# Probabilistic Algorithms

Paul Cotofrei

information management institute

PA 2016

# Outline

# Stochastic Optimization

**A.** Random noise in input information (e.g., measurements with noise for $L(\mathbf{d})$ or $g(\mathbf{d})$):

$$y(\mathbf{d}) \equiv L(\mathbf{d}) + \varepsilon(\mathbf{d})$$

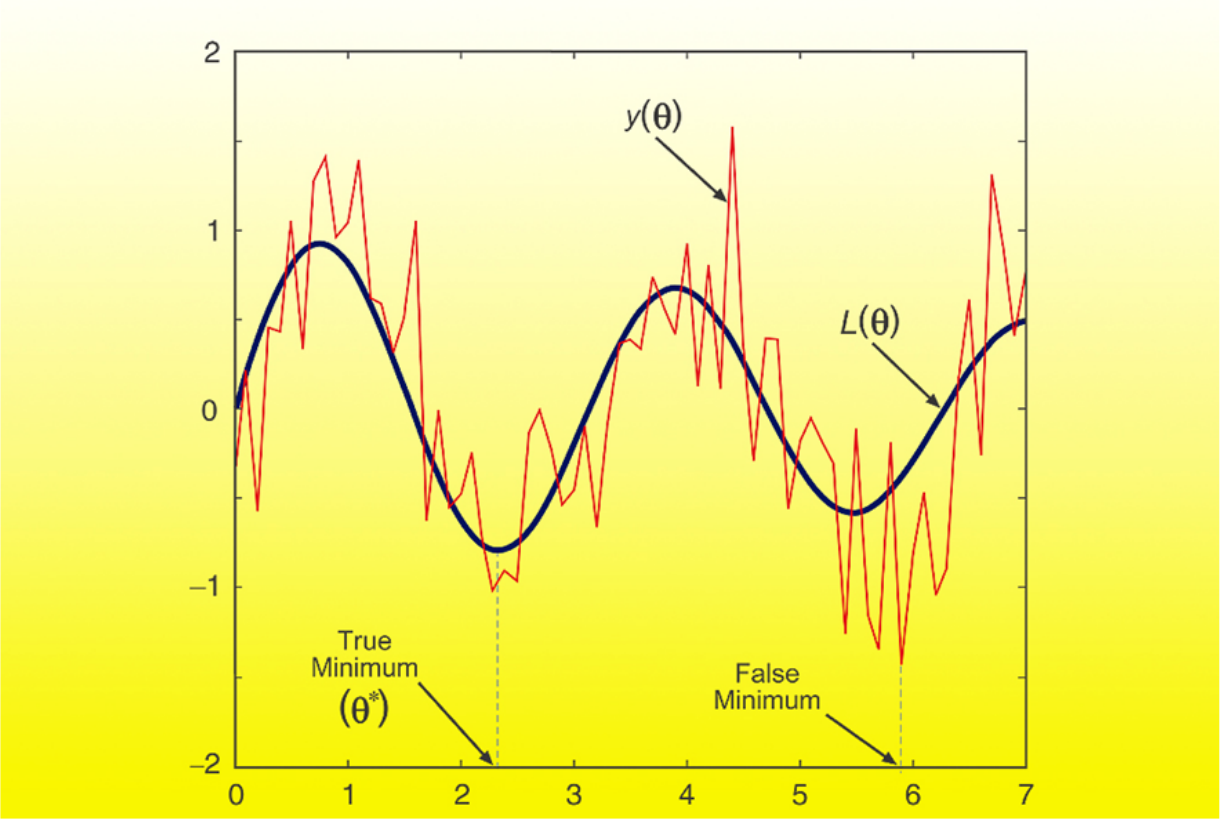
$$Y(\mathbf{d}) \equiv g(\mathbf{d}) + e(\mathbf{d}),$$

where $\varepsilon$ and $e$ represent the noise terms.
**B.** Injected randomness (Monte Carlo) in choice of algorithm iteration magnitude/direction

- Noise: independent, identically distributed (i.i.d.) measurement errors
- Injected randomness (B) in search magnitude/direction can offer benefits in efficiency and robustness
  - E.g., Capabilities for global (vs. local) optimization

# Effects of Noise on Simple Optimization Problem

- Noise-free loss function: $L(\mathbf{d}) = e^{-0.1}\sin(2\mathbf{d})$, with $\mathbf{d}^* = 3\pi/4$
- Noisy loss function: $y(\mathbf{d}) = L(\mathbf{d}) + \varepsilon$, where $\varepsilon \sim N(0, 0.5^2)$
- Estimate $\mathbf{d}^*$ using a grid with step 0.1: $[0, 0.1, ..., 6.9, 7]$



# Sources of noise

- Noise arises in almost any case where *physical system* measurements or *computer simulations* are used to approximate a steady-state criterion.
- Specific areas:
  - Real-time estimation and control problems where data are collected "on the fly" as a system is operating.
  - Problems where estimates of a criterion are formed by computer-based Monte Carlo sampling
  - Problems where large-scale simulations are run as estimates of actual system behavior

# Example: Tracking Problem

- Consider tracking problem where controller and/or system depends on design parameters **d** (e.g.: missile guidance, robot arm manipulation, attaining macroeconomic target values, etc.)
- Aim is to pick **d** to minimize mean-squared error (MSE):

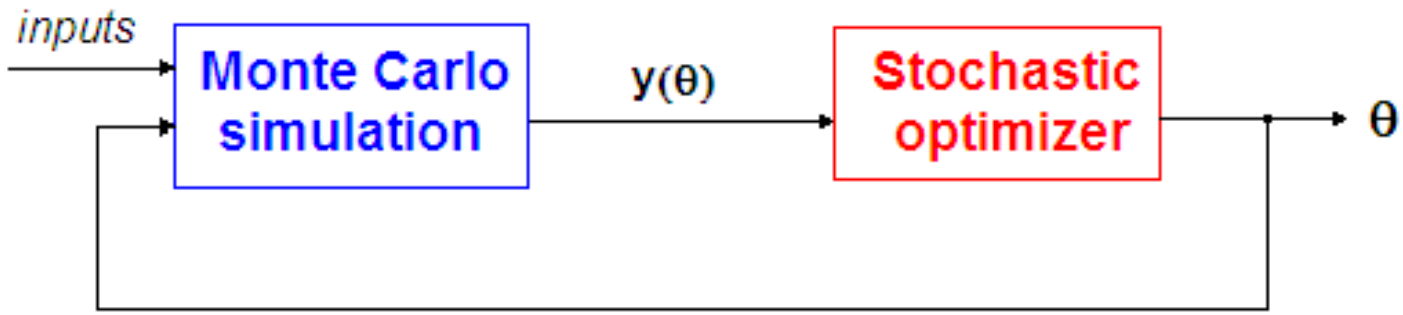
$$L(\mathbf{d}) = E\left(\|\text{current output - planned output}\|^2\right)$$

- In general for nonlinear and/or non-Gaussian systems, it's **not possible** to compute $L(\mathbf{d})$
- Get observed squared error $y(\mathbf{d}) \equiv \|\cdot\|^2$ by running system
- Note that $y(\mathbf{d}) \equiv \|\cdot\|^2 = L(\mathbf{d})$ + noise
  - Values of $y(\mathbf{d})$, not $L(\mathbf{d})$, used in optimization of **d**

# Example: Simulation-Based Optimization

- Have credible Monte Carlo simulation of real system
- Parameters **d** in simulation have physical meaning in system
  - E.g.: **d** is machine locations in plant layout, timing settings in traffic control, resource allocation in military operations, etc.
- Run simulation to determine best **d** for use in real system
- Want to minimize **average** measure of performance $L(\mathbf{d})$.

Let $y(\mathbf{d})$ represent one simulation output ($y(\mathbf{d}) = L(\mathbf{d})$ + noise)



# Outline

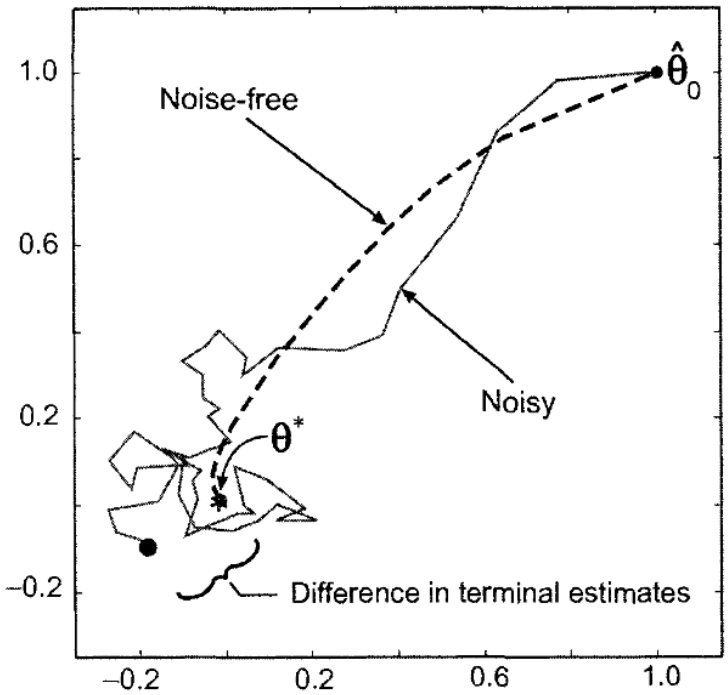
# Steepest Descent with Noisy/Noise-Free Gradient

