

Lecture 08 - A

§8.4 Stationary Iterative Methods

T. Takahashi

Department of Mathematics and Statistics
University of New Mexico



We will answer these questions:

- What is the main drawback of direct solvers? How do iterative solvers address this drawback?
- What is Jacobi iteration?
- What is Gauss-Seidel?
- What is a fixed-point iteration and how is it related to Jacobi and GS?



Direct Solvers



Definition: Direct Solver

Direct solvers are a class of methods that find the exact solutions x^* to the linear system $Ax = b$ (up to rounding error).

- We have been learning about direct solvers
 - Gaussian Elimination
 - LU
 - Cholesky Decomposition
- Why we like them:
 - Finds exact solution
 - Predictability
 - Great for smaller systems
 - Don't need to worry about convergence
- Matlab's backslash only uses direct solvers for dense A



Complexity of Methods for Linear Systems

Direct solvers

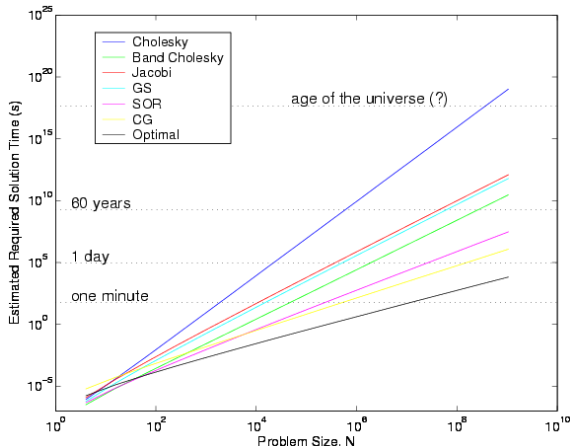
- diagonal solve: $\mathcal{O}(n)$
- tridiagonal solve: $\mathcal{O}(n)$
- triangular solve: $\mathcal{O}(n^2)$
- m -band system: $\mathcal{O}(m^2n)$

- Gaussian Elimination with: $\mathcal{O}(n^3)$
- partial pivoting adds $\mathcal{O}(n^2)$

- LU/LDL/Cholesky factorization: $\mathcal{O}(n^3)$
- one LU solve: $\mathcal{O}(n^2)$
- solve k different right-hand sides (with LU): $\mathcal{O}(n^3 + kn^2)$



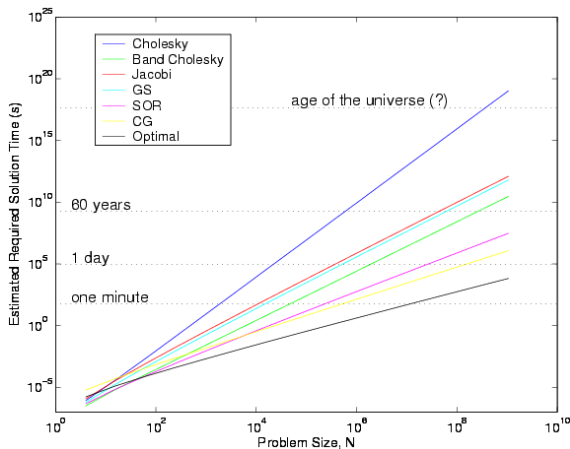
Complexity of Solvers



- log-log plot shows runtime versus size to solve an $N \times N$ system
- Direct solvers (even Cholesky) simply do not scale well for large N



Complexity of Solvers



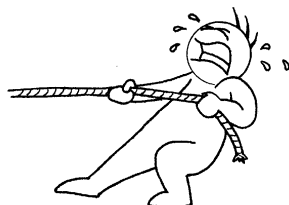
Key Idea

For applications with a million unknowns, direct solvers are prohibitively slow.

Need for faster methods



Accuracy



Cost

- Scalability is the biggest issue for direct solvers
- Willing to trade exactness for speed



Need for faster methods



Accuracy



Cost

- Scalability is the biggest issue for direct solvers
- Willing to trade exactness for speed
- Use methods called **iterative solvers**



Iteratively Improve



Question

If Gaussian elimination is taking too long and you stop halfway, do you have an approximate solution?



