Numerical Optimization

A review of common optimization algorithms

Jason DeBacker

October 16, 2024

Roadmap

Numerical Optimization

- We'll consider methods of numerical *minimization*
  - → Note that if you want to maximize a function (e.g., a likelihood function), you'll use the same methods because $max f(x) = \min -f(x)$
- Also, we'll focus on method s suitable for nonlinear problems
  - → Linear programming methods can be used to find the minimum of linear problems)

Numerical Optimization

- Optimization methods can generally be divided into two types:
  1. Gradient-based methods
     - Faster convergence to minimum
  2. Non-gradient based methods
     - More robust convergence to minimum
- Gradient-based methods are preferred if the function is smooth and you have a good initial guess
- When your function isn't smooth, optimization becomes much more difficult

Roadmap

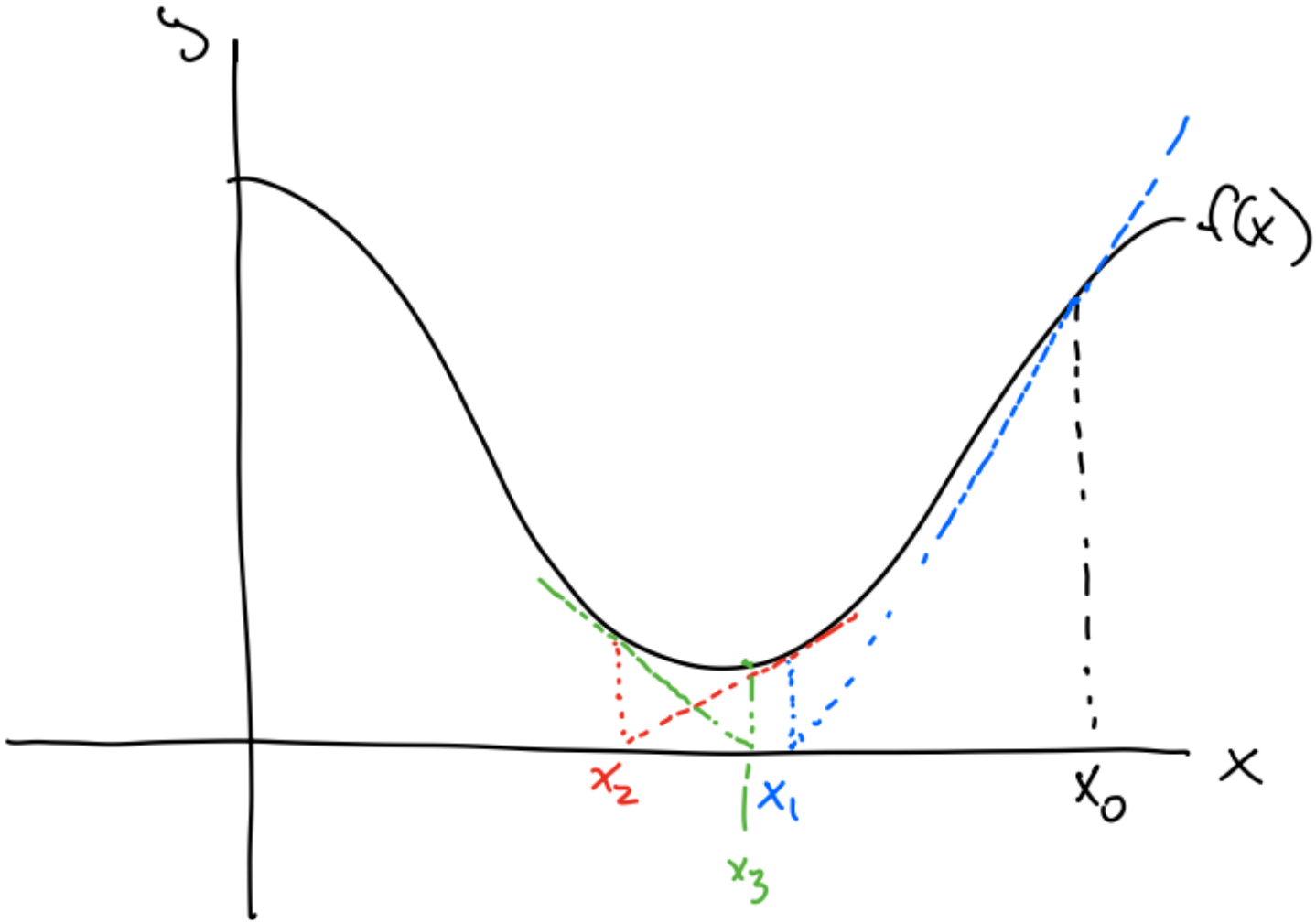
Newton's method: Algorithm

1. For an initial guess, x_0, let $q(x)$ be a Taylor expansion of degree 2 of $f(x)$ around x_0.

$$q(x) \equiv f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2} f''(x_0)(x - x_0)^2 \tag{1}$$

2. Find the value of x that minimizes $q(x)$. Call this x_1
   - $\rightarrow$ x_1 solves $q'(x) = 0$
   - $\rightarrow$ $q'(x) = f'(x_0) + f''(x_0)(x - x_0)$
   - $\rightarrow$ $\implies$ x_1 solves $f'(x_0) + f''(x_0)(x - x_0) = 0$
   - $\rightarrow$ $\implies$ $x_1 = x_0 - \frac{f'(x_0)}{f''(x_0)}$
3. Form a Taylor expansion around x_1 and repeat, from Step (2) to find $x_2, x_3, ...x_k$
4. Repeat until convergence: $|x_{k+1} - x_k| < \varepsilon$

Newton's method: In a Picture



Newton's method: Convergence

- Rate of convergence is quadratic
- Can be sensitive to initial guess
- Often the method of choice if feasible for your problem
  - → i.e., if have smooth function

