

Forward Euler Method

Forward Euler is an
“explicit method”

$$\begin{aligned}\mathbf{Q}^{n+1} &= \mathbf{Q}^n + \Delta t A \mathbf{Q}^n \\ &= (I + \Delta t A) \mathbf{Q}^n\end{aligned}$$

In order to ensure that $\| \mathbf{Q}^{n+1} \|$ doesn't grow, we needed

$$\| \mathbf{Q}^{n+1} \| = \| (I + \Delta t A) \mathbf{Q}^n \| < \| I + \Delta t A \| \| \mathbf{Q}^n \|$$

or $\| I + \Delta t A \| < 1$. We chose our time step so that

$$-1 < 1 + \Delta t \lambda^p(A) < 1$$

or that

$$(\Delta t)_{stable} < \frac{(\Delta x)^2}{2}$$

time step restriction

Forward Euler

Consider a typical problem on a 1d mesh $[0,1]$ with N mesh points. Below is the number of time steps M we would need to take to reach $T=1.0$

N	dx	dt	M
16	6.25E-02	1.95E-03	512
32	3.13E-02	4.88E-04	2048
64	1.56E-02	1.22E-04	8192
128	7.81E-03	3.05E-05	32768
256	3.91E-03	3.05E-05	131072
512	1.95E-03	1.91E-06	524288
1024	9.77E-04	4.77E-07	2097152

Backward Euler Method

Backward Euler is an
“implicit method”

$$\begin{aligned}\mathbf{Q}^{n+1} &= \mathbf{Q}^n + \Delta t A \mathbf{Q}^{n+1} \\ &= (I - \Delta t A)^{-1} \mathbf{Q}^n\end{aligned}$$

Just as with FE, we need to ensure that $\| \mathbf{Q}^{n+1} \|$ doesn't grow.

$$\| \mathbf{Q}^{n+1} \| = \| (I - \Delta t A)^{-1} \mathbf{Q}^n \| < \frac{\| \mathbf{Q}^n \|}{\| I - \Delta t A \|}$$

We have

$$0 < \frac{1}{1 - \Delta t \lambda^p(A)} < 1$$

no time step restriction

for all $\Delta t > 0$ and the Backward Euler method is *unconditionally stable*. Typically, however, we try to take $\Delta t \approx \Delta x$

Backward Euler

Comparison between using Forward Euler time step and a Backward Euler time step $\Delta t \approx \Delta x$

N	dx	dt (FE)	M (FE)	M (BE)	Ratio
16	6.25E-02	1.95E-03	512	16	32
32	3.13E-02	4.88E-04	2048	32	64
64	1.56E-02	1.22E-04	8192	64	128
128	7.81E-03	3.05E-05	32768	128	256
256	3.91E-03	3.05E-05	131072	256	512
512	1.95E-03	1.91E-06	524288	512	1024
1024	9.77E-04	4.77E-07	2097152	1024	2048

$\sim \mathcal{O}(N)$

If we can solve the linear system in $\mathcal{O}(N)$ work, we will come out ahead. More importantly, we will have a much more stable scheme.

The cost of an implicit method

Implicit methods have better stability properties than explicit methods, but each time step is more costly, and can be more difficult to implement (especially in parallel).

Explicit (Forward Euler)

```
Initialize  $Q(0)$   
for  $k = 0, 1, 2, \dots$ 
```

```
 $Q(n+1) = Q(n) + dt * A * Q(n);$ 
```

“embarrassingly parallel” and
requires only local communication

Implicit (Backward Euler)

```
Initialize  $Q(0)$   
for  $k = 0, 1, 2, \dots$ 
```

```
Solve  $(I - dt * A) Q_{n+1} = Q_n;$ 
```

```
 $Q(n+1) = Q_{n+1};$ 
```

non-trivial to parallelize and
requires global communication.

Solving linear systems

At each time step of Backward Euler, we have to solve the linear system

$$(I - \Delta t A)\mathbf{x} = \mathbf{Q}^n$$

for the unknowns \mathbf{x} .

In what follows, we assume that we are solving the “canonical” linear system given by

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

For the heat equation using backward Euler, we have

$$A \leftarrow I - \Delta t A$$

$$\mathbf{b} \leftarrow \mathbf{Q}^n$$

$$\mathbf{x} \leftarrow \mathbf{Q}^{n+1}$$

Direct methods?

