11. Linear algebra II: Power iteration and the QR factorisation

Last time

- Solving systems of linear equations $A\,\mathbf{x} = \mathbf{b}$
- Iterative methods: Jacobi & Gauss–Seidel
  - Do not always converge
- Direct method: Gaussian elimination
  - Reinterpreted as LU factorisation, $A = LU$

Goals for today

- Iterating matrix–vector multiplication: Power iteration
- Symmetric matrices
- Gram–Schmidt algorithm
- QR factorisation

Iterating matrix–vector multiplication

- Matrix–vector multiplication: basic linear algebra operation

Iterating matrix–vector multiplication

- Matrix–vector multiplication: basic linear algebra operation
- Sometimes called “matvec”

Iterating matrix–vector multiplication

- Matrix–vector multiplication: basic linear algebra operation
- Sometimes called "matvec"

- We saw that this occurs in the Jacobi method:

$$\mathbf{x}_{n+1} = \mathsf{A}\,\mathbf{x}_n$$

- Repeatedly multiply by the *same* matrix A

Iterating matrix–vector multiplication

- Matrix–vector multiplication: basic linear algebra operation
- Sometimes called “matvec”

- We saw that this occurs in the Jacobi method:

$$\mathbf{x}_{n+1} = \mathsf{A}\, \mathbf{x}_n$$

- Repeatedly multiply by the *same* matrix A

- What happens?
- For simplicity we restrict to real, symmetric matrices A

Collaboration I

Iterating matrix multiplication

1. Take the matrix

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$$

   Take any non-zero vector **x** and carry out the iteration

$$\mathbf{x}_{n+1} = \mathrm{A}\,\mathbf{x}_n$$

2. What happens?
3. How can we avoid that?
4. What equation does the resulting iteration solve?

The power iteration: Repeated matrix multiplication

- Applying the iteration $\mathbf{x}_{n+1} = \mathsf{A}\,\mathbf{x}_n$ often diverges

The power iteration: Repeated matrix multiplication

- Applying the iteration $\mathbf{x}_{n+1} = \mathsf{A}\,\mathbf{x}_n$ often diverges

- So **normalise** $\mathbf{x}_{n+1}$ at each iteration:

$$\mathbf{y}_{n+1} = \mathsf{A}\,\mathbf{x}_n$$
$$\mathbf{x}_{n+1} = \frac{\mathbf{y}_{n+1}}{\|\mathbf{y}_{n+1}\|}$$

- Then $\mathbf{x}_{n+1}$ converges to an $\mathbf{x}^*$ as $n \to \infty$

The power iteration: Repeated matrix multiplication

- Applying the iteration $\mathbf{x}_{n+1} = \mathsf{A}\,\mathbf{x}_n$ often diverges

- So **normalise** $\mathbf{x}_{n+1}$ at each iteration:

$$\mathbf{y}_{n+1} = \mathsf{A}\,\mathbf{x}_n$$
$$\mathbf{x}_{n+1} = \frac{\mathbf{y}_{n+1}}{\|\mathbf{y}_{n+1}\|}$$

- Then $\mathbf{x}_{n+1}$ converges to an $\mathbf{x}^*$ as $n \to \infty$

- Note that we are calculating $\mathsf{A}^n\,\mathbf{x}$
- Hence the name **power method** or **power iteration**

Iterated matrix multiplication II

- What does the limit $\mathbf{x}^*$ satisfy?

Iterated matrix multiplication II

- What does the limit $\mathbf{x}^*$ satisfy?

$$\mathbf{x}^* = \frac{\mathsf{A}\,\mathbf{x}^*}{\left\|\mathsf{A}\,\mathbf{x}^*\right\|}$$

Iterated matrix multiplication II

- What does the limit $\mathbf{x}^*$ satisfy?

$$\mathbf{x}^* = \frac{\mathsf{A}\,\mathbf{x}^*}{\|\mathsf{A}\,\mathbf{x}^*\|}$$

- So $\mathsf{A}\,\mathbf{x}^* = \lambda \mathbf{x}^* \quad$ with $\lambda \in \mathbb{R}$

Iterated matrix multiplication II

- What does the limit $\mathbf{x}^*$ satisfy?

