Interpolation
Iteration and Convergence
Solving One Equation

Solving Many Equations
Finding the Best: Optimization in 1D
Optimization in $n$ Dimensions
Recap: Interpolation

Starting point: Looking for a linear combination of functions \( \varphi_i \) to hit given data points \((x_i, y_i)\).

Interpolation becomes solving the linear system:

\[
y_i = f(x_i) = \sum_{j=0}^{N_{\text{func}}} \alpha_j \varphi_j(x_i) \quad \leftrightarrow \quad V\alpha = y.
\]

Want unique answer: Pick \( N_{\text{func}} = N \rightarrow V \) square. 

\( V \) is called the (generalized) Vandermonde matrix.

Main lesson:

\[
V \text{ (coefficients)} = \text{(values at nodes)}.
\]
We have so far always used monomials \((1, x, x^2, x^3, \ldots)\) and equispaced points for interpolation. It turns out that this has significant problems.

**Demo:** Monomial interpolation
Interpolation: Choosing Basis Function and Nodes

Both function basis and point set are under our control. What do we pick?

Ideas for basis functions:

- Monomials $1, x, x^2, x^3, x^4, \ldots$
- Functions that make $V = I \rightarrow \text{‘Lagrange basis’}$
- Functions that make $V$ triangular $\rightarrow \text{‘Newton basis’}$
- Splines (piecewise polynomials)
- Orthogonal polynomials
- Sines and cosines
- ‘Bumps’ (‘Radial Basis Functions’)

Ideas for nodes:

- Equispaced
- ‘Edge-Clustered’ (so-called Chebyshev/Gauss/… nodes)
What caused monomials to have a terribly conditioned Vandermonde?

Being close to linearly dependent.

What’s a way to make sure two vectors are *not* like that?

Orthogonality

But polynomials are functions!

How can those be orthogonal? Just need something like a dot product!

\[ f \cdot g = \sum_{i=1}^{n} f_i g_i = \langle f, g \rangle \]

\[ \langle f, g \rangle = \int_{-1}^{1} f(x)g(x)\,dx \]
Orthogonal then just means $\langle f, g \rangle = 0$.

**Q:** How can we find an orthogonal basis?

**A:** Apply Gram-Schmidt to the monomials.

Obtained Legendre polynomials.

**Demo:** Orthogonal polynomials

But how can I practically compute the Legendre polynomials?

→ DLMF, Chapter on orthogonal polynomials

Main lessons:

- There exist three-term recurrences. Easy to apply if you know the first two.
There is a whole zoo of polynomials there, depending on the weight function $w$ in the (generalized) inner product:

$$\langle f, g \rangle = \int w(x)f(x)g(x)\,dx.$$ 

Some sets of orthogonal polynomials live on intervals other than $(-1, 1)$.
Another Family of Orthogonal Polynomials: Chebyshev

Three equivalent definitions:

- Result of Gram-Schmidt with weight $\frac{1}{\sqrt{1 - x^2}}$

What is that weight?

- $1 / (\text{Half circle}), \text{i.e. } x^2 + y^2 = 1, \text{ with } y = \sqrt{1 - x^2}$

- $T_k(x) = \cos(k \cos^{-1}(x))$

- $T_k(x) = 2xT_k(x) - T_{k-1}(x)$

Demo: Chebyshev interpolation part I

What are good nodes to use with Chebyshev polynomials?

The answer would be particularly simple if the nodes were $\cos(*)$. So why not $\cos$ (equispaced)?

Might get

$$x_i = \cos \left( \frac{i}{k} \pi \right) \quad (i = 0, 1, \ldots, k)$$
Chebyshev Nodes

Might also consider zeros (instead of roots) of $T_k$:

$$x_i = \cos\left(\frac{2i + 1}{2k} \pi\right) \quad (i = 1 \ldots, k).$$

The Vandermonde for these (with $T_k$) can be applied in $O(N \log N)$ time, too.

It turns out that we were still looking for a good set of interpolation nodes.

We came up with the criterion that the nodes should bunch towards the ends. Do these do that?

Yes.

