \delta x_1\, dt$$
$$\int_0^T \psi_2 \delta\dot{x}_2\, dt = [\psi_2 \delta x_2]_0^T - \int_0^T \dot{\psi}_2 \delta x_2\, dt$$

Boundary terms $[\psi_1 \delta x_1]_0$ and $[\psi_2 \delta x_2]_0$ are zeros due to perturbation system – variational form of (FP)

$$\delta\dot{x}_1 = -\beta x_1 \delta x_2 + (\alpha - \beta\, x_2)\, \delta x_1 - x_1 \delta u - u\, \delta x_1$$
$$\delta\dot{x}_2 = \delta \cdot x_2 \delta x_1 - (\gamma - \delta \cdot x_1)\, \delta x_2 - x_2 \delta u - u\, \delta x_2$$
$$\delta x_1(0) = 0, \quad \delta x_2(0) = 0, \quad t \in (0, T]$$

LV Optimization Model: Deriving Gradient (cont'd)

And now we have fully consistent form

$$\delta\mathcal{L}(x_1, x_2, t; u, \psi_1, \psi_2) = 2\beta_1 \int_0^T (x_1 - \tilde{x}_1)\delta x_1 \, dt + 2\beta_2 \int_0^T (x_2 - \tilde{x}_2)\delta x_2 \, dt$$
$$+ \psi_1(T)\delta x_1 - \int_0^T \dot{\psi}_1 \delta x_1 \, dt + \psi_2(T)\delta x_2 - \int_0^T \dot{\psi}_2 \delta x_2 \, dt$$
$$+ \int_0^T [\beta x_1 \, \delta x_2 - (\alpha - \beta \, x_2)\delta x_1 + x_1 \delta u + u\delta x_1] \, \psi_1 \, dt$$
$$+ \int_0^T [-\delta \cdot x_2 \, \delta x_1 + (\gamma - \delta \cdot x_1)\delta x_2 + x_2 \delta u + u\delta x_2] \, \psi_2 \, dt$$

after grouping all terms by factoring δu, δx_1 and δx_2

$$\delta\mathcal{L}(x_1, x_2, t; u, \psi_1, \psi_2) = \int_0^T [x_1\psi_1 + x_2\psi_2] \, \delta u \, dt$$
$$+ \psi_1(T)\delta x_1 + \int_0^T \left[-\dot{\psi}_1 + 2\beta_1(x_1 - \tilde{x}_1) - (\alpha - \beta \, x_2)\psi_1 + u\psi_1 - \delta \cdot x_2\psi_2\right] \delta x_1 \, dt$$
$$+ \psi_2(T)\delta x_2 + \int_0^T \left[-\dot{\psi}_2 + 2\beta_2(x_2 - \tilde{x}_2) + \beta x_1\psi_1 + (\gamma - \delta \cdot x_1)\psi_2 + u\psi_2\right] \delta x_2 \, dt$$

LV Optimization Model: Optimizing in Discretized Settings

Adjoint-based gradient derived in L_2 functional space

$$\nabla_u \mathcal{L} = x_1(t)\psi_1(t) + x_2(t)\psi_2(t)$$

Adjoint ODE problem to be solved to find $\psi_1(t)$ and $\psi_2(t)$ with terminal conditions

$$\begin{aligned} \dot{\psi}_1 &= 2\beta_1(x_1 - \tilde{x}_1) - (\alpha - \beta x_2)\psi_1 + u\psi_1 - \delta x_2\psi_2 \\ \dot{\psi}_2 &= 2\beta_2(x_2 - \tilde{x}_2) + \beta x_1\psi_1 + (\gamma - \delta x_1)\psi_2 + u\psi_2 \qquad \text{(AP)} \\ \psi_1(T) &= 0, \quad \psi_2(T) = 0, \quad t \in [0, T) \end{aligned}$$

Discretizing:

- time: $\mathbf{t} = [t_0 \ t_1 \ \dots \ t_n]^T, \ t_0 = 0, \ t_1 = t_0 + \Delta t, \ \dots; \quad \Delta t = \dfrac{T - t_0}{n}$
- states, controls and adjoints:

$$\mathbf{x} = [\mathbf{x}_1^k \ \mathbf{x}_2^k] = \begin{bmatrix} x_1^0 & x_2^0 \\ x_1^1 & x_2^1 \\ \dots & \dots \\ x_1^n & x_2^n \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} u_0 \\ u_1 \\ \dots \\ u_n \end{bmatrix}, \quad \boldsymbol{\psi} = [\boldsymbol{\psi}_1^k \ \boldsymbol{\psi}_2^k] = \begin{bmatrix} \psi_1^0 & \psi_2^0 \\ \psi_1^1 & \psi_2^1 \\ \dots & \dots \\ \psi_1^n & \psi_2^n \end{bmatrix}$$

- objective: $\mathcal{J}(\mathbf{u}) = \beta_1 \sum_{i=1}^{n-1} \left(x_1^i(\mathbf{u}) - \tilde{x}_1^i\right)^2 \Delta t + \beta_2 \sum_{i=1}^{n-1} \left(x_2^i(\mathbf{u}) - \tilde{x}_2^i\right)^2 \Delta t$

LV Optimization Model: Complete Optimization Algorithm (iterative)

Solution: computations for iterative reconstructions

1. Discretize time; initialize vectors for states, controls and adjoints
2. Obtain and store measurement (analytic/synthetic) data $(\tilde{t}_i, \tilde{x}_1^i)$ and $(\tilde{t}_i, \tilde{x}_2^i)$
3. Choose initial guess $\mathbf{u}^0$ for control
4. Solve (discretized) forward problem (numerically) to find $\mathbf{x}^k = [\mathbf{x}_1^k \ \mathbf{x}_2^k]$

$$\begin{aligned} \dot{x_1} &= (\alpha - \beta\, x_2)\, x_1 - u(t)\, x_1 \\ \dot{x_2} &= -(\gamma - \delta\, x_1)\, x_2 - u(t)\, x_2 \qquad \text{(FP)} \\ x_1(0) = x_1^0, &\quad x_2(0) = x_2^0, \quad t \in (0, T] \end{aligned}$$

5. Evaluate objective: $\mathcal{J}(\mathbf{u}) = \beta_1 \sum_{i=1}^{n-1} \left(x_1^i(\mathbf{u}) - \tilde{x}_1^i\right)^2 \Delta t + \beta_2 \sum_{i=1}^{n-1} \left(x_2^i(\mathbf{u}) - \tilde{x}_2^i\right)^2 \Delta t$
6. For $k = 1, 2, \ldots$ check optimality of $\mathbf{u}^k$, if OK (optimal) $\Rightarrow$ **STOP**
7. Solve (discretized) adjoint problem (numerically) to find $\boldsymbol{\psi}^k = [\boldsymbol{\psi}_1^k \ \boldsymbol{\psi}_2^k]$

$$\begin{aligned} \dot{\psi_1} &= 2\beta_1(x_1 - \tilde{x}_1) - (\alpha - \beta\, x_2)\psi_1 + u\psi_1 - \delta x_2 \psi_2 \\ \dot{\psi_2} &= 2\beta_2(x_2 - \tilde{x}_2) + \beta x_1 \psi_1 + (\gamma - \delta x_1)\psi_2 + u\psi_2 \qquad \text{(AP)} \\ \psi_1(T) = 0, &\quad \psi_2(T) = 0, \quad t \in [0, T) \end{aligned}$$

8. Evaluate gradient: $\boldsymbol{\nabla}_u \mathcal{L}(\mathbf{x}^k; \mathbf{u}^k, \boldsymbol{\psi}^k) = \mathbf{x}_1^k \circ \boldsymbol{\psi}_1^k + \mathbf{x}_2^k \circ \boldsymbol{\psi}_2^k$
9. Improve solution by finding optimal stepsize α^k: $\mathbf{u}^{k+1} = \mathbf{u}^k - \alpha^k \boldsymbol{\nabla}_{\mathbf{u}} \mathcal{L}(\mathbf{x}^k; \mathbf{u}^k, \boldsymbol{\psi}^k)$
10. $k \leftarrow k + 1$ and go to 4

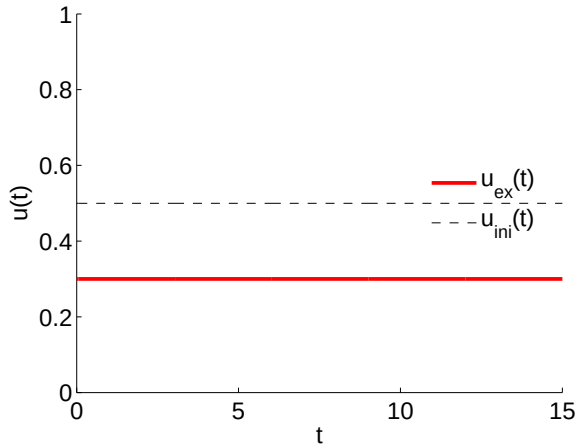
Benchmark Models for Control $u(t)$: `models_ctrl.m`

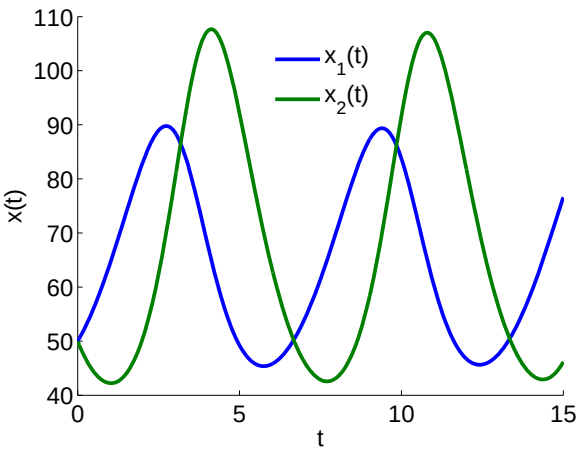
Goal: to check computational performance of optimization framework in reconstructing control functions $u(t)$ of different level of complexity