In serial, direct methods can often be preferable

- Tridiagonal solver (for 1d) - Thomas Algorithm
- Banded solvers (2d, 3d) - Lapack
- Cyclic reduction (2d, 3d) (FishPACK)
- Discrete Fourier transforms
- Integral methods with acceleration
- Multifrontal methods (UMFPack)

These methods have the advantage that they converge in a fixed number of steps, regardless of the problem. Industry likes direct methods for their “robustness”.

However, these methods can be considerably more difficult to implement in parallel.

Stationery Iterative Methods

Iterative methods work on the principle that we can solve a linear system $A\mathbf{x} = \mathbf{b}$ using an iterative method.

A stationary iterative method is based on two key principles :

- Applying A to a vector \mathbf{x} is easy and cheap, and
- We have a matrix M that approximates A and is easy to invert.

Common stationary methods include Jacobi, Gauss-Seidel and the SOR method.

Stationery iterative methods

Assume that we have an iterate \mathbf{x}_k that approximates our true solution \mathbf{x} . Then

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

where $\mathbf{r}_k \equiv \mathbf{b} - A\mathbf{x}_k$ is the *residual*.

Subtracting this from the original equation, we get

$$A(\mathbf{x} - \mathbf{x}_k) = \mathbf{r}_k$$

If we define an *error* as $\mathbf{e}_k \equiv \mathbf{x} - \mathbf{x}_k$, we have

$$A\mathbf{e}_k \equiv \mathbf{r}_k$$

If we solved this for the error, we would have $\mathbf{x} = \mathbf{x}_k + \mathbf{e}_k$, and we would be done. Instead, we solve an easier problem

$$M\mathbf{z}_k = \mathbf{r}_k$$

and *update* \mathbf{x}_k as $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{z}_k$.

Fixed point iteration

We can also interpret stationery methods as a fixed point iteration that solves

$$g(\mathbf{x}) = \mathbf{x} + M^{-1}(\mathbf{b} - A\mathbf{x}) = \mathbf{x}$$

The fixed point iteration is then given by

$$\mathbf{x}_{k+1} = \mathbf{x}_k + M^{-1}(\mathbf{b} - A\mathbf{x}_k)$$

The **Jacobi Method** chooses $M = D$, the diagonal of A.

The **Gauss-Seidel** method choose $M = L + D$, where L is the lower triangular portion of A.

The Jacobi method converges if the matrix is diagonally dominant, and the Gauss-Seidel method converges if the matrix is symmetric positive definite.

Stationary Method Algorithm

We choose a matrix M that approximates A . Given a tolerance $\tau \ll 1$, and an initial guess \mathbf{x}_0

Compute $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$

Solve $M\mathbf{z}_0 = \mathbf{r}_0$

For $k = 0, 1, 2, \dots$

Set $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{z}_k$

Compute $\mathbf{r}_{k+1} = \mathbf{b} - A\mathbf{x}_{k+1}$

Solve $M\mathbf{z}_{k+1} = \mathbf{r}_{k+1}$ Solve for an approximation to the error

if $\|\mathbf{z}_{k+1}\| < \tau$

Stop

end

end

Solution is $\mathbf{x} \equiv \mathbf{x}_{k+1}$.

Steepest-descent method

Steepest-descent finds the minimum of the function

$$F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T \mathbf{x}$$

A must be
symmetric,
positive
definite

The direction of greatest *decrease* of this function is

$$-\nabla F(\mathbf{x}) = \mathbf{b} - A\mathbf{x} = \mathbf{r}$$

Using the residual as a search direction, we get iterates

$$\mathbf{x}_{k+1} = \mathbf{x}_k + a_k \mathbf{r}_k$$

and

$$\begin{aligned}\mathbf{r}_{k+1} &= \mathbf{b} - A\mathbf{x}_{k+1} \\ &= \mathbf{b} - A(\mathbf{x}_k + \alpha_k \mathbf{r}_k) \\ &= \mathbf{r}_k - \alpha_k A\mathbf{r}_k\end{aligned}$$

The question is, how do we choose a_k ?

