Topics in Numerical
and Computational Mathematics

Computational Optimization:

Success in Practice

Chapter 3: Generalized Optimization Framework

VladislavBukshtynov@yahoo.com

CRC Press
https://www.crcpress.com/

Example 1.3 (revisited): Least-Squares Data Fitting

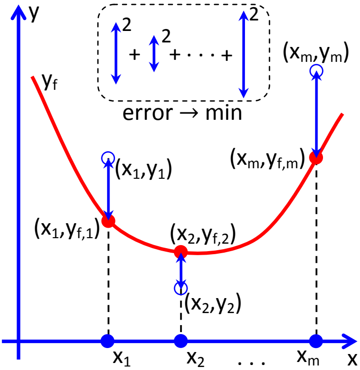
Input Data: m data points

$$(x_i, y_i), \quad i = 1, \ldots, m$$

Equation to Model Fitting:

$$y_f(x) = a_1 + a_2 x + a_3 x^2,$$

where a_1, a_2, a_3 are parameters to identify while pursuing the best data fit in the "least-squares" sense



General Approach: consider constrained optimization problem

$$\min_{\mathbf{a}\in\mathbb{R}^3} \sum_{i=1}^{m} (y_i - y_{f,i})^2$$

$$\text{s.t. } y_{f,i} = a_1 + a_2 x_i + a_3 x_i^2, \quad i = 1, \ldots, m$$

Example 1.3 (revisited): Least-Squares Data Fitting (cont'd)

Computational Approach: consider residual vector for m "pieces" of data

$$\mathbf{r} = \mathbf{y} - A\,\mathbf{a} = \begin{bmatrix} y_1 - (a_1 + a_2 x_1 + a_3 x_1^2) \\ y_2 - (a_1 + a_2 x_2 + a_3 x_2^2) \\ \dots \\ y_m - (a_1 + a_2 x_m + a_3 x_m^2) \end{bmatrix}, \quad A = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \dots & \dots & \dots \\ 1 & x_m & x_m^2 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_m \end{bmatrix}$$

and solve the problem in the form of unconstrained optimization problem

$$\min_{\mathbf{a} \in \mathbb{R}^3} f(\mathbf{a})$$

with objective function

$$f(\mathbf{a}) = \|\mathbf{r}\|^2 = r_1^2 + r_2^2 + \dots + r_m^2 = \sum_{i=1}^{m} \Big(y_i - (a_1 + a_2 x_i + a_3 x_i^2) \Big)^2,$$

where $\mathbf{a} = [a_1\; a_2\; a_3]^T$ is a control vector.

Case 1: $m = 3,\; y_1 \neq y_2 \neq y_3$ – unique solution could be found exactly (see Example 1.2)

Case 2: $m = 1, 2$ – infinitely many solutions

Case 3: $m > 3$ – uniqueness of the solution depends on data

Parameter Identification for Least-Squares Data Fitting

Objective function:

$$f(\mathbf{a}) = \sum_{i=1}^{m} \Big(y_i - (a_1 + a_2 x_i + a_3 x_i^2) \Big)^2$$

Gradient of the objective function w.r.t. control vector $\mathbf{a}$

$$\frac{\partial f}{\partial \mathbf{a}} = \nabla_{\mathbf{a}} f(\mathbf{a}) = \begin{bmatrix} \frac{\partial f}{\partial a_1} \\ \frac{\partial f}{\partial a_2} \\ \frac{\partial f}{\partial a_3} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{m} 2 \left(y_i - (a_1 + a_2 x_i + a_3 x_i^2) \right) \cdot (-1) \\ \sum_{i=1}^{m} 2 \left(y_i - (a_1 + a_2 x_i + a_3 x_i^2) \right) \cdot (-x_i) \\ \sum_{i=1}^{m} 2 \left(y_i - (a_1 + a_2 x_i + a_3 x_i^2) \right) \cdot (-x_i^2) \end{bmatrix}$$

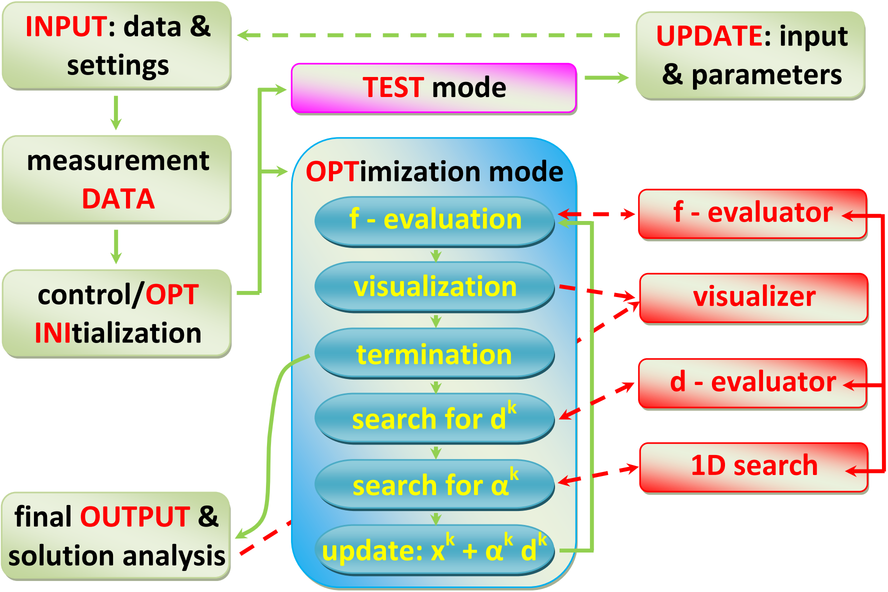
Find optimal solution $\mathbf{a}^*$ by using

