Computational Microscopy

Lars Loetgering (lars.loetgering@uni-jena.de)
Rainer Heintzmann (heintzmann@gmail.com)

January 13, 2025

Contents

[1] also known as *power iteration*

Part I

Mathematical and Programming Preliminaries

Chapter 1

Introduction to Matlab, Python, and Julia

With each of the programming languages used in this book, the single best advice to learn them is to use them: **Write your own code**! If you ever read a scientific book, wondering a few years later, how come you remember little about the subject, a possible answer could be: you simply did not internalize the material because you didn't use it. Have you ever tried learning a (spoken) language yourself without being exposed to a native speaker or visiting the country were the language is used? Probably that didn't go well. Have you learned cooking by reading a cooking book? Probably not. Have you learned riding a bycicle by watching others do so?

Despite University education being a rather theoretical experience at times, a working knowledge of any subject comes from hands-on experience. In contrast, many University courses books take a code-free approach. You may have learned about linear algebra and partial differential equations as well as electromagnetism and quantum mechanics, but how confident are you computing eigenfunctions of an unfamiliar linear operator or pulling off a Bessel function when a new problem crosses your way?

We live in a time were data-driven methods are prevalent in cutting-edge research. Many early-stage engineers and PhD students are simply not prepared to bridge the gap between the material they have learned in lectures and the practical problems they encounter once they start specializing on a topic. You take on a new position and suddenly have to learn programming, obtain domain-specific knowledge of your field, and come up with your own solutions. Then you talk to your colleagues and you make the frustrating discovery that you can not even help each other because you do not speak the same programming language: bad luck, did you learn the "wrong" programming language?

In light of the rapidly evolving number of programming languages, it is probably wishful thinking to hope one could get through an entire career learning only a sinlge programming language. Therefore we decided to write this book in a trilingual approach using Matlab, Python, and Julia. Our hope is that the reader develops an open attitude towards using multiple programming languages throughout the course of this book and identifies each language's individual strengths. Accordingly, this chapter reviews basics in each of the three aforementioned programming languages as well as further resources to

comprehensive manuals and literature. If the reader has no prior programming experience, we strongly recommend reading this chapter before moving on. If the reader is already familiar with programming in one of the languages covered here, a time-saving approach could be to skip this introduction and get familiar with the other programming languages by simply looking at the codes and examples in the upcoming chapter. However, let us repeat one more time: programming skill comes from actually programming yourself. Frequently distrust yourself when you think you have understood a code solution to a particular problem. Think of a variaton to the same problem and code up the solution yourself. Often this is harder than expected, but worth every minute spent[1].

1.1 Matlab

Matlab (from **mat**rix **lab**oratory) is a high-level programming language. Developed originally by Cleve Moler, its initial intent was to aid in the computation of matrix calculations. As the functionalities expanded, Matlab became a programming language capable to solve mathematical problems that go far beyond the scope of linear algebra. Matlab itself is written in the more low-level programming language C. The reader could be indifferent about the inner workings of Matlab, if it did not play crucial role for performance. There are compiled languages, such as C(++), which force the user to think about data types ahead of time. Depending on the data types, the code is compiled differently, as computations involving different data types require different precision. On the other end, there are interpreted languages, which do not require compilation and run code line by line. Matlab is residing somewhere in between these extremes. Certain operations are precompiled, while the code can be run without worrying too much about data types. This approach is called just-in-time compilations. As soon as the programmer writes his or her own functions, just-in-time compilation can become a bottleneck. If, however, the data types and sizes of the input arrays are fixed, Matlab allows to speed up self-written functions through precompilation in C, via so-called mex-files (described in subsection ??).

1.1.1 Integrated development environment

An integrated development environment (IDE) comes along with an installation of Matlab. Besides this, there are other ways to run matlab code, such as calling scripts from a shell (the command line interface in an operating system) and compiled executables, each with variable input. In this book we will only address codes run from the Matlab IDE.

It is also worth mentioning that alternative programming languages exist - such as Julia (covered below), Octave and Scilab - which to a certain degree parallel the syntax in Matlab. Code independent of Matlab's own toolboxes is oftentimes easily transferable between the aforementioned languages.

In the Matlab IDE,

- use arrows (up and down) to scroll through command history,

[1] Conrad Wolfram gave an inspiring TED talk about the use of computers in education that we highly recommend: https://www.youtube.com/watch?v=60OVlfAUPJg&t=413s&ab_channel=TED.

- use tab key for autocompletion,
- use semicolon at the end of a command to suppress output,
- (this one drives your colleagues mad if not followed) if you are working in an institute with limited amount of Matlab (toolbox) licenses, save results if needed, and leave script using `exit` or `quit` at the end of execution.

1.1.2 Style recommendations

Matlab does not provide an official style guide (contrary to other programming languages such as Python and Julia). Certain conventions are recommended here - although they are not mandatory. These recommendations are partially adapted from the book by Higham and Higham [22] [2]:

- Use camelCase[3] for descriptive variable names,
- one statement per line,
- indentations are used after conditionals (if-else, switch-case) and loops (for, while),
- follow commas by a space,
- always write a short description of a function (= first comment after function, which may be called with `help functionName`),
- try using descriptive function names. For instance a function that calculates the mean along each row of a matrix could be named `rowMean.m`, not `rm.m`, the latter of which gives no clue whatsoever to a colleague who has to reuse the function.

With such a minimal set of rules, the stage is set for anarchy. Some of the readers may find a stricter set of rules (as given in Python and Julia) better to enhance uniformity and thus readibility of code between co-workers.

1.1.3 Basic Datatypes and constants

There are various data types in Matlab. The default numerical data type is `double`, which represents a floating point number by 64 bits. For instance,

```
a = 1/3;
whos a
disp(a)
```

creates approximates the fraction 1/3 by a floating point number with 64 bits. Another typical data type is `single`, which approximates a numerical value by a 32 bit floating point number. Notice that even integers default to double precision. For example,

```
a = 1;
```

[2] Higham, Desmond J., and Nicholas J. Higham. MATLAB guide. Society for Industrial and Applied Mathematics, 2016.

[3] https://en.wikipedia.org/wiki/Camel_case

saves $a = 1$ at double precision. Division by 0 results in an `NaN` (not a number). The code

```
a = 1;
counter = 0;
while ~isinf(a)
        counter = counter + 1;
        a = a*2;
end
```

produces counter = 1024 and $a = \infty$ (`a=inf`). Complex numbers are represented by the data type complex, which is simply a tuple of doubles by default. This behaviour can be altered, when the constituting numbers each have another (but the same) data type. For instance,

```
x = double(2);
y = double(3);
z = x+i*y
whos z
```

produces a complex double, while

```
x = single(2);
y = single(3);
z = x+i*y
whos z
```

produces a complex single.

There are constants that should not be overwritten, inlcuding π =`pi` and the imaginary unit $\sqrt{-1}$ =`i`. Both `i` and `j` may by default be used in Matlab to reference the imaginary unit. We recommend never overwriting them in for-loops. If i and j are needed for loops, use for instance `iLoop` and `jLoop`, or similar running variables that avoid overwriting the imaginary unit. The command `eps` by default gives distance from 1.0 to the next largest floating point number at double precision. In addtion `eps(x)` is the distance from the variable x to the next largest floating point number. For instance, `eps(double(1.0))`$\approx 2.2204 \cdot 10^{-16}$, while `eps(single(1.0))`$\approx 1.1921 \cdot 10^{-7}$. In general,

$$\frac{|x - \text{float}(x)|}{x} \leq \epsilon(x). \tag{1.1}$$

1.1.4 Toolboxes

A full Matlab installation comes with a variety of useful toolboxes that provide functionality for various scientific computing tasks, including hardware control (e.g. image acquisition toolbox), machine learning (deep learning toolbox), curve fitting (curve fitting toolbox), and image processing (image processing toolbox) - among others. Due to the commercialization of Matlab, these toolboxes require license fees. In particular, larger institutions oftentimes select a limited number of licences for a selection of toolboxes - hopefully aligning with the needs of the employees. It happened to one of the authors

that working in an institution with limited number of toolbox turned out to be frustrating experience at times. Certain employees would hoard Matlab toolbox licenses, by not typing the command `quit` or `exit` at the end of a long computation thereby fetching those limited licenses (and making sure they are available for themselves the next day but potentially not for others). One can only speculate that those were the same people that hoarded toilet paper at the onset of the COVID-19 pandemic. All this is to say, yes, if you are a programming beginner (and even if you are not), consider using a non-commercial programming language to avoid these problems. This decision may turn out as a kind of toilet paper flat rate in a pandemic. The future is with non-commercial scientific computing languages.

1.1.5 Arrays

The early versions of Matlab (from **mat**rix **lab**oratory) were developed for Linear Algebra tasks. Hence much of the syntax in matlab is such that matrix operations are made to be easy. we may declare a 5×5 matrix of zeros using the command

```
A=zeros(5,5);
```

and a matrix of ones using the command

```
A=ones(5,5);
```

where the semicolon is used to suppress output displayed in the command window. A 5×5 identity matrix, in Linear algebra text books often denoted $\boldsymbol{I}$ ("eye"), can be declared as

```
A=eye(5,5);
```

The command

```
A=[];
```

declares an empty matrix. In rare occasions when the final size of a computation is not foreseeable, an array may be declared as empty and expanded depending of the computations that follow. Changing the size of an array excessively (for instance at every iteration of a for-loop) should be avoided for performance reasons.

An important distinction is the difference between matrix operations and element-wise opeartion. The code

```
A = ones(3,3);
B = ones(3,3);
C = A*B
D = A.*B
```

produces the matrices

$$\boldsymbol{C} = \begin{bmatrix} 3 & 3 & 3 \\ 3 & 3 & 3 \\ 3 & 3 & 3 \end{bmatrix} \tag{1.2}$$

and

$$\boldsymbol{D} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}. \tag{1.3}$$

In the first case

$$\boldsymbol{C} = \boldsymbol{A} \cdot \boldsymbol{B} \tag{1.4}$$

is the result of a matrix multiplication ($C_{i,j} = \sum_k A_{i,k} \cdot B_{k,j}$), while in the second case we get an element-wise multiplication

$$\boldsymbol{D} = \boldsymbol{A} \odot \boldsymbol{B}, \tag{1.5}$$

which is also called Hadamard multiplication ($C_{i,j} = B_{i,j} \cdot C_{i,j}$). In Linear Algebra we often (but not always) want matrix multiplications, while in image processing we often (but not always) deal with Hadamard multiplications.

1.1.6 The colon operator

The colon operator (:) can be used to produce arrays in a specified range. For example

```
a = 2:6
```

produces the row vector $\boldsymbol{a} = [2, 3, 4, 5, 6]$. In addition, we may use the colon operator to access arrays,

```
a(1:3)
```

gives the row vector $[2, 3, 4]$. Notice that Matlab uses 1-based indexing, meaning that the first element in the array `a` is `a(1)`. We will see many more examples as we go through specific applications in later chapters.

1.1.7 Functions

A recurrent theme in any programming language is to identify code snippets that may be recycled. This may be done by defining self-written functions containing functionality to be re-used at a later occasion. In Matlab we may write a self-written function using the keyword function at the beginning of a script. Here is an example

```
function r = sum2(x)
% r = sum2(x)
% two-dimensional sum

r = sum(sum( x ));

return
```

Listing 1.1: Two-dimensional sum: sum2.m

This function can be used in image processing, where we often need to calculate the sum over all pixels in the image. Alternatively, we can simply use

```
sum(x(:))
```

to achieve the same goal.

Another example is `circ.m`:

```
function r = circ(X, Y, D)
% r = circ(X, Y, D)
% D: diameter
% X, Y: meshgrid (in x/y-direction)

r = single((X.^2+Y.^2) <= D^2/4);

end
```

Listing 1.2: circ.m

This function computes a circular image of diameter `D` and casts the boolean result (0 or 1) to the data type single. The comment lines following the keyword are displayed when calling `help circ` from the command line. Make sure you understand this function, as it is used often throughout this book.

Notice that the scope of an m-function is local, meaning that only the value of the returned variable is passed on to outside the function (where it may have a different name).

1.1.8 Loops

Consider the sum

$$(1-q)\sum_{k=0}^{N} q^k = \sum_{k=0}^{N} q^k - \sum_{k=0}^{N} q^{k+1} \tag{1.6}$$

$$= \sum_{k=0}^{N} q^k - \sum_{k=1}^{N+1} q^k \tag{1.7}$$

$$= 1 - q^{N+1} \tag{1.8}$$

or

$$\sum_{k=0}^{N} q^k = \frac{1-q^{N+1}}{1-q}. \tag{1.9}$$

The same result may be obtained using a for-loop

```
% maximum power
N = 10;
% base
q = 0.5;
% sum
s = 0;
for k = 0:N
    s = s + q^k;
end

disp(['for-loop result: ', num2str(s)])
```

```
12 disp(['analytic result: ', num2str( (1-q^(N+1))/(1-q) )])
```

Listing 1.3: forLoopExample.m

which returns the result 1.999[4] in each case.

1.1.9 Further Resources

As a general reference, we recommend the book by Higham and Higham [22]. The book by Brunton and Kutz [10] gives an excellent resource for matlab code on scientific computing, machine learning, and control. Another book very much recommended is the book by Quateroni et al. [40]. Yair Altman wrote a book on speeding up Matlab code [2]. His export_fig toolbox is an excellent tool to export Matlab figures into publication-quality output. We would also like to mention the older but still useful book on image processing by Woods et al. [49]. The books by Schmidt [41] and Voelz [47] are outstanding references for numerical aspects of wave propagation and Fourier Optics.

1.2 Python

Python is an interpreted, high-level programming language. It was originally developed by Guido van Rossum in the 1980s and has become one of the most widespread programming languages. Python is currently one of the most popular programming languages. Its core is relatively lightweight and functionality is extended via the addition of packages. Thus a code often starts with the import of a variety of packages, for instance *NumPy* which supports multi-dimensional arrays and a variety of mathematical functions, or *Matplotlib* which supports tools for visualization.

1.2.1 Integrated development environment

In Python the user has the choice between a variety of IDEs. The Jupyter Notebook is a browser-based IDE, which integrates code with text (markdown) and visualization tools. The Jupyter Notebook can be used with Python, but also with other languages including Julia (discussed in section 1.3). A more recent development is Jupyter Lab, which allows you to open and run code in a browser-based console. Other popular Python IDEs are Spyder and PyCharm. If you are using Python for the first time, we recommend installing Anaconda, which is a Python distribution that comes with many packages that you are likely to use. You can then integrate new packages into your Python distribution using the conda pacakge manager. From the Anaconda prompt line, you can type `ipython` to open an interactive Python environment.

1.2.2 Style recommendations

A style guide is provided in the PEP 8 document, which contains recommendations for readible Python code. The following table contains some naming conventions from PEP 8 [46]:

[4] This is approximated to 4 digits.

file type	recommendation
Modules	Modules should have short, all-lowercase names.
Function	Function names should be lowercase.
Variable	Variable names should be lowercase.
Constant	Constants should be all-uppercase.

Table 1.1: PEP 8 style recommendations.

Module	description
Numpy	Multi-dimensional arrays, mathematical functionality similar to Matlab
Matplotlib	Various tools for visualizing functions, data
SciPy	Optimization, special functions, signal and image processing
Pandas	Data analysis and manipulation
TensorFlow	Machine Learning, Neural Networks
CuPy	NumPy and SciPy GPU support
JAX	Automatic differentiation

Table 1.2: Frequently used modules.

A module is a script with a collection of functions and possibly variables. Collecting functions that are often re-used in modules makes it easy to recycle code and save work. Generally, it is advisable to choose descriptive variable and function names that aid readibility. The string

```
s='I love chickpeas'
```

may be split into three words using

```
a, b, c = s.split()
```

Using short variable names like this is not recommended. A more readible version would be

```
sentence='I go home'
subject, verb, destination = s.split()
```

1.2.3 Modules

In Python we may import modules using the `import` command. Table 1.2 provides a list of modules that we will use frequently throughout the remainder of this book.

1.2.4 NumPy arrays

The following subsections follow the organization of the NumPy chapter in [23].

1.2.4.1 Importing NumPy

The first module that we are going to use is `numpy`, which provides support for multi-dimensional arrays and various functions to manipulate the latter. We can import this

module and assign an abbreviation (np) using the command

```
import numpy as np
```

1.2.4.2 Basic properties of ndarrays

We declare a simple ndarray object using the command

```
data = np.array([1,2,3,4])
```

Basic properties of this ndarray may be inferred using the commands

```
data.shape
data.dtype
data.nbytes
```

which result in the outputs

```
> (4,)
> dtype('int32')
> 16
```

respectively. A simple matrix is declared using the following syntax

```
data = np.array([[1,2,3],[4,5,6],[7,8,9]])
```

1.2.4.3 Data types

We can explicitly define the output data type.

```
data = np.array(np.array([-1,0,9]),dtype=float) data
```

returns

```
> array([-1.,  0.,  9.])
```

This data may be cast into another type using the command

```
data = data.astype(complex)
```

It is now possible to compute the square root of each entry

```
data = np.sqrt(data)
data
```

gives the result

```
array([0.+1.j,  0.+0.j,  3.+0.j])
```

1.2.4.4 Built-in functions for array generation

Numpy provides similar functions for array generation as previously discussed in Matlab. Examples are

```
np.linspace(1,9,9)
np.zeros(5)
np.zeros((5,5))
np.ones((3,3))
np.eye(3)
```

which result in

```
> array([1., 2., 3., 4., 5., 6., 7., 8., 9.])
> array([0., 0., 0., 0., 0.])
> array([[0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0.]])
> array([[1., 1., 1.],
         [1., 1., 1.],
         [1., 1., 1.]])
> array([[1., 0., 0.],
         [0., 1., 0.],
         [0., 0., 1.]])
```

1.2.4.5 Indexing and Slicing

>

Python uses 0-based indexing. For instance, the code

```
x = np.linspace(0,10,11)
x
```

results in the output

```
> array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.])
```

Then

```
x[0]
```

yields the output

```
> 0.0
```

The nth entry is given by

```
n = 5
x[n-1]
```

producing the output

```
> 4.0
```

Other possibilities

```
# last entry
x[-1]
# first five entries
x[:5]
# m:n:s = start index : end index : increment
x[1:-1:2]
```

produces

```
# last entry
> 10.0
# first five entries
> array([0., 1., 2., 3., 4.])
# m:n:s = start index : end index : increment
array([1., 3., 5., 7., 9.])
```

1.2.4.6 Views

Manipulating subsets of NumPy arrays oftentimes results in *views*. These views are not independent copies of the data, but the data *itself*. Hence one needs to be a little bit more careful than in Matlab when manipulating such views. Example:

```
N = 2
x = np.linspace(-N,N,2*N+1)
X, Y = np.meshgrid(x,x)
X
```

results in

```
> array([[-2., -1.,  0.,  1.,  2.],
         [-2., -1.,  0.,  1.,  2.],
         [-2., -1.,  0.,  1.,  2.],
         [-2., -1.,  0.,  1.,  2.],
         [-2., -1.,  0.,  1.,  2.]])
```

Then

```
Z = X
Z[1:,:] = 0
# show result for Z
Z
# show X
X
```

yields

```
# show Z
> array([[-2., -1.,  0.,  1.,  2.],
         [ 0.,  0.,  0.,  0.,  0.],
         [ 0.,  0.,  0.,  0.,  0.],
         [ 0.,  0.,  0.,  0.,  0.],
         [ 0.,  0.,  0.,  0.,  0.]])
# show X
> array([[-2., -1.,  0.,  1.,  2.],
         [ 0.,  0.,  0.,  0.,  0.],
         [ 0.,  0.,  0.,  0.,  0.],
         [ 0.,  0.,  0.,  0.,  0.],
         [ 0.,  0.,  0.,  0.,  0.]])
```

We can omit this bahavior by explicitly copying the `X` into a new variable `Z`:

```
X, Y = np.meshgrid(x,x)
Z = X.copy()
Z[1:,:] = 0
# show result for Z
Z
# show X
X
```

gives

```
# show Z
> array([[-2., -1.,  0.,  1.,  2.],
         [ 0.,  0.,  0.,  0.,  0.],
         [ 0.,  0.,  0.,  0.,  0.],
         [ 0.,  0.,  0.,  0.,  0.],
         [ 0.,  0.,  0.,  0.,  0.]])
# show X
> array([[-2., -1.,  0.,  1.,  2.],
         [-2., -1.,  0.,  1.,  2.],
         [-2., -1.,  0.,  1.,  2.],
         [-2., -1.,  0.,  1.,  2.],
         [-2., -1.,  0.,  1.,  2.]])
```

1.2.4.7 Reshaping and Flipping

An array is reshaped using the `reshape` command

```
x = np.linspace(1,9,9)
A = x.reshape(3,3)
A
```

results in

```
> array([[1., 2., 3.],
         [4., 5., 6.],
```

```
        [7., 8., 9.]])
```

Similar to Matlab, NumPy provides flipud and fliplr commands

```
# flip upside down
np.flipud(A)
> array([[7., 8., 9.],
        [4., 5., 6.],
        [1., 2., 3.]])
# flip left right
np.fliplr(A)
> array([[3., 2., 1.],
        [6., 5., 4.],
        [9., 8., 7.]])
```

1.2.4.8 Arithmetic Operations

Let us define two NumPy arrays x and y

```
x = np.linspace(1,9,9).reshape(3,3)
x
> array([[1., 2., 3.],
        [4., 5., 6.],
        [7., 8., 9.]])
y = np.linspace(4,12,9).reshape(3,3)
y
> array([[ 4.,  5.,  6.],
        [ 7.,  8.,  9.],
        [10., 11., 12.]])
```

With this we can perform element-wise addition

```
x + y
> array([[ 5.,  7.,  9.],
        [11., 13., 15.],
        [17., 19., 21.]])
```

and subtraction

```
x - y
> array([[-3., -3., -3.],
        [-3., -3., -3.],
        [-3., -3., -3.]])
```

Likewise, we can compute element-wise multiplication

```
x * y
> array([[  4.,  10.,  18.],
        [ 28.,  40.,  54.],
        [ 70.,  88., 108.]])
```

and division

```
np.round(x/y, decimals = 4)
> array([[0.25   , 0.4    , 0.5    ],
         [0.5714, 0.625  , 0.6667],
         [0.7    , 0.7273, 0.75   ]])
```

where we rounded the latter to 4 decimals. Matrix-multiplication can be performed using the @ symbol

```
x @ y
> array([[ 48.,  54.,  60.],
         [111., 126., 141.],
         [174., 198., 222.]])
```

Another example illustrates matrix-vector multiplication

```
A = 1.*np.array([[1,2,3],[0, 1, 2],[0,0,1]])
A
> array([[1., 2., 3.],
         [0., 1., 2.],
         [0., 0., 1.]])
x = np.array([1,1,0])
x = x[:,np.newaxis]
x
> array([[1],
         [1],
         [0]])
```

Then matrix-vector multiplication is given by

```
A @ x
> array([[3.],
         [1.],
         [0.]])
```

We may obtain the element-wise square using

```
A**2
> array([[1., 4., 9.],
         [0., 1., 4.],
         [0., 0., 1.]])
```

1.2.5 Functions

The following code imports NumPy and Matplotlib, defines and calls a circ function, and plots the result of the latter:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def circ(X,Y,D):
```

```
    return 1. * ( (X**2 + Y**2) <= D**2/4 )

N = 2**8
dx = 1
L = N*dx
x = np.linspace(-N/2,N/2,N)
X, Y = np.meshgrid(x,x)
f = circ(X,Y,L/4)
plt.imshow(f)
```

Listing 1.4: showCircFunction.py (Python)

1.2.6 Broadcasting

1.2.7 Loops

1.2.8 Further Resources

1.3 Julia

Julia is a modern computer language which contains many of the concepts of Python and Matlab but aims for on-the fly compilation and high performance.

1.3.1 Basic Datatypes and constants

Here is a list of datatypes. It should be evident from their name, what they refer to:

`Float32`, `Float64`, `UInt8`, `Int8`. You can find out the type of a variable using `typeof()`. For example, `typeof(1.0)` yields `Float64`. There are ways to specify the type of a number when typing it. For instance, `typeof(1f0)` yields `Float32` and `typeof(1im)` yields `Complex{Int64}`. In the latter case you already see an example of a more complicated parameterized type, of which you will see more below.

Basic constants are already in the `Base` environment, which you can find out with the help of a macro `@which` π, where you type the symbol of pi by typing `\pi` followed by pressing the tab key. The @ sign in `@which` indicates that this is a macro, it yields the following result: `Base.MathConstants` indicating that the symbol of pi is defined in Base and thus is automatically present in Julia. Note that `\pi` can also be used as a postfix operator. For example, typing 2π yields `6.283185307179586` which can be a useful abbreviation. Similarly you can use the imaginary constant `im` often written as `1im`.

1.3.2 Modules, Packages

Larger units of code are often collected in packages. From the interactive command-line REPL (Read Eval Print Loop), you can reach the package management system by typing `]` and then install a package named `PackageName` via `add PackageName` into the currently active environment. Each environment "lives" in a directory, thus you can change the environment by chaging directory in the REPL

`cd(raw"some/directory")`

and activate it from the package manager via

```
cd(raw"path-to-my-directory").
```
Note that the prefix `raw` before the string indicates that you want to ignore control characters in the string, which is particularly useful on Windows systems, as the backslashes are otherwise wrongly interpreted. If the package is a development version or not yet present in Julia's official package system, but you know its github link, you can simply add it via its GitHub link:
```
add http://github.com/BionanoImaging/ComputationalImaging.jl
```
from within the package manager and the package will automatically be downloaded and installed somewhere deep in your local .julia folder, which resides in your home folder (on Windows the folder one above "Documents"). Yet this does not allow you to modify the package. If you wish to modify it, simply clone (or download) it into a folder of your choice, change directory into this folder and type
```
dev .
```
in this folder, which will add this package for development purposes (now you can modify it!) to the currently active environment.

If you want to create your own package, you should surround your code by
```
module MyPackage end
```
and export the functions or variables you want to be visible to the outside once the package is used via
```
export MyFunctionOrVariable.
```

1.3.3 Arrays

At first sight arrays in Julia feel a lot like arrays in Matlab, example
```
A = zeros(5,5); A
```
Arrays are indexed with 1-based indexing:
```
A[1,1]=1
```
you can also allocate arrays fast leaving the uninitialized
```
A=Array{Float64}(undef,5,5).
```
Here `undef` indicates that the values in the array are currently undefined (representing whateven is currently in that memory). Here you already can see an example of a datatype `Array{Float64}` which is parameterized by another type `Float64`.
The intricate typesystem of Julia has parameters, which are determined at compile time. This allows Julia to determine the memory requirement for each element of the array and therefore determine also all that is necessary for indexing it and other important things. Since also integer numbers can be used to parameterize a type, Julia uses such an integer parameter, as the dimension of an array, to work with multi-dimensional arrays. Even sizes of arrays can be fixed at compile time to increas performance, which is the case for the module `StaticArrays`.

Julia's typesystem is quite smart as you can see by the following example:
```
typeof([1,2,3,4])
```
yields
```
Vector{Int64},
```
whereas
```
typeof([1,2,3.0,4])
```
yields

`Vector{Float64}`
and
`a=[1,2,3.0,"hello"];typeof(a)`
yields
`Vector{Any}`,
where the individual types stay dynamic:
`typeof(a[1])`
outputs `Int64`. Note that arrays can (just as in Matlab) be costructed by surrounding a comma-separated list by square brackets, but opposed to Matlab the array access is also using square brackets (just as in Python).

The indices of an array start at `1` and are interpreted such that the first index refers to the most inner dimension of the array and the last index refers to the outer dimension, similar to Matlab, but opposed to the convetion in Python und NumPy. During indexing, you can use the `colon` operator `:`, very much the same way as is in Matlab. For example,
`A[:,2,end,:]`
yields a two-dimensional array which is a subslice spanning over the full range of dimension one and four but taken at position two of dimension two and the last index of dimension three. Note that as opposed to NumPy negative indices or zero indices are not allowed for standard arrays, but you can use the keyword `end` to refer to the last index:
`A[end/2+1,end/2+1]`
which would refer to the value at the center position of the array `A`.

Indexing an array with a single number leads to the multi-dimensionality being dropped. If
`B = [1 2 3; 4 5 6]`,
then
`B[:]`
returns
`[1,4,2,5,3,6]` reflecting the the matrix was specified row by row but arranged in memory with the rows as the innermost index.
In multidimensional image processing it is often useful to keep the dimensions as specified in the original array. This can be achieved by specifying a range with size one, instead of a single number:
`size(rand(5,6,7)[:,3:3,:])`
yields
`(5, 1, 7)`,
while
`size(rand(5,6,7)[:,3,:])`
returns the tuple
`(5, 7)`. By using multiple semicolons you can stack along a higher dimension. E.g. `size([3;;;4;;;5])` yields `(1, 1, 3)`.

Multidimensional arrays can, similar to Matlab, always be indexed by one-dimensional indexing, in which case the array is interpreted as the array being flattened.

```
julia> A = reshape(1:100,10,10);

julia> A[52]  52

```

```
5  julia> transpose(A)[52]  16
```

Not every operation in Julia immediately leads to a calculation. Sometimes it just leads to a different view of the data with a modified indexing scheme. In fact the `reshape` and `transpose` commands are examples that just create a view, whereas accessed to subarrays leads to copy operations, but these can be avoided via the `@view` macro:

```
1  julia> typeof(A)
2  Base.ReshapedArray{Int64, 2, UnitRange{Int64}, Tuple{}}
3
4  julia> typeof(transpose(A))
5  Transpose{Int64, Base.ReshapedArray{Int64, 2, UnitRange{Int64}, Tuple{}}}
6
7  julia> typeof(A[2:3,2:3])
8  Matrix{Int64} (alias for Array{Int64, 2})
9
10 julia> typeof(@view A[2:3,2:3])
11 SubArray{Int64, 2, Base.ReshapedArray{Int64, 2, UnitRange{Int64}, Tuple{}},
12          Tuple{UnitRange{Int64}, UnitRange{Int64}}, false}
```

As you can see in the first and second lines, the array A is not a simple array but already a different (reshaped) view of an underlying array and the transpose of it is a view of a view of the underlying array. Only by accessing a range of values are we forcing Julia to evaluate the view and store a new result (lines 7-8). However, this can be avoided by using the macro `@view` before accessing with a range (lines 10-11). You can also index an array with a vector (a 1D array)
`rand(10,10)[[1,5,7,22]]`,
in which case a vector of array entries is returned. Arrays can be reduced over an arbitrary number of dimensions: `a=rand(10,10,10); sum(a,dims=(2,3))`,
which returns an array of the same number of dimensions, but the reduced dimensions have singleton size, i.e. size one. This can be very useful to be able to use broadcasting (see below) with the orginal array. For example
`a .- minimum(a,dims=(1,2))`
would subtract the minimum of each plane with `a` being a three-dimensional array. Note that the `dims` argument can be a single number or a tuple (see below).

Arrays are mutable, which means that they can be modified
`a=rand(10,10,10); a[2,3,4]=100.0`.
Modifying arrays, matrices and vectors (which are special cases of `Array`) does not only include modifying its entries but also allows to append or delete an entry: `a=Float64[]; push!(a,42); a`. The `push!()` command pushes an object into a list (a vector). The exclamation mark at the end of the funciton `push!()` is really just a part of its name, however by convention it is agreed that all functions which end by an exclamation mark are functions which modify their input. In this case the input vector `a` is modified by appending the value 42 to that vector. When declaring `a`, we specified the type of the array for performance reasons, but one could also use `a=[];` which would allocate an empty vector of type `Any`. You can obtain the type of the elements of an array `a` via `eltype(a)`. To delete an element you can use `deleteat!(a,ind)` with `ind` referring to the index to delete from the array.

1.3.4 Tuples

Tuples are very similar to arrays but they are immutable. This has the large advantage that the compiler can very often infer the size of the tuple, which makes Julia code very fast.

Tuples are often used as arguments to functions as this allows the compiler produce (without you noticing it) a specialized version of the function for a tuple of known size, which then involves very little overhead.

Tuples are constructed by enclosing a comma separated list by round brackets `mytuple=(4,5.0)`. If you construct a tuple with mixed types, this is reflected in the datatype.
`typeof(4,5.0)`
yields
`Tuple{Int64, Float64}`.
Even though tuples are immutable, you can still use them for calculations and modify them during copy operations: `(3,4).+(5,6)./2`. Note that you have to stricktly use the broadcasting operators `.+, .-, .*, ./` etc. to indicate elementwise application for tuples, whereas for vectors you can also use the `+, -, *, /` symbols as long as you are not adding or subtracting a scalar to a vector.

1.3.5 Writing Multidimensional Algorithms

Writing multidimensional algorithms sound easy at first until you realize that you cannot use ordinary indexing, if you do not know the number of dimensions you are dealing with. Luckily in Julia this is fairly simple and straight forward to deal with, if you know about the splatting operator `...` which you can append to a tuple. This will transform the list of elements in the tuple individual arguments of the function or array indexing operation: `a=rand(10,10,10);idx=(2,4,3); a[idx...]`.
Note that the `...` operator can also appear in a different context inside a function declaration where it is called "slurping operator" and can "slurp" a bunch of trainling arguments into a single tuple. To demonstrate how "splatting" can be very useful, here is an algorithm which "draws" a shape specified by an indexed curve into an N-dimensional dataset:

```
# Julia code: ND_curve.jl
""" ND_curve(data, curve=(x)->(size(data).-1).*x.+1, arange=range(0,1,
    length=maximum(size(data))); val=one(eltype(data)))
    plots a `curve`, a parametric function, into a multidimensional dataset
    by setting the pixels
    closest to the curve to the value defined by the named argument `val`.
    `arange` specifies the range at each point of which to sample the `curve`.
"""
function ND_curve(data, curve=(x)->(size(data).-1).*x.+1,
                  arange=range(0,1,length=maximum(size(data))),
                  val=one(eltype(data)))
    for p in arange
        dp = curve(p) # obtain a position as tuple
        dp = round.(Int64, dp) # nearest integer
        data[dp...] = val # mark the pixel at the N-dimensional coordinate dp
    end
    data
end

```

```
20  data = ND_curve(zeros(10,5)) # draws a diagonal in a 10x5 array
```

Listing 1.5: Splatting of tuples example

Notice that we provided default arguments for the function argument two and three and that these default arguments can use previous arguments to calculate default values. In this case the default is a curve, which spans the N-dimensional array and paints a diagonal.

1.3.6 Functions

Every functional unit you program should be captured into a function such that it can be reused at a later time. In Julia, functions can be specified in several ways
`function sqr(input) return input^2; end`
or
`sqr(input) =input^2`
or
`sqr= (input) -> input^2` ,
the latter definition being especially useful when wanting to define an anonymous function (a function without a name) for example within the call syntax of another function
`map((x)->x^2, rand(10,10))`
which applies the square operation to each element of the 5×5 array of random numbers.

1.3.7 Multiple Dispatch

1.3.8 Broadcasting

Broadcasting is a mechanism that allows you to work with arrays where singleton (or non-existing) dimensions are automatically applied to all corresponding entries along that dimension in a second array. In Julia broadcasting is indicated by appending a `.` to a function or inline operator. E.g. You can add a vector to each row of a matrix: `[1 2;3 4] .+ [1,2]` yields

2x2 Matrix{`Int64`}:
`2 3`
`5 6`

You can add a number to a tuple: `(1,2,3).+0.5` yields `(1.5, 2.5, 3.5)`. You can also apply any function that accepts a single argument to each element of a collection: `sin.((0,\pi /2,\pi))`, note the dot after the function name, yields
`(0.0, 1.0, 1.2246467991473532e-16)`.

1.3.9 Loops

Multidimensional loops are very easy to write down in Julia, as you can see by the following example:

```
1  julia> for x=1:2, y=1:2
2      println("I am at postion $x $y")
3      end
4  I am at postion 1 1
```

```
I am at postion 1 2
I am at postion 2 1
I am at postion 2 2
```

This nomenclature is particularly useful in generator expressions (see below):

1.3.10 Generators

Generators are a way to capture algorithms without executing them. Look at the following example:

```
julia> G = ((x,y) for x=1:4, y=1:4)
Base.Generator{Base.Iterators.ProductIterator{
Tuple{UnitRange{Int64}, UnitRange{Int64}}},
var"#39#40"}(var"#39#40"(),
Base.Iterators.ProductIterator{Tuple{UnitRange{Int64},
UnitRange{Int64}}}((1:4, 1:4)))

julia> collect(G)
4x4 Matrix{Tuple{Int64, Int64}}:
(1, 1)  (1, 2)  (1, 3)  (1, 4)
(2, 1)  (2, 2)  (2, 3)  (2, 4)
(3, 1)  (3, 2)  (3, 3)  (3, 4)
(4, 1)  (4, 2)  (4, 3)  (4, 4)
```

The first line creates a generator `G` which only stores the algorithm how to calculate something. It can then further be used or, as in the second line, simply be evaluated via the `collect()` function.

1.3.11 Performance and Type Stability

To be fast, the compiler needs to be able to infer the types. This is particularly important for numerical calculations. If you write a function it is good practice to infer the datatype that should be calculated with, from its arguments. To this aim the previously mentioned function `eltype()` can be useful. Basic math function are already written such that they preserve the datatype. You can see this for example by calling the sine function with the same argument of different types:

```
julia> sin(1.0)
0.8414709848078965

julia> sin(1f0)
0.84147096f0

julia> sin(1+0im)
0.8414709848078965 + 0.0im
```

If Julia cannot figure out a type beforehand, it will assume the type to be `Any`, which can have severe performance limitations.

To find out whether your program is type-stable, the `@code_warntype` macro can be very helpful:

```
julia> function foo(a)
```

```
2  if a
3      return 1
4  else
5      return 2.0
6  end
7  end;
8
9  julia> @code_warntype foo(true) Variables   #self#::Core.Const(foo)   a::Bool
10 Body::Union{Float64, Int64}
11 1 -     goto #3 if not a
12 2 -     return 1
13 3 -     return 2.0
```

The red color (not shown above) in the expression `Body::Union{Float64, Int64}` indicates that this function is not types-stable as the return type depends on how it is called, which the compiler cannot figure out beforehand. To avoid this one could have specified the return type already in the function declaration: `function foo(a)::Float64` which would have forced the `return 1` to be automatically converted to the preferred type making this code now type stable, as the compiler always can be sure about the return type.

1.3.12 Integrated development environment

1.3.13 Further Resources

Chapter 2

Linear Algebra

In this chapter we introduce vectors and matrices as well as the operations connecting them. With the increasingly important role of data-driven techniques in applied optical engineering, particular emphasis is placed on formulating and solving linear models. This oftentimes results in the decomposition of matrices into smaller pieces that allow efficient inversion of models. An example is the *singular value decomposition* (SVD). The SVD manifests a universal theme in computational imaging and finds applications such as orthogonalization, data compression and matrix inversion. A thorough understanding of this matrix decomposition will pay off in the study of various topics discussed later in the book.

In data analysis we often encounter situations where we have exact or approximate models. However, in the presence of noise even exact models are not expected to perfectly describe our data. Often the models contain a few unknown parameters, which can be extracted by minimizing a particular *cost function*. This allows to estimate and extract unknown parameters, that were not immediately accessible in the original data. A natural setting for this approach is the method of *least squares*. Many variations of this approach exists, ranging from linear to nonlinear and univariate to multivariate. In this chapter we will discuss the simplest case, linear least squares, which is extended in subsequent chapters.

2.1 Vectors

In the mathematical sense a vector $\boldsymbol{u}$ is an ordered and finite list of numbers. We may order such numbers as a column vector, for instance

$$\boldsymbol{u} = \begin{bmatrix} 1 \\ -1 \\ 2 \end{bmatrix}. \tag{2.1}$$

The same information is contained in the associated row vector or transpose,

$$\boldsymbol{u}^T = \begin{bmatrix} 1 & -1 & 2 \end{bmatrix}. \tag{2.2}$$

For complex quantities we typically encounter the complex transpose, for instance

$$\boldsymbol{u} = \begin{bmatrix} 1+i \\ -i \\ 2i \end{bmatrix}, \boldsymbol{u}^\dagger = \begin{bmatrix} (1-i) & i & -2i \end{bmatrix}. \tag{2.3}$$

In the physical sense, the above description of a vector as an ordered list of numbers is insufficient. The choice of a coordinate system is arbitrary, while physical vectors have properties independent of the particular coordinate system chosen to represent them. In particular, vectors are quantities that preserve geometric properties such as length and relative angle as we transform between coordinate systems.

A special vector is the unit vector $\boldsymbol{e}_k$, which is of unit length and points in the direction of the kth coordinate for a given predefined coordinate system. For instance, in a three-dimensional Cartesian coordinate system, we have

$$\boldsymbol{e}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \boldsymbol{e}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \boldsymbol{e}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}. \tag{2.4}$$

Sparsity

A vector $\boldsymbol{u}$ is called sparse provided that it only contains *few* non-zero elements. There is no general definition when a vector has few enough non-zero elements such that it is considered sparse.

```
% matlab code to check number of non-zero elements in a vector

a = [1, 2, 3, 0, 1];
disp( nnz(a) )
```

2.1.1 Vector addition

$$\boldsymbol{u} + \boldsymbol{v} = \sum_k (u_k + v_k)\, \boldsymbol{e}_k \tag{2.5}$$

Example

$$\boldsymbol{u} = \begin{bmatrix} 1 \\ -1 \\ 2 \end{bmatrix}, \boldsymbol{v} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \boldsymbol{u} + \boldsymbol{v} = \begin{bmatrix} 2 \\ 1 \\ 5 \end{bmatrix} \tag{2.6}$$

2.1.2 Scalar multiplication

A vector $\boldsymbol{u}$ can be synthesized by multiplying scalar coefficients u_k with the unit vectors $\boldsymbol{e}_k$ and a global scalar α is multiplied with each coefficient:

$$\alpha \boldsymbol{u} = \sum_k \alpha u_k \boldsymbol{e}_k \tag{2.7}$$

Example

$$5 \begin{bmatrix} 1 \\ -2 \end{bmatrix} = \begin{bmatrix} 5 \\ -10 \end{bmatrix} \tag{2.8}$$

2.1.3 Inner product

The *inner product*, also called *scalar product*, is obtained for real-valued vectors $\boldsymbol{u}$ and v as

$$\boldsymbol{u}^T \boldsymbol{v} = \sum_k u_k v_k. \tag{2.9}$$

and complex-valued vectors as

$$\boldsymbol{u}^\dagger \boldsymbol{v} = \sum_k u_k^* v_k, \tag{2.10}$$

where * denotes complex conjugation and † the conjugate transpose as described above.

Example

$$\begin{bmatrix} 1 & -1 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = 1 \cdot 1 - 1 \cdot 2 + 2 \cdot 3 = 5 \tag{2.11}$$

Computational complexity of inner products

Computing an inner product between two vectors with N components requires multiplication of each component (N operations) and addition of these ($N-1$ additions). Hence the number of operations required to compute an inner product of two vectors is $N + N - 1 = 2N - 1$ resulting in a computational complexity of $\mathcal{O}(N)$.

2.2 Vector Norms

A norm is a function that assigns a non-negative scalar to a vector or a matrix. It is typically used in inverse modelling to formulate cost functions, which are then optimized (minimized or maximized). Any norm $\|\ldots\|$ is defined through the following properties:

1. $\|\boldsymbol{u} + \boldsymbol{v}\| \leq \|\boldsymbol{u}\| + \|\boldsymbol{v}\|$ (triangle inequality)
2. $\|\boldsymbol{u}\| \geq 0$ (non-negativity)
3. $\|\boldsymbol{u}\| = 0$ if and only if $\boldsymbol{u} = 0$ (definiteness)
4. $\|\alpha \boldsymbol{u}\| = |\alpha| \, \|\boldsymbol{u}\|$ (homogeneity)

2.2.1 L2 norm

The *L2-norm* is derived from the inner product. For real-valued u

$$\|\boldsymbol{u}\|_2 = \left(\boldsymbol{u}^T\boldsymbol{u}\right)^{1/2} = \left(\sum_k u_k^2\right)^{1/2}, \tag{2.12}$$

for complex-valued u

$$\|\boldsymbol{u}\|_2 = \left(\boldsymbol{u}^\dagger\boldsymbol{u}\right)^{1/2} = \left(\sum_k |u_k|^2\right)^{1/2}. \tag{2.13}$$

Example

Suppose

$$\boldsymbol{u} = \begin{bmatrix} 3 \\ 4i \end{bmatrix}, \tag{2.14}$$

then

$$\|\boldsymbol{u}\|_2^2 = \boldsymbol{u}^\dagger\boldsymbol{u} = \begin{bmatrix} 3 & -4i \end{bmatrix}\begin{bmatrix} 3 \\ 4i \end{bmatrix} = 9 + 16 = 25 \tag{2.15}$$

and

$$\|\boldsymbol{u}\|_2 = 5. \tag{2.16}$$

Cost function

The L2-norm will be important in the context of formulating *cost functions*[1] in least square problems. It is typically a function of the difference of a forward model $\boldsymbol{f}(\boldsymbol{\theta})$, depending on a set of parameters $\boldsymbol{\theta}^T = [\theta_1, \theta_2, \ldots, \theta_N]$ to be estimated, and observed data $\boldsymbol{d}$

$$\mathcal{L} = \|\boldsymbol{f}(\boldsymbol{\theta}) - \boldsymbol{d}\|_2. \tag{2.17}$$

The scalar quantity $\mathcal{L}$ is called a *cost function*. Minimizing $\mathcal{L}$ with respect to the parameters $\boldsymbol{\theta}$ gives rise to a model that explains the measurement $\boldsymbol{d}$ in an optimal sense. Equivalently to Eq. 2.17, we may minimize the following cost function,

$$\mathcal{L}' = \|\boldsymbol{f}(\boldsymbol{\theta}) - \boldsymbol{d}\|_2^2. \tag{2.18}$$

This prevents evaluating the square root along with the need for differentiating under the square root. In machine learning, the extracted parameters $\boldsymbol{\theta}$ are then used for *generalization*. This means that the parameters $\boldsymbol{\theta}$ obtained from observed data (in the past) are used to make further prediction about future measurement outcomes. One may expect a set of parameters $\boldsymbol{\theta}$ that minimizes the cost function $\mathcal{L}$ to be somewhat optimal for future predictions. However, this is generally not necessarily the case. Complex models bear the risk of *overfitting*, where the parameters found describe the past data but generalize poorly to future data.

[1] Also known as *figure of merit* or *objective function*.

2.2.2 L1 norm

The L1-norm,

$$\|\boldsymbol{u}\|_1 = \sum_k |u_k|\,, \tag{2.19}$$

plays a central role in the field of *compressed sensing* where it can be of great use to promote *sparsity* in the solution of inverse problems. If a problem

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{2.20}$$

is given, where $\boldsymbol{x}$ represent parameters in our model, $\boldsymbol{A}$ is a matrix representing a linear system, and $\boldsymbol{b}$ is an observation, then we may seek to minimize a so-called *data term*

$$e_{\text{data}} = \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2\,.$$

The problem is underdetermined if multiple solutions for $\boldsymbol{x}$ exist. In some problems we may vary $\boldsymbol{x}$ on a subspace (for instance a line, a plane, or higher-dimensional analogs) without changing the value of data term e_{data}. Adding a L1 *regularization term*,

$$e = e_{\text{data}} + e_{\text{reg}} \tag{2.21}$$

$$= \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \lambda \|\boldsymbol{x}\|_1\,, \tag{2.22}$$

with λ a weight parameter, changes the situation and prefers a solution with few non-zero entries in $\boldsymbol{x}$. The following example gives a geometric reasoning for the sparsity promoting property of the L1 norm, which was first described by Tibshirani [44].

Comparison of L1 and L2

Suppose we are told that a two-component, real-valued vector $\boldsymbol{r}^T = (x, y)$ has an L2 norm equal to one, $\|\boldsymbol{r}\|_2 = 1$. This fixes the possible vectors to the unit circle

$$\|\boldsymbol{r}\|_2^2 = x^2 + y^2 = 1. \tag{2.23}$$

This equation describes the set of vectors with unit radius. Another way to see this is solve for one of the components, for instance y,

$$y = \pm\sqrt{1 - x^2}. \tag{2.24}$$

We know that $-1 \leq x \leq 1$ since the length of $\boldsymbol{r}$ is bounded by one. Thus Eq. 2.24 describes two semi-circles of radius 1, above and below the x axis.

Now suppose we had been told a similar statement for the L1 norm, $\|\boldsymbol{r}\|_1 = 1$. What geometric region is described by this equation? By definition (compare Eq. 2.19)

$$\|\boldsymbol{r}\|_1 = |x| + |y| = 1. \tag{2.25}$$

We can solve this equation for y

$$|y| = 1 - |x|\,. \tag{2.26}$$

We have to distinguish four cases:

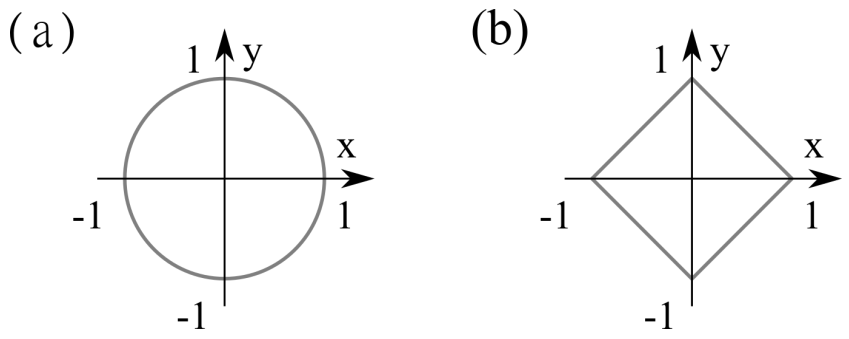


Figure 2.1: (a) L2 versus (b) L1 norm with unit length.

1. $0 \leq x \leq 1$ and $0 \leq y \leq 1$. Then $y = 1 - x$.
2. $0 \leq x \leq 1$ and $-1 \leq y \leq 0$. Then $-y = 1 - x$, or $y = x - 1$.
3. $-1 \leq x \leq 0$ and $0 \leq y \leq 1$. Then $y = 1 + x$.
4. $-1 \leq x \leq 0$ and $-1 \leq y \leq 0$. Then $-y = 1 + x$, or $y = -x - 1$.

In summary,

$$y = \begin{cases} 1 - x & , 0 < x < 1 \\ x - 1 & , 0 < x < 1 \\ 1 + x & , -1 < x < 0 \\ -x - 1 & , -1 < x < 0 \end{cases}. \tag{2.27}$$

We see that the set of points that satisfy $\|\boldsymbol{r}\|_1 = 1$ is given by four linear functions. Figure 2.1 illustrates what we have just found. As we will see, carefully choosing between L1- and L2-based cost functions can enforce useful properties upon the solution of a problem at hand, for example sparsity. This is because in problems with multiple constraints, each geometrically represented by a set, the square in Fig. 2.1 (b) will intersect another set with *large probability* at one of its vertices. But at the vertices in Fig. 2.1 (b) the solution vector is inherently sparse: either x or y is nonzero. This is in contrast to the circle in Fig. 2.1 (a) which is equally likely to intersect another set at any point on its circumference. In this case we expect the solution *probably not* to be sparse.

Although at this point we have not yet reviewed matrices, we allow ourselves to illustrate the point of sparsity by giving the particular example :

$$\mathcal{L} = \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \lambda \|\boldsymbol{x}\|_1 , \tag{2.28}$$

with e referring the cost to minimize. The L2 norm represents the *data term* and the L1 norm representing the *regularization term* weighted by a scalar λ which we assume to be equal to one for now. Let us choose some particular numbers:

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 \\ 0 & 0 \end{bmatrix} \tag{2.29}$$

and

$$\boldsymbol{b} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}, \tag{2.30}$$

We are searching for a vector $\begin{bmatrix} x & y \end{bmatrix}^T$ minimizing e. For

$$\boldsymbol{x_1} = \begin{bmatrix} 2 \\ 0 \end{bmatrix} \tag{2.31}$$

and

$$\boldsymbol{x_2} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2.32}$$

the data term $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2$ vanishes and is thus minimal (a norm is non-negative). It is a simple matter to show that all points in between the aforementioned solutions, namely the line (α a constant which can be chosen freely)

$$\begin{bmatrix} x \\ y \end{bmatrix} = (1-\alpha)\begin{bmatrix} 2 \\ 0 \end{bmatrix} + \alpha \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2.33}$$

$$= \begin{bmatrix} 2 \\ 0 \end{bmatrix} + \alpha \begin{bmatrix} -2 \\ 1 \end{bmatrix}, \tag{2.34}$$

likewise minimize the data term. This line is depicted by the blue line in Fig. 2.2. However, we have to minimize the whole expression in Eq. 2.28. If we chose a particular α, we can draw, depending on the norm a circle or a diamond that represents our choice in the following sense: All points on the gray curve have identical regularization terms, yet only the intersection points minimize the data term (in our case to zero). The contribution of the regularization term is represented by the axis intersection of the circle or diamond respectively. To minimize the whole system, we need to find a suitable compromise between the data term and the regularization term. In the limit of λ approaching zero, the data term always dominates and the solution will therefore be very close to the blue line. The smallest figure that still intersects the blue line will be the solution we are looking for, as seen by the two situations drawn in Fig. 2.2. It is seen that the solution (orange dot) to the problem

$$e_{\text{L2-reg}} = \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \lambda \|\boldsymbol{x}\|_2^2, \tag{2.35}$$

which is depicted in Fig. 2.2 (a), has only non-zero entries. In contrast, the solution to

$$e_{\text{L1-reg}} = \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \lambda \|\boldsymbol{x}\|_1, \tag{2.36}$$

has only one non-zero entry. For higher values λ, you can imagine a parabolic potential orthogonal to the blue line representing the data term. Also in these cases, the compromise that is found will continue to be sparse, just with a different position on the y-axis for the L1-normed situation. Thus the L1-regularization term yields a more sparse solution than the L2-regularization term. In the next chapter on optimization, we will discuss how to systematically optimize large-scale problems with sparsity regularization.

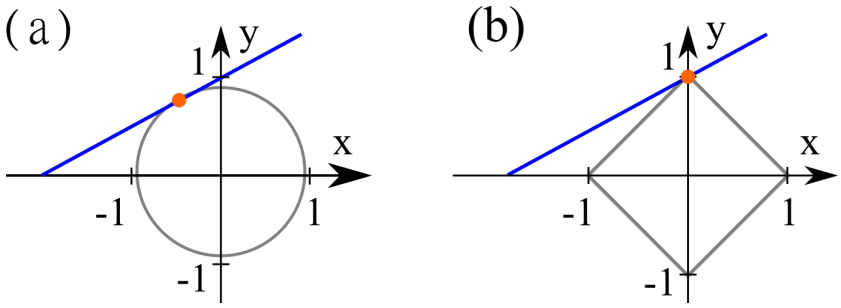


Figure 2.2: Solution space to a linear system (blue) subject to (a) L2 and (b) L1 regularization, respectively.

2.3 A first look at least squares: linear regression

In many physical applications we have access to observational data which is related but not equal to a quantity of interest. Moreover, the data is typically subject to noise. We may therefore not retrieve the quantity of interest directly, but can only estimate it. Depending on the noise level, the estimated quantity of interest may be more or less accurately estimated. We illustrate this situation by a few examples.

2.3.1 Linear least squares

Suppose we know that the input x and output y of a physical system are related by a linear relationship

$$y = ax, \tag{2.37}$$

where we can measure y for a given x, assumed to be perfectly known, but a is unknown. Our goal here is to estimate a. We will potentially carry out multiple measurements with input x_k and output y_k to improve statistics on the precise value of a.

The relationship in Eq. 2.37 is idealized, because measurements are subject to noise. We therefore distinguish the ideal model output y_k from the noisy measurements d_k. We assume that the noise present in d_k is zero as we average over many measurements. That is, our idealized model in Eq. 2.37 is not true for every data point, but it is true in an average sense. Assuming Gaussian statistics[2], this implies that we can estimate a by minimizing the squared difference between model and data over all data points,

$$\mathcal{L} = \sum_k (y_k - d_k)^2 = \sum_k (ax_k - d_k)^2 . \tag{2.38}$$

While the deviation between ax_k and d_k is typically not zero for all measurements k (again due to noise), we expect to do the right thing on average by minimizing $\mathcal{L}$. The squares prevent positive and negative *residuals* $ax_k - d_k$ to cancel. We can rewrite Eq. 2.38 in vector form using the L2 norm

$$\mathcal{L} = \|a\boldsymbol{x} - \boldsymbol{d}\|_2^2 , \tag{2.39}$$

[2]Later we will go a little deeper into how the choice of a particular cost function relates to a probabilistic model of the noise present in a measurement.

or

$$\mathcal{L} = (a\boldsymbol{x} - \boldsymbol{d})^T (a\boldsymbol{x} - \boldsymbol{d}) = a^2 \boldsymbol{x}^T \boldsymbol{x} - 2a\boldsymbol{x}^T \boldsymbol{d} + \boldsymbol{d}^T \boldsymbol{d}, \tag{2.40}$$

where we have used $\boldsymbol{x}^T \boldsymbol{d} = \boldsymbol{d}^T \boldsymbol{x}$. This is a quadratic function in a pointing upwards ($a^2 > 0$), so there is exactly one minimum. We find this minimum by differentiating with respect to a and setting the result to zero,

$$\frac{\partial \mathcal{L}}{\partial a} = 2a\boldsymbol{x}^T \boldsymbol{x} - 2\boldsymbol{x}^T \boldsymbol{d} = 0. \tag{2.41}$$

The solution is

$$a = \frac{\boldsymbol{x}^T \boldsymbol{d}}{\boldsymbol{x}^T \boldsymbol{x}}. \tag{2.42}$$

Note that both parts of the quotient, $\boldsymbol{x}^T \boldsymbol{d}$ and $\boldsymbol{x}^T \boldsymbol{x}$ are just scalar numbers.

```
% matlab code: linear regression

rng(666,'twister')
x = (-5:5)';
N = length(x);
d = 2*x + 4*(2*rand(N,1)-1);
% generate outlier
% d(10) = 20;
% lsq
aEstimate = x'*d / (x'*x);

plot(x, d, 'ko', 'MarkerFaceColor', 'k')
hold on
plot(x,aEstimate*x,'k--','LineWidth',2)
hold off, axis square, grid on, xlabel('x')
legend('data', 'fit','location','Northwest')
export_fig('linearRegression1.png')
```

Listing 2.1: Linear Regression (Matlab)

The above codes start by seeding the random number generator. This step makes sure the output produced in this example is the same even if computed on different machines. Then an ideal linear function $2\boldsymbol{x}$ plus random noise is generated to produce the data vector $\boldsymbol{d}$. The least square estimate for a is calculated according to Eq. 2.42. The fit produced by the linear regression code is shown in Fig. 2.3.

Intuitively, we get a feeling for least squares by imagining all data points to be connected with the fit line by elastic rubber bands. The rubber bands are pulling on the fit line to get it close to each respective data point. The result is a line suspended midway - a compromise between all data points.

2.3.2 Regression with outliers

In the previous subsection we assumed that the noise was zero mean. This, together with the assumption of a Gaussian noise distribution, justified the use of the L2 cost

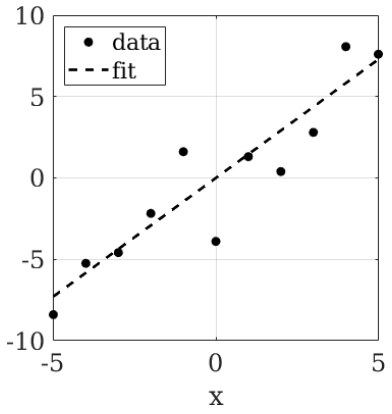


Figure 2.3: Linear regression. The data (black dots) is fit by a linear function according to Eq. 2.42.

function: Summing over enough data points, the noise approaches its mean, which is zero. However, in many practical situations data is subject to systematic errors and outliers[3]. A disadvantage of the L2 cost function is its sensitivity to outliers, which can distort L2 regression considerably. In some cases using the L1 norm can produce curve fits that are more robust due to smaller weights given to outliers. The L1 cost for the linear regression problem from the previous section is given by

$$\mathcal{L} = \|a\boldsymbol{x} - \boldsymbol{d}\|_1 = \sum_k |ax_k - d_k| \,. \tag{2.43}$$

How do we minimize this expression? One problem here is that the absolute value function is not differentiable at zero input. This forbids us to use standard least squares. Nevertheless, there are a plethora of algorithmic techniques to solve this problem, such as the simplex method, proximal gradients, and weighted least squares. Here we illustrate the latter route. Notice that we can rewrite the L1 cost function as

$$\mathcal{L} = \sum_k |ax_k - d_k| = \sum_k \frac{(ax_k - d_k)^2}{|ax_k - d_k|} = \sum_k w_k \left(ax_k - d_k\right)^2 , \tag{2.44}$$

which is a weighted least squares problem with

$$w_k = \frac{1}{|ax_k - d_k|}. \tag{2.45}$$

Intuitively, this cost function attributes less weight to strong outliers, where $|ax_k - d_k|$ is large and therefore w_k is small. Now comes a little trick: while both w_k and $(ax_k - d_k)^2$ depend on a, we pretend w_k is constant with respect to a change in a. Then

$$\frac{\partial \mathcal{L}}{\partial a} = 2 \sum_k w_k \left(ax_k - d_k\right) x_k. \tag{2.46}$$

[3]For instance, human failure in noting down a measurement properly or a broken pixel on a camera.

Setting this derivative to zero and solving for a yields

$$a_{n+1} = \frac{\sum_k w_k x_k d_k}{\sum_k w_k x_k^2} = \frac{\boldsymbol{w}_n^T (\boldsymbol{x} \odot \boldsymbol{d})}{\boldsymbol{w}_n^T (\boldsymbol{x} \odot \boldsymbol{x})}, \tag{2.47}$$

where $\odot$ is an element-wise multiplication[4] and the kth component of $\boldsymbol{w}_n$ is given by

$$(\boldsymbol{w}_n)_k = \frac{1}{|a_n x_k - d_k|}. \tag{2.48}$$

This approximation can now be used as an iterative update by recalculating the weights. In the code below we iteratively apply Eqs. 2.47 and 2.48 to obtain an a for which the L1 cost function in Eq. 2.43 is small. The L1-based approach here is also known as *least absolute deviation (LAD) regression*.

```
% matlab code: lad regression

rng(666, 'twister')
% generate input
x = (-5:5)';
N = length(x); % number of elements in x
% generate data
d = 2*x + 4*(2*rand(N,1)-1);
% generate outlier
d(10) = 50;
% d(9:end) = 25;
% lsq: least squares
aEstimate = x'*d / (x'*x);

% lad
numIter = 10;
aLAD = aEstimate; % initialize LAD-estimate
for k = 1:numIter
   w = 1 ./ ( abs(aLAD*x - d) + eps );
   aLAD = w' * (d.*x) / (w' * (x.^2));
end

plot(x, d, 'ko', 'MarkerFaceColor', 'k')
hold on
plot(x,aEstimate*x,'k--','LineWidth',2)
plot(x,aLAD*x,'k-','LineWidth',2)
hold off, axis square, grid on, xlabel('x')
legend('data', 'lsq fit','lad fit','location','Northwest')
```

Listing 2.2: LAD Regression (Matlab)

[4] $\boldsymbol{x}^T \odot \boldsymbol{d}^T = (x_1, x_2, \ldots, x_N) \odot (d_1, d_2, \ldots, d_N) = (x_1 d_1, x_2 d_2, \ldots, x_N d_N)$.

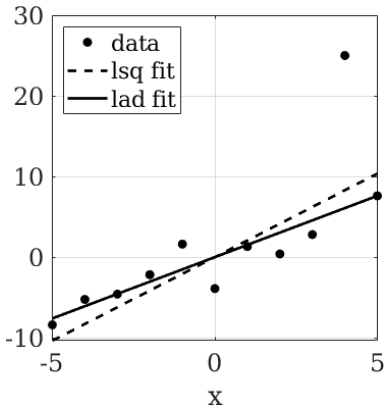


Figure 2.4: Linear regression. The data (black dots) is fit by linear function according to Eq. 2.42 (LSQ, dashed line) and Eq. 2.47 (LAD, solid line). In this example, the LAD estimate is more robust to the outlier.

In Fig.2.4 we compare the fit results for least squares (LSQ) and LAD regression. We see that the fit obtained through LAD regression is less affected by the outlier on the top right. Why does it work? Intuitively, we can again invoke the rubber band picture. However, this time the rubber band exerts a force that becomes weaker over large distances (here large residuals), as mathematically implied by the weighting factors w_k in Eq. 2.44. Because the outlier is very far away the rest of the points, which closely cluster around a line in Fig. 2.4, the solution is pushed towards.

2.4 Orthogonal Basis

2.4.1 Linear independence

A set of vectors $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_N$ is called *linearly independent* if

$$\sum_k \alpha_k \boldsymbol{u}_k = 0 \tag{2.49}$$

is only satisfied if and only if

$$\alpha_1 = \alpha_2 = \ldots = \alpha_N = 0. \tag{2.50}$$

In this case $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_N$ forms a *basis* of *dimension* N. Conversely the set $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_N$ is called *linearly dependent* if there is at least one non-zero coefficient α_k such that Eq. 2.49 holds true.

2.4.2 Orthogonality

A set of vectors $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_N$ is called *orthogonal* if

$$\boldsymbol{u}_j^\dagger \boldsymbol{u}_k = \begin{cases} \|\boldsymbol{u}_j\|_2^2 & , j = k \\ 0 & , j \neq k \end{cases} \tag{2.51}$$

and *orthonormal* if

$$\boldsymbol{u}_j^\dagger \boldsymbol{u}_k = \delta_{j,k}, \tag{2.52}$$

where

$$\delta_{j,k} = \begin{cases} 1 & , j = k \\ 0 & , j \neq k \end{cases} \tag{2.53}$$

is called *Kronecker delta*.

Example

The vectors[5]

$$\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ i \\ i^2 \\ i^3 \end{bmatrix}, \begin{bmatrix} 1 \\ i^2 \\ i^4 \\ i^6 \end{bmatrix}, \begin{bmatrix} 1 \\ i^3 \\ i^6 \\ i^9 \end{bmatrix}, \tag{2.54}$$

with the imaginary constant $i = \sqrt{-1}$, are orthogonal. For instance, the inner product between the second and third vector gives

$$\begin{bmatrix} 1 & -i & -1 & i \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} = 1 + i - 1 - i = 0, \tag{2.55}$$

where we used $i^* = -i$ and $i^2 = -1$. A similar calculation for all pairs of all the vectors listed shows orthogonality.

[5] These vectors form the columns of the Fourier matrix discussed in subsection 2.11.2.

2.4.3 Zernike polynomials

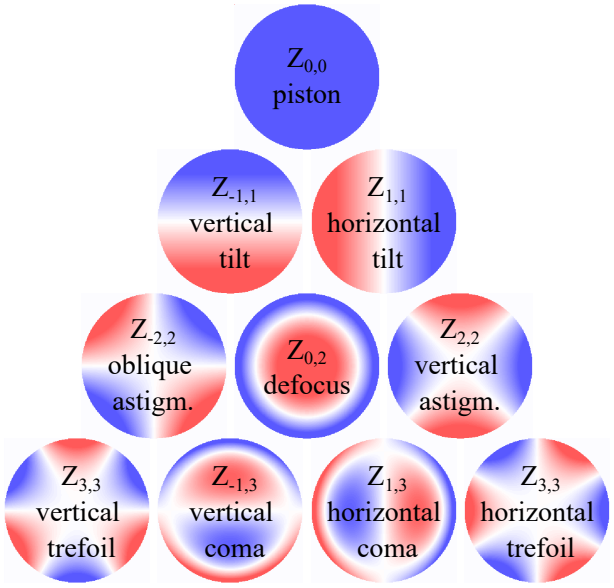


Figure 2.5: Zernike polynomials.

Zernike polynomials are a basis for real-valued functions defined on the unit circle. In Optics, they are typically used to desribe the phase modifications (aberrations) of optical elements with finite diameter. Figure 2.5 shows the shape of various Zernike basis vectors[6] along with the name of the aberration given to each one. Positive values are colored blue and negative values are colored red. Note that by combining a vertical and a corresponding horizontal basis term one can freely rotate the direction of the aberration. Also note that the standard Zernike functions, as defined below, describe only phase variations. They do not contain information about brightness variations in the pupil. We define the Zernike polynomials as

$$Z_n^m(r,\theta) = \begin{cases} \sqrt{\frac{n+1}{\pi}} R_n^m(r) & , m = 0 \\ \sqrt{\frac{2n+2}{\pi}} R_n^m(r) \cos(m\theta) & , m > 0 \\ \sqrt{\frac{2n+2}{\pi}} R_n^m(r) \sin(m\theta) & , m < 0 \end{cases}. \tag{2.56}$$

Here $0 \leq r \leq 1$ and $0 \leq \theta \leq 2\pi$ are polar coordinates. The radial function $R_n^m(r)$ is given by

$$R_n^m(r) = \begin{cases} \sum_{s=0}^{\frac{n-m}{2}} \frac{(-1)^s (n-s)!}{s!\left(\frac{n+m}{2}-s\right)!\left(\frac{n-m}{2}-s\right)} r^{n-2s} & , \text{for } \frac{n-m}{2} \text{ even} \\ 0 & , \text{else} \end{cases}. \tag{2.57}$$

[6] Think of two-dimensional functions (or images) as vectors by reshaping or flattening them into a single column.

We note that there are different conventions for Zernike polynomials. The above convention is such that the Zernike polynomials are orthonormal[7]. The indexing convention using m, n follows Bathia and Wolf [4]. Another representation is the OSA and ANSI standard convention, which uses only a single index obtained from n and m,

$$j = \frac{n\,(n+2)+m}{2}. \tag{2.58}$$

The Matlab code `zernike.m` computes Zernike polynomials and the corresponding OSA/ANSI output index j for input indices m, n. Notice that the two local functions `rfun` and `getOSAindex` have a scope only valid inside the script `zernike.m`.

```
function [Z, j] = zernike(r,theta,m,n)
% compute zernike polynomial with indices m, n
% m: azimuthal
% n: radial
% references:
% https://en.wikipedia.org/wiki/Zernike_polynomials
% difference in notation:
% notice the normalization by 1/sqrt(pi), which causes the set
   of all
% zernike polynomials as defined here to be orthonormal
if m == 0      % m zero
    Z = sqrt((n+1)/pi)*rfun(m,n,r);
else
    if m > 0 % m positive
        Z = sqrt(2*(n+1)/pi)*rfun(m,n,r) .* cos(m*theta);
    else     % m negative
        Z = sqrt(2*(n+1)/pi)*rfun(m,n,r) .* sin(m*theta);
    end
end
Z = Z .* (r<=1);
j = getOSAindex(m,n);
end

function R = rfun(m, n, r)
% radial factor of Zernike  polynomial
R = 0;
if mod(m-n,2) == 0
    for s = 0 : ((n-m)/2)
        R = R + r.^(n-2*s) * ...
            ( (-1)^s * gamma(n-s+1) ) / ...
            ( gamma(s+1) * gamma((n+m)/2-s+1) * ...
              gamma((n-m)/2-s+1) );
    end
```

[7] This is analytically true, although numerical implementations might not exhibit this orthogonality due to coarse sampling and/or finite precision.

```
else
    R = 0;
end

end

function j = getOSAindex(m,n)
% OSA index convention
% https://en.wikipedia.org/wiki/Zernike_polynomials
j = (n*(n+2) + m)/2;
end
```

Listing 2.3: zernike.m (Matlab)

The code `showZernike.m` calls the function `zernike.m` and produces a plot similar to Fig. 2.5.

```
% matlab code: showZernike.m

N = 2^8;
x = linspace(-1,1,N);
dx = x(2) - x(1);
[X,Y] = meshgrid(x);
[THETA, R] = cart2pol(X,Y);

%% calculate Zernike for m,n
cmap = cat(1,...
       cat(2,linspace(1,1,50),linspace(1,0,50)),... %R
       cat(2,linspace(0,1,50),linspace(1,0,50)),... %G
       cat(2,linspace(0,1,50),linspace(1,1,50)))';  %B

Z = [];
for n = 0:5
    for m = -n:2:n
        [temp, J] = zernike(R,THETA,m,n);
        Z = cat(3,Z, temp);
        figure%(1)
        imagesc(x,x,Z(:,:,end))  % scaled image display
        xticks([-1 0 1]), yticks([-1 0 1])
        axis image off
        colormap(cmap)
        title(['n = ',num2str(n),...
               ', m = ',num2str(m), ', J = ',num2str(J)])
        pause(1)
    end
end
```

Listing 2.4: showZernike.m (Matlab)

Exercise: Translate the above codes to generate and plot Zernike polynomials into Python or Julia. Exercise

Exercise: Write Python or Julia code that checks whether or not the Zernike polynomials as defined above are orthonormal. Exercise

Exercise: The script `showZernike.m` and its call to the function `zernike.m` is computationally inefficient in that the same polynomials are generated many times. The loop in lines 30 to 34 are repeated at every function call of `zernike.m`. Write Python or Julia code that computes all Zernike polynomials up to index $j_{\max}$ without repeating the computations in lines 30 to 34 in `zernike.m`. Exercise

2.4.4 Basis expansions

The Zernike basis is *complete*, which means that it can approximate any well-behaved, real-valued function f over the unit circle at arbitrary precision as long as we choose a sufficiently large sufficient $j_{\max}$. For a finite number of terms we can write

$$f(r,\theta) \approx \sum_{j=0}^{j_{\max}} a_j Z_j(r,\theta). \tag{2.59}$$

In a discrete representation, each Zernike basis function may take the shape of a square matrix $\boldsymbol{Z}_j$, which we can reshape into a column vector $\boldsymbol{z}_j$. These representations are equivalent. Thus we may write

$$\boldsymbol{f} = \sum_{j=0}^{j_{\max}} a_j \boldsymbol{z}_j. \tag{2.60}$$

Now multiply from the left hand side with $\boldsymbol{z}_k^T$ to get

$$\boldsymbol{z}_k^T \boldsymbol{f} = \sum_{j=0}^{j_{\max}} a_j \boldsymbol{z}_k^T \boldsymbol{z}_j = \sum_{j=0}^{j_{\max}} a_j \delta_{j,k} = a_k. \tag{2.61}$$

Substituting this into Eq. 2.60, we get

$$\boldsymbol{f} = \sum_{j=0}^{j_{\max}} a_j \boldsymbol{z}_j = \sum_{j=0}^{j_{\max}} \left(\boldsymbol{z}_j^T \boldsymbol{f}\right) \boldsymbol{z}_j. \tag{2.62}$$

The code `expandZernike.m` expands an arbitrary function into the Zernike basis. You can play with m and n to see how the approximation gets better with an increasing number of expansion coefficients. Figure 2.6 shows an approximation of a spiral-shaped function (panel a) for an increasing number of expansion terms in the Zernike basis (panels b to e).

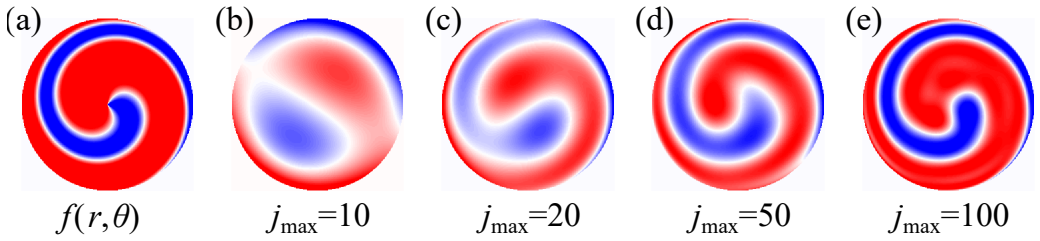


Figure 2.6: Zernike expansion. (a) Original function. (b-c) Expansion in Zernike basis with varying number of expansion terms. J denotes OSA/ANSI index.

```
N = 2^6;
x = linspace(-1,1,N);
dx = x(2) - x(1);
[X,Y] = meshgrid(x);
% convert Cartesian to polar coordinates
[THETA, R] = cart2pol(X,Y);

%% calculate Zernike for m,n
cmap = [[1 0 0];...
        [1 1 1];...
        [0 0 1]];
cmap = imresize(cmap, [100, 3],'bilinear');
% function to be expanded
f =  cos(THETA - 2*pi * (X.^2 + Y.^2)) .* (R<=1);
Z = [];
fExpansion = 0;
for n = 0:10
    for m = -n:2:n
        disp(m)
        disp(n)
        [temp, J] = zernike(R,THETA,m,n);
        c = sum(sum(temp .* f)) * dx^2;
        fExpansion = fExpansion + c * temp;

        figure(1)
        subplot(1,3,1)
        imagesc(x,x,temp)
        xticks([-1 0 1]), yticks([-1 0 1])
        axis image off
        colormap(cmap)
        title({['n = ',num2str(n)], ['m = ',num2str(m)],...
               ['J = ',num2str(J)]})

        subplot(1,3,2)
```

```
35          imagesc(f)
36          axis image off
37          title('f')
38
39          subplot(1,3,3)
40          imagesc(fExpansion)
41          axis image off
42          title('approx.')
43      end
44  end
```

Listing 2.5: expandZernike.m (Matlab)

Exercise: Translate the above code into Python. Exercise

Exercise: Write a Python or Julia code that expands a gray-value image of your choice in the Zernike basis. Restrict the non-zero region of the image to a circular domain. Exercise

---

2.4.5 Gram-Schmidt

In the previous section we have seen the usefulness of orthogonality. It allows us to quickly compute expansion coefficients in a given orthogonal basis. What if we have a linearly independent basis $\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_N$, but no orthogonality? In this case we can convert the given basis $\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_N$ into an orthogonal basis $\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_N$ using the Gram-Schmidt algorithm (see Alg. 2.1).

Algorithm 2.1 Classical Gram-Schmidt (CGS)

Require: $\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_N$

1: **for** $j = 1, \ldots, N$ **do**
2: $\quad \tilde{\boldsymbol{q}}_j = \boldsymbol{b}_j$
3: $\quad$ **for** $k = 1 \ldots j-1$ **do**
4: $\quad\quad \tilde{\boldsymbol{q}}_j = \tilde{\boldsymbol{q}}_j - \left(\boldsymbol{q}_k^\dagger \boldsymbol{b}_j\right) \boldsymbol{q}_k$
5: $\quad$ **end for**
6: $\quad \boldsymbol{q}_j = \frac{\tilde{\boldsymbol{q}}_j}{\left\|\tilde{\boldsymbol{q}}_j\right\|_2}$
7: **end for**

Notice that the nested for-loop in line 3-5 will only be exectued for $j > 1$. We emphasize that this code *assumes* $\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_N$ to be linearly independent. Otherwise there will be a division by zero in line 3, namely when $\left\|\tilde{\boldsymbol{q}}_j\right\|_2 = 0$ for some j. If this is the case we need a slightly more complicated version of the algorithm, which allows for exchanging columns by means of permutations. There is another pitfall, which is numerical in nature: The representation of numbers in a computer is generally only an approximation due to the finite floating point accuracy. The classical Gram-Schmidt (CGS) algorithm

may accumulate large errors such that the results are far away from the corresponding analytical solution. The modified Gram-Schmidt routine mitigates error accumulation. A thorough analysis would lead us too far away here, so we simply state the result (see Alg. 2.2), hoping to alert the reader to the issue of floating point precision. More information may be found in text books on numerical linear algebra [17, 45]. Here it should suffice for the reader to assure herself that the results in Alg. 2.1 and Alg. 2.2 are equivalent. Exercise

Algorithm 2.2 Modified Gram-Schmidt (MGS)

Require: $\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_N$

for $j = 1, \ldots, N$ **do**
  $\boldsymbol{q}_j = \frac{\boldsymbol{b}_j}{\|\boldsymbol{b}_j\|_2}$
  for $k = j+1, \ldots, N$ **do**
    $\boldsymbol{q}_k = \boldsymbol{q}_k - \left(\boldsymbol{q}_j^\dagger \boldsymbol{q}_k\right) \boldsymbol{q}_j$
  end for
end for

The Matlab code GramSchmidt.m demonstrates the issue that may arise due to finite floating point arithmetic. To see this, switch between 32 and 64 bit representations[8]

```
% matlab code: Gram-Schmidt

n = 3;
% rng(666 , 'twister'), B = rand(n); % alternative B
B = hilb(n);
precision = 'double'; % also try 'single'
Qc = zeros(size(B), precision);

% classical Gram-Schmidt (CGS)
for j = 1:size(B,2)
    q = B(:,j);
    % note: the following loop will be executed only for j>1
    for k = 1:j-1
        q = q - (Qc(:,k)'*q)*Qc(:,k);
    end
    Qc(:,j) = q / sqrt(q'*q);
end
errorCGS = abs(abs(det(Qc))-1);
disp(['error CGS: ', num2str(errorCGS)])

% modified Gram-Schmidt (MGS)
Qm = B;
for j = 1:size(B,2)
    Qm(:,j) = Qm(:,j)/sqrt(Qm(:,j)'*Qm(:,j));
```

[8] To this end, set `precision = 'single'` or `'double'`.

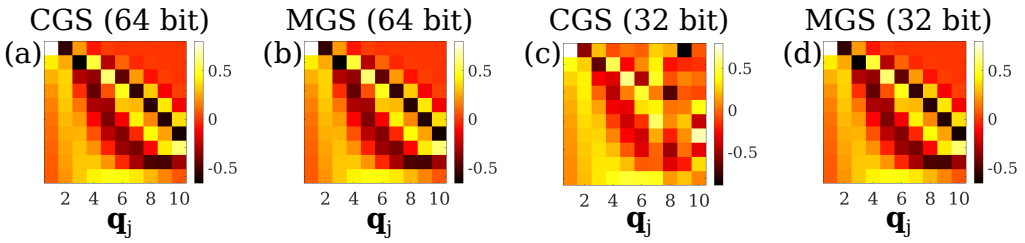


Figure 2.7: Q ($\boldsymbol{q}'s$ along columns) matrix produced by classical and modified Gram-Schmidt with varying numerical precision. (a) Classical Gram-Schmidt (CGS) with 64 bit, (b) Modified Gram-Schmidt (MGS) with 64 bit, (c) CGS with 32 bit, (d) MGS with 32 bit. Notice the floating point error produced by CGS at 32 bit in panel (c).

