hany.farid@dartmouth.edu
http://www.cs.dartmouth.edu/~farid

[*] *In progress*

0.1 Vectors

From the preface of Linear Algebra and its Applications:

> "Linear algebra is a fantastic subject. On the one hand it is clean and beautiful." – Gilbert Strang

This wonderful branch of mathematics is both beautiful and useful. It is the cornerstone upon which signal and image processing is built. This short chapter can not be a comprehensive survey of linear algebra; it is meant only as a brief introduction and review. The ideas and presentation order are modeled after Strang's highly recommended Linear Algebra and its Applications.

At the heart of linear algebra is machinery for solving *linear equations*. In the simplest case, the number of unknowns equals the number of equations. For example, here are a two equations in two unknowns:

$$\begin{aligned} 2x - y &= 1 \\ x + y &= 5. \end{aligned} \tag{1}$$

There are at least two ways in which we can think of solving these equations for x and y. The first is to consider each equation as describing a line, with the solution being at the intersection of the lines: in this case the point $(2, 3)$, Figure 0.1. This solution is termed a "row" solution because the equations are considered in isolation of one another. This is in contrast to a "column" solution in which the equations are rewritten in *vector* form:

$$\begin{pmatrix} 2 \\ 1 \end{pmatrix} x + \begin{pmatrix} -1 \\ 1 \end{pmatrix} y = \begin{pmatrix} 1 \\ 5 \end{pmatrix}. \tag{2}$$

The solution reduces to finding values for x and y that *scale* the vectors $(2, 1)$ and $(-1, 1)$ so that their *sum* is equal to the vector $(1, 5)$, Figure 0.2. Of course the solution is again $x = 2$ and $y = 3$.

These solutions generalize to higher dimensions. Here is an example with $n = 3$ unknowns and equations:

$$\begin{aligned} 2u + v + w &= 5 \\ 4u - 6v + 0w &= -2 \\ -2u + 7v + 2w &= 9. \end{aligned} \tag{3}$$

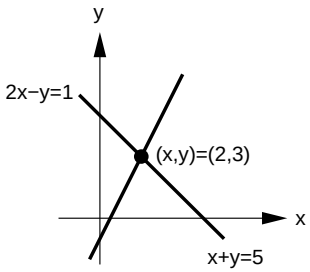


Figure 0.1 "Row" solution

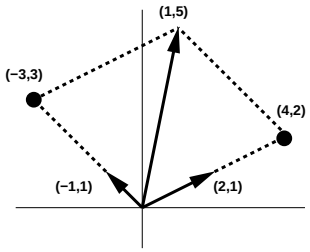


Figure 0.2 "Column" solution

Each equation now corresponds to a plane, and the row solution corresponds to the intersection of the planes (i.e., the intersection of two planes is a line, and that line intersects the third plane at a point: in this case, the point $u = 1$, $v = 1$, $w = 2$). In vector form, the equations take the form:

$$\begin{pmatrix} 2 \\ 4 \\ -2 \end{pmatrix} u + \begin{pmatrix} 1 \\ -6 \\ 7 \end{pmatrix} v + \begin{pmatrix} 1 \\ 0 \\ 2 \end{pmatrix} w \quad = \quad \begin{pmatrix} 5 \\ -2 \\ 9 \end{pmatrix}. \tag{4}$$

The solution again amounts to finding values for u, v, and w that scale the vectors on the left so that their sum is equal to the vector on the right, Figure 0.3.

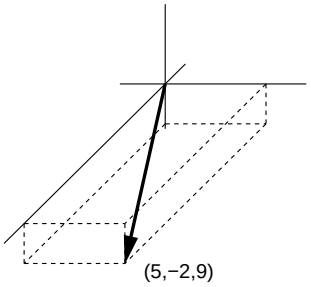


Figure 0.3 "Column" solution

In the context of solving linear equations we have introduced the notion of a *vector*, *scalar multiplication* of a vector, and *vector sum*. In its most general form, a n-dimensional *column vector* is represented as:

$$\vec{x} \quad = \quad \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \tag{5}$$

and a n-dimensional *row vector* as:

$$\vec{y} \quad = \quad (y_1 \quad y_2 \quad \dots \quad y_n). \tag{6}$$

Scalar multiplication of a vector $\vec{x}$ by a scalar value c, scales the length of the vector by an amount c (Figure 0.2) and is given by:

$$c\vec{v} \quad = \quad \begin{pmatrix} cv_1 \\ \vdots \\ cv_n \end{pmatrix}. \tag{7}$$

The *vector sum* $\vec{w} = \vec{x} + \vec{y}$ is computed via the parallelogram construction or by "stacking" the vectors head to tail (Figure 0.2) and is computed by a pairwise addition of the individual vector components:

$$\begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} \quad = \quad \begin{pmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \vdots \\ x_n + y_n \end{pmatrix}. \tag{8}$$

The *linear combination* of vectors by vector addition and scalar multiplication is one of the central ideas in linear algebra (more on this later).

In solving n linear equations in n unknowns there are three quantities to consider. For example consider again the following set of equations:

$$\begin{aligned} 2u + \ v + \ w &= 5 \\ 4u - 6v + 0w &= -2 \\ -2u + 7v + 2w &= 9. \end{aligned} \tag{9}$$

On the right is the column vector:

$$\begin{pmatrix} 5 \\ -2 \\ 9 \end{pmatrix}, \tag{10}$$

and on the left are the three unknowns that can also be written as a column vector:

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix}. \tag{11}$$

The set of nine coefficients (3 rows, 3 columns) can be written in *matrix* form:

$$\begin{pmatrix} 2 & 1 & 1 \\ 4 & -6 & 0 \\ -2 & 7 & 2 \end{pmatrix} \tag{12}$$

Matrices, like vectors, can be added and scalar multiplied. Not surprising, since we may think of a vector as a skinny matrix: a matrix with only one column. Consider the following 3×3 matrix:

$$A = \begin{pmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{pmatrix}. \tag{13}$$

The matrix cA, where c is a scalar value, is given by:

$$cA = \begin{pmatrix} ca_1 & ca_2 & ca_3 \\ ca_4 & ca_5 & ca_6 \\ ca_7 & ca_8 & ca_9 \end{pmatrix}. \tag{14}$$

And the sum of two matrices, $A = B + C$, is given by:

$$\begin{pmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{pmatrix} = \begin{pmatrix} b_1 + c_1 & b_2 + c_2 & b_3 + c_3 \\ b_4 + c_4 & b_5 + c_5 & b_6 + c_6 \\ b_7 + c_7 & b_8 + c_8 & b_9 + c_9 \end{pmatrix}. \tag{15}$$

With the vector and matrix notation we can rewrite the three equations in the more compact form of $A\vec{x} = \vec{b}$:

$$\begin{pmatrix} 2 & 1 & 1 \\ 4 & -6 & 0 \\ -2 & 7 & 2 \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} 5 \\ -2 \\ 9 \end{pmatrix}. \tag{16}$$

Where the multiplication of the matrix A with vector $\vec{x}$ must be such that the three original equations are reproduced. The first component of the product comes from "multiplying" the first row of A (a row vector) with the column vector $\vec{x}$ as follows:

$$\begin{pmatrix} 2 & 1 & 1 \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} 2u + 1v + 1w \end{pmatrix}. \tag{17}$$

This quantity is equal to 5, the first component of $\vec{b}$, and is simply the first of the three original equations. The full product is computed by multiplying each row of the matrix A with the vector $\vec{x}$ as follows:

$$\begin{pmatrix} 2 & 1 & 1 \\ 4 & -6 & 0 \\ -2 & 7 & 2 \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} 2u + 1v + 1w \\ 4u - 6v + 0w \\ -2u + 7v + 2w \end{pmatrix} = \begin{pmatrix} 5 \\ -2 \\ 9 \end{pmatrix}. \tag{18}$$

In its most general form the product of a $m \times n$ matrix with a n dimensional column vector is a m dimensional column vector whose i^{th} component is:

$$\sum_{j=1}^{n} a_{ij} x_j, \tag{19}$$

where a_{ij} is the matrix component in the i^{th} row and j^{th} column. The sum along the i^{th} row of the matrix is referred to as the *inner product* or *dot product* between the matrix row (itself a vector) and the column vector $\vec{x}$. Inner products are another central idea in linear algebra (more on this later). The computation for multiplying two matrices extends naturally from that of multiplying a matrix and a vector. Consider for example the following 3×4 and 4×2 matrices:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \\ b_{41} & b_{42} \end{pmatrix}. \tag{20}$$

The product $C = AB$ is a 3×2 matrix given by:

$$\begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} + a_{14}b_{41} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} + a_{14}b_{42} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} + a_{24}b_{41} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} + a_{24}b_{42} \\ a_{31}b_{11} + a_{32}b_{21} + a_{33}b_{31} + a_{34}b_{41} & a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32} + a_{34}b_{42} \end{pmatrix}. \tag{21}$$

That is, the i, j component of the product C is computed from an inner product of the i^{th} row of matrix A and the j^{th} column of matrix B. Notice that this definition is completely consistent with the product of a matrix and vector. In order to multiply two matrices A and B (or a matrix and a vector), the column dimension of A must equal the row dimension of B. In other words if A is of size $m \times n$, then B must be of size $n \times p$ (the product is of size $m \times p$). This constraint immediately suggests that matrix multiplication is not commutative: usually $AB \neq BA$. However matrix multiplication is both associative $(AB)C = A(BC)$ and distributive $A(B + C) = AB + AC$.

The *identity matrix* I is a special matrix with 1 on the diagonal and zero elsewhere:

$$I = \begin{pmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \dots & 0 & 1 \end{pmatrix}. \tag{22}$$

Given the definition of matrix multiplication, it is easily seen that for any vector $\vec{x}$, $I\vec{x} = \vec{x}$, and for any suitably sized matrix, $IA = A$ and $BI = B$.

In the context of solving linear equations we have introduced the notion of a *vector* and a *matrix*. The result is a compact notation for representing linear equations, $A\vec{x} = \vec{b}$. Multiplying both sides by the *matrix inverse* A^{-1} yields the desired solution to the linear equations:

$$\begin{aligned} A^{-1}A\vec{x} &= A^{-1}\vec{b} \\ I\vec{x} &= A^{-1}\vec{b} \\ \vec{x} &= A^{-1}\vec{b} \end{aligned} \tag{23}$$

A matrix A is *invertible* if there exists [1] a matrix B such that $BA = I$ and $AB = I$, where I is the identity matrix. The matrix B is the inverse of A and is denoted as A^{-1}. Note that this commutative property limits the discussion of matrix inverses to square matrices.

Not all matrices have inverses. Let's consider some simple examples. The inverse of a 1×1 matrix $A = (a)$ is $A^{-1} = (1/a)$; but the inverse does not exist when $a = 0$. The inverse of a 2×2

[1] The inverse of a matrix is unique: assume that B and C are both the inverse of matrix A, then by definition $B = B(AC) = (BA)C = C$, so that B must equal C.

matrix can be calculated as:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = \frac{1}{ad-bc}\begin{pmatrix} d & -b \\ -c & a \end{pmatrix}, \tag{24}$$

but does not exist when $ad - bc = 0$. Any diagonal matrix is invertible:

$$A = \begin{pmatrix} a_1 & & \\ & \ddots & \\ & & a_n \end{pmatrix} \quad \text{and} \quad A^{-1} = \begin{pmatrix} 1/a_1 & & \\ & \ddots & \\ & & 1/a_n \end{pmatrix}, \tag{25}$$

as long as all the diagonal components are non-zero. The inverse of a product of matrices AB is $(AB)^{-1} = B^{-1}A^{-1}$. This is easily proved using the associativity of matrix multiplication. [2] The inverse of an arbitrary matrix, if it exists, can itself be calculated by solving a collection of linear equations. Consider for example a 3×3 matrix A whose inverse we know must satisfy the constraint that $AA^{-1} = I$:

$$\begin{pmatrix} 2 & 1 & 1 \\ 4 & -6 & 0 \\ -2 & 7 & 2 \end{pmatrix}\begin{pmatrix} \vec{x}_1 & \vec{x}_2 & \vec{x}_3 \\ & & \end{pmatrix} = \begin{pmatrix} \vec{e}_1 & \vec{e}_2 & \vec{e}_3 \\ & & \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{26}$$

This matrix equation can be considered "a column at a time" yielding a system of three equations $A\vec{x_1} = \vec{e_1}$, $A\vec{x_2} = \vec{e_2}$, and $A\vec{x_3} = \vec{e_3}$. These can be solved independently for the columns of the inverse matrix, or simultaneously using the *Gauss-Jordan method*.

A system of linear equations $A\vec{x} = \vec{b}$ can be solved by simply left multiplying with the matrix inverse A^{-1} (if it exists). We must naturally wonder the fate of our solution if the matrix is not invertible. The answer to this question is explored in the next section. But before moving forward we need one last definition.

The *transpose* of a matrix A, denoted as A^t, is constructed by placing the i^{th} row of A into the i^{th} column of A^t. For example:

$$A = \begin{pmatrix} 1 & 2 & 1 \\ 4 & -6 & 0 \end{pmatrix} \quad \text{and} \quad A^t = \begin{pmatrix} 1 & 4 \\ 2 & -6 \\ 1 & 0 \end{pmatrix} \tag{27}$$

In general, the transpose of a $m \times n$ matrix is a $n \times m$ matrix with $(A^t)_{ij} = A_{ji}$. The transpose of a sum of two matrices is the sum of

[2] In order to prove $(AB)^{-1} = B^{-1}A^{-1}$, we must show $(AB)(B^{-1}A^{-1}) = I$: $(AB)(B^{-1}A^{-1}) = A(BB^{-1})A^{-1} = AIA^{-1} = AA^{-1} = I$, and that $(B^{-1}A^{-1})(AB) = I$: $(B^{-1}A^{-1})(AB) = B^{-1}(A^{-1}A)B = B^{-1}IB = B^{-1}B = I$.

the transposes: $(A+B)^t = A^t + B^t$. The transpose of a product of two matrices has the familiar form $(AB)^t = B^t A^t$. And the transpose of the inverse is the inverse of the transpose: $(A^{-1})^t = (A^t)^{-1}$. Of particular interest will be the class of *symmetric* matrices that are equal to their own transpose $A^t = A$. Symmetric matrices are necessarily square, here is a 3×3 symmetric matrix:

$$A = \begin{pmatrix} 2 & 1 & 4 \\ 1 & -6 & 0 \\ 4 & 0 & 3 \end{pmatrix}, \tag{28}$$

notice that, by definition, $a_{ij} = a_{ji}$.

0.3 Vector Spaces

The most common *vector space* is that defined over the reals, denoted as R^n. This space consists of all column vectors with n real-valued components, with rules for vector addition and scalar multiplication. A vector space has the property that the addition and multiplication of vectors always produces vectors that lie within the vector space. In addition, a vector space must satisfy the following properties, for any vectors $\vec{x}$, $\vec{y}$, $\vec{z}$, and scalar c:

1. $\vec{x} + \vec{y} = \vec{y} + \vec{x}$
2. $(\vec{x} + \vec{y}) + \vec{z} = \vec{x} + (\vec{y} + \vec{z})$
3. there exists a unique "zero" vector $\vec{0}$ such that $\vec{x} + \vec{0} = \vec{x}$
4. there exists a unique "inverse" vector $-\vec{x}$ such that $\vec{x} + (-\vec{x}) = \vec{0}$
5. $1\vec{x} = \vec{x}$
6. $(c_1 c_2)\vec{x} = c_1(c_2\vec{x})$
7. $c(\vec{x} + \vec{y}) = c\vec{x} + c\vec{y}$
8. $(c_1 + c_2)\vec{x} = c_1\vec{x} + c_2\vec{x}$

Vector spaces need not be finite dimensional, R^∞ is a vector space. Matrices can also make up a vector space. For example the space of 3×3 matrices can be thought of as R^9 (imagine stringing out the nine components of the matrix into a column vector).

A *subspace* of a vector space is a non-empty subset of vectors that is *closed* under vector addition and scalar multiplication. That is, the following constraints are satisfied: (1) the sum of any two vectors in the subspace remains in the subspace; (2) multiplication of any vector by a scalar yields a vector in the subspace. With the closure property verified, the eight properties of a vector space automatically hold for the subspace.

Example 0.1 Consider the set of all vectors in R^2 whose components are greater than or equal to zero. The sum of any two

vectors in this space remains in the space, but multiplication of, for example, the vector $\begin{pmatrix} 1 \\ 2 \end{pmatrix}$ by -1 yields the vector $\begin{pmatrix} -1 \\ -2 \end{pmatrix}$ which is no longer in the space. Therefore, this collection of vectors does not form a vector space.

Vector subspaces play a critical role in understanding systems of linear equations of the form $A\vec{x} = \vec{b}$. Consider for example the following system:

$$\begin{pmatrix} u_1 & v_1 \\ u_2 & v_2 \\ u_3 & v_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \tag{29}$$

Unlike the earlier system of equations, this system is *over-constrained*, there are more equations (three) than unknowns (two). A solution to this system exists if the vector $\vec{b}$ lies in the subspace of the columns of matrix A. To see why this is so, we rewrite the above system according to the rules of matrix multiplication yielding an equivalent form:

$$x_1 \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} + x_2 \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}. \tag{30}$$

In this form, we see that a solution exists when the scaled columns of the matrix sum to the vector $\vec{b}$. This is simply the closure property necessary for a vector subspace.

The vector subspace spanned by the columns of the matrix A is called the *column space* of A. It is said that a solution to $A\vec{x} = \vec{b}$ exists if and only if the vector $\vec{b}$ lies in the column space of A.

Example 0.2 Consider the following over-constrained system:

$$\begin{aligned} A\vec{x} &= \vec{b} \\ \begin{pmatrix} 1 & 0 \\ 5 & 4 \\ 2 & 4 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} &= \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \end{aligned}$$

The column space of A is the plane spanned by the vectors $(1 \quad 5 \quad 2)^t$ and $(0 \quad 4 \quad 4)^t$. Therefore, the solution $\vec{b}$ can not be an arbitrary vector in $\mathcal{R}^3$, but is constrained to lie in the plane spanned by these two vectors.

At this point we have seen three seemingly different classes of linear equations of the form $A\vec{x} = \vec{b}$, where the matrix A is either:

1. square and invertible (*non-singular*),

2. square but not invertible (*singular*),
3. over-constrained.

In each case solutions to the system exist if the vector $\vec{b}$ lies in the column space of the matrix A. At one extreme is the invertible $n \times n$ square matrix whose solutions may be any vector in the whole of $\mathcal{R}^n$. At the other extreme is the zero matrix $A = 0$ with only the zero vector in it's column space, and hence the only possible solution. In between are the singular and over-constrained cases, where solutions lie in a subspace of the full vector space.

The second important vector space is the *nullspace* of a matrix. The vectors that lie in the nullspace of a matrix consist of all solutions to the system $A\vec{x} = \vec{0}$. The zero vector is always in the nullspace.

Example 0.3 Consider the following system:

$$\begin{array}{rcl} A\vec{x} & = & \vec{0} \\ \begin{pmatrix} 1 & 0 & 1 \\ 5 & 4 & 9 \\ 2 & 4 & 6 \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix} & = & \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \end{array}$$

The nullspace of A contains the zero vector $(\,u \quad v \quad w\,)^t = (\,0 \quad 0 \quad 0\,)^t$. Notice also that the third column of A is the sum of the first two columns, therefore the nullspace of A also contains all vectors of the form $(\,u \quad v \quad w\,)^t = (\,c \quad c \quad -c\,)^t$ (i.e., all vectors lying on a one-dimensional line in $\mathcal{R}^3$).

0.4 Basis

Recall that if the matrix A in the system $A\vec{x} = \vec{b}$ is invertible, then left multiplying with A^{-1} yields the desired solution: $\vec{x} = A^{-1}\vec{b}$. In general it is said that a $n \times n$ matrix is invertible if it has *rank* n or is *full rank*, where the rank of a matrix is the number of *linearly independent* rows in the matrix. Formally, a set of vectors $\vec{u}_1, \vec{u}_2, ..., \vec{u}_n$ are *linearly independent* if:

$$c_1\vec{u}_1 + c_2\vec{u}_2 + ... + c_n\vec{u}_n \quad = \quad \vec{0} \tag{31}$$

is true only when $c_1 = c_2 = ... = c_n = 0$. Otherwise, the vectors are *linearly dependent*. In other words, a set of vectors are linearly dependent if at least one of the vectors can be expressed as a sum of scaled copies of the remaining vectors.

