Basic Math for 16-720

August 23, 2002

1 Linear Algebra

1.1 Vectors and Matrices

First, a reminder of a few basic notations, definitions, and terminology:

- Unless indicated otherwise, vectors are always column vectors ($n \times 1$ matrices). The identity matrix is denoted by $\boldsymbol{I}$.
- The dot product of two column vectors is defined as $\boldsymbol{u} \cdot \boldsymbol{v} = \boldsymbol{u}^T \boldsymbol{v}$
- The norm of of a vector is defined as $|\boldsymbol{u}|^2 = \boldsymbol{u}^T \boldsymbol{u}$
- If θ is the angle between $\boldsymbol{u}$ and $\boldsymbol{v}$, then $\cos\theta = \frac{\boldsymbol{u}^T \boldsymbol{v}}{|\boldsymbol{u}||\boldsymbol{v}|}$
- A matrix $\boldsymbol{A}$ is symmetric if $\boldsymbol{A} = \boldsymbol{A}^T$
- The null space of a matrix $\boldsymbol{A}$ is the set of vectors $\boldsymbol{x}$ such that $\boldsymbol{A}\boldsymbol{x} = 0$
- The rank of $\boldsymbol{A}$ is the dimension of its column(row) space
- A $n \times n$ matrix $\boldsymbol{A}$ is non-singular ($\boldsymbol{A}^{-1}$ exists) if its rank is n, i.e., $det(\boldsymbol{A}) \neq 0$
- A $p \times p$ minor of $\boldsymbol{A}$ is an $p \times p$ sub-matrix extracted from the rows and columns of $\boldsymbol{A}$. If $\boldsymbol{A}$ is of rank $r < n$ then all of its $p \times p$ minors have zero determinant for $p > r$.
- The range of $\boldsymbol{A}$ is the set of vector $\boldsymbol{y}$ such that there exists a vector $\boldsymbol{x}$ with $\boldsymbol{A}\boldsymbol{x} = y$
- As an often-used notation, we denote by $[\boldsymbol{A}|\boldsymbol{B}]$ the $p \times (m + n)$ matrix obtain by concatenating the m columns of the $p \times m$ matrix $\boldsymbol{A}$ and the n columns of the $p \times n$ matrix $\boldsymbol{B}$ (with the obvious corresponding notation for concatenation of the rows.)

1.2 Matrix Decompositions

1.2.1 Eigenvectors and Eigenvalues

$\boldsymbol{x}$ is an *eigenvector* of a $n \times n$ matrix $\boldsymbol{A}$ if there exists a scalar λ such that: $\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x}$; λ is the *eigenvalue* associated with the eigenvector $\boldsymbol{x}$[1].

If A is a symmetric matrix, then it has n real eigenvalues that are solution of the nth-degree polynomial equation $det(\boldsymbol{A} - \lambda\boldsymbol{I}) = 0$.

If A is of rank r, then $n - r$ eigenvalues are equal to 0.

Note that, if $\boldsymbol{x}$ is an eigenvector of $\boldsymbol{A}$, then so is $\alpha\boldsymbol{x}$ for any scalar α. Usually, when we talk about *THE* eigenvectors of the A, we refer to the vector such that $\|\boldsymbol{x}\|^2 = 1$.

[1] In MATLAB: `help eig`

Eigenvalues and eigenvectors are used for reducing a general symmetric matrix $\boldsymbol{A}$ to a simpler form. Namely, if λ_i are the eigenvalues and $\boldsymbol{v}_i$ are the corresponding eigenvectors, then $\boldsymbol{A}$ can be decomposed (diagonalized) in the form:

$$\boldsymbol{A} = \boldsymbol{R}\boldsymbol{D}\boldsymbol{R}^T$$

where $\boldsymbol{D}$ is a diagonal matrix whose ith diagonal elements is λ_i and $\boldsymbol{R}$ is an orthonormal matrix ($\boldsymbol{R}^T\boldsymbol{R} = \boldsymbol{R}\boldsymbol{R}^T = \boldsymbol{I}$)whose ith column is $\boldsymbol{v}_i$.

Eigenvalues and eigenvectors have a simple (and very useful) geometric interpretation. Suppose that all the eigenvalues of $\boldsymbol{A}$ are positive. In that case, the locus of points $\boldsymbol{x}$ such that $\boldsymbol{x}^T A\boldsymbol{x} = 1$ is an ellipsoid. The eigenvectors are the directions of the axes of the ellipsoid; the eigenvalues quantify the elongations in the directions of the axes. Specifically, if λ_i is the eigenvalue corresponding to the ith axis of the ellipsoid, the radius of the ellipsoid along this axis is $\frac{1}{\sqrt{\lambda_i}}$. Another interpretation of the decomposition is that $\boldsymbol{R}$ is the rotation matrix that aligns the coordinate axis with the axes of the ellipsoid.

Two additional properties of the eigenvalues are often useful:

- The sum of the eigenvalues of $\boldsymbol{A}$ is equal to the trace of $\boldsymbol{A}$, i.e., the sum of its diagonal elements
- The product of the eigenvalues is equal to the determinant of $\boldsymbol{A}$

The concept of eigenvalue can be generalized further:

- Given two symmetric $n \times n$ matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, $\boldsymbol{x}$ is a *generalized* eigenvector associated with *generalized* eigenvalue λ if:

$$\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{B}\boldsymbol{x}$$

The computation of the generalized eigenvalues is more difficult, of course, but is critical in some problems[2].

1.2.2 Rayleigh Quotient

In turns out that many problems in computer vision can be reduced to the mathematical problem statement:

- Given a symmetric matrix $\boldsymbol{A}$, find a vector $\boldsymbol{x}$ such that
- $\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}$ is maximum AND
- $\|\boldsymbol{x}\|^2 = 1$

This problem is a constrained maximization problem (constrained because of $\|\boldsymbol{x}\|^2 = 1$). This problem is strictly equivalent to the problem:

- Find $\boldsymbol{x}$ such that $\frac{\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}}{\boldsymbol{x}^T\boldsymbol{x}}$ is maximum.

(recall that $\|\boldsymbol{x}\|^2 = \boldsymbol{x}^T\boldsymbol{x}$.) The ratio above is sometimes called a *Rayleigh Quotient*.

The solution to this problem is given by the following theorem:

- $\frac{\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}}{\boldsymbol{x}^T\boldsymbol{x}}$ reaches its absolute maximum when $\boldsymbol{x}$ is an eigenvector of $\boldsymbol{A}$ corresponding to the *largest* eigenvalue λ_{max}.

