PHY604 Lecture 11

October 12, 2023

Review: Gaussian elimination

- Main general technique for solving **A x = b**
  - Does not involve matrix inversion
  - For "special" matrices, faster techniques may apply
- Involves forward-elimination and back-substitution
- Partial-pivoting:
  - Interchange of rows to move the one with the largest element in the current column to the top
  - (Full pivoting would allow for row and column swaps—more complicated)
- Scaled pivoting
  - Consider largest element relative to all entries in its row
  - Further reduces roundoff when elements vary in magnitude greatly
- Row echelon form: This is the upper-triangular form that the matrix is in after forward elimination

Review: Matrix determinants with Gaussian elimination

- Once we have done forward substitution and obtained a row echelon matrix it is trivial to calculate the determinant:

$$\det(\mathbf{A}) = (-1)^{N_{\text{pivot}}} \prod_{i=1}^{N} A_{ii}^{\text{row-echelon}}$$

- Every time we pivoted in the forward substitution, we change the sign

Review: Matrix inverse with Gaussian elimination

- We can also use Gaussian elimination to fin the inverse of a matrix
- We would like to find $\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$
- We can use Gaussian elimination to solve: $\mathbf{A}\,\mathbf{x}_i = \mathbf{e}_i$
  - $\mathbf{e}_i$ is a column of the identity:

$$\mathbf{e}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \end{bmatrix}, \quad \mathbf{e}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix}, \quad \mathbf{e}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \end{bmatrix}, \ldots, \; \mathbf{e}_N = \begin{bmatrix} \vdots \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

  - $\mathbf{x}_i$ is a column of the inverse:

$$\mathbf{A}^{-1} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & \ldots & \mathbf{x}_N \end{bmatrix}$$

Today's lecture:
More on linear and nonlinear algebra

- Singular and banded matrices
- LU decomposition
- Iterative methods
- Eigensystems

Singular matrix

- If a matrix has a vanishing determinant, then the system is not solvable
- Common way for this to enter, one equation in the system is a linear combination of some others
- Not always easy to detect from the start

Singular and close to singular matrices

- Condition number: Measures how close to singular we are
  - How much **x** would change with a small change in **b**

$$\mathrm{cond}(\mathbf{A}) = ||\mathbf{A}||\ ||\mathbf{A}^{-1}||$$

  - Requires defining a norm of **A**
    - https://en.wikipedia.org/wiki/Matrix_norm
  - See, e.g., numpy implementation:
    - https://numpy.org/doc/stable/reference/generated/numpy.linalg.cond.html

- Rule of thumb: $\dfrac{||\mathbf{x}^{\mathrm{exact}} - \mathbf{x}^{\mathrm{calc}}||}{||\mathbf{x}^{\mathrm{exact}}||} \simeq \mathrm{cond}(\mathbf{A}) \cdot \epsilon^{\mathrm{machine}}$

Tridiagonal and banded matrices

- We saw this type of matrix when solving for cubic spline coefficients:

$$\begin{pmatrix} 4\Delta x & \Delta x & & & & \\ \Delta x & 4\Delta x & \Delta x & & & \\ & \Delta x & 4\Delta x & \Delta x & & \\ & & \ddots & \ddots & \ddots & \\ & & & \Delta x & 4\Delta x & \Delta x \\ & & & & \Delta x & 4\Delta x \end{pmatrix} \begin{pmatrix} p''_1 \\ p''_2 \\ p''_3 \\ \vdots \\ p''_{n-2} \\ p''_{n-1} \end{pmatrix} = \frac{6}{\Delta x} \begin{pmatrix} f_0 - 2f_1 + f_2 \\ f_1 - 2f_2 + f_3 \\ f_2 - 2f_3 + f_4 \\ \vdots \\ f_{n-3} - 2f_{n-2} + f_{n-1} \\ f_{n-2} - 2f_{n-1} + f_n \end{pmatrix}$$

- Often come up in physical situations
- These types of matrices can be efficiently solved with Gaussian elimination

