Type-Safety for Inverse Imaging Problems

TYPE-SAFETY FOR INVERSE IMAGING PROBLEMS

BY
MARYAM MOGHADAS, B.Sc.

A THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTING & SOFTWARE
AND THE SCHOOL OF GRADUATE STUDIES
OF MCMASTER UNIVERSITY
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

© Copyright by Maryam Moghadas, May 2012
All Rights Reserved

Master of Science (2012)
(Computing & Software)

McMaster University
Hamilton, Ontario, Canada

TITLE:	Type-Safety for Inverse Imaging Problems
AUTHOR:	Maryam Moghadas B.A.Sc., (Electrical Engineering) Isfahan University of Technology, Isfahan, Iran
SUPERVISOR:	Dr. Christopher Anand, Dr.Wolfram Kahl
NUMBER OF PAGES:	x, 57

To my family

Abstract

This thesis gives a partial answer to the question: "Can type systems detect modeling errors in scientific computing, particularly for inverse problems derived from physical models?" by considering, in detail, the major aspects of inverse problems in Magnetic Resonance Imaging (MRI). We define a type-system that can capture all correctness properties for MRI inverse problems, including many properties that are not captured with current type-systems, e.g., frames of reference. We implemented a type-system in the Haskell language that can capture the errors arising in translating a mathematical model into a linear or nonlinear system, or alternatively into an objective function. Most models are (or can be approximated by) linear transformations, and we demonstrate the feasibility of capturing their correctness at the type level using what is arguably the most difficult case, the (discrete) Fourier transformation (DFT). By this, we mean that we are able to catch, at compile time, all known errors in applying the DFT. The first part of this thesis describes the Haskell implementation of vector size, physical units, frame of reference, and so on required in the mathematical modelling of inverse problems without regularization. To practically solve most inverse problems, especially those including noisy data or ill-conditioned systems, one must use regularization. The second part of this thesis addresses the question of defining new regularizers and identifying existing regularizers the correctness of which (in our estimation) can be formally verified at the type level. We describe such Bayesian regularization schemes based on probability theory, and describe a novel simple regularizer of this type. We leave as future work the formalization of such regularizers.

Acknowledgements

I would like to express my sincere and honest thanks to my supervisor, Prof. Christopher Anand for his guidance and support throughout my graduate studies at McMaster University. Indeed, his careful supervision and consistent help and support made it possible for me to finish this work.

I am very grateful to Prof. Wolfram Kahl for reviewing my thesis and for his valuable suggestions and comments.

Finally, I would like to give my special thanks to my mother and my husband, for their encouragement and support.

Notation and abbreviations

AMPL	A Mathematical Programming Language, page 48
CS	Compressive (Compressed) Sensing, page 3
DFT	Discrete Fourier Trasform, page 9
FT	Fourier Transform, page 16
GHC	Glasgow Haskell Compiler, page 8
HMM	Hidden Markov Model, page 42
MAP	Maximum a posteriori Probability, page 35
ML	Maximum Likelihood, page 36
MRI	Magnetic Resonance Imaging, page 2
MRF	Markov Random Field, page 36
SNR	Signal to Noise Ratio, page 49
TSVD	Truncated Singular Value Decomposition, page 3
TV	Total Variation, page 3

$\delta(x)$	delta distribution, meaning the limit of functions which are 0 except at 0, but whole integral over any interval containing 0 is 1, page 10
$\mathbb{R}$	real numbers, page 9

Contents

Abstract **iv**

Acknowledgements **v**

Notation and abbreviations **vi**

1 Introduction **1**
1.1 Motivation 1
1.2 Contribution of this Work 2
1.3 Organization of this Thesis 2

2 Our Type-Level System **5**
2.1 Existing Static and Dynamic Type Checking 5
2.1.1 Physical Units 6
2.2 Enhanced Typing for Scientific Computation 8
2.2.1 Motivating Example: The Fourier Transform 9
2.2.2 Frames 13
2.2.3 Discretizations and FTs 15
2.2.4 Detailed Type-Level Implementation 17

3 Inverse Problems And Regularization **24**
3.1 Inverse Problem 24
3.2 Ill-posed and ill-conditioned problems 25
3.3 Regularization 26
3.4 Classical Approach 27
3.4.1 TSVD 28
3.4.2 Tikhonov Method 29
3.5 Compressive Sensing 31
3.6 Statistical Approach 32
3.6.1 Hidden Markov Random Model 36

3.7 Statistical Methods Versus Classical Methods 40

4 The New Regularization Method **41**
4.1 Statistical Derivation 42
4.2 Complexity of Our Model 45
4.3 Acceptable Patterns 46

5 Results **48**

6 Conclusion **53**

List of Tables

5.1 Comparing the ℓ_2-norm of the remaining error in the optimized image with the ℓ_2-norm of the original noise 49

List of Figures

2.1 Fourier Transform [Skl88] 11
2.2 K-space and Image-space Data 12
2.3 Sampling of height of water in a canal 15
4.1 Templates to generate allowed patterns. 46
5.1 model-1pixel/feature-9features 49
5.2 model-12pixel/feature-9features 49
5.3 l2-norm of the remaining error and the original noise 50
5.4 number of visible spots for different pixels/feature w.r.t noise scale . . 51
5.5 optimized image- 3pixels/feature- sn=0.2 52
5.6 optimized image- 2pixels/feature- sn=0.7 52

Chapter 1

Introduction

The first intent of this thesis is to present a type-system capable of verifying the correctness of scientific software used in applications like medical imaging, where correctness is especially important.