[2]In MATLAB, the same function `eig` solves the generalized problem.

Note that the same result holds if the ratio is minimized (this time the *smallest* eigenvalue is used, of course.)

This result can be generalized in a straightforward manner:

- Given two matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, $\frac{\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}}{\boldsymbol{x}^T\boldsymbol{B}\boldsymbol{x}}$ reaches its absolute maximum when $\boldsymbol{x}$ is a *generalized* eigenvector of $\boldsymbol{A}$ corresponding to the largest *generalized* eigenvalue λ_{max}, that is: $\boldsymbol{A}\boldsymbol{x} = \lambda_{max}\boldsymbol{B}\boldsymbol{x}$

Note that this generalized problem is equivalent to maximizing $\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}$ under the constraint $\boldsymbol{x}^T\boldsymbol{B}\boldsymbol{x} = 1$.

As a simple example of application of the Rayleigh Quotient theorem, consider the following (contrived) problem: Suppose that we have a set of vectors in two dimensions, $\boldsymbol{u}_j$ with $j = 1 \ldots m$. We wish to find the vector $\boldsymbol{x}$ of magnitude 1 such that $\boldsymbol{x}$ is as close as possible to the $\boldsymbol{u}_j$'s. The distance between $\boldsymbol{x}$ and $\boldsymbol{u}_j$ could be measured by the dot product $\boldsymbol{x}^T\boldsymbol{u}_j$. This dot product measures the angle between $\boldsymbol{x}$ and $\boldsymbol{u}_j$ and is maximum (equal to the magnitude of $\boldsymbol{u}_j$) if $\boldsymbol{x} = \frac{\boldsymbol{u}_j}{|\boldsymbol{u}_j|}$. Therefore, we want to find a unit vector $\boldsymbol{x}$ that maximizes:

$$\sum_j (\boldsymbol{x}^T\boldsymbol{u}_j)^2 = \sum_j \boldsymbol{x}^T\boldsymbol{u}_j\boldsymbol{u}_j^T\boldsymbol{x} = \boldsymbol{x}^T \sum_j \boldsymbol{u}_j\boldsymbol{u}_j^T\boldsymbol{x}$$

This is exactly the Rayleigh Quotient problem, so the solution is the eigenvector $\boldsymbol{x}_{max}$ corresponding to the largest eigenvalue λ_{max} of the matrix:

$$\boldsymbol{A} = \sum_j \boldsymbol{u}_j\boldsymbol{u}_j^T$$

Incidentally $\boldsymbol{A}$ is the *scatter matrix* of the cloud of points formed by the vectors $\boldsymbol{u}_j$ and it defines the best fit ellipsoid to that cloud of points. In particular, $\boldsymbol{x}_{max}$ corresponds to the direction of largest variation of the cloud of points.

1.2.3 Singular Value Decomposition

The decomposition in eigenvalues (diagonalization) applies only to square matrices. A more general decomposition is the Singular Value Decomposition (SVD)[3]. A $m \times n$ matrix $\boldsymbol{A}$ with $m \geq n$ can always be decomposed in the form:

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$$

where:

- $\boldsymbol{\Sigma}$ is an $n \times n$ *diagonal* matrix. The elements $\sigma_1, \ldots, \sigma_n$ of the diagonal are called the singular values of $\boldsymbol{A}$ and $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n \geq 0$
- $\boldsymbol{U}$ is a $m \times n$ matrix. The columns of $\boldsymbol{U}$ are of unit norm and orthogonal to each other, that is: $\boldsymbol{U}^T\boldsymbol{U} = \boldsymbol{I}$ (sometimes called *column-orthonormal*)
- $\boldsymbol{V}$ is a $n \times n$ orthonormal matrix, that is: $\boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{V}\boldsymbol{V}^T = \boldsymbol{I}$

The SVD is a powerful tool for analyzing matrices. The key properties of the SVD are:

- SVD and eigenvalues: If $\boldsymbol{A}$ happens to be a square matrix $m = n$, then there is a simple relation between the eigenvalues α_i of $\boldsymbol{A}^T\boldsymbol{A}$ and the singular values σ_i: $\alpha_i = \sigma_i^2$. In particular, if $\boldsymbol{A}$ is symmetric, then: $\lambda_i^2 = \sigma_i^2$ where λ_i is the eigenvalue of $\boldsymbol{A}$.
- Rank: Suppose that p singular values are non-0, i.e., $\sigma_1 \geq \ldots \geq \sigma_p \geq 0$ and $\sigma_{p+1} = \ldots = \sigma_{n-p} = 0$, then the rank of $\boldsymbol{A}$ is p.

[3] In MATLAB: `help svd`

- Null Space: If $\boldsymbol{A}$ is of rank p, then the $n-p$ lowest singular values are equal to 0 and the last $n-p$ columns of $\boldsymbol{V}$ form an orthonormal basis of the null space of $\boldsymbol{A}$. This is extremely important in practice because it allows us to represent explicitly the set of vectors for which $\boldsymbol{Ax} = 0$, the null space.
- Range: This is the dual property to the null space property: the first p columns of $\boldsymbol{U}$ form an orthonormal basis for the range of $\boldsymbol{A}$, that is, the set of vectors $\boldsymbol{y}$ such that there exists a vector $\boldsymbol{x}$ with $\boldsymbol{y} = \boldsymbol{Ax}$.

A key application of SVD is to solve the following problem:

- Given a $m \times n$ matrix $\boldsymbol{A}$ with $m \geq n$.
- Given a number $p \leq n$
- Find the matrix $\boldsymbol{A}_p$ *of rank* p that is closest to $\boldsymbol{A}$.

To solve this problem, we need to define what we mean by "closest". We can define the distance between any two matrices $\boldsymbol{M}$ and $\boldsymbol{N}$ by: $\sum_{i,j}(m_{ij} - n_{ij})^2$, that is the sum of squared differences between the components of $\boldsymbol{M}$ and $\boldsymbol{N}$. This defines a norm on the space of matrices, which is usually called the *Frobenius* norm. The Frobenius norm simply captures the intuitive notion that matrices are "close to each other" if they all their elements are close to each other.