**Demo:** Chebyshev interpolation part II
Calculus on Interpolants

Suppose we have an interpolant \( \tilde{f}(x) \) with \( f(x_i) = \tilde{f}(x_i) \) for \( i = 1, \ldots, n \):

\[
\tilde{f}(x) = \alpha_1 \varphi_1(x) + \cdots + \alpha_n \varphi_n(x)
\]

How do we compute the derivative of \( \tilde{f} \)?

\[
\tilde{f}'(x) = \alpha_1 \varphi'_1(x) + \cdots + \alpha_n \varphi'_n(x).
\]

Easy because interpolation basis \( (\varphi_i) \) is known.

Suppose we have function values at nodes \( (x_i, f(x_i)) \) for \( i = 1, \ldots, n \) for a function \( f \). If we want \( f'(x_i) \), what can we do?

\( f'(x_i) \): Hard to get
\( \tilde{f}'(x_i) \): Easy to get
Calculus on Interpolants (II)

So:

1. Compute coefficients $\alpha = V^{-1} f$, where $f = (f(x_1), \ldots, f(x_n))^T$.

2. Build generalized Vandermonde with derivatives of basis:

$$V' = \begin{pmatrix} \varphi'_1(x_1) & \cdots & \varphi'_n(x_1) \\ \vdots & \ddots & \vdots \\ \varphi'_1(x_n) & \cdots & \varphi'_n(x_n) \end{pmatrix}.$$ 

3. Compute

$$V' \alpha = \begin{pmatrix} \alpha_1 \varphi'_1(x_1) + \cdots + \alpha_n \varphi'_n(x_1) \\ \tilde{f}'(x_1) \\ \vdots \\ \tilde{f}'(x_n) \end{pmatrix} = \begin{pmatrix} \tilde{f}'(x_1) \\ \vdots \\ \tilde{f}'(x_n) \end{pmatrix}.$$
Calculus on Interpolants (III)

All in one step: \( \tilde{f}' = V'V^{-1}f \).
In other words: \( V'V^{-1} \) is a matrix to apply a derivative!
We call \( D = V'V^{-1} \) a differentiation matrix.
About Differentiation Matrices

How could you find coefficients of the derivative?

$$\alpha' = V^{-1}V'V^{-1} f.$$ 

Give a matrix that finds the second derivative.

$$V'V^{-1}V'V^{-1}.$$
**Demo:** Taking derivatives with Vandermonde matrices
Finite Difference Formulas

It is possible to use the process above to find ‘canned’ formulas for taking derivatives. Suppose we use three points equispaced points \((x - h, x, x + h)\) for interpolation (i.e. a degree-2 polynomial).

- What is the resulting differentiation matrix?
- What does it tell us?

\[
D = V'V^{-1} = \begin{pmatrix}
\cdots & \cdots & \cdots \\
-1/2h & 0 & 1/2h \\
\cdots & \cdots & \cdots 
\end{pmatrix}
\]

(Can find the dependence on \(h\) by varying \(h\) and watching the entries.)

When we apply that, we get

\[
V'V^{-1} \begin{pmatrix}
f(x - h) \\
f(x) \\
f(x + h)
\end{pmatrix} = \begin{pmatrix}
\cdots \\
\frac{f(x+h) - f(x-h)}{2h} \\
\cdots
\end{pmatrix}
\]
Finite Difference Formulas (II)

So we can compute an approximate (second-order accurate!) derivative just by using this formula.

Generalizes to more (and non-center) points easily.
Can we use a similar process to compute (approximate) integrals of a function $f$?

The process of computing approximate integrals is called ‘quadrature’. Same idea as derivatives: interpolate, then integrate.

**Have:** interpolant $\tilde{f}(x) = \alpha_1 \varphi_1(x) + \cdots + \alpha_n \varphi_n(x)$ so that $\tilde{f}(x_i) = f(x_i) = y_i$. We’ll call the $x_i$ the quadrature nodes.

**Want:** Integral

$$\int_a^b f(x) \, dx \approx \int_a^b \tilde{f}(x) \, dx = \int_a^b \alpha_1 \varphi_1(x) + \cdots + \alpha_n \varphi_n(x) \, dx$$

$$= \alpha_1 \int_a^b \varphi_1(x) \, dx + \cdots + \alpha_n \int_a^b \varphi_n(x) \, dx.$$ 