1.1 Motivation

One basic aspect of type safety in scientific computation is physical units, which can prevent us from performing nonsense computations like adding 2m to 20s. To motivate its necessity, we introduce one of NASA's Mars Surveyor program, which failed because of a mismatch between physical units.

The Mars Climate Orbiter (MCO) was the second robotic space probe in NASA's Mars Surveyor program, which was launched in 1988. MCO was designed to manage simultaneous inspections of Mars's atmosphere, climate, and surface. It also served as a communications relay for another surveyor called the Mars Polar Lander. After entering Mars's orbit, no signals were received from the space craft and communication was lost. The MCO team's investigation found that the main cause of the failure was the improper use of physical units in the coding of ground software to calculate the trajectory of the space craft. The output from the ground software was to be in metric units of Newton-seconds, but the data was reported in Imperial units of pound-seconds instead. Therefore, an incorrect trajectory was computed for the MCO related to that output data with the wrong unit. As a result, the space craft went to an improperly low altitude above Mars and disintegrated because of the atmospheric stresses.

Now consider a hypothetical situation where a surgeon receives a medical image with left and right reversed which can lead to an operation on the wrong leg. This illustrates the importance of accounting for the frame of reference, which is an origin,

a vector space basis together with physical units to measure some characteristics of objects in it, before doing such computation.

1.2 Contribution of this Work

This thesis is about establishing a type-system that can capture these particular problems and many other related problems. Our primary target is verifying the correctness of medical imaging computation. In experimental MRI, tissue density information is not directly measurable. Instead, MRI experiments collect samples of the continuous Fourier Transform of the tissue density (and other interesting tissue properties). Therefore, we have two types of quantities in such experiments including the quantity of interest (e.g. tissue density) and the measurable quantity which is related to the quantity of interest via a mathematical model called a forward model. The process of extracting the quantity of interest from the observable measurement (data) in such experiments is called an inverse problem. Hence, to get the interesting tissue properties we have to deal with inverse problems in medical imaging. To solve an inverse problem, we need to use regularization, which is explained in chapter 3. Hence, we have to verify the correctness of both the model and the regularizer to have a complete verification of the resulting software. We implement a type-system that can capture the errors arising from model encoding. Most models are (or can be approximated by) linear transformations like the Fourier transformation, which was encoded into our type system, and they can be encoded into our type-system analogously. Hence, the first focus of this thesis is to implement a type-system coded in Haskell to verify the correctness of scientific computation including size, physical units, frame of reference, and so on.

To complete this task we need to formalize the regularization term using type theory. The problem is that we could not do this for most regularizers. We believe this is because they were motivated by numerical considerations rather than physical considerations so physical units can not be added in a consistent way. Therefore, the second focus of this thesis is to introduce and review inverse problems and regularizations such that we can propose a reasonable Bayesian regularization method with properties that we believe will make formalization possible.

1.3 Organization of this Thesis

In chapter 2, we present our type-system to verify the correctness of scientific computation, focusing initially on medical imaging. First of all, it summarizes two general aspects of scientific computation which can be captured in the existing type systems

encompassing both size-matching, which was previously implemented using type-level numbers, and physical units which were implemented in the existing Dimensional package [1]. Then we explain that, to have a satisfactory type-system for verifying scientific/medical computation, we needed additional features in our type-system. One is combining both sizes and units to produce types to capture properties of a frame of reference related to physical measurements. Another important feature is the assignment of physical units to symbolic expressions rather than just numerical values. To introduce our type-system, we present the implementation details including type-level numbers encoded into our type-system for both size and dimension, implementation of the arithmetics on such typed numbers, defining a class for formalizing the frame of reference, formalizing the discretization concept by defining a new **data** type, and formalizing the Fourier transform using a **class** definition.

In chapter 3, we introduce the concept of inverse problems. We explain that, in most cases, the inverse problems are ill-posed or ill-conditioned in the sense of Hadamard's conditions, meaning that they may not have a solution, solutions may not be unique or may not depend continuously on the data. Then we introduce the concept of regularization which is the most common technique to solve ill-posed inverse problems. The main idea of the regularization methods is to employ the additional a priori information to make a family of approximate solutions. Regularization methods are divided into two categories, classical (deterministic) and statistical. We introduce two classical methods: TSVD and Tikhonov methods. Then, we discuss both ℓ_2-norm and Total Variation regularizations, common in imaging, which stem from Tikhonov methods. We also introduce compressive sensing (CS) , which is one of the most recent methods introduced for under-sampling problems. Finally, we explain statistical and Bayesian methods and discuss hidden Markov random fields as one of the most common Bayesian methods in signal and image restoration. At the end, we compare these two classes of methods.