Having defined the Frobenius norm, we can now state the key result (a result that we will use repeatedly in vision applications, by the way):

- The matrix $\boldsymbol{A}_p$ of rank p that is the closest (in the Frobenius sense) to a given matrix $\boldsymbol{A}$ with SVD decompostion $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ is given by:

$$\boldsymbol{A}_p = \boldsymbol{U}_p\boldsymbol{\Sigma}_p\boldsymbol{V}_p^T, \text{ where}$$

  - $\boldsymbol{\Sigma}_p$ is a $p \times p$ diagonal matrix whose elements are the first p singular values of $\boldsymbol{A}$.
  - $\boldsymbol{U}_p$ is a $m \times p$ matrix formed by taking the first p *columns* of $\boldsymbol{U}$.
  - $\boldsymbol{V}_p$ is a $n \times p$ matrix formed by taking the first p *columns* of $\boldsymbol{V}$.

This looks complicated, but all it really says is that if you want to approximate a matrix by a matrix of rank p, all you need to do is to ignore the $n-p$ least important components of the matrix (corresponding to the $n-p$ smallest singular values), and that it is mathematically the optimal thing to do! See Figure 2 for a graphical illustration.

The reason why this is important is that, in many problems, we know the rank p that some matrix $\boldsymbol{A}$ should have because of prior knowledge about the structure of the underlying problem. However, because of noise in the data, the real matrix that we observe $\boldsymbol{A}'$ may have a higher rank. Using the theorem above, we can recover the "ideal" matrix of rank p that approximates the matrix observed from the actual data, $\boldsymbol{A}'$, in a simple way.

Let us take a look at a simple toy example. Suppose that we are working on a problem in which we want to exploit the fact that a matrix computed from the data is known to be of rank 2. For example, suppose that the ideal (unknown) value of the matrix is:

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \\ 4 & 2 & 6 \end{bmatrix}$$

Because of noise in the data, e.g., the matrix may have been derived from pixel values in an image, the actual matrix that we observe from the data is perturbed and is instead:

$$\boldsymbol{A}' = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \\ 4 & 2 & 6+10^{-4} \end{bmatrix}$$

$\boldsymbol{A}'$ is now of full rank because of the noise perturbation, and therefore any algorithm that depends on the rank-2 assumption will fail. The SVD decomposition of $\boldsymbol{A}'$ is:

$$\boldsymbol{A}' = \begin{bmatrix} -0.497 & -0.648 & -0.577 \\ -0.313 & 0.754 & -0.577 \\ -0.810 & 0.106 & 0.577 \end{bmatrix} \begin{bmatrix} 9.075 & 0 & 0 \\ 0 & 1.280 & 0 \\ 0 & 0 & 2.5 \times 10^{-5} \end{bmatrix} \begin{bmatrix} -0.497 & -0.648 & -0.577 \\ -0.313 & 0.754 & -0.577 \\ -0.810 & 0.106 & 0.577 \end{bmatrix}^T$$

The projection theorem above tells us that the closest rank-2 matrix to the actual data matrix $\boldsymbol{A}'$ is:

$$\boldsymbol{A}'' = \begin{bmatrix} -0.497 & -0.648 \\ -0.313 & 0.754 \\ -0.810 & 0.106 \end{bmatrix} \begin{bmatrix} 9.075 & 0 \\ 0 & 1.280 \end{bmatrix} \begin{bmatrix} -0.497 & -0.648 \\ -0.313 & 0.754 \\ -0.810 & 0.106 \end{bmatrix}^T$$

What the theorem gives us is an easy way to find the matrix $\boldsymbol{A}''$ that best fits our model (rank 2) given the data, $\boldsymbol{A}'$.

1.3 2-D and 3-D Geometry

1.3.1 Rigid and Affine Transformations

Let's consider for a moment the cases of 2 and 3-dimensional spaces. The simplest geometric transformation in such spaces is a translation $\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{t}$. The next type of transformation is a rotation $\boldsymbol{y} = \boldsymbol{R}\boldsymbol{x}$ where $\boldsymbol{R}$ is an orthonormal matrix, $\boldsymbol{R}^T\boldsymbol{R} = \boldsymbol{R}\boldsymbol{R}^T = \boldsymbol{I}$.

In 3-D, $\boldsymbol{R}$ is fully defined by three parameters, either three rotation angles about the three coordinate axes, or the direction of the rotation axis and the angle of rotation.

In 2-D, $\boldsymbol{R}$ is fully parameterized by one rotation angle θ and has the general form:

$$\boldsymbol{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Rigid transformations are combinations of rotations of translations: $\boldsymbol{y} = \boldsymbol{R}\boldsymbol{x} + \boldsymbol{t}$. Rigid transformations preserve lengths and angles.

If we replace the rotation matrix $\boldsymbol{R}$ by a general 2×2 (or 3×3) matrix $\boldsymbol{A}$, the resulting transformation $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{t}$ is called an *affine* transformation. Affine transformations do not preserve lengths or angles but they do preserve *ratios* of lengths and parallelism (parallel lines remain parallel after transformation.)

1.3.2 Homogeneous Coordinates

If $\boldsymbol{u} = [x\ y]^T$ is a 2-D vector, we could write the transformation $\boldsymbol{v} = \boldsymbol{A}\boldsymbol{u} + \boldsymbol{t}$ as:

$\boldsymbol{v} = \boldsymbol{M} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$, where $\boldsymbol{M} = [\boldsymbol{A}|\boldsymbol{t}]$ is the 2×3 matrix obtained by concatenating the columns of $\boldsymbol{A}$ and $\boldsymbol{t}$.

The interesting thing is that the transformation between $\boldsymbol{v}$ and $[x\ y\ 1]^T$ is expressed by a single 2×3 matrix $\boldsymbol{M}$. In general, the vector $[x\ y]^T$ is said to be expressed in *homogeneous* coordinates as $[x\ y\ 1]^T$. More generally, any set of homogeneous coordinates $[a\ b\ c]^T$ such that $a/c = x$ and $b/c = y$ represent the same 2-D vector $[x\ y]^T$. Two vectors expressed in homogeneous coordinates, $\tilde{\boldsymbol{u}} = [a\ b\ c]^T$ and $\tilde{\boldsymbol{u}}' = [a'\ b'\ c']^T$, represent the same vector if $a/c = a'/c'$ and $b/c = b'/c'$. In that case we write $\tilde{\boldsymbol{u}} \equiv \tilde{\boldsymbol{u}}'$ (or $\tilde{\boldsymbol{u}} \propto \tilde{\boldsymbol{u}}'$, depending on the text!) to indicate that $\tilde{\boldsymbol{u}}$ and $\tilde{\boldsymbol{u}}'$ are equivalent sets of homogeneous coordinates.