**Idea:** $d_i = \int_a^b \varphi_i(x) \, dx$ can be computed ahead of time, so that

$$\int_a^b \tilde{f}(x) \, dx = \alpha_1 d_1 + \cdots + \alpha_n d_n = d^T \alpha = d^T (V^{-1}y) = (d^T V^{-1}) y.$$
Can call $\mathbf{w} := V^{-T}d$ the quadrature weights and compute

$$\int_{a}^{b} \tilde{f}(x)dx = \mathbf{w}^T \mathbf{y} = \mathbf{w} \cdot \mathbf{y}.$$
Example: Building a Quadrature Rule

**Demo:** Computing the Weights in Simpson’s Rule

Suppose we know

\[ f(x_0) = 2 \quad f(x_1) = 0 \quad f(x_2) = 3 \]

\[ x_0 = 1 \quad x_1 = \frac{1}{2} \quad x_2 = 1 \]

How can we find an approximate integral?

1. Find coefficients

\[ \alpha = V^{-1} \begin{pmatrix} 2 \\ 0 \\ 3 \end{pmatrix}. \]
Example: Building a Quadrature Rule (II)

2. Compute integrals

\[
\begin{align*}
\int_0^1 1 \, dx &= 1 \\
\int_0^1 x \, dx &= \frac{1}{2} \\
\int_0^1 x^2 \, dx &= \left[ \frac{1}{3} x^3 \right]_0^1 = \frac{1}{3}
\end{align*}
\]

3. Combine it all together:

\[
\int_0^1 \tilde{f}(x) \, dx = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} \end{pmatrix} V^{-1} \begin{pmatrix} 2 \\ 0 \\ 3 \end{pmatrix} = \begin{pmatrix} .167 \\ .667 \\ .167 \end{pmatrix} \cdot \begin{pmatrix} f(0) \\ f(1/2) \\ f(1) \end{pmatrix}.
\]

It turns out that this rule has someone’s name attached to it. It’s called Simpson’s rule.
Facts about Quadrature

What does Simpson’s rule look like on $[0, 1/2]$?

\[
\frac{1}{2} \begin{pmatrix}
.167 \\
.667 \\
.167
\end{pmatrix} \cdot \begin{pmatrix}
f(0) \\
f(1/2) \\
f(1)
\end{pmatrix}
\]

What does Simpson’s rule look like on $[5, 6]$?

\[
\begin{pmatrix}
.167 \\
.667 \\
.167
\end{pmatrix} \cdot \begin{pmatrix}
f(5) \\
f(5.5) \\
f(6)
\end{pmatrix}
\]

How accurate is Simpson’s rule?
Facts about Quadrature (II)

**Demo:** Accuracy of Simpson’s rule

- **Quadrature:**

  \[
  \left| \int_{a}^{b} f(x) \, dx - \int_{a}^{b} \tilde{f}(x) \, dx \right| \leq C \cdot h^{n+2}
  \]
  
  (where \( h = b - a \))
  
  (Due to a happy accident, odd \( n \) produce an even smaller error.)

- **Interpolation:**

  \[
  \max_{x \in [a,b]} \left| f(x) - \tilde{f}(x) \right| \leq C \cdot h^{n+1}
  \]

- **Differentiation:**

  \[
  \max_{x \in [a,b]} \left| f'(x) - \tilde{f}'(x) \right| \leq C \cdot h^{n}
  \]

**General lesson:** More derivatives \( \Rightarrow \) Worse accuracy.
Outline

Interpolation
Iteration and Convergence
Solving One Equation
Solving Many Equations
Finding the Best: Optimization in 1D
Optimization in $n$ Dimensions
What is linear convergence? quadratic convergence?

Let $e_k = \hat{x}_k - x$ be the error in the $k$th estimate $\hat{x}_k$ of a desired solution $x$.

An iterative method converges with rate $r$ if

$$ \lim_{k \to \infty} \frac{\|e_{k+1}\|}{\|e_k\|^r} = C \begin{cases} > 0, & r > 1 \text{ is called superlinear convergence.} \\ < \infty, & r = 2 \text{ is called quadratic convergence.} \end{cases} $$

$r = 1$ is called linear convergence.

Examples:

- Power iteration is linearly convergent.
- Rayleigh quotient iteration is quadratically convergent.
About Convergence Rates

**Demo:** Rates of Convergence

Characterize linear, quadratic convergence in terms of the ‘number of accurate digits’.

- Linear convergence gains a constant number of digits each step:
  \[
  \|e_{k+1}\| \leq C \|e_k\|
  \]
  (and $C < 1$ matters!)

