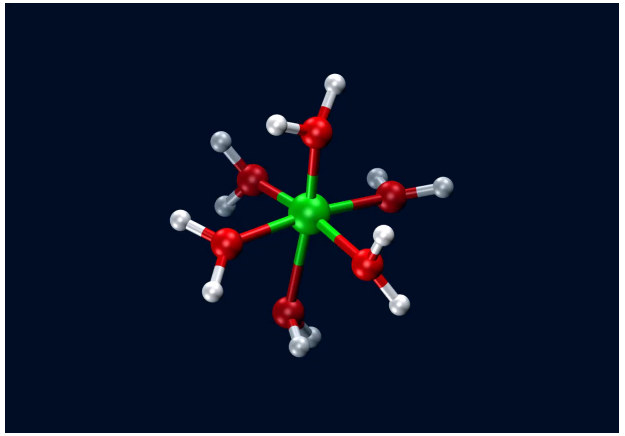
Introduction to Structural Optimization

Raymond Atta-Fynn (University of Texas, Arlington)

NSF Summer School on Disordered Materials Modeling

Summer 2019

attafynn@uta.edu



Outline

- What is structural optimization?
- Optimization algorithms
  - Steepest descent algorithm
  - Conjugate gradient algorithm
  - Monte Carlo method
- Closing remarks

What is structural optimization?

What is structural optimization?

Structural optimization is the process of using computer algorithms to minimize the energy of an atomistic structure.

Specifically, the atomic positions are displaced sequentially [following a set of rules] until a minimum energy state is reached.

- An atomistic structure is a set atoms with well-defined positions (or coordinates).
- Several properties of an atomistic structure are best described when the structure is in a minimum energy state; this is a major reason why structural optimization is performed

What is structural optimization?

- **Goal of structural optimization**: Minimize the total energy E of a set of N atoms with respect to the atomic positions $\{\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N\} \equiv \{(x_1, y_1, z_1), (x_2, y_2, z_2), , (x_N, y_N, z_N)\}$.
- E is a function of $\{\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N\}$, that is: $E \equiv E(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N)$. E depends on $3N$ variables.
- The condition for E to be a minimum is:

$$\nabla E(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N) = \mathbf{0} \qquad [1]$$

where ∇E is a *vector known as the gradient* of E given by:

$$\nabla E = \left(\frac{\partial E}{\partial \mathbf{r}_1}, \frac{\partial E}{\partial \mathbf{r}_2}, \ldots, \frac{\partial E}{\partial \mathbf{r}_N}\right) = \left(\underbrace{\frac{\partial E}{\partial x_1}, \frac{\partial E}{\partial y_1}, \frac{\partial E}{\partial x_1}}_{\frac{\partial E}{\partial \mathbf{r}_1}}, \underbrace{\frac{\partial E}{\partial x_2}, \frac{\partial E}{\partial y_2}, \frac{\partial E}{\partial x_2}}_{\frac{\partial E}{\partial \mathbf{r}_2}}, \ldots\ldots, \underbrace{\frac{\partial E}{\partial x_N}, \frac{\partial E}{\partial y_N}, \frac{\partial E}{\partial x_N}}_{\frac{\partial E}{\partial \mathbf{r}_N}}\right)$$

Thus equation [1] is equivalent to solving the $3N$ equations:

$$\frac{\partial E}{\partial \mathbf{r}_1} = 0; \frac{\partial E}{\partial \mathbf{r}_2} = 0; \ldots\ldots \frac{\partial E}{\partial \mathbf{r}_N} = 0$$

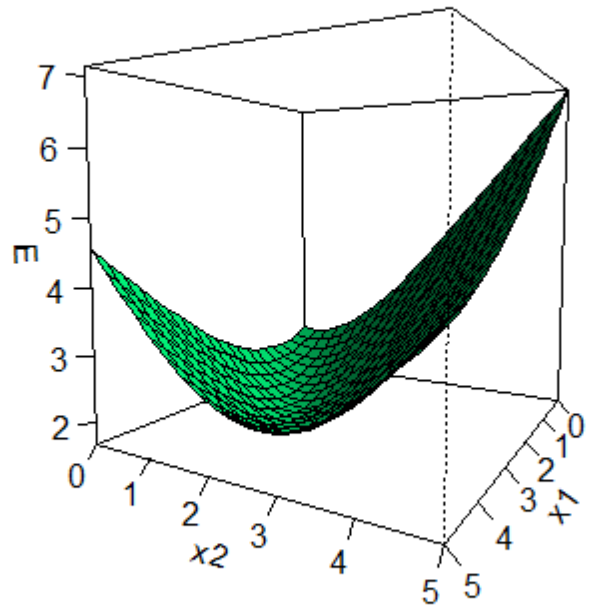
What is structural optimization?

- Example: Consider a system with $N = 2$ atoms. Suppose that atom 1 is located at position $\mathbf{r}_1 = (x_1, 0,0)$ and atom 2 is located at $\mathbf{r}_2 = (x_2, 0,0)$

- Suppose the (hypothetical) energy E of the 2-particle system is given by:

$$E(x_1, x_2) = {x_1}^2 + 2{x_2}^2 - 2x_1x_2 - 2x_1 - x_2 + 6$$

- A plot of E is shown below:



What is structural optimization?

