5D – Optimization: Multivariable Optimization

Derek Nowrouzezahrai
derek@cim.mcgill.ca

Multivariable Input – Scalar Functions

Identifying the extrema for functions of *many* input variables, but only a *single* output variable, i.e. $f : \mathbb{R}^m \to \mathbb{R}$ is more straightforward than for those with *many* output variables, i.e., $\mathbf{f} : \mathbb{R}^m \to \mathbb{R}^n$

- i.e., min/max of real-valued *vs.* vector-valued output
  - the latter depends on, e.g., a choice of norm

Optimizing such multivariate real-valued functions will be focus of the remainder of this module of the course:

$$\mathbf{x}_* = \underset{\mathbf{x}}{\operatorname{argmin}}\, f(\mathbf{x})$$

Newton for Multivariate Optimization

Newton's 1D optimization extends to multivariate inputs

- start again from a 2nd-order expansion of (twice differentiable) $f(\mathbf{x})$

$$f(\mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T \cdot (\mathbf{x} - \mathbf{x}_0) + (\mathbf{x} - \mathbf{x}_0)^T \cdot \mathbf{H}_{f(\mathbf{x}_0)} \cdot (\mathbf{x} - \mathbf{x}_0) \Big/ 2 = \bar{f}(\mathbf{x})$$

- we iteratively estimate stationary points of f by differentiating $\bar{f}$ w.r.t. $\mathbf{x}$, setting the result to 0 and solving for the extrema as $\mathbf{x}_\star$
- here, the *approximation* $\bar{f}$ is an m-dimensional *quadratic* manifold
    - tangent to f at $\mathbf{x} = \mathbf{x}_0$ and with vertex $\mathbf{x}_\star = \mathbf{x}_0 - [\mathbf{H}_{f(\mathbf{x}_0)}]^{-1} \nabla f(\mathbf{x}_0)$

Newton for Multivariate Optimization

The Newton estimates are updated, analogously and iteratively, as

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

$$\mathbf{H}_{f(\mathbf{x})} = \begin{pmatrix} \frac{\partial^2 f}{\partial^2 x_1} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_m} \\ \vdots & & \vdots \\ \frac{\partial^2 f}{\partial x_m \partial x_1} & \cdots & \frac{\partial^2 f}{\partial^2 x_m} \end{pmatrix}$$

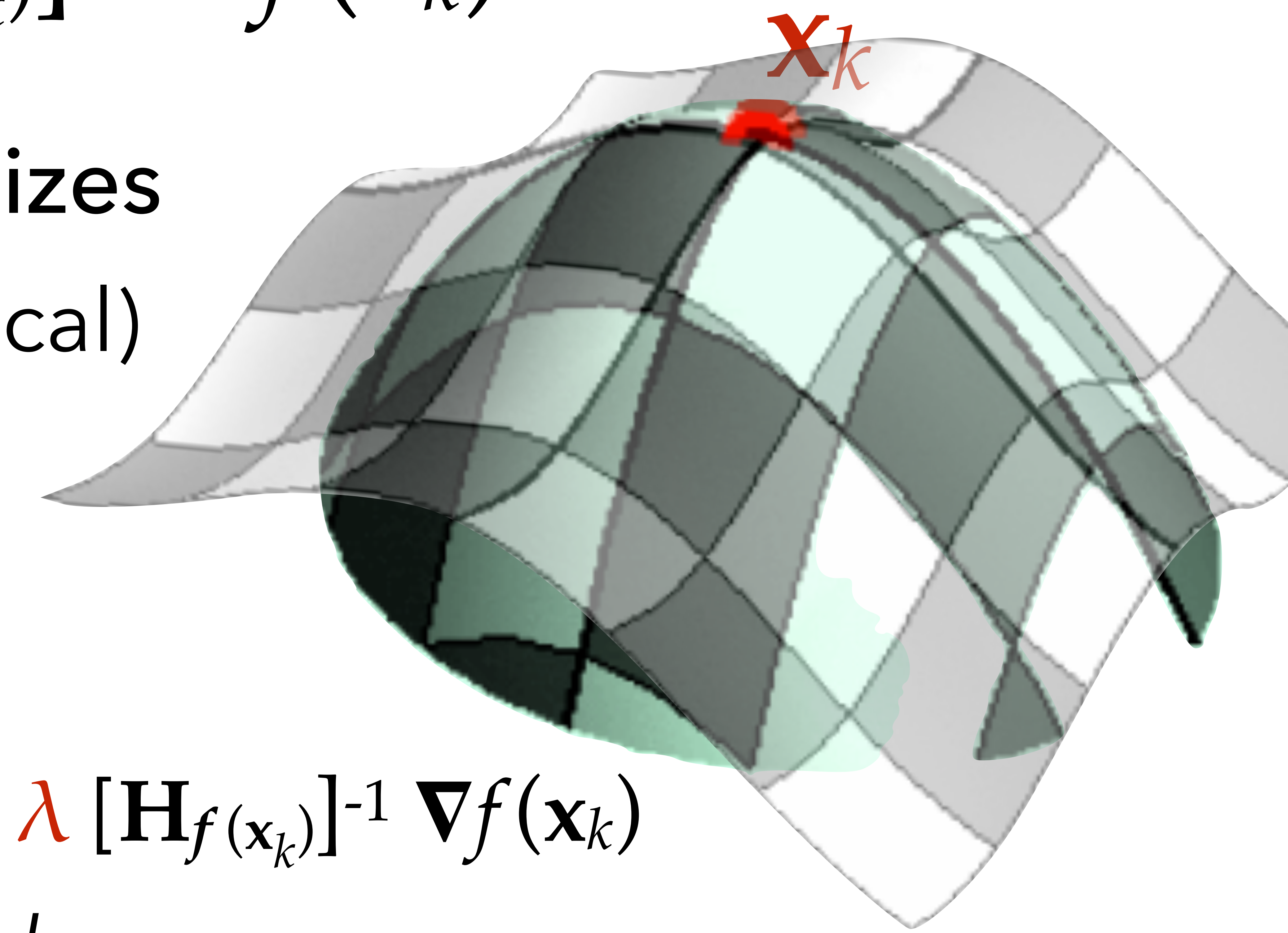
Need to form **and** invert the Hessian!

