Numerical Linear Algebra

2017

Book:

Numerical Linear Algebra, (1997), by LLoyd N. Trefethen and David Bau, III, SIAM

# I. Fundamentals

## 1. Matrix-Vector Multiplication

**A matrix times a vector is a linear combination of columns.**

Let

**A matrix times another matrix is piecing together individual (matrix \* vector) results.**

Suppose we want to double column 1 of matrix *B*.

To double column 1:

To preserve column 2 and column 3:

Now putting it altogether:

Checking with Python:

import numpy

B = numpy.array([[1, 4, 7],

[2, 5, 8],

[3, 6, 9]])

M = numpy.array([[2, 0, 0],

[0, 1, 0],

[0, 0, 1]])

B.dot(M)

**Row operations are the transpose of the corresponding column operations.**

Suppose the goal is to halve row 3. This is the transpose of the matrix that will halve column 3.

Now apply transpose to both sides

So, to halve row 3, pre-multiply by the matrix .

**Example 1.1 Vandermonde Matrix**

This matrix encodes a polynomial. For example:

is encoded as

**Example 1.2 Outer Product**

It's called outer product because the result looks like a multiplication table, with the input acting as the rows and columns.

|  |  |  |  |
| --- | --- | --- | --- |
|  | 8 | 9 | 10 |
| 1 | 8 | 9 | 10 |
| 2 | 16 | 18 | 20 |
| 3 | 24 | 27 | 30 |

## 2. Orthogonal Vectors and Matrices

**Adjoint (conjugate transpose, Hermitian conjugate)**

Complex conjugate of a scalar z

The **Hermitian conjugate** or **adjoint** of a matrix:

**Hermitian (complex symmetric)**

If , A is **Hermitian**.

This *A* is a square matrix.

For real *A*, , and *A* is symmetric.

For a complex *A*, the matrix has real values on the diagonal, and conjugate values off the diagonal. For example: .

**Inner Product (for complex vectors)**

Note that you must not write since that would be the outer product.

The inner product is bilinear:

**Orthogonal Vector**

A pair of vectors "*x*" and "y" are orthogonal if .

**Theorem 2.1. The vectors in an orthogonal set S are linearly independent.**

Proof: start by assuming vector "vk" is a linear combination of some other vectors

Next multiply by "vk\*"to get a contradiction.

On one hand, the "vk"is not a zero vector, so .

On the other hand, the set of vectors are orthogonal, so all the , with , would be zero.

**Components of a Vector**

Let be an orthonormal set.

Let

This "r" vector is orthogonal to all vectors in the set .

If the "q" set spans "v" space, then there will be nothing left in "r".

**Discussion**

Multiply both sides by

Note thatis allowed since the inner product result in a constant. Using the associative law of matrix multiplication, followed by moving out the dot product, is a common technique for simplifying matrix products.

Note that will mostly cancel out due to the vectors being from an orthonormal set.

The .

So in the end .

Note the decomposition term can be rewritten as .

The is not a constant value, but it's the outer product and results in a matrix.

The term views the component vectors as a projection value times the .

The term views the component vectors as something that is generated by the matrix.

**Unitary (Complex Orthonormal) Matrices**

In order to produce the first row, the , but all the other inner products with need to be zero.

**Multiplication by a Unitary Matrix**

Proof:

The preservation of the dot product means the angles between "x" and "y" are preserved.

Multiplying by "Q" is like a reflection or rotation. The vectors get moved, but the angles between those vectors remain the same.

Proof: means

So vectors are rotated or reflected, but they are not stretched.

**Pythagorean Theorem (Exercise 2.2)**

Suppose the vectors are orthogonal to each other.

The reason is that the dot product will result in only terms. Terms like becomes zero because the set of vectors is orthogonal to each other.

**Eigenvalues of Hermitian (symmetric) matrices are real (Exercise 2.3)**

Conjugate transpose both sides.

Post multiply by

*A* is Hermitian, so *A*\* = *A*. The left side becomes , and .

**For Hermitian matrices, eigenvectors of distinct eigenvalues are orthogonal (Exercise 2.3)**

Let and .

, where because the eigenvalues for Hermitian matrices must be real.

So , but . Therefore .

**Eigenvalues of unitary (orthogonal) matrices lie on the unit circle (Exercise 2.4)**

This is because of . The magnitude of the eigenvalues need to be one.

Note that the unitary matrix need not be symmetric. So the eigenvalues can be complex.

## 3. Norms

**Vector Norm Properties**

1. , and only if ,
2. The triangle inequality: ,
3. .

Any function that follows these properties can count as a vector norm.

**p-norms**

|  |  |
| --- | --- |
|  | Unit Ball |
|  |  |
|  |  |
|  |  |
|  |  |

**weighted p-norms**

where W is a diagonal matrix of weights.

|  |  |
| --- | --- |
| Weighted 2-norm | Unit Box |
|  |  |

The unit box is an ellipse because the "x" can go farther in certain directions to produce a norm of 1.

**Induced Matrix Norms**

This is how far a matrix can amplify a vector.

The subscripts like (m,n) and (m) are referring to the dimension of the matrix.

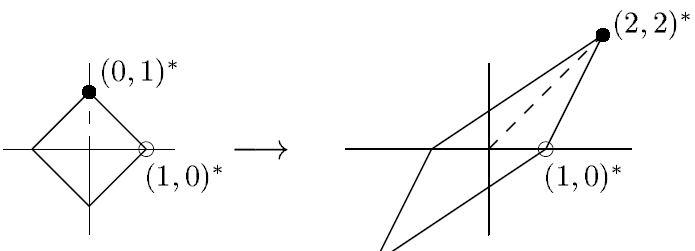
The "sup" (supremum) means maximum.

The search for the induced norm means finding the "optimal" input vector that generate the largest output vector .

The input vector search can be limited to unit vectors.