Linear independence is easy to visualize in lower-dimensional subspaces. In 2-D, two vectors are linearly dependent if they lie along a line, Figure 0.4. That is, there is a non-trivial combination of the

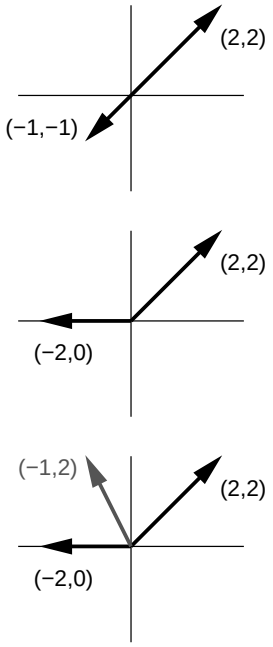


Figure 0.4 Linearly dependent (top/bottom) and independent (middle).

vectors that yields the zero vector. In 2-D, any three vectors are guaranteed to be linearly dependent. For example, in Figure 0.4, the vector $(-1 \quad 2)$ can be expressed as a sum of the remaining linearly independent vectors: $\frac{3}{2}(-2 \quad 0) + (2 \quad 2)$. In 3-D, three vectors are linearly dependent if they lie in the same plane. Also in 3-D, any four vectors are guaranteed to be linearly dependent.

Linear independence is directly related to the nullspace of a matrix. Specifically, the columns of a matrix are linearly independent (i.e., the matrix is full rank) if the matrix nullspace contains only the zero vector. For example, consider the following system of linear equations:

$$\begin{pmatrix} u_1 & v_1 & w_1 \\ u_2 & v_2 & w_2 \\ u_3 & v_3 & w_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}. \tag{32}$$

Recall that the nullspace contains all vectors $\vec{x}$ such that $x_1\vec{u} + x_2\vec{v} + x_3\vec{w} = 0$. Notice that this is also the condition for linear independence. If the only solution is the zero vector then the vectors are linearly independent and the matrix is full rank and invertible.

Linear independence is also related to the column space of a matrix. If the column space of a $n \times n$ matrix is all of $\mathcal{R}^n$, then the matrix is full rank. For example, consider the following system of linear equations:

$$\begin{pmatrix} u_1 & v_1 & w_1 \\ u_2 & v_2 & w_2 \\ u_3 & v_3 & w_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}. \tag{33}$$

If the the matrix is full rank, then the solution $\vec{b}$ can be any vector in $\mathcal{R}^3$. In such cases, the vectors $\vec{u}$, $\vec{v}$, $\vec{w}$ are said to *span* the space.

Now, a *linear basis* of a vector space is a set of linearly independent vectors that span the space. Both conditions are important. Given an n dimensional vector space with n basis vectors $\vec{v}_1, ..., \vec{v}_n$, any vector $\vec{u}$ in the space can be written as a linear combination of these n vectors:

$$\vec{u} = a_1\vec{v_1} + ... + a_n\vec{v_n}. \tag{34}$$

In addition, the linear independence guarantees that this linear combination is unique. If there is another combination such that:

$$\vec{u} = b_1\vec{v_1} + ... + b_n\vec{v_n}, \tag{35}$$

then the difference of these two representations yields

$$\begin{aligned} \vec{0} &= (a_1 - b_1)\vec{v_1} + ... + (a_n - b_n)\vec{v_n} \\ &= c_1\vec{v_1} + ... + c_n\vec{v_n} \end{aligned} \tag{36}$$

which would violate the linear independence condition. While the representation is unique, the basis is not. A vector space has infinitely many different bases. For example in $\mathcal{R}^2$ any two vectors that do not lie on a line form a basis, and in $\mathcal{R}^3$ any three vectors that do not lie in a common plane or line form a basis.

Example 0.4 The vectors $(1 \quad 0)$ and $(0 \quad 1)$ form the canonical basis for $\mathcal{R}^2$. These vectors are both linearly independent and span the entire vector space.

Example 0.5 The vectors $(1 \quad 0 \quad 0)$, $(0 \quad 1 \quad 0)$ and $(-1 \quad 0 \quad 0)$ do not form a basis for $\mathcal{R}^3$. These vectors lie in a 2-D plane and do not span the entire vector space.

Example 0.6 The vectors $(1 \quad 0 \quad 0)$, $(0 \quad -1 \quad 0)$, $(0 \quad 0 \quad 2)$, and $(1 \quad -1 \quad 0)$ do not form a basis. Although these vectors span the vector space, the fourth vector is linearly dependent on the first two. Removing the fourth vector leaves a basis for $\mathcal{R}^3$.

0.5 Inner Products and Projections

0.6 Linear Transforms

1.1 Discrete-Time Signals

A *discrete-time signal* is represented as a sequence of numbers, f, where the xth number in the sequence is denoted as $f[x]$:

$$f = \{f[x]\}, \qquad -\infty < x < \infty, \tag{1.1}$$

where x is an integer. Note that from this definition, a discrete-time signal is defined only for integer values of x. For example, the finite-length sequence shown in Figure 1.1 is represented by the following sequence of numbers

$$\begin{aligned} f &= \{\, f[1] \quad f[2] \quad \ldots \quad f[12] \,\} \\ &= \{\, 0 \quad 1 \quad 2 \quad 4 \quad 8 \quad 7 \quad 6 \quad 5 \quad 4 \quad 3 \quad 2 \quad 1 \,\}. \end{aligned} \tag{1.2}$$

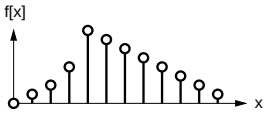


Figure 1.1
Discrete-time signal

For notational convenience, we will often drop the cumbersome notation of Equation (1.1), and refer to the entire sequence simply as $f[x]$. Discrete-time signals often arise from the periodic sampling of continuous-time (analog) signals, a process that we will cover fully in later chapters.

1.2 Discrete-Time Systems

In its most general form, a *discrete-time system* is a transformation that maps a discrete-time signal, $f[x]$, onto a unique $g[x]$, and is denoted as:

$$g[x] = T\{f[x]\} \tag{1.3}$$

Figure 1.2
Discrete-time system

Example 1.1 Consider the following system:

$$g[x] = \frac{1}{2N+1} \sum_{k=-N}^{N} f[x+k].$$

In this system, the kth number in the output sequence is computed as the average of $2N+1$ elements centered around the kth input element. As shown in Figure 1.3, with $N = 2$, the output value at $k = 5$ is computed as $1/5$ times the sum of the five input elements between the dotted lines. Subsequent output values are computed by sliding these lines to the right.

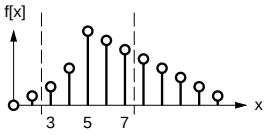


Figure 1.3 Moving Average

Although in the above example, the output at each position k depended on only a small number of input values, in general, this may not be the case, and the output may be a function of all input values.

Of particular interest to us will be a class of discrete-time systems that are both *linear* and *time-invariant*. A system is said to be linear if it obeys the rules of superposition, namely:

$$T\{af_1[x] + bf_2[x]\} \quad = \quad aT\{f_1[x]\} + bT\{f_2[x]\}, \qquad (1.4)$$

for any constants a and b. A system, $T\{\cdot\}$ that maps $f[x]$ onto $g[x]$ is shift- or time-invariant if a shift in the input causes a similar shift in the output:

$$g[x] = T\{f[x]\} \qquad \Longrightarrow \qquad g[x - x_0] = T\{f[x - x_0]\}. \qquad (1.5)$$

Example 1.2 Consider the following system:

$$g[x] \quad = \quad f[x] - f[x-1], \qquad -\infty < x < \infty.$$

In this system, the kth number in the output sequence is computed as the difference between the kth and kth-1 elements in the input sequence. Is this system **linear**? We need only show that this system obeys the principle of superposition:

$$\begin{aligned} T\{af_1[x] + bf_2[x]\} &= (af_1[x] + bf_2[x]) - (af_1[x-1] + bf_2[x-1]) \\ &= (af_1[x] - af_1[x-1]) + (bf_2[x] - bf_2[x-1]) \\ &= a(f_1[x] - f_1[x-1]) + b(f_2[x] - f_2[x-1]) \end{aligned}$$

which, according to Equation (1.4), makes $T\{\cdot\}$ linear. Is this system **time-invariant**? First, consider the shifted signal, $f_1[x] = f[x - x_0]$, then:

$$g_1[x] \;=\; f_1[x] - f_1[x-1] \;=\; f[x - x_0] - f[x - 1 - x_0],$$

and,

$$g[x - x_0] \;=\; f[x - x_0] - f[x - 1 - x_0] \;=\; g_1[x],$$

so that this system is time-invariant.

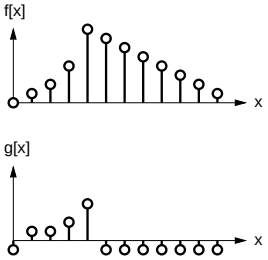


Figure 1.4 Backward difference

Example 1.3 Consider the following system:

$$g[x] \quad = \quad f[nx], \qquad -\infty < x < \infty,$$

where n is a positive integer. This system creates an output sequence by selecting every nth element of the input sequence. Is this system **linear**?

$$T\{af_1[x] + bf_2[x]\} \quad = \quad af_1[nx] + bf_2[nx]$$

which, according to Equation (1.4), makes $T\{\cdot\}$ linear. Is this system **time-invariant**? First, consider the shifted signal, $f_1[x] = f[x - x_0]$, then:

$$g_1[x] \;=\; f_1[nx] \;=\; f[nx - x_0],$$

but,

$$g[x - x_0] \;=\; f[n(x - x_0)] \;\neq\; g_1[x],$$

so that this system is not time-invariant.

The precise reason why we are particularly interested in linear time-invariant systems will become clear in subsequent chapters. But before pressing on, the concept of discrete-time systems is reformulated within a linear-algebraic framework. In order to accomplish this, it is necessary to first restrict ourselves to consider input signals of finite length. Then, any discrete-time linear system can be represented as a matrix operation of the form:

$$\vec{g} \quad = \quad M\vec{f}, \tag{1.6}$$

where, $\vec{f}$ is the input signal, $\vec{g}$ is the output signal, and the matrix M embodies the discrete-time linear system.

Example 1.4 Consider the following system:

$$g[x] \quad = \quad f[x-1], \qquad 1 < x < 8.$$

The output of this system is a shifted copy of the input signal, and can be formulated in matrix notation as:

$$\begin{pmatrix} g[1] \\ g[2] \\ g[3] \\ g[4] \\ g[5] \\ g[6] \\ g[7] \\ g[8] \end{pmatrix} \quad = \quad \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} f[1] \\ f[2] \\ f[3] \\ f[4] \\ f[5] \\ f[6] \\ f[7] \\ f[8] \end{pmatrix}$$

Note that according to the initial definition of the system, the output signal at $x = 1$ is undefined (i.e., $g[1] = f[0]$). In the above matrix formulation we have adopted a common solution to this problem by considering the signal as wrapping around itself and setting $g[1] = f[8]$.

Any system expressed in the matrix notation of Equation (1.6) is a discrete-time linear system, but not necessarily a time-invariant system. But, if we constrain ourselves to *Toeplitz* matrices, then the resulting expression will be a linear time-invariant system. A Toeplitz matrix is one in which each row contains a shifted copy of the previous row. For example, a 5×5 Toeplitz matrix is of the form

$$M \quad = \quad \begin{pmatrix} m_1 & m_2 & m_3 & m_4 & m_5 \\ m_5 & m_1 & m_2 & m_3 & m_4 \\ m_4 & m_5 & m_1 & m_2 & m_3 \\ m_3 & m_4 & m_5 & m_1 & m_2 \\ m_2 & m_3 & m_4 & m_5 & m_1 \end{pmatrix} \tag{1.7}$$

It is important to feel comfortable with this formulation because later concepts will be built upon this linear algebraic framework.

Our interest in the class of linear time-invariant systems (LTI) is motivated by the fact that these systems have a particularly convenient and elegant representation, and this representation leads us to several fundamental tools in signal and image processing.

2.1 Space: Convolution Sum

In the previous section, a discrete-time signal was represented as a sequence of numbers. More formally, this representation is in terms of the discrete-time *unit impulse* defined as:

$$\delta[x] \quad = \quad \begin{cases} 1, & x = 0 \\ 0, & x \neq 0. \end{cases} \tag{2.1}$$

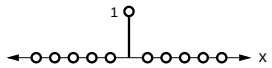


Figure 2.1 Unit impulse

Any discrete-time signal can be represented as a sum of scaled and shifted unit-impulses:

$$f[x] \quad = \quad \sum_{k=-\infty}^{\infty} f[k]\delta[x-k], \tag{2.2}$$

where the shifted impulse $\delta[x-k] = 1$ when $x = k$, and is zero elsewhere.

Example 2.1 Consider the following discrete-time signal, centered at $x = 0$.

$$f[x] \quad = \quad (\ldots \quad 0 \quad 0 \quad 2 \quad -1 \quad 4 \quad 0 \quad 0 \quad \ldots),$$

this signal can be expressed as a sum of scaled and shifted unit-impulses:

$$\begin{aligned} f[x] &= 2\delta[x+1] - 1\delta[x] + 4\delta[x-1] \\ &= f[-1]\delta[x+1] + f[0]\delta[x] + f[1]\delta[x-1] \\ &= \sum_{k=-1}^{1} f[k]\delta[x-k]. \end{aligned}$$

Let's now consider what happens when we present a linear time-invariant system with this new representation of a discrete-time signal:

$$\begin{aligned} g[x] &= T\{f[x]\} \\ &= T\left\{\sum_{k=-\infty}^{\infty} f[k]\delta[x-k]\right\}. \end{aligned} \tag{2.3}$$

By the property of linearity, Equation (1.4), the above expression may be rewritten as:

$$g[x] \quad = \quad \sum_{k=-\infty}^{\infty} f[k]T\{\delta[x-k]\}. \tag{2.4}$$

Imposing the property of time-invariance, Equation (1.5), if $h[x]$ is the response to the unit-impulse, $\delta[x]$, then the response to $\delta[x-k]$ is simply $h[x-k]$. And now, the above expression can be rewritten as:

$$g[x] \quad = \quad \sum_{k=-\infty}^{\infty} f[k]h[x-k]. \tag{2.5}$$

Consider for a moment the implications of the above equation. The unit-impulse response, $h[x] = T\{\delta[x]\}$, of a linear time-invariant system fully characterizes that system. More precisely, given the unit-impulse response, $h[x]$, the output, $g[x]$, can be determined for *any* input, $f[x]$.

The sum in Equation (2.5) is commonly called the *convolution sum* and may be expressed more compactly as:

$$g[x] \quad = \quad f[x] \star h[x]. \tag{2.6}$$

A more mathematically correct notation is $(f \star h)[x]$, but for later notational considerations, we will adopt the above notation.

Example 2.2 Consider the following finite-length unit-impulse response:

$$h[x] \quad = \quad (\,-2 \quad 4 \quad -2\,),$$

and the input signal, $f[x]$, shown in Figure 2.2. Then the output signal at, for example, $x = -2$, is computed as:

$$\begin{aligned} g[-2] \quad &= \quad \sum_{k=-3}^{-1} f[k]h[-2-k] \\ &= \quad f[-3]h[1] + f[-2]h[0] + f[-1]h[-1]. \end{aligned}$$

The next output sum at $x = -1$, is computed by "sliding" the unit-impulse response along the input signal and computing a similar sum.

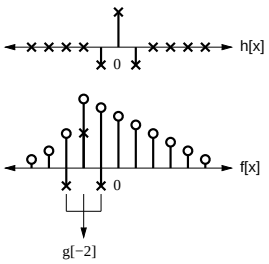


Figure 2.2 Convolution: $g[x] = f[x] \star h[x]$

Since linear time-invariant systems are fully characterized by convolution with the unit-impulse response, properties of such systems can be determined by considering properties of the convolution operator. For example, convolution is *commutative*:

$$f[x] \star h[x] \quad = \quad \sum_{k=-\infty}^{\infty} f[k]h[x-k], \qquad \text{let } j = x-k$$

$$= \sum_{j=-\infty}^{\infty} f[x-j]h[j] = \sum_{j=-\infty}^{\infty} h[j]f[x-j]$$
$$= h[x] \star f[x]. \tag{2.7}$$

Convolution also *distributes* over addition:

$$\begin{aligned} f[x] \star (h_1[x] + h_2[x]) &= \sum_{k=-\infty}^{\infty} f[k](h_1[x-k] + h_2[x-k]) \\ &= \sum_{k=-\infty}^{\infty} f[k]h_1[x-k] + f[k]h_2[x-k] \\ &= \sum_{k=-\infty}^{\infty} f[k]h_1[x-k] + \sum_{k=-\infty}^{\infty} f[k]h_2[x-k] \\ &= f[x] \star h_1[x] + f[x] \star h_2[x]. \end{aligned} \tag{2.8}$$

A final useful property of linear time-invariant systems is that a cascade of systems can be combined into a single system with impulse response equal to the convolution of the individual impulse responses. For example, for a cascade of two systems:

$$(f[x] \star h_1[x]) \star h_2[x] = f[x] \star (h_1[x] \star h_2[x]). \tag{2.9}$$

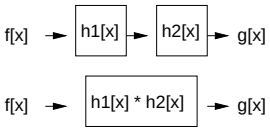


Figure 2.3 Identical LTIs

This property is fairly straight-forward to prove, and offers a good exercise in manipulating the convolution sum:

$$\begin{aligned} g_1[x] &= f[x] \star h_1[x] \\ &= \sum_{k=-\infty}^{\infty} f[k]h_1[x-k] \quad \text{and,} \end{aligned} \tag{2.10}$$

$$\begin{aligned} g_2[x] &= (f[x] \star h_1[x]) \star h_2[x] \\ &= g_1[x] \star h_2[x] \\ &= \sum_{j=-\infty}^{\infty} g_1[j]h_2[x-j] \quad \text{substituting for } g_1[x], \\ &= \sum_{j=-\infty}^{\infty} \left(\sum_{k=-\infty}^{\infty} f[k]h_1[j-k] \right) h_2[x-j] \\ &= \sum_{k=-\infty}^{\infty} f[k] \left(\sum_{j=-\infty}^{\infty} h_1[j-k]h_2[x-j] \right) \quad \text{let } i = j-k, \\ &= \sum_{k=-\infty}^{\infty} f[k] \left(\sum_{i=-\infty}^{\infty} h_1[i]h_2[x-i-k] \right) \\ &= f[x] \star (h_1[x] \star h_2[x]). \end{aligned} \tag{2.11}$$

Let's consider now how these concepts fit into the linear-algebraic framework. First, a length N signal can be thought of as a point

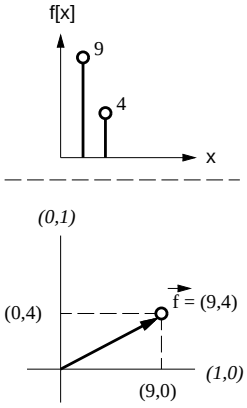


Figure 2.4
Signal and vector representation

in a $N-$dimensional vector space. As a simple example, consider the length 2 signal shown in Figure 2.4, represented as a vector in a 2-dimensional space. Earlier, the signal $f[x]$ was expressed as a sum of weighted and shifted impulses, $f[x] = 9\delta[x] + 4\delta[x-1]$, and in the vector space, it is expressed with respect to the canonical basis as $\vec{f} = 9\,(\,1 \quad 0\,) + 4\,(\,0 \quad 1\,)$. The weighting of each basis vector is determined by simply projecting the vector $\vec{f}$ onto each axis. With this vector representation, it is then natural to express the convolution sum (i.e., a linear time-invariant system) as a matrix operation. For example, let $h[x] = (\,h_{-1} \quad h_0 \quad h_1\,)$ be the finite-length unit-impulse response of a linear time-invariant system, $T\{\cdot\}$, then the system $g[x] = T\{f[x]\}$ can be expressed as $\vec{g} = M\vec{f}$, where the matrix M is of the form:

$$M \quad = \quad \begin{pmatrix} h_0 & h_{-1} & 0 & 0 & \dots & 0 & 0 & 0 & h_1 \\ h_1 & h_0 & h_{-1} & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & h_1 & h_0 & h_{-1} & \dots & 0 & 0 & 0 & 0 \\ \vdots & & & & \ddots & & & & \vdots \\ 0 & 0 & 0 & 0 & \dots & h_1 & h_0 & h_{-1} & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & h_1 & h_0 & h_{-1} \\ h_{-1} & 0 & 0 & 0 & \dots & 0 & 0 & h_1 & h_0 \end{pmatrix} \quad (2.12)$$

where each row contains a shifted and time-reversed copy of the unit-impulse response, $h[x]$. The convolution matrix can then be thought of as simply transforming the basis set. As expected, this matrix is a Toeplitz matrix of the form given earlier in Equation (1.7). The reason for the time-reversal can be seen directly from the convolution sum of Equation (2.5). More specifically, the output signal $g[x]$ at a fixed x, is determined by summing the products of $f[k]h[x-k]$ for all k. Note that the signal $h[x-k]$ can be equivalently written as $h[-k+x]$, which is a shifted (by x) and time-reversed (because of the $-k$) copy of the impulse response. Note also that when expressed in matrix form, it becomes immediately clear that the convolution sum is invertible, when h is not identically zero: $\vec{g} = M\vec{f}$ and $\vec{f} = M^{-1}\vec{g}$.