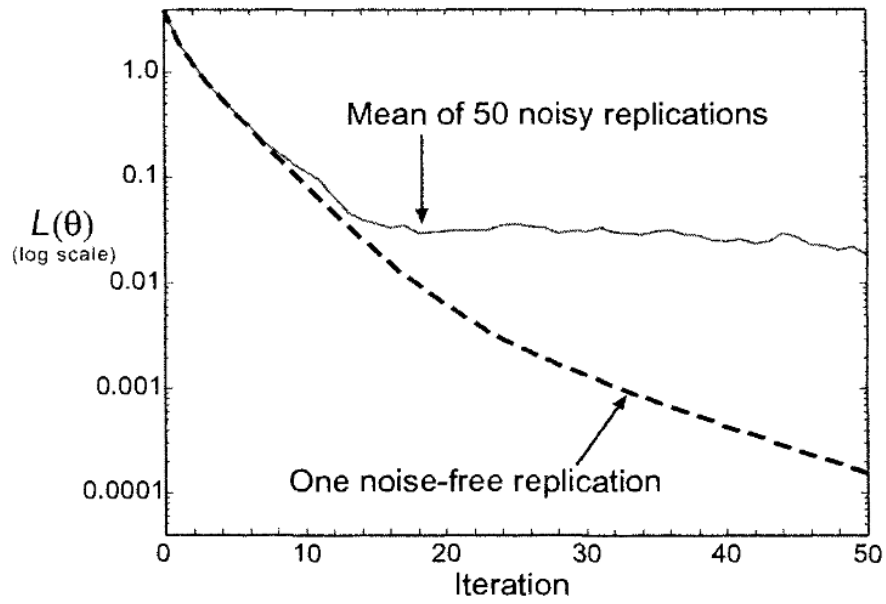
- $L([t_1, t_2]) = t_1^4 + t_1^2 + t_1 t_2 + t_2^2$, with the gradient

$$g([t_1, t_2]) = \begin{bmatrix} 4t_1^3 + 2t_1 + t_2 \\ t_1 + 2t_2 \end{bmatrix}$$

- Noisy gradient $Y([t_1, t_2]) = g([t_1, t_2]) + N(0, I_2)$
- Steepest descent: $\hat{\mathbf{d}}_{k+1} = \hat{\mathbf{d}}_k - aY(\hat{\mathbf{d}}_k)$, with $a = 0.05$
- $\hat{\mathbf{d}}_0 = [1, 1]'$
- Number of independent runs: 50

# Noisy/Noise-Free Gradient Comparison



# Outline

# Random Search Algorithms with Noisy Loss Function

- Basic implementation of random search assumes perfect (noise-free) values of $L$
- Some applications require use of noisy measurements: $y(\mathbf{d}) = L(\mathbf{d}) +$ noise
- The simplest modification is to form average of $N$ values $y(\mathbf{d})$ at each iteration as an approximation to $L(\mathbf{d})$
  - the error decreases only at a rate of $1/\sqrt{N}$
- Alternative modification is to set a threshold $\tau > 0$ for improvement before new value is accepted in algorithm
- Example: thresholding in localized algorithms B/C with modified step 2/3:

**[Step 2-mod.]** If $y(\mathbf{d}_{new}(k+1)) < y(\hat{\mathbf{d}}_k) - \tau_k$, set $\hat{\mathbf{d}}_{k+1} = \mathbf{d}_{new}(k+1)$; else set $\hat{\mathbf{d}}_{k+1} = \hat{\mathbf{d}}_k$, where $\tau_k \geq 0$

# Remarks on the heuristics for noise accommodation

- Threshold $\tau$ must be view in terms of the number of standard deviations in the measurement noise
  - if $sd = 0.1$ for all $k$, take $\tau_k = 0.2$
  - the new $\mathbf{d}$ is accepted only if shows a "two-standard deviation" improvement in the measured loss value
  - $P(L(\mathbf{d}_{new}) < L(\hat{\mathbf{d}}_k)) \geq 0.95$
  - convenient if noise is at least approximately normally distributed
- Averaging may greatly increase the number of loss evaluations required to obtain sufficient accuracy
- The altered threshold may conduct to reject too many changes in $\mathbf{d}$

# Nonlinear Simplex for Noisy Loss Measurements

- Nonlinear simplex algorithm uses ranks for sorting, so for small noise the algorithm is relative insensible; only replace $L(\mathbf{d})$ with $y(\mathbf{d}) = L(\mathbf{d}) + \varepsilon(\mathbf{d})$
- Large noise effects will frequently change the relative ranks of the loss values
- A first modification: average several loss measurements at given $\mathbf{d}$
- another modification: change the convergence criterion
  - common criterion for deterministic nonlinear simplex algorithm

$$\left( \sum_{i=1}^{p+1} [L(\mathbf{d}^i) - L(\mathbf{d}_{cent})]^2 \right)^{1/2} \leq \eta$$

  - stopping criterion for noisy loss values

$$\frac{\max_i \|\mathbf{d}^i - \mathbf{d}_{min}\|}{\max\{1, \|\mathbf{d}_{min}\|\}}$$

- Noise can lead to premature reduction of simplex size: change $\delta$ to 0.9

# Outline

# Stochastic Root-Finding Problem

- Focus is on finding $\mathbf{d}$ (i.e., $\mathbf{d}^*$) such that $g(\mathbf{d}) = 0$
  - $g(\mathbf{d})$ is typically a nonlinear function of $\mathbf{d}$
- Assume that only noisy measurements of $g(\mathbf{d})$ are available:

$$Y_k(\mathbf{d}) = g(\mathbf{d}) + e_k(\mathbf{d}), k = 0, 1, 2, \ldots,$$

- Sometimes an input measurement (random or deterministic) occurs

$$Y_k(\mathbf{d}) = \tilde{g}_k(\mathbf{d}, x_k) + \tilde{e}_k(\mathbf{d}, x_k), k = 0, 1, 2, \ldots,$$

- Above problem arises frequently in practice
  - Optimization with noisy measurements ($g(\mathbf{d})$ represents gradient of loss function)
  - Quantile-type problems
  - Equation solving in physics-based models
  - Machine learning

# Core Algorithm for Stochastic Root-Finding

- Basic algorithm published by Robbins and Monro (1951)
- Algorithm is a stochastic analogue to steepest descent when used for optimization
  - Noisy measurement $Y_k(\mathbf{d})$ replaces exact gradient $g(\mathbf{d})$
- Generally wasteful to average measurements at given value of $\mathbf{d}$
  - Average across iterations (changing $\mathbf{d}$)
- Core Robbins-Monro algorithm for unconstrained root-finding is

$$\hat{\mathbf{d}}_{k+1} = \hat{\mathbf{d}}_k - a_k Y_k(\hat{\mathbf{d}}_k), \text{ where } a_k > 0$$

# Example - production function in microeconomics

- Tradeoff between work hours ($W$) and capital equipment ($C$) in manufacturing a product
- Production function $h(\cdot)$

$$z_k = h(\mathbf{d}, x_k) + v_k$$

where $\mathbf{d} = [\lambda, \beta]$ ($\lambda > 0$ and $0 \leq \beta \leq 1$ are parameters having economic significance), $x_k = [W_k, C_k]$ represents the input used in the production at $k$ times period and $v_k$ is the random noise

- Cobb-Douglas form: $z_k = h(\mathbf{d}, x_k) + v_k = \lambda C_k^{\beta} W_k^{1-\beta} + v_k$
- The loss function is the expected squared error

$$L(\mathbf{d}) = \frac{1}{2} E\left\{[z_{k+1} - h(\mathbf{d}, x_{k+1})]^2\right\}$$

- The noisy gradient (or *stochastic gradient*)

$$Y_k(\mathbf{d}) = \frac{1}{2} \frac{\partial}{\partial \mathbf{d}} [z_{k+1} - h(\mathbf{d}, x_{k+1})]^2$$

- The noisy gradient at step $k$ is then:

$$Y_k(\hat{\mathbf{d}}_k) = [h(\hat{\mathbf{d}}_k, x_{k+1}) - z_{k+1}] \frac{\partial h(\mathbf{d}, x_{k+1})}{\partial \mathbf{d}}\Big|_{\mathbf{d}=\hat{\mathbf{d}}_k}$$

# Outline

# Convergence Conditions

- Central aspect of root-finding SA are conditions for formal convergence of the iterate to a root $\mathbf{d}^*$
- Provides rigorous basis for many popular algorithms (LMS, back-propagation, simulated annealing, evolutionary computation, MCMC, etc.)
- Two sets of conditions (apply when there is a unique root $\mathbf{d}^*$):
  - "Statistics" conditions based on classical assumptions about $g(\mathbf{d})$, noise, and gains $a_k$
  - "Engineering" conditions based on connection to deterministic ordinary differential equation (ODE)
- Neither of statistics or engineering conditions is special case of other
- These are *sufficient* conditions, (so satisfactory results when one or more conditions are not satisfied)

# Statistical Convergence Conditions

A.1 (Gain sequence) $a_k > 0, a_k \to 0, \sum_{i=1}^{\infty} a_k = \infty$ and $\sum_{i=0}^{\infty} a_k^2 < \infty$

A.2 (Mean-zero noise) $E[e_k(\mathbf{d})] = 0$ for all $\mathbf{d}$ and $k$.