Gaussian elimination for banded matrices

- Only need to do Gaussian elimination steps for *m* nonzero elements below given row (*m* is less than the number of diagonal bands)
- Example:

$$\begin{pmatrix} 2 & 1 & 0 & 0 \\ 3 & 4 & -5 & 0 \\ 0 & -4 & 3 & 5 \\ 0 & 0 & 1 & 3 \end{pmatrix} \to \begin{pmatrix} 2 & 1 & 0 & 0 \\ 0 & 2.5 & -5 & 0 \\ 0 & -4 & 3 & 5 \\ 0 & 0 & 1 & 3 \end{pmatrix} \to \begin{pmatrix} 2 & 1 & 0 & 0 \\ 0 & 2.5 & -5 & 0 \\ 0 & 0 & -5 & 5 \\ 0 & 0 & 1 & 3 \end{pmatrix} \to \begin{pmatrix} 2 & 1 & 0 & 0 \\ 0 & 2.5 & -5 & 0 \\ 0 & 0 & -5 & 5 \\ 0 & 0 & 0 & 4 \end{pmatrix}$$

Today's lecture:
More on linear and nonlinear algebra

- Singular and banded matrices

- LU decomposition

- Iterative methods

- Eigensystems

LU decomposition (Newman Ch. 6)

- Often happens that we would like to solve: $\mathbf{A}\mathbf{x}_i = \mathbf{v}_i$ for the same **A** but many **v**
  - For example, our implementation for the inverse
  - Wasteful to do Gaussian elimination over and over, we will always get the same row echelon matrix, just $\mathbf{v}_i$ will be different
  - Instead, we should keep track of operations we did to $\mathbf{v}_1$ and use them over and over
- Consider a general 4 x 4 matrix:

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix}$$

- Let’s perform Gaussian elimination

LU decomposition: First GE step

- Write the first step of the GE as:

$$\frac{1}{a_{00}}\begin{pmatrix} 1 & 0 & 0 & 0 \\ -a_{10} & a_{00} & 0 & 0 \\ -a_{20} & 0 & a_{00} & 0 \\ -a_{30} & 0 & 0 & a_{00} \end{pmatrix}\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} 1 & b_{01} & b_{02} & b_{03} \\ 0 & b_{11} & b_{12} & b_{13} \\ 0 & b_{21} & b_{22} & b_{23} \\ 0 & b_{31} & b_{32} & b_{33} \end{pmatrix}$$

- Where the *b*'s are some linear combination of *a* coefficients
- The first matrix on the LHS is a lower triangular matrix we call:

$$\mathbf{L}_0 \equiv \frac{1}{a_{00}}\begin{pmatrix} 1 & 0 & 0 & 0 \\ -a_{10} & a_{00} & 0 & 0 \\ -a_{20} & 0 & a_{00} & 0 \\ -a_{30} & 0 & 0 & a_{00} \end{pmatrix}$$

LU decomposition: Second LU step

$$\frac{1}{b_{11}}\begin{pmatrix} b_{11} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -b_{21} & b_{11} & 0 \\ 0 & -b_{31} & 0 & b_{11} \end{pmatrix} \begin{pmatrix} 1 & b_{01} & b_{02} & b_{03} \\ 0 & b_{11} & b_{12} & b_{13} \\ 0 & b_{21} & b_{22} & b_{23} \\ 0 & b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} 1 & c_{01} & c_{02} & c_{03} \\ 0 & 1 & c_{12} & c_{13} \\ 0 & 0 & c_{22} & c_{23} \\ 0 & 0 & c_{32} & c_{33} \end{pmatrix}$$

$$\mathbf{L}_1 \equiv \frac{1}{b_{11}}\begin{pmatrix} b_{11} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -b_{21} & b_{11} & 0 \\ 0 & -b_{31} & 0 & b_{11} \end{pmatrix}$$

LU decomposition: Last two steps for 4x4 matrix

$$\mathbf{L}_2 \equiv \frac{1}{c_{22}} \begin{pmatrix} c_{22} & 0 & 0 & 0 \\ 0 & c_{22} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -c_{32} & c_{22} \end{pmatrix}, \quad \mathbf{L}_3 \equiv \frac{1}{d_{33}} \begin{pmatrix} d_{33} & 0 & 0 & 0 \\ 0 & d_{33} & 0 & 0 \\ 0 & 0 & d_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- So, we can write:

$$\mathbf{L}_3\mathbf{L}_2\mathbf{L}_1\mathbf{L}_0\mathbf{A} = \mathbf{L}_3\mathbf{L}_2\mathbf{L}_1\mathbf{L}_0\mathbf{v}$$