$$\mathbf{x}^* = \frac{\mathrm{A}\,\mathbf{x}^*}{\|\mathrm{A}\,\mathbf{x}^*\|}$$

- So $\mathrm{A}\,\mathbf{x}^* = \lambda \mathbf{x}^* \quad$ with $\lambda \in \mathbb{R}$
- $\mathrm{A}\,\mathbf{x}^*$ and $\mathbf{x}^*$ *point in the same direction*

Iterated matrix multiplication II

- What does the limit $\mathbf{x}^*$ satisfy?

$$\mathbf{x}^* = \frac{\mathrm{A}\,\mathbf{x}^*}{\|\mathrm{A}\,\mathbf{x}^*\|}$$

- So $\mathrm{A}\,\mathbf{x}^* = \lambda \mathbf{x}^* \quad$ with $\lambda \in \mathbb{R}$
- $\mathrm{A}\,\mathbf{x}^*$ and $\mathbf{x}^*$ *point in the same direction*

- $\mathbf{x}^*$ is an **eigenvector** of A
- With **eigenvalue** $\|\mathrm{A}\,\mathbf{x}^*\| / \|\mathbf{x}^*\|$

Unit vectors

- Recall that any vector $\mathbf{v}$ can be decomposed as

$$\mathbf{v} = \|\mathbf{v}\| \, \hat{\mathbf{v}}$$

Unit vectors

- Recall that any vector $\mathbf{v}$ can be decomposed as

$$\mathbf{v} = \|\mathbf{v}\| \, \hat{\mathbf{v}}$$

- The **(Euclidean) norm** or **length** of $\mathbf{v}$ is

$$\|\mathbf{v}\|_2 := \sqrt{\sum_{i=1}^{n} v_i^2}$$

Unit vectors

- Recall that any vector $\mathbf{v}$ can be decomposed as

$$\mathbf{v} = \|\mathbf{v}\| \, \hat{\mathbf{v}}$$

- The **(Euclidean) norm** or **length** of $\mathbf{v}$ is

$$\|\mathbf{v}\|_2 := \sqrt{\sum_{i=1}^{n} v_i^2}$$

- $\|\mathbf{v}\|_2^2 = \mathbf{v}^\mathsf{T}\mathbf{v}$ – Householder notation

Unit vectors

- Recall that any vector $\mathbf{v}$ can be decomposed as

$$\mathbf{v} = \|\mathbf{v}\| \, \hat{\mathbf{v}}$$

- The **(Euclidean) norm** or **length** of $\mathbf{v}$ is

$$\|\mathbf{v}\|_2 := \sqrt{\sum_{i=1}^{n} v_i^2}$$

- $\|\mathbf{v}\|_2^2 = \mathbf{v}^\mathsf{T}\mathbf{v}$ – Householder notation
- $\hat{\mathbf{v}}$ is a **unit vector** (norm 1), the **direction** of $\mathbf{v}$

Convergence of the power iteration

- Questions we need to answer about power iteration:

- Which eigenvector does it converge to?

Convergence of the power iteration

- Questions we need to answer about power iteration:

- Which eigenvector does it converge to?
- Does power iteration always converge?

Convergence of the power iteration

- Questions we need to answer about power iteration:

- Which eigenvector does it converge to?
- Does power iteration always converge?
- How fast is the convergence?

Convergence of the power iteration

- Questions we need to answer about power iteration:

- Which eigenvector does it converge to?
- Does power iteration always converge?
- How fast is the convergence?

- Problem set!

What do we need to run power iteration?

- For power iteration we need a matvec $\mathsf{A}\,\mathbf{x}$

What do we need to run power iteration?

- For power iteration we need a matvec $A\,\mathbf{x}$
- In face we need the *result* of a matvec

What do we need to run power iteration?

- For power iteration we need a matvec $A\,\mathbf{x}$
- In face we need the *result* of a matvec

- Often matrices are huge, but have many zeros – **sparse**

What do we need to run power iteration?

- For power iteration we need a matvec $\mathsf{A}\,\mathbf{x}$
- In face we need the *result* of a matvec

- Often matrices are huge, but have many zeros – **sparse**
- Then we can do $\mathsf{A}\,\mathbf{x}$ more efficiently

What do we need to run power iteration?

