Mathematical Methods for Biology, Part 2

Dmitry Kondrashov

Table of contents

Preface

In this book you will find a collection of mathematical ideas, computational methods, and modeling tools for describing biological systems quantitatively. Biological science, like all natural sciences, is driven by experimental results. As with other sciences, there comes a point when accumulated data needs to be analyzed quantitatively, in order to formulate and test explanatory hypotheses. Biology has reached this stage, thanks to an explosion of data from molecular biology techniques, such as large-scale DNA sequencing, protein structure determination, data on gene regulatory networks, and signaling pathways. Quantitative skills have become necessary for anyone hoping to make sense of biological research.

Mathematical modeling necessarily involves making simplifying assumptions. Reality is generally too complex to be captured in a few equations, and this is especially true for living systems. Simplicity in modeling has at least two virtues: first, simple models can be grasped by our limited minds, and second, it allows for meaningful testing of the assumptions against the evidence. A complex model that fits the data may not provide any insights about how the system works, whereas a simple model which does not fit all the data can indicate where the assumptions break down.

1 Principal Component Analysis

Principal Component Analysis (PCA) is one of the most popular techniques to perform "dimensionality reduction" of complex data sets. If we see the data with many variables as points in a high-dimensional space, we can compute new variables as linear combinations of the original ones and represent each data point as a set of coordinates in the new variables. In this way, we can project large-dimensional data sets onto low-dimensional spaces and lose the least information about the data.

1.1 Motivation: simplifying complex data

Suppose we have a data set with n variables and m observations of each (typically, with $n \gg m$), in which the m rows are observations and the n columns are the variables. Each row of this matrix defines a point in the Euclidean space $\mathbb{R}^n$. Many biological data sets, e.g. gene expression {numref}`fig-micro-array`, RNAseq, medical imaging, can contain thousands or more variables, which poses major challenges both for visualization and computational tasks. PCA provides the best representation of such a data set in terms of a smaller set of variables, while capturing as much variance as possible.

The intuition behind finding these new collective variables rests on the fact that the original variables have relationships. This is typically measured using covariance, which quantified how much a pair variables tends to move in the same direction (positive covariance) or in opposite directions (negative covariance). If two variables are tighlty coupled, one can replace the two measurements with one, which will describe how much the two of them are deviating in some collective way.

It is helpful to think of this geometrically: if the variables are related, the scatterplot of observed data points will have a shape. The goal of PCA is to find directions in the n-dimensional space of observations that best match the shape of the data cloud. This requires tools from linear algebra, specifically the technique of change of basis.

1.2 Linearity and vector spaces

We have dealt with linear models in various guises, so now would be a good time to define properly what linearity means. The word comes from the shape of graphs of linear functions of

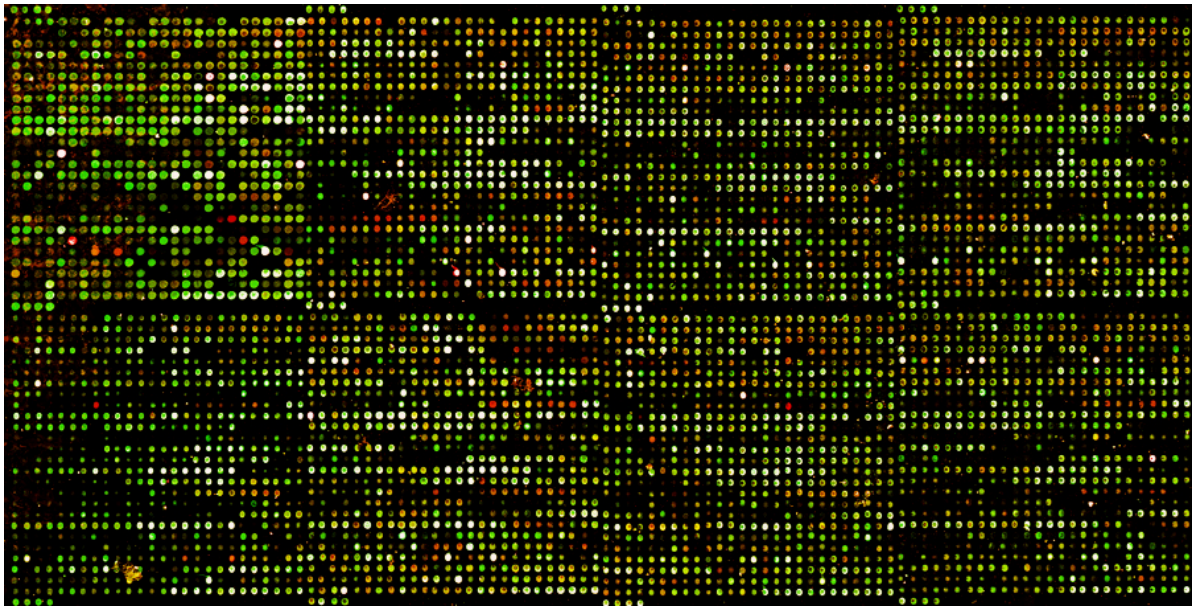


Figure 1.1: Image of a microarray plate, (http://exploreable.files.wordpress.com/2011/04/array.jpg). Here each dot is a different variable (different gene) and this image in just one set of observations that will be placed into a row of the data matrix.

one variable, e.g. $f(x) = ax + b$, but the algebraic meaning rests on the following two general properties:

Definition

A *linear transformation* or *linear operator* is a mapping L between two sets of vectors with the following properties:

1. *(scalar multiplication)* $L(c\vec{v}) = cL(\vec{v})$; where c is a scalar and $\vec{v}$ is a vector;
2. *(additive)* $L(\vec{v}_1 + \vec{v}_2) = L(\vec{v}_1) + L(\vec{v}_2)$; where $\vec{v}_1$ and $\vec{v}_2$ are vectors.

Here we have two types of objects: vectors and transformations/operators that act on those vectors. The basic example of this are vectors and matrices, because a matrix multiplied by a vector (on the right) results another vector, provided the number of columns in the matrix is the same as the number of rows in the vector. This can be interpreted as the matrix transforming the vector $\vec{v}$ into another one: $A \times \vec{v} = \vec{u}$.

Example: Let us multiply the following matrix and vector (specially chosen to make a point):

```
import numpy as np #package for work with arrays and matrices
import matplotlib.pyplot as plt #package with plotting capabilities
import seaborn as sns; sns.set() # colors for visualization
```

```
Mat = np.array([[2, 1],[2, 3]])
vec1 = np.array([1, -1])
vec2 = Mat@vec1
print(vec1)
```

```
[ 1 -1]
```

```
print(vec2)
```

```
[ 1 -1]
```

We see that this particular vector $(1, -1)$ is unchanged when multiplied by this matrix, or we can say that the matrix multiplication is equivalent to multiplication by 1. Here is another such vector for the same matrix:

```
vec1 = np.array([1, 2])
vec2 = Mat@vec1
print(vec1)
```

```
[1 2]
```

```
print(vec2)
```

```
[4 8]
```

In this case, the vector is changed, but only by multiplication by a constant (4). Thus the geometric direction of the vector remained unchanged.

The notion of linearity leads to the important idea of combining different vectors:

Definition

A *linear combination* of n vectors $\{\vec{v}_i\}$ is a weighted sum of these vectors with any real numbers $\{a_i\}$:

$$a_1\vec{v}_1 + a_2\vec{v}_2 ... + a_n\vec{v}_n$$

Linear combinations arise naturally from the notion of linearity, combining the additive property and the scalar multiplication property. Speaking intuitively, a linear combination of vectors produces a new vector that is related to the original set. Linear combinations give a simple way of generating new vectors, and thus invite the following definition for a collection of vectors closed under linear combinations:

Definition

A *vector space* is a collection of vectors such that a linear combination of any n vectors is contained in the vector space.