(the following conditions are usually ignored in practice)

A.3 (Search direction) For some symmetric, positive definite matrix $\mathbf{B}$ (often, $\mathbf{B} = \mathbf{I}_p$) and every $0 < \eta < 1$,

$$\inf_{\eta < \|\mathbf{d} - \mathbf{d}^*\| < 1/\eta} (\mathbf{d} - \mathbf{d}^*)^T \mathbf{B} g(\mathbf{d}) > 0$$

A.4 (Growth and variance bounds) $\|g(\mathbf{d})\|^2 + E\left(\|e_k(\mathbf{d})\|^2\right) \leq c(1 + \|\mathbf{d}\|^2)$ for all $\mathbf{d}$ and $k$ and some $c > 0$

# Engineering Convergence Conditions

B.1 (Gain sequence) $a_k > 0, a_k \to 0, \sum_{i=1}^{\infty} a_k = \infty$.

(the following conditions are usually ignored in practice)

B.2 (Relationship to ODE) $g(\mathbf{d})$ continuous and, for $\mathbf{Z}(\tau) \in \mathcal{R}^p$, the differential equation $d\mathbf{Z}/d\tau = -g(\mathbf{Z}(\tau))$ has an asymptotically stable equilibrium point at $\mathbf{d}^*$.

B.3 (Iterate boundedness) $\sup_{k \geq 0} \|\hat{\mathbf{d}}_k\| < \infty$ a.s. Further, $\hat{\mathbf{d}}_k$ lies in a compact subset of the domain of attraction for the differential equation in B.2 infinitely often.

B.4 (Bounded variance property of measurement error) Let $\mathcal{F}_k \equiv \{\hat{\mathbf{d}}_0, \ldots, \hat{\mathbf{d}}_k\}, b_k = E[e_k(\hat{\mathbf{d}}_k | \mathcal{F}_k|]$. Then $E\left[\| \sum_{k=0}^{\infty} a_k (e_k - b_k)\|^2\right] < \infty$

B.5 (Disappearing bias) $\sup_{k \geq 0} \|b_k\| < \infty$ a.s. and $b_k \to 0$ a.s. as $k \to \infty$

# Gain Selection

- Choice of the gain sequence $a_k$ is critical to the performance of SA
- Famous conditions for convergence are $\sum_{i=1}^{\infty} a_k = \infty$ and $\sum_{i=0}^{\infty} a_k^2 < \infty$
- A common practical choice of gain sequence is

$$a_k = \frac{a}{(k + 1 + A)^{\alpha}}$$

  where $1/2 < \alpha \leq 1$, $a > 0$, and $A \geq 0$
- Strictly positive $A$ ("stability constant") allows for larger $a$ (possibly faster convergence) without risking unstable behavior in early iterations
- $\alpha$ and $A$ can usually be pre-specified; critical coefficient $a$ usually chosen by "trial-and-error"

# Outline

# Adaptive Estimation and Higher-Order Algorithms

- Methods for adaptively estimating the gain $a_k$
  a) use the information acquired during the search process
  b) use SA analogues of the Newton-Raphson search
- Method (a) - use the signs of the differences $\hat{\mathbf{d}}_{k+1} - \hat{\mathbf{d}}_k$ ($\mathbf{d}$ real)
  - frequent sign changes: iterate $\hat{\mathbf{d}}_k$ near $\mathbf{d}^*$, so small gain to use
  - signs not changing: iterate $\hat{\mathbf{d}}_k$ far from $\mathbf{d}^*$, so large gain to use
- Method (b) - replace the scalar $a_k$ by a matrix approximating the (unknown) true inverse of Jacobian matrix

# Iterate Averaging

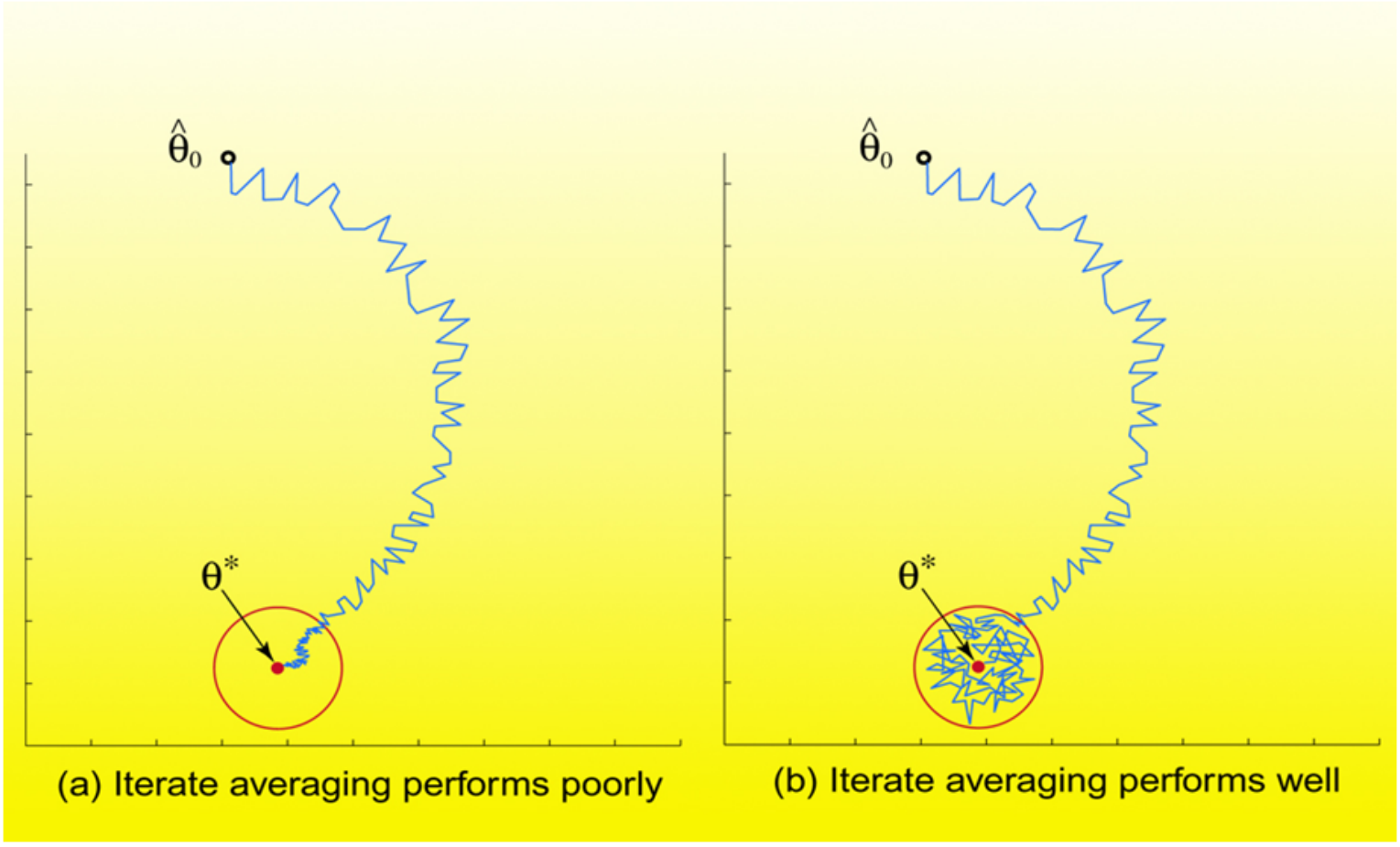
- Iterate averaging is important and relatively recent development in SA
- Provides means for achieving optimal **asymptotic** performance without using optimal gains $a_k$
- Basic iterate average uses following sample mean as final estimate:

$$\bar{\theta}_k \equiv \frac{1}{k+1} \sum_{j=0}^{k} \hat{\mathbf{d}}_j$$

- Supplementary condition: $a_{k+1}/a_k = 1 - o(a_k)$
- Results in **finite-sample** practice are mixed. Success relies on large proportion of individual iterates hovering in some balanced way around $\mathbf{d}^*$
  - Many practical problems have iterate approaching $\mathbf{d}^*$ in roughly monotonic manner
  - Monotonicity not consistent with good performance of iterate averaging;

# Ineffective and Effective Uses of Iterate Averaging

- (a) The process does not oscillate about $\mathbf{d}^*$
- (b): The process oscillates around $\mathbf{d}^*$



# Outline

# Stochastic Gradient Formulation

- Classical optimization problem of minimizing a loss function $L(\mathbf{d})$
- For differentiable $L(\mathbf{d})$, the minimum $\mathbf{d}^*$ is one of the roots of the equation:

$$g(\mathbf{d}) = \frac{\partial L}{\partial \mathbf{d}} = 0$$

- Above is special case of root-finding problem
- Suppose cannot observe $L(\mathbf{d})$ and $g(\mathbf{d})$ except in presence of noise
  - Adaptive control (target tracking)
  - Simulation-based optimization
- Seek **unbiased measurement** of $\partial L/\partial \mathbf{d}$ for optimization