**Example 3.1** - Induced norms of the matrix .

1-norm

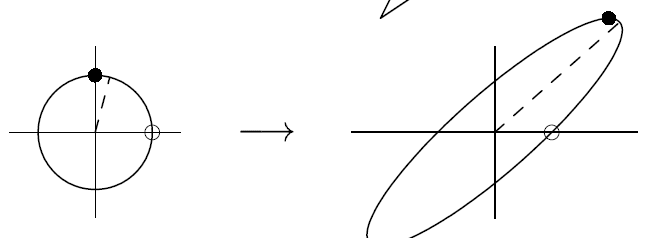


On the left is the range of that has unit length when measured using 1-norm.

On the right is the output .

The maximum amplification occurs for [0, 1], which produces [2,2].

2-norm

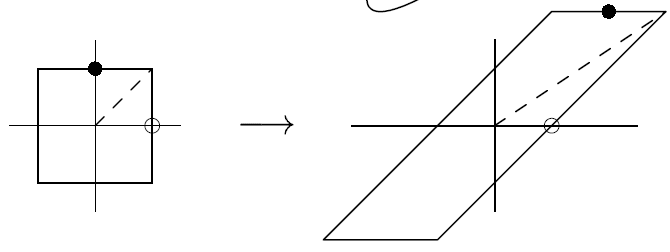


The maximum amplification is at slightly to the right of [0, 1]. The amplification at [0,1] would have been be

The ‖A‖2 is actually slightly larger than that.

This norm is computed in exercise 5.1, via computing the σ1 value of the SVD of *A*.

∞-norm:



The maximum amplification is on [1,1], which produces [3,2].

**Example 3.2 The p-Norm of a Diagonal Matrix**

The maximum happens when all of the unit vector is focused on the largest term of the diagonal.

There is just one non-zero term in the maximum vector. So it's always going to be . The *p* therefore doesn't affect the final answer.

**Example 3.3 - The 1-Norm of a Matrix**

The maximum output for a unit vector would be to concentrate on the largest column.

**Example 3.4 - The ∞-Norm of a Matrix**

The maximum unit vector for the ∞-norm is all ones. So all the columns get used.

The ∞-norm of is the maximum row sum.

where denotes the i-th row of A.

**The Holder Inequality**

**The Cauchy-Schwarz Inequality**

This is when *p = q* = 2.

**Example 3.5 - induced norm of a matrix with a single row**

Consider a matrix *A* that has just a single row, .

The induced 2-norm of matrix *A*:

The induced matrix norm is therefore consistent with the 2-norm of the row vector.

**Example 3.6 - The 2-Norm of an Outer Product**

You can factor out the since it's a constant.

It is factored out as the absolute value due to the property: .

Apply

The equality case is when the same vector is the optimizing vector for all three norms ‖AB‖, ‖A‖, ‖B‖.

The product of ‖A‖ and ‖B‖ will in general be larger because they are free to choose different optimizing vectors.

**Matrix Norm Properties**

1. , and only if ,
2. The triangle inequality: ,
3. .

Any function that follows these properties can count as a matrix norm.

**Hilbert-Schmidt or Frobenius Norm**

The norm treats the *m x n* matrix as a single vector that contains *m\*n* numbers, and takes a 2-norm of this vector.

where tr(B) denotes the trace of B, the sum of its diagonal terms.

This formula works because the diagonal terms of ATA are inner product of the column vectors with itself.

The and have the same values on the diagonal. So even though the off diagonal terms are not the same, the trace function only care about the diagonal terms.

The proof is to use the Cauchy-Schwarz Inequality to establish an upper bound and then separate this upper bound into two matrices.

Let i-th row of A

Let j-th column of B

Let a term from the AB matrix

This sum is separable. The ‖***ai***‖ exits the inner summation since that is summing over the index j. The Σ(‖***bj***‖) then exits the outer summation since that is summing over index i.

**Norm Invariance under Unitary Multiplication**

Given matrix *A* and an orthogonal (unitary) matrix *Q*:

The reason is due to the property presented in the previous lecture. The will produce some column vector . Multiplying a *Q* in front of this does not change the 2-norm.

Proof: , and

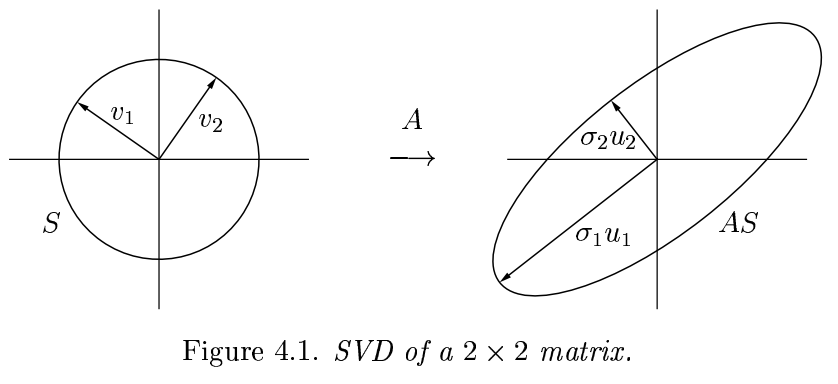
Intuitively, the multiplication does not change the norm for the column vector . The matrix *QA* is . The individual column norms remain the same, and so the overall Frobenius norm remain the same as well.

## 4. The Singular Value Decomposition

**Geometric Observation**

The image of the unit sphere under any *m x n* matrix is a hyperellipse.

2D situation:



Matrix *A* maps ***v1*** to *σ1****u1***, ***v2*** to *σ2****u2***, and the unit sphere to an ellipse.

It's a convention to list the singular values from largest to smallest, so that .

**Example (Problem 2.1 a)**

Determine the SVD of .

This is a set of parametric equations for an ellipse.

The longer axis is .

The shorter axis is .

**Example (Problem 2.1 e)**

Determine the SVD of .

This is a line. The question is at what angle is the maximum (or minimum) of the line.

Use the Manhattan distance:

Note that and will both lead to . There should be a maximum somewhere in the middle.

**Extension to 3D**

This geometric interpretation can be extended indefinitely. At 3D, the input is unit sphere. The largest output possible is associated with the σ1 and . The "left over input" is a plane that is normal to the . This plane traces out a unit circle, reducing to the 2D case.

