Second-Order Methods

Asst. Prof. Dr.-Ing. Sudchai Boonto

Department of Control System and Instrumentation Engineering
King Mongkut's Unniversity of Technology Thonburi
Thailand

September 20, 2025

Objective

At the end of this chapter you should be able to:

- Describe, implement, and use 2nd-order Method
- Explain the pros and cons of the various 2nd-order methods.
- Understand Newton, Levenberg-Marquardt, DFP, BFGS

Newton's Method: one variable

- The function value and gradient can help to determine the direction to travel, but it does not directly help to determine how far to step to reach a local minimum.
- Second-order information allows us to make a quadratic approximation of the objective function and approximate the right step size to reach a local minimum.
- As we have seen with a quadratic fit search, we can analytically obtain the location where a quadratic approximation has a zero gradient. We can use that location as the next iteration to approach a local minimum.
- The quadratic approximation about a point $\mathbf{x}_k$ comes from the second-order Taylor expansion (scalar case):

$$f(x_k + s) = f(x_k) + f'(x_k)s + \frac{1}{2}s^2 f''(x_k)$$

$$\frac{d}{ds} f(x_k + s) = 0 + f'(x_k) + s f''(x_k) = 0, s = -\frac{f'(x_k)}{f''(x_k)},$$ where s is the step size

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

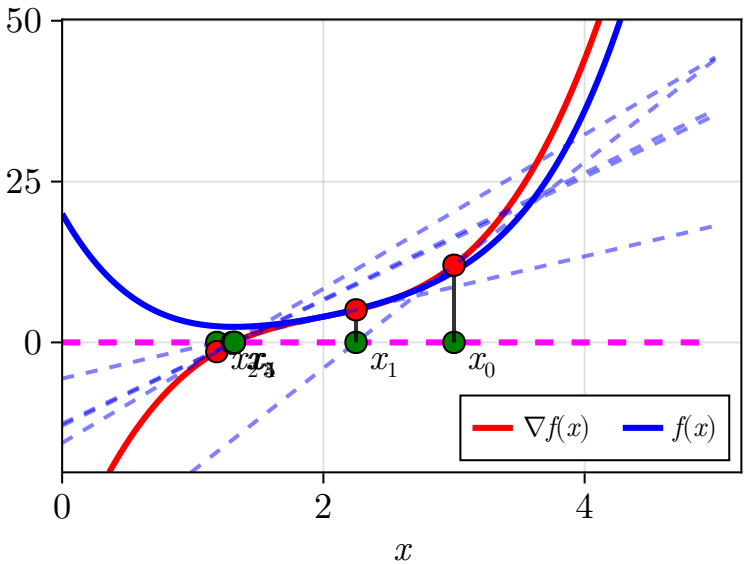
Newton's Method : One variable Example

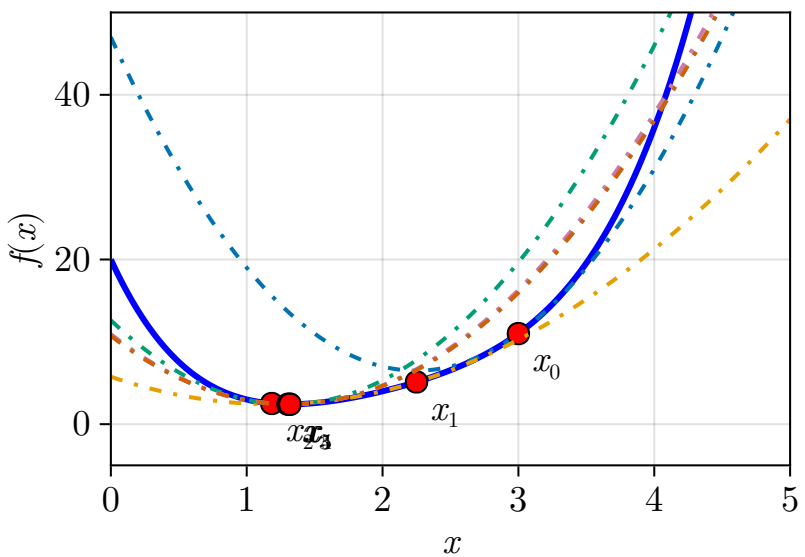
Suppose we want to minimize the following single-variable function:

$$f(x) = (x-2)^4 + 2x^2 - 4x + 4, \quad f'(x) = 4(x-2)^3 + 4x - 4,$$
$$f''(x) = 12(x-2)^2 + 4$$

with $x_0 = 3$, we can form the quadratic using the function value and the first and second derivatives evaluated at the point.

$$x_1 = x_0 - \frac{f'(x_0)}{f''(x_0)} = 2.25, x_2 = x_1 - \frac{f'(x_1)}{f''(x_1)} = 1.1842, x_3 = 1.3039, x^* = x_4 = 1.3177$$





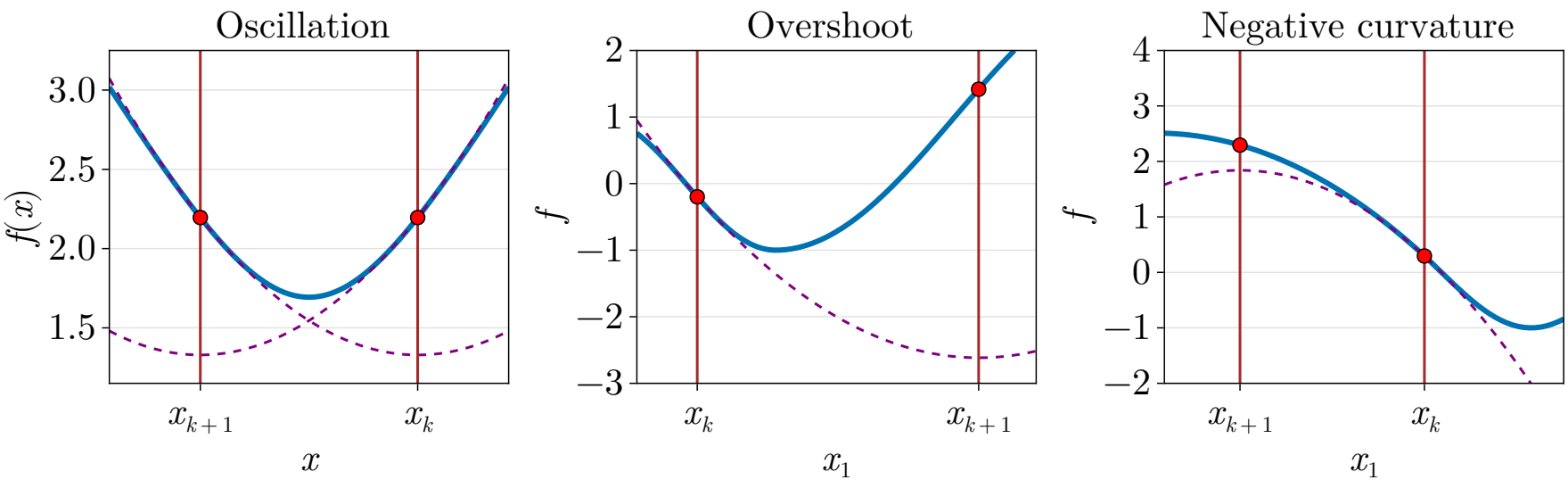
Newton's Method : One variable Example

Newton Iteration Table

i	x_k	$\frac{f''(x_k)}{f'(x_k)}$	$f(x_k)$
0	2.25	-0.75	5.12891
1	1.1842	-1.0658	2.51077
2	1.3039	0.1197	2.4195
3	1.3175	0.0136	2.41859
4	1.3177	0.0002	2.41859
5	1.3177	0.0	2.41859
6	1.3177	0.0	2.41859

Newton' s Method : Disadvantages

- The update rule in Newton's method involves dividing by the second derivative. The update is undefined if the second derivative is zero, which occurs when the quadratic approximation is a horizontal line.
- Instability also occurs when the second derivative is very close to zero, in which case the next iterate will lie very far from the current design point, far from where the local quadratic approximation is valid.
- Poor local approximations can lead to poor performance with Newton's method.



Newton's Method : Multivariate Optimization

- The multivariate second-order Taylor expansion at x_k is

$$f(\mathbf{x}_k + \mathbf{s}) \approx f(\mathbf{x}_k) + \nabla(f(\mathbf{x}_k)^T \mathbf{s} + \frac{1}{2}\mathbf{s}^T \mathbf{H}_k \mathbf{s}$$
$$\frac{d}{d\mathbf{s}}(\mathbf{x}_k + \mathbf{s}) = \nabla f(\mathbf{x}_k) + \mathbf{H}_k \mathbf{s} = 0$$

  We then solve for the next iterate, thereby obtaining Newton's method in multivariate form:

$$\mathbf{s} = -\mathbf{H}_k^{-1} \nabla f(\mathbf{x}_k), \quad \mathbf{s} = \mathbf{x}_{k+1} - \mathbf{x}_k$$
$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{H}_k^{-1} \nabla f(\mathbf{x}_k)$$

- If $f(\mathbf{x})$ is quadratic and its Hessian is positive definite, then the update converges to the global minimum in one step. For general functions, Newton's method is often terminated once $\mathbf{x}$ ceases to change by more than a given tolerance.

