Lecture 5: Vector and matrix norms, Classical iterative methods

Jamie Haddock

Table of contents

1 Vector and Matrix Norms

Measuring magnitude and length is fundamental to vector and matrix analysis, and will be fundamental to our analysis of numerical methods. Recall that a **norm** $\|\cdot\|$ is a real-valued function defined over a vector space with the following properties for all vectors and scalars α:

- $\|\mathbf{x}\| \geq 0$
- $\|\mathbf{x}\| = 0 \iff \mathbf{x} = \mathbf{0}$
- $\|\alpha\mathbf{x}\| = |\alpha|\|\mathbf{x}\|$
- $\|\mathbf{x}+\mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$

1.1 Vector Norms

Perhaps the most commonly encountered vector norms on $\mathbb{R}^n$ are these three:

- ℓ_2 norm: $\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^n x_i^2}$
- ℓ_∞ norm: $\|\mathbf{x}\|_\infty = \max_{i=1,\cdots,n} |x_i|$
- ℓ_1 norm: $\|\mathbf{x}\|_1 = \sum_{i=1}^n |x_i|$

```
using LinearAlgebra

x = [2, -3, 1, -1]
twonorm = norm(x)
```

```
3.872983346207417
```

```
infnorm = norm(x,Inf)
```

```
3.0
```

```
onenorm = norm(x,1)
```

7.0

We say that a sequence of vectors $\mathbf{x}_1, \mathbf{x}_2, \cdots$ **converges** to $\mathbf{x}$ if

$$\lim_{k\to\infty} \|\mathbf{x}_k - \mathbf{x}\| = 0.$$

This will be very important as we begin to study iterative methods!

Theorem: Norm equivalence

In a finite-dimensional space, convergence in any norm implies convergence in all norms.

1.2 Matrix norms

As you may recall from Linear Algebra, the space of real-valued matrices of a given size define a vector space. There are many norms for this vector space. One such norm is quite interesting since it has a nice interpretation.

This norm is the **Frobenius norm** and it is defined as

$$\|\mathbf{A}\|_F = \sqrt{\sum_{i,j} A_{ij}^2}.$$

This norm is what is by default computed by the `norm` function in Julia.

One of the most interesting aspects of this norm is that we can view it as the ℓ_2 norm on a *vectorization* of the matrix. If you imagine stacking columns of $\mathbf{A}$ to form a vector, then the ℓ_2 norm of this vector is equal to the Frobenius norm of the matrix.

Matrices are actually column-stacked when stored in memory in Julia – this is known as *column-major order*. `MATLAB` is also column-major, while `C` and `Python` are row-major.

However, note that this norm does not inherently involve the *action* of the matrix as an *operator*. There are other matrix norms which do, and these are sometimes more useful.

Definition: Induced (natural) matrix norms

Given a vector norm $\|\cdot\|$, the **induced** or **natural matrix norm** for any $m \times n$ matrix $\mathbf{A}$ is

$$\|\mathbf{A}\| = \max_{\|\mathbf{x}\|=1} \|\mathbf{A}\mathbf{x}\| = \max_{\mathbf{x}\neq 0} \frac{\|\mathbf{A}\mathbf{x}\|}{\|\mathbf{x}\|}.$$

The induced norm definition causes these norms to satisfy some useful inequalities:

Theorem: Norm inequalities

Let $\|\cdot\|$ designate a matrix norm and the vector norm that induced it. Then for all matrices and vectors of compatible sizes,

$$\|\mathbf{A}\mathbf{x}\| \leq \|\mathbf{A}\|\|\mathbf{x}\|.$$

For all matrices of compatible sizes,

$$\|\mathbf{A}\mathbf{B}\| \leq \|\mathbf{A}\|\|\mathbf{B}\|.$$

Exercise: Prove these inequalities!