Gradient Descent Method

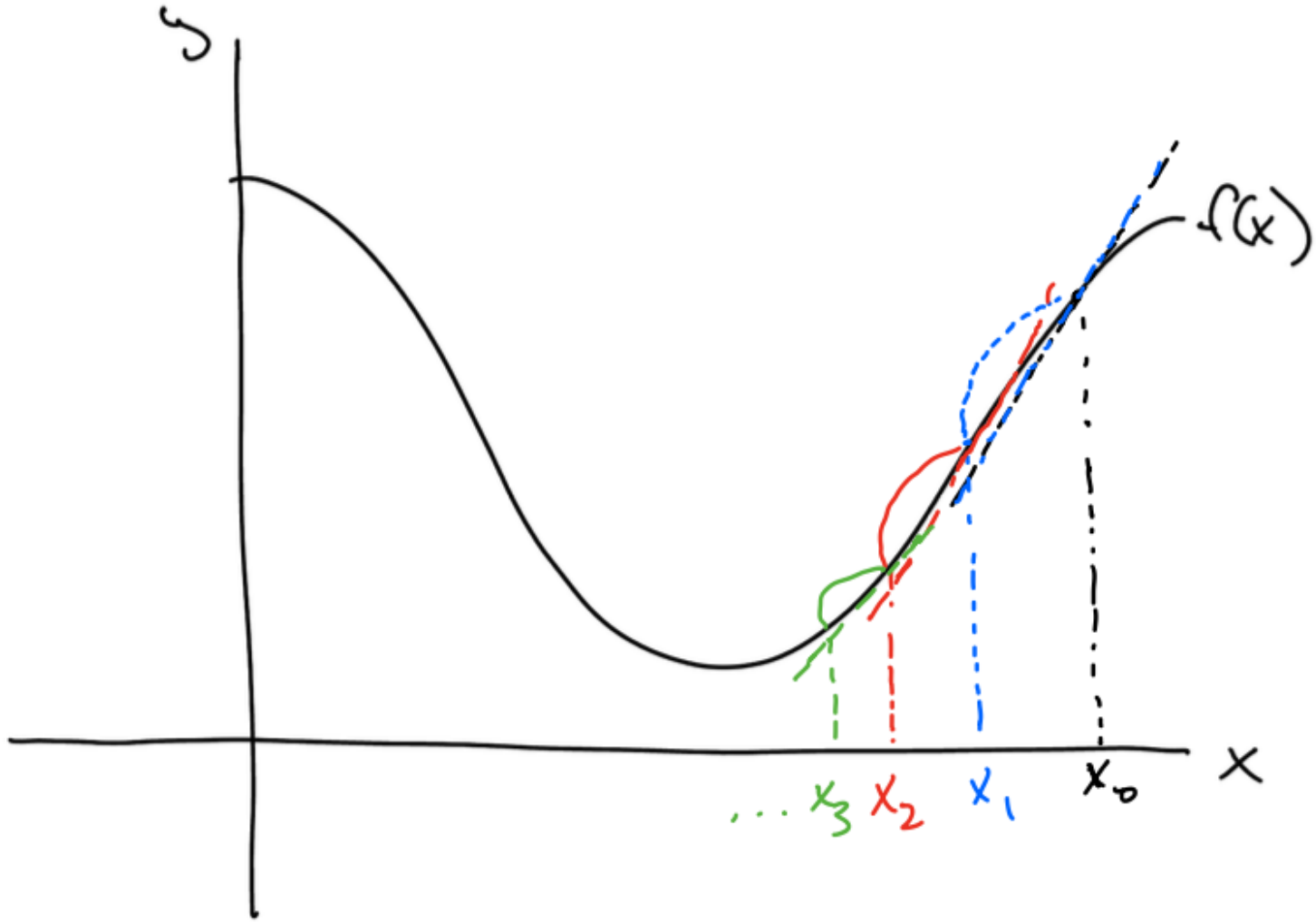
- Idea: The gradient of the function points in the direction of ascent in x
- To minimize, go in the opposite direction
  - $\rightarrow$ Going further might get you to solution more quickly
  - $\rightarrow$ But larger steps might have you jump over solution and have to backtrack

Gradient Descent Method: Algorithm

1. For an initial guess, x_0 find the gradient, $\bigtriangledown f(x_0)$
2. Find x_1 as $x_1 = x_0 - \beta \bigtriangledown f(x_0)$

   $\rightarrow$ β is a parameter of the search - tradeoffs involved in it's choice
3. Repeat from Step (a) to find $x_2, x_3, ..., x_k$ until convergence, $|x_{k+1} - x_k| < \varepsilon$

Gradient Descent Method: In a Picture



Gradient Descent Method: Convergence

- Linear convergence

Golden Ratio search

- The Golden Ratio search method finds the minimum of a function by narrowing down the state space in a specific way
- In particular, it will break up the state space into intervals of equal length