```
25      for k = (j+1):n
26          Qm(:,k) = Qm(:,k) - (Qm(:,j)'*Qm(:,k))*Qm(:,j);
27      end
28  end
29  errorMGS = abs(abs(det(Qm))-1);
30  disp(['error MGS: ', num2str(errorMGS)])
```

Listing 2.6: GramSchmidt.m (Matlab)

Exercise: Write a Python or Julia code that produces Zernike polynomials and subsequently orthogonalizes them via MGS. Compare the first 20 Zernikes (in OSA convention) before and after orthogonalization. Exercise

---

2.5 Matrices

2.5.1 Matrix Addition

Matrix addition

$$\boldsymbol{C} = \boldsymbol{A} + \boldsymbol{B} \tag{2.63}$$

is explained through element-wise addition

$$C_{i,j} = A_{i,j} + B_{i,j}. \tag{2.64}$$

Notice that matrix addition in the mathematical sense is only allowed if the two matrices have the same number of rows and columns. In contrast, many programming languages, including Matlab, Python and Julia use *broadcasting*, which automatically adjusts the dimensions in the case of missing or singleton-sized (size one) dimensions. It is important to be familiar with the broadcasting rules in the programming languages.

Example

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 1 & 3 & 5 \\ 7 & 9 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 5 & 8 \\ 11 & 14 & 6 \end{bmatrix} \tag{2.65}$$

Broadcasting matrix additions in Matlab [9]

```
% matlab code: broadcasting
A = eye(3,3);
B = [1;2;3];
C = A+B
D = A+B'
```

Listing 2.7: broadcasting.m (Matlab)

The above Matlab code evaluates to

$$\boldsymbol{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.66}$$

$$\boldsymbol{B} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \tag{2.67}$$

and

$$\boldsymbol{C} = \boldsymbol{A} + \boldsymbol{B} = \begin{bmatrix} 1+1 & 0+1 & 0+1 \\ 0+2 & 1+2 & 0+2 \\ 0+3 & 0+3 & 1+3 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 1 \\ 2 & 3 & 2 \\ 3 & 3 & 4 \end{bmatrix} \tag{2.68}$$

and

$$\boldsymbol{D} = \begin{bmatrix} 1+1 & 0+2 & 0+3 \\ 0+1 & 1+2 & 0+3 \\ 0+1 & 0+2 & 1+3 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 3 \\ 1 & 3 & 3 \\ 1 & 2 & 4 \end{bmatrix}. \tag{2.69}$$

To evaluate $\boldsymbol{C}$, Matlab has to add two matrices of incompatible dimension.

Notice that $\boldsymbol{B}$ and $\boldsymbol{C}$ are mathematically incompatible for matrix addition because they do not have the same number of rows and columns. Until Matlab 2016b, one way to broadcast variables was to invoke the `bsxfun` command. "bsx" stands for "binary singleton expansion". Here "binary" means that the operation is defined for two input arguments. The "singleton" dimension of an array has length 1. Thus, if two arrays of size 3×3 and 3×1 are added, `bsxfun` repeats the smaller array along the singleton dimension, in order to produce the same dimension of the larger array.

$$bsxfun(@plus, A, B) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \tag{2.70}$$

$$\rightarrow \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 1 \\ 2 & 3 & 2 \\ 3 & 3 & 4 \end{bmatrix}. \tag{2.71}$$

[9] More information: https://blogs.mathworks.com/loren/2016/10/24/matlab-arithmetic-expands-in-r2016b/

Compare this to

$$bsxfun(@plus, A, B') = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \tag{2.72}$$

$$\rightarrow \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 3 \\ 1 & 3 & 3 \\ 1 & 2 & 4 \end{bmatrix}. \tag{2.73}$$

Since Matlab 2016b the functionality of `bsxfun` has been transferred to elementary arithmetic operations. While this helped writing Matlab code that appears more concise, it sometimes bears the risk of overlooking bugs that previously would have caused an error. Getting rid of such bugs can be a painstaking business, so we stress once more that awareness of broadcasting rules is key.

2.5.2 Matrix-Vector Multiplication

Viewpoint 1

A matrix-vector multiplication can be understood as generating a linear combination of the columns of a given matrix, with weights determined by the components of the multiplying vector:

$$\boxed{\boldsymbol{A}\boldsymbol{x} = \sum_k x_k \boldsymbol{A}_k,} \tag{2.74}$$

where $\boldsymbol{A}_k$ are the columns of $\boldsymbol{A}$. For instance,

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = 1 \cdot \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + 2 \cdot \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} + 3 \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix}, \tag{2.75}$$

where the final result is not important here. For now, we focus on the operational details, namely, the fact how the components of $\boldsymbol{x}$ (colored) multiply with the columns of $\boldsymbol{A}$.

Why is this viewpoint important? Answer: We may regard matrix-vecor multiplication as a linear generator. The set of vectors we can generate is also called the *range*, *span*, or *column space* of $\boldsymbol{A}$. The number of linearly independent vectors in the column space of $\boldsymbol{A}$ is referred to as the *rank* of $\boldsymbol{A}$, written rank($\boldsymbol{A}$). The components of $\boldsymbol{x}$ are the weights in the linear combination. When we tackle small linear inverse problems of the form

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \tag{2.76}$$

we may quickly inspect whether or not $\boldsymbol{b}$ is in the column space of $\boldsymbol{A}$. For instance, does the problem

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 0 \end{bmatrix} \boldsymbol{x} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \tag{2.77}$$

have a solution for $\boldsymbol{x}$? The answer to this question should be evident.

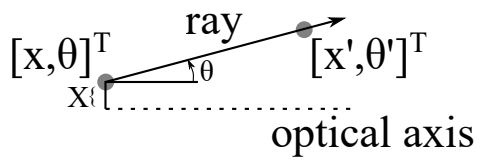


Figure 2.8: Free-space propagation of optical ray.

The role of unit vectors

Applying unit vector with only a single non-zero component as input to a matrix-vecor multiplication returns a particular column of the given matrix

$$\boldsymbol{A}\boldsymbol{e}_k = \boldsymbol{A}_k. \tag{2.78}$$

In optics, this viewpoint is important because it allows to interprete the impulse response to a linear system as a single column of the matrix that represents the linear system. See for instance section 5.9.

Viewpoint 2

Another viewpoint is to compute matrix-vector products $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ by means of inner products between the rows of $\boldsymbol{A}$ and the (single) column of $\boldsymbol{x}$

$$\boxed{b_i = \boldsymbol{A}_{i,:}\boldsymbol{x} = \sum_j A_{i,j}x_j.} \tag{2.79}$$

Computational complexity of matrix-vector products

A matrix-vector product of the for $\boldsymbol{A}\boldsymbol{x}$ with $\boldsymbol{A} \in \mathcal{R}^{M\times N}$ and $\boldsymbol{x} \in \mathcal{R}^N$ requires computation of M inner products, each given by Eq. 2.79. Each inner product requires $2N-1$ operations, so we have a total of $M\,(2N-1) \approx 2MN$ operations. The computational complexity is thus $\mathcal{O}(MN)$.

2.5.3 Parxial Ray Tracing Part 1

The theoretical development of linear algebra is paused here for a moment to appreciate the fact that we are slowly gaining powerful mathematical tools that help us solve real-life optical problems. Matrix-vector multiplication is a fundamental operational building block in paraxial ray tracing. An optical ray, drawn in a 2D plane, is characterized by a location x and angle θ with respect to the optical axis, the latter in units of radian. In the paraxial regime, θ can also be thought of as the slope of the ray. An optical system transforms an input ray $(x, \theta)^T$ into an output ray $(x', \theta')^T$. We may characterize optical systems by cascades of elementary matrices.

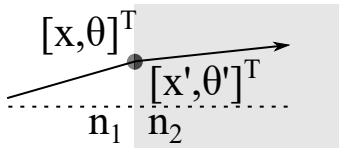


Figure 2.9: Refraction of an optical ray at an interface.

Paraxial Free-space Propagation

Free-space propagation over distance z can be described in terms of geometrical optics by means of matrix-vector multiplication. In order to find the sought-after free-space propagation matrix $\boldsymbol{F}$, we first try to understand how propagation acts on unit vectors with only a single non-zero component. The result is referred to here as *impulse response*. Once we know the corresponding impulse responses, we can stack up the matrix by virtue of Eq. 2.78.

How does a ray $(x, \theta)^T = (1, 0)^T$ propagate through free space? For $(1, 0)^T$ the direction $\theta = 0$ is parallel to the optical axis. The distance to the optical axis, $x = 1$, does not change. Hence,

$$\boldsymbol{F} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \boldsymbol{F}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}. \tag{2.80}$$

Next, consider the second impulse response, i.e. the output generated from the input $(x, \theta)^T = (0, 1)^T$. Under the *paraxial approximation* $\tan(\theta) \approx \sin(\theta) = \theta$. Then we have

$$\boldsymbol{F} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \boldsymbol{F}_2 = \begin{bmatrix} z \\ 1 \end{bmatrix}. \tag{2.81}$$

Putting these results together,

$$\boldsymbol{F} = \begin{bmatrix} \boldsymbol{F}_1 & \boldsymbol{F}_2 \end{bmatrix} = \begin{bmatrix} 1 & z \\ 0 & 1 \end{bmatrix}. \tag{2.82}$$

We should mention here that it is doubtful that the paraxial approximation performs well for angles as large as 1 rad, which corresponds to of about 57.3°. However, with decreasing angles the paraxial approximation becomes increasingly precise. Hence paraxial free-space propagation may be expressed as

$$\begin{bmatrix} 1 & z \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ \theta \end{bmatrix} = \begin{bmatrix} x' \\ \theta' \end{bmatrix}. \tag{2.83}$$

We see that $\operatorname{rank}(\boldsymbol{A}) = 2$. Does that mean that any ray is supported by free space? Answer: Yes, all combinations of $(x', \theta')^T$ can be generated by suitable combinations of $(x, \theta)^T$. Notice though that once the input $(x, \theta)^T$ is defined, the output angle θ' can not be different from the input angle θ for free-space propagation.

Paraxial Refraction

The transition from a first medium with refractive index n into a second medium with refractive index n' is described by Snell's law

$$n \sin(\theta) = n' \sin(\theta'). \tag{2.84}$$

Under the paraxial approximation, this becomes

$$n\theta \approx n'\theta'. \tag{2.85}$$

How does a ray $(x, \theta)^T = (1, 0)^T$ traverse through an interface? Answer: From Snell's law, for $\theta = 0$ it directly follows that $\theta' = 0$. Also, the beam is assumed to exit at the same point where it entered the medium[10]. Hence, representing refraction by a matrix, we have

$$\boldsymbol{R} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \boldsymbol{R}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}. \tag{2.86}$$

For angles $\theta \neq 0$ we can solve Snell's law directly for θ'. We also assume again that the location of the ray directly to the left and right of the interface is the same ($x = x'$). Under these assumptions, we get the second impulse response

$$\boldsymbol{R} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \boldsymbol{R}_2 = \begin{bmatrix} 0 \\ \frac{n}{n'} \end{bmatrix}. \tag{2.87}$$

With this we have the refraction matrix $\boldsymbol{R}$

$$\boldsymbol{R} = \begin{bmatrix} 1 & 0 \\ 0 & \frac{n}{n'} \end{bmatrix}. \tag{2.88}$$

Thus we can describe refraction of rays at an interface by

$$\begin{bmatrix} 1 & 0 \\ 0 & \frac{n}{n'} \end{bmatrix} \begin{bmatrix} x \\ \theta \end{bmatrix} = \begin{bmatrix} x' \\ \theta' \end{bmatrix}. \tag{2.89}$$

Paraxial and thin lens

We can infer the ray optical description of lenses by two simple properties: 1) rays parallel to the optical axis change direction in such a way that they intersect the focal point of the lens; 2) rays through the intersection of the optical axis and the lens do not change direction. Thus,

$$\boldsymbol{L} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \boldsymbol{L}_1 = \begin{bmatrix} 1 \\ -1/f \end{bmatrix} \tag{2.90}$$

and

$$\boldsymbol{L} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \boldsymbol{L}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{2.91}$$

[10] We note that this assumption is not exactly true under total internal reflection conditions, which is known as the Goos-Hähnchen effect.

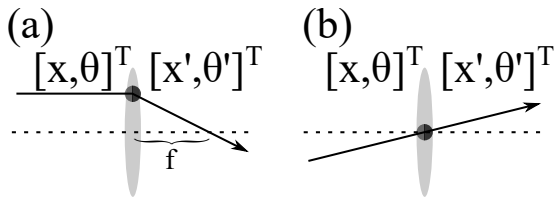


Figure 2.10: Transmission of optical rays through a thin lens. (a) Incident ray parallel to optical axis. (b) Ray through ntersection of optical axis and midpoint of lens.

This gives the lens matrix

$$\boldsymbol{L} = \begin{bmatrix} 1 & 0 \\ -1/f & 1 \end{bmatrix} \tag{2.92}$$

and the corresponding ray mapping through a lens

$$\begin{bmatrix} 1 & 0 \\ -1/f & 1 \end{bmatrix} \begin{bmatrix} x \\ \theta \end{bmatrix} = \begin{bmatrix} x' \\ \theta' \end{bmatrix}. \tag{2.93}$$

Example

How does a ray $(x, \theta)^T = (0, 1/f)^T$ propagate through the cascaded optical system $\boldsymbol{S} = \boldsymbol{FLF}$? This can be answered step-wise by observing that

$$\boldsymbol{FLF} \begin{bmatrix} x \\ \theta \end{bmatrix} \equiv \boldsymbol{FL}\boldsymbol{v}_1, \tag{2.94}$$

$$\boldsymbol{FL}\boldsymbol{v}_1 \equiv \boldsymbol{F}\boldsymbol{v}_2, \tag{2.95}$$

$$\boldsymbol{F}\boldsymbol{v}_2 \equiv \boldsymbol{v}_3, \tag{2.96}$$

where each colored variable on the right is defined by ($\equiv$) the corresponding colored quantity on the left. Thus we carry out the following sequence of computations for the input $(x, \theta)^T = (0, 1/f)^T$

$$\boldsymbol{v}_1 = \boldsymbol{F} \begin{bmatrix} 0 \\ 1/f \end{bmatrix} = \begin{bmatrix} 1 & z \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1/f \end{bmatrix} = \begin{bmatrix} z/f \\ 1/f \end{bmatrix}, \tag{2.97}$$

$$\boldsymbol{v}_2 = \boldsymbol{L}\boldsymbol{v}_1 = \begin{bmatrix} 1 & 0 \\ -1/f & 1 \end{bmatrix} \begin{bmatrix} z/f \\ 1/f \end{bmatrix} = \begin{bmatrix} z/f \\ -z/f^2 + 1/f \end{bmatrix}, \tag{2.98}$$

$$\boldsymbol{v}_3 = \boldsymbol{F}\boldsymbol{v}_2 = \begin{bmatrix} 1 & z \\ 0 & 1 \end{bmatrix} \begin{bmatrix} z/f \\ -z/f^2 + 1/f \end{bmatrix} = \begin{bmatrix} z/f - z^2/f^2 + z/f \\ -z/f^2 + 1/f \end{bmatrix}. \tag{2.99}$$

Let us check a special case to see if this result is reasonable. For $z = f$ we get

$$\boldsymbol{v}_3 = \boldsymbol{FLF} \begin{bmatrix} 0 \\ 1/f \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}. \tag{2.100}$$

Thus a ray traversing the focal point upstream of the lens propagates parallel to the optical axis downstream of the lens. Notice that this can already be seen from $\boldsymbol{v}_2$.

2.5.4 Matrix-Matrix Multiplication

Viewpoint 1

Suppose $\boldsymbol{A} \in \mathcal{R}^{M\times N}, \boldsymbol{B} \in \mathcal{R}^{N\times P}$, then

$$\boxed{\boldsymbol{A}\cdot\boldsymbol{B} = \boldsymbol{A}\cdot[\boldsymbol{B}_1,\ldots,\boldsymbol{B}_N] = [\boldsymbol{A}\cdot\boldsymbol{B}_1,\ldots,\boldsymbol{A}\cdot\boldsymbol{B}_N]\,.} \tag{2.101}$$

In words, one possibility to compute the product of two matrices is two compute the matrix-vector products between the first matrix $\boldsymbol{A}$ and each column of the second matrix $\boldsymbol{B}_k$. Another way to write this form of block multiplication is that the columns of $\boldsymbol{C} = \boldsymbol{A}\cdot\boldsymbol{B}$ are given by

$$\boldsymbol{C}_k = \boldsymbol{A}\cdot\boldsymbol{B}_k \tag{2.102}$$

for $k = 1,\ldots,P$.

Example: Matrix-Matrix Multiplication by columns

Suppose we wish to multiply not only a single but a set of K optical rays with the free-space ray propagation matrix $\boldsymbol{F}$. Each ray may potentially have a different distance from the optical axis x_k and a different direction θ_k. We can process all such rays in parallel by arranging this data into a matrix

$$\boldsymbol{B} = \begin{bmatrix} x_1 & x_2 & \ldots & x_{K-1} & x_K \\ \theta_1 & \theta_2 & \ldots & \theta_{K-1} & \theta_K \end{bmatrix}. \tag{2.103}$$

Then the transformed rays are given by

$$\begin{aligned} \boldsymbol{C} &= \boldsymbol{F}\cdot\boldsymbol{B} && (2.104)\\ &= \begin{bmatrix} \boldsymbol{F}\boldsymbol{B}_1 & \boldsymbol{F}\boldsymbol{B}_2 & \ldots & \boldsymbol{F}\boldsymbol{B}_{K-1} & \boldsymbol{F}\boldsymbol{B}_K \end{bmatrix} && (2.105)\\ &= \begin{bmatrix} x'_1 & x'_2 & \ldots & x'_{K-1} & x'_K \\ \theta'_1 & \theta'_2 & \ldots & \theta'_{K-1} & \theta'_K \end{bmatrix}. && (2.106) \end{aligned}$$

Computational complexity of matrix-matrix products

From Eq. 2.102 we have to compute P matrix-vector products, each requiring $M\,(2N-1)$ operations. Therefore, a matrix-matrix multiplication requires a total of $M\,(2N-1)\,P \approx 2MNP$ or $\mathcal{O}\,(MNP)$ operations. For square matrices with N elements per dimension, the computational complexity of matrix-matrix multiplication scales with $\mathcal{O}\left(N^3\right)$. Complexities of this type rapidly cause both performance and memory problems to ordinary computers such as notebooks. *Avoid large matrix multiplications whenever possible.* For instance, when cascaded expressions of the form $\boldsymbol{A}\cdot\boldsymbol{B}\cdot\boldsymbol{C}\cdot\boldsymbol{x}$ arise, we may compute

$$\boldsymbol{C}\cdot\boldsymbol{x} = \boldsymbol{z}_1,\ \boldsymbol{B}\cdot\boldsymbol{z}_1 = \boldsymbol{z}_2,\ \boldsymbol{A}\cdot\boldsymbol{z}_2 = \boldsymbol{z}_3 \tag{2.107}$$

to avoid storage and computation of the product $\boldsymbol{A}\cdot\boldsymbol{B}\cdot\boldsymbol{C}$ alltogether. Assuming square matrices with N elements per dimension, each of the computations of $\boldsymbol{z}_1$ to $\boldsymbol{z}_3$ requires three matrix-vector multiplications each with a complexity of $2N^2$. In contrast, computing $\boldsymbol{A}\cdot\boldsymbol{B}\cdot\boldsymbol{C}$ requires two matrix-matrix multiplications, each with a complexity $2N^3$.

Example: avoid matrix multiplications when possible

```
import numpy as np
import time

N = 2**12
A = np.random.rand(N,N)
B = np.random.rand(N,N)
C = np.random.rand(N,N)
x = np.random.rand(N,1)

t1 = time.time()

out1 = np.dot(C,B)
out2 = np.dot(out1,A)
out3 = np.dot(out2,x)

t2 = time.time()

out1 = np.dot(C,x)
out2 = np.dot(B,out1)
out3 = np.dot(A,out2)

t3 = time.time()

digits = 3
print('elapsed time method #1: '+\
      str(np.round((t2-t1)*10**digits)/10**digits)+' seconds')
print('elapsed time method #2: '+\
      str(np.round((t3-t2)*10**digits)/10**digits)+' seconds')
print('speedup by method #2: '+\
      str(np.round((t2-t1)/(t3-t2)*10)/10)+'x')
```

Listing 2.8: MatrixMultiplicationSpeed.py (Python)

This code returns on one of the authors' computer:

```
elapsed time method #1:  1.789 seconds
elapsed time method #2:  0.031 seconds
speedup by method #2:  57.3x
```

Viewpoint 2

The element at row i and column j of $\boldsymbol{C} = \boldsymbol{A} \cdot \boldsymbol{B}$ is an inner product,

$$\boxed{\boldsymbol{C}_{i,j} = \boldsymbol{A}_{i,:} \cdot \boldsymbol{B}_{:,j} = \sum_k A_{i,k} B_{k,j}.} \tag{2.108}$$

Here we use the colon operator (:) notation. $\boldsymbol{A}_{i,:}$ is the ith row of $\boldsymbol{A}$. Similarly, $\boldsymbol{B}_{:,j}$ is the jth column of $\boldsymbol{B}$.

Viewpoint 3

Another viewpoint is to regard matrix-matrix multiplication as a sum of outer products,

$$\boldsymbol{A}\cdot\boldsymbol{B}=\begin{bmatrix} | & & | \\ \boldsymbol{A}_{:,1} & \dots & \boldsymbol{A}_{:,N} \\ | & & | \end{bmatrix}\cdot\begin{bmatrix} - & \boldsymbol{B}_{1,:} & - \\ & \vdots & \\ - & \boldsymbol{B}_{P,:} & - \end{bmatrix}=\sum_k \boldsymbol{A}_{:,k}\boldsymbol{B}_{k,:} \tag{2.109}$$

Each of the summation terms $\boldsymbol{A}_{:,k}\boldsymbol{B}_{k,:}$ is a rank-one matrix (column by row). Decomposing matrices into rank-1 pieces is a dominant theme in numerical linear algebra, where for instance the spectral eigen decomposition of a symmetric matrix or the singular value decomposition build up higher-rank matrices from rank-1 building blocks. Data compression can be achieved by ignoring the less relevant rank-1 matrices in a matrix decomposition (discussed in subsection 2.8.2).

2.5.5 Ray Tracing Part 2

As an application of matrix-matrix multiplication, we show a paraxial ray tracing simulation that uses the viewpoint 1 on matrix-matrix multiplication and Eq. 2.101. In the code a set of rays is arranged along the columns of a matrix $\boldsymbol{V}$. The rays are propagated to a lens, giving rise to the set of rays $\boldsymbol{V}' = \boldsymbol{F}\cdot\boldsymbol{V}$. In a second step, the resulting matrix is transformed by a lens and once more propagated in free space, to wit $\boldsymbol{V}'' = \boldsymbol{F}\cdot\boldsymbol{L}\cdot\boldsymbol{V}' = \boldsymbol{F}\cdot\boldsymbol{L}\cdot(\boldsymbol{F}\cdot\boldsymbol{V})$. Both ray bundles emanating from points on and off the optical axis are traced through the system. The result of the simulation is shown in Fig. 2.11.

```
% free space
z = 10e-2;          % free-space propagation distance
F = [1 z;...        % free-space propagation matrix
     0 1];
numRays = 50;       % number of rays
maxAngle = 5e-2;    % maximum (paraxial) angle
% ray bundle at x = 0:
V = [0*ones(1,numRays); linspace(-maxAngle,maxAngle,numRays)];
% set half of the positions to x ~= 0
V(1,1:2:numRays) = 1.5e-3 * ones(1, numRays/2);
% propagation step: map V to Vp ("Vprime"):
Vp = F * V;         % free space propagation (from source to
   lens)
figure(1), clf      % clf clears previous figure if present
hold on             % hold on allows to draw multiple points
cmap = jet(numRays);% get colormap to draw rays in different
   colors
for k  = 1:numRays
    plot([0 z], [V(1,k), Vp(1,k)], 'o-',...
        'color',cmap(k,:),'MarkerFaceColor',cmap(k,:))
end
hold off
%%
```

```
% lens + free-space
f = z/2;            % focal length of lens
L = [1 0; -1/f 1];  % lens matrix
Vpp = F*(L*Vp);     % lens transormation + propagation step
figure(1)
hold on
for k  = 1:numRays
    plot([z, 2*z], [Vp(1,k), Vpp(1,k)], 'o-',...
        'color',cmap(k,:),'MarkerFaceColor',cmap(k,:))
end
%%
% draw ellipse to indicate lens position (not mandatory)
dz = 2e-2;
dx = 1.5e-2;
pos = [z-dz/2 0-dx/2 dz dx];
h = rectangle('Position',pos,'Curvature',[1 1],'LineWidth',1);
axis square
grid on
h = gca;
h.LineWidth = 3;
xlabel('z / m'), ylabel('x / m')
hold off
```

Listing 2.9: rayTrace.m (Matlab)

Exercise: Translate rayTrace.m into Python or Julia. Exercise

---

2.5.6 Hadamard Multiplication

Image processing applications often require to multiply only those elements of two matrices that have the same row and column indices. Suppose $\boldsymbol{A} \in \mathcal{R}^{M\times N}, \boldsymbol{B} \in \mathcal{R}^{M\times N}$, then the element-wise product

$$\boldsymbol{C} = \boldsymbol{A} \odot \boldsymbol{B}, \tag{2.110}$$

is called *Hadamard multiplication* and defined by

$$C_{i,j} = A_{i,j} \cdot B_{i,j}. \tag{2.111}$$

Numerical example

$$\begin{bmatrix} 1 & -1 \\ 2 & 3 \\ -1 & 4 \\ 1 & 0 \end{bmatrix} \odot \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1 & -2 \\ 6 & 12 \\ -5 & 24 \\ 7 & 0 \end{bmatrix} \tag{2.112}$$

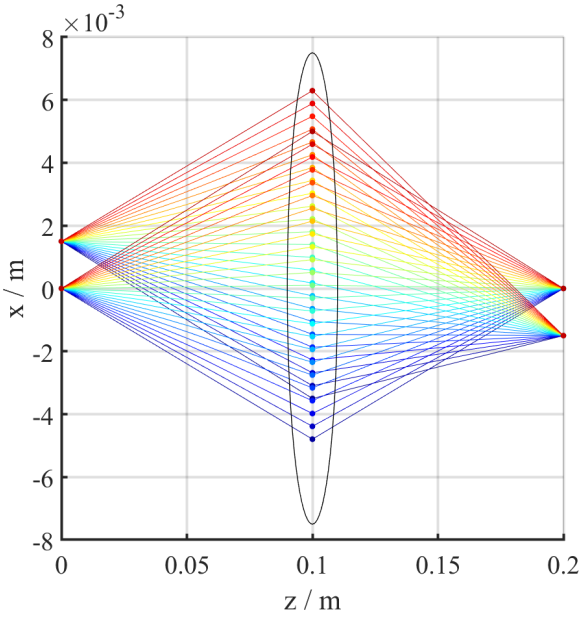


Figure 2.11: Ray tracing through a thin lens.

Image processing application: Matlab

The center of mass $\boldsymbol{r}_c = (x_c, y_c)^T$ of an image $I(x, y)$ can be calculated using the normalized first moments

$$x_c = \frac{\int xI(x,y)\,dx\,dy}{\int I(x,y)\,dx\,dy} \tag{2.113}$$

and

$$y_c = \frac{\int yI(x,y)\,dx\,dy}{\int I(x,y)\,dx\,dy}. \tag{2.114}$$

The following Matlab function calculates and returns the center of mass of an image. This example illustrates a typical use of the Hadamard product - namely, when continuous function expressions are multiplied, this corresponds often to an element-wise product in a discrete representation.

```
function [ys, xs]  = center( I )
% center image
% [r,ys,xs]  = center( I )
% input:
% I: gray image
% output:
% ys: center row
% xs: center column
[M, N] = size(I);
y = linspace(-M/2, M/2, M);
```

```
x = linspace(-N/2, N/2, N);
[X,Y] = meshgrid(x, y);
% normalize
I = I / sum2(I);
xs = sum2(I .* X);
ys = sum2(I .* Y);
return
```

Listing 2.10: center.m (Matlab)

Exercise: Reduce the number of lines in the above code and omit the meshgrid function. Instead use broadcasting[11]. Exercise

---

2.5.7 Basis Transformations

In section 2.1 we introduced vectors as a collection of numbers obeying certain rules upon addition and multiplication with other vectors. There is more to say about vectors. In practice, one of the first steps in modeling physical systems is to decide on a particular coordinate or reference frame. Typically there are multiple choices of such coordinate systems, such as Cartesian and cuvilinear coordinate systems (polar, spherical, etc.). So which coordinate system should we choose? Without insight into the symmetry of the problem this may be a difficult decision to make. The good news is that the physics of the problem is coordinate independent and a solution in one reference frame may be transformed into a solution in another reference frame.

In the physical sense vectors do not only result from an operational definition as to how they combine with other vectors. In addition, they must preserve properties such as norm and relative angle if we choose to change the coordinate system in which we describe them. It is helpful to think of a vector as a geometric quantity that exists independently of the particular reference frame in which we describe it. Once a coordinate system is selected, this fixes the components of the vector. But we could instead have chosen a different coordinate system, which would have resulted in different components. The relation between these components is described by a *basis transformation*. Here we consider transformations between Cartesian coordinate systems that share the same origin.

Example

Figure 2.12 (a) illustrates the representation of a vector (red) in two particular choices of bases (black and blue). We assume the vector $\boldsymbol{u}$ has a representation $\boldsymbol{u}_{B_1} = [0, 1]^T$ in basis B_1. Suppose the coordinate system B_2 arises from B_1 by counter-clockwise rotation by an angle θ. What is the coordinate representation of $\boldsymbol{u}$ in B_2? What is the coordinate transformation $\boldsymbol{T}$ that maps from B_1 to B_2? Solution: The red vector in Fig. 2.12 (b) has only one non-zero component in B_1. In contrast, it has two non-zero components in

[11] The broadcasting rules for Hadamard products are the same as for conventional matrix-matrix addition, i.e. the smaller array is expanded to a compatible size by repeating it along its singleton dimension.

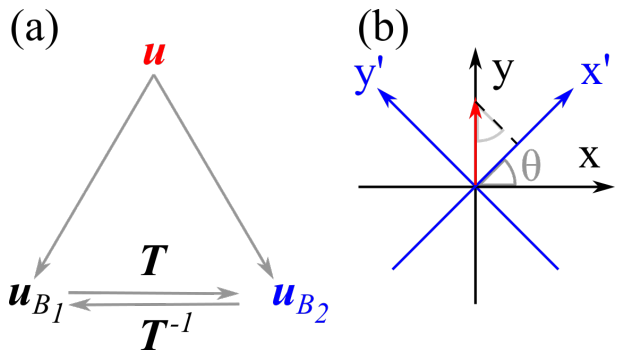


Figure 2.12: Basis transformation between coordinate representations of a vector. (a) The vector on top (red) may be thought of as existing independently of the choice of a particular coordinate system or basis in which we represent it. We may choose a particular basis (black) and later discover that the solution is more suitably expressed in another basis (blue). The vector representation can be mapped from the first basis B_1 into the second basis B_2 by means of a coordinate transformation T. Similarly, we may go the opposite direction via T^{-1}. (b) Example of a coordinate rotation.

B_2, namely $x' = \sin(\theta)$ and $y' = \cos(\theta)$. Similarly, we find that a vector $[1,0]^T$ in B_1 has the representation $[\cos(\theta), -\sin(\theta)]^T$ in B_2. In summary, we know how the unit vectors $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ in B_1 map into B_2

$$\boldsymbol{T}\boldsymbol{e}_1 = \boldsymbol{T}_1 = \begin{bmatrix} \cos(\theta) \\ -\sin(\theta) \end{bmatrix}, \tag{2.115}$$

and

$$\boldsymbol{T}\boldsymbol{e}_2 = \boldsymbol{T}_2 = \begin{bmatrix} \sin(\theta) \\ \cos(\theta) \end{bmatrix}. \tag{2.116}$$

With this we find

$$\boldsymbol{T} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}. \tag{2.117}$$

Notice that a counter-clockwise rotation of a coordinate system has the same effect on the components of a vector as a clock-wise rotation of the same vector.

2.5.8 Orthonormal matrices

The rotation matrix in Eq. 2.117 is a special instance of what is called an *orthonormal matrix*. An orthonormal matrix $\boldsymbol{Q} \in \mathcal{R}^{N\times N}$ is a square matrix defined by the property

$$\boldsymbol{Q}\cdot\boldsymbol{Q}^T = \boldsymbol{I}, \tag{2.118}$$

$$\boldsymbol{Q}^T\cdot\boldsymbol{Q} = \boldsymbol{I}, \tag{2.119}$$

where $\boldsymbol{I}$ is the identity matrix. Orthonormal matrix multiplications preserve the inner product of two vectors $\boldsymbol{u} = \boldsymbol{Q}\boldsymbol{x}$ and $\boldsymbol{v} = \boldsymbol{Q}\boldsymbol{y}$,

$$\boldsymbol{u}^T\boldsymbol{v} = \boldsymbol{x}^T\underbrace{\boldsymbol{Q}^T\boldsymbol{Q}}_{\boldsymbol{I}}\boldsymbol{y} = \boldsymbol{x}^T\boldsymbol{y}. \tag{2.120}$$

For the particular choice $\boldsymbol{u} = \boldsymbol{v} = \boldsymbol{Q}\boldsymbol{x}$, we have

$$\boldsymbol{u}^T\boldsymbol{u} = \boldsymbol{x}^T\boldsymbol{x}, \tag{2.121}$$

implying that orthonormal transformations preserve length. The reader may show that orthonormal transformations also preserve angles. The analog of an orthonormal matrix for complex-valued vectors are *unitary matrices*. An important unitary transformation is the Fourier transform, which will be discussed in section 2.11.2 and used in depth throughout this book. Exercise

2.5.9 QR decomposition

Suppose we are given a square matrix $\boldsymbol{A} \in \mathcal{R}^{N\times N}$ with linearly independent columns. Then the Gram-Schmidt algorithm (subsection 2.4.5) orthonormalizes the columns of $\boldsymbol{A} = [\boldsymbol{A}_1, \boldsymbol{A}_2, \ldots, \boldsymbol{A}_N]$ into an orthogonal matrix $\boldsymbol{Q} = [\boldsymbol{Q}_1, \boldsymbol{Q}_2, \ldots, \boldsymbol{Q}_N]$. The Gram-Schmidt started by chosing $\boldsymbol{Q}_1 = \boldsymbol{A}_1/\left\|\boldsymbol{A}_1\right\|_2$. The next basis vector $\boldsymbol{Q}_2$ was found by subtracting the projection of $\boldsymbol{A}_2$ along the direction of $\boldsymbol{Q}_1$ (from $\boldsymbol{A}_2$) and normalizing the result. Thus $\boldsymbol{Q}_2$ depends only on $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$. In the same fashion, $\boldsymbol{Q}_k$ depends on $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_k$. Thus we can decompose $\boldsymbol{A}$ into a product of an orthonormal matrix and a upper triangular matrix

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R} = [\boldsymbol{Q}_1, \boldsymbol{Q}_2, \ldots, \boldsymbol{Q}_N] \cdot \begin{bmatrix} R_{1,1} & R_{1,2} & \ldots & R_{1,N} \\ 0 & R_{2,2} & \ldots & R_{2,N} \\ 0 & 0 & \ldots & R_{3,N} \\ \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & R_{N,N} \end{bmatrix}, \tag{2.122}$$

which is called the *QR decomposition* (or *QR factorization*). The QR decomposition is simply the result of the Gram-Schmidt algorithm expressed in matrix form. Notice that the QR decomposition is special in that it is the decomposition into an orthonormal and an upper triangular matrix.

2.6 Inverse Matrix

The matrix inverse $\boldsymbol{A}^{-1}$ of a square matrix $\boldsymbol{A}$ is defined via the properties

$$\boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{I} \tag{2.123}$$

and

$$\boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{I}. \tag{2.124}$$

If a matrix inverse with these properties exist, we can formerly solve a linear system of equations by

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \quad | \quad \cdot \boldsymbol{A}^{-1} \tag{2.125}$$

$$\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}. \tag{2.126}$$

Likewise, matrix equalities of the form

$$\boldsymbol{A}\boldsymbol{X} = \boldsymbol{B} \tag{2.127}$$

may formally be solved via

$$\boldsymbol{X} = \boldsymbol{A}^{-1}\boldsymbol{B}, \tag{2.128}$$

provided $\boldsymbol{A}^{-1}$ exists. For the inverse matrix $\boldsymbol{A}^{-1}$ to exist, $\boldsymbol{A}$ needs to be full rank. In numerical practice, some matrices are full rank but close to singular, which means that computation of the inverse matrix is computationally unstable. In this case, *regularization* techniques can help. We will have to say more about this topic in subsection 2.9.4. For the following example we assume the inverse matrix exists. The computation of a matrix inverse comes at a cost of $\mathcal{O}\left(N^3\right)$.

2.7 Eigendecompositions

Diagonal matrices are at the core of high-performance computing. It turns out that a large class of non-diagonal matrices can be diagonalized, first and foremost symmetric matrices. With such diagonalizations matrix powers can be computed efficiently.

2.7.1 Eigenproblems

An eigenvector is a special input to a matrix that does not change its direction after multiplication with $\boldsymbol{A}$,

$$\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x}. \tag{2.129}$$

Here $\boldsymbol{x}$ and λ are referred to as *eigenvector* and *eigenvalue*, respectively. From this, it directly follows that $\boldsymbol{x}$ is an eigenvector for $\boldsymbol{A}^n$ with eigenvalue λ^n,

$$\boldsymbol{A}^n\boldsymbol{x} = \lambda^n\boldsymbol{x}. \tag{2.130}$$

Once known, eigenvectors can lead to significant performance improvements in the computation of matrix powers. The question then is how to compute eigenvectors. From the defining equation (Eq. 2.129), an eigenpair $\boldsymbol{x}$ and λ forms a solution to the equation

$$\left(\boldsymbol{A} - \lambda\boldsymbol{I}\right)\boldsymbol{x} = 0. \tag{2.131}$$

Solving this equation by hand for small matrices is typically performed by solving the characteristic polynomial

$$\det\left(\boldsymbol{A} - \lambda\boldsymbol{I}\right) = 0, \tag{2.132}$$

where det is the determinant function[12]. In practice, when large scale systems are concerned, the eigenvalues and vectors of a matrix are found numerically. In Matlab, the command `[X, D] = eig(A)` returns a sequence of eigenvectors along the columns of `X` and the eigenvalues along the main diagonal of `D`. Often only the k largest (or dominant) eigenvectors and eigenvalues are of concern, for which the Matlab command `[X, D] = eigs(A,k)` is more efficient.

[12] https://en.wikipedia.org/wiki/Determinant

2.7.2 The Power Method[13]

Suppose a matrix $\boldsymbol{A} \in \mathcal{C}^{N \times N}$ has linearly independent eigenvectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$. Then we may expand any vector $\boldsymbol{v}$ into a basis of eigenvectors

$$\boldsymbol{v} = c_1 \boldsymbol{x}_1 + \ldots + c_N \boldsymbol{x}_N. \tag{2.133}$$

Multiply this equation by $\boldsymbol{A}$ to get

$$\boldsymbol{A}\boldsymbol{v} = c_1 \lambda_1 \boldsymbol{x}_1 + \ldots + c_N \lambda_N \boldsymbol{x}_N. \tag{2.134}$$

Suppose now that λ_k has the largest magnitude among all eigenvalues, then for large n

$$\boldsymbol{A}^n \boldsymbol{v} = c_1 \lambda_1^n \boldsymbol{x}_1 + \ldots + c_k \lambda_k^n \boldsymbol{x}_k + \ldots + c_N \lambda_N^n \boldsymbol{x}_N \approx c_k \lambda_k^n \boldsymbol{x}_k. \tag{2.135}$$

This suggests the following algorithm, called *power method*, to compute the dominant eigenvalue: Start with a random initial vector $\boldsymbol{v}_0$. Set $j = 0$. Then for a predefined number of iterations,

1. Compute $v_{j+1/2} = \boldsymbol{A}\boldsymbol{v}_j$.
2. Compute $v_{j+1} = v_{j+1/2} / \left\| v_{j+1/2} \right\|_2$ to obtain a normalized version of this eigendirection, i.e. an estimation of the eigenvector with the largest eigenvalue.

Example

Consider the following Matlab code:

```
A1 = [1 2 7; 1 4 5; 3 2 5];
A2 = [1 1 -2; 1 -2 1; -2 1 1];
A3 = A2 + eye(size(A2));
A4 = [1 2 3; 0 1 2; 0 0 1];
% un/comment (ctrl + r/t) each of the following examples
% example 1:
% A = A1;
% example 2:
% A = A2;
% example 3:
% A = A3;
% example 4:
A = A4;
% initialize start vector
v = rand(3,1,'single');

numIter = 10;

% power method
for k = 1:numIter
```

[13] also known as *power iteration*

```
21      v = A*v;
22      v = v / sqrt(v'*v);
23  end
24  % eig solution
25  [X,D] = eig(A);
26
27  % show results
28  disp(v)
29  disp(diag(D))
30  disp(X)
```

Listing 2.11: powerMethod.m (Matlab)

The first example produces

$$\boldsymbol{v} = \begin{bmatrix} 0.5774 \\ 0.5774 \\ 0.5774 \end{bmatrix}, \operatorname{diag}(\boldsymbol{D}) = \begin{bmatrix} 10.0000 \\ -2.0000 \\ +2.0000 \end{bmatrix}, \boldsymbol{X} = \begin{bmatrix} -0.5774 & -0.8729 & +0.3015 \\ -0.5774 & -0.2182 & -0.9045 \\ -0.5774 & +0.4364 & +0.3015 \end{bmatrix}. \tag{2.136}$$

We see that the largest eigenvalue is $\lambda = 10$. The corresponding eigenvector is the first column of $\boldsymbol{X}$ (save for a minus sign). The power method produced a vector $\boldsymbol{v}$ colinear to the dominant eigenvector of $\boldsymbol{A}$ (as displayed in the first column of $\boldsymbol{X}$). Note that the power method can also yield a flipped, negative eigenvector which is equally well a solution of equation (Eq. 2.129).

In the second example,

$$\boldsymbol{v} = \begin{bmatrix} 0.7805 \\ -0.5979 \\ -0.1826 \end{bmatrix}, \operatorname{diag}(\boldsymbol{D}) = \begin{bmatrix} -3.0000 \\ 0.0000 \\ 3.0000 \end{bmatrix}, \boldsymbol{X} = \begin{bmatrix} +0.4082 & +0.5774 & -0.7071 \\ -0.8165 & +0.5774 & -0.0000 \\ +0.4082 & +0.5774 & +0.7071 \end{bmatrix}. \tag{2.137}$$

The power method failed to produce an eigenvector of $\boldsymbol{A}_2$! A closer look at the eigenvalues in $\boldsymbol{D}$ reveals the problem: When the largest eigenvalues are equal in magnitude, the power method will not give preference to any of the eigenvectors.

In some circumstances the problem in example 2 may be resolved with a little trick. If we add a constant times the identity matrix to the original problem, the eigenvectors remain the same since

$$(\boldsymbol{A} + \sigma \boldsymbol{I})\,\boldsymbol{x} = \boldsymbol{A}\boldsymbol{x} + \sigma \boldsymbol{x} = \lambda \boldsymbol{x} + \sigma \boldsymbol{x} = (\lambda + \sigma)\,\boldsymbol{x}. \tag{2.138}$$

The eigenvalues have shifted by an amount σ, which possibly gives preference to one eigenvector. In example 3 we get,

$$\boldsymbol{v} = \begin{bmatrix} -0.7071 \\ 0 \\ 0.7071 \end{bmatrix}, \operatorname{diag}(\boldsymbol{D}) = \begin{bmatrix} -2.0000 \\ 1.0000 \\ 4.0000 \end{bmatrix}, \boldsymbol{X} = \begin{bmatrix} 0.4082 & 0.5774 & -0.7071 \\ -0.8165 & 0.5774 & 0.0000 \\ 0.4082 & 0.5774 & 0.7071 \end{bmatrix}. \tag{2.139}$$

Subtracting 1 from the eigenvalues of $\boldsymbol{A}_2 + \boldsymbol{I}$ gives the correct eigenvalues for $\boldsymbol{A}_2$.

2.7.3 Symmetric and Hermitian matrices

In the previous subsection we motivated the power method by first expanding an arbitrary vector in a basis of eigenvectors (see Eq. 2.133) and subsequently looked into the evolution of the coefficients under multiplication by the original matrix. For matrices that a have dominant eigenvalue, we saw that the power method converges to the associated eigenvector. However, it is important to realize that not all matrices lead to linearly independent eigenvectors[14]. The key question is then, is there a sufficient condition for a matrix to have a complete set of eigenvectors?

Theorem: Hermitian matrices have real eigenvalues. Eigenvectors corresponding to different eigenvalues are orthogonal.

Proof: Suppose

$$\boldsymbol{A}\boldsymbol{x}_1 = \lambda_1 \boldsymbol{x}_1, \tag{2.140}$$

$$\boldsymbol{A}\boldsymbol{x}_2 = \lambda_2 \boldsymbol{x}_2. \tag{2.141}$$

Multiplying the first equation from the left by $\boldsymbol{x}_2^\dagger$ and the second by $\boldsymbol{x}_1^\dagger$, we get

$$\boldsymbol{x}_2^\dagger \boldsymbol{A}\boldsymbol{x}_1 = \lambda_1 \boldsymbol{x}_2^\dagger \boldsymbol{x}_1, \tag{2.142}$$

$$\boldsymbol{x}_1^\dagger \boldsymbol{A}\boldsymbol{x}_2 = \lambda_2 \boldsymbol{x}_1^\dagger \boldsymbol{x}_2. \tag{2.143}$$

Conjugate the latter equation and subtract it from the former to get

$$\boldsymbol{x}_2^\dagger \boldsymbol{x}_1 \left(\lambda_1 - \lambda_2^*\right) = 0, \tag{2.144}$$

where we used that by definition $\boldsymbol{A}^\dagger = \boldsymbol{A}$. If $\boldsymbol{x}_1 = \boldsymbol{x}_2$, then it directly follows that $\lambda_1 = \lambda_1^*$. Thus Hermitian matrices have real eigenvalues. If $\lambda_1 \neq \lambda_2$, then

$$\boldsymbol{x}_2^\dagger \boldsymbol{x}_1 = 0, \tag{2.145}$$

stating that $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are orthogonal. The special case of symmetric matrices is contained in the more general case of Hermitian matrices. This completes the proof.

Can two different eigenvectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ have the same eigenvalues $\lambda_1 = \lambda_2$? Yes, this is referred to as *degeneracy*. In this case, we may find orthogonal eigenvectors corresponding to the same eigenvalue. As an example, consider the identity matrix

$$\boldsymbol{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.146}$$

Clearly, the matrix is Hermitian and any vector satisfies

$$\boldsymbol{I}\boldsymbol{x} = 1\boldsymbol{x}, \tag{2.147}$$

[14] Can you find the eigenvectors and eigenvalues of the matrix $\boldsymbol{A} = \begin{bmatrix} \mathbf{1} & \mathbf{2} & \mathbf{3} \\ \mathbf{0} & \mathbf{1} & \mathbf{2} \\ \mathbf{0} & \mathbf{0} & \mathbf{3} \end{bmatrix}$, that is, do you find a pair $(\boldsymbol{x}, \lambda)$ that satisfies $\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x}$? Does this matrix have a complete set of eigenvectors?

so in this example we may choose any set of orthogonal $\mathcal{C}^3$ vectors, each corresponding to the same eigenvalue $\lambda = 1$.

Suppose now that we have found all eigenvectors of a Hermitian matrix $\boldsymbol{A}$ and arrange them as columns in a matrix $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N]$. Then

$$\boldsymbol{AX} = [\boldsymbol{A}\boldsymbol{x}_1, \boldsymbol{A}\boldsymbol{x}_2, \ldots, \boldsymbol{A}\boldsymbol{x}_N] \tag{2.148}$$

$$= [\lambda_1\boldsymbol{x}_1, \lambda_2\boldsymbol{x}_2, \ldots, \lambda_N\boldsymbol{x}_N] \tag{2.149}$$

$$= \boldsymbol{X}\Lambda, \tag{2.150}$$

where

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 & \ldots & 0 \\ 0 & \lambda_2 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & \lambda_N \end{bmatrix} \tag{2.151}$$

is a diagonal matrix with eigenvalues. Because we can choose the columns of $\boldsymbol{X}$ to be orthonormal, we have

$$\boldsymbol{A} = \boldsymbol{X}\Lambda\boldsymbol{X}^\dagger. \tag{2.152}$$

This matrix factorization is always possible for Hermitian matrices. For real-valued symmetric matrices

$$\boldsymbol{A} = \boldsymbol{X}\Lambda\boldsymbol{X}^T. \tag{2.153}$$

In this form we can efficiently compute matrix powers. For example

$$\boldsymbol{A}^2 = \boldsymbol{X}\Lambda\underbrace{\boldsymbol{X}^T\boldsymbol{X}}_{\boldsymbol{I}}\Lambda\boldsymbol{X}^T = \boldsymbol{X}\Lambda^2\boldsymbol{X}^T, \tag{2.154}$$

and more generally (by induction),

$$\boldsymbol{A}^n = \boldsymbol{X}\Lambda^n\boldsymbol{X}^T, \tag{2.155}$$

where we made use of the orthogonality of $\boldsymbol{X}$.

Another way to write Eq. 2.153 is by means of outer products (compare Eq. 2.109)

$$\boldsymbol{A} = \boldsymbol{X}\Lambda\boldsymbol{X}^T \tag{2.156}$$

$$= [\lambda_1\boldsymbol{x}_1, \lambda_2\boldsymbol{x}_2, \ldots, \lambda_N\boldsymbol{x}_N] \begin{bmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \ldots \\ \boldsymbol{x}_N^T \end{bmatrix} \tag{2.157}$$

$$= \sum_k \underbrace{\lambda_k}_{\text{weights}} \cdot \underbrace{\boldsymbol{x}_k \cdot \boldsymbol{x}_k^T}_{\text{rank-1 pieces}} . \tag{2.158}$$

Note: Sometimes, but not always, even non-symmetric matrices can be diagonalized by

$$\boldsymbol{AX} = \boldsymbol{X}\Lambda. \tag{2.159}$$

However, while for non-symmetric matrices we will generally not find columns of $\boldsymbol{X}$ that are orthonormal, we may be lucky and find independent columns. In this case an inverse of $\boldsymbol{X}$ may be found, so that

$$\boldsymbol{A} = \boldsymbol{X}\Lambda\boldsymbol{X}^{-1}. \tag{2.160}$$

We can then efficiently compute matrix powers,

$$\boldsymbol{A}^2 = \boldsymbol{X}\Lambda\underbrace{\boldsymbol{X}^{-1}\boldsymbol{X}}_{\boldsymbol{I}}\Lambda\boldsymbol{X}^{-1} = \boldsymbol{X}\Lambda^2\boldsymbol{X}^{-1} \tag{2.161}$$

and more generally

$$\boldsymbol{A}^n = \boldsymbol{X}\Lambda^n\boldsymbol{X}^{-1}. \tag{2.162}$$

2.7.4 Rayleigh quotients

Consider the eigenpair $(\boldsymbol{x}, \lambda)$ solving the eigenvalue problem to a symmetric matrix $\boldsymbol{A}$

$$\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x}. \tag{2.163}$$

Multiplying each side by $\boldsymbol{x}^\dagger$ from the left gives

$$\boldsymbol{x}^\dagger\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x}^\dagger\boldsymbol{x}, \tag{2.164}$$

or

$$\lambda = \frac{\boldsymbol{x}^\dagger\boldsymbol{A}\boldsymbol{x}}{\boldsymbol{x}^\dagger\boldsymbol{x}}. \tag{2.165}$$

Suppose now we take any vector $\boldsymbol{v}$ and define the *Rayleigh quotient*

$$R = \frac{\boldsymbol{v}^\dagger\boldsymbol{A}\boldsymbol{v}}{\boldsymbol{v}^\dagger\boldsymbol{v}}. \tag{2.166}$$

For which vector $\boldsymbol{v}$ does the Rayleigh quotient of $\boldsymbol{A}$ attain its maximum? Assuming that $\boldsymbol{A}$ is symmetric, there exist an orthogonal eigenbasis $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]$ into which $\boldsymbol{v}$ can be expanded,

$$\boldsymbol{v} = \sum_{k=1}^{N} c_k\boldsymbol{x}_{\boldsymbol{k}}. \tag{2.167}$$

Substituting this into the expression for the Rayleigh quotient, we get

$$R = \frac{\sum_{k=1}^{N}\lambda_k c_k^{\,2}}{\sum_{k=1}^{N} c_k^{\,2}}. \tag{2.168}$$

A necessary condition for this expression to be maximized is that the gradient with respect to the eigenbasis expansion coefficients vanishes,

$$\frac{\partial R}{\partial c_l} = \frac{2\lambda_l c_l\sum_{k=1}^{N} c_k^{\,2} - 2c_l\sum_{k=1}^{N}\lambda_k c_k^{\,2}}{\left(\sum_{k=1}^{N} c_k^{\,2}\right)^2} = 0 \tag{2.169}$$

or

$$\lambda_l\sum_{k=1}^{N} c_k^{\,2} - \sum_{k=1}^{N}\lambda_k c_k^{\,2} = 0 \tag{2.170}$$

This expression holds true when exactly one of the coefficients c_k is non-zero and all other coefficients vanish. But then $\boldsymbol{v}$ must be pointing in the direction of $\boldsymbol{x}_k$, which is to say $\boldsymbol{v}$

is an eigenvector. We may then simply pick the eigenvector corresponding to the largest eigenvalue to maximize the Rayleigh quotient in Eq. 2.168.

What is the Rayleigh quotient good for? In a number of problems the ratio

$$R = \frac{\boldsymbol{v}^\dagger \boldsymbol{A} \boldsymbol{v}}{\boldsymbol{v}^\dagger \boldsymbol{v}} \tag{2.171}$$

can be regarded as a particular kind of cost function (or figure of merit). For instance, in wavefront shaping applications we wish to maximize the amount of radiation that resides in a particular region, both in the focal plane and in another plane in which the beam is controlled. Before we illustrate this as an example, we will have to add yet another ingredient to get to the full picture.

2.7.5 Generalized eigenvalue problems and generalized Rayleigh quotients

A generalized eigenproblem is defined by

$$\boldsymbol{A}\boldsymbol{x} = \lambda \boldsymbol{B}\boldsymbol{x}. \tag{2.172}$$

If an inverse of $\boldsymbol{B}$ exists, we may convert this problem into a standard eigenproblem,

$$\boldsymbol{B}^{-1}\boldsymbol{A}\boldsymbol{x} = \lambda \boldsymbol{x}, \tag{2.173}$$

where $\boldsymbol{x}$ is an eigenvector of the matrix $\boldsymbol{B}^{-1}\boldsymbol{A}$ with corresponding eigenvalue λ. We may take a similar route as in the previous subjection to convert Eqn. 2.172 into

$$\lambda = \frac{\boldsymbol{x}^\dagger \boldsymbol{A} \boldsymbol{x}}{\boldsymbol{x}^\dagger \boldsymbol{B} \boldsymbol{x}}. \tag{2.174}$$

This motivates the definition of the generalized Rayleigh quotient

$$R = \frac{\boldsymbol{v}^\dagger \boldsymbol{A} \boldsymbol{v}}{\boldsymbol{v}^\dagger \boldsymbol{B} \boldsymbol{v}}. \tag{2.175}$$

A similar proof as in the previous subsection leads to the insight that generalized eigenvectors of $\boldsymbol{B}^{-1}\boldsymbol{A}$ maximize the generalized Rayleigh quotient.

Example: wavefront shaping

Although we have not yet described wave propagation, we will already mention here that far-field diffraction in the Fourier domain can be approximated by a two-dimensional Fourier transform, represented by a unitary matrix $\boldsymbol{F}$. In practice, this matrix is computed via the fast Fourier transform algorithm, which recursively splits it into smaller and smaller constituents of repeating building blocks; more on this will follow in later sections. In this section we will make use of the two-dimensional Fourier transform. We will discuss an approach to wavefront shaping that makes use of the notion of generalized eigenvectors. The underlying ideas have to be presented here in a somewhat "spoon-feeding" way. We will start with a simple picture, which is simple to understand but which unfortunately

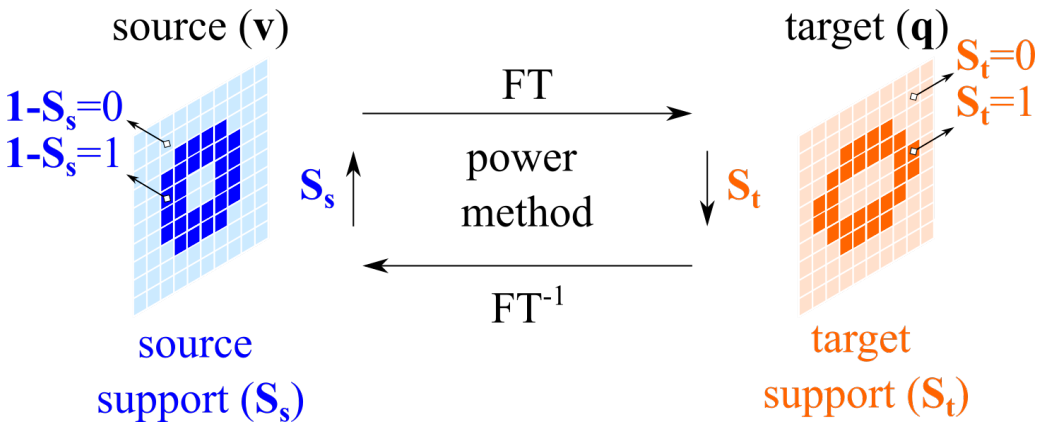


Figure 2.13: Wavefront shaping experimental geometry with circular support in source (real space $\boldsymbol{v}$) and elliptical support in target plane (Fourier space $\boldsymbol{q}$).

results in numerical problems. This simplified picture is subsequently refined to avoid the aforementioned complications.

In wavefront shaping applications, we are typically using one or multiple finite-sized adaptive optical elements, for instance a digital micromirror device (DMD) or a spatial light modulator (SLM) - or both to control an electric field. How should we manipulate a plane wave (in both amplitude and phase) by an adaptive optical element such that a desired radiation pattern is observed in the far field? In the following, we refer to the plane of the adaptive optics - under the simplified assumption that we can modify both amplitude and phase - as the *source*, while we refer to the far field as the *target*. A schematic of this wavefront shaping configuration is shown in Fig. 2.13.

Let us tackle this problem from a linear algebra perspective. We may interpret the generalized Rayleigh quotient

$$R = \frac{\boldsymbol{v}^\dagger \boldsymbol{A} \boldsymbol{v}}{\boldsymbol{v}^\dagger \boldsymbol{B} \boldsymbol{v}} \tag{2.176}$$

as a figure of merit. Both the numerator and the denominator is a scalar and we aim to maximize the ratio of the two. The matrix $\boldsymbol{A}$ in the numerator gives weight to particular pixels in the target plane support. In addition, we can embed a propagator into $\boldsymbol{A}$. Likewise, we may choose the matrix $\boldsymbol{B}$ in the denominator to measure how much our source complies with a desired constraint (to be specified below). To see this, suppose we choose

$$\boldsymbol{A} = \boldsymbol{F}^\dagger \boldsymbol{S}_t \boldsymbol{F} \tag{2.177}$$

and

$$\boldsymbol{B} = \boldsymbol{I} - \boldsymbol{S}_s, \tag{2.178}$$

where $\boldsymbol{F}$ is a two-dimensional discrete Fourier transform presented in matrix form, $\boldsymbol{S}_t$ and $\boldsymbol{S}_s$ are diagonal matrices, and $\boldsymbol{I}$ is the identity matrix. From these definitions it immediately follows that $\boldsymbol{A}$ and $\boldsymbol{B}$ are Hermitian matrices. These diagonal matrices $\boldsymbol{S}_t$ and $\boldsymbol{S}_s$ determine whether signal present in a component of $\boldsymbol{v}$ is penalized or promoted (explanation below).

How do we maximize the generalized Rayleigh quotient R? We want the numerator

$$\boldsymbol{v}^\dagger \boldsymbol{A} \boldsymbol{v} = \boldsymbol{v}^\dagger \boldsymbol{F}^\dagger \boldsymbol{S}_t \boldsymbol{F} \boldsymbol{v} = \boldsymbol{q}^\dagger \boldsymbol{S}_t \boldsymbol{q} \tag{2.179}$$

to be large. This is the intensity that ends up in the target plane pixels where $\boldsymbol{S}_t$ is non-zero. Here we substituted $\boldsymbol{q} = \boldsymbol{F}\boldsymbol{v}$, which propagates a vector from the source ($\boldsymbol{v}$) to the target plane ($\boldsymbol{q}$) via a two-dimensional discrete Fourier transform $\boldsymbol{F}$. Let us expand the numerator into the form

$$\boldsymbol{q}^\dagger \boldsymbol{S}_t \boldsymbol{q} = \begin{bmatrix} q_1^* & q_2^* & \dots & q_N^* \end{bmatrix} \begin{bmatrix} S_{t;1,1} & 0 & \dots & 0 \\ 0 & S_{t;2,2} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & S_{t;N,N} \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ \dots \\ q_N \end{bmatrix} \tag{2.180}$$

$$= q_1^* S_{t;1,1} q_1 + q_2^* S_{t;2,2} q_2 + \ldots + q_N^* S_{t;N,N} q_N \tag{2.181}$$

$$= \sum_k S_{t;k,k} \left| q_k \right|^2, \tag{2.182}$$

which is a weighted sum over the intensity in the target plane. We can choose the diagonal elements of $\boldsymbol{S}_t$ to determine how strong certain pixels in the target plane $\boldsymbol{q}$ may contribute to the numerator (which we seek to maximize). For simplicity, let us assume we choose $\boldsymbol{S}_t$ to be binary. If we shape the wavefront such that signal ends up in pixel k, then there are two possibilities: First, if we choose $S_{t;k,k} = 1$, then signal in pixel k is desired and it contributes to the sum in Eq. 2.182. Second, if we choose $S_{t;k,k} = 0$, then signal in pixel k is unwanted. If under this condition q_k is non-zero, meaning that signal ends up in pixel k, this component does not contribute in increasing the generalized Rayleigh quotient. This is because $S_{t;k,k} = 0$ gives zero weight to pixel k and thus does not increase the sum in Eq. 2.182. We also know that because $\boldsymbol{F}$ is a unitary transform, it preserves the energy (recall Eq. 2.120) from the source to the target plane, given by the L2 norms of $\boldsymbol{v}$ and $\boldsymbol{q}$. But then, giving signal to a pixel with $S_{t;k,k} = 0$ is a waste of energy. The energy goes somewhere, but is not streamlined in the right direction, namely the direction which would result in a larger weighted sum in the numerator.

Next we consider the denominator of the Rayleigh quotient,

$$\boldsymbol{v}^\dagger \boldsymbol{B} \boldsymbol{v} = \boldsymbol{v}^\dagger \left(\boldsymbol{I} - \boldsymbol{S}_s \right) \boldsymbol{v} = \sum_k \left(1 - S_{s;k,k} \right) \left| v_k \right|^2. \tag{2.183}$$

Suppose again for simplicity that $\boldsymbol{S}_s$ is a diagonal matrix with only binary elements. Now our goal is to push the denominator to zero in order to achieve a large generalized Rayleigh quotient. Now we run into several problems, as a result of allowing for binary values in $\boldsymbol{S}_s$. If, for instance, we choose a single diagonal element $S_{s;k,k} = 1$, then in computing the generalized Rayleigh quotient we divide by 0 given that $v_k \neq 0$ and all other components of $\boldsymbol{v}$ are zero. Hence allowing for unit diagonal elements in $\boldsymbol{S}_s$ is numerically problematic. This issue can be prevented when all elements $S_{s;k,k}$ are smaller than one or if alternatively we increase the diagonal elements of $\boldsymbol{I}$. In this case, we cannot divide by zero, but still we can allow for different weights in various locations where signal is or is not desired. Hence let us fix the requirement $1 - S_{s;k,k} > 0$. With this choice we see from Eq. 2.183 that $\boldsymbol{B}$ is *positive definite*, meaning that $\boldsymbol{v}^\dagger \boldsymbol{B} \boldsymbol{v} > 0$ for all $\boldsymbol{v}$.

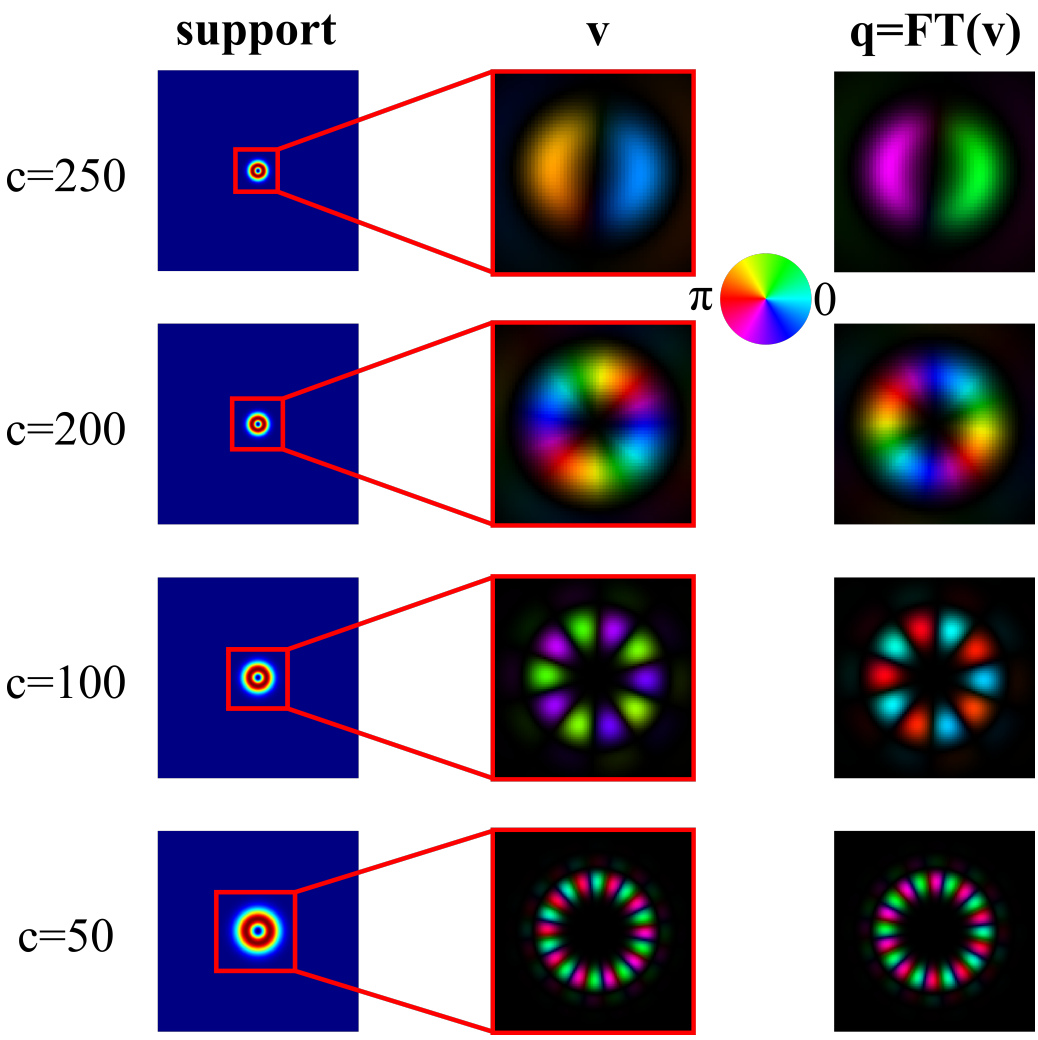


Figure 2.14: Example of wavefront shaping with annular support in both source (real space $\boldsymbol{v}$) and target plane (Fourier space $\boldsymbol{q}$). In the middle and right column, amplitude and phase and depicted as brightness and hue, respectively.

The following code gives an example where a wavefront $\boldsymbol{v}$ is sought that has maximum amplitude on annular supports both in the source and target plane. Both supports are chosen to be equal in shape in this example, but the code can easily be modified to accommodate more general situations. We apply a power method to find the maximum generalized Rayleigh quotient as the solution to the generalized eigenproblem $\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{B}\boldsymbol{x}$ or $\boldsymbol{B}^{-1}\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x}$. We emphasize that $\boldsymbol{B}^{-1}$ is guaranteed to exist as a result of $\boldsymbol{B}$ being Hermitian and positive definite. Figure 2.14 shows the optimal solution for a varying support (see left column; diameter parameterized by c) in both the source (middle column) and the target plane (middle column). In the middle column we show non-trivial, generalized eigenvectors $\boldsymbol{v}$ that, apart from a phase offset, reproduce themselves upon Fourier transformation $\boldsymbol{q} = \mathcal{F}\boldsymbol{v} \approx \exp(i\phi)\,\boldsymbol{v}$. Take a moment to appreciate what wonderful things we can already do with a little bit of linear algebra and programming skill!

```
N = 2^9;
x = linspace(-1,1,N);
[X,Y] = meshgrid(x);
[theta,r] = cart2pol(X,Y);

c = 50;
% source support
A = exp(-c*(X.^2 + Y.^2)) - exp(-2*c*(X.^2 + Y.^2));
% normalization
A = A / max(A(:));
% target support
B = A;

lambda = 1e0;
rng(0)
v = exp(1i*2*pi*rand(N,N));

figure(1)
imagesc(A)
axis image off
colormap(jet)

%% power method
v = v / sqrt(v(:)'*v(:));
numIter = 1e4;
figureUpdate = 1000;

for k = 0:numIter
    v = linOp(v,B,A,lambda);
    v = v / sqrt(v(:)'*v(:));

```

```
    if mod(k,figureUpdate) == 0
        figure(2)
        hsvplot(v)
        zoom(4*sqrt(c/50))

        figure(3)
        hsvplot(fft2c(v))
        zoom(4*sqrt(c/50))
        title(['iteration ', num2str(k)])
        drawnow
    end
end

function r = linOp(v,B,C,lambda)
r = 1./(1+lambda-C) .* ifft2c(B.*fft2c(v));
end

function G = fft2c(g)
G = fftshift(fft2(ifftshift(g)))/numel(g);
end

function g = ifft2c(G)
g = fftshift(ifft2(ifftshift(G)))/numel(G);
end
```

Listing 2.12: wavefrontShaping.m (Matlab)

The reader is invited to explore the possibilities of the code (wavefrontShaping.m).

Exercise: Translate wavefrontShaping.m into Python or Julia. Exercise

---

Exercise: Is it possible to find a choice for c such that the solution $\boldsymbol{v}$ has an odd symmetry (threefold, fivefold, ...)? Does the symmetry of the beam continuously change as a function of c or are there sudden (topological) changes? If so, at what values of c do the changes occur? We also encourage the reader to break the symmetry between the source and target plane supports in order to explore other exotic wavefront solutions. Exercise

---

Exercise: Can you map a circular source support into an elliptical target support? What happens when the source and support diameter jointly shrink? Exercise

---

Exercise: Finally, is the method guaranteed to converge to the right solution? Can you construct degenerate cases, that is, situations where two different solutions have the same generalized Rayleigh quotient? Exercise

---

What you should take away from this section is that the Rayleigh quotient can in some circumstances serve as a figure of merit. Despite its nonlinearity, namely the variable of interest appearing in the denominator of the generalized Rayleigh quotient that we wished

to maximize, we can apply linear algebra to obtain a solution to this particular problem at hand.

2.8 Singular Value Decomposition

2.8.1 Full SVD

In subsection 2.7.3 we saw that a square Hermitian matrix $\boldsymbol{A}$ has an eigendecomposition of the form (Eq. 2.152)

$$\boldsymbol{A} = \boldsymbol{X}\Lambda\boldsymbol{X}^\dagger, \tag{2.184}$$

where $\boldsymbol{X}$ is unitary and $\boldsymbol{\Lambda}$ is diagonal. This useful decomposition is not valid anymore when $\boldsymbol{A}$ is not Hermitian. However, another useful decomposition is the so-called *singular value decomposition* (SVD), which exists for all matrices. The SVD of a matrix $\boldsymbol{A} \in \mathcal{C}^{M\times N}$ is given by

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\dagger, \tag{2.185}$$

where $\boldsymbol{U} \in \mathcal{C}^{M\times M}$ and $\boldsymbol{V} \in \mathcal{C}^{N\times N}$ are unitary matrices and $\boldsymbol{S} \in \mathcal{R}^{M\times N}$ is a diagonal matrix with non-negative entries ($S_{i,i} = \sigma_i$). We will not give a proof of the existence of the SVD here, which the reader may find in classical texts on (numerical) Linear Algebra [17, 45].

We may compute the columns of $\boldsymbol{U}$ and $\boldsymbol{V}$ by observing that

$$\boldsymbol{A}\boldsymbol{A}^\dagger = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\dagger\boldsymbol{V}\boldsymbol{S}\boldsymbol{U}^\dagger = \boldsymbol{U}\boldsymbol{S}^2\boldsymbol{U}^\dagger = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\dagger \tag{2.186}$$

and

$$\boldsymbol{A}^\dagger\boldsymbol{A} = \boldsymbol{V}\boldsymbol{S}\boldsymbol{U}^\dagger\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\dagger = \boldsymbol{V}\boldsymbol{S}^2\boldsymbol{V}^\dagger = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^\dagger. \tag{2.187}$$

Thus the columns of $\boldsymbol{U}$ are the eigenvectors of the Hermitian matrix $\boldsymbol{A}\boldsymbol{A}^\dagger$ with eigenvalues given by the diagonal elements of $\Lambda = \boldsymbol{S}^2$. The columns of $\boldsymbol{V}$ are the eigenvectors of the Hermitian matrix $\boldsymbol{A}^\dagger\boldsymbol{A}$ with eigenvalues given by the diagonal elements of $\Lambda = \boldsymbol{S}^2$.

2.8.2 Truncated SVD

Suppose $\boldsymbol{A} \in \mathcal{C}^{M\times N}$ and let $\text{rank}\,(\boldsymbol{A}) = K < \min(M, N)$. Then the column space of $\boldsymbol{A}$ is spanned by at most K linearly independent vectors. Likewise, the row space of $\boldsymbol{A}$, which is equal to the column space of $\boldsymbol{A}^\dagger$, is spanned by at most K linearly independent

vectors. Hence the SVD may be written as

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\dagger \tag{2.188}$$

$$= \underbrace{\left[\underbrace{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_K}_{\in \mathrm{col}(\boldsymbol{A})}, \underbrace{\boldsymbol{u}_{K+1}, \ldots, \boldsymbol{u}_M}_{\perp \mathrm{col}(\boldsymbol{A})}\right]}_{M \times M} \underbrace{\begin{bmatrix} \sigma_1 & 0 & 0 & | \\ 0 & \ldots & 0 & \mathbf{0} \\ 0 & 0 & \sigma_K & | \\ - & \mathbf{0} & - & \mathbf{0} \end{bmatrix}}_{M \times N} \underbrace{\left[\underbrace{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_K}_{\in \mathrm{col}(\boldsymbol{A}^\dagger)}, \underbrace{\boldsymbol{v}_{K+1}, \ldots, \boldsymbol{v}_N}_{\perp \mathrm{col}(\boldsymbol{A}^\dagger)}\right]^\dagger}_{N \times N} \tag{2.189}$$

$$= \left[\tilde{\boldsymbol{U}}_K, \boldsymbol{u}_{K+1}, \ldots, \boldsymbol{u}_M\right] \begin{bmatrix} \tilde{\boldsymbol{S}}_K & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \left[\tilde{\boldsymbol{V}}_K, \boldsymbol{v}_{K+1}, \ldots, \boldsymbol{v}_N\right]^\dagger \tag{2.190}$$

$$= \underbrace{\tilde{\boldsymbol{U}}_K}_{M \times K} \underbrace{\tilde{\boldsymbol{S}}_K}_{K \times K} \underbrace{\tilde{\boldsymbol{V}}_K^\dagger}_{K \times N} \tag{2.191}$$

Another way to write Eq. 2.185 is to extend it into a sum of rank-1 pieces

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\dagger \tag{2.192}$$

$$= \tilde{\boldsymbol{U}}_K \tilde{\boldsymbol{S}}_K \tilde{\boldsymbol{V}}_K^\dagger \tag{2.193}$$

$$= [\sigma_1 \boldsymbol{u}_1, \sigma_2 \boldsymbol{u}_2, \ldots, \sigma_K \boldsymbol{u}_K] \begin{bmatrix} \boldsymbol{v}_1^\dagger \\ \boldsymbol{v}_2^\dagger \\ \ldots \\ \boldsymbol{v}_K^\dagger \end{bmatrix} \tag{2.194}$$

$$= \sum_{k=1}^{K} \underbrace{\sigma_k}_{\text{weights}} \cdot \underbrace{\boldsymbol{u}_k \cdot \boldsymbol{v}_k^\dagger}_{\text{rank-1 pieces}} . \tag{2.195}$$

Often it is the case that a data matrix $\boldsymbol{A}$ exhibits correlations or similarity along its rows or columns. While $\boldsymbol{A}$ might still be full (or high) rank, the singular spectrum may rapidly decay. In this case, we can get a good approximation to a full (or high) rank matrix from a small number of rank 1 pieces, i.e.

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\dagger \tag{2.196}$$

$$= \sum_{k=1}^{\min(M,N)} \sigma_k \cdot \boldsymbol{u}_k \cdot \boldsymbol{v}_k^\dagger \tag{2.197}$$

$$\approx \tilde{\boldsymbol{U}}_K \tilde{\boldsymbol{S}}_K \tilde{\boldsymbol{V}}_K^\dagger \tag{2.198}$$

$$= \sum_{k=1}^{K} \sigma_k \cdot \boldsymbol{u}_k \cdot \boldsymbol{v}_k^\dagger, \tag{2.199}$$

where $K < \min(M, N)$. The *Eckert-Young theorem* states that among all rank-K approximations to $\boldsymbol{A}$, the truncated rank-K SVD achieves the lowest error

$$\mathcal{L} = \left\| \boldsymbol{A} - \tilde{\boldsymbol{U}}_K \tilde{\boldsymbol{S}}_K \tilde{\boldsymbol{V}}_K^\dagger \right\|_F^2, \tag{2.200}$$

where $\|\ldots\|_F$ is the Frobenius norm discussed in subsection 2.10.2.

Example: Lossy compression

The following code illustrates how the truncated SVD can be used for lossy image compression. The word *lossy* refers to the fact that the original image cannot be reconstructed unambiguously from the compressed image without further a priori knowledge. The green channel from the rgb image (*Origami.jpg*) is loaded and selected. Next, a rank-K truncated SVD is used to approximate the image. The original image has $N = 720$ columns. We use the compression ratio $C = 1-K/N$ to quantify the reduction in singular vectors[15]. The original and compressed images are shown in Fig. 2.15.

```
im = single(imread('../testImages/Origami.jpg'));
im = im(:,:,2); % select green channel (out of rgb image)

[U,S,V] = svd(im);
N = min(size(im));
K = 720;

imCom = U(:,1:K)*S(1:K,1:K)*V(:,1:K)';
C = 1-K/N;  % compression ratio
D = 1/(1-C);% reduction factor

figure(1)
imagesc(imCom)
axis image off
colormap gray
title(['C = ', num2str(round(1000*(C))/10),'% (',...
               num2str(round(D)),'x)'])
```

Listing 2.13: svdCompression.m (Matlab)

The previous example illustrated the use of the SVD for (lossy) image compression. However, computing the SVD directly on the original image has the disadvantage that only entire columns or rows of the original image may be accounted for. To see this, consider again the form of the truncated SVD,

$$\boldsymbol{A} = \tilde{\boldsymbol{U}}_K \tilde{\boldsymbol{S}}_K \tilde{\boldsymbol{V}}_K^\dagger. \tag{2.201}$$

[15] We will specify a more appropriate compression measure below.

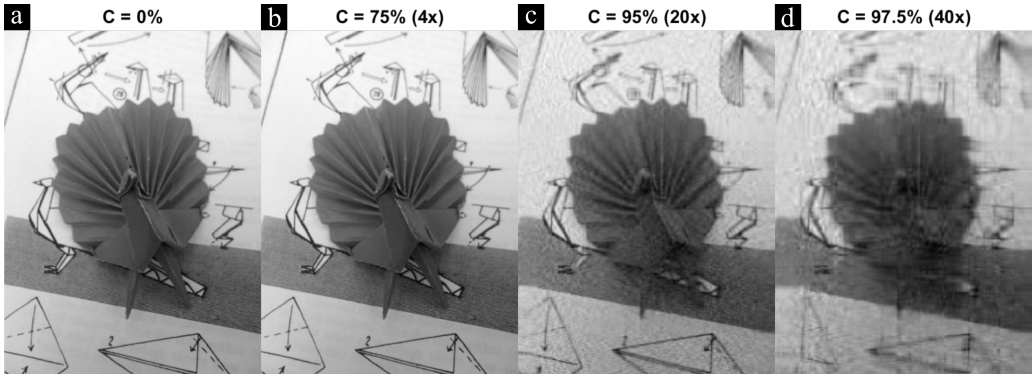


Figure 2.15: Image compression using truncated rank-K SVD on image with $N = 720$ columns. (a) Uncompressed image ($K = 720$). (b) Low compression ($K = 180$). (c) moderate compression ($K = 72$). (d) High compression ($K = 18$). The top of each panel shows the compression ratio $C = 1 - K/N$. The compressed images also show the data reduction factor N/K in parenthesis.

Defining $\boldsymbol{B} = \tilde{\boldsymbol{S}}_K \tilde{\boldsymbol{V}}_K^\dagger$, the ith column of $\boldsymbol{A}$ is approximated by

$$\begin{aligned}
\boldsymbol{A}_i &= \left(\tilde{\boldsymbol{U}}_K \boldsymbol{B}\right)_i && (2.202)\\
&= \left(\tilde{\boldsymbol{U}}_K \left[\boldsymbol{B}_1, \boldsymbol{B}_2, \ldots, \boldsymbol{B}_N\right]\right)_i && (2.203)\\
&= \left(\left[\tilde{\boldsymbol{U}}_K \boldsymbol{B}_1, \tilde{\boldsymbol{U}}_K \boldsymbol{B}_2, \ldots, \tilde{\boldsymbol{U}}_K \boldsymbol{B}_N\right]\right)_i && (2.204)\\
&= \underbrace{\tilde{\boldsymbol{U}}_K}_{M\times K} \underbrace{\boldsymbol{B}_i}_{K\times 1} && (2.205)\\
&= \sum_{j=1}^{K} \boldsymbol{B}_{i,j} \tilde{\boldsymbol{U}}_{K;j}. && (2.206)
\end{aligned}$$

So $\boldsymbol{A}_i$ is a linear combination of columns of $\tilde{\boldsymbol{U}}_K$[16] with expansion coefficients given by the ith row of $\boldsymbol{B}$. This is to say the SVD computed on the original image has the challenging task to find a truncated basis $\tilde{\boldsymbol{U}}_K$ in which every column of $\boldsymbol{A}$ can be approximately expanded. This is not easy because the columns of the original image range over a large distance where the image content changes. Image information of natural scenes is typically correlated mostly over short distances. Thus we may attempt to split an image into smaller cells, similar to a checker board. The SVD can then be used to find a suitable basis for the set of all such cells. Indeed, compression techniques like JPEG or JPEG2000 use this idea, except that the set of basis functions is predetermined (discrete cosine transform in the case of JPEG and wavelet transform in the case of JPEG2000).

Another issue with the previous example is that the compression ratio $C = 1 - K/N$ is not a realistic measure of compression. A truncated rank-K SVD decomposes the

[16] the index j in $\tilde{\boldsymbol{U}}_{K;j}$ references the jth column of $\tilde{\boldsymbol{U}}_K$.

original image with $M \cdot N$ into three matrices $\tilde{\boldsymbol{U}}_K, \tilde{\boldsymbol{S}}_K, \tilde{\boldsymbol{V}}_K$ which require a total of $M \cdot K + K \cdot K + N \cdot K = (M + K + N) \cdot K$ pixels. Thus a more precise compression measure is

$$C = 1 - (M + K + N) \cdot K / (M \cdot N) . \tag{2.207}$$

In addition, we may use

$$D = \frac{1}{1 - C} \tag{2.208}$$

to quantify the reduction in size. For example, $10\times$ would indicate a reduction in size by a factor of ten.

Example: Lossy compression with truncated SVD on cell-partitioned image

In the following code the truncated rank-K SVD is used to compress the original image subdivided into cells of size $c \times c$. As you modify the parameters, you notice how c controls the cell size of the subimages and K controls the number of possible modes inside each cell. In this way the original image is approximated by a mosaic of K modes (as contained along the columns of $\tilde{\boldsymbol{U}}_K$). Figure 2.16 shows the full-column SVD (a,b) as compared to the cell-partitioned SVD (c-f) for various choices of c and K as well as the updated definition of the compression ratio C and the reduction factor D (in parenthesis).

```
im = single(imread('../testImages/Origami.jpg'));
im = im(:,:,2); % select green channel (out of rgb image)

tic
% choose c = (2,3,4,5,6<slow!!)8,10,12,15,16,20,24,30
c = 30;
im2 = reshape_cXc(im, c);
[U,S,V] = svd(im2);
M = size(im2,1);
N = size(im2,2);
K = 10; % choose K <= c
imCom = U(:,1:K)*S(1:K,1:K)*V(:,1:K)';
C = 1-K*(M+K+N)/(M*N);
D = 1/(1-C);
% compressed image
imCom = reshape_cXc_inv(imCom,size(im,1),size(im,2),c);

figure(1)
imagesc(imCom)
axis image off
colormap gray
title({['c = ', num2str(c),', K = ',num2str(K)],...
    ['C = ', num2str(round(1000*(C))/10),'% (',num2str(round(D))
       ,'x)']})
toc

```

```
function B = reshape_cXc(A,c)
B = zeros(c^2,size(A,1)*size(A,2)/c^2);
counter = 0;
for k = 1:c:size(A,1)
    for l = 1:c:size(A,2)
        counter = counter+1;
        B(:,counter) = reshape(A(k:k+c-1,l:l+c-1),[c^2,1]);
    end
end
end

function A = reshape_cXc_inv(B,M,N,c)
A = zeros(M,N);
counter = 0;
for k = 1:c:M
    for l = 1:c:N
        counter = counter+1;
        A(k:k+c-1,l:l+c-1) = reshape(B(:,counter),[c,c]);
    end
end
end
```

Listing 2.14: svdCompressionCell.m (Matlab)

The execution time for small c is much longer than for larger c. Can you rewrite the functions in the previous code without for-loop? Exercise

2.8.3 Snapshot SVD

In order to compute the individual factors $\boldsymbol{U}, \boldsymbol{S}, \boldsymbol{V}$, we could solve the two eigenproblems

$$\boldsymbol{A}\boldsymbol{A}^\dagger \boldsymbol{u} = \lambda \boldsymbol{u} \tag{2.209}$$

and

$$\boldsymbol{A}^\dagger \boldsymbol{A}\boldsymbol{v} = \lambda \boldsymbol{v}. \tag{2.210}$$

In practice, this is oftentimes not needed. Let us assume the columns of $\boldsymbol{A}_1, \boldsymbol{A}_2, \ldots, \boldsymbol{A}_N$ of the matrix $\boldsymbol{A}$ are images reshaped into a column vector, with number of pixels in each image M much larger than the number of images N. We also assume $M \gg N$. Then the matrix $\boldsymbol{A} \in \mathcal{R}^{M \times N}$ will look rectangular. The cost in computing

$$\underbrace{\boldsymbol{A}}_{M \times N} \cdot \underbrace{\boldsymbol{A}^\dagger}_{N \times M} \tag{2.211}$$

scales with $M \cdot N \cdot M = M^2 \cdot N$ flops (recall the discussion at the end of subsection 2.5.4) and the result will be a large matrix (of size $M \times M$). In contrast, the cost in computing

$$\underbrace{\boldsymbol{A}^\dagger}_{N \times M} \cdot \underbrace{\boldsymbol{A}}_{M \times N} \tag{2.212}$$

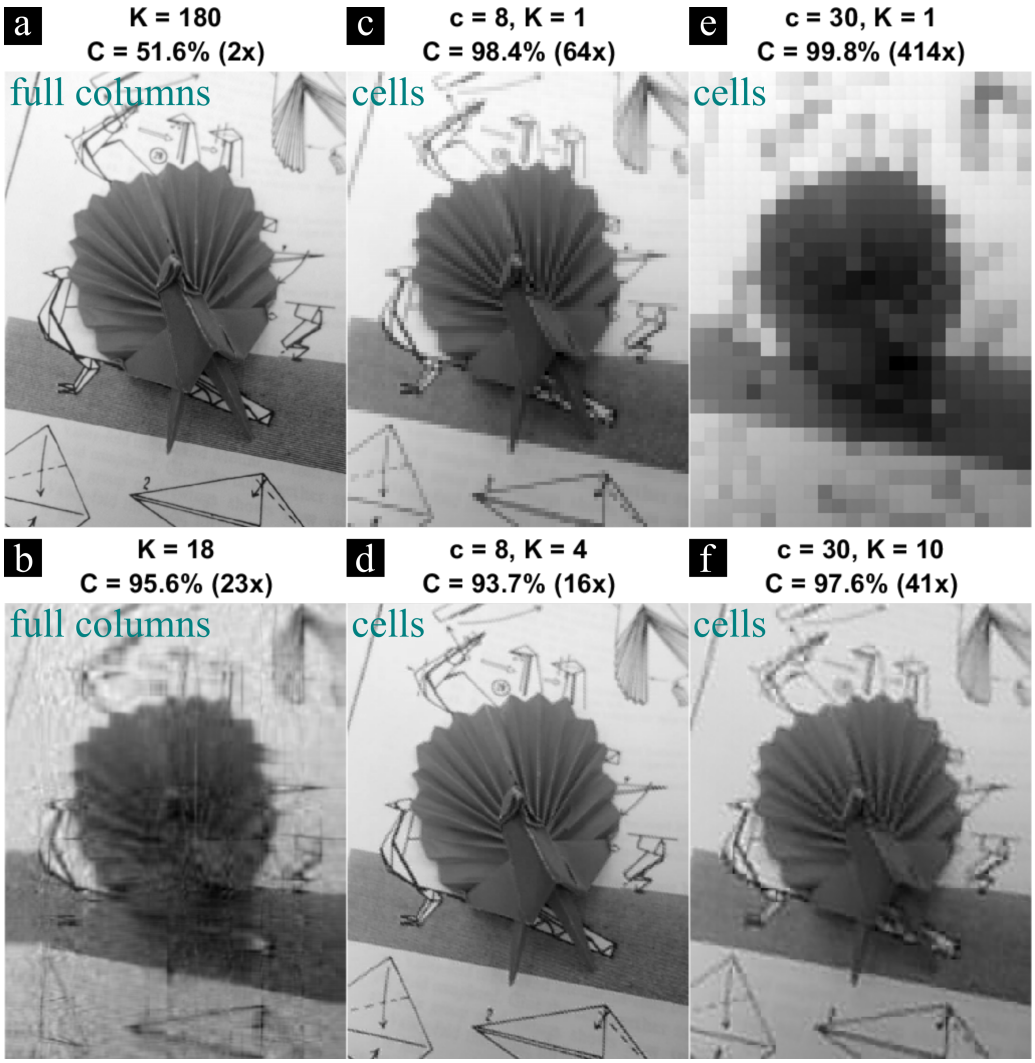


Figure 2.16: Comparison of compression based on full columns of image (left, a and b) and cells (c-f) for various choices of cell size c and rank K.

requires $N \cdot M \cdot N = N^2 \cdot M \ll M^2 \cdot N$ flops and the result $\boldsymbol{A}^\dagger \boldsymbol{A}$ $(N \times N)$ is much smaller in size than $\boldsymbol{A}\boldsymbol{A}^\dagger$ $(M \times M)$. Hence we take the route of least resistance and compute the eigendecomposition

$$\underbrace{\boldsymbol{A}^\dagger \boldsymbol{A}}_{N \times N} = \tilde{\boldsymbol{V}} \tilde{\boldsymbol{S}}^2 \tilde{\boldsymbol{V}}^\dagger, \tag{2.213}$$

giving us $\boldsymbol{V}$ and $\tilde{\boldsymbol{S}}$. But then

$$\underbrace{\boldsymbol{A}}_{M \times N} \underbrace{\tilde{\boldsymbol{V}} \tilde{\boldsymbol{S}}^{-1}}_{N \times K} = \underbrace{\tilde{\boldsymbol{U}}}_{M \times K}. \tag{2.214}$$

2.8.4 Optical application of SVD

The SVD has an important application wave propagation. We can generally formulate wave propagation in free space by the Rayleigh-Sommerfeld diffraction integral. However, the waves involved can principally scatter to arbitrary large dimensions. In contrast, both the source (a finite light source itself or a secondary light distribution insight an finite aperture or pupil) and the receiver (detector) of an optical system are finite sized. In order to maximize signal to noise in an optical system, the question arises which optical waveform to send from a finite source in order to maximize the signal strength in a finite receiver [33]. We will see that the SVD will provide an answer to this problem.

Consider once more the Helmholtz equation

$$\nabla^2 u + k^2 u = 0, \tag{2.215}$$

whose 3D solution, as discussed in section 2.7.4, may be written as

$$u(\boldsymbol{r}) = \int u(\boldsymbol{r}') \frac{\exp\left[i\boldsymbol{k}\left(\boldsymbol{r} - \boldsymbol{r}'\right)\right]}{i\lambda \left|\boldsymbol{r} - \boldsymbol{r}'\right|} \frac{z - z'}{\left|\boldsymbol{r} - \boldsymbol{r}'\right|} d\boldsymbol{r}'. \tag{2.216}$$

Here $\boldsymbol{r}' = (x', y', z')$ are source and $\boldsymbol{r} = (x, y, z)$ are target coordinates. The integral transformation in 2.216 states that each point in the field distribution in the target plane $u(\boldsymbol{r})$ depends on every point in the field distribution in the source plane, $u(\boldsymbol{r}')$. Approximating the integral by a finite sum, we get

$$u(\boldsymbol{r}_i) \approx \sum_j u\left(\boldsymbol{r}'_j\right) \frac{\exp\left[i\boldsymbol{k}\left(\boldsymbol{r}_i - \boldsymbol{r}'_j\right)\right]}{i\lambda \left|\boldsymbol{r}_i - \boldsymbol{r}'_j\right|} \frac{z - z'}{\left|\boldsymbol{r}_i - \boldsymbol{r}'_j\right|} \Delta x' \Delta y'. \tag{2.217}$$

or

$$\boldsymbol{u}_{z;\,i} = \sum_j \boldsymbol{G}_{i,j} \boldsymbol{u}_{0;\,j} \tag{2.218}$$

where

$$\boldsymbol{G}_{i,j} = \frac{\exp\left[i\boldsymbol{k}\left(\boldsymbol{r}_i - \boldsymbol{r}'_j\right)\right]}{i\lambda \left|\boldsymbol{r}_i - \boldsymbol{r}'_j\right|} \frac{z - z'}{\left|\boldsymbol{r}_i - \boldsymbol{r}'_j\right|} \Delta x' \Delta y'. \tag{2.219}$$

In matrix form,

$$\boldsymbol{u}_z = \boldsymbol{G}\boldsymbol{u}_0. \tag{2.220}$$

In practice, the source and the receiver plane are bounded, which can be modeled by means of diagonal matrices $\boldsymbol{S}$ and $\boldsymbol{R}$. With this the latter equation becomes

$$\boldsymbol{u}_z = \boldsymbol{M}\boldsymbol{u}_0, \tag{2.221}$$

where

$$\boldsymbol{M} = \boldsymbol{R}\boldsymbol{G}\boldsymbol{S}. \tag{2.222}$$

Here $\boldsymbol{S}$ truncates $\boldsymbol{u}_0$ to a finite source domain, while $\boldsymbol{R}$ truncates $\boldsymbol{G}\boldsymbol{S}\boldsymbol{u}_0$ to a finite size in the receiver domain. Let us now decompose the matrix $\boldsymbol{M}$ into an SVD,

$$\boldsymbol{M} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\dagger, \tag{2.223}$$

where $\boldsymbol{V}$ contains orthonormal modes in the finite source domain, the are mapped into orthonormal modes $\boldsymbol{U}$ in the finite receiver domain. The singular values $\boldsymbol{S}_{i,i}$ give the channel connectivity with which the source domain mode $\boldsymbol{V}_i$ is mapped into the receiver domain mode $\boldsymbol{U}_i$.

Example

We illustrate the meaning of the SVD in a numerical example (SVDmodes.m). Suppose we have a circular aperture (source) and a circular detector (receiver). Then we can simulate the impulse responses in the receiver plane due to all point sources in the aperture plane. These spherical wave responses can be cropped to the size of the detector. After a subsequent reshaping operation we can regard them as the columns of the aforementioned matrix $\boldsymbol{M}$. Applying an SVD to this matrix gives us a set of orthogonal source modes $\boldsymbol{V}$ and receiver modes $\boldsymbol{U}$ as well as the corresponding channel connectivity $\boldsymbol{S}$. The result is shown in Fig. 2.17. Notice in particular the singular spectrum which rapidly decays. A closer inspection shows 99% of the energy in 1481 source spherical waves can be represented by the first 45 modes. We can interpret the effective rank of the SVD as the *number of degrees of freedom* in the present optical system. Essentially any signal that can be sent over the present communication link between source and receiver can be expanded into a source basis of 45 modes.