Answer:

For the first, if $\mathbf{x} = \mathbf{0}$, then $\|\mathbf{A}\mathbf{x}\| = 0 = \|\mathbf{A}\|\|\mathbf{x}\|$. Now, we can deal only with the case $\mathbf{x} \neq 0$,

$$\frac{\|\mathbf{A}\mathbf{x}\|}{\|\mathbf{x}\|} \leq \max_{\mathbf{x}\neq\mathbf{0}} \frac{\|\mathbf{A}\mathbf{x}\|}{\|\mathbf{x}\|} = \|\mathbf{A}\|.$$

Answer:

For the second,

$$\|\mathbf{A}\mathbf{B}\| = \max_{\mathbf{x}\neq\mathbf{0}} \frac{\|\mathbf{A}\mathbf{B}\mathbf{x}\|}{\|\mathbf{x}\|} \leq \max_{\mathbf{x}\neq\mathbf{0}} \frac{\|\mathbf{A}\|\|\mathbf{B}\mathbf{x}\|}{\|\mathbf{x}\|} \leq \max_{\mathbf{x}\neq\mathbf{0}} \frac{\|\mathbf{A}\|\|\mathbf{B}\|\|\mathbf{x}\|}{\|\mathbf{x}\|} = \|\mathbf{A}\|\|\mathbf{B}\|.$$

The induced matrix ∞- and 1-norms can be equivalently defined in terms of the entries of the matrix.

Theorem: Matrix ∞- and 1-norms

$$\|\mathbf{A}\|_\infty = \max_{1\leq i\leq n} \sum_{j=1}^{n} |A_{ij}|$$

$$\|\mathbf{A}\|_1 = \max_{1\leq j\leq n} \sum_{i=1}^{n} |A_{ij}|$$

```
A = [2 0; 1 -1]
```

```
2×2 Matrix{Int64}:
 2   0
 1  -1
```

```
Fronorm = norm(A)
```

```
2.449489742783178
```

```
twonorm = opnorm(A)
```

```
2.2882456112707374
```

We can also see that the entry-wise definitions of the ∞- and 1-norms are equivalent to their *induced norm* definition.

```
onenorm = opnorm(A,1)
```

```
3.0
```

```
maximum( sum(abs.(A),dims=1) )
```

```
3
```

```
infnorm = opnorm(A,Inf)
```

```
2.0
```

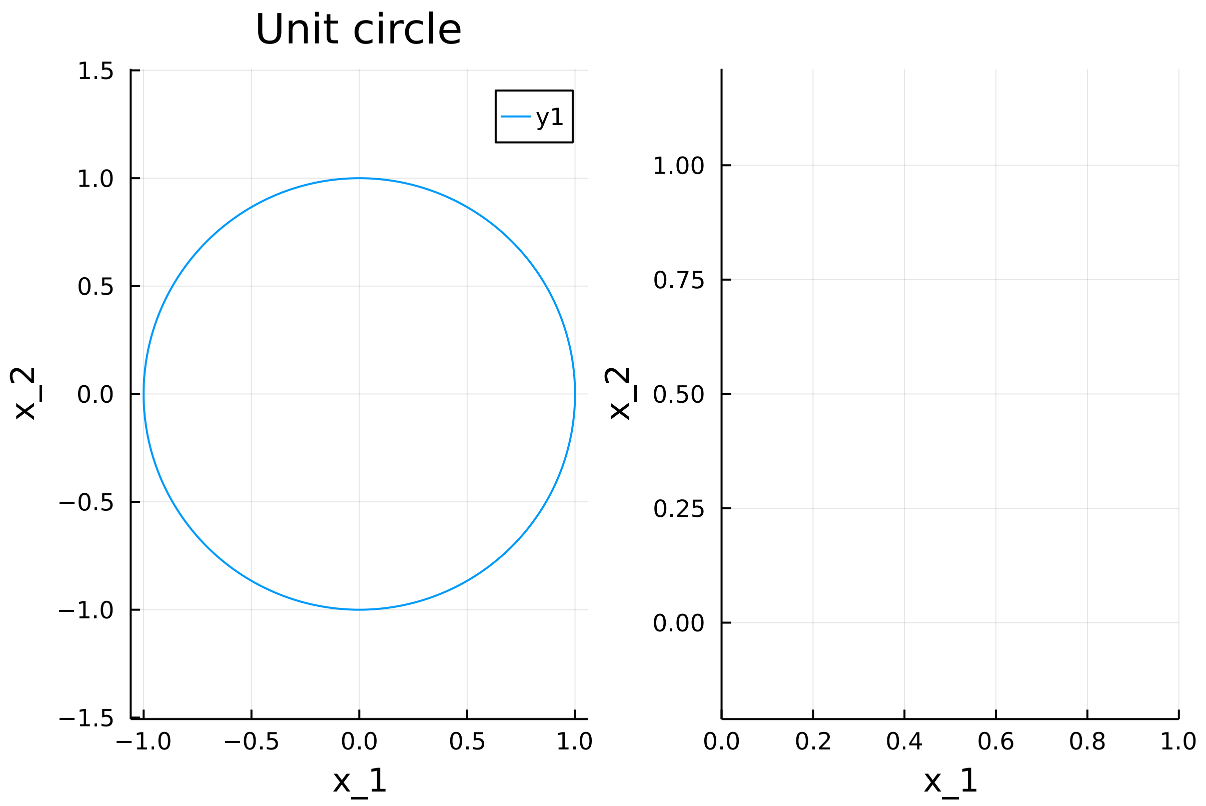
```
maximum( sum(abs.(A),dims=2) )
```

```
2
```