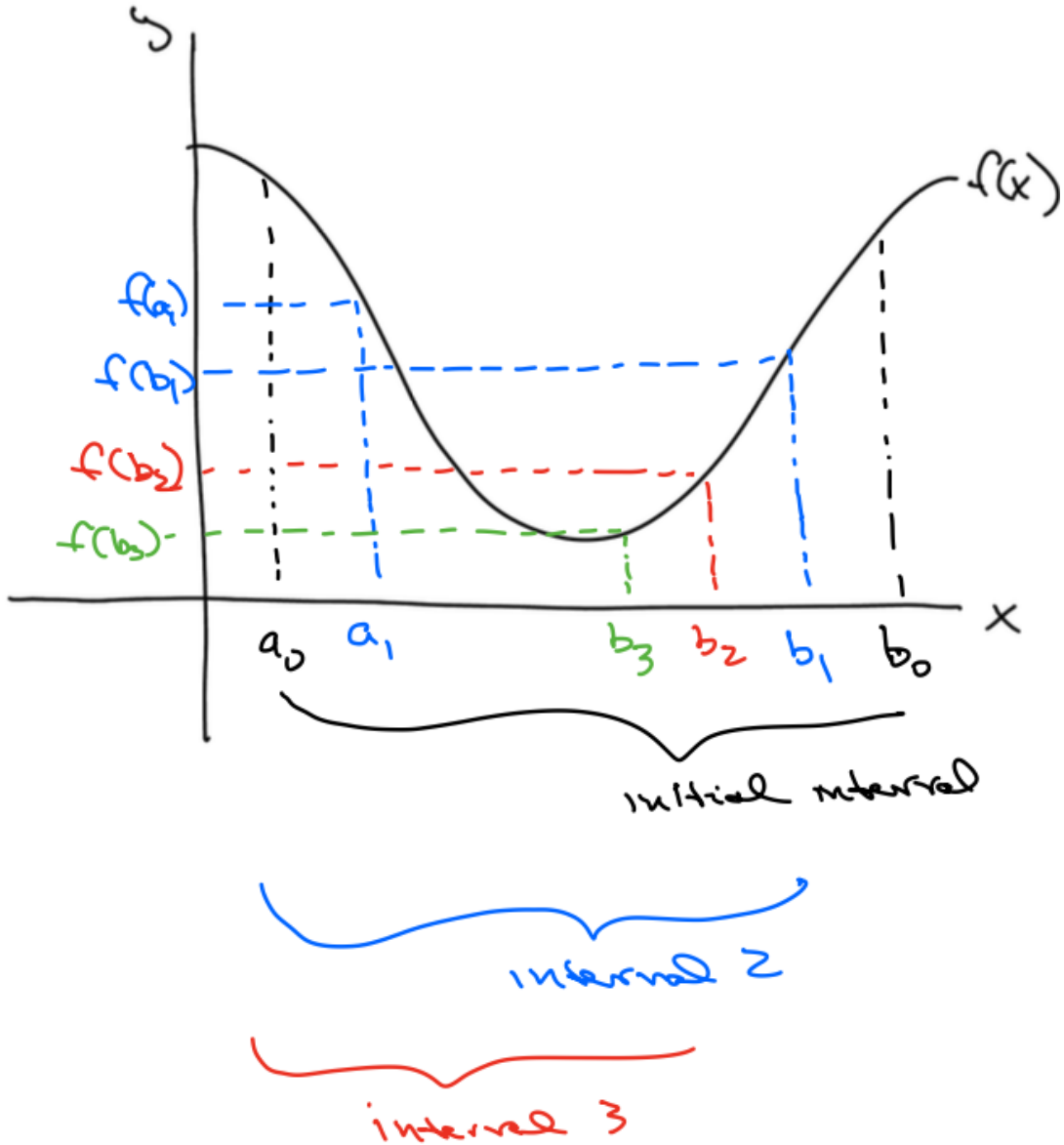
Golden Ratio search: Algorithm

- Algorithm to find the minimum of $f(x)$:
  1. Choose two points, a_1 and b_1 in the interval over which the problem is defined, $[a_0, b_0]$
     - The key to the Golden Ratio search method is in how these are chosen.
     - a_1 and b_1 are chosen so that the "outer intervals" are the same size. i.e., $a_1 - a_0 = b_0 - b_1 = \rho(b_0 - a_0)$
     - $\implies \rho = \frac{b_0 - a_0}{b_0 - b_1} = \frac{3-\sqrt{5}}{2} \simeq 0.381 = 1 - \frac{1}{\phi}$ where $\phi =$ the Golden Ratio
  2. Evaluate $f(a_1)$ and $f(b_1)$
  3. If $f(a_1) < f(b_1)$, then we know the minimum is in the interval $[a_0, b_1]$ (and if $f(a_1) > f(b_1)$, we'd know the min is in the interval) $[a_1, b_0]$)

Golden Ratio search: Algorithm (cont'd)

- Algorithm to find the minimum of $f(x)$:
  4. Choose another point in (a_0, b_1), call this point b_2
     - As before, we choose this point so that outer intervals are the same length
     - i.e., $a_1 - a_0 = b_2 - b_1$
  5. Evaluate $f(b_2)$
  6. Compare $f(b_2)$ to $f(a_1)$ to decide which side of b_2 the next point will be on.
  7. Repeat from Step (4) until convergence, $|b + n + 1 - a_n| < \varepsilon$

Golden Ratio search: In a Picture



Golden Ratio search: Convergence

- Convergence is linear
  - → The intervals of uncertainty shrink at a rate of $1 - \rho$. i.e., the size of the interval at iteration k is $(1 - \rho)^k (b_0 - a_0)$
- This is a robust method - it will always converge
- Golden Ratio search is similar to the Bisection method
  - → Both converge linearly
  - → Golden Ratio is more efficient in that it requires fewer function evaluations (one per iteration rather than two – see Step (e) above)

Brent's method

- Brent's method is a hybrid method, a combination of gradient and non-gradient methods
- This method combines inverse quadratic interpolation (IQI) with a search method (e.g., the Golden Ratio search method)
  - → At it's best, it converges super-linearly (by using IQI)
  - → At worst, it converges linearly (by using search method)
  - → By being a hybrid, it retains robustness

Brent's method: Algorithm

1. Approximate the function to be minimized with a quadratic function
2. This function can be evaluated at 3 points, a, b, and c, where $a < b < c$ and a and c bracket the minimum
3. The minimum of this parabola is given by:

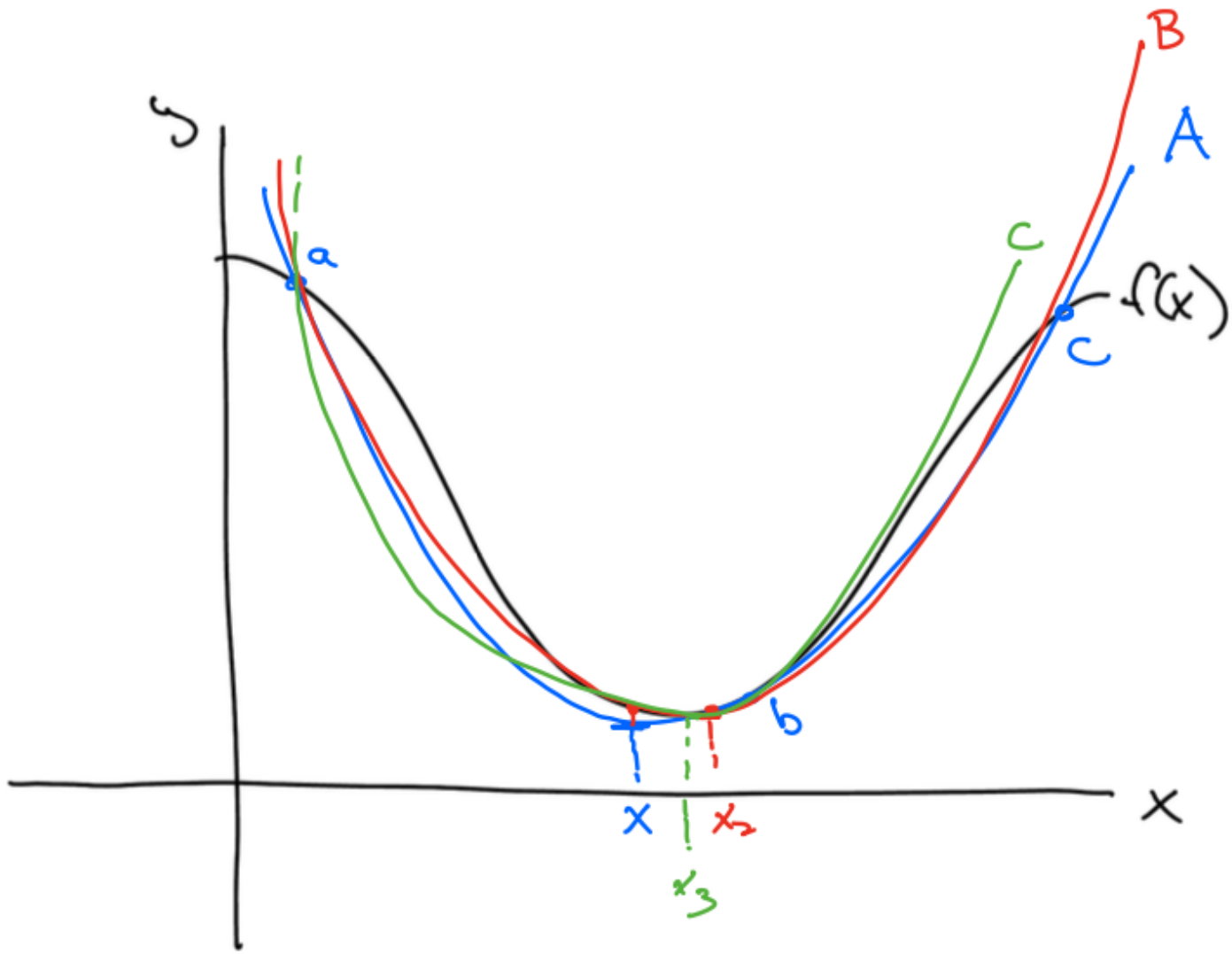
$$x = b - \frac{1}{2}\frac{(b-a)^2[f(b)-f(c)] - (b-c)^2[f(b)-f(a)]}{(b-a)[f(b)-f(c)] - (b-c)[f(b)-f(a)]}$$

4. If $a < x < b$, do steps (2) and (3) on a new interval with 3 points = a, x, b
5. If $b < x < c$, do steps (2) and (3) on a new interval with 3 points = b, x, c

Brent's method: Algorithm (cont'd)

6. Step (3) could send x away from the true minimum - thus check that x in the interval (a, c)
   - $\rightarrow$ If x outside this interval, switch to Golden Ratio method
   - $\rightarrow$ If not, go back to (2) with the 3 new points
7. Repeat until convergence, $|c_k - a_k| < \varepsilon$, where c_k and a_k are the bounds on the interval at iteration k

Brent's method: In a Picture



Brent's method: Convergence

- Converges at a rate of about 1.83 ($> 1, < 2 \implies$ super-linear convergence)

Roadmap

Numerical Optimization in Multiple Dimensions

- Solving non-linear problems in multiple dimensions is hard
- There is no guarantee of convergence to the true global min in any solution method
- It’s all about trade offs - and thus it’s important to understand how each method works

Newton's Method: Algorithm

- A gradient based method
- Like Newton method in one dimension, but now deal with vectors/matrices of first/second order conditions.
- Still find x_{k+1} by finding minimum of a degree-two Taylor approximation of $f(x)$. Thus:

$$x_{k+1} = x_k - (D^2 f(x_k))^{-1} Df(x_k)^T \tag{2}$$

- Algorithm is the same as one dimension otherwise - iterate on the above until convergence starting from some initial guess, x_0

Newton's Method: Potential Issues

- May not converge (the initial guess is important here)
- $D^2 f(x)$ may not be positive definite, and thus inverse may not exist
- $(D^2 f(x))^{-1} Df(x)^T$ may be expensive, unstable, or otherwise difficult to compute
  - $\rightarrow$ The curse of dimensionality applies here.
  - $\rightarrow$ If the problem has n dimensions, then the Hessian has n^2 elements.
  - $\rightarrow$ Thus to compute the Hessian at each iteration requires computing $n^2/2$ second derivatives.