Model #1

constant function

$u(t) = 0.3$

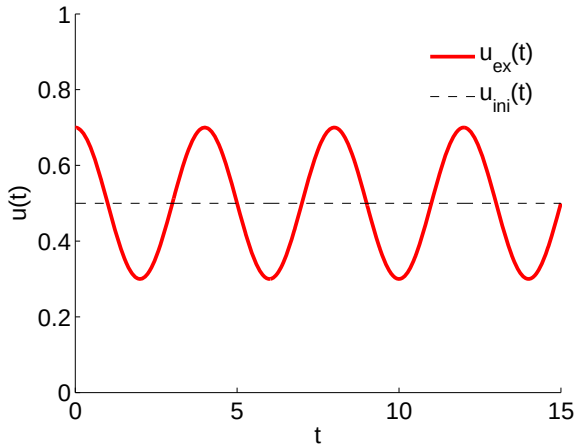


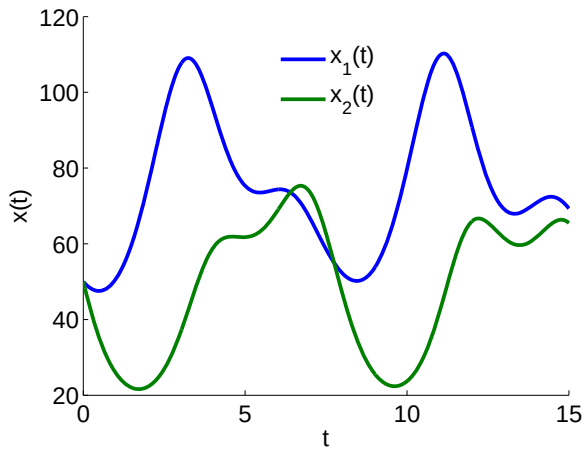


Model #2

smooth periodic

$u(t) = \frac{1}{5}\cos\left(\frac{\pi}{2}t\right) + 0.5$

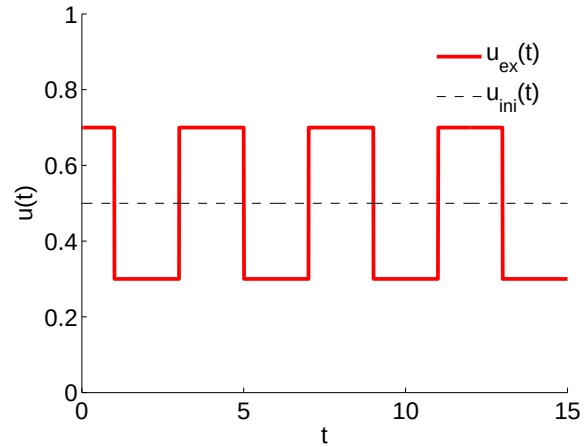


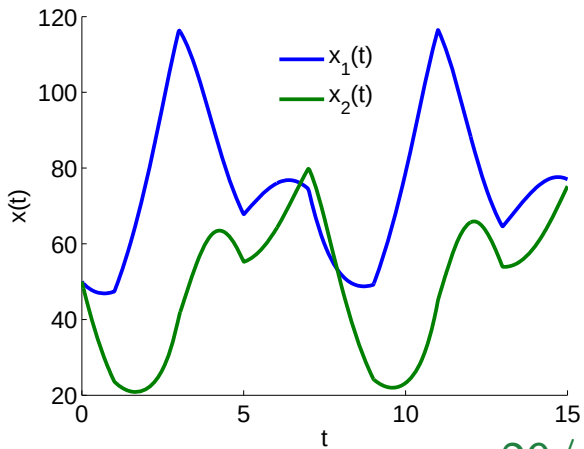


Model #3

discontinuous (step/signum)

$u(t) = \frac{1}{5}\mathrm{sign}\left[\cos\left(\frac{\pi}{2}t\right)\right] + 0.5$

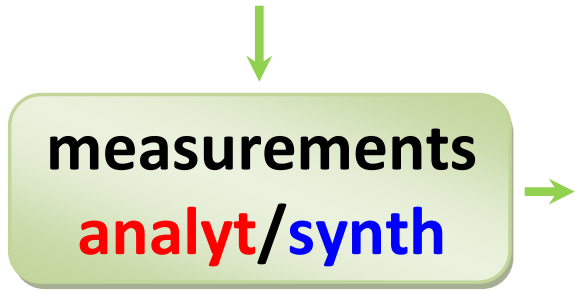




Measurements: Analytic vs. Synthetic

Real data are good, but

- not available for every model
- may not be used to validate accuracy and performance of computational framework



Analytic measurements:

1. Find analytic functions (exact solutions) $x_{1,ex}(t)$, $x_{2,ex}(t)$ and $u_{ex}(t)$ that satisfy forward problem (FP)
2. Discretize time $t \in [0, T]$ to obtain vector $\mathbf{t} = [t_0 \ t_1 \ \ldots \ t_n]^T$
3. Evaluate $x_{1,ex}(\mathbf{t})$, $x_{2,ex}(\mathbf{t})$ to obtain two vectors $\tilde{\mathbf{x}}_1$ and $\tilde{\mathbf{x}}_2$
4. Use analytic measurements $(\tilde{t}_i, \tilde{x}_1^i)$ and $(\tilde{t}_i, \tilde{x}_2^i)$ to substitute real data

- It is hard to implement for complex models, but
- may also be used to check accuracy of $\mathcal{J}$-evaluator

Measurements: Analytic vs. Synthetic (cont'd)

Synthetic measurements:

1. Derive analytic function (exact solution) $u_{ex}(t)$
2. Discretize time $t \in [0, T]$ to obtain vector $\mathbf{t} = [t_0 \ t_1 \ \ldots \ t_n]^T$
3. Solve numerically forward problem (FP) to obtain solutions for $x_1(t)$ and $x_2(t)$ in terms of two vectors $\tilde{\mathbf{x}}_1$ and $\tilde{\mathbf{x}}_2$
4. Use synthetic measurements $(\tilde{t}_i, \tilde{x}_1^i)$ and $(\tilde{t}_i, \tilde{x}_2^i)$ to substitute real data

- It is simple to implement, but
- requires interpolation if $\mathbf{t}$-discretization is changed over optimization process or pointwise measurements are used

- m-code `measurements.m` to obtain synthetic measurements

```
data = [tt zeros(Nt+1,1) zeros(Nt+1,1)]; % preallocation to structure data matrix

[Cf,x1meas,x2meas] = f(0.0, u_ex, zeros(size(u_ex)), params, data); % calling J-evaluator

data(:,2:3) = [x1meas x2meas]; % updating data matrix
```

Choosing Parameters and Tuning Optimization Algorithms

Computational algorithm: `opt_lotka_volterra.m`

main OPT-part:	written manually (adapted from previous model)	[MATLAB]
$\mathcal{J}$-evaluator:	m-function using ODE solver ode23/45 for (FP)	[MATLAB]
d-evaluator:	m-function using ODE solver ode23/45 for (AP) & analytically defined $\nabla_u \mathcal{J}$ for SD, CG, BFGS	[MATLAB]
1D search for α:	m-function for Golden Section & bracketing-Brent	[MATLAB]
visualizer:	m-code with subplots	[MATLAB]

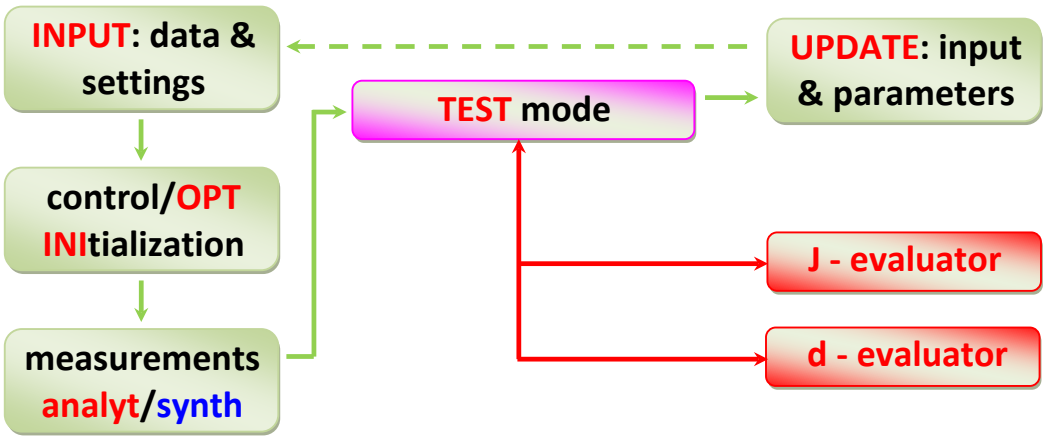
Main parameters: `params.m`

- Model: $t \in [0, 15]$, $n = 100$, $x_1^0 = x_2^0 = 50$, $\alpha = 1$, $\beta = 0.01$, $\gamma = 1$, $\delta = 0.02$
- Optimization: $\beta_1 = \beta_2 = 1.0$, $u_{ini}(t) = 0.5$
- SD, CG, BFGS: termination $\epsilon = 10^{-9}$, $k_{max} = 200$; BFGS restarts $= 2, 5, 10$
- GS: search interval $[0, 10^{-6}]$, termination $\epsilon_\alpha = 10^{-9}$
- BB: initial interval $[0, 10^{-6}]$; bracketing `MAXITER` $= 20$, `GLIMIT` $= 100.0$; Brent: `TOL` $= 10^{-9}$, `ITMAX` $= 2$

Choosing Parameters and Tuning Optimization Algorithms: params.m

```
% choosing mode & solution method
mode   = 'OPT';                    % OPT or TEST
method = 'SD';                     % SD, CG, BFGS
BFGSrst = 2;                       % restart parameter for BFGS

% MODEL parameters:
% t-domain
ti = 0; tf = 15;                   % t-domain
Nt = 100;                          % number of discrete intervals in t-domain
% x-solution
x1i = 50.0; x2i = 50.0;            % initial conditions for LV-IVP
% coefficients in the main LV-ODE system
paramsODE = [1.0 0.01 1.0 0.02]; % alpha, beta, gamma, delta
modelU = 2;                        % exact u(t) model number (1, 2, 3)

% OPT setup:
objWeight = [1.0 1.0];             % weight coefficients in objective
u_ini = 0.5;                       % initial guess for u(t) - constant
u_l = 0.0; u_u = 1.0;              % lower and upper bounds for control (for later use)

% [SD, CG, BFGS] - parameters for alpha stepsize search
methodAlpha = 'GS';                % method for alpha stepsize search: GS, BB
% (1) GS
alphaA = 0; alphaB = 1e-6;         % search interval [a, b]
alphaEps = 1e-9;                   % tolerance for search termination
% (2) bracketing-Brent toolbox
AX = 0; BX = 1e-6;                 % min_brack: initial bracketing interval [a, b]
MAXITER = 20;                      % min_brack: max number of iterations
GLIMIT = 100.0;                    % min_brack: maximum magnification for parabolic-fit step
TOL = 1e-9; ITMAX = 2;             % Brent: tolerance & max number of iterations

% termination
epsilonF = 1e-9;                   % tolerance (#1) relative objective decrease
epsilonA = 1e-9;                   % tolerance (#2) relative solution decrease
kMax     = 200;                    % max number of iterations (#3)
```

Checking Quality of Discretized Gradients: TEST Mode



1D case implementation (by FD-1):

$$f'(x) \approx \frac{f(x+\Delta x) - f(x)}{\Delta x},$$

$$\kappa = \frac{f(x+\epsilon\,\Delta x) - f(x)}{\epsilon\; f'(x)\cdot \Delta x} \to 1$$

if $\Delta x \to 0$ and $\epsilon \ll 1$.