- For power iteration we need a matvec $\mathsf{A}\,\mathbf{x}$
- In face we need the *result* of a matvec

- Often matrices are huge, but have many zeros – **sparse**
- Then we can do $\mathsf{A}\,\mathbf{x}$ more efficiently

- We may even be able to calculate $\mathsf{A}\,\mathbf{x}$ without storing A !

What do we need to run power iteration?

- For power iteration we need a matvec $\mathrm{A}\,\mathbf{x}$
- In face we need the *result* of a matvec

- Often matrices are huge, but have many zeros – **sparse**
- Then we can do $\mathrm{A}\,\mathbf{x}$ more efficiently

- We may even be able to calculate $\mathrm{A}\,\mathbf{x}$ without storing A !
- **Krylov subspace** methods take advantage of this
- Operate on the subspace spanned by $\{\mathbf{x}, \mathrm{A}\,\mathbf{x}, \mathrm{A}^2\,\mathbf{x}, \ldots\}$

Real, symmetric matrices

- From now on (in this lecture) we assume that

  A *is an* $(n \times n)$ *real, symmetric matrix*

- i.e. $\mathsf{A}^\mathsf{T} = \mathsf{A}$

Real, symmetric matrices

- From now on (in this lecture) we assume that

  A *is an* $(n \times n)$ *real, symmetric matrix*

- i.e. $\mathsf{A}^{\mathsf{T}} = \mathsf{A}$

- Such matrices occur naturally in many situations:
  - Hessian matrix of 2nd partial derivatives is symmetric
  - Covariance matrix in statistics

Symmetric matrices II

- Let's recall some facts about symmetric matrices:

Symmetric matrices II

- Let's recall some facts about symmetric matrices:

- They have (a basis of) n eigenvectors
- i.e. they are **diagonalisable**

Symmetric matrices II

- Let's recall some facts about symmetric matrices:

- They have (a basis of) n eigenvectors
- i.e. they are **diagonalisable**
- Eigenvectors are mutually **orthogonal**

Symmetric matrices II

- Let's recall some facts about symmetric matrices:

- They have (a basis of) n eigenvectors
- i.e. they are **diagonalisable**
- Eigenvectors are mutually **orthogonal**

- Recall: **v** and **w** are **orthogonal** if

$$\mathbf{v} \cdot \mathbf{w} = \mathbf{v}^\mathsf{T} \mathbf{w} = 0$$

Collaboration II

Simultaneous power iteration using QR

1. What happens if we perform power iteration with *two* distinct (non-zero) initial vectors (but the same matrix A)
2. What would we like to happen?
3. How can we achieve this?

Simultaneous power iteration

- If we do the power iteration with k vectors, we could hope that they converge to the first k eigenvectors
- But they all “bunch up” towards the first eigenvector

Simultaneous power iteration

- If we do the power iteration with k vectors, we could hope that they converge to the first k eigenvectors
- But they all “bunch up” towards the first eigenvector

- To avoid this we need to “push them apart”

Simultaneous power iteration

- If we do the power iteration with k vectors, we could hope that they converge to the first k eigenvectors
- But they all “bunch up” towards the first eigenvector

- To avoid this we need to “push them apart”
- Make them **orthogonal**
- How can we do so?

Collaboration III

Orthogonalisation

1. Given a pair of two vectors **u** and **v**.
   How could we find a new pair of vectors $\mathbf{q}_1$ and $\mathbf{q}_2$ that are orthogonal, but span the same plane?
2. How could we extend this to n vectors?

Orthogonal linear combinations

- Suppose $\mathbf{v}_1, \ldots, \mathbf{v}_n$ are all mutually orthogonal
- Suppose we want to solve the linear equations

$$x_1 \mathbf{v}_1 + \cdots + x_n \mathbf{v}_n = \mathbf{b}$$

- This is equivalent to $A\,\mathbf{x} = \mathbf{b}$

Orthogonal linear combinations

- Suppose $\mathbf{v}_1, \ldots, \mathbf{v}_n$ are all mutually orthogonal
- Suppose we want to solve the linear equations

$$x_1\mathbf{v}_1 + \cdots + x_n\mathbf{v}_n = \mathbf{b}$$

- This is equivalent to $\mathsf{A}\,\mathbf{x} = \mathbf{b}$
- It's now easy!: $x_i = \mathbf{b} \cdot \mathbf{v}_i$