Steepest-descent methods

Calculus shows that we should choose α_k so that we minimize the function $f(\alpha) = \mathbf{F}(\mathbf{x}_{k+1})$ along a direction \mathbf{r}_k . This minimum occurs at the point where $\nabla \mathbf{F}(\mathbf{x}_k + \alpha \mathbf{r}_k)$ is orthogonal to \mathbf{r}_k :

$$f'(\alpha) = \nabla \mathbf{F}(\mathbf{x}_k + \alpha \mathbf{r}_k)^T \mathbf{r}_k = 0$$

Using the update for the residual, given by

$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k A \mathbf{r}_k$$

we have $\nabla \mathbf{F}(x_{k+1}) = -\mathbf{r}_{k+1}$, so we can set $\mathbf{r}_{k+1}^T \mathbf{r}_k = 0$.

Solving for α_k , we get

$$(\mathbf{r}_k - \alpha_k A \mathbf{r}_k)^T \mathbf{r}_k = \mathbf{r}_k^T \mathbf{r}_k - \alpha_k A \mathbf{r}_k^T \mathbf{r}_k = 0$$

$$\alpha_k = \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{r}_k^T A \mathbf{r}_k}$$

Steepest Descent Algorithm

Given a positive definite matrix A , an initial guess \mathbf{x}_0 , and a tolerance $\tau \ll 1$