Now, we'll try to construct a geometric interpretation of the ℓ_2 norm.

```
# sample a lot of vectors on the unit circle in R^2
theta = 2pi*(0:1/600:1)
x = [ fun(t) for fun in [cos,sin], t in theta ]; #what a cool comprehension!
```

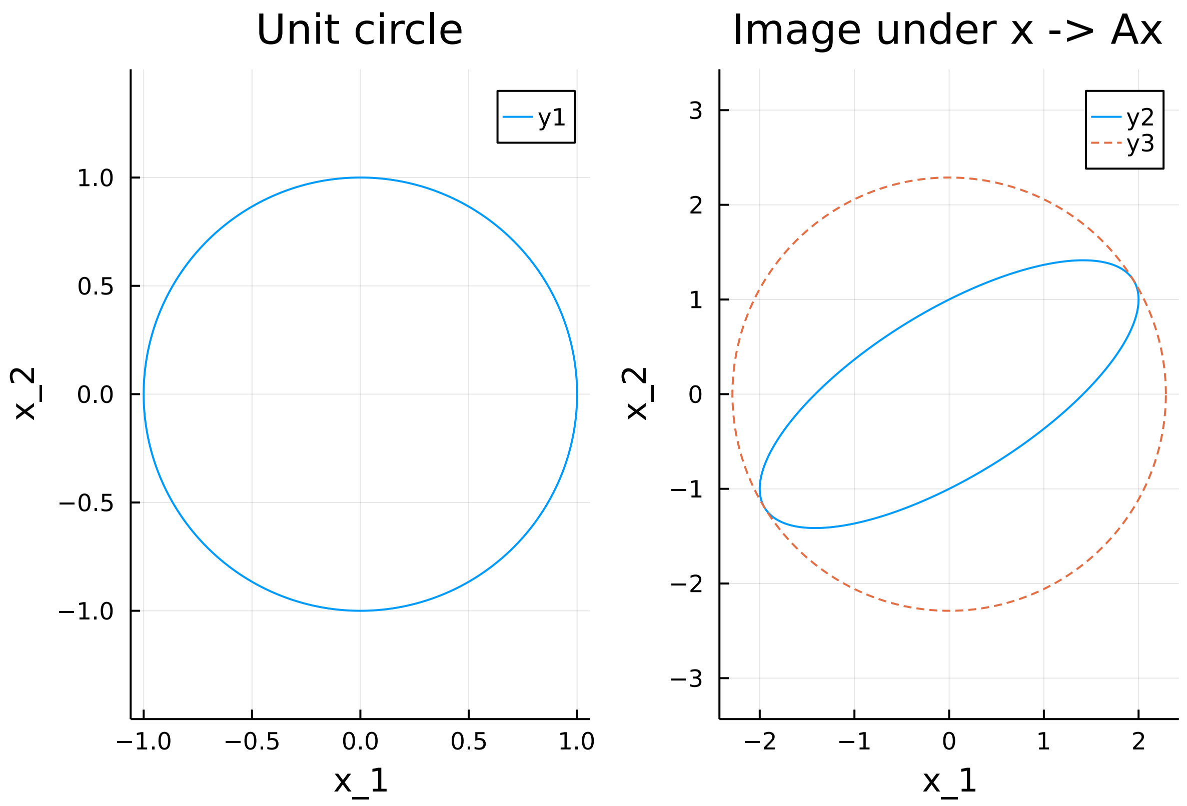
```
using Plots

plot(aspect_ratio=1, layout=(1,2), xlabel="x_1", ylabel="x_2") #creates a "layout" -- subsequent plot!
plot!(x[1,:],x[2,:], subplot=1,title="Unit circle")
```