- gradient-based (steepest descent) iterative approach

$$\mathbf{a}^{k+1} = \mathbf{a}^k + \alpha^k \cdot \mathbf{d}^k, \qquad \mathbf{d}^k = -\nabla_{\mathbf{a}} f(\mathbf{a}^k)$$

- optimal step size α^k computed by one of the discussed 1D minimization methods
- termination $\left| \frac{f(\mathbf{a}^{k+1}) - f(\mathbf{a}^k)}{f(\mathbf{a}^k)} \right| < \epsilon$ (relative decrease of objective)

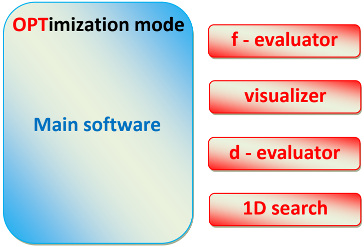
Computational Elements of the Generalized Optimization Framework



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



f-evaluator:

- to evaluate objective function(s) $f(\mathbf{x})$
- 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 *f*-evaluator

Choice of Proper Software (cont'd)

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

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

1D search:

- to find optimal step size α
- depends on the nature of the problem (differentiability, convexity, constraints, etc.)
- may require to communicate effectively with f-evaluator and d-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

Examples of core software platforms:

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

Example 1.3: MATLAB-based Optimization Framework

```
% Chapter_3_data_fit_by_gradient.m

close all; clc; clear;

params;                                          % setting INPUT parameters

data = load(dataFile);                           % loading DATA

initialize;                                      % INItialization

while(k < kMax+1) % termination condition #2     % main OPTimization loop

  obj = [obj f(a, data)];                        % f-evaluation

  visualize;                                     % visualization

  if k > 0        % termination condition #1     % checking optimality (by tolerance)
    err = abs(obj(end-1)-obj(end))/obj(end-1);
    if (err < epsilon)
      break;
    end
  end

  d = -grad(a,data);                             % search for d: computing gradient

  alpha = alphaConst;                            % search for alpha

  a = a + alpha*d;                               % update for controls

  k = k + 1;                                     % iteration counter increment

end
```

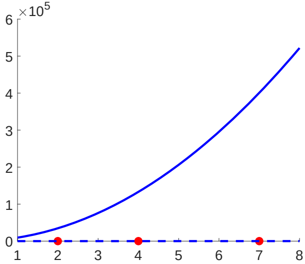
Example 1.3: Choosing and Adjusting Optimization Algorithms

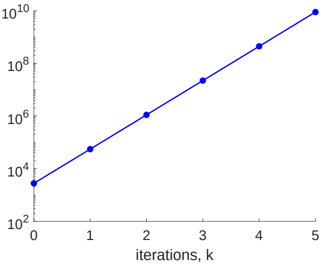
Computational elements: `Chapter_3_data_fit_by_gradient.m`

main OPT-part:	written manually	[MATLAB]
f-evaluator:	m-function, analytically defined function $f(\mathbf{a})$	[MATLAB]
d-evaluator:	m-function, analytically defined gradient $\nabla_{\mathbf{a}} f(\mathbf{a})$	[MATLAB]
1D search for α:	constant value, $\alpha = \mathrm{const}$	—
visualizer:	plain m-code	[MATLAB]

Parameters:

- initial run: set $\alpha = 10^{-3}$ and $\mathbf{a}^0 = [1\ 1\ 1]^T$ (dashed blue line)
- termination #1: $\left| \frac{f(\mathbf{a}^{k+1}) - f(\mathbf{a}^k)}{f(\mathbf{a}^k)} \right| < \epsilon = 10^{-6}$
- termination #2: $k_{max} = 5$

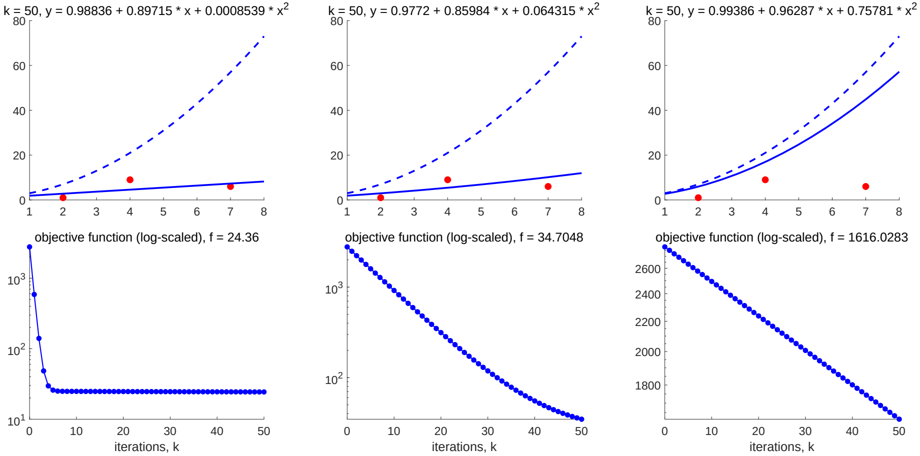




Q: Why does it diverge?

