SOLVING EQUATIONS

QUANTITATIVE ECONOMICS 2025

Piotr Żoch

October 29, 2025

INTRODUCTION

- Systems of linear equations.
- Nonlinear equations.

LINEAR EQUATIONS

LINEAR SYSTEMS OF EQUATIONS

- One of the most common problems in scientific computation: solve

$$\mathbf{A}\mathbf{x} = \mathbf{b},$$

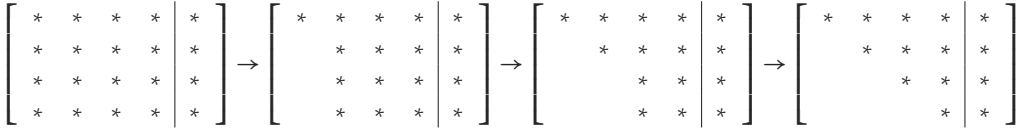
  for $\mathbf{x}$, where $\mathbf{A}$ is a square matrix and $\mathbf{b}$ is a vector.
- Seems like an easy problem, but it will teach us many things.
- Multiple specialized libraries for numerical linear algebra.

DIRECT METHODS

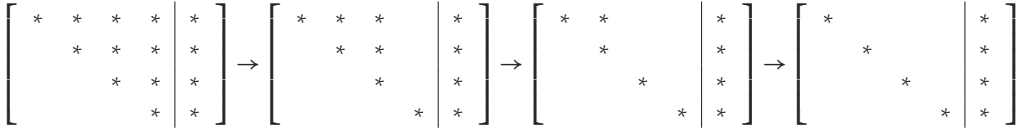
- Elementary operations:
  - multiply a row by a scalar,
  - add a scalar multiple of a row to another row,
  - interchange two rows.
- Solve $\mathbf{Ax} = \mathbf{b}$ by using elementary row operations on the augmented matrix $[\mathbf{A} \mid \mathbf{b}]$.
- Transform $\mathbf{A}$ into a reduced row echelon form.

DIRECT METHODS

- Two step procedure:
  - Forward elimination.



  - Backward substitution.



COMPLEXITY OF GAUSSIAN ELIMINATION

- Forward elimination: to deal with the first column, for each row k
  1. Calculate multiplier $m_{k1} = a_{k1}/a_{11}$ (one division)
  2. Add a multiple of the first row to the last row (n additions and n multiplications)
- We had to do this for $n - 1$ rows, so total number of operations to deal with the first column is $(n-1)(1+2n) \approx 2n^2$.
- We need to repeat it for the second column, third column and so on.
- Backward substitution: the number is smaller, because we have less rows and less columns to deal with.
- Forward elimination is $\mathcal{O}(n^3)$, backward substitution is $\mathcal{O}(n^2)$.

TRIANGULAR SYSTEMS

- Lower triangular system:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ l_{21} & 1 & 0 & 0 \\ l_{31} & l_{32} & 1 & 0 \\ l_{41} & l_{42} & l_{43} & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

  can be solved using forward elimination, starting from the top

$$y_i = b_i - \sum_{j=1}^{i-1} l_{ij} y_j.$$

UPPER TRIANGULAR SYSTEMS

- Upper triangular system:

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

  can be solved using backward substitution, starting from the bottom

$$x_i = \frac{1}{u_{ii}} \left(y_i - \sum_{j=i+1}^{n} u_{ij} x_j \right).$$

- Suppose we can write

$$\mathbf{A} = \mathbf{LU}$$

where **L** is a lower triangular matrix and **U** is an upper triangular matrix.

- We have

$$\mathbf{L}\left(\mathbf{Ux}\right) = \mathbf{b} \rightarrow \mathbf{Ly} = \mathbf{b}$$

which we can solve for **y** using forward elimination.

- We then solve

$$\mathbf{Ux} = \mathbf{y}$$

for **x** using backward substitution.

- This is known as Gaussian elimination.
    1. Compute **L** and **U**.
    2. Solve **Ly** = **b** for **y** using forward elimination.
    3. Solve **Ux** = **y** for **x** using backward substitution.
- Step 1. is known as LU decomposition.
- Nice thing: we can keep **L** and **U** and recycle them for different **b**.
- How to perform the decomposition?