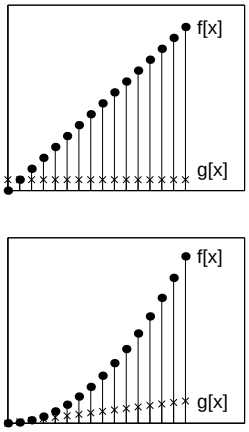


Figure 2.5 $g[x] = f[x] \star h[x]$

Before pressing on, let's try to combine the main ideas seen so far into a single example. We will begin by defining a simple discrete-time system, show that it is both linear and time-invariant, and compute its unit-impulse response

Example 2.3 Define the discrete-time system, $T\{\cdot\}$ as:

$$g[x] \quad = \quad f[x+1] - f[x-1].$$

This system is **linear** because it obeys the rule of superposition:

$$\begin{aligned} T\{af_1[x] + bf_2[x]\} &= (af_1[x+1] + bf_2[x+1]) - (af_1[x-1] + bf_2[x-1]) \\ &= (af_1[x+1] - af_1[x-1]) + (bf_2[x+1] - bf_2[x-1]) \\ &= a(f_1[x+1] - f_1[x-1]) + b(f_2[x+1] - f_2[x-1]) \end{aligned}$$

This system is also **time-invariant** because a shift in the input, $f_1[x] = f[x - x_0]$, leads to a shift in the output:

$$\begin{aligned} g_1[x] &= f_1[x+1] - f_1[x-1] \\ &= f[x+1-x_0] - f[x-1-x_0] \quad \text{and,} \\ g[x-x_0] &= f[x+1-x_0] - f[x-1-x_0] \\ &= g_1[x]. \end{aligned}$$

The **unit-impulse** response is given by:

$$\begin{aligned} h[x] &= T\{\delta[x]\} \\ &= \delta[x+1] - \delta[x-1] \\ &= (\ldots \quad 0 \quad 1 \quad 0 \quad -1 \quad 0 \quad \ldots). \end{aligned}$$

So, convolving the finite-length impulse response $h[x] = (\,1 \quad 0 \quad -1\,)$ with any input signal, $f[x]$, gives the output of the linear time-invariant system, $g[x] = T\{f[x]\}$:

$$g[x] = \sum_{k=-\infty}^{\infty} f[k]h[x-k] = \sum_{k=x-1}^{x+1} f[k]h[x-k].$$

And, in matrix form, this linear time-invariant system is given by $\vec{g} = M\vec{f}$, where:

$$M = \begin{pmatrix} 0 & 1 & 0 & 0 & \ldots & 0 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & \ldots & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & \ldots & 0 & 0 & 0 & 0 \\ \vdots & & & & \ddots & & & & \vdots \\ 0 & 0 & 0 & 0 & \ldots & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \ldots & 0 & -1 & 0 & 1 \\ 1 & 0 & 0 & 0 & \ldots & 0 & 0 & -1 & 0 \end{pmatrix}.$$

2.2 Frequency: Fourier Transform

In the previous section the expression of a discrete-time signal as a sum of scaled and shifted unit-impulses led us to the convolution sum. In a similar manner, we will see shortly how expressing a signal in terms of sinusoids will lead us to the Fourier transform, and then to a new way to think about discrete-time systems. The basic signal used as the building block is the sinusoid:

$$A\cos[\omega x + \phi], \qquad -\infty < x < \infty, \tag{2.13}$$

where A is the amplitude, ω is the frequency, and ϕ is the phase of the sinusoid. Shown in Figure 2.6, from top to bottom, are $\cos[x]$, $\cos[2x]$, and $\cos[x + \pi/2]$. Consider next the following, seemingly unrelated, *complex exponential* $e^{i\omega x}$ with frequency ω, and i the complex value $\sqrt{-1}$. This function is simply a compact notation for describing a sum of the sine and cosine function:

$$Ae^{i\omega x} = A\cos(\omega x) + iA\sin(\omega x). \tag{2.14}$$

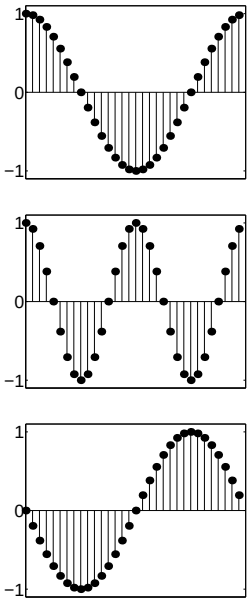


Figure 2.6
$f[x] = A\cos[\omega x + \phi]$

The complex exponential has a special relationship with linear time-invariant systems - the output of a linear time-invariant system with unit-impulse response $h[x]$ and a complex exponential as input is:

$$\begin{aligned} g[x] &= e^{i\omega x} \star h[x] \\ &= \sum_{k=-\infty}^{\infty} h[k] e^{i\omega(x-k)} \\ &= e^{i\omega x} \sum_{k=-\infty}^{\infty} h[k] e^{-i\omega k} \end{aligned} \tag{2.15}$$

Defining $H[\omega]$ to be the summation component, $g[x]$ can be expressed as:

$$g[x] = H[\omega] e^{i\omega x}, \tag{2.16}$$

that is, given a complex exponential as input, the output of a linear time-invariant system is again a complex exponential of the same frequency scaled by some amount. [3] The scaling of the complex exponential, $H[w]$, is called the *frequency response* and is generally a complex-valued function expressed in terms of its real and imaginary components:

$$H[\omega] = H_R[\omega] + iH_I[\omega], \tag{2.17}$$

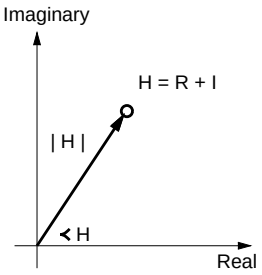


Figure 2.7 Magnitude and phase

or more commonly in terms of the *magnitude* and *phase*:

$$|H[\omega]| = \sqrt{H_R[\omega]^2 + H_I[\omega]^2} \quad \text{and} \quad \prec H[\omega] = \tan^{-1}\left(\frac{H_I[(\omega]}{H_R[\omega]}\right).$$

Example 2.4 Consider the following linear time-invariant system, $T\{\cdot\}$:

$$g[x] = f[x - x_0].$$

This system outputs a time-delayed copy of the input signal. The frequency response of this system can be determined by considering the input signal $f[x] = e^{i\omega x}$:

$$\begin{aligned} g[x] &= e^{i\omega(x-x_0)} \\ &= e^{-i\omega x_0} e^{i\omega x}, \end{aligned}$$

which is of the same form as Equation (2.16), with frequency response $H[\omega] = e^{-i\omega x_0}$. Decomposing this response in terms of the real and imaginary components gives:

$$H_R[\omega] = \cos[\omega x_0] \quad \text{and} \quad H_I[\omega] = -\sin[\omega x_0],$$

[3] In linear algebraic terms, the complex exponentials are said to be the eigenfunctions of LTIs, and $H[\omega]$ the eigenvalue.

or in terms of the magnitude and phase:

$$\begin{aligned} |H[\omega]| &= \sqrt{\cos^2[\omega x_0] + -\sin^2[\omega x_0]} \\ &= 1 \end{aligned}$$

$$\begin{aligned} \prec H[\omega] &= \tan^{-1}\left(\frac{-\sin[\omega x_0]}{\cos[\omega x_0]}\right) \\ &= -\omega x_0. \end{aligned}$$

Intuitively, this should make perfect sense. This system simply takes an input signal and outputs a delayed copy, therefore, there is no change in the magnitude of each sinusoid, while there is a phase shift proportional to the delay, x_0.

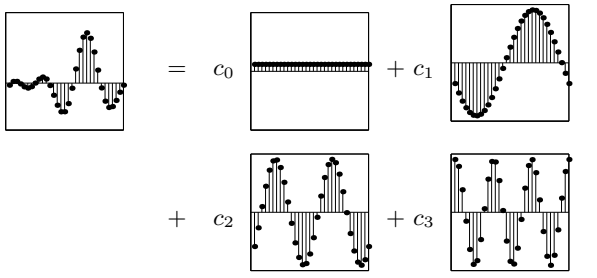
So, why the interest in sinusoids and complex exponentials? As we will show next, a broad class of signals can be expressed as a linear combination of complex exponentials, and analogous to the impulse response, the frequency response completely characterizes the system.

Let's begin by taking a step back to the more familiar sinusoids, and then work our way to the complex exponentials. Any *periodic* discrete-time signal, $f[x]$, can be expressed as a sum of scaled, phase-shifted sinusoids of varying frequencies:

$$f[x] = \frac{1}{2\pi}\sum_{k=-\pi}^{\pi} c_k \cos\left[kx + \phi_k\right] \quad -\infty < x < \infty, \quad (2.18)$$

For each frequency, k, the amplitude is a real number, $c_k \in \mathcal{R}$, and the phase, $\phi_k \in [0, 2\pi]$. This expression is commonly referred to as the *Fourier series*.

Example 2.5 Shown below is a signal, $f[x]$ (left) represented as a sum of the first four sinusoids: $f[x] = c_0 \cos[0x + \phi_0] + ... + c_3 \cos[3x + \phi_3]$.



In the language of linear algebra, the sinusoids are said to form a *basis* for the set of periodic signals, that is, any periodic signal can be written as a linear combination of the sinusoids. Recall

that in deriving the convolution sum, the basis consisted of shifted copies of the unit-impulse. But note now that this new basis is not fixed because of the phase term, ϕ_k. It is, however, possible to rewrite the Fourier series with respect to a fixed basis of zero-phase sinusoids. With the trigonometric identity $\cos(A+B) = \cos(A)\cos(B) - \sin(A)\sin(B)$, the Fourier series of Equation (2.18) may be rewritten as:

$$\begin{aligned} f[x] &= \frac{1}{2\pi}\sum_{k=-\pi}^{\pi} c_k \cos[kx + \phi_k] \\ &= \frac{1}{2\pi}\sum_{k=-\pi}^{\pi} c_k \cos[\phi_k]\cos[kx] + c_k \sin[\phi_k]\sin[kx] \\ &= \frac{1}{2\pi}\sum_{k=-\pi}^{\pi} a_k \cos[kx] + b_k \sin[kx] \end{aligned} \tag{2.19}$$

In this expression, the constants a_k and b_k are the *Fourier coefficients* and are determined by the *Fourier transform*. In other words, the Fourier transform simply provides a means for expressing a signal in terms of the sinusoids. The Fourier coefficients are given by:

$$a_k = \sum_{j=-\infty}^{\infty} f[j]\cos[kj] \quad \text{and} \quad b_k = \sum_{j=-\infty}^{\infty} f[j]\sin[kj] \tag{2.20}$$

Notice that the Fourier coefficients are determined by *projecting* the signal onto each of the sinusoidal basis. That is, consider both the signal $f[x]$ and each of the sinusoids as T-dimensional vectors, $\vec{f}$ and $\vec{b}$, respectively. Then, the projection of $\vec{f}$ onto $\vec{b}$ is:

$$f_0 b_0 + f_1 b_1 + ... = \sum_j f_j b_j, \tag{2.21}$$

where the subscript denotes the jth entry of the vector.

Often, a more compact notation is used to represent the Fourier series and Fourier transform which exploits the complex exponential and its relationship to the sinusoids:

$$e^{i\omega x} = \cos(\omega x) + i\sin(\omega x), \tag{2.22}$$

where i is the complex value $\sqrt{-1}$. Under the complex exponential notation, the Fourier series and transform take the form:

$$f[x] = \frac{1}{2\pi}\sum_{k=-\pi}^{\pi} c_k e^{ikx} \quad \text{and} \quad c_k = \sum_{j=-\infty}^{\infty} f[j]e^{-ikj}, \tag{2.23}$$

where $c_k = a_k - ib_k$. This notation simply bundles the sine and cosine terms into a single expression. A more common, but equivalent, notation is:

$$f[x] = \frac{1}{2\pi} \sum_{\omega=-\pi}^{\pi} F[\omega] e^{i\omega x} \quad \text{and} \quad F[\omega] = \sum_{k=-\infty}^{\infty} f[k] e^{-i\omega k}. \tag{2.24}$$

Comparing the Fourier transform (Equation (2.24)) with the frequency response (Equation (2.16)) we see now that the frequency response of a linear time-invariant system is simply the Fourier transform of the unit-impulse response:

$$H[\omega] \quad = \quad \sum_{k=-\infty}^{\infty} h[k] e^{-i\omega k}. \tag{2.25}$$

In other words, linear time-invariant systems are completely characterized by their impulse response, $h[x]$, and, equivalently, by their frequency response, $H[\omega]$, Figure 2.8.

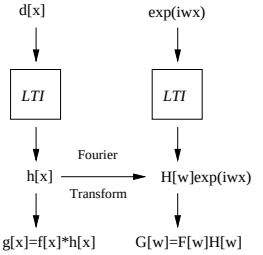


Figure 2.8 LTI: space and frequency

Example 2.6 Consider the following linear time-invariant system, $T\{\cdot\}$:

$$g[x] \quad = \quad \frac{1}{4} f[x-1] + \frac{1}{2} f[x] + \frac{1}{4} f[x+1].$$

The output of this system at each x, is a weighted average of the input signal centered at x. First, let's compute the **impulse response**:

$$\begin{aligned} h[x] &= \frac{1}{4}\delta[x-1] + \frac{1}{2}\delta[x] + \frac{1}{4}\delta[x+1] \\ &= (\ldots \quad 0 \quad 0 \quad \tfrac{1}{4} \quad \tfrac{1}{2} \quad \tfrac{1}{4} \quad 0 \quad 0 \quad \ldots). \end{aligned}$$

Then, the **frequency response** is the Fourier transform of this impulse response:

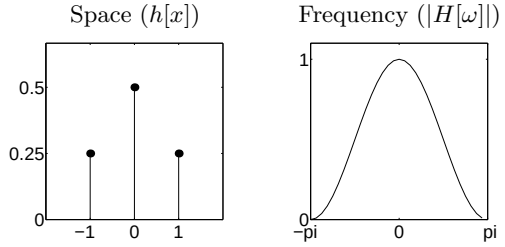
$$\begin{aligned} H[\omega] &= \sum_{k=-\infty}^{\infty} h[k] e^{-i\omega k} \\ &= \sum_{k=-1}^{1} h[k] e^{-i\omega k} \\ &= \frac{1}{4} e^{i\omega} + \frac{1}{2} e^{0} + \frac{1}{4} e^{-i\omega} \\ &= \frac{1}{4}(\cos(\omega) + i\sin(\omega)) + \frac{1}{2} + \frac{1}{4}(\cos(\omega) - i\sin(\omega)) \\ &= \mid \frac{1}{2} + \frac{1}{2}\cos(\omega) \mid. \end{aligned}$$

In this example, the frequency response is strictly real (i.e., $H_I[\omega] = 0$) and the magnitude and phase are:

$$\begin{aligned} |H[\omega]| &= \sqrt{H_R[\omega]^2 + H_I[\omega]^2} \\ &= \frac{1}{2} + \frac{1}{2}\cos(\omega) \end{aligned}$$

$$\begin{aligned} \prec H[\omega] &= \tan^{-1}\left(\frac{H_I[\omega]}{H_r[\omega]}\right) \\ &= 0 \end{aligned}$$

Both the impulse response and the magnitude of the frequency response are shown below, where for clarity, the frequency response was drawn as a continuous function.



Example 2.7 Consider the following linear time-invariant system, $T\{\cdot\}$:

$$g[x] \quad = \quad \frac{1}{2}f[x+1] - \frac{1}{2}f[x-1].$$

The output of this system at each x, is the difference between neighboring input values. The **impulse response** of this system is:

$$\begin{aligned} h[x] &= \frac{1}{2}\delta[x+1] - \frac{1}{2}\delta[x-1] \\ &= (\ldots \quad 0 \quad 0 \quad \tfrac{1}{2} \quad 0 \quad -\tfrac{1}{2} \quad 0 \quad 0 \quad \ldots). \end{aligned}$$

Then, the **frequency response** is the Fourier transform of this impulse response:

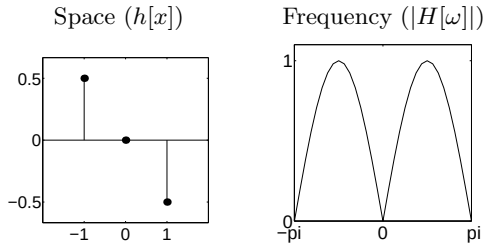
$$\begin{aligned} H[\omega] &= \sum_{k=-\infty}^{\infty} h[k]e^{-i\omega k} \\ &= \sum_{k=-1}^{1} h[k]e^{-i\omega k} \\ &= \frac{1}{2}e^{i\omega} + 0e^{0} - \frac{1}{2}e^{-i\omega} \\ &= \frac{1}{2}(\cos(\omega) + i\sin(\omega)) - \frac{1}{2}(\cos(\omega) - i\sin(\omega)) \\ &= i\sin(\omega). \end{aligned}$$

In this example, the frequency response is strictly imaginary (i.e., $H_R[\omega] = 0$) because the impulse response is anti-symmetric, and the magnitude and phase are:

$$\begin{aligned} |H[\omega]| &= \sqrt{H_R[\omega]^2 + H_I[\omega]^2} \\ &= |\ \sin(\omega)\ | \end{aligned}$$

$$\begin{aligned} \prec H[\omega] &= \tan^{-1}\left(\frac{H_I[\omega]}{H_r[\omega]}\right) \\ &= \frac{\pi}{2}. \end{aligned}$$

Both the impulse response and the magnitude of the frequency response are shown below, where for clarity, the frequency response was drawn as a continuous function.



This system is an (approximate) differentiator, and can be seen from the definition of differentiation:

$$\frac{df(x)}{dx} = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon) - f(x-\varepsilon)}{\varepsilon},$$

where, in the case of the system $T\{\cdot\}$, ε is given by the distance between samples of the discrete-time signal $f[x]$. Let's see now if we can better understand the frequency response of this system, recall that the magnitude was given by $|\sin(\omega)|$ and the phase by $\frac{\pi}{2}$. Consider now the derivative of a fixed frequency sinusoid $\sin(\omega x)$, differentiating with respect to x gives $\omega \cos(\omega x) = \omega \sin(\omega x - \pi/2)$. Note that differentiation causes a phase shift of $\pi/2$ and a scaling by the frequency of the sinusoid. Notice that this is roughly in-line with the Fourier transform, the difference being that the amplitude is given by $|\sin(\omega)|$ instead of ω. Note though that for small ω, $|\sin(\omega)| \approx \omega$. This discrepancy makes the system only an approximate, not a perfect, differentiator.