# Basic principles

- Suppose $L(\mathbf{d}) = E[Q(\mathbf{d}, \mathbf{V})]$
  - $\mathbf{V}$ represents all random effects
  - $Q(\mathbf{d}, \mathbf{V})$ represents "observed" cost (noisy measurement of $L(\mathbf{d})$)
- Seek a representation for $L(\cdot)$ such that $\partial Q/\partial \mathbf{d}$ (called stochastic gradient) is an unbiased measurement of $\partial L/\partial \mathbf{d}$
  - Generally true only when distribution function for $\mathbf{V}$ is independent on $\mathbf{d}$
- Consequently the desired representation is

$$E[Q(\mathbf{d}, \mathbf{V})] = \int Q(\mathbf{d}, v) p_{\mathbf{V}}(v) dv$$

**and not**

$$E[Q(\mathbf{d}, \mathbf{V})] = \int Q(\mathbf{d}, v) p_{\mathbf{V}}(v|\mathbf{d}) dv$$

where $p_{\mathbf{V}}(\cdot)$ is density function for $\mathbf{V}$

# Outline

# Stochastic Gradient Measurement and Algorithm

- When density $p_{\mathbf{V}}(\cdot)$ is independent of $\mathbf{d}$,

$$Y(\mathbf{d}) = \frac{\partial Q(\mathbf{d}, \mathbf{V})}{\partial \mathbf{d}}$$

  is unbiased measurement of $\partial L / \partial \mathbf{d}$, i.e.

$$E[Y(\mathbf{d})] = \partial L / \partial \mathbf{d} = g(\mathbf{d})$$

- Can use root-finding (Robbins-Monro) SA algorithm to attempt to find $\mathbf{d}^*$:

$$\hat{\mathbf{d}}_{k+1} = \hat{\mathbf{d}}_k - a_k \left. \frac{\partial Q_k(\mathbf{d}, \mathbf{V}_k)}{\partial \mathbf{d}} \right|_{\mathbf{d}=\hat{\mathbf{d}}_k} = \hat{\mathbf{d}}_k - a_k Y_k(\hat{\mathbf{d}}_k)$$

- Unbiased measurement satisfies key convergence conditions of SA

# Implementation variants of SGA

- Instantaneous (recursive) input

  recursive implementation: $\hat{\mathbf{d}}_{k+1} = \hat{\mathbf{d}}_k - a_k Y_k(\hat{\mathbf{d}}_k)$

  Init $L(\mathbf{d}) = E[Q(\mathbf{d}, \mathbf{V})]$; $Y(\mathbf{d}) = \partial Q(\mathbf{d}, \mathbf{V})/\partial \mathbf{d}$;
  Set $\hat{\mathbf{d}}_0$ and choose gain sequence $a_k$

  Step 0 Get input data: random vector $\mathbf{V}_0$ (and, sometimes, deterministic vector $\mathbf{X}_0$)

  $$\hat{\mathbf{d}}_1 = \hat{\mathbf{d}}_0 - a_0 \frac{\partial Q(\mathbf{d}, \mathbf{V}_0)}{\partial \mathbf{d}}\bigg|_{\mathbf{d}=\hat{\mathbf{d}}_0} = \hat{\mathbf{d}}_0 - a_0 Y_0(\hat{\mathbf{d}}_0)$$

  Step 1 Get input data: random vector $\mathbf{V}_1$ (and $\mathbf{X}_1$)

  $$\hat{\mathbf{d}}_2 = \hat{\mathbf{d}}_1 - a_1 \frac{\partial Q(\mathbf{d}, \mathbf{V}_1)}{\partial \mathbf{d}}\bigg|_{\mathbf{d}=\hat{\mathbf{d}}_1} = \hat{\mathbf{d}}_1 - a_1 Y_1(\hat{\mathbf{d}}_1)$$

  ...

  Step $k$ Get input data: random vector $\mathbf{V}_k$ (and $\mathbf{X}_k$)

  $$\hat{\mathbf{d}}_{k+1} = \hat{\mathbf{d}}_k - a_k \frac{\partial Q(\mathbf{d}, \mathbf{V}_k)}{\partial \mathbf{d}}\bigg|_{\mathbf{d}=\hat{\mathbf{d}}_k} = \hat{\mathbf{d}}_k - a_k Y_k(\hat{\mathbf{d}}_k)$$

  - Continue until no more input data is available or some stopping criteria is fulfil

# Implementation variants of SGA (cont.)

- Batch input
  - $\bar{Q}(\mathbf{d}, \mathbf{V})$: average (over the number of measurements) observed cost function, where $\mathbf{V}$ is the overall set of random inputs

    batch implementation: $\hat{\mathbf{d}}_{k+1} = \hat{\mathbf{d}}_k - a_k Y(\hat{\mathbf{d}}_k)$

Init $L(\mathbf{d}) = E[Q(\mathbf{d}, \mathbf{V})]$; Get all data input: random vectors $\mathbf{V}_0, \mathbf{V}_1, ..., \mathbf{V}_N$ (and eventually deterministic vectors $\mathbf{X}_0, ..., \mathbf{X}_N$). Let be

$$\bar{Q}(\mathbf{d}) = \frac{1}{N+1} \sum_{i=0}^{N} Q(\mathbf{d}, \mathbf{V}_i) \text{ and}$$

$$Y(\mathbf{d}) = \partial \bar{Q}(\mathbf{d}) / \partial \mathbf{d}$$

Set $\hat{\mathbf{d}}_0$ and choose gain sequence $a_k$.

Step 0 $\hat{\mathbf{d}}_1 = \hat{\mathbf{d}}_0 - a_0 \frac{\partial \bar{Q}(\mathbf{d})}{\partial \mathbf{d}}\Big|_{\mathbf{d}=\hat{\mathbf{d}}_0} = \hat{\mathbf{d}}_0 - a_0 Y(\hat{\mathbf{d}}_0)$

...

Step $k$ $\hat{\mathbf{d}}_{k+1} = \hat{\mathbf{d}}_k - a_k \frac{\partial \bar{Q}(\mathbf{d})}{\partial \mathbf{d}}\Big|_{\mathbf{d}=\hat{\mathbf{d}}_k} = \hat{\mathbf{d}}_k - a_k Y(\hat{\mathbf{d}}_k)$

- Continue until some stopping criteria is fulfil

# Implementation variants of SGA (cont.)

- Multiple-pass implementation:
  - after the algorithm has passed through data one time (recursive implementation), it returns to the first data point with the initial $\mathbf{d}$ equal the the last estimate of the previous pass
  - the gain value: reset to $a_0$ or keep the current ($a_k$) value

Init $L(\mathbf{d}) = E[Q(\mathbf{d}, \mathbf{V})]$; $Y(\mathbf{d}) = \partial Q(\mathbf{d}, \mathbf{V})/\partial \mathbf{d}$;
Set $\hat{\mathbf{d}}_0$ and choose gain sequence $a_k$

Step 0 Get input data: random vector $\mathbf{V}_0$ (and, sometimes, deterministic vector $\mathbf{X}_0$)

$$\hat{\mathbf{d}}_1 = \hat{\mathbf{d}}_0 - a_0 \frac{\partial Q(\mathbf{d}, \mathbf{V}_0)}{\partial \mathbf{d}}\bigg|_{\mathbf{d}=\hat{\mathbf{d}}_0} = \hat{\mathbf{d}}_0 - a_0 Y_0(\hat{\mathbf{d}}_0)$$

...

Step N Get last input data: random vector $\mathbf{V}_N$ (and $\mathbf{X}_N$)

$$\hat{\mathbf{d}}_{N+1} = \hat{\mathbf{d}}_N - a_N \frac{\partial Q(\mathbf{d}, \mathbf{V}_N)}{\partial \mathbf{d}}\bigg|_{\mathbf{d}=\hat{\mathbf{d}}_N} = \hat{\mathbf{d}}_N - a_N Y_N(\hat{\mathbf{d}}_N)$$

Step N+1 Get first input data: random vector $\mathbf{V}_0$ (and $\mathbf{X}_0$)

$$\hat{\mathbf{d}}_{N+2} = \hat{\mathbf{d}}_{N+1} - a_{N+1} \frac{\partial Q(\mathbf{d}, \mathbf{V}_0)}{\partial \mathbf{d}}\bigg|_{\mathbf{d}=\hat{\mathbf{d}}_{N+1}} = \hat{\mathbf{d}}_{N+1} - a_{N+1} Y_0(\hat{\mathbf{d}}_{N+1})$$

...