Newton's Method : Example

With $\mathbf{x}_1 = \begin{bmatrix} 9 & 8 \end{bmatrix}$, we will use Newton's method to minimize Booth's function:

$$f(\mathbf{x}) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2,$$

$$\nabla f(\mathbf{x}) = \begin{bmatrix} 10x_1 + 8x_2 - 34, & 8x_1 + 10x_2 - 38 \end{bmatrix}^T, \quad \mathbf{H}(\mathbf{x}) = \begin{bmatrix} 10 & 8 \\ 8 & 10 \end{bmatrix}$$

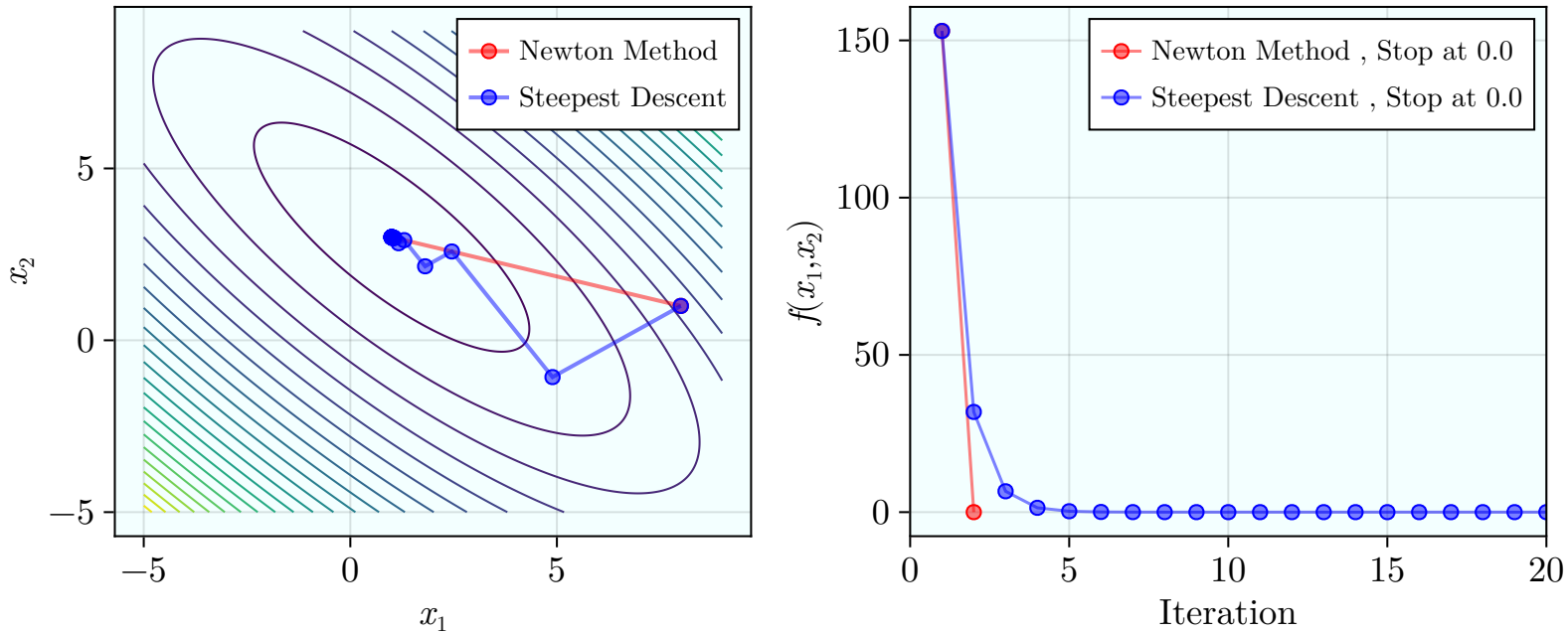
The first iteration of Newton's method yields:

$$\mathbf{x}_2 = \mathbf{x}_1 - \mathbf{H}_1^{-1}\mathbf{g}_1 = \begin{bmatrix} 9 \\ 8 \end{bmatrix} - \begin{bmatrix} 10 & 8 \\ 8 & 10 \end{bmatrix}^{-1} \begin{bmatrix} 10(9) + 8(8) - 34 \\ 8(9) + 10(8) - 38 \end{bmatrix}$$

$$= \begin{bmatrix} 9 \\ 8 \end{bmatrix} - \begin{bmatrix} 10 & 8 \\ 8 & 10 \end{bmatrix}^{-1} \begin{bmatrix} 120 \\ 114 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

The gradient at $\mathbf{x}_2$ is zero, so we have converged after a single iteration. The Hessian is positive definite everywhere, so $\mathbf{x}_2$ is the global minimum.

Newton's Method : Multivariate Optimization



- If $f(\mathbf{x})$ is quadratic and its Hessian is positive definite, the update converges to the global minimum in one step.

Newton's Method : Algorithm

Let $\mathbf{s} = x_{k+1} - x_k$

Require: $\mathbf{x}_0, \varepsilon_G, \nabla f_k, \mathbf{H}_k$

  $k = 0$

  while $\|\nabla f_k\| > \varepsilon_G$ and $k \leq k_{\max}$ **do**

    $\mathbf{s} = \mathbf{H}(\mathbf{x})^{-1} \nabla f(\mathbf{x})$

    $\mathbf{x} = \mathbf{x} + \mathbf{s}$

    $k = k + 1$

  end while

  return $\mathbf{x}$

- Newton's method is efficient because the second-order information results in better search directions. Two main steps in Newton's method
- Need to compute Hessian $\mathbf{H}$
- Solve the system of equations $\mathbf{Hs} = -\nabla f(\mathbf{x}) \;\Rightarrow\; \mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{H}^{-1} \nabla f(\mathbf{x}_k)$

Secant Methods

- Newton's method for univariate function minimization requires the first and second derivatives $f'(x_k)$ and $f''(x_k)$
- In many cases, $f'(x_k)$ is known but the second derivative is not.
- We can use first-order information (gradients) along each step in the iteration path to build an approximation of Hessian.
- The secant method uses the last two iterates to approximate the second derivative.

$$f''(x_k) \approx \frac{f'(x_k) - f'(x_{k-1})}{x_k - x_{k-1}}$$

This estimate is substituted into Newton's method

$$x_{k+1} = x_k - \frac{x_k - x_{k-1}}{f'(x_k) - f'(x_{k-1})} f'(x_k)$$

- The secant method requires and additional initial design point. The method has the same problems as Newton's method and may take more iterations to converge due to approximating the second derivative.

Levenberg-Marquardt Algorithm

- Newton's method tends to perform well whn our objective function is a quadratic equation or is closely approximation by one. It performs poorly when the approximation is poor.
- A quadratic approximation for a smooth function will tend to be good sufficiently close to a local optimum, where the objective function is typically convex and bowl-like. In more linear regions, quadratic approximations will be poor, and in the concave areas, quadratic approximations will have degenerate Hessians. In such cases, it is often more effective to use a simple gradient descent step.
- The interpolation update rule is parameterized by a damping factor δ:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - (\mathbf{H}_k + \delta \mathbf{I})^{-1} \nabla f(\mathbf{x}_k)$$

  $\delta > 0$ is small, the update mimics Newton's method. When δ is large, the update mimics gradient descent with a step factor $\alpha \approx 1/\delta$.

Levenberg-Marquardt Algorithm

- The damping factor is adjusted during descent based on whether each iterations improve the objective function value. If the objective is better at the next iterate, it is accepted, and δ can be decreased. If the objective is worse at the next iterate, it is rejected, the algorithm retains the current iterate, and δ is increased.
- In practice, we use

$$\mathbf{x}_{k+1} = \mathbf{x}_k - (\mathbf{H}_k + \delta \operatorname{diag}(\operatorname{diag}(\mathbf{H}_k)))^{-1} \nabla f(\mathbf{x}_k)$$

  This leverages information about the Hessian even when mimicking gradient descent, allowing iterates to move further in directions where the gradient is smaller.
- If the Hessian is not invertible and $\operatorname{diag}(\mathbf{H}_k)$ has any negative entries, increasing the damping factor will not produce an invertible matrix. We can take the component-wise maximum of these values with a small positive number.

Levenberg-Marquardt Algorithm

Each step of Levenberg-marquardt method:

Require: $\mathbf{x}_0$, ε_G, ∇f_k, $\mathbf{H}_k$, δ, γ_a, γ_r

$\mathbf{M} = \mathbf{H}(\mathbf{x}_0)$

$d = \max(\mathrm{diag}(\mathbf{M}), \varepsilon)$

$\mathbf{M} = \mathbf{M} + \delta * \mathrm{diag}(d)$

$\mathbf{x}' = \mathbf{x} - \mathbf{M}^{-1} \nabla f(\mathbf{x})$

if $f(\mathbf{x}') < f(\mathbf{x})$ **then** ▷ accept the new design and decrease damping

  return $\mathbf{x} = \mathbf{x}', \delta = \delta * \gamma_a$

end if ▷ reject the new design and increase damping

return $\mathbf{x} = \mathbf{x}, \delta = \delta * \gamma_r$

Levenberg-Marquardt Algorithm: Gauss-Newton Method

The Levenberg-marquardt algorithm was ariginally develope for the least-square problem, that we can derive an efficient approximation of the Hessian.