Example 1.3: Choosing and Adjusting Optimization Algorithms (cont'd)

Adjusting algorithm: change step size to $\alpha = 10^{-4}, 10^{-5}, 10^{-6}$ & $k_{max} = 50$

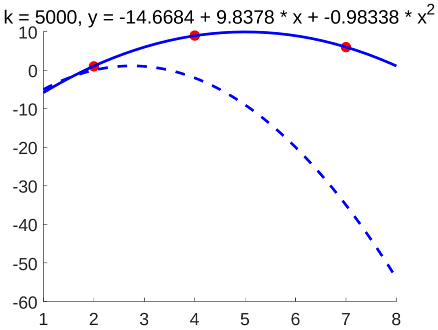


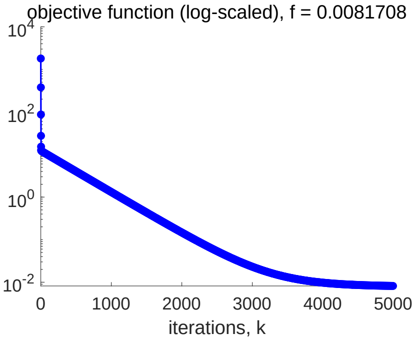
$\alpha = 10^{-4}$ | $\alpha = 10^{-5}$ | $\alpha = 10^{-6}$

Q: Now it converges, what about performance?

Adjusting algorithm:

- Fix step size to $\alpha = 10^{-4}$ and $k_{max} = 5000$
- Make initial guess closer to $\mathbf{a}^* = [-15\ 10\ \ -1]^T$, e.g. $\mathbf{a}^0 = [-14\ 11\ \ -2]^T$
- Explore the results (shown below)





Q: What could be done to check and increase further the performance?

Visualization and Analysis of Obtained Solutions

Data to be visualized (depending on problem):

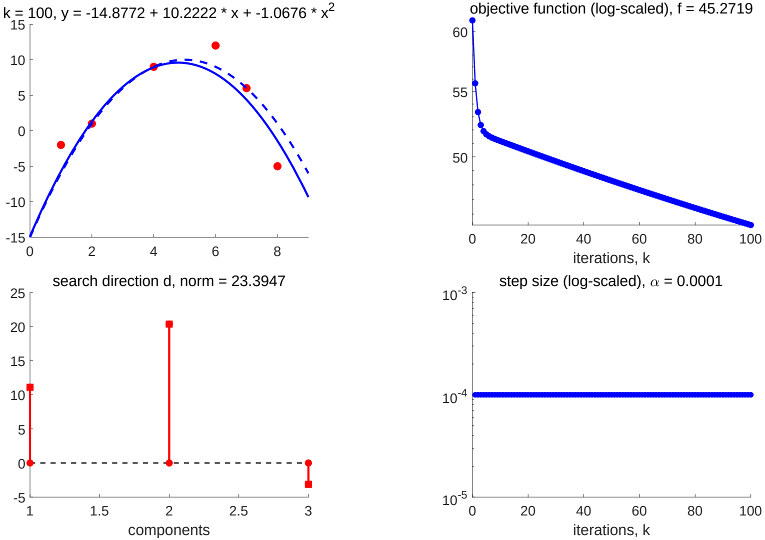
1. Optimization progress via objective function
   - measurement data (if compared with the modeled data)
   - separate parts of objective (how closely data is fitted)
   - entire objective vs. iteration number k (to check monotonicity)
2. Optimization progress via optimization/control variables
   - "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 (optimal step size, weighting coefficients, etc.)
   - controlling other techniques (regularization, preconditioning, etc.)

Before your big project starts, think how to:

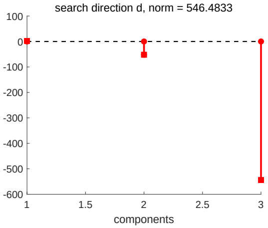
- save intermediate/final data instead of graphical images
- keep data in easily convertible formats, e.g. `dat` or `txt` files
- convert your data into high resolution images or send to external software

Visualization and Analysis of Obtained Solutions: Example 1.3 (modified)

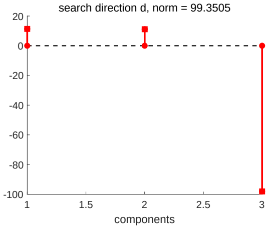
Modification: more data (6 points), initial guess $\mathbf{a}^0$ set to exact solution of original Example 1.3 (with 3 points).