Linear time-invariant systems can be fully characterized by their impulse, $h[x]$, or frequency responses, $H[\omega]$, both of which may be used to determine the output of the system to *any* input signal, $f[x]$:

$$g[x] = f[x] \star h[x] \quad \text{and} \quad G[\omega] = F[\omega]H[\omega], \tag{2.26}$$

where the output signal $g[x]$ can be determined from its Fourier transform $G[\omega]$, by simply applying the inverse Fourier transform. This equivalence illustrates an important relationship between the space and frequency domains. Namely, convolution in the space domain is equivalent to multiplication in the frequency domain. This is fairly straight-forward to prove:

$$\begin{aligned}
g[x] &= f[x] \star h[x] \qquad \text{Fourier transforming,} \\
\sum_{k=-\infty}^{\infty} g[k] e^{-i\omega k} &= \sum_{k=-\infty}^{\infty} (f[k] \star h[k]) e^{-i\omega k} \\
G[\omega] &= \sum_{k=-\infty}^{\infty} \left(\sum_{j=-\infty}^{\infty} f[j] h[k-j] \right) e^{-i\omega k}
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{j=-\infty}^{\infty} f[j] \sum_{k=-\infty}^{\infty} h[k-j]e^{-i\omega k} \qquad \text{let } l = k-j, \\
&= \sum_{j=-\infty}^{\infty} f[j] \sum_{l=-\infty}^{\infty} h[l]e^{-i\omega(l+j)} \\
&= \sum_{j=-\infty}^{\infty} f[j]e^{-i\omega j} \sum_{l=-\infty}^{\infty} h[l]e^{-i\omega l} \\
&= F[\omega]H[\omega]. \qquad (2.27)
\end{aligned}
$$

Like the convolution sum, the Fourier transform can be formulated as a matrix operation:

$$
\begin{pmatrix} F[0] \\ F[\omega] \\ F[2\omega] \\ \vdots \\ F[T\omega] \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ e^{-0i} & e^{-i} & e^{-2i} & \dots & e^{-Ti} \\ e^{-0i} & e^{-2i} & e^{-4i} & \dots & e^{-2Ti} \\ \vdots & & & \ddots & \vdots \\ e^{-0i} & e^{-Ti} & e^{-2Ti} & \dots & e^{-T^2i} \end{pmatrix} \begin{pmatrix} f[0] \\ f[1] \\ f[2] \\ \vdots \\ f[T] \end{pmatrix}
$$

$$
\vec{F} = M\vec{f}. \qquad (2.28)
$$

Notice that this equation embodies both the Fourier transform and the Fourier series of Equation (2.24). The above form is the Fourier transform, and the Fourier series is gotten by left-multiplying with the inverse of the matrix, $M^{-1}\vec{F} = \vec{f}$.

It is often more convenient to process a *continuous-time* signal with a *discrete-time* system. Such a system may consist of three distinct stages: (1) the conversion of a continuous-time signal to a discrete-time signal (C/D converter); (2) the processing through a discrete-time system; and (3) the conversion of the output discrete-time signal back to a continuous-time signal (D/C converter). Earlier we focused on the discrete-time processing, and now we will concentrate on the conversions between discrete- and continuous-time signals. Of particular interest is the somewhat remarkable fact that under certain conditions, a continuous-time signal can be fully represented by a discrete-time signal!

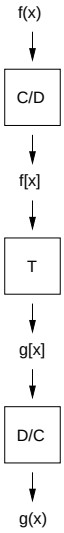


Figure 3.1 Processing block diagram

3.1 Continuous to Discrete: Space

A discrete-time signal, $f[x]$, is formed from a continuous-time signal, $f(x)$, by the following relationship:

$$f[x] \quad = \quad f(xT) \qquad -\infty < x < \infty, \tag{3.1}$$

for integer values x. In this expression, the quantity T is the *sampling period*. In general, continuous-time signals will be denoted with rounded parenthesis (e.g., $f(\cdot)$), and discrete-time signals with square parenthesis (e.g., $f[\cdot]$). This sampling operation may be considered as a multiplication of the continuous time signal with an *impulse train*, Figure 3.2. The impulse train is defined as:

$$s(x) \quad = \quad \sum_{k=-\infty}^{\infty} \delta(x - kT), \tag{3.2}$$

where $\delta(\cdot)$ is the unit-impulse, and T is the sampling period - note that the impulse train is a continuous-time signal. Multiplying the impulse train with a continuous-time signal gives a sampled signal:

$$f_s(x) \quad = \quad f(x)s(x), \tag{3.3}$$

Note that the sampled signal, $f_s(x)$, is indexed on the *continuous* variable x, while the final discrete-time signal, $f[x]$ is indexed on the *integer* variable x. It will prove to be mathematically convenient to work with this intermediate sampled signal, $f_s(x)$.

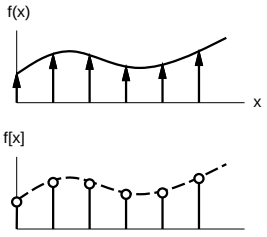


Figure 3.2 Sampling: space

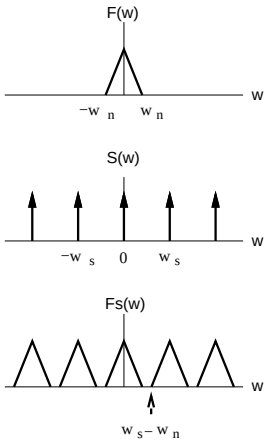


Figure 3.3 Sampling: no aliasing

In the space domain, sampling was described as a product between the impulse train and the continuous-time signal (Equation (3.3)). In the frequency domain, this operation amounts to a convolution between the Fourier transform of these two signals:

$$F_s(\omega) \quad = \quad F(\omega) \star S(\omega) \tag{3.4}$$

For example, shown in Figure 3.3 (from top to bottom) are the Fourier transforms of the continuous-time function, $F(\omega)$, the impulse train, $S(\omega)$, itself an impulse train, and the results of convolving these two signals, $F_s(\omega)$. Notice that the Fourier transform of the *sampled* signal contains multiple (yet exact) copies of the Fourier transform of the original *continuous* signal. Note however the conditions under which an exact replica is preserved depends on the maximum frequency response ω_n of the original continuous-time signal, and the sampling interval of the impulse train, ω_s which, not surprisingly, is related to the sampling period T as $\omega_s = 2\pi/T$. More precisely, the copies of the frequency response will not overlap if:

$$\begin{aligned} \omega_n &< \omega_s - \omega_n \qquad \text{or} \\ \omega_s &> 2\omega_n, \end{aligned} \tag{3.5}$$

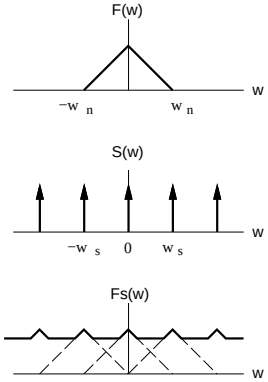


Figure 3.4 Sampling: aliasing

The frequency ω_n is called the *Nyquist frequency* and $2\omega_n$ is called the *Nyquist rate*. Shown in Figure 3.4 is another example of this sampling process in the frequency domain, but this time, the Nyquist rate is not met, and the copies of the frequency response overlap. In such a case, the signal is said to be *aliased*.

Not surprisingly, the Nyquist rate depends on both the characteristics of the continuous-time signal, and the sampling rate. More precisely, as the maximum frequency, ω_n, of the continuous-time signal increases, the sampling period, T must be made smaller (i.e., denser sampling), which in turn increases ω_s, preventing overlap of the frequency responses. In other words, a signal that changes slowly and smoothly can be sampled fairly coarsely, while a signal that changes quickly requires more dense sampling.

3.3 Discrete to Continuous

If the Nyquist rate is met, then a discrete-time signal fully characterizes the continuous-time signal from which it was sampled. On the other hand, if the Nyquist rate is not met, then the sampling leads to aliasing, and the discrete-time signal does not accurately represent its continuous-time counterpart. In the former case, it

is possible to reconstruct the original continuous-time signal, from the discrete-time signal. In particular since the frequency response of the discrete-time signal contains exact copies of the original continuous-time signals frequency response, we need only extract one of these copies, and inverse transform the result. The result will be identical to the original signal. In order to extract a single copy, the Fourier transform of the sampled signal is multiplied by an *ideal reconstruction filter* as shown in Figure 3.5. This filter has unit value between the frequencies $-\pi/T$ to π/T and is zero elsewhere. This frequency band is guaranteed to be greater than the Nyquist frequency, ω_n (i.e., $\omega_s = 2\pi/T > 2\omega_n$, so that $\pi/T > \omega_n$). In the space domain, this ideal reconstruction filter has the form:

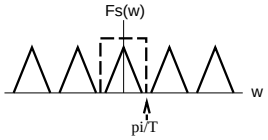


Figure 3.5 Reconstruction

$$h(x) \quad = \quad \frac{\sin(\pi x/T)}{\pi x/T}, \tag{3.6}$$

and is often referred to as the *ideal sync* function. Since reconstruction in the frequency domain is accomplished by multiplication with the ideal reconstruction filter, we could equivalently reconstruct the signal by convolving with the ideal sync in the space domain.

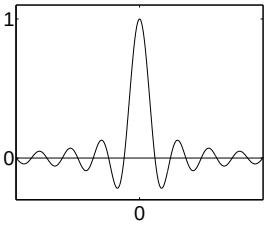


Figure 3.6 Ideal sync

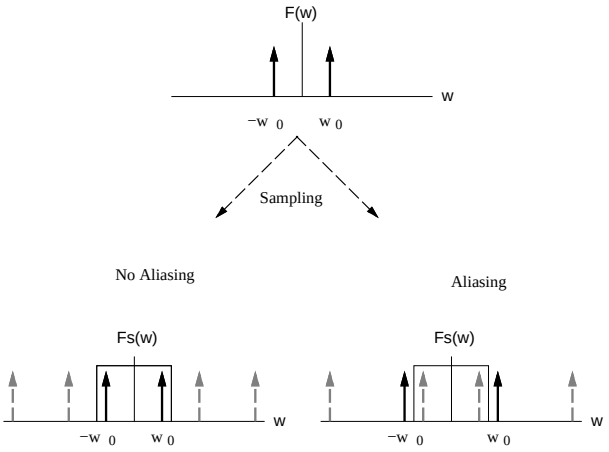
Example 3.1 Consider the following continuous-time signal:

$$f(x) \quad = \quad \cos(\omega_0 x),$$

a sinusoid with frequency ω_0. We will eventually be interested in sampling this function and seeing how the effects of aliasing are manifested. But first, let's compute the Fourier transform of this signal:

$$\begin{aligned} F(\omega) &= \sum_{k=-\infty}^{\infty} f(k)e^{-i\omega k} \\ &= \sum_{k=-\infty}^{\infty} \cos(\omega_0 k)(\cos(\omega k) - i\sin(\omega k)) \\ &= \sum_{k=-\infty}^{\infty} \cos(\omega_0 k)\cos(\omega k) - i\cos(\omega_0 k)\sin(\omega k) \end{aligned}$$

First let's consider the product of two cosines. It is easy to show from basic trigonometric identities that $\cos(A)\cos(B) = 0$ when $A \neq B$, and is equal to π when $|A| = |B|$. Similarly, one can show that $\cos(A)\sin(B) = 0$ for all A and B. So, the Fourier transform of $\cos(\omega_0 x) = \pi$ for $|\omega| = \omega_0$, and is 0 otherwise (see below). If the sampling rate is greater than $2\omega_0$, then there will be no aliasing, but if the sampling rate is less than $2\omega_0$, then the reconstructed signal will be of the form $\cos((\omega_s - \omega_0)x)$, that is, the reconstructed signal will be appear as a lower frequency sinusoid - it will be aliased.



We will close this chapter by drawing on the linear algebraic framework for additional intuition on the sampling and reconstruction process. First we will need to restrict ourselves to the sampling of an already sampled signal, that is, consider a m-dimensional signal sub-sampled to a n-dimensional signal. We may express this operation in matrix form as follows:

$$\begin{pmatrix} g_1 \\ \vdots \\ g_n \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 & 0 & 0 \\ \vdots & & & & \ddots & & & & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_{m-1} \\ f_m \end{pmatrix}$$

$$\vec{g}_n = S_{n\times m}\vec{f}_m, \tag{3.7}$$

where the subscripts denote the vector and matrix dimensions, and in this example $n = m/2$. Our goal now is to determine when it is possible to reconstruct the signal $\vec{f}$, from the sub-sampled signal $\vec{g}$. The Nyquist sampling theory tells us that if a signal is band-limited (i.e., can be written as a sum of a finite number of sinusoids), then we can sample it without loss of information. We can express this constraint in matrix notation:

$$\vec{f}_m = B_{m\times n}\vec{w}_n, \tag{3.8}$$

where the columns of the matrix B contains the basis set of sinusoids - in this case the first n sinusoids. Substituting into the above sampling equation gives:

$$\begin{aligned} \vec{g}_n &= S_{n\times m}B_{m\times n}\vec{w}_n \\ &= M_{n\times n}\vec{w}_n. \end{aligned} \tag{3.9}$$

If the matrix M is invertible, then the original weights (i.e., the representation of the original signal) can be determined by simply left-multiplying the sub-sampled signal $\vec{g}$ by M^{-1}. In other

words, Nyquist sampling theory can be thought of as simply a matrix inversion problem. This should not be at all surprising, the trick to sampling and perfect reconstruction is to simply limit the dimensionality of the signal to at most twice the number of samples.

Recall that the class of linear time-invariant systems are fully characterized by their impulse response. More specifically, the output of a linear time-invariant system to any input $f[x]$ can be determined via a convolution with the impulse response $h[x]$:

$$g[x] = f[x] \star h[x]. \tag{4.1}$$

Therefore the *filter* $h[x]$ and the linear-time invariant system are synonymous. In the frequency domain, this expression takes on the form:

$$G[\omega] = F[\omega]H[\omega]. \tag{4.2}$$

In other words, a filter modifies the frequencies of the input signal. It is often the case that such filters pass certain frequencies and attenuate others (e.g., a lowpass, bandpass, or highpass filters).

The design of such filters consists of four basic steps:

1. choose the desired frequency response
2. choose the length of the filter
3. define an error function to be minimized
4. choose a minimization technique and solve

The choice of frequency response depends, of course, on the designers particular application, and its selection is left to their discretion. We will however provide some general guidelines for choosing a frequency response that is amenable to a successful design. In choosing a filter size there are two conflicting goals, a large filter allows for a more accurate match to the desired frequency response, however a small filter is desirable in order to minimize computational demands [4]. The designer should experiment with varying size filters until an equitable balance is found. With a frequency response and filter size in hand, this chapter will provide the computational framework for realizing a finite length filter that "best" approximates the specified frequency response. Although there are numerous techniques for the design of digital filters we will cover only two such techniques chosen for their simplicity and generally good performance (see *Digital Filter Design* by T.W. Parks and C.S. Burrus for a full coverage of many other approaches).

[4] In multi-dimensional filter design, separability is also a desirable property.

A common class of filters are bandpass in nature, that is, they pass certain frequencies and attenuate others. An ideal lowpass, bandpass, and highpass filter are illustrated in Figure 4.1 Shown is the magnitude of the frequency response in the range $[0, \pi]$, since we are typically interested in designing real-valued, linear-phase filters, we need only specify one-half of the magnitude spectrum (the response is symmetric about the origin). The responses shown in Figure 4.1 are often referred to as *brick wall* filters because of their abrupt fall-off. A finite-length realization of such a filter produces undesirable "ringing" known as *Gibbs phenomena* as shown below in the magnitude of the frequency response of ideal lowpass filters of length 64, 32, and 16 (commonly referred to as the filter *tap size*).

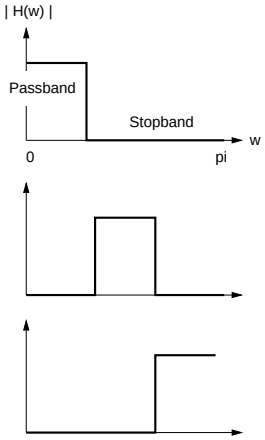
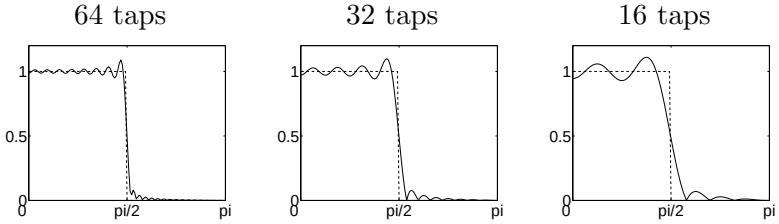


Figure 4.1
Ideal lowpass, bandpass, and highpass



These effects are particularly undesirable because neighboring frequencies may be passed or attenuated by wildly varying amounts, leading to general instabilities. To counter this problem, the designer is resigned to sacrificing the ideal response for a "softer" frequency response, Figure 4.2. Such a frequency response is amenable to a small finite-length filter free of significant ringing artifacts. The specific functional form of the soft falloff is somewhat arbitrary, however one popular form is a raised cosine. In its most general form, the frequency response takes the following form, where the bandpass nature of the response is controlled through ω_0, ω_1, $\Delta\omega_0$, and $\Delta\omega_1$.

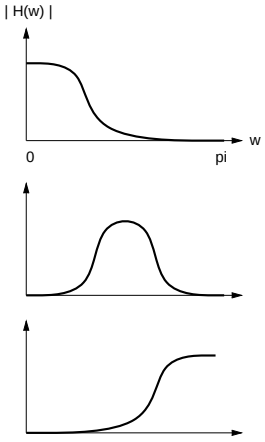


Figure 4.2 Soft lowpass, bandpass, and highpass

$$H(\omega) = \begin{cases} 0, & \omega < \omega_0 \\ \frac{1}{2}\left[1 - \cos(\pi(\omega - \omega_0)/\Delta\omega_0)\right], & \omega_0 \le \omega < \omega_0 + \Delta\omega_0 \\ 1, & \omega_0 + \Delta\omega_0 \le \omega < \omega_1 - \Delta\omega_1 \\ \frac{1}{2}\left[1 + \cos(\pi(\omega - (\omega_1 - \Delta\omega_1))/\Delta\omega_1)\right], & \omega_1 - \Delta\omega_1 \le \omega < \omega_1 \\ 0, & \omega_1 \le \omega. \end{cases}$$

In general, a larger transition width (i.e., $\Delta\omega_0$, and $\Delta\omega_1$) allows for smaller filters with minimal ringing artifacts. The tradeoff, of course, is that a larger transition width moves the frequency response further from the ideal brick-wall response. In specifying only half the frequency response (from $[0, \pi]$) we are implicitly imposing a symmetric frequency response and thus assuming that the desired filter is symmetric.

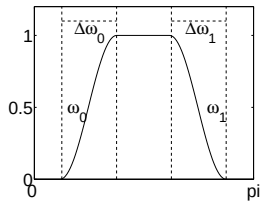


Figure 4.3 Frequency response

Once the desired frequency response has been chosen, the more difficult task of designing a finite-length filter that closely approximates this response begins. In general this problem is hard because by restricting ourselves to a small finite-length filter we are in effect asking to fit an arbitrarily complex function (the desired frequency response) with the sum of a small number of sinusoids.

We begin with the most straight-forward design method - frequency sampling. Our goal is to design a filter h whose Fourier transform best approximates the specified response H, that is:

$$\mathcal{F}(h) = H \tag{4.3}$$

where $\mathcal{F}$ is the Fourier transform operator. By applying the inverse Fourier transform we obtain a solution:

$$\begin{aligned} \mathcal{F}^{-1}(\mathcal{F}(h)) &= \mathcal{F}^{-1}(H) \\ h &= \mathcal{F}^{-1}(H). \end{aligned} \tag{4.4}$$

In other words, the design simply involves inverse Fourier transforming the specified frequency response. This series of steps can be made more explicit and practical for computer implementation by expressing the initial constraint (Equation (4.3)) in matrix notation:

$$M\vec{h} = \vec{H} \tag{4.5}$$

where $\vec{H}$ is the n-dimensional sampled frequency response, M is the $n \times n$ Fourier matrix (Equation (2.28)), and n is the chosen filter size. The filter h can be solved for by left multiplying both sides of Equation (4.5) by the inverse of the matrix F:

$$\vec{h} = M^{-1}\vec{H}. \tag{4.6}$$

Since the matrix M is square, this design is equivalent to solving for n unknowns (the filter taps) from n linearly independent equations. This fact illustrates the shortcomings of this approach, namely, that this method produces a filter with a frequency response that exactly matches the sampled response, but places no constraints on the response between sampled points. This restriction can often lead to poor results that are partially alleviated by the least-squares design presented next.