The most common examples are the spaces of all real-valued vectors of dimension n, which are denoted by $\mathbb{R}^n$. For instance, $\mathbb{R}^2$ (pronounced "r two") is the vector space of two dimensional real-valued vectors such as $(1, 3)$ and $(\pi, -\sqrt{17})$; similarly, $\mathbb{R}^3$ is the vector space consisting of three dimensional real-valued vectors such as $(0.1, 0, -5.6)$. You can convince yourself, by taking linear combinations of vectors, that these vector spaces contain all the points in the usual Euclidean plane and three-dimensional space. The real number line can also be thought of as the vector space $\mathbb{R}^1$.

1.2.1 Linear independence and basis vectors

How can we describe a vector space without trying to list all of its elements? We know that one can generate an element by taking linear combinations of vectors. It turns out that it is possible to generate (or "span") a vector space by taking linear combinations of a subset of its vectors. The challenge is to find a minimal subset of subset that is not redundant. In order to do this, we first introduce a new concept:

Definition

A set of vectors $\{\vec{v}_i\}$ is called *linearly independent* if the only linear combination involving them that equals the zero vector is if all the coefficients are zero. ($a_1\vec{v}_1+a_2\vec{v}_2+...+a_n\vec{v}_n = 0$ only if $a_i = 0$ for all i.)

In the familiar Euclidean spaces, e.g. $\mathbb{R}^2$, linear independence has a geometric meaning: two vectors are linearly independent if the segments from the origin to the endpoint do not lie on the same line. But it can be shown that any set of three vectors in the plane is linearly

dependent, because there are only two dimensions in the vector space. This brings us to the key definition of this section:

> **Definition**
>
> A *basis* of a vector space is a linearly independent set of vectors that generate (or span) the vector space. The number of vectors (cardinality) in such a set is called the *dimension* of the vector space.

A vector space generally has many possible bases, as illustrated in figure. In the case of $\mathbb{R}^2$, the usual (canonical) basis set is $\{(1,0);(0,1)\}$ which obviously generates any point on the plane and is linearly independent. But any two linearly independent vectors can generate any vector in the plane.

Example: The vector $\vec{r} = (2,1)$ can be represented as a linear combination of the two canonical vectors: $\vec{r} = 2\times(1,0)+1\times(0,1)$. Let us choose another basis set, say $\{(1,1);(-1,1)\}$ (this is the canonical basis vectors rotated by $\pi/2$.) The same vector can be represented by a linear combination of these two vectors, with coefficients 1.5 and -0.5: $\vec{r} = 1.5\times(1,1) - 0.5\times(-1,1)$. If we call the first basis C for canonical and the second basis D for different, we can write the same vector using different sets of coordinates for each basis:

$$\vec{r}_C = (2,1);\ \vec{r}_D = (1.5,-0.5)$$

1.2.2 Projections and changes of basis

The representation of an arbitrary vector (point) in a vector space as a linear combination of a given basis set is called the *decomposition* of the point in terms of the basis, which gives the coordinates for the vector in terms of each basis vector. The decomposition of a point in terms of a particular basis is very useful in high-dimensional spaces, where a clever choice of a basis can allow a description of a set of points (such as a data set) in terms of contributions of only a few basis vectors, if the data set primarily extends only in a few dimensions.

To obtain the coefficients of the basis vectors in a decomposition of a vector $\vec{r}$, we need to perform what is termed a *projection* of the vector onto the basis vectors. Think of shining a light perpendicular to the basis vector, and measuring the length of the shadow cast by the vector $\vec{r}$ onto $\vec{v}_i$. If the vectors are parallel, the shadow is equal to the length of $\vec{r}$; if they are orthogonal, the shadow is nonexistent. To find the length of the shadow, use the inner product of $\vec{r}$ and $\vec{v}$, which as you recall corresponds to the cosine of the angle between the two vectors multiplied by their norms: $\langle\vec{r},\vec{v}\rangle = |\vec{r}||\vec{v}|\cos(\theta)$. We do not care about the length of the vector $\vec{v}$ we are projecting onto, thus we divide the inner product by the square norm of $\vec{v}$, and then multiply the vector $\vec{v}$ by this projection coefficient:

$$Proj(\vec{r};\vec{v}) = \frac{\langle \vec{r},\vec{v}\rangle}{\langle \vec{v},\vec{v}\rangle}\vec{v} = \frac{\langle \vec{r},\vec{v}\rangle}{|\vec{v}|^2}\vec{v} = \frac{|\vec{r}|\cos(\theta)}{|\vec{v}|}\vec{v}$$

This formula gives the projection of the vector $\vec{r}$ onto $\vec{v}$, the result is a new vector in the direction of $\vec{v}$, with the scalar coefficient $a = \langle \vec{r},\vec{v}\rangle/|\vec{v}|^2$.

Example: Here is how one might calculate the projection of the point $(2, 1)$ onto the basis set $\{(1, 1); (-1, 1)\}$:

```
v1 <- c(1, 1)
v2 <- c(-1, 1)
u <- c(2, 1)
ProjMat <- matrix(cbind(v1, v2),
                  byrow = T, nrow = 2)
print(ProjMat)
```

```
     [,1] [,2]
[1,]    1    1
[2,]   -1    1
```

```
ProjMat %*% u
```

```
     [,1]
[1,]    3
[2,]   -1
```

This is not quite right: the projection coefficients are off by a factor of two compared to the correct values in the example above. This is because we have neglected to *normalize* the basis vectors, so we should modify the script as follows:

```
v1 <- c(1, 1)
v1 <- v1 / (sum(v1^2))
v2 <- c(-1, 1)
v2 <- v2 / (sum(v2^2))
u <- c(2, 1)
ProjMat <- matrix(cbind(v1, v2),
                  byrow = T, nrow = 2)
print(ProjMat)
```

```
     [,1] [,2]
[1,]  0.5  0.5
[2,] -0.5  0.5
```

```
print(ProjMat %*% u)
```

```
     [,1]
[1,]  1.5
[2,] -0.5
```

This is an example of how to convert a vector/point from representation in one basis set to another. The new basis vectors, expressed in the original basis set, are arranged in a matrix by row, scaled by their norm squared, and multiplied by the vector that one wants to express in the new basis. The resulting vector contains the coordinates in the new basis.

1.3 PCA algorithm

We start with a data set X in the form of a m by n matrix. The first step is to decide which are the variables and which are the observations. For example, in the case of the microarray experiment, it usually makes sense to consider different genes the variables, and to use principal components to see which genes tend to be expressed together with others.

The second step is to compute the variance-covariance matrix of the N variables.

Definition

The *variance-covariance* matrix C of a data set X with n variables x_i and m observations is an n by n matrix that contains pairwise variances between all n variables, so that its i, j element is:

$$C_{i,j} = Cov(X_i, X_j)$$

The third step is to diagonalize (find the eigenvalues and eigenvectors) of the covariance matrix C. The eigenvectors are the principal components of the n variables in the data set, representing linear combinations of the variables that best fit the data. Diagonalizing an n by n matrix results in n eigenvectors, so in order to simplify the description one needs to choose the most significant ones. This is accomplished by choosing a subset of k principal components with the largest eigenvalues. Here are the steps of principal component analysis (PCA):

PCA algorithm

1. Obtain a dataset as a m by n matrix, with n variables and m observations
2. Compute covariances for variable i and variable j, put them in the covariance matrix C
3. Compute the eigenvalues and eigenvectors (principal components) of the matrix C
4. Order the principal component by size of eigenvalues from largest to smallest and select a few as the new coordinates

1.4 Optimization by explained variance

The reason that we order the PCs by their eigenvalues is that they measure the amount of variance captured by each principal component. In that, they are equivalent to the coefficient of determination r^2 in linear regression. The sum of all the eigenvalues is equal to the total variance of all the variables:

$$\sum_i \lambda_i = \sum Var(X_i)$$

and the fraction of variance captured by the a principal component is:

$$Var(PC_i) = \frac{\lambda_i}{\sum_i \lambda_i}$$

The theory behind this rests on some relatively sophisticated linear algebra, in particular what is called the singular value decomposition (SVD) and the Eckart-Young Mirsky theorem. Here is a nice video by Gilbert Strang that explains this:Strang lecture