- Consider an objective function:

$$f(\mathbf{x}) = \frac{1}{2}\sum_i f_i(\mathbf{x})^2,\ \nabla f(\mathbf{x}) = \sum_i f_i(\mathbf{x})\nabla f_i(\mathbf{x}),\ f_i(\mathbf{x}) \approx f_i(\mathbf{x}_k) + \nabla f_i(\mathbf{x}_k)^T(\mathbf{x} - \mathbf{x}_k)$$

- Substituting the approximation back:

$$\begin{aligned}\nabla f(\mathbf{x}) &\approx \sum_i \left(f_i(\mathbf{x}_k) + \nabla f_i(\mathbf{x}_k)^T(\mathbf{x} - \mathbf{x}_k)\right)\nabla f_i(\mathbf{x}) \\ &= \sum_i f_i(\mathbf{x}_k)\nabla f_i(\mathbf{x}_k) + \sum_i \nabla f_i(\mathbf{x}_k)\nabla f_i(\mathbf{x}_k)^T(\mathbf{x} - \mathbf{x}_k)\end{aligned}$$

- Since we have only one scalar objective function of a vector parameter $f(\mathbf{x})$, then

$$\mathbf{J} = \begin{bmatrix}\frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \cdots & \frac{\partial f}{\partial x_n}\end{bmatrix} = (\nabla f(\mathbf{x}))^T$$

Levenberg-Marquardt Algorithm: Gauss-Newton Method

- Then we have

$$\nabla f(\mathbf{x}) = \mathbf{J}^T f(\mathbf{x}) + \mathbf{J}^T \mathbf{J}(\mathbf{x} - \mathbf{x}_k) = 0$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \left(\mathbf{J}^T \mathbf{J}\right)^{-1} \mathbf{J}^T f(\mathbf{x})$$

- This update matches Newton's method, except the Hessian is approximated using the outer product of the gradients.
- Using the damping factor produces the Levenberg-Marquardt:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \left(\mathbf{J}^T \mathbf{J} + \delta \operatorname{diag}\left(\operatorname{diag}(\mathbf{J}^T \mathbf{J})\right)\right)^{-1} \mathbf{J}^T f(\mathbf{x}_k)$$

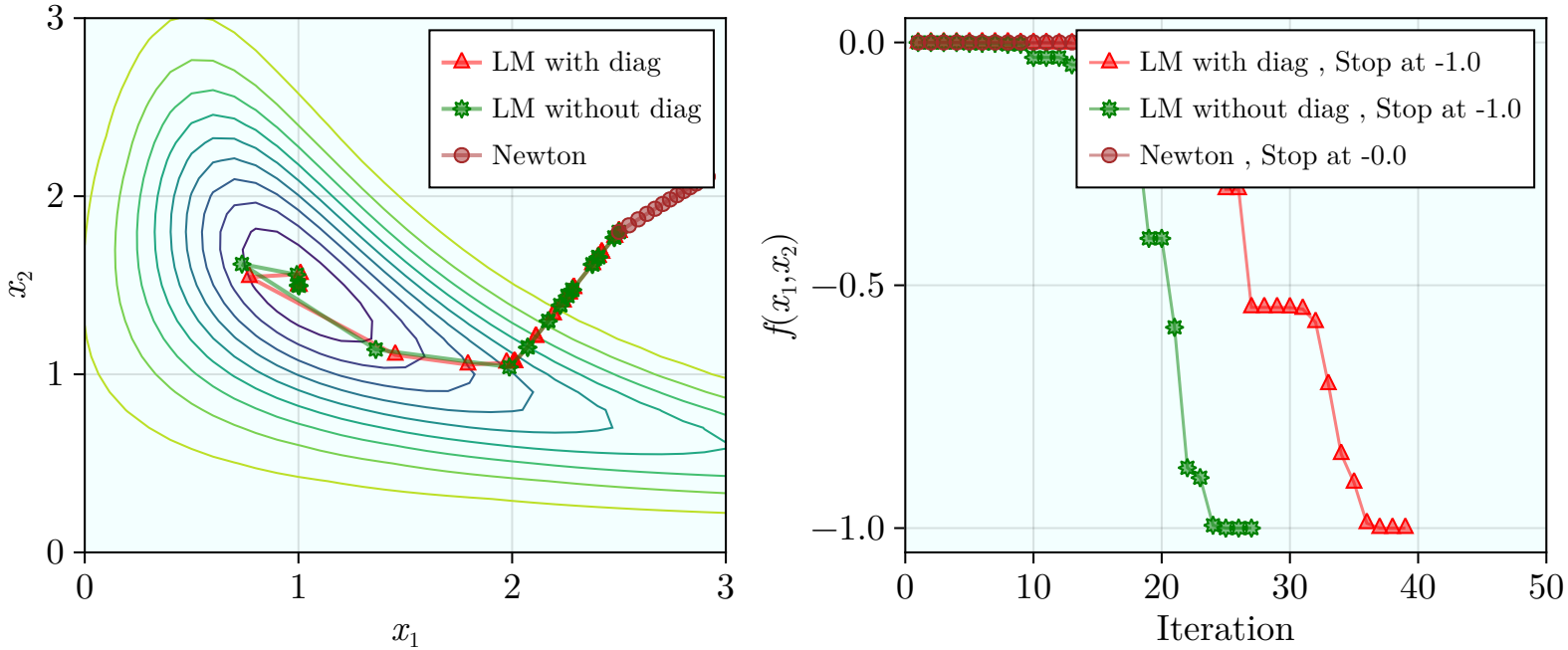
Quasi-Newton Methods

- Quasi-Newton methods are broad class of approaches to approximate the Hessian used in Newton's method.
- When $\mathbf{H}$ is too expensive to compute, or when $\mathbf{H}$ is not positive definite, we may approximate this Hessian with a matrix $\tilde{\mathbf{H}}$. The matrix $\tilde{\mathbf{H}}$ provides a quadratic approximation. The matrix $\tilde{\mathbf{H}}$ provides a quadratic approximation $\tilde{f}(\mathbf{x})$ to our objective function $f(\mathbf{y})$ at some point $\mathbf{y}$ near the current base point $\mathbf{x}$

$$\tilde{f}(\mathbf{x}_k + \mathbf{p}_k) = f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T \mathbf{p}_k + \frac{1}{2}\mathbf{p}_k^T \tilde{\mathbf{H}}_k \mathbf{p}_k$$

  where $\tilde{\mathbf{H}}$ is an approximation of the Hessian.
- To design an effective approximation $\tilde{\mathbf{H}}$ to $\mathbf{H}$, we need to put some constraints on this new matrix:
  - $\tilde{\mathbf{H}}$ should be symmetric, as $\mathbf{H}$ is symmetric.
  - **Condition I:** $\tilde{\mathbf{H}}$ should be positive definite, since we will invert it in Newton's method.

Levenberg-Marquardt Algorithm



- Using the exact Hessian provides the most accurate information about the curvature of the minimum surface. This can lead to more precise steps and potentially faster convergence if the function is well-behaved.
- Calculation the second derivatives is very expensive. In many real-world problems, it's simply not practical.

Quasi-Newton Methods

- Cont.
  - **Condition II:** The quadratic approximation $\tilde{f}$ should have the same gradient at the current point $\mathbf{x}_{k+1}$ and last point $\mathbf{x}_k$:

$$\nabla \tilde{f}(\mathbf{x}_{k+1}) = \nabla f(\mathbf{x}_{k+1})$$
$$\nabla \tilde{f}(\mathbf{x}_k) = \nabla f(\mathbf{x}_k)$$

  - Since $\tilde{\mathbf{H}}$ will be updated at each iteration step, we add a subscript $\tilde{\mathbf{H}}_k$. We want $\tilde{\mathbf{H}}_{k+1}$ to be close to $\tilde{\mathbf{H}}_k$.

Quasi-Newton Methods

The quadratic approximation of the objective function:

$$\tilde{f}(\mathbf{x}_k + \mathbf{p}_k) = f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T \mathbf{p}_k + \frac{1}{2}\mathbf{p}_k^T \tilde{\mathbf{H}}_k \mathbf{p}_k$$

Minimize this quadratic with respect to $\mathbf{p}$, and let it equal zero, we have

$$\tilde{\mathbf{H}}_k \mathbf{p}_k = -\nabla f(\mathbf{x}_k)$$

We get the $\mathbf{p}_k$ direction and update the point using $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$.
Quasi-Newton methods update the approximate Hessian at every iteration based on the latest information using an update of the form (instead of recalculate the Hessian.)

$$\tilde{\mathbf{H}}_{k+1} = \tilde{\mathbf{H}}_k + \Delta\tilde{\mathbf{H}}_k$$

The approximation of the Hessian must match the slope of the actual function at the last two point.

Quasi-Newton Methods

Condition II: Using $\mathbf{x}_k$ in the direction of $\mathbf{p}_k$, we have

$$\tilde{f}(\mathbf{x}_{k+1} + \mathbf{p}_k) = f(\mathbf{x}_{k+1}) + \nabla f(\mathbf{x}_{k+1})^T \mathbf{p}_k + \frac{1}{2}\mathbf{p}_k^T \tilde{\mathbf{H}}_{k+1}\mathbf{p}_k$$

$$\nabla \tilde{f}(\mathbf{x}_{k+1} + \mathbf{p}_k) = \nabla f(\mathbf{x}_{k+1}) + \tilde{\mathbf{H}}_{k+1}\mathbf{p}_k$$

If $\mathbf{p}_k = 0$, then we have $\nabla \tilde{f}(\mathbf{x}_{k+1}) = \nabla f(\mathbf{x}_{k+1})$. Move backward, $\mathbf{x}_k = \mathbf{x}_{k+1} - \alpha_k \mathbf{p}_k$. Then the approximation is

$$\nabla \tilde{f}(\mathbf{x}_{k+1} - \alpha_k \mathbf{p}_k) = \nabla \tilde{f}(\mathbf{x}_k) = \nabla f(\mathbf{x}_{k+1}) - \alpha_k \tilde{\mathbf{H}}_{k+1}\mathbf{p}_k$$

To enforce that the $\nabla \tilde{f}(\mathbf{x}_k) = \nabla f(\mathbf{x}_k)$, we need

$$\nabla f(\mathbf{x}_{k+1}) - \alpha_k \tilde{\mathbf{H}}_{k+1}\mathbf{p}_k = \nabla f(\mathbf{x}_k) \;\Rightarrow\; \alpha_k \tilde{\mathbf{H}}_{k+1}\mathbf{p}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$$

Let $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k = \alpha_k \mathbf{p}_k$, and $\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$. We have

$$\tilde{\mathbf{H}}_{k+1}\mathbf{s}_k = \mathbf{y}_k, \qquad \text{which is a secant equation}$$

Quasi-Newton Methods

Condition II: We need $\tilde{\mathbf{H}}$ to be positive definite then

$$\tilde{\mathbf{H}}_{k+1}\mathbf{s}_k = \mathbf{y}_k \;\Rightarrow\; \mathbf{s}_k^T\tilde{\mathbf{H}}_{k+1}\mathbf{s}_k = \mathbf{s}_k^T\mathbf{y}_k > 0$$

The latter is called the curvature condition, and it is automatically satisfied if the line search finds a step that satisfies the strong Wolfe conditions.