Question

If Gaussian elimination is taking too long and you stop halfway, do you have an approximate solution?

Nothing



Iterative Methods

Question

If Gaussian elimination is taking too long and you stop halfway, do you have an approximate solution?

Nothing

Definition: Iterative Methods

Iterative methods construct a sequence of approximate solutions

$$x^{(0)}, x^{(1)}, x^{(2)}, \dots$$

starting from an initial guess $x^{(0)}$. The integer k is the **iteration index**.

- Our goal is convergence to the true solution:

$$x^{(k)} \longrightarrow x^* \quad \text{as } k \rightarrow \infty.$$

- We can stop once the approximation is sufficiently accurate.
- For suitable problems, this can cost less than a direct solve.



The Error

- Assume A is nonsingular, and let x^* be the exact solution of $Ax = b$.
- The error at iteration k is

$$e^{(k)} := x^* - x^{(k)}.$$

- Ideally, we would stop when

$$\|e^{(k)}\| < \delta,$$

where $\delta > 0$ is a chosen error tolerance.

- **Issue:** We do not know x^* , so $e^{(k)}$ is unknown as well

Question

How can we decide when to stop using quantities we can actually compute?



The Residual

A Computable Measure

- **Idea:** use the residual instead

$$r^{(k)} := b - Ax^{(k)}.$$

- It is related to the error by

$$r^{(k)} = Ax^* - Ax^{(k)} = Ae^{(k)}.$$

- Or solving for

$$e^{(k)} = A^{-1}r^{(k)}$$

- $r^{(k)}$ is not the same as $e^{(k)}$ - **Conditioning matters**

Key Idea

We can compute the residual at every iteration and stop when

$$\|r^{(k)}\| = \|b - Ax^{(k)}\| < \delta,$$

where $\delta > 0$ is a chosen residual tolerance.

Typical Iterative Algorithm

Iterative Update

The update has the form

$$\underbrace{x^{(k+1)}}_{\text{new}} = \underbrace{x^{(k)}}_{\text{old}} + \underbrace{c^{(k)}}_{\text{correction}} .$$

```
1 r = b - A*x
2 for k = 1:kmax
3     if norm(r) <= tol
4         break
5     end
6     correction = compute_correction(A, r)
7     x = x + correction
8     r = b - A*x
9 end
```



The Correction

Part 1

Question

Suppose you start $x^{(0)}$. What is the ideal correction

$$x^{(1)} = x^{(0)} + c^{(0)}?$$



The Correction

Part 1

Question

Suppose you start $x^{(0)}$. What is the ideal correction

$$x^{(1)} = x^{(0)} + c^{(0)}?$$

- The ideal correction is $c^{(0)} = e^{(0)}$ because

$$\begin{aligned}x^{(1)} &= x^{(0)} + e^{(0)} \\&= x^{(0)} + x^* - x^{(0)} \\&= x^*\end{aligned}$$

- Therefore, the best first step is

$$\begin{aligned}x^{(1)} &= x^{(0)} + e^{(0)} \\&= x^{(0)} + A^{-1}r^{(0)}\end{aligned}$$



The Correction

Part 1

Question

Suppose you start $x^{(0)}$. What is the ideal correction

$$x^{(1)} = x^{(0)} + c^{(0)}?$$

- The ideal correction is $c^{(0)} = e^{(0)}$ because

$$\begin{aligned}x^{(1)} &= x^{(0)} + e^{(0)} \\&= x^{(0)} + x^* - x^{(0)} \\&= x^*\end{aligned}$$

- Therefore, the best first step is

$$\begin{aligned}x^{(1)} &= x^{(0)} + e^{(0)} \\&= x^{(0)} + A^{-1}r^{(0)}\end{aligned}$$

Question

Do you see a problem with this correction step?