- Afterwards, the equation is ready for back substitution

- Mathematically identical to Gaussian elimination, but we only have to find $\mathbf{L}_0$-$\mathbf{L}_3$ once, and then we can operate on many **v**'s

Slightly different formulation of LU decomposition

- From the properties of upper triangular matrices (same holds for lower):
  - Product of two upper triangular matrices is an upper triangular matrix.
  - Inverse of an upper triangular matrix is an upper triangular matrix

- Consider the lower-diagonal matrix **L** and the upper-diagonal matrix **U**:

$$\mathbf{L} = \mathbf{L}_0^{-1}\mathbf{L}_1^{-1}\mathbf{L}_2^{-1}\mathbf{L}_3^{-1}, \quad \mathbf{U} = \mathbf{L}_3\mathbf{L}_2\mathbf{L}_1\mathbf{L}_0\mathbf{A}$$

- Then trivially: **LU** = **A**, so for **Ax** = **v**,, we can write **LUx** = **v**

Expression for L

- We can confirm that for our 4 x 4 example,

$$\mathbf{L}_0^{-1} = \begin{pmatrix} a_{00} & 0 & 0 & 0 \\ a_{10} & 1 & 0 & 0 \\ a_{20} & 0 & 1 & 0 \\ a_{30} & 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{L}_1^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & b_{11} & 0 & 0 \\ 0 & b_{21} & 1 & 0 \\ 0 & b_{31} & 0 & 1 \end{pmatrix}, \quad \mathbf{L}_2^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & c_{22} & 0 \\ 0 & 0 & c_{32} & 1 \end{pmatrix}, \quad \mathbf{L}_3^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & d_{33} \end{pmatrix}$$

- Multiplying together we get

$$\mathbf{L} = \begin{pmatrix} a_{00} & 0 & 0 & 0 \\ a_{10} & b_{11} & 0 & 0 \\ a_{20} & b_{21} & c_{22} & 0 \\ a_{30} & b_{31} & c_{32} & d_{33} \end{pmatrix}$$

Solving the equation with **L** and **U**

- Break into two steps:
  - 1. **Ly** = **v** can be solved by back substitution:

$$\begin{pmatrix} l_{00} & 0 & 0 & 0 \\ l_{10} & l_{11} & 0 & 0 \\ l_{20} & l_{21} & l_{22} & 0 \\ l_{30} & l_{31} & l_{32} & l_{33} \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{pmatrix}$$

  - 2. Now solve **Ux** = **y** by back substitution:

$$\begin{pmatrix} u_{00} & u_{01} & u_{02} & u_{03} \\ 0 & u_{11} & u_{12} & u_{13} \\ 0 & 0 & u_{22} & u_{23} \\ 0 & 0 & 0 & u_{33} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

Some comments about **LU** decomposition

- Most common method for solving simultaneous equations
- Decomposition needs to be done once, then only back substitution is needed for different **v**
- In general, still may need to pivot
  - Every time you swap rows, you have to do the same to **L**
  - Need to perform the same sequence of swaps on **v**

Today's lecture:
More on linear and nonlinear algebra

- Singular and banded matrices
- LU decomposition
- Iterative methods
- Eigensystems

Jacobi and Gauss-Seidel iterative methods

- Gaussian elimination is a **direct** method

- We can also use an **iterative** method
  - Choose an initial guess and converge to better and better guesses
  - E.g., Jacobi or Gauss Seidel, Newton methods
  - Can be much more efficient for very large systems
  - Often puts restrictions on the form of the matrix for guaranteed convergence

Jacobi iterative method

- Starting with a linear system:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ \vdots \qquad \vdots \qquad\qquad \vdots \quad &\quad \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned}$$