- The original quasi-Newton update, known ad DFP, was first proposed by Davidon and then refined by Fletcher and slso Powell. The DFP update formula has been superseded by the BFGS formula, which was independently developed by Broyden, Fletcher, Goldfarb, and Shanno.
- The BFGS is currently considered the most effective quasi-Newton update.

Quasi-Newton Methods

- As the secant method approximates f'' in the univariate case, *quasi-Newton* methods approximate the inverse Hessian. Quasi-Newton method updates have the form:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{Q}_k \nabla f_k,$$

  where α_k is a scalar step factor and $\mathbf{Q}_k$ approximates the inverse of the Hessian at $\mathbf{x}_k$
- These methods typically set $\mathbf{Q}_0$ to the identity matrix, and they then apply updates to reflect information learned with each iteration. To simplify the equations for the various quasi-Newton methos, we define the following:

$$\mathbf{y}_{k+1} = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$$
$$\mathbf{s}_{k+1} = \mathbf{x}_{k+1} - \mathbf{x}_k$$

Quasi-Newton Methods : Davidon-Fletcher-Powell (DFP)

In stead of starting with the update for the Hessian, we use the inverse Hessian $\mathbf{Q}$.

$$\mathbf{Q}_{k+1} = \mathbf{Q}_k + \alpha \mathbf{u}\mathbf{u}^T + \beta \mathbf{v}\mathbf{v}^T$$
$$\tilde{\mathbf{H}}_{k+1}\mathbf{s}_k = \mathbf{y}_k \;\Rightarrow\; \mathbf{Q}_{k+1}\mathbf{y}_k = \mathbf{s}_k$$

Setting $\mathbf{u} = \mathbf{s}_k$ and $\mathbf{v} = \mathbf{Q}_k\mathbf{y}_k$, we have

$$\begin{aligned}
\mathbf{Q}_{k+1} &= \mathbf{Q}_k + \alpha \mathbf{s}_k\mathbf{s}_k^T + \beta \mathbf{Q}_k\mathbf{y}_k\mathbf{y}_k^T\mathbf{Q}_k^T && \text{Multiply both sides with } \mathbf{y}_k \\
\mathbf{s}_k &= \mathbf{Q}_k\mathbf{y}_k + \alpha \mathbf{s}_k\mathbf{s}_k^T\mathbf{y}_k + \beta \mathbf{Q}_k\mathbf{y}_k\mathbf{y}_k^T\mathbf{Q}\mathbf{y}_k && \text{from secant equation} \\
\mathbf{s}_k - \alpha \mathbf{s}_k\mathbf{s}_k^T\mathbf{y}_k &= \mathbf{Q}_k\mathbf{y}_k + \beta \mathbf{Q}_k\mathbf{y}_k\mathbf{y}_k^T\mathbf{Q}_k\mathbf{y}_k^T \\
\mathbf{s}_k(1 - \alpha \mathbf{s}_k^T\mathbf{y}_k) &= \mathbf{Q}_k\mathbf{y}_k(1 + \beta \mathbf{y}_k^T\mathbf{Q}_k\mathbf{y}_k^T)
\end{aligned}$$

The last equation is correct if both sides are zero or

$$\alpha = \frac{1}{\mathbf{s}_k^T\mathbf{y}_k}, \qquad \beta = \frac{-1}{\mathbf{y}_k^T\mathbf{Q}_k\mathbf{y}_k}$$

Quasi-Newton Methods : Davidon-Fletcher-Powell (DFP)

The *Davidon-Fletcher-Powell* (DFP) method uses:

$$\mathbf{Q}_{k+1} = \mathbf{Q}_k + \frac{\mathbf{s}_k \mathbf{s}_k^T}{\mathbf{s}_k^T \mathbf{y}_k} - \frac{\mathbf{Q}_k \mathbf{y}_k \mathbf{y}_k^T \mathbf{Q}_k}{\mathbf{y}_k^T \mathbf{Q}_k \mathbf{y}_k}$$

The update for $\mathbf{Q}$ in the DFP method havs three properties:

- $\mathbf{Q}$ remains symmetric and positive definite.
- If $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \mathbf{A}\mathbf{x} + \mathbf{b}^T \mathbf{x} + c$, then $\mathbf{Q} = \mathbf{A}^{-1}$. Thus the DFP has the same convergence properties as the conjugate gradient method.
- For high-dimensional problems, storing and updating $\mathbf{Q}$ can be significant compared to other methods like the conjugate gradient method.
- The DPF algorithm does not guarantee the positiveness of the Hessian $\tilde{\mathbf{H}}$.

Quasi-Newton Methods : Davidon-Fletcher-Powell (DFP)

Require: $\mathbf{x}_0, \varepsilon_G, f, \nabla f(\mathbf{x}_0)$

$k = 0, \mathbf{Q} = I$

while $\|\nabla f(\mathbf{x}_k)\| > \varepsilon_G \;\&\&\; k \leq k_{\max}$ do

$\mathbf{g}_k = \nabla f(\mathbf{x}_k)$

$\alpha = \text{line\_search}(f, \mathbf{x}_k, -\mathbf{Q} * \mathbf{g})$

$\mathbf{x}_{k+1} = x_k - \alpha Q * g$

$\mathbf{g}_{k+1} = \nabla f(\mathbf{x}_{k+1})$

$\mathbf{s} = \mathbf{x}_{k+1} - \mathbf{x}_k$

$\mathbf{y} = \mathbf{g}_{k+1} - \mathbf{g}_k$

$\mathbf{Q} = \mathbf{Q} - \mathbf{Q}\mathbf{y}_k\mathbf{y}_k^T\mathbf{Q}/\mathbf{y}_k^T\mathbf{Q}\mathbf{y}_k + \mathbf{s}_k\mathbf{s}_k^T/\mathbf{s}_k^T\mathbf{y}_k$

$k = k + 1$

$\mathbf{x}_k = \mathbf{x}_{k+1}$

end while

return $\mathbf{x}_{k+1}$

Quasi-Newton Methods : BFGS

In addition to the secant equation, we would like

- $\tilde{\mathbf{H}}_{k+1}$ is symmetric
- $\tilde{\mathbf{H}}_{k+1}$ is close to $\tilde{\mathbf{H}}_k$
- $\tilde{\mathbf{H}}$ is positive definite the $\tilde{\mathbf{H}}_{k+1}$ is positive definite.

Using rank-1 update ($\tilde{\mathbf{H}}_{k+1} = \tilde{\mathbf{H}}_k + \alpha \mathbf{u}\mathbf{u}^T$ is satisfied the secant equation, but is not guaranteed to be positive definite. The BFGS is a rank-2 update

$$\tilde{\mathbf{H}}_{k+1} = \tilde{\mathbf{H}}_k + \alpha \mathbf{u}\mathbf{u}^T + \beta \mathbf{v}\mathbf{v}^T, \quad \tilde{\mathbf{H}}_{k+1}\mathbf{s}_k = \mathbf{y}_k \;\Rightarrow\; \tilde{\mathbf{H}}_k\mathbf{s}_k + \alpha \mathbf{u}\mathbf{u}^T\mathbf{s}_k + \beta \mathbf{v}\mathbf{v}^T\mathbf{s}_k = \mathbf{y}_k$$

Setting $\mathbf{u} = \mathbf{y}$ and $\mathbf{v} = \tilde{\mathbf{H}}\mathbf{s}$ yields

$$\begin{aligned} \tilde{\mathbf{H}}_k\mathbf{s}_k + \alpha \mathbf{y}_k\mathbf{y}_k^T\mathbf{s}_k + \beta \tilde{\mathbf{H}}_k\mathbf{s}_k \left(\tilde{\mathbf{H}}_k\mathbf{s}_k\right)^T \mathbf{s}_k &= \mathbf{y}_k \\ \mathbf{y}_k \left(1 - \alpha \mathbf{y}_k^T\mathbf{s}_k\right) &= \tilde{\mathbf{H}}_k\mathbf{s}_k \left(1 + \beta \mathbf{s}_k^T\tilde{\mathbf{H}}_k\mathbf{s}_k\right) \end{aligned}$$

we need $\alpha = \frac{1}{\mathbf{y}_k^T\mathbf{s}_k}$, and $\beta = -\frac{1}{\mathbf{s}_k^T\tilde{\mathbf{H}}_k\mathbf{s}_k}$.