1.5 Dimensionality reduction

After sorting the principal components and selecting k largest eigenvalues, we are ready to simplify the data. This means that we can express a data set of n variables in terms of the coordinate set of k principal components. In order to express the data set in this new system of coordinates, we compute the projection coefficients for each measurement onto a give principal component. Suppose that Y is a set of measurements of N variables (e.g. genes) and P_i is the i-the principal component. Then the projection coefficient of Y onto P_i is the dot product of the two vectors (both of length N) divided by the squared norm (length) of the PC:

$$c_i = \frac{\langle Y, P_i \rangle}{||P_i||^2}$$

If the eigenvectors are normalized prior to the computation (as they are by most computational packages), then the projection coefficient is just the dot product. Then the coefficients can be obtained for all of the measurements in the data set X (m by n) by multiplying it by the matrix P containing the first k eigenvectors (principal components), which has n rows and k columns. The result is an m by k matrix C containing k coefficients for each of the m measurements:

$$C = D \times P$$

Here is the outline for the transformation:

Dimensionality reduction

1. Subtract the mean of each observation from the data matrix (if it has M observations in rows and N variables as columns, subtract the mean of each row from it)
2. Compute the projection coefficients $C = D \times P$ for each measurement and each of the k principal components
3. Plot or otherwise display these coefficients as coordinates in the new vector system of the k PCs. This can be used to cluster or otherwise find patterns in the observations.

The entire data set can be expressed in a low-dimensional setting, for instance plotted in the plane of two principal components with coordinates $(c_{i,1}, c_{i,2})$ for each data measurement i. This is often useful for clustering, or grouping experimental conditions based on the similarity of their principal component representations. Biologists frequently do this with complex data sets, for example grouping different cell lines together by their gene expression profiles.

2 Optimization using gradients

2.1 Overview

One often has to find the maximum or minimum of a function that describes an important biological property. Examples include:

- Free energy of a protein as function of its conformation (the folding problem)
- Evolutionary fitness as a function of the genome
- Optimizing the dose of therapeutic radiation or chemotherapy, to affect the maximal fraction of tumor cells and the minimal number of healthy cells

In all these cases, there is the same fundamental problem: given a complex function of many variables, find the values of the variables which optimize the function, that is, for which it attains its maximum or minimum, depending on the question.

Some terminology: f is known as the *objective function*, and the quantities the function depends on $(x_1, x_2, ..., x_n)$ are the governing variables.

It is important to note the notion of a minimum or a maximum is local: it is a point which is lowest or highest in some neighborhood. But very often we are interested in a global optimum - the best possible solution to our problem. The first problem is not too difficult, while the second is practically impossible for a complex function, because, aside from the areas explored by minimization, there is no way to guarantee that there is no better optimum elsewhere. This is the tragic state of optimizers everywhere - they have to live with the uncertainty.

2.2 Optimization with one variable

2.2.1 golden section search

Let us first consider searching for the minimum (or maximum, the problems are really the same except for a minus sign) of a function of one variable $f(x)$. The goal is to locate the minimum value of the function within a certain interval on the x-axis, to a desired tolerance. We assume only that we can evaluate the function at any given value of x. Here is the outline of the algorithm:

Golden section search

1. start with three points x_1, x_2, x_3 ($x_1 < x_2 < x_3$), such that $f(x_2) < f(x_1)$ and $f(x_2) < f(x_3)$
2. choose a point x_4 in the larger of the intervals (x_1, x_2) or (x_2, x_3) (let's assume it's (x_1, x_2), the process is the same for the other case)
3. if $f(x_4) > f(x_2)$, replace x_1 with x_4, the new triplet is (x_4, x_2, x_3)
4. if $f(x_4) < f(x_2)$, replace x_3 with x_2, the new triplet is (x_1, x_4, x_2)
5. repeat until the width of the interval is smaller than your tolerance

The question is, what is the optimal way to pick the new point x_4? Since we have no information about the shape of the function $f(x)$, we don't know where the minimum is more likely to be hiding. The best approach is to hedge your bets and make each option (3 or 4) result in the same interval scaled by the same factor.

Assume that we pick x_4 in (x_1, x_2), as the larger interval, and we want to pick x_4 so that the length is reduced by the same factor in both cases: (x_4, x_2, x_3) and (x_1, x_4, x_2) and the resulting triplet is also divided in the same proportion. This is the condition for the golden ratio (that the ratio of the larger segment to the whole is the same as the ratio of the smaller segment to the larger). Thus, the the choice for the new point is $x_4 = x_1 + (x_3 - x_1)/\phi$, where $\phi = (1 + \sqrt{5})/2$. Similarly, if the interval (x_2, x_3) is larger, we pick $x_4 = x_3 - (x_3 - x_1)/\phi$.

Notice that this means that the interval bracketing the minimum shrinks by a factor of $\phi \approx 1.61$ every step.

2.2.2 bisection method

The bisection method is applicable for a continuous function $f(x)$ which has a computable derivative function $F(x)$. In that case, finding a maximum or minimum of f(x) is the same as finding a root of the derivative function F(x). The bisection method simply divides the bracketing interval that contains the maximum or minimum into two, like this:

Golden section search

1. start with an interval (a, b) with $F(a)$ and $F(b)$ of opposite signs
2. choose $c = (b - a)/2$
3. pick the side on which the two endpoints are bracketing zero, so the new interval is either (a, c) or (c, b)
4. repeat until the interval width is smaller than the tolerance

In this case, the bracketing interval is always decreased by a factor of two, which is faster than the golden section search.