- $\mathbf{H}_{f(\mathbf{x}_k)}$ is symmetric: how would you compute $[\mathbf{H}_{f(\mathbf{x}_k)}]^{-1} \nabla f(\mathbf{x}_k)$
- we can interpret the iteration as moving from $\mathbf{x}_k$ in the direction of $-[\mathbf{H}_{f(\mathbf{x}_k)}]^{-1} \nabla f(\mathbf{x}_k)$ (and by a distance equal to its length)
- this method converges *quadratically* in the number of iterations, but each iteration can be very costly

Newton for Multivariate Optimization

$$f(\mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T \cdot (\mathbf{x} - \mathbf{x}_0) + (\mathbf{x} - \mathbf{x}_0)^T \cdot \mathbf{H}_{f(\mathbf{x}_0)} \cdot (\mathbf{x} - \mathbf{x}_0) \Big/ 2 = \bar{f}(\mathbf{x})$$

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



Near-singular $\mathbf{H}_{f(\mathbf{x}_k)}$ can lead to large step sizes

- this can be a problem: the quality of the (local) quadratic approximation of the function decreases as we move away from $\mathbf{x}_k$
- can address this issue by, e.g., either:
  - "dampening" the step size, as $\mathbf{x}_{k+1} = \mathbf{x}_k - \lambda [\mathbf{H}_{f(\mathbf{x}_k)}]^{-1} \nabla f(\mathbf{x}_k)$
  - performing a (more expensive) *line search*

Line Search

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

Every iteration takes *one step* along the vector $-[\mathbf{H}_{f(\mathbf{x}_k)}]^{-1}\nabla f(\mathbf{x}_k)$

- instead of taking one (possibly large) step, we can decide to "walk" along this direction until we hit a minimum
- this is commonly referred to as performing a *line search*

Line search amounts to solving a 1D minimization problem, since we're walking along a 1D slice $g_k(\lambda)$ of the multivariate $f(\mathbf{x})$,