Now, the function $\mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{x}$ defines a mapping from $\mathbb{R}^2$ to $\mathbb{R}^2$. Let's see what this does to the vectors in the unit circle!

```
Ax = A*x;
```

```
plot!(Ax[1,:],Ax[2,:],subplot=2,title="Image under x -> Ax")
plot!(twonorm*x[1,:],twonorm*x[2,:], subplot=2,l=:dash)
```



2 Classical Iterative Methods for Solving Linear Systems

We've talked a bit last week about *direct* methods for solving linear systems of equations. There is another class of methods known as *iterative methods* which use an iterative sequence of steps to make incremental improvement of an approximate solution to the system.

These methods typically use information from some subset of the system to make iterative improvements to the approximate solution (**iterate**).

2.1 Jacobi's Method

The first method hinges on the observation that if $\mathbf{Ax} = \mathbf{b}$ then

$$A_{i1}x_1 + A_{i2}x_2 + \cdots + A_{ii}x_i + \cdots + A_{in}x_n = b_i \text{ for } i = 1, \cdots, n.$$

This can be arranged as

$$x_i = \frac{1}{A_{ii}} \left[\sum_{j=1 j \neq i}^{n} (-A_{ij}x_j) + b_i \right] \text{ for } i = 1, \cdots, n.$$

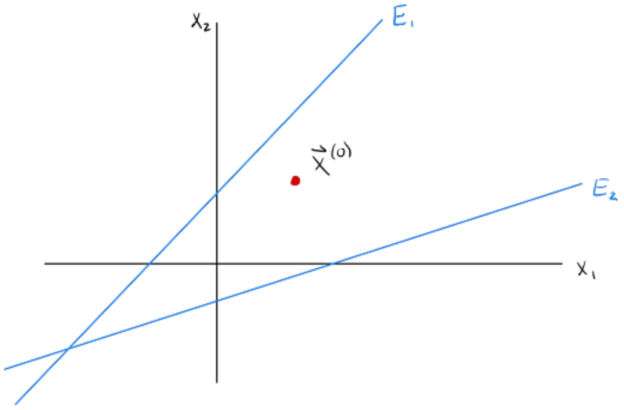
Jacobi's method uses this observation iteratively, forcing the ith component of the current approximation to satisfy the ith equation, that is $\mathbf{x}^{(k)}$ satisfies

$$x_i^{(k)} = \frac{1}{A_{ii}} \left[\sum_{j=1 j \neq i}^{n} (-A_{ij}x_j^{(k-1)}) + b_i \right] \text{ for } i = 1, \cdots, n.$$

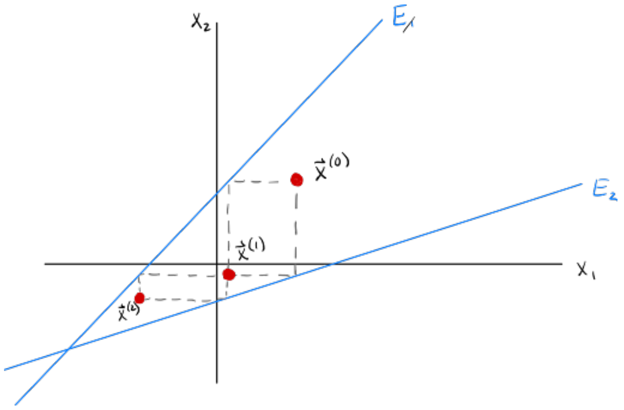
Exercise: Sketch Jacobi's method

Sketch two iterations of Jacobi's method applied to the system visualized below with the given initial iterate, $\mathbf{x}^{(0)}$.