- Given the energy of a 2-atom system: $E(x_1, x_2) = {x_1}^2 + 2{x_2}^2 - 2x_1x_2 - 2x_1 - x_2 + 6$

- We want to minimize E with respect to x_1 and x_2. The conditions are:

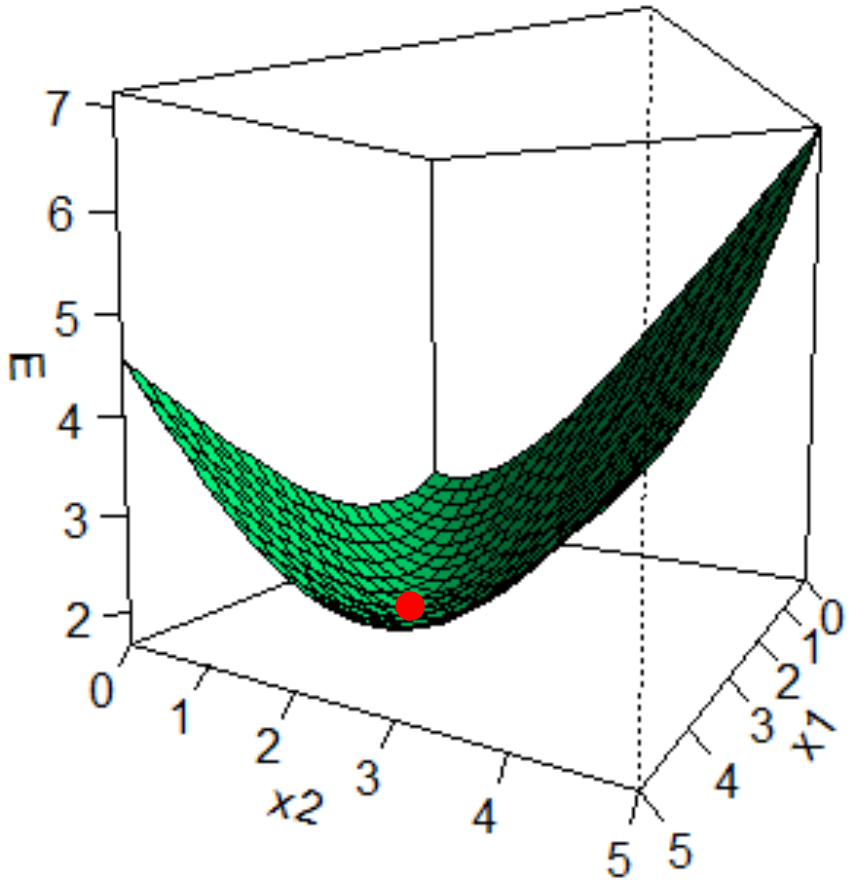
$$\frac{\partial E}{\partial x_1} = 2x_1 - 2x_2 - 2 = 0$$

$$\frac{\partial E}{\partial x_2} = 4x_2 - 2x_1 - 1 = 0$$

- The solutions to the equations are:

  $x_1 = 2.5$ and $x_2 = 1.5$;

- The minimum energy is $E(2.5, 1.5\ ) = 2.75$ [red dot is $(2.5, 1.5, 2.75)$]

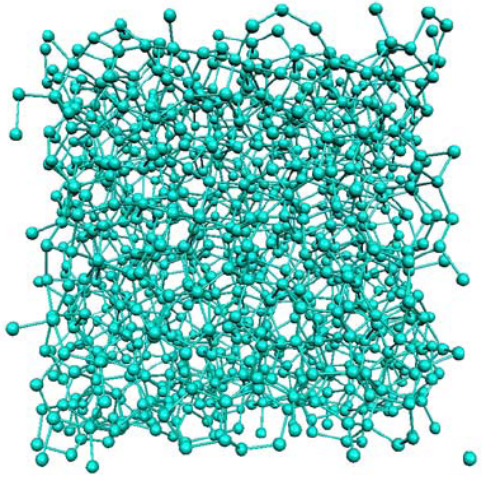


What is structural optimization?

- Now consider a more realistic scenario: begin with a random, high energy N-atom structure [$N = 1000$ in the pictures]

- The energy $E(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_{1000})$ of the system can be a bit complicated.
  - Optimizing the random structure implies: minimizing the total energy E by adjusting the positions of the atoms several times to yield an ordered (or semi-ordered), low energy structure.