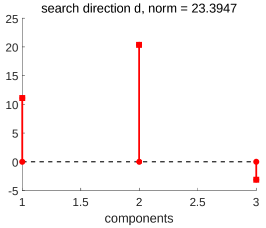
Analysis of Gradient Structure: Example 1.3 (modified)



$k = 1$



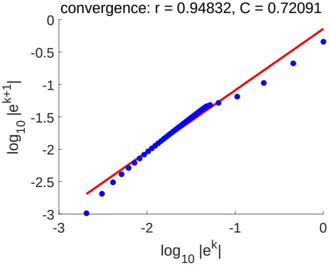
$k = 5$



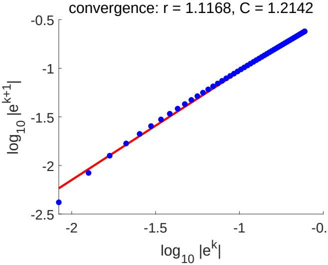
$k = 100$

- visualized **d**-components: pattern to update controls
- actual updates: **d**-components scaled by step size α
- convergence: diminishing range of **d**-component amplitudes
- termination condition: norm $\|\mathbf{d}^k\| = \|\nabla_{\mathbf{a}} f^k(\mathbf{a})\| = 0$
  (also 1-order optimality condition, Chapter 5)

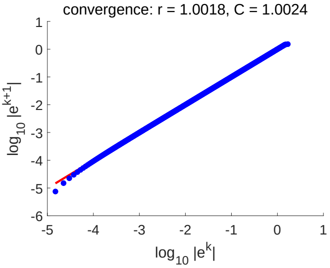
Analysis of Computational Convergence: Example 1.3 (original & modified)



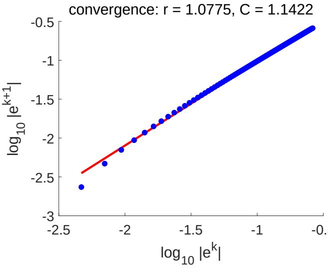
$\alpha = 10^{-4}$, $\mathbf{a}^0$ far from $\mathbf{a}^*$



$\alpha = 10^{-6}$, $\mathbf{a}^0$ far from $\mathbf{a}^*$



$\alpha = 10^{-4}$, $\mathbf{a}^0$ close to $\mathbf{a}^*$



$\alpha = 10^{-4}$, $m = 6$ case

Analysis of Computational Convergence (cont'd)

- Review the concept applied to 1D optimization problems

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

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

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

- Now, optimization in 3D ($\mathbf{a} \in \mathbb{R}^3$): back to generalized form using $\|\cdot\|_2$ (Euclidean distance in $\mathbb{R}^n$) norm and $\mathbf{a}^* = \mathbf{a}^{last}$ concept

$$\lim_{k\to\infty} \frac{\|\mathbf{e}^{k+1}\|}{\|\mathbf{e}^k\|^r} = \lim_{k\to\infty} \frac{\|\mathbf{a}^{k+1} - \mathbf{a}^*\|_2}{\|\mathbf{a}^k - \mathbf{a}^*\|_2^r} = C, \quad C < \infty.$$

- Convergence: **linear** due to steepest-descent (cannot move it beyond its limits)
- Faster convergence: consider two options
  1. investing further in the optimal step size search, or
  2. changing the method itself (method's order).

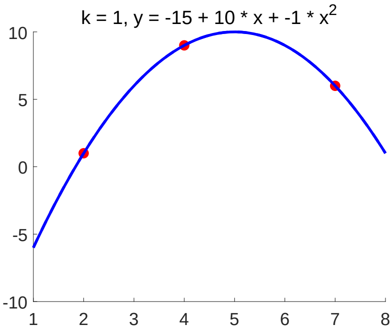
Q: What would be the best option for our current Example 1.3? For other problems?

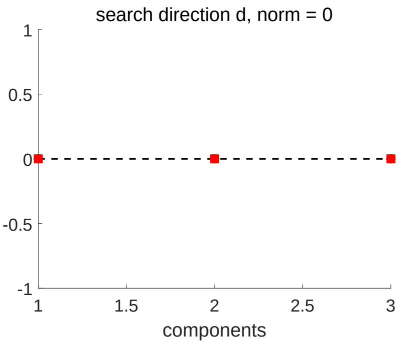
Testing and Dealing with Problems (Debugging)

f-evaluator

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

Test case #1: *f*-evaluator with $\mathbf{a}^0 = \mathbf{a}^*$





Testing and Dealing with Problems (Debugging, cont'd)

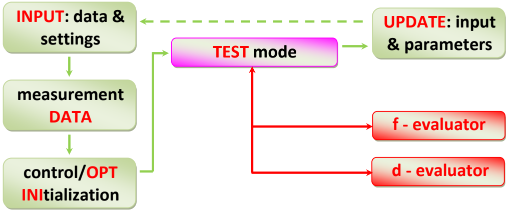
d-evaluator (problem- and method-dependent)

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

main OPT part

- Test every component separately: "change one part at a time"
- Test communication within the entire framework (variables, dimensions of vectors/matrices, names, solution files, etc.)
- Tuning Test: for the same problem, change one parameter/technique at a time (check sensitivity of performance to this particular change)
- Robustness Test: for fixed set of parameters/techniques run framework for the same problem varying initial data; then explore the results and repeat tuning (if necessary)
- Applicability Test: apply framework to problems at different levels of complexity (low, moderate, high)