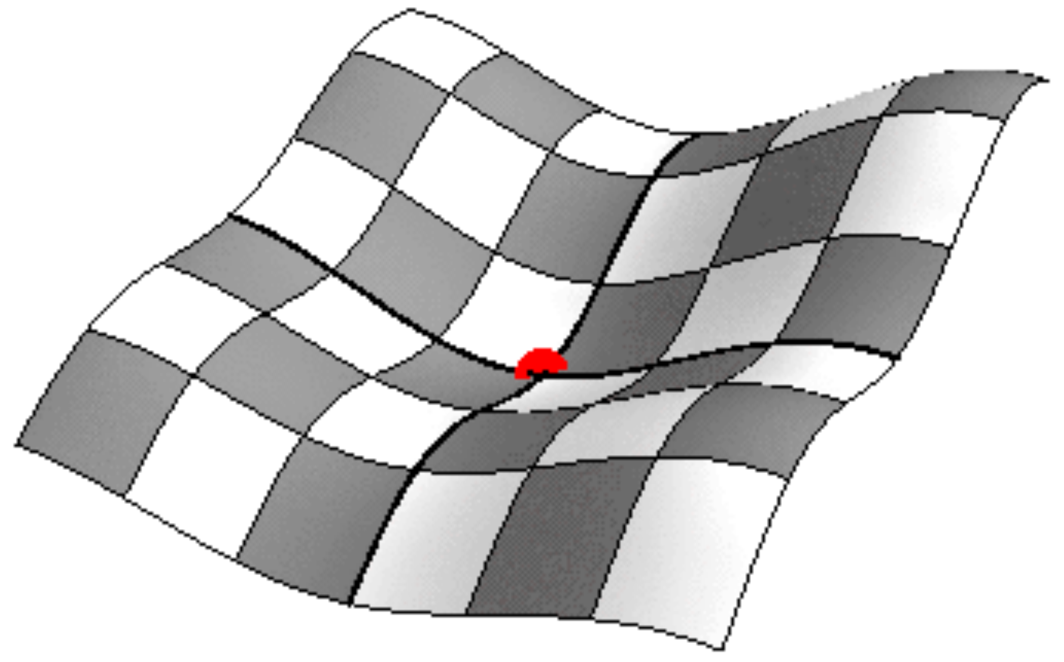
$\lambda^* = \underset{\lambda}{\operatorname{argmin}} \underbrace{f\left(\mathbf{x}_k - \lambda \left[\mathbf{H}_{f(\mathbf{x}_k)}\right]^{-1} \nabla f(\mathbf{x}_k)\right)}_{g_k(\lambda)}$ and $\mathbf{x}_{k+1} = \mathbf{x}_k - \lambda^*[\mathbf{H}_{f(\mathbf{x}_k)}]^{-1}\nabla f(\mathbf{x}_k)$

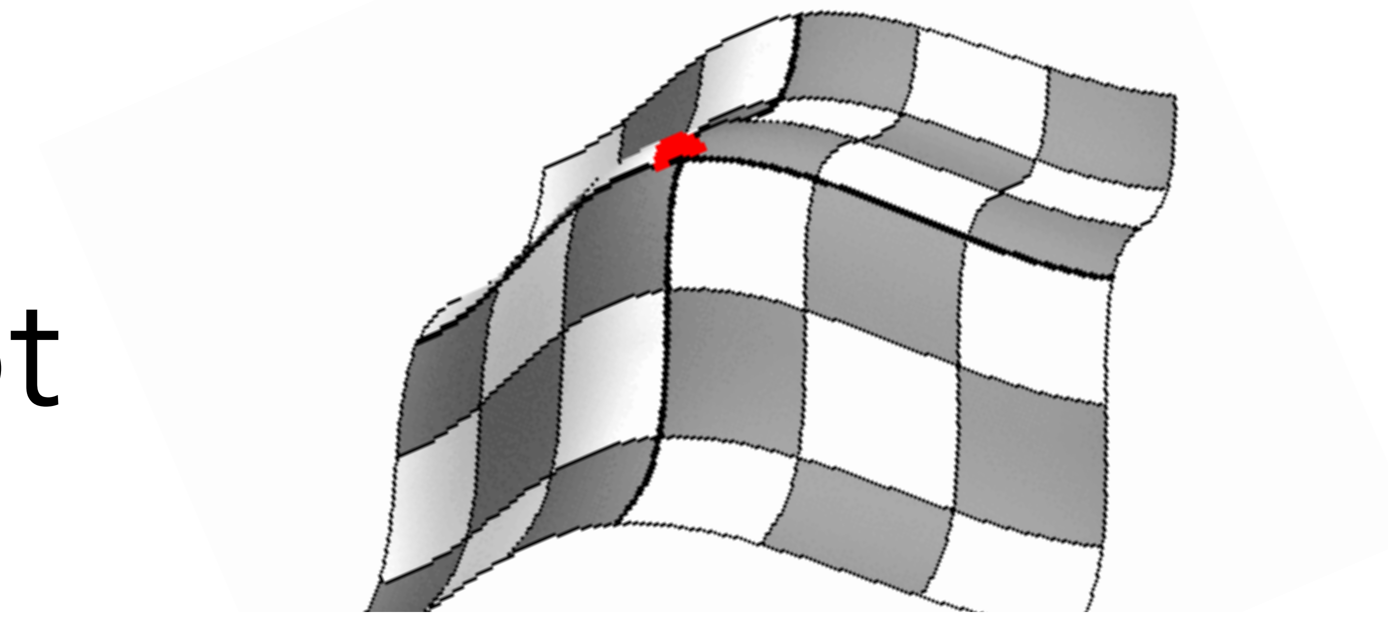
- you can apply a 1D optimization method to the λ^* optimization problem