```
%% sampling grid and physical parameters
% physical quantities
wavelength = 632.8e-9;  % wavelength
% detector coodinates
Nd = 2^7;               % number of samples in detector plane
dxd = 5e-6;             % detector pixel size
Ld = Nd * dxd;          % detector size
xd = linspace(-Nd/2,Nd/2,Nd)*dxd;% 1D coordinates (detector)
[Xd, Yd] = meshgrid(xd);% 2D coordinates (detector)
z = 10e-2;              % source-receiver distance
% source coordinates
dxs = wavelength*z/Ld/4;  % source sampling
Ns = Nd;        % number of samples in source plane
Ls = Ns * dxs;  % source field of view
xs = linspace(-Ns/2,Ns/2,Ns)*dxs;% 1D coordinates (source)
```

```
[Xs, Ys] = meshgrid(xs);% 2D coordinates (source)

%% get source - object eigenmodes
% generate pinhole
diameter = Ls/3;
source = circ(Xs, Ys, diameter);
source = normconv2(source, [1,1;1,1]);
figure(1); imagesc(source); colormap gray
axis image; title('source') % show result
% find non-zero source indices
[row, col] = find( source > 0 );
% number of source points
numSourcePoints = size(row,1);
% (mutually uncorrelated) spherical wavelets
sphericalWavelets = zeros(Ns, Ns, numSourcePoints,'single');
% determines how often source wavelets are shown
figureUpdateFrequency = 50;
for loop = 1:numSourcePoints
   % displacement
   R = sqrt( (Xd - Xs( row(loop), col(loop) )).^2 + ...
             (Yd - Ys( row(loop), col(loop) )).^2 + z.^2 );
   % Greens function (Rayleigh-Sommerfeld) for each point source
   sphericalWavelets(:,:,loop) = source(row(loop),col(loop))*...
                  exp(1i * 2*pi / wavelength .* R) .* z ./ R.^2;

   if mod(loop,figureUpdateFrequency)==0
       figure(2)
       hsvplot(sphericalWavelets(:,:,loop));
       axis off;
       title({'spherical wavelets in entrance pupil',...
               num2str(loop)});
       drawnow
   end
end

%% receiver aperture
receiverAperture = normconv2(circ(Xd, Yd, Ld/2), [1,1;1,1]);
figure(99)
imagesc(receiverAperture)
axis image, colormap gray
sphericalWavelets = bsxfun(@times, sphericalWavelets,
   receiverAperture);

%% get orthogonal modes
maxNumModes = 500; % maximum number of modes to be calculated
wrank = min(numSourcePoints, maxNumModes); % # modes for tsvd
```

```
[U,S,V] = tsvd(reshape(sphericalWavelets,...
                    [Nd^2, numSourcePoints]), wrank);
s = diag(S);
purity = sqrt(sum(s.^2))/sum(s)^2;

%% orthogonal source and receiver modes
w = zeros(Ns, Ns, wrank, 'single');
u = zeros(Nd, Nd, wrank, 'single');
idx = find( source > 0 );
for k = 1:wrank
    u(:,:,k) = reshape( U(:, k), Nd * [1,1] );
    temp = 0*source;
    temp(idx) = V(:,k);
    w(:,:,k) = temp;

end

%% show source and receiver modes
n = round(diameter/dxs/2) + 10;
figure(3)
subplot(1,3,1)
hsvmodeplot(w(end/2-n:end/2+n,end/2-n:end/2+n,1:min(wrank, 25)))
title('source modes'), axis off

subplot(1,3,2)
hsvmodeplot(u(:,:,1:min(wrank, 25)))
title('receiver modes'), axis off

subplot(1,3,3)
semilogy(diag(S(1:100,1:100))/trace(S),...
    'ko','MarkerFaceColor','k')
axis square, grid on
title('singular values')
```

Listing 2.15: opticalSVD.m (Matlab)

2.9 Least squares

2.9.1 Linear least squares

In many problems we wish to find a solution to a system of equations of the form

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \tag{2.224}$$

where for simplicity we assume $\boldsymbol{A} \in \mathcal{R}^{M\times N}, \boldsymbol{x} \in \mathcal{R}^{N\times 1}$ and $\boldsymbol{b} = \mathcal{R}^{M\times 1}$. In some problems, this system is *overdetermined*, meaning that we have more equations than unknowns. Due

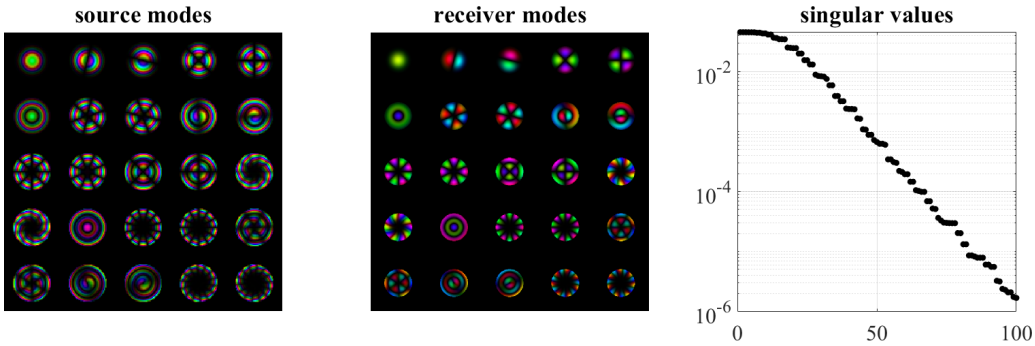


Figure 2.17: opticalSVD.m

to noise there is typically no exact solution. Instead, we are searching for a solution $\boldsymbol{x}$ that minimizes the error

$$\mathcal{L} = \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2, \tag{2.225}$$

which is referred to as the *linear least squares* problem. In section 3.1.2, we show that the gradient of the latter cost function with respect to $\boldsymbol{x}$ is

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}} = \boldsymbol{A}^T (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}). \tag{2.226}$$

Setting this to zero as a necessary condition for optimality, we get the *normal equations*

$$\boldsymbol{A}^T \boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^T \boldsymbol{b}. \tag{2.227}$$

If $\boldsymbol{A}^T\boldsymbol{A}$ is full rank, we obtain the least squares solution

$$\boldsymbol{x} = \left(\boldsymbol{A}^T \boldsymbol{A}\right)^{-1} \boldsymbol{A}^T \boldsymbol{b}. \tag{2.228}$$

Compare this to the earlier result Eq. 2.42, where $\boldsymbol{A}$ was a scalar.

2.9.2 Stacked least squares

A least squares problem with multiple objectives

$$\mathcal{L} = \lambda_1 \|\boldsymbol{A}_1\boldsymbol{x} - \boldsymbol{b}_1\|_2^2 + \lambda_2 \|\boldsymbol{A}_2\boldsymbol{x} - \boldsymbol{b}_2\|_2^2 + \ldots + \lambda_K \|\boldsymbol{A}_K\boldsymbol{x} - \boldsymbol{b}_K\|_2^2 \tag{2.229}$$

$$= \sum_{k=1}^{K} \lambda_k \|\boldsymbol{A}_k\boldsymbol{x} - \boldsymbol{b}_k\|_2^2 \tag{2.230}$$

may be rewritten in the form [6]

$$\mathcal{L} = \left\|\tilde{\boldsymbol{A}}\boldsymbol{x} - \tilde{\boldsymbol{b}}\right\|_2^2, \tag{2.231}$$

where

$$\tilde{\boldsymbol{A}} = \begin{bmatrix} \sqrt{\lambda_1}\boldsymbol{A}_1 \\ \sqrt{\lambda_2}\boldsymbol{A}_2 \\ \dots \\ \sqrt{\lambda_K}\boldsymbol{A}_K \end{bmatrix}, \tilde{\boldsymbol{b}} = \begin{bmatrix} \sqrt{\lambda_1}\boldsymbol{b}_1 \\ \sqrt{\lambda_2}\boldsymbol{b}_2 \\ \dots \\ \sqrt{\lambda_K}\boldsymbol{b}_K \end{bmatrix}. \tag{2.232}$$

Here λ_k $(k = 1, \dots, K)$ is a weighting parameter that controls the relative importance of each objective. Using once more the least squares solution in Eq. 3.52, we get

$$\boldsymbol{x} = \left(\tilde{\boldsymbol{A}}^T \tilde{\boldsymbol{A}}\right)^{-1} \tilde{\boldsymbol{A}}^T \tilde{\boldsymbol{b}} \tag{2.233}$$

$$= \left(\lambda_1 \boldsymbol{A}_1^T \boldsymbol{A}_1 + \ldots + \lambda_K \boldsymbol{A}_K^T \boldsymbol{A}_K\right)^{-1} \left(\lambda_1 \boldsymbol{A}_1^T \boldsymbol{b}_1 + \ldots + \lambda_K \boldsymbol{A}_K^T \boldsymbol{b}_K\right) \tag{2.234}$$

$$= \left(\sum_{k=1}^{K} \lambda_k \boldsymbol{A}_k^T \boldsymbol{A}_k\right)^{-1} \left(\sum_{k=1}^{K} \lambda_k \boldsymbol{A}_k^T \boldsymbol{b}_k\right). \tag{2.235}$$

2.9.3 Ridge regression

Suppose now the system

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{2.236}$$

is *underdetermined*, meaning that there are less equations (rows) than unknowns (entries in $\boldsymbol{x}$). In this case, it oftentimes happens that there are an infinite number of solutions, implying that the least squares problem

$$\mathcal{L} = \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 \tag{2.237}$$

is non-unique. We nevertheless may attempt to find a solution that penalizes entries in $\boldsymbol{x}$, for instance by *LASSO regression*[17]

$$\mathcal{L} = \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \lambda \|\boldsymbol{x}\|_1 \tag{2.238}$$

or *ridge regression*[18]

$$\mathcal{L} = \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \lambda \|\boldsymbol{x}\|_2^2. \tag{2.239}$$

For now we are in a position to solve the latter problem. Notice that this is an instance of stacked least squares with two objectives. Identifying

$$\boldsymbol{A}_1 = \boldsymbol{A},\ \boldsymbol{b}_1 = \boldsymbol{b},\ \lambda_1 = 1,\ \boldsymbol{A}_2 = \boldsymbol{I},\ \boldsymbol{b}_2 = 0,\ \lambda_2 = \lambda, \tag{2.240}$$

we use Eq. 2.235 to get the solution

$$\boldsymbol{x} = \left(\boldsymbol{A}^T \boldsymbol{A} + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{A}^T \boldsymbol{b}. \tag{2.241}$$

Ridge regression is typically used, when the components of $\boldsymbol{x}$ are correlated.

[17] This problem is solved later in subsection 3.10.2, when we have the necessary mathematical tools.
[18] This is also referred to as *Tikhonov regression.*

Example: polynomial curve fitting

Suppose we are given data from a function

$$y(x) = \exp(x) \tag{2.242}$$

subject to Gaussian noise. The exact mathematical form of $y(x)$ is unknown to us. We attempt to fit the data by a polynomial curve of the form

$$y_{\text{fit}}(x) = \sum_{k=0}^{K} c_k x^k. \tag{2.243}$$

Our goal is to estimate the coefficient $\boldsymbol{c} = (c_1, c_2, \ldots, c_K)^T$ from the available data $\boldsymbol{d} = (d_1, d_2, \ldots, d_N)^T$ measured at $(x_1, x_2, \ldots, x_N)^T$. According to our polynomial model, we have

$$d_i = \sum_{k=0}^{K} c_k x_i^k \tag{2.244}$$

for $i = 1, 2, \ldots, N$. In vector form this may be written as

$$\boldsymbol{d} = \boldsymbol{X}\boldsymbol{c}, \tag{2.245}$$

where

$$\boldsymbol{X} = \begin{bmatrix} x_1^0 & x_1^1 & \ldots & x_1^K \\ x_2^0 & x_2^1 & \ldots & x_2^K \\ \ldots & \ldots & \ldots & \ldots \\ x_N^0 & x_N^1 & \ldots & x_N^K \end{bmatrix} \in \mathcal{R}^{N\times(K+1)} \tag{2.246}$$

is the so-called *Vandermonde* matrix; the superscripts indicate powers and the subscripts distinguish data points. Due to noise we do not expect an exact fit. Instead we aim to minimize the L2 norm

$$\mathcal{L}_{\text{lsq}} = \|\boldsymbol{X}\boldsymbol{c} - \boldsymbol{d}\|_2^2. \tag{2.247}$$

If we have reason to assume that the coefficients in $\boldsymbol{c}$ are correlated, we may alternatively aim to minimize

$$\mathcal{L}_{\text{ridge}} = \|\boldsymbol{X}\boldsymbol{c} - \boldsymbol{d}\|_2^2 + \lambda \|\boldsymbol{c}\|_2^2. \tag{2.248}$$

The code ridgeRegression.m compares polynomial fits obtained from least squares and ridge regression. The results are shown in Fig. 2.18. It is seen that in this example a naive least squares fit (blue line) exhibits oscillatory behavior, as it attempts to follow the data, as much as permitted by the model. In contrast, ridge regression does not blindly attempt to follow and fit the noisy data (black points). Instead only the general trend is captured by the fit, resulting in a smoother appearance. A reason that this works out well in this example is that the ground truth (dashed line) has a Taylor expansion with only positive and monotonically decreasing coefficients,

$$y(x) = \exp(x) = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots \tag{2.249}$$

The coefficients are indeed correlated, an implicit *prior* that ridge regression takes advantage of when fitting the data. We stress that ridge regression is not a kind of magic trick

that saves our curve fits under general circumstances. Applying the regularization on the L2 norm of the fit coefficients $\boldsymbol{c}$ amounts to *a priori* knowledge about the underlying model. By selecting a ridge regression model we assume correlation between the components of the solution vector $\boldsymbol{c}$, which may not always be justified. Another difficulty lies in an optimal choice of the tuning parameter λ, which is a priori unknown. A poor choice of λ can result in a much worse curve fit than shown here. In the limit of $\lambda \to 0$, ridge regression produces the same results as least squares, while for $\lambda \to \infty$ the coefficients in $\boldsymbol{c}$ vanish. In Fig. 2.19 we show the expansion coefficients estimated from least squares (blue) as compared to ridge regression (red). The right bar plot shows the ground truth expansion coefficients (black).

```
N = 50;                      % number of data points
rng(1)
x = linspace(0,1,N)';        % data grid
xd = linspace(0,1,5*N)';     % dense grid
y = exp(x);                  % ground truth
data = y + 1/2*randn(N,1);   % noisy observation

figure(1)
plot(x,data,'ko','MarkerFaceColor','k')
axis square
%%
p = 10;
X = zeros(size(x,1),p+1);
Xd = zeros(size(xd,1),p+1);
for k = 0:p
    X(:,k+1) = x.^k;
    Xd(:,k+1) = xd.^k;
end
% least squares
c_lsq = (X'*X)\(X'*data);
% ridge regression
lambda = 1;
c_ridge = (X'*X + lambda*eye(p+1,p+1))\(X'*data);

figure(1), clf
hold on
plot(xd,Xd*c_lsq,'-b')
plot(xd,Xd*c_ridge,'-r')
plot(x,data,'ko','MarkerFaceColor','k')
plot(x,y,'k--')
hold off
axis square
h = legend('least squares','ridge','data','ground truth');
h.Location = 'SoutheastOutside';
axis([0 1 -inf inf])

```

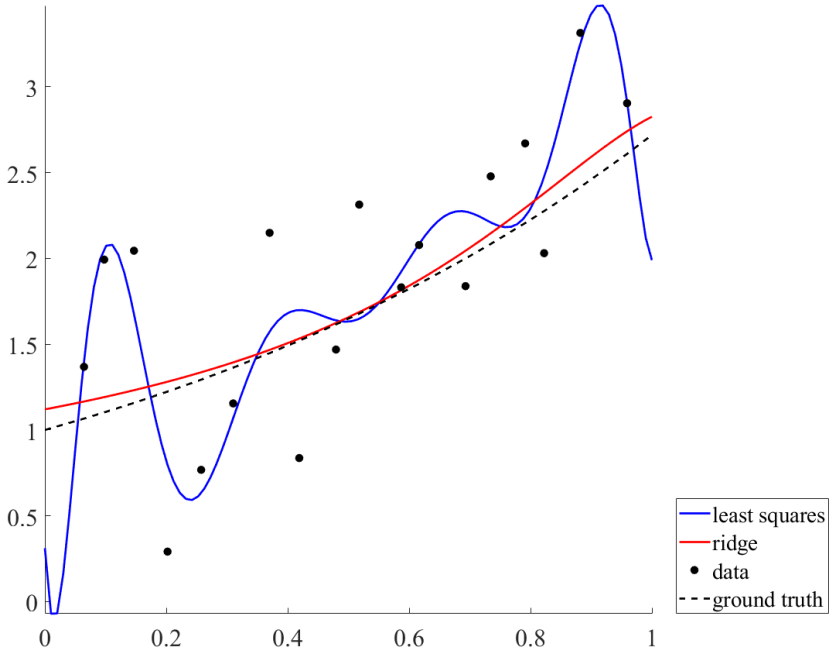


Figure 2.18: Plot generated from ridgeRegression.m

```
figure(2),clf
subplot(1,3,1)
bar(0:length(c_lsq)-1,c_lsq,'FaceColor','b')
xlabel('c_{k;lsq}'), axis square, title('LSQ')

subplot(1,3,2)
bar(0:length(c_ridge)-1,c_ridge,'FaceColor','r')
xlabel('c_{k;ridge}'), axis square, title('ridge')

subplot(1,3,3)
bar(0:length(c_ridge)-1,1./gamma(1:length(c_ridge)),'FaceColor',
    'k')
axis square, title('ground truth')
```

Listing 2.16: ridgeRegression.m (Matlab)

As a final note, we can map out the coefficients for ridge regression as a function of the regularization parameter λ. This is referred to as a *trade-off curve* as λ controls the degree to which we balance our model between objectives - the result being a compromise or trade-off. An example is shown in Fig. 2.20, where we show the estimated ridge regression coefficients (c_k, solid lines) versus the four[19] ground truth expansion coefficients (gt_k,

[19] Notice that both the first and the second ground truth coefficient have the same value 1.

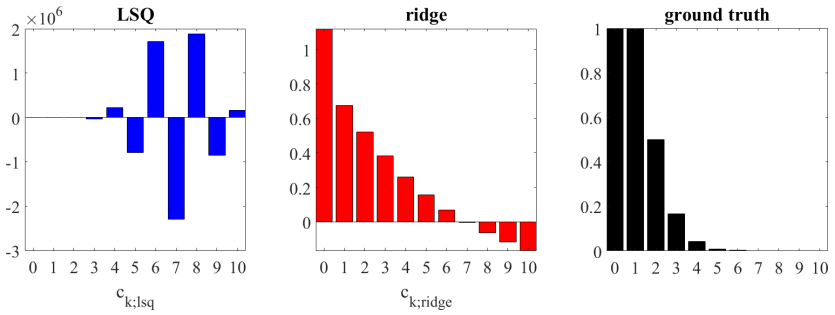


Figure 2.19: Comparison of expansion coefficients from least squares (left) and ridge regression (middle, $\lambda = 1$, $K = 10$). The right panel shows the ground truth Taylor expansion coefficients of $\exp(x)$.

dashed lines)[20]. From the left panel, it appears that somewhere in the range $\lambda \in [10^0, 10^1]$ there is an optimal choice for lambda, as judged from the distance to the ground truth. For example, in the aforementioned range both c_0 and c_1 are close to 1, the value of the first two terms in the ground truth Taylor expansion. In the right panel, we plot the ground truth cost (violet) $\|\boldsymbol{c} - \boldsymbol{gt}\|_2^2$.

So far we have assumed that we know the ground truth. What can we judge the quality of a curve fit in the absence of the ground truth, which is the typical case in practice? In the right panel of Fig. 2.20 we also show the L2 cost (green) $\|\boldsymbol{X}_t\boldsymbol{c} - \boldsymbol{d}_t\|_2^2$ evaluated on test data $\boldsymbol{d}_t$ *that was not used to train the model.* This is referred to as *cross-validation*. Cross-validation can be used to tune *hyperparameters*, such as the regularizer λ in the ridge regression model. While typically the ground truth cost function is not available - where the true solution is not known -, cross-validation can easily be achieved by withholding a fraction of data and using it for model selection. The full code (ridgeRegressionTradeOff.m) to produce Fig. 2.20 is available on GitHub.

2.9.4 Least squares, SVD, and pseudoinverse

We can gain additional insight into linear inverse problems

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \tag{2.250}$$

where $\boldsymbol{A} \in \mathcal{C}^{M\times N}$, by expanding $\boldsymbol{A}$ into an exact SVD,

$$\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\dagger\boldsymbol{x} = \boldsymbol{b}. \tag{2.251}$$

If the system is square ($M = N$) and full rank, then the solution to Eq. 2.250 is given by

$$\boldsymbol{x} = \boldsymbol{V}\boldsymbol{S}^{-1}\boldsymbol{U}^\dagger\boldsymbol{b}. \tag{2.252}$$

[20] Notice that for the Taylor expansion of a simple exponential, we have $gt_0 = gt_1 = 1$. For this reason we did not explicitly plot gt_0.

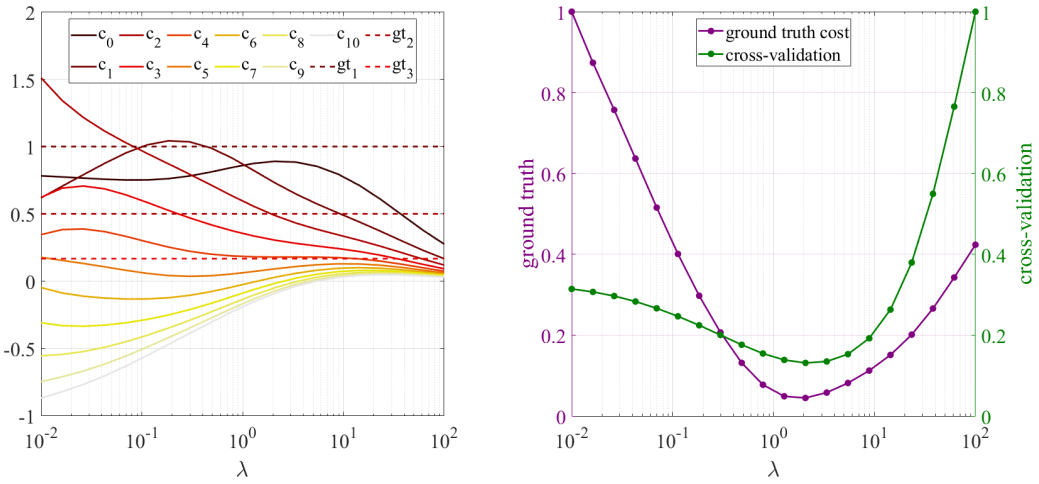


Figure 2.20: (Left) Trade-off curve: polynomial expansion coefficients estimated from ridge regression as a function of the regularization parameter λ. (Right) The purple curve shows the ground truth error between the coefficient estimated in ridge regression and the true Taylor expansion of $\exp(x)$. The green curve shows the evaluation of the model as a function of the regularization parameter λ. The cross-validation data (green) was not used to estimate or train the model.

In contrast, if $\boldsymbol{A}$ is not full rank, some of the singular values along the diagonal of $\boldsymbol{S}$ vanish and

$$\boldsymbol{S}^{-1} = \begin{bmatrix} \frac{1}{\sigma_1} & 0 & \dots & 0 \\ 0 & \frac{1}{\sigma_2} & \dots & 0 \\ \dots & \dots & \dots & 0 \\ 0 & 0 & 0 & \frac{1}{\sigma_N} \end{bmatrix} \tag{2.253}$$

is not defined, as this operation would cause division by zero. Moreover, for the aforementioned cases of overdetermined and underdetermined systems, the matrix $\boldsymbol{S}$ has a rectangular shape

$$\boldsymbol{S} = \begin{bmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 \\ \dots & \dots & \dots & 0 \\ 0 & 0 & 0 & \sigma_N \\ \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 \end{bmatrix} \in \mathcal{R}^{M \times N} \, (M > N) \tag{2.254}$$

or

$$\boldsymbol{S} = \begin{bmatrix} \sigma_1 & 0 & \dots & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 & \dots & 0 \\ \dots & \dots & \dots & 0 & \dots & 0 \\ 0 & 0 & 0 & \sigma_M & \dots & 0 \end{bmatrix} \in \mathcal{R}^{M \times N} \, (M < N). \tag{2.255}$$

But the inverse matrix $\boldsymbol{S}^{-1}$ is only defined for square matrices.

However, the truncated SVD

$$\boldsymbol{A} = \underbrace{\tilde{\boldsymbol{U}}_K}_{M\times K}\underbrace{\tilde{\boldsymbol{S}}_K}_{K\times K}\underbrace{\tilde{\boldsymbol{V}}_K^\dagger}_{K\times N}$$

is defined for each of the aforementioned cases, which opens up a path towards a generalized notion of matrix inversion, namely

$$\boldsymbol{x} = \boldsymbol{A}^+\boldsymbol{b} = \tilde{\boldsymbol{V}}_K\tilde{\boldsymbol{S}}_K^{-1}\tilde{\boldsymbol{U}}_K^\dagger\boldsymbol{b}, \tag{2.256}$$

provided that we select K such that all all singular values σ_k for $k = 1,\dots,K$ are nonzero. This approach simply sets to zero what would otherwise diverge and go to infinity in the calculation of $\boldsymbol{S}^{-1}$. The matrix

$$\boldsymbol{A}^+ = \tilde{\boldsymbol{V}}_K\tilde{\boldsymbol{S}}_K^{-1}\tilde{\boldsymbol{U}}_K^\dagger \in \mathcal{C}^{N\times M} \tag{2.257}$$

is called the *Moore-Penrose pseudoinverse*. Here

$$\left(\tilde{\boldsymbol{S}}_K^{-1}\right)_{k,k} = \begin{cases} \frac{1}{\sigma_k} & ,\text{if } \sigma_k \neq 0 \\ 0 & ,\text{else} \end{cases}. \tag{2.258}$$

First note that

$$\boldsymbol{A} = \underbrace{\boldsymbol{U}}_{M\times M}\underbrace{\boldsymbol{S}}_{M\times N}\underbrace{\boldsymbol{V}^\dagger}_{N\times N} \tag{2.259}$$

$$= \left[\tilde{\boldsymbol{U}}_K, \boldsymbol{u}_{K+1},\dots,\boldsymbol{u}_M\right]\begin{bmatrix} \tilde{\boldsymbol{S}}_K & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0}\end{bmatrix}\left[\tilde{\boldsymbol{V}}_K, \boldsymbol{v}_{K+1},\dots,\boldsymbol{v}_N\right]^\dagger \tag{2.260}$$

$$= \left[\tilde{\boldsymbol{U}}_K\right]\begin{bmatrix} \tilde{\boldsymbol{S}}_K & \boldsymbol{0}\end{bmatrix}\left[\tilde{\boldsymbol{V}}_K, \boldsymbol{v}_{K+1},\dots,\boldsymbol{v}_N\right]^\dagger \tag{2.261}$$

$$= \underbrace{\tilde{\boldsymbol{U}}_K}_{M\times K}\underbrace{\tilde{\boldsymbol{S}}_K}_{K\times K}\underbrace{\tilde{\boldsymbol{V}}_K^\dagger}_{K\times N}. \tag{2.262}$$

Then by definition,

$$\boldsymbol{A}^+\boldsymbol{A} = \tilde{\boldsymbol{V}}_K\tilde{\boldsymbol{S}}_K^{-1}\tilde{\boldsymbol{U}}_K^\dagger\tilde{\boldsymbol{U}}_K\tilde{\boldsymbol{S}}_K\tilde{\boldsymbol{V}}_K^\dagger = \boldsymbol{I}_{N\times N}. \tag{2.263}$$

We now distinguish three cases, which do not serve as a rigorous proof but illustrate the properties of the pseudoinverse.

Case 1: $M > N$

In this case we are dealing with an overdetermined system and $\boldsymbol{A}$ is a tall matrix with more rows than columns. The vector $\boldsymbol{b}$ may or may not be in the column space of $\boldsymbol{A}$. If $\boldsymbol{b}$ is in the column space of $\boldsymbol{A}$, then $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ holds with equality leading to the exact solution[21]

$$\boldsymbol{x} = \boldsymbol{I}_{N\times N}\boldsymbol{x} = \boldsymbol{A}^+\underbrace{\boldsymbol{A}\boldsymbol{x}}_{=\boldsymbol{b}} = \boldsymbol{A}^+\boldsymbol{b}. \tag{2.264}$$

[21] This solution may or may not be unique, which is further elaborated in case 3.

In contrast, if $\boldsymbol{b}$ is not in the column space of $\boldsymbol{A}$, then $\boldsymbol{Ax} = \boldsymbol{b}$ has no exact solution. However, we may still attempt to find an approximate solution. We have previously discussed the least square solution given by Eq. 3.52, which derives from the normal equations (Eq. 3.51)

$$\boldsymbol{A}^\dagger \boldsymbol{Ax} = \boldsymbol{A}^\dagger \boldsymbol{b} \tag{2.265}$$

$$\tilde{\boldsymbol{V}}_K \tilde{\boldsymbol{S}}_K \tilde{\boldsymbol{U}}_K^\dagger \tilde{\boldsymbol{U}}_K \tilde{\boldsymbol{S}}_K \tilde{\boldsymbol{V}}_K \boldsymbol{x} = \tilde{\boldsymbol{V}}_K \tilde{\boldsymbol{S}}_K \tilde{\boldsymbol{U}}_K^\dagger \boldsymbol{b} \tag{2.266}$$

$$\tilde{\boldsymbol{V}}_K \tilde{\boldsymbol{S}}_K^2 \tilde{\boldsymbol{V}}_K^\dagger \boldsymbol{x} = \tilde{\boldsymbol{V}}_K \tilde{\boldsymbol{S}}_K \boldsymbol{U}^\dagger \boldsymbol{b} \tag{2.267}$$

$$\boldsymbol{x} = \tilde{\boldsymbol{V}}_K \tilde{\boldsymbol{S}}_K^{-1} \tilde{\boldsymbol{U}}_K^\dagger \boldsymbol{b}. \tag{2.268}$$

Case 2: $M = N$

In this case we are considering a square system. If $\boldsymbol{A}$ is full rank, then we may use the full SVD to get the solution $\boldsymbol{x} = \boldsymbol{V}\boldsymbol{S}^{-1}\boldsymbol{U}^\dagger \boldsymbol{b}$. If $\boldsymbol{A}$ is singular, we have to truncate its SVD, as detailed in the next case.

Case 3: $M < N$

Now we are dealing with an underdetermined system and $\boldsymbol{A}$ is a fat matrix, which has less rows than columns. Again, the vector $\boldsymbol{b}$ may or may not be in the column space of $\boldsymbol{A}$. If $\boldsymbol{b}$ is in the column space of $\boldsymbol{A}$, the solution is either unique or non-unique. In both of theses cases a solution is provided by

$$\boldsymbol{x} = \boldsymbol{I}_{N\times N}\boldsymbol{x} = \boldsymbol{A}^+ \underbrace{\boldsymbol{Ax}}_{=\boldsymbol{b}} = \boldsymbol{A}^+ \boldsymbol{b}. \tag{2.269}$$

Notice that if $\boldsymbol{x}_n$ is in the nullspace of $\boldsymbol{A}$, then $\boldsymbol{Ax}_n = \tilde{\boldsymbol{U}}_K \tilde{\boldsymbol{S}}_K \tilde{\boldsymbol{V}}_K^\dagger \boldsymbol{x}_n = 0$, from which it follows that $\boldsymbol{x}_n$ is orthogonal to each of the rows of $\tilde{\boldsymbol{V}}_K^\dagger$. This implies that

$$\boldsymbol{x}^\dagger \boldsymbol{x}_n = \boldsymbol{b}^\dagger \left(\boldsymbol{A}^+\right)^\dagger \boldsymbol{x}_n = \boldsymbol{b}^\dagger \tilde{\boldsymbol{U}}_K \tilde{\boldsymbol{S}}_K^{-1} \tilde{\boldsymbol{V}}_K^\dagger \boldsymbol{x}_n = 0. \tag{2.270}$$

Hence the pseudoinverse solution $\boldsymbol{x} = \boldsymbol{A}^+\boldsymbol{b}$ is orthogonal to any vector $\boldsymbol{x}_n$ in the nullspace of $\boldsymbol{A}$. We also note that any $\boldsymbol{u} = \boldsymbol{x} + \boldsymbol{x}_n$ is a solution to the original problem since

$$\boldsymbol{A}\left(\boldsymbol{x} + \boldsymbol{x}_n\right) = \boldsymbol{A}\left(\boldsymbol{A}^+\boldsymbol{b} + \boldsymbol{x}_n\right) = \boldsymbol{b} + \boldsymbol{0} = \boldsymbol{b}. \tag{2.271}$$

With this we find

$$\|\boldsymbol{Ax} - \boldsymbol{b}\|_2^2 = \|\boldsymbol{Ax} + \boldsymbol{0} - \boldsymbol{b}\|_2^2 \tag{2.272}$$

$$= \|\boldsymbol{Ax} + \boldsymbol{Ax}_n - \boldsymbol{b}\|_2^2 \tag{2.273}$$

$$= \|\boldsymbol{A}\left(\boldsymbol{x} + \boldsymbol{x}_n\right) - \boldsymbol{b}\|_2^2 \tag{2.274}$$

$$= \|\boldsymbol{Au} - \boldsymbol{b}\|_2^2, \tag{2.275}$$

which states that the L2 cost for the pseudoinverse solution $\boldsymbol{x}$ does is invariant under addition of a nullspace component $\boldsymbol{u} = \boldsymbol{x} + \boldsymbol{x}_n$. However, the ridge regression problem

$$\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \lambda \|\boldsymbol{x}\|_2^2 \leq \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \lambda \left(\|\boldsymbol{x}\|_2^2 + \|\boldsymbol{x}_n\|_2^2 \right) \tag{2.276}$$

$$= \|\boldsymbol{A}\boldsymbol{u} - \boldsymbol{b}\|_2^2 + \lambda \|\boldsymbol{u}\|_2^2 \tag{2.277}$$

where the last line follows from the orthogonality of $\boldsymbol{x}$ and $\boldsymbol{x}_n$, which allows to write

$$\|\boldsymbol{u}\|_2^2 = \|\boldsymbol{x} + \boldsymbol{x}_n\|_2^2 = \|\boldsymbol{x}\|_2^2 + \|\boldsymbol{x}_n\|_2^2 . \tag{2.278}$$

This result states that the pseudoinverse solution $\boldsymbol{x}$ minimizes the ridge regression cost in case of $\boldsymbol{A}$ being singular. Here we can not add a nullspace component without increasing the cost[22]. Also notice that the least square solution to the ridge regression cost is given by (compare Eq. 2.241)

$$\boldsymbol{x} = \left(\boldsymbol{A}^\dagger \boldsymbol{A} + \lambda \boldsymbol{I} \right)^{-1} \boldsymbol{A}^\dagger \boldsymbol{b}, \tag{2.279}$$

which in the limit $\lambda \to 0$ is equal to

$$\boldsymbol{x} = \tilde{\boldsymbol{V}}_K \tilde{\boldsymbol{S}}_K^{-1} \tilde{\boldsymbol{U}}_K^\dagger \boldsymbol{b}. \tag{2.280}$$

Finally, in the case that $\boldsymbol{b}$ is not in the column space of $\boldsymbol{A}$, using the same derivation as in case 1 leads to the least square solution

$$\boldsymbol{x} = \tilde{\boldsymbol{V}}_K \tilde{\boldsymbol{S}}_K^{-1} \tilde{\boldsymbol{U}}_K^\dagger \boldsymbol{b}. \tag{2.281}$$

Summary

We see that the behavior of the pseudoinverse is complex and care has to be taken as to when and how to make use of it. We summarize the previous observations as follows:

- The pseudoinverse $\boldsymbol{A}^+$ is defined by $\boldsymbol{A}^+ = \left(\boldsymbol{A}^\dagger \boldsymbol{A} \right)^{-1} \boldsymbol{A}^\dagger = \tilde{\boldsymbol{V}}_K \tilde{\boldsymbol{S}}_K^{-1} \tilde{\boldsymbol{U}}_K^\dagger$.
- $\boldsymbol{x} = \boldsymbol{A}^+ \boldsymbol{b}$ solves $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ exactly in case that there is a unique solution.
- In case that there is no solution to $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, the pseudoinverse $\boldsymbol{x} = \boldsymbol{A}^+ \boldsymbol{b}$ minimizes the least cost $\mathcal{L} = \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2$.
- If there infinitely many solutions, the pseudoinverse $\boldsymbol{x} = \boldsymbol{A}^+ \boldsymbol{b}$ minimizes the ridge regression cost $\mathcal{L} = \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \lambda \|\boldsymbol{x}\|_2^2$ in the limit $\lambda \to 0$.

A practical difficulty is how to decide when a matrix is singular. The finite floating point accuracy in numerical computations may require to set λ to a finite value. Matlab provides the command `pinv(A, delta)`, which allows to manually control the value of λ for the ridge regression cost.

[22] The same statement hold in cases 1 and 2 when the columns of $\boldsymbol{A}$ are linearly dependent.

2.10 Matrix Norms

2.10.1 Trace

The trace of a square matrix $\boldsymbol{A}$ is defined as

$$\operatorname{tr}(\boldsymbol{A}) = \sum_i A_{i,i}, \tag{2.282}$$

i.e., $\operatorname{tr}(\boldsymbol{A})$ is the sum over all diagonal elements of $\boldsymbol{A}$. It is linear with respect to summation (α, β constant scalars),

$$\operatorname{tr}(\alpha \cdot \boldsymbol{A} + \beta \cdot \boldsymbol{B}) = \alpha \cdot \operatorname{tr}(\boldsymbol{A}) + \beta \cdot \operatorname{tr}(\boldsymbol{B}). \tag{2.283}$$

Moreover, the trace of $\boldsymbol{A}$ and the trace of $\boldsymbol{A}^T$ are the same,

$$\operatorname{tr}(\boldsymbol{A}) = \sum_i A_{i,i} = \sum_i A^T_{i,i} = \operatorname{tr}\left(\boldsymbol{A}^T\right). \tag{2.284}$$

The trace of a product can be changed cyclically. Recall that (compare Eq. 2.108)

$$(\boldsymbol{A} \cdot \boldsymbol{B})_{i,j} = \sum_k A_{i,k} \cdot B_{k,j}. \tag{2.285}$$

Therefore

$$\operatorname{tr}(\boldsymbol{A} \cdot \boldsymbol{B}) = \sum_i (\boldsymbol{A} \cdot \boldsymbol{B})_{i,i} = \sum_i \sum_k A_{i,k} \cdot B_{k,i}, \tag{2.286}$$

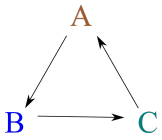
and

$$\operatorname{tr}(\boldsymbol{B} \cdot \boldsymbol{A}) = \sum_i (\boldsymbol{B} \cdot \boldsymbol{A})_{i,i} = \sum_i \sum_k B_{i,k} \cdot A_{k,i} \tag{2.287}$$

$$= \sum_k \sum_i A_{k,i} \cdot B_{i,k} = \operatorname{tr}(\boldsymbol{A} \cdot \boldsymbol{B}). \tag{2.288}$$

By induction, we can extend this results to products of arbitrary size, for example

$$\operatorname{tr}(\boldsymbol{A} \cdot \boldsymbol{B} \cdot \boldsymbol{C}) = \operatorname{tr}(\boldsymbol{C} \cdot \boldsymbol{A} \cdot \boldsymbol{B}) = \operatorname{tr}(\boldsymbol{B} \cdot \boldsymbol{C} \cdot \boldsymbol{A}). \tag{2.289}$$



2.10.2 Frobenius norm

There are several matrix norms relevant to the field of optimization. The first matrix norm discussed here is the Frobenius norm

$$\|\boldsymbol{A}\|_F^2 = \operatorname{tr}\left(\boldsymbol{A}^H \boldsymbol{A}\right) = \sum_i \sum_j A^H_{i,j} \cdot A_{j,i} \tag{2.290}$$

$$= \sum_i \sum_j A^*_{j,i} \cdot A_{j,i} = \sum_i \sum_j |A_{j,i}|^2 = \sum_i \sum_j |A_{i,j}|^2. \tag{2.291}$$

Thus the Frobenius norm is defined via the sum of squares over all matrix elements. For later problems we are going to need the Frobenius norm of the product of two (or more) matrices. Define $\boldsymbol{C} = \boldsymbol{A} \cdot \boldsymbol{B}$, then

$$\|\boldsymbol{A} \cdot \boldsymbol{B}\|_F^2 = \|\boldsymbol{C}\|_F^2 = \sum_i \sum_j |C_{i,j}|^2 = \sum_i \sum_j \left| \sum_k A_{i,k} \cdot B_{k,j} \right|^2 \tag{2.292}$$

$$= \sum_i \sum_j \left(\sum_k A_{i,k} \cdot B_{k,j} \right)^* \left(\sum_l A_{i,l} \cdot B_{l,j} \right) \tag{2.293}$$

$$= \sum_i \sum_j \sum_k A_{i,k}^* \cdot B_{k,j}^* \left(\sum_l A_{i,l} \cdot B_{l,j} \right) \tag{2.294}$$

$$= \sum_i \sum_j \sum_k \sum_l A_{i,k}^* \cdot B_{k,j}^* \cdot A_{i,l} \cdot B_{l,j} \tag{2.295}$$

where in the first line we have used (see Eq. 2.108)

$$C_{i,j} = (\boldsymbol{A} \cdot \boldsymbol{B})_{i,j} = \sum_k A_{i,k} \cdot B_{k,j}. \tag{2.296}$$

In summary,

$$\|\boldsymbol{A} \cdot \boldsymbol{B}\|_F^2 = \sum_i \sum_j \sum_k \sum_l A_{i,k}^* \cdot B_{k,j}^* \cdot A_{i,l} \cdot B_{l,j}. \tag{2.297}$$

By induction, we can extend this formula to larger products. Not pretty, but useful. Why do we need such expressions? Later on, we will solve least square problems where the unknown $\boldsymbol{X}$ is a matrix. This involves minimizing a cost function of the form

$$\mathcal{L} = \|\boldsymbol{A} \cdot \boldsymbol{X} - \boldsymbol{B}\|_F^2 . \tag{2.298}$$

Next, we consider the Frobenius norm of matrix $\boldsymbol{A}$ subject to a unitary transformation $\boldsymbol{U}$. We know from Eq. 2.291 that the Frobenius norm of a matrix $\boldsymbol{A}$ is defined via the sum of squares over its elements, which may be written in terms of the sum over the L2 norm of its columns

$$\|\boldsymbol{A}\|_F^2 = \sum_i \sum_j |A_{i,j}|^2 = \sum_j \|\boldsymbol{A}_j\|_2^2 . \tag{2.299}$$

We also know know from Eq. 2.120 that unitary transformation preserves the L2 norm. But then

$$\|\boldsymbol{U}\boldsymbol{A}\|_F^2 = \|[\boldsymbol{U}\boldsymbol{A}_1, \boldsymbol{U}\boldsymbol{A}_2, \ldots, \boldsymbol{U}\boldsymbol{A}_N]\|_F^2 \tag{2.300}$$

$$= \sum_j \|\boldsymbol{U}\boldsymbol{A}_j\|_2^2 \tag{2.301}$$

$$= \sum_j \|\boldsymbol{A}_j\|_2^2 \tag{2.302}$$

$$= \|\boldsymbol{A}\|_F^2 . \tag{2.303}$$

An important implication of this is that the Frobenius norm $\boldsymbol{A}$ may be expressed in terms of its singular values

$$\|\boldsymbol{A}\|_F^2 = \left\|\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\dagger\right\|_F^2 \tag{2.304}$$

$$= \left\|\boldsymbol{S}\boldsymbol{V}^\dagger\right\|_F^2 \tag{2.305}$$

$$= \|\boldsymbol{S}\|_F^2 \tag{2.306}$$

$$= \sum_{i=1}^{N} \sigma_i^2 \tag{2.307}$$

or

$$\|\boldsymbol{A}\|_F = \sqrt{\sum_{i=1}^{N} \sigma_i^2}. \tag{2.308}$$

2.11 The discrete Fourier transform

2.11.1 Shift Matrix

Consider the matrix

$$\boldsymbol{P} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}. \tag{2.309}$$

An example for matrix-vector multiplication

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = 1 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + 2 \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + 3 \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} \tag{2.310}$$

illustrates its operation. As can be seen, the effect of the matrix $\boldsymbol{P}$ is to shift each component of the input vector by one row up. The first component is re-inserted at the bottom. The matrix $\boldsymbol{P}$ is a so-called *shift matrix* or *cyclic matrix*. This is a special type of permutation matrix, which has exactly one element equal to 1 and all other elements equal to 0 along each row.

Next, consider powers of $\boldsymbol{P}$,

$$\boldsymbol{P}^2 = \boldsymbol{P}\left[\boldsymbol{e}_3, \boldsymbol{e}_1, \boldsymbol{e}_2\right] \tag{2.311}$$

$$= \left[\boldsymbol{P}\boldsymbol{e}_3, \boldsymbol{P}\boldsymbol{e}_1, \boldsymbol{P}\boldsymbol{e}_2\right] \tag{2.312}$$

$$= \left[\boldsymbol{e}_2, \boldsymbol{e}_3, \boldsymbol{e}_1\right] \tag{2.313}$$

$$= \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \tag{2.314}$$

and

$$\begin{aligned} \boldsymbol{P}^3 &= \boldsymbol{P}\boldsymbol{P}^2 && (2.315)\\ &= \boldsymbol{P}\left[\boldsymbol{e}_2, \boldsymbol{e}_3, \boldsymbol{e}_1\right] && (2.316)\\ &= \left[\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3\right] && (2.317)\\ &= \boldsymbol{I} && (2.318)\\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. && (2.319) \end{aligned}$$

The third power of $\boldsymbol{P}$ is the identity matrix.

More generally, we could have chosen a shift matrix acting on an N-component vector. This matrix would have looked like this

$$\boldsymbol{P} = \begin{bmatrix} & 1 & & & \\ & & 1 & & \\ & & & \cdots & \\ & & & & 1 \\ 1 & & & & \end{bmatrix}, \tag{2.320}$$

where the empty entries are zeros. In this case, $\boldsymbol{P}^N = \boldsymbol{I}$.

2.11.2 Fourier matrix

Now consider the eigenvalue problem

$$\boldsymbol{P}\boldsymbol{x} = \lambda \boldsymbol{x}, \tag{2.321}$$

which for the $N = 3$ example from the previous subsection takes the explicit form

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_2 \\ x_3 \\ x_1 \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \tag{2.322}$$

or

$$\begin{aligned} x_2 &= \lambda x_1 \\ x_3 &= \lambda x_2 \\ x_1 &= \lambda x_3 \end{aligned}. \tag{2.323}$$

We can back-substitute each of these equations to get

$$x_1 = \lambda x_3 = \lambda^2 x_2 = \lambda^3 x_1. \tag{2.324}$$

Thus

$$\lambda^3 = 1, \tag{2.325}$$

which has three complex roots

$$\lambda_0 = 1,\ \lambda_1 = \exp\left[i2\pi/3\right],\ \lambda_2 = \exp\left[i4\pi/3\right]. \tag{2.326}$$

Now we can solve

$$(\boldsymbol{P} - \lambda_k \boldsymbol{I})\, \boldsymbol{x}_k = 0 \tag{2.327}$$

for each $k = 0, 1, 2$. For $k = 0$

$$\begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \\ 1 & 0 & -1 \end{bmatrix} \boldsymbol{x}_0 = 0, \tag{2.328}$$

a solution is given by

$$\boldsymbol{x}_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \tag{2.329}$$

since

$$\begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = 1 \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix} + 1 \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix} + 1 \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}. \tag{2.330}$$

For $k = 1$

$$\begin{bmatrix} -\exp[i2\pi/3] & 0 & 0 \\ 0 & -\exp[i2\pi/3] & 1 \\ 1 & 0 & -\exp[i2\pi/3] \end{bmatrix} \boldsymbol{x}_1 = 0 \tag{2.331}$$

we find

$$\boldsymbol{x}_1 = \begin{bmatrix} 1 \\ \exp[i2\pi/3] \\ \exp[i4\pi/3] \end{bmatrix} \tag{2.332}$$

since

$$\begin{bmatrix} -\exp[i2\pi/3] & 1 & 0 \\ 0 & -\exp[i2\pi/3] & 1 \\ 1 & 0 & -\exp[i2\pi/3] \end{bmatrix} \begin{bmatrix} 1 \\ \exp[i2\pi/3] \\ \exp[i4\pi/3] \end{bmatrix} = 0. \tag{2.333}$$

The reader may verify that for $k = 2$ we get Exercise

$$\boldsymbol{x}_2 = \begin{bmatrix} 1 \\ (\exp[i2\pi/3])^2 \\ (\exp[i4\pi/3])^4 \end{bmatrix}. \tag{2.334}$$

In general, for an $N \times N$ shift matrix $\boldsymbol{P}$ we have the matrix of eigenvectors

$$\boldsymbol{X} = [\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{N-1}] = \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & w^1 & w^2 & \ldots & w^{N-1} \\ 1 & w^2 & w^4 & \ldots & w^{2(N-1)} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & w^{N-1} & w^{2(N-1)} & \ldots & w^{(N-1)(N-1)} \end{bmatrix}, \tag{2.335}$$

where

$$w = \exp(i2\pi/N). \tag{2.336}$$

Example: $N = 4$

For $N = 4$ we have $w = \exp(i\pi/2) = i$ and

$$\boldsymbol{X} = \begin{bmatrix} 1 & i^0 & i^0 & i^0 \\ 1 & i^1 & i^2 & i^3 \\ 1 & i^2 & i^4 & i^6 \\ 1 & i^3 & i^6 & i^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix}. \tag{2.337}$$

Notice that

$$\boldsymbol{X}\boldsymbol{X}^\dagger = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix} = 4 \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = 4\boldsymbol{I}.$$

For general N, we get

$$\boldsymbol{X}\boldsymbol{X}^\dagger = N\boldsymbol{I}. \tag{2.338}$$

With this we have

$$\boldsymbol{P}\boldsymbol{X} = [\lambda_0\boldsymbol{x}_0, \lambda_1\boldsymbol{x}_1, \ldots, \lambda_{N-1}\boldsymbol{x}_{N-1}] = \boldsymbol{X}\boldsymbol{\Lambda} \tag{2.339}$$

or

$$\boldsymbol{P} = \frac{1}{N}\boldsymbol{X}\boldsymbol{\Lambda}\boldsymbol{X}^\dagger, \tag{2.340}$$

where

$$\boldsymbol{\Lambda} = \begin{bmatrix} 1 & 0 & \ldots & 0 \\ 0 & w & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & w^{N-1} \end{bmatrix}. \tag{2.341}$$

The shift matrix $\boldsymbol{P}$ is diagonalized by the matrix $\boldsymbol{X}$.

In Matlab, the matrix $\boldsymbol{X}$ is closely related to the `fft` command,

$$\text{fft} = \boldsymbol{X}^\dagger = \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & w^1 & w^2 & \ldots & w^{N-1} \\ 1 & w^2 & w^4 & \ldots & w^{2(N-1)} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & w^{N-1} & w^{2(N-1)} & \ldots & w^{(N-1)(N-1)} \end{bmatrix}^\dagger \equiv \boldsymbol{F}. \tag{2.342}$$

Here we refer to $\boldsymbol{F}$ as the *Fourier matrix*. As a consequence of the orthogonality relation in Eq. 2.338

$$\boldsymbol{F}^{-1} = \boldsymbol{X}/N = \frac{1}{N} \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & w^1 & w^2 & \ldots & w^{N-1} \\ 1 & w^2 & w^4 & \ldots & w^{2(N-1)} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & w^{N-1} & w^{2(N-1)} & \ldots & w^{(N-1)(N-1)} \end{bmatrix} \equiv \text{ifft}. \tag{2.343}$$

To illustrate this, the following Matlab code returns $\boldsymbol{X}$ for $N = 4$.

```
N=4;
disp( N*ifft( eye(N) ) )
```

Listing 2.17: generateFourierMatrix.m (Matlab)

These code lines take advantage of the fact that the unit vectors along the columns of the identity matrix each select a single column of the inverse Fourier transform matrix.

With the definition of the Fourier matrix in Eq. 2.342 we can get an explicit formula for the components of the Fourier transform. Suppose that $\boldsymbol{v} = \boldsymbol{F}\boldsymbol{u}$, then from the definition of matrix vector multiplication

$$\boldsymbol{v} = \boldsymbol{F}\boldsymbol{u} = \sum_{n=0}^{N-1} \boldsymbol{F}_n \boldsymbol{u}_n, \tag{2.344}$$

where $\boldsymbol{F}_n$ is the nth column of the matrix $\boldsymbol{F}$ and $\boldsymbol{u}_n$ is the nth component of the vector $\boldsymbol{u}$. Then the mth component of $\boldsymbol{v}$ is given by

$$\boldsymbol{v}_m = (\boldsymbol{F}\boldsymbol{u})_m = \sum_{n=0}^{N-1} \boldsymbol{F}_{m,n} \boldsymbol{u}_n. \tag{2.345}$$

From Eq. 2.342 we have $\boldsymbol{F}_{m,n} = \boldsymbol{X}^*_{n,m}$, so

$$\boldsymbol{v}_m = \sum_{n=0}^{N-1} \boldsymbol{X}^*_{n,m} \boldsymbol{u}_n = \sum_{n=0}^{N-1} \left(w^{m*n}\right)^* \boldsymbol{u}_n = \sum_{n=0}^{N-1} \exp\left(-i2\pi \frac{m \cdot n}{N}\right) \boldsymbol{u}_n. \tag{2.346}$$

We would also like to emphasize that Eq. 2.338 implies *orthogonality* but not *orthonormality* between the columns of $\boldsymbol{X}$ and $\boldsymbol{X}^\dagger$. This is a somewhat unpleasant but important detail that results in the asymmetry in the definition of the Fourier matrix $\boldsymbol{F}$ and its inverse $\boldsymbol{F}^{-1}$, caused by the presence of the factor N^{-1} in the latter. In particular, this implies that the L2 norm is not conserved under Fourier transformation, as defined here. Suppose $\boldsymbol{v} = \boldsymbol{F}\boldsymbol{u}$, then

$$\|\boldsymbol{v}\|_2^2 = \boldsymbol{v}^\dagger \boldsymbol{v} = \boldsymbol{u}^\dagger \boldsymbol{F}^\dagger \boldsymbol{F} \boldsymbol{u} = \boldsymbol{u}^\dagger \underbrace{\boldsymbol{X}\boldsymbol{X}^\dagger}_{N\boldsymbol{I}} \boldsymbol{u} = N \|\boldsymbol{u}\|_2^2 . \tag{2.347}$$

This is a problem. In later sections, we will use the Fourier transform for wave propagation problems. However, there we need a Fourier transform that preserves the total energy in a wave field. To this end, we can define a modified Fourier transform $\mathcal{F} = \boldsymbol{F}/\sqrt{N}$ that preserves the L2 norm. Suppose $\boldsymbol{v} = \mathcal{F}\boldsymbol{u}$,then the previous argument changes into

$$\|\boldsymbol{v}\|_2^2 = \boldsymbol{v}^\dagger \boldsymbol{v} = \boldsymbol{u}^\dagger \mathcal{F}^\dagger \mathcal{F} \boldsymbol{u} = \boldsymbol{u}^\dagger \frac{1}{N} \underbrace{\boldsymbol{F}^\dagger \boldsymbol{F}}_{N\boldsymbol{I}} \boldsymbol{u} = \|\boldsymbol{u}\|_2^2 . \tag{2.348}$$

Notice also that now $\mathcal{F}^\dagger \mathcal{F} = \boldsymbol{I}$, so

$$\mathcal{F}^\dagger = \boldsymbol{F}^\dagger/\sqrt{N} = \boldsymbol{X}/\sqrt{N} = N\boldsymbol{F}^{-1}/\sqrt{N} = \sqrt{N}\boldsymbol{F}^{-1} = \sqrt{N} \cdot \text{ifft}$$

is the inverse of $\mathcal{F}$. The following code illustrates normalization of the Fourier matrix.

```
clc % clear command window
% generate signal
x = (1:3)';
% perform (normalized) fft
y = fftL2(x);
% perform inverse (normalized) fft
z = ifftL2(y);
% display L2 squared
disp(x'*x)
disp(y'*y)
disp(z'*z)

function v = fftL2(u)
% assumes input is 1D vector
v = fft(u)/sqrt( numel(u) );
end

function u = ifftL2(v)
% assumes input is 1D vector
u = ifft(v)*sqrt( numel(v) );
end
```

Listing 2.18: fftL2test.m (Matlab)

2.11.3 The Fast Fourier Transform (FFT)

The Matlab command `fft` is shorthand for the Fast Fourier Transform (FFT). The FFT is an algorithm to efficiently compute Fourier transforms [12]. Ordinary matrix-vector multiplication of an $N \times N$ matrix with an $N \times 1$ vector requires operations proportional to N^2[23]. But the Fourier matrix has a special structure. The FFT reduces the operations to be proportional to $N \log(N)$ - a large computational speed up!

In the simplest version, the FFT requires N to be a power of two. It then breaks up the full-sized FFT to smaller FFT blocks via recursion. This can be seen from using the component-wise definition of the Fourier transform (Eq. 2.346) and splitting the sum up

[23] Matrix-vector multiplications involving sparse matrices can be computed faster, but this requires many zero elements. The Fourier matrix has no zero elements.

in terms of a sum of smaller FFTs [13],

$$\boldsymbol{v}_m = \sum_{n=0}^{N-1} \exp\left(-i2\pi \frac{m\cdot n}{N}\right) \boldsymbol{u}_n \tag{2.349}$$

$$= \sum_{n=0}^{N/2-1} \exp\left(-i2\pi \frac{m\cdot [2n]}{N}\right) \boldsymbol{u}_{2n} + \sum_{n=0}^{N/2-1} \exp\left(-i2\pi \frac{m\cdot [2n+1]}{N}\right) \boldsymbol{u}_{2n+1} \tag{2.350}$$

$$= \sum_{n=0}^{N/2-1} \exp\left(-i2\pi \frac{m\cdot n}{N/2}\right) \underbrace{\boldsymbol{u}_{2n}}_{\text{even}} + \underbrace{\exp\left(-i2\pi \frac{m}{N}\right)}_{\boldsymbol{D}_{N/2}} \sum_{n=0}^{N/2-1} \exp\left(-i2\pi \frac{m\cdot n}{N/2}\right) \underbrace{\boldsymbol{u}_{2n+1}}_{\text{odd}} \tag{2.351}$$

This can be rewritten in matrix form,

$$\boldsymbol{F}_N = \begin{bmatrix} \boldsymbol{I}_{N/2} & \boldsymbol{D}_{N/2} \\ \boldsymbol{I}_{N/2} & -\boldsymbol{D}_{N/2} \end{bmatrix} \begin{bmatrix} \boldsymbol{F}_{N/2} & 0 \\ 0 & \boldsymbol{F}_{N/2} \end{bmatrix} \boldsymbol{P}_{\text{even,odd}} \tag{2.352}$$

where the opposite signs in front of $\boldsymbol{D}_{N/2}$ come from the fact that

$$\exp\left(-i2\pi \frac{m+N/2}{N}\right) = \exp\left(-i2\pi \frac{m}{N} - i\pi\right) = -\exp\left(-i2\pi \frac{m}{N}\right). \tag{2.353}$$

For example, for $N = 2^3 = 8$, the FFT may be written as

$$\boldsymbol{F}_8 = \begin{bmatrix} \boldsymbol{I}_4 & \boldsymbol{D}_4 \\ \boldsymbol{I}_4 & -\boldsymbol{D}_4 \end{bmatrix} \begin{bmatrix} \boldsymbol{F}_4 & 0 \\ 0 & \boldsymbol{F}_4 \end{bmatrix} \boldsymbol{P}_{\text{even,odd}} \tag{2.354}$$

where $\boldsymbol{I}_4 = \text{eye}(4)$, $\boldsymbol{D}_4$ is a diagonal matrix with w^m $(m = 0, 1, \ldots, N/2-1)$ on its diagonal, and the last matrix is a permutation matrix that first takes all input entries with even index, then all entries with odd index (start counting from 0). For the specific example of $N = 8$,

$$\boldsymbol{P}_{\text{even,odd}} = \begin{bmatrix} 1 & & & & & & & \\ & & 1 & & & & & \\ & & & & 1 & & & \\ & & & & & & 1 & \\ & 1 & & & & & & \\ & & & 1 & & & & \\ & & & & & 1 & & \\ & & & & & & & 1 \end{bmatrix}. \tag{2.355}$$

Equation 2.354 and Figure 2.21 illustrate a single step[24] of the FFT. The FFT algorithm then proceeds to split up smaller blocks $(\boldsymbol{F}_4, \boldsymbol{F}_2, \boldsymbol{F}_1)$. Each such recursive step almost halves the computational cost of the Fourier transform. The following code illustrates a single step of the FFT recursion.

[24] Figure 2.21 also indicates a second recursive step, as shown in the yellow box.

```
clc
% index "s": small
% index "l": large
tic
N = 4;
w = exp(1i * 2*pi / N);
Isub = eye(N/2);
Fsub = fft(Isub);
dsub = w.^(0:N/2-1)';
Dsub = diag( dsub );
% matrix-based implementation (slow)
% compute ground truth
Flarge = fft( eye(N) );
% permutation matrix
Plarge = zeros(N);
idx = sub2ind([N N], 1:N, [1:2:(N-1), 2:2:N]);
Plarge( idx ) = 1;
% compute fft step
Glarge = [Isub, Dsub; Isub, -Dsub]*...
         [Fsub, zeros(N/2); zeros(N/2), Fsub]*Plarge;
toc
% check Frobenius norm
disp(norm(Flarge - Glarge, 'fro'))
% more efficient implementation
% > FFT recursion directly applied to random vector v
rng(0)
v = rand(N,1);
% notice that index 1 in Matlab corresponds
% to 0 when using zero-based idexing
tic
veven = v(1:2:N-1);  % even
vodd = v(2:2:N);     % odd
veven_ft = fft(veven);
vodd_ft = fft(vodd);
v_ft = [veven_ft + dsub.*vodd_ft; veven_ft - dsub.*vodd_ft];
toc
% check L2 norm
disp(norm(v_ft - fft(v), 2))
```

Listing 2.19: fftRecursion.m (Matlab)

2.11.4 Relation to continuous Fourier transform

We shortly outline the connection between the discrete Fourier transform, as discussed in the previous subsection, to the so called *signal processing convention* of the continuous Fourier transform. In this convention defines the one-dimensional *continuous Fourier*

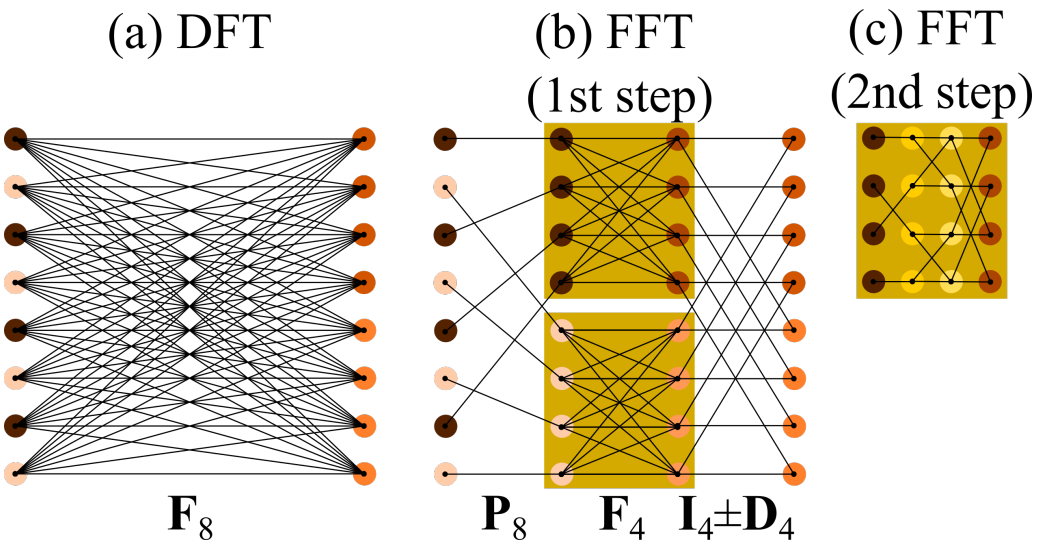


Figure 2.21: (a) Discrete Fourier transformation (DFT) $\boldsymbol{F}_8$ requires $\mathcal{O}\left[N^2\right]$ operations. (b) The fast Fourier transform (FFT) reduces the operations required to $\mathcal{O}\left[N \log N\right]$. The transition from (a) to (b) illustrates only one step of the FFT recursion as given by Eq. 2.352. (c) Step 2 breaks down the complexity of inside the yellow boxes by once more applying the FFT recursion. Notice how the number of connections, representing the number of flops needed to compute the Fourier transform, reduces drastically at each step of the FFT recursion.

transform (CFT) G of a function g is defined as

$$G(f) = \mathcal{F}\{g(x)\} = \int g(x) \exp(-j2\pi fx) dx, \tag{2.356}$$

where x and f denote space and spatial frequency dependencies. For two-dimensional functions we have

$$G(f_x, f_y) = \mathcal{F}\{g(x, y)\} = \iint g(x, y) \exp\left[-j2\pi\left(f_x x + f_y y\right)\right] dxdy. \tag{2.357}$$

The corresponding inverse Fourier transform of a function G is given by

$$g(x) = \mathcal{F}^{-1}\{G(f)\} = \int G(f) \exp(j2\pi fx) df \tag{2.358}$$

for one-dimensional functions, and

$$g(x, y) = \mathcal{F}^{-1}\{G(f_x, f_y)\} = \iint G(f_x, f_y) \exp\left[j2\pi\left(f_x x + f_y y\right)\right] df_x df_y \tag{2.359}$$

for two-dimensional functions. For notational simplicity, the operator $\mathcal{F}$ will not be distinguished for one- or two-dimensional transforms when it is clear from the context whether one- or two-dimensional Fourier transforms are meant. In later sections, the notation $\tilde{\psi} = \mathcal{F}\{\psi\}$ is used to distinguish a wave field ψ from its Fourier transform $\tilde{\psi}$.

Proofs of the following properties of Fourier transforms can be found in classical texts on Fourier analysis and Fourier Optics[25] [7, 19]:

- *Linearity.* Let $\mathcal{F}\{g\} = G$ and $\mathcal{F}\{h\} = H$, then

  $$\mathcal{F}\{\alpha g + \beta h\} = \alpha G + \beta H, \tag{2.360}$$

  where g and h are functions for which the Fourier transform exists and α and β are scalars.

- *Similarity Theorem.* If $\mathcal{F}\{g\} = G$, then

  $$\mathcal{F}\{g(\alpha x, \beta y)\} = \frac{1}{|\alpha\beta|} G\left(\frac{f_x}{\alpha}, \frac{f_y}{\beta}\right). \tag{2.361}$$

  Therefore increasing the extent of an object in the spatial domain shrinks its Fourier transform in the spatial frequency domain.

- *Shift Theorem.* If $\mathcal{F}\{g\} = G$, then

  $$\mathcal{F}\{g(x - \alpha, y - \beta)\} = G(f_x, f_y) \exp\left(-j2\pi\left(f_x x + f_y y\right)\right), \tag{2.362}$$

  i.e. shifting an object in the spatial domain yields multiplication by a constant phase factor in the spatial frequency domain.

[25] Except for Friedel's theorem, which is proven here.

- *Parseval's Theorem.* Let $\mathcal{F}\{g\} = G$ and $\mathcal{F}\{h\} = H$, then

$$\iint g(x,y) \cdot \overline{h}(x,y)\, dxdy = \iint G(f_x, f_y) \cdot \overline{H}(f_x, f_y)\, df_x df_y, \tag{2.363}$$

  where the bars denote complex conjugation. In particular, if $g = h$,

$$\iint |g(x,y)|^2 dxdy = \iint |G(f_x, f_y)|^2 df_x df_y, \tag{2.364}$$

  i.e., $\mathcal{F}$ is unitary since the L^2-norm is preserved under Fourier transformation.

- *Convolution Theorem.* Let $\mathcal{F}\{g\} = G$ and $\mathcal{F}\{h\} = H$, then

$$\mathcal{F}\{(g \otimes h)(x,y)\} = G(f_x, f_y) H(f_x, f_y), \tag{2.365}$$

  where $(g \otimes h)(x,y) := \iint g(\xi,\eta)\, h(x-\xi, y-\eta)\, d\xi d\eta$ defines the convolution operation.

- *Autocorrelation Theorem.* If $\mathcal{F}\{g\} = G$, then

$$\mathcal{F}\left\{\iint g(\xi,\eta)\, \overline{g}(\xi - x, \eta - y)\, d\xi d\eta\right\} = |G(f_x, f_y)|^2 \tag{2.366}$$

  and

$$\mathcal{F}\left\{|g(x,y)|^2\right\} = \iint G(\xi,\eta)\, \overline{G}(\xi - f_x, \eta - f_y)\, d\xi d\eta. \tag{2.367}$$

- *Fourier Integral Theorem.* At points of continuity of g,

$$\mathcal{F}\mathcal{F}^{-1}\{g(x,y)\} = \mathcal{F}^{-1}\mathcal{F}\{g(x,y)\} = g(x,y); \tag{2.368}$$

  i.e., $\mathcal{F}^{-1}$ is the inverse operator of $\mathcal{F}$.

- *Uncertainty relation.* Let $\mathcal{F}\{g(t)\} = G(f)$, define the variance[26]

$$(\Delta t)^2 = \frac{\int t^2 g(t)\, \overline{g}(t)\, dt}{\int g(t)\, \overline{g}(t)\, dt} \tag{2.369}$$

  and

$$(\Delta f)^2 = \frac{\int f^2 G(f)\, \overline{G}(f)\, df}{\int G(f)\, \overline{G}(f)\, df}, \tag{2.370}$$

  then

$$\Delta t \Delta f \geq \frac{1}{4\pi}. \tag{2.371}$$

- *Friedel's Theorem.* For real.valued functions $g(x,y) = \overline{g}(x,y)$,

$$G(-f_x, -f_y) = \overline{G}(f_x, f_y); \tag{2.372}$$

[26] In this definition of variance, g is amplitude while $g(t)\,\overline{g}(t)$ is intensity; similarly for G.

i.e., the Fourier transform of a real-valued function is Hermitian[27].

Proof: Let g be real-valued, i.e. $g(x) = \bar{g}(x)$, then its Fourier transform is given by

$$G(f) = \mathcal{F}\{g(x)\} = \int g(x) \exp(-j2\pi fx) dx, \tag{2.373}$$

or equivalently

$$G(-f) = \int g(x) \exp(j2\pi fx) dx. \tag{2.374}$$

Complex conjugation yields

$$\begin{aligned} \overline{G}(-f) &= \overline{\int g(x)\ \exp(j2\pi fx) dx} \\ &= \int \overline{g(x)}\ \overline{\exp(j2\pi fx)} dx \\ &= \int g(x)\ \exp(-j2\pi fx) dx \\ &= G(f). \end{aligned} \tag{2.375}$$

In a similar way the two-dimensional case is verified. This completes the proof.

The definitions of the *discrete Fourier transform* (DFT) vary in the literature. We define the two-dimensional DFT as

$$G[p,q] = \sum_{m=-M/2}^{M/2-1} \sum_{n=-N/2}^{N/2-1} g[m,n] \exp\left[-j2\pi\left(\frac{pm}{M} + \frac{qn}{N}\right)\right] \tag{2.376}$$

where $p = -\frac{M}{2}, ..., \frac{M}{2} - 1$ and $q = -\frac{N}{2}, ..., \frac{N}{2} - 1$. It is straightforward to show (see for instance [41]) that this definition directly follows from approximating the CFT (Eqs. 2.357 and 2.359) by a Riemann sum and neglecting the step size $dx \approx \Delta x$ and $dy \approx \Delta y$. The corresponding inverse discrete Fourier transform (iDFT) is given by

$$g[m,n] = \sum_{p=-M/2}^{M/2-1} \sum_{q=-N/2}^{N/2-1} G[p,q] \exp\left[j2\pi\left(\frac{pm}{M} + \frac{qn}{N}\right)\right] \tag{2.377}$$

where $m = -\frac{M}{2}, ..., \frac{M}{2} - 1$ and $n = -\frac{N}{2}, ..., \frac{N}{2} - 1$. We again neglected the step size $df_x \approx \Delta f_x$ and $df_y \approx \Delta f_y$.

From the definitions of the DFT and iDFT, we see that increasing the grid size in the spatial domain space by appending zeros to the signal g results in a finer sampling in reciprocal space. This operation is referred to as *zero-padding*, which is a common digital signal processing operation to refine the sampling step size in reciprocal space.

While the DFT, defined via an approximation to the CFT by means of a finite Riemann sum, is consistent with the definition of the CFT, Matlab uses a different convention

[27] Accordingly, the absolute value squared of a real-valued transmission function is centrosymmetric.

because it uses only positive indices for labeling arrays. We thus have to modify the DFT definition slightly. In Matlab the DFT is defined as

$$\widetilde{G}[p,q] = \sum_{m=1}^{M}\sum_{n=1}^{N} \widetilde{g}[m,n] \exp\left[-j2\pi\left(\frac{(p-1)(m-1)}{M} + \frac{(q-1)(n-1)}{N}\right)\right], \quad (2.378)$$

where

$$\widetilde{g}[m,n] = \begin{cases} g[m+M/2, n+N/2] & \text{for } m = 1,...,M/2-1;\ n = 1,...,N/2-1 \\ g[m-M, n-N] & \text{for } m = M/2,...M;\ n = N/2,...,N \end{cases}. \quad (2.379)$$

The corresponding inverse DFT is given by

$$\widetilde{g}[m,n] = \frac{1}{MN}\sum_{p=1}^{M}\sum_{q=1}^{N} \widetilde{G}[p,q] \exp\left[j2\pi\left(\frac{(p-1)(m-1)}{M} + \frac{(q-1)(n-1)}{N}\right)\right]. \quad (2.380)$$

2.12 Non-negative matrix factorization (NMF)

In this section, we discuss the non-negative matrix factorization

$$\underbrace{\boldsymbol{A}}_{M\times N} \approx \underbrace{\boldsymbol{W}}_{M\times r} \cdot \underbrace{\boldsymbol{H}}_{r\times N} \quad (2.381)$$

subject to $\boldsymbol{W}, \boldsymbol{H} \geq 0$ and where the factorization rank $r < N$. The main goal of this decomposition is to find a reduced order model for the data represented by the columns of $\boldsymbol{A}$. The non-negativity requirement arises from the goal to have intelligible components as further discussed below. We see that the NMF amounts to data compression, if the right-hand side of Eqn. 2.381 has less degrees of freedom than the right hand side,

$$M \cdot N > M \cdot r + r \cdot N = (M+N)\, r. \quad (2.382)$$

Assuming[28] $M \gg N$, the above inequality reduces to the requirement $r < N$.

Lee and Seung popularized non-negative matrix factorization in a seminal paper [**?**],in which simple iterative update rules to estimate $\boldsymbol{W}$ and $\boldsymbol{H}$ from $\boldsymbol{A}$ where given. We will simply state the update rules here and provide a derivation later when we have the necessary mathematical tools (namely, matrix-valued derivatives, see section **??**). For the case of Gaussian noise,

$$\text{argmin}_{\boldsymbol{W},\boldsymbol{H}\geq 0} \|\boldsymbol{A} - \boldsymbol{W}\boldsymbol{H}\|_2^2 \quad (2.383)$$

leads to the update rules

$$\boldsymbol{H} \leftarrow \boldsymbol{H}. \times \left(\left[\boldsymbol{W}^T\boldsymbol{A}\right] ./ \left[\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{H} + \epsilon \mathbf{1}_{k\times N}\right]\right) \quad (2.384)$$

$$\boldsymbol{W} \leftarrow \boldsymbol{W}. \times \left(\left[\boldsymbol{A}\boldsymbol{H}^T\right] ./ \left[\boldsymbol{W}\boldsymbol{H}\boldsymbol{H}^T + \epsilon \mathbf{1}_{M\times k}\right]\right), \quad (2.385)$$

[28] In this book, the NMF arises in the context of spectral unmixing. In this context we typically rearrange the images along columns, so that the rows correspond to pixels. Typically we have millions of pixels ($M \sim 10^6$ rows) and only a handful or a few tens of images ($N < 10^2$ columns).

where the arrows to the left ($\leftarrow$) mean "replace by", $.\times$ and $./$ denote element-wise multiplication and division, respectively, and $\mathbf{1}_{k\times N}$ is a $k \times N$ matrix filled with ones, ϵ is a small constant to prevent division by zero. Initializing with any random matrix, whose elements are non-negative, the above iterative NMF update rules preserve non-negativity.

We note that the basic NMF algorithm as presented here has certain ambiguities. For example, we may insert a permutation matrix and its inverse (which is equal to its transpose for a permutation) to get

$$\boldsymbol{A} = \boldsymbol{W}\boldsymbol{H} = \boldsymbol{W}c\boldsymbol{P}\boldsymbol{P}^T c^{-1}\boldsymbol{H} = \boldsymbol{W}^{'}\boldsymbol{H}',$$

where c is an arbitrary constant scalar. We see that if $\boldsymbol{W}$ and $\boldsymbol{H}$ is a NMF decomposition of $\boldsymbol{A}$, then $\boldsymbol{W}^{'} = \boldsymbol{W}c\boldsymbol{P}$ and $\boldsymbol{H}' = \boldsymbol{P}^T c^{-1}\boldsymbol{H}$ is also a solution. In practice, this issue rarely matters, but we recommend to be aware of the existing ambiguities. Regularization techniques and/or additional normalization constraints can be used to avoid ambiguities.

Listing 2.20 provides code for computing the NMF. Here we use the singular value decomposition for initialization of the NMF factors $\boldsymbol{W}$ and $\boldsymbol{H}$. In particular, the SVD can attain negative values, which is why the absolute values are computed here. For more involved variants of NMF, the reader is referred to the book on NMF by Gillis [**?**].

```
function [W,H,residual] = L2_NMF(A,r,numIter)
% NMF to minimize norm(A-W*H,'fro') wrt W,H
% input:
% A: MxN matrix (images arranged as row vectors)
% r: rank
% numIter: number of iterations
% output:
% W: interpretable output images
% H: weights

% initialize by absolute value of each
% truncated SVD factors
[U,S,V] = svd(A,'econ');
W = (abs(U(:,1:r)) + 1)/2;
H = (abs(S(1:r,1:r)*V(:,1:r)') + 1);
delta = 1;      % prevent division by zero

for k = 1:numIter
    H = H .* (W'*A + delta)./((W'*W)*H + delta);
    W = W .* ((A*H' + delta)./(W*(H*H') + delta));
end

residual = norm(A - W*H,'fro') / norm(A, 'fro');
end
```

Listing 2.20: L2_NMF.m (Matlab)

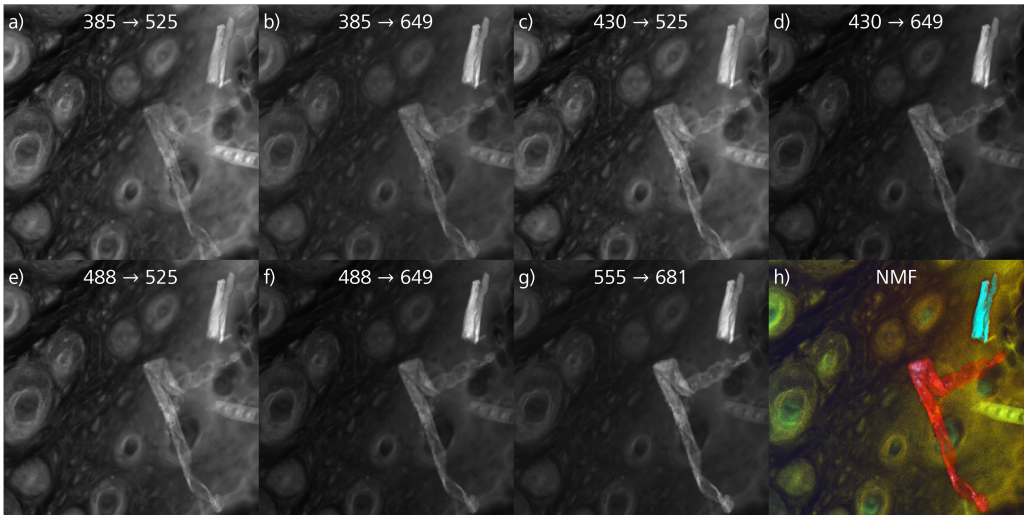


Figure 2.22: Example of non-negative matrix factorization on autofluorescent data from unlabeled mouse histological specimen. (a-g) Raw data, (h) rank-3 NMF, where the unmixed channels are encoded in the red, green and blue channels.

Application of NMF to spectral unmixing

Given a microscope with the ability to excite a specimen at multiple wavelengths and detect fluorescence at multiple emission windows , we can perform a variety of autofluorescence measurements on a specimen of interest. An example is shown in Fig. 2.22. Panels (a-g) show the raw data, where the numbers at the top of each image signify excitation and emission central wavelengths (excitation $\rightarrow$ emission). It can be seen that the raw data appears different as the excitation and emission central wavelengths are changed. Different chemical constituents inside the specimen carry distinct spectral signatures. However, each image contains similar information, suggesting that each image is a linear combination of fundamental autofluorescence images. NMF is a suitable tool to reduce such variable fluorescence data into a few interpretable non-negative components. Computing a rank-3 NMF decomposition of the raw data and combining the unmixed images into the individual channels of an rgb image results in the image shown in panel h). It appears that the specimen was contaminated with two specs of unknown origin. However, these components are clearly highlighted by their distinct color, as can be seen on the right hand side of panel h (turquoise and green structures). Moreover, the tissue itself appears to have mainly two different autofluorescence spectral signatures, as can be seen from the green and turquoise colors of the tissue.

In performing NMF, several pitfalls have to be avoided: 1) The individual spectral channels of the raw images have to be aligned well. A lateral shift would break the model that the images are linearly dependent in the given pixel basis. 2) The images should be background-subtracted. A variable background can corrupt the expansion coefficients in the linear combinations. We suggest to at least subtract the minimum intensity from each respective image. 3) The illumination profile should be homogeneous. If the illumination

has a spatially dependent profile changing as the illumination wavelength is varied, for example spectrally dependent shading towards the edges of the field of view, the linearity assumption is not valid.

The factorization rank of the NMF can be manually tuned by trial and error or automatically determined by considering the decay of singular values in the SVD [**?**].

2.13 Dynamic mode decomposition (DMD)

In this section we discuss a matrix factorization used to analyze time series of images. Suppose we are given a sequence of images reshaped into column vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$. We are looking for a linear operator $\boldsymbol{A}$ that advances the system state from one point in time to another

$$\boldsymbol{x}_{j+1} = \underbrace{\boldsymbol{A}}_{M\times M} \boldsymbol{x}_j. \tag{2.386}$$

Assuming M is the number of pixels much larger than the number of observations over time N, the operator $\boldsymbol{A}$ defined in this way would require M^2 elements. For example, for an image with $1000 \times 1000 = 10^6$ pixels, the operator $\boldsymbol{A}$ would require 10^{12} elements (or one "tera element") to be specified. Our goal is to find a more efficient representation of $\boldsymbol{A}$.

Formally, the operator $\boldsymbol{A}$ may be found as follows: (1) Allocate the column vectors $\boldsymbol{x}_j$ along matrices

$$\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{N-1}] \tag{2.387}$$

and

$$\boldsymbol{X}' = [\boldsymbol{x}_2, \boldsymbol{x}_3, \ldots, \boldsymbol{x}_N]\,. \tag{2.388}$$

Then we have

$$\underbrace{\boldsymbol{X}'}_{M\times(N-1)} \approx \underbrace{\boldsymbol{A}}_{M\times M} \cdot \underbrace{\boldsymbol{X}}_{M\times(N-1)}\,. \tag{2.389}$$

Formally, the least squares solution of estimating $\boldsymbol{A}$ in the Frobenius norm (assuming Gaussian noise) is

$$\boldsymbol{A} \approx \mathrm{argmin}_{\boldsymbol{A}} \left\|\boldsymbol{A}\boldsymbol{X} - \boldsymbol{X}'\right\|_F^2 = \boldsymbol{X}'\boldsymbol{X}^\dagger \left(\boldsymbol{X}\boldsymbol{X}^\dagger\right)^{-1}. \tag{2.390}$$

We now embark on a computationally more efficient path to find a representation of $\boldsymbol{A}$. We start with computing the truncated (rank r) SVD of the data matrix $\boldsymbol{X}$, namely

$$\boldsymbol{X} = \underbrace{\boldsymbol{U}}_{M\times r} \cdot \underbrace{\boldsymbol{S}}_{r\times r} \cdot \underbrace{\boldsymbol{V}^\dagger}_{r\times(N-1)} \tag{2.391}$$

and inserting this into the least squares solution for $\boldsymbol{A}$ (Eqn. 2.390),

$$\boldsymbol{A} \approx \boldsymbol{X}'\boldsymbol{V}\boldsymbol{S}\boldsymbol{U}^\dagger \left(\boldsymbol{U}\boldsymbol{S}^2\boldsymbol{U}^\dagger\right)^{-1} = \boldsymbol{X}'\boldsymbol{V}\boldsymbol{S}^{-1}\boldsymbol{U}^\dagger. \tag{2.392}$$

We note that (1) we neglect notational distinction of $\boldsymbol{A}$ as written in the latter equation being an estimate (which in the literature is often done using a hat or other symbols) and that (2) this is a low rank approximation. The second assumption is a strong one: we are assuming there is a low rank structure in the temporal evolution our image sequence $\boldsymbol{X}$.

Given the singular value decomposition, we can use the orthogonal basis $\boldsymbol{U}$ to project $\boldsymbol{A}$ into a lower-dimensional space (left-multiply the latter equation by $\boldsymbol{U}^\dagger$, right-multiply by $\boldsymbol{U}$),

$$\underbrace{\boldsymbol{\mathcal{A}}}_{r\times r} = \underbrace{\boldsymbol{U}^\dagger}_{r\times M}\cdot\underbrace{\boldsymbol{A}}_{M\times M}\cdot\underbrace{\boldsymbol{U}}_{M\times r} = \underbrace{\boldsymbol{U}^\dagger}_{r\times M}\cdot\underbrace{\boldsymbol{X}'}_{M\times(N-1)}\cdot\underbrace{\boldsymbol{V}}_{(N-1)\times r}\cdot\underbrace{\boldsymbol{S}^{-1}}_{r\times r} \tag{2.393}$$

If $\boldsymbol{y}$ is an eigenvector of $\boldsymbol{A}$ with eigenvalue λ,

$$\boldsymbol{A}\boldsymbol{y} = \lambda\boldsymbol{y}, \tag{2.394}$$

then

$$\boldsymbol{z} = \boldsymbol{U}^\dagger\boldsymbol{y} \tag{2.395}$$

is an eigenvector of $\boldsymbol{\mathcal{A}}$ which shares the same eigenvalue λ as $\boldsymbol{y}$ has with respect to $\boldsymbol{A}$,

$$\boldsymbol{\mathcal{A}}\boldsymbol{z} = \left(\boldsymbol{U}^\dagger\boldsymbol{A}\boldsymbol{U}\right)\left(\boldsymbol{U}^\dagger\boldsymbol{y}\right) = \boldsymbol{U}^\dagger\boldsymbol{A}\boldsymbol{y} = \boldsymbol{U}^\dagger\left(\lambda\boldsymbol{y}\right) = \lambda\left(\boldsymbol{U}^\dagger\boldsymbol{y}\right) = \lambda\boldsymbol{z}. \tag{2.396}$$

We say that $\boldsymbol{\mathcal{A}} = \boldsymbol{U}^\dagger\boldsymbol{A}\boldsymbol{U}$ is a *similarity transformation*, which preserves eigenvalues. Conversely, we can reconstruct a high-dimensional eigenvector $\boldsymbol{y}$ of $\boldsymbol{A}$ from the low-dimensional eigenvector $\boldsymbol{z}$ of $\boldsymbol{\mathcal{A}}$, simply by multiplying both sides of Eqn. 2.395 with $\boldsymbol{U}$,

$$\boldsymbol{y} = \boldsymbol{U}z. \tag{2.397}$$

This suggests that the r dominant high-dimensional eigenvectors are given by the columns of

$$\boldsymbol{Y} = \boldsymbol{U}\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{V}\boldsymbol{S}^{-1}\boldsymbol{Z}. \tag{2.398}$$

We test the latter statement and examine the expression

$$\boldsymbol{A}\boldsymbol{Y} = (\boldsymbol{X}'\boldsymbol{V}\boldsymbol{S}^{-1}\boldsymbol{U}^\dagger)(\boldsymbol{X}\boldsymbol{V}\boldsymbol{S}^{-1}\boldsymbol{Z}). \tag{2.399}$$

Unfortunately, here we do find an eigenvalue equation. However, the turquoise term is reminiscent of the defining equation of the projected matrix (Eqn. 2.393). Replacing $\boldsymbol{X}$ by $\boldsymbol{X}'$ in the definition of the eigenvectors,

$$\boldsymbol{Y}' = \boldsymbol{X}'\boldsymbol{V}\boldsymbol{S}^{-1}\boldsymbol{Z}. \tag{2.400}$$

With this we have

$$\begin{aligned}
\boldsymbol{A}\boldsymbol{Y}' &= (\boldsymbol{X}'\boldsymbol{V}\boldsymbol{S}^{-1}\underbrace{\boldsymbol{U}^\dagger)(\boldsymbol{X}'\boldsymbol{V}\boldsymbol{S}^{-1}}_{\boldsymbol{\mathcal{A}}}\boldsymbol{Z}) && (2.401)\\
&= \boldsymbol{X}'\boldsymbol{V}\boldsymbol{S}^{-1}\boldsymbol{\mathcal{A}}\boldsymbol{Z} && (2.402)\\
&= \underbrace{\boldsymbol{X}'\boldsymbol{V}\boldsymbol{S}^{-1}\boldsymbol{Z}}_{\boldsymbol{Y}'}\boldsymbol{\Lambda} && (2.403)\\
&= \boldsymbol{Y}'\boldsymbol{\Lambda} && (2.404)
\end{aligned}$$

We see that the columns of $\boldsymbol{Y}'$ provide eigenvectors of $\boldsymbol{A}$.

In hindsight, the dynamic mode decomposition allows for efficiently modeling exponentially decaying/increasing or oscillating time series of the form

$$\boldsymbol{x}_j = \boldsymbol{A}\boldsymbol{x}_{j-1} = \boldsymbol{A}^2\boldsymbol{x}_{j-2} = \ldots = \boldsymbol{A}^j\boldsymbol{x}_0. \tag{2.405}$$

The operator $\boldsymbol{A}$ is large in dimension and computationally inefficient to evaluate. We could have attempted to find $\boldsymbol{A}$ first and then diagonalize via its its eigendecomposition. However, that route is prohibitively expensive when the number of rows M of $\boldsymbol{A}$ is large. The dynamic mode decomposition computes an eigendecomposition of the projected low rank $(r \times r)$ matrix $\boldsymbol{\mathcal{A}} = \boldsymbol{U}^\dagger\boldsymbol{A}\boldsymbol{U}$, where $\boldsymbol{U}$ comes from the SVD of $\boldsymbol{A}$, which can be computed efficiently. Mathematically the time evolution then takes place on the projected vectors and matrix,

$$\underbrace{\boldsymbol{x}_j}_{M\times 1} = \underbrace{\boldsymbol{A}}_{M\times M} \cdot \underbrace{\boldsymbol{x}_{j-1}}_{M\times 1} \tag{2.406}$$

$$\approx \boldsymbol{U}\left(\boldsymbol{U}^\dagger\boldsymbol{A}\boldsymbol{U}\right)\left(\boldsymbol{U}^\dagger\boldsymbol{x}_{j-1}\right) \tag{2.407}$$

$$= \boldsymbol{U}\boldsymbol{\mathcal{A}}\boldsymbol{\chi}_{j-1} \tag{2.408}$$

or

$$\underbrace{\boldsymbol{\chi}_j}_{r\times 1} = \underbrace{\boldsymbol{\mathcal{A}}}_{r\times r} \cdot \underbrace{\boldsymbol{\chi}_j}_{r\times 1}, \tag{2.409}$$

where $\boldsymbol{\chi}_j = \boldsymbol{U}^\dagger\boldsymbol{x}_j$ is the projected state vector. The computational efficiency of the DMD comes from Eqn. 2.409 being of much lower dimensionality than Eqn. 2.406. We note that arbitrary time points can be computed by

$$\boldsymbol{\chi}_j = \boldsymbol{\mathcal{A}}^j \cdot \boldsymbol{\chi}_0 = \boldsymbol{Z}\boldsymbol{\Lambda}^j\boldsymbol{Z}^{-1}\boldsymbol{\chi}_0. \tag{2.410}$$

In summary, computation of the dynamic mode decomposition involves four steps:

1. Compute truncated rank-r SVD of $(N-1)$-point data matrix:

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\dagger \tag{2.411}$$

2. Compute projected operator:

$$\boldsymbol{\mathcal{A}} = \boldsymbol{U}^\dagger\boldsymbol{X}'\boldsymbol{V}\boldsymbol{S}^{-1} \tag{2.412}$$

3. Compute projected eigendecomposition:

$$\boldsymbol{\mathcal{A}}\boldsymbol{Z} = \boldsymbol{Z}\boldsymbol{\Lambda} \tag{2.413}$$

4. Compute high-dimensional eigenvectors of $\boldsymbol{A}$:

$$\boldsymbol{Y}' = \boldsymbol{X}'\boldsymbol{V}\boldsymbol{S}^{-1}\boldsymbol{Z} \tag{2.414}$$

Listing 2.21 provides code for computing the DMD.