- Pick initial guesses $\mathbf{x}^k$, solve equation *i* for *i*th unknown to get an improved guess:

$$\begin{aligned} x_1^{k+1} &= -\frac{1}{a_{11}}(a_{12}x_2^k + a_{13}x_3^k + \cdots + a_{1n}x_n^k - b_1) \\ x_2^{k+1} &= -\frac{1}{a_{22}}(a_{21}x_1^k + a_{23}x_3^k + \cdots + a_{2n}x_n^k - b_2) \\ \vdots \qquad & \qquad \vdots \qquad\qquad\qquad \vdots \qquad\qquad \vdots \\ x_n^{k+1} &= -\frac{1}{a_{nn}}(a_{n1}x_1^k + a_{n2}x_2^k + \cdots + a_{n,n-1}x_{n-1}^k - b_n) \end{aligned}$$

Jacobi iterative method

- We can write an element-wise formula for **x**:

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

- Or:

$$\mathbf{x}^{k+1} = \mathbf{D}^{-1} \left(\mathbf{b} - (\mathbf{A} - \mathbf{D})\mathbf{x}^k \right)$$

  - Where **D** is a diagonal matrix constructed from the diagonal elements of **A**

- Convergence is guaranteed if matrix is diagonally dominant (but works in other cases):

$$a_{ii} > \sum_{j=1, j \neq i}^{N} |a_{ij}|$$

Today's lecture:
More on linear and nonlinear algebra

- Singular and banded matrices
- LU decomposition
- Iterative methods
- Eigensystems

Eigenvalues and eigenvectors

- Very common matrix problem in physics
- Mostly concerned with real symmetric matrices, or Hermitian matrices
- For a symmetric matrix **A**, an eigenvector $\mathbf{v}_i$ satisfies:

$$\mathbf{A}\mathbf{v}_i = \lambda_i \mathbf{v}_i$$

  - λ_i are the eigenvalues
- Eigenvectors are orthogonal, and we will assume they are normalized:

$$\mathbf{v}_i \cdot \mathbf{v}_j = \delta_{ij}$$

- Combining eigenvectors into matrix **V**, and eigenvalues into diagonal matrix **D**:

$$\mathbf{AV} = \mathbf{VD}$$

QR algorithm for calculating eigenvalues/eigenvectors

- We will focus on real, symmetric, square **A**
- Makes use of QR decomposition to obtain **V** and **D**
  - Same idea as LU decomposition
  - Write **A** as a product of orthogonal matrix **Q**, and upper-triangular matrix **R**
  - Any square matrix can be written that way

- 1. Break **A** down into QR decomposition: $\mathbf{A} = \mathbf{Q}_1\mathbf{R}_1$
- 2. Multiply on the left by $\mathbf{Q}_1^{\mathrm{T}}$:

$$\mathbf{Q}_1^{\mathrm{T}}\mathbf{A} = \mathbf{Q}_1^{\mathrm{T}}\mathbf{Q}_1\mathbf{R}_1 = \mathbf{R}_1$$

  - Note that since **Q** is orthogonal, $\mathbf{Q}^{\mathrm{T}}=\mathbf{Q}^{-1}$

QR decomposition

- 3. Now we define a new matrix, product of $\mathbf{Q}_1$ and $\mathbf{R}_1$ in reverse order:

$$\mathbf{A}_1 = \mathbf{R}_1 \mathbf{Q}_1$$

  - Combine with step 2 to get:

$$\mathbf{A}_1 = \mathbf{Q}_1^{\mathrm{T}} \mathbf{A} \mathbf{Q}_1$$

- 4. Repeat the process, find QR decomposition of $\mathbf{A}_1$:

$$\mathbf{A}_2 = \mathbf{R}_2 \mathbf{Q}_2 = \mathbf{Q}_2^{\mathrm{T}} \mathbf{A}_1 \mathbf{Q}_2 = \mathbf{Q}_2^{\mathrm{T}} \mathbf{Q}_1^{\mathrm{T}} \mathbf{A} \mathbf{Q}_1 \mathbf{Q}_2$$