Be careful: $\tilde{\boldsymbol{u}} \equiv \tilde{\boldsymbol{u}}'$ does not mean that the components of $\tilde{\boldsymbol{u}}$ and $\tilde{\boldsymbol{u}}'$ are equal, only that they are proportional.

With these definitions, a generic affine transformation $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}+\boldsymbol{t}$ is expressed in homogeneous coordinates as a transformation of the form $\tilde{\boldsymbol{u}}' = \boldsymbol{M}\tilde{\boldsymbol{u}}$, where $\boldsymbol{M}$ is the 3×3 matrix defined as:

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{t} \\ 0\ 0 & 1 \end{bmatrix}$$

For example, the rigid transformation of rotation angle θ and translation $[2\ 3]^T$ is represented in homogeneous coordinates by:

$$\boldsymbol{M} = \begin{bmatrix} \cos\theta & -\sin\theta & 2 \\ \sin\theta & \cos\theta & 3 \\ 0 & 0 & 1 \end{bmatrix}$$

Homogeneous coordinates are defined in the same way in 3-D by adding one coordinate to the three-dimensional vectors. That is, two homogeneous vectors $\tilde{\boldsymbol{u}} = [a\ b\ c\ d]^T$ and $\tilde{\boldsymbol{u}}' = [a'\ b'\ c'\ d']^T$ represent the same 3-D vector if: $a/c = a'/d' = x$, $b/d = b'/d' = y$, and $c/d = c'/d' = z$. A generic affine transformation $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{t}$ is expressed in homogeneous coordinates as a transformation of the form $\tilde{\boldsymbol{u}}' = \boldsymbol{M}\tilde{\boldsymbol{u}}$, where $\boldsymbol{M}$ is the 4×4 matrix defined as:

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{t} \\ 0\ 0\ 0 & 1 \end{bmatrix}$$

1.3.3 Projective Transformations

Consider first the 2-D case. The last row of the transformation matrices defined above in homogeneous coordinates is $[0\ 0\ 1]^T$. We could replace those fixed values by arbitrary values to define a general transformation in homogeneous coordinates: $\tilde{\boldsymbol{u}}' = \boldsymbol{M}\tilde{\boldsymbol{u}}$, where $\boldsymbol{M}$ is a general 3×3 matrix. Consider a vector $\boldsymbol{x} = [x\ y]^T$, a representation in homogeneous coordinates is, for example, $\boldsymbol{u} = [x\ y\ 1]^T$ and

$\tilde{\boldsymbol{u}}' = \boldsymbol{M}\tilde{\boldsymbol{u}} = \begin{bmatrix} \boldsymbol{m}_1^T\tilde{\boldsymbol{u}} \\ \boldsymbol{m}_2^T\tilde{\boldsymbol{u}} \\ \boldsymbol{m}_3^T\tilde{\boldsymbol{u}} \end{bmatrix}$, where $\boldsymbol{m}_i^T$ is the ith row of $\boldsymbol{M}$.

The vector represented by the transformed set of homogeneous coordinates $\tilde{\boldsymbol{u}}'$ is obtained by dividing the first coordinates by the third one, yielding:

$$x' = \frac{\boldsymbol{m}_1^T\tilde{\boldsymbol{u}}}{\boldsymbol{m}_3^T\tilde{\boldsymbol{u}}} \text{ and } y' = \frac{\boldsymbol{m}_2^T\tilde{\boldsymbol{u}}}{\boldsymbol{m}_3^T\tilde{\boldsymbol{u}}}$$

Note that, if we use arbitray numbers instead of $[0\ 0\ 1]^T$ in the last row of the matrix, the transformation becomes a non-linear transformation of the original coordinates, even though it is a *linear* transformation in homogeneous coordinates. Such a transformation, represented by a 3×3 matrix in homogeneous coordinates is called a *projective* transformation[4]. Projective transformations do not preserve lengths, angles, ratios, or parallelism. They do preserve colinearity (things that are on the same line remain on the same line after transformation) and incidence (curves that are tangent to each other remain tangent to each other.)

Projective transformations are defined in three dimensions in the same manner: A 4×4 matrix is applied in homogeneous coordinates. The vector coordinates corresponding to the homogeneous coordinates are recovered by dividing the first three coordinates by the fourth one. The special case in which the last row of the matrix is $[0\ 0\ 0\ 1]^T$ corresponds to affine transformations just like in 2-D. It is important to note that a transformation expressed in homogeneous coordinates is defined only up to a scale factor because two homogeneous vectors are equivalent if they are equal up to a scale factor ($\boldsymbol{u} \equiv \tilde{\boldsymbol{u}}'$ means that $\tilde{\boldsymbol{u}} = \alpha\tilde{\boldsymbol{u}}'$ for some scalar α.) In other words, if $\boldsymbol{M}$ is the matrix of a transformation in homogeneous coordinates, then $\alpha\boldsymbol{M}$ represents the same transformation for any $\alpha \neq 0$.

[4]If you want to sound sophisticated, the correct technical name is "collineation", but projective transformation is good enough for now.....

1.3.4 3-D to 2-D Transformations

Transformations that map a point in 3-D to a point in the 2-D plane are expressed in homogeneous coordinates by 3×4 matrices. A 3-D to 2-D transformation maps a 3-D point of coordinates $[x\ y\ z]^T$ to a point in the plane of coordinates $[x'\ y']^T$ by applying a 3×4 matrix $\boldsymbol{M}$ to the homogeneous coordiantes of the 3-D point, $[x\ y\ z\ 1]^T$, i.e., the coordinates x' and y' are computed by:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \boldsymbol{M} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad x' = \frac{u}{w}\ y' = \frac{v}{w}$$

3-D to 2-D transformations come in three varieties. If the last row of $\boldsymbol{M}$ is $[0\ 0\ 0\ 1]^T$ and the first two rows are the first two rows of a 3-D rigid transformation matrix, the transformation is an *orthographic* projection. This type of transformation amounts to applying a rigid transformation in 3-D and keeping the first two coordinates of the transformed 3-D point as the result of the projection.

If the last row of $\boldsymbol{M}$ is $[0\ 0\ 0\ 1]^T$ and the first two rows are arbitrary, i.e., not necessarily from a rigid transformation, then $\boldsymbol{M}$ is an *affine* projection. This type of transformation amounts to applying an affine transformation in 3-D and keeping the first two coordinates of the transformed 3-D point as the result of the projection.