- Continue until some stopping criteria is fulfil

# Example

- Cobb-Douglas model: $h(\mathbf{d}, \mathbf{X}) = h(\lambda, \beta; C, W) = \lambda C^{\beta} W^{1-\beta}$
- For the input: $\mathbf{d} = (\lambda, \beta)$ (parameter of interest), $\mathbf{X} = (C, W)$ (deterministic factors) and $\mathbf{Z}$ (random factor), the cost (measurable) function is

$$Q(\mathbf{d}; \mathbf{X}; \mathbf{Z}) = \frac{1}{2}[\mathbf{Z} - h(\mathbf{d}; \mathbf{X})]^2$$

and the loss function

$$L(\mathbf{d}) = E\left[Q(\mathbf{d}; \mathbf{X}; \mathbf{Z})\right]$$

- Stochastic gradient:

$$Y(\mathbf{d}; \mathbf{X}; \mathbf{Z}) = \frac{\partial Q(\mathbf{d}; \mathbf{X}; \mathbf{Z})}{\partial \mathbf{d}} = [h(\mathbf{d}; \mathbf{X}) - \mathbf{Z}] \frac{\partial h(\mathbf{d}; \mathbf{X})}{\partial \mathbf{d}} =$$

$$= \left[\lambda C^{\beta} W^{1-\beta} - \mathbf{Z}\right] \begin{bmatrix} \frac{\partial(\lambda C^{\beta} W^{1-\beta})}{\partial \lambda} \\ \\ \frac{\partial(\lambda C^{\beta} W^{1-\beta})}{\partial \beta} \end{bmatrix} = \left[\lambda C^{\beta} W^{1-\beta} - \mathbf{Z}\right] \begin{bmatrix} C^{\beta} W^{1-\beta} \\ \lambda C^{\beta} W^{1-\beta} (\ln(C) - \ln(W)) \end{bmatrix}$$

$$= C^{\beta} W^{1-\beta} \left[\lambda C^{\beta} W^{1-\beta} - \mathbf{Z}\right] \begin{bmatrix} 1 \\ \lambda(\ln(C) - \ln(W)) \end{bmatrix}$$

# Example (cont.)

- recursive implementation: $\hat{\mathbf{d}}_{k+1} = \hat{\mathbf{d}}_k - a_k Y_k(\hat{\mathbf{d}}_k)$
  - Suppose $\hat{\mathbf{d}}_0 = [\lambda_0, \beta_0]' = [0,\ 1]'$, $\mathbf{X}_0 = (C_0, W_0) = (1, 2e)$ and $\mathbf{Z}_0 = e$
  - Consider the gain sequence $a_k = \frac{1}{k+1}$
  - $\hat{\mathbf{d}}_1 = \hat{\mathbf{d}}_0 - a_0 Y_0(\hat{\mathbf{d}}_0) = \begin{bmatrix} \lambda_0 \\ \beta_0 \end{bmatrix} - a_0 Y(\hat{\mathbf{d}}_0, \mathbf{X}_0, \mathbf{Z}_0) = \begin{bmatrix} \lambda_0 \\ \beta_0 \end{bmatrix} - a_0 \cdot C_0^{\beta_0} W_0^{1-\beta_0} \left[\lambda_0 \cdot C_0^{\beta_0} W_0^{1-\beta_0} - Z_0\right] \begin{bmatrix} 1 \\ \lambda_0 \cdot (\ln(C_0) - \ln(W_0)) \end{bmatrix}$
    $= \begin{bmatrix} 0 \\ 1 \end{bmatrix} - 1 \cdot 1^1 \cdot (2e)^0 \cdot \left[0 \cdot 1^1 \cdot (2e)^0 - e\right] \begin{bmatrix} 1 \\ 0 \cdot (\ln(1) - \ln(2e)) \end{bmatrix} =$
    $\begin{bmatrix} 0 \\ 1 \end{bmatrix} + e \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} e \\ 1 \end{bmatrix}$

# Example (cont.)

- batch implementation: $\hat{\mathbf{d}}_{k+1} = \hat{\mathbf{d}}_k - a_k Y(\hat{\mathbf{d}}_k)$
- Suppose we have only two data input: $(\mathbf{X}_0, \mathbf{Z}_0) = (C_0, W_0, \mathbf{Z}_0) = (1, 2e, e)$ and $(\mathbf{X}_1, \mathbf{Z}_1) = (C_1, W_1, \mathbf{Z}_1) = (2, 2e, 2e)$
- $Y(\mathbf{d}) = \frac{1}{2}\left(Y_0(\mathbf{d}) + Y_1(\mathbf{d})\right) = \frac{1}{2}\left(Y(\mathbf{d}, \mathbf{X}_0, \mathbf{Z}_0) + Y(\mathbf{d}, \mathbf{X}_1, \mathbf{Z}_1)\right) =$
  $\frac{1}{2}\left(C_0^{\beta} W_0^{1-\beta}[\lambda C_0^{\beta} W_0^{1-\beta} - \mathbf{z}_0]\left[\begin{smallmatrix} 1 \\ \lambda(\ln(C_0) - \ln(W_0)) \end{smallmatrix}\right] + C_1^{\beta} W_1^{1-\beta}[\lambda C_1^{\beta} W_1^{1-\beta} - \mathbf{z}_1]\left[\begin{smallmatrix} 1 \\ \lambda(\ln(C_1) - \ln(W_1)) \end{smallmatrix}\right]\right)$
  $=$
  $\frac{1}{2}\left\{1^{\beta}(2e)^{1-\beta}[\lambda 1^{\beta}(2e)^{1-\beta} - e]\left[\begin{smallmatrix} 1 \\ \lambda(\ln(1) - \ln(2e)) \end{smallmatrix}\right] + 2^{\beta}(2e)^{1-\beta}[\lambda 2^{\beta}(2e)^{1-\beta} - 2e]\left[\begin{smallmatrix} 1 \\ \lambda(\ln(2) - \ln(2e)) \end{smallmatrix}\right]\right\}$
- Suppose $\hat{\mathbf{d}}_0 = [\lambda_0, \beta_0]' = [0, 1]'$ and $a_k = \frac{1}{k+1}$.
- We have $Y(\hat{\mathbf{d}}_0) = \frac{1}{2}\left\{1 \cdot [0 - e]\begin{bmatrix} 1 \\ 0 \end{bmatrix} + 2[0 - 2e]\begin{bmatrix} 1 \\ 0 \end{bmatrix}\right\} = \begin{bmatrix} -2.5e \\ 0 \end{bmatrix}$
  $\hat{\mathbf{d}}_1 = \hat{\mathbf{d}}_0 - a_0 Y(\hat{\mathbf{d}}_0) = \begin{bmatrix} 0 \\ 1 \end{bmatrix} - 1 \cdot \begin{bmatrix} -2.5e \\ 0 \end{bmatrix} = \begin{bmatrix} 2.5e \\ 1 \end{bmatrix}$

# Outline

# Motivation for Gradient-Free Algorithms

- Interest in optimization problems for which the stochastic gradient for $L(\mathbf{d}) = E[Q(\mathbf{d}, \mathbf{V})]$, which is $Y(\mathbf{d}) = \partial Q / \partial \mathbf{d}$, is not available
  - cannot use techniques such as Robbins-Monro SA, steepest descent, etc.
  - can (in principle) use techniques such as Kiefer and Wolfowitz SA, genetic algorithms, ...
- Many such "gradient-free" problems arise in practice
  - Generic difficult parameter estimation
  - Model-free feedback control
  - Simulation-based optimization
  - Experimental design: sensor configuration

# Outline

# Finite Difference SA (FDSA) Method

- FDSA looks as standard stochastic gradient form of root-finding (Robbins-Monro) SA
  - Finite difference approximation replaces direct gradient measurement
  - Resulting algorithm sometimes called Kiefer-Wolfowitz SA
- General form of FDSA algorithm:

$$\hat{\mathbf{d}}_{k+1} = \hat{\mathbf{d}}_k - a_k \hat{g}_k(\hat{\mathbf{d}}_k)$$

  - $\hat{\mathbf{d}}_k$ - estimate for $\theta$ at the $k^{th}$ iteration
  - $\hat{g}_k(\theta)$ - FD estimate of $g(\theta)$ at $k^{th}$ iteration
  - $a_k$ - nonnegative gain value
- Under conditions, $\hat{\mathbf{d}}_k \to \mathbf{d}^*$ in stochastic sense (a.s.)