Since the leading regularizers were developed in an abstract mathematical context, they do not contain physical units or other type attributes which can be used to infer correctness properties. Some models are derived entirely using probabilistic arguments, which seem to be the most promising method of either automatically deriving regularizers or being able to check for validity in some sense. As should be clear from the discussion in chapter 3, most regularization methods contain some dimensionless parameters, the interpretation of which depends on computational heuristics, and not model properties. This is in conflict with the general understanding that regularization is a way of incorporating a priori information about the model. Therefore, to have a completely formalized inverse problem, we develop a new Bayesian regularization method without such magic numbers.

[1] http://hackage.haskell.org/package/dimensional-0.10.2

In chapter 4, we present a new Bayesian regularizer based on observed tissue segmentation in medical images, together with an informal version of the type of argument we hope to have automatically applied in our future type system. We will also give an example of common regularizers whose statistical derivation is likely to be very difficult, or to result in "a priori" knowledge which would not be acceptable to end users (e.g., the radiologist) if presented in a form understandable to them.

In chapter 5, we present the numerical results of our novel regularization method with respect to the image de-noising problem. To identify the effect of our regularization method to remove the noise from the measurement data, we used the ℓ_2 difference which is an objective numerical measurement, but the ℓ_2 difference cannot really capture one of the most interesting properties in imaging which is whether features of a certain size will be visible after de-noising. Therefore, we considered a test where there is a repeating pattern of features of the certain size.

To sum up, we implement a strong type-system which is able to capture errors in mathematical modelling like the Fourier transformation errors. We also propose a Bayesian regularization method which has a stronger theoretical foundation that we hope will lead to formalization and encoding in the type system.(see chapter 6)

Chapter 2

Our Type-Level System

Our main goal was to implement a type-system capable of verifying the correctness of scientific software used in applications like medical imaging, where correctness is especially important.

The most widely used method for verifying properties of programs automatically is type checking. The most common form, static type checking, occurs at compile time rather than run time. Static type checking is always preferred because for example, MRI image reconstruction software needs to execute while the patient is in the machine. Therefore, It is the best to capture errors before the code runs to avoid wasting expensive scanner time and minimize the discomfort to sick patients.

The power of static typing depends on the language used and varies greatly, from simple checking of storage-type compatibility in C to elaborate proofs of correctness embedded into types in Agda [BDN09]. We chose Haskell [HHPJW07] as our implementation platform because Haskell is a mature language well supported by libraries that allows the encoding of properties such as list length which very few languages can reason about at compile time.

Haskell has three interfaces for advanced type-level programming. The first is multi-parameter type classes together with functional dependencies, introduced to Haskell by Mark Jones [Jon00], while a newer approach uses type families [KPcS10]. Another one is the Generalized Algebraic Datatypes (GADT). We used the first two interfaces to implement our type-system.

2.1 Existing Static and Dynamic Type Checking

One basic aspect of type safety in scientific computation is size-matching in linear algebra like vector space and matrix computation. For example, if we have two vectors with different sizes, our type-system should be able to capture the error when

we add those two vectors. This has been demonstrated for lists (see [Kis05] for a complete implementation based on ideas introduced in [McB02] [Oka99]), but is not yet used by standard packages like Hmatrix and Vector to do linear algebra and vector computation. In fact, these packages produce results in cases we would expect at least run-time errors, e.g., when adding two vectors with different lengths. In Haskell, we can augment vector and array types with run-time size information, so we will be able to verify such computational errors at run time (dynamic type checking). For example we can define a vector by:

```
data Vector a = Vector Int [a] deriving (Show,Eq)
```

Then, we can make this vector type an instance of the Num class, to make it possible to add two vectors.

```
instance (Num a) => Num (Vector a) where
  (Vector nx x) + (Vector ny y) =
                if nx == ny then Vector nx $ zipWith (+) x y
                else error $ "adding mismatched vectors of sizes "
                ++ show nx ++ " and " ++ show ny

v1 = Vector 3 [1,2,3]
v2 = Vector 4 [1,2,3,4]
```

This implementation will cause a run time error when we add those two vectors:

```
*** Exception: adding mismatched vectors of sizes 3 and 4
```

As we mentioned, we are interested in catching all errors involving vector computation at compile time (static type checking) which will be discussed later, following the approach of [Kis05].