Extension for current multidimensional case, $\mathbf{u} \in \mathbb{R}^n$, "kappa-test":

$$\kappa(\epsilon) = \frac{\mathcal{J}(\mathbf{u}+\epsilon\,\delta\mathbf{u}) - \mathcal{J}(\mathbf{u})}{\epsilon\,\langle \boldsymbol{\nabla}_{\mathbf{u}}\mathcal{J}(\mathbf{u}),\, \delta\mathbf{u}\rangle}$$

- "cheap test": requires minimum 2 $\mathcal{J}$-evaluations

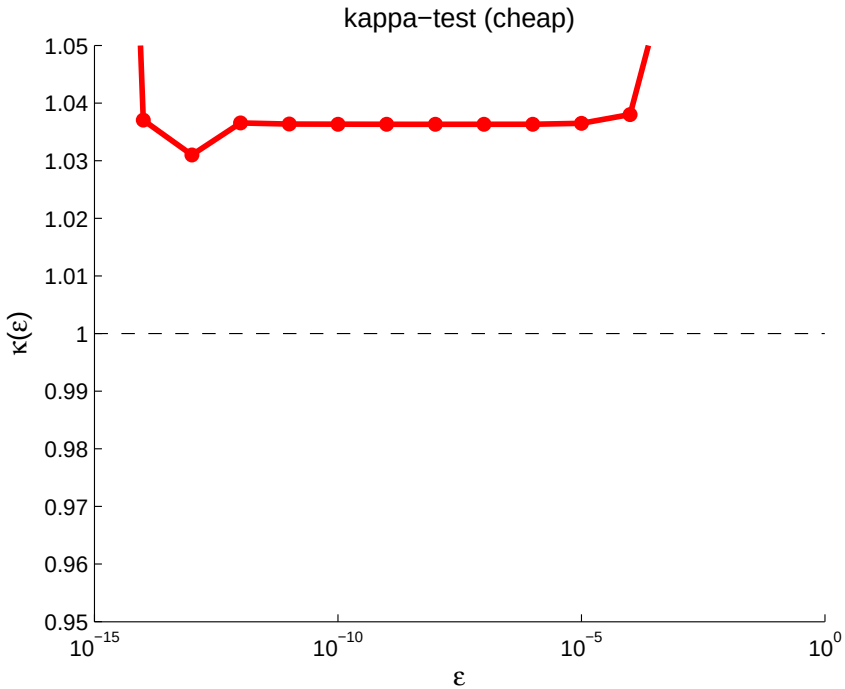
  for fixed $\delta\mathbf{u}$, e.g. $\delta\mathbf{u} = \mathbf{u}$, compute $\kappa(\epsilon)$ for a range of ϵ, e.g. $\epsilon = 10^{-15} \div 10^{0}$

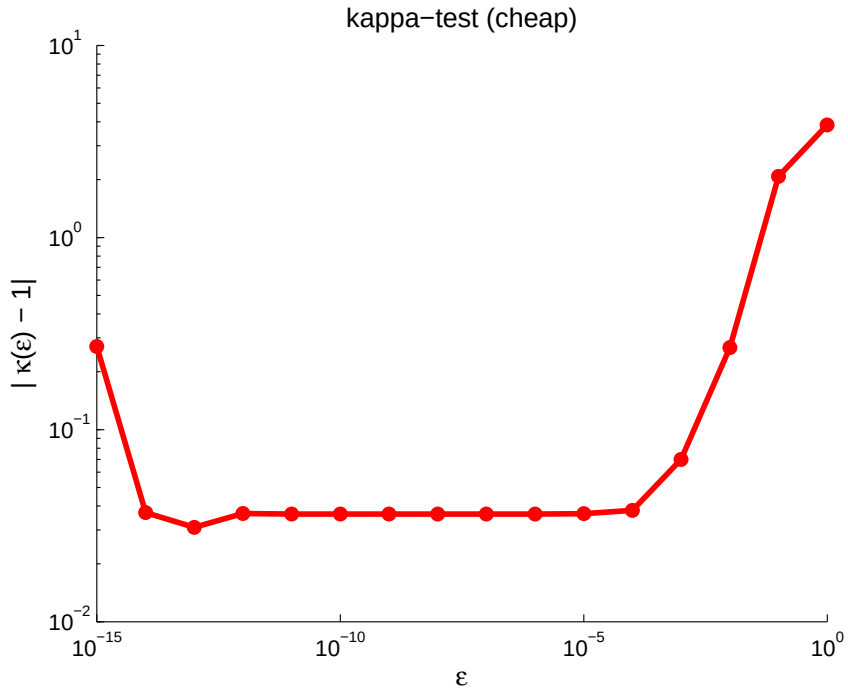
- "expensive test": requires $n+1$ $\mathcal{J}$-evaluations

  for fixed ϵ, e.g. $\epsilon = 10^{-6}$, perform test changing $\delta\mathbf{u}$: $[u_1\ 0\ 0\ \ldots\ 0]^T$, $[0\ u_2\ 0\ \ldots\ 0]^T$, $\ldots$, $[0\ 0\ 0\ \ldots\ u_n]^T$ (checking sensitivity to every $\mathbf{u}$-component)

Checking Quality of Discretized Gradients: TEST Mode (cont'd)

LV-problem: "cheap test" for gradient in Model #2

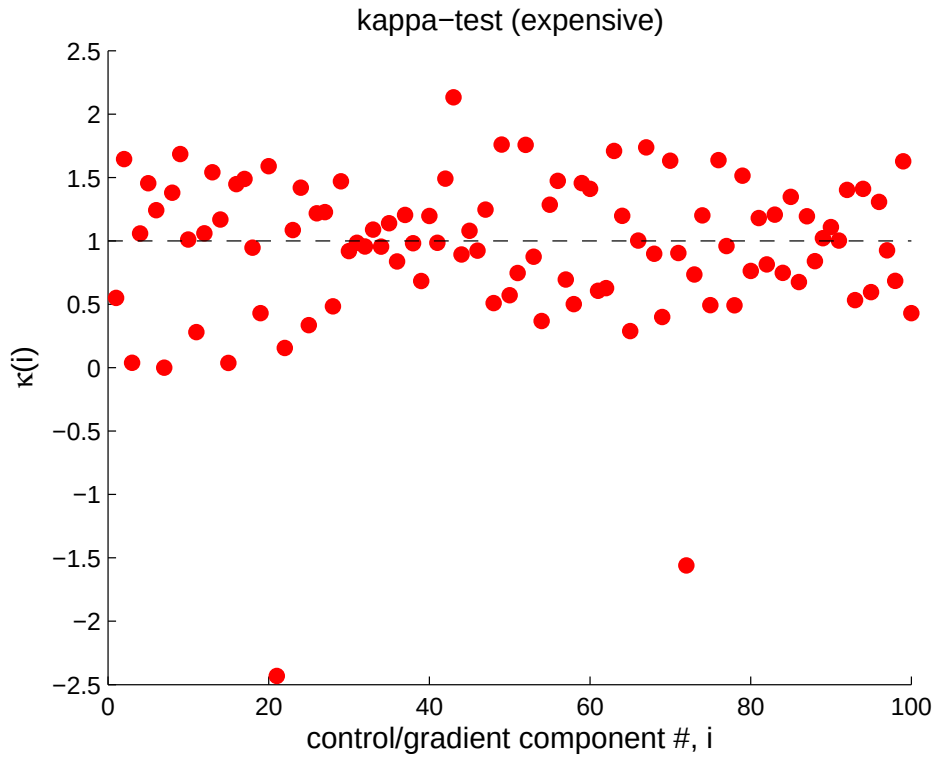




- correctness of gradient: range of ϵ spans 8-9 orders of magnitude
- well-known effects: $\kappa(\epsilon)$ deviates from the unity:
  - for very small values of ϵ due to subtractive cancelation (roundoff) errors
  - for large values of ϵ due to truncation errors
- quantity $\log_{10} |\kappa(\epsilon) - 1|$ shows how many significant digits of accuracy are captured in gradient evaluation

Checking Quality of Discretized Gradients: TEST Mode (cont'd)

LV-problem: "expensive test" in Model #2



- correctness of *i*-th gradient component: component-wise sensitivity analysis (accuracy)
- easy problem identification: accuracy of gradient vs. sensitivity by single controls
- both tests, "cheap" and "expensive", may be repeated throughout the optimization process to control error/loss of sensitivity propagation

Improving Quality of Discretized Gradients

Q: What can we do to improve quality of gradients, and thus, to expect better performance of optimization?