Orthogonal linear combinations

- Suppose $\mathbf{v}_1, \ldots, \mathbf{v}_n$ are all mutually orthogonal
- Suppose we want to solve the linear equations

$$x_1\mathbf{v}_1 + \cdots + x_n\mathbf{v}_n = \mathbf{b}$$

- This is equivalent to A $\mathbf{x} = \mathbf{b}$
- It's now easy!: $x_i = \mathbf{b} \cdot \mathbf{v}_i$
- x_i = component in the direction of $\mathbf{v}_i$

Orthogonal linear combinations

- Suppose $\mathbf{v}_1, \ldots, \mathbf{v}_n$ are all mutually orthogonal
- Suppose we want to solve the linear equations

$$x_1\mathbf{v}_1 + \cdots + x_n\mathbf{v}_n = \mathbf{b}$$

- This is equivalent to A $\mathbf{x} = \mathbf{b}$
- It's now easy!: $x_i = \mathbf{b} \cdot \mathbf{v}_i$
- x_i = component in the direction of $\mathbf{v}_i$
- Orthogonal decomposition:

$$\mathbf{b} = (\mathbf{b} \cdot \mathbf{v}_1)\mathbf{v}_1 + \cdots + (\mathbf{b} \cdot \mathbf{v}_n)\mathbf{v}_n$$

Orthogonalization

- Suppose we have a set of linearly independent vectors
- Can we create an *orthogonal* set of vectors from them?

Orthogonalization

- Suppose we have a set of linearly independent vectors
- Can we create an *orthogonal* set of vectors from them?

- For two vectors $\mathbf{u}$ and $\mathbf{v}$, take $\mathbf{u}$ as the first vector

Orthogonalization

- Suppose we have a set of linearly independent vectors
- Can we create an *orthogonal* set of vectors from them?

- For two vectors **u** and **v**, take **u** as the first vector
- Make an orthogonal decomposition of **v** as

  v = (part of **v** in the direction of **u**) + (the rest)

Orthogonalization II

- Define $\mathbf{q}_1 := (\mathbf{v} \cdot \hat{\mathbf{u}})\hat{\mathbf{u}}$ = part of $\mathbf{v}$ in direction of $\mathbf{u}$

Orthogonalization II

- Define $\mathbf{q}_1 := (\mathbf{v} \cdot \hat{\mathbf{u}})\hat{\mathbf{u}}$ = part of $\mathbf{v}$ in direction of $\mathbf{u}$
- Define $\mathbf{q}_2 := \mathbf{v} - \mathbf{q}_1 = \mathbf{v} - (\mathbf{v} \cdot \hat{\mathbf{u}})\hat{\mathbf{u}}$

Orthogonalization II

- Define $\mathbf{q}_1 := (\mathbf{v} \cdot \hat{\mathbf{u}})\hat{\mathbf{u}}$ = part of **v** in direction of **u**
- Define $\mathbf{q}_2 := \mathbf{v} - \mathbf{q}_1 = \mathbf{v} - (\mathbf{v} \cdot \hat{\mathbf{u}})\hat{\mathbf{u}}$
- Then

$$\mathbf{q}_1 \cdot \mathbf{q}_2 = \mathbf{q}_1 \cdot \mathbf{v} - \mathbf{q}_1 \cdot \mathbf{q}_1 = (\mathbf{v} \cdot \hat{\mathbf{u}})(\hat{\mathbf{u}} \cdot \mathbf{v}) - (\mathbf{v} \cdot \hat{\mathbf{u}})^2 = 0$$

Orthogonalization II

- Define $\mathbf{q}_1 := (\mathbf{v} \cdot \hat{\mathbf{u}})\hat{\mathbf{u}}$ = part of $\mathbf{v}$ in direction of $\mathbf{u}$
- Define $\mathbf{q}_2 := \mathbf{v} - \mathbf{q}_1 = \mathbf{v} - (\mathbf{v} \cdot \hat{\mathbf{u}})\hat{\mathbf{u}}$
- Then

$$\mathbf{q}_1 \cdot \mathbf{q}_2 = \mathbf{q}_1 \cdot \mathbf{v} - \mathbf{q}_1 \cdot \mathbf{q}_1 = (\mathbf{v} \cdot \hat{\mathbf{u}})(\hat{\mathbf{u}} \cdot \mathbf{v}) - (\mathbf{v} \cdot \hat{\mathbf{u}})^2 = 0$$