2.1.1 Physical Units

Another aspect of type safety is physical units. As represented by the Dimensional package, it is possible to check the physical units in arithmetic computation at compile time. Physical (dimensional) information is encoded in types which wrap numerical quantities, allowing the type checker to verify the correctness of operations on those physical quantities at compile time. Therefore, using this library can prevent us from doing nonsense computation like adding 2m to 20s. We will give an example using this package, after explaining its physical unit encoding.

There are seven basic physical dimensions including length, mass, time, electric current, thermodynamic temperature, amount of substance and luminous intensity. They can be combined to produce compound dimensions. In the Dimensional package, physical dimensions are represented by the powers of the seven basic dimensions. It implements this using type level numbers to represent the powers of the basic dimensions. This package uses a type level encoding called NumTypes which is defined

in the **Numeric**.NumType module of the Dimensional package. Data type Dim represents the seven basic dimensions.

```
data Dim l m t i th n j
```

where, each type variable represents the power of its corresponding basic unit. Type variables l, m, t, i, th , n and j represent the powers of length, mass, time, electric current, temperature, luminous intensity and amount of substance respectively. They defined some type synonyms for the basic dimensions. For example the length is represented by:

```
type DLength      = Dim Pos1 Zero Zero Zero Zero Zero Zero
```

where Pos1 is a type level positive number representing number 1, meaning that the length has power equal to 1 and the power of the other basic units are 0, which is specified using Zero. Therefore, to represent the volume we should set the type as:

```
type DVolume      = Dim Pos3 Zero Zero Zero Zero Zero Zero
```

where, Pos3 represents that length has power equal to 3 in the DVolume type. Other built-in type synonyms are:

```
type DOne         = Dim Zero Zero Zero Zero Zero Zero Zero
type DMass        = Dim Zero Pos1 Zero Zero Zero Zero Zero
type DTime        = Dim Zero Zero Pos1 Zero Zero Zero Zero
⋮
```

Then, type synonyms for quantities of particular physical dimensions were defined:

```
type Dimensionless          = Quantity DOne
type Length                 = Quantity DLength
type Mass                   = Quantity DMass
type Time                   = Quantity DTime
⋮
```

We cannot present all of the implementation details of this package, but it is instructive to look at one function, the implementation of addition:

```
(+) :: Num a => Quantity d a -> Quantity d a -> Quantity d a
```

The data type Dimensional encodes both units and quantities in one data type.

```
newtype Dimensional v d a = Dimensional a deriving (Eq, Ord, Enum)
```

where v and d are phantom type variables. The phantom type variable d represents the physical dimension of the Dimensional, and v distinguishes between units and quantities using one of the two following phantom types:

```
data DUnit
data DQuantity
```

There are type synonyms for units and quantities:

```
type Unit     = Dimensional DUnit
type Quantity = Dimensional DQuantity
```

In general, a Quantity is a number (value) which has a physical unit, and it is represented by the product of a number and a Unit. The '$(* \sim)$' operator is a convenient way to declare quantities as such a product.

```
(*~) :: Num a => a -> Unit d a -> Quantity d a
```

Therefore, we can define two physical quantities representing length and mass with the following definition:

```
h = 2 *~ meter
m = 3 *~ gram
```

To show the power of static type checking for such a computation, we can try to add those two variables, h and m, resulting in a compiler error:

```
Couldn't match expected type 'Numeric.NumType.Pos
                                Numeric.NumType.Zero'
            with actual type 'Numeric.NumType.Zero'
Expected type: Quantity DLength Prelude.Integer
  Actual type: Quantity DMass Prelude.Double
In the second argument of '(+)', namely 'm'
In the expression: h + m
```

As you can see, interpretation of the error message requires knowledge of the Haskell type system and the implementation of the Dimensional package. In particular, it is not intuitive to a domain expert (e.g. an applied mathematician), and this is a relatively simple error. One of our goals is to make the error messages easier for non-programmer users to understand. This will guide the encoding of our type-system in the type system for Haskell as implemented by GHC (with extensions).

We will not use the Dimensional package in the reminder of this thesis.

2.2 Enhanced Typing for Scientific Computation

We have summarized two general aspects of type safety (container size and physical units), including existing implementations in Haskell. Some aspects of previous implementations are not compatible with our goals, e.g., separation of numbers and units in the Dimensional package which allows unsafe computations with no physical interpretation to be wrapped in types after the computation, thereby bypassing the type checker. Therefore, we implemented our own type wrappers including two implementations of type level numbers, incorporating previous ideas in a coherent implementation. To have a satisfactory type-system for verifying scientific/medical computation, we needed additional features in our type-system. One of those features is combining both sizes and units to produce types to capture properties involving the interaction of size and units to produce a new class, called a Frame which captures the properties of a discretization sufficient to guarantee the correct use of such discretizations in mathematical models. The combination of these features is much greater than the

sum of the parts. In our system, all measurements need to take place in a frame of reference with an origin so that only comparable quantities are combined.Unlike the Dimensional package, there is no way of separating units from quantities and performing unsafe computation.