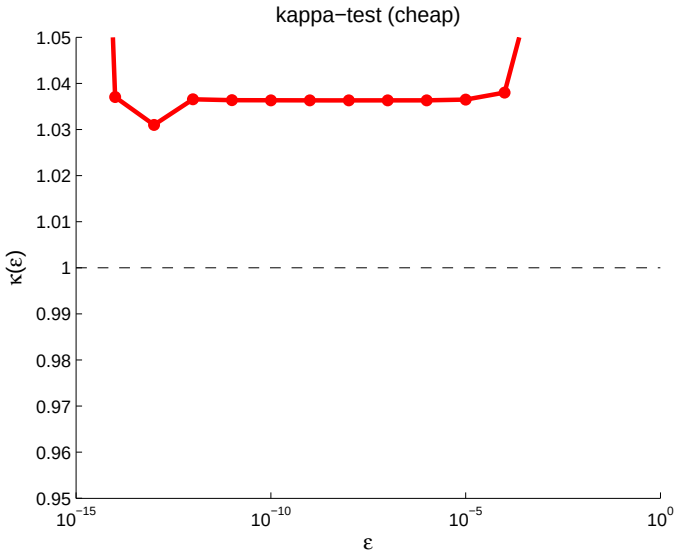
For current problem:

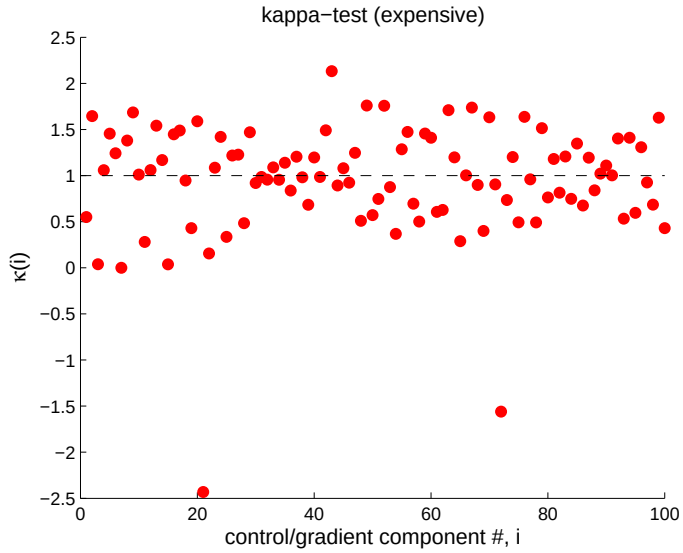
- refine time discretization: check $n = 50$, 100 (current), 500, 1000 (goes to homework)
- use higher order ODE solver: change `ode23` (current) to `ode45` (goes to homework)
- change parameters for ODE solver: use keyword `odeset` to check/change settings (see example below & next slide)

```
>> odeset

      AbsTol:   [ positive scalar or vector {1e-6} ]
      RelTol:   [ positive scalar {1e-3} ]
 NormControl:   [ on | {off} ]
 NonNegative:   [ vector of integers ]
         ...    ...
```

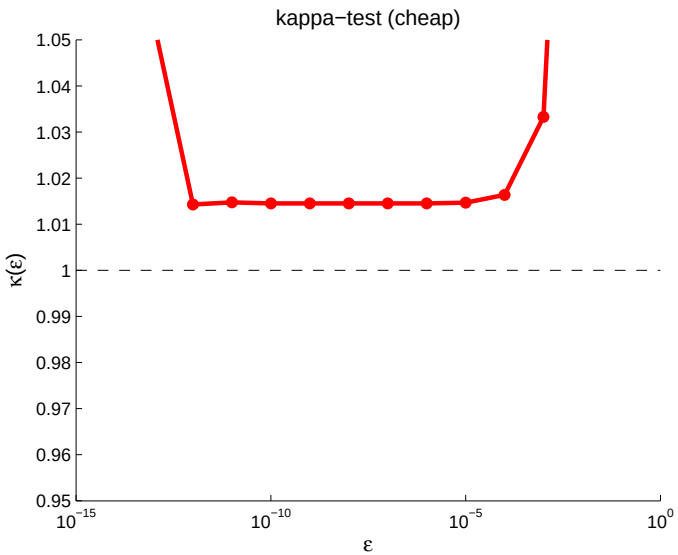
Improving Quality of Discretized Gradients: Tuning ode23

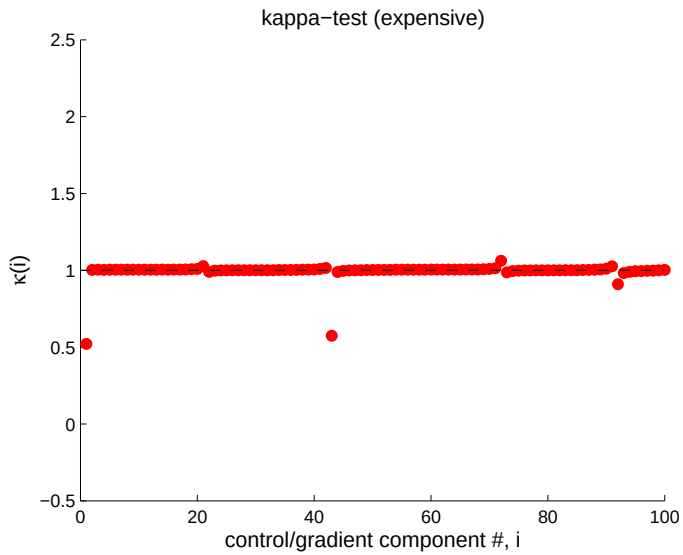
- `odeset('RelTol',1e-3,'AbsTol',1e-6); CPU elapsed time =` 4.0355s



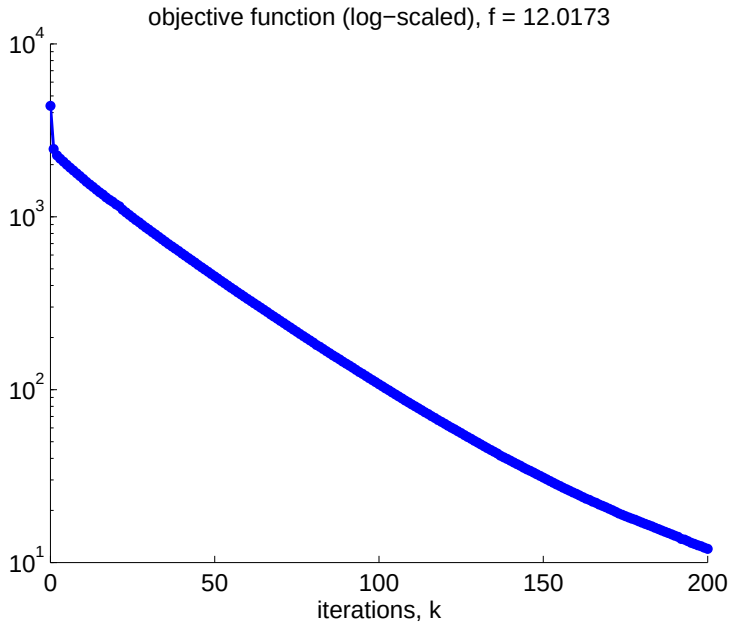


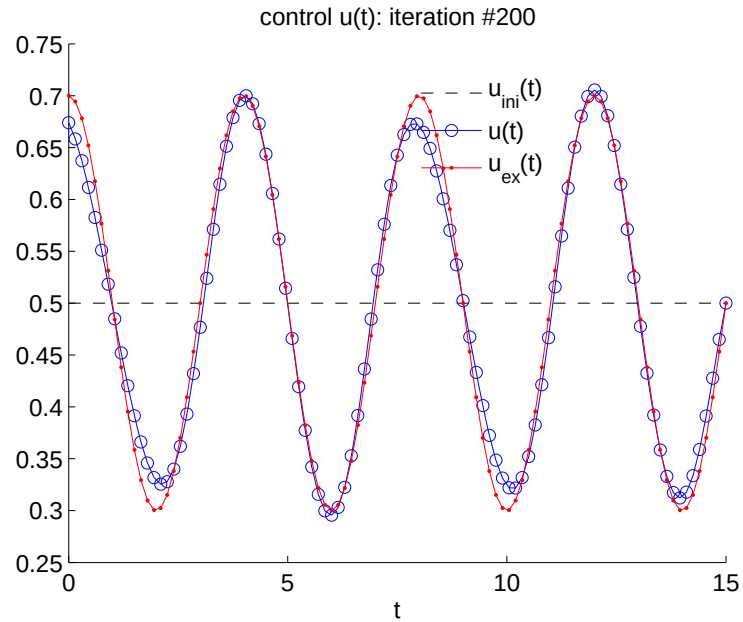
- `odeset('RelTol',1e-9,'AbsTol',1e-12); CPU elapsed time =` 260.4798s

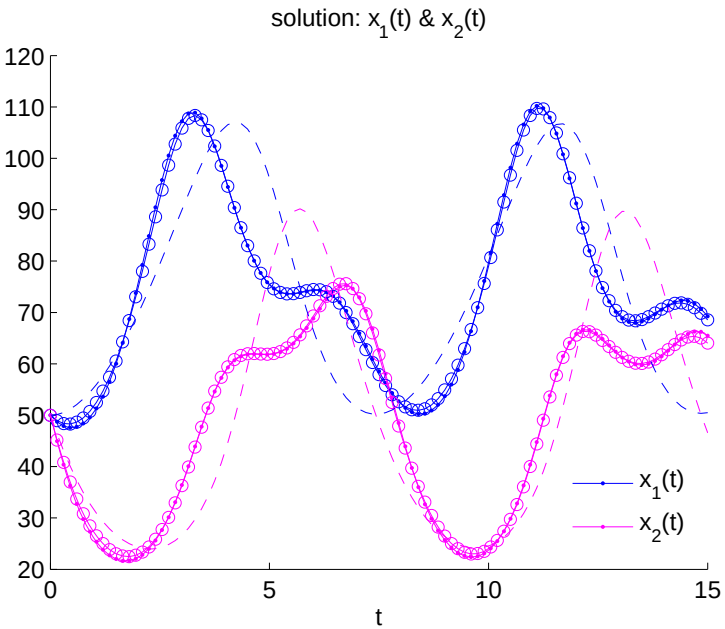


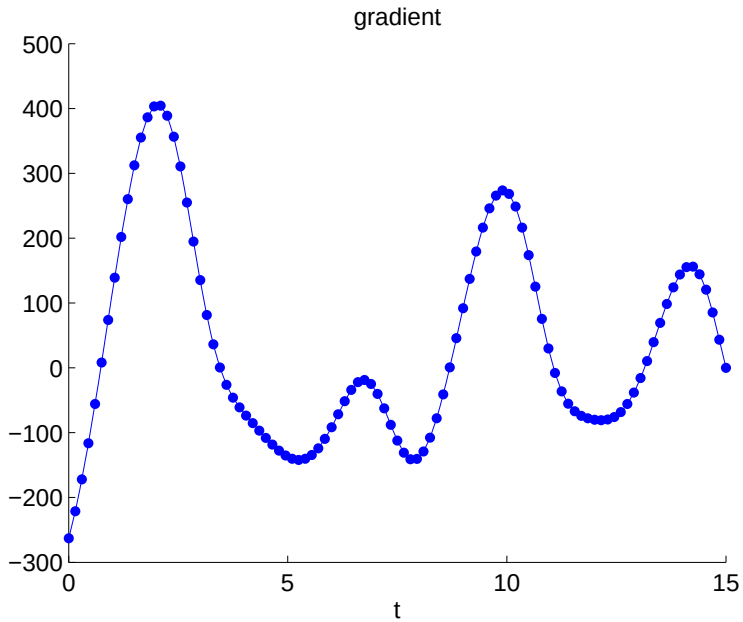


First Optimization Results: Model #2 (SD + GS)