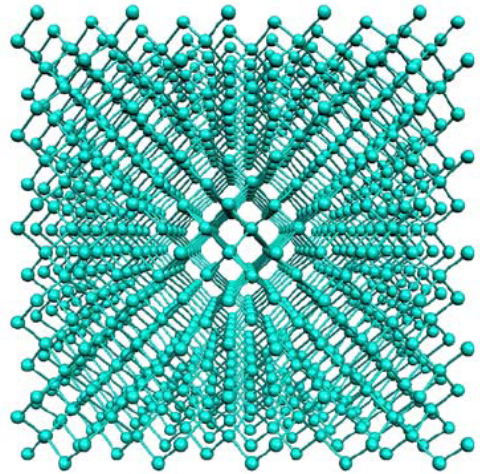
Random



Minimize E with respect to positions

$$\nabla E(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N) \approx \mathbf{0}$$

Ordered



Optimization Methods

Optimization problem: Given $E : R^N \rightarrow R$, minimize E over all possible values of $\vec{x} \in R^N$, where $\vec{x} = (\mathbf{r}_1, \mathbf{r}_2, .., \mathbf{r}_N)$.

Possible optimization methods

- Gradient based methods
  - Steepest descent method
  - Conjugate gradient method
  - Quasi-Newton methods [will not be discussed]
- Stochastic based methods
  - Metropolis Monte Carlo method
  - Particle swarm/population based methods [will not be discussed]

Optimization Methods

First a quick refresher on Taylor series: Suppose that x is a one-dimensional variable and x_0 is a constant. If the scalar function E is differentiable, then the Taylor expansion of $E(x+a)$ is

$$E(x+x_0) = E(x_0) + x\left.\frac{dE}{dx}\right|_{x=x_0} + x^2\left.\frac{d^2E}{dx^2}\right|_{x=x_0} + \cdots = E(x_0) + x\nabla E(x_0) + x^2\nabla^2 E(x_0) + \cdots$$

Now suppose that $\vec{x}$ is an N-dimensional variable vector and $\vec{x}_0$ is a constant vector. Then

$$E(\vec{x}+\vec{x}_0) = E(\vec{x}_0) + \vec{x}\cdot E(\vec{x}_0) + \vec{x}^{\mathrm{T}}\cdot\nabla^2 E(\vec{x}_0)\cdot\vec{x} + \cdots$$

Steepest Descent Algorithm

Steepest descent in 1-dimension

- Suppose we want to minimize a 1-dimensional function $E(x)$.
- Given an initial point x_0, the direction of steepest descent, i.e. direction of greatest change from x_0 is $-\nabla E(x_0)$.
- Key point: if we follow $-\nabla E$ in a *small enough steps*, E is *guaranteed to decrease*. To see this, consider the first order Taylor expansion of $E(x_0)$:

$$E(x_0 + \delta x) = E(x_0) + \delta x \nabla E(x_0)$$

where δx is a small change in x_0 [ignore $(\delta x)^2$ and higher powers]

- If δx is chosen to be $\delta x = -\alpha \nabla E(x_0)$, where α is a positive parameter called the step size, then the decrease in E follows:

$$E(x_0 + \delta x) = E(x_0) - \alpha |\nabla E(x_0)|^2 < E(x_0)$$

Steepest Descent Algorithm

Steepest descent in *N*-dimensions

- We want to minimize a *N*-dimensional function $E(\vec{x})$, where $\vec{x} \in R^N$ is a vector of dimension *N*.
- Given an initial point $\vec{x}_0$, the direction of steepest descent from $\vec{x}_0$ is the vector $-\nabla E(\vec{x}_0)$.
- Taylor expansion: the first order Taylor expansion of $E(\vec{x}_0)$:

$$E(\vec{x}_0 + \delta\vec{x}) = E(\vec{x}_0) + \delta\vec{x} \cdot \nabla E(\vec{x}_0)$$

- The choice of $\delta\vec{x} = -\alpha\nabla E(\vec{x}_0)$, where $\alpha > 0$, ensures that E decrease steadily toward the minimum.