```
function [Y, Lambda] = DMD(X, r)
% dynamic mode decomposition of a time series
% DMD model: X(:,2:end) = A*X(:,1:end-1)
% input:
% X: input data (entire time series M x N)
% r: rank
% output:
% Y: high-dimensional eigenmodes of A
% Lambda: oscillation/decay rates

% step 1: truncated SVD
[U,S,V] = svd(X(:,1:end-1),'econ');
Ur = U(:,1:r);
Sr = S(1:r,1:r);
Vr = V(:,1:r);
% Step 2: compute projected matrix
Aprojected = Ur' * X(:,2:end) * (Vr/Sr);
% Step 3: eigendecomposition
[Z,Lambda] = eig(Aprojected);
% step 4: high-dimensional eigenmodes
Y = X(:,2:end)*(Vr/Sr)*Z;
end
```

Listing 2.21: DMD.m (Matlab)

Application of DMD to spectral unmixing

We have already discussed spectral unmixing in the context of non-negative matrix factorization. In the latter case, NMF was used to distinguish a few independent images in a variety of spectral channels. In particular, NMF required the number of independent images to be lower than the number of channels measured. In addition, the independent channels were required to be non-negative since negative fluorescence intensities would lack physical interpretability.

Dynamic mode decomposition (DMD) can be used to unmix time series of a single fluorescence channel. In Fig. 2.23 we recorded a 100-frame video of a fixated adherent cell, where phalloidin was used to label actin. Only a single excitation wavelength and a single emission filter (suited for the emission spectrum of phalloidin) was used while recording the time sequence. Panel (a) and (b) show the first and last frame of the video, respectively. While the brightness of the images was normalized for ease of inspection, it is seen that the image content has changed. In the first frame, the nuclei are prominently visible, while in the last frame the actin structures have gained in relative brightness. The change in relative brightness is due to bleaching of both the specific label (phalloidin) and bleaching of the autofluorescent signal. We used DMD to extract two independent components ($r = 2$) from the video sequence. The results are shown in panel (c) and (d). Panel (c) gives now a clean view on the actin structures, while panel (d) is the

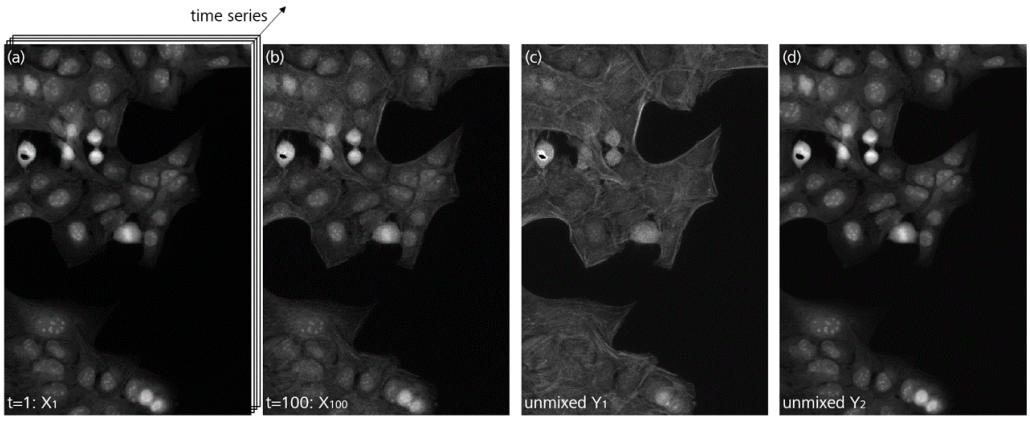


Figure 2.23: Unmixing of a time series via dynamic mode decomposition. (a) First image and (b) last image in a 100-frame video sequence. (c,d) unmixed independent components with different decay rates. Panel (c) shows actin labeled with phalloidin, panel (d) shows autofluorescence.

autofluorescent signal. We note that the high-dimensional eigenvectors ($\boldsymbol{Y}_1$ actin and $\boldsymbol{Y}_2$ autofluorescence) are the main quantities of interest in this example. We also extracted decay rates, stating that the phalloidin label is more stable than the autofluorescence, with eigenvalues of $\lambda_1 = 0.995$ and $\lambda_2 = 0.971$. The larger the difference in eigenvalues, the easier it is to perform spectral unmixing based on DMD.

2.14 Summary of important matrix factorizations

In this chapter, we have seen a variety of matrix factorizations (or decompositions), which are summarized here:

- QR decomposition: $\boldsymbol{A} = \boldsymbol{QR}$ arises from Gram-Schmidt orthogonalization. Here $\boldsymbol{Q}$ is an orthogonal basis, $\boldsymbol{R}$ contains expansion coefficients, namely the k-th column $\boldsymbol{R}_k$ contains the expansion coefficients of the k-th column of $\boldsymbol{A}_k$, so that $\boldsymbol{A}_k = \boldsymbol{QR}_k$.
- eigen decomposition: $\boldsymbol{A} = \boldsymbol{X\Lambda X}^{-1}$ arises in the computation of matrix powers and practically in the computation of the SVD.
- singular value decomposition (SVD): $\boldsymbol{A} = \boldsymbol{USV}^\dagger$ arises from finding an orthonormal basis $\boldsymbol{V}$ in the domain of $\boldsymbol{A}$ that maps into an orthonormal basis $\boldsymbol{U}$ in the range of $\boldsymbol{A}$.
- non-negative matrix factorization (NMF): Similar to the truncated SVD, the NMF is a low-rank approximation. Unlike the truncated SVD, the NMF decomposes a matrix into non-negative factors. In applications, such as fluorescence microscopy, the non-negativity of the factors is important for interpretability of the factors.

- dynamic mode decomposition (DMD): Enables time series analysis, for example of video sequences. Originally proposed for fluid dynamics, the DMD has not found many applications in microscopy. However, we believe the DMD will attain popularity for analysis of fluorescence decay (bleaching / life time).

2.15 Computational costs of common vector and matrix operations

Table 2.1 summarizes computational costs for some of the matrix and vector operations discussed in this chapter. We assume $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{C}^{N\times 1}$, $\boldsymbol{A} \in \mathcal{C}^{M\times N}$, $\boldsymbol{B} = \mathcal{C}^{N\times P}$. $\boldsymbol{F} \in \mathcal{C}^{N\times N}$ is the fast Fourier transform matrix.

name	mathematical form	computational complexity
inner product	$\boldsymbol{x}^\dagger \boldsymbol{y}$	$\mathcal{O}(N)$
matrix-vector multiplication	$\boldsymbol{A}\boldsymbol{x}$	$\mathcal{O}(M \cdot N)$
matrix-matrix multiplication	$\boldsymbol{A}\boldsymbol{B}$	$\mathcal{O}(M \cdot N \cdot P)$
matrix inversion[29]	$\boldsymbol{A}^{-1}$	$\mathcal{O}\left(N^3\right)$
QR decomposition	$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R}$	$\mathcal{O}\left(M \cdot N^2\right)$
singular value decomposition (SVD)	$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\dagger$	$\mathcal{O}\left(M \cdot N^2 + M^2 \cdot N\right)$
Power method[30]	$\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x}$	$\mathcal{O}\left(N^2\right)$ per iteration
Fast Fourier transform (DFT)	$\boldsymbol{F}\boldsymbol{x}$	$\mathcal{O}(N \cdot \log[N])$

Table 2.1: Overview of computational costs of various matrix and vector operations.

2.16 List of notation and symbols

[29] Only defined for $M = N$

[30] For finding eigenvalues/-vectors; only defined for $M = N$.

symbol	description
$\boldsymbol{x}$	vector
$\boldsymbol{x}^T$	transpose vector
$\boldsymbol{x}^\dagger$	conjugate transpose vector
$\boldsymbol{e}$	unit vector
$\boldsymbol{A}$	matrix
$\boldsymbol{A}^{-1}$	inverse matrix
$\boldsymbol{A}^T$	transpose matrix
$\boldsymbol{A}^\dagger$	conjugate transpose matrix
$\boldsymbol{A}_k$	k-th columnn of $\boldsymbol{A}$
$\boldsymbol{A}_{jk}$	j-th row, k-th column of $\boldsymbol{A}$
$\boldsymbol{I}$	identity matrix

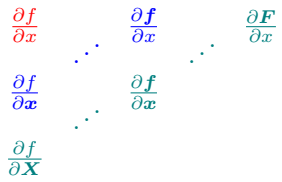
Table 2.2: List of common vector and matrix symbols.

Chapter 3

Optimization

3.1 Real-valued derivatives

In this section we assume that the reader is familiar with scalar-valued calculus, where each the input and output of a function are real (red in the table below). Beyond this, we are going to make use of a variety of higher-dimensional derivatives, which we may summarize as follows:



Vector-valued derivatives (blue) are either a result of a vector-valued function that is differentiated with respect to a scalar quantity (top row, middle) or by a scalar-valued function whose gradient is computed (middle row, left). Some readers will also be familiar with vector calculus, which we quickly review. However, machine learning applications require yet another level of sophistication, which is described by matrix-valued derivatives (turquoise)[1]. Beyond scalar- and real-valued calculus, in optics we are often dealing with complex-valued quantities. Although complex-valued quantities may be dealt with by treating their real and imaginary part separately, this approach is often cumbersome and results in tedious calculations. An alternative calculus uses the so called *Wirtinger derivatives* to facilitate the optimization of complex-valued quantities. In continuous optimization problems the so-called *functional derivatives* are relevant. We will review the different types of derivatives in this chapter. After we have discussed the aforementioned variety of derivatives, the chapter will discuss selected topics from the field of optimization.

[1] Tensor-valued calculus is needed in the optimization of weights in neural networks.

3.1.1 Real-valued derivatives: scalar input, scalar output

The derivative of a scalar-valued function $f : \mathcal{R} \to \mathcal{R}, x \to f(x)$ with respect to a scalar-valued input is defined as the limit

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}. \tag{3.1}$$

We assume the reader is familiar with the following differentiation rules (f, g, h all scalar- and real-valued)

$$f(x) = g(x) \cdot h(x) \blacktriangleright f'(x) = g'(x) h(x) + g(x) h'(x) \qquad \text{(product rule)} \tag{3.2}$$

$$f(x) = \frac{g(x)}{h(x)} \blacktriangleright f'(x) = \frac{g'(x) h(x) - g(x) h'(x)}{[h(x)]^2} \qquad \text{(quotient rule)} \tag{3.3}$$

$$f(x) = h(g(x)) \blacktriangleright f'(x) = h'(g(x)) \cdot g'(x) \qquad \text{(chain rule).} \tag{3.4}$$

We also note the second order Taylor approximation

$$f(x + \Delta x) \approx f(x) + f'(x) \Delta x + \frac{1}{2} f''(x) \Delta x^2. \tag{3.5}$$

Example: phase synchronization

Suppose we are given two complex-valued vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathcal{C}^n$. Occasionally we need to perform an operation referred to as *phase synchronization*. This means that we are searching for a constant ϕ such that

$$\mathcal{L} = \|\exp(i\phi)\, \boldsymbol{a} - \boldsymbol{b}\|_2^2 \tag{3.6}$$

$$= (\exp(i\phi)\, \boldsymbol{a} - \boldsymbol{b})^\dagger (\exp(i\phi)\, \boldsymbol{a} - \boldsymbol{b}) \tag{3.7}$$

$$= \|\boldsymbol{a}\|_2^2 + \|b\|_2^2 - \exp(-i\phi)\, \boldsymbol{a}^\dagger \boldsymbol{b} - \exp(i\phi)\, \boldsymbol{b}^\dagger \boldsymbol{a} \tag{3.8}$$

is minimal. How do we obtain ϕ? The necessary condition for optimality is that the derivative of the above cost with respect to the unknown phase ϕ vanishes,

$$\frac{\partial \mathcal{L}}{\partial \phi} = i \exp(-i\phi)\, \boldsymbol{a}^\dagger \boldsymbol{b} - i \exp(i\phi)\, \boldsymbol{b}^\dagger \boldsymbol{a} = 0, \tag{3.9}$$

from which we get

$$\exp(i\phi) = \pm \sqrt{\frac{\boldsymbol{a}^\dagger \boldsymbol{b}}{\boldsymbol{b}^\dagger \boldsymbol{a}}}. \tag{3.10}$$

Notice that the complex root has two branches, so we have to check which of these solutions minimizes the original cost function. This can be done by either (1) directly inserting both possible solutions into the cost function and choosing the candidate for which the resulting L2 cost is minimal, or (2) by computing the second derivative and checking for positivity. Trying the latter we find[2]

$$\frac{\partial^2 \mathcal{L}}{\partial \phi^2} = \exp(-i\phi)\, \boldsymbol{a}^\dagger \boldsymbol{b} + \exp(i\phi)\, \boldsymbol{b}^\dagger \boldsymbol{a} \in \mathcal{R}$$

[2]Recall that a complex number plus its complex conjugate is real.

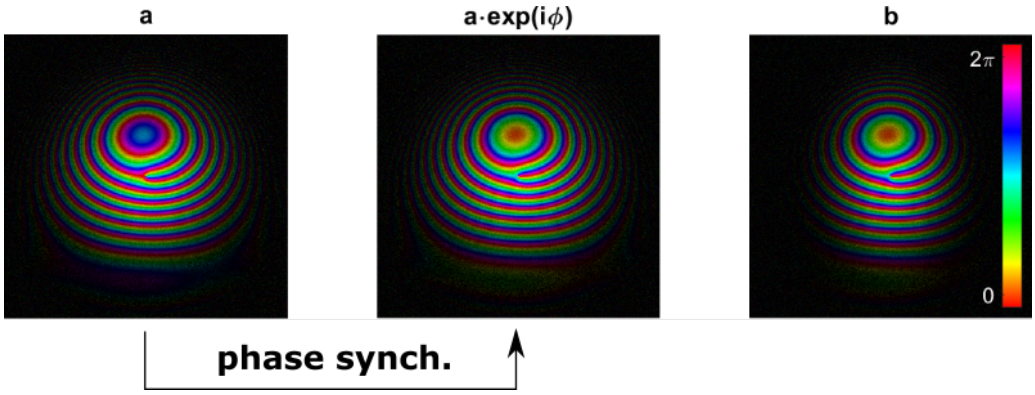


Figure 3.1: Estimation of a global phase offset in an optical beam via phase synchronization. (left) Beam $\boldsymbol{a}$ to be synchronized. (middle) Phase synchronized beam $\boldsymbol{a} \cdot \exp(i\phi)$. (c) Reference beam $\boldsymbol{b}$.

Thus, while the second derivative of $\mathcal{L}$ with respect to the unknown parameter ϕ is real, we do not know for a general combination of vectors $\boldsymbol{a}, \boldsymbol{b}$ whether it is positive or negative. Thus in practice, we have to take route (2) and perform a test, as done in the code phaseSynchronization.m. The utility of this function is illustrated in phaseSynchronizationExample.m, which was used to produce Fig. 3.1.

```
function exp_iPhi = phaseSynchronization(A, B)
% A, B: complex-valued images of same dimension
% exp_iPhi: minimizes the cost
% L = norm( exp_iPhi * A(:) - B(:), 2 )
a = A(:); % flatten image A to vector
b = B(:); % flatten image A to vector
% compute phase synch. term
exp_iPhi = sqrt( (a'*b) / (b'*a) );
% compare L2 norms of candidate solutions
L2_1 = norm(  exp_iPhi * a - b, 2 );
L2_2 = norm( -exp_iPhi * a - b, 2 );
if L2_2 < L2_1
    exp_iPhi = -exp_iPhi;
end

end
```

Listing 3.1: phaseSynchronization.m (Matlab)

3.1.2 Vector-valued derivatives: vector input, scalar output

The gradient of a scalar-valued function $f : \mathcal{R}^n \rightarrow \mathcal{R}, \boldsymbol{x} \rightarrow f(\boldsymbol{x})$ with respect to a vector-valued input is defined as

$$\nabla f(\boldsymbol{x}) = \frac{\partial f}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \cdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}, \tag{3.11}$$

where the partial derivatives are given by

$$\frac{\partial f}{\partial x_k} = \lim_{\Delta x \rightarrow 0} \frac{f(x_1, x_2, \ldots, x_k + \Delta x, \ldots, x_n) - f(x_1, x_2, \ldots, x_k, \ldots, x_n)}{\Delta x}. \tag{3.12}$$

The following differentiation rules (f, g, h all scalar- and real-valued) apply

$$f(\boldsymbol{x}) = g(\boldsymbol{x}) \cdot h(\boldsymbol{x}) \blacktriangleright \nabla f(\boldsymbol{x}) = \nabla g(\boldsymbol{x}) h(\boldsymbol{x}) + g(\boldsymbol{x}) \nabla h(\boldsymbol{x}) \quad \text{(product rule)} \tag{3.13}$$

$$f(\boldsymbol{x}) = \frac{g(\boldsymbol{x})}{h(\boldsymbol{x})} \blacktriangleright \nabla f(\boldsymbol{x}) = \frac{\nabla g(\boldsymbol{x}) h(\boldsymbol{x}) - g(\boldsymbol{x}) \nabla h(\boldsymbol{x})}{[h(\boldsymbol{x})]^2} \quad \text{(quotient rule)} \tag{3.14}$$

$$f(\boldsymbol{x}) = h(g(\boldsymbol{x})) \blacktriangleright \nabla f(\boldsymbol{x}) = h'(g(\boldsymbol{x})) \cdot \nabla g(\boldsymbol{x}) \quad \text{(chain rule)}. \tag{3.15}$$

A second-order Taylor approximation is given by [31]

$$f(\boldsymbol{x} + \Delta \boldsymbol{x}) \approx f(\boldsymbol{x}) + \nabla f(\boldsymbol{x}) \Delta \boldsymbol{x} + \frac{1}{2} \Delta \boldsymbol{x}^T \boldsymbol{H} \Delta \boldsymbol{x}, \tag{3.16}$$

where

$$\boldsymbol{H}_{i,j} = \frac{\partial^2 f(\boldsymbol{x})}{\partial x_i \partial x_j} \tag{3.17}$$

is the *Hessian matrix*. This matrix is symmetric and contains all second-order partial derivatives of f.

Note that

$$\frac{\partial x_i}{\partial x_j} = \delta_{i,j} = \begin{cases} 1 & , i = j \\ 0 & , \text{otherwise} \end{cases}, \tag{3.18}$$

where $\delta_{i,j}$ is the *Kronecker delta* (compare Eq. 2.53). Equation 3.18 asserts that components of $\boldsymbol{x}$ with different indices are mutually independent. The Kronecker delta is useful to simplify sum expressions. For instance,

$$\sum_j x_j \delta_{i,j} = x_i. \tag{3.19}$$

We will use the Kronecker delta to derive gradient identities in the examples below.

Example

Suppose we are given the quadratic form ($\boldsymbol{A} \in \mathcal{R}^{n \times n}$)

$$\mathcal{L} = \boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x} \tag{3.20}$$

$$= \boldsymbol{x}^T \sum_j \boldsymbol{A}_{:,j} x_j \tag{3.21}$$

$$= \sum_i \sum_j x_i A_{i,j} x_j. \tag{3.22}$$

Then the kth component of the gradient is given by

$$\frac{\partial \mathcal{L}}{\partial x_k} = \frac{\partial}{\partial x_k} \sum_i \sum_j x_i A_{i,j} x_j \tag{3.23}$$

$$= \sum_i \sum_j \frac{\partial x_i}{\partial x_k} A_{i,j} x_j + \sum_i \sum_j x_i A_{i,j} \frac{\partial x_j}{\partial x_k} \tag{3.24}$$

$$= \sum_i \sum_j \delta_{i,k} A_{i,j} x_j + \sum_i \sum_j x_i A_{i,j} \delta_{j,k} \tag{3.25}$$

$$= \sum_j A_{k,j} x_j + \sum_i x_i A_{i,k} \tag{3.26}$$

$$= \sum_j A_{k,j} x_j + \sum_i A^T_{k,i} x_i \tag{3.27}$$

$$= \boldsymbol{A}_{k,:} \boldsymbol{x} + \boldsymbol{A}^T_{k,:} \boldsymbol{x} \tag{3.28}$$

and

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}} = \begin{bmatrix} \partial \mathcal{L}/\partial x_1 \\ \partial \mathcal{L}/\partial x_2 \\ \cdots \\ \partial \mathcal{L}/\partial x_n \end{bmatrix}, = \begin{bmatrix} \boldsymbol{A}_{1,:} \boldsymbol{x} \\ \boldsymbol{A}_{2,:} \boldsymbol{x} \\ \cdots \\ \boldsymbol{A}_{n,:} \boldsymbol{x} \end{bmatrix} + \begin{bmatrix} \boldsymbol{A}^T_{1,:} \boldsymbol{x} \\ \boldsymbol{A}^T_{2,:} \boldsymbol{x} \\ \cdots \\ \boldsymbol{A}^T_{n,:} \boldsymbol{x} \end{bmatrix} \tag{3.29}$$

$$= \begin{bmatrix} \boldsymbol{A}^T_{1,:} \\ \boldsymbol{A}^T_{2,:} \\ \cdots \\ \boldsymbol{A}^T_{n,:} \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} \boldsymbol{A}_{:,1} \\ \boldsymbol{A}_{:,2} \\ \cdots \\ \boldsymbol{A}_{:,n} \end{bmatrix} \boldsymbol{x} \tag{3.30}$$

$$= \boldsymbol{A}^T \boldsymbol{x} + \boldsymbol{A} \boldsymbol{x} = \left(\boldsymbol{A}^T + \boldsymbol{A} \right) \boldsymbol{x}. \tag{3.31}$$

For the special case of a symmetric matrix $\boldsymbol{S} = \boldsymbol{A} = \boldsymbol{A}^T$, we get

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}} = 2 \boldsymbol{S} \boldsymbol{x}. \tag{3.32}$$

Example

Compute the gradient with respect to $\boldsymbol{x}$ of

$$\mathcal{L} = \boldsymbol{b}^T \boldsymbol{A}\boldsymbol{x} \tag{3.33}$$

$$= \boldsymbol{b}^T \sum_{j=1}^{n} \boldsymbol{A}_{:,j} x_j \tag{3.34}$$

$$= \sum_{i=1}^{m} \sum_{j=1}^{n} b_i A_{i,j} x_j, \tag{3.35}$$

where $\boldsymbol{A} \in \mathcal{R}^{m \times n}, \boldsymbol{x} \in \mathcal{R}^{n \times 1}, \boldsymbol{b} \in \mathcal{R}^{m \times 1}$. Then the kth component of the gradient with respect to $\boldsymbol{x}$ is given by

$$\frac{\partial \mathcal{L}}{\partial x_k} = \frac{\partial}{\partial x_k} \sum_i \sum_j b_i A_{i,j} x_j \tag{3.36}$$

$$= \sum_i \sum_j b_i A_{i,j} \frac{\partial x_j}{\partial x_k} \tag{3.37}$$

$$= \sum_i \sum_j b_i A_{i,j} \delta_{j,k} \tag{3.38}$$

$$= \sum_i b_i A_{i,k}. \tag{3.39}$$

$$= \sum_i b_i A^T_{k,i} \tag{3.40}$$

$$= \boldsymbol{A}^T_{k,:} \boldsymbol{b}. \tag{3.41}$$

Thus

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}} = \begin{bmatrix} \partial \mathcal{L}/\partial x_1 \\ \partial \mathcal{L}/\partial x_2 \\ \cdots \\ \partial \mathcal{L}/\partial x_n \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}^T_{1,:} \boldsymbol{b} \\ \boldsymbol{A}^T_{2,:} \boldsymbol{b} \\ \cdots \\ \boldsymbol{A}^T_{n,:} \boldsymbol{b} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}^T_{1,:} \\ \boldsymbol{A}^T_{2,:} \\ \cdots \\ \boldsymbol{A}^T_{n,:} \end{bmatrix} \boldsymbol{b} = \boldsymbol{A}^T \boldsymbol{b}. \tag{3.42}$$

Notice that $\mathcal{L} = \boldsymbol{b}^T \boldsymbol{A}\boldsymbol{x} = \boldsymbol{x}^T \boldsymbol{A}^T \boldsymbol{b}$ by the commutativity of the inner product. Therefore we have

$$\frac{\partial}{\partial \boldsymbol{x}} \left(\boldsymbol{b}^T \boldsymbol{A}\boldsymbol{x} \right) = \frac{\partial}{\partial \boldsymbol{x}} \left(\boldsymbol{x}^T \boldsymbol{A}^T \boldsymbol{b} \right) = \boldsymbol{A}^T \boldsymbol{b}. \tag{3.43}$$

Example

Consider the minimization of ($\boldsymbol{A} \in \mathcal{R}^{m \times n}$)

$$\mathcal{L} = \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 \tag{3.44}$$

$$= \boldsymbol{x}^T \boldsymbol{A}^T \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}^T \boldsymbol{b} - \boldsymbol{b}^T \boldsymbol{A}\boldsymbol{x} - \boldsymbol{x}^T \boldsymbol{A}^T \boldsymbol{b} \tag{3.45}$$

We can use the results from the previous examples[3]

$$\frac{\partial}{\partial \boldsymbol{x}}\left(\boldsymbol{x}^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x}\right) = 2\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} \tag{3.46}$$

$$\frac{\partial}{\partial \boldsymbol{x}}\left(\boldsymbol{b}^T\boldsymbol{A}\boldsymbol{x}\right) = \boldsymbol{A}^T\boldsymbol{b} \tag{3.47}$$

$$\frac{\partial}{\partial \boldsymbol{x}}\left(\boldsymbol{x}^T\boldsymbol{A}^T\boldsymbol{b}\right) = \boldsymbol{A}^T\boldsymbol{b} \tag{3.48}$$

to get

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}} = 2\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} - 2\boldsymbol{A}^T\boldsymbol{b} = 2\boldsymbol{A}^T\left(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\right). \tag{3.49}$$

The optimality condition

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}} = 0 \tag{3.50}$$

gives the so-called *normal equations*

$$\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^T\boldsymbol{b} \tag{3.51}$$

which has the solution

$$\boldsymbol{x} = \left(\boldsymbol{A}^T\boldsymbol{A}\right)^{-1}\boldsymbol{A}^T\boldsymbol{b}, \tag{3.52}$$

provided $\left(\boldsymbol{A}^T\boldsymbol{A}\right)^{-1}$ exists. The matrix $\boldsymbol{A}^+ = \left(\boldsymbol{A}^T\boldsymbol{A}\right)^{-1}\boldsymbol{A}^T$ is called *Moore-Penrose pseudoinverse*. We used this result in the discussion of linear least squares (compare section 2.9.1).

3.1.3 Vector-valued derivatives: scalar input, vector output

The derivative of a vector-valued function with scalar input $\boldsymbol{f} : \mathcal{R} \to \mathcal{R}^n, x \to \boldsymbol{f}(x)$ is evaluated via component-wise differentiation

$$\frac{\partial \boldsymbol{f}}{\partial x} = \begin{bmatrix} \partial f_1/\partial x \\ \partial f_2/\partial x \\ \cdots \\ \partial f_n/\partial x \end{bmatrix}. \tag{3.53}$$

3.1.4 Matrix-valued derivatives: vector input, vector output

A vector-valued function with vector-valued input $\boldsymbol{f} : \mathcal{R}^n \to \mathcal{R}^m, \boldsymbol{x} \to \boldsymbol{f}(\boldsymbol{x})$ has the first-order Taylor approximation [31]

$$\boldsymbol{f}(\boldsymbol{x} + \Delta\boldsymbol{x}) \approx \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{J}(\boldsymbol{x})\Delta\boldsymbol{x}, \tag{3.54}$$

where

$$\boldsymbol{J}_{i,j} = \frac{\partial f_i}{\partial x_j} \tag{3.55}$$

is the *Jacobian matrix* ($\boldsymbol{J} \in \mathcal{R}^{m \times n}$).

[3] We use that $\boldsymbol{S} = \boldsymbol{A}^T\boldsymbol{A}$ is symmetric.

3.1.5 Matrix-valued derivatives: matrix input, scalar output

Given a scalar-valued function of matrix-valued input $f : \mathcal{R}^{m\times n} \to \mathcal{R}, \boldsymbol{X} \to f(\boldsymbol{X})$, we define the matrix-valued derivative

$$\frac{\partial f}{\partial \boldsymbol{X}} = \begin{bmatrix} \partial f/\partial X_{1,1} & \partial f/\partial X_{1,2} & \dots & \partial f/\partial X_{1,n} \\ \partial f/\partial X_{2,1} & \partial f/\partial X_{2,2} & \dots & \partial f/\partial X_{2,n} \\ \dots & \dots & \dots & \dots \\ \partial f/\partial X_{m,1} & \partial f/\partial X_{m,2} & \dots & \partial f/\partial X_{m,n} \end{bmatrix} \in \mathcal{R}^{m\times n}. \tag{3.56}$$

Notice that the dimensions of $\partial f/\partial \boldsymbol{X}$ are equal to the dimension of $\boldsymbol{X}$. This is referred to as *numerator layout*. We could have defined the derivative to have the same dimensions as $\boldsymbol{X}^T$, which would have resulted in the so-called *denominator layout*[4].

In the derivations that follow, we use the identity

$$\frac{\partial X_{i,j}}{\partial X_{k,l}} = \delta_{i,k}\delta_{j,l} \tag{3.57}$$

to find differentiation results. It follows from the properties of the Kronecker delta that $\delta_{i,k}\delta_{j,l}$ is non-zero only when $i = k$ and $j = l$. This is useful to simplify double sums. For instance,

$$\sum_i \sum_j \delta_{i,k}\delta_{j,l}A_{i,j} = A_{k,l}. \tag{3.58}$$

Example

Consider the least square cost function

$$\mathcal{L} = \sum_\alpha \|\boldsymbol{A}\boldsymbol{x}_\alpha - \boldsymbol{b}_\alpha\|_2^2, \tag{3.59}$$

where for simplicity we assume $\boldsymbol{A} \in \mathcal{R}^{n\times n}$, i.e. $\boldsymbol{A}$ is real-valued and square. In an optical context $\boldsymbol{A}$ could for example represent an incoherent optical system. Our goal would then be to characterize this optical system from known input and output data $\boldsymbol{x}_\alpha, \boldsymbol{b}_\alpha$ $(\alpha = 1, 2, ...)$. Knowing the intensity mapping relationship for a large number of input and output fields, we may infer the system matrix $\boldsymbol{A}$. To this end, expand Eq. 3.59

$$\mathcal{L} = \sum_\alpha (\boldsymbol{A}\boldsymbol{x}_\alpha - \boldsymbol{b}_\alpha)^T (\boldsymbol{A}\boldsymbol{x}_\alpha - \boldsymbol{b}_\alpha) \tag{3.60}$$

$$= \sum_\alpha \boldsymbol{x}_\alpha^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x}_\alpha + \boldsymbol{b}_\alpha^T\boldsymbol{b}_\alpha - \boldsymbol{b}_\alpha^T\boldsymbol{A}\boldsymbol{x}_\alpha - \boldsymbol{x}_\alpha^T\boldsymbol{A}^T\boldsymbol{b}_\alpha. \tag{3.61}$$

Let us tackle each of these terms individually. The first term is[5]

$$\boldsymbol{x}^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} = \boldsymbol{x}^T\boldsymbol{B}\boldsymbol{x} = \sum_i \sum_j x_i B_{i,j} x_j \tag{3.62}$$

[4] For more information, see https://en.wikipedia.org/wiki/Matrix_calculus.

[5] We can drop the index α for the moment.

with

$$B_{i,j} = \boldsymbol{A}_{i,:}^T \boldsymbol{A}_{:,j} = \sum_k A_{i,k}^T A_{k,j} = \sum_k A_{k,i} A_{k,j}, \tag{3.63}$$

so

$$\boldsymbol{x}^T \boldsymbol{A}^T \boldsymbol{A} \boldsymbol{x} = \sum_i \sum_j \sum_k x_i A_{k,i} A_{k,j} x_j. \tag{3.64}$$

From this we get the derivative with respect to a single matrix element

$$\frac{\partial}{\partial A_{l,m}} \left(\boldsymbol{x}^T \boldsymbol{A}^T \boldsymbol{A} \boldsymbol{x} \right) = \frac{\partial}{\partial A_{l,m}} \sum_i \sum_j \sum_k x_i A_{k,i} A_{k,j} x_j \tag{3.65}$$

$$= \sum_i \sum_j \sum_k x_i \frac{\partial A_{k,i}}{\partial A_{l,m}} A_{k,j} x_j + \sum_i \sum_j \sum_k x_i A_{k,i} \frac{\partial A_{k,j}}{\partial A_{l,m}} x_j \tag{3.66}$$

$$= \sum_i \sum_j \sum_k x_i \delta_{k,l} \delta_{i,m} A_{k,j} x_j + \sum_i \sum_j \sum_k x_i A_{k,i} \delta_{k,l} \delta_{j,m} x_j \tag{3.67}$$

$$= \sum_j A_{l,j} x_j x_m + \sum_i A_{l,i} x_i x_m \tag{3.68}$$

$$= 2 \sum_j A_{l,j} x_j x_m. \tag{3.69}$$

This can be reassembled in matrix form

$$\frac{\partial}{\partial \boldsymbol{A}} \left(\boldsymbol{x}^T \boldsymbol{A}^T \boldsymbol{A} \boldsymbol{x} \right) = 2 \boldsymbol{A} \boldsymbol{x} \boldsymbol{x}^T. \tag{3.70}$$

The next relevant term is

$$\frac{\partial}{\partial A_{k,l}} \left(\boldsymbol{b}^T \boldsymbol{A} \boldsymbol{x} \right) = \frac{\partial}{\partial A_{k,l}} \sum_i \sum_j b_i A_{i,j} x_j \tag{3.71}$$

$$= \sum_i \sum_j b_i \frac{\partial A_{i,j}}{\partial A_{k,l}} x_j \tag{3.72}$$

$$= \sum_i \sum_j b_i \delta_{i,k} \delta_{j,l} x_j \tag{3.73}$$

$$= b_k x_l, \tag{3.74}$$

which can be rewritten as

$$\frac{\partial}{\partial \boldsymbol{A}} \left(\boldsymbol{b}^T \boldsymbol{A} \boldsymbol{x} \right) = \boldsymbol{b} \boldsymbol{x}^T. \tag{3.75}$$

Because

$$\boldsymbol{b}^T \boldsymbol{A} \boldsymbol{x} = \text{some real scalar} = \left(\boldsymbol{b}^T \boldsymbol{A} \boldsymbol{x} \right)^T = \boldsymbol{x}^T \boldsymbol{A}^T \boldsymbol{b}, \tag{3.76}$$

we immediately see that

$$\frac{\partial}{\partial \boldsymbol{A}} \left(\boldsymbol{x}^T \boldsymbol{A}^T \boldsymbol{b} \right) = \boldsymbol{b} \boldsymbol{x}^T. \tag{3.77}$$

Putting everything together and re-using the α summation that we momentarily dropped above, we find the matrix-valued derivative to the original cost function

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{A}} = 2 \sum_{\alpha} \boldsymbol{A}\boldsymbol{x}_\alpha \boldsymbol{x}_\alpha^T - \boldsymbol{b}_\alpha \boldsymbol{x}_\alpha^T \tag{3.78}$$

Setting this to zero, we get the least square solution for $\boldsymbol{A}$,

$$\boldsymbol{A} = \left(\sum_{\alpha} \boldsymbol{b}_\alpha \boldsymbol{x}_\alpha^T \right) \left(\sum_{\alpha} \boldsymbol{x}_\alpha \boldsymbol{x}_\alpha^T \right)^{-1}, \tag{3.79}$$

provided the inverse matrix factor exists. Each $\boldsymbol{x}_\alpha \boldsymbol{x}_\alpha^T$ and $\boldsymbol{b}_\alpha \boldsymbol{x}_\alpha^T$ are rank-1 matrices. Summing up N linearly independent rank-1 matrices gives a rank-N matrix. Thus for the inverse to be computed, we need at least $N \geq \min(m, n) \geq \text{rank}(\boldsymbol{A})$ measurements.

We can write the solution in Eq. 3.79 in a neater way. First note that the cost in Eq. 3.59 can be formulated as See exercise below.

$$\mathcal{L} = \|\boldsymbol{A}\boldsymbol{X} - \boldsymbol{B}\|_F^2 \tag{3.80}$$

where

$$\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots], \; \boldsymbol{B} = [\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots] \tag{3.81}$$

and $\|\ldots\|_F$ is the Frobenius norm. Then it follows from matrix block multiplication that

$$\boldsymbol{X}\boldsymbol{X}^T = \begin{bmatrix} | & | & | \\ \boldsymbol{x}_1 & \boldsymbol{x}_2 & \ldots \\ | & | & | \end{bmatrix} \begin{bmatrix} - & \boldsymbol{x}_1^T & - \\ - & \boldsymbol{x}_2^T & - \\ - & \ldots & - \end{bmatrix} = \sum_{\alpha} \boldsymbol{x}_\alpha \boldsymbol{x}_\alpha^T, \tag{3.82}$$

and

$$\boldsymbol{B}\boldsymbol{X}^T = \begin{bmatrix} | & | & | \\ \boldsymbol{b}_1 & \boldsymbol{b}_2 & \ldots \\ | & | & | \end{bmatrix} \begin{bmatrix} - & \boldsymbol{x}_1^T & - \\ - & \boldsymbol{x}_2^T & - \\ - & \ldots & - \end{bmatrix} = \sum_{\alpha} \boldsymbol{b}_\alpha \boldsymbol{x}_\alpha^T. \tag{3.83}$$

So we have

$$\underbrace{\frac{\partial \mathcal{L}}{\partial \boldsymbol{A}}}_{M \times N} = 2 \left(\underbrace{\boldsymbol{A}}_{M \times N} \underbrace{\boldsymbol{X}}_{N \times P} - \underbrace{\boldsymbol{B}}_{M \times P} \right) \underbrace{\boldsymbol{X}^T}_{P \times N} \tag{3.84}$$

with the least squares solution

$$\boldsymbol{A} = \boldsymbol{B}\boldsymbol{X}^T \left(\boldsymbol{X}\boldsymbol{X}^T \right)^{-1}, \tag{3.85}$$

provided $\left(\boldsymbol{X}\boldsymbol{X}^T \right)^{-1}$ exists. The dimensions indicated below Eq. 3.84 confirm our use of numerator layout, resulting in a derivative with the same dimensions as the original matrix.

An example is given in the following code, which produces a 9×9 Toeplitz matrix and 9 random measurements (with additional noise). The input vectors are collected along the columns of the matrix $\boldsymbol{X}$, while the output vectors are collected along the columns of $\boldsymbol{B}$. Both the ground truth matrix and the least square solution from the measurements are shown in Fig. 3.2.

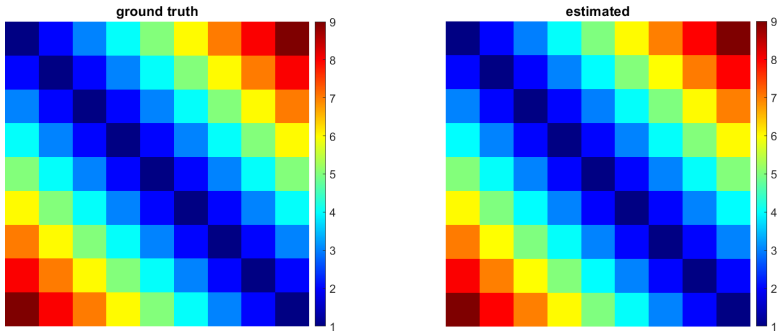


Figure 3.2: Output from estimateMatrix.m. (Left) ground truth Toeplitz matrix. (Right) Least square estimate from 9 random measurement.

```
A = toeplitz(1:9);

rng(0)
X = rand(9);
noise = 1e-1*randn(9);
B = A * X + noise;
SNR = 1/numel(A) * sum(abs(B(:)).^2) / sum(abs(noise(:)).^2);

A_sol = B*X'/(X*X');

figure(1)
subplot(1,2,1)
imagesc(A)
axis image off
colorbar
title('ground truth')

subplot(1,2,2)
imagesc(A_sol)
axis image off, colormap jet
colorbar
title('estimated')
```

Listing 3.2: estimate matrix (Matlab)

If you feel matrix-derivatives are complicated and hard to memorize, we encourage you to take a look at http://www.matrixcalculus.org/. This website allows you to compute real-valued derivatives with respect to a matrix. If on the other hand you feel like you just encountered your first love, we encourage you to take a look at the matrix cookbook (add reference).

Exercise: Show that Exercise

$$\sum_{\alpha} \|\boldsymbol{A}\boldsymbol{x}_{\alpha} - \boldsymbol{b}_{\alpha}\|_2^2 = \|\boldsymbol{A}\boldsymbol{X} - \boldsymbol{B}\|_F^2 . \tag{3.86}$$

Exercise: Show that Exercise

$$\frac{\partial}{\partial \boldsymbol{X}} \|\boldsymbol{A}\boldsymbol{X} - \boldsymbol{B}\|_F^2 = 2\boldsymbol{A}^T (\boldsymbol{A}\boldsymbol{X} - \boldsymbol{B}) . \tag{3.87}$$

Exercise: Show that Exercise

$$\frac{\partial}{\partial \boldsymbol{X}} \|\boldsymbol{A} \odot \boldsymbol{X} - \boldsymbol{B}\|_F^2 = 2 (\boldsymbol{A} \odot \boldsymbol{X} - \boldsymbol{B}) \odot \boldsymbol{X}, \tag{3.88}$$

where $\odot$ denotes element-wise (Hadamard) multiplication.

3.1.6 Matrix-valued derivatives: scalar input, matrix output

We can define a function, which takes a scalar as input and returns a matrix, $\boldsymbol{F} : \mathcal{R} \rightarrow \mathcal{R}^{m \times n}, x \rightarrow \boldsymbol{F}(x)$ .This is illustrated here by a simple example. The matrix

$$\boldsymbol{F}(x) = \begin{bmatrix} \cos^2 x & \cos x \sin x \\ \cos x \sin x & \sin^2 x \end{bmatrix} . \tag{3.89}$$

may be used to represent the passage of coherent light through a linear polarizer (rotated through an angle x). The derivative with respect to the parameter x is given by

$$\frac{\partial \boldsymbol{F}(x)}{\partial x} = \begin{bmatrix} -2\cos x \sin x & \cos^2 x - \sin^2 x \\ \cos^2 x - \sin^2 x & 2\cos x \sin x \end{bmatrix} , \tag{3.90}$$

i.e. each matrix element is individually differentiated according to the scalar differentiation rule.

3.2 Wirtinger derivatives

In the previous section on real-valued derivatives we have examined 6 types of derivatives that we will frequently come across when solving optimization problems in computational optics. The formalism in the previous section can in principle be used to also solve problems with complex-valued input, simply by separately optimizing for real and imaginary part of each input parameter. However, it turns out that this approach is cumbersome. A more efficient method is to use Wirtinger derivatives, discussed below. Unlike in the previous section, we restrict ourselves to Wirtinger derivatives of functions with scalar output:

$$\begin{matrix} \frac{\partial f}{\partial z^*} & \cdots & \frac{\partial f}{\partial z_N^*} \\ \frac{\partial f}{\partial \boldsymbol{z}^*} & \cdots & \\ \frac{\partial f}{\partial \boldsymbol{Z}^*} & & \end{matrix}$$

3.2.1 Wirtinger derivatives: scalar input, scalar output

We will illustrate the idea of Wirtinger derivatives by example. Suppose we have a real-valued scalar function $f : \mathcal{C} \to \mathcal{R}, z \to f(z)$ with complex-valued input. Intuitively, we could split up the complex input variable z into real and imaginary part

$$z = x + i \cdot y. \tag{3.91}$$

With this we may redefine our original function in terms of a vector-valued[6] real input $f : \mathcal{R}^2 \to \mathcal{R}, \boldsymbol{x} \to f(\boldsymbol{x})$. Often our goal is to optimize a function f with respect to complex-valued input. The necessary condition for optimality would then read

$$\nabla f = 0. \tag{3.92}$$

This would result in a set of two equations with solution $(x, y)^T$. Alternatively we can use Wirtinger derivatives and set them to zero,

$$\frac{\partial f}{\partial z^*} = 0. \tag{3.93}$$

This notation treats z and its conjugate z^* as independent and computes the partial derivative with respect to the latter. We will first illustrate the operational part of this calculation by example and then dive a bit deeper into its meaning.

Example: phase and amplitude synchronization

Suppose we are searching for a complex scalar z to be multiplied with a complex-valued vector $\boldsymbol{a}$, the result of which should be as close as possible to another complex-valued vector $\boldsymbol{b}$. In contrast to the phase synchronization example in subsection 3.1.1, z now allows for a scaling of the length of $\boldsymbol{a}$ in addition to the phase shifting property previously considered. Similar to the calculation in subsection 3.1.1,

$$\mathcal{L} = \|z\boldsymbol{a} - \boldsymbol{b}\|_2^2 \tag{3.94}$$

$$= z \cdot z^* \|\boldsymbol{a}\|_2^2 + \|b\|_2^2 - z^* \boldsymbol{a}^\dagger \boldsymbol{b} - z \boldsymbol{b}^\dagger \boldsymbol{a}. \tag{3.95}$$

Solution 1:

Splitting z into its real and imaginary part ($z = x + i \cdot y$), we find

$$\mathcal{L} = \left(x^2 + y^2\right) \|\boldsymbol{a}\|_2^2 + \|b\|_2^2 - (x + i \cdot y)^* \boldsymbol{a}^\dagger \boldsymbol{b} - (x + i \cdot y) \boldsymbol{b}^\dagger \boldsymbol{a} \tag{3.96}$$

$$= x^2 \|\boldsymbol{a}\|_2^2 + y^2 \|\boldsymbol{a}\|_2^2 + \|b\|_2^2 - x\boldsymbol{a}^\dagger \boldsymbol{b} + i \cdot y\boldsymbol{a}^\dagger \boldsymbol{b} - x\boldsymbol{b}^\dagger \boldsymbol{a} - i \cdot y\boldsymbol{b}^\dagger \boldsymbol{a}. \tag{3.97}$$

Set the gradient to zero

$$\nabla \mathcal{L} = \begin{bmatrix} \partial \mathcal{L}/\partial x_1 \\ \partial \mathcal{L}/\partial x_2 \end{bmatrix} = \begin{bmatrix} 2x \|\boldsymbol{a}\|_2^2 - \boldsymbol{a}^\dagger \boldsymbol{b} - \boldsymbol{b}^\dagger \boldsymbol{a} \\ 2y \|\boldsymbol{a}\|_2^2 + i \cdot \left(\boldsymbol{a}^\dagger \boldsymbol{b} - \boldsymbol{b}^\dagger \boldsymbol{a}\right) \end{bmatrix} = 0 \tag{3.98}$$

[6] In the variable in the vector-valued input $\boldsymbol{x}$ we hide the not so elegant notation $\boldsymbol{x} = [x, y]^T$. We could have used the notation $z = x_1 + i \cdot x_2$, which unfortunately generalizes even less elegantly when z is turned into a vector $\boldsymbol{z} = [z_1, z_2, \ldots]^T$, which is what is going to happen in subsection 3.2.3.

to get the solution

$$x_{\text{sol}} = \frac{1}{2}\frac{\boldsymbol{a}^\dagger \boldsymbol{b} + \boldsymbol{b}^\dagger \boldsymbol{a}}{\|\boldsymbol{a}\|_2^2} \tag{3.99}$$

$$y_{\text{sol}} = \frac{i}{2}\frac{-\boldsymbol{a}^\dagger \boldsymbol{b} + \boldsymbol{b}^\dagger \boldsymbol{a}}{\|\boldsymbol{a}\|_2^2}. \tag{3.100}$$

Thus

$$z_{\text{sol}} = x_{\text{sol}} + i \cdot y_{\text{sol}} \tag{3.101}$$

$$= \frac{1}{2}\frac{\boldsymbol{a}^\dagger \boldsymbol{b} + \boldsymbol{b}^\dagger \boldsymbol{a}}{\|\boldsymbol{a}\|_2^2} + i \cdot \frac{i}{2}\frac{-\boldsymbol{a}^\dagger \boldsymbol{b} + \boldsymbol{b}^\dagger \boldsymbol{a}}{\|\boldsymbol{a}\|_2^2} \tag{3.102}$$

$$= \frac{\boldsymbol{a}^\dagger \boldsymbol{b}}{\|\boldsymbol{a}\|_2^2}. \tag{3.103}$$

Solution 2:

Let us see what Wirtinger derivatives can do for us:

$$\frac{\partial \mathcal{L}}{\partial z^*} = z\,\|\boldsymbol{a}\|_2^2 - \boldsymbol{a}^\dagger \boldsymbol{b} = 0, \tag{3.104}$$

so

$$z = \frac{\boldsymbol{a}^\dagger \boldsymbol{b}}{\|\boldsymbol{a}\|_2^2}. \tag{3.105}$$

Done. Compare this to the previous computation. Wirtinger derivatives can reduce computations quite dramatically!

3.2.2 Wirtinger derivatives: theoretical background

It is important to distinguish *Wirtinger derivatives* from *complex derivatives*. Typical courses on complex analysis deal with *holomorphic functions* [16, 26]

$$f(x,y) = u(x,y) + i \cdot v(x,y), \tag{3.106}$$

whose real and imaginary parts satisfy the Cauchy-Riemann equations

$$\frac{\partial u(x,y)}{\partial x} = \frac{\partial v(x,y)}{\partial y}, \quad \frac{\partial v(x,y)}{\partial x} = -\frac{\partial u(x,y)}{\partial y}. \tag{3.107}$$

Only if this set of partial differential equations is satisfied, then the function f is differentiable with respect to the complex variable z, i.e.

$$f'(z) = \frac{\partial f(z)}{\partial z}, \tag{3.108}$$

where $z = x + i \cdot y$, exists. Notice that the Cauchy-Riemann equations pose a very special requirement and many beautiful properties can be derived from it (e.g. residue theorem,

Laurent series, ...). However, the Cauchy-Riemann equations are typically not satisfied for real-valued functions f (constant functions being a trivial exception).

Wirtinger derivatives are built on a different theoretical foundation. There is no requirement for the Cauchy-Riemann equations to be satisfied. Instead, the only requirement for a Wirtinger derivative to exist is that the real and imaginary parts of f are both differentiable with respect to the real-valued variables x and y. This is typically all we need when it comes to inverse problems in optics, where the quantity of interest is oftentimes complex-valued (for instance a coherent wave field, a coherent transfer function of an optical system, or an object exhibiting phase contrast). In fact, the Wirtinger derivative may formally be defined through real-valued derivatives [9],

$$\frac{\partial f}{\partial z^*} = \frac{1}{2}\left(\frac{\partial f}{\partial x} + i \cdot \frac{\partial f}{\partial y}\right). \tag{3.109}$$

In what follows we assume f to be real-valued, which is the only case of interest throughout this text. For us f is typically a cost function, which we often denote by f or $\mathcal{L}$. We note that the Wirtinger calculus can also be used for the more general case of complex-valued f, as described in [26].

Example

In the amplitude- and phase-synchronization problem above (Eq. 3.94), we found the gradient of the cost function $\mathcal{L}$,

$$\nabla\mathcal{L} = \begin{bmatrix} \partial\mathcal{L}/\partial x \\ \partial\mathcal{L}/\partial y \end{bmatrix} = \begin{bmatrix} 2x\,\|\boldsymbol{a}\|_2^2 - \boldsymbol{a}^\dagger\boldsymbol{b} - \boldsymbol{b}^\dagger\boldsymbol{a} \\ 2y\,\|\boldsymbol{a}\|_2^2 + i\cdot\left(\boldsymbol{a}^\dagger\boldsymbol{b} - \boldsymbol{b}^\dagger\boldsymbol{a}\right) \end{bmatrix}. \tag{3.110}$$

Using Eq. 3.109, we find the Wirtinger derivative

$$\frac{\partial\mathcal{L}}{\partial z^*} = \frac{1}{2}\left(\frac{\partial\mathcal{L}}{\partial x} + i\cdot\frac{\partial\mathcal{L}}{\partial y}\right) \tag{3.111}$$

$$= x\,\|\boldsymbol{a}\|_2^2 - \frac{1}{2}\boldsymbol{a}^\dagger\boldsymbol{b} - \frac{1}{2}\boldsymbol{b}^\dagger\boldsymbol{a} + i\cdot\left(y\,\|\boldsymbol{a}\|_2^2 + \frac{i}{2}\cdot\left(\boldsymbol{a}^\dagger\boldsymbol{b} - \boldsymbol{b}^\dagger\boldsymbol{a}\right)\right) \tag{3.112}$$

$$= (x + i\cdot y)\cdot\|\boldsymbol{a}\|_2^2 - \boldsymbol{a}^\dagger\boldsymbol{b} \tag{3.113}$$

$$= z\cdot\|\boldsymbol{a}\|_2^2 - \boldsymbol{a}^\dagger\boldsymbol{b} \tag{3.114}$$

Setting this to zero, we confirm once more that

$$z = \frac{\boldsymbol{a}^\dagger\boldsymbol{b}}{\|\boldsymbol{a}\|_2^2}. \tag{3.115}$$

From this example, we gain insight how the Wirtinger derivative operates. Assuming that $\mathcal{L}$ is real-valued, which the cost functions that we are interested in always are, both $\partial\mathcal{L}/\partial x$ and $\partial\mathcal{L}/\partial y$ are real-valued. Thus Eq. 3.109 implicitly rearranges the components of the vector-valued (real) gradient

$$\nabla\mathcal{L} = \begin{bmatrix} \partial\mathcal{L}/\partial x \\ \partial\mathcal{L}/\partial y \end{bmatrix} \tag{3.116}$$

into the real and imaginary parts of the scalar-valued (complex) Wirtinger derivative

$$\frac{\partial \mathcal{L}}{\partial z^*} = \frac{1}{2}\left(\underbrace{\frac{\partial \mathcal{L}}{\partial x}}_{\text{real}} + i \cdot \underbrace{\frac{\partial \mathcal{L}}{\partial y}}_{\text{imag}} \right). \tag{3.117}$$

In this way the real and imaginary parts, which hold information about the components of the gradient, do not get mixed up. The (necessary) optimality condition $\nabla \mathcal{L} = 0$ directly translates into $\partial \mathcal{L}/\partial z^* = 0$, since a complex scalar is zero if and only if both its real and imaginary parts vanish. We may also write the Wirtinger derivative in vector notation

$$\frac{\partial \mathcal{L}}{\partial z^*} = \frac{1}{2}\begin{bmatrix} 1 & i \end{bmatrix} \nabla \mathcal{L} = \frac{1}{2}\begin{bmatrix} 1 & i \end{bmatrix} \begin{bmatrix} \partial \mathcal{L}/\partial x \\ \partial \mathcal{L}/\partial y \end{bmatrix}, \tag{3.118}$$

which is another way to see that $\nabla \mathcal{L} = 0$ directly implies $\partial \mathcal{L}/\partial z^* = 0$.

But at this point the reader might be skeptical whether or not the (possibly different) algebraic operations $\partial \mathcal{L}/\partial z^*$[7] and $1/2 \cdot (\partial \mathcal{L}/\partial x + \partial \mathcal{L}/\partial y)$[8] always evaluate to the same result? This is case, for it is an immediate consequence of the chain rule. Rewrite a given function of two real-valued input variables, $h(x, y)$, in terms of z and z^*

$$h(x, y) = h\left(\frac{1}{2}[z + z^*], \frac{1}{2i}[z - z^*] \right) = f(z, z^*), \tag{3.119}$$

then

$$\frac{\partial f}{\partial z^*} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial z^*} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial z^*} \tag{3.120}$$

$$= \frac{1}{2}\frac{\partial f}{\partial x} - \frac{1}{2i}\frac{\partial f}{\partial y} \tag{3.121}$$

$$= \frac{1}{2}\left(\frac{\partial f}{\partial x} + i \cdot \frac{\partial f}{\partial y} \right). \tag{3.122}$$

Finally let us point out that the Wirtinger derivative assumes z and z^* to be independent. A somewhat more precise but lengthy notation would be

$$\left. \frac{\partial f(z, z^*)}{\partial z^*} \right|_{z=\text{constant}=c} = \frac{\partial f_c(z^*)}{\partial z^*}. \tag{3.123}$$

This means that when one embeds the function $f : \mathcal{R}^2 \to \mathcal{R}, (x, y) \to f(x, y)$ into a new space $h : \mathcal{C}^2 \to \mathcal{R}, (z, z^*) \to h(z, z^*)$ (compare Eq. 3.119), then the components z and z^* can be varied independently. In particular, we can fix z to some constant value c and independently vary z^*. One can then formally prove that the optimality condition for h, namely $\nabla h = 0$, translates into an new optimality condition $\partial f/\partial z^* = 0$ for f, as shown by Brandwood [9].

[7] The symbol $\partial \mathcal{L}/\partial z^*$ treats z^* as an independent variable and computes the partial derivative of $\mathcal{L}$ with respect to z^*.

[8] The symbol $\partial \mathcal{L}/\partial x$ treats x as an independent variable and computes the partial derivative of $\mathcal{L}$ with respect to x; and similarly for $\partial \mathcal{L}/\partial y$.

Notice also that in Eq. 3.119 we may just as much compute the Wirtinger derivative with respect to z, keeping z^* constant, to get

$$\frac{\partial f}{\partial z} = \frac{1}{2}\left(\frac{\partial f}{\partial x} - i \cdot \frac{\partial f}{\partial y}\right), \tag{3.124}$$

which for real-valued f is simply a complex conjugate of Eq. 3.109,

$$\frac{\partial f}{\partial z^*} = \left(\frac{\partial f}{\partial z}\right)^*. \tag{3.125}$$

We note that the Wirtinger derivative of a composition of functions is subject to the following chain rule

$$\frac{\partial f\left(g\left(z^*\right), g^*\left(z^*\right)\right)}{\partial z^*} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial z^*} + \frac{\partial f}{\partial g^*}\frac{\partial g^*}{\partial z^*}. \tag{3.126}$$

More information of Wirtinger derivatives may be found in the excellent review by Kreutz-Delgado [26][9] and the article by Brandwood [9].

3.2.3 Wirtinger derivatives: vector input, scalar output

Given a scalar function of vector-valued (complex) input $f : \mathcal{C}^n \to \mathcal{R}, z \to f\left(\boldsymbol{z}\right)$, where $\boldsymbol{z} = \left[z_1, z_2, \ldots, z_n\right]^T$, the vector-valued Wirtinger derivative is given by

$$\frac{\partial f}{\partial \boldsymbol{z}^*} = \begin{bmatrix} \partial f/\partial z_1^* \\ \partial f/\partial z_2^* \\ \cdots \\ \partial f/\partial z_n^* \end{bmatrix}. \tag{3.127}$$

Example

Let us reconsider the quadratic form

$$\mathcal{L} = \boldsymbol{z}^\dagger \boldsymbol{A}\boldsymbol{z} \tag{3.128}$$

$$= \sum_i \sum_j z_i^* A_{i,j} z_j, \tag{3.129}$$

[9] https://arxiv.org/abs/0906.4835

this time with complex-valued quantities involved $\boldsymbol{z} \in \mathcal{C}^n, \boldsymbol{A} \in \mathcal{C}^{n\times n}$. Then[10]

$$\frac{\partial \mathcal{L}}{\partial z_k^*} = \frac{\partial}{\partial z_k^*} \sum_i \sum_j z_i^* A_{i,j} z_j \tag{3.130}$$

$$= \sum_i \sum_j \frac{\partial z_i^*}{\partial z_k^*} A_{i,j} z_j \tag{3.131}$$

$$= \sum_i \sum_j \delta_{i,k} A_{i,j} z_j \tag{3.132}$$

$$= \sum_j A_{k,j} z_j \tag{3.133}$$

$$= \boldsymbol{A}_{k,:}\boldsymbol{z} \tag{3.134}$$

and so

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^*} = \begin{bmatrix} \partial\mathcal{L}/\partial z_1^* \\ \partial\mathcal{L}/\partial z_2^* \\ \cdots \\ \partial\mathcal{L}/\partial z_n^* \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_{1,:}\boldsymbol{z} \\ \boldsymbol{A}_{2,:}\boldsymbol{z} \\ \cdots \\ \boldsymbol{A}_{n,:}\boldsymbol{z} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_{1,:} \\ \boldsymbol{A}_{2,:} \\ \cdots \\ \boldsymbol{A}_{n,:} \end{bmatrix} \boldsymbol{z} = \boldsymbol{A}\boldsymbol{z}. \tag{3.135}$$

Notice the difference in the formulae for real- and complex valued innerproducts

$$\tfrac{\partial}{\partial \boldsymbol{x}} \left(\boldsymbol{x}^T \boldsymbol{A}\boldsymbol{x}\right) = \left(\boldsymbol{A}^T + \boldsymbol{A}\right)\boldsymbol{x} \quad \text{(real-valued gradient)} \tag{3.136}$$

$$\tfrac{\partial}{\partial \boldsymbol{z}^*} \left(\boldsymbol{z}^\dagger \boldsymbol{A}\boldsymbol{z}\right) = \boldsymbol{A}\boldsymbol{z} \quad \text{(Wirtinger gradient)} \ . \tag{3.137}$$

This difference comes essentially from the observation that $\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}$ is second-order in $\boldsymbol{x}$ while only first-order in the independent variables $\boldsymbol{z}$ and $\boldsymbol{z}^*$.

Example

Let us consider the following linear form ($\boldsymbol{A} \in \mathcal{C}^{m\times n}$; $\boldsymbol{b} \in \mathcal{C}^{m\times 1}, \boldsymbol{z} \in \mathcal{C}^{n\times 1}$)

$$\mathcal{L}_1 = \boldsymbol{b}^\dagger \boldsymbol{A}\boldsymbol{z} \tag{3.138}$$

$$= \boldsymbol{b}^\dagger \sum_{j=1}^{n} \boldsymbol{A}_{:,j} z_j \tag{3.139}$$

$$= \sum_{i=1}^{m}\sum_{j=1}^{n} b_i^* A_{i,j} z_j \tag{3.140}$$

The Wirtinger gradient is then given by

$$\frac{\partial \mathcal{L}_1}{\partial z_k^*} = \frac{\partial}{\partial z_k^*} \sum_{i=1}^{m}\sum_{j=1}^{n} b_i^* A_{i,j} z_j \tag{3.141}$$

$$= \sum_{i=1}^{m}\sum_{j=1}^{n} b_i^* A_{i,j} \frac{\partial z_j}{\partial z_k^*} \tag{3.142}$$

$$= 0, \tag{3.143}$$

[10] Recall that Wirtinger derivatives treat z and z^* as independent variables, so we are not using the product rule to what appears to be a quadratic form at first sight.

where we used that $\partial z_j / \partial z_k^* = 0$.

Now consider the conjugate transpose of $\mathcal{L}_1$

$$\mathcal{L}_2 = \mathcal{L}_1^* = \boldsymbol{z}^\dagger \boldsymbol{A}^\dagger \boldsymbol{b} \tag{3.144}$$

$$= \boldsymbol{z}^\dagger \sum_{j=1}^{m} \boldsymbol{A}^\dagger_{:,j} b_j \tag{3.145}$$

$$= \sum_{i=1}^{n} \sum_{j=1}^{m} z_i^* \boldsymbol{A}^\dagger_{i,j} b_j \tag{3.146}$$

With this we have

$$\frac{\partial \mathcal{L}_2}{\partial z_k^*} = \frac{\partial}{\partial z_k^*} \sum_{i=1}^{n} \sum_{j=1}^{m} z_i^* \boldsymbol{A}^\dagger_{i,j} b_j \tag{3.147}$$

$$= \sum_{i=1}^{n} \sum_{j=1}^{m} \frac{\partial z_i^*}{\partial z_k^*} \boldsymbol{A}^\dagger_{i,j} b_j \tag{3.148}$$

$$= \sum_{i=1}^{n} \sum_{j=1}^{m} \delta_{i,k} \boldsymbol{A}^\dagger_{i,j} b_j \tag{3.149}$$

$$= \sum_{j=1}^{m} \boldsymbol{A}^\dagger_{k,j} b_j \tag{3.150}$$

$$= \boldsymbol{A}^\dagger_{k,:} \boldsymbol{b}, \tag{3.151}$$

so

$$\frac{\partial \mathcal{L}_2}{\partial \boldsymbol{z}^*} = \boldsymbol{A}^\dagger \boldsymbol{b}. \tag{3.152}$$

The reader can confirm that in order to get the latter result we could have also computed

$$\frac{\partial \mathcal{L}_2}{\partial \boldsymbol{z}^*} = \left(\frac{\partial \mathcal{L}_1}{\partial \boldsymbol{z}} \right)^* . \tag{3.153}$$

Notice again the difference between real-valued and Wirtinger gradients,

$$\tfrac{\partial}{\partial \boldsymbol{x}} \left(\boldsymbol{b}^T \boldsymbol{A} \boldsymbol{x} \right) = \tfrac{\partial}{\partial \boldsymbol{x}} \left(\boldsymbol{x}^T \boldsymbol{A}^T \boldsymbol{b} \right) = \boldsymbol{A}^T \boldsymbol{b} \quad \text{(real-valued gradient)} \tag{3.154}$$

$$\tfrac{\partial}{\partial \boldsymbol{z}^*} \left(\boldsymbol{b}^\dagger \boldsymbol{A} \boldsymbol{z} \right) = 0, \; \tfrac{\partial}{\partial \boldsymbol{z}^*} \left(\boldsymbol{z}^\dagger \boldsymbol{A}^\dagger \boldsymbol{b} \right) = \boldsymbol{A}^\dagger \boldsymbol{b} \quad \text{(Wirtinger gradient)} \; . \tag{3.155}$$

Example

Since some of our rules familiar from real-valued gradients do not apply to Wirtinger gradients we feel the urge to re-evaluate the least squares problem,

$$\mathcal{L} = \| \boldsymbol{A} \boldsymbol{z} - \boldsymbol{b} \|_2^2 \tag{3.156}$$

$$= \boldsymbol{z}^\dagger \boldsymbol{A}^\dagger \boldsymbol{A} \boldsymbol{z} + \boldsymbol{b}^\dagger \boldsymbol{b} - \boldsymbol{b}^\dagger \boldsymbol{A} \boldsymbol{z} - \boldsymbol{z}^\dagger \boldsymbol{A}^\dagger \boldsymbol{b}, \tag{3.157}$$

this time around with complex-valued quantities $\boldsymbol{A} \in \mathcal{C}^{n\times n}; \boldsymbol{b}, \boldsymbol{z} \in \mathcal{C}^{n\times 1}$. From the previous examples, we find

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^*} = \boldsymbol{A}^\dagger \boldsymbol{A}\boldsymbol{z} - \boldsymbol{A}^\dagger \boldsymbol{b}. \tag{3.158}$$

Surprisingly, despite the significant differences in real-valued and Wirtinger gradients, the optimality conditions have remained (almost) the same:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^*} = \boldsymbol{A}^\dagger \boldsymbol{A}\boldsymbol{z} - \boldsymbol{A}^\dagger \boldsymbol{b} = 0 \tag{3.159}$$

leads to the (complex-valued) normal equations

$$\boldsymbol{A}^\dagger \boldsymbol{A}\boldsymbol{z} = \boldsymbol{A}^\dagger \boldsymbol{b} \tag{3.160}$$

and to the solution

$$\boxed{\boldsymbol{z} = \left(\boldsymbol{A}^\dagger \boldsymbol{A}\right)^{-1} \boldsymbol{A}^\dagger \boldsymbol{b},} \tag{3.161}$$

provided that $\left(\boldsymbol{A}^\dagger \boldsymbol{A}\right)^{-1}$ exists.

3.2.4 Wirtinger derivatives: matrix input, scalar output

Given a (real) scalar-valued function of (complex) matrix-valued input, $f : \mathcal{C}^{m\times n} \to \mathcal{R}, \boldsymbol{X} \to f(\boldsymbol{X})$, we define the matrix-valued Wirtinger derivative

$$\frac{\partial f}{\partial \boldsymbol{X}^*} = \begin{bmatrix} \partial f/\partial X^*_{1,1} & \partial f/\partial X^*_{1,2} & \dots & \partial f/\partial X^*_{1,n} \\ \partial f/\partial X^*_{2,1} & \partial f/\partial X^*_{2,2} & \dots & \partial f/\partial X^*_{2,n} \\ \dots & \dots & \dots & \dots \\ \partial f/\partial X^*_{m,1} & \partial f/\partial X^*_{m,2} & \dots & \partial f/\partial X^*_{m,n} \end{bmatrix} \in \mathcal{C}^{m\times n}. \tag{3.162}$$

This convention is again numerator layout, since the derivative $\partial f/\partial \boldsymbol{X}^*$ has equal dimensions to the input $\boldsymbol{X}$.

We are going to use that

$$\frac{\partial X_{i,j}}{\partial X^*_{k,l}} = 0, \; \frac{\partial X^*_{i,j}}{\partial X^*_{k,l}} = \delta_{i,k}\delta_{j,l}. \tag{3.163}$$

Example

Suppose we are given $\boldsymbol{A} \in \mathcal{C}^{M\times N}, \boldsymbol{B} \in \mathcal{C}^{M\times P}$ and $\boldsymbol{X} \in \mathcal{C}^{N\times P}$. We would like to solve the least squares problem

$$\mathcal{L} = \|\boldsymbol{A}\boldsymbol{X} - \boldsymbol{B}\|_F^2$$

with respect to $\boldsymbol{X}$. Let us first expand the scalar cost function in terms of the elements of each matrix involved. Define $\boldsymbol{C} = \boldsymbol{A}\boldsymbol{X}$, implying $C_{i,l} = \sum_j A_{i,j}X_{j,l}$, then

$$\mathcal{L} = \|\boldsymbol{C} - \boldsymbol{B}\|_F^2 \tag{3.164}$$

$$= \sum_i \sum_l |C_{i,l} - B_{i,l}|^2 \tag{3.165}$$

$$= \sum_i \sum_l \left| \sum_j A_{i,j}X_{j,l} - B_{i,l} \right|^2 \tag{3.166}$$

$$= \sum_i \sum_l \left(\sum_j A_{i,j}X_{j,l} - B_{i,l} \right)^* \left(\sum_k A_{i,k}X_{k,l} - B_{i,l} \right) \tag{3.167}$$

$$= \sum_i \sum_j \sum_k \sum_l \left(A_{i,j}X_{j,l} - B_{i,l} \right)^* \left(A_{i,k}X_{k,l} - B_{i,l} \right) \tag{3.168}$$

$$= \sum_i \sum_j \sum_k \sum_l \left(A_{i,j}^* X_{j,l}^* - B_{i,l}^* \right) \left(A_{i,k}X_{k,l} - B_{i,l} \right) \tag{3.169}$$

$$= \sum_{i,j,k,l} A_{i,j}^* X_{j,l}^* A_{i,k}X_{k,l} - A_{i,j}^* X_{j,l}^* B_{i,l} - B_{i,l}^* A_{i,k}X_{k,l} + B_{i,l}^* B_{i,l}. \tag{3.170}$$

Then

$$\frac{\partial \mathcal{L}}{\partial X_{m,n}^*} = \sum_{i,j,k,l} A_{i,j}^* \delta_{j,m}\delta_{l,n} A_{i,k}X_{k,l} - A_{i,j}^* \delta_{j,m}\delta_{l,n} B_{i,l} \tag{3.171}$$

$$= \sum_{i,k} A_{i,m}^* A_{i,k}X_{k,n} - A_{i,m}^* B_{i,n} \tag{3.172}$$

$$= \sum_{i,k} A_{i,m}^* \left(A_{i,k}X_{k,n} - B_{i,n} \right) \tag{3.173}$$

$$= \sum_i A_{i,m}^* \sum_k A_{i,k}X_{k,n} - B_{i,n}, \tag{3.174}$$

which can be rewritten in matrix form,

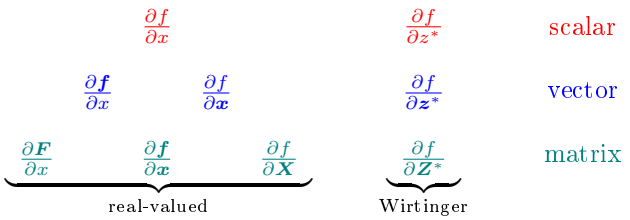
$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{X}^*} = \boldsymbol{A}^\dagger \left(\boldsymbol{A}\boldsymbol{X} - \boldsymbol{B} \right). \tag{3.175}$$

Notice the similarity to the real matrix-valued gradient in Eq. 3.87.

Admittedly, taking care of all the indices in the derivation above is tedious. But imagine we had omitted Wirtinger derivatives and split the whole problem into real and imaginary parts, seperately differentiated via real matrix-valued derivatives: such a derivation would have taken several pages. We hope you see once more the power of Wirtinger derivatives.

3.2.5 Overview of derivatives

We would like to give a short diagram that summarizes the previous two sections.



In practice, try to first figure out where in this diagram your problem is located and then solve it accordingly.

3.3 Functional derivatives

There is one more type of derivative that we need to discuss: a *functional derivative*.

Functionals

An ordinary function $f(x)$ with scalar input and scalar output can be thought of as a sort of machine: number in $\rightarrow$ number out. A functional $L\left[f\left(x\right)\right]$ is another type of machine that works like this: function in $\rightarrow$ number out. Here is an example for a simple type of functional:

$$L\left[f\left(x\right)\right]=\int_{-\infty}^{\infty}\left[f\left(x\right)-g\left(x\right)\right]^{2}dx. \tag{3.176}$$

This could for instance be a cost functional in a least square problem where the input x is a continuous variable.

Functional derivative

We can optimize functionals. In order to do so, we need to first find a meaningful differentiation rule. Here we define a functional derivative as [28]

$$\frac{\delta L\left[f\left(x\right)\right]}{\delta f\left(y\right)}=\lim_{\epsilon\rightarrow0}\frac{L\left[f\left(x\right)+\epsilon\delta\left(x-y\right)\right]-\mathcal{L}\left[f\left(x\right)\right]}{\epsilon}, \tag{3.177}$$

where $\delta\left(x\right)$ is the Dirac delta function, which satisfies the *sifting property*

$$\int_{-\infty}^{\infty}\delta\left(x-y\right)f\left(x\right)dx=f\left(y\right). \tag{3.178}$$

You can think of the sifting property of the Dirac delta function to be continuous analog to the Kronecker delta (compare Eq. 3.18).

Optimality condition for functional

Consider a functional of the form

$$L\left[f\left(x\right)\right]=\int_{-\infty}^{\infty}\mathcal{L}\left[f\left(x\right),f_{x}\left(x\right)\right]dx, \tag{3.179}$$

which we wish to minimize. Here $f_x = \partial f/\partial x$ and $\mathcal{L}$ is referred to as a *Lagrangian density*. The optimality condition for functionals is stated here without proof to be that the functional derivative vanishes

$$\frac{\delta L\left[f\left(x\right)\right]}{\delta f\left(y\right)} = 0. \tag{3.180}$$

We will show that this optimality condition can be rewritten in form of a partial differential equation (PDE)

$$\frac{\partial \mathcal{L}}{\partial f} = \frac{\partial}{\partial x}\frac{\partial \mathcal{L}}{\partial f_x}, \tag{3.181}$$

which is the so-called *Euler-Lagrange equation* (ELE). Solving this PDE for f gives a stationary point of $L\left[f\left(x\right)\right]$.

Euler-Lagrange equation

For the derivation of the Euler-Lagrange equation, we need a few basic properties. We will not provide rigorous proofs here, but outline the argument. An essential assumption for all that follows is that the Lagrangian density $\mathcal{L}$ approaches its first order Taylor approximation in the limit of small ϵ,

$$\mathcal{L}\left[f\left(x\right) + \epsilon \cdot \delta\left(x-y\right)\right] =_{\epsilon\to 0} \mathcal{L}\left[f\left(x\right)\right] + \epsilon \cdot \delta\left(x-y\right)\frac{\partial \mathcal{L}\left[f\left(x\right)\right]}{\partial f}, \tag{3.182}$$

which can be made arbitrary precise as ϵ gets smaller.

First, suppose

$$L\left[f\left(x\right)\right] = \int_{-\infty}^{\infty} \mathcal{L}\left[f\left(x\right)\right] dx, \tag{3.183}$$

then

$$\begin{aligned}
\frac{\delta L\left[f\left(x\right)\right]}{\delta f\left(y\right)} &= \lim_{\epsilon\to 0}\frac{1}{\epsilon}\int_{-\infty}^{\infty} \mathcal{L}\left[f\left(x\right) + \epsilon\delta\left(x-y\right)\right]dx - \frac{1}{\epsilon}\int_{-\infty}^{\infty}\mathcal{L}\left[f\left(x\right)\right]dx && (3.184)\\
&= \lim_{\epsilon\to 0}\frac{1}{\epsilon}\int_{-\infty}^{\infty} \mathcal{L}\left[f\left(x\right)\right] + \epsilon\delta\left(x-y\right)\frac{\partial \mathcal{L}\left[f\left(x\right)\right]}{\partial f}dx && (3.185)\\
&\quad - \frac{1}{\epsilon}\int_{-\infty}^{\infty}\mathcal{L}\left[f\left(x\right)\right]dx && (3.186)\\
&= \lim_{\epsilon\to 0}\frac{1}{\epsilon}\int_{-\infty}^{\infty} \epsilon\delta\left(x-y\right)\frac{\partial \mathcal{L}\left[f\left(x\right)\right]}{\partial f}dx && (3.187)\\
&= \frac{\partial \mathcal{L}\left[f\left(y\right)\right]}{\partial f}. && (3.188)
\end{aligned}$$

Second, suppose

$$L\left[f\left(x\right)\right] = \int_{-\infty}^{\infty} \mathcal{L}\left[f_x\left(x\right)\right] dx, \tag{3.189}$$

where $f_x(x) = \partial f(x)/\partial x$, then

$$\frac{\delta L[f(x)]}{\delta f(y)} = \lim_{\epsilon\to 0}\frac{1}{\epsilon}\int_{-\infty}^{\infty}\mathcal{L}[f_x(x)+\epsilon\delta'(x-y)]dx - \frac{1}{\epsilon}\int_{-\infty}^{\infty}\mathcal{L}[f_x(x)]\,dx \tag{3.190}$$

$$= \lim_{\epsilon\to 0}\frac{1}{\epsilon}\int_{-\infty}^{\infty}\mathcal{L}[f_x(x)] + \epsilon\delta'(x-y)\frac{\partial\mathcal{L}[f_x(x)]}{\partial f_x}dx\ldots$$

$$-\frac{1}{\epsilon}\int_{-\infty}^{\infty}\mathcal{L}[f_x(x)]\,dx \tag{3.191}$$

$$= \lim_{\epsilon\to 0}\frac{1}{\epsilon}\int_{-\infty}^{\infty}\epsilon\delta'(x-y)\frac{\partial\mathcal{L}[f_x(x)]}{\partial f_x}dx \tag{3.192}$$

$$= \int_{-\infty}^{\infty}\underbrace{\delta'(x-y)}_{u'}\underbrace{\frac{\partial\mathcal{L}[f_x(x)]}{\partial f_x}}_{v}dx \tag{3.193}$$

$$=_{\text{ibp}} u\cdot v|_{-\infty}^{\infty} - \int_{-\infty}^{\infty}u\cdot\frac{\partial v}{\partial x}\,dx \tag{3.194}$$

$$= \delta(x-y)\frac{\partial\mathcal{L}[f_x(x)]}{\partial f_x}|_{-\infty}^{\infty} - \int_{-\infty}^{\infty}\delta(x-y)\frac{\partial}{\partial x}\frac{\partial\mathcal{L}[f_x(x)]}{\partial f_x}dx \tag{3.195}$$

$$= -\frac{\partial}{\partial y}\frac{\partial\mathcal{L}[f_y(y)]}{\partial f_y}, \tag{3.196}$$

where "ibp" stands for integration by parts. We have also that $u\cdot v|_{-\infty}^{\infty} = 0$ assuming that $y\in(-\infty,\infty)$.

Third, suppose

$$L[f(x)] = \int_{-\infty}^{\infty}\mathcal{L}[f(x), f_x(x)]\,dx. \tag{3.197}$$

Using a bivariate Taylor expansion of $\mathcal{L}$ and the results of the previous derivations one then shows that Exercise

$$\frac{\delta L[f(x)]}{\delta f(y)} = \frac{\partial\mathcal{L}}{\partial f(y)} - \frac{\partial}{\partial y}\frac{\partial\mathcal{L}}{\partial f_y(y)}. \tag{3.198}$$

Using $\delta L/\delta f = 0$ (optimality condition) and reassigning the variable x to y we get the ELE

$$\frac{\partial\mathcal{L}}{\partial f(x)} = \frac{\partial}{\partial x}\frac{\partial\mathcal{L}}{\partial f_x(x)}. \tag{3.199}$$

3.4 Optimality conditions for unconstrained discrete and continuous optimization

At this point we may have been found guilty by the reader of talking about derivatives for too long without doing anything useful with it. The use of all of the aforementioned derivatives will be discussed now. One important point of derivatives is to optimize problems. However, we first need to first decide what we consider *optimal*. This will

typically involve scalar and real-valued cost functions $\mathcal{L}$, which can take on a large variety of different mathematical forms. We will seek minima of these cost functions. Of course, maximum problems can simply be turned into minimum problems by reversing the sign of the cost function $-\mathcal{L}$. For all the derivatives we discussed so far the *necessary condition* for optimality is that the first derivative vanishes. Consider what a general statement that is! It means that

$$\frac{\partial \mathcal{L}}{\partial x} = 0, \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}} = 0, \frac{\partial \mathcal{L}}{\partial \boldsymbol{X}} = 0 \tag{3.200}$$

when dealing with real-valued quantities,

$$\frac{\partial \mathcal{L}}{\partial z^*} = 0, \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^*} = 0, \frac{\partial \mathcal{L}}{\partial \boldsymbol{Z}^*} = 0 \tag{3.201}$$

when dealing with real-valued quantities, and

$$\frac{\delta \mathcal{L}}{\delta f} = 0 \tag{3.202}$$

in problems where an entire function f is to be found to minimize $\mathcal{L}$. The latter may be rewritten as (ELE)

$$\frac{\partial \mathcal{L}}{\partial f} - \frac{\partial}{\partial x}\frac{\partial \mathcal{L}}{\partial f_x} = 0. \tag{3.203}$$

In the case of complex-valued f it can be shown that the optimality condition amounts to solving (complex ELE)

$$\frac{\partial \mathcal{L}}{\partial f^*} - \frac{\partial}{\partial x}\frac{\partial \mathcal{L}}{\partial f_x^*} = 0, \tag{3.204}$$

where the latter derivatives are Wirtinger derivatives.

We do not discuss sufficient conditions for optimality. These may be found for the case of scalar and vector valued derivatives in [31]. A second order theory of Wirtinger derivatives is reviewed in [26]. Sufficient conditions for Functional derivatives are found in [25].

3.5 Gradient descent

We will now get to work and use the material discussed so far. In many practical situations, we are given a cost function $\mathcal{L}$ which needs to be minimized. We can attempt to do so iteratively by using *gradient descent* (GD)[11]

$$x_{n+1} = x_n - \alpha \frac{\partial \mathcal{L}}{\partial x}. \tag{3.205}$$

The step size α controls how fast we want to walk downhill in the given cost landscape. We will say more about the step size later. Suffice it here to note that we can not choose it arbitrarily large, which can result in numerical instability. The GD update rule generalizes to other derivatives such as gradients, matrix-valued derivatives, and Wirtinger derivatives. It even works for functionals as we will see.

[11] Gradient descent is also referred to as *steepest descent*.

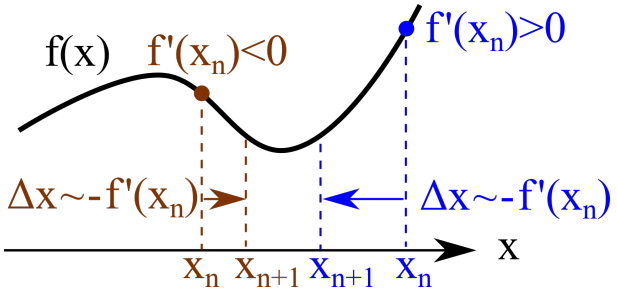


Figure 3.3: Illustration of one iteration of gradient descent.

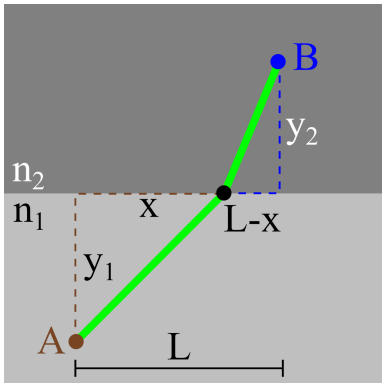


Figure 3.4: Fermat via gradient descent. We would like to computationally find the x-coordinate of the black point.

We note that Eq. 3.205 could be written more precisely

$$x_{n+1} = x_n - \alpha \frac{\partial \mathcal{L}}{\partial x}|_{x_n},$$

meaning that we evaluate the slope at the old point x_n. Then we walk along the negative gradient at x_n to get a new point x_{n+1}.

But where does the GD update come from? The basic idea for univariate functions is shown in Fig. 3.3. If we start at the blue point, the derivative is positive. That means that we have to reduce x_n by an amount proportional to the (positive) derivative. If we start at the brown point, the derivative is negative, meaning that we have to increase x. Thus in both cases, the iterate x_n is moved into the opposite direction of the gradient f'. This reduces the error iteratively until a local (or global) minimum is found.

3.5.1 GD with scalar-valued derivatives

Example: Fermat's principle via GD

Suppose a laser beam travels between two media n_1 and n_2 from point A to point B, as illustrated in Fig. 3.4. We would like to know the coordinate x at the intersection between

the two media. From Fermat's principle we know that the ray travels in such way that it minimizes the optical path length between A and B. Thus we formulate the total optical path as a cost function

$$\mathcal{L} = n_1\sqrt{x^2 + y_1^2} + n_2\sqrt{(x-L)^2 + y_2^2}. \tag{3.206}$$

Differentiating with respect to x we get

$$\frac{\partial \mathcal{L}}{\partial x} = n_1 \frac{x}{\sqrt{x^2 + y_1^2}} + n_2 \frac{x-L}{\sqrt{(x-L)^2 + y_2^2}}. \tag{3.207}$$

We can be confident that this is true as the optimality condition $\partial\mathcal{L}/\partial x = 0$ reduces to Snell's law

$$n_1 \sin(\alpha_1) = n_2 \sin(\alpha_2) \tag{3.208}$$

where

$$\sin(\alpha_1) = \frac{x}{\sqrt{x^2 + y_1^2}} \tag{3.209}$$

and

$$\sin(\alpha_2) = \frac{L-x}{\sqrt{(x-L)^2 + y_2^2}}. \tag{3.210}$$

However, what we are interested in is how to find x. We could square Eq. 3.208 and try to solve for x. This would result in a quartic equation, which is already beyond what we would like to tackle by hand. Instead we can use GD. We simply start with a initial estimate for x which we iteratively refine by walking the direction given by Eq. 3.207 downhill. This is illustrated in the following code.