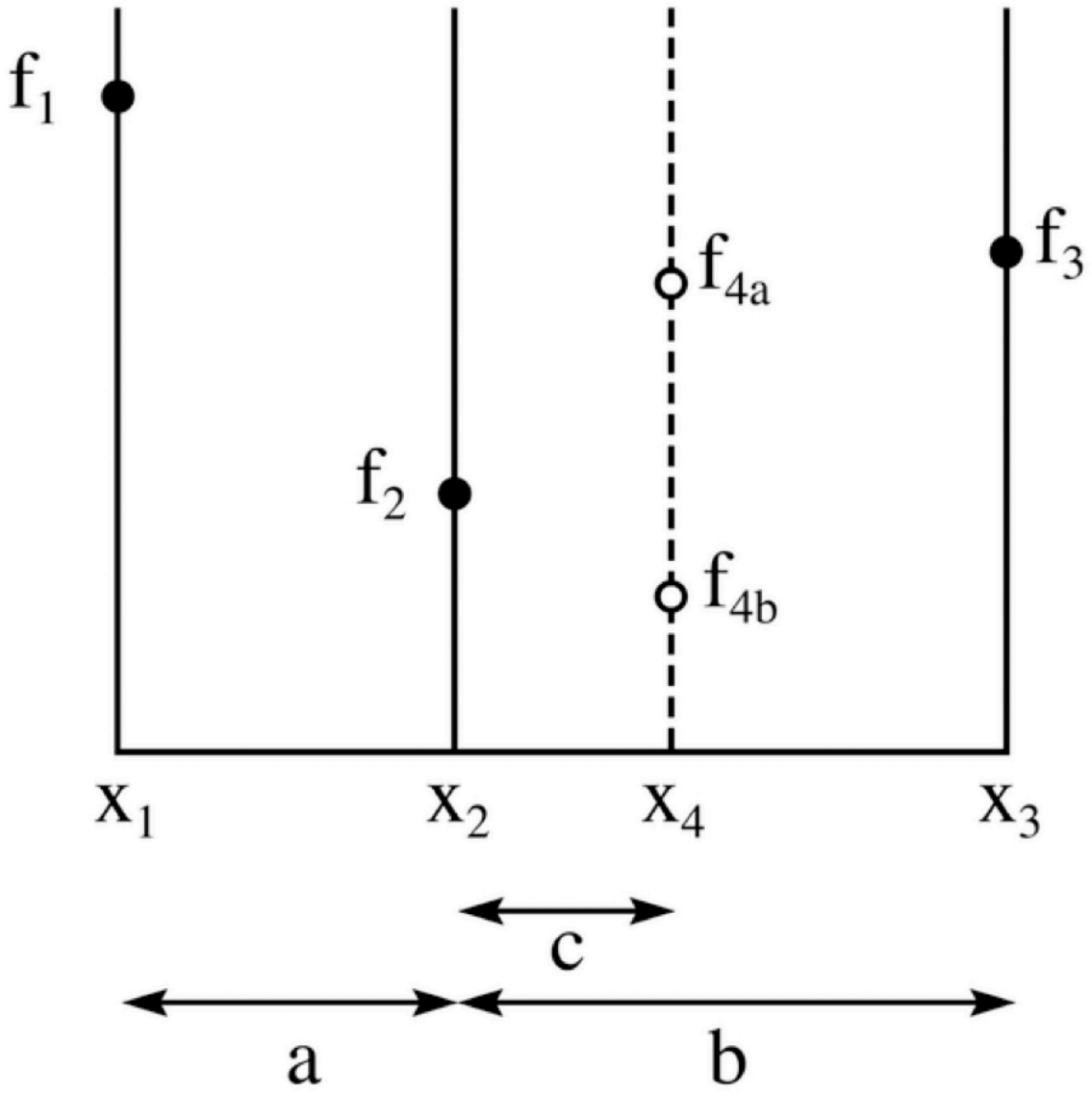


Figure 2.1: Golden ratio method for a 1-variable function f(x); Figure source

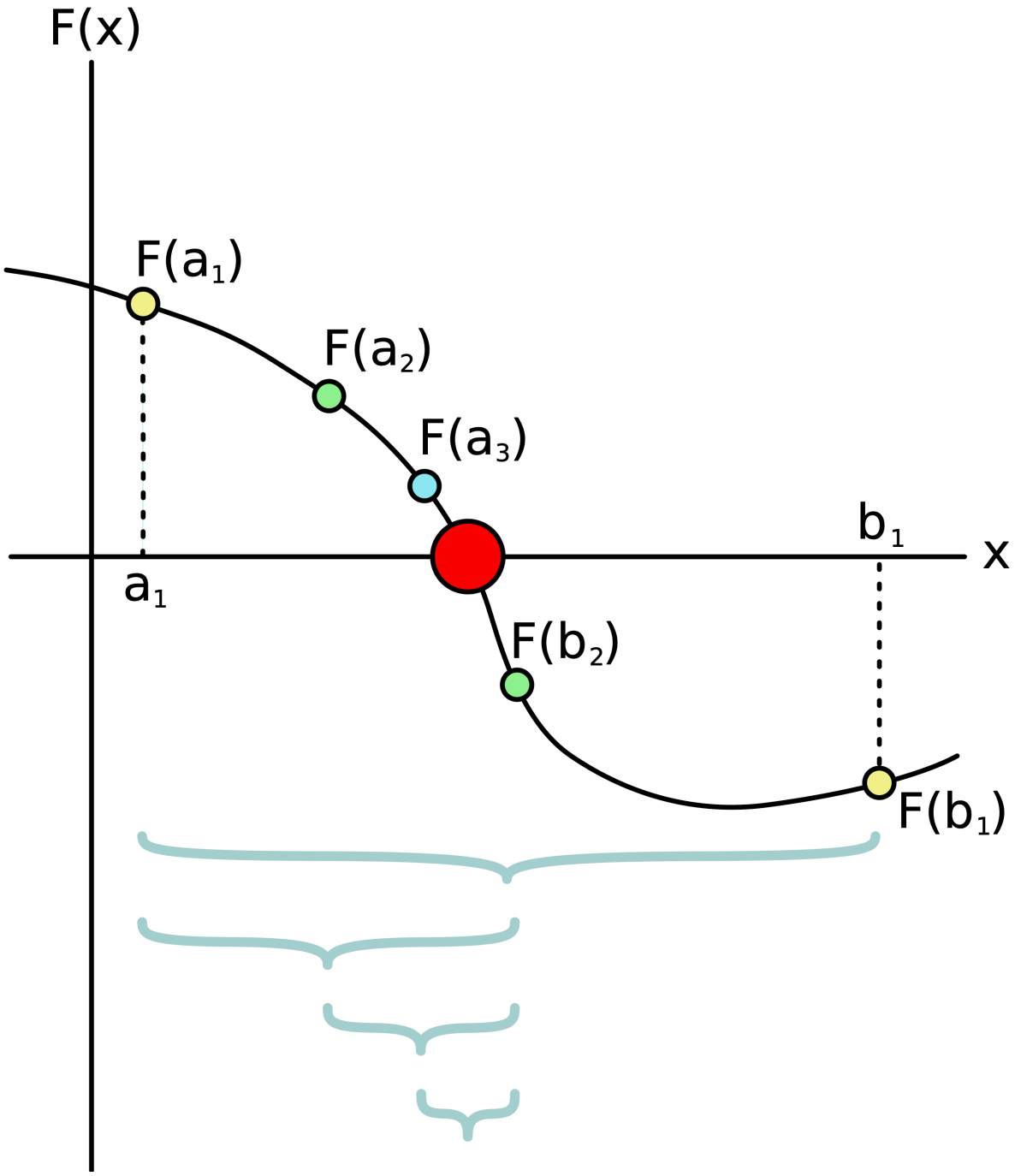


Figure 2.2: Bisection method involves cutting the interval bracketing a zero in two; Figure source

2.2.3 secant method

Secant method is also a root-finding method, and can thus be used to find optima of functions with a computable derivative. If the derivative function is f(x), then the algorithm is:

> **Secant algorithm**
>
> 1. start with a bracketing interval (x_0, x_1)
> 2. calculate the slope of the (secant) line that connects the two points $a = \frac{f(x_1)-f(x_0)}{x_1-x_0}$
> 3. pick the next value $x_i = x_{i-1} - af(x_{i-1})$, which is where the secant line crosses 0
> 4. repeat until $f(x_i)$ is close enough to zero

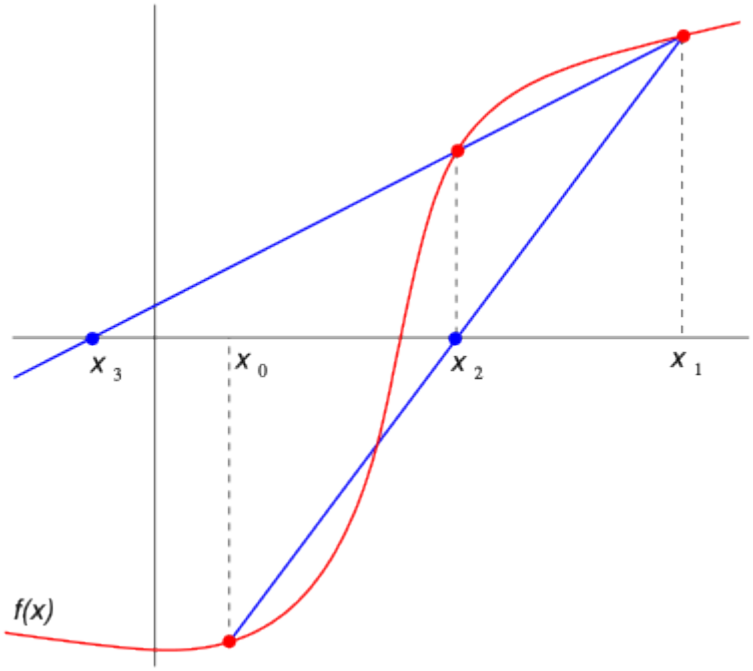


Figure 2.3: The secant method uses the line connecting the two bracketing points to approximate the root; Figure source

The secant method is faster to converge than bisection for most functions.