Steepest Descent Algorithm

Steepest descent algorithm

Pick an initial point $\vec{x}_0$

$i \rightarrow 0$

$\alpha \rightarrow 0.5$

$\varepsilon = 0.000001$

loop

$-\nabla E(\vec{x}_i) \rightarrow \Delta$

if $|\Delta| < \varepsilon$ stop

$\alpha \rightarrow 2\alpha$

while $E(\vec{x}_i + \alpha\Delta) \geq E(\vec{x}_i)$

$\alpha \rightarrow \alpha/2$

end while

$\vec{x}_i + \alpha\Delta \rightarrow \vec{x}_{i+1}$

$i + 1 \rightarrow i$

end loop

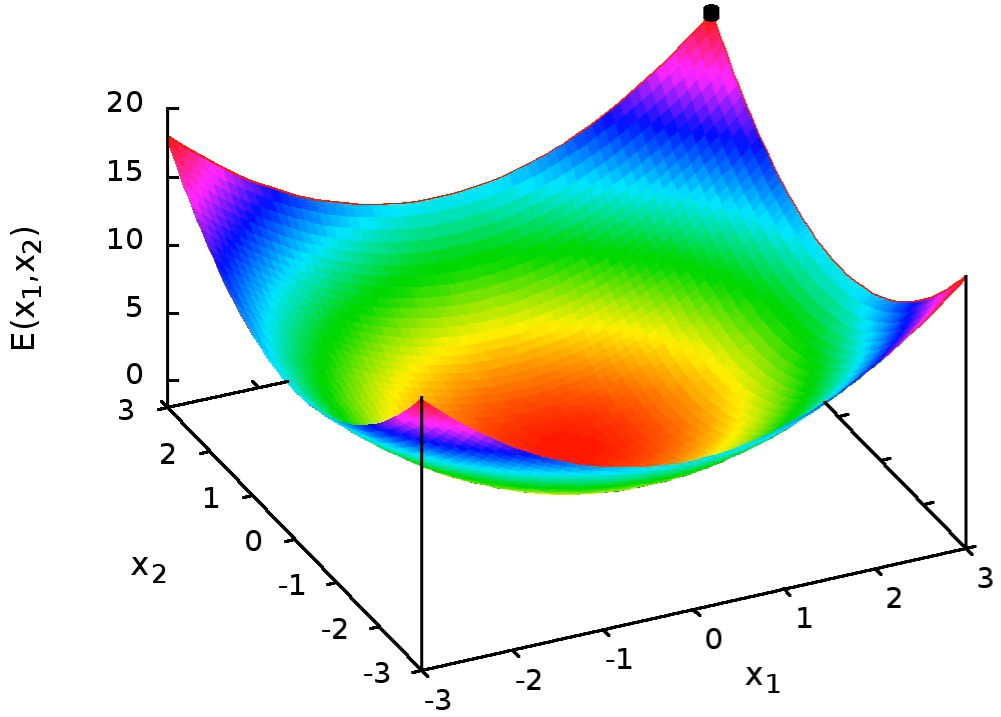
Steepest Descent Algorithm

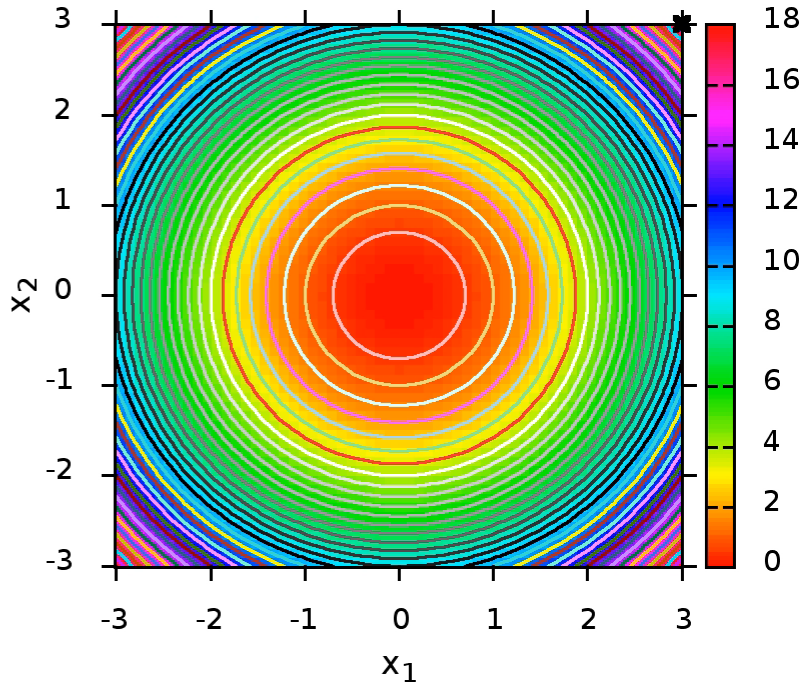
Steepest descent minimization: $E(x_1, x_2) = {x_1}^2 + {x_2}^2$

Raymond Atta-Fynn (UT Arlington), NSF Summer School 2019

Bowl Function: $E(x_1,x_2)={x_1}^2+{x_2}^2$; starting point for steepest descent minimization $(x_1,x_2)=(3,3)$

Steepest descent: step #1; $(x_1,x_2)=(3.000,3.000)$; E=18.000000





The minimum value of E is $E_{min} = 0$; this occurs at the location $(x_1, x_2) = (0,0)$

Left plot: 3D graph.

Right plot: corresponding projection onto the 2D plane spanned by x_1 and x_2 (contour plot).