The Correction

Part 2

- **Problem:** the update

$$x^{(k+1)} = x^{(k)} + A^{-1}r^{(k)}.$$

is nonsense because it requires A^{-1}

- **Solution:** use an approximation of the inverse $Q^{-1} \approx A^{-1}$

$$x^{(k+1)} = x^{(k)} + Q^{-1}r^{(k)}.$$

Key Idea

The strategy to iteratively improve the vector is

$$x^{(k+1)} = x^{(k)} + Q^{-1}r^{(k)}$$

where $Q \approx A$

Stationary Iterative Solvers

Definition: Stationary Iteration

Choose Q and initial guess $x^{(0)}$.

- 1 Calculate $r^{(k)} = b - Ax^{(k)}$
- 2 Solve $Qc^{(k)} = r^{(k)}$
- 3 Update $x^{(k+1)} = x^{(k)} + c^{(k)}$

This is called a **stationary iteration** because the matrix Q is fixed.

- The choice of Q determines the method.
- It affects both the cost per iteration and whether the iteration converges.

Question

Which choices of Q make the correction easy to compute?



Choices for Q



A Simple Choice for Q

- **Idea:** Keep only the diagonal:

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 3 & -1 \\ 0 & -1 & 2 \end{bmatrix} \approx \begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 2 \end{bmatrix} = Q = D.$$

- **Why?:** Inverse D^{-1} is cheap

$$D^{-1} = \begin{bmatrix} 1/2 & 0 & 0 \\ 0 & 1/3 & 0 \\ 0 & 0 & 1/2 \end{bmatrix}.$$

An Inexpensive Correction

$$Dc^{(k)} = r^{(k)} \quad \implies \quad c_i^{(k)} = \frac{r_i^{(k)}}{a_{ii}}.$$

The Jacobi Iteration

Jacobi Update

$$x^{(k+1)} = x^{(k)} + D^{-1}r^{(k)}.$$

Method: Jacobi Iteration

- **Given:** System $Ax = b$, tolerance δ
- **Output:** Approximate solution $\hat{x}$
- **Idea:** Diagonal approximation $A \approx D$ to compute inexpensive corrections

```
1  $x = x^{(0)}$ 
2 for  $k = 1, \dots, k_{\max}$ 
3    $r = b - Ax$ 
4   if  $\|r\| \leq \delta$ , stop
5    $c_i = r_i/a_{ii}$  for  $i = 1, \dots, n$ 
6    $x = x + c$ 
7 end
```

Jacobi Example

Part 1

Example

Perform two Jacobi iterations to approximate the solution of $Ax = b$:

$$A = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad x^{(0)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

- First, construct Q and Q^{-1}

$$Q = D = \text{diag}(A) = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \implies Q^{-1} = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \end{bmatrix}$$

- Iteration 1:** compute $r^{(0)}$ and $x^{(1)}$

$$r^{(0)} = b - Ax^{(0)} = b = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad x^{(1)} = x^{(0)} + Q^{-1}r^{(0)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1/2 \\ 0 \end{bmatrix} = \begin{bmatrix} 1/2 \\ 0 \end{bmatrix}.$$



Jacobi Example

Part 2

- Compute the new residual:

$$r^{(1)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} - \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} 1/2 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1/2 \end{bmatrix}.$$

- Apply the correction:

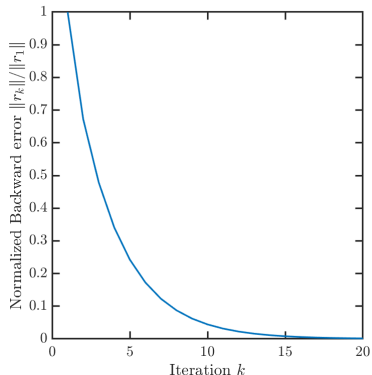
$$x^{(2)} = \begin{bmatrix} 1/2 \\ 0 \end{bmatrix} + \begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \end{bmatrix} \begin{bmatrix} 0 \\ 1/2 \end{bmatrix} = \begin{bmatrix} 1/2 \\ 1/4 \end{bmatrix}.$$

How Close Are We?

$$x^* = \begin{bmatrix} 2/3 \\ 1/3 \end{bmatrix}, \quad \|x^{(2)} - x^*\|_2 = \frac{\sqrt{5}}{12} \approx 0.186.$$



Reduction in Normalized Residual



- Track the **normalized residual**.