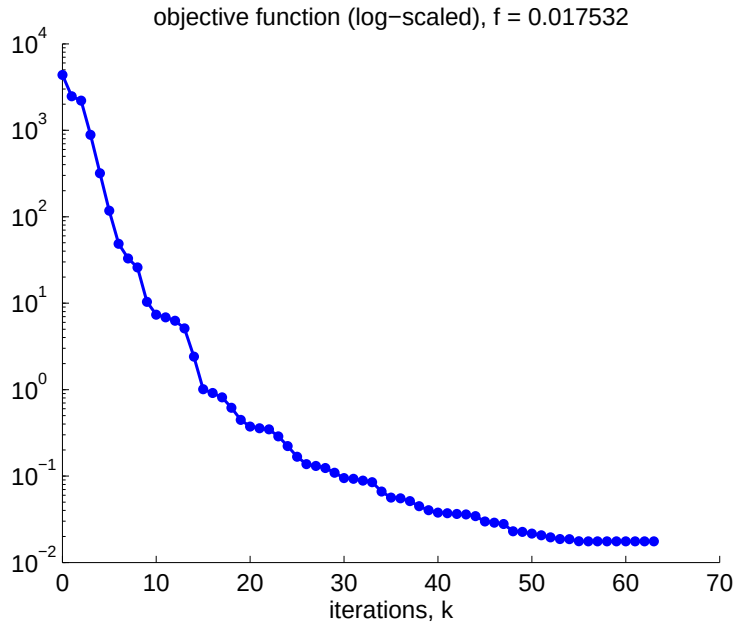


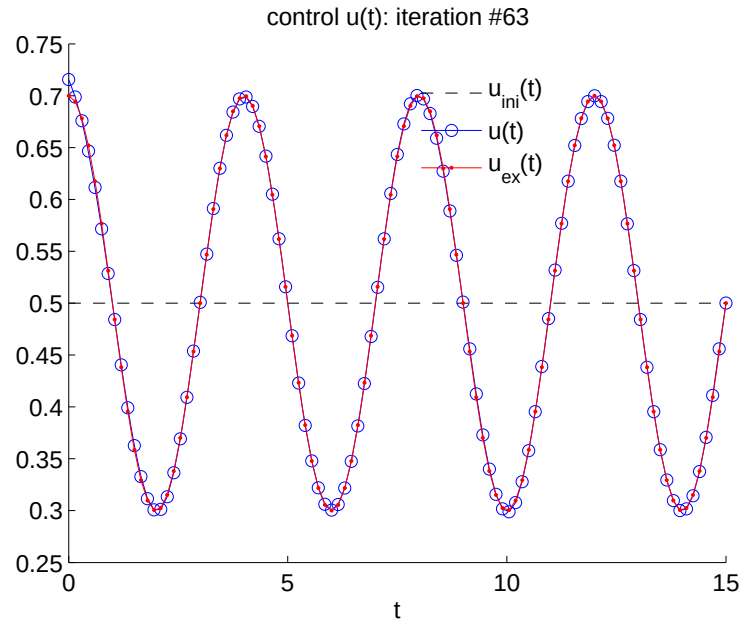


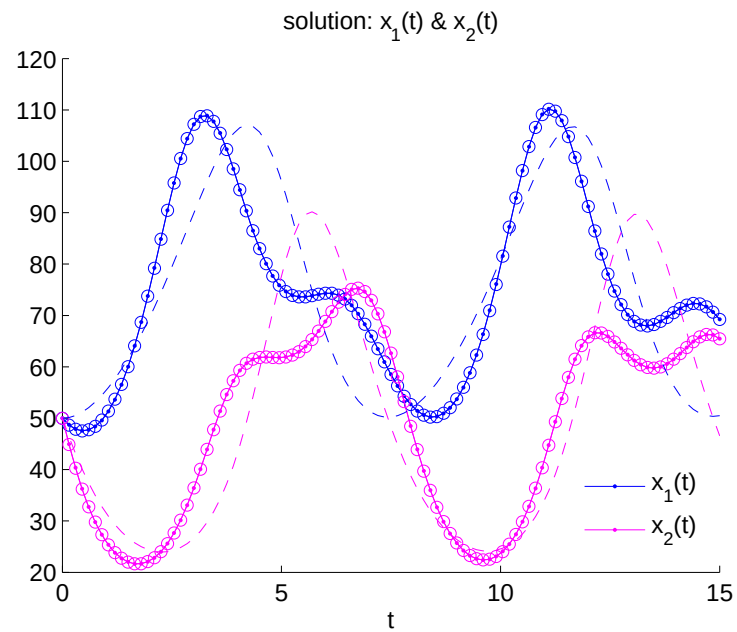


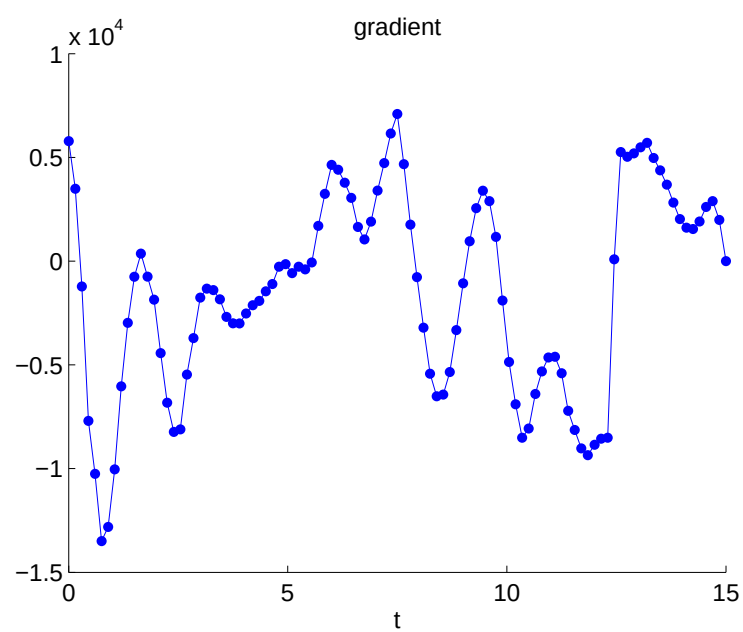


First Optimization Results: Model #2 ($\text{BFGS}_{rst=5}$ + BB)