The Method of Steepest Descent

- An extension of the gradient decent method - where choose optimal step size
- Idea: The gradient of the function points in the direction of ascent in x
- To minimize, go in the opposite direction
- Question - how far to go?
  - → Going further might get you to solution more quickly
  - → But larger steps might have you jump over solution and have to backtrack

The Method of Steepest Descent: Algorithm

1. For an initial guess, x_0 find the gradient, $\bigtriangledown f(x_0)$
2. Find x_1 as $x_1 = x_0 - \alpha_0 \bigtriangledown f(x_0)$
3. But, what is α_0 (or what should it be)?
   - $\rightarrow$ Find optimal step, α_0 by solving for the α that minimizes $f(x)$ along the line $x = x_0 - \alpha \bigtriangledown f(x_0)$:

$$\min_{\alpha} f(x_0 - \alpha \bigtriangledown f(x_0)) \tag{3}$$

   - $\rightarrow$ This is a one-dimensional minimization problem
   - $\rightarrow$ This problem can be solved by Brent's Method, Golden Ratio Search, or other algorithm.
4. Repeat from Step (1) to find $x_2, x_3, ..., x_k$ until convergence, $|x_{k+1} - x_k| < \varepsilon$

The Method of Steepest Descent

- Convergence:
  - → Can converge very quickly for some problems
  - → Slower for others
- Gradient Descent Method
  - → Solving for the optimal α_k at each iteration can be costly
  - → Thus we might not want to solve for the optimal step size, but rather just use a rule of thumb to chose the step size
  - → This method is the Gradient Descent Method (as opposed to the Method of Steepest Descent)

Conjugate Gradient Method

- A gradient based method
- This method is midway between Newton method and Gradient Descent
  - → Newton method gives faster convergence
  - → Gradient descent is less expensive to compute per step
- It works well to find local minima when the function is approximately quadratic near the minimum
- Similar to the Method of Steepest Descent

Conjugate Gradient Method: Algorithm

1. Begin at point x_0 and find the gradient at this point, $\bigtriangledown f(x_0)$
2. Move in the opposite direction the gradient points (since it points uphill and you want to find the min), $\Delta x_0 = - \bigtriangledown f(x_0)$
   - $\rightarrow$ We'll call $s_0 \equiv \Delta x_0$ the conjugate of the gradient, $\bigtriangledown f(x_0)$
   - $\rightarrow$ How far to move? Solve this as solve for in the Method of Steepest Descent
   - $\rightarrow$ Find α_0 as $\min_\alpha f(x_0 + \alpha \Delta x_0)$
3. Find point move to as $x_1 = x_0 + \alpha_0 \Delta x_0$
4. Calculate the opposite direction of the gradient at x_1:

   $\Delta x_1 = - \bigtriangledown f(x_1)$

Conjugate Gradient Method: Algorithm (cont'd)

5. Compute β_1
   - $\rightarrow$ There are several ways to do this.
   - $\rightarrow$ Some examples:

5.1 Fletcher-Reeves:

$$\beta_n^{FR} = \frac{\Delta x_n^T \Delta x_n}{\Delta x_{n-1}^T \Delta x_{n-1}} \tag{4}$$

5.2 Polak-Ribière:

$$\beta_n^{PR} = \frac{\Delta x_n^T (\Delta x_n - \Delta x_{n-1})}{\Delta x_{n-1}^T \Delta x_{n-1}} \tag{5}$$

5.3 Hestenes-Stiefel:

$$\beta_n^{HS} = \frac{\Delta x_n^T (\Delta x_n - \Delta x_{n-1})}{s_{n-1}^T (\Delta x_n - \Delta x_{n-1})} \tag{6}$$

5.4 Dai-Yuan:

$$\beta_n^{DY} = \frac{\Delta x_n^T \Delta x_n}{s_{n-1}^T (\Delta x_n - \Delta x_{n-1})} \tag{7}$$

Conjugate Gradient Method: Algorithm (cont'd)

6. Update conjugate direction: $s_1 = \Delta x_1 + \beta_1 s_0$
7. Solve for optimal step size, $\alpha_1 = arg\min_\alpha f(x_1 + \alpha s_1)$
8. Update the position: $x_2 = x_1 + \alpha_1 s_1$
9. Repeat Steps (4)-(8) until convergence, $|x_{k+1} - x_k| < \varepsilon$

BFGS

- A gradient-based method
- The longer name is the Broyden-Fletcher-Goldfarb-Shanno Method
- This method is similar to Newton method, but rather than computing the Hessian at each iteration (which can be expensive), it only computes the Hessian initially and then approximates it after that
- Given this approximation, it has a slower rate of convergence than Newton method, but the cost at each iteration is lower.

BFGS: Algorithm

- Recall that Newton method had:

$$x_{k+1} = x_k - (D^2 f(x_k))^{-1} Df(x_k)^T \tag{8}$$

- With BFGS, we have this exactly for the first iteration. i.e.,

$$x_1 = x_0 - (D^2 f(x_0))^{-1} Df(x_0)^T \tag{9}$$

BFGS: Algorithm (cont'd)

- But for all following iterations we have:

$$x_{k+1} = x_k - A_k^{-1} Df(x_k)^T \tag{10}$$

  → where

$$A_{k+1} = A_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{A_k s_k s_k^T A_k}{s_k^T A_k s_k} \tag{11}$$

  → $y_k = Df(x_{k+1})^T - Df(x_k)^T$

  → $s_k = x_{k+1} - x_k$

- A nice property of this is that the A_k is guaranteed to be positive definite
- Otherwise, this is just like Newton's method

Gauss-Newton Method

- A gradient-based method
- This is a special method for nonlinear least squares problems, a form of Newton method
- e.g., consider a problem where you have a set of residuals, $r_i(x)$
  - $\rightarrow$ The function you are minimizing is of the form

$$f(x) = \sum_{i=1}^{M} r_i(x)^2 \tag{12}$$

Gauss-Newton Method (cont'd)