2.2.4 Newton-Raphson method

We will now consider an old method for finding roots of functions, devised by the Great Newton Himself, is also useful for finding maxima and minima of functions for which an exact formula is given. To use for optimization, it requires not only the knowledge of the derivative function

$f(x)$ (as for secant and bisection method) but also the knowledge of the second derivative function $f'(x)$ of the function we want to optimize.

The idea is based on the first-order Taylor expansion of a function near a point x_0: $f(x_0+\Delta x) = f(x_0)+\Delta x f'(x_0)+....$ The dots indicate higher-order terms that we will ignore, for sufficiently small Δx. Now, suppose the function has a root (zero) at x_0, so $f(x_0) = 0$. Then, if we are at a point near the root, $x_0 + \Delta x$, we can calculate how far away it is from the root, by simply solving for $\Delta x = f(x_0 + \Delta x)/f'(x_0)$. The idea of Newton's method is to find the step size needed to reach the root, by using the value of the derivative at the current point $x_0 + \Delta x$ (assuming the points are close enough that the slopes are almost equal). Then the steps for the algorithm are:

Newton-Raphson algorithm in 1 dimension

1. Start at some point x_0 (not the root, as above)
2. Let $x_{i+1} = x_i - f(x_i)/f'(x_i)$
3. Repeat, until $|x_{i+1} - x_i| < \epsilon_{tol}$, where ϵ_{tol} is the specified tolerance

The method turns out to be very efficient. Its only limitation is that it cannot find a root for which the derivative is zero (graphically, a root at which the graph of the function only touches the x-axis). Another complication is that, if the starting point is not close enough to a root, it can be hard to predict which root the method will find, but once it settles in near one, it will reach it quickly. If the value of the derivative $f'(x_i)$ is small, then the method can bounce around, sometimes almost chaotically, so its efficiency strongly depends on starting in proximity to a root.

To be used for optimization of a function $F(x)$, we need to be able to calculate $F'(x) = f(x)$ and $F''(x) = f'(x)$ and the find the root(s) of $f(x)$. Then one has to check whether the extremum is a max or a min, which is straightforward.

2.3 Optimization of multivariable functions using derivatives

Let us now consider functions of multiple variables, which is the case for most real applications. Such, a function, e.g. $f(x, y)$, takes in values of the variables x and y, and returns a single number. Graphically, this could be plotted as a surface, with the height at any pair of coordinates (x, y) given by $f(x, y)$. Intuitively, the goal of optimization is to find the lowest (or highest) point on the surface, at least in a neighborhood (see above discussion about difficulties of global optimization). Let us assume that we are searching for a minimum, as the story is completely equivalent for maximization.

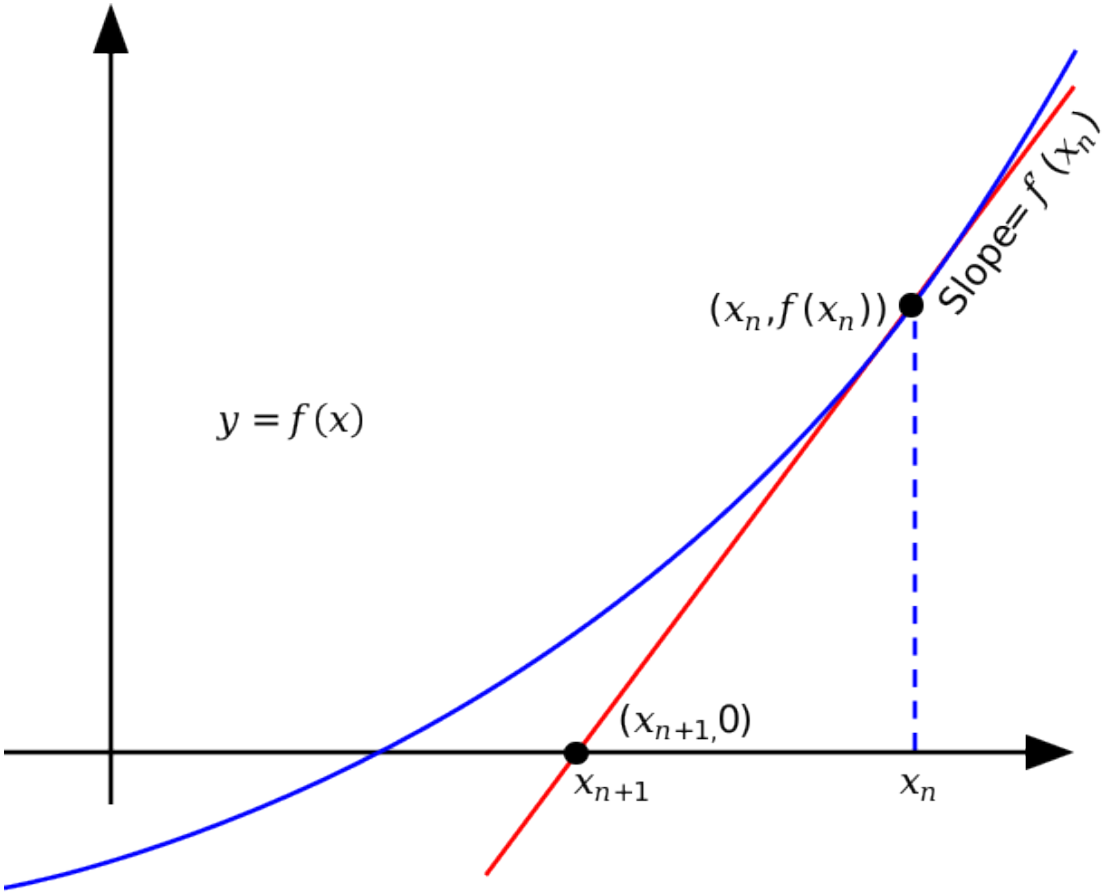


Figure 2.4: Newton-Raphson method for finding roots uses the derivative of the function to find the intersection of the tangent line with the x-axis; Figure source

2.3.1 multivariable optimization problem

A function of more than one variable is denoted by $f(\vec{x})$, where $\vec{x}$ is a vector of n variables $x_1, x_2, ..., x_n$. This means the function takes in a vector of several numbers and returns a scalar - a single number. One simple visual analogy is the function that gives elevation (height) for a given latitude and longitude (x, y). By plotting that function we would produce the shape of a mountain range or any other terrain, with the appropriate height at each point in the $x - y$ plane.

The basic condition for finding the maximum or minimum of a function is known from basic calculus. For a function of n variables $f(x_1, x_2, ..., x_n)$ to be at an optimum, all partial derivatives must vanish:

$$\frac{\partial f(x_1, x_2, ..., x_n)}{\partial x_1} = \frac{\partial f(x_1, x_2, ..., x_n)}{\partial x_2} = ... = \frac{\partial f(x_1, x_2, ..., x_n)}{\partial x_n} = 0$$

This gives a set of n equations to be solved for n unknowns. However, in practice optimization problems are rarely solved this way, because solving n nonlinear equations can be difficult, particularly for complicated functions that frequently arise in biology. However, one can use iterative methods to converge to the optimum using derivatives of the objective function.

2.3.2 gradient and contours

In order to describe the changes in multivariable functions, we need more than one number. Consider the slope on the surface of a mountain: it may be steep in one direction, and flat in another. In order to deal with this, *partial derivatives* are used. These represent the rate of change of f with respect to x and y separately. Geometrically, this means the slope of the landscape we visualized above, if sliced in the x or y direction.