4.3 Least-Squares

Our goal is to design a filter $\vec{h}$ that "best" approximates a specified frequency response. As before this constraint can be expressed as:

$$M\vec{h} = \vec{H}, \tag{4.7}$$

where M is the $N \times n$ Fourier matrix (Equation (2.28)), $\vec{H}$ is the N sampled frequency response, and the filter size is n. Note that unlike before, this equation is over constrained, having n unknowns in $N > n$ equations. We can solve this system of equations in a least-squares sense by first writing a squared error function to be minimized:

$$E(\vec{h}) = \mid M\vec{h} - \vec{H} \mid^2 \tag{4.8}$$

In order to minimize [5], we differentiate with respect to the unknown $\vec{h}$:

$$\begin{aligned} \frac{dE(\vec{h})}{d\vec{h}} &= 2M^t \mid M\vec{h} - \vec{H} \mid \\ &= 2M^tM\vec{h} - 2M^t\vec{H}, \end{aligned} \tag{4.9}$$

then set equal to zero, and solve:

$$\vec{h} = (M^tM)^{-1}M^t\vec{H} \tag{4.10}$$

Shown in Figure 4.4 are a set of lowpass, bandpass, and highpass 16-tap filters designed using this technique. In this design, the frequency response was of the form given in Equation (4.3), with start and stop bands of $[\omega_0, \omega_1] = [0, \pi/2]$, $[\pi/4, 3\pi/4]$, and $[\pi/2, \pi]$, and a transition width of $\Delta\omega_0 = \Delta\omega_1 = \pi/4$. The frequency response was sampled at a rate of $N = 512$.

Finally, note that the previous frequency sampling design is equivalent to the least-squares design when the sampling of the Fourier basis is the same as the filter size (i.e., $N = n$).

4.4 Weighted Least-Squares

One drawback of the least-squares method is that the error function (Equation (4.8)) is uniform across all frequencies. This is easily rectified by introducing a weighting on the least-squares error function:

$$E(\vec{h}) = W \mid M\vec{h} - \vec{H} \mid^2 \tag{4.11}$$

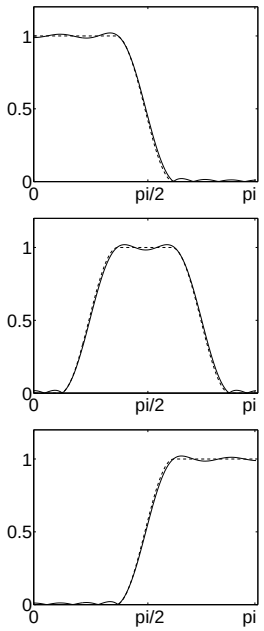


Figure 4.4 Least-squares: lowpass, bandpass, and highpass

[5] Because of Parseval's theorem (which amounts to the orthonormality of the Fourier transform), the minimal error in the frequency domain equates to a minimal error in the space domain.

where W is a diagonal weighting matrix. That is, the diagonal of the matrix contains the desired weighting of the error across frequency. As before, we minimize by differentiating with respect to $\vec{h}$:

$$\begin{aligned} \frac{dE(\vec{h})}{d\vec{h}} &= 2M^t W|\ M\vec{h} - \vec{H}\ | \\ &= 2M^t W M\vec{h} - 2WM^t\vec{H}, \end{aligned} \tag{4.12}$$

then set equal to zero, and solve:

$$\vec{h} = (M^t W M)^{-1} M^t W \vec{H}. \tag{4.13}$$

Note that this solution will be equivalent to the original least-squares solution (Equation (4.10)) when W is the identity matrix (i.e., uniform weighting).

Shown in Figure 4.5 is a comparison of an 8-tap lowpass filter designed with a uniform weighting, and with a weighting that emphasizes the errors in the low frequency range, $W(\omega) = \frac{1}{(|\omega|+1)^8}$. Note that in the case of the later, the errors in the low frequencies are smaller, while the errors in the high frequencies have increased.

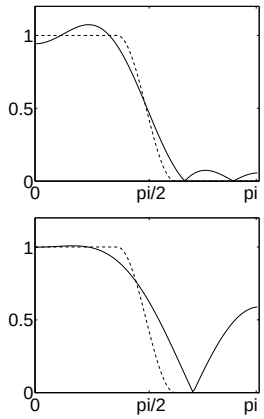


Figure 4.5
Least-squares and weighted least squares

5.1 Pinhole Camera

The history of the pinhole camera (or camera obscura) dates back as early as the fifth century B.C., and continues to be popular today among students, artists, and scientists. The Chinese philosopher Mo Ti is believed to be the first to notice that objects reflect light in all directions and that the light rays that pass through a small hole produce an inverted image. In its simplest form a pinhole camera is a light-tight box with a tiny hole in one end and a photo-sensitive material on the other. Remarkably, this simple device is capable of producing a photograph. However, the pinhole camera is not a particularly efficient imaging system (often requiring exposure times as long as several hours) and is more popular for its artistic value than for its practical value. Nevertheless, the pinhole camera is convenient because it affords a simple model of more complex imaging systems. That is, with a pinhole camera model, the projection of points from the three-dimensional world onto the two-dimensional sensor takes on a particularly simple form.

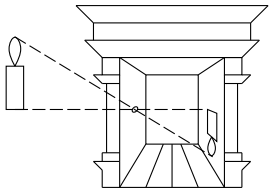


Figure 5.1 Pinhole image formation

Denote a point in the three-dimensional world as a column vector, $\vec{P} = (\,X \quad Y \quad Z\,)^t$ and the projection of this point onto the two dimensional image plane as $\vec{p} = (\,x \quad y\,)^t$. Note that the world and image points are expressed with respect to their own coordinate systems, and for convenience, the image coordinate system is chosen to be orthogonal to the Z-axis, i.e., the origins of the two systems are related by a one-dimensional translation along the $Z-$axis or *optical axis*. It is straight-forward to show from a similar triangles argument that the relationship between the world and image point is:

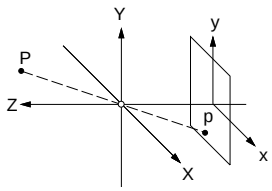


Figure 5.2 Perspective projection

$$x = -\frac{dX}{Z} \qquad \text{and} \qquad y = -\frac{dY}{Z}, \tag{5.1}$$

where d is the displacement of the image plane along the Z-axis [6] These equations are frequently referred to as the *perspective projection* equations. Although non-linear in their nature, the perspective projection equations may be expressed in matrix form

[6] The value d in Equation (5.1) is often referred to as the focal length. We do not adopt this convention primarily because it is a misnomer, under the pinhole model all points are imaged in perfect focus.

using the homogeneous equations:

$$\begin{pmatrix} x_s \\ y_s \\ s \end{pmatrix} = \begin{pmatrix} -d & 0 & 0 & 0 \\ 0 & -d & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}, \tag{5.2}$$

where the final image coordinates are given by $(x \quad y)^t = (\frac{x_s}{s} \quad \frac{y_s}{s})^t$.

An approximation to the above perspective projection equations is *orthographic projection*, where light rays are assumed to travel from a point in the world parallel to the optical axis until they intersect the image plane. Unlike the pinhole camera and perspective projection equations, this model is not physically realizable and is used primarily because the projection equations take on a particularly simple linear form:

$$x = X \qquad \text{and} \qquad y = Y. \tag{5.3}$$

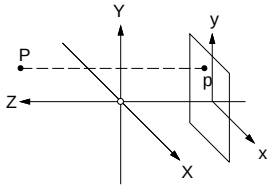


Figure 5.3 Orthographic projection

And in matrix form:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{5.4}$$

Orthographic projection is a reasonable approximation to perspective projection when the difference in depth between points in the world is small relative to their distance to the image plane. In the special case when all the points lie on a single frontal-parallel surface relative to the image plane (i.e., $\frac{d}{Z}$ is constant in Equation (5.1)), the difference between perspective and orthographic is only a scale factor.

5.2 Lenses

It is important to remember that both the perspective and orthographic projection equations are only approximations of more complex imaging systems. Commercial cameras are constructed with a variety of lenses that collect and focus light onto the image plane. That is, light emanates from a point in the world in all directions and, whereas a pinhole camera captures a single light ray, a lens collects a multitude of light rays and focuses the light to a small region on the image plane. Such complex imaging systems are often described with the simpler *thin-lens* model. Under the thin-lens model the projection of the central or *principal* ray obeys the rules of perspective projection, Equation (5.1): the point $\vec{P} = (X \quad Y \quad Z)^t$ is projected onto the image plane centered about the point $(x \quad y)^t = (\frac{-dX}{Z} \quad \frac{-dY}{Z})^t$. If the point $\vec{P}$

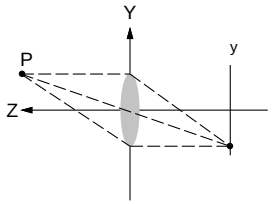


Figure 5.4 Thin lens

is in perfect focus, then the remaining light rays captured by the lens also strike the image plane at the point $\vec{p}$. A point is imaged in perfect focus if its distance from the lens satisfies the following *thin-lens equation*:

$$\frac{1}{Z} + \frac{1}{d} = \frac{1}{f}, \tag{5.5}$$

where d is the distance between the lens and image plane along the optical axis, and f is the focal length of the lens. The focal length is defined to be the distance from the lens to the image plane such that the image of an object that is infinitely far away is imaged in perfect focus. Points at a depth of $Z_o \neq Z$ are imaged onto a small region on the image plane, often modeled as a blurred circle with radius r:

$$r = \frac{R}{\frac{1}{f} - \frac{1}{Z_o}} \left| \left(\frac{1}{f} - \frac{1}{Z_o} \right) - \frac{1}{d} \right|, \tag{5.6}$$

where R is the radius of the lens. Note that when the depth of a point satisfies Equation (5.5), the blur radius is zero. Note also that as the lens radius R approaches 0 (i.e., a pinhole camera), the blur radius also approaches zero for all points independent of its depth (referred to as an infinite depth of field).

Alternatively, the projection of each light ray can be described in the following more compact matrix notation:

$$\begin{pmatrix} l_2 \\ \alpha_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ -\frac{1}{R}\left(\frac{n_2 - n_1}{n_2}\right) & \frac{n_1}{n_2} \end{pmatrix} \begin{pmatrix} l_1 \\ \alpha_1 \end{pmatrix}, \tag{5.7}$$

where R is the radius of the lens, n_1 and n_2 are the index of refraction for air and the lens material, respectively. l_1 and l_2 are the height at which a light ray enters and exits the lens (the thin lens idealization ensures that $l_1 = l_2$). α_1 is the angle between the entering light ray and the optical axis, and α_2 is the angle between the exiting light ray and the optical axis. This formulation is particularly convenient because a variety of lenses can be described in matrix form so that a complex lens train can then be modeled as a simple product of matrices.

Image formation, independent of the particular model, is a three-dimensional to two-dimensional transformation. Inherent to such a transformation is a loss of information, in this case depth information. Specifically, all points of the form $\vec{P}_c = (\, cX \quad cY \quad cZ \,)^t$, for any $c \in \mathcal{R}$, are projected to the same point $(\, x \quad y \,)^t$ - the projection is not one-to-one and thus not invertible. In addition to this geometric argument for the non-invertibility of image formation, a similarly straight-forward linear algebraic argument holds.

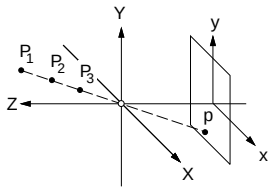


Figure 5.5
Non-invertible projection

In particular, we have seen that the image formation equations may be written in matrix form as, $\vec{p} = M_{n\times m}\vec{P}$, where $n < m$ (e.g., Equation (5.2)). Since the projection is from a higher dimensional space to a lower dimensional space, the matrix M is not invertible and thus the projection is not invertible.

5.3 CCD

To this point we have described the geometry of image formation, how light travels through an imaging system. To complete the image formation process we need to discuss how the light that strikes the image plane is recorded and converted into a digital image. The core technology used by most digital cameras is the charge-coupled device (CCD), first introduced in 1969. A basic CCD consists of a series of closely spaced metal-oxide-semiconductor capacitors (MOS), each one corresponding to a single image pixel. In its most basic form a CCD is a charge storage and transport device: charge is stored on the MOS capacitors and then transported across these capacitors for readout and subsequent transformation to a digital image. More specifically, when a positive voltage, V, is applied to the surface of a P-type MOS capacitor, positive charge migrates toward ground. The region depleted of positive charge is called the depletion region. When photons (i.e., light) enter the depletion region, the electrons released are stored in this region. The value of the stored charge is proportional to the intensity of the light striking the capacitor. A digital image is subsequently formed by transferring the stored charge from one depletion region to the next. The stored charge is transferred across a series of MOS capacitors (e.g., a row or column of the CCD array) by sequentially applying voltage to each MOS capacitor. As charge passes through the last capacitor in the series, an amplifier converts the charge into a voltage. An analog-to-digital converter then translates this voltage into a number (i.e., the intensity of an image pixel).

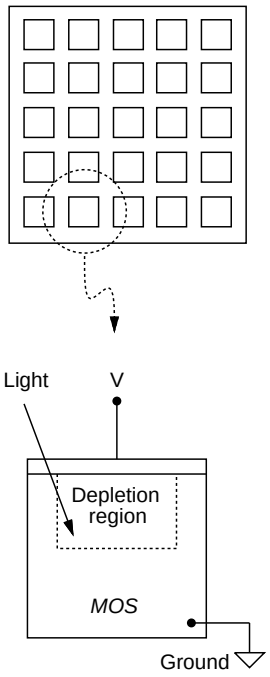


Figure 5.6 MOS capacitor

6.1 Lookup Table

The internal representation of a digital image is simply a matrix of numbers representing grayscale or color values. But when an image is displayed on a computer monitor we typically do not see a direct mapping of the image. An image is first passed through a *lookup table* (LUT) that maps the image intensity values to brightness values, Figure 6.1. If the lookup table is linear with unit slope and zero intercept then the image is directly mapped to the display, otherwise, the displayed image will not be an exact representation of the underlying image. For example, most computer monitors intentionally impose a non-linear LUT of the general form $D = I^{\alpha}$ (i.e., gamma correction), where α is a real number, and D and I are the displayed and image values. A variety of interesting visual effects can be achieved by simple manipulations of the functional form of the LUT. Keep in mind though that in manipulating the lookup table, the underlying image is left untouched, it is only the *mapping* from pixel value to display brightness that is being effected.

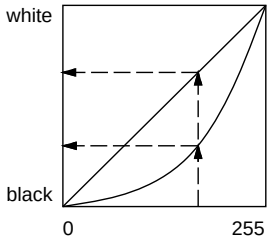


Figure 6.1 Lookup table

6.2 Brightness/Contrast

Perhaps the most common and familiar example of a LUT manipulation is to control the brightness or darkness of an image as shown in Figure 6.3. The bright and dark images of Einstein were created by passing the middle image through the LUTs shown in the same figure. The functional form of the LUT is a unit-slope line with varying intercepts: $g(u) = u + b$, with the image intensity values $u \in [0, 1]$. A value of $b > 0$ results in a brightening of the image and $b < 0$ a darkening of the image. Another common manipulation is that of controlling the contrast of an image as shown in Figure 6.4. The top image is said to be high contrast and the bottom image low contrast, with the corresponding LUTs shown in the same figure. The functional form of these LUTs is linear: $g(u) = mu + b$, where the relationship between the slope and intercept is $b = 1/2(1 - m)$. The image contrast is increased with a large slope and negative intercept (in the limit, $m \to \infty$ and $b \to -\infty$), and the contrast is reduced with a small slope and positive intercept ($m \to 0$ and $b \to 1/2$). The image is *inverted* with $m = -1$ and $b = 1$, that is white is mapped to black, and black is mapped to white. Of course, an image can be both contrast enhanced and brightened or darkened by simply passing the image through two (or more) LUTs.

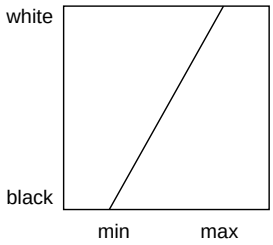


Figure 6.2 Autoscale

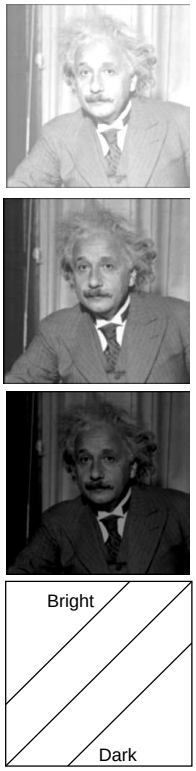


Figure 6.3 Brightness

Figure 6.4 Contrast

Autoscaling is a special case of contrast enhancement where the minimum image intensity value is mapped to black and the maximum value is mapped to white, Figure 6.2. Autoscaling maximizes the contrast without saturating at black or white. The problem with this sort of autoscaling is that a few stray pixels can dictate the contrast resulting in a low-contrast image. A less sensitive approach is to sort the intensity values in increasing order and map the 1% and 99% intensity values to black and white, respectively. Although this will lead to a small amount of saturation, it rarely fails to produce a high-contrast image.

6.3 Gamma Correction

Typically, high contrast images are visually more appealing. However a drawback of linear contrast enhancement described above is that it leads to saturation at both the low and high end of the intensity range. This may be avoided by employing a non-linear contrast adjustment scheme, also realizable as a lookup table (LUT) manipulation. The most standard approach is *gamma correction*, where the LUT takes the form:

$$g(u) \quad = \quad u^{\alpha}, \tag{6.1}$$

where $\alpha > 1$ increases contrast, and $\alpha < 1$ reduces contrast. Shown in Figure 6.5 are contrast enhanced (top: $\alpha = 2$) and contrast reduced (bottom: $\alpha = 1/2$) images. Note that with the intensity values u scaled into the range $[0, 1]$, black (0) and white (1) are mapped to themselves. That is, there is no saturation at the low or high end. Gamma correction is widely used in a number of devices because it yields reasonable results and is easily parameterized. One drawback to this scheme is that the gray values are mapped in an asymmetric fashion with respect to mid-level gray (0.5). This may be alleviated by employing a *sigmoidal* non-linearity of the form

$$g(u) \quad = \quad \frac{1}{1 + e^{-\alpha u + \beta}}. \tag{6.2}$$

In order that $g(u)$ be bound by the interval $[0, 1]$, it must be scaled as follows: $(g(u) - c_1)/c_2$, where $c_1 = 1/(1 + e^{\beta})$ and $c_2 = 1/(1+e^{-\alpha+\beta}) - c1$. This non-linearity, with its two degrees of freedom, is more versatile and can produce a more balanced contrast enhancement. Shown in Figure 6.6 is a contrast enhanced image with $\alpha = 12$ and $\beta = 6$.

6.4 Quantize/Threshold

A digital image, by its very nature, is *quantized* to a discrete number of intensity values. For example an image quantized to 8-bits contains $2^8 = 256$ possible intensity values, typically in the range $[0, 255]$. An image can be further quantized to a lower number of bits (b) or intensity values (2^b). Quantization can be accomplished by passing the image through a LUT containing a step function, Figure 6.7, where the number of steps governs the number of intensity values. Shown in Figure 6.7 is an image of Einstein quantized to five intensity values, notice that all the subtle variations in the curtain and in his face and jacket have been eliminated. In the limit, when an image is quantized to one bit or two intensity values, the image is said to be *thresholded*. Shown in Figure 6.8 is a thresholded image of Einstein and the corresponding LUT, a two-step function. The point at which the step function transitions from zero to one is called the threshold and can of course be made to be any value (i.e., slid left or right).

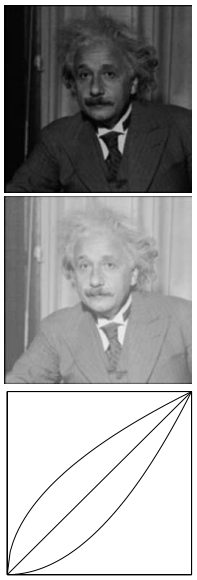


Figure 6.5 Contrast: Gamma

6.5 Histogram Equalize

The intensity values of a typical image are often distributed unevenly across the full range of 0 to 255 (for an 8-bit image), with most the mass near mid-gray (128) and falling off on either side, Figure 6.9. An image can be transformed so that the distribution of intensity values is flat, that is, each intensity value is equally represented in the image. This process is known has *histogram equalization* [7]. Although it may not be immediately obvious an image is histogram equalized by passing it through a LUT with the functional form of the cumulative distribution function. More specifically, define $N(u)$ as the number of pixels with intensity value u, this is the image histogram and a discrete approximation to the probability distribution function. Then, the cumulative distribution function is defined as:

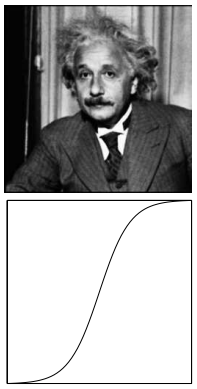


Figure 6.6 Contrast: Sigmoid

$$C(y) = \sum_{i=0}^{u} N(i), \tag{6.3}$$

that is, $C(u)$ is the number of pixels with intensity value less than or equal to u. Histogram equalization then amounts to simply inserting this function into the LUT. Shown in Figure 6.9 is Einstein before and after histogram equalization. Notice that the effect is similar to contrast enhancement, which intuitively should make sense since we increased the number of black and white pixels.