MATRIX MULTIPLICATION BY OUTER PRODUCTS

- Write the columns of $\mathbf{A}$ as $\mathbf{a}_1, \ldots, \mathbf{a}_n$.
- Write the rows of $\mathbf{B}$ as $\mathbf{b}_1^\top, \ldots, \mathbf{b}_n^\top$.
- We have

$$\mathbf{AB} = \sum_{k=1}^{n} \mathbf{a}_k \mathbf{b}_k^\top.$$

- Useful: for triangular matrices $\mathbf{L}, \mathbf{U}$ only the first outer product contributes to the first row and the first column of $\mathbf{LU}$

$$\mathbf{e}_1^\top \sum_{k=1}^{n} \mathbf{l}_k \mathbf{u}_k^\top = l_{11} \mathbf{u}_1^\top, \quad \left(\sum_{k=1}^{n} \mathbf{l}_k \mathbf{u}_k^\top \right) \mathbf{e}_1 = u_{11} \mathbf{l}_1.$$

LU FACTORIZATION WITHOUT PIVOTING

Algorithm LU Factorization without Pivoting

Require: Matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$

for $j = 1$ to n **do**
  for $i = j + 1$ to n **do**
    $a_{ij} = \frac{a_{ij}}{a_{jj}}$
    for $k = j + 1$ to n **do**
      $a_{ik} = a_{ik} - a_{ij}a_{jk}$
    end for
  end for
end for

- This algorithm uses **A** matrix to store **L** and **U**.
- Problem when $a_{jj} = 0$ at any step!

SOLVING THE SYSTEM

- We need

$$\sum_{j=1}^{n} \sum_{i=j+1}^{n} \left(1 + \sum_{k=j+1}^{n} 2 \right) = \frac{2}{3}n^3 - \frac{1}{2}n^2 - \frac{1}{6}n$$

  operations to perform the factorization.

- We then need

$$\sum_{i=1}^{n} \left(2 + \sum_{j=1}^{i-1} 2 \right) = n^2 + n$$

  operations for forward and backward substitution each.

- LU decomposition is the most costly step.

SOLVING THE SYSTEM

- In practice we use a similar method, but with pivoting
- PLU factorization:

$$\tilde{\mathbf{A}} = \mathbf{LU},$$

  where $\tilde{\mathbf{A}}$ is a matrix $\mathbf{A}$ with its rows permuted.
- It works if and only if $\mathbf{A}$ is non-singular.
- Asymptotically uses the same number of operations and LU without pivoting.

NORMS

- A vector norm is a function $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$ that satisfies:
  1. $\|\mathbf{x}\| \geq 0$,
  2. $\|\mathbf{x}\| = 0 \iff \mathbf{x} = \mathbf{0}$,
  3. $\|a\mathbf{x}\| = |a| \, \|\mathbf{x}\|$,
  4. $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$,

  for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and $a \in \mathbb{R}$.
- Common vector norms: $\ell_1, \ell_2, \ell_\infty$:

$$\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i|, \quad \|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}, \quad \|\mathbf{x}\|_\infty = \max_{i=1,\ldots,n} |x_i|.$$

NORMS

- For matrices we have matrix norms.
- A Frobenius norm is

$$\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}^2}.$$

- Imagine representing a matrix as a vector with columns stacked on top of each other.
- An induced matrix norm is

$$\|\mathbf{A}\|_p = \max_{\|\mathbf{x}\|_p=1} \|\mathbf{A}\mathbf{x}\|_p .$$

- In Julia: `norm(A)` is the Frobenius norm, `opnorm(A,p)` is the induced norm.