Compute $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$

Set $\mathbf{p}_0 = \mathbf{r}_0$ **Search directions**

For $k = 0, 1, 2, \dots$

 Compute $A\mathbf{p}_k$

 Set $\alpha_k = (\mathbf{r}_k^T \mathbf{r}_k) / (\mathbf{p}_k^T A\mathbf{p}_k)$

 Set $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$

 Set $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k A\mathbf{p}_k$

 Set $\mathbf{p}_{k+1} = \mathbf{r}_{k+1}$

Search direction is the residual

 if $\|\mathbf{r}_{k+1}\| < \tau$

 Stop

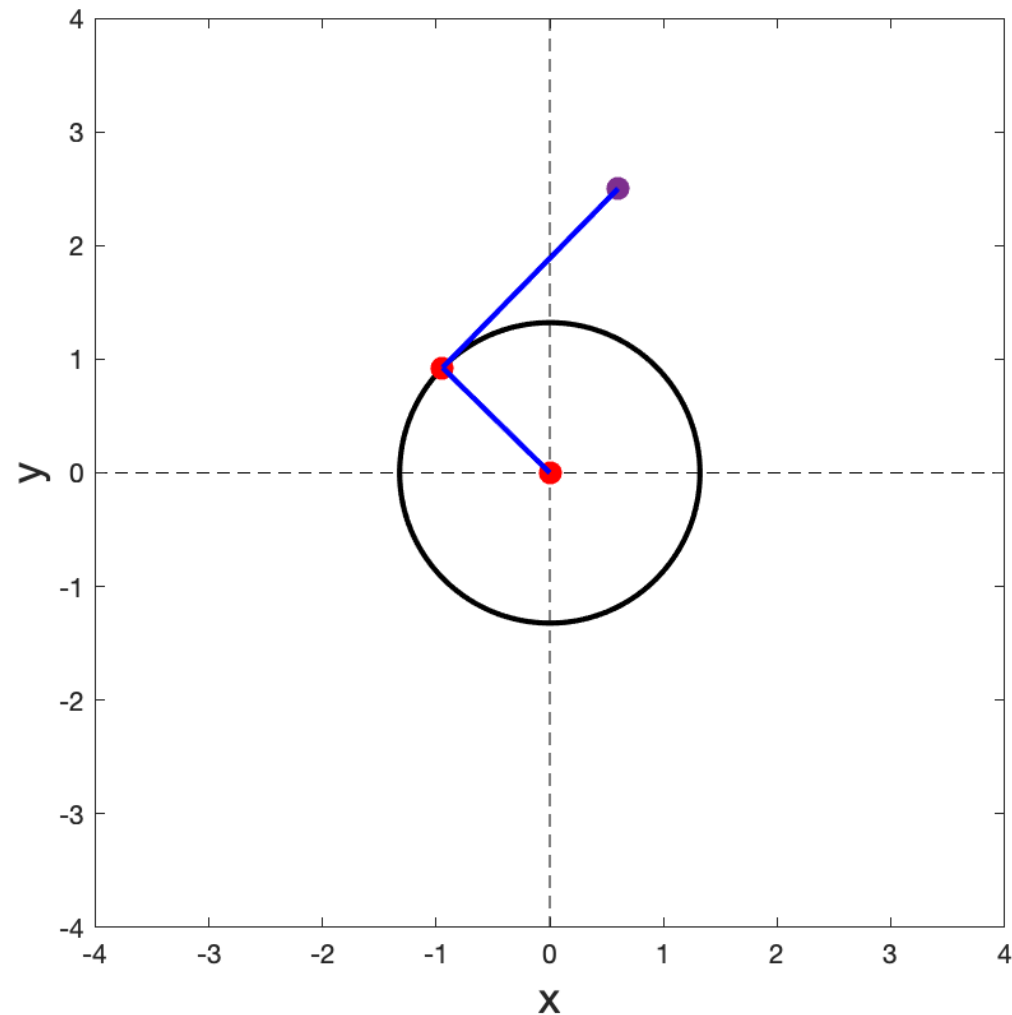
 end

end

Solution is $\mathbf{x} \equiv \mathbf{x}_{k+1}$.

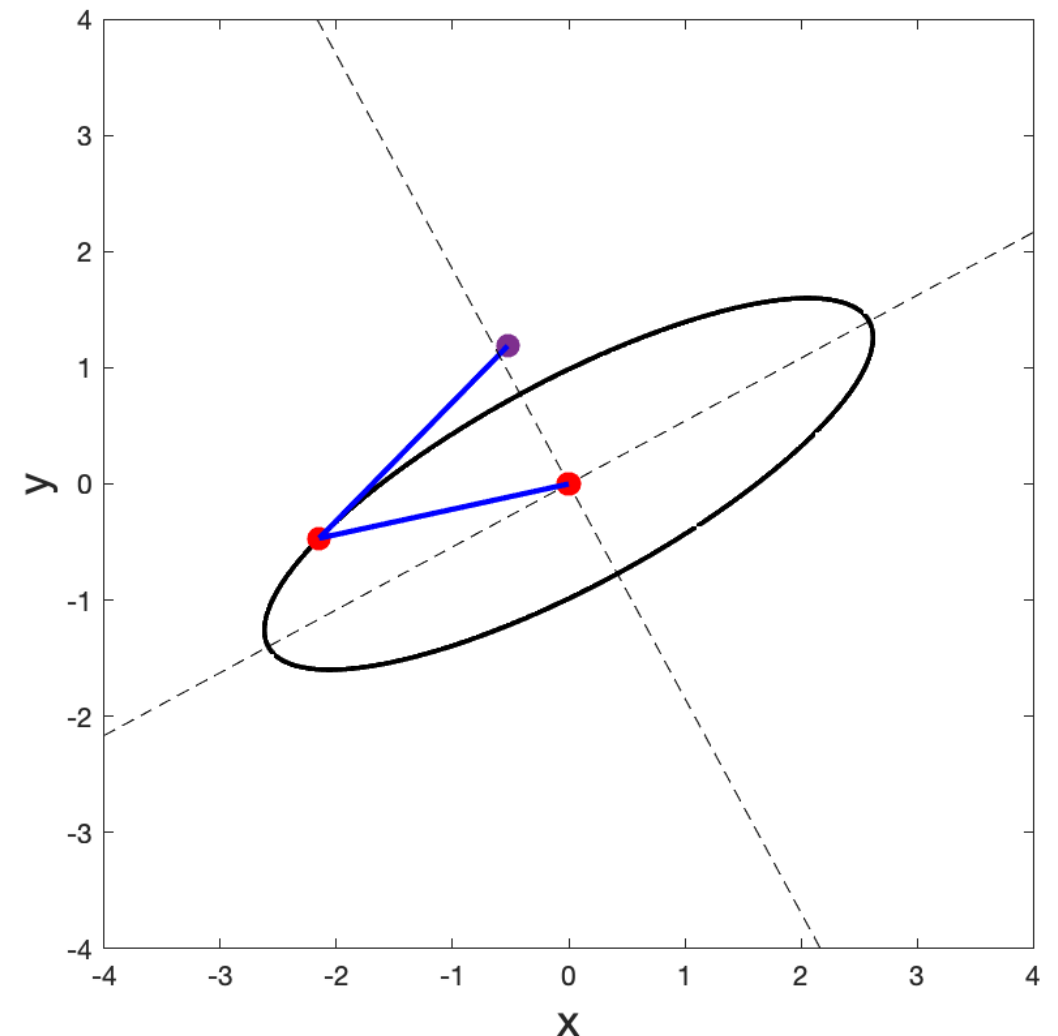
A-Conjugate directions

Diagonal matrix



For a diagonal matrix, orthogonal directions are what is needed to converge in n steps.