2.2.1 Motivating Example: The Fourier Transform

To motivate the power of our enhanced types, consider the Discrete Fourier Transform (DFT) which is the basis of MRI imaging. With type-level sizes, we can capture incorrect applications producing 256-sized arrays from 128-sized inputs. With physical units, we can capture errors in which tissue density and velocity data are erroneously added together before or after the DFT, but neither is capable of detecting the most common and hard to detect errors. This includes scaling problems related to the Nyquist Theorem and erroneously combining data in time- and frequency-space. We can capture all of these errors, and even use type inferencing (built into Haskell compilers) to infer missing type information (including sample resolution). To appreciate the value of this correctness checking, we first review the properties of the DFT, beginning with sampling theory.

Sampling is the process of converting a signal into a numeric sequence. The Fourier transform of a signal $x(t)$, $x : \mathbb{R} \to \mathbb{R}$ is defined by:

$$X(f) = \int_{-\infty}^{\infty} x(t) \cdot e^{-2 \cdot \pi \cdot i \cdot f \cdot t} dt \tag{2.2.1}$$

Where $t, f \in \mathbb{R}$. For restoring and processing the information by computer, we must work with the discrete version of a signal which is obtained by sampling from a continuous one. It transforms a finite sequence of inputs into another finite sequence of the same length. The definition is:

$$X_k = \sum_{0}^{N-1} x_n \cdot e^{-i \cdot 2 \cdot \pi \cdot k \cdot n / N} \tag{2.2.2}$$

In the following, we show how to use the DFT to approximate the FT of a continuous signal $x(t)$, and how aliasing errors can arise.

In Fig. 2.1, part (a) represents a band-limited signal $x(t)$, part (b) represents the Fourier transform of that signal $X(f)$ which is zero outside the interval $(-f_m < f < f_m)$ because it is band-limited, part (c) represents the periodic unit impulse train, part (d) illustrates the FT of the impulse train illustrated in (c), part (e) represents the sampled version of $x(t)$ and part (f) represents the FT of the sampled version of $x(t)$. The samples $x_s(t)$ can be viewed as the product of the function $x(t)$ with a

periodic impulse train (Fig. 2.1.(c)). The train is represented as [Skl88]:

$$x_\delta(t) = \sum_{-\infty}^{\infty} \delta(t - nt_s) \ , \tag{2.2.3}$$

where $\delta(t) : \mathbb{R} \to \mathbb{R}$ is the unit impulse function and t_s is the sampling period. The sifting property of the impulse function states:

$$x(t)\delta(t - t_0) = x(t_0)\delta(t - t_0) \tag{2.2.4}$$

The sampled version of $x(t)$ can be represented by:

$$x_s(t) = x(t)x_\delta(t) = \sum_{n=-\infty}^{\infty} x(t)\delta(t - nt_s) = \sum_{n=-\infty}^{\infty} x(nt_s)\delta(t - nt_s) \tag{2.2.5}$$

Using the Convolution theorem, the time domain equation (2.2.5) can be transformed to the convolution in frequency domain $X(f) * X_\delta(f)$, where $X_\delta(f)$, is the impulse train with step $f_s = 1/t_s$, which is the Fourier transform of x_δ. Therefore:

$$X_\delta(f) = \frac{1}{t_s} \sum_{n=-\infty}^{\infty} \delta(f - nf_s) \tag{2.2.6}$$

Another property of convolution is that convolution with an impulse function shifts the original function:

$$X(f) * \delta(f - nf_s) = X(f - nf_s) \tag{2.2.7}$$

Hence, the Fourier transform of the sampled version of $x(t)$, $X_s(f)$, is represented by:

$$X_s(f) = X(f) * X_\delta(f) = X(f) * \left[\frac{1}{t_s} \sum_{n=-\infty}^{\infty} \delta(f - nf_s) \right] = \frac{1}{t_s} \sum_{n=-\infty}^{\infty} X(f - nf_s) \tag{2.2.8}$$

In conclusion, the Fourier transform of the sampled signal is multiple spectra of the original signal (within a constant factor), which centred at the sampling frequency, and its harmonics. As it is illustrated in Fig. 2.1.(f), if $f_s = 2f_m$ then there is no overlapping between the multiple spectra. The Nyquist theorem states that, any band limited signal $x(t)$ that has no frequency component for $f > f_s$ can be uniquely reconstructed from its samples $X(nT)$ if the sampling frequency satisfies $f_s > 2f_m$ which is called the Nyquist criterion. This criterion is one of the important FT properties which should be encoded in our the type-system, because when the sampling resolutions exactly satisfy the Nyquist criterion, the sampled spaces are connected by