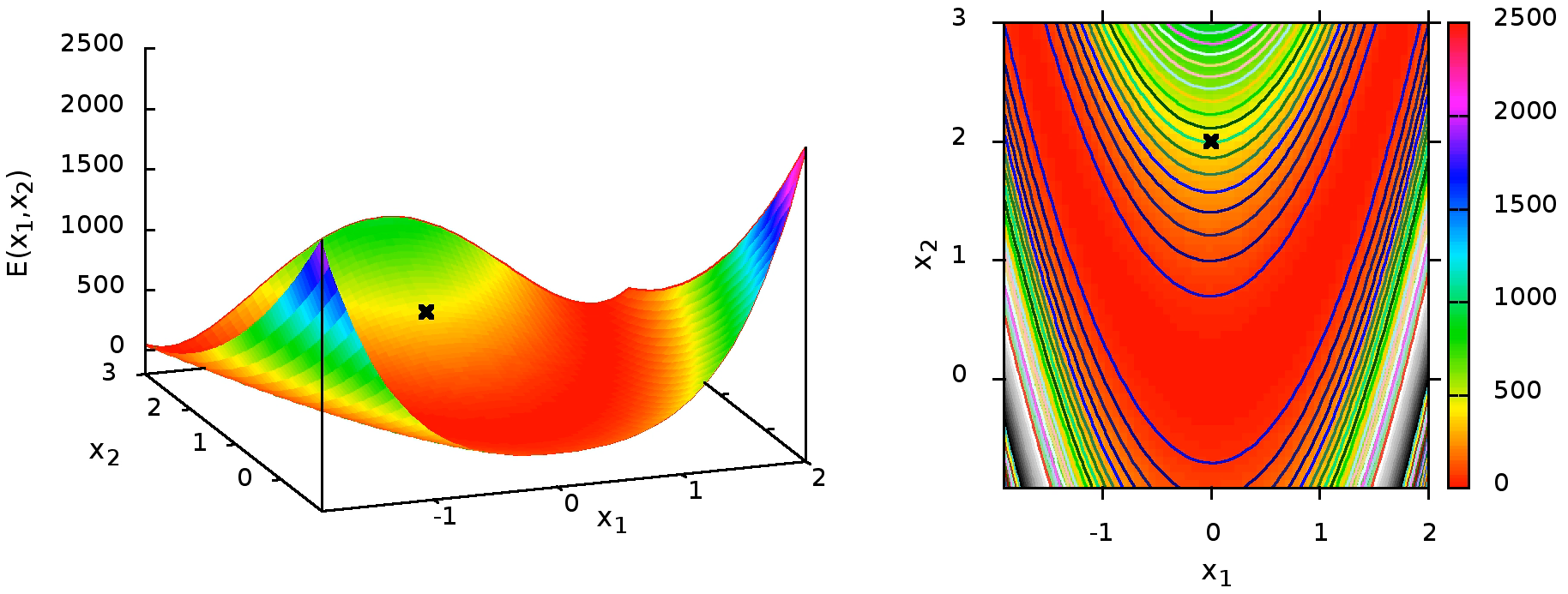
Steepest Descent Algorithm

Steepest descent minimization: Rosenbrock function $E(x_1, x_2) = (1 - x_1)^2 + 100(x_2{}^2 - x_1)^2$

Raymond Atta-Fynn (UT Arlington), NSF Summer School 2019

Rosenbrock function: $E(x_1,x_2)=(1-x_1)^2+100*(x_2-x_1{}^2)^2$; minimization starting point $(x_1,x_2)=(0,2)$

Steepest descent: step #1; $(x_1,x_2)=(0.0000,2.0000)$; E=401.0000



The minimum value of the Rosenbrock function is $E_{min} = 0$; this occurs at the location $(x_1, x_2) = (1,1)$

Left plot: 3D graph.

Right plot: Contour plot in the 2D plane spanned by x_1 and x_2.

Steepest Descent Algorithm

Steepest descent advantages

- Easy to implement.

- Many other methods switch to steepest descent when they do not make sufficient progress.

Steepest descent advantages

- **Picking** the step size α is a **bit of a dark art**. A small α makes the determination of the solution longer; a large α can make the algorithm worse

- The convergence of steepest descent algorithms close to the minimum can be quite slow.

Conjugate Gradient Algorithm

Conjugate gradient method

- The steepest descent minimization algorithm:

$$\vec{x}_{i+1} = \vec{x}_i - \alpha_i \nabla E(\vec{x}_i) = \vec{x}_0 - \alpha_1 \nabla E(\vec{x}_1) - \alpha_2 \nabla E(\vec{x}_2) - \cdots - \alpha_i \nabla E(\vec{x}_i)$$

$$\text{Until} \quad \nabla E(\vec{x}_{i+1}) \approx 0$$

where $i = 0, 1, 2, \ldots$

often finds itself taking steps in the same direction as earlier steps.

- This makes the steepest descent method woefully inefficient! It will be more efficient if a step is taken only once (but optimally). This lies at the heart of the **conjugate gradient method**.

- Original philosophy of conjugate gradient method: pick a set of directions $\{\vec{d}_0, \vec{d}_1, \ldots., \vec{d}_{N-1}\}$, known as **conjugate directions**, such that at only one step is taken along each direction $\vec{d}_i$ to reach the minimum.

Conjugate Gradient Algorithm

Conjugate gradient method: exact arithmetics

- For a given function $E(\vec{x})$ with *N* variables, **you are guaranteed to reach the minimum $E(\vec{x}^*)$ in exactly *N* steps if $E(\vec{x})$ is quadratic.**
- Specifically, given an initial point $\vec{x}_0$ and a set of *N* conjugate directions $\{\vec{d}_0, \vec{d}_1, \ldots, \vec{d}_{N-1}\}$, E attains a minimum at $\vec{x} = \vec{x}^*$ given by:

$$\vec{x}^* = \vec{x}_0 + \alpha_0\vec{d}_0 + \alpha_1\vec{d}_1 + \cdots + \alpha_{N-1}\vec{d}_{N-1} = \vec{x}_0 + \sum_{i=0}^{N-1} \alpha_i\vec{d}_i$$

$$\nabla E(\vec{x}^*) = 0$$

where α_i is the step size along the conjugate direction $\vec{d}_i$.