- i.e. $\mathbf{q}_1$ and $\mathbf{q}_2$ are indeed orthogonal

Orthogonalization II

- Define $\mathbf{q}_1 := (\mathbf{v} \cdot \hat{\mathbf{u}})\hat{\mathbf{u}}$ = part of **v** in direction of **u**
- Define $\mathbf{q}_2 := \mathbf{v} - \mathbf{q}_1 = \mathbf{v} - (\mathbf{v} \cdot \hat{\mathbf{u}})\hat{\mathbf{u}}$
- Then

$$\mathbf{q}_1 \cdot \mathbf{q}_2 = \mathbf{q}_1 \cdot \mathbf{v} - \mathbf{q}_1 \cdot \mathbf{q}_1 = (\mathbf{v} \cdot \hat{\mathbf{u}})(\hat{\mathbf{u}} \cdot \mathbf{v}) - (\mathbf{v} \cdot \hat{\mathbf{u}})^2 = 0$$

- i.e. $\mathbf{q}_1$ and $\mathbf{q}_2$ are indeed orthogonal
- We can write $\mathbf{q}_1 = (\frac{1}{\|\mathbf{u}\|^2}\mathbf{u}\mathbf{u}^\mathsf{T})\mathbf{v}$
- In terms of a **projector**
- $\mathbf{q}_2 = (\mathbf{I} - \frac{\mathbf{u}\mathbf{u}^\mathsf{T}}{\mathbf{u}^\mathsf{T}\mathbf{u}})\mathbf{v}$

Orthogonalization III

- Generalize to n vectors:

Orthogonalization III

- Generalize to n vectors:
- Define $\mathbf{u}_1 = \mathbf{v}_1; \quad \mathbf{q}_1 := \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|}$

Orthogonalization III

- Generalize to n vectors:
- Define $\mathbf{u}_1 = \mathbf{v}_1; \quad \mathbf{q}_1 := \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|}$
- Define $\mathbf{u}_2 = \mathbf{v}_2 - (\mathbf{v}_2 \cdot \mathbf{q}_1)\mathbf{q}_1; \quad \mathbf{q}_2 := \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|}$

Orthogonalization III

- Generalize to n vectors:
- Define $\mathbf{u}_1 = \mathbf{v}_1; \quad \mathbf{q}_1 := \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|}$
- Define $\mathbf{u}_2 = \mathbf{v}_2 - (\mathbf{v}_2 \cdot \mathbf{q}_1)\mathbf{q}_1; \quad \mathbf{q}_2 := \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|}$
- Define $\mathbf{u}_3 = \mathbf{v}_3 - (\mathbf{v}_3 \cdot \mathbf{q}_2)\mathbf{q}_2 - (\mathbf{v}_3 \cdot \mathbf{q}_1)\mathbf{q}_1; \quad \mathbf{q}_3 := \frac{\mathbf{u}_3}{\|\mathbf{u}_3\|}$
- etc.

Orthogonalization III

- Generalize to n vectors:
- Define $\mathbf{u}_1 = \mathbf{v}_1; \quad \mathbf{q}_1 := \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|}$
- Define $\mathbf{u}_2 = \mathbf{v}_2 - (\mathbf{v}_2 \cdot \mathbf{q}_1)\mathbf{q}_1; \quad \mathbf{q}_2 := \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|}$
- Define $\mathbf{u}_3 = \mathbf{v}_3 - (\mathbf{v}_3 \cdot \mathbf{q}_2)\mathbf{q}_2 - (\mathbf{v}_3 \cdot \mathbf{q}_1)\mathbf{q}_1; \quad \mathbf{q}_3 := \frac{\mathbf{u}_3}{\|\mathbf{u}_3\|}$
- etc.