System:



Answer:



You should also think about what happens here if the order of the equations is changed or if the initial iterate is somewhere else. Spoiler alert: Jacobi's method doesn't always converge!

2.2 Matrix-vector form of Jacobi update

We can collect all of the updates to the coordinates of $\mathbf{x}^{(k)}$ together into a vector update.

Note that we can rewrite $\mathbf{A}\mathbf{x} = \mathbf{b}$ as

$$\mathbf{D}(\mathbf{A})\mathbf{x} + \mathbf{L}(\mathbf{A})\mathbf{x} + \mathbf{U}(\mathbf{A})\mathbf{x} = \mathbf{b}$$

where $\mathbf{D}(\mathbf{A})$ is the diagonal matrix containing the diagonal elements of $\mathbf{A}$, $\mathbf{L}(\mathbf{A})$ is the strictly lower-triangular matrix containing the strictly lower-triangular elements of $\mathbf{A}$, and $\mathbf{U}(\mathbf{A})$ is the strictly upper-triangular matrix containing the strictly upper-triangular elements of $\mathbf{A}$.

In this notation, the Jacobi iterates satisfy

$$\mathbf{D}(\mathbf{A})\mathbf{x}^{(k)} + [\mathbf{L}(\mathbf{A}) + \mathbf{U}(\mathbf{A})]\mathbf{x}^{(k-1)} = \mathbf{b}$$

and so,

$$\mathbf{x}^{(k)} = -\mathbf{D}(\mathbf{A})^{-1}[\mathbf{L}(\mathbf{A}) + \mathbf{U}(\mathbf{A})]\mathbf{x}^{(k-1)} + \mathbf{D}(\mathbf{A})^{-1}\mathbf{b}.$$

```
"""
    jacobi(A,b,x_0,N)

Run N iterations of Jacobi's method on system Ax=b with initial iterate x_0.
"""
function jacobi(A,b,x_0,N,x_true)
    D = Diagonal(A);
    L = tril(A,-1);
    U = triu(A,1);

    x_old = x_0
    errs = zeros(N+1)
    errs[1] = norm(x_old - x_true)/norm(x_true)
    for i in 1:N
        x_new = D\(-(L+U)*x_old + b);
        errs[i+1] = norm(x_new - x_true)/norm(x_true);
        x_old = x_new;
    end
    return x, errs
end
```