- Thus the employment of the conjugate gradient method boils to the determination of the directions $\{\vec{d}_0, \vec{d}_1, \ldots, \vec{d}_{N-1}\}$ and the step sizes $\{\alpha_0, \alpha_1, \ldots, \alpha_{N-1}\}$.
- Later on, we will present methods for determining $\{\vec{d}_i\}$ and $\{\alpha_i\}$.

Conjugate Gradient Algorithm

Conjugate gradient method: in practice

- Advantage: The conjugate gradient method is much faster than the steepest descent method; it requires much less steps to converge.

- Disadvantage:

(i) Its implementation is slightly more involving compared to the steepest descent method;

(ii) Due to rounding errors, the conjugate gradient method may take longer to converge

(iii) For highly disordered structures, the conjugate method can fail miserably.

- Implementation: We will present two iterative conjugate gradient methods that can be applied in practice; they are Fletcher–Reeves method and the Polak–Ribiere method.

Conjugate Gradient Algorithm

Fletcher–Reeves and Polak–Ribiere conjugate gradient algorithm

Step 1

Pick an initial point $\vec{x}_0$ and calculate $\vec{g}_0 = \nabla E(\vec{x}_0)$; set $\vec{d}_0 = -\vec{g}_0$

Step 2

For $i = 0, 1, 2, \ldots$:

(a) Find the value of α_i which minimizes $E(\vec{x}_i + \alpha_i \vec{d}_i)$

(b) Set $\vec{x}_{i+1} = \vec{x}_i + \alpha_i \vec{d}_i$ and compute the gradient $\vec{g}_{i+1} = \nabla E(\vec{x}_{i+1})$

(c) Test for convergence: if $|\nabla E(\vec{x}_{i+1})| < \varepsilon$ then stop $[\varepsilon = 10^{-4}]$.

(d) Compute the next conjugate direction $\vec{d}_{i+1}$ given by $\vec{d}_{i+1} = -\vec{g}_{i+1} + \beta_i \vec{d}_i$, where

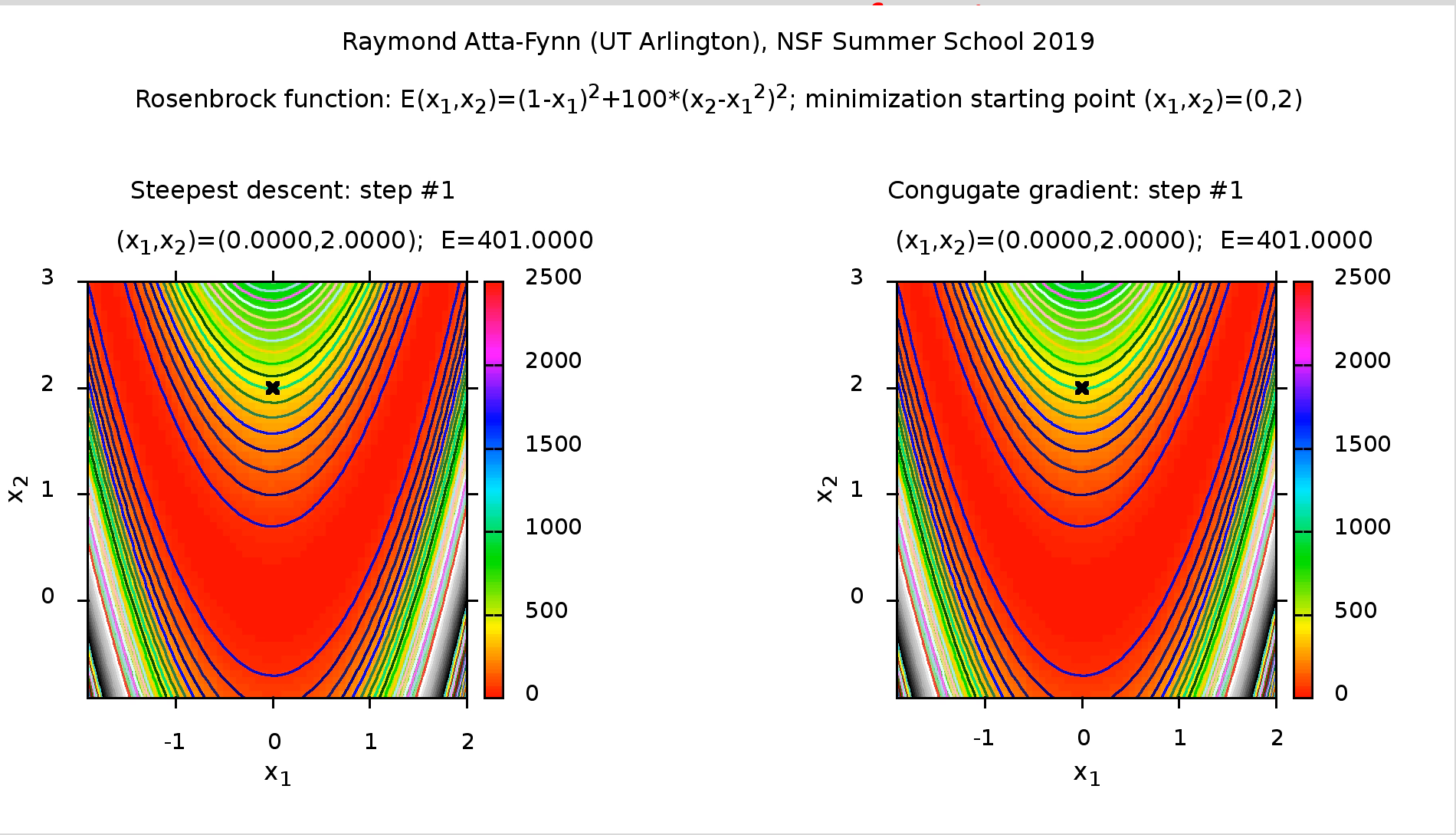
$$\beta_i = \frac{|\vec{g}_{i+1}|^2}{|\vec{g}_i|^2}$$ [Fletcher–Reeves method]