General matrix



For general matrices, “A-Conjugate” directions are what is needed to converge in n steps.

Conjugate Gradient Algorithm

The conjugate gradient algorithm improves on Steepest Descent by choosing better search directions \mathbf{p}_k .

Compute $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$

Set $\mathbf{p}_0 = \mathbf{r}_0$ **Search directions**

For $k = 0, 1, 2, \dots$

 Compute $A\mathbf{p}_k$

 Set $\alpha_k = (\mathbf{r}_k^T \mathbf{r}_k) / (\mathbf{p}_k^T A\mathbf{p}_k)$

 Set $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$

 Set $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k A\mathbf{p}_k$

 Set $\beta_k = (\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}) / (\mathbf{r}_k^T \mathbf{r}_k)$

 Set $\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$

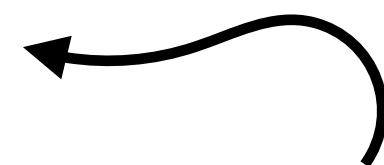
 if $\|\mathbf{r}_{k+1}\| < \tau$

 Stop

 end

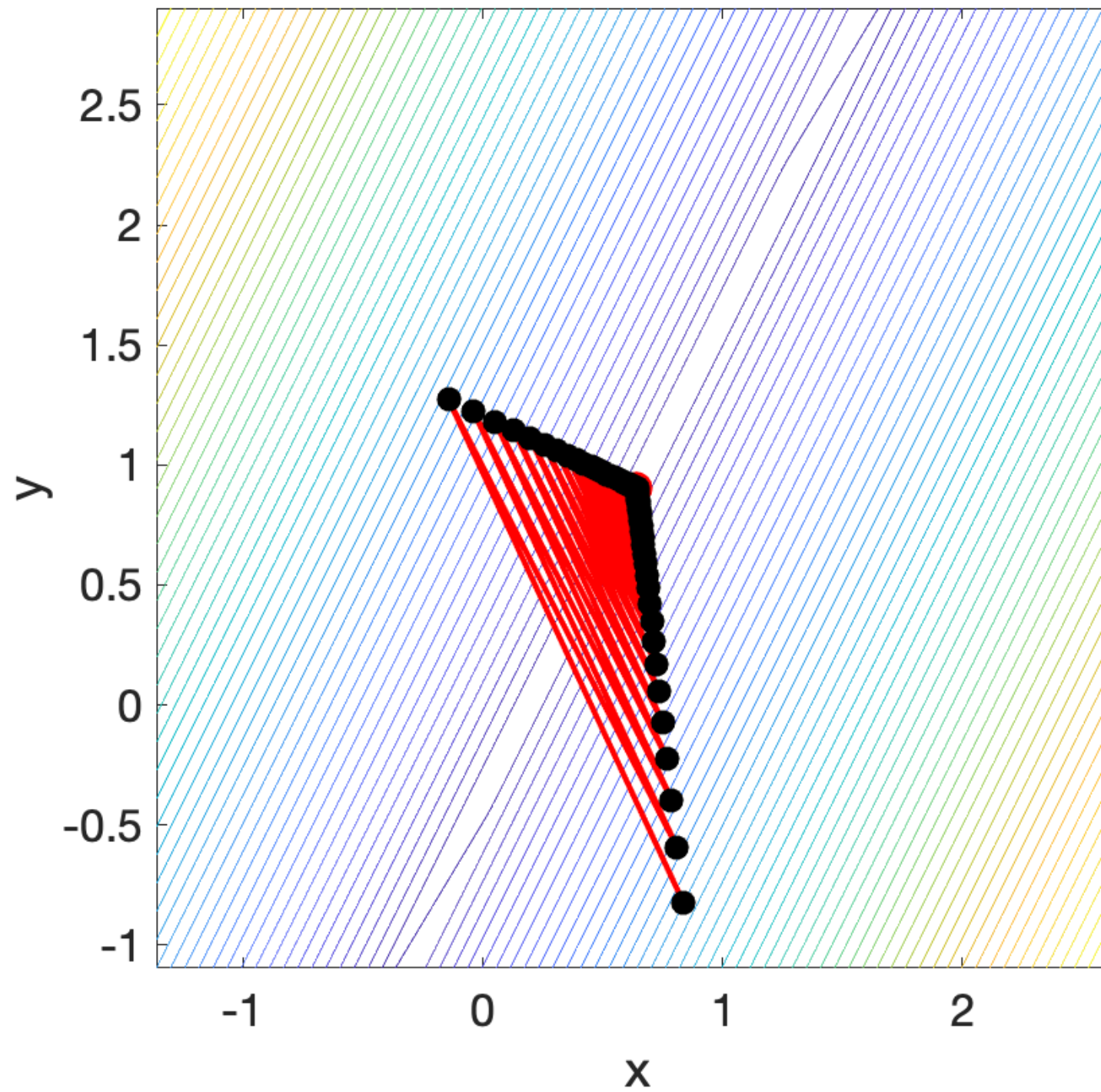
end

Solution is $\mathbf{x} \equiv \mathbf{x}_{k+1}$.