TEST Mode for Gradient-based Framework



1D case implementation (by FD-1):

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

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

if Δx is finite (small) and $\epsilon \to 0$

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

$$\kappa(\epsilon) = \frac{f(\mathbf{x} + \epsilon\, \delta\mathbf{x}) - f(\mathbf{x})}{\epsilon\, \langle \boldsymbol{\nabla}_{\mathbf{x}} f(\mathbf{x}),\, \delta\mathbf{x} \rangle}, \quad \delta\mathbf{x} = [\Delta x_1\ \Delta x_2\ \ldots\ \Delta x_n]^T, \quad \epsilon \to 0$$

- "cheap test": requires 2 f-evaluations

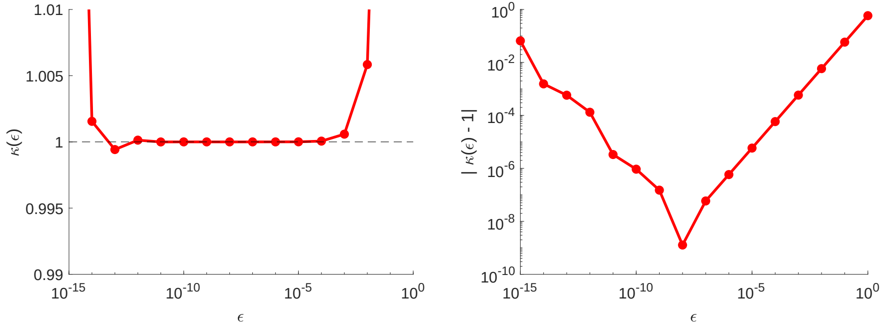
  for fixed $\delta\mathbf{x}$, e.g., $\delta\mathbf{x} = \mathbf{x}$, compute $\kappa(\epsilon)$ for a range of ϵ, e.g., $\epsilon = 10^{-12} \div 10^2$

- "expensive test": requires $n + 1$ f-evaluations

  for fixed ϵ, e.g., $\epsilon = 10^{-6}$, perform kappa-test changing $\delta\mathbf{x}$:
  $[x_1\ 0\ 0\ \ldots\ 0]^T$, $[0\ x_2\ 0\ \ldots\ 0]^T$, $\ldots$, $[0\ 0\ 0\ \ldots\ x_n]^T$
  to check sensitivity for every component of $\mathbf{x}$

TEST Mode for Gradient-based Framework (cont'd)

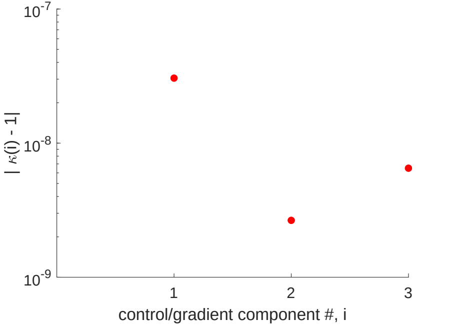
Example 1.3: "cheap test" for gradient (`Chapter_3_data_fit_by_gradient_test.m`)



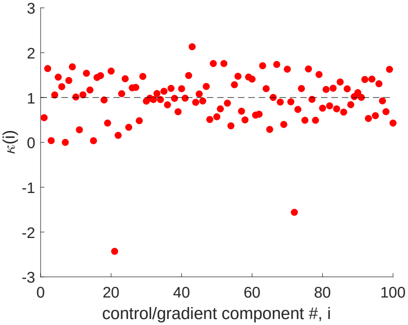
- correctness of gradient: range of ϵ spans 9-10 orders of magnitude
- quantity $\log_{10}|\kappa(\epsilon) - 1|$ shows how many significant digits of accuracy are captured in gradient evaluation
- well-known effects: $\kappa(\epsilon)$ deviates from the unity:
  - for very small values of ϵ due to subtractive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

TEST Mode for Gradient-based Framework (cont'd)

Example 1.3: "expensive test"
`Chapter_3_data_fit_by_gradient_test.m`



Another Example: typical "expensive test" for problem with $\mathbf{x} \in \mathbb{R}^n$, $n = 100$



- 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 (to avoid propagation)

Example 1.3: Improving Performance – Step Size α

Computational algorithm: (updated) `Chapter_3_data_fit_by_gradient_ver_2.m`

main OPT-part:	written manually	[MATLAB]
f-evaluator:	m-function, analytically defined function $f(\mathbf{a})$	[MATLAB]
d-evaluator:	m-function, analytically defined gradient $\nabla_{\mathbf{a}} f(\mathbf{a})$	[MATLAB]
1D search for α:	plain m-code for Golden Section Search	[MATLAB]
visualizer:	plain m-code	[MATLAB]

Implementation of Golden Section Search (line minimization search):

- find optimal step size α^k at every optimization iteration k by solving 1D minimization problem

$$\alpha^k = \underset{\alpha > 0}{\operatorname{argmin}} \; f\left(\mathbf{a}^k + \alpha \cdot \mathbf{d}^k\right)$$

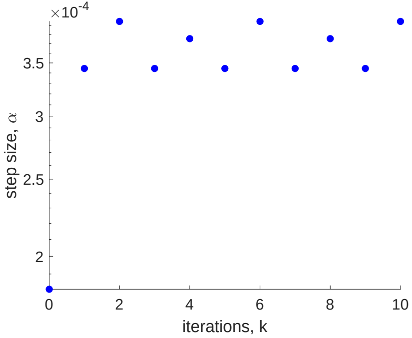
- could also use: Bisection, Brute-Force, Monte Carlo methods, etc.