$$\beta_i = \frac{(\vec{g}_{i+1} - \vec{g}_i) \cdot \vec{g}_{i+1}}{|\vec{g}_i|^2}$$ [Polak–Ribiere method; preferred]

End for

Conjugate Gradient Algorithm

conjugate gradient method and the steepest descent method comparison: Rosenbrock



Left plot: contour plot of steepest descent minimization; it requires 3300 iterations to converge

Right plot: contour plot of conjugate gradient minimization; it requires only 15 iterations to converge

Monte Carlo Algorithm

Stochastic Optimization: Metropolis Monte Carlo Method (MMC)

- The MMC employs *random moves* to minimize a function; it is quite cheap [but not *necessarily efficient*] as no gradients are required. It is based on the concept of Markov chains.

- A sequence of $N+1$ successive *moves* or *events* or *states* $\vec{x}_0, \vec{x}_1, \vec{x}_2 \ldots \vec{x}_{N-1}, \vec{x}_N$ form a **Markov chain** if the present state $\vec{x}_N$ depends on only the immediate past state $\vec{x}_{N-1}$ regardless of all the other past states $\vec{x}_0, \vec{x}_1, \vec{x}_2 \ldots \vec{x}_{N-2}$:

$$P(\vec{x}_N | \vec{x}_0, \vec{x}_1, \vec{x}_2 \ldots \vec{x}_{N-1}) = P(\vec{x}_N | \vec{x}_{N-1})$$

- Here $P(B|A)$ is the conditional probability that the event B occurs given that A has occurred.

- In Monte Carlo language, the conditional probability $P(B|A)$ is also known as the *transition probability from event A to event B* (A is the present event, while B future event).

Monte Carlo Algorithm

Stochastic Optimization: Metropolis Monte Carlo Method (MMC)

- MMC operates on two key principles, namely, ***egordicity*** and ***detailed balance***.
- ***Egordicity:*** For a given system, a given state $\vec{x}_j$ can be reached from the state $\vec{x}_i$ in a finite number of steps.
- ***Detailed balance:*** For a given system, the average number of times the state $\vec{x}_j$ can be reached from the state $\vec{x}_i$ equals the average number of times $\vec{x}_i$ can be reached from $\vec{x}_j$.

Monte Carlo Algorithm

Stochastic Optimization: Metropolis Monte Carlo Method (MMC)

Philosophy behind the practical application of MMC minimization

(i) Suppose that an atomic structure begins in a state with coordinates $\vec{x}_0$ and energy $E(\vec{x}_0)$. We assign a fictitious temperature T to the system to measure its "hotness."

(ii) Conceptually, **MMC** operates by gradually cooling the system to from a "hot, unstable" state $\vec{x}_0$ to a "cold, minimum energy" state $\vec{x}_1$ using random atomic displacements. This "hot-to-cool" process in falls under a general minimization method known as *simulated annealing*.

(iii) The transition probability $P(\vec{x}_1|\vec{x}_0)$ from $\vec{x}_0$ to $\vec{x}_1$ is given by the **Metropolis criterion**:

$$P(\vec{x}_1|\vec{x}_0) = \min\left(1, e^{-\beta\Delta E}\right)$$

where $\beta = 1/(k_B T)$ and k_B is a *fundamental constant known as Boltzmann's constant*.

Optimization Methods

The Metropolis Monte Carlo algorithm

Step 1: Pick a fictious temperature T and begin in an initial state $\vec{x}_0$ with total energy $E(\vec{x}_0)$.

Step 2: Generate a new state $\vec{x}_1$ from $\vec{x}_0$ via *random* displacements of the atomic positions. Denote the total energy of $\vec{x}_1$ by $E(\vec{x}_1)$.

Step 3: Compute the energy difference $\Delta E = E(\vec{x}_1) - E(\vec{x}_0)$ and compute the transition probability from state $\vec{x}_0$ to state $\vec{x}_1$ as $P(\vec{x}_1|\vec{x}_0) = \min\left(1, e^{-\beta\Delta E}\right)$, where $\beta = 1/(k_B T)$.