NORMS

- We have
  1. $\|\mathbf{Ax}\| \leq \|\mathbf{A}\| \, \|\mathbf{x}\|$,
  2. $\|\mathbf{AB}\| \leq \|\mathbf{A}\| \, \|\mathbf{B}\|$,
  3. for a square matrix, $\left\|\mathbf{A}^k\right\| \leq \|\mathbf{A}\|^k$ for any integer $k \geq 0$.
- Two common matrix norm are the 1-norm and the ∞-norm:

$$\|\mathbf{A}\|_1 = \max_{j=1,\ldots,n} \sum_{i=1}^{m} |a_{ij}|, \quad \|\mathbf{A}\|_\infty = \max_{i=1,\ldots,m} \sum_{j=1}^{n} |a_{ij}|.$$

CONDITIONING OF LINEAR SYSTEMS

- Consider the perturbed system

$$\mathbf{A}\left(\mathbf{x}+\mathbf{h}\right)=\mathbf{b}+\mathbf{d}.$$

- The condition number is the relative change in the solution divided by the relative change in the data:

$$\kappa=\frac{\|\mathbf{h}\|/\|\mathbf{x}\|}{\|\mathbf{d}\|/\|\mathbf{b}\|}=\frac{\|\mathbf{h}\|\,\|\mathbf{b}\|}{\|\mathbf{d}\|\,\|\mathbf{x}\|}.$$

- Note that $\mathbf{h}=\mathbf{A}^{-1}\mathbf{d}$ so

$$\|\mathbf{h}\|\leq\left\|\mathbf{A}^{-1}\right\|\|\mathbf{d}\|.$$

- Use $\|\mathbf{h}\| \leq \left\|\mathbf{A}^{-1}\right\| \|\mathbf{d}\|$ to write

$$\frac{\|\mathbf{h}\| \|\mathbf{b}\|}{\|\mathbf{d}\| \|\mathbf{x}\|} \leq \frac{\left\|\mathbf{A}^{-1}\right\| \|\mathbf{d}\| \|\mathbf{A}\| \|\mathbf{x}\|}{\|\mathbf{d}\| \|\mathbf{x}\|} = \left\|\mathbf{A}^{-1}\right\| \|\mathbf{A}\| .$$

- We can prove that inequality is tight.
- The matrix condition number of an invertible square matrix $\mathbf{A}$ is

$$\kappa(\mathbf{A}) = \left\|\mathbf{A}^{-1}\right\| \|\mathbf{A}\| .$$

CONDITIONING OF LINEAR SYSTEMS

- If $\mathbf{A}\left(\mathbf{x}+\Delta\mathbf{x}\right)=\mathbf{b}+\Delta\mathbf{b}$ then

$$\frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}\|}\leq\kappa\left(\mathbf{A}\right)\frac{\|\Delta\mathbf{b}\|}{\|\mathbf{b}\|}.$$

- The condition number is a measure of how sensitive the solution is to changes in the data.
- We can derive a similar result for perturbed $\mathbf{A}$.
- The condition number is at least equal to 1.
- A condition number of 10^k means that we lose k digits of precision.

CONDITIONING OF LINEAR SYSTEMS

- Suppose that we compute a "solution" $\tilde{\mathbf{x}}$ to the system $\mathbf{Ax} = \mathbf{b}$.
- We would like to compare $\tilde{\mathbf{x}}$ to the true solution $\mathbf{x}$ - but we do not know $\mathbf{x}$.
- We can calculate the residual

$$\mathbf{r} = \mathbf{b} - \mathbf{A}\tilde{\mathbf{x}}$$

.

- We have

$$\frac{\|\mathbf{x} - \tilde{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \kappa(\mathbf{A}) \frac{\|\mathbf{r}\|}{\|\mathbf{b}\|}.$$

ITERATIVE METHODS

- We saw that Gaussian elimination is $\mathcal{O}(n^3)$.
- This is prohibitive for large n (unless a matrix has a special structure).
- But matrix-vector multiplication is $\mathcal{O}(n^2)$.
- In some cases we can apply repeated matrix-vector multiplication to solve the system.
- We call these iterative methods.