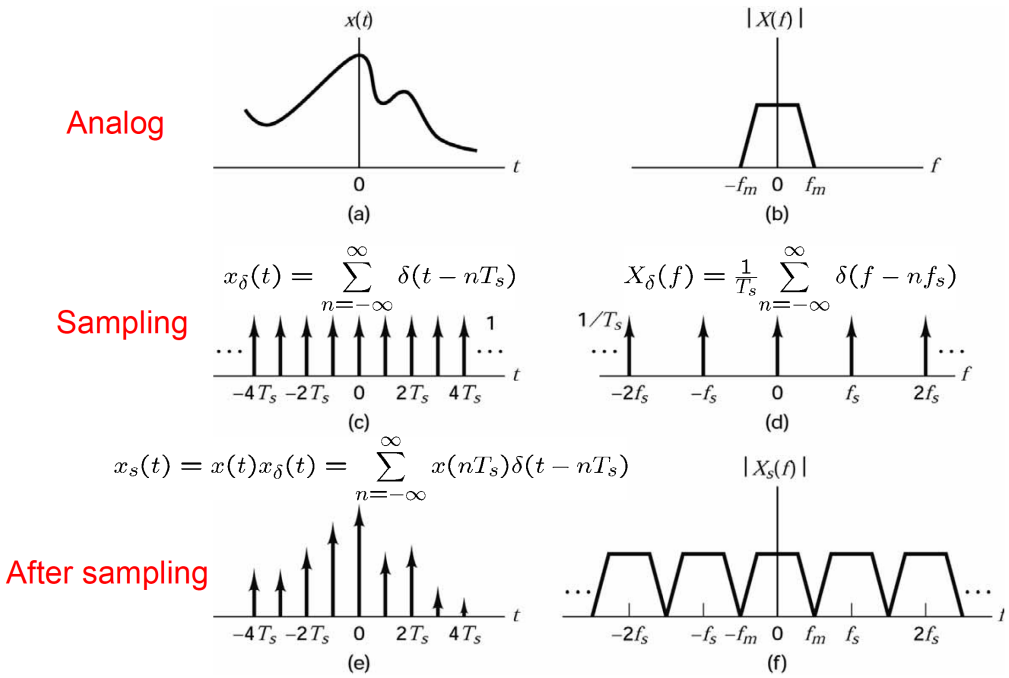

Figure 2.1: Fourier Transform [Skl88]

the *Fast* Fourier Transform.

In experimental MRI, tissue density information is not directly measurable. Instead, MRI experiments collect samples of the continuous Fourier Transform of the tissue density (and other interesting tissue properties). The dual-space to physical space is called k-space, as are the measurements of the Fourier Transform. Note that, unlike applications of the Fourier Transform to sound production, MRI requires multi-dimensional transforms (3D for normal space, 2D for planar imaging, and even higher dimensions for some types of velocity and diffusion imaging). Correctness in MRI reconstructions depends on precisely encoding the meaning of data at the type level to prevent errors in processing, from the obvious error of mixing k-space and image-space data, to subtle errors involving incompatible resolutions in k- and image-space.

First of all, the number of samples (resolution) should be equal in both k- and image-space which is represented by N. The second one is to encode the Nyquist criteria into our type-system resulted in the relation between sampling rates in k- and image-space. According to Fig. 2.1, the Fourier transform of a non-periodic discrete function $x_s(t)$ is a periodic continuous function. The $X_s(f)$ is a continuos signal in the frequency domain which should be discretized to be processed numerically. Likewise, the sampling in the time domain, the sampled version of $X_S(f)$, can be obtained by

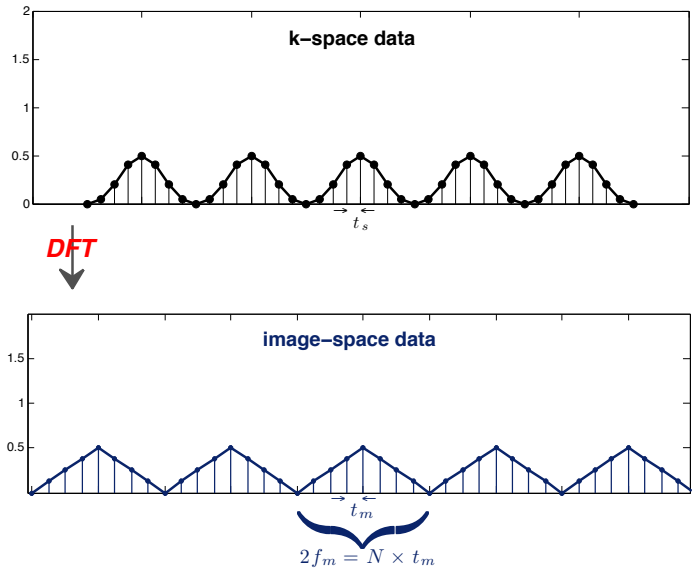

Figure 2.2: K-space and Image-space Data

multiplying by another impulse train. Such multiplication in the frequency domain affects the original signal in the time domain by making it periodic. Hence we can represent the sampled data in k- and image-space by Fig. 2.2.

In the image-space:

N : number of samples

t_m : the sampling step size in the image-space

$2f_m$: the bandwidth collected in k-space,

where image-width $= N \times t_m$.