- where each r_i is a smooth function and where M is the number of observations or moment conditions
- Be careful with the notation here - x are the parameters, i denotes observations (data)
- So $f(x)$ is a sum of squared residuals
- A function of this form can be solved more easily than a more general nonlinear function
- Express the problem in matrix notation:

$$f(x) = r(x)^T r(x) \tag{13}$$

Gauss-Newton Method (cont'd)

- The derivative of $f(x)$ is $Df(x) = 2r(x)^T J(x)$
- where $J(x)$ is the Jacobian matrix for r (if x has n dimensions, then J is $M \times n$)
- $J(x)_{i,j} = \frac{\partial r_i}{\partial x_j}$
- The Hessian is given by

$$D^2 f(x) = 2 \left(J(x)^T J(x) + \underbrace{\sum_{i=1}^{M} r_i(x) \overbrace{D^2 r_i(x)}^{\text{Deriv of Jacobian}}}_{Q(x)} \right) \tag{14}$$

Gauss-Newton Method (cont'd)

- Newton method implies:

$$x_{k+1} = x_k - (J(x_k)^T J(x_k) + Q(x_k))^{-1} J(x_k)^T r(x_k) \tag{15}$$

- If $r(x)$ is close to 0, then $Q(x) \simeq 0$ and so we can drop it and find

$$x_{k+1} = x_k - (J(x_k)^T J(x_k))^{-1} J(x_k)^T r(x_k) \tag{16}$$

- We can iterate on this equation until convergence, having to compute the Jacobian, but not the Hessian, at each step

Gauss-Newton Method (cont'd)

- Gauss-Newton is standard for nonlinear least squares problems
- If $Q(x) \simeq 0$ you get close to quadratic convergence
- If $Q(x)$ far from 0, you might not get convergence at all (initial guesses are important)
- Also, if $Q(x)$ is large, $J^T J$ may not be positive definite or even invertible. To fix this, one can use the Levenberg-Marquart method
  - $\rightarrow$ To do this, replace $J^T J$ with $J^T J + \mu I$ where $\mu > 0$ is a dampening factor

The Nelder-Mead Method

- A non-gradient based method
- This function is particularly useful for multidimensional functions that aren't differentiable

The Nelder-Mead Method: Algorithm

1. Choose an initial simplex of $n+1$ points (where n is the number of dimensions of x)
2. Order according to the values at the vertices:
   $f(x_1) \leq f(x_2) \leq f(x_3) \leq ... \leq f(x_n) \leq f(x_{n+1})$
3. Calculate the centroid of all points except x_{n+1}. Call this x_o
4. Compute the reflected point, $x_r = x_o + \alpha(x_o - x_{n+1})$, where $\alpha > 0$
   - → If x_r is better than the second worst, but not better than the best (i.e., $f(x_1) \leq f(x_r) < f(x_n)$), then:
     - Replace the worst point, x_{n+1} with x_r and go to Step (b)
   - → If x_r is the best point (i.e., $f(x_r) < f(x_1)$) then:
     - Compute the expanded point: $x_e = x_o + \gamma(x_r - x_o)$ with $\gamma > 1$
     - If $f(x_e) < f(x_r)$ then replace x_{n+1} with x_e and go to Step (b)
     - Else, replace x_{n+1} with x_r and go to (b)

The Nelder-Mead Method: Algorithm (cont'd)

4. Computing the reflected point...
   - → If x_r is at least as bad as the second worst point (i.e., $f(x_r) \geq f(x_n)$):
     - Compute the contraction point: $x_c = x_o + \rho(x_{n+1} - x_o)$ with $0 < \rho \leq 0.5$
     - If $f(x_c) < f(x_{n+1})$, replace x_{n+1} with x_c and go back to Step (b)
     - Else, shrink the space where searching by computing a new simplex, replacing each point except x_1 with $x_i = x_1 + \sigma(x_i - x_1)$ (where $0 < \sigma < 1$) and go to Step (b)
5. Repeat until convergence, $|x_{n+1} - x_1| < \varepsilon$

The Nelder-Mead Method: Notes

- Standard values for the reflection, expanding, contraction, and shrink coefficients are $\alpha = 1, \gamma = 2, \rho = 0.5, \sigma = 0.5$
- The initial simplex is important. If it's too small, the algorithm can get stuck there.

Powell's Method

- A non-gradient based method
- Useful because it does not require that the function be differentiable
- Essentially breaks up the solution to an n-dimensional problem into n single dimensional minimization problems at each step, each solved via Golden Ratio search or Brent's Method
- Uses solution at each step to see the n directions in which minimums are found in the next step

Roadmap

Global Optimizers

- As noted: Solving non-linear problems in multiple dimensions is hard and no guarantee of convergence to the true global min in any solution method
- There exist a few *heuristics* for finding a global optimum
  - →
  - → This means these are rules of thumb that will hopefully help you find the solution, but that is not mathematically proven to do so.
- We'll discuss two here, both of which try to avoid getting stuck in a local optimum

Simulated annealing

- A non-gradient method (though one could use gradients with this)
- It's a probabilistic method to find a global maximum (or minimum)
- The name derives from the process of annealing in metallurgy, which is the heating and cooling of metals to share and strengthen them

Simulated annealing: Algorithm

1. Start at an initial point, x_0
2. Draw a random perturbation around this point. Let the point that is this perturbation from x_0 be called x_1.
3. Determine the "energy" at points x_1 and x_0, call these e' and e, respectively.
   - $\rightarrow$ How energy is defined is loose, but it will be something that relates to the value or slope of the functions are a given point
   - $\rightarrow$ Since you'll compute this often, it's helpful is it's something that is easy to compute
   - $\rightarrow$ To fix ideas, just think of the energy of point x being $f(x)$
4. Compare the energy at these two points:
   - 4.1 If $e' > e$, move to x_1
   - 4.2 If $e' < e$, *maybe* move to x_1