  - And so on:

$$\begin{aligned}
\mathbf{A}_1 &= \mathbf{Q}_1^{\mathrm{T}} \mathbf{A} \mathbf{Q}_1 \\
\mathbf{A}_2 &= \mathbf{Q}_2^{\mathrm{T}} \mathbf{Q}_1^{\mathrm{T}} \mathbf{A} \mathbf{Q}_1 \mathbf{Q}_2 \\
\mathbf{A}_3 &= \mathbf{Q}_3^{\mathrm{T}} \mathbf{Q}_2^{\mathrm{T}} \mathbf{Q}_1^{\mathrm{T}} \mathbf{A} \mathbf{Q}_1 \mathbf{Q}_2 \mathbf{Q}_3 \\
&\vdots \\
\mathbf{A}_k &= (\mathbf{Q}_k^{\mathrm{T}} \ldots \mathbf{Q}_1^{\mathrm{T}}) \mathbf{A} (\mathbf{Q}_1 \ldots \mathbf{Q}_k)
\end{aligned}$$

Eigenvalues and eigenvectors from QR decomposition

- If you continue this process long enough, the matrix $\mathbf{A}_k$ will eventually become diagonal:

$$\mathbf{A}_k \simeq \mathbf{D}$$

- Continue until the off-diagonal elements are below some accuracy
- Eigenvector matrix is given by:

$$\mathbf{V} = \mathbf{Q}_1\mathbf{Q}_2\mathbf{Q}_3 \ldots \mathbf{Q}_k = \prod_{i=1}^{k} \mathbf{Q}_i$$

- **V** Orthogonal since the product of orthogonal matrices is orthogonal. Then:

$$\mathbf{D} = \mathbf{A}_k = \mathbf{V}^{\mathrm{T}}\mathbf{A}\mathbf{V}$$

- So:

$$\mathbf{A}\mathbf{V} = \mathbf{V}\mathbf{D}$$

How do we do the QR decomposition?

- Think of the matrix as a set of *N* columns:

$$\mathbf{A} = \begin{pmatrix} | & | & | & \cdots \\ \mathbf{a}_0 & \mathbf{a}_1 & \mathbf{a}_2 & \cdots \\ | & | & | & \cdots \end{pmatrix}$$

- Now define two new sets of vectors:

$$\mathbf{u}_0 = \mathbf{a}_0, \qquad \mathbf{q}_0 = \frac{\mathbf{u}_0}{|\mathbf{u}_0|}$$

$$\mathbf{u}_1 = \mathbf{a}_1 - (\mathbf{q}_0 \cdot \mathbf{a}_1)\mathbf{q}_0, \qquad \mathbf{q}_1 = \frac{\mathbf{u}_1}{|\mathbf{u}_1|}$$

$$\mathbf{u}_2 = \mathbf{a}_2 - (\mathbf{q}_0 \cdot \mathbf{a}_2)\mathbf{q}_0 - (\mathbf{q}_1 \cdot \mathbf{a}_2)\mathbf{q}_1, \qquad \mathbf{q}_2 = \frac{\mathbf{u}_2}{|\mathbf{u}_2|}$$

$$\vdots \qquad \vdots$$

(Gram-Schmidt orthogonalization!)

How do we do the QR decomposition?

- General formula for $\mathbf{u}_i$ and $\mathbf{q}_i$:

$$\mathbf{u}_i = \mathbf{a}_i - \sum_{j=0}^{i-1} (\mathbf{q}_j \cdot \mathbf{a}_i)\mathbf{q}_j, \qquad \mathbf{q}_i = \frac{\mathbf{u}_i}{|\mathbf{u}_i|}$$

- We can show that the **q** vectors are orthonormal:

$$\mathbf{q}_i \cdot \mathbf{q}_j = \delta_{ij}$$

- Now we rearrange the definitions of the vectors:

$$\begin{aligned}
\mathbf{a}_0 &= |\mathbf{u}_0|\mathbf{q}_0, \\
\mathbf{a}_1 &= |\mathbf{u}_1|\mathbf{q}_1 + (\mathbf{q}_0 \cdot \mathbf{a}_1)\mathbf{q}_0 \\
\mathbf{a}_2 &= |\mathbf{u}_2|\mathbf{q}_2 + (\mathbf{q}_0 \cdot \mathbf{a}_2)\mathbf{q}_0 + (\mathbf{q}_1 \cdot \mathbf{a}_2)\mathbf{q}_1
\end{aligned}$$

How do we do the QR decomposition?