If the last row of $\boldsymbol{M}$ is not $[0\ 0\ 0\ 1]^T$, $\boldsymbol{M}$ is a projective projection. Projective transformations from 3-D to 2-D are simply arbitrary 3×4 matrices applied to homogeneous coordinates. One important example that illustrates the use of projective projection is the representation of the perspective projection used in the pinhole camera model. Recall that, if $[x\ y\ z]^T$ is a point in space, the coordinates of the corresponding point $[x'\ y']^T$ after projection through a pinhole are given by:

$$x' = f\frac{x}{z} \text{ and } y' = f\frac{y}{z}$$

In homogeneous coordinates, this becomes a linear transformation:

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \text{ with } x' = \frac{a}{c} \text{ and } y' = \frac{b}{c}$$

This is the reason why the projective (homogeneous representation) is so important to us: It enables us to convert the non-linear, perspective projection problems to linear problems[5]

1.3.5 Cross-Product and Skew-Symmetric Matrices

In three dimensions, the cross product of two vectors $\boldsymbol{u} = [u_x\ u_y\ u_z]^T$ and $\boldsymbol{v} = [v_x\ v_y\ v_z]^T$ is defined by:

$$\boldsymbol{u} \times \boldsymbol{v} = \begin{bmatrix} u_y v_z - u_z v_y \\ u_z v_x - u_x v_z \\ u_x v_y - u_y v_x \end{bmatrix}$$

A few basic properties of the cross product are useful and worth remembering:

- $\boldsymbol{u} \times \boldsymbol{v} = -\boldsymbol{v} \times \boldsymbol{u}$ (antisymmetry)
- $\boldsymbol{w} \cdot (\boldsymbol{u} \times \boldsymbol{v})$ is the determinant of the matrix formed by the three vectors $\boldsymbol{u}\ \boldsymbol{v}\ \boldsymbol{w}$

[5]Of course, there is a much broader theory of projective representation behind this, which we will need later, but this is sufficient for now.

Transformation	Vector Coordinates	Homogeneous Coordinates	Degrees of Freedom	Invariants
Translation	$\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{t}$	$\begin{bmatrix} \boldsymbol{I} & \boldsymbol{t} \\ 0\ 0 & 1 \end{bmatrix}$	2	lengths, angles
Rotation	$\boldsymbol{y} = \boldsymbol{R}\boldsymbol{x}$ $\boldsymbol{R}^T\boldsymbol{R} = \boldsymbol{R}\boldsymbol{R}^T = \boldsymbol{I}$	$\begin{bmatrix} \boldsymbol{R} & \boldsymbol{0} \\ 0\ 0 & 1 \end{bmatrix}$	1	lengths, angles
Rigid	$\boldsymbol{y} = \boldsymbol{R}\boldsymbol{x} + \boldsymbol{t}$	$\begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ 0\ 0 & 1 \end{bmatrix}$	3	lengths, angles
Affine	$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{t}$	$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{t} \\ 0\ 0 & 1 \end{bmatrix}$	6	ratios of lengths, parallelism
Projective		3×3 matrix $\boldsymbol{M}$	8	colinearity, incidence

Table 1: Classification of 2-D transformations

Transformation	Vector Coordinates	Homogeneous Coordinates	Degrees of Freedom	Invariants
Translation	$\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{t}$	$\begin{bmatrix} \boldsymbol{I} & \boldsymbol{t} \\ 0\ 0\ 0 & 1 \end{bmatrix}$	3	lengths, angles
Rotation	$\boldsymbol{y} = \boldsymbol{R}\boldsymbol{x}$ $\boldsymbol{R}^T\boldsymbol{R} = \boldsymbol{R}\boldsymbol{R}^T = \boldsymbol{I}$	$\begin{bmatrix} \boldsymbol{R} & \boldsymbol{0} \\ 0\ 0\ 0 & 1 \end{bmatrix}$	3	lengths, angles
Rigid	$\boldsymbol{y} = \boldsymbol{R}\boldsymbol{x} + \boldsymbol{t}$	$\begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ 0\ 0\ 0 & 1 \end{bmatrix}$	6	lengths, angles
Affine	$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{t}$	$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{t} \\ 0\ 0\ 0 & 1 \end{bmatrix}$	12	ratios of lengths, parallelism
Projective		4×4 matrix $\boldsymbol{M}$	15	colinearity, incidence

Table 2: Classification of 3-D transformations

Transformation	Homogeneous Coordinates	Degrees of Freedom
Orthographic	$\begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ 0\ 0\ 0 & 1 \end{bmatrix}$ $\boldsymbol{R}\boldsymbol{R}^T = \boldsymbol{I}$ (row-orthonormal)	5
Affine	$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{t} \\ 0\ 0\ 0 & 1 \end{bmatrix}$	8
Projective	3×4 matrix $\boldsymbol{M}$	11

Table 3: Classification of 3-D to 2-D transformations

- $\boldsymbol{a} \times (\boldsymbol{b} \times \boldsymbol{c}) = (\boldsymbol{a} \cdot \boldsymbol{b})\boldsymbol{c} - (\boldsymbol{a} \cdot \boldsymbol{c})\boldsymbol{b}$
- $|\boldsymbol{u} \times \boldsymbol{v}| = \sin\theta |\boldsymbol{u}||\boldsymbol{v}|$

Given a vector $\boldsymbol{u}$, The mapping that associates $\boldsymbol{u} \times \boldsymbol{x}$ to $\boldsymbol{x}$ is obviously a linear mapping. Therefore, there must exist a matrix $\boldsymbol{M}$ such that: $\boldsymbol{u} \times \boldsymbol{x} = \boldsymbol{M}\boldsymbol{x}$ for any $\boldsymbol{x}$. Such a matrix $\boldsymbol{M}$ is called a *skew-symmetric matrix*. The skew-symmetric matrix associated with a vector $\boldsymbol{u}$ is denoted typically by $\boldsymbol{u}_\times$or $[\boldsymbol{u}]_\times$ (sorry, no standardized notations!) The expression for the skew-symmetric matrix is:

$$\boldsymbol{u}_\times = \begin{bmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{bmatrix}$$

So, for any $\boldsymbol{x}$, $\boldsymbol{u} \times \boldsymbol{x} = \boldsymbol{u}_\times \boldsymbol{x}$. The matrix is called skew-symmetric because $\boldsymbol{u}_\times^T = -\boldsymbol{u}_\times$ owing to the antisymmetry of the cross product.