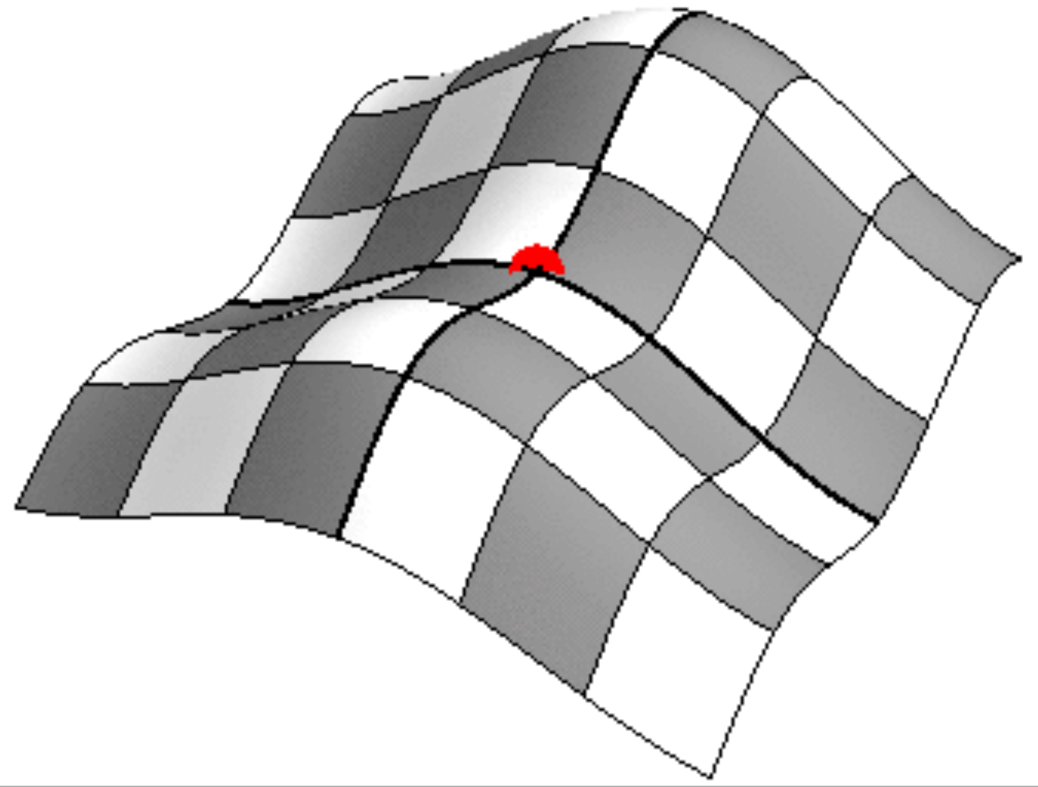
Newton Optimization – Convergence

Recall that, in the 1D setting, seeking $f'(x_\mathrm{r}) = 0$ was a *necessary* – but not *sufficient* – condition for minimization

- similarly in the $f : \mathbb{R}^m \to \mathbb{R}$ setting, finding an $\mathbf{x}_\mathrm{r}$ root that satisfies $\boldsymbol{\nabla} f(\mathbf{x}_\mathrm{r}) = 0$ is necessary but not sufficient to identify an extremum
- here, it is the form of the Hessian $\mathbf{H}_{f(\mathbf{x}_\mathrm{r})}$ that classifies the root