Example: If $f(x, y) = x^2 - 2x + y^2 + 4x + 5$

$$\frac{\partial f}{\partial x} = 2x - 2 \frac{\partial f}{\partial y} = 2y + 4$$

This allows us to define the multi-dimensional equivalent of the derivative: the *gradient* of a function,

> **i** Definition
>
> The *gradient* of a function $f(\vec{x})$ of multiple variables $\vec{x} = x_1, .., x_n$ is a vector with n components, each one the partial derivative with respect to the corresponding variable:

$$\nabla f(\vec{x}) = (\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, ..., \frac{\partial f}{\partial x_n})$$

To gain some geometric intuition about gradients, let us introduce the notion of contours of $f(\vec{x})$

Definition

The *contours* or *level curves* of a function $f(\vec{x})$ of multiple variables $\vec{x} = x_1, .., x_n$ are sets of values of $\vec{x}$ that satisfy equations, for any given constant c:

$$f(\vec{x}) = c$$

These are curves (for two-variable functions) or surfaces (for three-variable functions) in the space of the variables on which the function is equal to a particular constant. One common example of this are contours of a landscape (e.g. on the Earth's surface) on a topographical map.

There is an important relationship between gradients and contours. Since there there is no change in f as one travels along it, the derivative of the function in the direction of the contour curve at any point is 0. As a consequence, the direction of the fastest change of f at any point, ∇f is orthogonal to the level curve at that point.

Example: Let us find the contours (level curves) for the function we gave above, which means the solution of the equation $f(x, y) = (x - 1)^2 + (y + 2)^2 = c$. These are circles centered at $(1, -2)$, with the radius depending on the value of c.

Let us compare the direction of the gradient at a point, let us say at $(0, -2)$. This point is horizontally to the left of the center of the circular level curves. Thus, the tangent line to the circle at the point is vertical. The gradient is $\nabla f = (-2, 0)$: a vertical vector, perpendicular to the direction of the circle at that point.

2.3.3 gradient descent

The simplest idea for finding the lowest point in a valley is to follow the slope downward. In multiple dimensions, the direction of greatest downward change (steepest descent) is given by the gradient, which leads to the *gradient descent* algorithm.

The first method for minimization of a multidimensional function simply takes steps in the direction of the gradient, until the point is sufficiently close to the bottom of the valley. This is called the method of *gradient descent*:

Gradient descent algorithm

1. Start at some point $\vec{x}_0$.
2. Compute the gradient at the current point $\vec{x}_i$, $\nabla f(\vec{x}_i)$. Find the minimum of the function f along that direction, which may be done through a one-dimensional search.
3. Take the next point to be $x_{i+1} = x_i + \alpha \nabla f(\vec{x}_i)$, where α is the multiple found by the one-dimensional search.
4. Repeat until $||\vec{x}_{i+1} - \vec{x}_i|| < \epsilon_{tol}$, where ϵ_{tol} is the specified tolerance

The gradient descent method, although intuitively simple, is in practice inefficient. Specifically, it has difficulties with descending into a long, narrow valley: it takes many short, zigzagging steps on the way to the lowest point. Relying on the steepest way down is not the fastest way of reaching the minimum, although it is guaranteed to converge eventually, if it takes small steps.

2.3.4 Newton-Raphson method

A different approach at finding the minimum of a multi-variable function is to extend the Newton-Raphson method from one variable to multiple. The idea is once again to convert the optimization problem for $f(\vec{x})$ into a root-finding problem for $\nabla f(\vec{x})$ where the gradient function plays the role of the derivative in the 1-dimensional Newton's method. In order for this to work, we need an equivalent to the second derivative as well.

Definition

For a function $f(\vec{x})$ of a vector variable $\vec{x}$ with n components, the *Hessian* is a n by n matrix, with each element defined as the second partial with respect to two variables:

$$H_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}$$

Example: For the same function we saw $f(x, y) = x^2 - 2x + y^2 + 4x + 5$, with the gradient:

$$\frac{\partial f}{\partial x} = 2x - 2 \frac{\partial f}{\partial y} = 2y + 4$$

The Hessian requires computing two more partials for each of the elements of the gradient:

$$\frac{\partial^2 f}{\partial x^2} = 2 \frac{\partial^2 f}{\partial x \partial y} = 0 \frac{\partial^2 f}{\partial y \partial x} = 0 \frac{\partial^2 f}{\partial y^2} = 2$$

So the Hessian matrix is

$$H(f(\vec{x})) = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$$

Now that we have the multi-variable equivalents of $f'(x)$ and $f''(x)$, we can formulate the multidimensional Newton-Raphson algorithm using vectors and matrices:

Multi-variable Newton-Raphson algorithm

1. Define the gradient function $\nabla f(\vec{x})$ and the Hessian function $H(\vec{x})$
2. Start at some point $\vec{x}_0$ (as long as it is not the minimum)
3. Update the point using$\vec{x}_{i+1} = \vec{x}_i - H^{-1}(\vec{x}_i)\nabla f(\vec{x}_i)$
4. Repeat until $||\vec{x}_{i+1} - \vec{x}_i|| < \epsilon_{tol}$, where ϵ_{tol} is the specified tolerance

The geometric intuition of the Newton-Raphson method relies on the fact that the second derivative (Hessian) matrix H represents the curvature of the objective function, similar to the coefficient a of the 1-variable quadratic function $ax^2 + bx + c$. Instead of simply using the gradient, or the linear component of the objective function, this method uses the curvature to speed up the search. In fact, Newton-Raphson performs very efficiently once it is within a nearly quadratic-shaped well, but doesn't do very well far from the minimum.

2.3.5 Levenberg-Marquardt method

The Levenberg-Marquardt method is a combination of the gradient descent and Newton-Raphson methods. It uses their respective strengths, by using the gradient to quickly reach the vicinity (approximately quadratic valley) of a minimum, and then use the curvature information from the Hessian to finish the job efficiently. The insight of this method is that one can interpolate between the two methods by introducing a parameter λ which at high values weigh the method toward gradient descent, and at small values the Hessian and therefore Newton's method dominates. Here is an outline of the algorithm, which starts with a large value of λ, assuming that the initial guess $\vec{x}_0$ is far from the minimum:

Levenberg-Marquardt algorithm

1. Define the gradient function $\nabla f(\vec{x})$ and the Hessian function $H(\vec{x})$
2. Start at some point $\vec{x}_0$ (as long as it is not the minimum) and set λ to be large (e.g. 1000)
3. Update the point using $\vec{x}_{i+1} = \vec{x}_i - (H(\vec{x}_i) + \lambda diag[H(\vec{x}_i)])^{-1} \nabla f(\vec{x}_i)$
4. If $f(\vec{x}_{i+1}) < f(\vec{x}_i)$ (the new point is an improvement), accept $\vec{x}_{i+1}$ and divide λ by a factor S (which can be set to be any positive real number, usually in the range of 2-10)
5. else (if the point is not an improvement) keep $\vec{x}_i$ as the current point and multiply λ by S
6. Repeat until $||\vec{x}_{i+1} - \vec{x}_i|| < \epsilon_{tol}$, where ϵ_{tol} is the specified tolerance

When $\lambda = 0$, the updating expression is identical to the Newton-Raphson step. One unusual insight in the method is the use of the diagonal of the Hessian instead of just the identity matrix, which is in line with the gradient descent updating step. It turns out that using the diagonal of the Hessian matrix (which contains the second derivatives w.r.t. to all the variables) guarantees more efficient convergence than using just the identity matrix by using the geometry of the curvature to scale the gradient accordingly. This method is widely used in nonlinear least squares fitting problems, such as fitting data to exponential or other complex functions.

3 Fourier transforms and convolutions

3.1 Overview

In this chapter we will build on the concepts of Fourier analysis introduced in the last quarter. We will outline the theory of Fourier transforms and give examples of pairs of functions and their transforms. In the process, new theoretical constructs, such as the delta-function, will come into play. This will lead to the most important tool in the applications of Fourier transforms, convolution of two functions. I will give examples from imaging and crystallography, where Fourier transforms of data, described as convolutions, are indispensable.