- Quadratic convergence doubles the number of digits each step:
  \[
  \|e_{k+1}\| \leq C \|e_k\|^2
  \]
  (Only starts making sense once $\|e_k\|$ is small. $C$ doesn’t matter much.)
Outline

Interpolation
Iteration and Convergence
Solving One Equation

Solving Many Equations
Finding the Best: Optimization in 1D
Optimization in $n$ Dimensions
Solving Nonlinear Equations

What is the goal here?

Solve \( f(x) = 0 \) for \( f : \mathbb{R} \rightarrow \mathbb{R} \).

If looking for solution to \( f(x) = y \), simply consider \( f(x) = \tilde{f}(x) - y \).

*Intuition*: Each of the \( n \) equations describes a surface. Looking for intersections.
Demo: Bisection Method

What’s the rate of convergence? What’s the constant?

Linear with constant $\frac{1}{2}$.
Idea: Approximate $f$ at current iterate using Taylor.

$$f(x_k + h) \approx f(x_k) + f'(x_k)h$$

Now find root of this linear approximation in terms of $h$:

$$f(x_k) + f'(x_k)h = 0 \iff h = -\frac{f(x_k)}{f'(x_k)}.$$ 

So

$$x_0 = \langle \text{starting guess} \rangle$$

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} = g(x_k)$$
What are some **drawbacks** of Newton?

- Convergence argument only good *locally*
  Will see: convergence only local (near root)
- Have to have derivative!
What would Newton without the use of the derivative look like?

Approximate

\[ f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}. \]

So

\[ x_0 = \text{starting guess} \]

\[ x_{k+1} = x_k - \frac{f(x_k)}{f(x_k) - f(x_{k-1})}. \]

Rate of convergence (not shown) is \( \frac{1 + \sqrt{5}}{2} \approx 1.618 \).

**Drawbacks** of Secant:

- Convergence argument only good *locally*
  
  Will see: convergence only local (near root)
- Slower convergence
- Need two starting guesses
**Demo:** Secant Method

**In-class activity:** Nonlinear equations in 1D
Outline

Interpolation
Iteration and Convergence
Solving One Equation

Solving Many Equations
Finding the Best: Optimization in 1D
Optimization in \( n \) Dimensions
Solving Nonlinear Equations

What is the goal here?

Solve $f(x) = 0$ for $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$.

If looking for solution to $\tilde{f}(x) = y$, simply consider $f(x) = \tilde{f}(x) - y$.

*Intuition:* Each of the $n$ equations describes a surface. Looking for intersections.

*Demo:* Intersection of quadratics
Newton’s method

What does Newton’s method look like in $n$ dimensions?

Approximate by linear function:

$$f(x + s) = f(x) + J_f(x)s$$

where $J_f$ is the Jacobian matrix of $f$:

$$J_f(x) = \begin{pmatrix}
\frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\
\vdots & \ddots & \vdots \\
\frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n}
\end{pmatrix}(x).$$

Set to 0:

$$J_f(x)s = -f(x) \implies s = -(J_f(x))^{-1}f(x)$$

That’s a linear system! (Surprised?)
Newton’s method (II)

So

\[
\begin{align*}
x_0 &= \text{〈starting guess〉} \\
x_{k+1} &= x_k - (J_f(x_k))^{-1} f(x_k)
\end{align*}
\]

Downsides:

- Still only locally convergent
- Computing and inverting $J_f$ is expensive.
Set up Newton’s method to find a root of

\[ f(x, y) = \begin{pmatrix} x + 2y - 2 \\ x^2 + 4y^2 - 4 \end{pmatrix}. \]

Mostly just need the Jacobian:

\[ J_f(x, y) = \begin{pmatrix} 1 & 2 \\ 2x & 8y \end{pmatrix}. \]

**Demo:** Newton’s method in \( n \) dimensions
Secant in $n$ dimensions?

What would the secant method look like in $n$ dimensions?

Need an ‘approximate Jacobian’ satisfying

$$\tilde{J}(x_{k+1} - x_k) = f(x_{k+1}) - f(x_k).$$

Suppose we have already taken a step to $x_{k+1}$. Could we ‘reverse engineer’ $\tilde{J}$ from that equation?

- No: $n^2$ unknowns in $\tilde{J}$, but only $n$ equations
- Can only hope to ‘update’ $\tilde{J}$ with information from current guess.