Quasi-Newton Methods : BFGS

We get the BFGS update:

$$\tilde{\mathbf{H}}_{k+1} = \tilde{\mathbf{H}}_k + \frac{\mathbf{y}_k \mathbf{y}_k^T}{\mathbf{y}_k^T \mathbf{s}_k} - \frac{\tilde{\mathbf{H}}_k \mathbf{s}_k \mathbf{s}_k^T \tilde{\mathbf{H}}_k}{\mathbf{s}_k^T \tilde{\mathbf{H}}_k \mathbf{s}_k}$$

It is more efficient to approximate the inverse of the Hessian directly instead. The inverse approximation Hessian $\mathbf{Q}$ can be found analytically from the update $\tilde{\mathbf{H}}$ using the Sherman-Morrison-Woodbury formula

$$\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{U}\mathbf{V}^T, \qquad \tilde{\mathbf{A}}^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}(\mathbf{I} + \mathbf{V}^T\mathbf{A}^{-1}\mathbf{U})\mathbf{V}^T\mathbf{A}^{-1}$$

We have

$$\mathbf{Q}_{k+1} = \left(I - \sigma_k \mathbf{s}_k \mathbf{y}_k^T\right) \mathbf{Q}_k \left(I - \sigma_k \mathbf{y}_k \mathbf{s}_k^T\right) + \sigma_k \mathbf{s}_k \mathbf{s}_k^T,$$

where $\sigma_k = \frac{1}{\mathbf{y}_k^T \mathbf{s}_k}$. We have $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \mathbf{Q}_k \nabla f(\mathbf{x}_k)$

Broyden-Fletcher-Goldfarb-Shanno (BFGS) proof

By letting $\tilde{\mathbf{H}}_{k+1} = \tilde{\mathbf{A}}$, and $\tilde{\mathbf{H}}_k = \mathbf{A}$, we have

$$\mathbf{Q}_{k+1} = \mathbf{Q}_k - \underbrace{\mathbf{Q}_k \mathbf{U}_k}_{\mathbf{V}_k} \left(\mathbf{I} + \mathbf{V}_k^T \mathbf{Q}_k \mathbf{U}_k\right)^{-1} \mathbf{V}_1 \underbrace{\mathbf{V}_2 \mathbf{Q}_k}_{\mathbf{V}_k^T}, \ \mathbf{V}_1 \mathbf{V}_2 = \mathbf{V}_k^T, \ \mathbf{V}_2 = \mathbf{U}^T, \ \tilde{\mathbf{H}}^{-1} = \mathbf{Q}$$

$$\mathbf{V}_1 = \begin{bmatrix} -\frac{1}{\mathbf{s}_k^T \tilde{\mathbf{H}}_k \mathbf{s}_k} & 0 \\ 0 & \frac{1}{\mathbf{y}_k^T \mathbf{s}_k} \end{bmatrix}, \ \mathbf{V}_2 = \begin{bmatrix} \mathbf{s}_k^T \tilde{\mathbf{H}}_k \\ \mathbf{y}_k^T \end{bmatrix} = \mathbf{U}_k^T, \ \mathbf{U}_k = \begin{bmatrix} \tilde{\mathbf{H}}_k \mathbf{s}_k & \mathbf{y}_k \end{bmatrix},$$

Let $\mathbf{V}_k = \mathbf{Q}_k \mathbf{U}_k = (\mathbf{V}_2 \mathbf{Q}_k)^T = \begin{bmatrix} \mathbf{s}_k & \mathbf{Q}_k \mathbf{y}_k \end{bmatrix}$, and $\mathbf{V}_2 \mathbf{Q}_k = \begin{bmatrix} \mathbf{s}_k^T \tilde{\mathbf{H}}_k \mathbf{Q}_k & \mathbf{y}_k^T \mathbf{Q}_k \end{bmatrix}^T = \begin{bmatrix} \mathbf{s}_k \\ \mathbf{Q}_k \mathbf{y}_k \end{bmatrix}$ Since, $\mathbf{M} = \left(\mathbf{I} + \mathbf{V}_k^T \mathbf{Q}_k \mathbf{V}_k\right)^{-1} \mathbf{V}_1$, we have

$$\begin{aligned}
\mathbf{Q}_{k+1} &= \mathbf{Q}_k - \mathbf{V}_k \mathbf{M} \mathbf{V}_k^T \\
\mathbf{M} &= \left(\mathbf{I} + \mathbf{V}_k^T \mathbf{Q}_k \mathbf{U}_k\right)^{-1} \mathbf{V}_1 = \left(\mathbf{V}_1^{-1} + \mathbf{V}_2 \mathbf{Q}_k \mathbf{V}_2^T\right)^{-1} \\
\mathbf{M}^{-1} &= \mathbf{V}_1^{-1} + \mathbf{V}_2 \mathbf{Q}_k \mathbf{V}_2^T = \begin{bmatrix} \frac{-1}{\mathbf{s}_k^T \tilde{\mathbf{H}}_k \mathbf{s}_k} & 0 \\ 0 & \frac{1}{\mathbf{y}_k^T \mathbf{s}_k} \end{bmatrix}^{-1} + \begin{bmatrix} \mathbf{s}_k^T \tilde{\mathbf{H}}_k \\ \mathbf{y}_k^T \end{bmatrix} \mathbf{Q}_k \begin{bmatrix} \tilde{\mathbf{H}}_k \mathbf{s}_k & \mathbf{y}_k \end{bmatrix} \\
&= \begin{bmatrix} -\mathbf{s}_k^T \tilde{\mathbf{H}}_k \mathbf{s}_k & 0 \\ 0 & \mathbf{y}_k^T \mathbf{s}_k \end{bmatrix} + \begin{bmatrix} \mathbf{s}_k^T \tilde{\mathbf{H}}_k \\ \mathbf{y}_k^T \end{bmatrix} \begin{bmatrix} \mathbf{s}_k & \mathbf{Q}_k \mathbf{y}_k \end{bmatrix} = \begin{bmatrix} 0 & \mathbf{s}_k^T \mathbf{y}_k \\ \mathbf{y}_k^T \mathbf{s}_k & \mathbf{y}_k^T \mathbf{s}_k + \mathbf{y}_k^T \mathbf{Q}_k \mathbf{y}_k \end{bmatrix}
\end{aligned}$$

Broyden-Fletcher-Goldfarb-Shanno (BFGS) proof

$$\mathbf{M} = \frac{-1}{\mathbf{y}_k^T \mathbf{s}_k \mathbf{s}_k^T \mathbf{y}_k} \begin{bmatrix} \mathbf{y}_k^T \mathbf{s}_k + \mathbf{y}_k^T \mathbf{Q}_k \mathbf{y}_k & -\mathbf{s}_k^T \mathbf{y}_k \\ -\mathbf{y}_k^T \mathbf{s}_k & 0 \end{bmatrix} = -\rho \begin{bmatrix} 1 + \rho \mathbf{y}_k^T \mathbf{Q}_k \mathbf{y}_k & -1 \\ -1 & 0 \end{bmatrix}, \rho = \frac{1}{\mathbf{s}_k^T \mathbf{y}_k}$$

$$\begin{aligned} \mathbf{Q}_{k+1} &= \mathbf{Q}_k - \mathbf{V}_k^T \mathbf{M} \mathbf{V}_k^T = \mathbf{Q}_k + \rho_k \begin{bmatrix} \mathbf{s}_k & \mathbf{Q}_k \mathbf{y}_k \end{bmatrix} \begin{bmatrix} 1 + \rho_k \mathbf{y}_k^T \mathbf{Q}_k \mathbf{y}_k & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{s}_k^T \\ \mathbf{y}_k^T \mathbf{Q}_k \end{bmatrix} \\ &= \mathbf{Q}_k + \rho_k \left(\mathbf{s}_k \mathbf{s}_k^T + \rho_k \mathbf{s}_k \mathbf{y}_k^T \mathbf{Q}_k \mathbf{y}_k \mathbf{s}_k^T - \mathbf{Q}_k \mathbf{y}_k \mathbf{s}_k^T - \mathbf{s}_k \mathbf{y}_k^T \mathbf{Q}_k \right) \\ &= \left(\mathbf{I} - \rho_k \mathbf{s}_k \mathbf{y}_k^T \right) \tilde{\mathbf{Q}}_k \left(\mathbf{I} - \rho_k \mathbf{y}_k \mathbf{s}_k^T \right) + \rho_k \mathbf{s}_k \mathbf{s}_k^T, \end{aligned}$$

where $\rho_k^2 = \frac{1}{\mathbf{y}_k^T \mathbf{s}_k \mathbf{s}_k^T \mathbf{y}_k}$

Broyden-Fletcher-Goldfarb-Shanno (BFGS)

Require: $\mathbf{x}_0$, ε_G, f, ∇f_k

$k = 0, \mathbf{Q} = I$

while $\|\nabla f(\mathbf{x}_k)\| > \varepsilon_G$ && $k \leq k_{\max}$ do

$\mathbf{g}_k = \nabla f(\mathbf{x}_k)$

$\alpha = \text{line\_search}(f, \mathbf{x}, -\mathbf{Q} * \mathbf{g})$

$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \mathbf{Q} * \mathbf{g}$

$\mathbf{g}_{k+1} = \nabla f(\mathbf{x}_{k+1})$

$\mathbf{s} = \mathbf{x}_{k+1} - \mathbf{x}_k$

$\mathbf{y} = \mathbf{g}_{k+1} - \mathbf{g}_k$

$\sigma = 1/\mathbf{s}_k^T \mathbf{y}_k$

$\mathbf{Q} = \left(I - \sigma_k \mathbf{s}_k \mathbf{y}_k^T\right) \mathbf{Q}_k \left(I - \sigma_k \mathbf{y}_k \mathbf{s}_k^T\right) + \sigma_k \mathbf{s}_k \mathbf{s}_k^T$