Warning: If you have never seen this skew-symmetric construct before, you should make sure you understand it. Most of the derivations in multi-camera geometry will rely on this notation.

A final note in this discussion of the cross product has to do with its use in expressing parallelism between vectors. Specifically, let $\boldsymbol{a}$ and $\boldsymbol{b}$ be two parallel vectors. The fact that they are parallel can be expressed by saying that they are colinear, i.e., there exists a scalar λ such that: $\boldsymbol{a} = \lambda \boldsymbol{b}$. This is fine, except that we have introduced an extra variable λ which really is a dummy variable which adds absolutely no information if all that we are interested in is the parallelism of the vectors. This seemingly trivial observation will become critical in some of the problems we'll have to solve. Specifically, in solving any problem that involves parallelism, we would be forced to introduce spurious extra degrees of freedom, the λ's, which would greatly complicate the algorithms.

A better way to describe parallelism is to observe that $\boldsymbol{a}$ and $\boldsymbol{b}$ are parallel iff $\boldsymbol{a} \times \boldsymbol{b} = 0$, or even better: $\boldsymbol{a}_\times \boldsymbol{b} = 0$, that way we have converted the parallelism condition to a simple set of linear equations without any extra parameters. In the future, we will automatically convert problems like "those rays are parallel to this direction" to a set of linear equations using this trick. In particular, given two homogeneous vectors in the plane $\tilde{\boldsymbol{u}} = [a\ b\ c]^T$ and $\tilde{\boldsymbol{u}}' = [a'\ b'\ c']^T$, we have seen that $\tilde{\boldsymbol{u}} \equiv \tilde{\boldsymbol{u}}'$ means that the two vectors are proportional to each other, in other words: $\tilde{\boldsymbol{u}} \times \tilde{\boldsymbol{u}}' = 0$. A little trick that will help a lot.

1.3.6 Lines and Planes

Here are some very simple (but sometimes forgotten) facts about lines and planes:

- A line in the plane (resp. plane in space) has equation $ax + by + c = 0$ (resp. $ax + by + cz + d = 0$

- In homogeneous coordinates, the equation becomes $\boldsymbol{l}^T\boldsymbol{v}$ (resp. $\boldsymbol{p}^T\boldsymbol{v}$) with $\boldsymbol{v} = [x\ y\ 1]^T$ and $\boldsymbol{l} = [a\ b\ c]^T$ (resp. $\boldsymbol{v} = [x\ y\ z\ 1]^T$ and $\boldsymbol{p} = [a\ b\ c\ d]^T$.)
- The normal direction to the line (resp. plane) is the vector $\frac{1}{\sqrt{a^2+b^2}}[a\ b]^T$ (resp. $\frac{1}{\sqrt{a^2+b^2+c^2}}[a\ b\ c]^T$)
- The distance of a generic point $\boldsymbol{v}$ to the line $\boldsymbol{l}$ (resp. plane $\boldsymbol{p}$) is $\frac{1}{\sqrt{a^2+b^2}}\boldsymbol{l}^T\boldsymbol{v}$ (resp. $\frac{1}{\sqrt{a^2+b^2+c^2}}\boldsymbol{p}^T\boldsymbol{v}$.)
- The intersection of two lines $\boldsymbol{l}_1$ and $\boldsymbol{l}_2$ is the point of homogeneous coordinates $\boldsymbol{l}_1 \times \boldsymbol{l}_2$ [6]

2 Optimization

2.1 Linear Least-Squares

Given a m-dimensional vector $\boldsymbol{b}$ and a $m \times n$ matrix $\boldsymbol{A}$, the problem is to find the n-dimensional vector $\boldsymbol{x}$ such that the residual $r = |\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}|^2$ is minimum[7]. This is the standard least-squares problem which we need to solve whenever we have m n-dimensional input data vectors $\boldsymbol{a}_i$, output values b_i and we want to fit a linear model of the form $\boldsymbol{a}_i^T\boldsymbol{x} = \boldsymbol{b}_i$ by minimizing $\sum_1^m |\boldsymbol{a}_i^T\boldsymbol{x} - \boldsymbol{b}_i|^2$. The matrix $\boldsymbol{A}$ is obtained by stacking the m row vectors $\boldsymbol{a}_i^T$ and $\boldsymbol{b}$ is obtained by concatenating the b_i's into a column vector.

The solution to the least-squares problem is given by:

$$\boldsymbol{x} = (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{b}$$

The matrix $\boldsymbol{A}^+ = (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T$ is the *pseudo-inverse* of $\boldsymbol{A}$. This solution assumes that the problem is over-constrained, i.e., there are more independent equations than unknowns ($\text{rank}(\boldsymbol{A}) = n$ and $m > n$). If the problem is underconstrained, the pseudo-inverse cannot be computed.

A solution that works with both over- and under-constrained problems can be derived using the SVD. This solution can also be more efficient and is more robust numerically.

Let $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ be the SVD of $\boldsymbol{A}$. Define the matrix $\boldsymbol{A}^+ = \boldsymbol{V}\boldsymbol{\Sigma}^+\boldsymbol{U}^T$, where $\boldsymbol{\Sigma}^+$ is the matrix formed by taking the inverse of the non-zero elements of $\boldsymbol{\Sigma}$ (and leaving the 0 elements unchanged.) With this definition of $\boldsymbol{A}^+$, the solution to the least-squares problem is:

$$\boldsymbol{x}_o = \boldsymbol{A}^+\boldsymbol{b}.$$

If the system is overconstrained, this is the same solution as with the previous definition of the pseudo-inverse.

If the system is under-constrained, and therefore has multiple solutions, this gives the solution $\boldsymbol{x}_o$ of minimum norm. Note that, any other solution will be of the form $\boldsymbol{x}_o + \boldsymbol{V}'\boldsymbol{y}$, where $\boldsymbol{V}'$ is the matrix formed from the columns corresponding to the null singular values of $\boldsymbol{A}$ (i.e., the columns of $\boldsymbol{V}'$ span the null space of $\boldsymbol{A}$) and $\boldsymbol{y}$ is an arbitrary vector.