- Finally write all the equations as a single matrix equation:

$$\mathbf{A} = \begin{pmatrix} | & | & | & \dots \\ \mathbf{a}_0 & \mathbf{a}_1 & \mathbf{a}_2 & \dots \\ | & | & | & \dots \end{pmatrix} = \begin{pmatrix} | & | & | & \dots \\ \mathbf{q}_0 & \mathbf{q}_1 & \mathbf{q}_2 & \dots \\ | & | & | & \dots \end{pmatrix} \begin{pmatrix} |\mathbf{u}_0| & \mathbf{q}_0 \cdot \mathbf{a}_1 & \mathbf{q}_0 \cdot \mathbf{a}_2 & \dots \\ 0 & |\mathbf{u}_1| & \mathbf{q}_1 \cdot \mathbf{a}_2 & \dots \\ 0 & 0 & |\mathbf{u}_2| & \dots \end{pmatrix}$$

- Our QR decomposition is thus

$$\mathbf{Q} = \begin{pmatrix} | & | & | & \dots \\ \mathbf{q}_0 & \mathbf{q}_1 & \mathbf{q}_2 & \dots \\ | & | & | & \dots \end{pmatrix}, \qquad \mathbf{R} = \begin{pmatrix} |\mathbf{u}_0| & \mathbf{q}_0 \cdot \mathbf{a}_1 & \mathbf{q}_0 \cdot \mathbf{a}_2 & \dots \\ 0 & |\mathbf{u}_1| & \mathbf{q}_1 \cdot \mathbf{a}_2 & \dots \\ 0 & 0 & |\mathbf{u}_2| & \dots \end{pmatrix}$$

- **Q** is orthogonal since the columns are orthonormal
- **R** is upper triangular

QR decomposition algorithm:

- For a give *N* x *N* starting matrix **A**:

- 1. Create an *N* x *N* array to hold **V**; initialize as identity
- 2. Calculate QR decomposition **A** = **QR**
- 3. Update **A** with new value **A** = **RQ**
- 4. Multiply **V** on the RHS with **Q**
- 5. Check off-diagonal elements of **A**. If they are less than some tolerance, we are done. Otherwise go back to 2.

Libraries for linear algebra: BLAS (basic linear algebra subroutines)

- These are the standard building blocks (API) of linear algebra on a computer (Fortran and C)
- Most linear algebra packages formulate their operations in terms of BLAS operations
- Three levels of functionality:
  - Level 1: vector operations ($\alpha\mathbf{x} + \mathbf{y}$)
  - Level 2: matrix-vector operations ($\alpha\mathbf{A}\,\mathbf{x} + \beta\,\mathbf{y}$)
  - Level 3: matrix-matrix operations ($\alpha\mathbf{A}\,\mathbf{B} + \beta\,\mathbf{C}$)
- Available on pretty much every platform (http://www.netlib.org/blas/)
  - See (https://en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms)
  - Some compilers provide specially optimized BLAS libraries (-lblas) that take great advantage of the underlying processor instructions
  - ATLAS: automatically tuned linear algebra software

Libraries for linear algebra: LAPACK

- The standard for linear algebra
- Built upon BLAS
- Routines named in the form xyyzzz
  - x refers to the data type (s/d are single/double precision floating, c/z are single/double complex)
  - yy refers to the matrix type
  - zzz refers to the algorithm (e.g. sgebrd = single precision bi-diagonal reduction of a general matrix)

- Routines: http://www.netlib.org/lapack/

Libraries for linear algebra: Python

- Basic methods in numpy.linalg (based on BLAS and LAPACK)
  - https://numpy.org/doc/stable/reference/routines.linalg.html
  - Has a matrix type built from the array class
  - * operator works element by element for arrays but does matrix product for matrices
  - As of python 3.5, @ operator will do matrix multiplication for NumPy arrays
  - Vectors are automatically converted into 1×N or N×1 matrices
  - Matrix objects cannot be > rank 2
  - Matrix has .H (or .T), .I, and .A attributes (transpose, inverse, as array)

- More general stuff in SciPy (scipy.linalg)
  - http://docs.scipy.org/doc/scipy/reference/linalg.html

After class tasks

- Homework 3 will be posted soon
- Readings:
  - Newman Ch. 6
  - Garcia Ch. 4
  - Pang Ch. 5