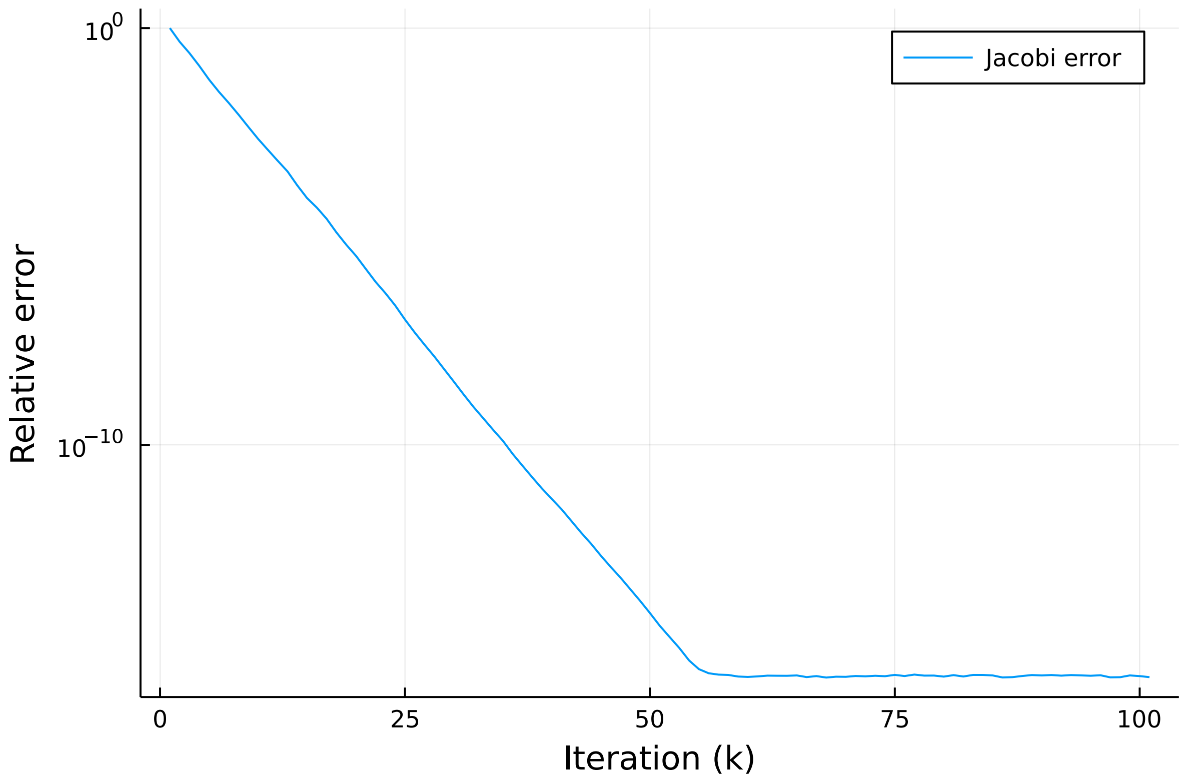
```
jacobi
```

Now, we'll run a small experiment to check that our code works!

```
n = 100
#A = randn(n,n)
A = diagm(ones(n)) + 0.05*randn(n,n)
x_true = ones(n)
b = A*x_true;
```

```
N = 100
x, errs = jacobi(A,b,zeros(n),N,x_true);
```

```
plot(1:N+1,errs,yscale=:log10,label="Jacobi error",xaxis="Iteration (k)", yaxis="Relative error")
```



2.3 Gauss-Seidel method

We can possibly improve Jacobi's method by taking a second look at the equation

$$x_i^{(k)} = \frac{1}{A_{ii}} \left[\sum_{j=1 j\neq i}^{n} (-A_{ij} x_j^{(k-1)}) + b_i \right] \text{ for } i = 1, \cdots, n.$$

If we think about doing the updates to the entries of $\mathbf{x}^{(k)}$ sequentially, we see that we update $x_1^{(k)}$ before updating $x_2^{(k)}$, but in the update of $x_2^{(k)}$, we use $x_1^{(k-1)}$, despite having the (likely better) estimate for the first entry computed in $x_1^{(k)}$.

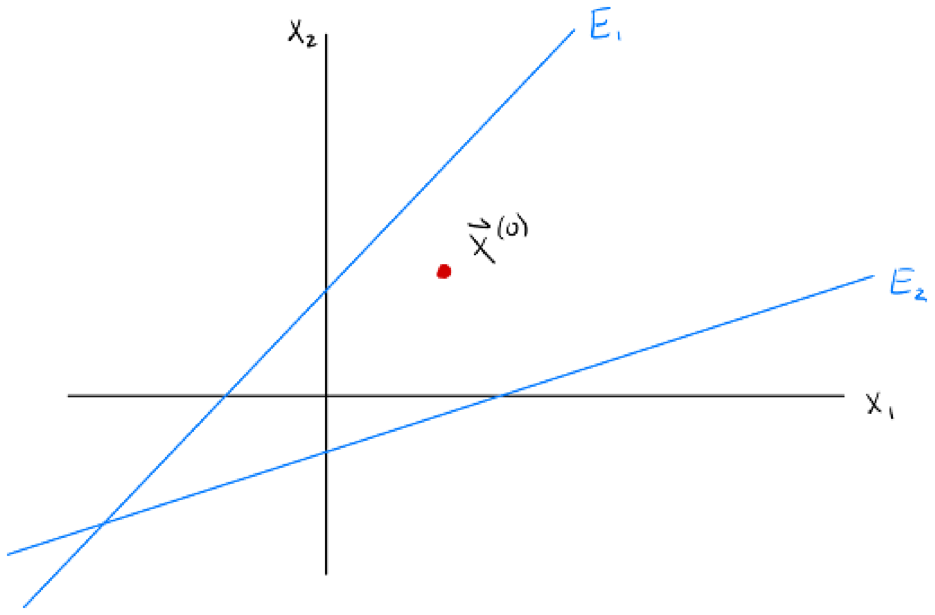
This motivates using the most up-to-date estimates available in this update:

$$x_i^{(k)} = \frac{1}{A_{ii}} \left[\sum_{j=1}^{i-1} (-A_{ij} x_j^{(k)}) \sum_{j=i+1}^{n} (-A_{ij} x_j^{(k-1)}) + b_i \right] \text{ for } i = 1, \cdots, n.$$