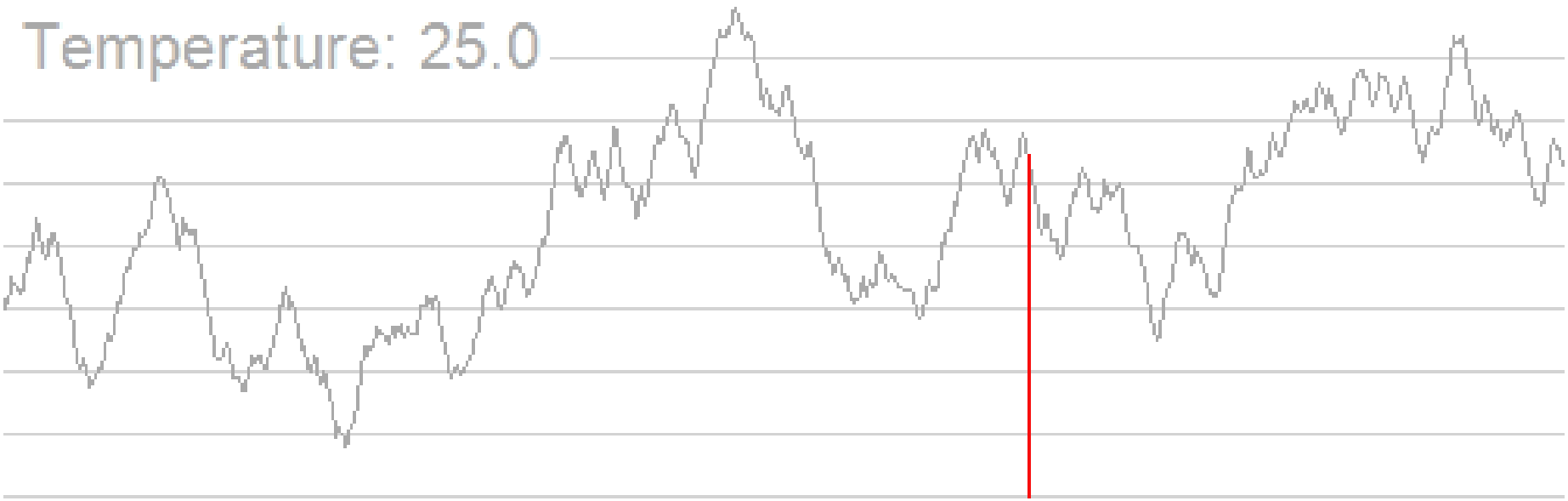
Simulated annealing: Algorithm (cont'd)

5. Accept new point at random if has lower energy
   - → Probability of acceptance should depend on 3 things: e, e', temperature (T)
   - → Temperate will start at a relatively high number and be lowered at each successive iteration through this algorithm
   - → $Pr(e, e', T)$ should be decreasing in e, increasing in e', and decreasing in T
   - → The first two conditions mean that high energy points are more likely to get selected
   - → The last means that the probability of accepting a low energy point declines as the number of iterations increase

Simulated annealing: Algorithm (cont'd)

6. If x_1 is selected, move there and go back to Step (a) with x_1 as the initial point
7. Repeat until T is zero (i.e., a maximum number of iterations has been reached) or when $|x_{k+1} - x_k| < \varepsilon$, where k denotes the iteration.

Simulated annealing: Algorithm in Action



Simulated annealing: Pros and Cons

- Benefits of simulated annealing:
  - → More likely to find global max (min) and not get stuck in local max (min)
- Costs of simulated annealing:
  - → Not guaranteed to find the optimum
  - → Convergence may not be quick - depends on how quickly reduce temp, but if reduce temp more quickly then more likely to get stuck in local max (min)
- Notes:
  - → Sometimes this algorithm is called basin-hopping (e.g. in SciPy)
  - → This method is related to the Metroplis-Hastings algorithm

Differential Evolution: Algorithm (1)

- Notation:
  - → Optimizing a function with D parameters
  - → Select the size of the population, $N > 4$ (this is a parameter of the optimizer)
  - → Parameter vectors given by:

$$x_{i,G} = [x_{1,i,G}, x_{2,i,G}, ...x_{D,i,G}], i = 1, 2, ..., N$$

  - → G is the "generation" number

Differential Evolution: Algorithm (2)

- Initialize:
  - → Define upper and lower bounds for each parameter; $x_j^L \leq x_{j,i,1} \leq x_j^U$
  - → Randomly select initial parameter values from a uniform distribution over $[x_j^L, x_j^U]$
- Mutate:
  - → For a given parameter vector $x_{i,G}$ randomly select three vectors $x_{r1,G}$, $x_{r2,G}$, and $x_{r3,G}$ such that the indices i, $r1$, $r2$ and $r3$ are distinct
  - → Add the weighted difference of two of the vectors to the third:

$$v_{i,G+1} = x_{r1,G} + F(x_{r2,G} - x_{r3,G})$$

  - → Where $F \in [0, 2]$ is a constant (and a parameter chosen when using DE called the "mutation factor")
  - → $v_{i,G+1}$ is called the "donor vector"

Differential Evolution: Algorithm (3)

- Recombination:
  - → Recombination incorporates successful solutions from the previous generation
  - → The trial vector $u_{i,G+1}$ is developed from the elements of the target vector, $x_{i,G}$, and the elements of the donor vector, $v_{i,G+1}$
  - → Elements of the donor vector enter the trial vector with probability CR

$$u_{j,i,G+1} = \begin{cases} v_{j,i,G+1}, & \text{if } rand_{j,i} \leq CR \text{ or } j = I_{rand} \\ x_{j,i,G}, & \text{if } rand_{j,i} > CR \text{ or } j \neq I_{rand} \end{cases} \quad i = 1, 2, ...N; j = 1, 2$$