Residual History

$$R_k = \frac{\|r^{(k)}\|}{\|r^{(1)}\|}.$$

- Plot against iteration k .
- See `iterative_methods.m`.



Matrix Splitting

- We can split up a square matrix like so

$$\begin{bmatrix} \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \end{bmatrix} = \begin{bmatrix} \bullet & & & \\ \bullet & \bullet & & \\ \bullet & \bullet & \bullet & \\ \bullet & \bullet & \bullet & \bullet \end{bmatrix} + \begin{bmatrix} \bullet & & & \\ & \bullet & & \\ & & \bullet & \\ & & & \bullet \end{bmatrix} + \begin{bmatrix} & \bullet & \bullet & \bullet \\ & & \bullet & \bullet \\ & & & \bullet \\ & & & \bullet \end{bmatrix}$$

- Let $A = D + C_U + C_L$, where C_U and C_L are the strictly upper and lower triangular components of A , respectively.



Example of Matrix Splitting

Example

Split the following matrix

$$A = \begin{bmatrix} 4 & -1 & 0 & 0 \\ -1 & 4 & -1 & 0 \\ 0 & -1 & 4 & -1 \\ 0 & 0 & -1 & 3 \end{bmatrix}$$

We split it into three parts:

$$A = D + C_L + C_U$$

where:

$$D = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix}, \quad C_L = \begin{bmatrix} 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix}, \quad C_U = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Splitting in Matlab

```
1 >> A = ones(4,4)
```

```
2     1     1     1     1
```

```
3     1     1     1     1
```

```
4     1     1     1     1
```

```
5     1     1     1     1
```

```
6  
7 >> tril(A) % lower triangular
```

```
8     1     0     0     0
```

```
9     1     1     0     0
```

```
10    1     1     1     0
```

```
11    1     1     1     1
```

```
12  
13 >> triu(A) % upper triangular
```

```
14     1     1     1     1
```

```
15     0     1     1     1
```

```
16     0     0     1     1
```

```
17     0     0     0     1
```

```
18  
19 >> tril(A,-1) % lower triangular without the diagonal
```

```
20     0     0     0     0
```

```
21     1     0     0     0
```

```
22     1     1     0     0
```

```
23     1     1     1     0
```

Matrix Splitting Approximations

Diagonal Only

This approximation $A \approx Q = D$ yields **Jacobi iteration**

$$\begin{bmatrix} \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \end{bmatrix} \approx \begin{bmatrix} \bullet & & & \\ & \bullet & & \\ & & \bullet & \\ & & & \bullet \end{bmatrix}$$

Diagonal + Lower

This approximation $A \approx Q = D + C_L$ leads to **Gauss-Seidel iteration**

$$\begin{bmatrix} \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \end{bmatrix} \approx \begin{bmatrix} \bullet & & & \\ \bullet & \bullet & & \\ \bullet & \bullet & \bullet & \\ \bullet & \bullet & \bullet & \bullet \end{bmatrix} + \begin{bmatrix} \bullet & & & \\ & \bullet & & \\ & & \bullet & \\ & & & \bullet \end{bmatrix}$$

Gauss-Seidel (GS)

- Approximate A with $Q = \text{tril}(A)$
- In Matlab, `tril(A, -1)` returns the lower portion of matrix A
- Then the algorithm is

```
1  $x = x^{(0)}$ 
2  $Q = D + C_L$ 
3 for  $k = 1$  to  $k_{max}$ 
4      $r = b - Ax$ 
5     if  $\|r\| \leq tol$ , stop
6      $x = x + Q^{-1}r$ 
7 end
```

Method: Gauss-Seidel

- **Given:** Given system $Ax = b$, tolerance ϵ
- **Output:** Approximate solution $\hat{x}$
- **Idea:** GS has a better approximation to A^{-1} than Jacobi. Q^{-1} can be computed relatively quickly because it is triangular.