**Reduced SVD**

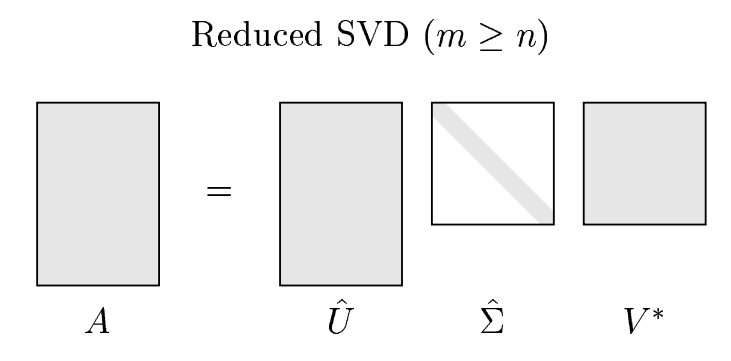
Per basis vector, the SVD looks like

This is like the eigenvalue equation, except that the "v" and "u" vectors can go in different directions. This flexibility is what enables the SVD to always exist, and the σ to be always positive.

Collecting all the basis vectors together, we have

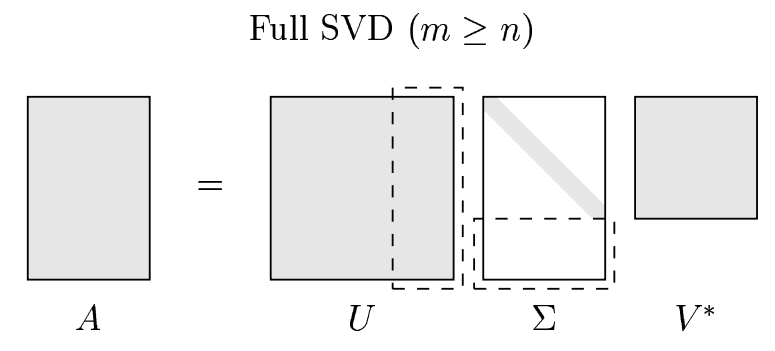
The V is a matrix with orthonormal columns, so it can be inverted as a transpose.

This is known as the reduced SVD. The Σ matrix is square, and the *U* is not necessarily square.



**Full SVD**

In the full SVD, the *U* is passed into a square orthogonal matrix by adding extra orthonormal basis column vectors. The Σ is padded with zeros.



The dashed lines marked the padding added.

If *A* is rank-deficient, the SVD factorization is still valid. The matrix *A* does not provide enough "natural" basis vectors, so arbitrary basis vectors are made up to fill up the matrices.

The full SVD has the advantage that the U is unitary (square), which makes it easier to work with in proofs.

**Transpose and multiplying by unitary matrices do not change singular values (Exercise 4.2)**

Suppose and . *U*, *V*, and *Q* are unitary.

To prove that is a valid SVD, you have to prove that and are unitary.

Since is a valid SVD, the singular values are still from the same Σ matrix. So *B* has the same singular values as *A*.

**Existence and Uniqueness**

Every matrix *A* has a singular value decomposition.

The singular values {*σ­j*} are uniquely determined.

For a distinct *σ­j*, the ***uj*** and ***vj*** are uniquely determined. You can, of course, apply a -1 to both the "**u**" and "**v**" vectors.

**Discussion**

This can be thought of as a recursive problem. The largest singular value σ1 is the norm of the matrix *A*.

Then you extend the "v" and "u" vectors into orthogonal matrices

Then create matrix *S*

The zero vector in the first column comes from the "u" vectors being from the orthogonal set. So will produce zeroes when it is multiplied with .

The actually turns out to be zero as well - see the book for more detail.

The whole argument can then be repeated for matrix *B*, that it has a such that , which then creates a unique pair of vectors , and so on.

## 5. More on the SVD

**Change of Bases**

Start with .

Every matrix *A* has an SVD.

The simplification is using the change of basis:

**Eigenvalue Decomposition (Diagonalization)**

where *Λ* is a diagonal matrix whose entries are the eigenvalues of *A*.

|  |  |
| --- | --- |
| **SVD** | **Eigenvalue Decomposition** |
| Two different bases: *U\** and *V\** | One set of basis - eigenvectors |
| Orthonormal bases | Generally not orthogonal |
| All matrices, even rectangular ones, has SVD | Not all matrices, even square ones, have eigenvalue decomposition |

**Matrix Properties via the SVD**

The SVD is helpful in understanding several matrix properties.

**Assumptions**

Matrix *A* has dimensions *m x n*.

Let *p* be the minimum of *m* and *n*. This is the size of the SVD Σ matrix diagonal.

Let *r* < *p* denote the number of nonzero singular values of *A*. When *r* < *p*, the last σ entries on the Σ matrix diagonal are zeros.

Let denote the space spanned by the vectors .

**Theorem 5.1 -** The rank of *A* is *r*, the number of nonzero singular values.

In , the Σ is the rank bottleneck. The U and V are of full rank.

rank(Σ) = *r* = rank(*A*)

So if *A* is not of full rank, some singular values will be zeros.

This theorem means that the rank of a matrix can be determined by counting the number of singular values.

**Theorem 5.2** - range(*A*) = and null(*A*) = .

Regarding range(*A*) =

The *U*Σ can only produce linear combination of only. The gets paired with a zero. Multiplication with the rightmost column of Σ looks like

Regarding null(*A*) =

We are looking for that will satisfy .

The , so we are looking for .

Consider

The submatrix is invertible. The only way to produce a zero in the first "r" terms is to start with a bunch of zeros in the vector.

The solution to is .

This theorem means the basis vector for range or null space of a matrix can be found via SVD.

**Theorem 5.3** - ‖A‖2 = σ1 and

There is a prior result at the end of the "Norms" section:

In both definitions of the norm, multiplication by orthogonal matrices do not change the norm.

The norm therefore is determined by only the Σ matrix:

This theorem means the 2-norm of a matrix can be computed by finding the maximum singular value.