Step 4: Generate a uniform random number r such that $0 \leq r < 1$.
(a)If $r < P(\vec{x}_1|\vec{x}_0)$, then replace $\vec{x}_0$ with $\vec{x}_1$ and $E(\vec{x}_0)$ with $E(\vec{x}_1)$ and go to step 2.

(a)If $r \geq P(\vec{x}_1|\vec{x}_0)$, then discard $\vec{x}_1$ and go to step 2.

Additional information: As the simulation proceeds, the temperature T is gradually reduced. Convergence is established by closely monitoring E.

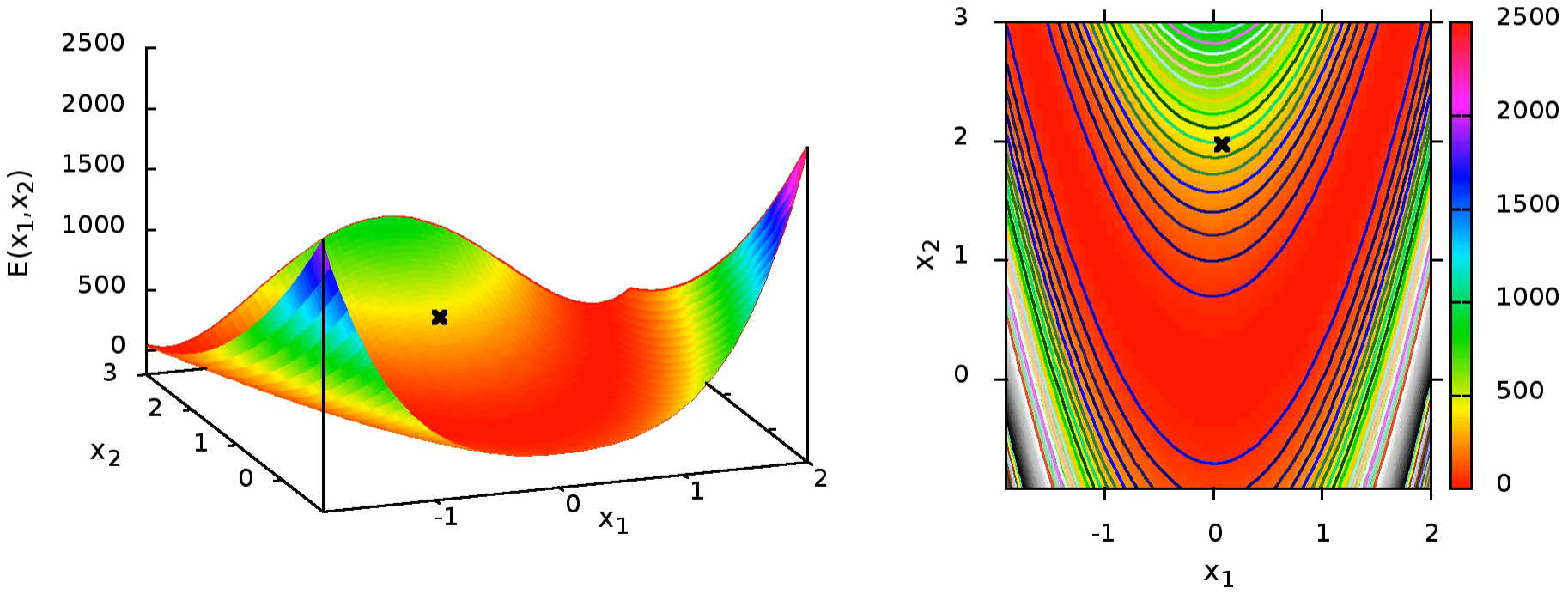
Monte Carlo Algorithm

Metropolis Monte Carlo method in action: minimizing the Rosenbrock function

Raymond Atta-Fynn (UT Arlington), NSF Summer School 2019

Rosenbrock function: $E(x_1,x_2)=(1-x_1)^2+100*(x_2-x_1^2)^2$; minimization starting point $(x_1,x_2)=(0,2)$

Monte Carlo with k_BT=8.617E-4: step #4; acceptance probability=0.2500; (x_1,x_2)=(0.0869,1.9767); E=357.9537



The minimum value of the Rosenbrock function is $E_{min} = 0$; this occurs at the location $(x_1, x_2) = (1,1)$

Left plot: 3D graph.

Right plot: Contour plot in the 2D plane spanned by x_1 and x_2.

Concluding Remarks

- Three minimization schemes, all of which are fairly easy to implement in computer codes, were presented: (i) steepest descent (ii) conjugate gradient (iii) Metropolis Monte Carlo

- The steepest descent and conjugate gradient methods are gradient-based (i.e. based on the evaluation of first partial derivative), while the Monte Carlo method does not require gradients.

- For practical applications, the conjugate gradient method is preferred; steepest descent can be used as a supplement in instances where the conjugate gradient method gets "stuck."

- For "quick and approximate results," the Monte Carlo method, which is the easiest to implement, can be employed.