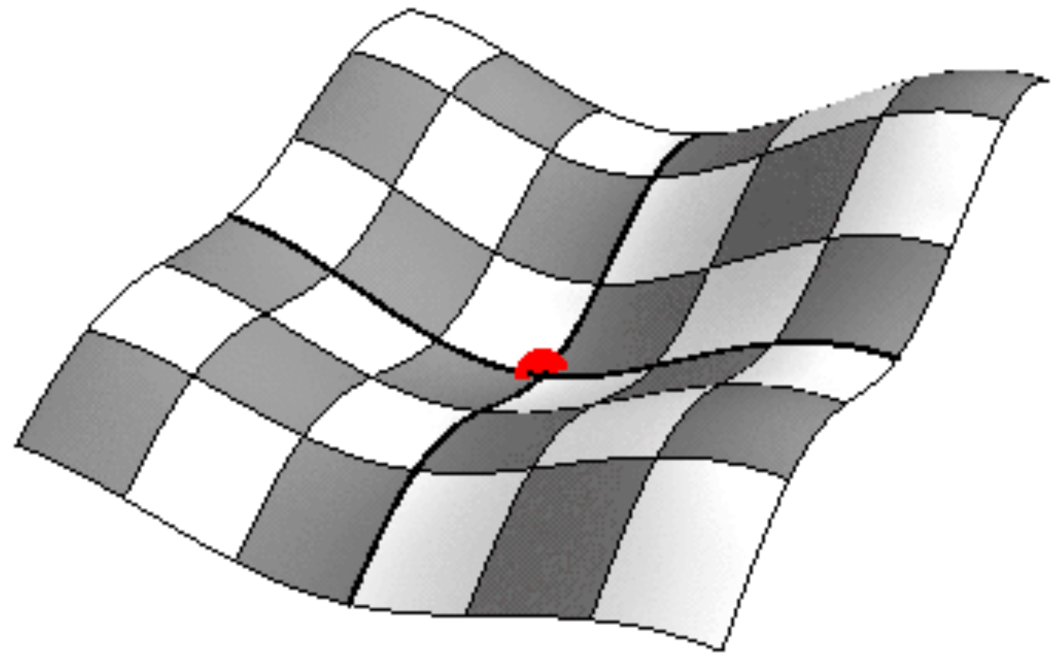


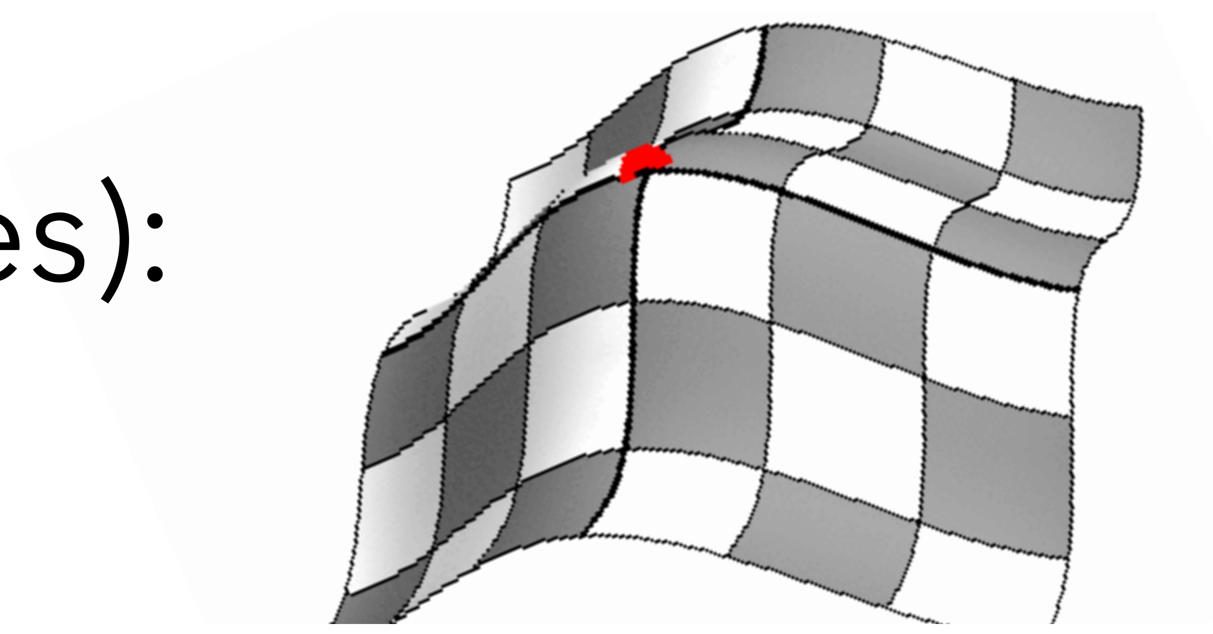


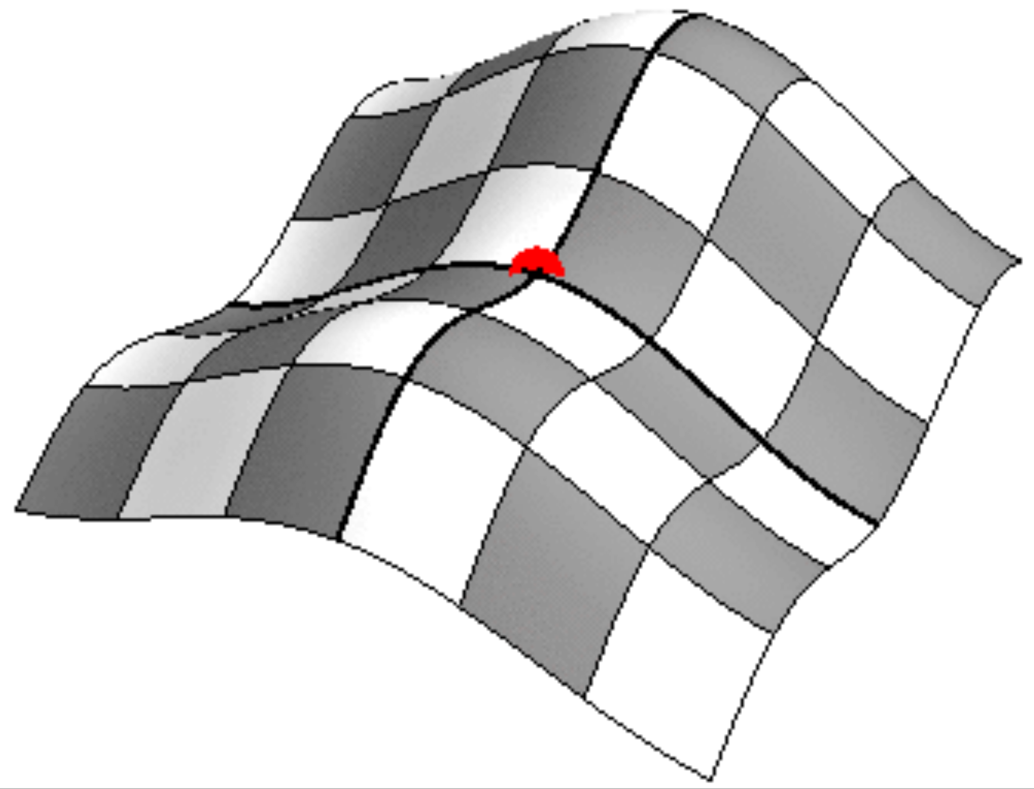


Newton Optimization – Convergence

- here, it is the form of the Hessian $\mathbf{H}_f(\mathbf{x}_\mathrm{r})$ that classifies the root
  - if it's positive definite (i.e., all positive eigenvalues): <u>local minimum</u>
  - if it's negative definite (i.e., all negative eigenvalues): <u>local maximum</u>
  - neither (i.e., eigenvalues with mixed signs): <u>saddle point</u>
  - can also be singular, but we won't get into the pathological cases







Newton Optimization – Summary

Newton's multivariate optimization method, for functions $f : \mathbb{R}^m \rightarrow \mathbb{R}$, *seeks out* a zero of the gradient $\nabla f(\mathbf{x})$ by successively stepping (or walking, with line search) along different directions

- these directions are guided by both the Hessian and gradient of the function, sampled at points in the search domain
- these 1st- and 2nd-order gradients model a quadratic approximation of the underlying manifold f
- the main costs are in forming & inverting the Hessian, each iteration

Approximating Hessians

We can approximate the inverse of the Hessian in Newton's multivariable optimization method

- the idea of numerically approximating higher-order information – to trade accuracy for performance – isn't new:
  - secant approximated derivatives in higher-order root finding
  - quasi-Newton methods (e.g., Broyden's) approximate the inverse of the Jacobian in Newton's vector-valued root finding method
  - secant for 1D optimization approximates 2nd-order expansions for extremum search
    - we'll extend this idea to multivariable scalar-valued optimization