**Theorem 5.4** - The nonzero singular values of A are the square roots of the nonzero eigenvalues of *A\*A* or *AA\**.

The A\*A and Σ\*Σ are similar matrices with the same eigenvalues. This is because:

Let

Let , then

The eigenvalues of the diagonal matrix Σ\*Σ are σ12, σ22, ..., σp2.

**Example (Exercise 5.1, Computation of Example 3.1)**

Find the 2-norm of .

Eigenvalue λ characteristic polynomial:

**Theorem 5.5** - If *A = A\**, then the singular values of A are absolute values of the eigenvalues of *A*.

A Hermitian (complex symmetric) matrix has real eigenvalues and orthogonal eigenvectors (this is a prior result from exercise 2.3).

So , which is almost the same as SVD. The difference is that the Λ matrix can have negative values, whereas the Σ matrix is all positive.

To "convert" the Λ matrix to SVD format, move out the "-1" values as needed and apply the same "-1" to the columns of *Q*.

For example, becomes

The newly created is called the sign matrix.

The is still unitary.

The lengths remain 1 because the "-1" value changes only the direction of an entry in , but not the magnitude of that entry.

The new column vectors still form a dot product of since:

**Theorem 5.6** -

The determinant of unitary matrix is 1 because , so . The transpose operation does not change the determinant - they have matching cofactor expansions.

**Low-Rank Approximations**

**Theorem 5.7** - *A* is the sum of rank-one matrices:

Proof:

where Σj = diag(0, ..., 0, σj, 0, ..., 0)

The UΣj is mostly zero. Only the j-th column is non zero.

The v-th partial sum

is the best approximation of a matrix *A* by matrices of lower rank.

This approximation will minimize the 2-norm and the Frobenius norm.

**Theorem 5.8**

, with the restriction that rank(B) < v

**Theorem 5.9**

, with the restriction that rank(B) < v

# II. QR Factorization and Least Squares

## 6. Projectors

**Projector**

A projector is a square matrix *P* that satisfies:

So the first application of *P* will move a vector to , but additional applications of *P* will not move the vector any further.

*P*

*P*

range(*P*)

**Complementary Projector (*I* - *P*)**

If *P* is a projector, then *I-P* is also a projector.

**range (*I-P*) = null (*P*)**

In order for to be true,

**range (*P*) = null (*I-P*)**

In order to have ,

First show that

For a to satisfy both

means

null (*I-P*) = range (*P*), so

null (*P*) = S2 = range (*I-P*)

range (*P*) = S1

We say that *P* is the projector onto *S1* along *S2*.

**Theorem 6.1** A projector *P* is orthogonal if and only if .

With an orthogonal projector, the .

Let and .

Consider building the SVD for *P*.

Let S1, the range(*P*), contain orthonormal basis vectors and S2 contain basis vector .

This approach also shows that *P=P\**.

**Constructing the projection matrix using orthonormal basis vectors**

From previous:

Drop the silent columns:

where the columns are orthonormal basis vectors.

Alternative construction procedure (same conclusion):

Project a vector onto orthonormal basis vectors .

**Note: the above formula is only for orthonormal basis.** The basis vector is a separate issue from the nature of the projector. An orthogonal projector can also be constructed using a non-orthogonal (arbitrary) basis vector. There's a different formula for that in the book.

The complement of an orthogonal projector, , is also an orthogonal projector.

**Example (Exercise 6.2) - Proving that a matrix operation is an orthogonal projector.**

R3 example:

So it seems to be a projector.

picks the last column, which is .

picks the second last column, which is .

, so *E* is a projector.

, so *E* is an orthogonal projector.

**Example (Exercise 6.4) - Constructing a projector**

The goal is to construct an orthogonal projector *P* into range(*A*).

The orthonormal basis vectors are

, which is the matrix in the previous example (exercise 6.2).

**Example (Exercise 6.5) - Norm of a projector**

**For a projector *P*, .**

Vectors that are in range(*P*) gets mapped to itself. So the norm of a projector is always at least one.

**For an orthogonal projector P,**

For any matrix *A*, , where σ1 is the largest singular value in the SVD.

For a projector matrix *P*, we can further say:

There are two (UΣV\*) SVDs in series. In general there's no way to know what is the norm - because the vector that optimizes the first (UΣV\*) SVD is not necessarily applied to the second SVD.

In other words, if optimize the norm for , in general we don't have .

However, for an orthogonal projector, the projection always results in a smaller norm. The optimizing vector is indeed .

For an orthogonal projector matrix *P*,

The two SVD collapse into a single SVD. So it's possible to say that

This is true even if the singular value is complex, since .

Putting the following two together:

, for any matrix *A*

, for orthogonal projector matrix *P*

the conclusion is that for orthogonal projector *P*, σ1 = 1.

## 7. QR Factorization

**Reduced QR Factorization**

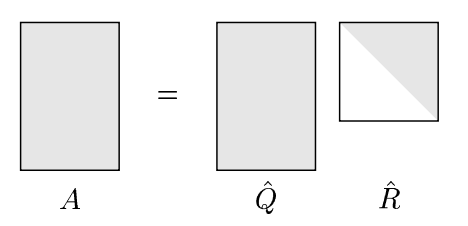
Notation: is the two-dimensional space spanned by and .

For matrix , we want to find basis vectors such that

The vectors can be found by Gram-Schmidt algorithm. The basis vector only depends on . That is why the matrix on the right-hand side can be upper triangular.

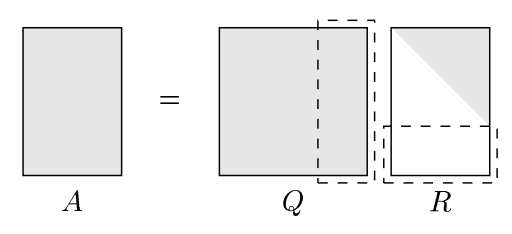
Having an algorithm to construct the basis vectors means that all matrices have QR factorizations.

The "hat" symbol means a reduced matrix.



**Full QR Factorization**

The *Q* matrix is expanded into a square matrix by padding it with orthonormal column vectors.



The newly padded columns are orthogonal to range(*A*).