- This is the (classical) **Gram–Schmidt** algorithm
- It produces a set of n mutually **orthogonal** vectors $\mathbf{q}_i$

QR factorization

- Given a matrix A, apply Gram–Schmidt to the columns $\mathbf{v}_i$ of A to get orthogonal $\mathbf{q}_i$

QR factorization

- Given a matrix A, apply Gram–Schmidt to the columns $\mathbf{v}_i$ of A to get orthogonal $\mathbf{q}_i$
- We can write the columns in terms of the $\mathbf{q}_i$:

$$\mathbf{v}_1 = (\mathbf{v}_1 \cdot \mathbf{q}_1)\mathbf{q}_1 \tag{1}$$

$$\mathbf{v}_2 = (\mathbf{v}_2 \cdot \mathbf{q}_1)\mathbf{q}_1 + (\mathbf{v}_2 \cdot \mathbf{q}_2)\mathbf{q}_2 \tag{2}$$

$$\mathbf{v}_3 = (\mathbf{v}_3 \cdot \mathbf{q}_1)\mathbf{q}_1 + (\mathbf{v}_3 \cdot \mathbf{q}_2)\mathbf{q}_2 + (\mathbf{v}_3 \cdot \mathbf{q}_3)\mathbf{q}_3 \tag{3}$$

QR factorization

- Given a matrix A, apply Gram–Schmidt to the columns $\mathbf{v}_i$ of A to get orthogonal $\mathbf{q}_i$
- We can write the columns in terms of the $\mathbf{q}_i$:

$$\mathbf{v}_1 = (\mathbf{v}_1 \cdot \mathbf{q}_1)\mathbf{q}_1 \tag{1}$$
$$\mathbf{v}_2 = (\mathbf{v}_2 \cdot \mathbf{q}_1)\mathbf{q}_1 + (\mathbf{v}_2 \cdot \mathbf{q}_2)\mathbf{q}_2 \tag{2}$$
$$\mathbf{v}_3 = (\mathbf{v}_3 \cdot \mathbf{q}_1)\mathbf{q}_1 + (\mathbf{v}_3 \cdot \mathbf{q}_2)\mathbf{q}_2 + (\mathbf{v}_3 \cdot \mathbf{q}_3)\mathbf{q}_3 \tag{3}$$

- We get $\mathsf{A} = \mathsf{QR}$
- With an **orthogonal** matrix Q with orthonormal columns $\mathbf{q}_i$

QR factorization

- Given a matrix A, apply Gram–Schmidt to the columns $\mathbf{v}_i$ of A to get orthogonal $\mathbf{q}_i$
- We can write the columns in terms of the $\mathbf{q}_i$:

$$\mathbf{v}_1 = (\mathbf{v}_1 \cdot \mathbf{q}_1)\mathbf{q}_1 \tag{1}$$

$$\mathbf{v}_2 = (\mathbf{v}_2 \cdot \mathbf{q}_1)\mathbf{q}_1 + (\mathbf{v}_2 \cdot \mathbf{q}_2)\mathbf{q}_2 \tag{2}$$

$$\mathbf{v}_3 = (\mathbf{v}_3 \cdot \mathbf{q}_1)\mathbf{q}_1 + (\mathbf{v}_3 \cdot \mathbf{q}_2)\mathbf{q}_2 + (\mathbf{v}_3 \cdot \mathbf{q}_3)\mathbf{q}_3 \tag{3}$$

- We get $\mathsf{A} = \mathsf{QR}$
- With an **orthogonal** matrix Q with orthonormal columns $\mathbf{q}_i$
- And upper-triangular R (since we have *column* operations)

QR factorization

- Given a matrix A, apply Gram–Schmidt to the columns $\mathbf{v}_i$ of A to get orthogonal $\mathbf{q}_i$
- We can write the columns in terms of the $\mathbf{q}_i$:

$$\mathbf{v}_1 = (\mathbf{v}_1 \cdot \mathbf{q}_1)\mathbf{q}_1 \tag{1}$$
$$\mathbf{v}_2 = (\mathbf{v}_2 \cdot \mathbf{q}_1)\mathbf{q}_1 + (\mathbf{v}_2 \cdot \mathbf{q}_2)\mathbf{q}_2 \tag{2}$$
$$\mathbf{v}_3 = (\mathbf{v}_3 \cdot \mathbf{q}_1)\mathbf{q}_1 + (\mathbf{v}_3 \cdot \mathbf{q}_2)\mathbf{q}_2 + (\mathbf{v}_3 \cdot \mathbf{q}_3)\mathbf{q}_3 \tag{3}$$