JACOBI METHOD

- Suppose we want to solve $\mathbf{Ax} = \mathbf{b}$.
- This can be written as

$$\sum_{j=1}^{n} a_{ij}x_j = b_i, \quad \text{or} \quad x_i = \frac{1}{a_{ii}}\left(b_i - \sum_{j\neq i} a_{ij}x_j\right).$$

- Start from some initial $\mathbf{x}^{(0)}$ and iterate:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j\neq i} a_{ij}x_j^{(k)}\right)$$

until $\mathbf{x}^{(k+1)}$ is close enough to $\mathbf{x}^{(k)}$.

GAUSS-SEIDEL METHOD

- We can also write

$$x_i = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j - \sum_{j=i+1}^{n} a_{ij} x_j \right).$$

- The Gauss-Seidel method uses the newest values of x_j:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right).$$

ITERATIVE METHODS

- Rewrite $\mathbf{A} = \mathbf{P} - \mathbf{N}$ so that the system is

$$\mathbf{P}\mathbf{x} = \mathbf{N}\mathbf{x} + \mathbf{b}.$$

- An iterative method is

$$\mathbf{P}\mathbf{x}^{(k+1)} = \left(\mathbf{N}\mathbf{x}^{(k)} + \mathbf{b}\right).$$

- $\mathbf{P}$ is called a preconditioner
- Jacobi and Gauss-Seidel differ in the choice of $\mathbf{P}$.
- Since $\mathbf{x}^{k+1} = \mathbf{P}^{-1}\left(\mathbf{N}\mathbf{x}^{(k)} + \mathbf{b}\right)$ a good preconditioner should be easy to invert.

- Why do we call $\mathbf{P}$ a preconditioner?
- Suppose we have a system $\mathbf{Ax} = \mathbf{b}$.
- The condition number is $\kappa(\mathbf{A})$.
- Suppose we have a preconditioned system $\mathbf{P}^{-1}\mathbf{Ax} = \mathbf{P}^{-1}\mathbf{b}$.
- The condition number is $\kappa\left(\mathbf{P}^{-1}\mathbf{A}\right)$.
- If $\mathbf{P} \approx \mathbf{A}^{-1}$, then $\kappa\left(\mathbf{P}^{-1}\mathbf{A}\right) \approx 1$.

ITERATIVE METHODS

- Iterative methods do not always converge.
- They are guaranteed to converge if **A** is diagonally dominant:

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|.$$

- There are better methods than Jacobi and Gauss-Seidel - the idea is to choose **P** so that the convergence is faster.

NONLINEAR EQUATIONS

NONLINEAR EQUATIONS

- We want to find a solution x^* to

$$f(x) = 0.$$

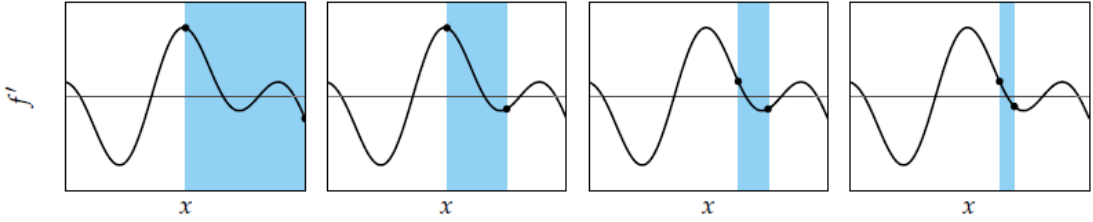
- We call x^* a root of f.
- Usually we cannot get the exact solution in a finite number of steps.
- We will focus on $f : \mathbb{R} \to \mathbb{R}$.
- Even simple polynomial equations can have no closed-form solution:

$$x^5 + 2x^2 + 1 = 0.$$

BISECTION METHOD

- Suppose we have a continuous function $f : \mathbb{R} \to \mathbb{R}$.
- We do not require it to be differentiable.
- We also have two points $a < b$ such that $f(a)f(b) < 0$ (opposite signs).
- We call $[a, b]$ a bracket.
- By the intermediate value theorem there exists a point x^* such that $f(x^*) = 0, x^* \in (a, b)$.
- Evaluate $f(m)$ where $m = \frac{a+b}{2}$.
  - If $f(m) = 0$ we are done.
  - If $f(m)f(a) < 0$ then $x^* \in (a, m)$. Use m as the new b.
  - If $f(m)f(b) < 0$ then $x^* \in (m, b)$. Use m as the new a.
- Repeat until $b - a$ is small enough.