Approximating Hessians

Newton iterates to zeros of $\nabla f(\mathbf{x})$ to optimize $f : \mathbb{R}^m \to \mathbb{R}$ as

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \lambda^* \left[\mathbf{H}_{f(\mathbf{x}_k)}\right]^{-1} \nabla f(\mathbf{x}_k)$$

where the step size scale λ^* can* be set, e.g., with line search

- the two costly operations in this update – other than how λ^* is chosen – are computing and inverting the Hessian $\mathbf{H}_{f(\mathbf{x}_k)}$ at $\mathbf{x}_k$

In a similar spirit to Broyden's approximation of the Jacobian in quasi-Newton multivariable root finding problems, two methods try reduce the cost Newton's multivariate optimization method

Approximate Hessians #1 – DFP

One such approach – the Davidon-Fletcher-Powell (DFP) scheme – reduces the cost of computing the Hessian $\mathbf{H}_{f(\mathbf{x}_k)}$ every iteration by computing a (cheaper) iterate to update an estimate of the Hessian alongside the extremum's iterate

- here, we alternate between updating $\mathbf{x}_{k+1}$ and $\mathbf{H}_{f(\mathbf{x}_{k+1})}$, and the cost of forming the Hessian is reduced significantly
- the approximate Hessian $\mathbf{B}_{f(\mathbf{x}_{k+1})} \approx \mathbf{H}_{f(\mathbf{x}_{k+1})}$ still needs to be inverted

Without going into the full derivation, we highlight the general steps that are taken to derive the iterative scheme for $\mathbf{B}_{f(\mathbf{x}_{k+1})}$

Approximate Hessians #1 – DFP

Note: the Hessian $\mathbf{H}_{f(\mathbf{x}_{k+1})}$ is the all-pairs and element-wise derivative of the gradient $\nabla f(\mathbf{x}_{k+1})$, and so DFP begin by imposing a finite difference-like condition on the form of their Hessian approximation, using previous iterates, as

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

In addition to this condition, DFP require that $\mathbf{B}_{f(\mathbf{x}_{k+1})}$ be:

- symmetric, as $\mathbf{H}_{f(\mathbf{x}_k)}$, and
- symmetric positive definite – so that the corresponding extremum estimate corresponds to a *minimum* (see slide on *Convergence*)

Approximate Hessians #1 – DFP

The SPD condition imposes the following implicit constraint between $(\mathbf{x}_{k+1} - \mathbf{x}_k)$ and $\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$:

$$(\mathbf{x}_{k+1} - \mathbf{x}_k) \cdot (\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)) > 0$$

After which they solve for $\mathbf{B}_{f(\mathbf{x}_{k+1})}$ as

$$\min_{\mathbf{B}_{f(\mathbf{x}_{k+1})}} \|\mathbf{B}_{f(\mathbf{x}_{k+1})} - \mathbf{B}_{f(\mathbf{x}_k)}\|$$

such that $\mathbf{B}_{f(\mathbf{x}_{k+1})} = [\mathbf{B}_{f(\mathbf{x}_{k+1})}]^{\mathrm{T}}$ and $\mathbf{B}_{f(\mathbf{x}_{k+1})}(\mathbf{x}_{k+1} - \mathbf{x}_k) = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$

Approximate Hessians #2 – BFGS

This DFP scheme still requires that $\mathbf{B}_{f(\mathbf{x}_{k+1})}$ be inverted

An alternative approach – Broyden-Fletcher-Goldfarb-Shanno (BFGS) – instead seeks an iterative approach to *directly update the* ***inverse*** *of the Hessian,* $[\mathbf{H}_{f(\mathbf{x}_k)}]^{-1}$

- the exposition in [Solomon; section 9.4.3] is informative and points to relevant references

BFGS is among the most used optimization methods in practice

- a *limited-memory* BFGS (L-BFGS) variant side-steps the storage requirements for the inverse Hessian with a clever buffering trick

Optimization – Convergence

Many real-world functions we seek to optimize can have complex manifolds ("landscapes", "shapes") – e.g., many local extrema riddled about the global landscape

- any iterative method reliant on local approximations of the manifold can get "stuck"
  - diverging or "ping-ponging" about the landscape
  - converging into a local minimum
- hybrid methods exist but – usually – there's no free lunch; trade accuracy/stability for performance