Key Difference

- Same system and initial guess:

$$A = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad x^{(0)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

- First residual: $r^{(0)} = b$.

Jacobi: Diagonal Solve

$$\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \implies c_1 = \frac{1}{2}, \quad c_2 = 0.$$

Gauss-Seidel: Forward Substitution

$$\begin{bmatrix} 2 & 0 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \implies c_1 = \frac{1}{2}, \quad c_2 = \frac{c_1}{2} = \frac{1}{4}.$$

- Update $x^{(1)} = x^{(0)} + c$:

$$x_{\text{Jacobi}}^{(1)} = \begin{bmatrix} 1/2 \\ 0 \end{bmatrix}, \quad x_{\text{GS}}^{(1)} = \begin{bmatrix} 1/2 \\ 1/4 \end{bmatrix}.$$



Actual Performance

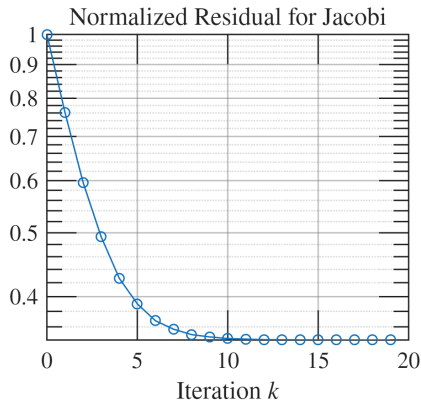


Figure: Numerical results for solving $Ax = b$ with Jacobi and GS

Actual Performance

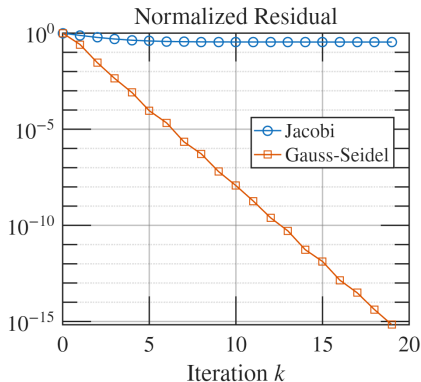
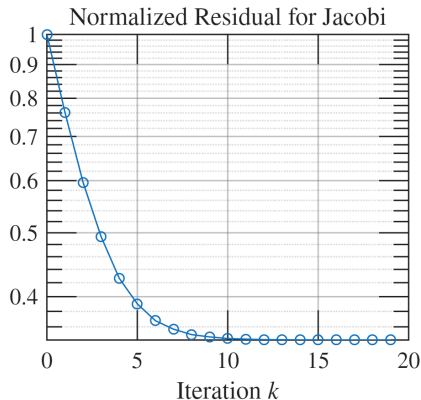


Figure: Numerical results for solving $Ax = b$ with Jacobi and GS

Fixed-Point Iteration



Old Calculator



- Old calculators had a cosine button
- It calculates the cosine of whatever number is on screen
- If keep you mashing cosine you eventually get...

0.739085133215



Fixed-Point Iteration

Problem

Suppose your goal is to solve

$$x - \cos x = 0$$

Here is one way:

- Rearrange the equation

$$x - \cos x = 0$$

$$x = \cos x$$

- Turn the equation into an iteration

$$x = \cos x \quad \longrightarrow \quad x_{n+1} = \cos(x_n)$$

- Choose an initial guess and iterate

$$x_{n+1} = \cos(x_n)$$

for $n = 1, 2, 3, \dots$



Example Calculation

- 1 Start with $x_0 = 0$
- 2 Calculate:

$$x_1 = \cos(0) = 1$$

$$x_2 = \cos(1) \approx 0.5403$$

$$x_3 = \cos(0.5403) \approx 0.8576$$

$$x_4 = \cos(0.8576) \approx 0.6543$$

$$\vdots$$

- 3 Continue until convergence (approximately $x \approx 0.739$).

Method: Fixed Point Iteration

- **Given:** Function $f(x)$, starting guess x_0 , tolerance ϵ
- **Output:** Approximate root of $f(x)$
- **Idea:** A simple method that transforms an equation into an iteration.