$k = k + 1$

$\mathbf{x}_k = \mathbf{x}_{k+1}$

end while

return $\mathbf{x}_{k+1}$

Limited-Memory Quasi-Newton Methods

- When the problem is large whose Hessian matrices cannot be computed at a reasonable cost.
- Instead of storing fully dense $n \times n$ approximations, we can save only a few vectors of length n that represent the approximations implicitly.
- Here we introduce the limited-Memory BFGS or L-BFGS.
- The BFGS method has the form

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{Q}_k \nabla f_k,\ \mathbf{Q}_{k+1} = \mathbf{V}_k^T \mathbf{Q}_k \mathbf{V}_k + \rho_k \mathbf{s}_k \mathbf{s}_k^T, \text{ where}$$
$$\rho_k = \frac{1}{\mathbf{y}_k^T \mathbf{s}_k},\ \mathbf{V}_k = \mathbf{I} - \rho_k \mathbf{y}_k \mathbf{s}_k^T, \text{ and } \mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k,\ \mathbf{y}_k = \nabla f_{k+1} - \nabla f_k$$

- The inverse Hessian approximation $\mathbf{Q}_k$ will generally be dense, the cost of storing and manipulating it is prohibitive when the number of variables is large.
- To solve this problem, we store a *modified* version of $\mathbf{Q}_k$ implicitly, by storing a certain number m of the vector pairs $\{\mathbf{s}_i, \mathbf{y}_i\}$. The product $\mathbf{Q}_k \nabla f_k$ can be obtained by performing a sequence of inner products and vector summations involving ∇f_k and the pairs $\{\mathbf{s}_i, \mathbf{y}_i\}$.

Limited-Memory Quasi-Newton Methods

Recall and expand the BFGS update:

$$
\begin{aligned}
\mathbf{Q}_k &= \mathbf{V}_{k-1}^T \mathbf{Q}_{k-1} \mathbf{V}_{k-1} + \rho_{k-1} \mathbf{s}_{k-1} \mathbf{s}_{k-1}^T \\
&= \mathbf{V}_{k-1}^T \mathbf{V}_{k-2}^T \mathbf{Q}_{k-2} \mathbf{V}_{k-2} \mathbf{V}_{k-1} + \rho_{k-2} \mathbf{V}_{k-2} \mathbf{s}_{k-2} \mathbf{s}_{k-2}^T \mathbf{V}_{k-1} + \rho_{k-1} \mathbf{s}_{k-1} \mathbf{s}_{k-1}^T \\
&= \left(\mathbf{V}_{k-1}^T \mathbf{V}_{k-2}^T \cdots \mathbf{V}_{k-m}^T \right) \mathbf{Q}_{k-m} \left(\mathbf{V}_{k-m} \mathbf{V}_{k-m+1} \cdots \mathbf{V}_{k-1} \right) \\
&\quad + \rho_{k-m} \left(\mathbf{V}_{k-1}^T \cdots \mathbf{V}_{k-m+1}^T \right) \mathbf{s}_{k-m} \mathbf{s}_{k-m}^T \left(\mathbf{V}_{k-m+1} \cdots \mathbf{V}_{k-1} \right) \\
&\quad + \rho_{k-m+1} \left(\mathbf{V}_{k-1}^T \cdots \mathbf{V}_{k-m+2}^T \right) \mathbf{s}_{k-m+1} \mathbf{s}_{k-m+1}^T \left(\mathbf{V}_{k-m+2} \cdots \mathbf{V}_{k-1} \right) \\
&\quad + \cdots \\
&\quad + \rho_{k-2} \mathbf{V}_{k-1}^T \mathbf{s}_{k-2} \mathbf{s}_{k-2}^T \mathbf{V}_{k-1} + \rho_{k-1} \mathbf{s}_{k-1} \mathbf{s}_{k-1}^T .
\end{aligned}
$$

In L-BFGS, we replace $\mathbf{Q}_{k-m}$ (a dense $d \times d$ matrix) with some sparse matrix $\mathbf{Q}_k^0$, e.g., a diagonal matrix. Thus, $\mathbf{Q}_k$ can be constructe using the most recent $m \ll d$ pairs $\{\mathbf{s}_i, \mathbf{y}_i\}_{i=k-m}^{k-1}$. That is

Limited-Memory Quasi-Newton Methods

$$
\begin{aligned}
\mathbf{Q}_k = &\left(\mathbf{V}_{k-1}^T \mathbf{V}_{k-2}^T \cdots \mathbf{V}_{k-m}^T\right) \mathbf{Q}_k^0 \left(\mathbf{V}_{k-m} \mathbf{V}_{k-m+1} \cdots \mathbf{V}_{k-1}\right) \\
&+ \rho_{k-m} \left(\mathbf{V}_{k-1}^T \cdots \mathbf{V}_{k-m+1}^T\right) \mathbf{s}_{k-m} \mathbf{s}_{k-m}^T \left(\mathbf{V}_{k-m+1} \cdots \mathbf{V}_{k-1}\right) \\
&+ \rho_{k-m+1} \left(\mathbf{V}_{k-1}^T \cdots \mathbf{V}_{k-m+2}^T\right) \mathbf{s}_{k-m+1} \mathbf{s}_{k-m+1}^T \left(\mathbf{V}_{k-m+2} \cdots \mathbf{V}_{k-1}\right) \\
&+ \cdots \\
&+ \rho_{k-2} \mathbf{V}_{k-1}^T \mathbf{s}_{k-2} \mathbf{s}_{k-2}^T \mathbf{V}_{k-1} + \rho_{k-1} \mathbf{s}_{k-1} \mathbf{s}_{k-1}^T.
\end{aligned}
$$

- We only need the d-dimensional vector $\mathbf{Q}_k \nabla f(\mathbf{x}_k)$ to update $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{Q}_k \nabla f(\mathbf{x}_k)$.
- We only stor the vectors $\{\mathbf{s}_i, \mathbf{y}_i\}_{i=k-m}^{k-1}$ from which $\mathbf{Q}_k \nabla f(\mathbf{x}_k)$ can be computed using only vector-vector multiplications.
- A popular choice for $\mathbf{Q}_k^0$ is $\mathbf{Q}_k^0 = \gamma_k \mathbf{I}$, where $\gamma_k = \frac{\mathbf{s}_{k-1}^T \mathbf{y}_{k-1}}{\mathbf{y}_{k-1}^T \mathbf{y}_{k-1}}$. This choice appears to be quite effective in practice.

Limited-Memory Quasi-Newton Methods

Algorithm 1 L-BFGS two-loop recursion

set $\mathbf{q} = \nabla f(\mathbf{x}_k)$ want to compute $\mathbf{Q}_k \nabla f(\mathbf{x})$

for $i = k-1, k-2, \ldots, k = m$ do

$\alpha_i = \rho_i \mathbf{s}_i^T \mathbf{q}$

$\mathbf{q} = \mathbf{q} - \alpha_i \mathbf{y}_i$ ▷ RHS = $\mathbf{q} - \rho_i \mathbf{s}_i^T \mathbf{q} \mathbf{y}_i = \underbrace{\left(\mathbf{I} - \rho_i \mathbf{y}_i \mathbf{s}_i^T\right)}_{\mathbf{V}_i} \mathbf{q}$

end for

$\mathbf{r} = \mathbf{Q}_k^0 \mathbf{q}$

for $i = k - m$ to $k - 1$: do

$\beta = \rho_i \mathbf{y}_i^T \mathbf{r}$

$\mathbf{r} = \mathbf{r} + \mathbf{s}_i(\alpha_i - \beta)$ ▷ RHS = $\mathbf{r} + \rho_i \alpha_i - \rho_i \mathbf{y}_i^T \mathbf{r} \mathbf{s}_i = \underbrace{\left(\mathbf{I} - \rho_i \mathbf{s}_i \mathbf{y}_i^T\right)}_{\mathbf{V}_i^T} \mathbf{r} + \rho_i \alpha_i$

end for

return $\mathbf{r}$ ▷ which equals $\mathbf{Q}_k \nabla f(\mathbf{x}_k)$

Limited-Memory Quasi-Newton Methods

Require: $\mathbf{x}_0$, m, ε_G

$k = 0$

while $\|\nabla f(\mathbf{x}_k)\| \leq \varepsilon_G$ do

Choose $\mathbf{Q}_k^0$

$\mathbf{p}_k = -\mathbf{Q}_k \nabla f(\mathbf{x}_k)$, where $\mathbf{Q}_k \nabla f(\mathbf{x}_k)$ is compute using Algorithm 1

$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$, where α_k satisfies Wolfe Conditions

if $k > m$: then

discard $\{\mathbf{s}_{k-m}, \mathbf{y}_{k-m}\}$ from storage

end if

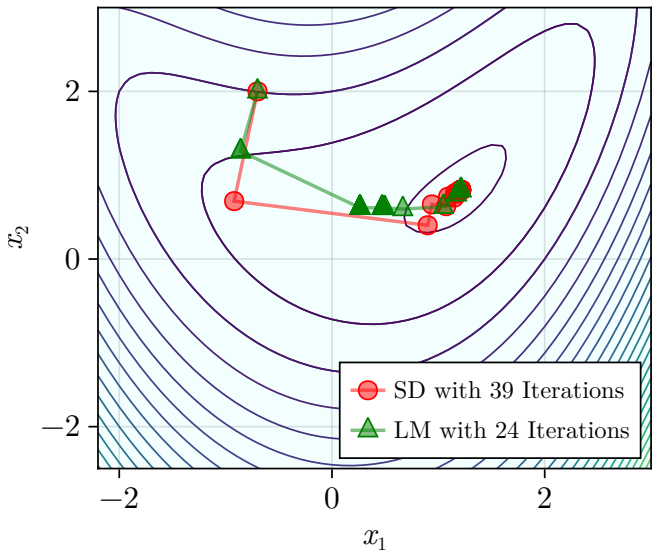
Compute and store $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$ and $\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1} - \nabla f(\mathbf{x}_k)$