# Finite Difference Gradient Approximation

- Finite Difference gradient approximation used in SA recursion as gradient measurement
- Standard one-side and two-sided gradient approximation at iteration $k$ are

$$\hat{g}_k(\hat{\mathbf{d}}_k) = \begin{bmatrix} \frac{y(\hat{\mathbf{d}}_k + c_k\xi_1) - y(\hat{\mathbf{d}}_k)}{c_k} \\ \vdots \\ \frac{y(\hat{\mathbf{d}}_k + c_k\xi_p) - y(\hat{\mathbf{d}}_k)}{c_k} \end{bmatrix} \quad \hat{g}_k(\hat{\mathbf{d}}_k) = \begin{bmatrix} \frac{y(\hat{\mathbf{d}}_k + c_k\xi_1) - y(\hat{\mathbf{d}}_k - c_k\xi_1)}{2c_k} \\ \vdots \\ \frac{y(\hat{\mathbf{d}}_k + c_k\xi_p) - y(\hat{\mathbf{d}}_k - c_k\xi_p)}{2c_k} \end{bmatrix}$$

  where $\xi_j$ is $p$-dimensional with 1 in $j^{th}$ entry, 0 elsewhere
- Each computation of one-side FD approximation takes $p + 1$ measurements of $y(\cdot)$
- Each computation of two-sided FD approximation takes $2p$ measurements of $y(\cdot)$

# Example of FD approximation

- Consider $L(\theta) = 2t_1^2 + t_2^2 - 3t_1 t_2$, where $\theta = [t_1, t_2]$
- Suppose $L(\cdot)$ is noise-free (i.e. $y(\theta) = L(\theta)$ for all $\theta$)
- Let be $\hat{\mathbf{d}}_0 = [1, 1]$ and $c_0 = 0.5$.
- One-side FD gradient approximation:

$$\hat{g}_0(\hat{\mathbf{d}}_0) = \begin{bmatrix} \frac{y(\hat{\mathbf{d}}_0 + c_0\xi_1) - y(\hat{\mathbf{d}}_0)}{c_0} \\ \frac{y(\hat{\mathbf{d}}_0 + c_0\xi_2) - y(\hat{\mathbf{d}}_0)}{c_0} \end{bmatrix} = \frac{1}{0.5} \begin{bmatrix} L([1,1]+0.5[1,0]) - L([1,1]) \\ L([1,1]+0.5[0,1]) - L([1,1]) \end{bmatrix}$$

$$= 2 \begin{bmatrix} L([1.5,1]) - L([1,1]) \\ L([1,1.5]) - L([1,1]) \end{bmatrix} = 2 \begin{bmatrix} 1-0 \\ -0.25-0 \end{bmatrix} = \begin{bmatrix} 2 \\ -0.5 \end{bmatrix}$$

- Two-sides FD gradient approximation:

$$\hat{g}_0(\hat{\mathbf{d}}_0) = \begin{bmatrix} \frac{y(\hat{\mathbf{d}}_0 + c_0\xi_1) - y(\hat{\mathbf{d}}_0 - c_0\xi_1)}{2c_0} \\ \frac{y(\hat{\mathbf{d}}_0 + c_0\xi_2) - y(\hat{\mathbf{d}}_0 - c_0\xi_2)}{2c_0} \end{bmatrix} = \frac{1}{1} \begin{bmatrix} L([1,1]+0.5[1,0]) - L([1,1]-0.5[1,0]) \\ L([1,1]+0.5[0,1]) - L([1,1]-0.5[0,1]) \end{bmatrix}$$

$$= \begin{bmatrix} L([1.5,1]) - L([0.5,1]) \\ L([1,1.5]) - L([1,0.5]) \end{bmatrix} = \begin{bmatrix} 1-0 \\ -0.25-0.75 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

# Convergence conditions

- Statistics conditions

  A.1 **(Gain sequence)**: $a_k > 0$, $c_k > 0$, $a_k \to 0$, $c_k \to 0$,

$$\sum_{k=0}^{\infty} a_k = \infty, \sum_{k=0}^{\infty} a_k c_k < \infty \text{ and } \sum_{k=0}^{\infty} a_k^2 / c_k^2 < \infty$$

  A.2 **(Unique minimum)**
  A.3 **(Mean-zero and finite variance noise)**
  A.4 **(Bounded hessian matrix)**

- Engineering conditions

  B.1 **(Gain sequence)**: $a_k > 0$, $c_k > 0$, $a_k \to 0$, $c_k \to 0$,

$$\sum_{k=0}^{\infty} a_k = \infty \text{ and } \sum_{k=0}^{\infty} a_k^2 / c_k^2 < \infty$$

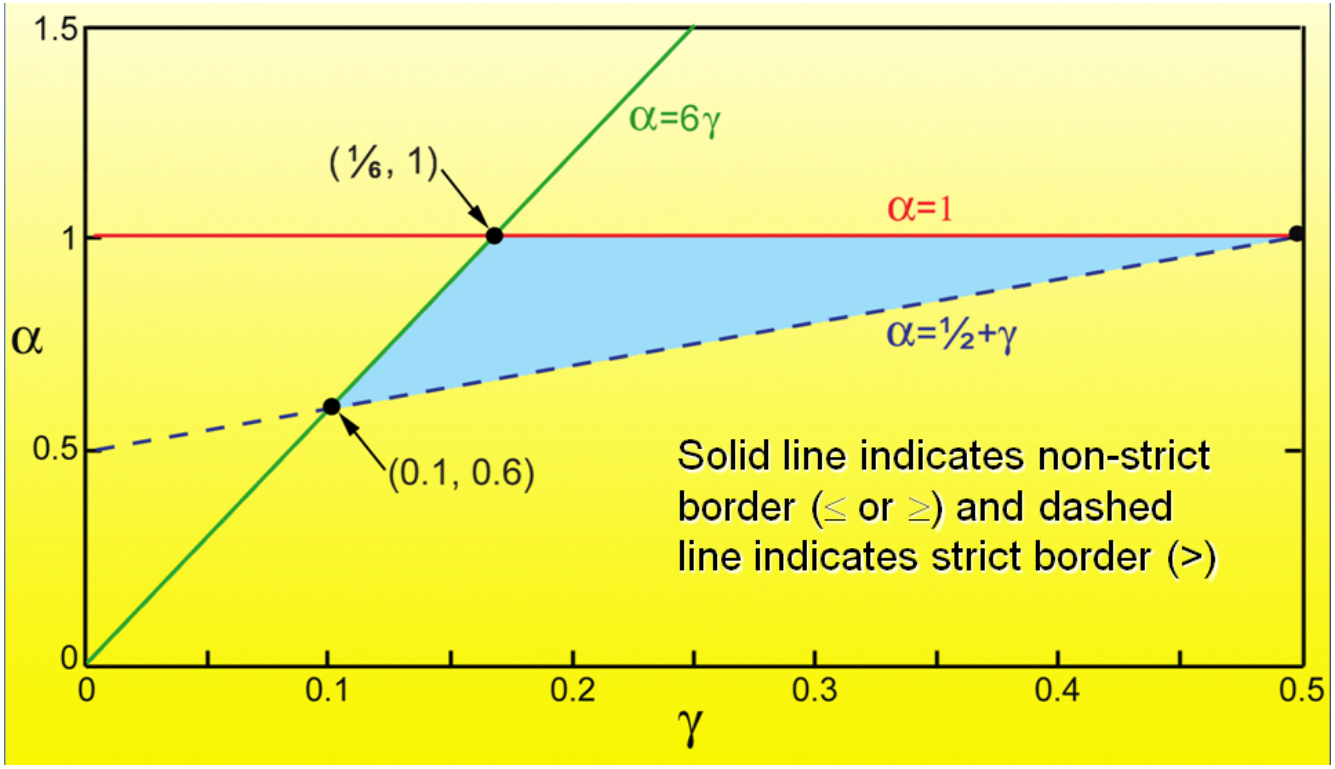
  B.2 **(Relationship to ODE)**
  B.3 **(Iterate boundedness)**
  B.4 **(Mean-zero and finite variance noise)**
  B.5 **(Smoothness of L)**

# Gain sequences

- Asymptotic normality applies under specific choices of gain sequences

$$a_k = \frac{a}{(k+1+A)^{\alpha}}, c_k = \frac{c}{(k+1)^{\gamma}}$$

- Constraints: $\alpha > 1/2$, $\gamma > 0$, $\alpha - 2\gamma > 0$, $3\gamma - \alpha/2 > 0$



# Semiautomatic gain selection process

- General values: $\alpha = 0.602, \gamma = 0.101$
- To cope with noise effects: $c \approx$ standard deviation of the measurement noise in $y(\mathbf{d})$ (estimate it using several $y(\mathbf{d}_0)$ measurements)
- Choose $A \approx 10\%$ of maximum number of expected (allowed) iterations
- Choose $a$ as follow:
  - calculate $a_{temp,i}$ from $\frac{a_{temp,i}}{(A+1)^{0.602}} \hat{g}_{0i}(\hat{\mathbf{d}}_0) = \hat{\mathbf{d}}_{1i} - \hat{\mathbf{d}}_{0i} \approx$ desired change in magnitude in the $i^{th}$ element of $\mathbf{d}$
  - $a = min\{a_{temp,1}, ...., a_{temp,p}\}$
- Refinement of values using "trial runs"

# Example

- Suppose that
  - the maximum number of loss measurements used by FDSA algorithm is 6000
  - the noise has an approximate standard deviation of 0.5
  - the dimension of $\mathbf{d}$ is $p = 3$
  - the expected change in magnitude in the early iterations is 0.1 for each coordinate of $\theta$.
  - the estimate gradient $\hat{g}_0(\hat{\theta}_0)$ is $[10, 20, 10]$
- The semiautomatic gain selection process sets
  - $\alpha = 0.602$ and $\gamma = 0.101$
  - $c = 0.5$
  - $A = 0.1 \cdot 6000/(2 \cdot 3) = 100$
  - from $\frac{a_{temp,1}}{(101)^{0.602}} \hat{g}_{01}(\hat{\mathbf{d}}_0) = 0.1$ we get $a_{temp,1} = 0.16$; similarly, $a_{temp,2} = 0.08$ and $a_{temp,1} = 0.16$
  - $a = \min\{a_{temp,i}\}$ so $a = 0.08$
- The gain sequences:

$$a_k = \frac{0.08}{(k + 1 + 100)^{0.602}}, \quad c_k = \frac{0.5}{(k + 1)^{0.101}}$$