BISECTION METHOD



Source: Kochenderfer and Wheeler (2019). Note: f' corresponds to our f.

BISECTION METHOD

- Good: bisection is *guaranteed* to converge to a root.
- Bad: it is slow. But how slow?
- If there are multiple roots, it will find just one of them – this is a general problem with root-finding algorithms.

CONVERGENCE

- A sequence $\{x_1, x_2, \ldots\}$ converges to x^* with order p, if there exists a constant r such that

$$\left|x_{k+1} - x^*\right| \leq r \left|x_k - x^*\right|^p .$$

  for all k large enough.
- We call r the rate of convergence.
- A method is locally convergent if the sequence converges to x^* for x_0 close enough to x^*.
- A method is globally convergent if the sequence converges to x^* for any x_0.

BISECTION METHOD

- The initial bracket in the bisection method is $I_0 = [a, b]$. Its length is $b - a$.
- The bracket after the first iteration is either $I_1 = \left[a, \frac{a+b}{2}\right]$ or $I_1 = \left[\frac{a+b}{2}, b\right]$. Its length is $\frac{b-a}{2}$.
- The bracket after the k-th iteration has length $\frac{b-a}{2^k}$.
- We have

$$\left|x_k - x^*\right| \leq \left(\frac{1}{2}\right)\left|x_{k-1} - x^*\right|$$

- The order of convergence is $p = 1$ and the rate of convergence is $r = \frac{1}{2}$.

BISECTION METHOD

- We add one bit of precision in each iteration.
- To get one digit of precision we need to perform ≈ 3.3 iterations (since $\log_2 10 \approx 3.3$).
- If we want to have $\left|x^k - x^*\right| \leq \gamma$, we need to perform

$$k = \log_2 \left(\frac{|b-a|}{\gamma} \right)$$

  iterations.
- The above follows from

$$\left|x^k - x^*\right| \leq 2^{-k} |b-a| .$$

BISECTION METHOD

- Bisection has $p = 1$ and $r = \frac{1}{2}$. If $p = 1$ and $r < 1$ the order of convergence is called linear.
- If $p = 1$ but $r_k < 1$ and $r_k \to 0$ as $k \to \infty$ the order of convergence is called superlinear.
- If $p = 2$ the order of convergence is called quadratic.
- If $p = 3$ the order of convergence is called cubic...
- The order of convergence does not have to be an integer.

NEWTON METHOD

- The bisection method did not use much information about the function f.
- We can use more information to design faster methods.
- Consider a Taylor expansion of $f(x^*)$ around x:

$$f(x^*) = f(x) + f'(x)(x^* - x) + \frac{1}{2}f''(\xi)(x^* - x)^2, \quad \xi \in (x^*, x)$$

- Use $f(x^*) = 0$ to solve for x^*:

$$x^* = x - \frac{f(x)}{f'(x)} + o(|x^* - x|).$$

- This suggests that x^* is close to $x - \frac{f(x)}{f'(x)}$.

NEWTON METHOD

- Start with x_0 and consider the sequence given by

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

- This is the Newton method. Also known as the Newton-Raphson method.
- Be careful: the method can diverge. Easy to see if $f'(x) = 0$.

NEWTON METHOD

- What is the order of convergence of the Newton method?
- We have $x_k = x^* + e_k$, where e_k is the error at the k-th iteration.
- We have

$$\begin{aligned} e_{k+1} &= e_k - \frac{f(x_k)}{f'(x_k)} \\ &= e_k - \frac{f(x^* + e_k)}{f'(x^* + e_k)} \\ &= e_k - \frac{f'(x^*)e_k + \frac{1}{2}f''(x^*)e_k^2 + o\left(e_k^2\right)}{f'(x^*) + f''(x^*)e_k + o(e_k)} \\ &= \frac{f''(x^*)}{2f'(x^*)}e_k^2 + o\left(e_k^2\right), \end{aligned}$$

if $f'(x^*) \neq 0$ and $f''(x^*) \neq 0$. Note that e_k is raised to the power of 2.