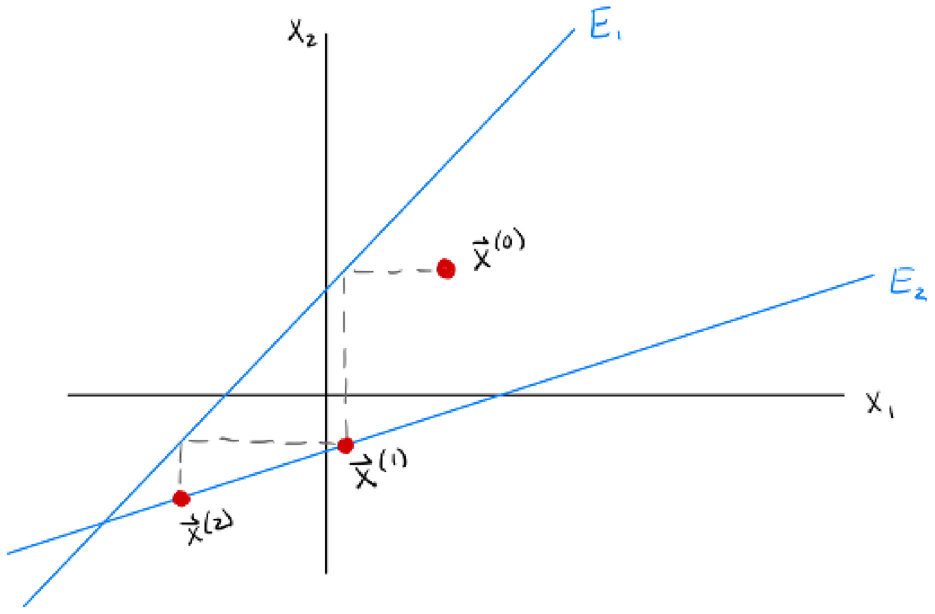
Exercise: Sketch the Gauss-Seidel method

Sketch two iterations of the Gauss-Seidel method applied to the system visualized below with the given initial iterate, $\mathbf{x}^{(0)}$.

System:



Answer:



2.4 Matrix-vector form of G-S update

In the same way we did for Jacobi, we can succinctly write the Gauss-Seidel update in matrix-vector form. Noting, now, that the update

$$x_i^{(k)} = \frac{1}{A_{ii}} \left[\sum_{j=1}^{i-1} (-A_{ij} x_j^{(k)}) \sum_{j=i+1}^{n} (-A_{ij} x_j^{(k-1)}) + b_i \right] \text{ for } i = 1, \cdots, n$$

involves the entire lower triangular part of $\mathbf{A}$ interacting with entries of $\mathbf{x}^{(k)}$,

we can rewrite this as

$$[\mathbf{D}(\mathbf{A}) + \mathbf{L}(\mathbf{A})]\mathbf{x}^{(k)} + \mathbf{U}(\mathbf{A})\mathbf{x}^{(k-1)} = \mathbf{b}.$$

Thus, the update is

$$\mathbf{x}^{(k)} = -[\mathbf{D}(\mathbf{A}) + \mathbf{L}(\mathbf{A})]^{-1}\mathbf{U}(\mathbf{A})\mathbf{x}^{(k-1)} + [\mathbf{D}(\mathbf{A}) + \mathbf{L}(\mathbf{A})]^{-1}\mathbf{b}.$$

Exercise: Implement the Gauss-Seidel method

Complete the Gauss-Seidel method code below and run this method on the same experimental system above.