3.2 Continuous Fourier Transform

3.2.1 Fourier integral

So far we have only used the idea of frequency decomposition on functions which are either periodic (repeating) with period L, or equivalently, if we are only interested in the function over that interval. We may want to similarly analyze a continuous function which is not periodic, and in which we are interested over the whole real line. To achieve that, we generalize the notion of a Fourier series from a sum to an integral:

Definition

Forward Fourier transform:

$$G(f) = \int_{-\infty}^{\infty} g(x) e^{i2\pi f x} dx$$

Inverse Fourier transform:

$$g(x) = \int_{-\infty}^{\infty} G(f) e^{-i2\pi f x} df$$

The first equation is called the *Fourier transform*, note that it changes the variable from x (e.g. time or space) to f (frequency). It is equivalent to the integral definition of c_n, except that n was restricted to the integers, and f can be any real number. This is why, for the second

equation, defining the *Inverse Fourier transform*, the Fourier series becomes an integral: we have to add up the contributions of all real frequencies.

A note on notation: first, the signs of the frequencies in the definition of the Fourier series and the Fourier integral are reversed. I believe this is purely a matter of convention. Second, mathematicians prefer to define the Fourier transform and the inverse transform as follows:

> **i** (Alternative) definition
>
> Forward Fourier transform:
>
> $$G(\omega) = \int_{-\infty}^{\infty} g(x) e^{i\omega x} dx$$
>
> Inverse Fourier Transform:
>
> $$g(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} G(\omega) e^{-i\omega x} d\omega$$

The difference between the two definitions is small, and conceptually can be distinguished by noting that the variable f in the first definition refers to a pure frequency (times per unit of x) while the variable ω in the second definition refers to angular frequency (radians per unit of x). The textbook uses the second definition, but we will stick with the first, because it uses pure frequency f and is more consistent with the definition of the Fourier series.

3.2.2 the delta function

As we saw above, the units of the variables x and f are reciprocal to each other (e.g. second and 1/second, or meter and 1/meter), and although the functions $g(x)$ and $\hat{g}(f)$ are in some sense the same function, they have an inverse relationship with each other. The Fourier transform and the inverse Fourier transform take functions between the two domains, and thus set up pairs of "inversely" related functions. In some cases, they are simply to compute and understand analytically.

For instance, if we take the pure wave $\cos(2\pi x)$, it clearly contains only the frequencies ± 1 (plus or minus because both positive and negative frequencies are inherent in any pure wave). You would expect its Fourier transform representation to be a double peak (symmetric about the origin) in the frequency space f, while everywhere else it is strictly zero. If we're dealing with Fourier series, of course, we just have two coefficients $c_1 = 1/2$ and $c_{N-1} = 1/2$, with all the others being zero. However, a single value has no weight on the real line (mathematically speaking, it has "measure zero", which means the same thing). So, mathematicians borrowed from physicists (Paul Dirac) the idea of an infinitely tall and infinitely thin spike which has finite area under the curve. This is called the delta function, and is written $\delta(x - a)$, where a is the value at which it spikes. The Fourier transform of the pure cosine wave is then described as:

$$\frac{1}{2}\delta(f-1)+\frac{1}{2}\delta(f+1)$$

As a corollary, if we take the boring function $f(x)=1$, which can be thought of as a wave of zero frequency, its Fourier transform is given by simply $\delta(f)$ - that is, an infinite spike at frequency zero. Since $f(x)$ has an infinite mean and nothing else, this makes intuitive sense.

Conversely, the Fourier transform of a delta function $\delta(x-a)$ is given by:

$$\int_{-\infty}^{\infty}\delta(x-a)e^{i2\pi xf}dx=e^{i2\pi af}=\cos(2\pi af)+i\sin(2\pi af)$$

This uses the property of the delta function to "pick out" the value of the function it is integrated with, at the point matching its spike (in this case a).

3.2.3 Fourier pairs and the Gaussian function

A function and its Fourier transformed counterpart are know as *Fourier pairs*. What we see is that functions which are widely distributed in one domain (like the constant function or the sines or cosines) have Fourier transforms which are infinitely thin, and vice versa. This a manifestation of the inverse property of the two domains, as we vaguely saw above. There is a general property of Fourier pairs that the wider a function, the narrower its Fourier counterpart. We have investigated the extremes, let us see where the happy medium lies.

The Gaussian function $h(x)=e^{-ax^2}$ is a continuous bell-shaped curve which falls off sharply from its peak at zero, and has a finite area under the curve. It has the special property that its functional form is preserved by the Fourier transform: the Fourier partner of a Gaussian is also a Gaussian. Here is the verification:

$$\int_{-\infty}^{\infty}e^{-ax^2}e^{i2\pi xf}dx=\int_{-\infty}^{\infty}e^{[-ax^2+i2\pi xf+\frac{\pi^2}{a}f^2]-\frac{\pi^2}{a}f^2}dx=e^{-\frac{\pi^2}{a}f^2}\int_{-\infty}^{\infty}e^{-a(x-i\frac{\pi}{a}f)^2}dx=e^{-\frac{\pi^2}{a}f^2}\sqrt{\frac{\pi}{a}}$$

In the first step we did a trick called completing the square in the exponent, by adding the term π/af^2 to make the first part of the expression be equal to $(\sqrt{a}x-i\frac{\pi}{\sqrt{a}}f)^2=a(x-i\frac{\pi}{a}f)^2$, and subtracting the same term. Then we split up the exponential into two factors, of which one ($e^{-\frac{\pi}{a}f^2}$) is independent of the integration variable x, and can thus be brought out of the integral. What remains inside the integral is just a Gaussian function shifted over by a constant ($-i\frac{\sqrt{\pi}}{a}$). You can make a change of variables $y=x-i\frac{\sqrt{\pi}}{a}$, and because we are integrating over all values of x, the shift does not matter, the integral still contains all the area under the Gaussian curve. Finally, we know that the total area under a Gaussian $ e^{-ax2}$ equals $\sqrt{\frac{\pi}{a}}$, hence the final step.

Thus, the functions $h(x) = e^{-ax^2}$ and $\hat{h}(f) = \sqrt{\frac{\pi}{a}} e^{-\frac{\pi^2}{a} f^2}$ are a Fourier pair. If we choose $a = \pi$, you see that the exponentials of the two functions match, and the constant in $\hat{h}(f)$ reduces to unity. The Gaussian function $ e^{- x2}$ is its own Fourier transform, not too wide, not too narrow - the Goldilocks function.

3.3 Convolution and Fourier Transform

We now come to one more special property of the Fourier Transform which is useful in many different applications.

Definition

The *convolution* of two functions is defined as follows:

$$h \circ g = \int_{-\infty}^{\infty} h(x) g(y - x) dx$$

$g(y - x)$ is the function $g(y)$ horizontally translated by x. Thus, you can think about this integral as follows: for every number y, we sum over all the values of $h(x)$ multiplied by the value of $g(y - x)$, which is effectively shifted by x.

Some important properties of the convolution, which can be easily shown by writing down the integral definition:

Properties of convolutions

- convolution of two functions is commutative: $h \circ g = g \circ h$
- convolution of two functions is associative: $h \circ (g \circ f) = (h \circ g) \circ f$
- convolution of two functions is distributive: $h \circ (g + f) = h \circ f + h \circ f$
- convolution preserves shifts (translations): if $f = h \circ g$, then $f(x-a) = h(x-a) \circ g = h \circ g(x - a)$

3.3.1 Examples

One simple example is based on the delta function $\delta(x - a)$ we saw in the previous lecture, which, if you recall picks out the value of the function it is integrated with that matches its spike at a. Thus, if we consider:

$$h(x) \circ \delta(x - a) = \int_{-\infty}^{\infty} h(x) \delta(y - x - a) dx = h(y - a)$$

That is, the delta function performed a shift, or repositioning of the function h by its own peak position a. This property is often used in convolution applications, as we will see below.

On the other extreme, we will consider a convolution with the opposite function: one which is infinitely wide. The convolution of a function with one which is unity everywhere, $(g(x) = 1)$ results in the mean value of the function $h(x)$:

$$h(x) \circ g(x) = \int_{-\infty}^{\infty} h(x)dx$$

3.3.2 The Convolution Theorem

The reason why the convolution is so central to the application of Fourier Transforms is the following.