NEWTON METHOD

- The order of convergence is $p = 2$, so it is quadratic.
- The number of correct digits approximately doubles in each iteration.
- Good guess: 6-7 iterations to get an answer within machine precision.
- Note: if x^* is a double root, then $e_{k+1} \approx \frac{1}{2} e_k$ - linear convergence.

NEWTON METHOD

- Good: the Newton method is fast.
- Bad: you need a good guess, not guaranteed to converge.
- You also need to calculate the derivative at each x_k. How to do this?
- f has to be sufficiently smooth (we used the Taylor expansion to show quadratic convergence).

SECANT METHOD

- Instead of calculating the derivative, we can approximate it.
- Suppose we have two points x_k and x_{k-1}.
- We can approximate the derivative at x_k as

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

- The secant method is

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

- The order of convergence is $p \approx 1.6$.
- Requires two initial guesses.

SECANT METHOD

- $p \approx 1.6$ is between linear and quadratic convergence. Seems worse than Newton.
- In each iteration of Newton we use $f'(x_k)$ and $f(x_k)$.
- In each iteration of the secant method we use $f(x_k)$ and $f(x_{k-1})$, but we already calculated $f(x_{k-1})$ in the previous iteration.
- If function evaluations are used to measure computational work, the secant iteration converges more rapidly than Newton's method.

BRENT METHOD

- Bisection is slow, but guaranteed to converge.
- Newton is fast, but requires a good guess.
- The Brent method (Dekker-Brent) combines the two:
  1. Start with a bracket.
  2. Do a step of Newton / secant to get x_k. If it is within the bracket, accept it. Otherwise do bisection.
  3. Update the bracket using the new point.
  4. Repeat until convergence.
- The Brent method uses an inverse quadratic interpolation to get an approximate f'.

MULTIDIMENSIONAL METHODS

- Newton's method can be extended to multiple dimensions; $f : \mathbb{R}^n \to \mathbb{R}^n$.
- Suppose we want to solve $\mathbf{f}(\mathbf{x}) = \mathbf{0}$.
- The Newton method is

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

- $\mathbf{J}$ is the Jacobian matrix of f:

$$\mathbf{J}_{ij}(\mathbf{x}) = \frac{\partial \mathbf{f}_i(\mathbf{x})}{\partial x_j}.$$

- In practice we do not calculate the inverse of the Jacobian.
- We solve

$$\mathbf{J}(\mathbf{x}_k)\mathbf{s}_k = -\mathbf{f}(\mathbf{x}_k).$$

  and update

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

- Challenge: it is expensive to calculate the Jacobian.

BROYDEN METHOD

- Note that what we do is:

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

- We had $\mathbf{A}_k = \mathbf{J}(\mathbf{x}_k)$. But there are other ways to choose $\mathbf{A}_k$.
- One possible choice is

$$\mathbf{A}_{k+1} = \mathbf{A}_k + \frac{1}{\mathbf{s}_k^\top \mathbf{s}_k} \left[\mathbf{f}(\mathbf{x}_{k+1}) - \mathbf{f}(\mathbf{x}_k) - \mathbf{A}_k \mathbf{s}_k\right] \mathbf{s}_k^\top.$$

- This is the Broyden method.
- Under some conditions the Broyden method converges superlinearly (but $\mathbf{A}_k$ is not guaranteed to converge to the Jacobian).

LEVENBERG STEP

- Problem: it is possible to diverge.
- One way to prevent divergence is to add a Levenberg step.
- Check if an usual Newton step would lead to an improvement.
- If not, go in a direction that is guaranteed to an improvement.

$$\left(\mathbf{A}_k^\top \mathbf{A}_k + \lambda \mathbf{I}\right) \mathbf{s}_k = -\mathbf{A}_k^\top \mathbf{f}(\mathbf{x}_k).$$

- Adjust λ to ensure that the function value gets closer to zero.