One choice: Broyden’s method (minimizes change to $\tilde{J}$)
Outline

Interpolation
Iteration and Convergence
Solving One Equation
Solving Many Equations
Finding the Best: Optimization in 1D
Optimization in $n$ Dimensions
State the problem.

Have: Objective function $f : \mathbb{R}^n \rightarrow \mathbb{R}$

Want: Minimizer $x^* \in \mathbb{R}^n$ so that

$$f(x^*) = \min_x f(x) \quad \text{subject to} \quad g(x) = 0 \quad \text{and} \quad h(x) \leq 0.$$ 

- $g(x) = 0$ and $h(x) \leq 0$ are called constraints. They define the set of feasible points $x \in S \subseteq \mathbb{R}^n$.
- If $g$ or $h$ are present, this is constrained optimization. Otherwise unconstrained optimization.
- If $f$, $g$, $h$ are linear, this is called linear programming. Otherwise nonlinear programming.
- **Q:** What if we are looking for a maximizer?
  **A:** Minimize $-f$ instead.
Examples:

What is the fastest/cheapest/shortest... way to do...?

Q: What about multiple objectives?
A: Make up your mind—decide on one (or build a combined objective). Then we’ll talk.

Solve a (nonlinear!) system of equations ‘as well as you can’ (if no exact solution exists)—similar to what least squares does for linear systems:

\[
\min \| F(x) \| 
\]
What are some potential problems in optimization?

- No minimum exists: Function just ‘keeps going’.
Optimization: What could go wrong? (II)

- Find a local minimum when we meant to find a global minimum.
Optimization: What is a solution?

How can we tell that we have a (at least local) minimum? (Remember calculus!)

- Necessary condition: $f'(x) = 0$
- Sufficient condition: $f'(x) = 0$ and $f''(x) > 0$. 
Let’s steal the idea from Newton’s method for equation solving: Build a simple version of $f$ and minimize that.

Use Taylor approximation—with what degree?

**Note:** Line (i.e. degree 1 Taylor) wouldn’t suffice—lines have no minimum. Must use at least parabola. (degree 2)
Newton’s Method (II)

\[ f(x + h) \approx f(x) + f'(x)h + f''(x) \frac{h^2}{2} =: \tilde{f}(h) \]

Solve \( 0 = \tilde{f}'(h) = f'(x) + f''(x)h \):

\[ h = -\frac{f'(x)}{f''(x)} \]
Newton’s Method (III)

1. $x_0 = \langle \text{some starting guess} \rangle$

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

Q: Notice something? Identical to Newton for solving $f'(x) = 0$. Because of that: locally quadratically convergent.
Demo: Newton’s method in 1D

In-class activity: Optimization Methods
Golden Section Search

Would like a method like bisection, but for optimization. In general: No invariant that can be preserved. Need *extra assumption*.

\( f \) is called **unimodal** if for all \( x_1 < x_2 \)

\[ x_2 < x^* \Rightarrow f(x_1) > f(x_2) \]

\[ x^* < x_1 \Rightarrow f(x_1) < f(x_2) \]
Golden Section Search (II)

Suppose we have an interval with $f$ unimodal:

Would like to maintain unimodality.

1. Pick $x_1, x_2$
2. If $f(x_1) > f(x_2)$, reduce to $(x_1, b)$
3. If $f(x_1) \leq f(x_2)$, reduce to $(a, x_2)$

Remaining question: Where to put $x_1, x_2$?
Golden Section Search (III)

- Want symmetry:
  \[ x_1 = a + (1 - \tau)(b - a) \]
  \[ x_2 = a + \tau(b - a) \]

- Want to reuse function evaluations: \( \tau^2 = 1 - \tau \)
  Find: \( \tau = \left(\sqrt{5} - 1\right)/2 \). Also known as the ‘golden section’.

- Hence **golden section search**.

Linearly convergent. Can we do better?
Demo: Golden Section Search Proportions
Outline

Interpolation
Iteration and Convergence
Solving One Equation
Solving Many Equations
Finding the Best: Optimization in 1D
Optimization in $n$ Dimensions
Optimization in $n$ dimensions: What is a solution?

How can we tell that we have a (at least local) minimum? (Remember calculus!)