## 8. Gram-Schmidt Orthogonalization

**Gram-Schmidt Projections**

Suppose the *A* column vectors are related to the basis vectors by projection matrices:

At each step of the Gram-Schmidt algorithm, the is projected onto a space defined by , and the subtracted out.

So each step of the algorithm can be described as:

The *P* matrices are given by:

**Modified Gram-Schmidt Algorithm**

Notation:

So will remove the component of that is parallel to .

The classical Gram-Schmidt is doing a single projection and then that gets normalized.

The modified Gram-Schmidt is doing a series of projections, each removing a single component away from

**Triangular Orthogonalization (Gram-Schmidt in matrix form)**

Let matrix *V* start out as .

In the first iteration, normalize and project the component away from the rest of the vectors.

In equation terms:

, where r11 is the length of

, where r12 is the length of projected onto .

In matrix terms:

The matrix for the second step we normalize and project it away from .

In general, the i-th step in the algorithm will normalize the i-th vector, and project away the component of that vector. Each step is a triangular matrix, and the modified Gram-Schmidt algorithm can be viewed as:

## 9. Python

**Experiment 1: Discrete Legendre Polynomials**

The following code approximates the functions using discrete points. It then uses QR decomposition to compute a basis for these functions.

import numpy

import numpy.linalg as linalg

import matplotlib.pyplot as pyplot

col\_0 = numpy.ones(256) # col\_0 = [1, ..., 1], 256 points total

col\_1 = numpy.linspace(-1, 1, 256) # col\_1 = [-1, ... , 1], 256 points total

col\_2 = col\_1 \*\* 2

col\_3 = col\_1 \*\* 3

A = numpy.column\_stack((col\_0, col\_1, col\_2, col\_3))

# python slice notation (start:stop:increment)

A[:10] # first 10 rows

A[-10:] # last 10 rows

Q,R = linalg.qr(A) # QR factorization

scale = Q[-1:][0] # last row of "Q"

Q = Q.dot(numpy.diag(scale \*\* -1)) # rescale Q to between -1 and 1

# plotting the columns of Q

pyplot.plot(col\_1, Q[:,0], "b-", label="P0")

pyplot.plot(col\_1, Q[:,1], "g-", label="P1")

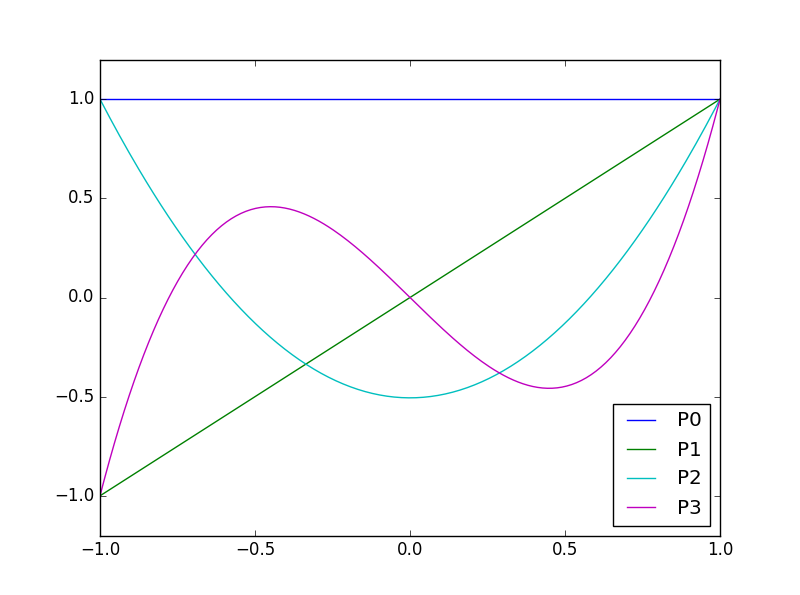
pyplot.plot(col\_1, Q[:,2], "c-", label="P2")

pyplot.plot(col\_1, Q[:,3], "m-", label="P3")

pyplot.axis([-1,1,-1.2,1.2])

pyplot.legend(loc=4)

pyplot.show()

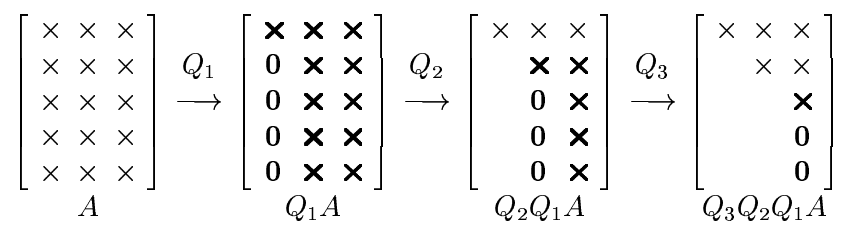


## 10. Householder Triangularization

**Orthogonal Triangularization**

The Householder method applies a succession of unitary matrices *Qk* on the left side of *A*, transforming *A* into an upper triangular matrix.

Each *Q* matrix zeroes out a column.



The matrix *Qk* needs to zero out the zeroes in the k-th column, while leaving the previous columns untouched.

In the product *QkA*, for the matrix *A* columns before the k-th column, we have:

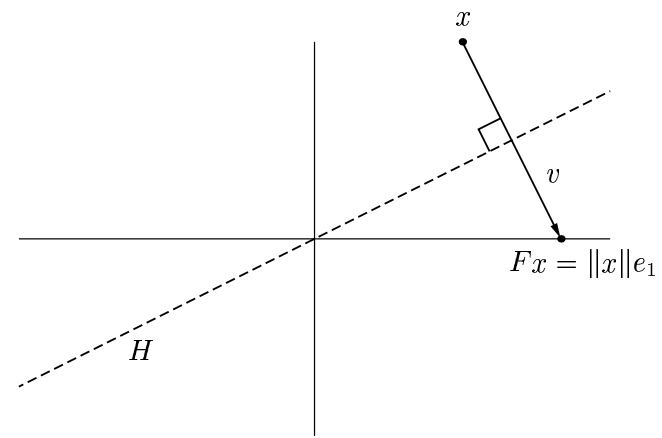
The *F* matrix will interact with zeroes, so the *Qk* will not affect columns before the

At the k-th column, we have

The top part of is unaffected. The lower part of , called , needs to interact with *Qk* to produce zeroes.