[7] Why anyone would want to histogram equalize an image is a mystery to me, but here it is in case you do.

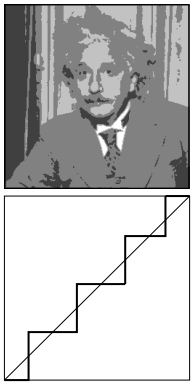


Figure 6.7 Quantize

In all of these examples the appearance of an image was altered by simply manipulating the LUT, the mapping from image intensity value to display brightness value. Such a manipulation leaves the image content intact, it is a non-destructive operation and thus completely invertible. These operations can be made destructive by applying the LUT operation directly to the image. For example an image can be brightened by adding a constant to each pixel, and then displaying with a linear LUT. Since such an operation is destructive it may not be inveritble, for example when brightening an 8-bit image, all pixels that exceed the value 255 will be truncated to 255.

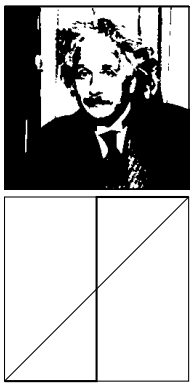


Figure 6.8 Threshold

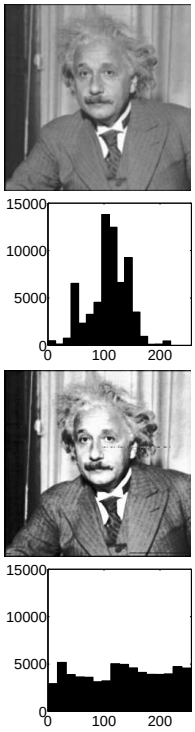


Figure 6.9 Histogram equalize

7.1 Convolution

The one-dimensional convolution sum, Equation (2.5), formed the basis for much of our discussion on discrete-time signals and systems. Similarly the two-dimensional convolution sum will form the basis from which we begin our discussion on image processing and computer vision.

The 1-D convolution sum extends naturally to higher dimensions. Consider an image $f[x, y]$ and a two-dimensional filter $h[x, y]$. The 2-D convolution sum is then given by:

$$g[x, y] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k, l]h[x - k, y - l]. \quad (7.1)$$

In 1-D, the intuition for a convolution is that of computing inner products between the filter and signal as the filter "slides" across the signal. The same intuition holds in 2-D. Inner products are computed between the 2-D filter and underlying image as the filter slides from left-to-right/top-to-bottom.

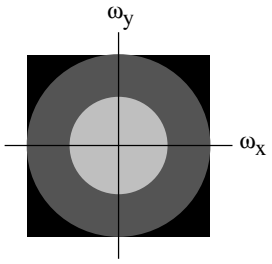


Figure 7.1 2-D Frequency

In the Fourier domain, this convolution is equivalent to multiplying the, now 2-D, Fourier transforms of the filter and image, where the 2-D Fourier transform is given by:

$$F[\omega_x, \omega_y] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k, l]e^{-i(\omega_x k + \omega_y l)}. \quad (7.2)$$

The notion of low-pass, band-pass, and high-pass filtering extends naturally to two-dimensional images. Shown in Figure 7.1 is a simplified decomposition of the 2-D Fourier domain parameterized by ω_x and $\omega_y \in [-\pi, \pi]$. The inner disc corresponds to the lowest frequencies, the center annulus to the middle (band) frequencies, and the outer dark area to the highest frequencies.

Figure 7.2 Low-, Band-, High-pass

Two of the most common (and opposing) linear filtering operations are *blurring* and *sharpening*. Both of these operations can be accomplished with a 2-D filter and 2-D convolution, or more efficiently with a 1-D filter and a pair of 1-D horizontal and vertical convolutions. For example, a 2-D convolution with the blur filter:

$$\begin{pmatrix} 0.0625 & 0.1250 & 0.0625 \\ 0.1250 & 0.2500 & 0.1250 \\ 0.0625 & 0.1250 & 0.0625 \end{pmatrix}$$

can be realized by convolving in the horizontal and vertical directions with the 1-D filter:

$$\text{blur} \quad = \quad (\,0.25 \quad 0.50 \quad 0.25\,)\,. \tag{7.3}$$

That is, an outer-product of the 1-D filter with itself yields the 2-D filter - the filters are *xy-separable*. The separability of 2-D filters is attractive for two reasons: (1) it is computationally more efficient and (2) it simplifies the filter design. A generic blur filter may be constructed from any row of the binomial coefficients:

$$\begin{array}{ccccccc} & & & 1 & 1 & & \\ & & 1 & 2 & 1 & & \\ & 1 & 3 & 3 & 1 & & \\ 1 & 4 & 6 & 4 & 1 & & \\ 1 & 5 & 10 & 10 & 5 & 1 & \\ 1 & 6 & 15 & 20 & 15 & 6 & 1 \end{array}$$

where each row (filter) should be normalized by it's sum (i.e., blur filters should always be unit-sum so as not to increase or decrease the mean image intensity). The amount of blur is then directly proportional to the size of the filter. Blurring simply reduces the high-frequency content in an image. The opposing operation, sharpening, is meant to enhance the high-frequencies. A generic separable sharpening filter is of the form:

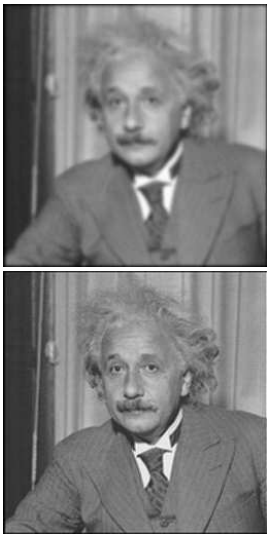


Figure 7.3 Blur and Sharpen

$$\text{sharp} \quad = \quad (\,0.08 \quad -1.00 \quad 0.08\,)\,. \tag{7.4}$$

This filter leaves the low-frequencies intact while enhancing the contribution of the high-frequencies. Shown in Figure 7.3 are results from blurring and sharpening.

7.2 Derivative Filters

Discrete differentiation forms the foundation for many applications in image processing and computer vision. We are all familiar with the definition of the derivative of a *continuous* signal $f(x)$:

$$D\{f(x)\} \quad = \quad \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon) - f(x)}{\varepsilon}. \tag{7.5}$$

This definition requires that the signal $f(x)$ be well defined for all $x \in \mathcal{R}$. So, does it make sense to differentiate a *discretely* sampled signal, $D\{f[x]\}$, which is only defined over an integer sampling lattice? Strictly speaking, no. But our intuition may be that this is not such an unreasonable request. After all, we know how to differentiate $f(x)$, from which the sampled signal $f[x]$ was derived, so why not just differentiate the continuous signal $f(x)$ and then sample the result? Surely this is what we have in mind when we

ask for the derivative of a sampled signal. But one should not be fooled by the seeming simplicity of our intuition, as we will soon discover the design of an accurate and efficient discrete derivative operator will prove to be sufficiently challenging.

Recall from earlier chapters that under certain conditions (Nyquist theory), the relationship between the continuous and sampled signals can be expressed precisely as:

$$f(x) \quad = \quad f[x] \star h(x), \tag{7.6}$$

where $h(x) = \sin(\pi x/T)/(\pi x/T)$ is the ideal sync, and $\star$ is the convolution operator. Now, applying the continuous differential operator to both sides yields:

$$D\{f(x)\} \quad = \quad D\{f[x] \star h(x)\}, \tag{7.7}$$

and expressing the right-hand side in terms of the convolution sum:

$$\begin{aligned} D\{f(x)\} &= D\left\{ \sum_{k=-\infty}^{\infty} f[k]h(x-k) \right\} \\ &= \sum_{k=-\infty}^{\infty} f[k]D\{h(x-k)\} \\ &= f[x] \star D\{h(x)\}. \end{aligned} \tag{7.8}$$

Notice that the derivative operator $D\{\cdot\}$ is being applied only to continuous entities. Having computed the desired derivatives, we need only sample the results, denoting $S\{\cdot\}$ as the sampling operator:

$$\begin{aligned} S\{\ D\{f(x)\}\ \} &= f[x] \star S\{\ D\{h(x)\}\ \} \\ S\{f'(x)\} &= f[x] \star S\{h'(x)\} \\ f'[x] &= f[x] \star h'[x]. \end{aligned} \tag{7.9}$$

On the left-hand side of the above equation is the desired quantity, the derivative of the sampled signal. On the right-hand side is a discrete convolution between two known quantities, the sampled derivative of the sync and the original sampled signal. The derivative of the sync can be expressed analytically by simply differentiating the sync function:

$$h'(x) \quad = \quad \frac{\pi^2 x/T^2 \cos(\pi x/T) - \pi/T \sin(\pi x/T)}{(\pi x/T)^2}, \tag{7.10}$$

where T is the sampling period at which $f(x)$ was sampled. So, if the signal $f(x)$ is sampled above the Nyquist rate and if it is in

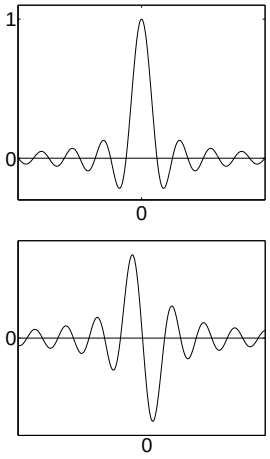


Figure 7.4 Ideal sync and its derivative

fact differentiable, then Equation (7.9) tells us that we can exactly compute the derivative of the sampled signal $f[x]$, an altogether happy ending.

If you are feeling a bit uneasy it is for a good reason. Although mathematically correct, we have a solution for differentiating a discretely sampled signal that is physically unrealizable. In particular the derivative of the sync, $h'(x)$, is spatially infinite in extent, meaning that it cannot be implemented on a finite machine. And even worse, $h'(x)$ falls off slowly from the origin so that truncation will cause significant inaccuracies. So we are going to have to part with mathematical perfection and design a finite-length filter.

To begin we need to compute the frequency response of the ideal derivative filter. We can compute the response indirectly by first expressing $f[x]$ in terms of its Fourier series:

$$f[x] = \frac{1}{2\pi} \sum_{\omega=-\pi}^{\pi} F[\omega] e^{i\omega x}, \tag{7.11}$$

and then differentiating both sides with respect to x:

$$\begin{aligned} D\{f[x]\} &= \frac{1}{2\pi} \sum_{\omega=-\pi}^{\pi} F[\omega] D\{e^{i\omega x}\} \\ &= \frac{1}{2\pi} \sum_{\omega=-\pi}^{\pi} i\omega F[\omega] e^{i\omega x}. \end{aligned} \tag{7.12}$$

Differentiation in the space domain is then seen to be equivalent to multiplying the Fourier transform $F[\omega]$ by an imaginary ramp $i\omega$. And since multiplication in the frequency domain is equivalent to convolution in the space domain, an imaginary ramp is the frequency response of the ideal derivative filter. Trying to directly design a finite length filter to this response is futile because of the discontinuity at $-\pi/\pi$, which of course accounts for the spatially infinite extent of $h'(x)$. So we are resigned to designing a filter with a *periodic* frequency response that "best" approximates a ramp. The simplest such approximation is that of a sinusoid where, at least in the low-frequency range, the match is reasonably good (i.e., $\sin(\omega) = \omega$, for small ω). Employing the least-squares filter design technique (Equation (4.8)) we formulate a quadratic error function to be minimized:

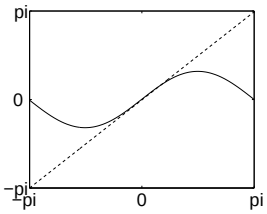


Figure 7.5 Ideal and approximate derivative frequency response

$$E(\vec{h}) = \mid M\vec{h} - \vec{H} \mid^2, \tag{7.13}$$

where M is the $N \times n$ Fourier matrix (Equation (2.28)), $\vec{H}$ is the N sampled desired frequency response, and n the filter size is.

To minimize we differentiate, set equal to zero and solve for the minimal solution:

$$\vec{h} \;=\; (M^t M)^{-1} M^t \vec{H} \tag{7.14}$$

Since the desired frequency response, a sinusoid, has only two degrees of freedom, amplitude and phase, a 2-tap filter will suffice (i.e., $n = 2$). The resulting filter is of the form $\vec{h} = (\,0.5 \quad -0.5\,)$. Intuitively this is exactly what we should have expected - for example, applying this filter via a convolution and evaluating at, arbitrarily, $n = 0$ yields:

$$\begin{aligned} f'[x] &= h[x] \star f[x] \\ &= \sum_{k=-\infty}^{\infty} h[x-k] f[k] \\ f'[0] &= h[1] f[-1] + h[0] f[0] \\ &= 0.5 f[0] - 0.5 f[-1]. \end{aligned} \tag{7.15}$$

Note that the derivative is being approximated with a simple two-point difference, that is, a discrete approximation to the continuous definition in Equation (7.5). We could of course greatly improve on this filter design. But since we are really interested in multi-dimensional differentiation, let's put aside further analysis of the one-dimensional case and move on to the two-dimensional case.

It has been the tendency to blindly extend the one-dimensional design to higher-dimensions, but, as we will see shortly, in higher-dimensions the story becomes slightly more complicated. In the context of higher-dimensional signals we first need to consider partial derivatives. For example the partial derivative of a two dimensional signal $f(x, y)$ in it's first argument is defined as:

$$\begin{aligned} f_x(x,y) &\equiv \frac{\partial f(x,y)}{\partial x} \\ &= \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon, y) - f(x,y)}{\varepsilon}. \end{aligned} \tag{7.16}$$

According to the Nyquist theory, the continuous and discrete signals (if properly sampled) are related by the following equality:

$$f(x,y) \;=\; f[x,y] \star h(x,y), \tag{7.17}$$

where $h(x,y) = \frac{\sin(\pi x/T)\sin(\pi y/T)}{\pi^2 xy/T^2}$ is the two-dimensional ideal sync. As before we apply the continuous partial differential operator to both sides:

$$D_x\{f(x,y)\} \;=\; f[x,y] \star D_x\{h(x,y)\}, \tag{7.18}$$

noting again that the differential operator is only applied to continuous entities. Since the two-dimensional sync is separable (i.e., $h(x,y) = h(x) \star h(y)$), the above equation can be rewritten as:

$$\begin{aligned} f_x(x,y) &= f[x,y] \star D_x\{h(x) \star h(y)\} \\ &= f[x,y] \star D_x\{h(x)\} \star h(y) \\ &= f[x,y] \star h'(x) \star h(y). \end{aligned} \tag{7.19}$$

And finally, sampling both sides gives an expression for the partial derivative of the discretely sampled two-dimensional signal:

$$\begin{aligned} S\{f_x(x,y)\} &= f[x,y] \star S\{h'(x)\} \star S\{h(y)\} \\ f_x[x,y] &= f[x,y] \star h'[x] \star h[y]. \end{aligned} \tag{7.20}$$

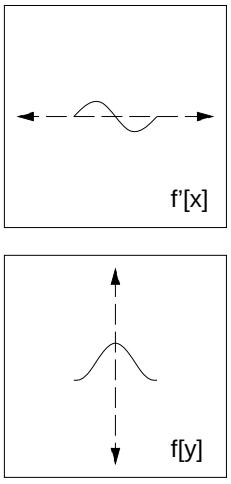


Figure 7.6 Horizontal partial differentiation

Notice that calculating the partial derivative requires a pair of one-dimensional convolutions: a derivative filter, $h'[x]$, in the dimension of differentiation, and an interpolation filter, $h[y]$, in the other dimension (for multi-dimensional signals, *all* remaining dimensions would be convolved with the interpolation filter). Since two-dimensional differentiation reduces to a pair of one-dimensional convolutions it is tempting to simply employ the same differentiation filter used in the one-dimensional case. But since a pair of filters are now required perhaps we should give this some additional thought.

In some ways the choice of filters seems trivial: chose an interpolation function $h(x)$, differentiate it to get the derivative function $h'(x)$, and sample these functions to get the final digital filters $h[x]$ and $h'[x]$. So how is this different from the one-dimensional case? In the one-dimensional case only the derivative filter is employed, whereas in the two-dimensional case we require the pair of filters. And by our formulation we know that the pair of filters should satisfy the relationship that one is the derivative of the other $h'(x) = D(h(x))$. And in fact this constraint is automatically enforced by the very nature in which the *continuous* functions are chosen, but in the final step, these functions are sampled to produce *discrete* filters. This sampling step typically destroys the required derivative relationship, and, although a seemingly subtle point, has dramatic effects on the accuracy of the resulting derivative operator. For example consider the often used Sobel derivative filters with $h[x] = (1 \quad \sqrt{2} \quad 1)/(2+\sqrt{2})$ and $h'[x] = (1 \quad 0 \quad -1)/3$. Shown in Figure 7.7 are the magnitudes of the Fourier transform of the derivative filter (solid line) and the interpolation filter times $i\omega$ (i.e., frequency domain differentiation). If the filters obeyed the required derivative relationship than these curves would be exactly matched, which they clearly

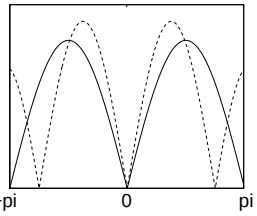


Figure 7.7
Sobel frequency response

are not. The mismatching of the filters results in gross inaccuracies in derivative measurements. Let's see then if we can design a better set of filters.

We begin by writing down the desired relationship between the derivative and interpolation filters, most conveniently expressed in the frequency domain:

$$H'(\omega) \quad = \quad i\omega H(\omega), \tag{7.21}$$

from which we can write a weighted least-squares error functional to be minimized:

$$E(H, H') \quad = \quad \int d\omega \ \left[W(\omega)(i\omega H(\omega) - H'(\omega))\right]^2, \tag{7.22}$$

where $W(\omega)$ is a frequency weighting function. Next, we write a discrete approximation of this continuous error functional over the n-vectors $\vec{h}$ and $\vec{h'}$ containing the sampled derivative and interpolation filters, respectively:

$$E(\vec{h}, \vec{h'}) \quad = \quad |W(F'\vec{h} - F\vec{h'})|^2, \tag{7.23}$$

where the columns of the matrix $F_{m\times n}$ contain the first n Fourier basis functions (i.e., a discrete-time Fourier transform), the matrix $F'_{m\times n} = i\omega F_{m\times n}$, and $W_{m\times m}$ is a diagonal frequency weighting matrix. Note that the dimension n is determined by the filter size and the dimension m is the sampling rate of the continuous Fourier basis functions, which should be chosen to be sufficiently large to avoid sampling artifacts. This error function can be expressed more concisely as:

$$E(\vec{u}) \quad = \quad |M\vec{u}|^2, \tag{7.24}$$

where the matrix M and the vector $\vec{u}$ are constructed by "packing together" matrices and vectors:

$$M = (\, WF' \quad | \quad -WF\,) \qquad \text{and} \qquad \vec{u} = \begin{pmatrix} \vec{h} \\ \vec{h'} \end{pmatrix}. \tag{7.25}$$

The minimal unit vector $\vec{u}$ is then simply the minimal-eigenvalue eigenvector of the matrix M^tM. After solving for $\vec{u}$, the derivative and interpolation filters can be "unpacked" and normalized so that the interpolation filter is unit sum. Below are the resulting filter values for a 3-tap and 5-tap filter pair.

Shown in Figure 7.8 are the matching of these filters in the frequency domain. Notice that the 5-tap filters are nearly perfectly matched.

h	0.223755	0.552490	0.223755
h'	0.453014	0.0	-0.453014

h	0.036420	0.248972	0.429217	0.248972	0.036420
h'	0.108415	0.280353	0.0	-0.280353	-0.108415

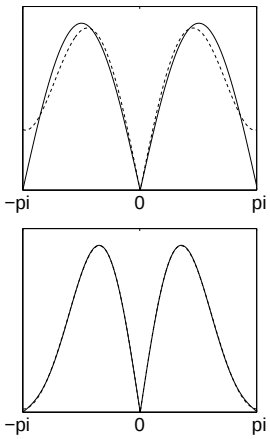


Figure 7.8
Frequency response of matched derivative filters

Higher-order derivative filters can be designed by replacing the initial constraint in Equation (7.21) with $H'(\omega) = (i\omega)^k H(\omega)$ for a k^{th} order derivative.