- **Necessary condition:** $\nabla f(x) = 0$
  $\nabla f$ is a vector, the gradient:

$$\nabla f(x) = \begin{pmatrix}
\frac{\partial f}{\partial x_1} \\
\vdots \\
\frac{\partial f}{\partial x_n}
\end{pmatrix}$$

- **Sufficient condition:** $\nabla f(x) = 0$ *and* $H_f(x)$ positive definite.

$$H_f(x) = \begin{pmatrix}
\frac{\partial^2 f}{\partial x_1 \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\
\vdots & \ddots & \vdots \\
\frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n}
\end{pmatrix}$$

is called the Hessian matrix.
Given a scalar function $f : \mathbb{R}^n \to \mathbb{R}$ at a point $x$, which way is down?

Direction of steepest descent: $-\nabla f$

Q: How far along the gradient should we go?

Unclear–do a line search. For example using Golden Section Search.

1. $x_0 = \langle$some starting guess$\rangle$
2. $s_k = -\nabla f(x_k)$
3. Use line search to choose $\alpha_k$ to minimize $f(x_k + \alpha_k s_k)$
4. $x_{k+1} = x_k + \alpha_k s_k$
5. Go to 2.

Observation: (from demo)

- Linear convergence
Demo: Steepest Descent
Newton’s method (nD)

What does Newton’s method look like in n dimensions?

Build a Taylor approximation:

\[ f(x + s) \approx f(x) + \nabla f(x)^T s + \frac{1}{2} s^T H_f(x) s =: \hat{f}(s) \]

Then solve \( \nabla \hat{f}(s) = 0 \) for \( s \) to find

\[ H_f(x) s = -\nabla f(x). \]

1. \( x_0 = \langle \text{some starting guess} \rangle \)
2. Solve \( H_f(x_k) s_k = -\nabla f(x_k) \) for \( s_k \)
3. \( x_{k+1} = x_k + s_k \)

Drawbacks: (from demo)
Newton’s method \((nD)\) (II)

- Need second (!) derivatives (addressed by Conjugate Gradients, later in the class)
- Local convergence
- Works poorly when \(H_f\) is nearly indefinite
**Demo:** Newton’s method in $n$ dimensions

**Demo:** Nelder-Mead Method
What if the $f$ to be minimized is actually a 2-norm?

$$f(x) = \|r(x)\|_2, \quad r(x) = y - f(x)$$

Define ‘helper function’

$$\varphi(x) = \frac{1}{2} r(x)^T r(x) = \frac{1}{2} f^2(x)$$

and minimize that instead.

$$\frac{\partial}{\partial x_i} \varphi = \frac{1}{2} \sum_{j=1}^{n} \frac{\partial}{\partial x_i} [r_j(x)^2] = \sum_{j} \left( \frac{\partial}{\partial x_i} r_j \right) r_j,$$

or, in matrix form:

$$\nabla \varphi = J_r(x)^T r(x).$$
Nonlinear Least Squares/Gauss-Newton (II)

For brevity: \( J := J_r(x) \). Can show similarly:

\[
H_\varphi(x) = J^T J + \sum_i r_i H_{r_i}(x).
\]

Newton step \( s \) can be found by solving

\[
H_\varphi(x)s = -\nabla \varphi
\]

Observation: \( \sum_i r_i H_{r_i}(x) \) is inconvenient to compute and unlikely to be large (since it’s multiplied by components of the residual, which is supposed to be small) \( \rightarrow \) forget about it.

Gauss-Newton method: Find step \( s \) by

\[
J^T Js = -\nabla \varphi = -J^T r(x)
\]

Does that remind you of the normal equations?

\[
Js \cong -r(x)
\]
Nonlinear Least Squares/Gauss-Newton (III)
Solve that using our existing methods for least-squares problems.

**Observations:** (from demo)
- Newton on its own is only locally convergent
- Gauss-Newton is clearly similar
- It’s worse because the step is only approximate
  → Much depends on the starting guess.

If Gauss-Newton on its own is poorly, conditioned, can try **Levenberg-Marquardt**:

\[
(J_r(x_k)^T J_r(x_k) + \mu_k I) s_k = -J_r(x_k)^T r(x_k)
\]

for a ‘carefully chosen’ \( \mu_k \). This makes the system matrix ‘more invertible’ but also less accurate/faithful to the problem. Can also be translated into a least squares problem (see book).

What Levenberg-Marquardt does is generically called ‘Regularization’: Make \( H \) more positive definite.

**Demo:** Gauss-Newton