The *Qk* matrix is unitary, so the length of needs to be preserved. That's why .

Note that is another acceptable option.



The is a vector pointing from to . So .

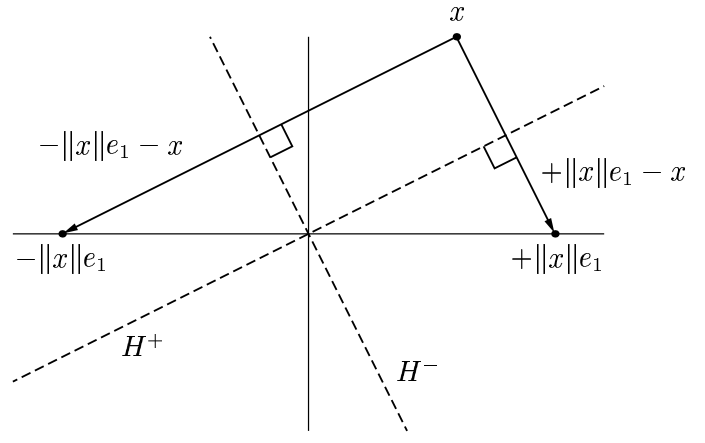
The is the normal to a hyperplane *H*. The dash line represents the edge of the hyperplane *H*.

An orthogonal projection from to the hyperplane *H* would be:

The *P* matrix above projects to the *H* plane. To reflect across the *H* plane, we need to go twice as far.

**The Better of Two Reflectors**

To zero out the column, both and will work. Pictorially, there are two possible reflections.



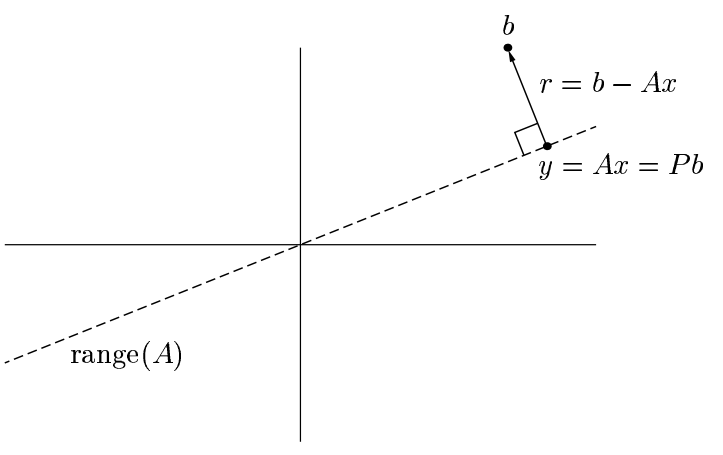
Numerically, it's better to choose the reflection that is farther away. For example, if the angle between *H+* and is very small, then reflecting to would mean a very small .

If *x1* is positive, that means is on the right-hand side, and would be the better destination.

## 11. Least Square Problems

**Orthogonal Projection and the Normal Equations**

The picture below depicts the least square solution to :



One solution method is , where *P* is the orthogonal projector that maps to range(A).

The other method is:

Let

This results in the **normal equations**: .

**Pseudoinverse**

Regarding the normal equations , if *A* has full rank, meaning the columns of *A* are linearly independent, then there will be a unique solution , given by

The matrix is known as the **pseudoinverse**.

**Three ways to solve Least Square Problems**

**#1. Normal Equations**

Solve using Cholesky factorization, which breaks into .

The *R* is an upper-triangular matrix.

**#2. QR Factorization**

Factor . This *Q* gives a way to express the orthogonal projection into range(A).

**#3. SVD**

. The gives a way to express the orthogonal projection into range(A).

**Comparison of Algorithms**

Method #1, normal equations, might be the fastest. It's not always stable though.

Method #2, QR factorization, is more stable and is that standard method.

If *A* is close to rank-deficient, then method #2 has stability issues as well. Method #3 is better in these cases.

# III. Conditioning and Stability

## 12. Conditioning and Condition Numbers

**Absolute Condition Number**

where "sup" means upper bound.

Think of it as (absolute change in output) / (absolute change in input).

**Relative Condition Number**

Think of it as (percent change in output) / (percent change in input).

**Example 12.4 - computing for *x* near 10100**

The variable *x* is so large that even very small change in *x* can lead to large change in f(x). Therefore, the condition number is very high.

**Example 12.5 - Polynomial root finding**

Polynomial root finding is potentially ill-conditioned because a small change in polynomial coefficients can change the root location dramatically.

The mathematical model for perturbing a polynomial:

The solid green line is the unperturbed polynomial and it crosses the x-axis at *xj*.

The dotted green line is the perturbed polynomial that went up by . This perturbed polynomial crosses the x-axis at a point to the right of the old zero.

Assuming the slope of the polynomial remain the same:

The condition number of *xj* being the root:

**Condition number of a root from the Wilkinson polynomial**

The Wilkinson polynomial is

It's also given by the book that the most sensitive root is , and the most sensitive coefficient is .

The question now is how to find the derivative *p'(x=15)*. There's a trick:

**Condition of Matrix-Vector Multiplication**

In , perturb but not *A*.

The exact condition number depends on both *A* and *x*.

To establish an upper bound, use:

**Condition Number of a Matrix**

Through SVD,

where σ1 is the largest singular value and σm is the smallest singular value.

The σ1/σm can be interpreted as the eccentricity of the hyperellipse.

# V. Eigenvalues

## 24. Eigenvalue Problems

**Eigenvalues and Eigenvectors**

The set of all eigenvalues is the **spectrum** of *A*, .

**Eigenvalue Decomposition (Diagonalization)**

**Geometric Multiplicity**

An eigenspace Eλ is an **invariant subspace**, meaning

The dimension of Eλ is the **geometric multiplicity** of λ.

Since:

, and the eigenvector is not allow to be zero

Eλ is also the null space of .

**Characteristic Polynomial**

From previous: , and the eigenvector is not allow to be zero

So is not invertible, and .