```
% clc
% fixed parameters
y1 = 1;
y2 = 2;
n1 = 1;      % air
n2 = 1.5;    % glass
L = 4;

% algorithmic properties
x = 2;           % initial estimate for x
stepSize = 2;    % step size
numIter = 200;    % number of iterations
for k=1:numIter
    % compute gradient of cost function
    gradCost = n1*x/sqrt(x^2+y1^2) + ...
               n2*(x-L)/sqrt((x-L)^2 + y2^2);
    % walk downhill
    x = x - stepSize * gradCost;
end
```

```
% display result with n digits precision
format long
n = 10;
disp('final estimate for x: ')
disp(round(x*10^n)/10^n)
```

Listing 3.3: scalarGradientDescent.m (Matlab)

Here we chose trivial boundary conditions to check the special case where by symmetry the beam must travel through the center. Namely, we set $n_1 = n_2 = 1$, which is to say that the beam travels in air. In addition $y_1 = y_2 = 1$, which makes the problem perfectly symmetric. Then choosing $L = 4$ must result in $x = 2$, which the code finds.

Exercise: Write a vectorized code that jointly walks downhill for 11 equispaced initial estimates in the range $-5 \leq x \leq 5$. Save the estimates for each initial condition as a function of iteration and visualize the results. Also monitor and visualize the cost of each estimate as a function of iteration. Do all the initial estimates converge to the solution? Can you find an x that does not converge? Exercise

---

How do we know what we have programmed works also for other values? We can get at least a hint that what we are doing is right by plotting out the cost landscape. For example, change the parameters to $n_1 = 1$, $n_2 = 1.5$, $y_1 = 1$, and $y_2 = 2$. The modified code (`scalarGradientDescent.m`) with the new parameters then returns $x = 2.4335$. We can now test in a separate code[12] how the cost depends for various values of x. Here we test 50 equidistant points $x \in [0, L]$ and plot the resulting cost for each x. This is shown in Fig. 3.5. Our estimate $x = 2.4335$ seems to not be too far off.

```
% fixed parameters
y1 = 1;
y2 = 2;
n1 = 1;
n2 = 1.5;
L = 4;
% algorithmic properties
x = linspace(0,L,50);          % initial estimate for x
costs = zeros(size(x));    % preallocate costs
stepSize = 2;   % step size
numIter = 20;   % number of iterations
for k=1:length(x)
    % compute cost function
    costs(k) = n1*sqrt(x(k)^2+y1^2) + ...
               n2*sqrt((x(k)-L)^2 + y2^2);
end
% display result with n digits precision
plot(x, costs, 'ko','MarkerFaceColor','k')
```

[12] see `scalarGradientDescentOtherParameterCostLandscape.m` (sorry for the long name)

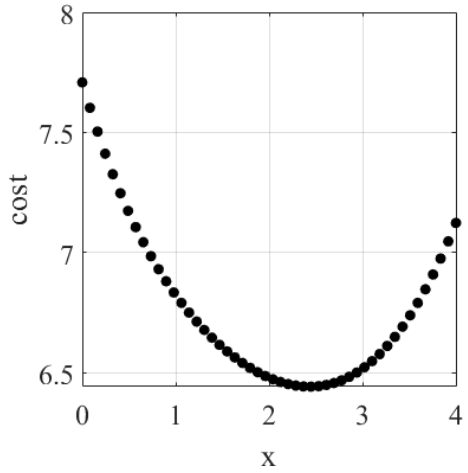


Figure 3.5: Cost in Eq. 3.206 for various values of x and the parameters $n_1 = 1$, $n_2 = 1.5$, $y_1 = 1$, and $y_2 = 2$.

```
19 xlabel('x'),ylabel('cost'), grid on, axis square
```

Listing 3.4: scalarGradientDescentOtherParameterCostLandscape.m (Matlab)

3.5.2 GD with vector-valued derivatives

Exercise: Write a code that finds x_1 and x_2 via GD for three layers with refractive indices n_1, n_2, and n_3 (see Fig. 3.6). First derive the gradient of the multivariate cost function. You can choose the parameters $y_1 = 1$, $y_2 = 2$, and $y_3 = 1$ as well as $n_1 = 1$, $n_2 = 1.5$ and $n_3 = 1$. Test whether your solution is close to the global optimum by mapping out the two-dimensional cost landscape. Exercise

---

Multivariate optimization is too important (and fun) to leave it with just one example. Let us consider another problem.

Example: Multivariate curve fitting

Suppose we are given data $\boldsymbol{d}$ as shown in Fig. 3.7. We are given the task to fit a line through the data. Our model is a linear function of the form

$$y = ax + b. \tag{3.211}$$

We assume Gaussian noise statistics and attempt to minimize the cost function

$$\mathcal{L} = \sum_i \left(ax_i + b - d_i\right)^2. \tag{3.212}$$

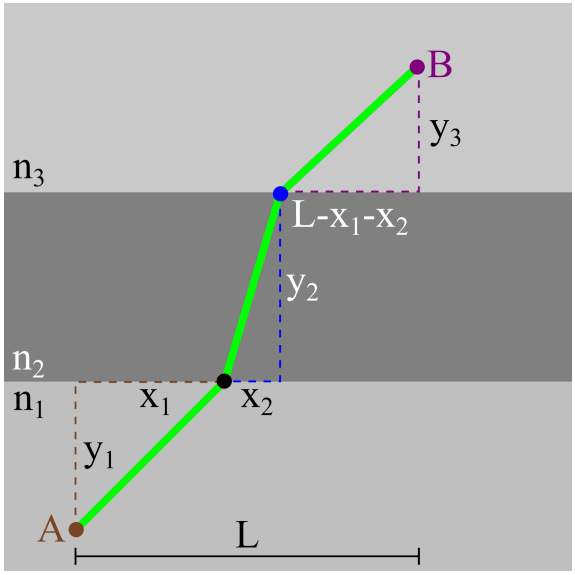


Figure 3.6: Multivariate Fermat via GD for three layers.

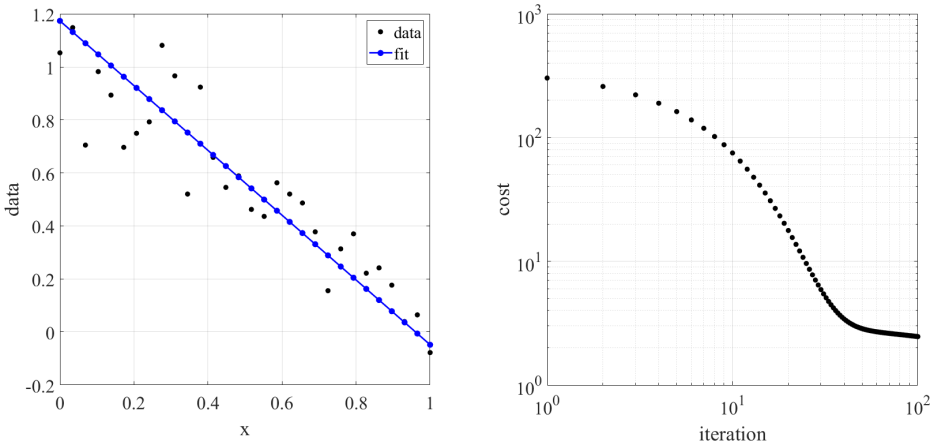


Figure 3.7: Illustration of multivariate curve fitting problem. In this example we attempt to fit a linear model to the data.

To this end, we compute the gradient

$$\nabla \mathcal{L} = \begin{pmatrix} \partial \mathcal{L}/\partial a \\ \partial \mathcal{L}/\partial b \end{pmatrix} \sim \begin{pmatrix} \sum_i r_i x_i \\ \sum_i r_i \end{pmatrix}$$

where we ignored a constant factor of 2 and

$$r_i = a x_i + b - d_i \tag{3.213}$$

is the *residual* between each data point and our linear model. We can then perform multivariate GD to search for a solution

$$\begin{pmatrix} a_{n+1} \\ b_{n+1} \end{pmatrix} = \begin{pmatrix} a_n \\ b_n \end{pmatrix} - \alpha \begin{pmatrix} \sum_i r_i x_i \\ \sum_i r_i \end{pmatrix}, \tag{3.214}$$

where α is a step size. This procedure is illustrated in the Matlab code `multivariateCurveFitting.m`, which after 100 iterations produces the result in Fig.

```
rng(0)
N = 3e1;
noise = 0.1*randn(1,N);
x = linspace(0,1,N);
a = -1; b = 1;
d = a*x + b + noise;
figure(1)
plot(x,d,'ko','MarkerFaceColor','k')
axis square
xlabel('x'), ylabel('data'), grid on
%% choose initial parameters
a = -2;
b = 0;
%% gradient descent
numIter = 100;
stepSize = 1e-3;
figureUpdate = 10;
cost = zeros(numIter,1);
for loop = 1:numIter
    % compute residual
    r = 2*(a*x+b-d);
    % compute gradient
    aGrad = sum(r .* x);
    bGrad = sum(r);
    % search along negative gradient
    a = a - stepSize * aGrad;
    b = b - stepSize * bGrad;
    % get cost
    cost(loop) = sum( r.^2 );
    % show current estimate
    if mod(loop,figureUpdate)==0
```

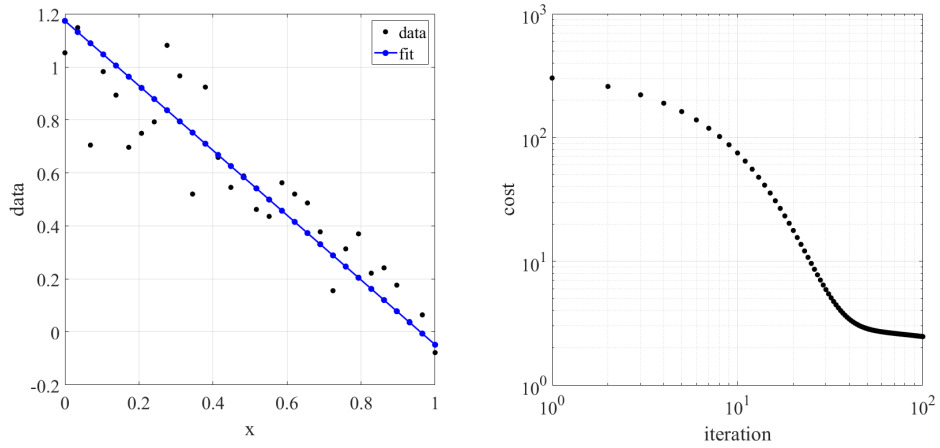


Figure 3.8: Multivariate curve fitting.

```
32          figure(2)
33          subplot(1,2,1)
34          plot(x,d,'ko','MarkerFaceColor','k')
35          hold on
36          plot(x,a*x+b,'bo-','MarkerFaceColor','b','linewidth',2)
37          legend('data', 'fit')
38          axis square, xlabel('x'), ylabel('data'), grid on
39          hold off
40
41          subplot(1,2,2)
42          loglog(1:loop, cost(1:loop), 'ko','MarkerFaceColor','k')
43          axis square, grid on
44          xlabel('iteration'), ylabel('cost')
45          drawnow
46      end
47  end
```

Listing 3.5: multivariateCurveFitting.m (Matlab)

3.5.2.1 Backtracking

You may have noticed that the choice for the step size α in the previous example was arbitrary. Generally, finding an optimal step size in gradient descent is not easy. If we choose α too large, then GD can diverge and no solution is found. If we choose α too small, converge is slow of the algorithm is slow. One possible solution is *backtracking*. In backtracking we start with $\alpha = 1$ and compute a test solution

$$\begin{pmatrix} a_t \\ b_t \end{pmatrix} = \begin{pmatrix} a_n \\ b_n \end{pmatrix} - \alpha \begin{pmatrix} \sum_i r_i x_i \\ \sum_i r_i \end{pmatrix}. \tag{3.215}$$

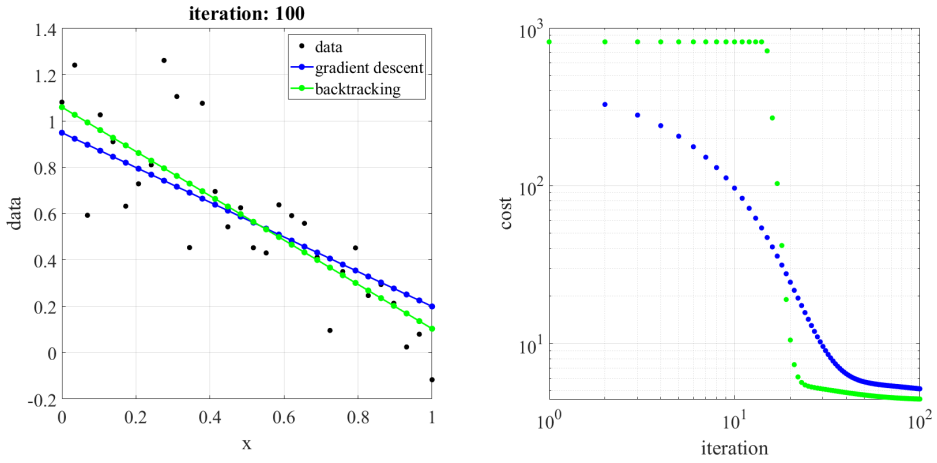


Figure 3.9: Multivariate curve fitting via backtracking.

If

$$\mathcal{L}\left(a_t, b_t\right) < \mathcal{L}\left(a_n, b_n\right), \tag{3.216}$$

then we accept the search direction,

$$\begin{pmatrix} a_{n+1} \\ b_{n+1} \end{pmatrix} = \begin{pmatrix} a_t \\ b_t \end{pmatrix}. \tag{3.217}$$

However, if otherwise

$$\mathcal{L}\left(a_t, b_t\right) \geq \mathcal{L}\left(a_n, b_n\right), \tag{3.218}$$

then we reduce the step size

$$\alpha_{n+1} = \gamma \alpha_n, \tag{3.219}$$

where $\gamma \in (0, 1)$.

A comparison of gradient descent (blue, $\alpha = 10^{-3}$) and backtracking (green, $\gamma = 2/3$, $\alpha_1 = 1$, $\alpha_{100} \approx 5 \cdot 10^{-3}$) for the problem from the previous subsection (100 iterations) is shown in Fig. 3.9.

Exercise: Write Python or Julia code that compares gradient descent with a fixed step size and backtracking, producing a similar plot as shown in Fig. 3.9. If you encounter difficulties with the implementation, take a look at the Matlab implementation multivariateCurveFittingBacktracking.m in the CI GitHub repository. Exercise

---

3.5.3 GD with matrix-valued derivatives

Suppose we are given some linear operation that transforms our data $\boldsymbol{X}$ in some way. As an example, let us assume that transformation is given by the Matlab command `B = cat(1, X, flipud(X))`. After convincing yourself that this is indeed a linear operation, you are curious how this operation can be represented by a matrix $\boldsymbol{A}$. We may Exercise

thus write down the cost function

$$\mathcal{L} = \|\boldsymbol{A}\boldsymbol{X} - \boldsymbol{B}\|_F^2 \,, \tag{3.220}$$

which is solved by matrix least squares (compare Eq. 3.85). However, we may also attempt to find a solution iteratively via gradient descent (with backtracking). This is shown in the following code[13] resulting in Fig. 3.10.

```
rng(0)
m = 5;
n = 10;
X = rand(m,n);
B = cat(1, X, flipud(X));

Alsq = B*X'/(X*X');
% gradient descent
numIter = 1e3;
cost = zeros(numIter,1);
A = rand(size(B,1),size(X,1));
% A = Alsq;
stepSize = 1e-1;
R = A*X - B;
cost(1) = norm(R,'fro');
for loop = 2:numIter
    % gradient descent step
    A_test = A - stepSize * R*X';
    % test residual
    R = A_test*X - B;
    % test cost
    cost_test = norm(R,'fro');
    % backtracking
    if cost_test <= cost(loop-1)
        cost(loop) = cost_test;
        A = A_test;
    else
        stepSize = 0.9*stepSize;
        cost(loop) = cost(loop-1);
        disp(loop)
    end
end

figure(1)
subplot(1,3,1)
imagesc(A)
axis image off, colormap winter
```

[13] We use Eq. 3.84 to compute the matrix-valued gradient

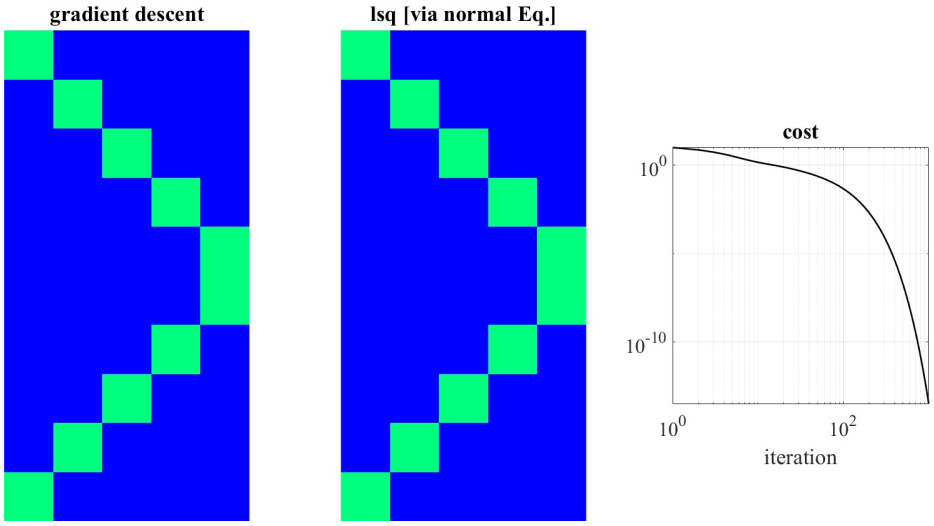


Figure 3.10: Estimating a matrix via gradient descent (left) and matrix least squares (right, compare Eq. 3.85). The green elements in each matrix estimate for $\boldsymbol{A}$ represent ones, while the blue entries are zeros.

```
38 title('gradient descent')
39
40 subplot(1,3,2)
41 imagesc(Alsq)
42 axis image off, colormap winter
43 title('lsq [via normal Eq.]')
44
45 subplot(1,3,3)
46 loglog(cost,'k','linewidth',2)
47 axis square, title('cost')
48 xlabel('iteration'), grid on
```

Listing 3.6: matrixEstimationViaGradientDescent.m (Matlab)

Chapter 4

Signal and Image Processing

4.1 Image Registration

4.2 Clustering

4.2.1 K-means

4.2.2 Fisher's linear discriminant analysis

Suppose we measure data that exhibits certain clustering behavior, as illustrated in Fig. 4.1 a. From this existing data $\boldsymbol{x}$ (already revealed to us) our goal is to build a *discriminant function* $f(\boldsymbol{x})$ that helps us in classifying future data (not yet revealed to us). For instance the function f could take on the value -1 or +1 if the data belongs to the first or second class, respectively. This is a machine learning task, namely a problem referred to as *classification*. From past data with known attribution to one out of two (or more) classes, also called *training data*, we want to learn how to classify future data. As we will see, it is typically a good idea to withhold some of the training data to learn the classification task. This allows the withheld data to be used for a separate assessment of whether or not the discrimination function we trained works only on the training data or if it generalizes to unseen data (this is called *validation*).

The displayed features in Fig. 4.1 may be the data in its original format or the result of some dimensionality reduction process. For instance, feature 1 could represent the average number of hours of sun in some polar region of the earth and feature 2 could be the average temperature. Another possibility is that the features themselves are extracted from some more complicated data. An example would be that we are given diffraction patterns with millions of pixels. The diffraction data could be a result from a laser beam scattering off a surface and measured on a pixelated camera. We could condense the large amount of information in the diffraction data into two (or more) features. Feature 1 could be the total reflected signal, which we obtain by integrating over all pixels in the diffraction pattern. Feature 2 could be the spatial standard deviation of the diffraction data, for instance providing information about the characteristic surface roughness. A smooth surface will mainly scatter into the specular direction, while with higher surface

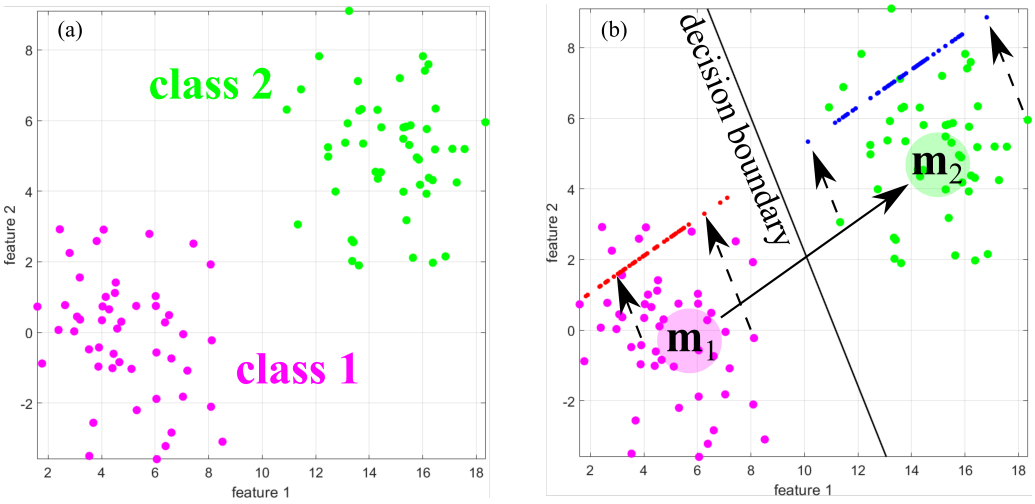


Figure 4.1: (a) Example for a feature space that forms two clusters. (b) The goal in classification is to find a decision rule that allows us to categorize future data by building a discrimination function from past data. The main text discusses several geometric quantities of interest that are highlighted in the right panel. The small spheres are individual data points. The large spheres depict the center of mass of each cluster, which are connected seperated along the direction $\boldsymbol{d}$. The red and blue points are the projections of the magenta and green data, respectively, onto the a line through the origin along the direction $\boldsymbol{d}$. The black line marks the decision boundary. Future data that lies to the left or right of the decision boundary is categorized into class 1 or class 2, respectively.

roughness we will observe increasing angular spread. Thus the spatial standard deviation could provide us with a useful feature. Other features might be useful, for instance indicating anisotropic scattering or other properties. Using diffraction data to extract statistical information and geometric parameters for metrology purposes is referred to as *scatterometry*. Finding the most relevant features is a non-trivial problem on its own. Deep neural networks (to be discussed later) perform a classification task along with the search for ideal features, which makes them very powerful tools. Here we shall be concerned with the situation where the features are already given.

Let us return to the two clusters shown in Fig. 4.1. How should we define the discrimination function f? Our first approach discussed here is be somewhat heuristic and will be refined below. We observe that a measure of the separation of the two clusters is given by the difference between the average feature in each point cloud

$$\boldsymbol{s} = \boldsymbol{m}_2 - \boldsymbol{m}_1, \tag{4.1}$$

where

$$\boldsymbol{m}_i = \frac{1}{K}\sum_{k=1}^{K} \boldsymbol{x}_{i,k} \tag{4.2}$$

is the center of mass of each respective point cloud and $\boldsymbol{x}_{i,k}$ is the kth point in the ith cloud ($i = 1, 2$). The vector $\boldsymbol{s}$ is a measure of the separation of the clusters. Unfortunately it is a vectorial quantity. Our original goal was to find a scalar function f that returns -1 or +1 in case the input data belongs class 1 or 2. To this end, let us convert the vector $\boldsymbol{s}$ by projecting it onto a direction $\boldsymbol{d}$ where the data is well-separated on average using the inner product $\boldsymbol{d}^T\boldsymbol{s}$. Our hope is that if we separate the projected center of mass of each point cloud as much as possible by optimally choosing the direction $\boldsymbol{d}$, then the same will hold true for the projected point clouds. However, depending on the choice of $\boldsymbol{d}$ the scalar $\boldsymbol{d}^T\boldsymbol{s}$ may take on all values from $-\infty$ to ∞. We have to normalize it and make sure that it only takes on positive values, which naturally yields the Rayleigh quotient

$$R = \frac{\left(\boldsymbol{d}^T\boldsymbol{s}\right)^2}{\boldsymbol{d}^T\boldsymbol{d}} = \frac{\boldsymbol{d}^T\boldsymbol{s}\boldsymbol{d}^T\boldsymbol{s}}{\boldsymbol{d}^T\boldsymbol{d}} = \frac{\boldsymbol{d}^T\boldsymbol{s}\boldsymbol{s}^T\boldsymbol{d}}{\boldsymbol{d}^T\boldsymbol{d}} = \frac{\boldsymbol{d}^T\boldsymbol{S}\boldsymbol{d}}{\boldsymbol{d}^T\boldsymbol{d}}. \tag{4.3}$$

Here we made use of the fact that $\boldsymbol{d}^T\boldsymbol{s} = \boldsymbol{s}^T\boldsymbol{d}$ and defined the symmetric rank one matrix $\boldsymbol{S} = \boldsymbol{s}\boldsymbol{s}^T = (\boldsymbol{m}_2 - \boldsymbol{m}_1)(\boldsymbol{m}_2 - \boldsymbol{m}_1)^T$. How do we maximize this ratio? We showed in subsection 2.7.4 that the eigenvector with the largest eigenvalue of $\boldsymbol{S}$ maximizes R. It is simple to see that that $\boldsymbol{S} = (\boldsymbol{m}_2 - \boldsymbol{m}_1)(\boldsymbol{m}_2 - \boldsymbol{m}_1)^T$ has only one eigenvector $\boldsymbol{d} = \boldsymbol{m}_2 - \boldsymbol{m}_1 = \boldsymbol{s}$,

$$\boldsymbol{S}(\boldsymbol{m}_2 - \boldsymbol{m}_1) = (\boldsymbol{m}_2 - \boldsymbol{m}_1)(\boldsymbol{m}_2 - \boldsymbol{m}_1)^T(\boldsymbol{m}_2 - \boldsymbol{m}_1) = \|\boldsymbol{m}_2 - \boldsymbol{m}_1\|^2(\boldsymbol{m}_2 - \boldsymbol{m}_1). \tag{4.4}$$

Thus we project the data $\boldsymbol{x}$ onto the direction given by the difference of the two center of masses,

$$\boldsymbol{g}_1(\boldsymbol{x}) = \frac{\boldsymbol{d}^T\boldsymbol{x}}{\boldsymbol{d}^T\boldsymbol{d}}\boldsymbol{d}, \tag{4.5}$$

which in Fig. 4.1 b gives the red and blue points upon substitution of the magenta and green points, respectively, for $\boldsymbol{x}$. The coefficient $\boldsymbol{d}^T\boldsymbol{x}/\boldsymbol{d}^T\boldsymbol{d}$ is the projection of $\boldsymbol{x}$ along

the direction $\boldsymbol{d}$. Mostly its sign matters and we may absorb the scaling by $\boldsymbol{d}^T\boldsymbol{d}$ together with a possible offset from the origin into a constant parameter c. Thus we can define the function

$$f = \text{sign}\left(\boldsymbol{d}^T\boldsymbol{x} - c\right) \tag{4.6}$$

where

$$\text{sign}\,(x) = \begin{cases} 1 & , x > 0 \\ -1 & , x \leq 0 \end{cases}. \tag{4.7}$$

How should we choose c? A natural choice is to require that the argument of Eq. 4.6 vanishes at the midpoint between the two clusters $1/2\,(\boldsymbol{m}_1 + \boldsymbol{m}_2)$. This leads to

$$c = \frac{1}{2}\boldsymbol{d}^T\,(\boldsymbol{m}_1 + \boldsymbol{m}_2) \tag{4.8}$$

giving the final discriminant function

$$f\,(\boldsymbol{x}) = \text{sign}\left(\boldsymbol{d}^T\,[2\boldsymbol{x} - \boldsymbol{m}_1 - \boldsymbol{m}_2]\right). \tag{4.9}$$

We refer to the argument $\boldsymbol{d}^T\,[2\boldsymbol{x} - \boldsymbol{m}_1 - \boldsymbol{m}_2] = 0$ as the *decision boundary*. The decision boundary is marked as a solid black line in Fig. 4.1 b. In two-dimensional problems, the decision boundary is a line (one-dimensional). In N-dimensional problems the decision boundary becomes an $(N-1)$-dimensional hyperplane.

The aforementioned discrimination function in Eq. 4.9 leads to good classification performance when the features are uncorrelated. If the features are correlated, then the decision boundary is not optimal. This is illustrated in Fig. 4.2 a. Here the point clouds are correlated, meaning that the features tend to jointly vary transforming what was previously circular point clouds (compare Fig. 4.1) into now elliptical point clouds (Fig. 4.2). In this situation the heuristic approach described above fails. Fisher [15] understood this problem and proposed a solution. He modified Eq. 4.3 to the more general form

$$R = \frac{\boldsymbol{d}^T\boldsymbol{S}\boldsymbol{d}}{\boldsymbol{d}^T\boldsymbol{C}\boldsymbol{d}}, \tag{4.10}$$

where $\boldsymbol{S} = (\boldsymbol{m}_2 - \boldsymbol{m}_1)\,(\boldsymbol{m}_2 - \boldsymbol{m}_1)^T$ is the between-class variance matrix (as before) and

$$\boldsymbol{C} = \sum_{k \in \text{class}_1} (\boldsymbol{x}_k - \boldsymbol{m}_1)\,(\boldsymbol{x}_k - \boldsymbol{m}_1)^T + \sum_{l \in \text{class}_2} (\boldsymbol{x}_l - \boldsymbol{m}_1)\,(\boldsymbol{x}_2 - \boldsymbol{m}_2)^T \tag{4.11}$$

is the intra-class covariance matrix (unlike before). Maximizing the generalized Rayleigh quotient in Eq. 4.10 implies finding a direction $\boldsymbol{d}$ that maximizing the between class separation expressed by the numerator $\boldsymbol{d}^T\boldsymbol{S}\boldsymbol{d}$ and minimizing the intra-class covariance $\boldsymbol{d}^T\boldsymbol{C}\boldsymbol{d}$. We know from subsection 2.7.5 that we can maximize the generalized Rayleigh quotient by finding the generalized eigenvector that gives the maximum generalized eigenvalue in

$$\boldsymbol{S}\boldsymbol{d} = \lambda\boldsymbol{C}\boldsymbol{d}. \tag{4.12}$$

As previously, we know that $\boldsymbol{S}\boldsymbol{d} \sim \boldsymbol{m}_2 - \boldsymbol{m}_1$, so the solution for $\boldsymbol{d}$ must satisfy

$$\boldsymbol{d} \sim \boldsymbol{C}^{-1}\,(\boldsymbol{m}_2 - \boldsymbol{m}_1), \tag{4.13}$$

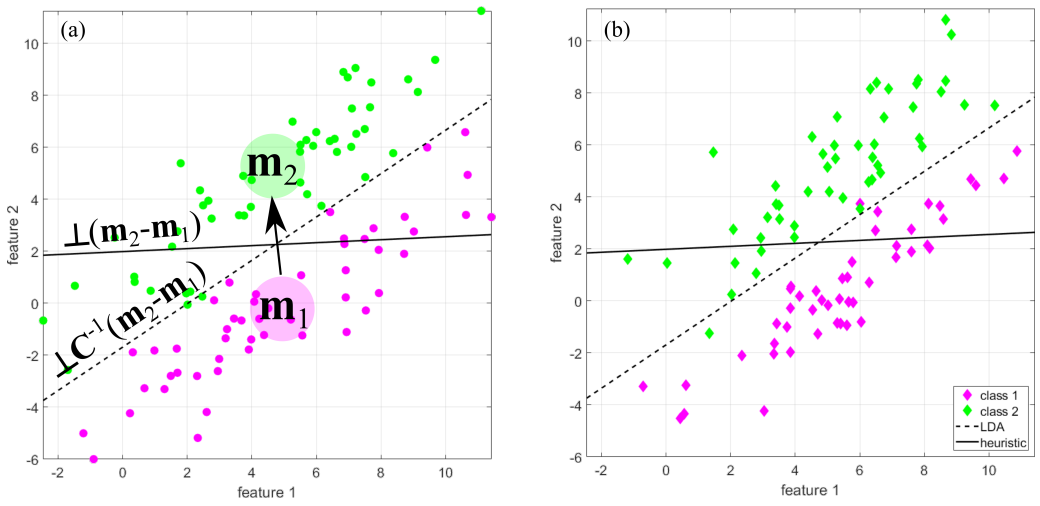


Figure 4.2: (a) Two clusters each with non-zero covariance. (b) The decision boundaries found by LDA and the heuristic method in panel (a) where validated on unseen test data (diamonds). LDA outperforms the heuristic method due to the incorporation of variance into the classification model.

where the proportionality constant is irrelevant (only the direction matters). This new projection direction can be inserted into the discriminant function

$$f = \text{sign}\left(\boldsymbol{d}^T \boldsymbol{x} - c\right). \tag{4.14}$$

Assuming (as previously) the midpoint between the clusters lies on the decision boundary, we get

$$c = \frac{1}{2}\left[\boldsymbol{C}^{-1}\left(\boldsymbol{m}_2 - \boldsymbol{m}_1\right)\right]^T \left(\boldsymbol{m}_2 + \boldsymbol{m}_1\right). \tag{4.15}$$

Figure 4.2 compares the previous (heuristic) to Fisher linear discriminant analysis (LDA). The decision boundaries for the heuristic approach and for LDA are shown as solid and dashed lines, respectively. We plotted unseen data (diamonds in panel b) drawn from the same probability distribution as the training data (circles in panel a). The unseen data is used for validation. In this particular example the data was drawn from a bivariate Gaussian probability density

$$p\left(\boldsymbol{x}_i \,|\, \boldsymbol{\mu}_i, \boldsymbol{C}_i\right) = \frac{1}{2\pi\sqrt{\det\left(\boldsymbol{\Sigma}\right)}} \exp\left[-\frac{1}{2}\left(\boldsymbol{x} - \boldsymbol{\mu}\right)^T \boldsymbol{C}^{-1}\left(\boldsymbol{x} - \boldsymbol{\mu}\right)\right], \tag{4.16}$$

where

$$\boldsymbol{\mu}_1 = \begin{bmatrix} 5 \\ 0 \end{bmatrix}, \boldsymbol{\mu}_2 = \begin{bmatrix} 5 \\ 5 \end{bmatrix}, \boldsymbol{C}_1 = \boldsymbol{C}_2 = \begin{bmatrix} 10 & 9 \\ 9 & 10 \end{bmatrix}. \tag{4.17}$$

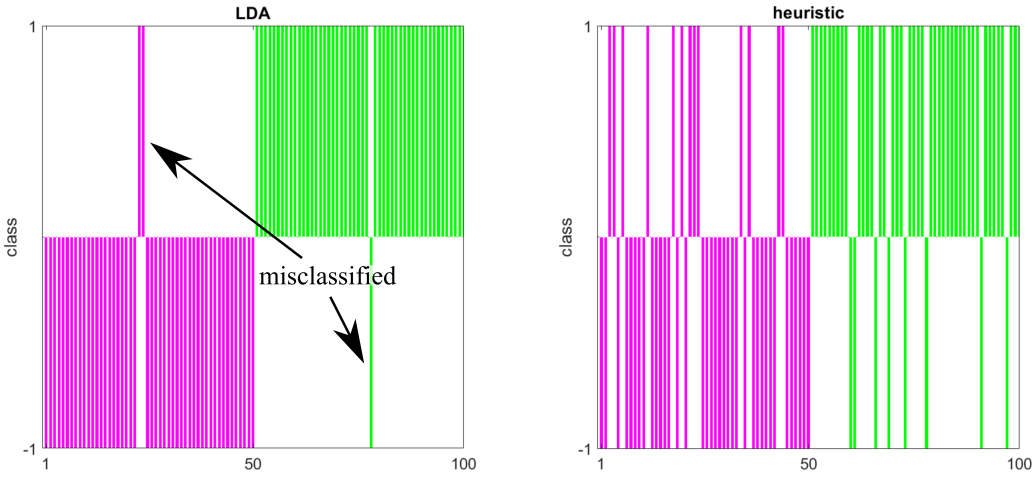


Figure 4.3: Validation results for the test data shown in Fig. 4.2 b. The left and right panels show the performance of LDA (3% misclassified) and the heuristic method (21% misclassified).

The validation result is shown in Fig. 4.3. It is seen that LDA classifies 97% (3% misclassified) of the data correctly, while the heuristic achieves only 79% (21% misclassified).

The plots in this section where generated with the following code.

```
% % example 2 (with covariance)
% mu1 = [5;0];
% mu2 = [5;5];
% p = 0.9;
% sigma1 = 10*[1,p;p,1];
% sigma2 = 10*[1,p;p,1];
% training data
X1 = mvnrnd(mu1,sigma1,K)';
X2 = mvnrnd(mu2,sigma2,K)';
% validation data
X1val = mvnrnd(mu1,sigma1,K)';
X2val = mvnrnd(mu2,sigma2,K)';

m1 = mean(X1,2);
m2 = mean(X2,2);
% between class separation matrix
Sb = (m2 - m1)*(m2 - m1)';
% within class covariance
Sw = (X1 - m1)*(X1 - m1)' + ...
     (X2 - m2)*(X2 - m2)';
% separation surface normal
w = Sw\(m2 - m1);
```

```
wlsq = m2 - m1;
% global mean
m = 1/2*(m1 + m2);
% threshold
c = -w'*m;
clsq = (m1 - m2)'*m;
% generate
s1 = linspace(min([X1(1,:),X2(1,:)]),max([X1(1,:),X2(1,:)]),100);
s2 = -1/w(2)*(c+w(1)*s1);
slsq = -1/wlsq(2)*(clsq+wlsq(1)*s1);

figure(1), clf
scatter(X1(1,:),X1(2,:),'mo','filled'), hold on
scatter(X2(1,:),X2(2,:),'go','filled')
plot(s1,s2,'--k','LineWidth',2)
plot(s1,slsq,'-k','LineWidth',2)
minAx1 = min([X1(1,:),X2(1,:)]);
minAx2 = min([X1(2,:),X2(2,:)]);
maxAx1 = max([X1(1,:),X2(1,:)]);
maxAx2 = max([X1(2,:),X2(2,:)]);
axis([minAx1, maxAx1, minAx2, maxAx2])
xlabel('feature 1'), ylabel('feature 2')
grid on, axis square
hold off

figure(2), clf
scatter(X1val(1,:),X1val(2,:),'md','filled'), hold on
scatter(X2val(1,:),X2val(2,:),'gd','filled')
plot(s1,s2,'--k','LineWidth',2)
plot(s1,slsq,'-k','LineWidth',2)
axis([minAx1, maxAx1, minAx2, maxAx2])
xlabel('feature 1'), ylabel('feature 2')
grid on, axis square
hold off

figure(3)
subplot(1,2,1)
bar( 1:K, sign(w'*X1val+c),'FaceColor','m','EdgeColor','w' ), hold on
bar( K+1:2*K, sign(w'*X2val+c),'FaceColor','g','EdgeColor','w' ), hold on
title('LDA validation')
axis square, hold off
axis([0 2*K -1 1])
xticks([1 50 100]), yticks([-1 1])
```

```
66 ylabel('class')
67
68 subplot(1,2,2)
69 bar( 1:K, sign(wlsq'*X1val+clsq),'FaceColor','m','EdgeColor','w'
       ), hold on
70 bar( K+1:2*K, sign(wlsq'*X2val+clsq),'FaceColor','g','EdgeColor'
      ,'w' ), hold on
71 title('LSQ validation')
72 axis square, hold off
73 axis([0 2*K -1 1])
74 xticks([1 50 100]), yticks([-1 1])
75 ylabel('class')
```

Listing 4.1: linearDiscriminantAnalysis.m (Matlab)

4.3 Phase unwrapping

Part II

Physical Optics

Chapter 5

Diffraction of scalar wave fields

The purpose of this chapter is to review the propagation of scalar electromagnetic waves. Quantities of frequent usage such as scalar wave fields and intensities are related to electrodynamic field quantities. Further, approximate models for fast computation of wave propagation and their limitations are derived.

5.1 Scalar wave propagation

In the absence of free charge and currents, *Maxwell's equations* in a linear medium are given by [5]

$$\nabla \times \boldsymbol{E} = -\frac{\partial (\mu \boldsymbol{H})}{\partial t} \tag{5.1}$$

$$\nabla \times \boldsymbol{H} = \frac{\partial (\epsilon \boldsymbol{E})}{\partial t} \tag{5.2}$$

$$\nabla \circ (\epsilon \boldsymbol{E}) = 0 \tag{5.3}$$

$$\nabla \circ (\mu \boldsymbol{H}) = 0, \tag{5.4}$$

where $\boldsymbol{E} = \boldsymbol{E}(\boldsymbol{r}, t) \in \mathbb{C}^3$ is the electric field, $\boldsymbol{H} = \boldsymbol{H}(\boldsymbol{r}, t) \in \mathbb{C}^3$ is the magnetic field, both being functions of position $\boldsymbol{r} \in \mathbb{R}^3$ and time $t \in \mathbb{R}$, and $\nabla\times$ as well as $\nabla\circ$ are shorthand notation for curl and divergence, respectively. For mathematical convenience, $\boldsymbol{E}$ and $\boldsymbol{H}$ are taken to be complex-valued. In this way, harmonic functions $A(\boldsymbol{r}, t) \cdot \cos(\boldsymbol{kr} - \omega t)$ can be represented as the real part of $A(\boldsymbol{r}, t) \cdot \exp[i(\boldsymbol{kr} - \omega t)]$ [48]. The complex phasor form is more convenient in mathematical manipulations. The permeability μ and permittivity ϵ, which describe material properties, are assumed to be scalar. With regard to the upcoming wave propagation models, assuming μ and ϵ to be scalar includes only isotropic media and excludes, for instance, effects such as birefringence and optical activity [20]. Taking the curl of Eq. 5.1

$$\nabla(\nabla \circ \boldsymbol{E}) - \nabla^2 \boldsymbol{E} = -\frac{\partial}{\partial t}((\nabla \mu) \times \boldsymbol{H} + \mu(\nabla \times \boldsymbol{H})) \tag{5.5}$$

and substituting Eq. 5.2 into the latter yields

$$\nabla\left(\nabla\circ\boldsymbol{E}\right)-\nabla^{2}\boldsymbol{E}=-\frac{\partial}{\partial t}\left(\left(\nabla\mu\right)\times\boldsymbol{H}+\mu\frac{\partial\epsilon\boldsymbol{E}}{\partial t}\right). \tag{5.6}$$

Assuming μ and ϵ to be time-independent gives

$$\nabla\left(\nabla\circ\boldsymbol{E}\right)-\nabla^{2}\boldsymbol{E}=-\left(\nabla\mu\right)\times\frac{\partial\boldsymbol{H}}{\partial t}-\epsilon\mu\frac{\partial^{2}\boldsymbol{E}}{\partial t^{2}}. \tag{5.7}$$

Using Eq. 5.1 and noting that $\left(\nabla\mu\right)/\mu=\nabla\ln\left(\mu\right)$ leads to

$$\nabla\left(\nabla\circ\boldsymbol{E}\right)-\nabla^{2}\boldsymbol{E}=\nabla\ln\left(\mu\right)\times\left(\nabla\times\boldsymbol{E}\right)-\frac{1}{c^{2}}\frac{\partial^{2}\boldsymbol{E}}{\partial t^{2}}, \tag{5.8}$$

where c is given by $c^{2}=1/\epsilon\mu$. From Eq. 5.3,

$$\nabla\circ\boldsymbol{E}=\nabla\circ\left(\frac{\epsilon\boldsymbol{E}}{\epsilon}\right)=-\boldsymbol{E}\circ\nabla\ln\left(\epsilon\right) \tag{5.9}$$

and therefore

$$\nabla^{2}\boldsymbol{E}-\frac{1}{c^{2}}\frac{\partial^{2}\boldsymbol{E}}{\partial t^{2}}+\nabla\ln\left(\mu\right)\times\left(\nabla\times\boldsymbol{E}\right)+\nabla\left(\boldsymbol{E}\circ\nabla\ln\left(\epsilon\right)\right)=0. \tag{5.10}$$

A similar result holds for the magnetic field. Finally, if the medium is homogeneous or approximately so, the logarithmic terms can be neglected, which gives the vector wave equations

$$\nabla^{2}\boldsymbol{E}-\frac{1}{c^{2}}\frac{\partial^{2}\boldsymbol{E}}{\partial t^{2}}=0,\quad\nabla^{2}\boldsymbol{H}-\frac{1}{c^{2}}\frac{\partial^{2}\boldsymbol{H}}{\partial t^{2}}=0. \tag{5.11}$$

The assumption of homogeneity is a good approximation for weakly scattering or very thin inhomogeneous specimens which are discussed in section 5.8. In Eq. 5.11 the components of both the electric and the magnetic field obey the same differential equation. Hence it is sufficient to solve the scalar wave equation

$$\nabla^{2}u-\frac{1}{c^{2}}\frac{\partial^{2}u}{\partial t^{2}}=0, \tag{5.12}$$

where here, by definition, $u=u\left(\boldsymbol{r},t\right)$ represents a component of $\boldsymbol{E}$. Eq. 5.12 describes the time evolution of an electromagnetic disturbance in space. Alternatively, an electromagnetic disturbance can be described in the space-frequency domain using its Fourier representation [7]. Substituting

$$u\left(\boldsymbol{r},t\right)=\int U\left(\boldsymbol{r},\nu\right)\exp\left(i2\pi\nu t\right)d\nu \tag{5.13}$$

into Eq. 5.12 leads to the *Helmholtz equation*

$$\nabla^{2}U+\frac{\left(2\pi\nu\right)^{2}}{c^{2}}U=0 \tag{5.14}$$

or

$$\boxed{\nabla^2 U + k^2 U = 0,} \tag{5.15}$$

where $k = 2\pi\nu/c$. Elementary solutions to Eq. 5.15 are given by

$$U_{\boldsymbol{s}}(\boldsymbol{r}) = A \exp(ik\boldsymbol{s}\boldsymbol{r}), \tag{5.16}$$

where $A \in \mathbb{C}$ is constant and $\boldsymbol{s}$ is a unit vector indicating the propagation direction. These harmonic solutions are periodic in the propagation direction with spatial period λ, i.e. $U(r+\lambda) = U(r)$. It follows that $\lambda k = 2\pi$. Substituting $k = 2\pi\nu/c$, the *dispersion relation*

$$c = \lambda\nu \tag{5.17}$$

is obtained. In the development so far, it was *assumed* that light is an electromagnetic wave. As a historical note, by comparing experimentally obtained values for the vacuum permittivity ϵ_0 and permeability μ_0 with those obtained for the speed of light c_0 and using the above obtained relation $c_0^2 = 1/\epsilon_0\mu_0$, Maxwell concluded that this is the case [24]. In a material medium the speed of light is slower than in vacuum and given by

$$c^2 = \frac{1}{\epsilon\mu} = \frac{1}{\epsilon_0\mu_0\epsilon_r\mu_r} = \frac{c_o^2}{n^2} \tag{5.18}$$

where $\epsilon = \epsilon_o\epsilon_r$ and $\mu = \mu_o\mu_r$. The material properties are contained in the relative permittivity ϵ_r and the relative permeability μ_r, respectively. It is commonplace in the optics literature to describe the material properties by the *refractive index* $n = \sqrt{\epsilon_r\mu_r}$. For a time-independent inhomogeneous medium the refractive index is a function of space. Therefore the Helmholtz equation for a wave propagating in an inhomogeneous medium is given by

$$\nabla^2 U(\boldsymbol{r}) + k_0^2 n^2(\boldsymbol{r}) U(\boldsymbol{r}) = 0, \tag{5.19}$$

where $k_0 = 2\pi/\lambda$. Defining the *scattering potential* $V(\boldsymbol{r}) = k_0^2\left(n^2(\boldsymbol{r}) - 1\right)$ [5], the inhomogeneous Helmholtz equation can be written in the form

$$\boxed{\nabla^2 U(\boldsymbol{r}) + k_0^2 U(\boldsymbol{r}) = -V(\boldsymbol{r}) U(\boldsymbol{r}).} \tag{5.20}$$

Solving Eq. 5.20 allows to predict the radiation pattern emanating from a given three-dimensional spatial refractive index distribution or its corresponding scattering potential[1]. In *optical diffraction tomography* (ODT) [5], the inverse problem is of interest: given information about $U(\boldsymbol{r})$, the refractive index or scattering potential is recovered [5, 34].

5.2 Intensity of an electromagnetic wave

Differences in the components of the electric field $\boldsymbol{E}$ and the magnetic field $\boldsymbol{B}$ occur as a result of Maxwell's equations and underlying boundary conditions. Substituting an elementary solutions, also called *plane wave solutions*, of the wave equation of the form

$$\boldsymbol{E}(\boldsymbol{r}, t) = \boldsymbol{\mathcal{E}} \exp(ik\boldsymbol{s}\boldsymbol{r}) \exp(-i2\pi\nu t) \tag{5.21}$$

[1] Compare the Foldy-Lax scattering theory in section 2.6.

and

$$\boldsymbol{H}(\boldsymbol{r},t) = \boldsymbol{\mathcal{H}} \exp(ik\boldsymbol{s}\boldsymbol{r}) \exp(-i2\pi\nu t), \tag{5.22}$$

where $\boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{H}} \in \mathbb{C}^3$ are constant, into Eq. 5.1 leads to

$$ik\boldsymbol{s} \times \boldsymbol{E}(\boldsymbol{r},t) = i2\pi\nu\mu\boldsymbol{H}(\boldsymbol{r},t). \tag{5.23}$$

Thus the magnetic field can be obtained from the electric field by

$$\boldsymbol{H}(\boldsymbol{r},t) = \sqrt{\tfrac{\epsilon}{\mu}}\boldsymbol{s} \times \boldsymbol{E}(\boldsymbol{r},t). \tag{5.24}$$

This result holds for general solutions of the wave equation since, as a result from Fourier analysis [7], any well-behaved function can be written as a linear combination of the elementary solutions in Eqs. 5.21 and 5.22. The magnetic field is therefore orthogonal to the electric field. Together with the divergence equations (Eqs. 5.3 and 5.4), which show that $\boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{H}}$ are orthogonal to the propagation direction $\boldsymbol{s}$, it follows that electromagnetic waves are transverse waves.

The electric and magnetic field define the *Poynting vector*

$$\boldsymbol{S}(\boldsymbol{r},t) = \boldsymbol{E}(\boldsymbol{r},t) \times \boldsymbol{H}(\boldsymbol{r},t) = \sqrt{\tfrac{\epsilon}{\mu}} |\boldsymbol{E}(\boldsymbol{r},t)|^2 \boldsymbol{s}, \tag{5.25}$$

which represents the direction and magnitude of energy flow per unit area and unit time. Since electromagnetic waves in the visible and x-ray spectral range oscillate beyond terahertz frequencies, existing electronic detectors can only record the time-integrated Poynting vector. The *intensity* of an electromagnetic wave is defined by the time-integrated magnitude of the Poynting vector

$$I(\boldsymbol{r}) = \langle |\boldsymbol{S}| \rangle = \frac{1}{T} \int_0^T \sqrt{\tfrac{\epsilon}{\mu}} |\boldsymbol{E}(\boldsymbol{r},t)|^2 \, dt, \tag{5.26}$$

where T is the detector integration time. For a scalar monochromatic wave with harmonic time dependence

$$I(\boldsymbol{r}) = \langle |\boldsymbol{S}| \rangle = \frac{1}{T} \int_0^T \sqrt{\tfrac{\epsilon}{\mu}} |U(\boldsymbol{r}) \exp(-i2\pi\nu t)|^2 \, dt = \sqrt{\tfrac{\epsilon}{\mu}} |U(\boldsymbol{r})|^2. \tag{5.27}$$

The term $\sqrt{\epsilon/\mu}$ is a proportionality constant and may be absorbed into U. It is therefore appropriate to define the *optical intensity*

$$I(\boldsymbol{r}) = |U(\boldsymbol{r})|^2. \tag{5.28}$$

5.2.1 Photoelectric counting statistics

In the previous section, a classical model for the optical intensity was discussed. Since the detection of optical intensity is based on the photoeffect [14], the classical model needs to be extended to account for the quantum mechanical nature of photodetection. A semi-classical theory proposed by Mandel is reviewed here, which explains the effect of shot noise [30]. The transition from deterministic classical fields to probabilistic electron

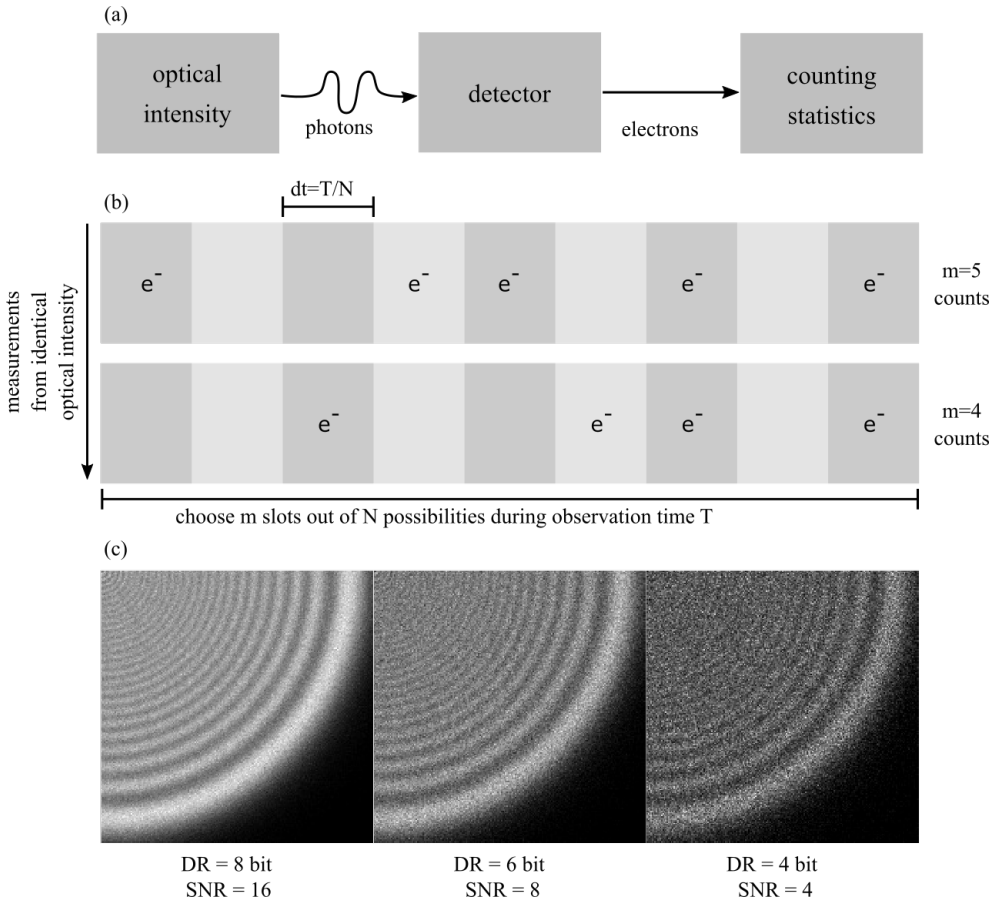


Figure 5.1: Photodetection as a statistical process: (a) due to the quantum mechanics of the photoeffect, the conversion from photons to electrons is probabilistic, (b) the underlying discrete random process is binomial and can be approximated by a Poisson distribution, (c) simulated measurements with Poisson noise of an optical intensity with varying dynamic range (DR) and signal-to-noise ratio (SNR).

counting distributions is illustrated in panel (a) of Figure 5.1. The absorption of photons in a semiconducting device allows bound electrons to overcome the band gap and be transported under the influence of an external electrostatic field [18]. It is assumed that the observation period T is divided into N time intervals $\Delta t = T/N$ (see panel (b) in Figure 5.1), which are small enough for the probability of absorbing two photons within time Δt to be negligible. The probability p of absorbing a photon within any time interval is modeled to be proportional to the optical intensity I and the length of the time interval, i.e. $p \sim I\Delta t$. Assuming a linear detector response, this can be refined to $p = \alpha I \Delta t$, where α is a proportionality constant related to the *quantum efficiency* of the detector (discussed further at the end of this subsection) [48]. Then the number of photon counts n is a discrete binomial random variable with N trials conditional on the optical intensity which can be approximated by a continuous Poisson distribution, i.e.

$$\begin{aligned}
P(n\,|\,I) &= \binom{N}{n} p^n (1-p)^{N-n} \\
&= \binom{N}{n} (\alpha I \Delta t)^n (1-\alpha I \Delta t)^{N-n} \\
&= \frac{N!}{n!\,(N-n)!} \left(\frac{\alpha I T}{N-\alpha I T}\right)^n \left(1-\frac{\alpha I T}{N}\right)^N \\
&= \frac{\left(1-\frac{\alpha I T}{N}\right)^N}{n!} \left[\prod_{k=0}^{n-1} (N-k)\right] \left(\frac{\alpha I T}{N-\alpha I T}\right)^n \\
&= \frac{\left(1-\frac{\alpha I T}{N}\right)^N}{n!} \left[\prod_{k=0}^{n-1} \frac{N-k}{N-\alpha I T} \alpha I T\right] \\
&\approx \frac{\exp(-\alpha I T)}{n!} (\alpha I T)^n ,
\end{aligned} \tag{5.29}$$

where in the last step $N \gg n$, $\alpha I T$ was assumed. For a quantum efficiency of $\alpha = 1$ and a unit observation time interval this result becomes

$$\boxed{P(n\,|\,I) \approx \frac{I^n}{n!} \exp(-I),} \tag{5.30}$$

which is a special case of *Mandel's formula* for photoelectric counting statistics [48]. For the Poisson distribution

$$\langle n \rangle = \left\langle (n - \langle n \rangle)^2 \right\rangle = I, \tag{5.31}$$

where brackets denote expectation values; this means for a Poisson distribution the expectation value of the observed number of counts equals its variance. Therefore the signal to noise ratio (SNR) for photoelectric conversion, defined here as expectation value over standard deviation of n, is

$$\mathrm{SNR} = \frac{\langle n \rangle}{\sqrt{\left\langle (n - \langle n \rangle)^2 \right\rangle}} \sim \sqrt{I}, \tag{5.32}$$

where we used a proportionality sign, which remains valid for quantum efficiencies smaller than 1. A simulated detected optical intensity with varying SNR and underlying Poisson noise is shown in panel (c) of Fig. 5.1. A near-field propagated pinhole diffraction pattern (see section 5.3) was used as an optical intensity. The Poisson measurements were simulated for maximum intensity signals $I_{max} = 2^8$, 2^6 and 2^4 assuming $\alpha = 1$ and $T = 1$.

Note on quantum efficiency

The quantum efficiency η is defined here as

$$\eta = \frac{N_e}{N_p} \tag{5.33}$$

where N_e is the number of electrons produced and N_p is the number of photons absorbed in a given time interval. For visible light CMOS and CCD detectors α is typically smaller than 100 %. Due to the silicon band gap of 1.1 eV, a visible light photon creates at most one photoelectron. For x-rays η can be larger, since multiple photoelectrons are generated by a single x-ray photon. The proportionality constant α used in Eq. 5.30 is related to the quantum efficiency via [18]

$$\alpha = \frac{\eta A}{h\nu}, \tag{5.34}$$

where A is the area over which the optical intensity is integrated, h is Planck's constant and ν is the frequency of the radiation.

Note on independence of photon arrivals

The derivation of the Poisson statistic implied independence of photon arrival events and a stationary optical intensity. If intensity fluctuations are given, the number of counts has to be calculated as a weighted average over all possible values that the optical intensity I can attain, i.e.

$$\begin{aligned} P(n) &= \int P(n \mid I)\, p(I)\, dI \\ &= \int \frac{I^n}{n!} \exp(-I)\, p(I)\, dI, \end{aligned} \tag{5.35}$$

which is a more general form of Mandel's equation. For a stabilized single-mode laser it is valid to assume $p(I) = \delta(I - I_0)$,where I_0 is constant. This essentially results again in Eq. 5.30

$$P(n) = \frac{I_0^n}{n!} \exp(-I_0). \tag{5.36}$$

For other sources the photocounting statistics generally depend on the degree of coherence of the source [18].

5.3 Angular spectrum propagation

5.3.1 Derivation

In vacuum, where the refractive index across the electromagnetic spectrum is $n = 1$, and in homogeneous media, where the refractive index is independent of space, an integral-form solution for the Helmholtz equation (Eq. 5.15) can be obtained. A two-dimensional coordinate system (x, z) is assumed with the optical axis pointing in z-direction. Suppose a scalar wave field is given in a plane at $z = 0$ in the form

$$U(x, 0) = \psi(x, 0) \exp(iks_x x), \tag{5.37}$$

where s_x is the direction cosine with the $x-$axis of the tilted plane wave envelope $\exp(iks_x x)$. Writing Eq. 5.37 in a factorized form saves work for later purposes. Inserting the latter expression into the Helmholtz equation and canceling exponentials gives

$$\nabla^2\psi + 2iks_x \frac{\partial\psi}{\partial x} - (ks_x)^2 + k^2\psi = 0.$$

Now Fourier transform ψ with respect to x, i.e.

$$\widetilde{\psi}(f_x, z = 0) = \mathcal{F}\{\psi(x, 0)\} = \int \psi(x, z = 0) \exp(-i2\pi f_x x)\, dx, \tag{5.38}$$

and rearrange terms to get an ordinary differential equation (ODE)

$$\frac{d^2\widetilde{\psi}}{dz^2} = -k^2 \left[1 - (s_x - \lambda f_x)^2\right] \widetilde{\psi}. \tag{5.39}$$

The solution to this differential equation is found to be

$$\widetilde{\psi}(f_x, z) = \widetilde{\psi}(f_x, z = 0) \exp\left[ikz\sqrt{1 - (s_x - \lambda f_x)^2}\right]. \tag{5.40}$$

Inverse Fourier transform the latter to get the solution

$$\psi(x, z) = \mathcal{F}^{-1}\{\mathcal{F}\{\psi(x, 0)\} H(f_x, s_x, z)\}, \tag{5.41}$$

where

$$H(f_x, s_x, z) = \exp\left[\pm ikz\sqrt{1 - (s_x - \lambda f_x)^2}\right] \tag{5.42}$$

is the *coherent transfer function* of free space[2]. In general, the plus sign models forward propagation the minus sign includes backward propagation. In what follows we drop the minus sign and only account for forward propagation. An analog derivation for a three-dimensional coordinate system (x, y, z) yields

$$\psi(x, y, z) = \mathcal{F}^{-1}\{\mathcal{F}\{\psi(x, y, 0)\} H(f_x, f_y, s_x, s_y, z)\}, \tag{5.43}$$

[2]In a medium replace the corresponding refractive index in $k = n2\pi/\lambda$.

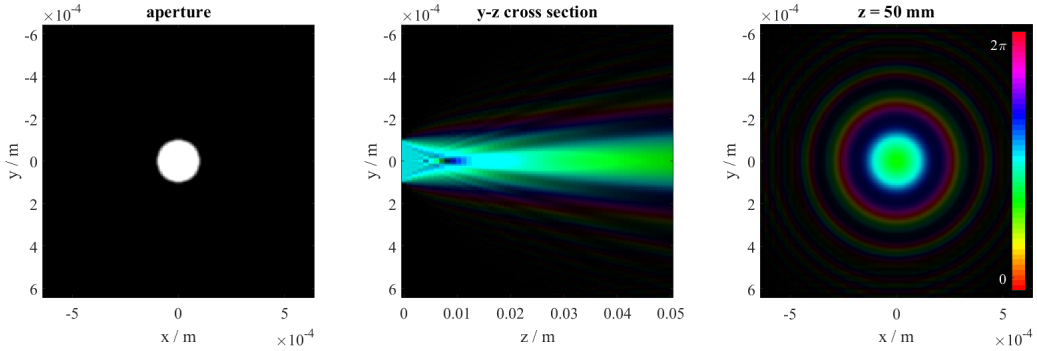


Figure 5.2: ASPW propagation: (left) circular aperture in $xy-$plane at $z = 0$, (middle) section of the beam in $xz-$plane in the range $0 \leq z \leq 0.5\,\mathrm{m}$, (right) diffracted beam in $xy-$plane at $z = 0.5\,\mathrm{m}$. The colorbar indicates the phase of the beam as hue.

where

$$H\left(f_x, f_y, s_x, s_y, z\right) = \exp\left[ik\sqrt{1-\left(s_x-\lambda f_x\right)^2-\left(s_y-\lambda f_y\right)^2}\right]. \tag{5.44}$$

Finally, the tilted plane wave solution of the propagation problem in Eq. 5.37 for a three-dimensional coordinate system is given by

$$U\left(x,y,z\right) = \mathcal{F}^{-1}\left\{\mathcal{F}\left\{\psi\left(x,y,0\right)\right\}H\left(f_x,f_y,s_x,s_y,z\right)\right\}\exp\left[ik\left(s_x x + s_y z\right)\right]. \tag{5.45}$$

The latter form is called *angular spectrum of plane wave* (ASPW) solution for $s_x = s_y = 0$ [**?**]. If s_x or s_y are nonzero, we refer to the *tilted angular spectrum of plane wa*ve (tASPW) solution.

Example

An illustration of the ASPW is given in Fig. 5.2. Here a simulated circular aperture of size $200\,\mu$m is numerically propagated over a range $0 \leq z \leq 0.5\,\mathrm{m}$ at a wavelength of $\lambda = 633\,\mathrm{nm}$ using the ASPW method. The spread in the angular distribution of the beam upon interacting with the rectangular aperture is a manifestation of an uncertainty principle: confining the spatial extent of a beam results in a spread in its angular distribution - which is to say the field diffracts upon interaction with the aperture. Numerical implementation details and digital sampling conditions for the ASPW propagation method (`aspw.m`) are adapted from [32]. This ASPW method imposes a bandlimit in the transfer function to prevent sampling artefacts due to circular convolution artefacts, which typically arise when using the ASPW over long propagation distances.

```
1
2 N = 2^9;
3 dx = 5e-6;
4 L = N*dx;
5 x = linspace(-L/2,L/2,N);
6 [X,Y] = meshgrid(x);
```

```
wavelength = 633e-9;

% define aperture
w = 200e-6;
% aperture = rect(X/(L/2)).*rect(Y/(L/2));
aperture = circ(X,Y,w);
aperture = imfilter(aperture,...
            fspecial('disk',2),'same');

figure(1)
h = subplot(1,3,1);
imagesc(x,x,aperture)
xlabel('x / m'),ylabel('y / m')
title('aperture')
axis image, colormap gray
%%
M = 50;
z = linspace(0,10e-2,M);
dz = z(2) - z(1);

aperture_diffracted = zeros(N,N,M);
tic
alpha =0.6;
for k = 1:length(z)
    aperture_diffracted(:,:,k) = ...
        exp(-1i*2*pi/wavelength*z(k))*...
        aspw(aperture, z(k),wavelength,L);

    subplot(1,3,2)
    hsvplot_axis(z,x,...
        squeeze(aperture_diffracted(:,N/2,:)),...
        'intensityScale', [0,alpha])
    axis square, title('y-z cross section')

    subplot(1,3,3)
    hsvplot_axis(x,x,...
        aperture_diffracted(:,:,k),...
        'intensityScale', [0,alpha])
    title(['z = ', num2str(round(z(k)*1e3)),' mm'])
    drawnow
end
% reset colormap of panel 1
h.Colormap = gray;
toc
```

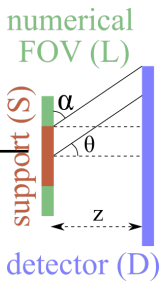
Listing 5.1: aspwExample.m (Matlab)

Exercise: Compute the spatial standard deviation

Exercise

$$\sigma_x(z) = \left(\frac{\int (x - \langle x \rangle)^2 |U(x,y,z)|^2 \, dx\, dy}{\int |U(x,y,z)|^2 \, dx\, dy} \right)^{1/2} \tag{5.46}$$

as a function of propagation distance. Can you find the asymptotic limit for $\sigma_x(0) \cdot \sigma_x(z)$?



5.3.2 Numerical implementation of ASPW

Consider the geometry on the margin of this page. A wave propagating from the source plane (left, green) to the destination plane (blue, typically a detector) is subject to diffraction, implying a spread of the signal towards the edge of the numerical field of view. A critical question is what exactly happens to signals that diffract at angles larger than the solid angle spanned by the detector. Our hope would be that these signals end up in some kind of numerical abyss and can safely be neglected. But unfortunately this is not the case. It turns out that the signals leaving one side of the numerical field of view revisit on the other side in form of circular convolution, where they cause nonphysical interferences and artefacts. Our goal here is to derive a filter that allows us to correct for these artefacts. For simplicity, we restrict this analysis to one dimension[3] and assume that the source and destination size are the same $L = D$. The field is non-zero only within a finite support of size S. Under these conditions, we have

$$t \equiv \tan\theta = \frac{\sin\theta}{\cos\theta} = \frac{D - S}{2z}. \tag{5.47}$$

Using the direction cosine [19] $\cos\alpha = \sin\alpha = \lambda f_x \equiv u$, we have

$$t = \frac{u}{\sqrt{1 - u^2}} \tag{5.48}$$

or

$$u = \sqrt{\frac{t^2}{1 + t^2}} = \frac{|D - S|}{\sqrt{4z^2 + (D - S)^2}}. \tag{5.49}$$

From this we find the spatial frequency limit

$$\boxed{f_{\max} = \frac{1}{\lambda} \frac{|D - S|}{\sqrt{4z^2 + (D - S)^2}}.} \tag{5.50}$$

This result states that in order to avoid circular convolution artefacts, we need to restrict the angular spectrum method to spatial frequencies $|f_x| \leq f_{\max}$. The following code gives an example where we assume $S = L/2$, for which the limit frequency simplifies to

$$f_{\max} = \frac{1}{\lambda} \frac{L}{\sqrt{16z^2 + L^2}}. \tag{5.51}$$

[3] A more general derivation is discussed in [32], where the resulting filter takes on an elliptical shape.

```
function [u, H] = aspw(u, z, lambda, L)
% ASPW wave propagation
% u = aspw(u,z,lambda,L)
% u: field distribution at z = 0
% (u is assumed to be square, i.e. N x N)
% z: propagation distance
% lambda: wavelength
% L: total size [m] of
% following: Matsushima et al.,
% "Band-Limited Angular Spectrum Method for
% Numerical Simulation of Free-Space
% Propagation in Far and Near Fields",
% Optics Express, 17 (22), pp. 19662-19673
% (2009), https://doi.org/10.1364/OE.17.019662

k = 2*pi/lambda;
N = size(u,1);
[Fx, Fy] = meshgrid((-N/2:N/2-1)/L);
f_max = L/(lambda*sqrt(L^2+16*z^2));
W = circ(Fx, Fy, 2*f_max); % low pass filter
H = exp(1i*k*z*sqrt(1-(Fx*lambda).^2-(Fy*lambda).^2));
u = ifft2c(bsxfun(@times, fft2c(u), H.*W));

end
```

Listing 5.2: aspw.m (Matlab)

Exercise: Write a test case for the aspw.m function. Test the method's accuracy for various propagation geometries with and without filter function. Which improvements to the method are described in [32]? Exercise

---

5.4 Rayleigh-Sommerfeld diffraction formula

The solution to the propagation problem in section 5.3 for illumination in the z-direction was written in the form

$$\psi(x,y,z) = \mathcal{F}^{-1}\left\{\mathcal{F}\left\{\psi(x,y,0)\right\} H(f_x,f_y,z)\right\}, \tag{5.52}$$

where

$$\boxed{H(f_x,f_y,z) = \exp\left[ikz\sqrt{1-(\lambda f_x)^2-(\lambda f_y)^2}\right]} \tag{5.53}$$

is the transfer function for propagation in a homogeneous medium[4]. The inverse Fourier transform of the transfer function, namely the impulse response of free space, is given by

[4] For simplicity we assume $s_x = s_y = 0$.

[43]

$$h(x,y,z) = \frac{\exp(ik|\boldsymbol{r}|)}{i\lambda|\boldsymbol{r}|}\frac{z}{|\boldsymbol{r}|}\left(1-\frac{1}{ik|\boldsymbol{r}|}\right), \tag{5.54}$$

where $|\boldsymbol{r}| = \sqrt{x^2+y^2+z^2}$. The last term in parenthesis can be approximated as $\left(1-\frac{1}{ik|\boldsymbol{r}|}\right) \approx 1$ given that $|\boldsymbol{r}| \gg \lambda$. In particular, if the propagation distance z is much larger than a wavelength this condition is satisfied, so that

$$h(x,y,z) \approx \frac{\exp(ik|\boldsymbol{r}|)}{i\lambda|\boldsymbol{r}|}\frac{z}{|\boldsymbol{r}|}. \tag{5.55}$$

Combining this approximate impulse response and the convolution theorem, Eq. 5.52 simplifies to the *Rayleigh-Sommerfeld diffraction integral*

$$\psi(x,y,z) = \int \psi(x',y',0)\frac{\exp[ik|\boldsymbol{r}-\boldsymbol{r}'|]}{i\lambda|\boldsymbol{r}-\boldsymbol{r}'|}\frac{z}{|\boldsymbol{r}-\boldsymbol{r}'|}d\boldsymbol{r}', \tag{5.56}$$

where the primed variables denote source coordinates, $\boldsymbol{r}' = \begin{bmatrix} x' & y' & 0 \end{bmatrix}^T$, and the unprimed variables denote observation plane coordinates, $\boldsymbol{r} = \begin{bmatrix} x & y & z \end{bmatrix}^T$. While the ASPW method in its simplest form (Eq. 5.52) is useful only to compute scalar fields between parallel planes orthogonal to the propagation direction, the Rayleigh-Sommerfeld diffraction integral can also be used between non-parallel planes and freeform geometries. However, its numerical evaluation, for instance by a simple for-loop, is computationally less efficient than the ASPW method, which can be evaluated at the cost of two fast Fourier transforms.

Exercise: Implement the Rayleigh-Sommerfeld integral using a for-loop with equal geometric parameters as in the example in section 5.3 (for a single propagation distance $z = 50\,\text{cm}$). Compare the speed of the angular spectrum method with the Rayleigh-Sommerfeld integral. Exercise

---

5.5 Fresnel approximation

If the square root term in the exponent of the Rayleigh-Sommerfeld integral is expanded into a Taylor series (here only the x-variable is considered)

$$k\sqrt{z^2+(x-x')^2} = kz\left(1+\frac{1}{2}\left(\frac{x-x'}{z}\right)^2+\frac{1}{8}\left(\frac{x-x'}{z}\right)^4+\ldots\right), \tag{5.57}$$

simpler expressions can be derived. Retaining only second-order terms and approximating the distances in the denominator by $|\boldsymbol{r}-\boldsymbol{r}'| \approx z$ of Eq. 5.56, the *Fresnel approximation*

$$\psi(x,y,z) = \frac{\exp(ikz)}{i\lambda z}\int \psi(x',y',0)\cdot\exp\left[i\frac{k}{2z}\left((x-x')^2+(y-y')^2\right)\right]dx'dy' \tag{5.58}$$

is obtained. This is the so-called convolution form of the Fresnel integral (CV-FR). Expanding the terms in the exponent we get

$$\psi(x,y,z) = \frac{\exp(ikz)}{i\lambda z}\exp\left[i\frac{k}{2z}\left(x^2+y^2\right)\right]\int\psi(x',y',0)\cdot \exp\left[i\frac{k}{2z}\left(x'^2+y'^2\right)\right]\exp\left[-i\frac{2\pi}{\lambda z}\left(xx'+yy'\right)\right]dx'dy', \tag{5.59}$$

which is the single Fourier transform form of the Fresnel integral (sFT-CV). From the Taylor series above it is seen that this approximation is valid when

$$\frac{\pi}{4\lambda}\frac{(x-x')^4}{z^3} \ll 1 \tag{5.60}$$

for all points in the source plane x'.

In theory, the CV-FR and sFT-CV methods are equivalent. In practice, they differ in various ways. Let us first consider an operational perspective. The CV-FR is numerically implemented via convolution with the kernel

$$h_{\text{CV-FR}}(x,y) = \frac{\exp(ikz)}{i\lambda z}\exp\left[i\frac{k}{2z}\left(x^2+y^2\right)\right], \tag{5.61}$$

which is equivalent to the following three steps:

CV-FR

1. Fourier transform $\psi(x',y',0)$ (practically via FFT).
2. Multiply with the transfer function
$$H(f_x,f_y) = \mathcal{F}\left[h_{\text{CV-FR}}(x,y)\right] = \exp(ikz)\exp\left[-i\pi\lambda z\left(f_x^2+f_y^2\right)\right]. \tag{5.62}$$
3. Inverse Fourier transform (practically via iFFT) to get
$$\psi(x,y,z) = \mathcal{F}^{-1}\left[\mathcal{F}\left[\psi(x',y',0)\right]\cdot H(f_x,f_y)\right]. \tag{5.63}$$

In contrast, the sFT-FR method uses the following three steps:

sFT-FR

1. Pre-multiply $\psi(x',y',0)$ with the quadratic phase
$$Q_{\text{in}} = \exp\left[i\frac{k}{2z}\left(x'^2+y'^2\right)\right].$$
2. Fourier transform (practically via FFT).
3. Post-multiply the result with quadratic phase $Q_{\text{out}} = \exp\left[i\frac{k}{2z}\left(x^2+y^2\right)\right]$ (and an additional factor of $\exp(ikz)/i\lambda z$).

To summarize, the CV-FR method uses a transfer function approach, involving two Fourier transform operations. The sFT-FR method uses only a single Fourier transform. When only the intensity in the observation plane is often interest, step 3 of the sFT-FR method can be neglected.

Sampling

The difference in the number of Fourier transforms involved affects the sampling of the CV-FR and the sFT-FR methods. In the CV-FR method, the sampling in the source and target planes are equal, because the forward and inverse Fourier transform operation cancel any scaling effects. In the sFT-FR method, only a single Fourier transform is used, which results in a scaling of the target coordinates as compared to the source coordinates. To understand this scaling effect, let us discretize the source coordinates

$$x' = \lambda \cdot z \cdot \frac{p}{M \cdot \Delta x} \tag{5.64}$$

and target coordinates

$$x = m \cdot \Delta x \tag{5.65}$$

in the sFT-FR method (compare Eq. 5.59), where $m = 1, \ldots, M$. Then we have have[5]

$$\exp\left[-i\frac{2\pi}{\lambda z}xx'\right] = \exp\left[-j2\pi\frac{pm}{M}\right], \tag{5.66}$$

the latter kernel of which was previously discussed in the context of the discrete Fourier transform (2.376). In particular, let us denote the source field of view with

$$L = M \cdot \Delta x' \tag{5.67}$$

and the target plane field of view (typically a detector) with

$$D = M \cdot \Delta x, \tag{5.68}$$

The source pixel pitch $\Delta x'$ is found by substituting $p = 1$ into Eq. 5.64, which gives

$$\Delta x' = \frac{\lambda \cdot z}{M \cdot \Delta x}. \tag{5.69}$$

This equation can be rewritten as

$$\boxed{\Delta x' = \frac{\lambda z}{D}.} \tag{5.70}$$

In Eq. we can also isolate the target pixel pitch on the left hand side to get

$$\Delta x = \frac{\lambda \cdot z}{M \cdot \Delta x'}, \tag{5.71}$$

which may be rewritten as

$$\Delta x = \frac{\lambda \cdot z}{L} \tag{5.72}$$

or

$$\boxed{L = \frac{\lambda \cdot z}{\Delta x}.} \tag{5.73}$$

[5]Only one dimension is considered here. The derivation for the y-direction is similar.

We refer to Eqs. 5.70 and 5.73 as *reciprocity relations*: The smallest element in the specimen plane, the source pixel pitch $\Delta x'$, depends on the largest element in the target plane, typically the size of the detector D. Likewise, the largest element in the specimen plane, the numerical field of view L, depends on the smallest element in the target plane, the detector pixel pitch Δx. Equations 5.70 and 5.73 are crucial for the discussion of lensless imaging in chapter **??**.

Code

The following code illustrates an implementation of Fresnel propagation via the sFT-FR method.