And, in the k-space:

N : number of samples

t_s : the sampling step size in the k-space

$t_s = 1/f_s$

According to Nyquist criteria $2f_m = f_s$, therefore:

$$N \times t_m \times t_k = 1 \tag{2.2.9}$$

Which enforces constraints on k- and image-space step sizes. To encode this constraint in our type-system, we need to introduce the concept of Discretization to define an FT class.

As an example, we consider a discretized function (meas1) and infer the type of its Fourier transform, then we observe that their step-sizes satisfy the equation (2.2.9). The discretized function meas1 has type defined by:

```
meas1 :: Discretization1D     LabFrame
                              (NumSamples (SIZE3 D0 D0 D4))
                              (StepSizeNum(SIZE3 D0 D0 D1))
                              (StepSizeDenom(SIZE3 D5 D0 D0))
                              (U.Meter ())
                              [Complex Double]
```

We have not yet introduced our type-level numbers, but, for example, SIZE3 D0 D0 D4 is a three-digit type level number which represents the number 004. We can get the type of its Fourier transform from the ghci interpreter:

```
> :type (ft meas1)
(ft meas1)
  :: Discretization1D
       LabFrameT
       (NumSamples (SIZE3 D0 D0 D4))
       (StepSizeNum (SIZE3 D5 D0 D0))
       (StepSizeDenom (SIZE3 D0 D0 D4))
       (U.Meter ())
       [Complex Double]
```

The number of samples in both cases is 4. Step-size is represented as the ratio of StepSizeNum and StepSizeDenom. Therefore, the step-size of meas1 is 0.002 and the step-size of ft mean1 is 125 which satisfy (2.2.9) ($4 \times 0.002 \times 125 = 1$).

2.2.2 Frames

A frame of reference is a coordinate system or a vector space basis together with physical units and an origin to measure some characteristics of objects in it. In the existing type systems, there is no way of recording the frame of reference. To formalize it in our type-system, we considered a class for all frames:

```
class Frame frame where
  type Base frame
  name :: frame -> String
```

Note that the units are explicitly encoded on the Base, but the basis and origin are implicitly encoded. As a reminder, vector space is a set of vectors on which two operations are defined, vector addition and scalar multiplication and it should satisfy several axioms w.r.t these two operations. Moreover, given any vector space V over a field F, the dual space V^* is defined as the set of all linear maps $\phi : V \to F$ (linear functionals).

Because a frame is mathematically similar to a vector space, we can define the dual frame concept analogous to the dual vector space. Hence, for any given Frame a, set of linear maps from Base a, to Unitless is the dual frame of Frame a. To formalize the definition of dual frame, first we need to formalize the duality betwean units as a class:

```
class DualUnits a b where
instance  (Frame a, Frame b, U.Mult (Base a) (Base b) (U.Unitless ()))
          => DualUnits a b where
```

Using this definition, we can assert the duality between frames by:

```
class (DualUnits a b) => AssertDualFrames a b | a -> b, b -> a where
```

where in our type-system, the user has to specify the frame of reference, and assert duality between frames such that if one frame changes alone, it will cause a compile-time error. The functional dependency, a -> b, b -> a, denotes that knowing the type of a frame, it is possible to infer the type if its dual frame and vice versa.

To justify the necessity of formalizing the frame into an enhanced type-system, consider the orientation in medical imaging. Image orientation identifies the spatial orientation of the imaging plane with respect to the patient. The orientation is important to have a correct diagnosis. Because, when a physician examines a medical image related to a patient's leg, he or she needs to know whether it is the left leg or right one; otherwise an operation may be done on the wrong leg. Currently, this level of correctness cannot be guaranteed by the existing type systems. This property of imaging is formalized using the Frame in our type-system.

To show an application of the Frame as a class, consider a discretized measurement called meas1 which is related to the LabFrame.

```
meas1 :: Discretization1D     LabFrame
                              (NumSamples (SIZE3 D0 D0 D4))
                              (StepSizeNum(SIZE3 D0 D0 D1))
                              (StepSizeDenom(SIZE3 D5 D0 D0))
                              (U.Meter ())
                              [Complex Double]

meas1 = Discretization1D [0,1,2,3]
```

The Fourier transform of meas1 should relate to the dual frame of LabFrame meaning that adding meas1 with its Fourier transform is not a valid operation because they are not measured in the same frame. So, whereas a conventional language which does size checking would allow this operation, we flag a type error:

```
> :type (ft meas1) U.+ meas1
> Couldn't match type 'LabFrameT' with 'LabFrame'
    When using functional dependencies to combine
      AssertDualFrames LabFrame LabFrameT,
        arising from the dependency 'a -> b'
        in the instance declaration at Practice.lhs:96:10
      AssertDualFrames LabFrame LabFrame,
        arising from a use of 'ft' at <interactive>:1:1-2
```