- We get $\mathsf{A} = \mathsf{QR}$
- With an **orthogonal** matrix Q with orthonormal columns $\mathbf{q}_i$
- And upper-triangular R (since we have *column* operations)
- However, classical Gram–Schmidt is **numerically**

Simultaneous power iteration again

- Start with a set of vectors $\mathbf{v}_k$

Simultaneous power iteration again

- Start with a set of vectors $\mathbf{v}_k$
- Form the matrix with columns $\mathbf{v}_k$
- Find QR factorisation to extract Q_0

Simultaneous power iteration again

- Start with a set of vectors $\mathbf{v}_k$
- Form the matrix with columns $\mathbf{v}_k$
- Find QR factorisation to extract Q_0
- Simultaneous power iteration is equivalent to:

$$\mathsf{Z}_{k+1} = \mathsf{A}\mathsf{Q}_k$$

$$\mathsf{Q}_{k+1}\mathsf{R}_{k+1} = \mathsf{Z}_k$$

Simultaneous power iteration again

- Start with a set of vectors $\mathbf{v}_k$
- Form the matrix with columns $\mathbf{v}_k$
- Find QR factorisation to extract Q_0
- Simultaneous power iteration is equivalent to:

$$\mathsf{Z}_{k+1} = \mathsf{A}\mathsf{Q}_k$$

$$\mathsf{Q}_{k+1}\mathsf{R}_{k+1} = \mathsf{Z}_k$$

- Closely related to **QR algorithm** for eigen-decomposition
- Trefethen & Bau, *Numerical Linear Algebra*

Solving linear equations with QR

- QR factorisation gives another method to solve $\mathsf{A}\,\mathbf{x} = \mathbf{b}$:

Solving linear equations with QR

- QR factorisation gives another method to solve $A\,\mathbf{x} = \mathbf{b}$:

- Factorize $A = QR$

Solving linear equations with QR

- QR factorisation gives another method to solve $\mathsf{A}\,\mathbf{x} = \mathbf{b}$:
- Factorize $\mathsf{A} = \mathsf{QR}$
- Solve $\mathsf{Q}\,\mathbf{y} = \mathbf{b}$

Solving linear equations with QR

- QR factorisation gives another method to solve $\mathsf{A}\,\mathbf{x} = \mathbf{b}$:

- Factorize $\mathsf{A} = \mathsf{Q}\mathsf{R}$

- Solve $\mathsf{Q}\,\mathbf{y} = \mathbf{b}$
- Solution: $\mathbf{y} = \mathsf{Q}^{\mathsf{T}}\,\mathbf{b}$

Solving linear equations with QR

- QR factorisation gives another method to solve $\mathsf{A}\,\mathbf{x} = \mathbf{b}$:

- Factorize $\mathsf{A} = \mathsf{QR}$

- Solve $\mathsf{Q}\,\mathbf{y} = \mathbf{b}$
- Solution: $\mathbf{y} = \mathsf{Q}^{\mathsf{T}}\,\mathbf{b}$
- Or $y_i = (\mathbf{b} \cdot \mathbf{q}_i)\mathbf{q}_i$

Solving linear equations with QR

- QR factorisation gives another method to solve $\mathsf{A}\,\mathbf{x} = \mathbf{b}$:

- Factorize $\mathsf{A} = \mathsf{QR}$

- Solve $\mathsf{Q}\,\mathbf{y} = \mathbf{b}$
- Solution: $\mathbf{y} = \mathsf{Q}^{\mathsf{T}}\,\mathbf{b}$
- Or $y_i = (\mathbf{b} \cdot \mathbf{q}_i)\mathbf{q}_i$

- Then solve $\mathsf{R}\,\mathbf{x} = \mathbf{y}$ by backward substitution

Summary

- Power iteration $\mathbf{x}_{n+1} = \mathsf{A}\,\mathbf{x}_n$
- Converges to an eigenvector of A

- **Orthogonality** arises as a key concept
- We can **orthogonalise** a set of vectors using the Gram–Schmidt algorithm

- This corresponds to a QR factorisation of a matrix