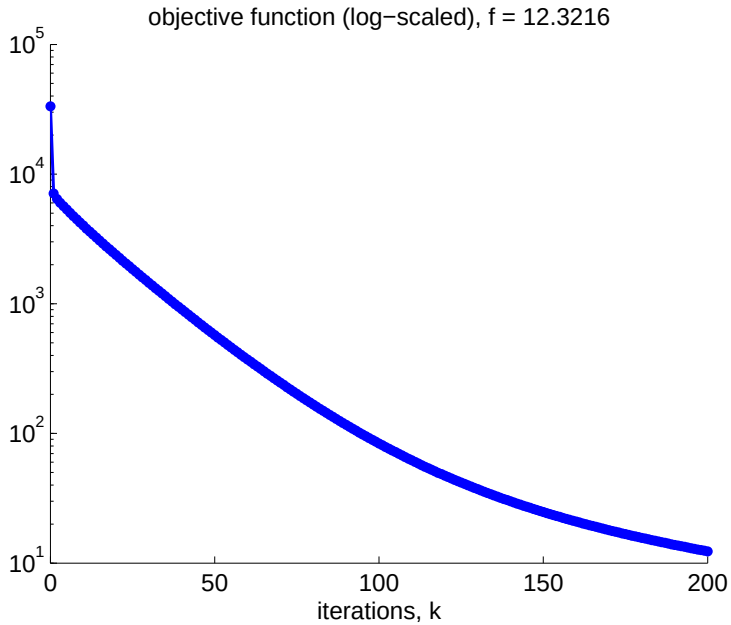


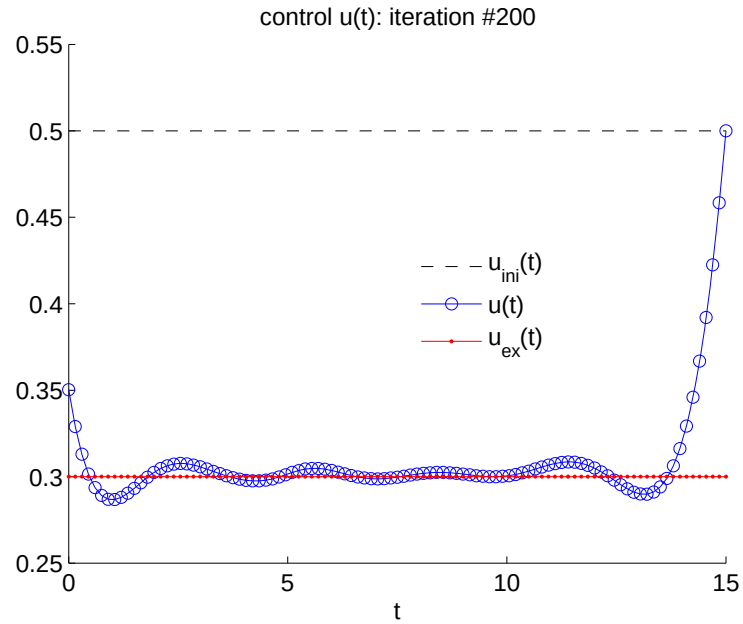


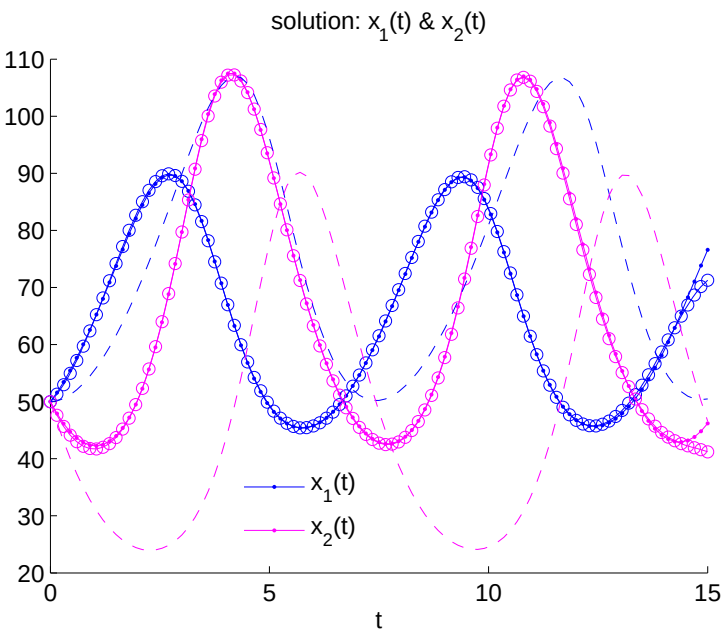


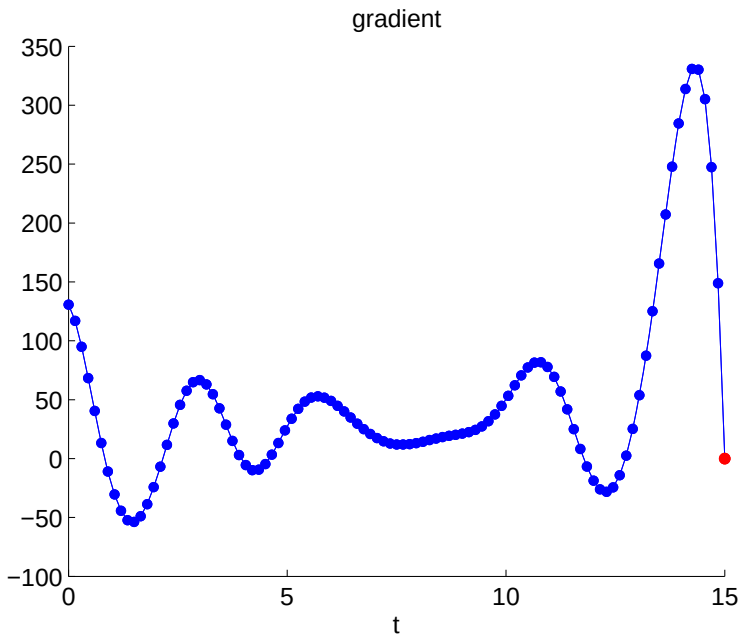


First Optimization Results: Model #1 (SD + GS)

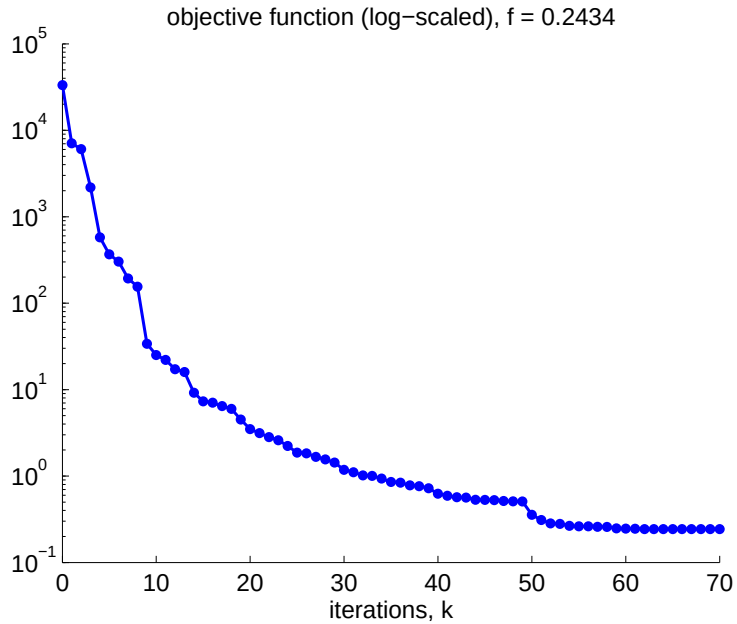


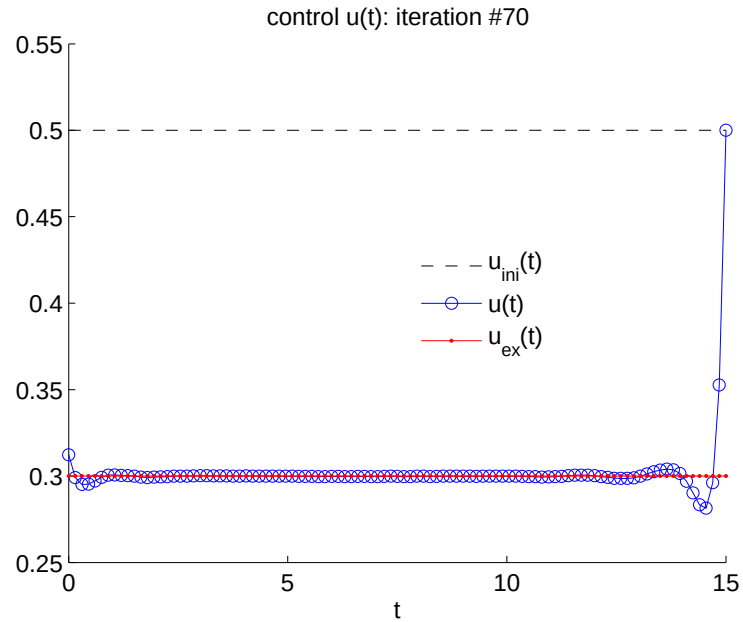


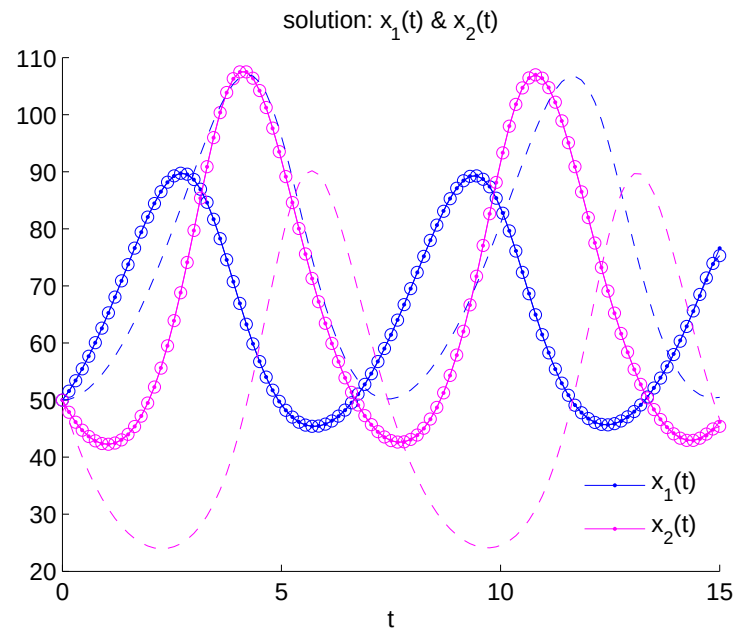


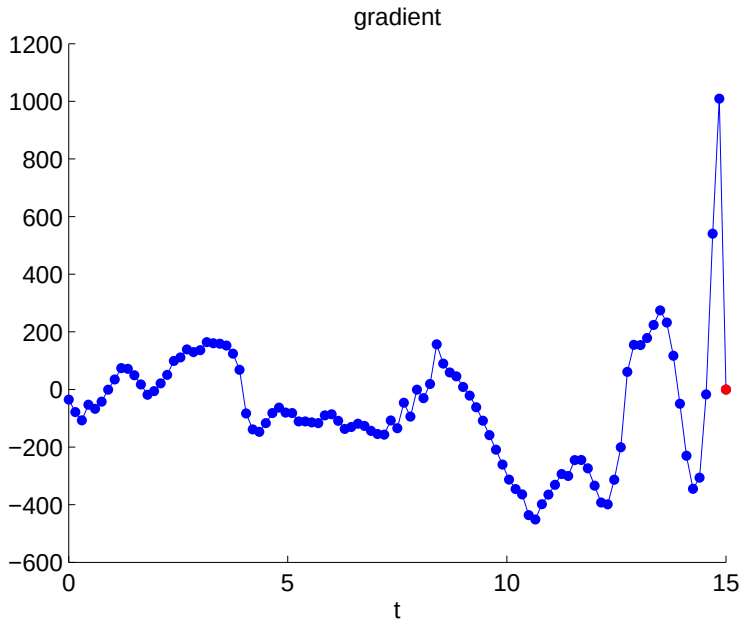


First Optimization Results: Model #1 (BFGS$_{rst=5}$ + BB)