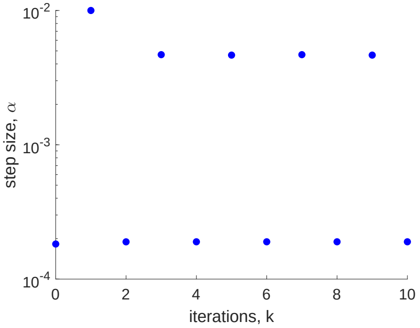
Parameters for Golden Section Search:

- search interval $[a, b]$: $a = 0$, $b = 0.01$
- termination: $\epsilon_\alpha = 10^{-2}, 10^{-3}$ (why diverging?), 10^{-4} (next slide figure), 10^{-6} (next two slides figures)

Example 1.3: Improving Performance – Step Size α (cont'd)

- step size α^k via GS: $\epsilon_\alpha = 10^{-4}$
- step size α^k via GS: $\epsilon_\alpha = 10^{-6}$



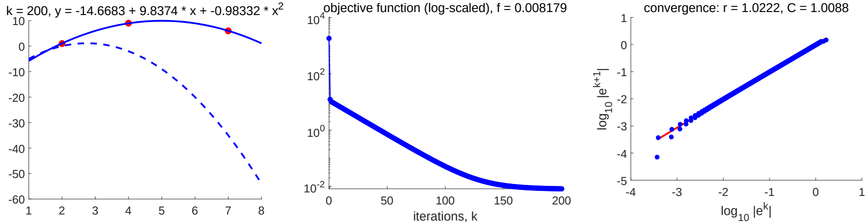


Tuning-up GS method:

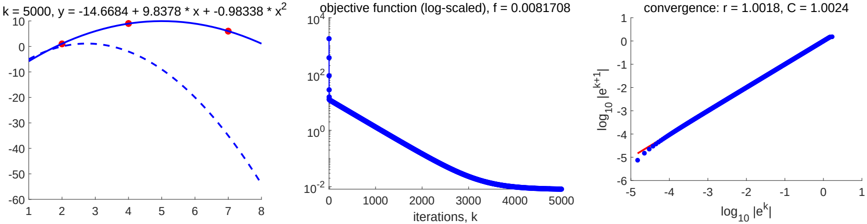
- search interval $[a, b]$: are bounds a and b appropriate?
- ϵ_α: best alignment with the gradient-based search
- $\epsilon_\alpha = 10^{-4}, 10^{-6}$: 11 vs. 21 f-evaluations (per kth iteration)
- $\alpha^k \in [0, 10^{-2}]$ vs. $\alpha^k = const = 10^{-4}$ (next slide)

Example 1.3: Comparing Performance – Step Size α

- step size α^k: Golden Section Search ($a = 0,\ b = 0.01,\ \epsilon_\alpha = 10^{-6}$)



- step size α^k: constant value $\alpha = 10^{-4}$ (see slides 11 & 15)



Q: How to improve further the performance of GS method? "Flexibility" for a and b?

Example 1.3: Improving Performance – Newton's Method

Computational algorithm: (updated) `Chapter_3_data_fit_by_gradient_ver_3.m`

main OPT-part:	written manually	[MATLAB]
f-evaluator:	m-function, analytically defined function $f(\mathbf{a})$	[MATLAB]
d-evaluator:	m-function, analytically defined $\nabla_{\mathbf{a}} f(\mathbf{a})$ & $\left[\nabla^2_{\mathbf{a}} f(\mathbf{a})\right]^{-1}$	[MATLAB]
1D search for α:	not required	—
visualizer:	plain m-code	[MATLAB]

Implementation of 2-order Newton's method for search direction:

- evaluate gradient $\nabla_{\mathbf{a}} f(\mathbf{a}^k)$ (slide 4) and Hessian $\nabla^2_{\mathbf{a}} f(\mathbf{a}^k)$

$$\nabla^2_{\mathbf{a}} f(\mathbf{a}) = \begin{bmatrix} \frac{\partial^2 f}{\partial a_1\, \partial a_1} & \frac{\partial^2 f}{\partial a_1\, \partial a_2} & \frac{\partial^2 f}{\partial a_1\, \partial a_3} \\ \frac{\partial^2 f}{\partial a_2\, \partial a_1} & \frac{\partial^2 f}{\partial a_2\, \partial a_2} & \frac{\partial^2 f}{\partial a_2\, \partial a_3} \\ \frac{\partial^2 f}{\partial a_3\, \partial a_1} & \frac{\partial^2 f}{\partial a_3\, \partial a_2} & \frac{\partial^2 f}{\partial a_3\, \partial a_3} \end{bmatrix} = 2 \cdot \begin{bmatrix} m & \sum_{i=1}^m x_i & \sum_{i=1}^m x_i^2 \\ \sum_{i=1}^m x_i & \sum_{i=1}^m x_i^2 & \sum_{i=1}^m x_i^3 \\ \sum_{i=1}^m x_i^2 & \sum_{i=1}^m x_i^3 & \sum_{i=1}^m x_i^4 \end{bmatrix}$$