> **Convolution Theorem**
>
> The Fourier transform has the following special property with convolutions:
>
> $$\widehat{g \circ h} = \hat{g}\hat{h}$$

This means that the Fourier transform of a convolution of two function is the same as the product of the two separate FT. This enables a huge computational benefit for computing Fourier transforms of signals which are naturally convolved, because it is usually much easier to compute the FT of the two individual functions.

There are many physical applications where convolutions arise naturally, for instance in communications, where one function can be thought of as a signal and the other the response, and the convolution given the observed communication. It is also invaluable in imaging, where one can get a repetition of different objects by defining one function to describe the object, and the other to describe the distribution of its locations. Convolution in this case gives the overall image with multiple objects, or with objects whose location is not precise, but has a smeared distribution.

3.4 Computational: Fourier transform in two dimensions, spectral analysis

3.4.1 power spectrum in two dimensions

As we saw above, Fourier transforms can be computed exactly for some simple functions. In practice computational methods like the Fast Fourier Transform are used to find a numerical

approximation of the Fourier transform. This means that only a discrete set of values of $\hat{h}(f)$ are computed. We will talk about this in the next lecture.

Here I want to focus on what we take away from Fourier transform analysis. Essentially, looking at $\hat{h}(f)$ gives us a frequency-based representation of the original function $h(x)$. We can see what types of frequencies are prevalent, which are absent, etc. To quantify this, however, we need to be more precise. The function $\hat{h}(f)$ is complex-valued, so we cannot even graph it in one dimension (Prove to yourself that: FT of an even function is purely real-valued, like the Gaussian example above, FT of an odd function is purely imaginary, and FT of mixed functions are complex, like the delta-function example above). Instead it is customary to consider the absolute value squared $|\hat{h}(f)|^2$, which is called the *power* of each frequency f.

This is convenient because the Parseval equality applies to Fourier integrals as well:

$$\int_{-\infty}^{\infty} |h(x)|^2 dx = \int_{-\infty}^{\infty} |\hat{h}(f)|^2 df$$

Therefore, we can look at the power distribution in frequency space (called the *power spectrum* as capturing an essential measure of the function $h(x)$ (its L^2 norm, if you want to get mathematical). This is the most common output of Fourier analysis, and the power spectrum plotted to describe how strong different frequencies are in the observed data.

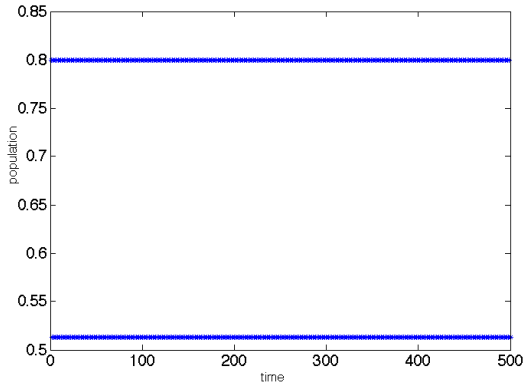
3.4.2 Application: analysis of time series data

One example of such application is the analysis of measurements recorded in time (called a *time series*, somewhat confusing for mathematicians since a series to them means a sum of multiple terms). For illustration, let us use the logistic iterated map that we studied in the first half of this course: $x_{n+1} = rx_n(1 - x_n)$. We will iterate it for several hundred steps (500 to be safe) to get it to converge to the attractor set, then record 500 steps of the steady-state time sequence and look at its power spectrum. Here is the time series and the power spectrum from $r = 3.2$:

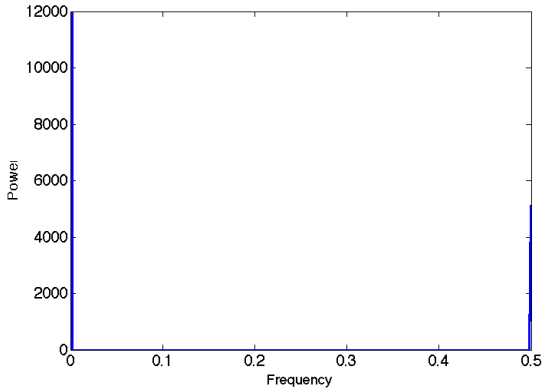
You can see that a period 2 oscillation is represented by two nonzero values in the power spectrum: zero frequency (sum of all values) and frequency 1/2 (period 2). This time of sharply peaked power spectrum is expected from

Now let us take the value $r = 3.8$, which leads to chaotic behavior, which is not periodic. The time series and the power spectrum are shown below:

You can see that the power spectrum of a chaotic time series is dramatically different: all frequencies contribute, fast and slow, but the overall appearance of the power spectrum is ragged, as they are all represented with different powers. This is a tell-tale mark of chaos: lack of well-defined peaks, and an overall unstructured appearance.

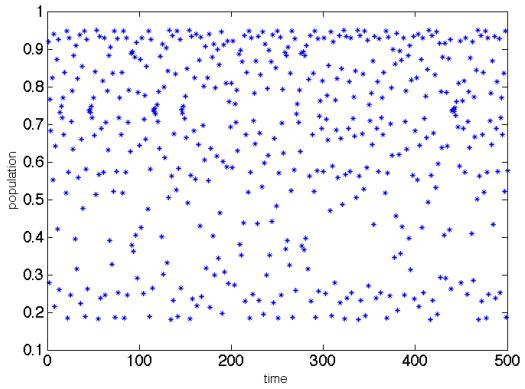


(a) The steady state sequence of values producef by iteration of the logistic map $x_{n+1} = rx_n(1-x_n)$ for $r = 3.2$

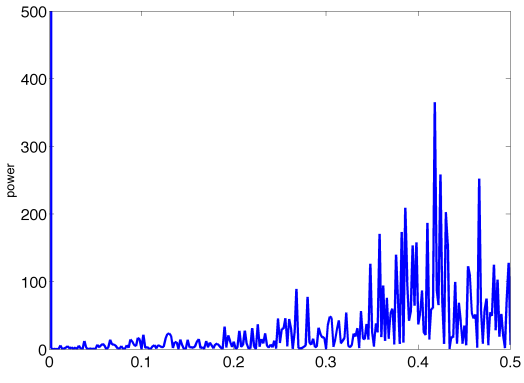


(b) The power spectrum of the data produced by iteration of the logistic map with $r = 3.2$

Figure 3.1: Time series and power spectrum



(a) Sequence of values produced by iteration of the logistic map $x_{n+1} = rx_n(1-x_n)$ for $r = 3.8$



(b) The power spectrum of the data produced by iteration of the logistic map with $r = 3.8$

Figure 3.2: Time series and power spectrum

3.5 Synthesis: Fourier transform of crystal diffraction pattern

One important application of Fourier transforms is to optical scattering. Any time a light is scattered by an object, it results in a Fourier transform of the image of the object, and if enough photons are scattered, one can get a pretty complete picture. In the visible range, we can use lenses, such as in our eyes or in microscopes, to essentially perform the inverse Fourier transform. In X-ray crystallography, which is the primary tool for determining structures of biological molecules, it is not possible, because no optical lenses can interact with photons of that wavelength. We are left to perform the inverse Fourier transforms computationally.

Each atom in the crystal scatters photons with its electron cloud. While it is easy to describe the distribution of electrons in each atom, we have many atoms arranged in a lattice, each contributing to the diffraction intensity. If we consider them all, it will be a very difficult calculation. Instead, the convolution property allows us to find the Fourier transform of a single atom (a Gaussian function, so its Fourier pair is another Gaussian) and find the transform of the idealized lattice of points, both of which are easy calculations, and then multiply them together to find the Fourier transform of the whole lattice of atoms. This gives a simplified idea of how X-ray crystallographers can make sense out of a diffraction pattern and re-create the electron density inside the molecules in the crystal.

References