```
function [r, Qin, Qout] = fresnelPropagator(u, z, lambda, L)
% Fresnel propagator
% u = fresnelPropagator(u,z,lambda,L)
% u: field distribution at z = 0
% (u is assumed to be square, i.e. N x N)
% z: propagation distance
% lambda: wavelength
% L: total size [m] of source field
% assume square array input,
% even number of pixel per dimension

% get source (primed) coordinates
N = size(u,2);
dxp = L / N;
xp = (-N/2:N/2-1)*dxp;
[Xp,Yp] = meshgrid(xp);
% get target (unprimed) coordinates
dx = lambda * z / L;
x = (-N/2:N/2-1)*dx;
[X,Y] = meshgrid(x);
% get wavenumber
k = 2*pi/lambda;
% step 1: quadratic phase inside integral
Qin = exp(1i * k/(2*z) * (Xp.^2 + Yp.^2));
% step 2: Fourier transform
r = fft2c(u .* Qin);
% step 3: multiply quadratic phase
% outside integral
Qout = exp(1i * k/(2*z) * (X.^2 + Y.^2));
r = Qout.*r;
end
```

Listing 5.3: fresnelPropagator.m (Matlab)

Exercise: Implement the CV-FR method. Hint: the implementation of the CV-FR Exercise

method is similar to the ASPW method discussed in section 5.3, but the transfer function in Eq. 5.62 is used.

5.6 Fraunhofer approximation

If only first-order terms in Eq. 5.57 are retained, the *Fraunhofer approximation*

$$\psi(x,y,z) = \frac{\exp(ikz)}{i\lambda z} \exp\left[i\frac{k}{2z}\left(x^2+y^2\right)\right] \cdot \int \psi(x',y',0) \exp\left[-i\frac{2\pi}{\lambda z}(xx'+yy')\right] dx'dy' \tag{5.74}$$

follows, which is a good approximation given that

$$\frac{\pi}{\lambda}\frac{(x-x')^2}{z} \ll 1. \tag{5.75}$$

In practice, the conditions in Eqs. 5.60 and 5.75 are rather conservative. Less stringent conditions can be formulated using the *Fresnel number* defined as

$$F = \frac{x'^2}{\lambda z}, \tag{5.76}$$

which is seen to be on the order of the quadratic phase exponential inside the integral in Eq. 5.59. In practice we substitute the maximum lateral dimensions for x', for instance the diameter of a circular pinhole. If $F \sim 1$, the Fresnel integral is a good approximation. For $F \ll 1$ the Fraunhofer integral is typically used. Using the projection approximation discussed later in this chapter (Eq. 5.85) with $\Delta x = a = x'$, it is seen that for $F \gg 1$ diffraction effects can be neglected. In the latter case ray optics is an appropriate choice to model light propagation.

While the Fresnel number only contains the source geometry of the diffracting aperture, the more restrictive condition 5.60 predicts deviations of the Fresnel integral as compared to the Rayleigh-Sommerfeld integral as a function of the difference between source and observation plane coordinates. In particular, this leads to artifacts using the Fresnel integral in numerical propagation problems with large detectors. For further details the reader is referred to [41].

Finally we note that the Fraunhofer integral, like the Fresnel integral, is subject to the reciprocity relations (compare Eqs. 5.70 and 5.73) derived in the last subsection, when computed by means of the fast Fourier transform.

5.7 Thin lenses

While lenses are not the center of this chapter, we are in a good place to shortly discuss the ideal transmission function of a thin lens. Consider the spatial filter arrangement in Fig. 5.3. From the left an aberrated wavefront propagates to the right. The first lens on the left focus the aberrated wavefront. In the focal plane a pinhole (PH) select only

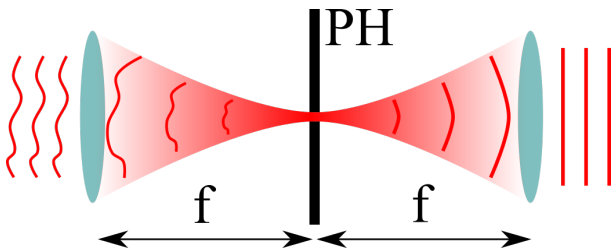


Figure 5.3: Spatial filter configuration consisting of two lenses and a pinhole (PH). The pinhole aperture is confocal to the two lenses. An aberrated wavefront from the results in an extended focal spot. The pinhole aperture selects a small portion of the extended spot so that only the spherical portion of the original wavefront can reach the second lens, where the beam is collimated.

the central portion of the focal spot. Let us assume the pinhole has a wavelength-scale aperture, for which we approximate the pinhole transmission function by a delta function

$$t_{\mathrm{PH}}\left(x',y'\right)=\delta\left(x'\right)\cdot\delta\left(y'\right). \tag{5.77}$$

Inserting this into both the Fresnel and Fraunhofer integral we get

$$\psi\left(x,y,f\right)=\frac{\exp\left(ikf\right)}{i\lambda f}\exp\left[i\frac{k}{2f}\left(x^2+y^2\right)\right], \tag{5.78}$$

which is the factor preceeding both integrals in Eqs. 5.5 and 5.74. We see that the wave upstream of the lens $\psi\left(x,y,f\right)$ on the right in Fig. 5.3 is expanding; namely, a paraxial approximation to a diverging spherical wave. In order to collimate this divergent wave we need a lens that compensates the quadratic phase factor. Hence we immediately find the transmission function of a thin lens can be approximated by

$$t_{\mathrm{lens}}\left(x,y\right)=\exp\left[-i\frac{k}{2f}\left(x^2+y^2\right)\right]. \tag{5.79}$$

Practical lenses have aberrations, which are defined as the deviation from the above ideal thin lens. These aberrations can be incorporated into a modified lens transmission function

$$t_{\mathrm{lens}}^{'}\left(x,y\right)=\exp\left[-i\frac{k}{2f}\left(x^2+y^2\right)\right]\exp\left[iW\left(x,y\right)\right], \tag{5.80}$$

where $W\left(x,y\right)$ is the so-called *pupil function*.

5.8 Thin element approximation and axial resolution

In this section a condition is derived which allows to consider a specimen as *optically thin*. This condition simplifies the analysis of wave-matter interaction and allows to neglect multiple-scattering and diffraction effects inside a thin, three-dimensional sample. In this case, the sample is mathematically represented by a two-dimensional transmission

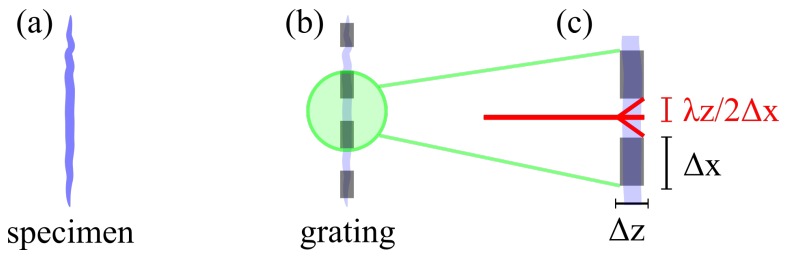


Figure 5.4: For the derivation of the projection approximation (Ineq. 5.85) the specimen (a) is considered as a grating (b, c) with spatial period $2\Delta x$ and thickness Δz. If the displacement due to diffraction of a coherent beam traversing the grating is smaller than its half period, then the specimen is considered as optically thin.

function, which is obtained by a projection of the refractive index along one spatial dimension. This is referred to as the *thin element approximation* (TEA) or *projection approximation* [39].

Consider a sample occupying a finite volume of axial length L. Its Fourier representation decomposes the sample into a linear combination of phase gratings. The highest spatial frequency in Fourier space corresponds to an effective harmonic grating with spatial period[6] $2\Delta x$, which in the context of imaging can be interpreted as the smallest resolvable scattering structure inside the specimen. Let the grating be located in the plane at $z = 0$. Assuming small angles, the first diffraction orders propagate under angles

$$\pm\alpha \approx \sin(\alpha) = \lambda/2\Delta x \tag{5.81}$$

with respect to the optical axis. To consider the object as optically thin, the displacement in x-direction at $z = \Delta z$ is required to be smaller than the smallest resolvable structure, i.e.

$$\tan(\alpha)\,\Delta z \approx \sin(\alpha)\,\Delta z = \lambda\Delta z/2\Delta x < \Delta x, \tag{5.82}$$

which leads to the condition

$$\sqrt{\frac{\lambda\Delta z}{2}} < \Delta x. \tag{5.83}$$

In addition, the illuminating beam itself diffracts according to its finite spatial extent. Let the size of the beam be denoted by a. Then the same argument as above leads to

$$\sqrt{\frac{\lambda\Delta z}{2}} < a. \tag{5.84}$$

Multiplying Ineqs. 5.83 and 5.84 leads to

$$\Delta z < \frac{2a\Delta x}{\lambda}, \tag{5.85}$$

[6] Here the full period is $2\Delta x$ and the half period Δx are discerned. This is in analogy with the definition of half period and full period resolution which should not be confused when assessing spatial resolution [?].

which is referred to here as the *projection approximation*. This condition limits the maximum thickness Δz under which multiple scattering effects inside a specimen may be neglected. A similar criterion for the maximum sample thickness can be derived using the inhomogeneous Helmholtz equation[7]. If the beam is focused down to the size of the achievable spatial resolution $\Delta x = \lambda/2\mathrm{NA}$, limited by the numerical aperture of the lens or detector of the optical system, then inequality 5.85 can be rewritten as

$$\Delta z < \frac{\lambda}{2\mathrm{NA}^2}, \tag{5.86}$$

where $\lambda/2\mathrm{NA}^2$ is the *depth of field* (DoF) or the *axial resolution* of the imaging system.

An optically thin specimen can mathematically be represented by means of a two-dimensional transmission function defined by

$$T(x,y) = \frac{\psi_n(x,y)}{\psi_0(x,y)}, \tag{5.87}$$

where ψ_n and ψ_0 are the waves in the presence and absence of the sample, respectively. Assuming plane wave illumination[8] and that diffraction effects can be neglected within the sample, the absorption and phase accumulated are proportional to the integrated optical path, i.e.

$$T(x,y) = \frac{\exp\left(ik\int n(x,y,z)\,dz\right)}{\exp(ikz)}, \tag{5.88}$$

where n is the generally complex-valued refractive index.

5.9 Free-space propagation in matrix form

5.10 Space variance

Exercise: In section 2.6 we discussed Foldy-Lax scattering theory. Compute the response to a point source upstream of the scattering medium in a plane downstream of the scattering medium (each in a single row of the computation window). Arrange each impulse response along the columns of a matrix. This is the the so-called *scattering matrix*. What conditions should the scattering medium satisfy so that adjacent columns are correlated? Exercise

---

[7] See supplementary information in [**?**]

[8] The derivation is not restricted to plane wave illumination. Any other type of illumination can be synthesized by a superposition of plane waves.

Chapter 6

Coherence

This chapter reviews the propagation and efficient numerical representation of partially coherent scalar waves.

6.1 Partially coherent scalar wave fields

The emission of light from excited atoms and accelerated charge is a statistical process. The fluctuations in wave fields can be quantified in terms of correlations in space and time. The study of these correlations is the subject of coherence theory [18]. Since the optical detectors available to date integrate wave fields over many periods of oscillation, it is desirable to formulate the study of coherence in terms of observable quantities which necessitates the use of time averages. The *mutual coherence function*

$$\Gamma\left(\boldsymbol{r}_1, \boldsymbol{r}_2, \tau\right) = \left\langle u^{\dagger}\left(\boldsymbol{r}_1, t\right) u\left(\boldsymbol{r}_2, t+\tau\right)\right\rangle_t, \tag{6.1}$$

measures the correlation of a wide-sense stationary[1] wave field $u\left(\boldsymbol{r}, t\right)$ at two different points in space and time [48]. Here the dagger denotes complex conjugation and the brackets denote averages over time. Since u satisfies the wave equation with respect to space and time, a set of wave equations is obeyed by the mutual coherence, i.e. [5]

$$\nabla_1^2 \Gamma\left(\boldsymbol{r}_1, \boldsymbol{r}_2, \tau\right) - \frac{1}{c^2} \frac{\partial^2 \Gamma\left(\boldsymbol{r}_1, \boldsymbol{r}_2, \tau\right)}{\partial \tau^2} = 0 \tag{6.2}$$

and

$$\nabla_2^2 \Gamma\left(\boldsymbol{r}_1, \boldsymbol{r}_2, \tau\right) - \frac{1}{c^2} \frac{\partial^2 \Gamma\left(\boldsymbol{r}_1, \boldsymbol{r}_2, \tau\right)}{\partial \tau^2} = 0, \tag{6.3}$$

where the Laplacians ∇_1^2 and ∇_2^2 yield second-order derivatives with respect to $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$, respectively.

Alternatively and equivalently, random wave fields may be regarded as a function in the space frequency domain by taking the Fourier transform with respect to time

[1] A wave whose statistical properties do only depend on separation in time is said to be *wide-sense stationary*.

separation τ of Eq. 6.1, which gives the *cross-spectral density (CSD) [?]*

$$W(\boldsymbol{r}_1, \boldsymbol{r}_2, \nu) = \int \Gamma(\boldsymbol{r}_1, \boldsymbol{r}_2, \tau) \exp(-i2\pi\tau\nu)\, d\tau. \tag{6.4}$$

Inserting Eq. 6.4 into Eqs. 6.2 and 6.3, it follows that the CSD obeys a set of Helmholtz equations, i.e.

$$\nabla_1^2 W(\boldsymbol{r}_1, \boldsymbol{r}_2, \nu) + k^2 W(\boldsymbol{r}_1, \boldsymbol{r}_2, \nu) = 0 \tag{6.5}$$

and

$$\nabla_2^2 W(\boldsymbol{r}_1, \boldsymbol{r}_2, \nu) + k^2 W(\boldsymbol{r}_1, \boldsymbol{r}_2, \nu) = 0. \tag{6.6}$$

The CSD can be decomposed into a series of coherent modes, i.e.

$$W(\boldsymbol{r}_1, \boldsymbol{r}_2, \nu) = \sum_n \lambda_n(\nu)\, U_n^\dagger(\boldsymbol{r}_1, \nu)\, U_n(\boldsymbol{r}_2, \nu), \tag{6.7}$$

where the modes U_n and eigenvalues λ_n are given by the integral equations

$$\int W(\boldsymbol{r}_1, \boldsymbol{r}_2, \nu)\, U_n(\boldsymbol{r}_1, \nu)\, d\boldsymbol{r}_1 = \lambda_n(\nu)\, U_n(\boldsymbol{r}_2, \nu) \tag{6.8}$$

and

$$\int U_m^\dagger(\boldsymbol{r}, \nu)\, U_n(\boldsymbol{r}, \nu)\, d\boldsymbol{r} = \delta_{mn}, \tag{6.9}$$

where δ_{mn} is the Kronecker delta [16]. Inserting Eq. 6.7 into Eq. 6.5 yields

$$\nabla^2 U_n(\boldsymbol{r}, \nu) + k^2 U_n(\boldsymbol{r}, \nu) = 0, \tag{6.10}$$

which shows that the modes $U_n(\boldsymbol{r}, \nu)$ satisfy a Helmholtz equation. This result is important because it allows to decompose a partially coherent wave field of any state of coherence into uncorrelated coherent modes, which can be propagated separately through an optical system[2]. For quasi-monochromatic waves, for which the frequency bandwidth is much smaller than the average frequency, i.e. $\Delta\nu \ll \overline{\nu}$, the CSD can approximately be written in the form

$$W(\boldsymbol{r}_1, \boldsymbol{r}_2, \nu) = J(\boldsymbol{r}_1, \boldsymbol{r}_2)\, \delta(\nu - \overline{\nu}), \tag{6.11}$$

where the spatial part is called *mutual intensity J*. The mutual intensity (MI) is independent of frequency (or wavelength) whereas the CSD contains the frequency as an independent variable. Hence the MI mode decomposition is written in the frequency-independent form

$$J(\boldsymbol{r}_1, \boldsymbol{r}_2) = \sum_n \lambda_n U_n^\dagger(\boldsymbol{r}_1)\, U_n(\boldsymbol{r}_2). \tag{6.12}$$

For a fully coherent wave the MI consists of only one mode

$$J_c(\boldsymbol{r}_1, \boldsymbol{r}_2) = U_1^\dagger(\boldsymbol{r}_1)\, U_1(\boldsymbol{r}_2). \tag{6.13}$$

[2]It is useful to think of modes as coherent waves, which independently propagate through an optical system and add incoherently upon detection, in analogy with polychromatic radiation. However, modes may contain contributions from different uncorrelated source emitters which appear in the same mode due to the geometry of the source. This is illustrated in section 6.5.

It is often useful to think of the CSD and MI as matrices, where $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ denote row and column indices [38]. The intensity of a coherent wave field can be written in terms of the mutual intensity as

$$I(\boldsymbol{r}) = J_c(\boldsymbol{r}) = \operatorname{diag}(J_c(\boldsymbol{r}_1, \boldsymbol{r}_2)) = U_1^\dagger(\boldsymbol{r}) U_1(\boldsymbol{r}), \tag{6.14}$$

where $\operatorname{diag}(...)$ indicates to set $\boldsymbol{r} = \boldsymbol{r}_1 = \boldsymbol{r}_2$. The intensity of a partially coherent wave field can be written as a superposition of intensities of each coherent mode

$$I(\boldsymbol{r}) = J(\boldsymbol{r}) = \operatorname{diag}(J(\boldsymbol{r}_1, \boldsymbol{r}_2)) = \sum_n \lambda_n U_n^\dagger(\boldsymbol{r}) U_n(\boldsymbol{r}) = \sum_n I_n(\boldsymbol{r}), \tag{6.15}$$

where $I_n(\boldsymbol{r}) = \lambda_n U_n^\dagger(\boldsymbol{r}) U_n(\boldsymbol{r})$ is the intensity due to the n^{th} mode.

It can be shown that the *normalized mutual intensity* (also called *complex coherence factor*)

$$j(\boldsymbol{r}_1, \boldsymbol{r}_2) = \frac{J(\boldsymbol{r}_1, \boldsymbol{r}_2)}{\sqrt{I(\boldsymbol{r}_1) I(\boldsymbol{r}_2)}} \tag{6.16}$$

obeys the relation [5]

$$0 \leq |j(\boldsymbol{r}_1, \boldsymbol{r}_2)| \leq 1. \tag{6.17}$$

Two points in a monochromatic wave front for which $|j(\boldsymbol{r}_1, \boldsymbol{r}_2)| = 1$ are fully correlated or coherent. In contrast, two points for which $|j(\boldsymbol{r}_1, \boldsymbol{r}_2)| = 0$ are uncorrelated or incoherent. In particular, a point is coherent with itself, i.e. $|j(\boldsymbol{r}, \boldsymbol{r})| = 1$. For thermal emitters it is often appropriate to model the source as uncorrelated, i.e. [18]

$$J_{inc}(\boldsymbol{r}_1, \boldsymbol{r}_2) = \sqrt{I(\boldsymbol{r}_1) I(\boldsymbol{r}_2)} \delta(\boldsymbol{r}_2 - \boldsymbol{r}_1). \tag{6.18}$$

6.2 Evolution of partially coherent scalar waves

In the last section it was mentioned that the cross-spectral density and consequently the mutual intensity satisfy a set of coupled Helmholtz equations (Eqs. 6.5 and 6.6). These can be solved in a way analog to the solution of the Helmholtz equation for coherent waves. The angular spectrum solution for the propagation of mutual intensity is [39]

$$J(\boldsymbol{x}_1, \boldsymbol{x}_2, z) = \mathcal{F}^{-1}\left\{\mathcal{F}\left\{J(\boldsymbol{x}_1, \boldsymbol{x}_2, 0)\right\} H(\boldsymbol{f_1}, \boldsymbol{f_2}, z)\right\}, \tag{6.19}$$

where $\mathcal{F}$ denotes four-dimensional Fourier-transformation with respect to $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^2$ and

$$H(\boldsymbol{f_1}, \boldsymbol{f_2}, z) = \exp\left[ikz\left(\sqrt{1 - |\lambda \boldsymbol{f}_1|^2} - \sqrt{1 - |\lambda \boldsymbol{f}_2|^2}\right)\right] \tag{6.20}$$

is the transfer function of free space. The choice of opposite signs derives from the asymmetry in the definition of the mutual coherence function (Eq. 6.1). As is the case for coherent waves, the ASPW solution for the propagation of mutual coherence can be written in a form analog to the Rayleigh-Sommerfeld integral[3]

$$J(\boldsymbol{r}_1, \boldsymbol{r}_2, z) = \frac{z^2}{(i\lambda)^2} \int J(\boldsymbol{r}_1', \boldsymbol{r}_2', 0) \frac{\exp\left[i\boldsymbol{k}(\boldsymbol{r}_1 - \boldsymbol{r}_1') - i\boldsymbol{k}(\boldsymbol{r}_2 - \boldsymbol{r}_2')\right]}{|\boldsymbol{r}_1 - \boldsymbol{r}_1'|^2 |\boldsymbol{r}_2 - \boldsymbol{r}_2'|^2} d\boldsymbol{r}_1' d\boldsymbol{r}_2'. \tag{6.21}$$

[3] Eq. 6.21 is often called *generalized Van Zittert-Zernike theorem* [48].

Assuming an incoherent source with $J(\boldsymbol{r}'_1, \boldsymbol{r}'_2, 0) = I(\boldsymbol{r}'_1)\,\delta(\boldsymbol{r}'_2 - \boldsymbol{r}'_1)$ far away from the source, paraxial approximation of Eq. 6.21 leads to the *Van Zittert-Zernike theorem*,

$$J(\Delta x, \Delta y, z) \propto \int I(x', y', 0) \exp\left[i\frac{2\pi}{\lambda z}(x'\Delta x + y'\Delta y)\right] dx'dy', \tag{6.22}$$

where $\Delta x = x_2 - x_1$ and $\Delta y = y_2 - y_1$. The Van Zittert-Zernike theorem states that far away from an incoherent source the mutual intensity is proportional to the scaled Fourier transform of the source intensity, with independent variables as a function of separation of two points in the observation plane.

6.3 Transmission of partially coherent radiation through thin specimens

Given a light source of partial spatial coherence (*psc*), the far field diffraction intensity resulting from paraxially approximating the mutual intensity in Eq. 6.21 and setting $\boldsymbol{r} = \boldsymbol{r}_1 = \boldsymbol{r}_2$ leads to

$$I_{\mathrm{psc}}(\boldsymbol{r}) = \int J(\boldsymbol{r}'_1, \boldsymbol{r}'_2) \exp\left(-j2\pi\frac{(\boldsymbol{r}'_2 - \boldsymbol{r}'_1)\,\boldsymbol{r}}{\lambda z}\right) d\boldsymbol{r}'_1 d\boldsymbol{r}'_2, \tag{6.23}$$

where $J(\boldsymbol{r}'_1, \boldsymbol{r}'_2)$ is the mutual intensity in the source plane. The mutual intensity downstream a stationary sample with transmission function $T(\boldsymbol{r})$ is given by [18]

$$J(\boldsymbol{r}'_1, \boldsymbol{r}'_2) = \psi(\boldsymbol{r}'_1, \boldsymbol{t})\,\psi^*(\boldsymbol{r}'_2, \boldsymbol{t})\,j(\boldsymbol{r}'_1, \boldsymbol{r}'_2). \tag{6.24}$$

As a special case, it is assumed that the mutual intensity can be approximated by

$$J(\boldsymbol{r}'_1, \boldsymbol{r}'_2) = \psi(\boldsymbol{r}'_1, \boldsymbol{t})\,\psi^*(\boldsymbol{r}'_2, \boldsymbol{t})\,j(\boldsymbol{r}'_2 - \boldsymbol{r}'_1), \tag{6.25}$$

where j is the normalized mutual intensity. The relation $j(\boldsymbol{r}'_1, \boldsymbol{r}'_2) = j(\boldsymbol{r}'_1 - \boldsymbol{r}'_2)$ is the defining a *Schell model beam* [18]. It is noted that all fields satisfying the Van Zittert-Zernike theorem (Eq. 6.22) are by definition Schell model beams[4].

Changing to centered coordinates $\overline{\boldsymbol{r}} = (\boldsymbol{r}'_1 + \boldsymbol{r}'_2)/2$ and $\Delta\boldsymbol{r} = \boldsymbol{r}'_2 - \boldsymbol{r}'_1$, the partially coherent diffraction intensity in the observation plane is given by

$$I_{\mathrm{psc}}(\boldsymbol{r}) = \int \psi\left(\overline{\boldsymbol{r}} - \frac{\Delta\boldsymbol{r}}{2}\right)\psi^*\left(\overline{\boldsymbol{r}} + \frac{\Delta\boldsymbol{r}}{2}\right) j(\Delta\boldsymbol{r}) \exp\left(-j2\pi\frac{\Delta\boldsymbol{r}\boldsymbol{r}}{\lambda z}\right) d\overline{\boldsymbol{r}} d\Delta\boldsymbol{r}. \tag{6.26}$$

Using the autocorrelation and convolution theorems for Fourier transforms this is equivalent to

$$I_{\mathrm{psc}}(\boldsymbol{r}) = I^c(\boldsymbol{r}) \otimes \tilde{j}(\boldsymbol{r}), \tag{6.27}$$

where

$$I_{\mathrm{c}}(\boldsymbol{r}) = \int \psi\left(\overline{\boldsymbol{r}} - \frac{\Delta\boldsymbol{r}}{2}, \boldsymbol{t}\right)\psi^*\left(\overline{\boldsymbol{r}} + \frac{\Delta\boldsymbol{r}}{2}, \boldsymbol{t}\right) \exp\left(-j2\pi\frac{\Delta\boldsymbol{r}\boldsymbol{q}}{\lambda z}\right) d\overline{\boldsymbol{r}} d\Delta\boldsymbol{r} \tag{6.28}$$

[4] In particular, this approximation holds for most synchrotron beams since the exit slits are typically far away from the condenser or object plane in imaging experiments.

is the the coherent diffraction intensity and $\tilde{j}(\boldsymbol{r}) = \mathcal{F}_{\Delta\boldsymbol{r}\to\boldsymbol{r}} j(\Delta\boldsymbol{r})$. Equation 6.27 is referred to as *Schell's theorem* which states that the partially coherent diffraction intensity is equal to the coherent diffraction intensity convolved with the Fourier transform of the complex coherence factor [18].

6.4 Effective number of modes

Using the mutual intensity in numerical propagation problems is computationally challenging. While the digital representation of a two-dimensional coherent wave with N samples per dimension requires N^2 samples, a partially coherent wave with N samples per dimension described by means of a four-dimensional mutual intensity requires N^4 samples. However, the orthogonal mode decomposition (Eq. 6.7) can render partially coherent propagation problems tractable. Defining the *purity* [1]

$$\nu = \frac{\sqrt{\sum_l \lambda_l^2}}{\sum_l \lambda_l} \in [0, 1], \tag{6.29}$$

as a measure for coherence, the *effective number of modes* M_{eff} required to approximately represent a partially coherent wave field is on the order of [**?**]

$$M_{\text{eff}} = \frac{1}{\nu^2}. \tag{6.30}$$

6.5 Numerical aspects of partially coherent wave propagation

The purpose of this section is to illustrate numerical aspects of partially coherent wave propagation by example. The intuition gained from the discussion facilitates later analyses of partially coherent beams. Another objective is to show how diffraction data critically depends on the coherence state of the source deployed in an optical system [**?**]. It will be evident that accurate analysis of partially coherent lensless imaging setups, as discussed in later chapters, relies on a proper model of the coherence state of the underlying radiation source.

Simulated setup

The optical system depicted in Fig. 6.1 consists of an optical radiation source (condenser), an aperture, a diffuser and a detector. The condenser is assumed to be monochromatic ($\lambda = 625\,\text{nm}$) and spatially extended. Spatially separated source points are assumed to be mutually incoherent. Two condenser geometries are simulated below: a square with side length $60\,\mu\text{m}$ (cf. Fig. 6.2 a) and a ring condenser with $240\,\mu\text{m}$ diameter (cf. Fig. 6.3 a). The square condenser was selected as an example for a source with moderate degree of spatial coherence, whereas the ring condenser was selected as an example with low spatial coherence. An aperture at distance $z_1 = 10\,\text{cm}$ serves as a spatial coherence filter with a diameter of $1.5\,\mu\text{m}$. A phase-modulating diffuser is simulated closely behind the aperture ($z_2 = 1\,\text{mm}$) with a mean granularity of $67\,\mu\text{m}$, as measured by the full width

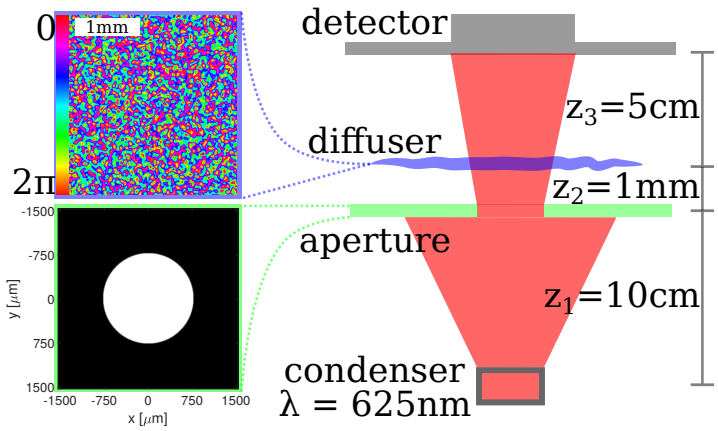


Figure 6.1: Simulated experimental setup. An incoherent source illuminates an aperture serving as a spatial filter. A phase-only diffuser behind the aperture modulates the partially coherent wave. Diffraction data is collected downstream the diffuser. Relevant physical units are as indicated.

at half maximum of its autocorrelation. The detector is placed at $z_3 = 5\,\text{cm}$ downstream the diffuser to record diffraction patterns from variable source geometries.

Square condenser

The simulation results for the square condenser are shown in Fig. 6.2. At a pixel size of $6\,\mu\text{m}$ the square source with side length $60\,\mu\text{m}$ in panel (a) consists of 100 source points. Each individual source point is propagated into the aperture plane using the impulse response of free space (Eq. 5.54). The resulting set of spherical waves is orthogonalized by means of a singular value decomposition (SVD).[5] The complex-valued orthogonalized mode structure in the aperture plane is shown in the hue-brightness plot in panel (b), where intensity is encoded as brightness and phase is encoded as hue. The corresponding *mode occupancy*

$$E_m = \frac{\lambda_m}{\sum_{\text{all } m} \lambda_m}, \tag{6.31}$$

which is a measure for the relative energy in each mode, is shown in panel (c). From the semi-logarithmic plot it is seen that the relative mode energy decreases exponentially. The purity (Eq. 6.29) for the square condenser beam is $\nu = 0.36$. As noted in the last section, the purity is useful to estimate the number of modes required to numerically represent a partially coherent beam to a good approximation. In this example $M_{\text{eff}} = 1/\nu^2 \approx 8$ and

[5] Other orthogonalization methods such as Gram-Schmidt or QR decomposition may alternatively be used [**?**].

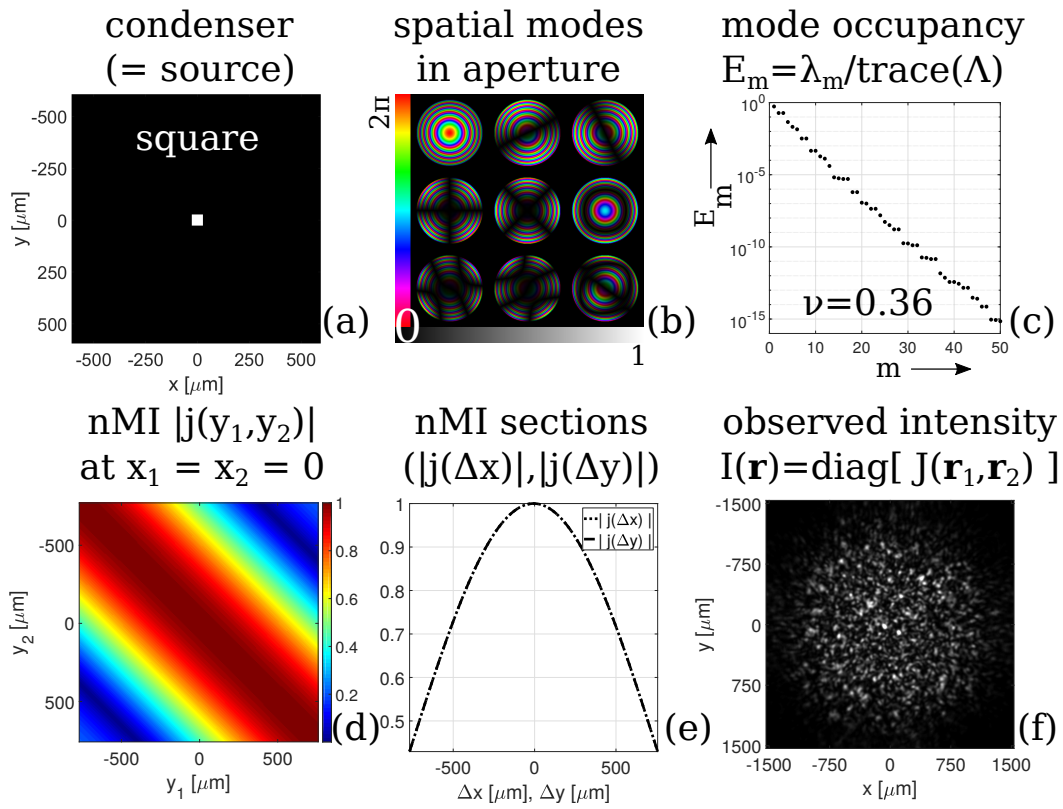


Figure 6.2: (a) square condenser; (b,c,d,e) show coherent modes, mode occupancy, nMI section (2D) and nMI section (1D), respectively in the aperture plane; (f) shows the diffraction pattern measured in the detector plane resulting from the square condenser.

the first 8 aperture modes contain

$$\sum_{m=1}^{8} E_m \approx 99.9\% \tag{6.32}$$

of the total energy contained in the partially coherent beam. Hence the truncated orthogonal mode decomposition is much more efficient than propagating individual source wavelets through the optical system. In particular, the orthogonal mode representation of the partially coherent beam requires only $8N^2$ samples, while the source wavelet required $100N^2$ samples in this example[6]. Panel (d) shows a two-dimensional section (y_1-y_2-plane) of the four-dimensional normalized mutual intensity (nMI, Eq. 6.16). A consequence of the Van Zittert-Zernike theorem (Eq. 6.22) is that a far-field nMI is only a function of separation ($\Delta r = r_1 - r_2$). Therefore the far-field partially coherent beam is a Schell-model beam and the nMI plotted in panel (d) takes the form of a Toeplitz matrix [38]. One-dimensional sections of the mutual intensity inside the aperture plane are shown in panel (e), where $|j(\Delta x)|$ and $|j(\Delta y)|$ are shown by dashed an dotted lines, which in this example fully overlap. For $|j(\Delta x)|$ both y_1 and y_2 are set to zero (and vice versa for $|j(\Delta y)|$). From the Van Zittert-Zernike theorem it is expected that the nMI takes on the form of a sinc-function, since the source intensity and far-field nMI are related by Fourier transformation (Eq. 6.22). Here only the portion of the sinc function is shown, that is spatially filtered by the 1.5 mm aperture. It is seen that the filtered beam exhibits correlation throughout the entire aperture. The smallest correlation is found at the extremal points of the aperture, where a double pinhole experiment (without diffuser) would yield a diffraction pattern I with a contrast $(\max(I) - \min(I)) / (\max(I) + \min(I)) \approx 0.5$. The diffraction pattern that results from the modulation of the phase-only diffuser as observed in the detector plane is shown in panel (f). The high contrast and fringe visibility is a consequence of the moderately high degree of spatial coherence on the beam.

Ring condenser

A similar simulation as above was carried out with the ring condenser shown in Fig. 6.3. All parameters including the diffuser shape were held constant except the source geometry. The 240 μm diameter ring (panel a) condenser has a thickness of 43 μm and consists of 848 source points. The spherical waves emanating from the source were orthogonalized in the aperture plane. The resulting mode structure is shown in panel (b). The beam produced by the ring condenser geometry has lower spatial coherence in the aperture plane as compared to the square condenser discussed above. This is both seen from the mode occupancy in panel (c), which is much more flat than in the previous example (Fig. 6.2 f), and from the low purity of $\nu = 0.167$. The resulting required number of modes is on the order of $M_{\text{eff}} = 1/\nu^2 \approx 36$. The relative energy contained in the first 36 modes is

$$\sum_{m=1}^{36} E_m \approx 94.9\%. \tag{6.33}$$

[6] In practical problems of large data volumes the marginal loss of numerical precision is often a good trade-off against the order of magnitude in computational memory and overhead gained in using the truncated orthogonal mode representation.

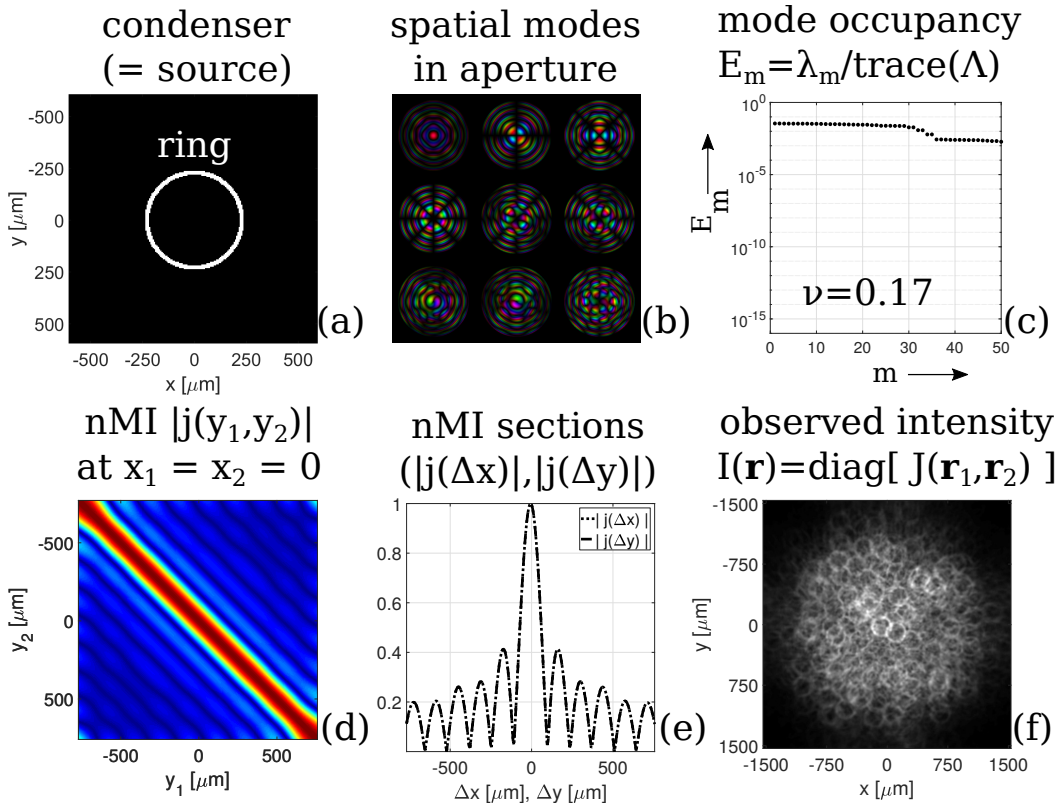


Figure 6.3: (a) Ring condenser; (b,c,d,e) show coherent modes, mode occupancy, nMI section (2D) and nMI section (1D), respectively in the aperture plane; (f) shows the diffraction pattern measured in the detector plane resulting from the square condenser.

As in the previous example, the orthogonal mode representation dramatically reduces the computational complexity of the partially coherent wave propagation simulation [7]. Another indication of the smaller spatial coherence length is given by the normalized mutual intensity shown in panels (d) and (e), which which exhibits a smaller FWHM than in the previous example. Finally, the diffraction pattern looks different than in the previous example. For the ring condenser it is apparent that the diffraction pattern is the convolution of both the diffraction pattern of the diffuser and the source . Mathematically, this is a consequence of Schell's theorem (Eq. 6.27). [8]

Summary

For a low coherence source the normalized mutual intensity $j(x_1, x_2)$ is diagonal, or full rank [38]. For a coherent source the rows and columns of $j(x_1, x_2)$ are linearly dependent, i.e. low rank or sparse. In between the extreme cases of incoherence (full rank) and coherence (sparsity), efficient numerical representations for partially coherent wave fields can be found by means of truncated singular value decompositions, where the effective rank is proportional to the reciprocal square of the purity of the wave field.

It was seen that the appearance of diffraction patterns in lensless imaging setups is affected by the source geometry in the case of partially coherent fields. From the example, it is evident that lensless partially coherent systems are sensitive to the source geometry at hand. In particular, if the source geometry is not exactly known, the coherence state of the system has to be recovered in order to allow for lensless imaging.

The simulations in this section further illustrated that the correlation in partially coherent wave fields increases upon propagation. While the source emitters where completely incoherent, the waves in the observation plane exhibited coherence in both examples. This is a consequence of the fact that not only optical wave fields but also their correlations obey wave equations (Eqs. 6.2 and 6.3). The standard deviations of the independent variables in the Van Zittert-Zernike theorem (cf. Eq. 6.22) must therefore satisfy the uncertainty relations [9]

$$\sigma_{\Delta x}\sigma_{x'/\lambda z} \geq \frac{1}{4\pi}. \tag{6.34}$$

Assuming that the standard deviation of the source is on the order of its size s, the standard deviation for the correlation distance in the observation plane can be estimated to have a lower bound given by

$$\sigma_{\Delta x} \geq \frac{1}{4\pi}\frac{\lambda z}{s}. \tag{6.35}$$

[7] 848 spherical waves each of which contain N^2 pixels are approximated by 36 modes each of which have N^2 pixels.

[8] Intuitively, this effect can be understood in analogy to a pinhole camera. In this case, the source radiation is once Fourier transformed into the aperture plane upon propagation. Then through spatial filtering at the aperture and propagation to the observation plane a second Fourier transform occurs which displays an inverted source. This effect is for example used in *coded aperture imaging*, with the difference that a non-redundant aperture array is used, for which the observation data can be deconvolved more easily than in the present example [8].

[9] Any Fourier transform pair $G(f) = \mathcal{F}g(x)$ satisfies the uncertainty relation $\sigma_x\sigma_f \geqslant 1/4\pi$ [7]. Here $\sigma_x^2 = \int x^2 |g(x)|^2 dx$ and $\sigma_f^2 = \int f^2 |g(f)|^2 dx$ are the variances of the independent variables in real and Fourier space, respectively. It was further assumed for simplicity that $g(x)$ and $G(f)$ are normalized and have zero mean. Further details can be found in [48].

It is common to define the *transverse coherence length*

$$l_{\mathrm{t,max}} = \frac{\lambda z}{s}, \tag{6.36}$$

as an estimate on the order of magnitude of the length scales over which an incoherent source exhibits some degree of correlation. For the square condenser problem Eq. 6.36 estimates a coherence length of $l_{t,max} = 1040\,\mu$m. It is noted that at this distance and larger, the source exhibits almost vanishing correlation. Therefore, throughout this thesis Eq. 6.36 is interpreted as the *maximum* distance over which an incoherent source exhibits correlation in the far field. A more conservative estimate of the transverse coherence length is [3]

$$l_{\mathrm{t,min}} = \frac{1}{2\pi}\frac{\lambda z}{s}. \tag{6.37}$$

The latter expression for the square simulation in this section (cf. fig. 6.2) yields $|j\,(l_{\mathrm{t,min}} = 166\,\mu\mathrm{m})| = 0.96$. Therefore Eq. 6.37 is interpreted as the *minimum* distance over which the source is sufficiently correlated.

Chapter 7

Scattering

7.1 Foldy-Lax scattering theory

In section 5.1 we have encountered the inhomogeneous Helmholtz equation

$$\nabla^2 u(\boldsymbol{r}) + k^2 u(\boldsymbol{r}) = -V(\boldsymbol{r})\, u(\boldsymbol{r})$$

where $k = 2\pi/\lambda$ and $V(\boldsymbol{r}) = k^2 [n(\boldsymbol{r}) - 1]$ is the *scattering potential*. In this chapter we discuss how to solve for the scattered field $U(\boldsymbol{r})$ in various exact and approximate ways. We start from discrete point scatter distributions and generalize the results to continuous scattering distributions.

Suppose we want to calculate the scalar field distribution scattered from a distribution of N isolated spheres in space. In general, the total field amplitude U can be written as the sum of the incident field u^{in} and the scattered field u^s

$$u(\boldsymbol{r}) = u^{in}(\boldsymbol{r}) + u^s(\boldsymbol{r}). \tag{7.1}$$

We assume the scattered field is a solution to the Helmholtz equation,

$$\nabla^2 u + k^2 u = 0. \tag{7.2}$$

As such, the scattered field due to the jth sphere is given by

$$u_j^s(\boldsymbol{r} - \boldsymbol{r}_j) = V(\boldsymbol{r}_j)\, u(\boldsymbol{r}_j)\, G(\boldsymbol{r} - \boldsymbol{r}_j), \tag{7.3}$$

where for three-dimensional problems

$$G(\boldsymbol{r} - \boldsymbol{r}_j) = \begin{cases} \frac{\exp(ik|\boldsymbol{r}-\boldsymbol{r}_j|)}{ik|\boldsymbol{r}-\boldsymbol{r}_j|} & , \boldsymbol{r} \neq \boldsymbol{r}_j \\ 0 & , \boldsymbol{r} = \boldsymbol{r}_j \end{cases} \tag{7.4}$$

and for two-dimensional problems

$$G(\boldsymbol{r} - \boldsymbol{r}_j) = \begin{cases} i\pi H_0^1(k|\boldsymbol{r} - \boldsymbol{r}_j|) & , \boldsymbol{r} \neq \boldsymbol{r}_j \\ 0 & , \boldsymbol{r} = \boldsymbol{r}_j \end{cases}. \tag{7.5}$$

In the latter equation, H_0^1 is a zeroth-order Hankel function of the first kind[1]. The different form of the Green's function in two- and three-dimensional problems is a result of the different formulations of the Helmholtz equation in cylindrical and spherical coordinates. Thus

$$u(\boldsymbol{r}) = u^{in}(\boldsymbol{r}) + \sum_{j=1}^{N} V(\boldsymbol{r}_j)\, G(\boldsymbol{r} - \boldsymbol{r}_j)\, u(\boldsymbol{r}_j), \tag{7.6}$$

which is the so-called *Lippmann-Schwinger equation* . Sampling this function at discrete points in space $\boldsymbol{r} = \boldsymbol{r}_i$ and defining $\boldsymbol{M}_{i,j} = V(\boldsymbol{r}_j)\, G(\boldsymbol{r}_i - \boldsymbol{r}_j)$, we get

$$u_i = u_i^{in} + \sum_{j=1}^{N} \boldsymbol{M}_{i,j} u_j. \tag{7.7}$$

This may be rewritten in matrix form

$$\boldsymbol{u} = \boldsymbol{u}^{in} + \boldsymbol{M}\boldsymbol{u}. \tag{7.8}$$

We can solve this for the total field by first collecting the $\boldsymbol{u}$ terms on the left

$$(\boldsymbol{I} - \boldsymbol{M})\, \boldsymbol{u} = \boldsymbol{u}^{in} \tag{7.9}$$

and then calculating the inverse of the matrix $\boldsymbol{I} - \boldsymbol{M}$,

$$\boldsymbol{u} = (\boldsymbol{I} - \boldsymbol{M})^{-1}\, \boldsymbol{u}^{in}. \tag{7.10}$$

The latter is the *Foldy-Lax solution* to the multiple scattering problem. The computation of the inverse $(\boldsymbol{I} - \boldsymbol{M})^{-1}$ entails a computational cost of $\mathcal{O}\left(N^3\right)$.

7.1.1 First-order Born approximation

An approximate solution may be obtained by substituting $\boldsymbol{u} = \boldsymbol{u}^{in}$ on the right hand side of Eq. 7.8,

$$\boldsymbol{u}^{(1)} \approx \boldsymbol{u}^{in} + \boldsymbol{M}\boldsymbol{u}^{in}, \tag{7.11}$$

which is referred to as the *first-order Born approximation*. The superscript (1) expresses that a single scattering event took place. Below we will discuss a higher-order series expansion, where multiple scattering events can take place. The first-order Born approximation omits matrix inversion and requires only a simple matrix-vector multiplication at the cost $\mathcal{O}\left(N^2\right)$.

Example

The following Matlab code compares the solutions predicted by the Foldy-Lax theory to the first-order Born approximation. A point source originating from outside of the numerical array illuminates a distribution of point scatterers (propagation direction of incoming beam $\boldsymbol{u}^{in}$ from top to bottom, see left panel in Fig. 7.1). In Fig. 7.1 it is seen that the Born approximation (right) overestimates the amount of light in the forward

[1] https://en.wikipedia.org/wiki/Bessel_function#Hankel_functions:_H(1)%CE%B1,_H(2)%CE%B1

scattering direction as compared to the Foldy-Lax model. We encourage the reader to decrease the value of α. In this case, the prediction by Foldy-Lax and Born become increasingly similar.

```
N = 2^6; dx = 100e-9; L = N * dx;
x = (-N/2:N/2-1)*dx; z = x;
[X,Z] = meshgrid(x);

% determine wavevector
k = 2*pi/633e-9;

% source coordinate
xs = 0;zs = -1.25*L;

% spherical wave
R = sqrt((X-xs).^2 + (Z-zs).^2);
Uin = exp(1i*k*R)./(1i*k*R);

%% determine random point scatter distribution

rng(0,'twister')
idx = 1:10:N^2;
numScatterers = length(idx);
idx = min(idx+randi([1 4],size(idx)),N^2);
aleph = 5e-2;
pointScatterDist = zeros(N,N);
pointScatterDist(idx) = aleph;
pointScatterDist = pointScatterDist .* (X.^2+Z.^2 < (L/4)^2);
idx = find(pointScatterDist(:)>0);

%% compute scattering matrix
G = mean(aleph)*exp(1i * k * R) ./ (1i * k * R + eps);
% method 1: Foldy Lax
M = zeros(N^2,N^2);
for loop = 1:length(idx)
    R = sqrt((X-X(idx(loop))).^2 + (Z-Z(idx(loop))).^2);
    % 2D Green's function: Hankel
    G = aleph * 1i*pi*besselh(0,k * R);
    G(idx(loop)) = 0;
    M(:,idx(loop)) = G(:);
end
Ufoldy = reshape( sparse(eye(N^2) - M) \ Uin(:), [N,N]);

% method 2: 1st Born approximation
Uborn = Uin(:) + M*Uin(:);
Uborn = reshape( Uborn, [N,N]);

```

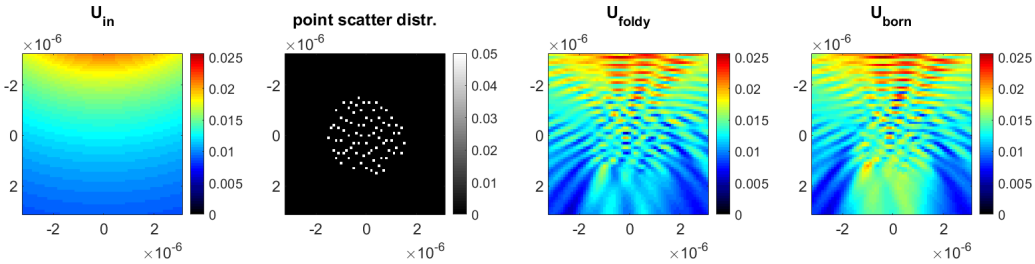


Figure 7.1: From left to right we show incoming wave, distribution of scattering spheres, the Foldy-Lax solution, and the first-order Born approximation ($\alpha = 5\%$).

```
maxAbs = max(max(abs(Ufoldy(:)),abs(Uborn(:))));

%% show results
% modify colormap
cmap = jet(50);
c = linspace(0,1,10);
cmap = [c' * cmap(1,:);cmap];

figure(3)
subplot(1,4,1)
imagesc(x,x,abs(Uin),[0 maxAbs]), axis image, colormap(cmap)
title({'U_{in}',''}), colorbar

subplot(1,4,2)
imagesc(x,x,pointScatterDist), axis image, colormap(gray)
h = gca; title({'point scatter distr.',''}), colorbar

subplot(1,4,3)
imagesc(x,x,abs(Ufoldy),[0 maxAbs]), axis image, colormap(cmap)
title({'U_{foldy}',''}), colorbar

subplot(1,4,4)
imagesc(x,x,abs(Uborn),[0 maxAbs]), axis image, colormap(cmap)
title({'U_{born}',''}), colorbar

colormap(h, gray)
```

Listing 7.1: foldyLax.m (Matlab)

Exercise: Translate foldyLax.m into Python or Julia. Exercise

---

Exercise: Create an animation (gif/mp4/etc...) that produces the plot as shown in Fig. 7.1 while the origin of the point source is laterally shifted (to the left). For each shifted point source, save the last row of the solution matrix (bottom row in Fig. 7.1). Exercise

Denote each row with $\boldsymbol{s}_j$, where $j = 0$ is the unshifted point source, $j = 1$ is the point source shifted one pixel to the left, and so forth. Compute and plot absolute value of the complex inner product between the first ($j = 0$) and each other row ($j \geq 0$). You will observe that the curve decreases with increasing shift. Over which distance will the curve attain half its maximum value? What you are looking at is the characteristic distance over which the sample exhibits a so-called *translational memory effect* [36].

7.1.2 Higher-order Born series

In the previous example we have seen that the (computationally cheap) first order Born approximation can approximate the (computationally expensive but more precise) Foldy-Lax theory. We may attempt to obtain a higher-order Born approximation by continuing the same strategy as before. Namely, we can insert the first-order Born approximation into Eq. 7.8 to obtain the second-order Born approximation,

$$\boldsymbol{u}^{(0)} = \boldsymbol{u}^{in}, \tag{7.12}$$

$$\boldsymbol{u}^{(1)} = \boldsymbol{u}^{(0)} + \boldsymbol{M}\boldsymbol{u}^{(0)}, \tag{7.13}$$

$$\begin{aligned} \boldsymbol{u}^{(2)} &= \boldsymbol{u}^{(0)} + \boldsymbol{M}\boldsymbol{u}^{(1)}, && (7.14)\\ &= \boldsymbol{u}^{(0)} + \boldsymbol{M}\left[\boldsymbol{u}^{(0)} + \boldsymbol{M}\boldsymbol{u}^{(0)}\right] && (7.15)\\ &= \boldsymbol{u}^{(0)} + \boldsymbol{M}\boldsymbol{u}^{(0)} + \boldsymbol{M}^2\boldsymbol{u}^{(0)} && (7.16) \end{aligned}$$

and so forth,

$$\begin{aligned} \boldsymbol{u}^{(n)} &= \boldsymbol{u}^{(0)} + \boldsymbol{M}\boldsymbol{u}^{(n-1)}. && (7.17)\\ &= \left(\sum_{k=0}^{n} \boldsymbol{M}^k\right)\boldsymbol{u}^{(0)} && (7.18) \end{aligned}$$

We note that the series of matrix powers in parenthesis is called *Neumann series*, which is a Taylor expansion[2] of the inverse we are trying to approximate

$$(\boldsymbol{I} - \boldsymbol{M})^{-1} = \sum_{k=0}^{\infty} \boldsymbol{M}^k.$$

Unfortunately this series diverges when the largest eigenvalue of $\boldsymbol{M}$ is greater than 1 in magnitude. Osnabrugge et al. [37] proposed a simple trick[3] to avoid this issue. First multiply Eq. 7.8 by a constant scalar $0 < \gamma < 1$,

$$\gamma\boldsymbol{u} = \gamma\boldsymbol{u}^{in} + \gamma\boldsymbol{M}\boldsymbol{u}. \tag{7.19}$$

[2]Notice the similarity of the Neumann series for matrices with the geometric series for univariate variables $1/(1-x) = \sum_{k=1}^{\infty} x^k$, which converges in the interval $-1 < x < 1$.

[3]We will discuss the idea behind this trick a bit deeper in subsection 7.1.3.

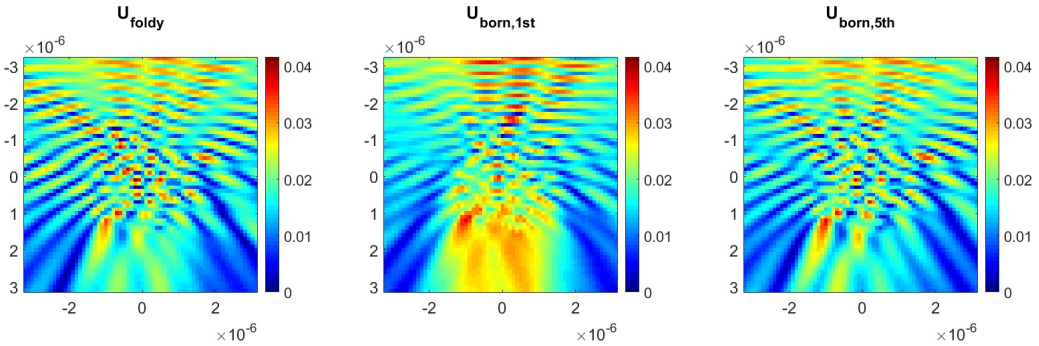


Figure 7.2: Comparison of Foldy-Lax scattering (left) with 1st ($\gamma = 1$, middle) and 5th ($\gamma = 0.75$, right) order Born approximation ($\alpha = 10\%$).

Next, add $\boldsymbol{u}$ to each side of the equation and rearrange to get

$$\boldsymbol{u} = \gamma \boldsymbol{u}^{in} + [\gamma \boldsymbol{M} + (1-\gamma)\,\boldsymbol{I}]\,\boldsymbol{u}. \tag{7.20}$$

Now we pretend we know the $\boldsymbol{u}$-term on the right-hand side by replacing it with an iterative estimate $\boldsymbol{u}^{(n-1)}$. This is known as a *fixed-point iteration* [4]. With this we get,

$$\boldsymbol{u}^{(n)} = \gamma \boldsymbol{u}^{in} + [\gamma \boldsymbol{M} + (1-\gamma)\,\boldsymbol{I}]\,\boldsymbol{u}^{(n-1)}. \tag{7.21}$$

The code `foldyLax_convergentBorn.m`[5] illustrates the idea. Figure 7.2 shows the Foldy-Lax solution (left) as compared to the 1st-order Born approximation ($\gamma = 1$, middle) and the 5th-order modified Born approximation ($\gamma = 0.75$, right). In contrast to the previous section where we used $\alpha = 5\%$, here we increased the scattering strength to $\alpha = 10\%$. At the end of the next section, we will return to the modified Born series in Eq. 7.21 and suitable choices of γ to achieve convergence. We have reduced the computational cost in solving the Lippmann-Schwinger equation from $\mathcal{O}\left(N^3\right)$ as required by the inverse of the Foldy-Lax theory to $\mathcal{O}\left(N^2\right)$ as required by the higher-order Born series.

7.1.3 Convergence of Born series

In the previous section we derived an iterative form of a higher-order Born series. So far we did not discuss why the modified Born series in Eq. 7.21 should have any advantage over the standard Born series in Eq. 7.17. Suppose $\boldsymbol{x}$ is an eigenvector of the scattering matrix $\boldsymbol{M}$. Then we have

$$\boldsymbol{M}\boldsymbol{x} = \lambda \boldsymbol{x}. \tag{7.22}$$

Insert the same eigenvector $\boldsymbol{x}$ into the right term of the convergent Born series in Eq. 7.21, then

$$[\gamma \boldsymbol{M} + (1-\gamma)\,\boldsymbol{I}]\,\boldsymbol{x} = \left[\gamma\lambda + \underbrace{(1-\gamma)}_{<1}\right]\boldsymbol{x}. \tag{7.23}$$

[4] https://en.wikipedia.org/wiki/Fixed-point_iteration

[5] This code is provided online in the ComputationalImaging.m Github repository.

we see that the eigenvalues of the modified Born series scattering matrix $\gamma \boldsymbol{M} + (1-\gamma)\,\boldsymbol{I}$ are $\gamma\lambda + (1-\gamma)$. In situations where the largest eigenvalue of $\boldsymbol{M}$ is larger than 1, the standard higher-order Born series diverges. In such situations, the choice $\gamma < 1/\lambda_{\max}$ guarantees that the modified term $\gamma \boldsymbol{M} + (1-\gamma)\,\boldsymbol{I}$ is an eigenvalue smaller than 1, so that matrix powers of it converge.

7.2 Fourier diffraction theorem

In the previous section we have discussed the Born series as an approximate solution to the Lippmann-Schwinger equation (Eqn. 7.8). There are intuitive real and reciprocal space interpretations of the first order Born approximation, which we discuss in this section.

In section 5.3 we have discussed the angular spectrum propagator. There we used a factorization Ansatz to distinguish between the exit wave, object wave, and the plane wave illumination (Eqn. 5.37)

$$\underbrace{U(x,y,0)}_{\text{exit wave}} = \underbrace{\psi(x,y,0)}_{\text{object wave}} \cdot \underbrace{\exp\left(ik\left[s_x x + s_y y\right]\right)}_{\text{plane wave illumination}}. \tag{7.24}$$

We used the factorization Ansatz to directly solve for a propagator of the object wave. Explicitly modeling the illumination lead to a modified ordinary differential equation that followed from the Helmholtz equation, namely

$$\frac{d^2\widetilde{\psi}}{dz^2} = -k^2\left[1 - (s_x - \lambda f_x)^2 - (s_y - \lambda f_y)^2\right]\widetilde{\psi}. \tag{7.25}$$

If we perform an analogous derivation directly on the exit wave, not using the factorization Ansatz, we find

$$\frac{d^2\widetilde{U}}{dz^2} = -k^2\left[1 - (\lambda f_x)^2 - (\lambda f_y)^2\right]\widetilde{U}. \tag{7.26}$$

These ODEs lead to different transfer functions for the propagation of the object and exit waves

$$H_\psi(f_x, f_y, z) = \exp\left[ikz\sqrt{1 - (s_x - \lambda f_x)^2 - (s_y - \lambda f_y)^2}\right] \tag{7.27}$$

and

$$H_U(f_x, f_y, z) = \exp\left[ikz\sqrt{1 - (\lambda f_x)^2 - (\lambda f_y)^2}\right]. \tag{7.28}$$

We note that both of these transfer functions are *mixed representations*. By Fourier transforming along the z direction, we find a pure Fourier representation of the transfer functions, where the independent variables are all spatial frequencies,

$$\mathcal{H}_\psi(f_x, f_y, f_z) = \mathcal{F}_{z\to f_z} H_\psi(f_x, f_y, z) \tag{7.29}$$

$$= \delta\left(f_z - \sqrt{\frac{1}{\lambda^2} - \left(\frac{s_x}{\lambda} - f_x\right)^2 - \left(\frac{s_y}{\lambda} - f_y\right)^2}\right) \tag{7.30}$$

and

$$\mathcal{H}_U\left(f_x, f_y, f_z\right) = \mathcal{F}_{z\to f_z} H_U\left(f_x, f_y, z\right) = \delta\left(f_z - \sqrt{\frac{1}{\lambda^2} - f_x^2 - f_y^2}\right). \tag{7.31}$$

The latter result is oftentimes referred to as the *Ewald sphere*. If we measure a coherent exit wave, for example interferometrically or by means of phase retrieval techniques, information contained in such a coherent wave field measurement is located on a spherical shell in three-dimensional Fourier space .

Here we distinguish the object and exit wave Ewald spheres which are related via the *Fourier diffraction theorem (FDT)*

$$\mathcal{H}_\psi\left(\boldsymbol{f}\right) = \mathcal{H}_U\left(\boldsymbol{f} + \frac{\boldsymbol{s}}{\lambda}\right). \tag{7.32}$$

In words, the Ewald sphere of an exit wave is related to the Ewald sphere of an object by a shift. The physical significance of the FDT is that three-dimensional information contained in a measurement of an exit wave is related to the three-dimensional information about an object wave via a shift of the corresponding Ewald spheres. If we extend the measurement to exit waves from a sequence of directions $\boldsymbol{s}_k$, we can coherently add the partial information from each observation to get an estimate of the three-dimensional Fourier transform of an object of interest

$$\tilde{\psi}\left(f_x, f_y, f_z\right) \approx \sum_k \tilde{\boldsymbol{U}}_k\left(f_x, f_y\right) \delta\left(f_z - \sqrt{\frac{1}{\lambda^2} - \left(\frac{s_{x,k}}{\lambda} - f_x\right)^2 - \left(\frac{s_{y,k}}{\lambda} - f_y\right)^2}\right). \tag{7.33}$$

7.3 Beam propagation method (BPM)

7.4 Rytov approximation

7.5 Wave propagation method (WPM)

Chapter 8

Wide field microscopy

8.1 Transmission cross coefficient (TCC)

We are considering partially coherent image formation in a wide field microscope in Köhler configuration 8.1. The outgoing mutual intensity downstream of a thin specimen is

$$J_{\text{out}}\left(\boldsymbol{r}_1, \boldsymbol{r}_2\right) = J_{\text{in}}\left(\boldsymbol{r}_1, \boldsymbol{r}_2\right) T\left(\boldsymbol{r}_1\right) T^*\left(\boldsymbol{r}_2\right). \tag{8.1}$$

J_{in} is the incoming mutual intensity resulting from the source which is "*Köhlered*", i.e. the source is located at the focal plane of the condenser lens. T is the specimen transmission function. In Fig. 8.1 we are separately showing the illumination path (red) and the specimen imaging optical path (dashed black). This configuration is named after August Köhler who invented it to minimize illumination inhomogeneity in the specimen plane [**?**].

Because we are dealing with partially coherent light, the mathematical description to follow involves bilinear expressions (revisit chapter **??** on coherence if needed). Transforming into the camera plane via an imaging system we have a bilinear convolution of

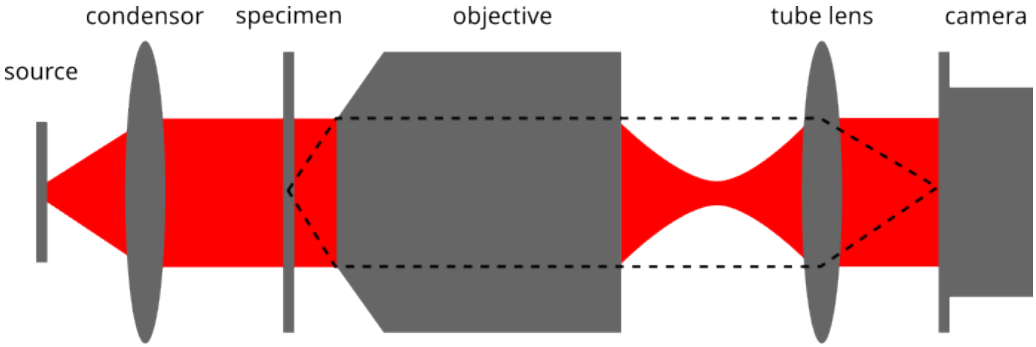


Figure 8.1: Wide field optical system in Köhler configuration.

the outgoing mutual intensity

$$\begin{aligned} I_{\text{cam}}(\boldsymbol{r}) =& J_{\text{cam}}(\boldsymbol{r}_1 = \boldsymbol{r}, \boldsymbol{r}_2 = \boldsymbol{r}) && (8.2)\\ =& \iint J_{\text{out}}(\boldsymbol{r}_1', \boldsymbol{r}_2')\, h\left(\boldsymbol{r} - \boldsymbol{r}_1'\right) h^*\left(\boldsymbol{r} - \boldsymbol{r}_2'\right) d\boldsymbol{r}_1'\, d\boldsymbol{r}_2' && (8.3)\\ =& \iint J_{\text{in}}(\boldsymbol{r}_1', \boldsymbol{r}_2')\, T(\boldsymbol{r}_1')\, T^*(\boldsymbol{r}_2')\, h\left(\boldsymbol{r} - \boldsymbol{r}_1'\right) h^*\left(\boldsymbol{r} - \boldsymbol{r}_2'\right) d\boldsymbol{r}_1' d\boldsymbol{r}_2' && (8.4)\\ =& \iiint \mathcal{S}(\boldsymbol{f})\, \mathcal{T}(\boldsymbol{f}_1)\, \mathcal{T}^*(\boldsymbol{f}_2) \times \ldots \\ & \mathcal{H}(\boldsymbol{f} + \boldsymbol{f}_1)\, \mathcal{H}^*(\boldsymbol{f} + \boldsymbol{f}_2) \exp\left(j2\pi\left(\boldsymbol{f}_1 - \boldsymbol{f}_2\right)\boldsymbol{r}\right) d\boldsymbol{f}\, d\boldsymbol{f}_1\, d\boldsymbol{f}_2 && (8.5)\\ =& \iint \mathcal{T}(\boldsymbol{f}_1)\, \mathcal{T}^*(\boldsymbol{f}_2)\, C(\boldsymbol{f}_1, \boldsymbol{f}_2) \exp\left(j2\pi\left(\boldsymbol{f}_1 - \boldsymbol{f}_2\right)\boldsymbol{r}\right) d\boldsymbol{f}_1\, d\boldsymbol{f}_2 && (8.6) \end{aligned}$$

where we used the Van-Cittert Zernike theorem stating that the mutual intensity incident on the specimen J_{in} is the Fourier transform of the source $\mathcal{S}$ (Eq. 6.22)

$$J_{\text{in}}(\boldsymbol{r}_1, \boldsymbol{r}_2) \propto \int S(\boldsymbol{f}) \exp\left[i2\pi \boldsymbol{f}\left(\boldsymbol{r}_1 - \boldsymbol{r}_2\right)\right] d\boldsymbol{f},$$

and

$$C(\boldsymbol{f}_1, \boldsymbol{f}_2) = \int \mathcal{S}(\boldsymbol{f})\, \mathcal{H}(\boldsymbol{f} + \boldsymbol{f}_1)\, \mathcal{H}^*(\boldsymbol{f} + \boldsymbol{f}_2)\, d\boldsymbol{f} \tag{8.7}$$

is the *transmission cross coefficient* (TCC) first derived by Hopkins [**?**]. The calligraphic symbols in Eqn. 8.6 are each Fourier transforms of the non-calligraphic symbols, to wit $\mathcal{T} = \mathcal{F}T$ and $\mathcal{H} = \mathcal{F}h$. Before moving further in the theoretic development, we note an important symmetry property, namely

$$\begin{aligned} C(\boldsymbol{f}_2, \boldsymbol{f}_1) &= \int \mathcal{S}(\boldsymbol{f})\, \mathcal{H}(\boldsymbol{f} + \boldsymbol{f}_2)\, \mathcal{H}^*(\boldsymbol{f} + \boldsymbol{f}_1)\, d\boldsymbol{f} \\ &= \left(\int \mathcal{S}(\boldsymbol{f})\, \mathcal{H}^*(\boldsymbol{f} + \boldsymbol{f}_2)\, \mathcal{H}(\boldsymbol{f} + \boldsymbol{f}_1)\, d\boldsymbol{f}\right)^* \\ &= C^*(\boldsymbol{f}_1, \boldsymbol{f}_2), && (8.8) \end{aligned}$$

i.e. the TCC is Hermitian.

The significance of the TCC is that it describes partially coherent imaging in terms of the optical system parameters, namely the source $\mathcal{S}$ and the pupil $\mathcal{H}$. The pupil can contain aberrations such as defocus, which allows to describe three-dimensional imaging systems [**?**]. Moreover, the source can attain any shape, for example on-axis coherent illumination, half-pupil oblique illumination, and dark field configurations. Below we will discuss several special cases that are of interest.

8.1.1 Coherent imaging

Suppose we have a small light source described by an on-axis delta impulse $\mathcal{S}(\boldsymbol{f}) = \delta(\boldsymbol{f})$, then the TCC takes on the form

$$\begin{aligned} C(\boldsymbol{f}_1, \boldsymbol{f}_2) &= \int \delta(\boldsymbol{f})\, \mathcal{H}(\boldsymbol{f} + \boldsymbol{f}_1)\, \mathcal{H}^*(\boldsymbol{f} + \boldsymbol{f}_2)\, d\boldsymbol{f} && (8.9)\\ &= \mathcal{H}(\boldsymbol{f}_1)\, \mathcal{H}^*(\boldsymbol{f}_2). && (8.10) \end{aligned}$$

With this the image intensity may be written in a separable form

$$I_{\text{cam}}(\boldsymbol{r}) = \iint \mathcal{T}(\boldsymbol{f}_1)\,\mathcal{T}^*(\boldsymbol{f}_2)\,\mathcal{H}(\boldsymbol{f}_1)\,\mathcal{H}^*(\boldsymbol{f}_2)\exp\left(j2\pi\left(\boldsymbol{f}_1-\boldsymbol{f}_2\right)\boldsymbol{r}\right)d\boldsymbol{f}_1\,d\boldsymbol{f}_2 \tag{8.11}$$

$$= \int \mathcal{T}(\boldsymbol{f}_1)\,\mathcal{H}(\boldsymbol{f}_1)\exp\left(j2\pi\boldsymbol{f}_1\boldsymbol{r}\right)d\boldsymbol{f}_1 \times \ldots$$

$$\left[\int \mathcal{T}(\boldsymbol{f}_2)\,\mathcal{H}(\boldsymbol{f}_2)\exp\left(j2\pi\boldsymbol{f}_2\boldsymbol{r}\right)d\boldsymbol{f}_2\right]^* \tag{8.12}$$

$$= \left|\int \mathcal{T}(\boldsymbol{f})\,\mathcal{H}(\boldsymbol{f})\exp\left(j2\pi\boldsymbol{f}\boldsymbol{r}\right)d\boldsymbol{f}\right|^2 \tag{8.13}$$

$$= \left|\int T(\boldsymbol{r}')\,h(\boldsymbol{r}-\boldsymbol{r}')\,d\boldsymbol{r}'\right|^2 = \left|T(\boldsymbol{r})\otimes h(\boldsymbol{r})\right|^2. \tag{8.14}$$

In words, for a coherent point source on-axis illumination, the recorded image intensity is proportional[1] to the absolute value squared of the specimen transmission function T with the coherent spread function (CSF) h.

8.1.2 Incoherent imaging

Next we consider an incoherent homogeneous source which we assume to have a constant intensity $\mathcal{S}(\boldsymbol{f}) = 1$. By the Van-Cittert Zernike theorem, for an incoherent source, the incoming mutual intensity is a delta function

$$J_{\text{in}}(\boldsymbol{r}_1,\boldsymbol{r}_2) = \delta(\boldsymbol{r}_1-\boldsymbol{r}_2). \tag{8.15}$$

But then

$$I_{\text{cam}}(\boldsymbol{r}) = \iint \delta(\boldsymbol{r}_1'-\boldsymbol{r}_2')\,T(\boldsymbol{r}_1')\,T^*(\boldsymbol{r}_2')\,h\left(\boldsymbol{r}-\boldsymbol{r}_1'\right)h^*\left(\boldsymbol{r}-\boldsymbol{r}_2'\right)d\boldsymbol{r}_1 d\boldsymbol{r}_2 \tag{8.16}$$

$$= \int \left|T(\boldsymbol{r}')\right|^2\left|h(\boldsymbol{r}-\boldsymbol{r}')\right|^2 d\boldsymbol{r}' \tag{8.17}$$

$$= \left|T(\boldsymbol{r})\right|^2 \otimes \left|h(\boldsymbol{r})\right|^2. \tag{8.18}$$

We find that for fully incoherent imaging the camera intensity is the absolute value squared of the specimen transmission function convolved with the incoherent point spread function (PSF) $\left|h(\boldsymbol{r})\right|^2$, the latter being the absolute value squared of the CSF h. Notice that the absolute value square of the specimen transmission function makes it impossible for fully incoherent imaging systems to transfer phase information into the image plane. Typically we need partially or fully coherent illumination configurations to confer phase sensitivity to an optical system. Phase imaging will be described in more detail in chapter 12.

Implications

The TCC framework is a rather general description for partially coherent imaging systems. The above examples demonstrate that the extreme cases of fully coherent and incoherent

[1] We write "proportional" because constants have been neglected entirely in this discussion.

imaging are included in the TCC framework. Beyond these cases, the TCC can be used to describe more general imaging configurations, including partial spatial coherence or dark field imaging, the latter of which we will find to be nonlinear. For two-dimensional imaging systems the TCC is four-dimensional. It is thus challenging to visualize the TCC. However, we will now turn to a simplified version of the TCC which allows to gain further insight into imaging systems.

8.2 Absorption and phase transfer function

The goal of this section is to approximate the TCC with simpler imaging models which will lead us to the amplitude and phase transfer functions. Suppose a specimen transmission function is given by

$$T(\boldsymbol{r}) = \exp\left[\mu(\boldsymbol{r}) + j\phi(\boldsymbol{r})\right], \tag{8.19}$$

where we assume μ as well as ϕ to be small so that we can approximate the latter expression by

$$T(\boldsymbol{r}) \approx (1 + \mu(\boldsymbol{r}) + j\phi(\boldsymbol{r})). \tag{8.20}$$

Then the Fourier transform of this expression is given by

$$\mathcal{T}(\boldsymbol{u}) \approx \delta(\boldsymbol{f}) + \tilde{\mu}(\boldsymbol{f}) + j\tilde{\phi}(\boldsymbol{f}). \tag{8.21}$$

With this we have

$$\begin{aligned}\mathcal{T}(\boldsymbol{f}_1)\mathcal{T}^*(\boldsymbol{f}_2) &\approx \left[\delta(\boldsymbol{f}_1) + \tilde{\mu}(\boldsymbol{f}_1) + j\tilde{\phi}(\boldsymbol{f}_1)\right] \cdot \left[\delta(\boldsymbol{f}_2) + \tilde{\mu}^*(\boldsymbol{f}_2) - j\tilde{\phi}^*(\boldsymbol{f}_2)\right] &(8.22)\\ &\approx \delta(\boldsymbol{f}_1)\delta(\boldsymbol{f}_2) + \delta(\boldsymbol{f}_1)\tilde{\mu}^*(\boldsymbol{f}_2) - j\delta(\boldsymbol{f}_1)\tilde{\phi}^*(\boldsymbol{f}_2)\\ &\quad + \tilde{\mu}(\boldsymbol{f}_1)\delta(\boldsymbol{f}_2) + j\tilde{\phi}(\boldsymbol{f}_1)\delta(\boldsymbol{f}_2) &(8.23)\end{aligned}$$

where second-order terms in $\tilde{\mu}$ and $\tilde{\phi}$ were dropped, assuming such contributions are small. Inserting the latter expression into Eq. 8.6, we get

$$
\begin{aligned}
I(\boldsymbol{r}) \approx & \iint \delta(\boldsymbol{f}_1)\,\delta(\boldsymbol{f}_2) C(\boldsymbol{f}_1,\boldsymbol{f}_2) \exp\left(j2\pi(\boldsymbol{f}_1-\boldsymbol{f}_2)\boldsymbol{r}\right) d\boldsymbol{f}_1\, d\boldsymbol{f}_2 \\
& + \iint \delta(\boldsymbol{f}_1)\,\tilde{\mu}^*(\boldsymbol{f}_2) C(\boldsymbol{f}_1,\boldsymbol{f}_2) \exp\left(j2\pi(\boldsymbol{f}_1-\boldsymbol{f}_2)\boldsymbol{r}\right) d\boldsymbol{f}_1\, d\boldsymbol{f}_2 \\
& - j \iint \delta(\boldsymbol{f}_1)\,\tilde{\phi}^*(\boldsymbol{f}_2) C(\boldsymbol{f}_1,\boldsymbol{f}_2) \exp\left(j2\pi(\boldsymbol{f}_1-\boldsymbol{f}_2)\boldsymbol{r}\right) d\boldsymbol{f}_1\, d\boldsymbol{f}_2 \\
& + \iint \tilde{\mu}(\boldsymbol{f}_1)\,\delta(\boldsymbol{f}_2) C(\boldsymbol{f}_1,\boldsymbol{f}_2) \exp\left(j2\pi(\boldsymbol{f}_1-\boldsymbol{f}_2)\boldsymbol{r}\right) d\boldsymbol{f}_1\, d\boldsymbol{f}_2 \\
& + j \iint \tilde{\phi}(\boldsymbol{f}_1)\,\delta(\boldsymbol{f}_2) C(\boldsymbol{f}_1,\boldsymbol{f}_2) \exp\left(j2\pi(\boldsymbol{f}_1-\boldsymbol{f}_2)\boldsymbol{r}\right) d\boldsymbol{f}_1\, d\boldsymbol{f}_2 \qquad (8.24) \\
& = C(\boldsymbol{0},\boldsymbol{0}) \\
& + \underbrace{\int \tilde{\mu}^*(\boldsymbol{f}) C(\boldsymbol{0},\boldsymbol{f}) \exp\left(-j2\pi\boldsymbol{f}\boldsymbol{r}\right) d\boldsymbol{f}}_{\text{substitute } \boldsymbol{f}\leftarrow-\boldsymbol{f}} \\
& - j \underbrace{\int \tilde{\phi}^*(\boldsymbol{f}) C(\boldsymbol{0},\boldsymbol{f}) \exp\left(-j2\pi\boldsymbol{f}\boldsymbol{r}\right) d\boldsymbol{f}}_{\text{substitute } \boldsymbol{f}\leftarrow-\boldsymbol{f}} \\
& + \int \tilde{\mu}(\boldsymbol{f}) C(\boldsymbol{f},\boldsymbol{0}) \exp\left(j2\pi\boldsymbol{f}\boldsymbol{r}\right) d\boldsymbol{f} \\
& + j \int \tilde{\phi}(\boldsymbol{f}) C(\boldsymbol{f},\boldsymbol{0}) \exp\left(j2\pi\boldsymbol{f}\boldsymbol{r}\right) d\boldsymbol{f} \qquad (8.25) \\
& = C(\boldsymbol{0},\boldsymbol{0}) \\
& + \int \tilde{\mu}^*(-\boldsymbol{f}) C(\boldsymbol{0},-\boldsymbol{f}) \exp\left(j2\pi\boldsymbol{f}\boldsymbol{r}\right) d\boldsymbol{f} \\
& - j \int \tilde{\phi}^*(-\boldsymbol{f}) C(\boldsymbol{0},-\boldsymbol{f}) \exp\left(j2\pi\boldsymbol{f}\boldsymbol{r}\right) d\boldsymbol{f} \\
& + \int \tilde{\mu}(\boldsymbol{f}) C(\boldsymbol{f},\boldsymbol{0}) \exp\left(j2\pi\boldsymbol{f}\boldsymbol{r}\right) d\boldsymbol{f} \\
& + j \int \tilde{\phi}(\boldsymbol{f}) C(\boldsymbol{f},\boldsymbol{0}) \exp\left(j2\pi\boldsymbol{f}\boldsymbol{r}\right) d\boldsymbol{f} \qquad (8.26) \\
& = C(\boldsymbol{0},\boldsymbol{0}) \\
& + \int \tilde{\mu}(\boldsymbol{f}) \left[C(\boldsymbol{f},\boldsymbol{0}) + C(\boldsymbol{0},-\boldsymbol{f})\right] \exp\left(j2\pi\boldsymbol{f}\boldsymbol{r}\right) d\boldsymbol{f} \\
& + \int \tilde{\phi}(\boldsymbol{f}) j \left[C(\boldsymbol{f},\boldsymbol{0}) - C(\boldsymbol{0},-\boldsymbol{f})\right] \exp\left(j2\pi\boldsymbol{f}\boldsymbol{r}\right) d\boldsymbol{f} \\
& = C(\boldsymbol{0},\boldsymbol{0}) \\
& + \int \tilde{\mu}(\boldsymbol{f}) ATF(\boldsymbol{f}) \exp\left(j2\pi\boldsymbol{f}\boldsymbol{r}\right) d\boldsymbol{f} \\
& + \int \tilde{\phi}(\boldsymbol{f}) PTF(\boldsymbol{f}) \exp\left(j2\pi\boldsymbol{f}\boldsymbol{r}\right) d\boldsymbol{f}, \qquad (8.27)
\end{aligned}
$$

where we defined the *absorption transfer function* (ATF)

$$ATF\left(\boldsymbol{f}\right) = C\left(\boldsymbol{f},\boldsymbol{0}\right) + C\left(\boldsymbol{0},-\boldsymbol{f}\right) \tag{8.28}$$

$$= \int \mathcal{S}\left(\boldsymbol{f}'\right)\mathcal{H}\left(\boldsymbol{f}'+\boldsymbol{f}\right)\mathcal{H}^{*}\left(\boldsymbol{f}'\right)d\boldsymbol{f}' + \int \mathcal{S}\left(\boldsymbol{f}'\right)\mathcal{H}\left(\boldsymbol{f}'\right)\mathcal{H}^{*}\left(\boldsymbol{f}'-\boldsymbol{f}\right)d\boldsymbol{f}' \tag{8.29}$$

and the *phase transfer function* (PTF)

$$PTF\left(\boldsymbol{f}\right) = j\left[C\left(\boldsymbol{f},\boldsymbol{0}\right) - C\left(\boldsymbol{0},-\boldsymbol{f}\right)\right] \tag{8.30}$$

$$= j\int \mathcal{S}\left(\boldsymbol{f}'\right)\mathcal{H}\left(\boldsymbol{f}'+\boldsymbol{f}\right)\mathcal{H}^{*}\left(\boldsymbol{f}'\right)d\boldsymbol{f}' - j\int \mathcal{S}\left(\boldsymbol{f}'\right)\mathcal{H}\left(\boldsymbol{f}'\right)\mathcal{H}^{*}\left(\boldsymbol{f}'-\boldsymbol{f}\right)d\boldsymbol{f}'. \tag{8.31}$$

The utility in the ATF and PTF is to reduce the partially coherent imaging model (Eqn. 8.6)

$$I_{\text{cam}}\left(\boldsymbol{r}\right) = \iint \mathcal{T}\left(\boldsymbol{f}_1\right)\mathcal{T}^{*}\left(\boldsymbol{f}_2\right)C\left(\boldsymbol{f}_1,\boldsymbol{f}_2\right)\exp\left(j2\pi\left(\boldsymbol{f}_1-\boldsymbol{f}_2\right)\boldsymbol{r}\right)d\boldsymbol{f}_1\,d\boldsymbol{f}_2 \tag{8.32}$$

into the much simpler model

$$I_{\text{cam}}\left(\boldsymbol{r}\right) \approx C\left(\boldsymbol{0},\boldsymbol{0}\right) + \int \tilde{\mu}\left(\boldsymbol{f}\right)ATF\left(\boldsymbol{f}\right)\exp\left(j2\pi\boldsymbol{f}\boldsymbol{r}\right)\,d\boldsymbol{f} + \int \tilde{\phi}\left(\boldsymbol{f}\right)PTF\left(\boldsymbol{f}\right)\exp\left(j2\pi\boldsymbol{f}\boldsymbol{r}\right)\,d\boldsymbol{f}. \tag{8.33}$$

We note that upon discretization the latter model can be expressed in terms of vectors and matrices

$$\boldsymbol{I} = c + \boldsymbol{F}^{\dagger}\text{diag}\left(ATF\right)\boldsymbol{F}\boldsymbol{\mu} + \boldsymbol{F}^{\dagger}\text{diag}\left(PTF\right)\boldsymbol{F}\boldsymbol{\phi}, \tag{8.34}$$

where $c = C\left(\boldsymbol{0},\boldsymbol{0}\right)$ is a constant and $\boldsymbol{F}$ is a Fourier matrix.

Listing 8.1 contains code to compute the weak object transfer function. We note that this is a minimal code example. For an even and odd number of pixels per dimension, different types of computations have to be performed to guarantee the transfer functions have certain symmetry properties.

```
function [PTF, ATF] = getTransferFunctions(source, pupil)
% ATF: absorption transfer function
% PTF: phase transfer function
% assumes object exp(mu-j*phi) ~ 1 + mu - j*phi

term1 = ifft2( fft2( source .* conj(pupil) ) .* ...
         conj( fft2( conj(pupil) ) ) );
term1 = rot90( term1, 2 );
term2 = ifft2( fft2( source .* pupil ) .* ...
         conj( fft2( pupil ) ) );
```