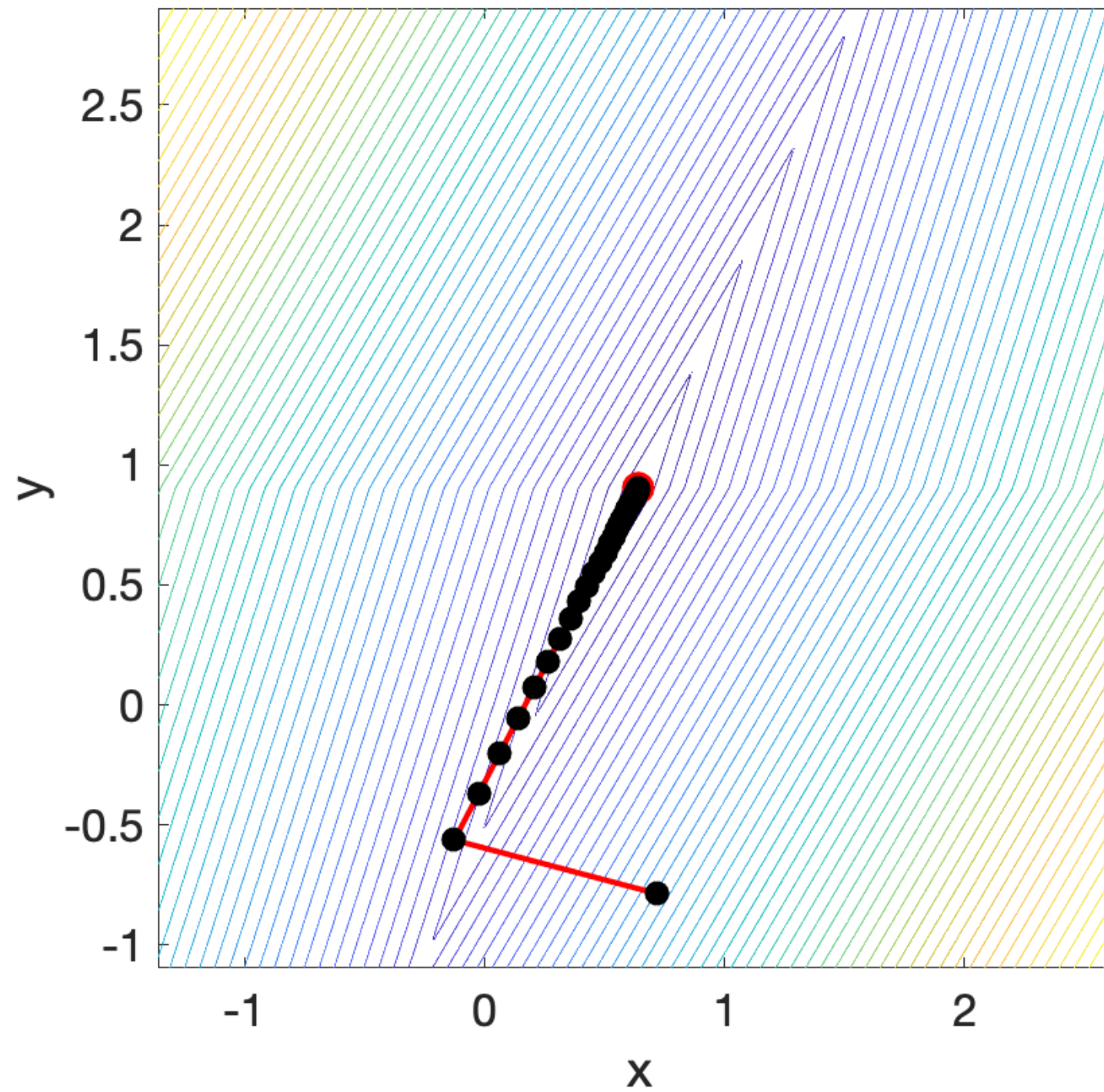


Update the search directions to get A-conjugate directions

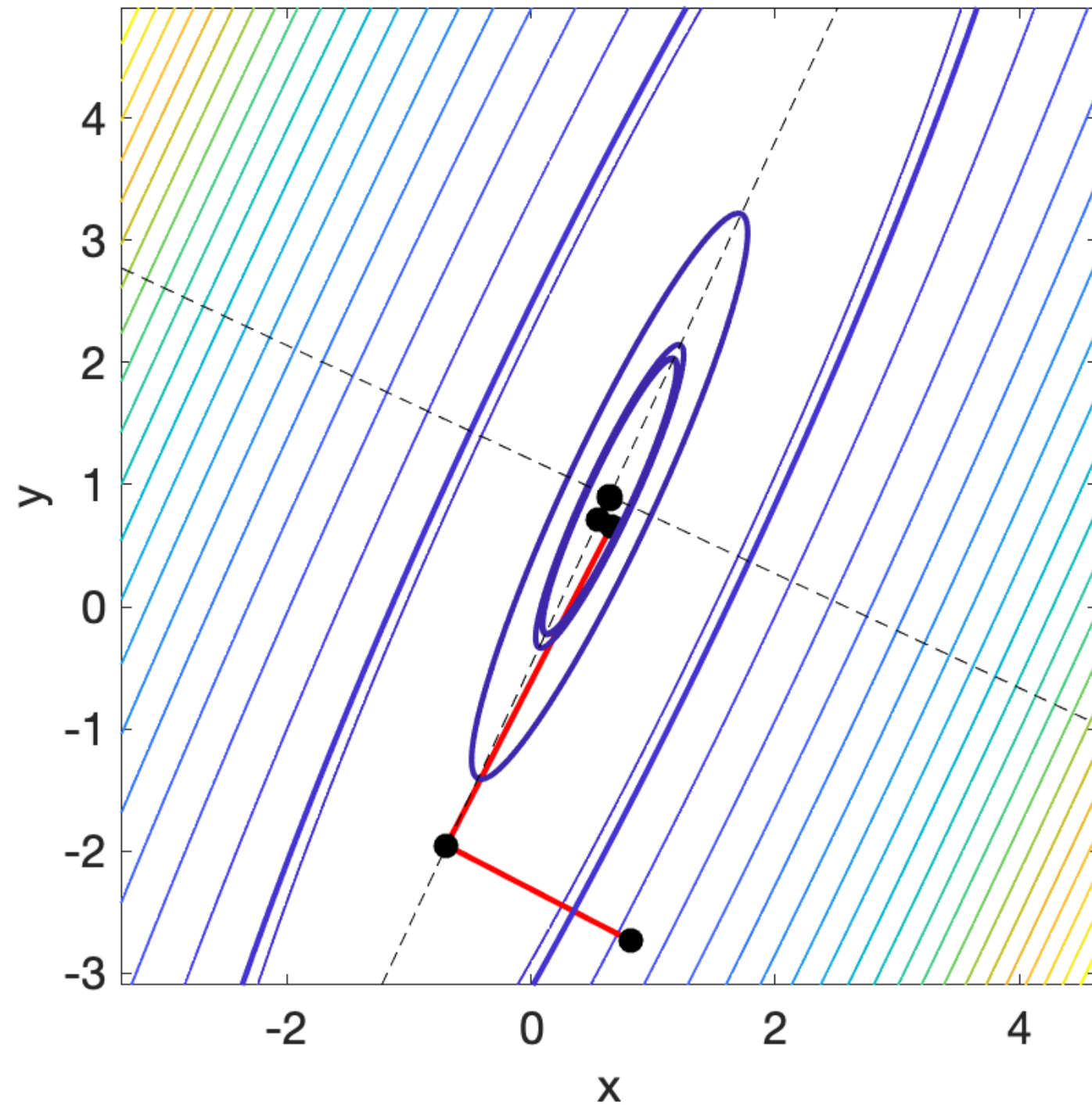