More generally, define a characteristic polynomial, such that

So the eigenvalues are the roots of the characteristic polynomial. For a real matrix, these roots exist in conjugate pairs.

Even if a matrix is real, its eigenvalue λ might be complex. The complex λ then force the solution to be complex, even when the matrix *A* is real.

The polynomial can be factored as

The **algebraic multiplicity** is the multiplicity of λ as a root of *pA(z)*.

**Theorem 24.2** - An matrix *A* has *m* eigenvalues, counting algebraic multiplicity.

**Similarity Transformations**

Matrices *A* and *B* are similar if .

**Theorem 24.3** - Similar matrices have the same characteristic polynomial, eigenvalues, and algebraic and geometric multiplicities.

To show that the characteristic polynomials are the same:

To see that the geometric multiplicities are the same:

If the eigenspace for *A* is *Eλ*, the *X* matrices do a unique one to one transform, so the eigenspace for B should have the same dimension.

**Theorem 24.4** - The eigenvalue algebraic multiplicity is at least as great as its geometric multiplicity.

Let a matrix *A* have an eigenvalue λ with geometric multiplicity 2.

The eigenspace is 2D, so span using an orthonormal basis set.

Construct an unitary matrix

In general,

The conclusion is that matrix *B*, similar to *A*, is guarantee to get λ with multiplicity 2.

**Defective Eigenvalues and Matrices**

Two matrices can have the same eigenvalues, but different eigenvectors.

When the eigenvalue's algebraic multiplicity exceeds its geometric multiplicity, there is not enough eigenvectors to span the space implied by the algebraic multiplicity.

Such eigenvalues are called **defective**. The matrix is also called defective, because it will not be possible to diagonalize the matrix (aka the eigenvalue decomposition) due to lack of eigenvectors.

A diagonal matrix is automatically non-defective.

**Determinant and Trace**

The trace of a matrix *A* is the sum of the terms on the diagonal.

**Theorem 24.6** - The determinant of a matrix is equal to the product of the eigenvalues, counted with algebraic multiplicity.

The trace of a matrix is equal to the sum of the eigenvalues, again counted with algebraic multiplicity.

Regarding the determinant being equal to the product of the eigenvalues:

From the characteristic polynomial section:

Evaluate both polynomials at *z=0*:

Combine the two polynomials to get

Regarding the trace being equal to the sum of the eigenvalues:

Consider a matrix *A*.

Do a cofactor expansion down the first column:

The other cofactor terms cannot product a polynomial contain *z3*. The best they can do is *z2*.

Using the same line of argument:

The *z4* and *z3* terms are all captured in:

The coefficient of the *z3* term is

Now consider the other expression for *pA(z)*:

The coefficient for *z3* is .

The conclusion is

**Unitary Diagonalization**

In this case, the Λ contains the eigenvalues and the *Q* matrix is unitary.

This factorization is very similar to the SVD. The only difference is that the SVD singular values are always positive.

**Theorem 24.7** - A Hermitian matrix is unitarily diagonalizable, and its eigenvalues are real.

A matrix *A* is **normal** if .

**Theorem 24.8** - A matrix is unitarily diagonalizable if and only if it is normal.

**Schur Factorization**

The *Q* is unitary and *T* is upper-triangular.

**Theorem 24.9** - Every square matrix *A* has a Schur factorization.

For a matrix *A*, it's guaranteed to have one eigenvalue at least, say .

Create a set of orthonormal basis using this . Let .

At this point, the first column is a single number, followed by zeroes.

Next, we need to show that this process can be extended indefinitely.

The *C* matrix is one size smaller than the *A* matrix. We need to show that assuming a Schur factorization of *C* is possible, that means a Schur factorization of *A* is possible as well.

Assume , where is unitary.

We need to extend such that the product will contain the *T* instead of *C*.

Extend as

## 25. Overview of Eigenvalue Algorithms

**Left Eigenvector**

The left eigenvector is a row vector that satisfies .

Compared with the standard (right) eigenvector , the two λ values are the same.

, and

which is the same characteristic polynomial as .

Matrix *A* and *AT* has the same eigenvalue.

This can be shown by considering the characteristic equations for *A* and *AT*. A key point is that since the two has the same cofactor expansions.

So λ is the "right eigenvalue" for A, and "left eigenvalue" for A\*, and it's known that left and right eigenvectors in fact share the same eigenvalue.

**Polynomial Root Finding as an Eigenvalue Problem**

Let

When , we have .

To restate this root finding problem as an eigenvalue problem, we want matrix *A* such that:

The matrix *A* is

It is the right most column that establish the relationship between the *z*'s. It enforces the requirement that .

The left eigenvector is used, but it's previously shown that the use of the left eigenvector does not change the eigenvalue. The *z* is the eigenvalue in this matrix equation, and also the root in the polynomial equation.

There is no formula for polynomial roots in general. That also implies there is no direct formula for finding eigenvalue sin general. Any eigenvalue solver must be iterative.

**Schur Factorization**

The *T* is upper triangular.

This means

The Schur factorization is computed iteratively, where

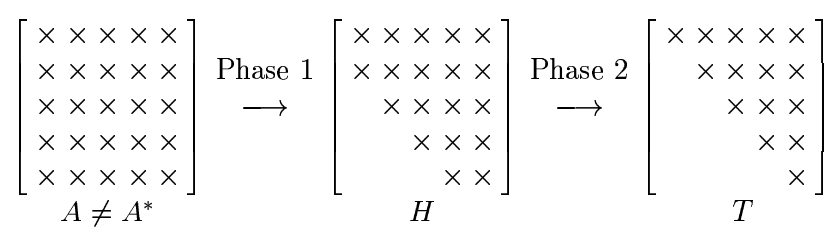
converges to an upper-triangular matrix *T* as more *Q* matrices are added.

**Upper Hessenberg Matrix**

This is a matrix that is almost triangular, with zeros below the first sub-diagonal.