```
11
12 term1 = fftshift(term1);
13 term2 = fftshift(term2);
14
15 normalization = sum( source .* abs(pupil).^2, [1,2] );
16 ATF = (term1 + term2)/normalization;
17 PTF = 1j*(term1 - term2)/normalization;
18 end
```

Listing 8.1: getTransferFunctions.m (Matlab)

8.3 Bright field

We define a bright field optical system to (1) have a centro-symmetric source, (2) a symmetric pupil, and (3) the pupil to be real (i.e. aberration-free). Mathematically these three conditions can be written as

$$1.\, \mathcal{S}\left(-\boldsymbol{f}\right) = \mathcal{S}\left(\boldsymbol{f}\right), \tag{8.35}$$

$$2.\, \mathcal{H}\left(-\boldsymbol{f}\right) = \mathcal{H}\left(\boldsymbol{f}\right), \tag{8.36}$$

$$3.\, \mathcal{H}^{*}\left(\boldsymbol{f}\right) = \mathcal{H}\left(\boldsymbol{f}\right). \tag{8.37}$$

But then the PTF can be written as

$$\begin{aligned} PTF\left(\boldsymbol{f}\right) &= j\int \mathcal{S}\left(\boldsymbol{f}'\right)\mathcal{H}\left(\boldsymbol{f}'+\boldsymbol{f}\right)\mathcal{H}^{*}\left(\boldsymbol{f}'\right)d\boldsymbol{f}' \\ &\quad - j\int \mathcal{S}\left(\boldsymbol{f}'\right)\mathcal{H}\left(\boldsymbol{f}'\right)\mathcal{H}^{*}\left(\boldsymbol{f}'-\boldsymbol{f}\right)d\boldsymbol{f}' && (8.38) \\ &= j\int \mathcal{S}\left(\boldsymbol{f}'\right)\mathcal{H}\left(\boldsymbol{f}'+\boldsymbol{f}\right)\mathcal{H}^{*}\left(\boldsymbol{f}'\right)d\boldsymbol{f}' \\ &\quad - j\int \mathcal{S}\left(-\boldsymbol{f}'\right)\mathcal{H}\left(-\boldsymbol{f}'\right)\mathcal{H}^{*}\left(-\boldsymbol{f}'-\boldsymbol{f}\right)d\boldsymbol{f}' && (8.39) \\ &= j\int \mathcal{S}\left(\boldsymbol{f}'\right)\mathcal{H}\left(\boldsymbol{f}'+\boldsymbol{f}\right)\mathcal{H}^{*}\left(\boldsymbol{f}'\right)d\boldsymbol{f}' \\ &\quad - j\int \mathcal{S}\left(\boldsymbol{f}'\right)\mathcal{H}\left(\boldsymbol{f}'+\boldsymbol{f}\right)\mathcal{H}^{*}\left(\boldsymbol{f}'\right)d\boldsymbol{f}' && (8.40) \\ &= 0. && (8.41) \end{aligned}$$

We find that aberration-free bright field optical system transmits no phase information. In a similar fashion, we find that the absorption transfer function becomes

$$ATF\left(\boldsymbol{f}\right) = 2\int \mathcal{S}\left(\boldsymbol{f}'\right)\mathcal{H}\left(\boldsymbol{f}'+\boldsymbol{f}\right)\mathcal{H}^{*}\left(\boldsymbol{f}'\right)d\boldsymbol{f}' \tag{8.42}$$

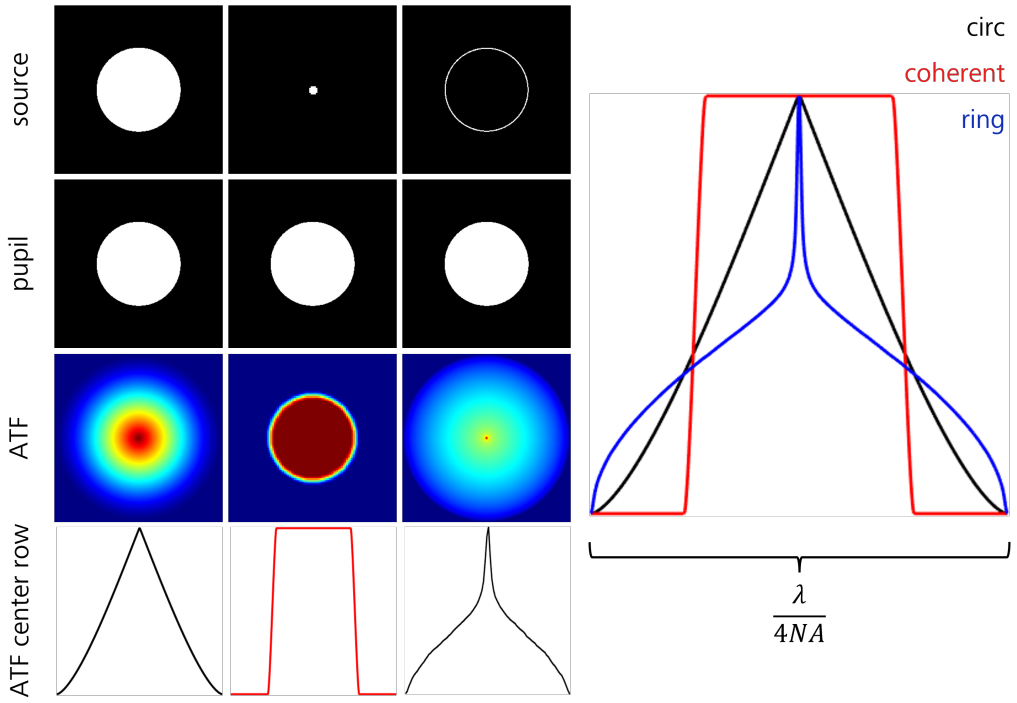


Figure 8.2: ATF coherence engineering. Varying the shape of the source in a bright field microscope allows for engineering spatial frequency transfer of absorption information.

for a bright field optical system.

Figure 8.2 shows an example of *coherence engineering* in a bright field microscope. The left column shows a source, for which the illumination and objective numerical apertures are matched. The central row of the ATF is shown on the bottom panel. In the middle column a quasi coherent source shown, for which the ATF resembles a top hat function. Notably, the resolution of coherent illumination (middle column) is only half as high as that for matched NA illumination (left column), as witness by the sudden drop of the ATF. However, certain mid spatial frequencies are transferred better for coherent illumination than for incoherent illumination. The third column features a ring-shaped illumination. It is seen that the corresponding ATF (blue line) drastically loses mid spatial frequencies. However, for certain high-spatial frequencies the transfer function has better signal to noise than for the incoherent case.

The takeaway from Fig. 8.2 is that the source in a bright field system can be engineered for the imaging task at hand. For example, if we are designing a microscope to image amplitude gratings in transmission with a spatial frequency close to the incoherent diffraction limit $\Delta x = \lambda/4\text{NA}$, it can be of benefit to consider a ring shaped illumination. On the other hand, we will see in chapter 12 that quasi-coherent sources can be beneficial for phase imaging. We may consider the area (or volume) under the ATF curve as a figure of merit for different sources in a bright field microscope. Then ring-shaped sources would actually not perform very well: the slight improvement at high spatial frequencies is relatively small as compared to loss of spatial frequencies in the low and mid spatial frequency range.

An experimental example of an unlabeled mouse histological section using matched NA full condensor and ring-shaped condensor illumination is shown in Fig. 8.3. Here panel a) was recorded full matched NA full condensor. Panel b) was recorded with ring-shaped condensor. Clearly, the image with ring.shaped illumination features improved high-frequency absorption information.

8.4 Oblique illumination

In an oblique illumination microscope, only half of the condensor aperture is opened. We used code listing 8.1 to simulate the absorption and phase transfer functions for such an optical system under varying the source shape, as shown inn Fig. 8.4. In the left column we see a matched NA configuration, that is, the condensor and objective numerical apertures have the same diameter. Half of the condensor aperture is blocked. The PTF undergoes a sign change (negative blue, green zero, red positive) about the central half of Fourier space. If we depart from the matched illumination condition and decrease the source size to half of the objective NA in this example (middle column), we see that the PTF has a large region in the center of Fourier space which is close to zero. Thus significantly less low frequency phase information is contained in the image when using an unmatched illumination configuration. The right column shows the case of a ring-shaped source matched in diameter to the objective NA. Here the PTF has the largest coverage in Fourier space. The central row of each PTF is overlayed in the image on the right. Clearly, the ring-shaped illumination outperforms half pupil illumination both at low and high frequencies (with a small tradeoff in mid spatial frequencies). For later reference, we also show the ATF in each illumination configuration. For the moment we only need

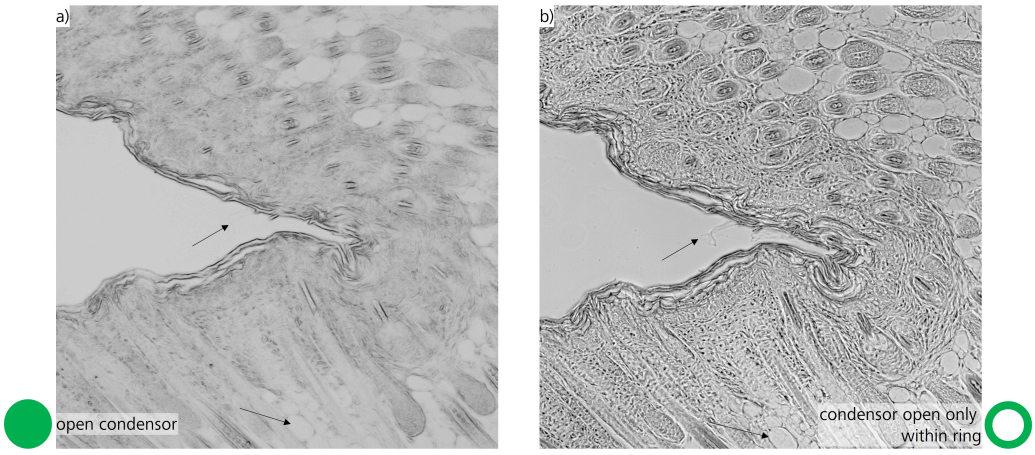


Figure 8.3: Example for bright field micrograph of an unlabeled saggital mouse section with A) full condensor aperture and B) ring-shaped condensor aperture. The right images features improved absorption information at higher spatial frequencies, as highlighted by the black arrows.

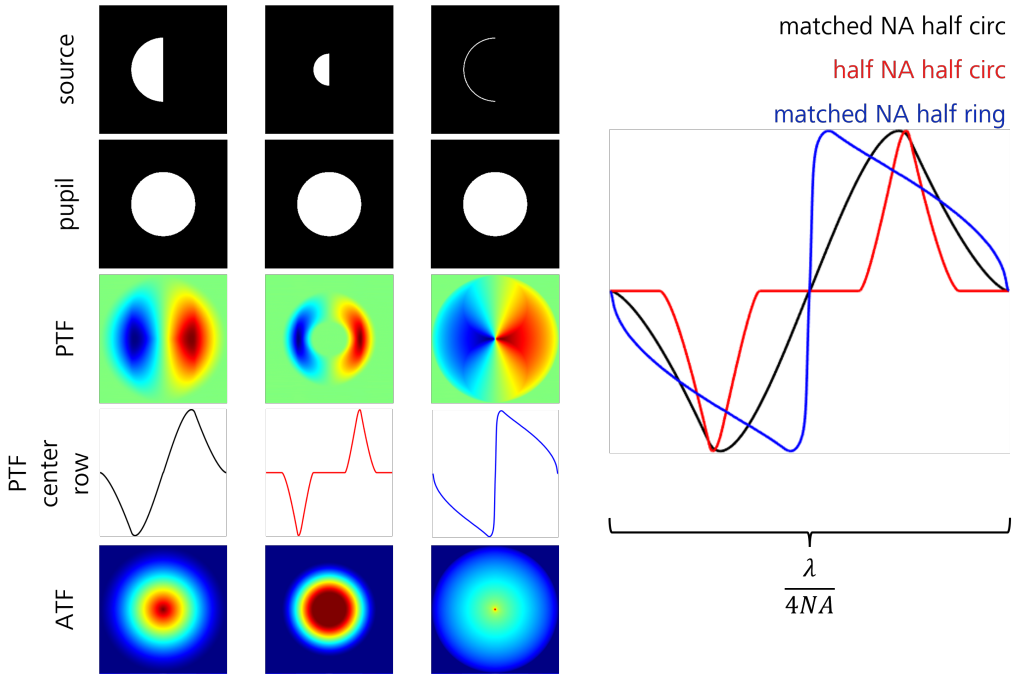


Figure 8.4: PTF coherence engineering. Varying the shape of the source in an oblique illumination microscope allows for engineering spatial frequency transfer of phase information. Note that the absorption transfer function does not vanish. The image intensities contain mixed absorption and phase information.

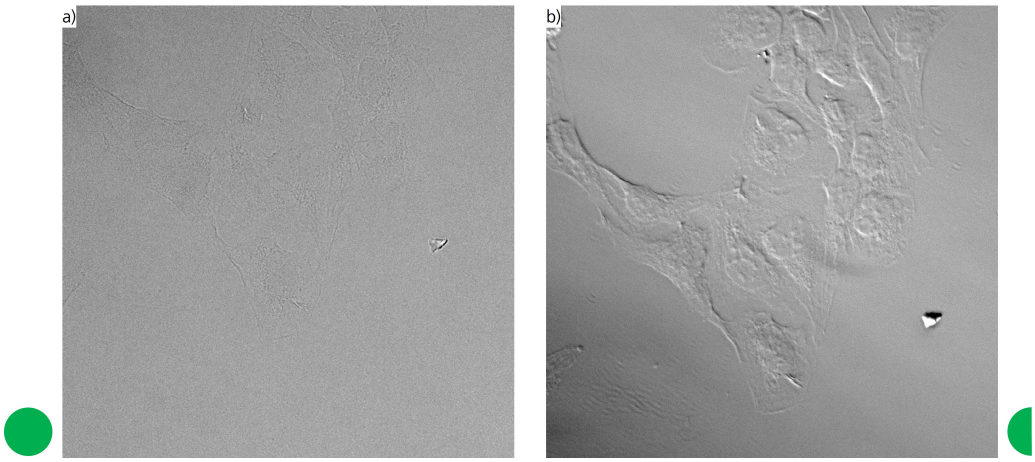


Figure 8.5: Example for bright field micrograph of an unlabeled adherent cell with A) full condensor aperture opened and B) half pupil oblique illumination. The right image features improved phase information.

to take note that the ATF does not vanish. We will discuss in chapter 12 how to get rid of absorption information and harness pure phase information from oblique illumination data. In some cases, for example for unlabeled thin cellular culture, the phase shift is naturally larger than the absorption, such that we may also simply neglect the absorption contribution to the measured image intensities. An example for such a situation is given in Fig. 8.5. Panel a) shows a bright field image of an unlabeled adherent cell. Panel b) shows the same cell under oblique illumination. Note the relief-like appearance of the oblique illumination image (seemingly casting shadows like a 3D surface topography under oblique illumination). Clearly, panel a) indicates that there is very little absorption information in the image, while the contrast in panel b) is primarily due to the phase shift induced by the specimen.

8.5 Dark field

Despite the mathematical progress that we were able to make in the previous sections, there are contrast mechanism beyond the theoretical scope of the ATF and PTF linear framework. An example is *dark field* microscopy. For simplicity, we assume the optical system has an illumination consisting of an off-axis point source

$$\mathcal{S}(\boldsymbol{f}) = \delta(\boldsymbol{f} - \boldsymbol{s}). \tag{8.43}$$

We assume that the off axis angle $\lambda \boldsymbol{s}$ is larger than the objective numerical aperture. Under these circumstances, there is zero overlap between the region where the source is non-zero and the region where the objective pupil is non-zero. Hence the ATF and PTF integrals vanish because each integral consists of terms such as $\mathcal{S}(\boldsymbol{f}')\mathcal{H}(\boldsymbol{f}') = 0$ (or similar) that vanish. Hence for dark field microscopy we need to resort back to the

more general, but also more complicated, TCC framework. We first revisit the weak object approximation (Eq. 8.23) and include second-order terms. For further simplicity, we assume that the phase shift induced by the specimen is stronger than the absorption of the specimen. Under these conditions we have

$$\mathcal{T}(\boldsymbol{f}_1)\,\mathcal{T}^*(\boldsymbol{f}_2) \approx \left[\delta(\boldsymbol{f}_1) + \tilde{\mu}(\boldsymbol{f}_1) + j\tilde{\phi}(\boldsymbol{f}_1)\right] \cdot \left[\delta(\boldsymbol{f}_2) + \tilde{\mu}^*(\boldsymbol{f}_2) - j\tilde{\phi}^*(\boldsymbol{f}_2)\right] \tag{8.44}$$

$$= \underbrace{\text{1st order terms}}_{\text{neglect}} + \underbrace{\tilde{\mu}(\boldsymbol{f}_1)\,\tilde{\mu}^*(\boldsymbol{f}_2)}_{\text{neglect}} + \tilde{\phi}(\boldsymbol{f}_1)\,\tilde{\phi}^*(\boldsymbol{f}_2) \tag{8.45}$$

$$\approx \tilde{\phi}(\boldsymbol{f}_1)\,\tilde{\phi}^*(\boldsymbol{f}_2), \tag{8.46}$$

where we neglected 1st-order terms since their ATF and PTF vanish and we neglected the absorption term by assumption. In principle the absorption can be included here, but we would like the result to be simple. Using Eq. 8.6 we then get

$$I_{\text{cam}}(\boldsymbol{r}) = \iiint \delta(\boldsymbol{f}-\boldsymbol{s})\,\tilde{\phi}(\boldsymbol{f}_1)\,\tilde{\phi}^*(\boldsymbol{f}_2) \times \ldots$$
$$\mathcal{H}(\boldsymbol{f}+\boldsymbol{f}_1)\,\mathcal{H}^*(\boldsymbol{f}+\boldsymbol{f}_2)\exp\left(j2\pi\,(\boldsymbol{f}_1-\boldsymbol{f}_2)\,\boldsymbol{r}\right)\,d\boldsymbol{f}\,d\boldsymbol{f}_1\,d\boldsymbol{f}_2 \tag{8.47}$$

$$= \iint \tilde{\phi}(\boldsymbol{f}_1)\,\tilde{\phi}^*(\boldsymbol{f}_2) \times \ldots \tag{8.48}$$
$$\mathcal{H}(\boldsymbol{s}+\boldsymbol{f}_1)\,\mathcal{H}^*(\boldsymbol{s}+\boldsymbol{f}_2)\exp\left(j2\pi\,(\boldsymbol{f}_1-\boldsymbol{f}_2)\,\boldsymbol{r}\right)\,d\boldsymbol{f}_1\,d\boldsymbol{f}_2$$

$$= \left|\int \tilde{\phi}(\boldsymbol{f})\,\mathcal{H}(\boldsymbol{s}+\boldsymbol{f})\exp\left(j2\pi\boldsymbol{f}\right)\,d\boldsymbol{f}\right|^2. \tag{8.49}$$

The last line indicates that *the intensity measured in a dark field system is non-linearly related to the phase of the specimen.* This result is still true if we (1) include absorption and (2) multiple point or continuous sources that are located beyond the pupil support. Because of the non-linear relation to the specimen properties, dark field micrographs are harder to interpret than bright field or oblique illumination images.

Chapter 9

Imaging Systems

ToDo: Write an abstract here, which summarizes the topics covered here.

The task of an imaging system is to generate an image of a sample, which is also called "object". In this chapter we will consider first the two extreme cases of fully coherent imaging and fully incoherent imaging.

9.1 Fully coherent imaging

Here the three-dimensional object is represented by its spatially dependent susceptibility $\chi(x,y,z)$. This susceptibility characterizes the emitted electric field if driven by an incident local electric field as described by the Lippmann schwinger equation. Assuming only single scattering events, it can be written in the first Born approximation: $E_{out}(x,y,z) = \chi(x,y,z)\, E_{in}(x,y,z)$. As we are only interested in the far field and we assume a monochromatic wave, we have to consider only the propagating electromagnetic waves. This is conveniently achieved by intersecting the Fourier-transform of E_{out} with the k-sphere, the sphere of possible plane waves of constant wavelengths $|k| = k_0$, which is related to the Ewald-sphere in scattering theory.

We further assume a linear shift invariant imaging system imaging a plane to a plane. Such systems need to fulfill a condition, called the Abbe sine condition, ensuring that the in-plane k-vectors are given by a constant magnification M to the corresponding k-vectors in the source space: $k'_{x,y} = k_{x,y}/M$. This further restricts our k-sphere to a maximum angle α from the optical axis of our imaging system. This curved cab of the k-shere is called the McCutchen pupil $p(k_x, k_y, k_z)$.

The detected image is the intensity $I(x',y') = |E(x',y')|^2$, measured at the detector, which we assume to be a pixelized camera placed in the image plane. Thus all together we can approximate the physics of imaging of our sample $\chi(x,y,z)$ in Fourier space as:

$$\hat{I}\left(k'_x, k'_y\right) = Proj_{k_z}\left\{\left[\hat{\chi}\left(k_x,k_y,k_z\right) \circledast_{3D} \hat{E_{in}}\left(k_x,k_y,k_z\right)\right] p\left(k_x,k_y,k_z\right)\right\}$$

with the hat over a symbol denoting the 2- or 3D Fourier transformation, $\circledast_{3D}$ a three-dimensional convolution, the lateral k-vectors in the image plane being related to the k-vectors in object space by Abbe's sine condition $k'_x = k_x/M$, $k'_y = k_y/M$. $Proj_{k_z}$ refers

to the projection along the k_z direction. Since we can assume the McCutchen pupil p to be infinitely thin, we can think of the projection[1] as a constraint to the axial component of the k-vector, which is obtained by the restrictions $k_x^2 + k_y^2 + k_z^2 = k_0^2$ of the sphere in k-space: $k_z = \sqrt{k_0^2 - k_x^2 - k_y^2}$. In real space this amounts to:

$$I(x', y') = [\chi(x, y, z)\, E_{in}(x, y, z)] \circledast_{3D} h_{amp}(x, y, z)|_{x'=Mx,\, y'=My,\, z=0}$$

where $h_{amp}(x, y, z)$ denotes the amplitude point spread function, which is the 3D inverse Fourier-transformation of the McCutchen pupil $p(k_x, k_y, k_z)$. Here, the image is always formed with the detector placed in a conjugate postion to the zero positon of the object coordinate system. To achieve an image stack, the objekt is typically translated through this position.

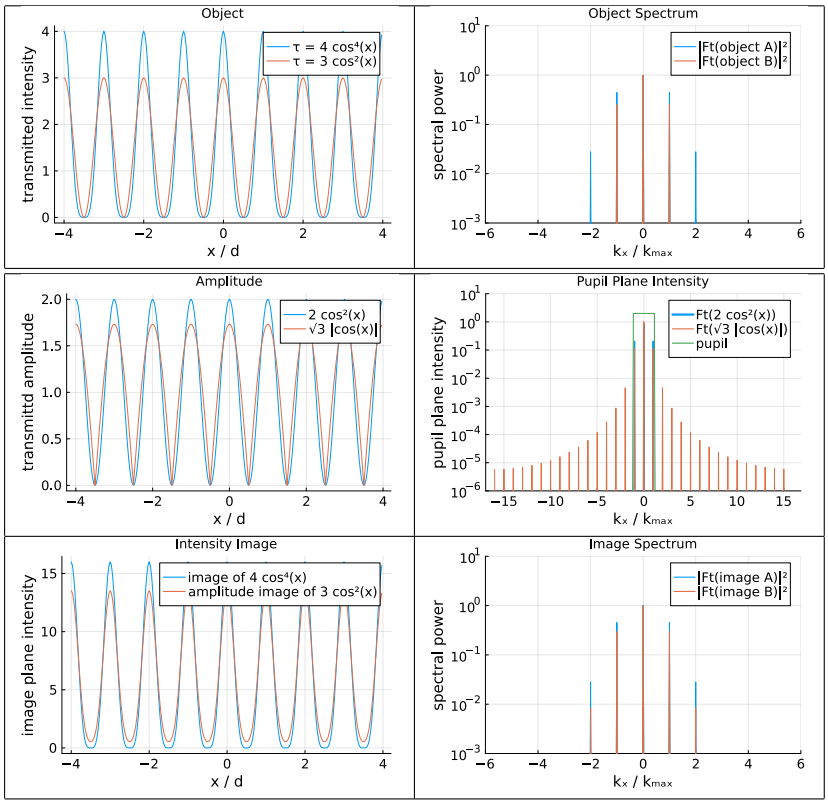


Figure 9.1: (a) example objects; (b) power spectra of objects; (c) amplitudes after object; (d) pupil plane intensities; (e) image plane intensities; (f) power spectra of images indicating a distortion. Note the red bars at $2k_{max}$ indicating artefact frequencies in the image (panel f), which are not present in the object (panel b).

Note that the susceptibility $\chi(x, y, z)$ linearly influences the amplitude. Yet we classically understand by the term "imaging" a transfer of intensity from the object plane

[1] if we have only forward scattered components in the detected field

($z = 0$) to the image plane. If we consider for example (Fig. 9.1) a purely transmissive object described by the intensity transmission $\tau(x, y)$,we would obtain $\chi(x, y) = \sqrt{\tau(x, y)}$. Note that taking the positive root is, apart from a global phase factor, the only solution producing the correct intensity transmittance not altering the phase of the light after the sample. Even though there is a linear transfer of amplitude in a coherent imaging system, there is typically no linear relationship between the intensity at the object and the intensity at the image plane. To illustrate this, let us consider two example intensity-tranmitting objects being illuminated by a plane wave ($E_{in} = 1$) in transmission geometry: $\tau_A(x, y) = 4\cos^4(\frac{k_{max}}{2}x) = \frac{3}{2} + 2\cos(k_{max}x) + \frac{\cos(2k_{max}x)}{2}$ and $\tau_B(x, y) = 3\cos^2(\frac{k_{max}}{2}x) = \frac{3}{2} + \frac{3}{2}\cos(k_{max}x)$, with k_{max} referring to the highest spatial frequency being transferred by the linear amplitude-imaging system, i.e. the first diffraction order at the edge of the pupil. Since we are assume a purely transmissive structure not changing the phase, we obtain $\chi_A(x, y) = 2\cos^2(\frac{k_{max}}{2}x) = 1 + \cos(k_{max}x)$ and $\chi_B(x, y) = \sqrt{3}\,|\cos(k_{max}x)|$. Note that the transmitted amplitude $E_{in}\chi_A$ is fully described by three peaks in Fourier space whereas $E_{in}\chi_B$ contains an infinite series of spatial frequencies. This may seem surprising, since the intensity τ_B contains lower spatial frequencies than $\boldsymbol{\tau_A}$. Finally the image of these intensity structures result in a perfect reproduction of τ_A in the image plane, whereas the image of τ_B is demodulated and distorted. Note that τ_A contains higher spatial frequencies and is perfectly reproduced in the image plane. The "distortion" of τ_B becomes obvious, if we analyze the spatial frequencies in the finally obtained intensity distributions.

These examples offer some interesting insights into coherent imaging of intensity structures: In some cases as in the τ_A example high-frequencies are encoded in lower frequencies to be perfectly reproduced in the image plane, seemingly allowing to "image" beyond the limit as given by k_{max}. We can call this limit the "coherent Abbe limit". However, in other cases as in τ_A, objects containing only frequencies below the coherent Abbe limit that should be perfectly reproduced are demodulated and distorted, creating frequencies in the image which are not present in the original object. This clearly demonstrates the problems of a non-linear relationship between the transmitted object- and image intensity.

9.2 Fully incoherent imaging

In incoherent imaging we assume that there is no phase correlation between any of the light being emitted by distinct points in the object. In any far field illumination geometry, this assumption can never be fullfilled well for imaging of the scattered or transmitted light, since object points at distances smaller than the wavelength will always be illuminated with highly correlated electric fields. Yet, this assumption is fulfilled extremely well for the case of fluorescence microscopy, where neighbouring fluorophores generally show no correlation in their emitted electric fields. In this case the image is formed by each emitter forming an independent intensity image and we simply have to add all those intensity images on the detector. This leads to a convolution of the object emitter density $\rho(x, y, z)$ with the incoherent point spread function (PSF) $h(x, y, z)$:

$$I(x', y') = \rho(x, y, z) \circledast_{3D} h(x, y, z)|_{x'=Mx,\, y'=My,\, z=0}$$

where the incoherent point spread function is obtained by coherently summing all amplitudes of a single emitter as described in the previous section:

$$h(x,y,z) = |h_{amp}(x,y,z)|^2 = h_{amp}(x,y,z)\, h^*_{amp}(x,y,z)$$

Note that we assumed that all emitters have the same properties and that the detected light is monochromatic. If the latter is not the case, one needs to describe the PSF by an incoherent weighted sum over a range of wavelengths. If we simply assume that our detector is sampling in object space coordinates (x, y, z) and we move the sample through the nominal focus position $z = 0$, we can write a whole acquired image stack obtained by shifting the object along z through the focus as a simple three-dimensional convolution

$$I(x', y') = I(x, y) = \rho(x, y, z) \circledast_{3D} h(x, y, z)$$

9.3 Noise Types in Imaging

9.3.1 Photon noise

Photon noise, also called shot-noise, is a property of the light rather than the detector. Its statistical properties are in almost all situations very well described by a Poisson distribution, which is governed by

$$P(Y|\mu) = \frac{\mu^Y e^{-\mu}}{Y!}$$

Here P describes the probability, μ the expected number and Y the detected number of photons. The most important property of Poisson noise is that the variance is equal to the expected mean:

$$\sigma^2 = \mu$$

This means if no photons are expected, there will never be a detection event and the detected signal is noise-free, if no other sources of noise are present. It also means, that more expected photons always lead to more noise in your signal. Nevertheless, the signal to noise ratio (SNR) as defined by the strength of the signal in relation to the standard deviation of the noise is

$$SNR = \frac{\mu}{\sigma} = \frac{\mu}{\sqrt{\mu}} = \sqrt{\mu}$$

Thus the SNR increases with higher photon numbers.

Fig. 9.2 shows the Poisson distribution for various expected mean number of photons μ. Note, that the shape of the Poisson distribution only differs significantly from a Gaussian distribution with a scaled variance for very low photon numbers.

9.3.2 Readnoise

Imaging typically uses two types of detectors, **bucket** and **array** detectors. A **P**hoto**M**ultiplyer **T**ube (PMT) or a Single Photon Avalanche photoDiode (SPAD) are examples of bucket detectors. They detect light inpinging on their surface and can be read out at very high rates. Bucket detectors are a core component of scanning microscopes. In contrast,

Charge **C**oulpled **D**evices (CCD) or **s**cientific **C**omplementary **M**etal **O**n **S**ilicon (sCMOS) are array detector, which provide thousands or millions of locations where light is measured in parralel (picture elements, pixels), which is then sequentially or partially in parallel digitized.

Both types of detectors come with noise originating from the electronics of the readout process. This "read noise" σ_{RN} is often specified as the standard deviation (**R**oot **M**ean **S**quare, RMS) in the amount of detected electrons e^- RMS.

It is interesting to discuss how such RMS values of the read noise can be interpreted. One way to interpret these read noise values would be to calculate the level of background intensity required to generate this amount of noise. This would correspond to the square of these values due to the scaling properties of Poisson-distributed photon noise. A detector with $1e^-$ RMS readnoise would correspond to a background level of one photon, whereas a detector with $2e^-$ RMS readnoise corresponds to a background level of $2^2 = 4$ photons. Stated in these terms, the former detector is 4 times as good as the latter. However, a more sensible way to compare the performance of detectors is the signal to noise ratio. Thus we should analyse the minimum integration time we need to achieve a predefined SNR being the ratio of the measured signal over the standard deviation of the noise. $SNR = \frac{N}{\sqrt{\sigma_{RN}^2+N}}$,where N signifies the number of detected signal photons. Note that for statistically independend sources of noise, the variances add and the variance of Poisson-distributed noise of N photons is N. We can now solve this for the number of required photons to achieve a predefined SNR:

$$N = \frac{SNR^2}{2} + SNR\sqrt{\sigma_{RN}^2 + \frac{SNR^2}{4}}$$

For signal to noise ratios being smaller than the read noise, we obtain approximately a proportionality of the required numer of photons to the read noise (Fig. 9.3). For small SNR, the readnoise is roughly proportional to the required number of photons with an additional photon-based offset. However, for large required SNRs, the read noise is not important.

9.3.3 Multiplicative Amplification Noise, Excess Noise

Photo multiplier tubes (PMTs), intensified (iCCD) cameras, as well as electron multiplying (emCCD) cameras amplify the generated photoelectrons before sending them to the detection electronics. Such a scheme is a very effective way to reduce the influence of the read noise on the detected signal. However, a single generated photoelectron can be substantially influenced by the statistics of the amplification process, since stochastic decisions are made in each amplification step. This type of amplification noise, also called "excess noise", is multiplicative in the sense that twice as many input photons yield twice as much noise. If we do not count detection events, due to the scaling property of Poisson noise, this multiplicative noise has the **same effect as a reduced detection efficiency**. The excess noise is quantitatively approximated by the excess noise factor

$$F^2 = 2(G-1)G^{-(N+1)/N} + 1/G$$

with the expected gain in signal G and the number of multiplication stages N. ("The noise performance of electron multiplying charge-coupled devices" June 2003IEEE Transactions

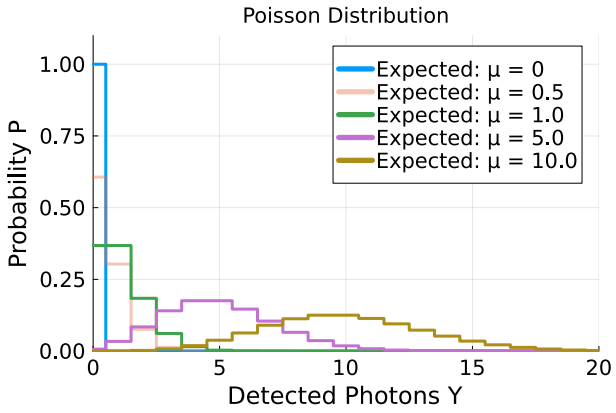


Figure 9.2: Photon noise. Photon noise (shot noise) is described by a Poisson distribution defining the probability for detected photons. The variance is equal to the expected mean μ.

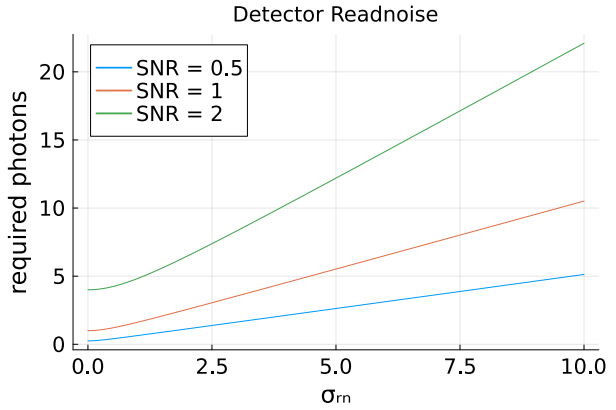


Figure 9.3: Detector readnoise. Required number of photons to achieve a predefined SNR in dependence on the readnoise of the detector.

on Electron Devices 50(5):1227 - 1232 DOI: 10.1109/TED.2003.813462). As a rule of thumb one can assume the effect of high gain ($G > 10$) to amount to a detection efficiency being reduced by $\sqrt{2}$. The amplification is achieved by a succession of many individual amplification steps each of them with a certain chance to cause an additional electron for each input electron. This leads to the resulting detected signal following an exponential distribution for each input electron.

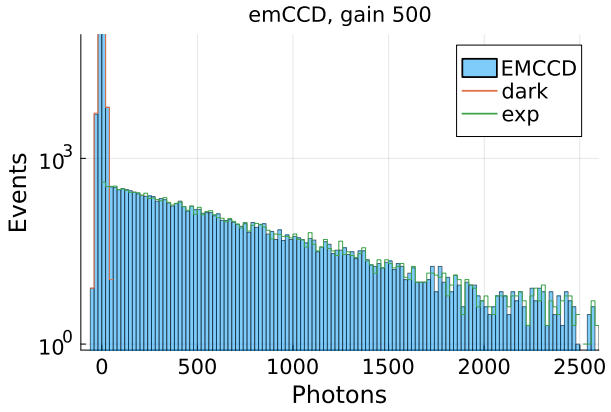


Figure 9.4: Intensity histogram of a detector with amplification noise.

Figure 9.4 shows the intensity histogram, that an emCCD camera at a gain of 500 would detect, if a very low number of photons (here 10,000, less than 4% propability per pixel to detect a photon) reached the camera. This histogram shows two contributions: the first one stems from the readout noise (magenta curve), whereas the second one shows the noise from the amplified photoelectrons (green curve).

Even though, as opposed to very low-noise detectors such as the newest scientific CMOS cameras, these detectors show no clear distinction between zero or one photon in the histogram, they can nevertheless be operated very efficiently in the single-photon counting regime. Operating cameras in the single-photon counting regime has the advantage of avoiding the excess noise factor described above, since any value above threshold is counted as a single photon. However, if the photon rates are too hight, too many pixels will have two photons detected and only one is counted.

Figure 9.5 illustrates the effect of single photon counting by thresholding on detector systems with amplification noise. Even though there is no clear distinction between zero and one photons in the histogram (Fig. 9.4), a threshold (e.g. 40 in the example) can be selected such that the loss in detection effciency (here about 9%) can be tolerated and keeping the fraction of false positive detections below 0.1% of the totally detected photons. In practice there are of course also other sources of noise such as dark noise buildup and clock-induced charge to consider as described below.

9.3.4 Dark Noise Buildup

Many detections, in particular cameras, exhibit a **dark noise** measured in electrons per second. This is also called “dark current”. Detection events which look like incoming

photons can arise due to thermal excitation of electrons on the detection surface (PMTs, APDs, SPADs), possible amplification electronics or within the storage of the captured photoelectrons in cameras. Such such events are exceedingly less likely to happen at cooler temperatures, scientific-grade detectors are often cooled to well below the freezing point of water. This often reduces the dark noise by orders of magnitude. E.g. in deep-cooled cameras the rate of dark events can be well below one electron in 10 seconds. However, such deep cooling comes with a price. Particularly in highly sensitive back-illuminated cameras the electrons need to travel a bit to the location of charge collection. Deep cooling (e.g. at liquid nitrogen temperatures) reduces the mobility of the charges, which decreases their chances to make to from the detection location to the storage area. In particularly for near infrared photons which are detected close to the surface of back illuminated CCDs, deep cooling can significantly reduce the effective quantum detection efficiency in the near infrared region. Note that dark noise builtup is really not only generating noise, but also an unwanted offset. The unwanted offset can be removed by carefully calibrating the camera at the same temperature and integration time as the data is acquired. This calibration should be performed by averaging many (typically between 20 and 100) dark images acquired in this way. Particularly in situations of strong dark noise builtup in cameras, one should perform this calibration on a pixel-by-pixel basis, since the generations of such unwanted electrons often varies greatly between the individual pixels leading to a pixel-dependent offset and noise. Dark noise builtup is usually not a problem for cameras running at video rate but it can be the decisive factor of noise for long exposure times beyond a second.

9.3.5 Clock Induced Charge

Clock induced charge (CIC) is a type of noise which is specific to CCD cameras. Large chips with many pixels require many clocking cycles to transport the charges to the read-out amplifiers. In each clocking cycle there is a small chance of generating a spurious electron. Over the many clocking cylcles this CIC can become significant. It can often be seen by careful inspection of a series of acquired dark frames, in which case the offset and noise varies with the location of the pixels. The pixels being most distant to the readout amplifier show more offset and noise than the ones close. Depending on the make of the camera, the manufactures may partially compensate for the offset by dark calibration lines and appropriate firmware on the cameras. For emCCDs used in the photon-counting mode, CIC is a major source of noise. Notably CMOS detectors do not show CIC as they do not need to move the electrons over the detector before readout.

9.3.6 Fixed Pattern Noise

Noise and offset can vary with position or time. This leads to an effect which is termed fixed pattern noise. Even dark frames can show pattern which is well visible. In scanning microscopes, this can be caused by electronic signals which are not well shielded. E.g. the 60 Hz of the main power supply is sometimes visible as offset fluctuations. Ideally one should detect this (e.g. with the help of Fourier-transforming the dark-frames) and remove the cause of the problem by implementing a better electronic shielding of the detectors. In sCMOS camera systems, another effect leading to fixed pattern noise is

particularly prominent. It is caused by the camera possessing separate amplifyers for each row (or often with rolling shutters being split in two, for each half row) of pixels. Even though the camera manufacturers carefully calibrate the amplifyer offset and gain for each amplifyer, there is typically a drift over time or other effects that lead to clearly visible lines in offset and gain, if a series of dark frames is acquired and averaged. Similarly a series of flat-field frames should also be acquired and evaluated (see below) to use the current values of offset and gain for each such readout row.

9.3.7 Digitization Noise

Signals are ofter acquired by a combination of amplification and digitization steps. The digitization is performed by an analogue to digital (AD) conversion process with a predefined number of bits. If the detected signals are of high quality, the smalles step of the AD converter may be above the noise level of the signal. This may lead to a signal-dependent bias, e.g. when the signal is 81 ± 9 photons, but the AD converter ranges are 0..50, 50..100, 100..150 etc. We would consistently interpret this signal as corresponding to 75 photons, even though it was consistently stronger. As a rule of thumb, the noise of the detection should be higher than the steps provided by the AD converter. To quantify how many bits are needed, it is useful to define the dynamic range:

$$DR_{lin} = \frac{I_{max}}{I_{min}}$$

were $I_{max}$is the brightest signal in the image and $I_{min}$is the darkest. If we cannot foresee the darkest signal or are unable to change the offset accordingly, we replace I_{min} by the readnoise standard deviation σ_{RN} (assuming no other noise sources) of our detector. If we want to evaluate a given camera performance we can replace I_{max} by the full-well capacity I_{FW}, i.e. the maximal amount of collectable photoelectrons. This leads to

$$DR_{lin} = \frac{I_{FW}}{\sigma_{RN}}$$

This value should be less than the number of digitization steps $2^{b_{min}}$ with b_{min} referring to the number of bits required in the AD converter. In practice the dynamic range of a system is often specified in decibel (dB):

$$DR_{dB} = 20 \log_{10} \left(\frac{I_{FW}}{\sigma_{RN}} \right)$$

The factor of 20 instead of 10 stems from the intensity ratio referring to powers rather than amplitudes of a signal. This digitization-bias can be regarded as a source of noise in the image and is therefore also called digitization noise. This means for a camera with a given dynamic range in dB one can estimate the number of bits needed to accurately represent the image

$$b_{min} > \log_2 \left(10^{\frac{DR_{dB}}{20}} \right) = DR_{dB} \log_2 \left(10^{\frac{1}{20}} \right) = \frac{DR_{dB}}{6.02}$$

E.g. an sCMOS camera with a full well capacity of $23\,ke^-$ and a read noise of $1e^-$ RMS has a dynamic range of $DB_{dB} \approx 87\,dB$ requiring a minimum of $b_{min} > 87dB/6.02 \approx 14.5$ bits.

In practice most confocal fluorescence microscopy data has very low signal in the range of 100 photons per pixel, in which case an 8 bit AD converter is sufficient to avoid such digitization noise. In camera-based systems and particularly in transmission microscopy the signal levels are much stronger. For the bright signals this is typically not a problem, since these signals also come with increased photon noise. However, scenes may have a very large dynamic range with low and high intensity signals being present. In this case the low photon numbers (e.g. zero photons) also have very low noise, in particular with current low-noise detection systems. A large number of bits are then required. Camera-based systems typcially use 12 to 16 bit AD converters to support the full dynamic range.

9.4 Acquiring Good Data

In imaging we typically acquire our images with electronic detectors. Even though many of such detector systems have the ability to change **gain** and **offset**, the detection noise is typically not dependent on the gain setting over orders of magnitude. In scientific-grade detection systems the noise in the image is almost always governed by the Poisson statistics of the photon.

The goal of all imaging efforts should be to acquire the most useful, not the most beautiful data. Since the noise is largely independent on the settings of the amplifyer, it is important to ensure that no signal is lost. Wrong settings, however, can easily lead to such loss of signal. In digital image acquisition, any pixels displaying the value of 0 or the highest value of the analogue to digital converter (e.g. 255 for 8 bit) are close to useless. Such pixels could well have corresponded to values of minus one, -10 or 260 in our example. This information cannot be regained in retrospect. Therefore it is highly recommended to use underflow/overflow indicator colormaps during image acquisition while adjusting the offset and gain of the detector and AD converter. These colormaps clearly show pixels at both limits of the digitalization range which should be avoided. Some situations require a sufficient dynamic range to not suffer from digitization artefacts.

9.5 Preprocessing: Offset and Flatfield Correction

As described above, to obtain high quality image data, we have to first preprocess it to correct for offset and gain variations. To this aim we need to acquire two sets of calibration data at the same settings which we used for acquiring our measurement data. This should be done in close proximity to our data acquisition to minimize possible drift effects. Ideally two calibration data set are acquired, one before and on after the measurement.

Each data sets consists of two series, a background and a flat field series. A minimum of 20 images is recommended for each series, such that the influence of statistical noise remains small. The following code demonstrates how to use the calibration series are used in preprocessing each input image:

```
using Statistics
function preprocess(image, dark_series, flat_series, dims=ndims(dark_series); result
        # Dark correction
        dark = mean(result_type, dark_series, dims=dims)
        # Flat field correction
```

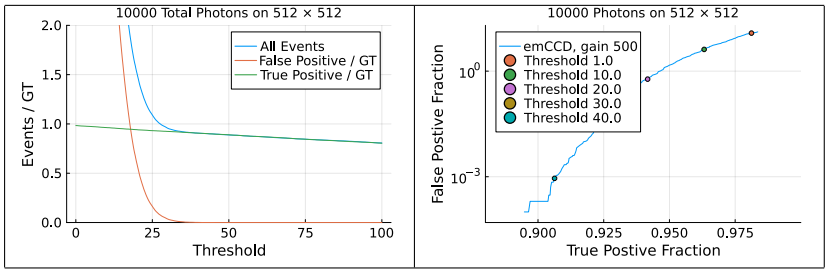


Figure 9.5: Detector readnoise. Required number of photons to achieve a predefined SNR in dependence on the readnoise of the detector.

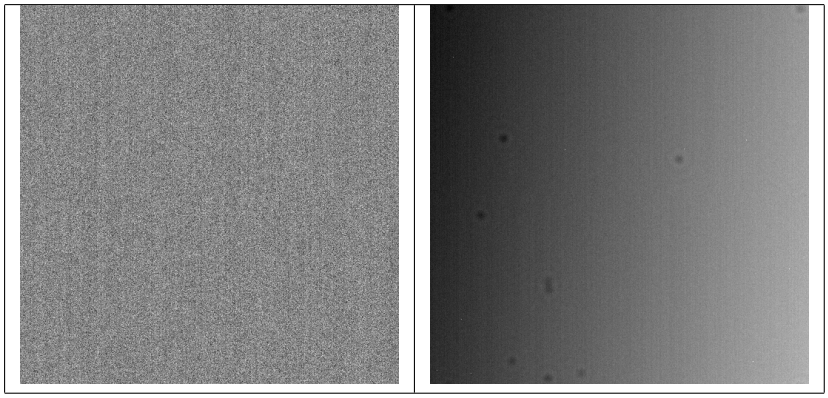


Figure 9.6: Offset and gain calibration. Series of at least 20 dark frames (left) and flatfield frames (right) are acquired at the time of data acquisition to be used in preprocessing for offset subtraction and flatfield correction. The dark image was simulated assuming 1.0 electron readnoise and offset fluctuations of 0.5 around the value of 100. The flatfield data simulation included specks of dust on the detector, row-wise line fluctuations and hot pixels.

```
        flat = mean(result_type, flat_series .- dark, dims=dims)
        preprocessed = (image .- dark) .* (mean(flat) ./ flat)
        return preprocessed
end
```

Note that this preprocessing converts image input data of typically integer type elements to image data with 32 bit floating point element type. To avoid any bias in our data processing, it is important to keep negative values rather than setting them to zero. Since read-noise (see section 9.3.2) is zero-mean we expect negative values to appear in our offset-corrected data. The code above could be improved specifically for sCMOS cameras by specifically adapting to the way that offset and gain variations manifest themselves in the data. Furthermore one could also modify the code to specifically address the drift of offset and gain, if calibration data was acquired before and after the image data.

9.6 Additions (to be put somewhere else)

2.10.1 additional trace property: $tr(A) = \sum_i \lambda_i$, with $\lambda_i$refering to the Eigenvalues of A.

Also: The trace of a Matrix with all positive Eigenvalues is maximal, if the matrix is diagonal. Any Orthonormal transformation, will naturally make the trance smaller, since you are rotating all axes away from these principle directions.

9.6.1 Finding the rotation between two sets of vectors (registration chapter?)

If we want to determine the relative rotation and shift vector Δ between two sets of N mutually corresponding vectors a_i and b_i, we may want to minimize the distance of one of the two sets rotated by a rotation matrix R, which is to be determined:

$$argmin_R \sum_{i=1}^{N} (Ra_i - \Delta - b_i)^2$$

To eliminate the problem of finding the shift, we now center source and target vectors by subtracting their mean (i.e. the weighted center of mass) of the respective vectors yielding $s_i = a_i - \sum_{n=1}^{N} a_n/N$ and $t_i = b_i - \sum_{n=1}^{N} b_n/N$. This centering simplified the

problem into only needing to find the rotation between the two datasets and minimize:

$$\begin{aligned}
&\sum_{i=1}^{N} |Rs_i - t_i|^2 \\
=&\sum_{i=1}^{N} (Rs_i - t_i)^T (Rs_i - t_i) \\
=&\sum_{i=1}^{N} \left(s_i^T R^T R s_i - \left(s_i^T R^T\right) t_i - t_i^T (Rs_i) - t_i^T t_i\right) \\
\hat{=}&\sum_{i=1}^{N} \left(-2 s_i^T R^T t_i\right) \\
\hat{=}&-\sum_{i=1}^{N}\sum_{n=1}^{N} s_{n,i} \sum_{m=1}^{N} R_{m,n} t_{m,i} \\
\hat{=}&-\sum_{n=1}^{N}\left(\sum_{m=1}^{N} R_{m,n}\left(\sum_{i=1}^{N} t_{m,i} s_{n,i}\right)_{m,n}\right) \\
\hat{=}&-tr\left(R^T H\right)
\end{aligned}$$

In step 3 we used $R^T R = I$, dropped the constant terms $s_i^T s_i$ and $t_i^T t_i$ and used for the scalar value $t_i^T R s_i = \left(t_i^T R s_i\right)^T = s_i^T R^T t_i$. The last step introduces the covariance matrix $H = \sum_{i=1}^{N} t_i s_i^T$. We now perform a singular value decomposition on H

$$H = USV^T$$

leaving us, by removing the minus sign in the minimization problem, with the task to **maximize**

$$\begin{aligned}
&tr\left(R^T USV^T\right) \\
=&tr\left(SV^T R^T U\right)
\end{aligned}$$

where we used the cyclic property of the trace. The trace of a matrix is the sum of its eigenvalues. To maximize it, it would be ideal if the term $V^T R^T U$ corresponds to the identity. We can achieve this by chosing $R^T = VU^T$thus yielding the rotation matrix

$$R = UV^T$$

It turns out that there are some possible flips of vectors depending how the Eigenvectors are calculated. To avoid potential problems, we can negating the last vector entry in the Matrix V^Tin case that $det\,(U)\,det\,(V)$ is negative.

We obtain the shift Δ as difference between the centering terms but accouting for the rotation which was already determined:$\Delta = \left(\sum_{n=1}^{N} t'_n/N - \sum_{n=1}^{N} Rs'_n/N\right)$.

This method of finding the shift and rotation (also called rigid alignment) between two corresponding sets of (landmark) vectors is known as the Kabsch[2] algorithm. The code can look like this:

[2]https://en.wikipedia.org/wiki/Kabsch_algorithm

```
using LinearAlgebra, Statistics
function find_rotation_kabsch(A::Matrix, B::Matrix)
# Calculate the center of mass of each set of points
        centroid_A = mean(A, dims=2)
        centroid_B = mean(B, dims=2)
# Center the points
        A_c = A .- centroid_A
        B_c = B .- centroid_B
# Calculate the covariance matrix
        H = B_c * A_c'
# Perform SVD
        F = svd(H)
        U = F.U
        Vt = F.Vt
# Check for reflection
        if det(U)*det(Vt) < 0
                Vt[:,end] *= -1
        end
# Calculate the rotation matrix
        R = U * Vt
        return R, (centroid_B .- R*centroid_A)
end
```

Part III

Computational Microscopy

Chapter 10

Maximum a Posteriori Likelihood

In inverse modeling want to obtain a reasonable estimate of a number of parameters based on measured data. These K parameters x_k can refer to position and brightness estimation in localization microscopy, individual reconstructed voxel emission intensities in deconvolution or even discretized object scattering intensities and phases. These parameters are often obtained by maximizing the probability over measuring all N individual datapoints Y_n

$$P(x|Y) = \prod_{n=1}^{N} P_n(x|Y_n)$$

Here P refers to the overall probability of a vector x of parameters x_k to estimate, given (represented by the vertical line |) a set of measurements Y. The left hand side P is also called the "posterior". On the right hand side the P_n refer to the individual probabilities with respect to a single measured datapoint. According to Bayes theorem

$$P(x|Y_n)P(Y_n) = P(Y_n|x)P(x)$$

we can write the posterior as a product of individual likelihoods $P(Y_n|x)$ as

$$P(x|Y) = P(x) \prod_{n=1}^{N} \frac{P(Y_n|x)}{P(Y_n)}. \tag{10.1}$$

We aim to obtain the best estimate of the parameters by maximizing $P(x|Y_n)$. The probability of the parameters to be correct (without considering the data) on their own $P(x)$ is called the "prior". As we are not really interested in the exact value of this posterior (the "maximum a posteriori likelihood") but rather the value of each of the parameters x_k we can apply any monotonic positive transform to the posterior, which will alter its value but the position of the maximum will be unaltered. In this context it is very useful to apply the logarithm as a strictly monotonic function in the positive half space:

$$argmax_x \{P(x|Y)\} = argmax_x \{\log(P(x|Y))\} = argmin_x \{-\log(P(x|Y))\}$$

. The latter term to minimize, which also yields the same optimized parameters is called the negative log a posteriori likelihood $\mathcal{L}$ and can be written according to eqn. 10.1 as

$$\mathcal{L} = -\log\left(P(x|Y)\right) = -\sum_{n=1}^{N} \log\left(P(Y_n|x)\right) + \sum_{n=1}^{N} \log\left(P(Y_n)\right) - \log\left(P(x)\right)$$

. Since we are only interested in the minimum position we can omit the constant data term $\sum_{n=1}^{N} \log\left(P(Y_n)\right)$. It is an additive constant not affecting our minimization task.

$$\mathcal{L} = -\sum_{n=1}^{N} \log\left(P(Y_n|x)\right) - \log\left(P(x)\right) \tag{10.2}$$

We refer to the first term as the posterior or the data-loss

$$\mathcal{L}_{\mathcal{D}} = -\sum_{n=1}^{N} \log\left(P(Y_n|x)\right) \tag{10.3}$$

and to the second as the prior or regularization-loss $\mathcal{L}_{\mathcal{R}} = -\log\left(P(x)\right)$. Thus we need to optimize the first term which depends on the forward model, a noise model and the measured data as well as consider our prior knowledge about the probability of possible parameter vectors x, which is described by the latter term in eq. 10.2, also called the regularizer. See section 11.5 for a detailed list of useful regularizers to be applied to images.

10.1 Forward Model

The forward model predicts the most important parameters which are then used in the noise model to predict probabilities of measurements. Typically the forward model f only predicts the expectancy μ_n of each datapoint.

$$\mu_n = f_n(x)$$

E.g. if the parameter vector describes the emission intensities of fluorophores in a widefield microscope, the vector of predicted image intensities μ is obtained via a convolution matrix H describing the convolution with the point spread function:

$$\mu_n = \sum_{k=1}^{K} H_{nk} x_k.$$

This is a linear model, but the forward model f can in general also be non-linear or arbitrarily complicated. You can find more detailes on various forward model in sections 11.0.1, 9.1 and 9.2 and others. Returning to the general case, we now decouple the overall model predicting the probabilities of our measurements $P(Y_n|x)$ from the foward and noise model, by only assuming that our forward model predicts expectancies $\mu_n = f_n(x)$ and that those are sufficient to describe the probabilities of our measurements:

$$\mathcal{L} = -\sum_{n=1}^{N} \log\left(P(Y_n|f(x))\right) - \log\left(P(x)\right) \tag{10.4}$$

Note that the probability P slightly changed its meaning by now depending on the predicted expectancy μ_n rather than the model parameters x_n. Aiming to minimize $\mathcal{L}$ it is often useful to know the gradient $(\nabla_x L)|_k = \frac{\partial \mathcal{L}}{\partial x_k}$ which is

$$\frac{\partial \mathcal{L}}{\partial x_k} = -\sum_{n=1}^{N} \left(\frac{\partial \log\left(P(Y_n|\mu_n)\right)}{\partial \mu_n} \right) \frac{\partial f_n(x)}{\partial x_k} - \frac{\partial \log\left(P(x)\right)}{\partial x_k}. \tag{10.5}$$

We can now use these general equations and look at specific noise models. The first term under the sum, we call the generalized residual, the second term refers to the adjunct operator of the forward model. The right hand term is the gradient of the prior. Note that this term neither depends on the forward model nor the noise model.

10.2 Noise Model

We now look at a few specific noise models to be used for eq. 10.4 and its gradient eq. 10.5.

10.2.1 Gaussian Noise

As a first example we assume that the measurement is described well by Gaussian noise of an a-priory known standard deviation σ_n, which can, however, vary between individual measurements. The probability density is then described by

$$P(Y_n|\mu_n) = \frac{1}{\sqrt{2\pi\sigma_n^2}} e^{-\frac{(Y_n-\mu_n)^2}{2\sigma_n^2}}$$

. By inserting this into eq. 10.2, we obtain the following Gaussian data-loss:

$$\begin{aligned}
\mathcal{L}_{Gauss} &= -\sum_{n=1}^{N} \log\left(P(Y_n|x)\right) \\
&= \sum_{n=1}^{N} \frac{\left(Y_n - \mu_n\right)^2}{2\sigma_n^2} + \sum_{n=1}^{N} \log\left(2\pi\sigma_n^2\right) \\
&\widehat{=} \sum_{n=1}^{N} \frac{\left(Y_n - \mu_n\right)^2}{\sigma_n^2}
\end{aligned}$$

, where in the last step the constant offset as given by $\sum_{n=1}^{N} \log\left(2\pi\sigma_n^2\right)$ as well as the division by 2 were dropped, since only the minimum position of the negative log likelihood matters. We thus obtained the well known weighted sum-of-squares loss, which for the case constant noise (i.e. $\sigma_n = \text{const}$) simplifies to as simple $\mathcal{L}_2^2$ applied to the difference between measurement and prediction which we strive to minimize:

$$\mathcal{L}_G = \sum_{n=1}^{N} \left(Y_n - \mu_n\right)^2$$

. For conveniance we can also state the corresponding generalized residual for Gaussian noise (see eq.)

$$\mathcal{R}_{Gauss} = \frac{\partial \log\left(P(Y_n|\mu_n)\right)}{\partial \mu_n} = 2\left(\mu_n - Y_n\right)$$

. Note that it turn zero when the predicted expectancy μ_n exactly matches the measured data Y_n.

10.2.2 Poisson Noise

In optics we often measure light in random superposition of coherent states. Assuming detectors which might have a non-ideal quantum efficiency but do not add addition noise, this leads to a Poisson distribution describing the measured ligth (see section 9.3.1):

$$P(Y_n|\mu_n) = \frac{\mu_n^{Y_n}}{Y_n!} e^{-\mu_n}$$

. inserting this into eq. 10.2, we obtain the following Poisson data-loss:

$$\begin{aligned}
\mathcal{L}_{Poisson} &= -\sum_{n=1}^{N} \log\left(P(Y_n|x)\right) \\
&= \sum_{n=1}^{N} \left(\mu_n - Y_n \log(\mu_n) + \log(Y_n!)\right) \\
&\widehat{=} \sum_{n=1}^{N} \left(\mu_n - Y_n \log(\mu_n)\right)
\end{aligned}$$

, where we also dropped the constant term $\log(Y_n!)$ in the last step. It bears a similarity to the Kullback-Leibler divergence used for comparing distributions in statistics. The corresponding generalized residual for Poisson noise becomes

$$\mathcal{R}_{Poisson} = \left(1 - \frac{Y_n}{\mu_n}\right)$$

. Note that also this becomes zero if the predicted expectancy exactly matches the data.

10.2.3 Additional Detection Noise

Often additional detection noise cannot be neglected and should therefore be part of the noise model. Such background noise with standard deviation σ_D is often of Gaussian statistics, but its mixture with the statistics of photon detection would lead to a convolution of the stochastic distribution, making the exact treatment rather involved. Therefore we present here three approximations that can account for a constant amount of detection noise.

10.2.3.1 Poisson noise with background

As a first approximation, we treat the detection noise as an additional Poisson noise. This leads to adding its constant variance to both, the data and the predicted expectancy. Omitting constant terms, we obtain

$$\mathcal{L}_{Poisson} = \sum_{n=1}^{N} \left(\mu_n + \left(Y_n + \sigma_D^2\right) \log(\mu_n + \sigma_D^2)\right)$$

and the generalized residual

$$\mathcal{R}_{Poisson} = \left(1 - \frac{Y_n + \sigma_D^2}{\mu_n + \sigma_D^2}\right)$$

10.2.3.2 Scaled Gaussian Noise

Another approach is to keep the Gaussian nature of the constant noise and approximate the Poisson distribution as a Gaussian, but keeping its scaling property such the the variance σ_n^2 of the Gaussian is described by the predicted expectancy of arriving photons and the detection noise $\sigma_n^2 = \sigma_D^2 + \mu_n$.

$$P_{ScaledGauss}(Y_n|\mu_n) = \frac{1}{\sqrt{2\pi\left(\sigma_D^2 + \mu_n\right)}} e^{-\frac{(Y_n - \mu_n)^2}{2\left(\sigma_D^2 + \mu_n\right)}}$$

. We then obtain the following data loss:

$$\begin{aligned}\mathcal{L}_{ScaledGauss} &= \sum_{n=1}^{N} \left(\frac{\left(Y_n - \mu_n\right)^2}{2\left(\sigma_D^2 + \mu_n\right)} + \frac{1}{2}\log\left(2\pi\left(\sigma_D^2 + \mu_n\right)\right)\right) \\ &\widehat{=} \sum_{n=1}^{N} \left(\frac{\left(Y_n - \mu_n\right)^2}{\left(\sigma_D^2 + \mu_n\right)} + \log\left(\sigma_D^2 + \mu_n\right)\right)\end{aligned}$$

and the corresponding generalized residual

$$\mathcal{R}_{ScaledGauss} = \frac{2\left(\mu_n - Y_n\right) + 1}{\left(\sigma_D^2 + \mu_n\right)} - \frac{\left(Y_n - \mu_n\right)^2}{\left(\sigma_D^2 + \mu_n\right)^2}$$

. As we see this generalized residual does not yield zero for the case of $\mu_n = Y_n$. This can be mitigated by normalizing the Gaussian with a normalization factor that can depend on the measurements Y_n:

$$P_{SG}(Y_n|\mu_n) = \frac{1}{\sqrt{2\pi Y_n + 1}} e^{-\frac{(Y_n - \mu_n)^2}{2\left(\sigma_D^2 + \mu_n\right)}}$$

. We then obtain the following data loss, omitting the constant normalization terms:

$$\mathcal{L}_{SG} = \sum_{n=1}^{N} \frac{\left(Y_n - \mu_n\right)^2}{2\left(\sigma_D^2 + \mu_n\right)}$$

and the corresponding generalized residual

$$\mathcal{R}_{SG} = \frac{(\mu_n - Y_n)}{(\sigma_D^2 + \mu_n)} - \frac{(Y_n - \mu_n)^2}{2(\sigma_D^2 + \mu_n)^2}$$

, which now has the property of being zero upon perfect prediction. For $\sigma_D^2 = 0$ we obtain

$$R_{SG} = \frac{1 - Y_n^2/\mu_n^2}{2}$$

.

10.2.3.3 Anscombe Transformation on Data and Expectancy

As a final approach we apply a variance-stabilizing transformation $x \to 2\sqrt{x+c}$, known as the generalized Anscombe transformation to both, the measured data and the predicted expectancy and model the resulting probability distribution as a Gaussian with unit variance:

$$P(Y_n|\mu_n) = \frac{1}{\sqrt{2\pi(Y_n + c)}} e^{-\frac{(2\sqrt{Y_n+c} - 2\sqrt{\mu_n+c})^2}{2}}$$

Note that the $\sqrt{Y_n + c}$ comes from the need to scale the probability with the inverse Jakobian to warrant a sum of one. Howeverm this term is constant and can therefore be dropped in the negative log likelihood, which yields the Anscombe loss

$$\mathcal{L}_{Anscombe} = \sum_{n=1}^{N} \left(\sqrt{Y_n + c} - \sqrt{\mu_n + c}\right)^2$$

. The corresponding generalized residual is

$$\begin{aligned}\mathcal{R}_{Anscombe} &= 2\left(\sqrt{Y_n + c} - \sqrt{\mu_n + c}\right)\frac{-1}{2\sqrt{\mu_n + c}} \\ &= \left(1 - \frac{\sqrt{Y_n + c}}{\sqrt{\mu_n + c}}\right)\end{aligned}$$

. Note the similarity to the generalized residual of the Poisson distribution. Upon perfect prediction ($Y_n = \mu_n$) also this generized residual becomes zero. Note also that for a model which predicts the intensity from a complex-valued amplitude, $f(x) = \mu_n = xx^*$, we obtain the Wirtinger derivative $\frac{\partial f}{\partial x^*} = x$, leading to a simple gradient descent update scheme for $c = 0$ of

$$\begin{aligned}x^{(n+1)} &= x^{(n)} - \left(1 - \frac{\sqrt{Y_n}}{\sqrt{\mu_n}}\right)x^{(n)} \\ &= \frac{\sqrt{Y_n}}{\left|x^{(n)}\right|}x^{(n)}\end{aligned}$$

, which means that the magnitude is replaced by the measured magnitude, but the phase of the complex-valued quantity x is kept. This shows that the well known Gerchberg-Saxton update scheme can be interpreted as applying a gradient descent update using

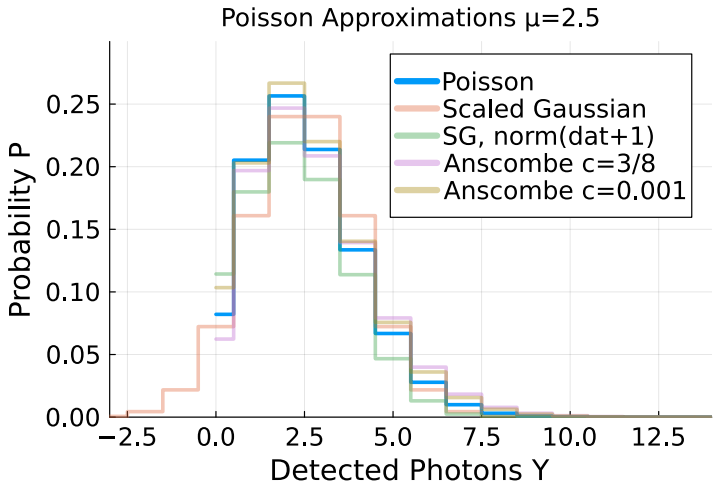


Figure 10.1: Comparison of Poisson approximations for the expectancy of $\mu = 2.5$. The SG method corresponds to a scaled Gaussian

the Maximum Likelihood updaten with the generalized Anscomb transform ($c = 0$) noise model approximating the Poisson noise.

A slightly better choice of $c = \frac{3}{8}$ leads to a better stabilization of the variance. If we want to include an additional Gaussian noise term, we can add the corresponding variance: $c = \frac{3}{8} + \sigma_D^2$.

The various noise models approximate Poisson noise in different ways (fig. 10.1), which we will now compare in terms of an introduced bias.

10.2.4 Bias of the Noise Models

To see if these noise models are leading to an estimation bias when applied to pure Poisson-distributed data we consider a model of N pixel all used to estimate a single value of μ. Our forward model is therefor

$$\mu_n(x) = f_n(x) = x$$

with only a single value of x being the model parameter predicting the same expectancy for each pixel. To obtain the maximum likelihood estimate we have to minimize $\mathcal{L}$ which, since $\partial f / \partial x = 1$ amounts to setting the sum of generalized residuals to zero

$$0 = \sum_{n=1}^{N} \mathcal{R}$$

. Applying this to the generalized residual of the Poisson noise model yields

$$0 = \sum_{n=1}^{N} \mathcal{R}_{Poisson} = \sum_{n=1}^{N} \left(1 - \frac{Y_n}{\mu_p}\right)$$

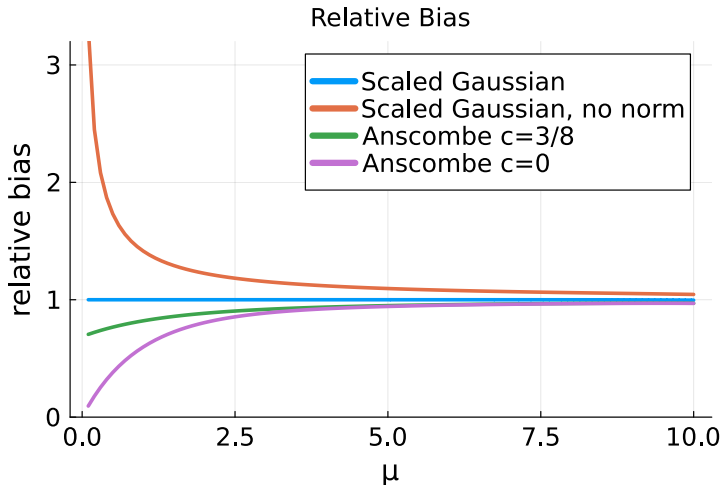


Figure 10.2: Statistical bias in relation to the ground truth applying the corresponding estimators with Poisson distributed data to estimate a common μ. The Poisson distribution (not shown) is also free of any estimation bias. Surprisingly the scaled Gaussian with the normalization based on the μ to estimate is asymptotically free of any bias, even though an estimate based on a single point is not equal to that point.

, which results into the ML estimator $\mu_p = \sum_{n=1}^{N} Y_n/N$. This estimator is just averaging all measured photon numbers, which is unbiased and optimal. We can now apply the same procedure to our other noise models. The Anscombe transform noise model yields $\mu_{Anscomb} = \left(\sum_{n=1}^{N} \sqrt{Y_n + c}/N\right)^2 - c$, the scaled Gaussian noise model with the normalization based on the expectancy yields $\mu_{ScaledGaussian} = \sqrt{\sum_{n=1}^{N} Y_n^2/N + 1/4} - 1/2$ and the scaled Gaussian noise model with the normalization based on the measured data yields $\mu_{SG} = \sqrt{\left(\sum_{n=1}^{N} Y_n^2/N\right)}$. By weighting the measurements with the Poisson distribution, we can calculate the relative bias for each true value of μ as shown in fig. 10.2. It is observed that all approximations except the scaled Gaussian with its normalization based on the predicted expectancy μ yield a significant bias for $\mu < 5$. Of course the estimator (not shown) based on Poisson statistics is also free of any bias.

Chapter 11

Deconvolution

Deconvolution refers to a subclass of inverse problems which involve convolutions in their forward model. Such forward problems could generally involve coherent, partially coherent or incoherent imaging systems. More generally deconvolution is also applied for spectral, time-dependent or in general data where the measurements system includes a convolution with an instrument response function. In this chapter on deconvolution, we assume that the imaging system is fully incoherent (section 9.2). The most noticable effect of deconvolution is an improvement of contrast and visibility. This is particularly prominent in widefield fluorescence image stacks. Fig. 11.1 shows a practical example. As can be seen, the deconvolution removes all the out-of-focus haze and also sharpens the borders of the vesicles significantly such that they are seen as hollow structures. Also confocal sereial sections looks significantly sharper after deconvolution such that the granal membranes of the chloroplasts become visible.

11.0.1 The Forward Model

For fully incoherent imageing the image can be modeled as

$$I(x,y) = \rho(x,y,z) \circledast_{3D} h(x,y,z) \tag{11.1}$$

More generally speaking, in the framework of inverse modeling, this is a **forward model** which describes how to get from the unknown object $\rho(x,y,z)$, parameterized by a set of parameters to a prediction of the measured intensities $I(x,y)$. The incoherent point spread function $h(x,y,z)$ serves as a blurring kernel, which is unfortunately, due to the extend of the McCutchen pupil (see section 9.1) strictly band-limited in optics. The task of deconvolution is to invert this problem, i.e. to determine the best possible set of parameters ρ_n given the measured data I_n. Due to the band limit mentioned above, this problem is an under-determined problem, which is typically called an **ill-posed** problem in the context of inverse modeling. To this aim, we discretize the problem as

$$I_n = \sum_{i=1}^{N} H_{n,i}\rho_i \tag{11.2}$$

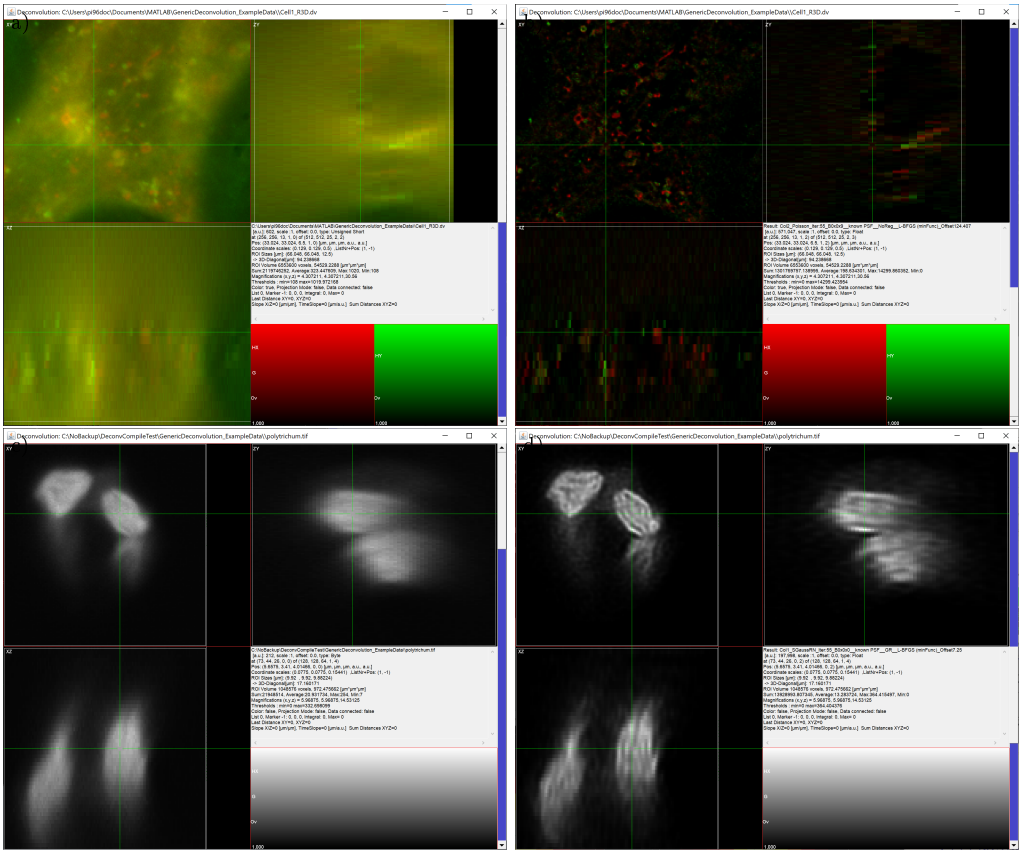


Figure 11.1: Example of a widefield fluorescence dataset, raw and deconvolved. a) On the left you see orthogonal sections of a focus series (512×512×25 pixels, 66µm×66µm×12µm) in two colors. Shown are living MDCK cells transfected with Rab5-GFP (early endosome) and Rab7-mCherry (late endosome) markers acquired on a deltavision widefield microscope. b) shows the deconvolved data (55 iterations Poisson noise model, L-BFGS, positively constrained, theoretical PSF, no regularization, 0×0×9 guard pixels). Data courtesy of Xian Hu (Edna). c) confocal data of a moss spore (polytrichum commune) near infrared emission, autofluorescence of chlorophyll (128×128×64 pixels, 10µm × 10µm × 10µm). d) deconvolution with an experimentally acquired PSF, 55 iterations, Good's roughness prior $\lambda = 0.001$.

, where we indexed the three-dimensional pixel positions $(x_n, y_n, z_n)$by a single index (n or i) and generalized the three-dimensional discrete convolution $\circledast_{3D}$ as a linear mapping encapsulated into the multiplication with the matrix H_{ni}. Note that this matrix would be enormously big and computations this way could be extremely slow or even too big to perform. Yet, for now the matrix H only serves as a placeholder denoting the convolution operation. Practically the matrix multiplication with H is implemented via the discrete fast Fourier transform multiplication in Fourier space but it is very useful to derive the algorithms below by using this generalized linear mapping approach.

Of course what we measure is not really $I_n$but rather a noisy version of this perfect forward model

$$y_n = I_n + \eta_n = \eta_n + \sum_{i=1}^{N} H_{n,i}\rho_i \tag{11.3}$$

Written in this form, the noise η does not need to be Gaussian noise. In photonics the noise is typically given by measured values drawn from a Poisson distribution. The noise scales with the expectancy value. In a Poisson distribution, the variance is equal to the mean, leading to the term "multiplicative" noise. Yet, we still write even this noise in an additive form in the equation above. Thus the values of η_n are drawn from a shifted Poisson distribution with zero expectancy value at each pixel, yet with a changing variance. Writing the noise in this additive format helps to keep track of the perfect signal and the influence of the noise.

11.1 Direct Inversion

As we want to recover the object, the source intensity distribution, ρ_i from the measured data y_n, it seems that we only need to invert Eq. 11.2:

$$\rho = H^{-1}y$$

yet, a closer look reveals that H is rank deficient. To see this, we can write the forward model by remembering Eq. 11.1 which can be written in Fourier space as.

$$\mathcal{F}(I) = \mathcal{F}(\rho)\,\mathcal{F}(h) \tag{11.4}$$

with $\mathcal{F}(h)$ being the optical transfer function. As the transfer function is band-limited, the inversion

$$\mathcal{F}(\rho) = \frac{\mathcal{F}(I)}{\mathcal{F}(h)}$$

with the division being a point-wise (Hadamard-) division in Fourier space, leads to the problem that we would need to divide outside the band limit of $\mathcal{F}(h)$. This can be avoided by performing this division only in places where $|\mathcal{F}(h)| > \epsilon$ with a small value of ϵ near the numerical precision limit. As seen in Figs. 11.2a,b, this indeed yields good results for simulated noise-free data, whereas applying this technique to noisy data, as typically acquired in a fluorescence microscope yields unacceptable results as seen in Fig. 11.2c.

The wave-like artifacts crossing and dominating the image stem from a few noisy pixels in Fourier space that are close to the band limit and thus divided by a very small number

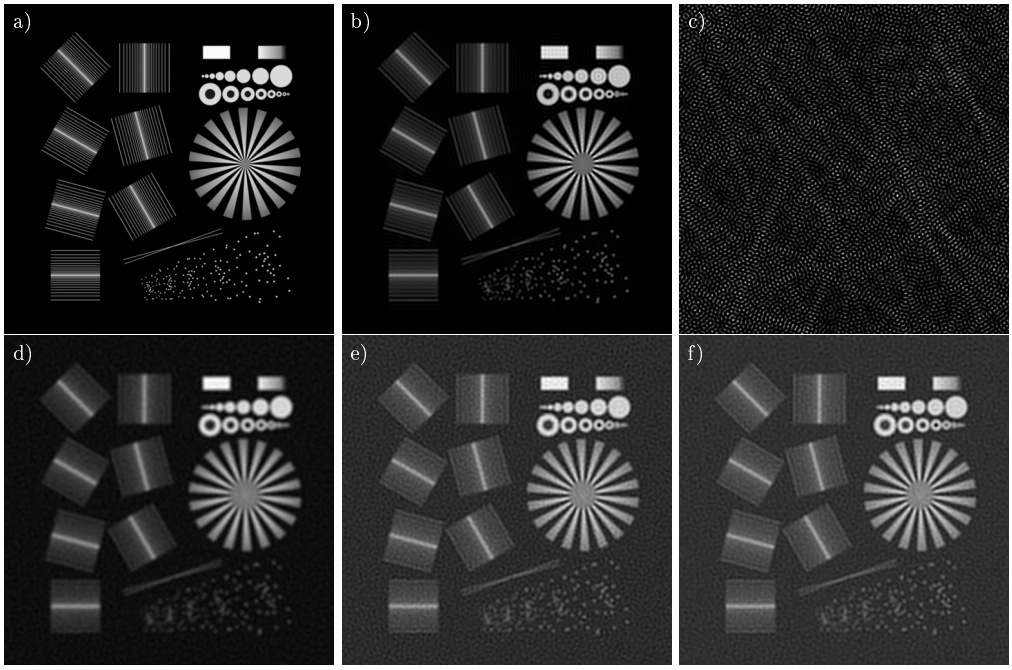


Figure 11.2: Direct deconvolution and Wiener filtering of data with Gaussian noise (variance 1000) of an image with a value of 1000 in the maximum pixel. (a) Ground truth object. (b) Division in Fourier space on the OTF support ($> 10^{-6}$) of noise-free data. (c) The same division applied to noisy data. Note that negative values in b) to c) are not shown here. d) Generalized Wiener filtering with the simulated sigma [mean square distance, msd: 10065], e) Generalized Wiener filter with a σ reduced by a factor of 3 msd: 7988] and f) using the correct object spectrum [msd: 7749]

close to ϵ, i.e. multiplied by a very large number. The noise term as described by Eq. 11.3 is also amplified:

$$\mathcal{F}(x') = \left.\frac{\mathcal{F}(I)}{\mathcal{F}(h)}\right|_{|\mathcal{F}(h)|>\epsilon} - \left.\frac{\mathcal{F}(\eta)}{\mathcal{F}(h)}\right|_{|\mathcal{F}(h)|>\epsilon}$$

with x' denoting the reconstructed estimate of ρ. One strategy to avoid this noise amplification problem would be to rise ϵ to a level on the order of the standard deviation of the noise in Fourier space, but better results are obtained by the approaches outlined below.

11.2 Generalized Wiener Filtering

As seen above, we need some sort of regularization to balance the influence of signal and noise in our attempt to invert the forward model. Due to the convolution in the forward model, Fourier space forms a natural basis, which means that the values of the transfer function represent the eigenvalues to which a smooth division has to be applied to. This is achieved by the generalized Wiener filtering:

$$\mathcal{F}(x') = \frac{\mathcal{F}(I)\,\mathcal{F}^*(h)}{|\mathcal{F}(h)|^2 + \lambda}$$

with the regularization constant λ being chosen by trial and error until a visually pleasing result is obtained. This leaves us with the unsatisfying feeling of a lack of insight of how to choose the constant λ. Therefore the next approach will bring us a step closer to a statistics-based approach to solving this inversion problem.

11.3 Wiener Filtering

As we now saw that we need a more rigorous treatment of the statistical inversion problem in deconvolution, we start with some simple assumptions. The noise is assumed to be of Gaussian distribution and independent of the pixel position. Now our goal is not any longer to obtain a perfect inversion of the forward problem but rather to obtain an estimate reconstruction filter kernel w', which upon filtering the measured data, gets as close as possible to the object in a statistical sense. Here we formulate the notion of "as close as possible" by a sum of squares difference. We can thus formulate the following minimization problem:

$$w' = \underset{w}{\operatorname{argmin}} |y \circledast w - \rho|_2^2$$

As the convolution operation is a simple multiplication in Fourier space, we can exploit Parseval's theorem and realize that the sum of squares can also be computed in Fourier space, with only a constant factor, depending on the precise definition of the Fourier transform. For finding a minimum such factors can be omitted. Inserting the forward

model including noise (Eq. 11.3) for y, we obtain in Fourier space

$$\begin{aligned}\widetilde{x'} &= \underset{\widetilde{x}}{\operatorname{argmin}} \left| \left(\tilde{\rho}\tilde{h} + \widetilde{\eta}\right) \tilde{w} - \tilde{\rho} \right|_2^2 \\ &= \underset{\widetilde{x}}{\operatorname{argmin}} \sum \left(\tilde{w}\tilde{\rho}\tilde{h} + \tilde{w}\widetilde{\eta} - \tilde{\rho}\right) \left(\tilde{w}\tilde{\rho}\tilde{h} + \tilde{w}\widetilde{\eta} - \tilde{\rho}\right)^* \\ &= \underset{\widetilde{x}}{\operatorname{argmin}} \sum \left(\tilde{w}\tilde{w}^* \left|\tilde{\rho}\right|^2 \left|\tilde{h}\right|^2 - \tilde{w}\tilde{h} \left|\tilde{\rho}\right|^2 - \tilde{w}^*\tilde{h}^* \left|\tilde{\rho}\right|^2 + \tilde{w}\tilde{w}^* \left|\widetilde{\eta}\right|^2 + \left|\tilde{\rho}\right|^2\right)\end{aligned}$$

There the tilde ˜ indicates a variable in Fourier space to keep the notation short. In the last step we exploited that all sums over products of a single noise terms with another value will evaluate in the statistical sense to zero. The Fourier transform of independent Gaussian noise of zero mean can be proven to be independent Gaussian noise of zero mean. Thus a sum over such a term will on average, i.e. if we perform many reconstructions, amount to zero.

To solve this equation, we need to find the extremum with respect to a complex variable x, which brings to mind Wirtinger derivatives (section **??**). The sum of terms in Fourier space will be minimal if each of the terms is minimal. We thus obtain at each spatial frequency:

$$\begin{aligned}&\frac{\partial \left(\tilde{w}^* \left(\tilde{w} \left|\tilde{\rho}\right|^2 \left|\tilde{h}\right|^2 - \tilde{h}^* \left|\tilde{\rho}\right|^2 + \tilde{w} \left|\widetilde{\eta}\right|^2\right) - \tilde{w}\tilde{h} \left|\tilde{\rho}\right|^2 + \left|\tilde{\rho}\right|^2\right)}{\partial \widetilde{x}^*} \\ &\qquad = \tilde{w} \left|\tilde{\rho}\right|^2 \left|\tilde{h}\right|^2 - \tilde{h}^* \left|\tilde{\rho}\right|^2 + \tilde{w} \left|\widetilde{\eta}\right|^2 \overset{!}{=} 0\end{aligned}$$

, which can be solved for the Fourier space weights of the deconvolution operation $\tilde{w}$

$$\tilde{w} = \frac{\tilde{h}^* \left|\tilde{\rho}\right|^2}{\left|\tilde{\rho}\right|^2 \left|\tilde{h}\right|^2 + \left|\widetilde{\eta}\right|^2} = \frac{\tilde{h}^*}{\left|\tilde{h}\right|^2 + \frac{\left|\widetilde{\eta}\right|^2}{\left|\tilde{\rho}\right|^2}}$$

The operation of multiplying in Fourier space by $\tilde{w}$ is called Wiener filtering. But did we really solve the problem? Of course not, since the weights include two unknown terms, $\left|\widetilde{\eta}\right|^2$and $\left|\tilde{\rho}\right|^2$. The latter term depends on the very thing we are trying to estimate. The noise term can be obtained via calibration measurement or other methods that estimate the noise from the data [**?**] but in Wiener filtering we have to make assumptions on the power spectrum of the object $\left|\tilde{\rho}\right|^2$. The typical assumption is to assume this spectrum being a constant equal to the sum S of measured intensity. Note that we then obtain a result very similar to generalized Wiener filtering but with the advantage of knowing which ϵ to use: $\epsilon = \frac{\left|\widetilde{\eta}\right|^2}{S^2}$. For typical real world objects the power spectrum often decays as $\frac{1}{|k|^p}$with p being related to the Hausdorff dimension. So we could base our Wiener filter also on such an assumption. However, as seen in Fig. 11.2d-f, the differences between Wiener filtering with assuming a constant power spectrum or the ground truth $\left|\tilde{\rho}\right|^2$, which is only known since this is simulated data, are rather small, especially, if we allow to lower the value of sigma as done in Fig. 11.2e. In the caption of Fig. 11.2, the mean square distance, i.e. the distance of the reconstruction to the ground_truth is given, supporting the visual impression.

Recapitulating Wiener filtering, we can see that is has the advantage of being a linear operation preserving the band limit of the original optical transfer function. So we are not "guessing" frequencies beyond what we measured. Since Wiener filtering is a non-iterative procedure, requiring only two Fourier transforms and a multiplication, it is very fast to apply to deconvolution problems. However the underlying assumption of the noise being of equal strength, independent of the pixel position is often not valid in optical imaging. High quality cameras have negligible read noise and the measured signal is dominated by the measured photons with a variance in each pixel being equal to the mean, apart for a global gain factor. This means that bright parts of a measured image contain significantly more noise than dim parts. This may seem counter intuitive, but the signal to noise ration (SNR) is still better for bright parts than for dim parts, since the noise being defined as the standard deviation grows with the square-root of the signal. Thus Wiener filtering is particularly suitable for data with a very small dynamic range. I.e. widefield fluorescence images of larger objects, which are usually dominated by a high background and also have a fairly good SNR. For dim and crisp images, as for example obtained by confocal laser scanning microscopy of points or filaments, the iterative deconvolution methods outlined below achieve significantly better results, also because they can exploit prior knowledge such as the object emission being a positive quantity.

11.4 Maximum Likelihood deconvolution

As seen in the chapter on maximum likelihood (section 3.11) we can formulate the deconvolution as a maximum likelihood problem which can then be solved, typically be an iterative optimization algorithm. For a vector x of parameters to estimate we repeat the basic equation (section 3.11, Eq. 10.3) here for the case of Gaussian and Poisson noise negative log likelihood $\mathcal{L}$ and the Anscombe approximation applied to the model and the data, with so far unspecified forward model $f(x)$:

$$\begin{aligned} \mathcal{L}_{\text{Gauss}} &= \sum_n |f_n(x) - y_n|^2 \\ \mathcal{L}_{\text{Poisson}} &= \sum_n (y_n - y_n log f_n(x)) \\ \mathcal{L}_{\text{Anscombe}} &= \sum_n \left| \sqrt{y_n + c} - \sqrt{f_n(x) + c} \right|^2 \end{aligned} \tag{11.5}$$

, where c is a constant which can be assumed to be zero or $\frac{3}{8}$. See section for a discussion of how c affects the solution. The gradients with respect to the real-valued vector x being

$$\begin{aligned} \frac{\partial \mathcal{L}_{\text{Gauss}}}{\partial x_k} &= \sum_n 2\left(f_n(x) - y_n\right) \frac{\partial f_n(x)}{\partial x_k} \\ \frac{\partial \mathcal{L}_{\text{Poisson}}}{\partial x_k} &= \sum_n \left(1 - \frac{y_n}{f_n(x)}\right) \frac{\partial f_n(x)}{\partial x_k} \\ \partial \mathcal{L}_{\text{Anscombe}} &= \sum_n \left(1 - \sqrt{\frac{y_n + c}{f_n(x) + c}}\right) \frac{\partial f_n(x)}{\partial x_k} \end{aligned} \tag{11.6}$$

We now need to insert the details of our model $f_n(x_m) = \sum_m H_{n,m} x_m$, here written in the general form of a linear operator. The gradient thus yields

$$\frac{\partial f_n(x)}{\partial x_k} = \sum_m H_{n,m} \frac{\partial x_m}{\partial x_k} = \sum_m H_{n,m} \delta_{m,k} = H_{n,k} \tag{11.7}$$

Reinserting it into the gradients above and assuming $c = \frac{3}{8}$, we obtain

$$\frac{\partial \mathcal{L}_{\text{Gauss}}}{\partial x_k} = \sum_n 2 \left(f_n(x) - y_n \right) H_{k,n} \tag{11.8}$$

$$\frac{\partial \mathcal{L}_{\text{Poisson}}}{\partial x_k} = \sum_n \left(1 - \frac{y_n}{f_n(x)} \right) H_{n,k} \tag{11.9}$$

$$\partial \mathcal{L}_{\text{Anscombe}} = \sum_n \left(1 - \sqrt{\frac{y_n + \frac{3}{8}}{f_n(x) + \frac{3}{8}}} \right) H_{n,k} \tag{11.10}$$

Note that Eq. 11.8 sums over the second index, whereas the convolution operation as used in 11.7 was summing over the first index of $H_{n,m}$.We thus need to transpose the matrix, which turns out to be another convolution operation, but with the PSF mirrored at the zero position of space: $\circledast h(-r)$ with the spatial coordinate r.

11.4.1 Gradient-based optimization

We can now minimize the negative log likelihood $\mathcal{L}$ by one of the optimization routines as discussed in section 3.11. For a simulated image assuming Gaussian noise (Fig. 11.3a) of a test object (Fig. 11.2a) with a variance of 1000 (i.e. 1000 expected photons per pixel), we obtain after 200 iterations with the steepest decent algorithm assuming Gaussian statistics, the result shown in Fig. 11.3b. In Fig. 11.3c, you see the algorithm which assumes Gaussian noise applied to a simulation using Poisson noise with 1000 expected photons in the maximum pixel.

Positivity constraints

If we try to apply the Poisson gradient of Eq. 11.8 we run into the problem, that negative numbers can arise for the object and thus for the forward model, which violates the assumptions made for Poisson statistics. This needs to be explicitly prevented.

Such a positivity constraint can be introduced in various ways. The simplest way is to restrict the algorithm explicitly from reaching negative numbers. Practically it is also very useful to prevent predictions very close to zero, since even measuring a non-zero value for a prediction of zero would lead to an infinite gradient and for the case of zero prediction and zero measurement we would obtain a division of zero by zero, which needs to be handles specially. Therefore it is useful to restrict the predictions to be larger than a small value ϵ. Another way to warrant positivity is achieved by modifying the forward model such that negative values for the object are prevented by definition.

$$x = x'^2 + \epsilon \tag{11.11}$$

$$\frac{\partial x}{\partial x'} = 2x' \tag{11.12}$$

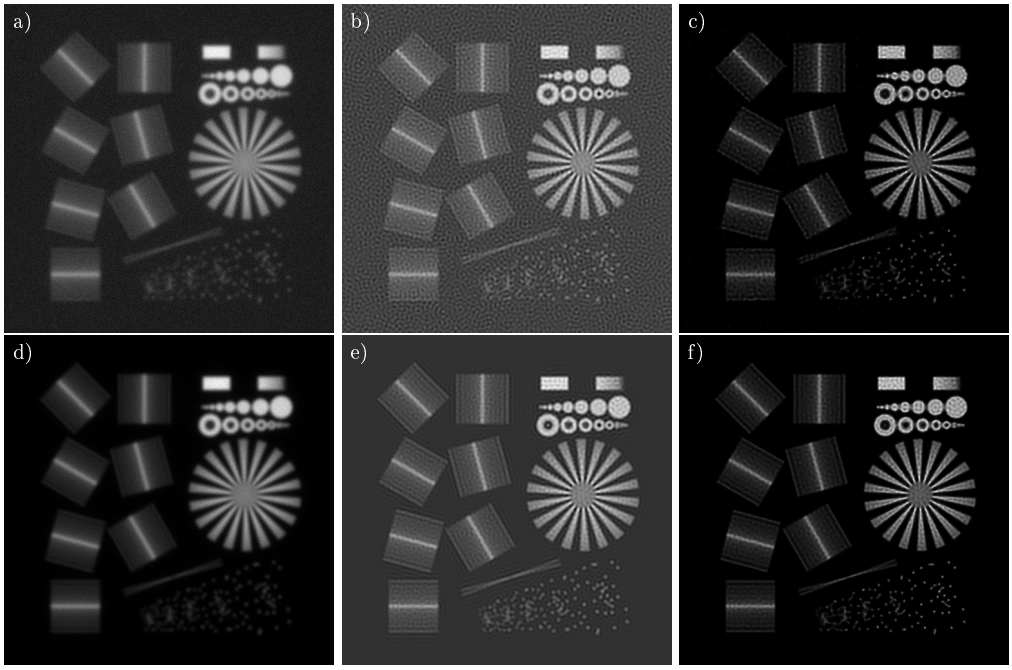


Figure 11.3: Steepest descent algorithm. (a) Simulated image with Gaussian noise of a variance of 1000 at a maximum object brightness of 1000. (b) Result of the steepest descent algorithm after 200 iterations of steplength one. (c) Result after 200 iterations with positivity constraint. Here and adpative step-size control was necessary to prevent divergence. (d-f) Corresponding measurement and reconstructions based on Poisson noise data with 1000 expected photons in the maximum pixel.

We are then optimizing the auxiliary function x' and after optimization obtain the reconstruction via $x = x'^2$. We can thus modify our gradients accordingly:

$$\frac{\partial \mathcal{L}_{\text{Gauss}}}{\partial x'_k} = 2x' \sum_n 2\left(f_n(x'^2) - y_n\right) H_{n,k} \tag{11.13}$$

$$\frac{\partial \mathcal{L}_{\text{Poisson}}}{\partial x'_k} = 2x' \sum_n \left(1 - \frac{y_n}{f_n(x'^2)}\right) H_{n,k} \tag{11.14}$$

In this way we can prevent negative values (or even very small values) from occurring. Since the mapping in not bijective, we have turned a problem which may be convex, into a clearly non-convex problem. Both values, x' and $-x'$ are clearly equivalent solutions. However this loss of convexity may be a minor issue, since both of these solutions lead to the same x and this is the reconstruction we are interested in.

Of course various other mappings exist and they can also be chosen to be bijective. One such choice could be the exponential function, mapping the set of numbers to all-positive numbers, yet in practice this turns out to be a particularly bad choice as the algorithms tend to easily get "stuck" when value are close to zero. The gradients become exceedingly small which then takes the algorithm forever to dig itself out of this x value close to zero, corresponding to a large negative value of x'. The following choice defines as piece-wise function which turns out to be better suited:

$$x = \left\{ \begin{array}{ll} 1 + x', & x' > 0 \\ \frac{1}{1-x}, & x' < 0 \end{array} \right| \tag{11.15}$$

with the gradients:

$$\frac{\partial x}{\partial x'} = \left\{ \begin{array}{ll} 1, & x' > 0 \\ \frac{1}{|1-x|^2}, & x' < 0 \end{array} \right|$$

In Fig. 11.8c,f, you can see a positively constrained iteration using the square positivity constraint (Eq. 11.11) for Gaussian and Poisson noise data.

All reconstructions were obtained by stopping the reconstructions at the optimal iteration number as judged by the maximal NCC compared to the ground truth. Interestingly there seems to be an optimal number of iterations in terms of quality, even though one may have expected that the quality improves the closer the iterations are to the minimum. The reason for this effect is that the problem is under-constrained, or "ill-posed". Thus the minimization problem over-fits the data and presents a solution, which is indeed well capable of predicting a close match to the measured images, but the reconstructed objects are far from our expectations of how a typical object should look like. We will return to this point in the section 11.5 on regularization below.

Avoiding border wrap-around and invalid pixels

Since the forward model 11.1 is implemented via FFTs the convolution leads to wrap-around problems near the borders of the array, i.e. an object emission intensity on the left leads to residual blur on the opposing side. This is particularly problematic when deconvolving widefield data, since the PSF possesses a constant integral along the XY coordinates. This leads to particularly strong border artifacts at the top or bottom. One

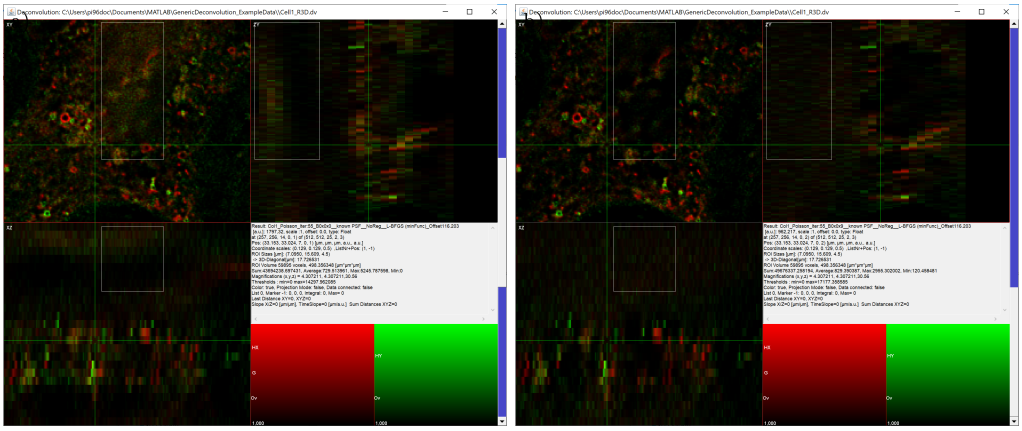


Figure 11.4: Border wrap-around problem. a) Deconvolved example without guard pixels. Note the artefactual intensities caused by wrap-around problems in the white box. b) The same dataset deconvolved with 9 guard-pixels used along Z. Note no artefacts in the XZ view. All other experimental and deconvolution parameters where as given in Fig. 11.1.

may be tempted to double the extent of the data along Z by inventing some interpolated fake data. However, this is usually not very successful and also computationally expensive. A much better approach is based on the idea of accepting missing data in the maximum likelihood framework. The forward model does not need to map from some array voxels to the same arrangement of array voxels. In fact it is much more general. This then leads to the idea to reconstruct a larger region extending the borders by a seam which is forward projected as well. However, in the comparison operation, theses guard-pixels should not be considered in the summation to determine the loss. When considering the gradient, automatic differentiation can sometimes handle such cases.

Fig. 11.4 shows a comparison of the Poisson-noise based ML-deconvolution of an experimental dataset without (left) and with (right) 9 such border-pixels applied along Z. Otherwise one can also just set these invalid pixels to the same values as the measured values which leads to a residuum of zero for those pixels before they are propagated via the adjunct operation. Similarly some data may be saturated, corrupted by high energy photons or can by other means of image processing be identified as invalid measurement data. The same approach of ignoring such pixels, marked by a validity map, can be used here to avoid corresponding reconstruction artifacts. Such validity maps can also be implemented as soft weights by considering them in the gradient calculation.

11.4.2 Acceleration

As we previously saw, a simple steepest descent algorithm following the gradient with a constant step-length of one gradient is not the best way to quickly approach the minimum. We were using the following steepest descent update scheme by simply following the gradient $\nabla\mathcal{L}$ downhill:

$$x_k^{(i+1)} = x_k^{(i)} - \frac{\partial\mathcal{L}}{\partial x_k}$$

Note that the iteration number superscript indices in brackets (i) should not be confused with powers. Thus we can speed up our algorithms significantly by choosing better schemes with adaptive step-length control or variations on the search direction, such conjugate gradient. A simple way to speed up is by introducing the over-relaxation factor α:

$$x_k^{(i+1)} = x_k^{(i)} - \alpha \frac{\partial \mathcal{L}}{\partial x_k}$$

There are several ways to determine α. One way is to keep increasing α by a factor of two without altering the gradient search direction, which is amounting to a line search, until a the loss starts increasing again [27]. But there are various other schemes.

The best compromise in terms of computational speed, so far, is achieved by algorithms that also consider previously calculated gradients and build up a low-dimensional version of the Hessian. An algorithm that is particularly recommended is the limited memory Broyden–Fletcher–Goldfarb–Shanno algorithm (L-BFGS), see [**?**]. This acceleration reduces the number of iterations needed to reach the NCC optimum from 32 to 10, see Fig. 11.5. Note that the steepest descent would also continue to optimize but only significantly slower that accelerated algorithms. Note also that LBFGS at 200 iterations already reached a result which is over-deconvonvolved due to the ill-posedness of the problem. See Section 11.5 below.

11.4.3 Fixed-point scheme (Richardson Lucy)

However, for Poisson-noise data, the fixed point iteration is also a particularly interesting scheme. Instead of providing an additive (gradient-based) update, fixed-point schemes apply a multiplicative correction:

$$x^{(i+1)} = x^{(i)} \mathrm{cor}^{(i)}$$

in which case the iteration i finishes when the correction $\mathrm{cor}^{(i)}$ reaches the value of one. To convert the Poisson gradient to a fixed point scheme, we need to subtract it from one:

$$\mathrm{cor}^{(i)} = 1 - \alpha \sum_n \left(1 - \frac{y_n}{f_n(x^{(i)})}\right) H_{k,n}$$

If we normalize the point spread function to an integral of one, and scale the Fourier transforms appropriately we can ensure that the convolution with $h(-x)$ as described in this equation is energy conserving, which means that 1 convolved with $h(-r)$ is one. If we set the over-relaxation factor α to one, the ones cancel and we obtain the following updating scheme:

$$x^{(i+1)} = x^{(i)} \left[\left(\frac{y_n}{x^{(i)} \circledast h(r)}\right) \circledast h(-r)\right]$$

The scheme is called the Ritchardson-Lucy (RL) update scheme. It has the advantage that convexity and convergence can be proven and the current object guess $x^{(i)}$ will automatically stay positive, thus the tricks as described to ensure positivity are not really needed here. However, this update scheme is converging relatively slowly and the problem of ill-posed-ness, which we saw in panel d) of Fig. 11.5, still remains to be an issue. To illustrate this Fig. 11.6b shows the RL result with the iteration number adjusted for

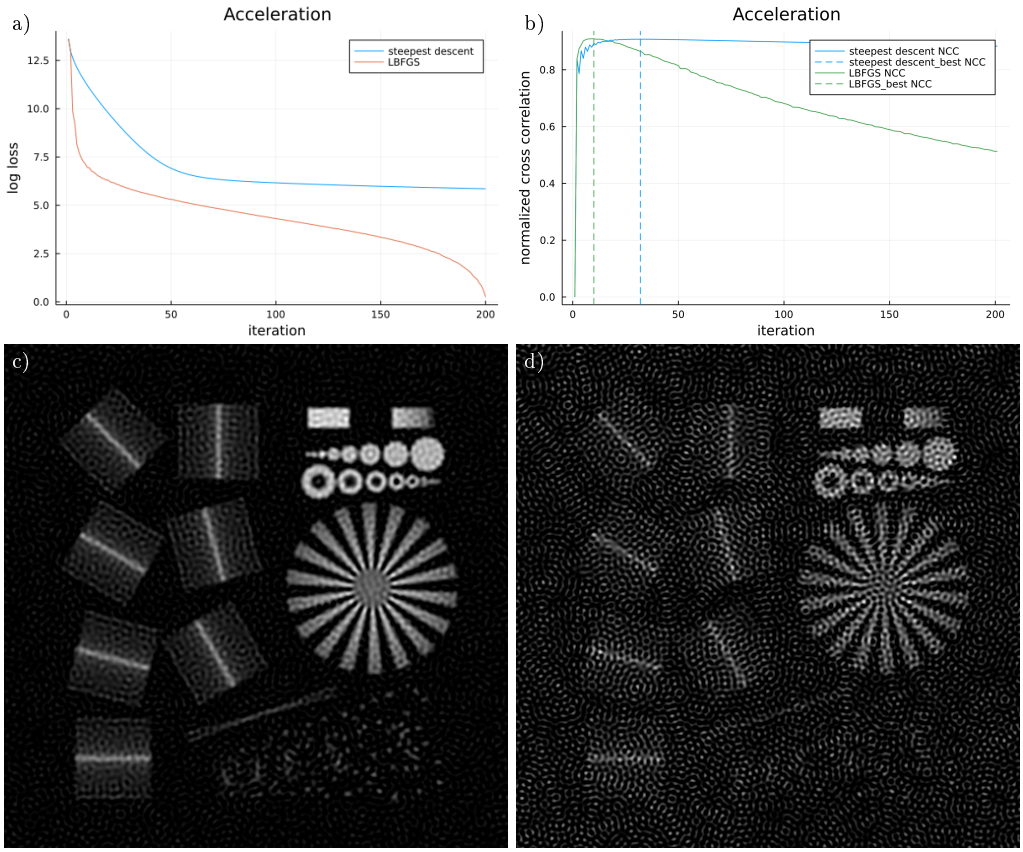


Figure 11.5: Acceleration and over-fitting. (a) Comparison of $(\log(\mathcal{L} - min(\mathcal{L}_{LBFGS}) + 1)$ for steepest decent and LBFGS for Gaussian noise (variance 1000, brightest pixel 1000) and Gaussian noise model, here shown clipped below zero. (b) Comparison of normalized cross correlations between reconstruction at every step of the iterations and the ground truth. The dashed lines indicate the best results. (c) Final result reached after 200 iterations for steepest descent. (d) Result reached after 200 iterations LBFGS.

an optimal NCC, whereas Fig. 11.6c shows the result for ten times as many iterations. Unconstrained maximum likelihood deconvolution has the tendency to decompose the image into a selection of very bright pixels, especially if the object being images is a point-like object.

This is very important to remember, since it means that it does not make sense to quantify resolution based on iteratively deconvolving images of point-like objects using any ML scheme. It is fairly easy to generate point-like deconvolution results (e.g. of 50nm width) of objects that are in reality even 100nm wide. Resolution should thus never be quantified based on ML-deconvolved images.

11.5 Regularization

As noted above, due to the ill-posed-ness of the problem, finding the minimum in the negative log likelihood is not necessarily yielding a solution that we would judge as a good one because such a solution is relatively far from how objects typically look like (see Fig. 11.5d, Fig. 11.6c and Fig. 11.7c). We therefore have to rethink the problem. By looking for our minimum we actually aimed at finding a maximal probability of measuring exactly the data y which we measured, given a set of parameters x. In statistics this is formalized as $P(y\,|\,x)$. This did not allow us to include prior knowledge about possible objects. Yet, what we really are interested is $P(x\,|\,y)$, called the posterior, meaning to maximize the probability of a reconstructed object x given the measured data y. Luckily there is an equation of statistics which allows us to relate these two quantities

$$P(x\,|\,y) = \frac{P(y\,|\,x)P(x)}{P(y)}$$

This equation is called Bayes' equation and the branch of statistics is called Bayesian statistics. Here $P(x)$ is called the prior and it describes our previous knowledge about which objects (sets of parameters x) are probable and which ones are not. The term $P(y)$ just refers to the probability of the data being measured but since this is a constant, it will not affect our attempt to maximize the probability. We can now use the same trick and minimize the negative log likelihood $\mathcal{L}$which, omitting the data term $P(y)$ becomes

$$\mathcal{L}_{\mathrm{MAP}} = \mathcal{L} + \lambda\mathcal{L}_{\mathrm{prior}}$$

With the negative log likelihood of the prior $\mathcal{L}_{\mathrm{prior}}$ encompassing our knowledge and the regularization constant λ describing how important the prior is to us. The subscript MAP in the posterior $\mathcal{L}$ refers to the name of this method: maximum a posteriori likelihood.

Even though the priors can sometimes be analytically derived from the probability $P(x)$, most of the time they are rather heuristically introduced at the level of $\mathcal{L}_{\mathrm{prior}}$. One class of priors are called Tikhonov regularization:

$$\mathcal{L}_{\mathrm{Tikhonov}} = |Cx|_2^2$$

with a linear mapping C being applied to the reconstructed object and then calculating the sum of squares. If C is just the identity matrix, we are penalizing bright object emission values and favoring low values. The underlying probability assumption $P(x)$

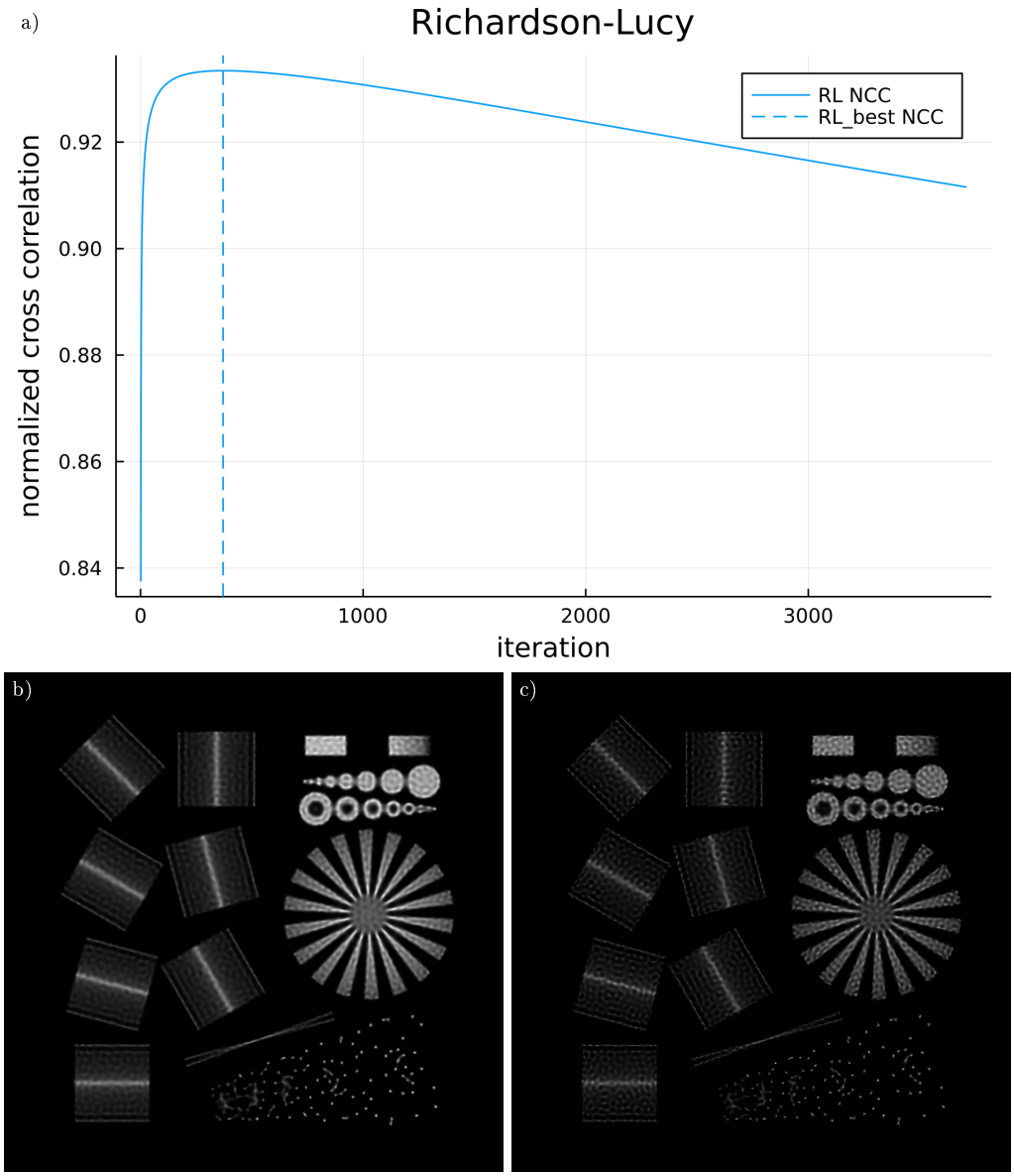


Figure 11.6: Richardson-Lucy iterations of data with Poisson noise (maximal pixel 1000 expected photons). (a) 371 iterations (optimal $NCC = 0.933$, $loss = -1.091312575$) and (b) 3710 iterations ($NCC = 0.911$, $loss = -1.09132315293$).

would be a Gaussian distribution centered at 0, the width of which is regulated by the regularization constant λ. If we consider two neighboring pixels $x_1 = 1$ and $x_2 = 1$ and compare them to a reconstruction of $x_1 = 2$, $x_2 = 0$ and assume that both models have very similar loss when compared to the measured data, they would differ in the prior: $\mathcal{L}_{\text{prior}} = 1^2 + 1^2 = 2$ compared to $\mathcal{L}_{\text{prior}} = 2^2 + 0^2 = 4$. We therefor see that even without the mapping C, Tikhonov regularization prevents the reconstruction from being decomposed into sets of individual speckles (Fig. 11.7g). The matrix C can for example be calculating the gradient along the spatial dimensions of the reconstruction in which case the Tikhonov regularization would force the reconstruction to be smooth. You could call this penalty a gradient-squared penalty term.

However, in addition to Tikhonov regularization there are other possibilities. A particularly prominent one is total variation:

$$\mathcal{L}_{\text{TV}} = |\nabla x|$$

The absolute length of the gradient along spatial coordinates (not to be confused with the gradient in the vector space of objects and images) is used here as a penalty. This prefers objects with equal brightness. The result, as seen as an approximation (see below, $\epsilon = 10^{-8}$) in Fig. 11.7e yield reconstructed objects which look a bit like a van Gogh painting. TV yield piece-wise constant results, whereas gradient-squared regularization yields smooth results. TV has the problem that gradient-based reconstruction algorithms may suffer from the discontinuity in the first derivative.

Thus, to alleviate this problem and also the allow for a better compromise between blurry results and van-Gogh appearance the modified TV has been introduced [**?**]:

$$\mathcal{L}_{\text{TV}'} = \sqrt{|\nabla x|_2^2 + \epsilon}$$

in which the extra parameter ϵ allows to smoothly transition between TV ($\epsilon = 0$) and a smooth appearance for large values of ϵ. A deconvolved result for an optimal choice of ϵ (here $\epsilon = 0.5$) is shown in Fig. 11.7f

In fluorescence microscopy a particularly good penalty term is called Good's roughness penalty [**?**]:

$$\mathcal{L}_{\text{GR}} = \frac{|\nabla x|_2^2}{|x|}$$

the gradient operator ∇ referring again to a spatial derivative (along X,Y,Z). Like the gradient-squared penalty it yields blurry results, but you can interpret the division by $|x|$ as an adaptive form of the regularization constant λ, which means that low intensity objects are strongly forced to a smooth appearance, whereas bright objects are almost not penalized. Since the background intensity in fluorescence microscopy is dark with some occasionally detected background photons, this penalty seems to be working quite well as it yield a very smooth background appearance whereas foreground objects will be crisp and sharp. This simple and extra-parameter-free penalty is fairly easy to implement and should be considered the gold standard to compare other penalties to. See Fig. 11.7d for a result.

Real world objects are often discrete structures, the the two-norm based regularizations seem non ideal as they lead to two much blurring. Yet the TV penalty yield piecewise constant object reconstructions (staircase effect), which is also a problem. Therefore

a thought has been to advance one derivative order and construct a penalty which likes piece-wise constant derivatives. This is achieved by the Hessian Schatten norm of the gradient [29]:

$$\mathcal{L}_{\text{Hessian-Schatten},p} = \left(\sum_{n=1}^{N} D_{n,n} \right)^{1/p}$$

where N is the number of spatial dimensions to regularize, p refers to the order of the Schatten norm and D refers to the diagonal matrix of eigenvalues of the Hessian matrix evaluated over spatial dimensions. The best results seem to be obtained for $p = 1$. See Fig. 11.7f for a result.

11.6 Choice of initial values

The choice of an initial guess of how to start the iterative optimization can be crucial. One the one hand, if this choice is already close to the solution, one can possibly speed up the process significantly. On the other hand, this choice may also bias our solution to a particular corner of the null-space in this under-determined problem, especially if we did not introduce a regularization term. A bad choice of the initial guess for a problem which is not strictly convex, may lead us to a part of the search space which contains a close-by local minimum or is bounded by a non-convex set of constraints.

The obvious choice may be to start the iterations with assuming the measurement to be the solution. However, this turns out to be a very bad choice especially in non-regularized problems. The high frequencies which are present in the noise will often remain to be present in the result as they are not eliminated by the algorithm since the forward model and the gradient are limited to low frequencies only by the band-limited nature of the PSF. A possibility would be to band-filter the measured data, yet this would introduce negative values, which are inconsistent with the solution space that is often restricted to all-positive values. A possibility is to start with convolving the measured value by the PSF which is also used in the deconvolution process. Since the PSF is also all-positive, so is the convolved measurement from which the high frequencies have then been reliably removed.

It turns out that a much simpler starting guess is to assume to initial object to be a **constant intensity distribution with its mean equal to the mean of the measured data** (or one, if the normalization as described below is accounted for). For most algorithms, the first step of their iteration will then yield the convolved version as the second estimate and this is quite simple to program.

11.7 Normalization

Even though the loss as well as the gradient depend on a joint scaling of data and model, the solution, i.e. the point of minimal loss, does often not depend on this. Looking at Eq. 11.6, you can see that introducing a scaling constant α into both, data and model cancels in case of the Poisson model and the Anscombe model for $b = 0$, and only yields a gradient scaled by α for the Gaussian noise model, which therefore yields the same solution, but possibly a different speed reaching this solution. This is a very useful property, since often

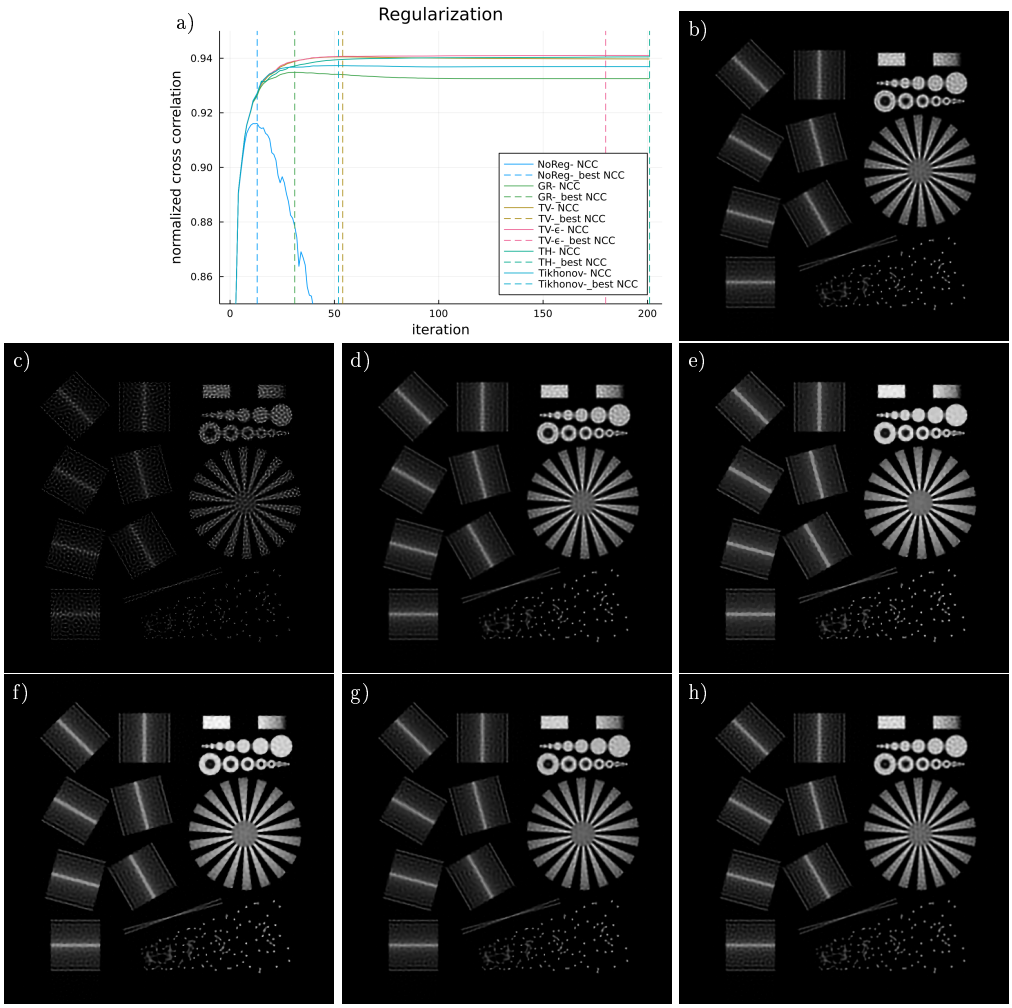


Figure 11.7: Comparison of regularizations roughly hand-optimized for best λ. (a) Quality comparison as a function of iterations. (b) no regularization, best @29 iterations: $NCC = 0.9345$, (c) 201 iterations non-regularized, $NCC = 0.8094$. (d) Good's roughness regularization, best @31 iterations: $\lambda = 7\,10^{-4}$, $NCC = 0.9348$, (e) Total variation (TV) regularization, best @54 iterations: $\lambda = 7\,10^{-4}$, $\epsilon = 10^{-8}$, $NCC = 0.9405$ (f) Total variation (TV) regularization with ϵ, best @180 iterations: $\lambda = 7\,10^{-4}$, $\epsilon = 0.5, NCC = 0.9410$ (g) Hessian Schatten norm regularization, best @201 iterations: $\lambda = 4\,10^{-4}$. $NCC = 0.9405$ (h) Tikhonov intensity regularization, best @52 iterations: $\lambda = 2\,10^{-4}$, $NCC = 0.9373$

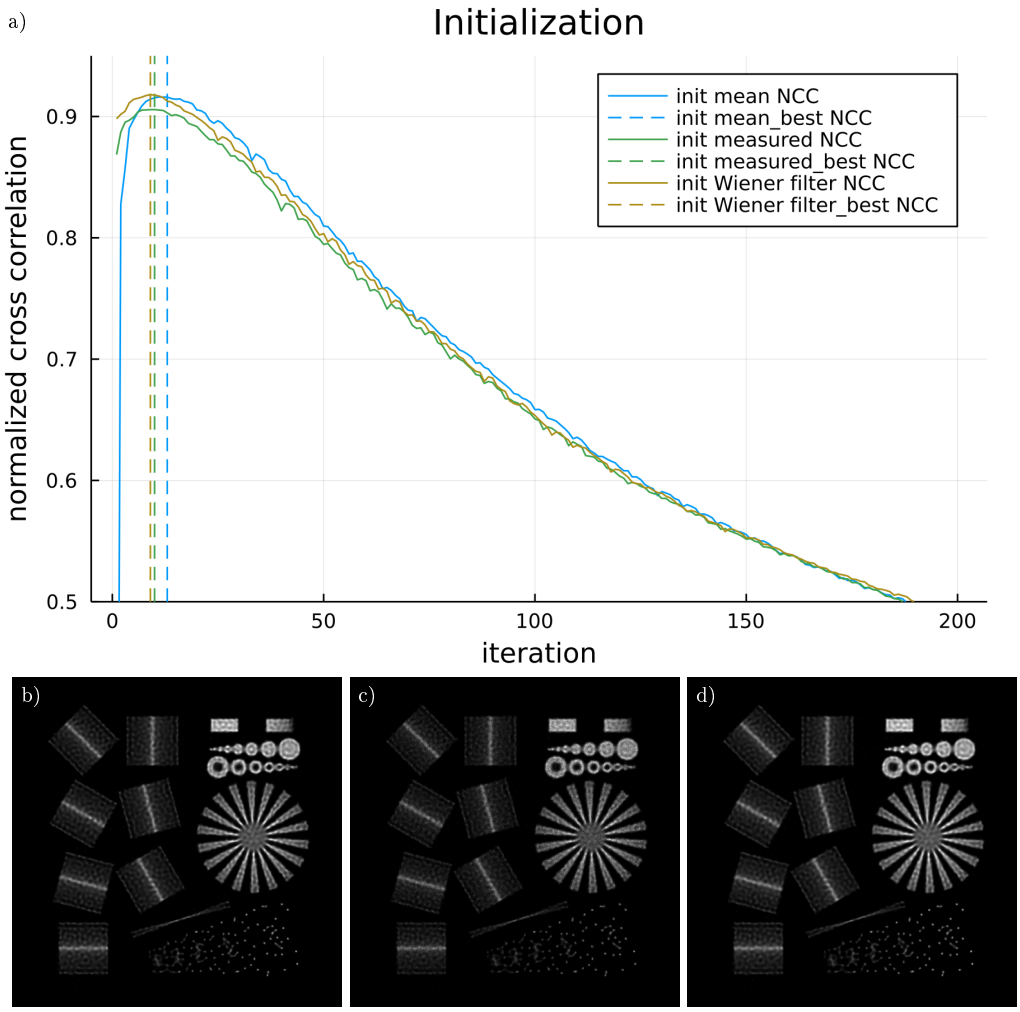


Figure 11.8: Comparison of initial values. LBFGS positivity constrained deconvolution of Poisson data with 100 expected photons in the maximum. (a) Quality comparison as a function of iterations. (b) Initial guess as a constant equal to the mean value of the measured data, best @13 iterations: $NCC = 0.9160$, (c) Initial guess as measured data, best at @10 iterations $NCC = 0.9057$. Note the residual high-frequency noise in the reconstruction. (d) Initial guess being a Wiener filter (with $\sigma = 1/3$ of the expected photons), best @9 iterations $NCC = 0.9181$.

the absolute scaling of the data and therefore the relative amount of photon noise are not known and algorithms would be needed to determine this. However, since the solution of the iterative maximum likelihood deconvolution does in most cases not depend on this scaling, we do not need to worry about it except for the argument of the convergence speed. Indeed, more advanced optimization algorithms which account for the value as well as the gradient may change their convergence rate, if the value changes and the gradient stays constant, as it is the case for the Poisson noise model. For the Anscombe method, the value of b influences the minimum. However, for reasonable values of b (0, 3/8 and 1) with the data being properly scaled to photons, simulations yielded only very minor changes in the result and convergence rate with a slight preference to the choice of $b = 0$. This choice is recommendable also since then the Anscombe method is also yields a solution, which does not depend on the scaling constant α.

To tackle the dependence of the convergence rate, it is useful to normalize the data to a mean value of one. This then removes the influence of scaling on the convergence speed.

An interesting aspect is the dependence of the algorithms on the data size and one may be tempted to scale the data to a sum of one rather than an average of one, or to use the mean rather than the sum of the individual pixel loss terms as the data loss to optimize on. However, if we assume that the data and the model are both twice as large in area, e.g. by replicating data and model along X, we obtain twice the loss, when it is defined as the sum and the vector of the gradient has twice as many dimensions since also the gradient essentially gets replicated twice (with the exceptions of possible wrap-around and border-region influences). If we approximate the two sub-problems (left and right part) as being independent, then the algorithms would actually do the right thing and solve these two problems simultaneously with identical steps as they would for each individual problem, if the loss defined as the sum. Thus the loss functions should remain to be sums rather than mean values, but of course the mean value may still be a useful progression indicator to report during convergence since it does not depend on the size of the problem, but it should not be used as the loss to optimize.

Normalization is relatively important especially for large data-sets as often encountered in microscopy image deconvolution, since a bad normalization can lead to very large ratios of gradient length compared to significant changes in the loss. If the a gradient step of an iterative algorithm causes a change in the loss, which ends up in the range of the numerical precision of summing many millions of data points, the algorithms are typically stuck as they detect essentially inconsistencies between changes in the loss (which are then zero or random) and gradient steps which are used for the minimization. Normalization as described can help to alleviate such problems.

11.8 The incoherent PSF

For the quality of deconvolution the incoherent point spread function is essential. Interestingly, if the PSF is too broad, the deconvolution result will be over-deconvolved, i.e. look skeletonized. If the PSF used for deconvolution is too narrow (e.g. delta-like), that the deconvolution has little effect and the result is still blurry. Since the point spread function should not contain noise and measurements of PSFs are not easy, often theoretically modeled PSFs are used for deconvolution. Yet the experiment is very often not

performed under ideal conditions. E.g. the objectives used in microscopy need to be operated within very stringent limits of temperature and for oil objectives one can only image less than one micrometer into a water-embedded sample without starting to get significant aberrations. For all of these reasons results are often much better if measured point spread functions are used for deconvolution rather than theoretically calculated ones. To alleviate the aforementioned noise problem of the PSF, one typically measures many tens of sub-resolution fluorescent micro-spheres ("beads") which are then superimposed by 2D or 3D cross-correlation routines and the result might be low-pass filtered to remove residual noise. Depending on the quality of the optical setup one can also try to radially average the PSF to further remove the noise, yet this has the disadvantage that non-central aberrations such as astigmatism are averaged away and also the influence of the pixel form factor is removed. The fact that the microscope integrates over pixels (e.g. a square pixel in a camera-based widefield system and a line defined by the pixel dwell time in a confocal scanning system) should be accounted for in the PSF calculation (which often is not done), but luckily it is automatically accounted for in PSF measurements, since the same effects also happen here, if these PSFs are measured at the same pixel pitch. Ideally measured PSFs can also be projected onto a basis set or themselves be fitted by a coherent PSF model for denoising, but this is also rather complicated and therefore rarely done. Ultimately there is also the possibility of estimating the PSF with the data as done in blind deconvolution, see the section below.

11.9 Sampling

Deconvolution seemingly can yield super-resolution results due to the constraints and regularizations. Even though the forward model can include resampling small object pixels onto larger measured pixels, this is time consuming and rarely done. Therefore a recommendation is to already acquire data on a fine pixel grid. For optimal results in terms of achievable final resolution, the data should be acquired on approximately half the pixel pitch as dictated by Nyquist sampling. In a widefield fluorescence microscope the lateral Nyquist sampling is

$$d_{Nq} = \frac{\lambda}{4\,\mathrm{NA}}$$

with the vacuum wavelength λ and the numerical aperture NA, which yields surprisingly small recommendations such as a maximum object-space pixel pitch of

$$\frac{d_{Nq}}{2} = \frac{520nm}{8 \cdot 1.4} = 46\,nm$$

for a confocal system and an image scanning microscope (ISM) system, assuming the excitation and emission wavelength being the same, this value would essentially need to be divided by two again. However the signal to noise is rarely good enough to be able to exploit such high resolution which is why sampling confocal or ISM data at half the widefield Nyquist limit, i.e. as stated in the example above, is a more realistic goal. For a detailed treatment of the Nyquist limits including the axial direction see the chapter on sampling in [11].

Sometimes there are other requirements, such as fast imaging and large sample sizes, which may prevent choosing the recommended small pixel pitches. Often one sees severely

under-sampled two-photon data where the point spread functions are less than a single pixel in width and still deconvolution routines are being applied. The reason is that the deconvolution routine, especially when combined with regularizes also has denoising properties. In such cases the algorithms can significantly be sped up, by omitting the convolution steps in the forward model and the gradient. This leads to denoising algorithms which still can be powerful and useful. It is interesting to see that the fraction of deconvolutions which are actually just denoisings are larger than one may think particularly in two-photon imaging.

It is worth noting that the observed signal to noise ratio (SNR) drops significantly when decreasing the sampling pitch and keeping the overall light dosage that the sample receives constant. However, the data with seemingly worse quality due to the bad SNR is still better data compared to the data sampled with large pixel pitch. One way to see this is that if an coarse pixel contains 100 photons, these are now in 3D distributed into 8 pixels each containing approximately 13 photons. Even though the SNR is worse, we only gained information on more precise locations of the same photons. They can always be binned back accordingly, but the deconvolution can make better use of such data.

Yet it is dangerous to show XZ comparisons of such highly sampled, untreated, raw data with deconvolution results, which look stunning and conclude that a specific deconvolution routine is particularly powerful based on such images. A simple Gaussian filtering may actually perform just as well or the application of just the regularizer or a state-of-the-art denoising algorithm may even outperform the deconvolution.

11.10 Other modes

Deconvolution can not only be applied to 2D data-sets but also, with a much more stunning effect, to three-dimensional widefield, confocal, two-photon or other fluorescence microscopy data. Within limits it can also be applied to incoherent transmission data [?] but strictly speaking the assumption of an incoherent superposition of PSFs, e.g. absorption of point absorbers, is not correct as there are interference effects between neighboring absorbers.

However, if we restrict ourselves to fluorescence imaging modalities, there are a multitude of microscopy modes, which obtain multiple data-sets simultaneously or successively. An example is image scanning microscopy (ISM) where the confocal pinhole is replaced with a spatially resolving detector acquiring data at every scan position. Another example is axial tomography [21], where the same sample is imaged with the PSF but rotated differently. Yet another example is multi-color imaging.

The data of all of these modes can be regarded as data where the same object is imaged with different point spread function, which also may differ in their relative intensity. In the case of axial tomography, the object orientation can be registered prior to deconvolution, which in turn rotates the PSFs. Deconvolving such joined data in a multi-view deconvolution, i.e. estimating a single reconstructed object from the multitude of data is straight forward. The considerations presented above can simply be extended by assuming that the forward model now consists of convolving with multiple PSFs and the measured data consists of multiple images each of which contribute to the total loss. The only thing that gets slightly more complicated is the PSF normalization, since we do want to preserve the relative brightness values.

We can also accommodate more details of the physical model into the deconvolution. For example, deconvolution, spectral unmixing and estimation of Förster transfer can be combined into a joined deconvolution yielding superior results [**?**]. It is also straight forward to modify the forward model to account for illumination pattern such as the ones used in structured illumination.

However, we can also increase the number of unknowns by reconstructing unknown illumination patterns along with the object. It may be surprising that reconstruction of such problems are possible as there are seemingly more unknowns than measurements. However, constraints such as the positivity of the object and the illumination pattern as well as other constraints which can be introduced (e.g. Fourier-space constraints for the SIM illumination) are helping a lot here [**?**]. Such deconvolutions are called "blind deconvolution". Of course the extreme case of a blind deconvolution is to estimate the PSF and the object simultaneously. The PSF can be positivity constrained as well as band-limit constrained in 3D. Another option is to project it onto a predefined set of modes or even model it as a coherent interference of waves with unknown phases [42].

Obtaining a fast blind deconvolution routine is still an essentially unsolved problem. The field of blind deconvolution is advancing quickly and there currently are interesting new ideas arising frequently.

11.11 Spatially variing deconvolution

Deconvolution, as presented above, relied on the assumption of the PSF not variing with position. However, for real-world data this assumtion cannot always be made. For example in light sheet microscopy, illumination is achieved by a weakly focused beam from the side. The smaller its beam waist, the better the final Z-resolution of the data will be. Yet, a tighter illumination focus, will also reduce the distance (the Rayleigh distance) over which the light-sheet has approximately the same extend thickness. This balanc often causes the lightsheet thickness to vary significantly within the field of view of the detected region. This causes the PSF to vary spatially raising the question how to best deconvolve such spatially variing data. One approach is based on separately deconvolving parts (tiles) of the image data (with a little mutual overlap), each with a slightly different PSF and merging the deconvolved data. Yet, this approach seems somewhat crude as it does not reflect the continous change of the PSF with position.

Another approach decomposes the illumination light I_{illu} into a sum of separable components. To this aim, we assume that the lightsheet has infinite extend along the Y direction with a thickness Z variing along the optical axis of the illumination beam, which is here called X. Thus $I_{illu}(x, z)$ is only a function of x and z, which we can decompose via singular value decomposition:

$$I_{illu} = USV^t$$

If we consider each single value of the central diagonal matrix S_n, we see that

$$I_{illu}(x, z) = \sum_{n=1}^{N} U_n(z) S_n V_n^t(x)$$

Figure 11.9: Comparison of illumination PSFs. Z is pointing upwards. The top image shows a lightsheet generated by calculating the inensity of the focus of a cylinder lens (256×256×256 pixels, 500nm sampling, NA=0.1 in water n=1.33). For better visualization, only 50 pixels are shown in Z for each lightsheet. The thickness variation within the field of view is clearly visible. The middle image is a separable approximation with N=1 (maximum relative error is 17%) and the lower one with N=2 (maximum relative error is 0.8%).

meaning that we decomposed a non-separable function into a sum of separable functions. It is now useful to consider how a single of these separable terms behaves during imaging in a lightsheet microscope. The term variing with z becomes part of the detection PSF, since the object is moved along the z-direction during acquisition of a focus series. In contrast, the x-dependent term as well as the eigenvalue S_n can be regarded as being part of the object. A forward model considering the spatial dependence of the convolution can now be obtaind by convolving a series of modified objects, each with a PSF modified along Z:

$$I(x,y) = \sum_{n=1}^{N} \left(V_n^t(x) S_n \rho(x,y,z)\right) \circledast_{3D} \left(U_n(z) h_{det}(x,y,z)\right) \tag{11.16}$$

with ρ referring to the fluorescent object and h_{det} to the detection PSF. Since the number of eigenvalues N is generally as large as the number of pixels, this would be a very inefficient method. However, as seen in Fig. 11.9, as few as two coefficients can approximate the shape of the illumination PSF rather well. The PSF was simulated by simulating the field distribution of the focus of a lens in 3D and then summing each of the field components along one dimension of space before summing their absolute squared values. This summation in real space corresponds to slicing the pupil in the center according to the Fourier-slice theorem, thus simulating the focus of a cylinder lens. Using the a forward model accoding to Eqn. 11.16 is continuous in space and can thus be used as a fairly efficient way of deconvolving lightsheet data.

This spatially depend deconvolution is admittedly a rather special case, since it only applies to lightsheet fluorescence microscopy. However by performing tensor decompositions also other types of spatially dependen deconvolutions can be efficiently modeled,

handing field-dependen aberrations according to Seidel or a depth-dependent amount of spherical aberration.

11.12 Out of band reconstruction

The constraints, most importantly the positivity constraint but also other regularizers have interesting consequences for the object reconstruction. The optical transfer function is band limited which means that seemingly no information about spatial frequencies above the band limit can be obtained. This information was multiplied by zero in the process of imaging. However, by exploiting prior knowledge about the object true computational super-resolution is possible. The measured data is only compatible with reconstructed objects with reconstructed spatial frequencies well above the band limit. To see this, consider a point object at zero position. Its Fourier transform is a constant and lets approximate the incoherent optical transfer function by a triangle (which would be obtained using a square aperture). Let´s first assume we have infinite signal to noise in which we can even apply the direct inverse reconstruction method within the band limit, thus recovering the spatial frequencies inside the band limit perfectly. The result would be a box in Fourier space and thus in real space a sinc function with many negative values. We can now try to fix the issue with negative values of the reconstructed object by adding various spatial frequencies above the band limit. These have no effect on the predicted measurement, but can reduce the negativity. It turns out that an all-positive object can under these conditions only be achieved, if we reconstruct a constant in Fourier space, thus recovering the delta function in real space.

As we saw, under ideal conditions infinite bandwidth extension is possible. However in reality we measure noisy data which places limits on the ability to perform out-of-band reconstruction. To make matters worse, we often have a non-zero background which much reduces our ability to reliably do out-of band reconstruction and the objects are also not point objects. It turns out that points are better deconvolved than lines, which are better than areas. One can even construct objects which are particularly resistant to out-of-band reconstructions. A more detailed treatment is found in [**?**].

Chapter 12

Phase imaging

12.1 Phase imaging in wide field optical systems

12.1.1 Differential phase contrast (DPC) microscopy

12.1.2 Transport of intensity (TIE) phase imaging

Starting from the homogeneous Helmholtz equation

$$\nabla^2 U(\boldsymbol{r}) + k^2 U(\boldsymbol{r}) = 0 \tag{12.1}$$

and assuming the wave field U can be factorized into a plane wave $\exp(jkz)$ along the propagation direction z and a wave ψ whose gradient varies slowly,

$$U(\boldsymbol{r}) = \underbrace{\psi(\boldsymbol{r})}_{\partial_z^2\psi \approx 0} \exp(jkz), \tag{12.2}$$

we can derive

$$\underbrace{\partial_z^2\psi}_{\approx 0} + 2jk\partial_z\psi + \underbrace{\psi(jk)^2 + k^2\psi}_{=0} + \nabla_\perp^2\psi = 0 \tag{12.3}$$

or

$$2jk\partial_z\psi + \nabla_\perp^2\psi = 0, \tag{12.4}$$

where $\nabla^2 = \partial_z^2 + \nabla_\perp^2$. Equation 12.4 is referred to as the *paraxial Helmholtz equation* . We can multiply Eq. 12.4 with ψ^* and subtract the result's complex conjugate to get

$$2jk\psi^*\partial_z\psi - 2(-j)k\psi\partial_z\psi^* + \psi^*\nabla_\perp^2\psi - \psi\nabla_\perp^2\psi^* = 0$$

or

$$2jk\left(\psi^*\partial_z\psi + \psi\partial_z\psi^*\right) + \left(\psi^*\nabla_\perp^2\psi - \psi\nabla_\perp^2\psi^*\right) = 0. \tag{12.5}$$

Now writing the wave field in polar form,

$$\psi = \sqrt{I}\exp(j\phi), \tag{12.6}$$

we get

$$\psi^*\nabla_\perp^2\psi - \psi\nabla_\perp^2\psi^* \tag{12.7}$$
$$= \nabla_\perp \circ (\psi^*\nabla_\perp\psi - \psi\nabla_\perp\psi^*) \tag{12.8}$$
$$= \nabla_\perp \circ (\psi^* \left[\exp(j\phi)\,\nabla_\perp\sqrt{I} + \sqrt{I}\exp(j\phi)\,j\nabla_\perp\phi\right] - \dots$$
$$\dots\psi\left[\exp(-j\phi)\,\nabla_\perp\sqrt{I} + \sqrt{I}\exp(j\phi)\,(-j)\,\nabla_\perp\phi\right]) \tag{12.9}$$
$$= \nabla_\perp \circ (\left[\sqrt{I}\nabla_\perp\sqrt{I} + jI\nabla_\perp\phi\right] - \left[\sqrt{I}\nabla_\perp\sqrt{I} - jI\nabla_\perp\phi\right]) \tag{12.10}$$
$$= 2j\nabla_\perp \circ (I\,\nabla_\perp\phi) \tag{12.11}$$

and

$$\psi^*\partial_z\psi + \psi\partial_z\psi^* = \frac{\partial(\psi\psi^*)}{\partial z} = \frac{\partial I}{\partial z}. \tag{12.12}$$

Combining the results leads to the *transport of intensity equation* (TIE) [**?**]

$$k\frac{\partial I}{\partial z} = -\nabla_\perp \circ (I\,\nabla_\perp\phi)\,. \tag{12.13}$$

For pure phase specimens which are close to fully transparent, the intensity I can be treated as a constant, which simplified the transport of intensity equation further.

$$k\frac{\partial I}{\partial z} \approx -I\nabla_\perp^2\phi \tag{12.14}$$

The term on the left can be experimentally measured and approximated by finite differences. Defining

$$J(x,y) = \frac{k}{2\Delta z}\frac{I(x,y,\Delta z) - I(x,y,-\Delta z)}{I(x,y,0)}, \tag{12.15}$$

we get

$$-\nabla_\perp^2\phi \approx J, \tag{12.16}$$

which is a Poisson equation (with unknown ϕ, and known right hand side J). Upon 2D Fourier transformation $(x,y) \to (f_x, f_y)$, we get

$$\left(f_x^2 + f_y^2\right)\tilde{\phi} = \tilde{J} \tag{12.17}$$

which we can formally solve to get

$$\tilde{\phi} = \frac{\tilde{J}}{f_x^2 + f_y^2}. \tag{12.18}$$

The latter equation has a singularity at the origin, which is practically avoided by adding a constant term $\alpha > 0$ to the denominator. This term amounts to regularization

$$\tilde{\phi} = \frac{\tilde{J}}{f_x^2 + f_y^2 + \alpha}. \tag{12.19}$$

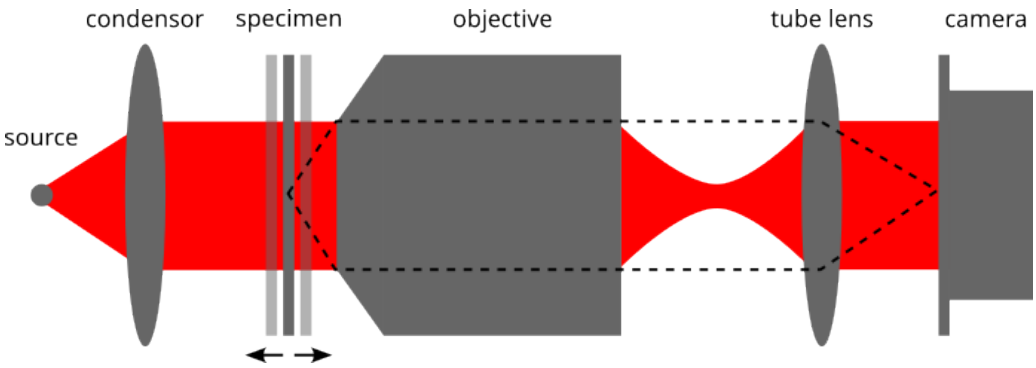


Figure 12.1: TIE experimental setup. The specimen is imaged in transmission via a wide field microscope. Three intensities are recorded at the focal plane and at slightly defocused positions $\pm\Delta z$. The red and dashed beam paths show the illumination and imaging paths, respectively.

Inverse Fourier transforming, we get a closed-form expression for the phase estimate

$$\phi = \mathcal{F}^{-1}\left[\frac{\mathcal{F}[J]}{f_x^2 + f_y^2 + \alpha}\right]. \tag{12.20}$$

Thus transport of intensity phase imaging practically amounts to a two-dimensional deconvolution of a quadratic transfer function $H(f_x, f_y) = f_x^2 + f_y^2 + \alpha$.

The experimental configuration for transport of intensity phase imaging is shown in Fig. 12.1. A transparent specimen is located at the focal plane $z = 0$. A total of three wide field intensity images are recorded in trans-illumination with the specimen positioned at $z = -\Delta z, 0, +\Delta z$ to compute $J(x, y)$ as given by Eqn. 12.15. Instead of axially moving the specimen, it is also possible to move the objective or tube lens along the optical axis. Given J, the phase image is computed via the deconvolution given by Eqn. 12.20.

An example of experimental bright field versus TIE phase imaging of an unstained COS7 adherent cell is shown in Fig. 12.2. It is seen that that TIE improves the contrast drastically. It is possible to identify nucleus, nucleoli and cell membrane, while the bright field image only features faint details. In addition, there are artefacts visible in the bright field image stemming from out of focus dust particles of the dish the cell was grown in. The axial subtraction of the raw intensities effectively removes such out of focus artefacts. It is seen that TIE phase imaging is an excellent contrast modality for labelfree microscopy. This section covered only the basic idea of TIE phase imaging. Various improvements are possible, for example lateral resolution improvement via the incorporation of partially coherent light sources and improved robustness against experimental noise via acquisition of more than three intensities at multiple defocus distances. For a comprehensive treatment, the reader is referred to the review by Zuo and co-workers [**?**].

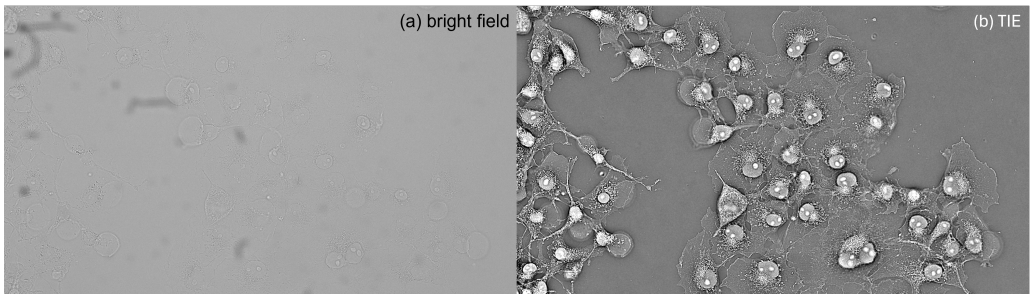


Figure 12.2: Example of (a) bright field versus (b) transport of intensity phase imaging.

12.2 Lensless phase imaging

Chapter 5 considered the forward problem of propagating a known coherent scalar wave field from one plane to another. In section 5.2 it was also discussed that photoelectric detection of infrared radiation, visible light, extreme ultraviolet radiation, and x-rays involves the conversion from complex amplitude to optical intensity, which results in a loss of phase information in the measured diffraction data. Therefore, the diffraction pattern can not directly be back-propagated to the sample plane to form an image. The problem of attributing a phase to a diffracted wave field is known as the *phase problem*. It arises in many scientific areas: In astrophysics, *stellar Michelson interferometers* measure the coherence properties of distant stars [48]. By the Van Zittert-Zernike theorem this information is related to the Fourier transform of the source intensity (cf. Eqn. 6.22). However, since the phase of the mutual intensity is not directly measured, a critical ingredient is a priori information about the source. In this case, a positivity constraint may be applied since optical intensities can not be negative. While it is hard to directly construct a confined intensity distribution in the source plane that complies with an observed far-field radiation pattern, it is possible to use iterative algorithms which impose both constraints (positivity and measured far field information) sequentially [**?**]. As a second example, in *blind deconvolution*, iterative phase retrieval algorithms can be applied to recover two functions whose convolution is measured. Yet again, this technique requires a priori knowledge such as positivity or information about a finite support [**?**]. In *inline holography*, the diffraction pattern contains phase information which can be decoded *if* the object is weakly scattering and the reference wave is known a priori [**?**].

We could ask the question how well we can live without phase information. The answer is "not so well". The importance of phase information was illustrated in a classical paper by Oppenheim [35], as reproduced in Fig. 12.3. Here two images are Fourier transformed. In the Fourier domain the phases of the signal (orange and turquoise) are exchanged while the moduli (red and green) are kept. After a final inverse Fourier transform, we see that the resulting image intelligibility is captured by each respective phase of the original signals. In the following section we discuss techniques for optical images that can computationally recover phase information, also known as *phase retrieval*.