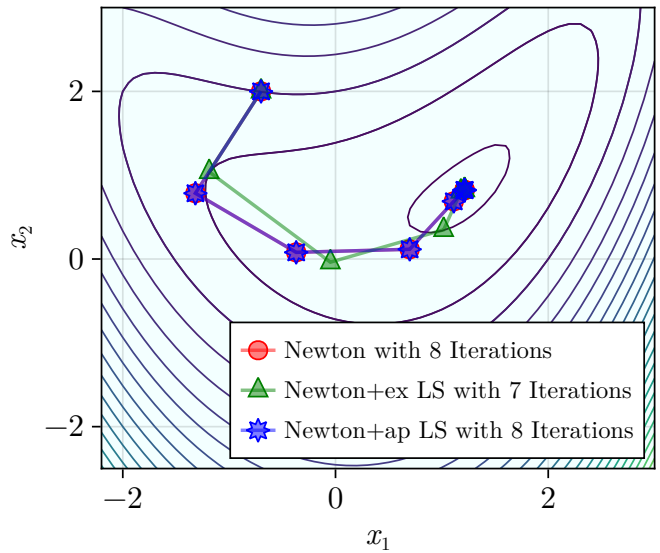
$k = k + 1$

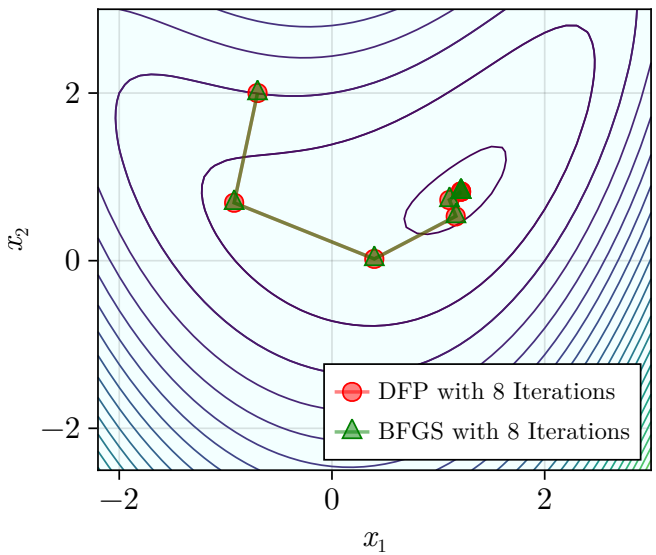
end while

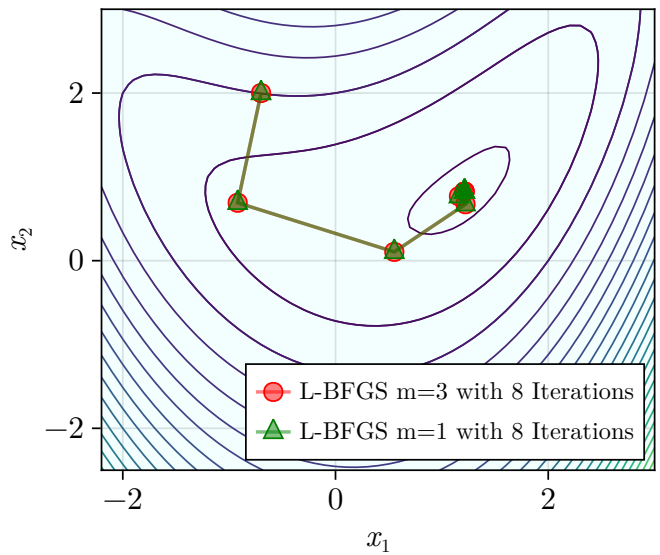
Compare Four Methods

Minimizing of bean function $f(x_1, x_2) = (1 - x_1)^2 + (1 - x_2)^2 + 0.5(2x_2 - x_1^2)^2$



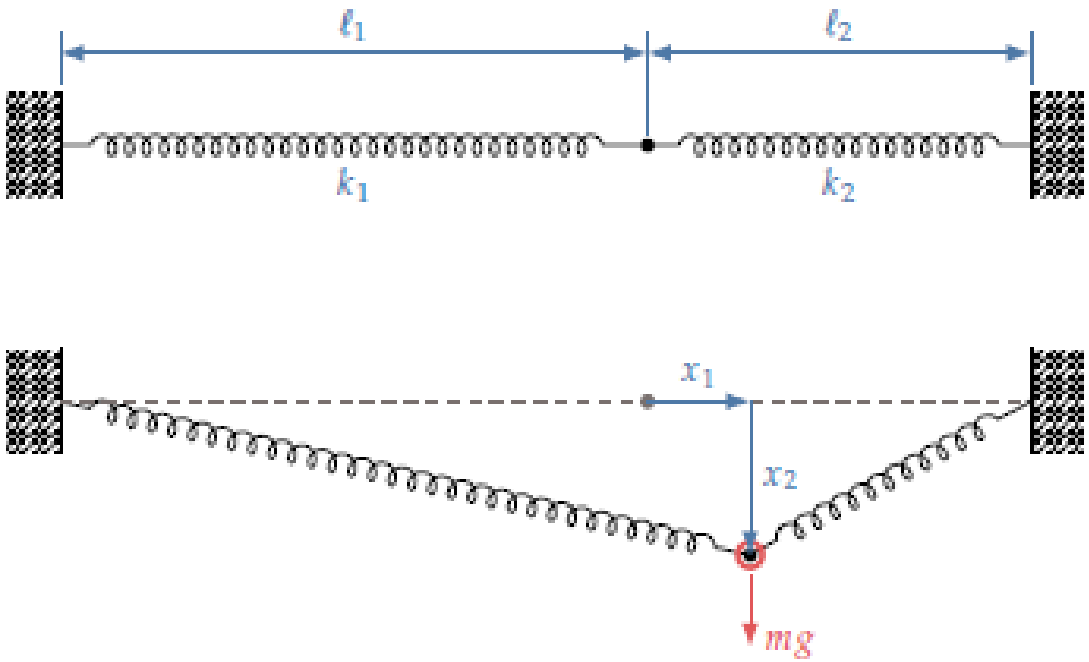






Compare Four Methods

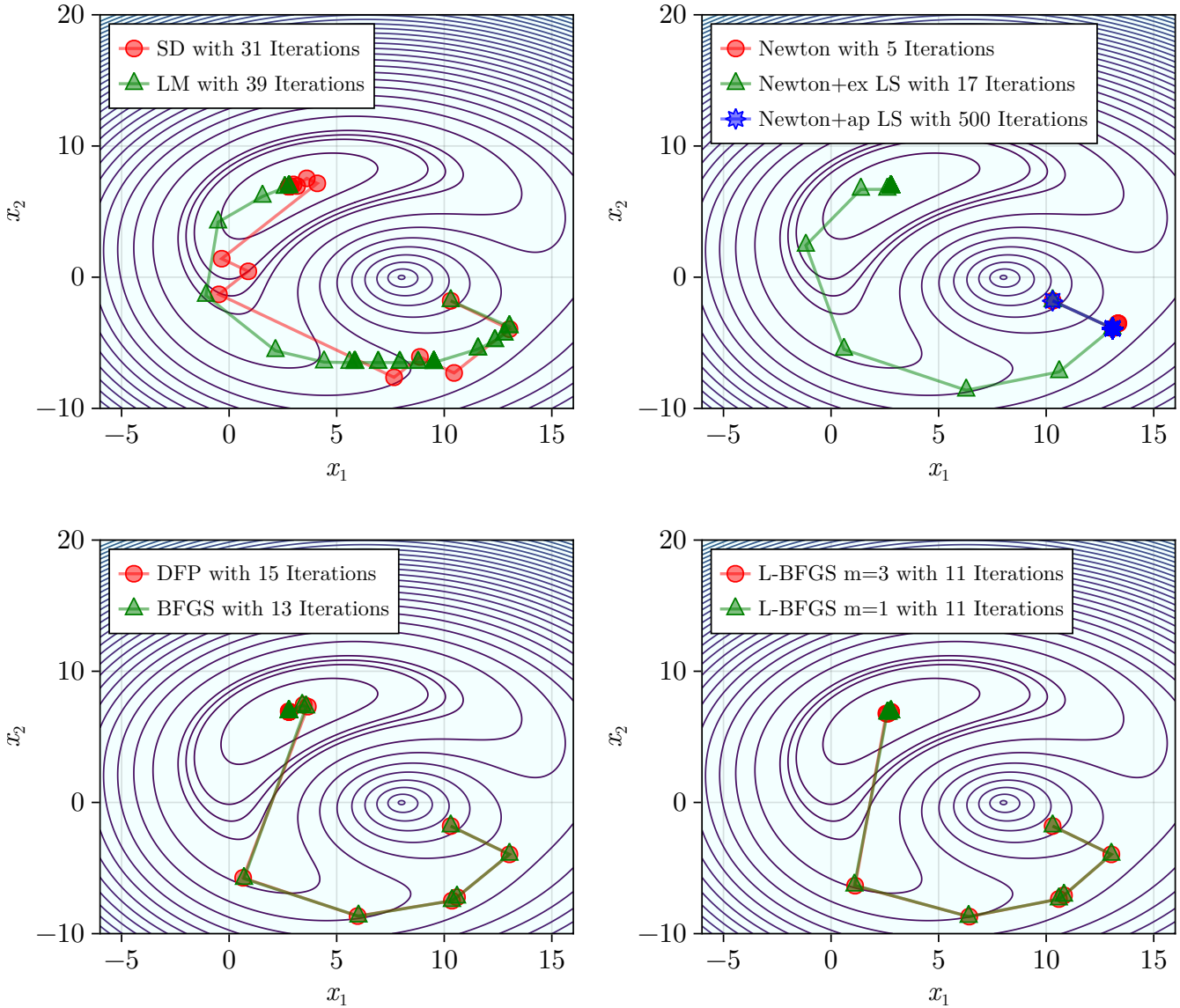
Minimizing the total potential energy for a spring system:



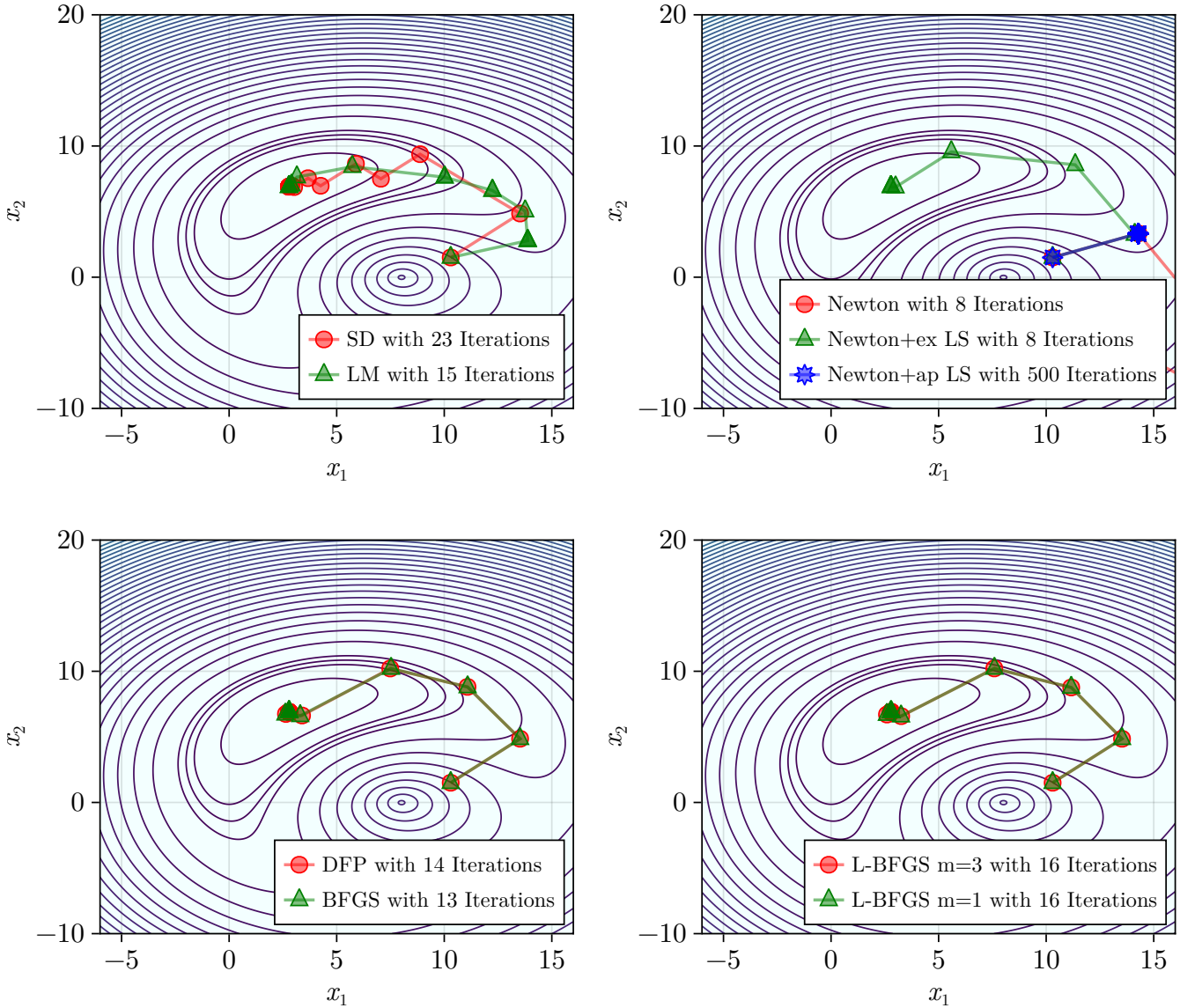
$$\underset{x_1,x_2}{\text{minimize}} \quad \frac{1}{2}k_1\left(\sqrt{(l_1+x_1)^2+x_2^2}-l_1\right)^2+\frac{1}{2}k_2\left(\sqrt{(l_2-x_1)^2+x_2^2}-l_2\right)^2-mgx_2$$

By letting $l_1 = 12, l_2 = 8$, $k_1 = 1$, $k_2 = 10$, $mg = 7$ (with appropriate units).

Compare Four Methods

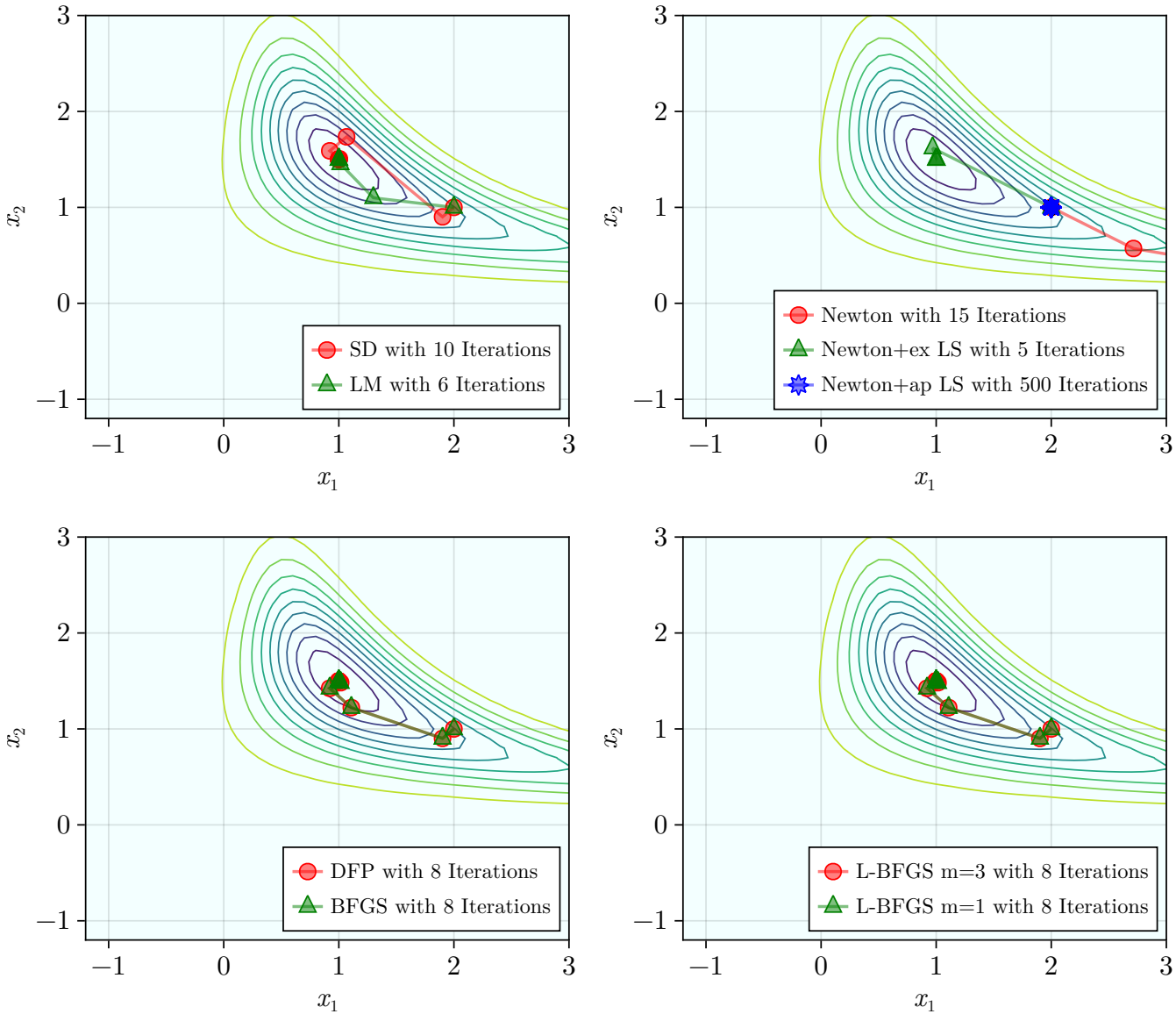


Compare Four Methods



Compare Four Methods

Minimizing of bean function $f(x_1, x_2) = -e^{(-(x_1 x_2 - 1.5)^2 - (x_2 - 1.5)^2)}$



Reference

1. Joaquim R. R. A. Martins, and Andrew Ning, "*Engineering Design Optimization*," Cambridge University Press, 2021.
2. Jorge Nocedal, and Stephen J. Wright, "*Numerical Optimization*," 2nd, Springer, 2026
3. Mykel J. Kochenderfer, and Tim A. Wheeler, "*Algorithms for Optimization*," The MIT Press, 2019