  - → $rand_{j,i} \sim U[0, 1]$, I_{rand} is a random integer from $[1, 2, ..., D]$
  - → I_{rand} ensures that $v_{i,G+1} \neq x_{i,G}$

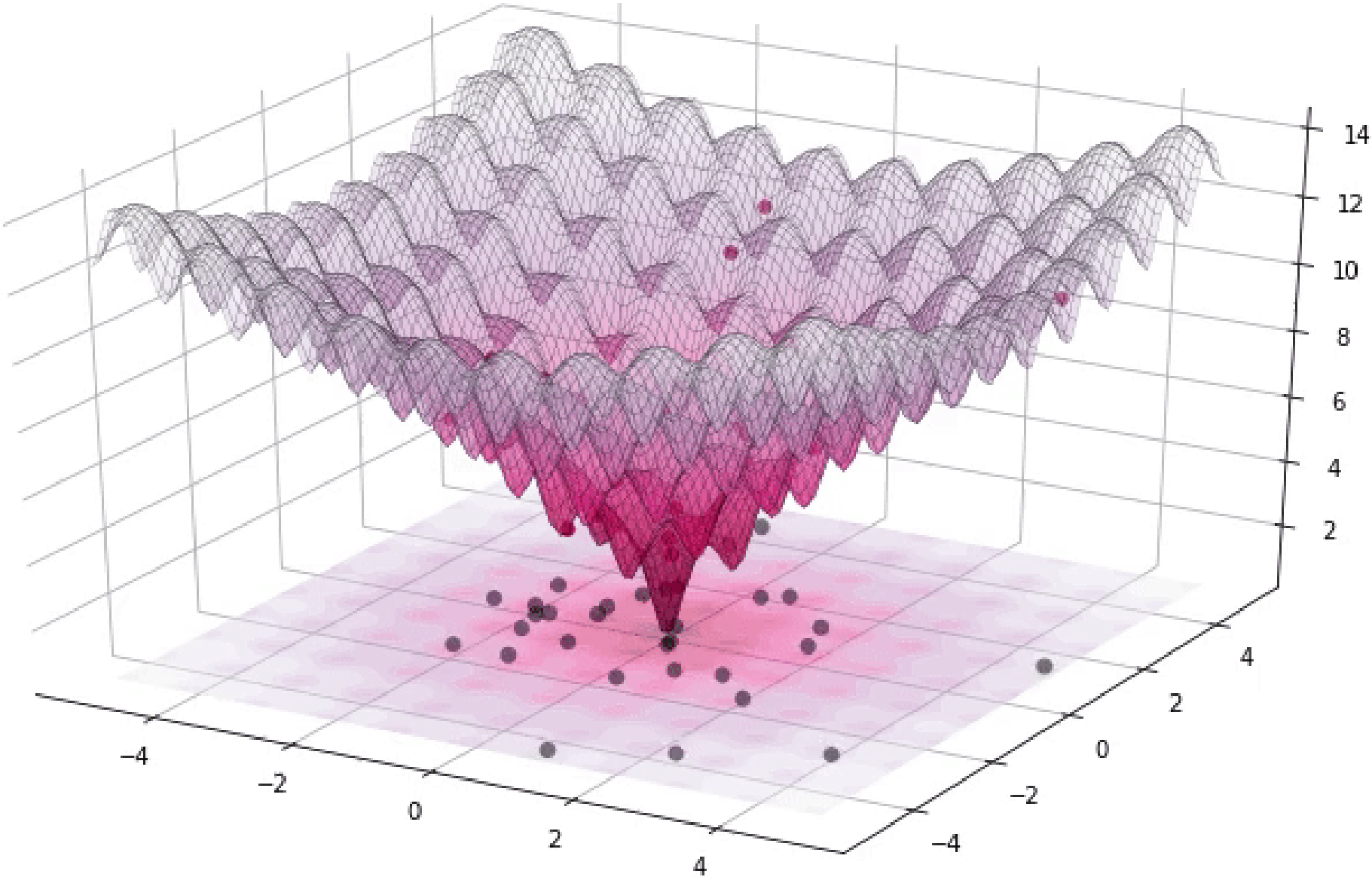
Differential Evolution: Algorithm (4)

- Selection:
  - → The target vector $x_{i,G}$ is compared with the trial vector $u_{i,G+1}$ and the one with the lowest function value is admitted to the next generation

$$x_{i,G+1} = \begin{cases} u_{i,G+1}, & \text{if } f(u_{i,G+1}) \leq f(x_{i,G}) \\ x_{i,G}, & \text{otherwise} \end{cases} \quad i = 1, 2, ...N$$

  - → Mutation, recombination and selection continue until some stopping criterion is reached

Differntial evolution: Algorithm in Action



Differential Evolution: Pros and Cons

- Benefits of DE:
  - → Can be used on problems that are not continuous or differentiable
  - → More likely to find global max (min) and not get stuck in local max (min)
- Costs of DE:
  - → Not guaranteed to find the optimum
  - → Can be computationally expensive (lots of function evaluations)

Roadmap

Rules of Thumb for Algorithm Selection

If the dimension of the problem is not too big:

1. If x_0 is close to $x*$
   - 1.1 If computing $(D^2 f(x))^{-1} Df(x)$ is cheap and accurate, use Newton method as it has fastest convergence
   - 1.2 If computing $(D^2 f(x))^{-1} Df(x)$ is expensive or error-prone, then
2. If x_0 is not close to $x*$, use gradient descent method (not necessarily steepest decent) for several steps to get closer to $x*$ then switch back to 1(a)
3. If other methods are not converging rapidly, try conjugate gradient

Rules of Thumb for Algorithm Selection (cont'd)

- If the dimension of the problem is large and the Hessian sparse, use conjugate gradient methods (no need to compute Hessian with these methods).
- If the function to minimize is not differentiable, try non-gradient methods like Nelder-Mead