```
% read images
g1 = imread('..\testImages\Dog.png');
```

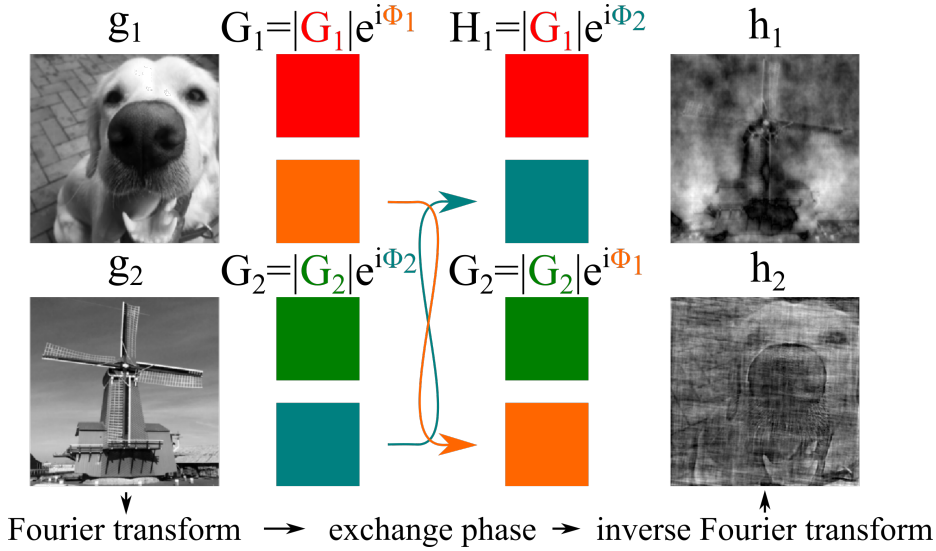


Figure 12.3: Numerical experiment adapted from Oppenheim's paper "The importance of phase" [35]. Take two images, Fourier transform each and exchange the phases in the Fourier domain. Inverse Fourier transform the result. The final images appear to mainly contain the intelligibility of the phase of the Fourier domain signals.

```
g2 = imread('..\testImages\Windmill.png');
% exctract only red channel
g1 = single( g1(:,:,1) );
g2 = single( g2(:,:,1) );
% interpolate to same size
g1 = imresize(g1, [256 256],'bilinear');
g2 = imresize(g2, [256 256],'bilinear');
%% compute Fourier transform
G1 = fft2c(g1);
G2 = fft2c(g2);
%% extract modulus and phase
mod1 = abs(G1);
phi1 = angle(G1);
mod2 = abs(G2);
phi2 = angle(G2);
%% exchange modulus and phase
H1 = mod1 .* exp(1i * phi2);
H2 = mod2 .* exp(1i * phi1);
%% compute inverse Fourier transform
h1 = ifft2c(H1);
h2 = ifft2c(H2);
%% show results
```

```
% original image 1
figure(1)
imagesc(g1)
axis image off
colormap gray
% original image 2
figure(2)
imagesc(g2)
axis image off
colormap gray
% final image 1
figure(3)
imagesc(abs(h1))
axis image off
colormap gray
% final image 2
figure(4)
imagesc(abs(h2))
axis image off
colormap gray
```

Listing 12.1: OppenheimExperiment.m (Matlab)

12.2.1 Modulus projection

In section 5.2.1 we found that in the presence of Poisson noise the conditional probability of measuring $n(x,y)$ counts on a pixelated photodetector given an incident intensity $I(x,y) = \psi^*(x,y) \cdot \psi(x,y)$ is given by

$$P\left(n(x,y) \mid I(x,y)\right) \approx \frac{I^n(x,y)}{n(x,y)!} \exp\left(-I(x,y)\right). \tag{12.21}$$

We drop the coordinate dependency in what follows. Our goal is do derive a gradient descent update on the negative log-likelihood of the above conditional probability,

$$\mathcal{L} = -\log\left(\prod_{\boldsymbol{r}} \frac{I^n}{n!} \exp\left(-I\right)\right) \tag{12.22}$$

$$= \sum_{\boldsymbol{r}} I - n\log\left(I\right) + \log\left(n!\right) \tag{12.23}$$

$$= \sum_{\boldsymbol{r}} \psi^*\psi - n\log\left(\psi^*\psi\right) + \log\left(n!\right) \tag{12.24}$$

where in the last line the summation is over all detector pixels with coordinates $\boldsymbol{r} = (x,y)^T$. We compute the Wirtinger derivative with respect to the unknown field incident of the detector, ψ, at a particular pixel location $\boldsymbol{r}'$ to get

$$\frac{\partial \mathcal{L}}{\partial \psi^*(\boldsymbol{r}')} = \psi - \psi\frac{n}{I}, \tag{12.25}$$

where we used that

$$\frac{\partial \psi^*(\boldsymbol{r})}{\partial \psi^*(\boldsymbol{r}')} = \delta(\boldsymbol{r} - \boldsymbol{r}'), \tag{12.26}$$

which resolved the summation sign in the previous equations. We can now iteratively update the estimate for the wave incident on the detector via gradient descent[1],

$$\psi_{n+1}(\boldsymbol{r}) = \psi_n(\boldsymbol{r}) - \alpha \frac{\partial \mathcal{L}}{\partial \psi^*(\boldsymbol{r})}, \tag{12.27}$$

where α is a step size. The particular choice $\alpha = 1$ leads to

$$\boxed{\psi_{n+1}(\boldsymbol{r}) = \psi_n(\boldsymbol{r}) \frac{n(\boldsymbol{r})}{I(\boldsymbol{r})}.} \tag{12.28}$$

This is a suitable update rule assuming Poisson noise in the measured data. In practice, a wide body of phase retrieval literature is found where the so called *Anscombe transformation*, a particular kind of variance stabilization transformation, is used to get a more robust update direction. This update direction assumes a cost function of the form

$$\mathcal{L} = \sum_{\boldsymbol{r}} \left(\sqrt{\psi^* \psi} - \sqrt{I} \right)^2, \tag{12.29}$$

for which a similar derivation as above leads to the so-called *modulus projection*

$$\boxed{\psi_{n+1}(\boldsymbol{r}) = \psi_n(\boldsymbol{r}) \sqrt{\frac{n(\boldsymbol{r})}{I(\boldsymbol{r})}}.} \tag{12.30}$$

Here the turquoise term is the field estimate at the n-th iteration, at which it is divided by its own modulus $\sqrt{I(\boldsymbol{r})} = \sqrt{\psi^*(\boldsymbol{r})\psi(\boldsymbol{r})}$ and multiplied by the square root of the measured data. Thus, operationally the modulus projection mandates to manipulate the field so as to "keep the phase and replace the modulus by the square root of the measured data", which explains its name. A single intensity measurement is not enough to unambiguously estimate the phase of a wave field. Below we will review phase retrieval techniques that use intensity data measured at multiple sample-detector distances (propagation-based phase retrieval), under lateral translation of the specimen (ptychography), and constrained frameworks where a priori knowledge is used in addition to the intensity measurement (for example, single shot coherent diffraction imaging).

12.2.2 Non-convexity of the modulus projection

12.3 Gerchberg-Saxton algorithm

12.4 Coherent diffraction imaging (CDI)

- Hybrid input output
- Shrink wrap

[1] Here we drop the prime notation again, as it was only needed to distinguish the earlier summation variable from the pixel location, for which we seek an update direction.

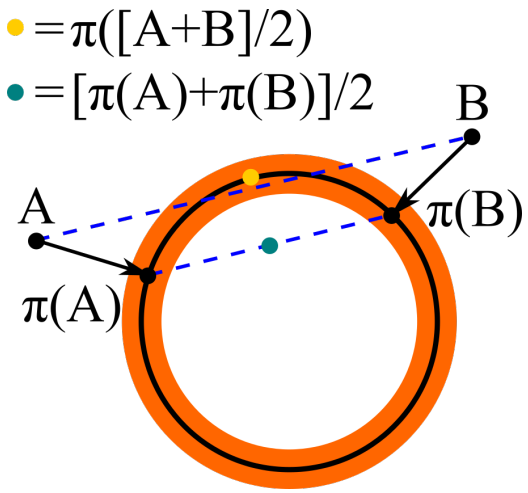


Figure 12.4: Geometric visualization of the modulus projection. The black circle represents the set of complex numbers with fixed modulus $\sqrt{n}$. The modulus projection (π) maps A and B are onto this circle via the shortest connecting line (black points). The nonlinearity (and nonconvexity) of the modulus projection operation can be seen from the fact that the average of A and B projected onto the black circle (yellow point) is not equal to the average of the projection of A and the projection of B (turquoise). In the presence of noise the correct modulus projection is uncertain and becomes a band.

12.5 Propagation-based phase retrieval

12.6 Ptychography

Chapter 13

Structured Illumination Microscopy (SIM)

Structured illumination microscopy refers to techniques which image an object by illuminating it with spatially structured light. Most modern microscopy methods use some form of structured illumination. Most commonly scanning microscopes focus onto the object and collect the emitted signal while raster-scanning the object. Confocal laser scanning micriscopy (CLSM) and two-photon laserscanning microscopy are examples. Instead of collecting just a single integrated or pinholed signal while scanning, one can also acquire images for each scan position as suggested by Sheppard [Sheppard 1989] and reassign the collected light to an appropriate location in the final image. This techniques are currently referred to under the name Image Scanning Microscopy (ISM) [Enderlein XXXX]. All of these techniques typically use a tightly focused spot as illumination structure. Imaging the fluorescence response of the sample leads under structured illumination causes an improved resolution, which can be attributed to the loss of coherence in the fluorescence emission. In CLSM the PSF of the whole raster scanned image is given by the product of the excitation PSF_{ex} with the detection PSF_{det}, which in itself can contain a finite pinhole size via a convolution. Accordingly optical transfer function of the whole system is given by the convolution of the excitation and the detection OTFs

$$OTF_{tot} = OTF_{ex} \circledast OTF_{det}.$$

In terms of the support (non-zero) region, the opotical resolution is effectively doubled [1]. However, due to this convolution of the OTFs decaying with rising spatial freqcuencies, thoses frequencies close to the new border of the passband are transferred very inefficiently. For classical CLSMs it is even worse, since a decent OTF_{det} is only achivable at very small pinhole sizes, which severely limit the detection efficiently. Luckily ISM is able to ameliorate this problem.

The strength of OTF_{tot}at high frequencies can be improved by illuminating with patterns that have more power in high spatial frequenices. In lasers scanning microscopy,

[1] neglecting the change in wavelength caused by the Stokes shift in fluroescence emission

this can be achieved by illuminating with a system of optical modes as marked by the company Bioaxial [Bioaxial XXX].

Yet structured illumination microscopy in the less general sense refers to system, which illuminate the sample with a pattern of structured light over the whole field of view and acquiring a number of images under different illumination patterns which is significantly smaller than the number of pixels in the final image. This can be done with incoherent or coherent illumination and the structures can be periodic or non-periodic (e.g. speckles). Also in this case the processed final image can essentially be described by an OTF_{tot} which is a convolution of the Fourier-transformed excitation pattern with the detection OTF_{det}. The incoherent and speckle illumination patterns are therefore not as well suited to improve the transfer strength at high spatial frequencies, whereas coherent of almost coherent illumination can generate 100% contrast close the the limit frequency, leading to strong transfer strength near the border frequency of OTF_{tot}.

Therefore we now consider only this type of SIM in which the sample is illuminated by a periodic structure of light which is typically generated by the interference of coherent laser beams and we image the fluorescence response of the sample. We first present the forward model, then a highly efficient yet general algorithm for reconstructing SIM images and finally we show how to estimate the parameters required for the reconstruction from the measured data.

13.1 Forward Model of SIM

In fluorescence SIM the fluorophores respond to the excitation intensity, which is described by a real-valued excitation intensity being periodic in one, two or three dimensions. To cover the general case, we describe the illumination pattern of illumination index p as

$$I_p = \sum_n m_{p,n} \cos(k_n x + \phi_{p,n}). \tag{13.1}$$

Each pattern of phase index p consists of a sum of cosine waves, each with its spatial frequency vector k_n. The matrix $m_{p,n}$ refers to how much each such pattern direction $k_n$contributes to this illumination pattern phase index and the matrix $\phi_{p,n}$ specifies the phase for each spatial frequency vector and phase index. This general description covers situations such as squentially illuminating three times along one direction with variing phases followed by three phases along a second direction etc. Here the matrix $m_{p,n}$ accounts for all pattern directions but contains zeros for all such vectors that do not contribute in the respective illumination pattern p. However the same equation can also describe 2D or 3D patterns where each modulation direction contributes with variing phases to each image.

The measured image is then obtained by calculating the emission pattern in the sample and convolving it with the detection PSF.

$$Y_p(x) = (\rho(x) I_p(x)) \circledast h_{det}(x) \tag{13.2}$$

, where ρ refers to the fluorophore density being a function of the spatial coordinate x and h_{det} is the detection PSF.

13.1.1 SIM and Focal Stacks

In SIM one often acquires focus stacks in which the illumination pattern is obtained by imaging a pattern generator (e.g. a spatial light modulator, SLM) though the objective onto the sample. If we imagine a very small fluoresc ent bead being fixed in space at a constant illumination phase as described by eqn. 13.1, we observe that its image will change for two reasons, when changing the focus of our microscope: We will observe a variing degree of blurring with defocus as well as a modulation caused by the different z-position. To accurately describe focal stacks, we therefore should consider the z-positions of the various modulation directions $k_{z,n}$ as modifying the detection PSF along the z direction rather than being part of the illumination pattern.

Of course a forward model could also be made by describing the varyations of the pattern focus with the acquisition of the focal series and then calculating the sum of 2D convolutions with different 2D PSFs for each z-position of the sample. Yet this approach seems complicated and it is in practic much easier to assume that each modulation order comes with a potentially different 3D detection $h_{det,m(n)}$, knowing that in practise there are typcially only two such detection PSFs, meaning $m(n) \in \{1, 2\}$. Combining eq. 13.1 with eq. 13.2 and considering the individuality of detection PSFs, we obtain

$$Y_p(x) = \sum_n \left(\rho(x) \left(m_{p,n} \cos(k_n x + \phi_{p,n})\right)\right) \circledast h_{det,m(n)}(x)$$

with the integer vector m_n signifying which PSF to use. A forward simulation can be performed efficiently by seperately summing over all orders connected to one such 3D PSF prior to convolving with it. In Fourier space this equation becomes

$$\begin{aligned} \widetilde{Y_p} &= \sum_n m_{p,n} \left(\widetilde{\rho} \circledast \widetilde{\cos}(k_n x + \phi_{p,n})\right) \widetilde{h_{det,m(n)}} \\ &= \sum_n m_{p,n} \left(\widetilde{\rho}(k - k_n) e^{i\phi_{p,n}} + \widetilde{\rho}(k + k_n) e^{-i\phi_{p,n}}\right) \widetilde{h_{det,m(n)}} \end{aligned} \tag{13.3}$$

. Note the convolution with a Fourier-transformed cosine leads to two laterally displaced copies in Fourier space for each modulation direction describing the emitted light prior to mutliplication by the corresponding 3D OTF. Note that these two displaced copies together correspond to a real-valued quantity in real space and can therefore accurately be represented in RFFT space[2] rather than full Fourier space.

13.2 SIM Image Reconstruction

13.2.1 In Full Fourier Space

To reconstruct SIM images, we need to estimate the sample $\rho(\hat{x})$ based on a number of N measured images Y_p. One straight-forward way to this aim is to integrate the forward model, as described above, into an iterative maximum likelihood framework as described in chapter 10. Yet a faster, non-iterative reconstruction is possible by assuming a uniform noise level. Before we discuss an optimized algorithm of SIM data reconstruction, we first

[2] Half of Fourier space, exploiting the Hermitian symmetry of real-valued Fourier transformations

illustrate the necessary steps in a non-optimized way. As seen in eq. 13.3 each recorded image consists of a sum modulations characterized by various $m_{p,n}e^{i\phi_{p,n}}$ and $m_{p,n}e^{-i\phi_{p,n}}$ factors. This can be regarded as an equation system

$$\widetilde{Y_p} = \sum_{n'} M_{p,n'} \widetilde{o_{n'}},$$

where $\widetilde{o_{n'}}(k) = \widetilde{\rho}(k - k_{n'})\widetilde{h_{det,m(n')}}$, refers to the individual object components which are mixed in the recorded images. Here the index n' was introduced to indicate that each positive and negative shift direction $k_{n'}$ is counted separately. In Fig. XXX we see a simulated object (XXXa) and the Fourier-transformation of one example image (XXXb), to which three such shifted object components contribute. For the reconstruction we need to retrieve the individual orders from the measured data, which we do by solving the equation system.

$$\widetilde{o_{n'}} = \sum_{n'} M^{-1}_{n',p} \widetilde{Y_p}$$

The recovered object components now need to be mulitpied by a frequency-dependent weight, shifted to their correct location in Fourier space (XXXb) and added to yield the FFT of the reconstruction result (XXXc).

$$\widetilde{R} = \sum_{n'} w(k + k_{n'}) \widetilde{o_{n'}}(k + k_{n'})$$

The shifting in Fourier-space needs to be performed with sub-pixel precision, which requires an FFT, a multiplication with a corresponding $e^{ik_{n'}x}$ term and inverse FFT. If we assume Nyquist sampling of the raw data, we need to preempt the additional sampling requirements due to the shift in Fourier-space, which typically requires padding with zeros in Fourier space, such that the number of pixels is doubled along each dimension. The frequency-dependent weights $w(k)$ are obtained by deviding by the detection OTF and weighting with the inverse variance of this rescaled data. Assuming equally distributed phases in the recorded data, the noise of the $\widetilde{o_{n'}}$ can be shown to be uniform and identical for all such separated object components. This then corresponds to a weight equal to the detection OTF which it is multiplied by prior to shifting. Finally the weight should also incorporate a term akin to generalized Wiener-filtering (see chaper XXX) to balance the effect of the noise.

Even though theoretically the weight should correspond the the detection OTF to optimize SNR, it may be advantageous to opt for a different weight which suppresses low detection frequencies. Low lateral frequencies are associated with the missing cone region of spatial frequencies in 3D. If the data is acquired only at a single focal slice, it is helpful to suppress out-of-focus blur. This can be achieved by incorporating a high-pass filter into the w factors of each separated order (prior to shifting). By the appropriate normalization over all w factors the signal from other lateral detection frequencies will fill in this region. This generates an overall transfer function, which typically fills the missing cone region. Although for data acquired as a focal stack, this high-pass filtering seems less necessary, it is in practice still useful to apply such a filter, since it also suppresses problems occuring due to bleaching or influence of further out-of-focus signal which has not been recorded in

focus. Typically the Gaussian suppression function $w(k) = \widetilde{h}(k)\left(1 - e^{|k|^2/(2\sigma^2)}\right)$ is used, with the standard deviation σ corresponding to about 10% of Abbe's frequency limit.

This reconstruction algorithm was presented to aid the understanding, yet it is not optimal in terms of computational performance, both in time and memory. In the above version requires for P images and N object components that

- Each input image needs to be subjected to a fully complex FFT before unmixing: P small FFTs
- The shifting requires two additional Fourier transforms at twice the number of pixels along each dimension for each of the N unixed components: $2N$ large FFTs
- The final image is assembled in Fourier-space and a final inverse FFT, followed by taking the real part yields the final reconstruction: 1 large FFT

In total this amounts to approximately $P + 8N + 4$ small FFTs. Here we made the approximation that a large full FFT takes about as long as 4 small full FFTs.

13.2.2 Optimized rFFT Reconstruction

A number of optimizations of the above scheme are straight forward and presented here. This yields a much faster reconstruction. We start by exploiting the linearity of the FFT, which allows us to unmix the data in real space, to retrieve "orders".

$$o_n = \sum M^{-1}_{n,p} Y_p$$

, where $M^{-1}_{n,p} = \left(M^t M\right)^{-1} M^t$ is the Moore-Penrose pseudo-inverse of the mixing matrix $M_{p,n} = m_{p,n} e^{i\phi_{p,n}}$, where index n now refers to one of the two terms contributing in the sum of eq. 13.3 in real space

$$o_n = \left(\rho(x) e^{-ik_n x}\right) \circledast h_{det,m(n)}(x).$$

This reduced matrix M works, since the full problem with both terms as separate orders to recover is of block-diagonal form, yielding a separate inverse for each part of the problem. Even though only half of the contributing orders are retrieved here, the second half is implicitely known by point mirroring and complex conjugation and assuming a shift in the opposite direction.

For SIM image reconstruction, we need to shift the Fourier-transformed orders $\widetilde{o_n}(k) = \widetilde{\rho}(k - k_n)\widetilde{h}_{det,m(n)}$ to their correct place in Fourier-space with subpixel precision. To this aim it is useful to separate this shift $k_n = k_{int,n} + k_{sub,n}$ operation into an integer $k_{int,n}$ and remainder subpixel part $k_{sub,n}$ in terms of Fourier-space pixels to shift. The subpixel partial shift is applied directly to the unmixed orders $o'_n = o_n e^{ik_{sub,n}x}$ prior to them being Fourier-transformed via a fully complex-valued FFT.

The remaining shift by integer pixels is then applied by writing $\widetilde{o'_n}$ into the appropriate place of a cumulative result image in rFFT-space. The size of this cumulative result rFFT can be appropriately expanded to account for the shifting of the separated orders. The raw data only needs to be sampled accourding to the Nyqusit sampling limit for widefield data, whereas the reconstruction needs to consider the intended resolution increase. Depending

on the precise value of $k_{int,n}$ parts of the shifted $\widetilde{o'_n}$ will not be inside the rFFT space and are ignored.

However the associated second term in the sum in eq. 13.3 still needs to be accounted for. This is achieved by point mirroring and complex conjugating the order $\widetilde{o'_n}$ in Fourier space and shifting it to the associated negative $k_{int,n}$ position. The flip and complex conjugation in Fourier space corresponds to only a complex conjugation in real space, which not only yields the correct corresponding order but also accounts for a sub-pixel shift of the right amount and in the correct direction. This conjugated, point-mirrored order $\widetilde{o'_n}$ is then also added into the rFFT result image, where contributing. Since the images may be of even size along on or multiple dimensions, the point mirror operation leads to a different location of the Fourier center for such sizes, which has to be accounted for in the $-k_{in,n}$ shift operation. To optimize the signal to noise ratio of the resulting image, one includes the weighting factor $w_n(k)$ to be applied in Fourier space to each separated order before placing it. Summarizing this, the resulting rFFT becomes

$$\widetilde{R}(k) = \sum_{n=1} \left(\widetilde{o'_n}(k - k_{int,n}) w_n(k - k_{int,n}) + \widetilde{o'_n}^*(k + k_{int,n}) w_n^*(k + k_{int,n}) \right)$$

Strictly speaking would the weights need to also be sub-pixel shifted to account for the subpixel pre-shifting of the separated components. However, we can neglect this tiny shift of the weights and still correctly account for the correct total sum of weights, which does not lead to any visible artefact.

This scheme based on rFFTs is more efficient and requires

- calculation of a single real-space component o_n by Matrix-vector multiplication
- that real-space component to be multiplied with a sub-pixel shift factor. The separability of the complex exponential can be used to minimize memory and computation requirements.
- a single small sized full FFT for this component: N small FFTs
- Hadamard-multiplication with the corresponding weight w and adding into the single result rFFT image.
- One inverse rFFT of the final result: One large rFFT.

If we assume that the FFTs dominate the performance, we have: N small FFTs, one large rFFT, which is computationally roughly equivalent to $N + 2$ **small full FFTs**, with N being the number of components, which is typically much smaller than the number of raw images. This scheme also has the advantage of a minimal memory requirement, since each order is processed and added to a final result in rFFT space and these separated orders do not need to be stored.

13.3 The Real-Space Picture

Returning to the first formulation of SIM reconstruction, it is interesting to note that SIM reconstruction can also mainly be performed in real space. The linearity of convolution and unmixing allows us to reorder the steps as follows [Master Thesis Simon Labouesse

- Pre-Filter (multiply by w in Fourier space) and upsample (pad with zeros) each raw-image: N small rFFTs, N large rFFTs
- multiply each such upsampled and pre-filtered image in real space by weighted sum of exponentials and add the result to a pre-final image. Here each image is contributing its part to various orders such that each order is correctly unmixed and shifted. After adding all the P images processed in this way, the pre-final image is only lacking the normalization and Wiener-filtering part of the SIM processing.
- Subject the pre-final image to a final Fourier-based filtering step, which can be performed using rFFTs.

This amounts roughly to $(N/2+2N+4)$ **small full FFTs**. Even though this approach is considerably slower that the above processing, it still has some advantages. One advantage is that it can easily be pipelined since the expensive pre-filtering step can start for each image as soon as it was acquired. Another advantage is that only one pre-final image is added into and no more intermediate copies of previously processed data has to be kept in memory. Apart from this it is also very interesting to compare this real-space picture of SIM data processing with classical data acquisition of confocal or spinning disc systems.

ToDo

13.4 SIM Parameter Estimation

The reconstruction methods described above are relatively straight-forward to implement, but they assume that the parameters required for the SIM reconstruction are exactly known. Yet in practice it is close to impossible to know these parameters accurately enough. It can often be assumed that the relative phase shift is well known (e.g. the changes in $\phi_{p,n}$) but not the absolute phase with respect to a reference position (e.g. the center). Most importantly the various shifts of the components in Fourier-space k_n are often only roughly known but not precisely enough for an artefact-free reconstruction. About a tenth of a pixel accuracy is needed to avoid visible artefacts.

13.4.1 Estimating the shift vectors

To estimate k_n at sub-pixel precision, one first needs a good image in which such a peak should be estimated. One could be tempted to directly use a single one of the acquired images to this aim, since under some conditions there are peak visible in the Fourier-transformation, which correspond to the spatial frequency peaks of the excitation pattern. However, the finer the exciation pattern is, the less discernable will be the peaks in the detected emission. This is caused by the low-pass filtering effect of the incoherent detection OTF. In the extreme case of the illumination pattern being generated by the interference of two in XY anticolinearly oriented plane waves from outside the objective or under total internal refraction conditions, the detection OTF will completely suppress these peaks. Yet, observing a single point-like particle or a fluorescent line-like object forming a small angle with the crests of the standing excitation wave still shows that there is an excitation pattern with close to 100% contrast, if the exciting waves are of approximately equal strength and their individual polarisation in 3D is collinear, i.e.

azimuthal with respect to the optica axis. Even in such extreme cases, each measured images contains some of the same information directly and shifted by k_g. Thus a two-dimensional spatial autocorrelation can be used to retrieve k_g by accurately localizing peaks in this autocorelation. Since we are interested in the autocorrelation of the Fourier-space, Fourier-tranforming the absolute square of the measured image should be sufficient to this aim. However, if we assume Nyquist sampling according to the OTF of widefield detection, the autocorrelation in Fourier-space will be undersampled by a factor of two, which leads wrap-around problems and aliasing of the autocorelation. This can be avoided by upsampling the raw-data prior to taking the absolute square. Yet for magnitudes of $k_g$that are below the Nyquist frequency this is often not a problem in practise such that this upsampling step can oftn be omitted.

13.4.1.1 Frequency Weighting

Yet, bigger problems are caused by the autocorrelation term of the strong zero order and the photon noise, which is of constant variance in Fourier space. It is therefore advisable to perform a frequency-weighted cross-correlation rather than the above described autocorrelation. The cross-correlation should be done with two separated components whos mutual overlap should yield the $k_g$vector to extract. Typically the central zero component and the first component. The zero component is often obtained by simply averaging all recorded orders in a squence. The first component can be obtained by a preliminary unmixing with assuming zero for the phase of the first image in a sequence of image to unmix, e.g. the first three images for SIM configurations with only two simultaneously interfering beams. However, it is often also sufficient to use an individual raw-image with the widefield component subtracted as the second image to correlate with. This has the advantage that no prior knowledge of the phase configuration is required, as long as their phases are roughly evenly distributed over the unit circle.

The frequency-weighting is achieved by pre-filtering each of the two images to correlate, by multiplying it in Fourier-space with the detection OTF, shifted to its approximate zero object-frequency position [cite Wicker Thesis here XXX] prior to multiplying one with the complex conjugate of the other to perform the cross-correlation. This detection can be further modified with a low-frequency suppression filter, which may be advantageous for thick fluorescent samples to reduce the influence of low-frequency out-of-focus signals on the cross-correlation. This modification improves the SNR in the region where the peak is expected.

To finally determine a precise estimate of k_g we assume that we already found k_g with about one-pixel precision by simply looking for a local maximum near the position where we expect it or by manually selecting it. Using a center-of-mass based algorithm or fiting a parabola to the pixels surrounding the peak is not recommended.

13.4.1.2 Iterative Subpixel Optimization

As already done by Gustafsson [Gustafsson 2000] one can then iteratively maximize the strength of the peak by shifting the higher-order to correlate with using subpixel precision shifting. This amounts to multiplying this order in real space by $e^{ik'_g r}$ with k'_g being the current estimate of k_g which is iteratively improved. Here Fourier-transformations can entirely be avoided by realizing that the peak at zero frequency, which is to optimize due

to the pre-shifting of the first order, corresponds to the sum in real space. The speed of this iterative algorithm can be further improved by expressing $e^{ik'_g r}$ as a separable product $e^{ik'_{g,x} x} e^{ik'_{g,y} y}$ and using a gradient-based optimization.

13.4.1.3 Zoom-In by CZT

This method of determining the peak is accurate but rather slow due to its iterative nature. An alternative is to zoom into the cross-correlation near the expected peak position by using a Chirp Z-Transformation (CZT) and just look at the maximum by locating the pixel with the largest magnitude. Zooming in 40-fold yields sufficient precision. The CZT exhibits the computational effort of roughly three FFTs, yet here the effort is less, as one can typically reduce the resulting array size along with applying the CZT.

13.4.1.4 Window Shifting

A third way of deterimining the sub-pixels location of the peak is based on the Fourier-Shift theorem. In a first step we correct for the integer position of the peak in Fourier space. We do this by multiplying our signal $s(x_n)$, with the pixel positions x_n with $e^{-ik'_g x_n}$ in real space. We then only have to determine a residual shift of a sub-pixel amount $\Delta k_g = k_g - k'_g$. The idea of what follows now, is best explained in one dimension: We crop the signal with a one pixel relative shift and apply an identical window to both cropped parts. Finally we sum over these two windowed results which amounts to determining the amplitude and phase of the zero frequency content in Fourier-space.

If we imagine a sharp peak to exist in continuous Fourier, which resides on a non-integer position in discrete Fourier space the cropping and windowing has the following effects. The continous signal is, due to the cropping, being shifted by exactly one pixel. Thus in continuous Fourier space, we would, due to the Fourier-shift theorom, expect a phase change of $e^{i\Delta k}$ to appear between the two Fourier-transformations. However, we cannot measure directly this peak in continuous Fourier-space. The windowing is a multiplication of the shifted signal with a fixed (and symmetric) window, which leads to a convoution with a symmetric and real-valued function in Fourierspace. Summing over the windowed signal in real space amounts to measuring the amplitude and phase of the nearby integer pixel (zero in Fourier-space) position, which should exhibit the same phase shift of $e^{i\Delta k}$. We therefore only need to determine this phaseshift Δk, which directly corresponds to the subpixel position. If we want this position normalized to pixels, it is $\Delta p = (N-1)\frac{\Delta k}{2\pi}$.

In multiple dimensions, we should apply a "slicing" operation in Fourier-space, which amounts to first summing over all other directions in which we do not want to apply the window shifting operation. The following code demostrates the determination of the subpixel peak position in multiple dimensions:

```
using SeparableFunctions
using
"""     subpixel_kg(data, k0)
Estimate the subpixel position of a peak in a multidimensional array.

Parameters:
    data: array of the data
```

```
    k0: integer Fourierspace position of the peak
"""
function subpixel_kg(data, k0)
    sz = size(data)
    # generate an object that looks like an array representing an exponential shifting operat
    # Yet the separability of such an exponential is exploitet to make to code faster.
    my_nd_exp = exp_ikx_sep(sz, shift_by=k0)
    kg = zeros(length(k0))
    for dim=1:ndims(data)
        # list all projection dimensions
        alldims = ntuple(d -> (d<dim) ? d : d+1, ndims(data)-1)
        # project over all dimensions except the current one
        proj = sum(my_nd_exp .* data, dims=alldims)
        # create a one-dimensional Hanning window over N-1 points
        win = (1 .+ cos.(range(-pi, stop=pi, length=sz[dim]-1))) ./ 2
        # sum over the shifted windowed 1D signal:
        sum1 = sum(proj[1:end-1] .* win)
        sum2 = sum(proj[2:end] .* win)
        kg[dim] = k0[dim] + (sz[dim]-1)*(angle(sum2)-angle(sum1))/(2pi)
    end
    return kg
end

### 2D Example
sz = (512,512)
kg = (110.267, 102.760)
signal = cos.(2pi * kg[1]/sz[1] .* xx(sz)) .* cos.(2pi * kg[2]/sz[2] .* yy(sz))
k0 = round.(kg)
@show kg = subpixel_kg(signal, k0)
```

13.4.2 Recovery of the Illumination Phase

Sinc the zero-order component has no phase associated to it, the peak in cross-correlation with an image containing the first order, from which $k_g$is to be extracted also exhibits the phase. If an individual raw-data, potentially with the zero-order component having been removed, image is used, these correlation phases can be used to construct the mixing- and therefore also the unmixing-matrix. If an unmixed first order component is used in the cross-correlation, the phase of the peak reveals the global phase, i.e. the true phase of the image for which a phase of zero was assumed in the preliminary unmixing.

13.4.3 Recovery of the Illumination Ampitude

The cross-correlation can also unviel the modulation strength of the higher orders by determining the magnitude of the correlation peak. Yet, in practise this may be difficult due to out-of-focus background and one may want to use a theoretical model for these amplitudes.

Chapter 14

Machine Learning

Chapter 15

Tomography

Bibliography

[1] M. A. Alonso, *Wigner functions in optics: describing beams as ray bundles and pulses as particle ensembles*, Advances in Optics and Photonics, 3 (2011), p. 272.

[2] Y. M. Altman, *Accelerating MATLAB Performance: 1001 tips to speed up MATLAB programs*, CRC Press, 2014.

[3] D. Attwood and A. Sakdinawat, *Soft X-Rays and Extreme Ultraviolet Radiation*, Cambridge University Press, 2nd ed., 2017.

[4] A. B. Bhatia and E. Wolf, *On the circle polynomials of Zernike and related orthogonal sets*, Mathematical Proceedings of the Cambridge Philosophical Society, 50 (1954), pp. 40–48.

[5] M. Born and E. Wolf, *Principles of Optics*, Cambridge University Press, 7th ed., 1999.

[6] S. Boyd and L. Vandenberghe, *Introduction to applied linear algebra: vectors, matrices, and least squares*, Cambridge university press, 2018.

[7] R. Bracewell, *The Fourier Transform and Its Applications*, Mcgraw-Hill, New York, 3 ed., 1999.

[8] D. J. Brady, *Optical Imaging and Spectroscopy*, John Wiley & Sons, 1st ed., 2009.

[9] D. H. Brandwood, *A complex gradient operator and its application in adaptive array theory*, in IEE Proceedings H-Microwaves, Optics and Antennas, vol. 130, IET, 1983, pp. 11–16.

[10] S. L. Brunton and J. N. Kutz, *Data-driven science and engineering: Machine learning, dynamical systems, and control*, Cambridge University Press, 2019.

[11] J. E. Celis, N. Carter, K. Simons, J. V. Small, T. Hunter, and D. Shotton, *Cell biology: a laboratory handbook*, Elsevier, 2005.

[12] J. W. Cooley and J. W. Tukey, *An Algorithm for the Machine Calculation of Complex Fourier Series*, Mathematics of Computation, 19 (1965), p. 297.

[13] G. C. Danielson and C. Lanczos, *Some improvements in practical Fourier analysis and their application to X-ray scattering from liquids*, Journal of the Franklin Institute, 233 (1942), pp. 435–452.

[14] A. Einstein, *Über einen die Erzeugung und Verwandlung des Lichtes betreffenden heuristischen Gesichtspunkt*, Annalen der Physik, 322 (1905), pp. 132–148.

[15] R. A. Fisher, *The use of multiple measurements in taxonomic problems*, Annals of eugenics, 7 (1936), pp. 179–188.

[16] G. Gbur, *Mathematical Methods for Optical Physics and Engineering*, Cambridge University Press, New York, 1 ed., 2011.

[17] G. H. Golub and C. F. Van Loan, *Matrix computations. Johns Hopkins studies in the mathematical sciences*, 1996.

[18] J. W. Goodman, *Statistical Optics*, Wiley, New York, 2nd ed., 2015.

[19] ——, *Introduction to Fourier Optics*, WH Freeman, 4th ed., 2017.

[20] E. Hecht, *Optics*, Addison-Wesley, San Francisco, 4th ed., 2001.

[21] R. Heintzmann and C. Cremer, *Axial tomographic confocal fluorescence microscopy*, Journal of microscopy, 206 (2002), pp. 7–23.

[22] D. J. Higham and N. J. Higham, *MATLAB guide*, SIAM, 2016.

[23] R. Johansson, *Numerical Python*, Apress, 2 ed., 2019.

[24] I. Kenyon, *The Light Fantastic: A Modern Introduction to Classical and Quantum Optics*, Oxford University Press, New York, 2nd ed. ed., 2011.

[25] M. Kot, *A first course in the calculus of variations*, vol. 72, American Mathematical Society, 2014.

[26] K. Kreutz-Delgado, *The complex gradient operator and the CR-calculus*, arXiv:0906.4835, (2009).

[27] V. Krishnamurthi, Y.-H. Liu, S. Bhattacharyya, J. N. Turner, and T. J. Holmes, *Blind deconvolution of fluorescence micrographs by maximum-likelihood estimation*, Applied optics, 34 (1995), pp. 6633–6647.

[28] T. Lancaster and S. J. Blundell, *Quantum field theory for the gifted amateur*, OUP Oxford, 2014.

[29] S. Lefkimmiatis and M. Unser, *Poisson image reconstruction with Hessian Schatten-norm regularization*, IEEE transactions on image processing, 22 (2013), pp. 4314–4327.

[30] L. Mandel and E. Wolf, *Optical Coherence and Quantum Optics*, Cambridge University Press, Cambridge, 1995.

[31] J. E. Marsden and A. Tromba, *Vector calculus*, Macmillan, 2003.

[32] K. Matsushima and T. Shimobaba, *Band-limited angular spectrum method for numerical simulation of free-space propagation in far and near fields.*, Optics express, 17 (2009), pp. 19662–73.

[33] D. A. B. Miller, *Spatial channels for communicating with waves between volumes*, Optics letters, 23 (1998), pp. 1645–1647.

[34] P. Müller, M. Schürmann, and J. Guck, *The Theory of Diffraction Tomography*, arXiv:1507.00466, (2015).

[35] A. V. Oppenheim and J. S. Lim, *Importance of Phase in Signals*, Proceedings of the IEEE, 69 (1981), pp. 529–541.

[36] G. Osnabrugge, R. Horstmeyer, I. N. Papadopoulos, B. Judkewitz, and I. M. Vellekoop, *Generalized optical memory effect*, Optica, 4 (2017), pp. 886–892.

[37] G. Osnabrugge, S. Leedumrongwatthanakun, and I. M. Vellekoop, *A convergent Born series for solving the inhomogeneous Helmholtz equation in arbitrarily large media*, Journal of Computational Physics, 322 (2016), pp. 113–124.

[38] H. M. Ozaktas, *Linear algebraic theory of partial coherence: discrete fields and measures of partial coherence*, JOSA A, 19 (2002), pp. 1563–1571.

[39] D. Paganin, *Coherent X-Ray Optics*, Oxford University Press, 1st ed., 2006.

[40] A. Quarteroni, F. Saleri, and P. Gervasio, *Scientific computing with MATLAB and Octave*, vol. 3, Springer, 2006.

[41] J. Schmidt, *Numerical Simulation of Optical Wave Propagation With Examples in MATLAB*, SPIE Press, Bellingham, 1 ed., 2010.

[42] F. Soulez and M. Unser, *Superresolution with optically-motivated blind deconvolution*, in Laser Applications to Chemical, Security and Environmental Analysis, paper JT3A38, 2016.

[43] H. Stark, *Applications of optical Fourier transforms*, Academic Press, 1982.

[44] R. Tibshirani, *Regression shrinkage and selection via the lasso*, Journal of the Royal Statistical Society: Series B (Methodological), 58 (1996), pp. 267–288.

[45] L. N. Trefethen and D. Bau III, *Numerical linear algebra*, vol. 50, Siam, 1997.

[46] G. Van Rossum, B. Warsaw, and N. Coghlan, *PEP 8: style guide for Python code*.

[47] D. G. Voelz, *Computational Fourier Optics: A MATLAB Tutorial*, SPIE Press, 2011.

[48] E. Wolf, *Introduction to the Theory of Coherence and Polarization of Light*, Cambridge University Press, Cambridge, 1st ed., 2007.

[49] R. E. Woods, S. L. Eddins, and R. C. Gonzalez, *Digital image processing using MATLAB*, (2009).