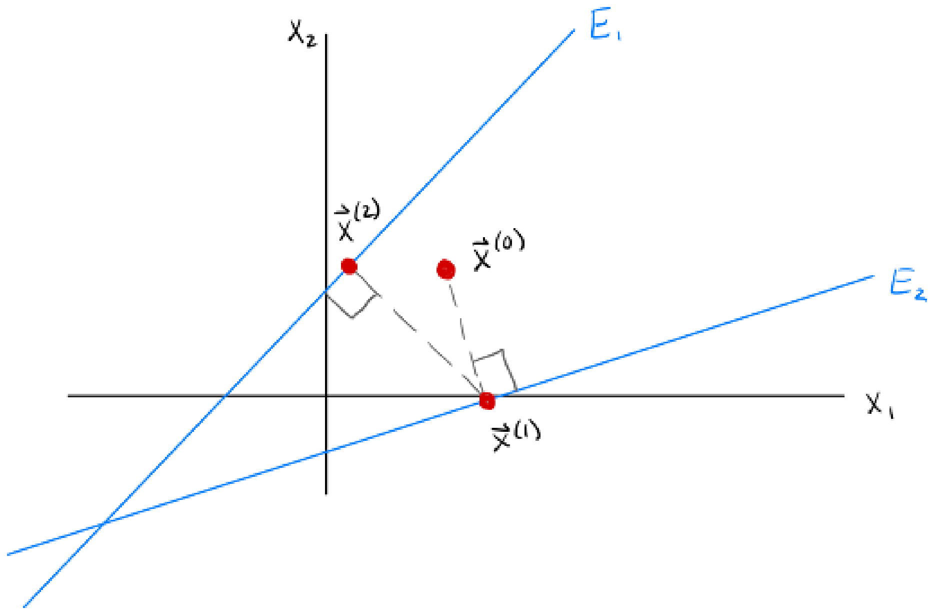
```
"""
    gaussseidel(A,b,x_0,N)

Run N iterations of the Gauss-Seidel method on system Ax=b with initial iterate x_0.
"""
function gaussseidel(A,b,x_0,N,x_true)
    D = Diagonal(A);
    L = tril(A,-1);
    U = triu(A,1);

    x_old = x_0
    errs = zeros(N+1)
    errs[1] = norm(x_old - x_true)/norm(x_true)
    for i in 1:N
        x_new = ############################################### complete this code!
        errs[i+1] = norm(x_new - x_true)/norm(x_true);
        x_old = x_new;
    end
    return x, errs
end
```

```
gaussseidel
```

2.5 Kaczmarz method

The Jacobi and Gauss-Seidel methods update using individual coordinate direction updates onto the hyperplanes defined by the individual equations. This is not the only way that the iterates can move towards these hyperplanes!

Perhaps the most natural updating strategy is to project each iterate onto a hyperplane to produce the subsequent iterate. This strategy is known as the *Kaczmarz method*.



The update for this type of method is given by $\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + \frac{b_i - \mathbf{a}_i^\top \mathbf{x}^{(k-1)}}{\|\mathbf{a}_i\|^2}\mathbf{a}_i$ where $\mathbf{a}_i$ is the ith row of $\mathbf{A}$ and i is chosen from $1, \cdots, n$.

2.6

Exercise: Decreasing error

If $\mathbf{x}^*$ satisfies $\mathbf{A}\mathbf{x}^* = \mathbf{b}$, then

$$\|\mathbf{x}^{(k)} - \mathbf{x}^*\|_2 \leq \|\mathbf{x}^{(k-1)} - \mathbf{x}^*\|_2.$$

Show that this is true!

Answer:

Note that $\mathbf{x}^*$ lies in the intersection of all hyperplane solution sets, and the three vector $\mathbf{x}^{(k)} - \mathbf{x}^*$, $\mathbf{x}^{(k-1)} - \mathbf{x}^*$, and $\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}$ form the edges of a right triangle. By the Pythagorean theorem, we have

$$\|\mathbf{x}^{(k-1)} - \mathbf{x}^*\|^2 = \|\mathbf{x}^{(k)} - \mathbf{x}^*\|^2 + \|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\|^2.$$

The result follows since $\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\|^2 \geq 0$.