2.2 Unconstrained Non-Linear Problems

The problem now is to find the n-vector $\boldsymbol{x}$ that is an extremum (minimum or maximum) of an objective function $z = F(\boldsymbol{x})$. We will encounter many such problems and we can generally use a standard minimization [8] as a black box. However, it is still interesting to understand the general characteristics of minimization approaches in order to assess the feasibility of a particular solution. For example, the dimensionality of some problems is such that, without some manipulation of the data or some approximation to the problem, optimization of the objective function will not be feasible. To help understand

[6] A similar relation exists for the intersection of planes but not with the cross product, which works only for 3-D vectors. One needs to use a different representation of lines in 3D, the Plucker coordinates, see Forsyth and Ponce, p. 165.

[7] MATLAB: `help regress`

[8] `help optim` in MATLAB will list pretty much all the non-linear optimization techniques known to mankind.

key issues and limitations, this section, along with the next two sections, provides a very quick overview of the basic approaches commonly used to solve non-linear optimization problems.

An optimization algorithm starts from an initial choice of the argument, $\boldsymbol{x}_o$ and iteratively generates successive values $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ corresponding to values of the objective function $F(\boldsymbol{x}_1), \ldots, F(\boldsymbol{x}_n)$ that converge to an extremum. It should be clear from the outset that the best we can hope for is to find a *local* extremum of the objective function. It is not possible in general to guarantee that the algorithm converges to a globally optimal value. It is also generally not possible to guarantee that the local extremum is a "deep" minimum or maximum. Moreover, many techniques may fail to converge if the starting point $\boldsymbol{x}_o$ is too far from a local extremum. It is therefore critical to understand how good (or poor) the choice of $\boldsymbol{x}_o$ is.

Different algorithms may have dramatically different convergence rates, as measured by the rate of change of the objective function as a function of the number of iterations. Typically, the fastest algorithms have quadratic convergence rate. Unfortunately, it is often the case that the computations involved at each step are prohibitively expensive. Therefore, assessing the feasibility of solving a problem through non-linear optimization involves understanding how good is the choice of the starting point, the convergence properties of the optimization algorithms, and the amount of computation to be performed at each iteration - and whether the computation can be carried out at all in practice.

To compute $\boldsymbol{x}_{k+1}$ from $\boldsymbol{x}_k$, we need to first consider the Taylor expansion of F around $\boldsymbol{x}_k$:

$$F(\boldsymbol{x}_{k+1}) = F(\boldsymbol{x}_k) + \nabla F^T(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k) + \tfrac{1}{2}(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k)^T \boldsymbol{H}(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k)$$

∇F is the gradient of F at $\boldsymbol{x}_k$, and $\boldsymbol{H}$ is the Hessian, i.e., matrix of second derivatives, of F at $\boldsymbol{x}_k$: $\boldsymbol{H} = \nabla^2 F(\boldsymbol{x}_k)$. Optimization algorithms attempt to find the update vector $(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k)$ that would yield the largest change in F based on this Taylor expansion.

We assume from now on that we want to find a minimum of F (a few sign reversals are needed if we want a maximum of F.) The first approach is based on the observation that the fastest variation of F is in the direction opposite to that of the gradient (*steepest descent*), i.e., $-\nabla F$. In other words, the iteration is defined by: $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \nabla F$, where α is chosen to maximize the change in F. Substituting $\alpha \nabla F$ for $(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k)$ in the Taylor expansion yields:

$$F(\boldsymbol{x}_{k+1}) = F(\boldsymbol{x}_k) - \alpha |\nabla F|^2 + \tfrac{1}{2}\alpha^2 \nabla F^T \boldsymbol{H} \nabla F$$

From this expression it is easy to see that the α that minimizes the right-hand side is $\alpha = \frac{|\nabla F|^2}{\nabla F^T \boldsymbol{H} \nabla F}$, and the update rule at each iteration is:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \frac{|\nabla F|^2}{\nabla F^T \boldsymbol{H} \nabla F} \nabla F$$

This approach is the *gradient descent* algorithm. Gradient descent has the advantage that it does not require any expensive matrix inversion. As shown here, it requires the computation of the second derivatives of F. However, it should be noted that other techniques can be used to choose α that do not require the second derivatives (with potentially slower convergence.) Gradient descent has two drawbacks: It has slow (linear) convergence and it is not guaranteed to converge if the starting point is far from the minimum.

An alternative to gradient descent is to not try to force the iterations to follow the gradient and to, instead, find the difference $(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k)$ that minimizes the right-hand side of the Taylor expansion. Setting the derivative of the right-hand side with respect to $(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k)$ to zero yields directly:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{H}^{-1} \nabla F$$

This approach is the *Newton* method. The advantage of the Newton method is that it has fast (quadratic) convergence. It still requires the computation of the second derivatives. More importantly, a major drawback is that it requires the inversion of the Hessian matrix which may cause serious computational and numerical problems. From a computational standpoint, if d is the dimension of $\boldsymbol{x}$, then $\boldsymbol{H}$ is a

$d \times d$ matrix and its inversion is $O(d^3)$. In real problems (in vision), d could be on the order of several hundreds or thousands, thus requiring a large amount of computation at each iteration, irrespective of the convergence rate.

Another problem is that $\boldsymbol{H}$ may be close to singular, in which case the inversion is numerically unstable especially if d is large. To see why this is a problem in practice, it is useful to understand the geometric interpretation of the Newton method. The right-hand side of the Taylor expansion of F is an approximation of the surface $z = F(\boldsymbol{x})$ by a quadric surface. The Newton method simply finds the lowest on that approximating surface. Assuming that we have translated the axes so that the minimum is at $\boldsymbol{x} = \boldsymbol{0}$, the Taylor expansion around $\boldsymbol{0}$ becomes $F(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{H}\boldsymbol{x}$, which is a quadratic approximation of the objective function.

The eigenvalues of $\boldsymbol{H}$ give us an indication of the local shape of $z = F(\boldsymbol{x})$ (Figure 3) Specifically, large eigenvalues indicate directions along which the function varies quickly. Along these directions, the minimum is sharp and can be localized easily. Furthermore, the inversion of $\boldsymbol{H}$ is numerically stable. On the other hand, small eigenvalues correspond to directions along which the function varies slowly, i.e., the minimum of the surface is very shallow. Along these directions, it is more difficult to accurately localize the minimum and the inversion of $\boldsymbol{H}$ becomes numerically unstable.

In the extreme, an eigenvalue of 0 corresponds to a direction in which the function does not vary at all, i.e., the surface is flat in that direction. In that case, of course, $\boldsymbol{H}$ becomes singular and the iteration rule above can no longer be used[9]. This dicussion illustrates why the performance of the Newton method may vary widely depending on the shape of the objective function. Since the computation of the Hessian and its inversion may be problematic, other approaches (*quasi-Newton*) use various approximations of the Hessian. We review one class of techniques that is particularly important for our applications in the next section.