How it Works

Transform equation into iteration

The Equation to Solve

$$x = g(x)$$



Iteration

$$x_{n+1} = g(x_n)$$



The Method

- 1 Choose starting guess x_0
- 2 Iterate $x_{n+1} = g(x_n)$
- 3 Repeat until $|x_{n+1} - x_n| < \delta$

Why Fixed-point Iteration?

- Fixed-point iteration is one of the simplest ways to solve equations iteratively.
- Just as we rewrite $f(x) = 0$ as $x = g(x)$ to iterate scalar functions, we can rewrite the linear equation $Ax = b$ to iteratively.

Key Idea

Jacobi and GS are actually just fixed-point iterations with matrices and vectors.



Alternate Derivation of Jacobi

- Split up the A matrix like so $A = D + C_L + C_U$
- Then $Ax = b$ becomes $(D + C_L + C_U)x = b$
- We can rewrite that as

$$(D + C_L + C_U)x = b$$

$$Dx = -(C_L + C_U)x + b$$

$$x = -D^{-1}(C_L + C_U)x + D^{-1}b$$

- Treat it as a fixed-point iteration and we get **Jacobi Iteration**

$$x^{(k+1)} = -D^{-1}(C_L + C_U)x^{(k)} + D^{-1}b$$



Alternate Derivation of Jacobi

The Equation to Solve

$$Ax = b$$

↕ same

Equation with Matrix Splitting

$$x = -D^{-1}(C_L + C_U)x + D^{-1}b.$$

↓

Jacobi Iteration

$$x^{(k+1)} = -D^{-1}(C_L + C_U)x^{(k)} + D^{-1}b$$



Jacobi as a Fixed-Point Iteration

- Start with our example:

$$\begin{cases} 2x_1 - x_2 = 1, \\ -x_1 + 2x_2 = 0. \end{cases}$$

- Solve each equation for its diagonal variable:

$$x_1 = \frac{1 + x_2}{2}, \quad x_2 = \frac{x_1}{2}.$$

Turn the Equations into an Iteration

$$\boxed{x_1^{(k+1)} = \frac{1 + x_2^{(k)}}{2}}, \quad \boxed{x_2^{(k+1)} = \frac{x_1^{(k)}}{2}}.$$

Key Idea

This is Jacobi in component form: use the **old values** in both equations.

Jacobi Example (Again)

Example

Perform two Jacobi iterations starting from $x^{(0)} = (0, 0)^T$:

$$\begin{cases} 2x_1 - x_2 = 1, \\ -x_1 + 2x_2 = 0. \end{cases}$$

Fixed-Point Formulas

$$x_1^{(k+1)} = \frac{1 + x_2^{(k)}}{2}, \quad x_2^{(k+1)} = \frac{x_1^{(k)}}{2}.$$

- **Iteration 1:** Use $(x_1^{(0)}, x_2^{(0)}) = (0, 0)$.

$$x_1^{(1)} = \frac{1 + 0}{2} = \frac{1}{2}, \quad x_2^{(1)} = \frac{0}{2} = 0.$$

- **Iteration 2:** Use $(x_1^{(1)}, x_2^{(1)}) = (1/2, 0)$.

$$x_1^{(2)} = \frac{1 + 0}{2} = \frac{1}{2}, \quad x_2^{(2)} = \frac{1/2}{2} = \frac{1}{4}.$$



Moving Forward



Summary

- For large systems, factorization can be expensive in both time and memory.
- Stationary solvers improve upon an approximate solution

$$\begin{cases} r^{(k)} = b - Ax^{(k)} \\ x^{(k+1)} = x^{(k)} + Q^{-1}r^{(k)} \end{cases}$$

- Choosing $Q = D$ (diagonal) gives you the Jacobi iteration
- Choosing $Q = D + C_L$ (diagonal + lower) gives you Gauss-Seidel.



- Does the Jacobi iteration always converge?
- What is the best way to implement stationary solvers?
- What are sparse matrices?
- How do sparse matrices work with stationary solvers?