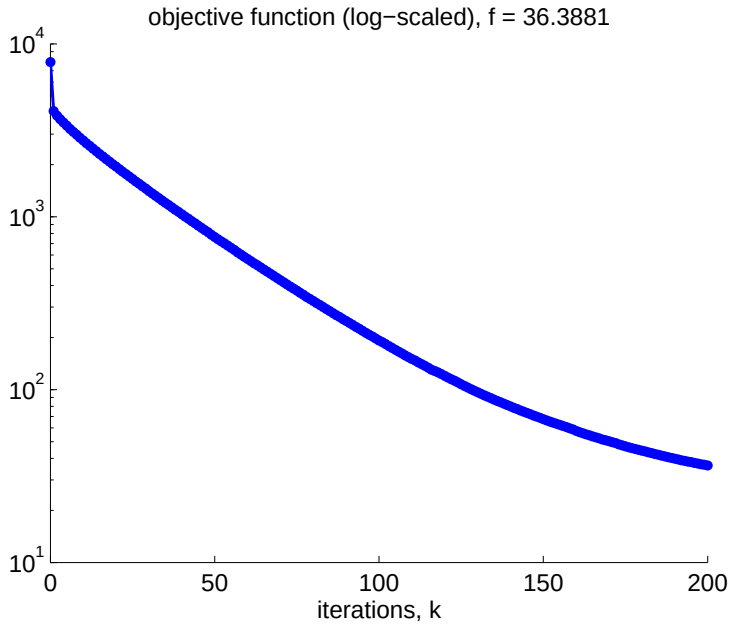


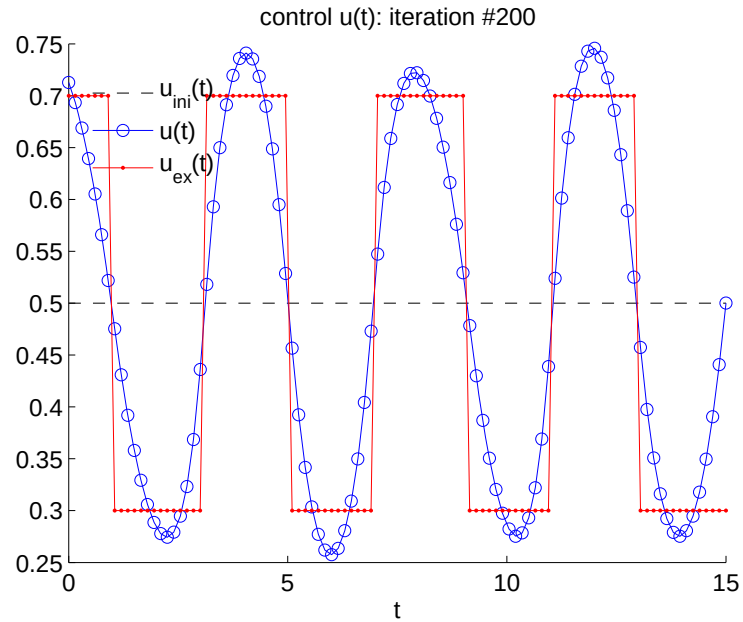


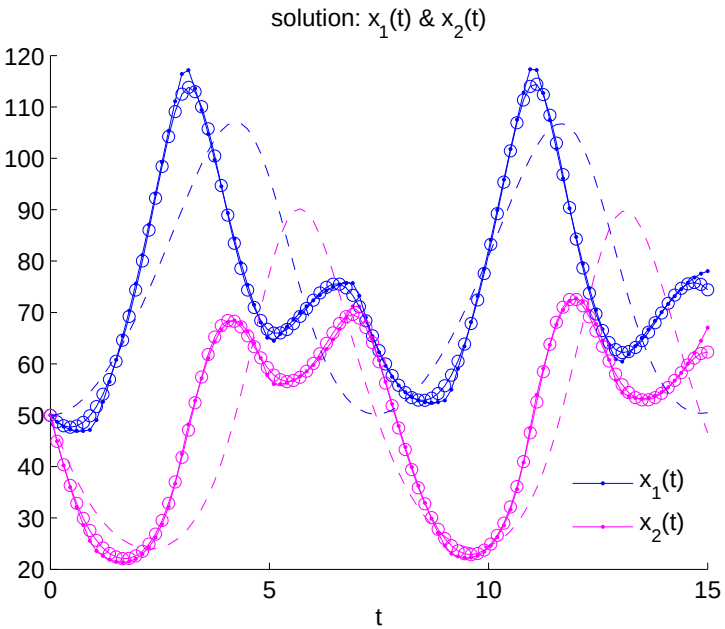


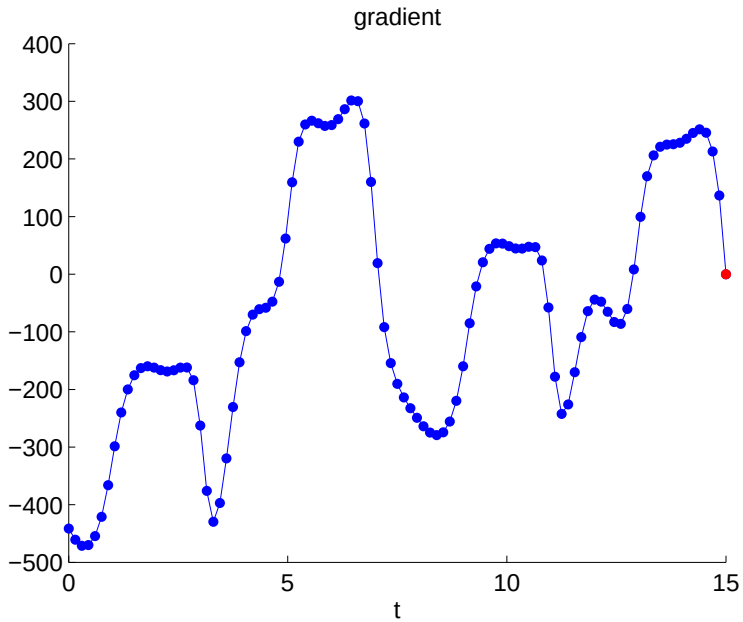


First Optimization Results: Model #3 (SD + GS)

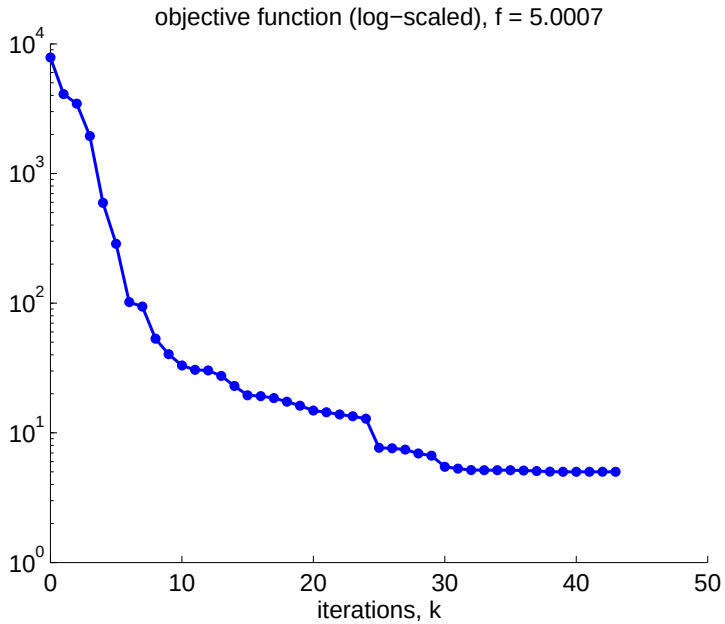


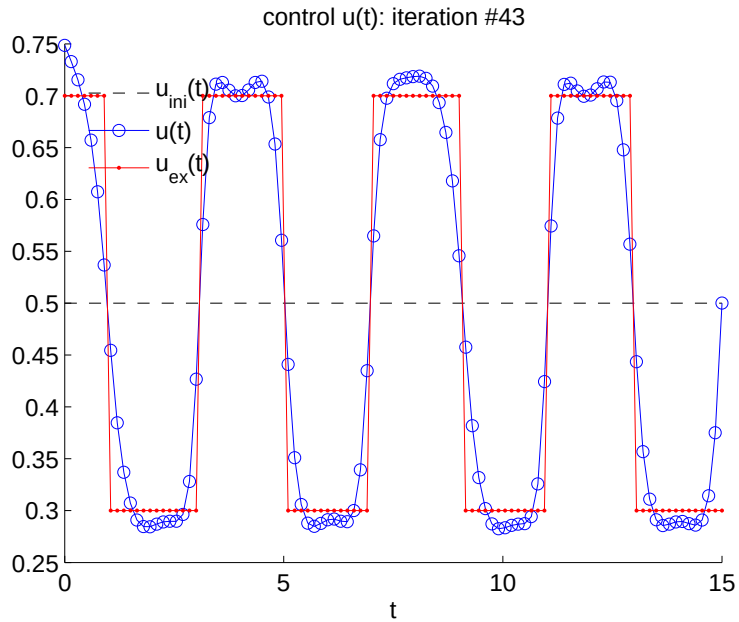


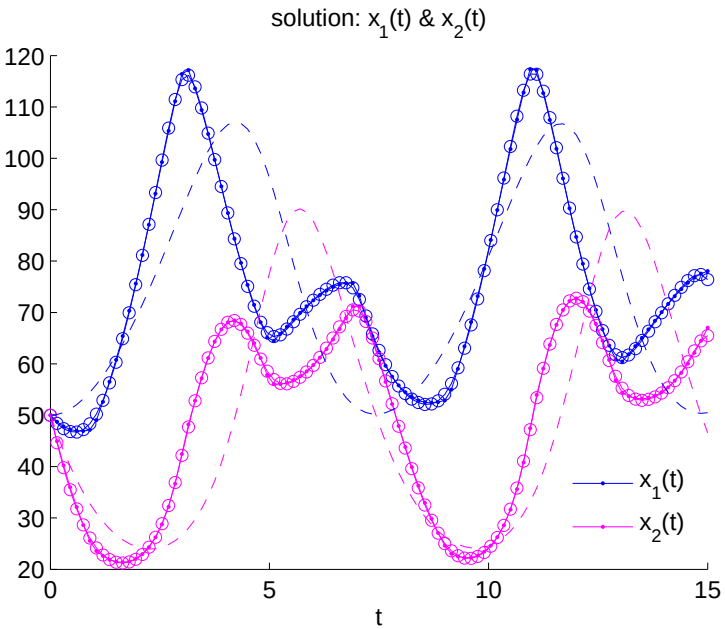


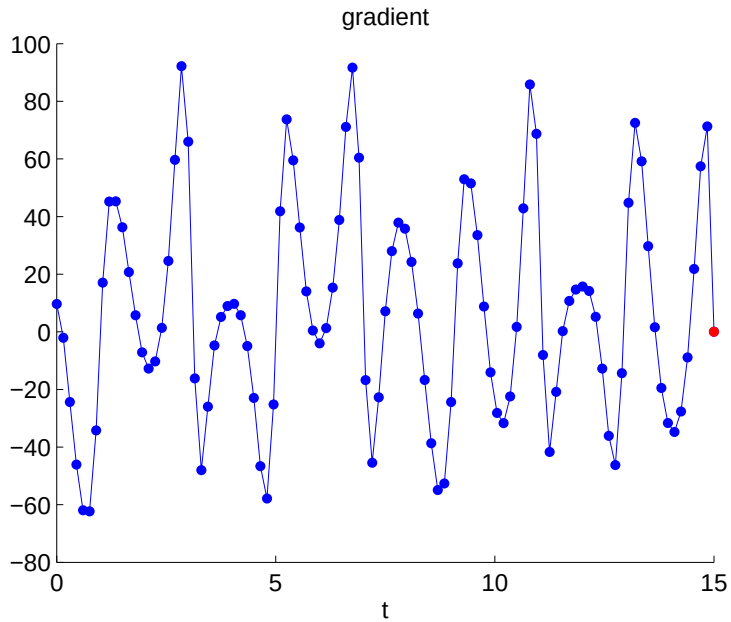


First Optimization Results: Model #3 (BFGS$_{rst=5}$ + BB)