# Outline

# Simultaneous Perturbation Stochastic Approximation Algorithm

- Let $\hat{g}_k(\theta)$ denote SP estimate of $g(\theta)$ at $k^{th}$ iteration
- Let $\hat{\theta}_k$ denote estimate for $\theta^*$ at $k^{th}$ iteration
- The iterative SPSA algorithm has the form

$$\hat{\theta}_{k+1} = \hat{\theta}_k - a_k \hat{g}_k(\hat{\theta}_k)$$

  where $a_k$ is nonnegative gain sequence
- Simultaneous perturbation estimate $\hat{g}_k(\hat{\mathbf{d}}_k)$ is critical !
- Under specific conditions, $\hat{\theta}_k \to \theta^*$ in "almost sure" (a.s.) stochastic sense as $k \to \infty$

# Computation of $\hat{g}_k(\cdot)$ (Heart of SPSA)

- Let $\Delta_k$ be a vector of $p$ independent random variables at $k^{th}$ iteration

$$\Delta_k = [\Delta_{k1}, \ldots, \Delta_{kp}]$$

- $\Delta_k$ is typically random generated
- Let $\{c_k\}$ be a sequence of positive scalars
- For iteration $k \rightarrow k+1$, take measurements in the points $\hat{\mathbf{d}}_k \pm c_k\Delta_k$ (two perturbations of $\hat{\mathbf{d}}_k$)

$$\begin{aligned} y(\hat{\theta}_k + c_k\Delta_k) &= L(\hat{\theta}_k + c_k\Delta_k) + \epsilon_k^{(+)} \\ y(\hat{\theta}_k - c_k\Delta_k) &= L(\hat{\theta}_k - c_k\Delta_k) + \epsilon_k^{(-)} \end{aligned}$$

  where $\epsilon_k^{(\pm)}$ are measurement noise terms

- Common special case is when $\epsilon_k^{(\pm)} = 0, \forall k$ (e.g., system identification with perfect measurements of the likelihood function)

# Computation of $\hat{g}_k(\cdot)$ cont.

- The standard two-sided SP form for $\hat{g}_k(\cdot)$:

$$\hat{g}_k(\hat{\theta}_k) = \begin{bmatrix} \frac{y(\hat{\theta}_k + c_k\Delta_k) - y(\hat{\theta}_k - c_k\Delta_k)}{2c_k\Delta_{k1}} \\ \vdots \\ \frac{y(\hat{\theta}_k + c_k\Delta_k) - y(\hat{\theta}_k - c_k\Delta_k)}{2c_k\Delta_{kp}} \end{bmatrix}$$

$$= \frac{y(\hat{\theta}_k + c_k\Delta_k) - y(\hat{\theta}_k - c_k\Delta_k)}{2c_k} [\Delta_{k1}^{-1}, \ldots, \Delta_{kp}^{-1}]^T$$

- Note that $\hat{g}_k(\cdot)$ only requires ***two*** measurements of $L(\cdot)$ ***independent*** of $p$
- Above SP form contrasts with standard finite-difference approximations taking $2p$ (or $p + 1$) measurements
- Intuitive reason why $\hat{g}_k(\cdot)$ is appropriate is that $E[\hat{g}_k(\hat{\theta}_k)|\hat{\theta}_k] \approx g(\hat{\theta}_k)$

# Example of SP approximation

- Consider $L(\theta) = 2t_1^2 - 3t_2^2 + t_3 + 4t_1 t_2 t_3$, where $\theta = [t_1, t_2, t_3]$
- Suppose $L(\cdot)$ is noise-free (i.e. $y(\theta) = L(\theta)$ for all $\theta$)
- Let be $\hat{\mathbf{d}}_k = [2, 0, 1]$, $c_k = 0.5$ and suppose the following Bernouilli distributed random vector $\Delta_k = [-1, 1, 1]$ was generated
- Two-sided SP gradient approximation:

$$\hat{g}_k(\hat{\mathbf{d}}_k) = \begin{bmatrix} \frac{y(\hat{\mathbf{d}}_k + c_k\Delta_k) - y(\hat{\mathbf{d}}_k - c_k\Delta_k)}{2c_k\Delta_{k1}} \\ \frac{y(\hat{\mathbf{d}}_k + c_k\Delta_k) - y(\hat{\mathbf{d}}_k - c_k\Delta_k)}{2c_k\Delta_{k2}} \\ \frac{y(\hat{\mathbf{d}}_k + c_k\Delta_k) - y(\hat{\mathbf{d}}_k - c_k\Delta_k)}{2c_k\Delta_{k3}} \end{bmatrix} = \begin{bmatrix} \frac{L([2,0,1]+0.5[-1,1,1]) - L([2,0,1]-0.5[-1,1,1])}{2(0.5)(-1)} \\ \frac{L([2,0,1]+0.5[-1,1,1]) - L([2,0,1]-0.5[-1,1,1])}{2(0.5)(1)} \\ \frac{L([2,0,1]+0.5[-1,1,1]) - L([2,0,1]-0.5[-1,1,1])}{2(0.5)(1)} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{L([1.5,0.5,1.5]) - L([2.5,-0.5,0.5])}{(-1)} \\ \frac{L([1.5,0.5,1.5]) - L([2.5,-0.5,0.5])}{1} \\ \frac{L([1.5,0.5,1.5]) - L([2.5,-0.5,0.5])}{1} \end{bmatrix} = \begin{bmatrix} \frac{0.75 - 14.75}{(-1)} \\ \frac{0.75 - 14.75}{1} \\ \frac{0.75 - 14.75}{1} \end{bmatrix} = \begin{bmatrix} 14 \\ -14 \\ -14 \end{bmatrix}$$

# Outline

# Essential Conditions for SPSA

- Roughly speaking the conditions are:

A. Gain sequences: standard SA conditions:

$$a_k, c_k > 0, a_k, c_k \to 0 \text{ as } k \to \infty$$

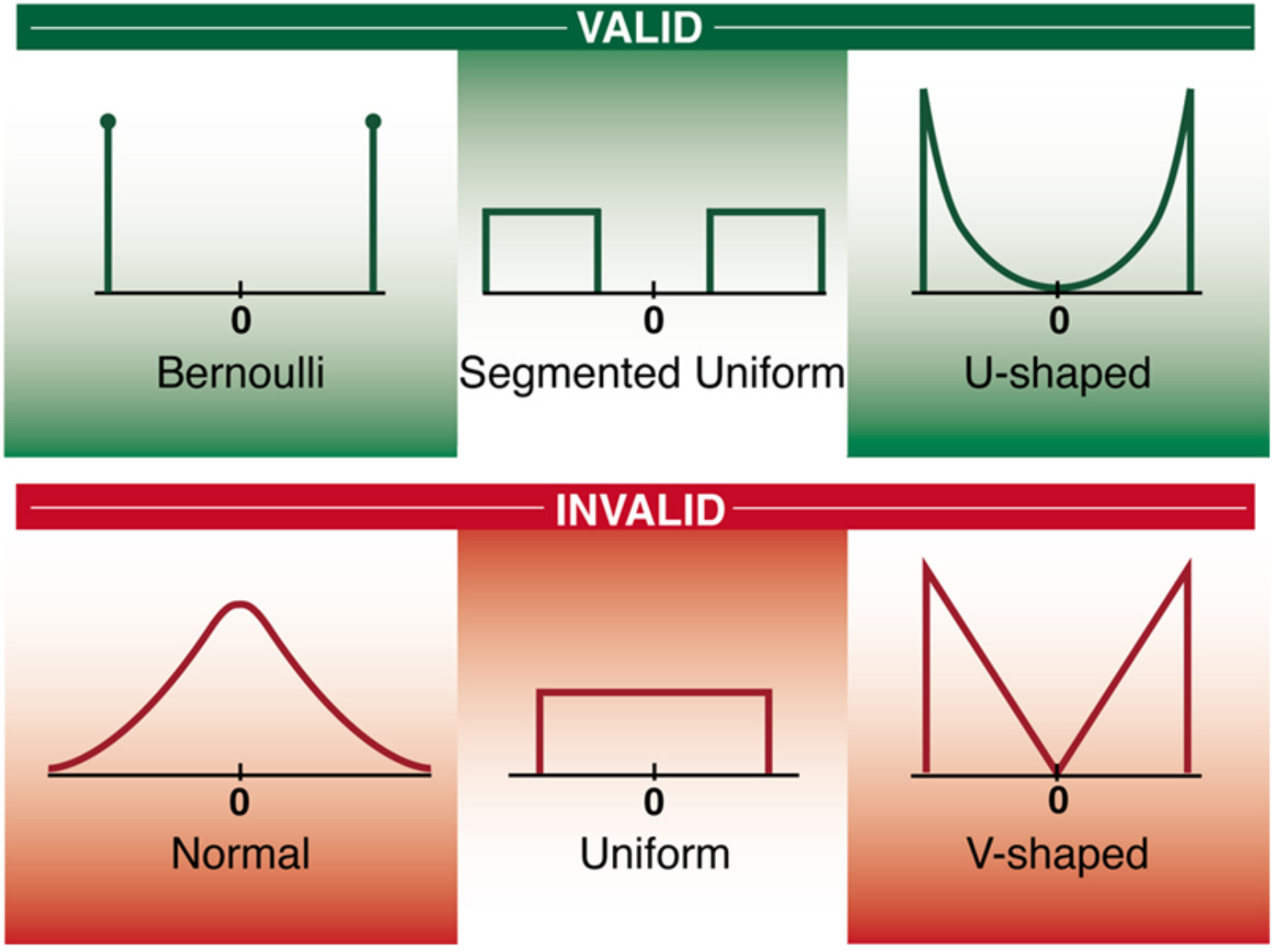
$$\sum_{k=0}^{\infty} a_k = \infty, \sum_{k=0}^{\infty} \left(\frac{a_k}{c_k}\right)^2 < \infty$$