A peculiar aspect of this filter design is that nowhere did we explicitly try to model a specified frequency response. Rather, the design fell naturally from the relationship between the continuous- and discrete-time signals and the application of the continuous derivative operator, and in this way is quite distinct from the one-dimensional case. The proper choice of derivative filters can have a dramatic impact on the applications which utilize them. For example, a common application of differential measurements is in measuring motion from a movie sequence $f(x, y, t)$. The standard formulation for motion estimation is:

$$f_x(x, y, t)v_x(x, y) + f_y(x, y, t)v_y(x, y) + f_t(x, y, t) = 0, \quad (7.26)$$

where the motion vector is $\vec{v} = (\, v_x \quad v_y \,)^t$, and $f_x(\cdot)$, $f_y(\cdot)$, and $f_t(\cdot)$ are the partial derivatives with respect to space and time. Shown in Figure 7.9 are the resulting motion fields for a simple translational motion with the Sobel (top panel) and matched (bottom) derivative filters used to compute the various derivatives. Although these filters are the same size, the difference in accuracy is significant.

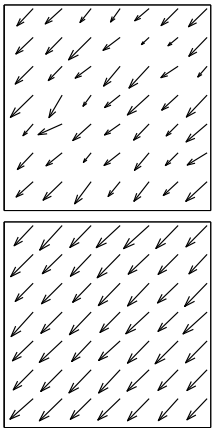


Figure 7.9
Differential motion estimation

7.3 Steerable Filters

In the previous section we showed how to compute horizontal and vertical partial derivatives of images. One may naturally wonder how to compute a derivative in an arbitrary direction. Quite remarkably it turns out that we need not design a new set of filters for each possible direction because the derivative in any direction can be synthesized from a linear combination of the horizontal and vertical derivatives. This property of derivatives has been termed *steerability*. There are several formulations of this property, we chose to work in the frequency domain where differentiation takes on a particularly simple form.

To begin, we express a two-dimensional image with respect to its Fourier series:

$$f(x,y) = \sum_{\omega_x=-\pi}^{\pi} \sum_{\omega_y=-\pi}^{\pi} F(\omega_x,\omega_y) e^{-j(\omega_x x+\omega_y y)}. \quad (7.27)$$

Differentiating [8] both sides with respect to x gives:

$$f_x(x,y) = \sum_{\omega_x=-\pi}^{\pi} \sum_{\omega_y=-\pi}^{\pi} -j\omega_x F(\omega_x,\omega_y) e^{-j(\omega_x x+\omega_y y)}. \quad (7.28)$$

That is, in the frequency domain differentiation in the horizontal direction u is equivalent to multiplying the Fourier transform $F(\omega_x,\omega_y)$ by an imaginary horizontal ramp, $-j\omega_x$. Similarly, the vertical partial derivative in v is:

$$f_y(x,y) = \sum_{\omega_x=-\pi}^{\pi} \sum_{\omega_y=-\pi}^{\pi} -j\omega_y F(\omega_x,\omega_y) e^{-j(\omega_x x+\omega_y y)}. \quad (7.29)$$

Now, differentiation in the vertical direction v is equivalent to multiplying the Fourier transform by an imaginary vertical ramp. This trend generalizes to arbitrary directions, that is, the partial derivative in any direction α can be computed by multiplying the Fourier transform by an imaginary oriented ramp $-j\omega_\alpha$:

$$f_\alpha(x,y) = \sum_{\omega_x=-\pi}^{\pi} \sum_{\omega_y=-\pi}^{\pi} -j\omega_\alpha F(\omega_x,\omega_y) e^{-j(\omega_x x+\omega_y y)}, \quad (7.30)$$

where the oriented ramp can be expressed in terms of the horizontal and vertical ramps:

$$\omega_\alpha = \cos(\alpha)\omega_x + \sin(\alpha)\omega_y. \quad (7.31)$$

Substituting this definition back into the partial derivative in α, Equation (7.30), gives:

$$\begin{aligned} f_\alpha(x,y) &= \sum_{\omega_x=-\pi}^{\pi} \sum_{\omega_y=-\pi}^{\pi} -j[\cos(\alpha)\omega_x + \sin(\alpha)\omega_y] F(\omega_x,\omega_y) e^{-j(\omega_x x+\omega_y y)} \\ &= \cos(\alpha) \sum_{\omega_x=-\pi}^{\pi} \sum_{\omega_y=-\pi}^{\pi} -j\omega_x F(\omega_x,\omega_y) e^{-j(\omega_x x+\omega_y y)} \\ &\quad + \sin(\alpha) \sum_{\omega_x=-\pi}^{\pi} \sum_{\omega_y=-\pi}^{\pi} -j\omega_y F(\omega_x,\omega_y) e^{-j(\omega_x x+\omega_y y)} \\ &= \cos(\alpha) f_x(x,y) + \sin(\alpha) f_y(x,y). \end{aligned} \quad (7.32)$$

[8] Recall that the derivative of an exponential is an exponential, so that according to the chain rule, $D_x\{e^{ax}\} = ae^{ax}$.

Notice that we obtain the horizontal and vertical derivatives when $\alpha = 0$ and $\alpha = 90$. This equation embodies the principle of steerability - the derivative in *any* direction α can be synthesized from a linear combination of the partial horizontal and vertical derivatives, $f_x(x, y)$ and $f_y(x, y)$. Pause to appreciate how remarkable this is, a pair of directional derivatives is sufficient to represent an infinite number of other directional derivatives, i.e., α can take on any real-valued number.

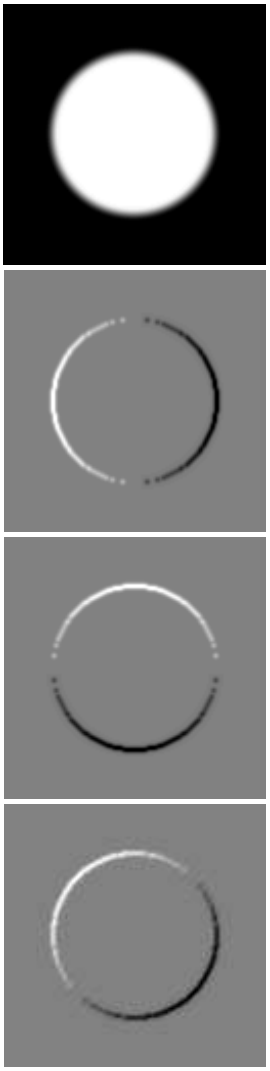


Figure 7.10 Steerability

From the previous section we know how to compute the horizontal and vertical derivatives via convolutions with an interpolation and derivative filter. To compute any other directional derivative no more convolutions are required, simply take the appropriate linear combinations of the horizontal and vertical derivatives as specified in Equation (7.32). Shown in Figure 7.10 from top to bottom is a disc $f(x, y)$, its horizontal derivative $f_x(x, y)$, its vertical derivative $f_y(x, y)$, and its steered derivative $f_{45}(x, y)$, where the steered derivative was synthesized from the appropriate linear combinations of the horizontal and vertical derivatives. The obvious benefit of steerability is that the derivative in any direction can be synthesized with minimal computational costs.

Steerability is not limited to first-order derivatives. Higher-order derivatives are also steerable; the N^{th}-order derivative is steerable with a basis set of size $N + 1$. For example, the second-order derivative in an arbitrary direction can be synthesized as follows:

$$f_{\alpha\alpha} = \cos^2(\alpha) f_{xx} + 2\cos(\alpha)\sin(\alpha) f_{xy} + \sin^2(\alpha) f_{yy}, \qquad (7.33)$$

where for notational simplicity the spatial arguments (x, y) have been dropped, and the multiple subscripts denote higher-order differentiation. Note that three partial derivatives are now needed to steer the second-order derivative. Similarly, the third-order derivative can be steered with a basis of size four:

$$\begin{aligned} f_{\alpha\alpha\alpha} &= \cos^3(\alpha) f_{xxx} + 3\cos^2(\alpha)\sin(\alpha) f_{xxy} \\ &\quad + 3\cos(\alpha)\sin^2(\alpha) f_{xyy} + \sin^3(\alpha) f_{yyy}. \end{aligned} \qquad (7.34)$$

You may have noticed that the coefficients needed to steer the basis set look familiar, they are the binomial coefficients that come from a polynomial expansion. More specifically, as in Equation(7.30) the N^{th}-order derivative in the frequency domain is computed by multiplying the Fourier transform by an imaginary oriented ramp raised to the N^{th} power, $(-j\omega_\alpha)^N$. Expressing this oriented ramp in terms of the horizontal and vertical ramps provides the basis and coefficients needed to steer derivatives of arbitrary order:

$$(\omega_\alpha)^N = (\cos(\alpha)\omega_x + \sin(\alpha)\omega_y)^N. \qquad (7.35)$$

Although presented in the context of derivatives, the principle of steerability is not limited to derivatives. In the most general case, a two-dimensional filter $f(x, y)$ is steerable in orientation if it can be expressed as a polar-separable function, $g(r)h(\theta)$, where $h(\theta)$ is band-limited. More specifically, for an arbitrary radial component $g(r)$, and for $h(\theta)$ expressed as:

$$h(\theta) = \sum_{n=1}^{N} a_n \cos(n\theta) + b_n \sin(n\theta) \tag{7.36}$$

then the filter is steerable with a basis size of $2N$.

7.4 Edge Detection

Discrete differentiation forms the foundation for many applications in computer vision. One such example is *edge detection* - a topic that has received an excessive amount of attention, but is only briefly touched upon here. An edge is loosely defined as an extended region in the image that undergoes a rapid directional change in intensity. Differential techniques are the obvious choice for measuring such changes. A basic edge detector begins by computing first-order spatial derivatives of an image $f[x, y]$:

$$f_x[x, y] = (f[x, y] \star h'[x]) \star h[y] \tag{7.37}$$
$$f_y[x, y] = (f[x, y] \star h[x]) \star h'[y], \tag{7.38}$$

where $h'[\cdot]$ and $h[\cdot]$ are the derivative and prefilter defined in Section 7.2. The "strength" of an edge at each spatial location is defined to be the magnitude of the gradient vector $\bigtriangledown[x, y] = (\, f_x[x, y] \quad f_y[x, y] \,)$, defined as:

$$|\bigtriangledown [x, y]| = \sqrt{f_x^2[x, y] + f_y^2[x, y]}. \tag{7.39}$$

Figure 7.11 Edges

As shown in Figure 7.11, the gradient magnitude is only the beginning of a more involved process (not discussed here) of extracting and localizing the salient and relevant edges.

7.5 Wiener Filter

For any of a number of reasons a digital signal may become corrupted with noise. The introduction of noise into a signal is often modeled as an additive process, $\hat{s} = s+n$. The goal of de-noising is to recover the original signal s from the corrupted signal $\hat{s}$. Given a single constraint in two unknowns this problem is equivalent to my asking you "37 is the sum of two numbers, what are they?" Lacking clairvoyant powers or knowledge of how the individual numbers were selected we have little hope of a solution. But by

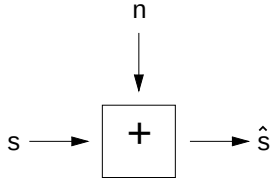


Figure 7.12 Additive noise

making assumptions regarding the signal and noise characteristics and limiting ourselves to a linear approach, a solution can be formulated known as the Wiener filter, of famed Mathematician Norbert Wiener (1894-1964).

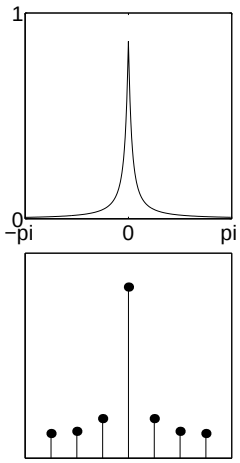


Figure 7.13 Wiener filter

Having restricting ourselves to a linear solution, our goal is to design a filter $h[x]$ such that:

$$\begin{aligned} s[x] &= h[x] \star \hat{s}[x] \\ &= h[x] \star (s[x] + n[x]), \end{aligned} \tag{7.40}$$

that is, when the filter is convolved with the corrupted signal the original signal is recovered. With this as our goal, we reformulate this constraint in the frequency domain and construct a quadratic error functional to be minimized:

$$E(H(\omega)) = \int d\omega \; [H(\omega)(S(\omega) + N(\omega)) - S(\omega)]^2. \tag{7.41}$$

For notational simplicity we drop the frequency parameter ω and express the integral with respect to the expected value operator $\mathcal{E}\{\cdot\}$:

$$\begin{aligned} E(H) &= \mathcal{E}\left\{(H(S+N) - S)^2\right\} \\ &= \mathcal{E}\left\{H^2(S+N)^2 - 2HS(S+N) + S^2\right\} \\ &= \mathcal{E}\left\{H^2(S^2 + 2SN + N^2) - 2H(S^2 + SN) + S^2\right\} \end{aligned} \tag{7.42}$$

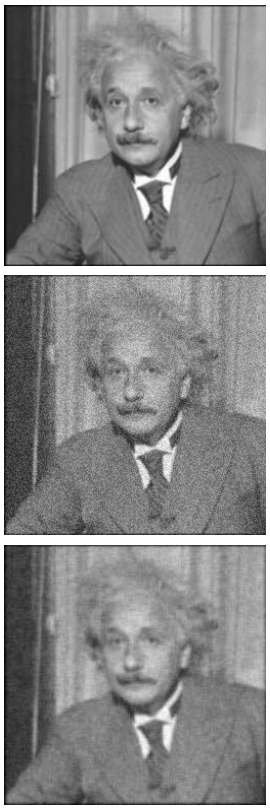


Figure 7.14 Einstein plus noise

In order to simplify this expression we can assume that the signal and noise are statistically independent (i.e., $\mathcal{E}\{SN\} = 0$), yielding:

$$E(H) = \mathcal{E}\left\{H^2(S^2 + N^2) - 2HS^2 + S^2\right\}. \tag{7.43}$$

To minimize, we differentiate:

$$\frac{dE(H)}{dH} = 2H(S^2 + N^2) - 2S^2, \tag{7.44}$$

set equal to zero and solve:

$$H(\omega) = \frac{S^2(\omega)}{S^2(\omega) + N^2(\omega)}. \tag{7.45}$$

At an intuitive level this frequency response makes sense - when the signal is strong and the noise is weak the response is close to 1 (i.e., frequencies are passed), and when the signal is weak and the noise is strong the response is close to 0 (i.e., frequencies are stopped). So we now have an optimal (in the least-squares sense)

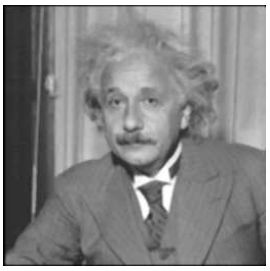
frequency response in terms of the signal and noise characteristics, but of course we don't typically know what those are. But we can instantiate them by making assumptions about the general statistical nature of the signal and noise, for example a common choice is to assume white noise, $N(\omega)$ is constant for all ω, and, for natural images, to assume that $S(\omega) = 1/\omega^p$. The frequency response in the top panel of Figure 7.13 was constructed under these assumptions. Shown in the bottom panel is a 7-tap filter derived from a least-squares design. This one-dimensional formulation can easily be extended to two or more dimensions. Shown in Figure 7.14 from top to bottom, is Einstein, Einstein plus noise, and the results of applying a 7×7 Wiener filter. Note that the noise levels are reduced but that much of the sharp image structure has also been lost, which is an unfortunate but expected side effect given that the Wiener filter is low-pass in nature.

8.1 Median Filter

Noise may be introduced into an image in a number of different ways. In the previous section we talked about how to remove noise that has been introduced in an additive fashion. Here we look at a different noise model, one where a small number of pixels are corrupted due to, for example, a faulty transmission line. The corrupted pixels randomly take on a value of white or black, hence the name *salt and pepper* used to describe such noise patterns (Figure 8.1). Shown in the middle panel of Figure 8.1 is the disastrous result of applying the solution from the additive noise model (Wiener filter) to the salt and pepper noise image in the top panel. Trying to average out the noise in this fashion is equivalent to asking for the average salary of a group of eight graduate students and Bill Gates. As the income of Gates will skew the average salary so does each noise pixel when its value is so disparate from its neighbors. In such cases, the mean is best replaced with the median, computed by sorting the set of numbers and reporting on the value that lies midway. Shown in the bottom panel of Figure 8.1 is the much improved result of applying a 3×3 median filter to the salt and pepper noise image. More specifically, the center pixel of each 3×3 neighborhood is replaced with the median of the nine pixel values in that neighborhood.

Salt & Pepper Noise

Wiener



Median

Figure 8.1 Median filter

Depending on the density of the noise the median filter may need to be computed over a larger neighborhood. The tradeoff being that a larger neighborhood leads to a loss of detail, however this loss of detail is quite distinct from that of averaging. For example, shown in Figure 8.2 is the result of applying a 15×15 median filter. Notice that although many of the internal details have been lost the boundary contours (edges) have been retained, this is often referred to as *posterization*. This effect could never be achieved with an averaging filter which would indiscriminately smooth over all image structures.

Figure 8.2 15×15 *median filter*

Because of the non-linear sorting step of a median filter it cannot be implemented via a simple convolution and is thus often more costly to implement. Outside the scope of this presentation there are a number of tricks for reducing the computational demands of a median filter.

Dithering is a process by which a digital image with a finite number of gray levels is made to appear as a continuous-tone image. For example, shown at the top of Figure 8.3 is an 8-bit (i.e., 256 gray values) grayscale image of Richard P. Feynman. Shown below are two 1 bit (i.e., 2 gray values) images. The first was produced by thresholding, and the second by dithering. Although in both images each pixel takes on only one of two gray values (black or white), it is clear that the final image quality is critically dependent on the way in which pixels are quantized.

Figure 8.3 Thresholding and Dithering

There are numerous dithering algorithms (and even more variants within each algorithm). Sadly there are few quantitative metrics for measuring the performance of these algorithms. A standard and reasonably effective algorithm is a *stochastic error diffusion* algorithm based on the Floyd/Steinberg algorithm. The basic Floyd/Steinberg error diffusion dithering algorithm tries to exploit local image structure to reduce the effects of quantization. For simplicity, a 1-bit version of this algorithm is described here, the algorithm extends naturally to an arbitrary number of gray levels.

This algorithm operates by scanning through the image, left to right and top to bottom. At each pixel, the gray value is first thresholded into "black" or "white", the difference between the new pixel value and the original value is then computed and distributed in a weighted fashion to its neighbors. Typically, the error is distributed to four neighbors with the following weighting:

$$\frac{1}{16} \times \begin{pmatrix} & \bullet & 7 \\ 3 & 5 & 1 \end{pmatrix},$$

where the $\bullet$ represents the thresholded pixel, and the position of the weights represent spatial position on a rectangular sampling lattice. Since this algorithm makes only a single pass through the image, the neighbors receiving a portion of the error must consist only of those pixels not already visited (i.e., the algorithm is casual). Note also that since the weights have unit sum, the error is neither amplified nor reduced. As an example, consider the quantization of an 8-bit image to a 1-bit image. An 8-bit image has 255 gray values, so all pixels less than 128 (mid-level gray) are thresholded to 0 (black), and all values greater than 128 are thresholded to 255 (white). A pixel at position (x, y) with intensity value 120 is thresholded to 0. The error, 120-0 = 120, is distributed to four neighbors as follows: (7/16)120 is added to the pixel at position $(x + 1, y)$, (3/16)120 is added to the pixel at

position $(x-1, y+1)$, (5/16)120 to pixel $(x, y+1)$ and (1/16)120 to pixel $(x+1, y+1)$. The intuition behind this algorithm is that when the pixel value of 120 is thresholded to 0 that pixel is made darker. By propagating this error the surrounding pixels are brightened making it more likely that they will be thresholded to something brighter. As a result the local neighborhood maintains it's average gray value.

Qualitatively, the error diffusion algorithm reduces the effects of quantization via simple thresholding. However this algorithm does introduce correlated artifacts due to the deterministic nature of the algorithm and scanning order. These problems may be partially alleviated by introducing a stochastic process in a variety of places. Two possibilities include randomizing the error before distributing it to its neighbors (e.g., randomly scaling the error in the range 0.9 to 1.1), and alternating the scanning direction (e.g., odd lines are scanned left to right, and even lines from right to left).