BACKUP SLIDES

B1: Termination Conditions

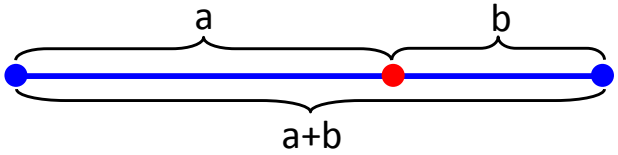
Based on

1. sufficient changes in the solution:
   - absolute decrease $\left\|\mathbf{u}^k - \mathbf{u}^{k-1}\right\| < \epsilon$
   - relative decrease $\frac{\left\|\mathbf{u}^k - \mathbf{u}^{k-1}\right\|}{\|\mathbf{u}^{k-1}\|} < \epsilon$
2. sufficient changes in the objective:
   - absolute decrease $\left|\mathcal{J}(\mathbf{u}^k) - \mathcal{J}(\mathbf{u}^{k-1})\right| < \epsilon$
   - relative decrease $\left|\frac{\mathcal{J}(\mathbf{u}^k) - \mathcal{J}(\mathbf{u}^{k-1})}{\mathcal{J}(\mathbf{u}^{k-1})}\right| < \epsilon$ (★)
3. computational efforts:
   - max number of optimization iterations K: $k < K$ (★)
   - max number of objective evaluations
   - limit on elapsed computational time T: $t < T$

Q: (★) are recommended options (problem dependent). Why?

B2: Golden Section (GS) Search Method

Q: Bisection requires $n = 2k$ function $f(x)$ evaluation. Could we reduce n?

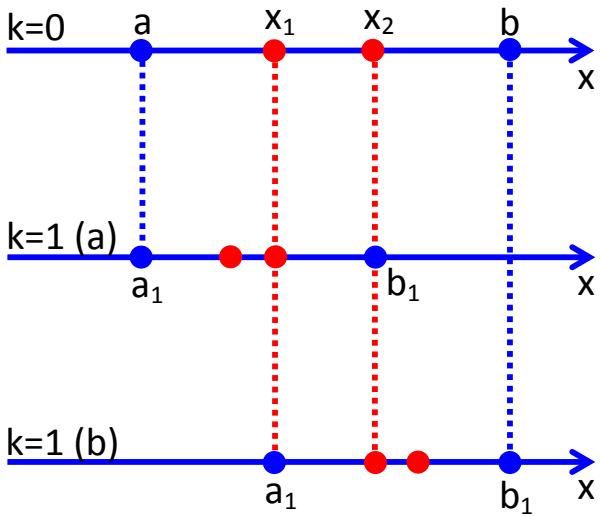


Golden section (ratio):

$$\frac{a+b}{a} = \frac{a}{b} = \frac{1+\sqrt{5}}{2} \approx 1.6180\ldots$$

Property: If X_1 is a GS for AB & X_2 is a GS for AB
$\Rightarrow$ X_1 is a GS for AX_2 & X_2 is a GS for X_1B

Finally: introduce $\alpha = \dfrac{\sqrt{5}-1}{2}$, then $x_1 = \alpha x_A + (1-\alpha)x_B, \quad x_2 = (1-\alpha)x_A + \alpha x_B$



Parameter: tolerance $\epsilon > 0$ to terminate iterations

Method:

1. For initial interval $I_0 = [a, b]$ compute:
   $x_1 = \alpha a + (1-\alpha)b, \qquad x_2 = (1-\alpha)a + \alpha b$
2. Compare $f(x_1)$ and $f(x_2)$, if
   (a) $f(x_1) \le f(x_2) \quad \Rightarrow \quad a_1 = a,\ b_1 = x_2,\ \bar{x}_1 = x_1$
   (b) $f(x_1) > f(x_2) \quad \Rightarrow \quad a_1 = x_1,\ b_1 = b,\ \bar{x}_1 = x_2$

B2: Golden Section (GS) Search Method (cont'd)

3. Get a new interval $I_1 = [a_1, b_1]$ of length $d_1 = b_1 - a_1 = \alpha(b-a)$
4. For $k = 2$ use $\bar{x}_1$ as a new value x_1 or x_2
5. Perform the same process iteratively for kth step, if
   - $f(x_{2k-1}) \le f(x_{2k}) \quad \Rightarrow \quad f(x_2) = f(\bar{x}_{k-1})$ known $\quad \Rightarrow \quad$ just compute $f(x_1)$
   - $f(x_{2k-1}) > f(x_{2k}) \quad \Rightarrow \quad f(x_1) = f(\bar{x}_{k-1})$ known $\quad \Rightarrow \quad$ just compute $f(x_2)$
6. Get a new interval $I_k = [a_k, b_k]$ of length $d_k = b_k - a_k = \alpha^k(b-a)$
7. Terminate if $d_k = b_k - a_k < \epsilon$:
   - number of iterations/f-evaluations $k = n > \log_{\frac{1}{\alpha}} \dfrac{b-a}{\epsilon}, \quad \alpha = \frac{\sqrt{5}-1}{2} \approx 0.62$
8. Approximate x^*, e.g. $x^* = \bar{x}_k$ with error $\sim \alpha^k(b-a) = \left(\frac{\sqrt{5}-1}{2}\right)^k (b-a)$

Compare the method error for (B)isection and GS search (same # of f-evaluations n):

$$\frac{e_{\mathrm{B}}}{e_{\mathrm{GS}}} = \frac{2^{-\frac{n}{2}}(b-a)}{\left(\frac{\sqrt{5}-1}{2}\right)^n (b-a)} = \left(\frac{\sqrt{2}}{\sqrt{5}-1}\right)^n \approx (1.144\ldots)^n, \qquad \left(\frac{\sqrt{2}}{\sqrt{5}-1}\right)^{10} \approx 3.84$$

B3: Computational Analysis for Rate of Convergence

$$\frac{\|\mathbf{e}^{k+1}\|}{\|\mathbf{e}^{k}\|^{r}} = \frac{\|\mathbf{u}^{k+1}-\mathbf{u}^{*}\|}{\|\mathbf{u}^{k}-\mathbf{u}^{*}\|^{r}} = C, \quad C<\infty,$$

for 1D case:

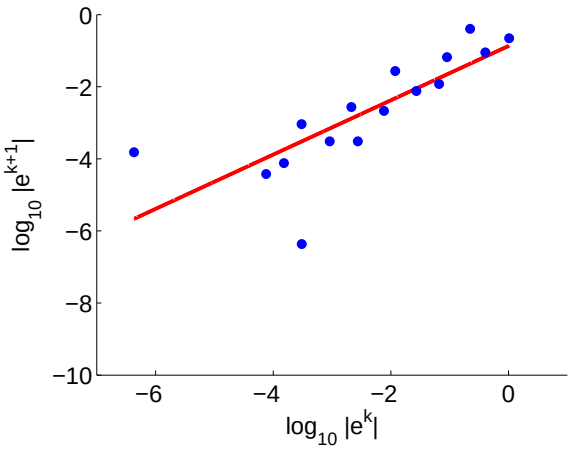
$$|e^{k+1}| = C|e^{k}|^{r} \quad \Rightarrow \quad \log_{10}|e^{k+1}| = \log_{10} C + r\cdot\log_{10}|e^{k}|.$$

MATLAB's `polyfit` function to approximate $b=\log_{10} C$ and r as coefficients in

$$y = b + rx, \qquad x=\log_{10}|e^{k}|, \quad y=\log_{10}|e^{k+1}|.$$

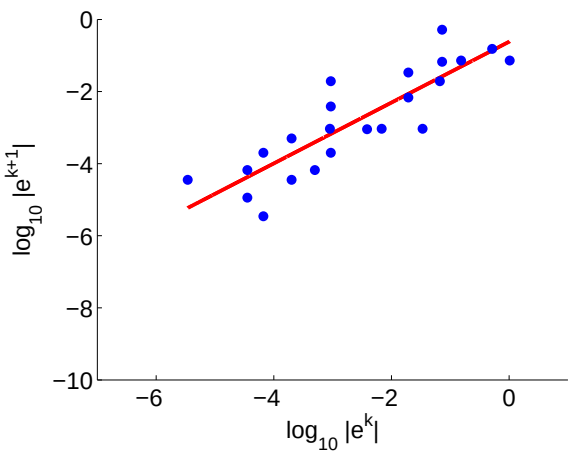
Example: comparing performance/convergence rate r for some model

bisection



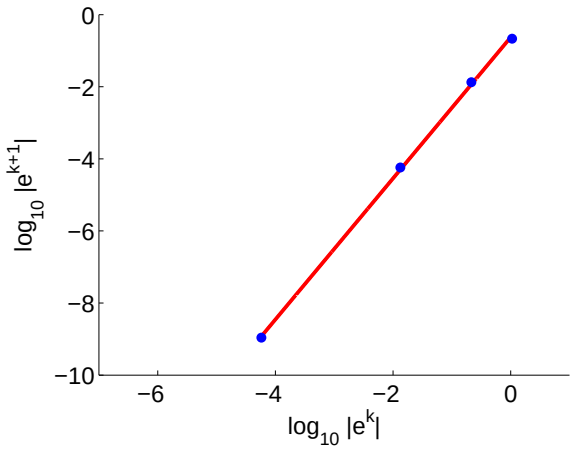
$r = 0.7533,\ C = 0.1348$

golden section



$r = 0.8441,\ C = 0.2404$

Newton



$r = 1.9573,\ C = 0.2347$

B4: Optimization Algorithms – Overview

Algorithms	Examples	Strengths	Limitations
Classical Methods	Gradient-based methods, Line Search, Pattern Searches, etc.	Optimality guaranteed (KKT condition)	Global optimum guaranteed only in convex cases
Evolutionary Algorithms	PSO, GA, Differential Evolution, etc.	Can be customized and easily adapted	Optimality guaranteed in limited cases (if any)
Global Search Optimizers	Branch & Bound, Cutting Plane, etc.	Can find global optimum of nonconvex problems	Can be computationally intractable
Hybrid Methods	PSO-MADS, rGA-SQP, etc.	Combine strengths of different algorithms	Heuristically done through trial and error