(better to violate some of these gain conditions in certain practical problems; e.g., non-stationary tracking and control where $a_k = a > 0, c_k = c > 0, \forall k, i$)

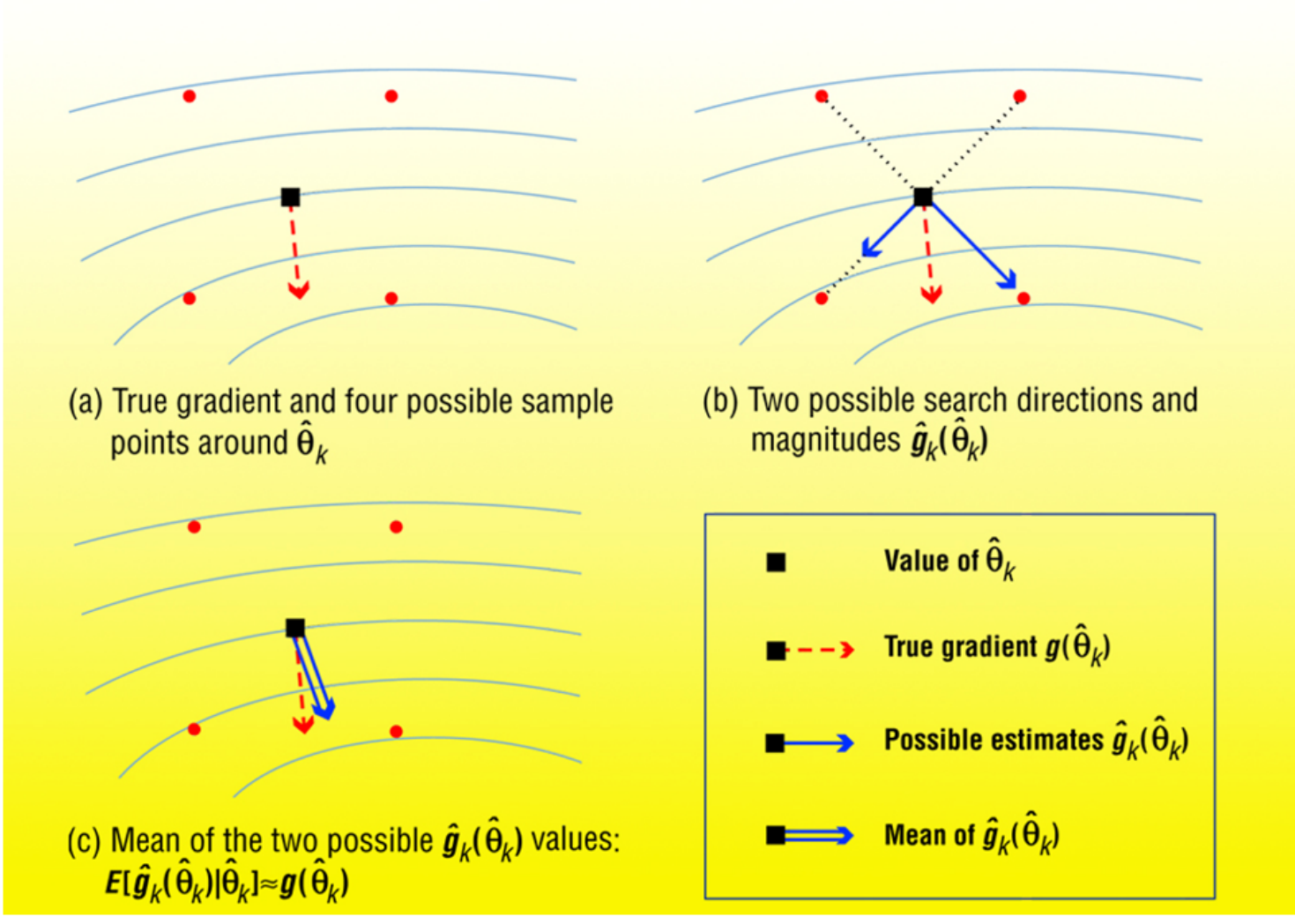
B. Choice of $\Delta_k$ distribution: For all $k$, $\Delta_k$ has independent components, symmetrically distributed around 0, and $E(\Delta_{ki}^2) < \infty, E(\Delta_{ki}^{-2}) < \infty$

  - Bounded inverse moments condition is critical (excludes $\Delta_{ki}$ being normally or uniformly distributed)
  - Symmetric Bernoulli $\Delta_{ki} = \pm 1$ (probability = $1/2$ for each outcome) is allowed; asymptotically optimal

# Valid and Invalid Perturbation Distributions



# Near-Unbiasedness for $\hat{g}_k(\cdot)$ ($p = 2, \Delta \sim$ Bernoulli)



# Efficiency Analysis

The critical cost in comparing relative efficiency of SPSA and FDSA is the number of loss function measurements $y(\cdot)$, not the number of iterations *per se*

- Can use asymptotic normality to analyze relative efficiency of SPSA and FDSA
- To achieve *same asymptotic MSE* (i.e. $\lim_{n\to\infty} E(\|\hat{\theta}_n - \theta^*\|^2)$)

$$\frac{\text{no. measurements } y(\theta) \text{ in SPSA}}{\text{no. measurements } y(\theta) \text{ in FDSA}} = \frac{1}{p} \quad (*)$$

- Result $(*)$ is main theoretical results justifying SPSA
  - Loss function measurements represent main cost (by far) – other costs are trivial

# Paraphrase of $(*)$ above:

Under reasonably general conditions, the SPSA and FDSA recursions achieve the same level of statistical accuracy for a given number of iterations even though SPSA uses only $1/p$ times the number of function evaluations of FDSA

– or –

One properly generated simultaneous random change of all variables in a problem contains as much information *for optimization* as a full set of one-at-a-time changes of each variable

# Outline

# Step-by-step implementation

Step 0. (Initialization and coefficient selection) Pick $\hat{\mathbf{d}}_0$. Set $k = 0$, $a_k = a/(k+1+A)^{\alpha}$, $c_k = c/(k+1)^{\gamma}$.

Practical effective values: $\alpha = 0.602, \gamma = 0.101$, $A = 10\%$ of max. number of expected iterations, $c =$ standard deviation of the noise in $y(\hat{\mathbf{d}}_0)$, $a/(1+A)^{0.602} \times \hat{g}_0(\hat{\mathbf{d}}_{0i})$ equal smallest of desired change magnitudes for $\hat{\mathbf{d}}_{0i}$ elements in the first iteration.

Step 1. (Generation of the simultaneous perturbation vector) Generate $p$-dimensional random perturbation vector $\Delta_k$; each $\Delta_{ki}$ - independently generated from zero-mean probability distribution. Practical choice: Bernoulli $\pm 1$ distribution, with $prob = 1/2$ for each $\pm 1$

Step 2. (Loss function evaluations) Calculate $y(\hat{\mathbf{d}}_k + c_k\Delta_k)$ and $y(\hat{\mathbf{d}}_k - c_k\Delta_k)$

# Step-by-step (cont.)

Step 3. (Gradient approximation) Calculate simultaneous perturbation approximation $\hat{g}_k(\hat{\mathbf{d}}_k)$. Sometimes useful to average several gradient approximations at $\hat{\mathbf{d}}_k$ using independent $\Delta_k$ (if noise effects $\varepsilon_k$ relatively large).

Step 4. (Update $\theta$ estimate) Calculate $\hat{\mathbf{d}}_{k+1} = \hat{\mathbf{d}}_k - a_k \hat{g}_k(\hat{\mathbf{d}}_k)$. Check for constraint violation and modify the update $\hat{\mathbf{d}}_{k+1}$ (map to the nearest valid point)

Step 5. (Iteration or termination) Terminate if small changes during several successive iterates or reached max. allowable number of iterations; if not, $k = k + 1$ and goto Step 1.

Simple enhancements possible to increase algorithm stability and/or speed convergence

- Block iteration $k$ (i.e. $\hat{\mathbf{d}}_{k+1} = \hat{\mathbf{d}}_k$) if $y(\theta_{k+1})$ is too much greater than $y(\theta_k)$ (requires extra loss measurement per iteration)
- Block iteration $k$ if $\|\hat{\theta}_{k+1} - \hat{\theta}_k\|$ is too large (does not require extra loss measurement)