10.1 Differential Motion

Our visual world is inherently dynamic. People, cars, dogs, etc. are (usually) moving. These may be gross motions, walking across the room, or smaller motions, scratching behind your ear. Our task is to estimate such image motions from two or more images taken at different instances in time.

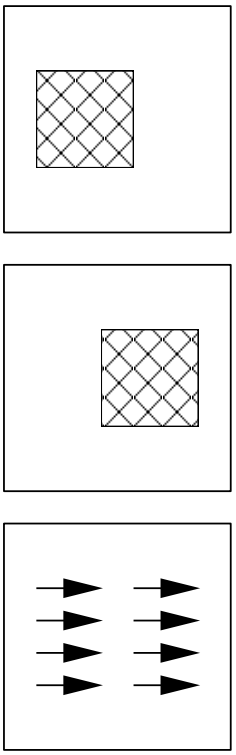


Figure 10.1 Flow field

With respect to notation, an image is denoted as $f(x, y)$ and an image sequence is denoted as $f(x(t), y(t), t)$, where $x(t)$ and $y(t)$ are the spatial parameters and t is the temporal parameter. For example, a sequence of N images taken in rapid succession may be represented as $f(x(t), y(t), t+i\Delta t)$ with $i \in [0, N-1]$, and Δt representing the amount of time between image capture (typically on the order of 1/30th of a second). Given such an image sequence, our task is to estimate the amount of motion at each point in the image. For a given instant in space and time, we require an estimate of motion (*velocity*) $\vec{v} = (\, v_x \quad v_y \,)$, where v_x and v_y denote the horizontal and vertical components of the velocity vector $\vec{v}$. Shown in Figure 10.1 are a pair of images taken at two moments in time as a textured square is translating uniformly across the image. Also shown is the corresponding estimate of motion often referred to as a *flow field*. The flow field consists of a velocity vector at each point in the image (shown of course are only a subset of these vectors).

In order to estimate motion, an assumption of *brightness constancy* is made. That is, it is assumed that as a small surface patch is moving, its brightness value remains unchanged. This constraint can be expressed with the following partial differential equation:

$$\frac{\partial f(x(t), y(t), t)}{\partial t} \;=\; 0. \tag{10.1}$$

This constraint holds for each point in space and time. Expanding this constraint according to the chain rule yields:

$$\frac{\partial f}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial t} + \frac{\partial f}{\partial t} \;=\; 0, \tag{10.2}$$

where the partials of the spatial parameters x and y with respect to time correspond to the velocity components:

$$f_x v_x + f_y v_y + f_t \;=\; 0. \tag{10.3}$$

The subscripts on the function f denote partial derivatives. Note again that this constraint holds for each point in space and time but that for notational simplicity the spatial/temporal parameters are dropped. This transformed brightness constancy constraint is rewritten by packing together the partial derivatives and velocity components into row and column vectors.

$$\begin{aligned} \begin{pmatrix} f_x & f_y \end{pmatrix} \begin{pmatrix} v_x \\ v_y \end{pmatrix} + f_t &= 0 \\ \vec{f}_s^t \vec{v} + f_t &= 0. \end{aligned} \tag{10.4}$$

The space/time derivatives $\vec{f}_s$ and f_t are measured quantities, leaving us with a single constraint in two unknowns (the two components of the velocity vector, $\vec{v}$). The constraint can be solved by assuming that the motion is locally similar, and integrating this constraint over a local image neighborhood. A least-squares error function takes the form:

$$E(\vec{v}) = \left[\sum_{x,y} \vec{f}_s^t \vec{v} + \sum_{x,y} f_t \right]^2, \tag{10.5}$$

To solve for the motion this error function is first differentiated

$$\begin{aligned} \frac{\partial E(\vec{v})}{\partial \vec{v}} &= 2 \sum \vec{f}_s \left[\sum \vec{f}_s^t \vec{v} + \sum f_t \right] \\ &= 2 \sum \vec{f}_s \vec{f}_s^t \vec{v} + 2 \sum \vec{f}_s f_t. \end{aligned} \tag{10.6}$$

Setting equal to zero and recombining the terms into matrix form yields:

$$\begin{aligned} \begin{pmatrix} \sum f_x \\ \sum f_y \end{pmatrix} \begin{pmatrix} \sum f_x & \sum f_y \end{pmatrix} \begin{pmatrix} v_x \\ v_y \end{pmatrix} &= -\begin{pmatrix} \sum f_x f_t \\ \sum f_y f_t \end{pmatrix} \\ \begin{pmatrix} \sum f_x^2 & \sum f_x f_y \\ \sum f_x f_y & \sum f_y^2 \end{pmatrix} \begin{pmatrix} v_x \\ v_y \end{pmatrix} &= -\begin{pmatrix} \sum f_x f_t \\ \sum f_y f_t \end{pmatrix} \\ M \vec{v} &= -\vec{b}. \end{aligned} \tag{10.7}$$

If the matrix M is invertible (full rank), then the velocity can be estimated by simply left multiplying by the inverse matrix:

$$\vec{v} = -M^{-1} \vec{b} \tag{10.8}$$

The critical question then is, when is the matrix M invertible? Generally speaking the matrix is rank deficient, and hence not invertible, when the intensity variation in a local image neighborhood varies only one-dimensionally (e.g., $f_x = 0$ or $f_y = 0$) or zero-dimensionally ($f_x = 0$ and $f_y = 0$). These singularities are sometimes referred to as the aperture and blank wall problem. The motion at such points simply can not be estimated.

Motion estimation then reduces to computing, for each point in space and time, the spatial/temporal derivatives f_x, f_y, and f_t. Of course the temporal derivative requires a minimum of two images, and is typically estimated from between two and seven images. The spatial/temporal derivatives are computed as follows. Given a temporal sequence of N images, the spatial derivatives are computed by first creating a temporally prefiltered image. The spatial derivative in the horizontal direction f_x is estimated by prefiltering this image in the vertical y direction and differentiating in x. Similarly, the spatial derivative in the vertical direction f_y is estimated by prefiltering in the horizontal x direction and differentiating in y. Finally, the temporal derivative is estimated by temporally differentiating the original N images, and prefiltering the result in both the x and y directions. The choice of filters depends on the image sequence length: an N tap pre/derivative filter pair is used for an image sequence of length N (See Section 7).

10.2 Differential Stereo

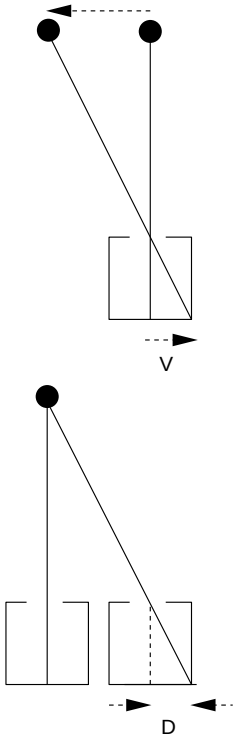


Figure 10.2 Motion and Stereo

Motion estimation involves determining, from a single stationary camera, how much an object moves over time (its velocity). Stereo estimation involves determining the displacement *disparity* of a stationary object as it is imaged onto a pair of spatially offset cameras. As illustrated in Figure 10.2, these problems are virtually identical: velocity ($\vec{v}$) $\equiv$ disparity (Δ). Motion and stereo estimation are often considered as separate problems. Motion is thought of in a continuous (differential) framework, while stereo, with its discrete pair of images, is thought of in terms of a discrete matching problem. This dichotomy is unnecessary: stereo estimation can be cast within a differential framework.

Stereo estimation typically involves a pair of cameras spatially offset in the horizontal direction such that their optical axis remain parallel (Figure 10.2). Denoting an image as $f(x, y)$, the image that is formed by translating the camera in a purely horizontal direction is given by $f(x + \Delta(x, y), y)$. If a point in the world (X, Y, Z) is imaged to the image position (x, y), then the shift $\Delta(x, y)$ is inversely proportional to the distance Z (i.e., nearby objects have large disparities, relative to distant objects). Given this, a *stereo pair* of images is denoted as:

$$f_L(x + \delta(x, y), y) \qquad \text{and} \qquad f_R(x - \delta(x, y), y), \tag{10.9}$$

where the disparity $\Delta = 2\delta$. Our task is to determine, for each point in the image, the disparity (δ) between the left and right images. That is, to find the shift that brings the stereo pair back

into register. To this end, we write a quadratic error function to be minimized:

$$E(\delta(x,y)) = [f_L(x+\delta(x,y),y) - f_R(x-\delta(x,y),y)]^2. \quad (10.10)$$

In this form, solving for δ is non-trivial. We may simplify things by expressing the image pair in terms of their truncated first-order Taylor series expansion:

$$f(x+\delta(x,y),y) \;=\; f(x,y) + \delta(x,y) f_x(x,y), \quad (10.11)$$

where $f_x(x,y)$ denotes the partial derivative of f with respect to x. With this first-order approximation, the error function to be minimized takes the form:

$$\begin{aligned} E(\delta) &= [(f_L + \delta(f_L)_x) - (f_R - \delta(f_R)_x)]^2 \\ &= [(f_L - f_R) + \delta(f_L + f_R)_x]^2, \end{aligned} \quad (10.12)$$

where for notational convenience, the spatial parameters have been dropped. Differentiating, setting the result equal to zero and solving for δ yields:

$$\begin{aligned} \frac{dE(\delta)}{d\delta} &= 2(f_L + f_R)_x[(f_L - f_R) + \delta(f_L + f_R)_x] \\ &= 0 \\ \delta &= -\frac{f_L - f_R}{(f_L + f_R)_x} \end{aligned} \quad (10.13)$$

Stereo estimation then reduces to computing, for each point in the image, spatial derivatives and the difference between the left and right stereo pair (a crude derivative with respect to viewpoint).

Why, if motion and stereo estimation are similar, do the mathematical formulations look so different? Upon closer inspection they are in fact quite similar. The above formulation amounts to a constrained version of motion estimation. In particular, because of the strictly horizontal shift of the camera pair, the disparity was constrained along the horizontal direction. If we reconsider the motion estimation formulation assuming motion only along the horizontal direction, then the similarity of the formulations becomes evident. Recall that in motion estimation the brightness constancy assumption led to the following constraint:

$$\frac{\partial f}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial t} + \frac{\partial f}{\partial t} \;=\; 0, \quad (10.14)$$

Constraining the motion along the vertical y direction to be zero yields:

$$\frac{\partial f}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial f}{\partial t} \;=\; 0, \quad (10.15)$$

where the partial derivative of the spatial parameter x with respect to time correspond to the motion (speed) in the horizontal direction:

$$f_x v_x + f_t = 0. \tag{10.16}$$

Unlike before, this leads to a single constraint with a single unknown which can be solved for directly:

$$v_x = -\frac{f_t}{f_x}. \tag{10.17}$$

This solution now looks very similar to the solution for differential stereo in Equation 10.13. In both solutions the numerator is a derivative, in one case with respect to time (motion) and in the other with respect to viewpoint (stereo). Also in both solutions, the denominator is a spatial derivative. In the stereo case, the denominator consists of the spatial derivative of the sum of the left and right image pair. This may seem odd, but recall that differentiation of a multi-dimensional function requires differentiating along the desired dimension and prefiltering along all other dimensions (in this case the viewpoint dimension).

In both the differential motion and stereo formulations there exists singularities when the denominator (spatial derivative) is zero. As with the earlier motion estimation this can be partially alleviated by integrating the disparities over a local image neighborhood. However, if the spatial derivative is zero over a large area, corresponding to a surface in the world with no texture, then disparities at these points simply can not be estimated.

11.1 Expectation/Maximization

The *Expectation/Maximization (EM)* algorithm simultaneously segments and fits data generated from multiple parametric models. For example, shown in Figure 11.1 are a collection of data points (x, y) generated from one of two linear models of the form:

$$y(i) = a_1 x(i) + b_1 + n_1(i) \quad \text{or} \quad y(i) = a_2 x(i) + b_2 + n_2(i), \tag{11.1}$$

where the model parameters are a_1, b_1 and a_2, b_2, and the system is modeled with additive noise $n_1(i)$ and $n_2(i)$.

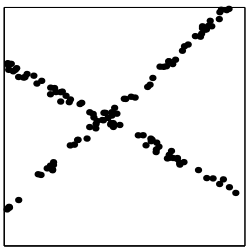


Figure 11.1 Data from two models

If we are told the model parameters, then determining which data point was generated by which model would be a simple matter of choosing, for each data point i, the model k that minimizes the error between the data and the model prediction:

$$r_k(i) \quad = \quad |a_k x(i) + b_k - y(i))|, \tag{11.2}$$

for $k = 1, 2$ in our current example. On the other hand, if we are told which data points were generated by which model, then estimating the model parameters reduces to solving, for each model k, an over-constrained set of linear equations:

$$\begin{pmatrix} x_k(1) & 1 \\ x_k(2) & 1 \\ \vdots & \vdots \\ x_k(n) & 1 \end{pmatrix} \begin{pmatrix} a_k \\ b_k \end{pmatrix} \quad = \quad \begin{pmatrix} y_k(1) \\ y_k(2) \\ \vdots \\ y_k(n) \end{pmatrix}, \tag{11.3}$$

where the $x_k(i)$ and $y_k(i)$ all belong to model k. In either case, knowing one piece of information (the model assignment or parameters) makes determining the other relatively easy. But, lacking either piece of information makes this a considerably more difficult estimation problem. The EM algorithm is an iterative two step algorithm that estimates both the model assignment and parameters.

The "E-step" of EM assumes that the model parameters are known (initially, the model parameters can be assigned random values) and calculates the likelihood of each data point belonging to each model. In so doing the model assignment is made in a "soft" probabilistic fashion. That is, each data point is not explicitly assigned a single model, instead each data point i is assigned a

probability of it belonging to each model k. For each model the *residual error* is first computed as:

$$r_k(i) = a_k x(i) + b_k - y(i) \tag{11.4}$$

from which the likelihoods are calculated. We ask, what is the likelihood of point i belonging to model k given the residual error. For our two model example:

$$\begin{aligned} P(a_k, b_k | r_k(i)) &= \frac{P(r_k(i)|a_k, b_k)P(a_k, b_k)}{P(r_k(i))} \\ &= \frac{P(r_k(i)|a_k, b_k)}{P(r_1(i)|a_k, b_k) + P(r_2(i)|a_k, b_k)}, \end{aligned} \tag{11.5}$$

for $k = 1, 2$. The expansion of the conditional probability is from Bayes rule: $P(B|A_k) = \frac{P(A_k|B)P(B)}{\sum_l P(A_l|B)P(B)}$. If we assume a Gaussian probability distribution, then the likelihood takes the form:

$$w_k(i) = \frac{e^{-r_k(i)^2/\sigma_N}}{e^{-r_1(i)^2/\sigma_N} + e^{-r_2(i)^2/\sigma_N}} \tag{11.6}$$

where, σ_N is proportional to the amount of noise in the data, and for each data point i, $\sum_k w_k(i) = 1$.

The "M-step" of EM takes the likelihood of each data point belonging to each model, and re-estimates the model parameters using weighted least-squares. That is, the following *weighted* error function on the model parameters is minimized:

$$E_k(a_k, b_k) = \sum_i w_k(i)[a_k x(i) + b_k - y(i)]^2. \tag{11.7}$$

The intuition here is that each data point contributes to the estimation of each model's parameters in proportion to the belief that it belongs to that particular model. This quadratic error function is minimized by computing the partial derivatives with respect to the model parameters, setting the result equal to zero and solving for the model parameters. Differentiating:

$$\begin{aligned} \frac{\partial E_k(a_k, b_k)}{\partial a_k} &= \sum_i 2w_k(i)x(i)[a_k x(i) + b_k - y(i)] \\ \frac{\partial E_k(a_k, b_k)}{\partial b_k} &= \sum_i 2w_k(i)[a_k x(i) + b_k - y(i)], \end{aligned} \tag{11.8}$$

setting both equal to zero yields the following set of linear equations:

$$a_k \sum_i w_k(i)x(i)^2 + b_k \sum_i w_k(i)x(i) = \sum_i w_k(i)x(i)y(i) \tag{11.9}$$

$$a_k \sum_i w_k(i)x(i) + b_k \sum_i w_k(i) = \sum_i w_k(i)y(i). \tag{11.10}$$

Rewriting in matrix form:

$$\begin{pmatrix} \sum_i w_k(i)x(i)^2 & \sum_i w_k(i)x(i) \\ \sum_i w_k(i)x(i) & \sum_i w_k(i) \end{pmatrix} \begin{pmatrix} a_k \\ b_k \end{pmatrix} = \begin{pmatrix} \sum_i w_k(i)x(i)y(i) \\ \sum_i w(i)y(i) \end{pmatrix}$$

$$A\vec{x_k} = \vec{b}$$

$$\vec{x_k} = A^{-1}\vec{b}, \qquad (11.11)$$

yields a weighted least squares solution for the model parameters. Note that this solution is identical to solving the set of linear equations in Equation (11.3) using weighted least-squares.

The EM algorithm iteratively executes the "E" and "M" step, repeatedly estimating and refining the model assignments and parameters. Shown in Figure 11.2 are several iterations of EM applied to fitting data generated from two linear models. Initially, the model parameters are randomly assigned, and after six iterations, the algorithm converges to a solution. Beyond the current scope are proofs on the convergence and rate of convergence of EM. At the end of this chapter is a MATLAB implementation of EM for fitting multiple linear models to two-dimensional data.

11.2 Principal Component Analysis

11.3 Independent Component Analysis

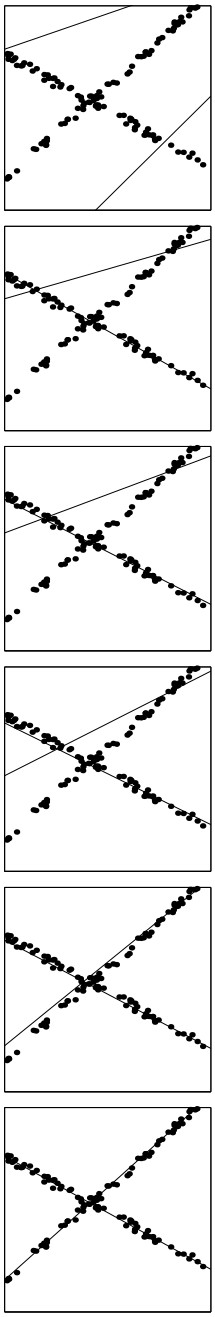


Figure 11.2 Six iterations of EM

```
%%% THE EM ALGORITHM
clear;
NumModel  = 2;
NumData   = 64;
Sigma     = 0.1; % IN E-STEP
Noise     = 0.1; % IN DATA
NIterate  = 10;

%%% MAKE SYNTHETIC DATA
A     = 2*rand(1,NumModel) - 1;
B     = 2*rand(1,NumModel) - 1;
X     = [];
Y     = [];
for i = 1 : NumModel
     x = 2*rand(1,NumData) - 1;
     y = A(i) * x + B(i) + Noise*(rand(1,NumData)-0.5);
     X = [X x];
     Y = [Y y];
end

%%% INITIALIZE MODEL
a     = 2*rand(1,NumModel) - 1;
b     = 2*rand(1,NumModel) - 1;

for j = 1 : NIterate
     %%% E-STEP
     for i = 1 : NumModel
          r(i,:) = a(i)*X + b(i) - Y;
     end
     den     = sum( exp( -(r.^2)/Sigma^2 ) );
     for i = 1 : NumModel
          w(i,:)     = exp( (-(r(i,:).^2)/Sigma^2) ) ./ (den+eps);
     end

     %%% M-STEP
     for i = 1 : NumModel
          M = [ sum(w(i,:).*X.^2) sum(w(i,:).*X) ; sum(w(i,:).*X) sum(w(i,:)) ];
          V = [ sum(w(i,:).*X.*Y) ; sum(w(i,:).*Y) ];
          U = pinv(M) * V;
          a(i) = U(1);
          b(i) = U(2);
     end

     %%% SHOW MODEL FIT AND ASSIGNMENT
     xc     = [-1 : 0.1 : 1];
     subplot(2,1,1); cla; hold on;
     plot( X, Y, 'bo' );
     for i = 1 : NumModel
          yc = a(i)*xc + b(i);
          plot( xc, yc, 'r' );
     end
     hold off;
     subplot(2,1,2); cla; imagesc(w); colormap gray; pause(1);
end
```