- find search direction $\mathbf{d}^k$ at every optimization iteration k by

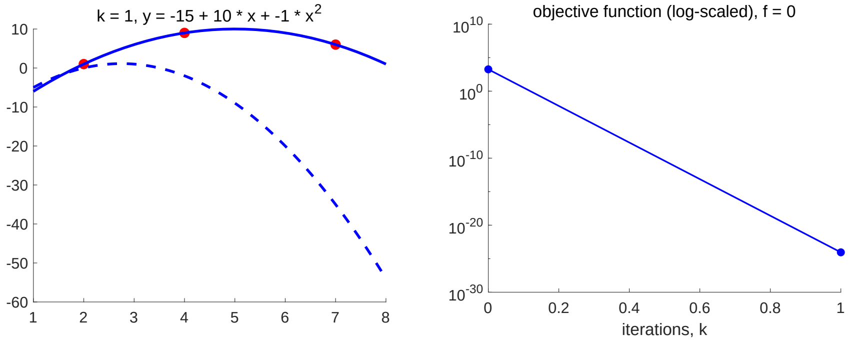
$$\mathbf{d}^k = -\left[\nabla^2_{\mathbf{a}} f(\mathbf{a}^k)\right]^{-1} \nabla f(\mathbf{a}^k)$$

- in general method works well with step size $\alpha^k = 1$

Example 1.3: Improving Performance – Newton's Method (cont'd)

Parameters for Newton's method (main OPT-part):

- initial run: set "no update" for α ($\alpha^k = 1$) and $\mathbf{a}^0 = [-14\ 11\ \ -2]^T$
- termination #1: $\left|\frac{f(\mathbf{a}^{k+1})-f(\mathbf{a}^k)}{f(\mathbf{a}^k)}\right| < \epsilon_1 = 10^{-6}$ [fails if $f(\mathbf{a}^{k+1}) = f(\mathbf{a}^k) = 0$]
- termination #2: $\frac{\|\mathbf{a}^{k+1}-\mathbf{a}^k\|_2}{\|\mathbf{a}^k\|_2} < \epsilon_2 = 10^{-6}$
- termination #3: $k_{max} = 100$

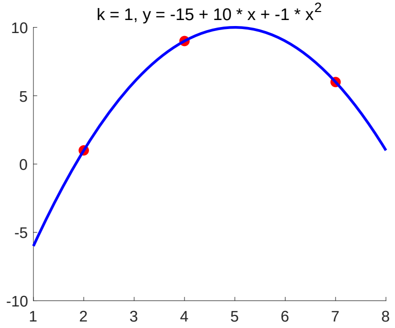


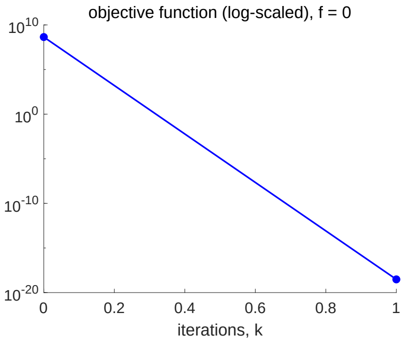
Q: Optimal solution $\mathbf{a}^* = [-15\ 10\ \ -1]^T$ is found in one iteration. Why?

Example 1.3: Improving Performance – Newton's Method (cont'd)

Exploring Newton's method:

- Update α using GS method with $a = 0$, $b = 10$, $\epsilon_\alpha = 10^{-6}$. Check that α returns optimal value close to 1 (e.g., 1.000000052835619) for the same settings.
- Check convergence to $\mathbf{a}^*$ from $\mathbf{a}^0 = [-100\ 1000\ 250]^T$.



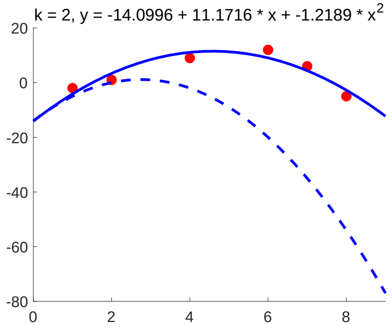


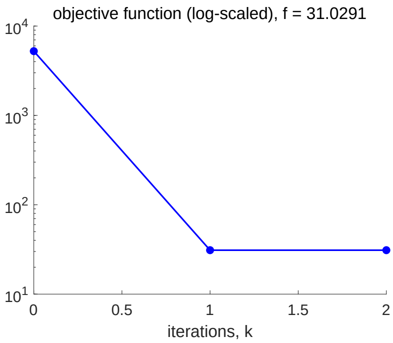
Q: Explain convergence in 1 iteration.

Example 1.3: Improving Performance – Newton's Method (cont'd)

Exploring Newton's method:

- Check convergence in 1–2 iterations for different data, e.g., $m = 6$ (`data_6pt.dat`).
- Make general conclusion on Newton's method applied to quadratic problems.