Jacobi



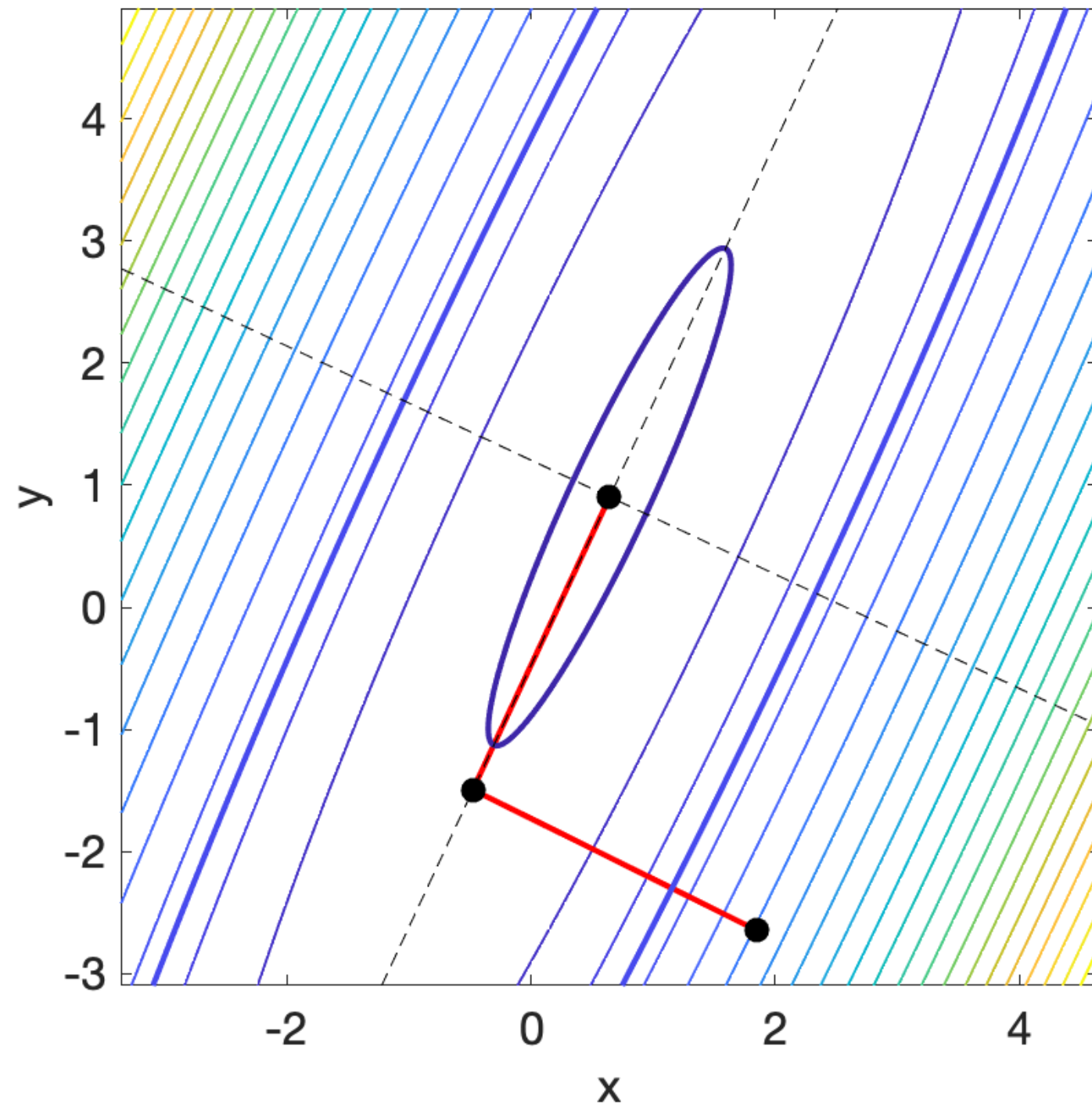
Gauss-Seidel



Steepest Descent



Conjugate Gradient



Convergence Behavior