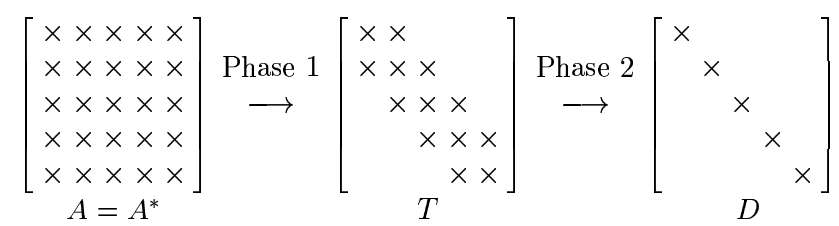
Example:

Eigenvalue solvers will convert to the upper Hessenberg form first using a direct method. Then converge to the upper triangular form.



Conversion to the *H* form first helps speed up convergence to the triangular form.

Hermitian matrices converge extra fast.



## 27. Rayleigh Quotient, Inverse Iteration

**Restriction to real symmetric matrices**

This restriction is in place for lecture #27 ~ #30.

Such matrix will have real eigenvalue sand a complete set of orthogonal eigenvectors.

**Rayleigh Quotient**

The input is an estimated eigenvector, and the output is an estimated eigenvalue.

If the input is the eigenvector exactly, then the output will be the eigenvalue exactly.

**As a least square problem solution**

Suppose there is no exact solution, since is not the eigenvector.

The question is what α is the best?

One solution is to use normal equations:

This method of estimating α will produce the formula.

**The gradient**

The gradient is . So we need a general expression for .

Drop terms that do not contain *xj*

means . The is the actual eigenvector.

**Limiting the search to**

In the equation

etc. will all give the same *r* value. Therefore, it's sufficient to search the a unit sphere for .

**Quadratic convergence (when is near the eigenvector).**

Expand using the eigenvectors as a basis:

We are restricting the search to the unit sphere. The values range from 0 to 1. When is close to the eigenvector , the coordinate will be close to 1, while the other *a* values will be close zero.

, when is close to

grows quadratically.

**Inverse Iteration**

The µ is an estimated eigenvalue. If the estimate is good, then using this matrix for power iteration will produce an estimated eigenvector .

Suppose matrix *A* has eigenvalue λ and eigenvector .

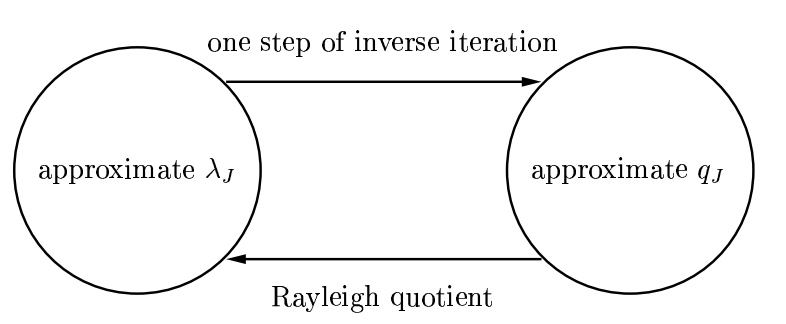
So the matrix has the same eigenvector as matrix *A*, and the eigenvalue is .

If the eigenvalue estimate µ is close to the real eigenvalue λ, the eigenvalue for will be very large. Power iteration using the matrix can quickly produce an approximate eigenvector.

The inverse iteration method is a vastly improved power iteration method because (1) it can find any eigenvector, not just the largest one, and (2) it can converge much faster.

**Rayleigh Quotient Iteration**

This is an algorithm that combines the inverse iteration with the Rayleigh quotient.



## 28. QR Algorithm Without Shifts

**"Pure" QR Algorithm**

The *Ak* will become triangular with eigenvalues on the diagonal.

**The new matrix *A* generated is similar to the old one:**

This means *Ak* and *Ak-1* both have the same eigenvalue (but not the same eigenvector).

**Unnormalized Simultaneous Iteration**

This algorithm applies power iteration to multiple vectors.

Start with a set of linearly independent vectors .

Compute for large *k*.

Extract an orthogonal basis using .

The *Qk* will converge to the eigenvectors of *A* .

**Discussion**

Suppose *V0* is just two linearly independent vectors, and .

Suppose matrix *A* has three eigenvalues, from largest to smallest: . There are also three eigenvectors .

We compute:

Express the *v* vectors in terms of eigenvectors:

For extremely large *k*, the λ1 overwhelms everything, so don't think too far down that path.

Just think of it as a large *k*. The and lean towards the most due to λ1 being the largest. They will lean toward "second most".

It's not that and converge towards and , but they converge to the space spanned by and .

After multiplying by *Ak* a sufficient number of times, the final step is to extract a set of orthonormal basis using .

Let the basis be . The QR has to be done in such a way that the is , which shows up in the product due to λ1 being the largest.

Now, knowing that , , , and , the conclusion is that .

This argument can be extended to more than two vectors to justify the

**Simultaneous Iteration**

Eventually *Qk* contains the eigenvectors.

**Discussion**

The problem with is that all the column vectors of *V* will approach , the eigenvector associated with the largest eigenvalue λ1.

The solution is to constantly orthogonalize. This is to prevent the λ1 vector from growing too fast.

Every iteration the multiplication by *A* will stretch the components of *Q* in the direction of eigenvector the most, the second most, the third most, and so on. The set of basis in *Q* will gradually become more like the set of eigenvectors.

**QR Algorithm**

The QR algorithm can be viewed as a further evolution of the simultaneous iteration algorithm.

|  |  |
| --- | --- |
| Simultaneous Iteration | QR |

As the diagram shows, the main difference is that in the QR iteration, we multiply by just R, instead of A=QR.

The R part of the transformation is the one that stretch the vectors. The Q is an orthogonal matrix, and multiplication by orthogonal matrix will preserve the length as well as the angles between the vectors.

By dropping the "Q", the length of the vectors is the same in the two algorithms, but the alignment of those vectors are different.

That's why the eigenvalues are present in the QR iteration, but the eigenvectors are temporarily "lost".

Multiplying by R has the advantage that R contain a lot of zeros, so it's less work to multiply by R. The eigenvectors can be recovered at the very end, once the eigenvalues are figured out.