2.3 Non-Linear Least Squares

A particularly important case is the case in which F has the special form:

$$F(\boldsymbol{x}) = \sum_{j=1}^{m} r_j^2(\boldsymbol{x})$$

The functions r_j are usually termed *residuals*. This special form of the objective function is very common. For example, suppose that we observe the projections $\boldsymbol{p}_j = [x'_j \; y'_j]^T$ in m images of an unknown point $\boldsymbol{P} = [x \; y \; z]^T$. To recover the $\boldsymbol{P}$ from its image projections, we need to find $\boldsymbol{P}$ that minimizes the distance between $\boldsymbol{p}_j$ and $g_j(\boldsymbol{P})$, where g_j is the function that maps a point in the scene to its projection in image j (a non-linear function that is a general form of the perspective projection: $\boldsymbol{p}_j = f_j[\frac{x}{z} \; \frac{y}{z}]^T$.) In other words, we need to minimize the objective function:

$$F(\boldsymbol{P}) = \sum_{j=1}^{m} |\boldsymbol{p}_j - g_j(\boldsymbol{P})|^2$$

This formulation will be central to solving several problems in structure recovery from multiple images. In these problems, there may be hundreds of variables, i.e., hundreds of points $\boldsymbol{P}$, and direct computation of the Hessian may become problematic. Instead, we can use an approximation of the Hessian as shown below.

Let $\boldsymbol{J}$ denote the matrix whose rows are the gradients of the residuals, i.e., $\boldsymbol{J}$ is obtained by stacking the row vectors $\nabla r_j(\boldsymbol{x})^T$, and let $\boldsymbol{r}(\boldsymbol{x})$ denote the column vector $[r_1(\boldsymbol{x}), \ldots, r_m(\boldsymbol{x})]^T$. We can write the gradient of the objective function as:

$$\nabla F(\boldsymbol{x}) = \sum_{j=1}^{m} \nabla r_j(\boldsymbol{x}) r_j(\boldsymbol{x}) = \sum_{j=1}^{m} \boldsymbol{J}^T r_j(\boldsymbol{x}) = \boldsymbol{J}^T \boldsymbol{r}(\boldsymbol{x})$$

Differentiating this expression one more time, we get the Hessian of F:

[9] A third case occurs when the eigenvalues are negative, in which case the point is a saddle point at which the gradient vanishes but is not an extremum of the function.

$$\boldsymbol{H} = \nabla^2 F(\boldsymbol{x}) = \boldsymbol{J}^T\boldsymbol{J} + \sum_{j=1}^{m} \nabla^2 r_j(\boldsymbol{x}) r_j(\boldsymbol{x})$$

The key observation is that only the second term in this expression of $\nabla^2 F(\boldsymbol{x})$ requires the knowledge of the second derivatives. Therefore, a natural approach is to drop this hard to compute second term and to approximate the Hessian by the first term only:

$$\boldsymbol{H} \approx \boldsymbol{J}^T\boldsymbol{J}$$

In particular, replacing this approximation of the Hessian in the update rule used in the Newton method, $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{H}^{-1}\nabla F$, we obtain:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - (\boldsymbol{J}^T\boldsymbol{J})^{-1}\boldsymbol{J}^T\boldsymbol{r}$$

This approach is the *Gauss-Newton* method, which has many advantages. Gauss-Newton does not require the computation of any second derivatives and, in many problems the first term used in the approximation of $\boldsymbol{H}$ is indeed much larger than the second term. It appears that we still need a matrix inversion (and a matrix multiplication to evaluate $\boldsymbol{J}^T\boldsymbol{J}$.) In fact, $(\boldsymbol{J}^T\boldsymbol{J})^{-1}\boldsymbol{J}^T$ is the pseudo-inverse of $\boldsymbol{J}$, as defined in Section 2.1, and can be computed directly from $\boldsymbol{J}$ using more efficient SVD techniques as described in 2.1 or other linear least squares techniques not listed here. The bottom line is that one can avoid completely any explicit computation of $\boldsymbol{J}^T\boldsymbol{J}$ or of its inverse!

Gauss-Newton provides a computationally efficient approach in the case where the objective function is a sum of squared residuals. The fact still remains that Gauss-Newton is based on an approximation of the Hessian. Presumably, if this approximation could be refined, we ought to be able to achieve faster convergence. A general approach to refining the Hessian approximation is the *Levenberg-Marquart* algorithm in which the approximation $\boldsymbol{H} \approx \boldsymbol{J}^T\boldsymbol{J}$ is replaced by:

$$\boldsymbol{H} \approx \boldsymbol{J}^T\boldsymbol{J} + \lambda\boldsymbol{I}$$

In this approximation, λ is a scalar that is adjusted at each iteration based on the rate of decrease of F at prior iterations. The heuristics used for controlling λ are rather involved, but the general idea of the algorithm is that the order of magnitude of the discrepancy between the true Hessian and its approximation is evaluated at each iteration and the Hessian approximation is corrected by adding a diagonal matrix of that magnitude. Levenberg-Marquart combines the good convergence properties of the original Newton method and it avoids completely the computation of the Hessian while maintaining a better estimate than Gauss-Newton.

2.4 Constrained Problems

In their simplest forms, constrained problems involve finding an extremum of an objective function $z = F(\boldsymbol{x})$ under a constraint $G(\boldsymbol{x}) = 0$. The problem is solved by optimizing a combined function, the *Lagrangian*:

$$L(\boldsymbol{x}) = F(\boldsymbol{x}) - \lambda G(\boldsymbol{x})$$

λ is an additional variable, the *Lagrange multiplier*. The gradient of L vanishes at the solution, yielding the general relation' between the gradients of the objective function and of the constraint:

$$\nabla F(\boldsymbol{x}) - \lambda \nabla G(\boldsymbol{x}) = 0$$

For example, if the objective function is $F(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}$, where $\boldsymbol{A}$ is a $n \times n$ matrix, and the constraint is $\boldsymbol{x}^T\boldsymbol{x} = 1$, the relation above between the gradients becomes: $\boldsymbol{A}\boldsymbol{x} - \lambda\boldsymbol{x} = 0$. Therefore, the solution to this constrained problem is an eigenvector of $\boldsymbol{A}$ since $\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x}$. This is basically the Rayleigh Quotient theorem from 1.2.2.