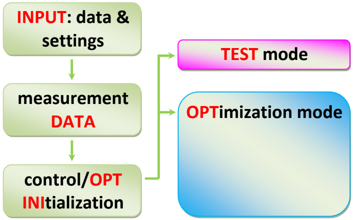


Q: How will convergence change if gradient $\nabla_{\mathbf{a}} f(\mathbf{a}^k)$ and Hessian $\nabla^2_{\mathbf{a}} f(\mathbf{a}^k)$ are computed for quadratic/non-quadratic problems using any FD approximations?

Generalized Optimization Framework: Communication

Framework "parameterization"

- modes: OPT, TEST
- methods: SD, NEWTON, . . . , future and your own methods
- α-search: const, GS, . . . , other algorithms
- other parts: main solver, regularization, etc.




```
% Chapter_3_data_fit_by_gradient_ver_final.m

close all; clc; clear; tic;

params_ver_final;                               % setting INPUT parameters

data = load(dataFile);                          % loading DATA

initialize_ver_final;                           % INItialization

if strcmp(mode,'OPT')                           % choosing mode OPT/TEST
  mode_OPT;                                     % based on Chapter_3_data_fit_by_gradient_ver_3.m
elseif strcmp(mode,'TEST')
  mode_TEST;                                    % based on Chapter_3_data_fit_by_gradient_test.m
else
  disp(['error: Unknown mode ' mode ' is chosen!']); return;
end

% final output
fprintf(['We are fully done! CPU elapsed time = ' num2str(toc) 's\n\n']);
```

Homework for Chapter 3

- Run MATLAB code `Chapter_3_data_fit_by_gradient.m` to experiment with $m > 3$ (modified Example 1.3 using steepest descent & constant step size α) for different parameters α, k_{max}, and initial guess $\mathbf{a}^0$. Check the performance based on the analysis of the visualized solutions: solution curves, objective function, search direction (gradient structure), parameters for the computational convergence.

- Modify MATLAB code `Chapter_3_data_fit_by_gradient.m` to use any FD approximations of $\nabla_{\mathbf{a}} f(\mathbf{a}^k)$ for the SD method. For constant step size α, check the convergence and approximate convergence parameters r and C for both cases: analytically defined and FD-approximated gradients $\nabla_{\mathbf{a}} f(\mathbf{a}^k)$. Compare the results and make a conclusion.

- Modify MATLAB code `Chapter_3_data_fit_by_gradient_ver_2.m` and repeat the previous experiments (problem 2) now with optimal step size α chosen by using the GS method.

Homework for Chapter 3 (cont'd)

- Modify MATLAB code `Chapter_3_data_fit_by_gradient_ver_3.m` and apply Newton's method to check the convergence and approximate convergence parameters r and C for both cases: analytically defined and FD-approximated gradients $\nabla_{\mathbf{a}} f(\mathbf{a}^k)$ and Hessians $\nabla^2_{\mathbf{a}} f(\mathbf{a}^k)$. Compare the results and conclude on the convergence when using 1-order, 2-order, mixed-order (e.g., 2-order for gradient and 1-order for Hessian) approximations.

- Explore the structure of the upgraded MATLAB code `Chapter_3_data_fit_by_gradient_ver_final.m` to incorporate computations for FD-approximated gradients $\nabla_{\mathbf{a}} f(\mathbf{a}^k)$ and Hessians $\nabla^2_{\mathbf{a}} f(\mathbf{a}^k)$. Discuss the proper communication concept applied for using FD approximations throughout the entire framework.

- In `Chapter_3_data_fit_by_gradient_ver_final.m`, upgrade the procedure for finding optimal step size α^k by solving 1D minimization problem

$$\alpha^k = \operatorname*{argmin}_{\alpha > 0} f\left(\mathbf{a}^k + \alpha \cdot \mathbf{d}^k\right)$$

using the bisection, brute-force, and Monte Carlo methods.

Where to Read More for Chapter 3

- **Bukshtynov (2023):** Chapter 3
- **Press (2007):** Chapter 9 (Root Finding and Nonlinear Sets of Equations), Chapter 10 (Minimization or Maximization of Functions), Chapter 15 (Modeling of Data)

MATLAB codes for Chapter 3

- `Chapter_3_data_fit_by_gradient.m`
- `Chapter_3_data_fit_by_gradient_test.m`
- `Chapter_3_data_fit_by_gradient_ver_2.m`
- `Chapter_3_data_fit_by_gradient_ver_3.m`
- `Chapter_3_data_fit_by_gradient_ver_final.m`
- `params.m`
- `initialize.m`
- `visualize.m`
- `fn_eval_f.m`
- `fn_eval_grad.m`
- `fn_convergence_sol_norm.m`
- `data_main.dat`
- `data_6pt.dat`
- `kappa_test.m`
- `golden_section_search.m`
- `params_ver_2.m`
- `initialize_ver_2.m`
- `params_ver_3.m`
- `initialize_ver_3.m`
- `fn_eval_hess.m`
- `params_ver_final.m`
- `initialize_ver_final.m`
- `mode_OPT.m`
- `mode_TEST.m`