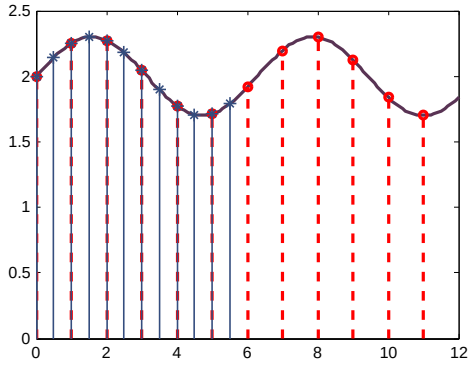

Figure 2.3: Sampling of height of water in a canal

```
In the first argument of '(U.+)', namely 'ft meas1'
In the expression: ft meas1 U.+ meas1
```

(ghc also flags errors associated with sampling sizes, but the important error above appears first.)

2.2.3 Discretizations and FTs

The Discretization data type is defined as:

```
data  (Frame frame, U.Unit (Base frame), U.Unit rangeU,
       NumSamplesC numSamples,  StepSizeNumC stepSizeNum,
       StepSizeDenomC stepSizeDenom)
    ⇒ Discretization1D  frame
                        numSamples
                        stepSizeNum
                        stepSizeDenom
                        rangeU
                        val
      = Discretization1D val
```

Where frame is a dimensional vector space; numSamples refers to the number of samples; stepSizeNum and stepSizeDenom specify the step-size; rangeU clarifies the physical unit of the discretized values; val represents discretized values. The class Unit, which is responsible for physical units encoded in our type-system, will also be described later.

We start with an application of this enhanced data type.

To increase the signal to noise ratio, one solution is to take the average of multiple samplings of the signal of interest. There are some constraints on the sampling to have a meaningful average. For example if two samplings have the same number of samples but different sampling steps, then adding those two samplings does not make sense. Suppose we have two samplings of the height of water in a canal illustrated in Fig. 2.3. In the blue sampling, there are 12 samples and the sampling step is 0.01. In the red sampling, there are 12 samples, but the sampling step is 0.005. Those two

samplings can be defined in our type system using the Discretization data type:

```
canalSample1 :: Discretization1D  CanalFrame
                                  (NumSamples (SIZE3 D0 D1 D2))
                                  (StepSizeNum (SIZE3 D0 D0 D1))
                                  (StepSizeDenom (SIZE3 D1 D0 D0))
                                  (U.Meter ())
                                  [Double]

canalSample1 =
  Discretization1D [1,1.2,1.3,1.12,1.23,1.12,1.15,1.25,1.18,1.20,1.24,1.28]


canalSample2 :: Discretization1D  CanalFrame
                                  (NumSamples (SIZE3 D0 D1 D2))
                                  (StepSizeNum (SIZE3 D0 D0 D1))
                                  (StepSizeDenom (SIZE3 D2 D0 D0))
                                  (U.Meter ())
                                  [Double]

canalSample2 =
  Discretization1D [1,1.21,1.2,1.42,1.3,1.32,1.12,1.25,1.23,1.20,1.12,1.28]
```

Trying to add those two samplings to each other will cause a compile time error. Therefore, writing

```
> canalSample1 U.+ canalSample2
```

creates an error of the form:

```
    Couldn't match expected type 'D1' with actual type 'D2'
    Expected type: Discretization1D
                     CanalFrame
                     (NumSamples (SIZE3 D0 D1 D2))
                     (StepSizeNum (SIZE3 D0 D0 D1))
                     (StepSizeDenom (SIZE3 D1 D0 D0))
                     (U.Meter ())
                     [Double]
      Actual type: Discretization1D
                     CanalFrame
                     (NumSamples (SIZE3 D0 D1 D2))
                     (StepSizeNum (SIZE3 D0 D0 D1))
                     (StepSizeDenom (SIZE3 D2 D0 D0))
                     (U.Meter ())
                     [Double]
    In the second argument of '(U.+)', namely 'canalSample2'
    In the expression: canalSample1 U.+ canalSample2
```

The error message denotes that the stepSizeDenomS are different in those two samplings, and adding them is not a valid operation.

As we mentioned, another powerful feature in our type system is the FT (Fourier Transform) class which is defined by:

```
class FT a b | a -> b, b -> a where
  ft :: a -> b
  invFt :: b -> a

instance ( Size numSamp, U.Unit rangeU, AssertDualFrames frame1 frame2
         , MultDNonZero (stepSizeNum1, stepSizeDenom1)
```

```
                        (numSamp,SIZE3 D0 D0 D1)
                        (stepSizeDenom2, stepSizeNum2))
  => FT (Discretization1D frame1
                          (NumSamples numSamp)
                          (StepSizeNum stepSizeNum1)
                          (StepSizeDenom stepSizeDenom1)
                          rangeU [Complex Double])
        (Discretization1D frame2
                         (NumSamples numSamp)
                         (StepSizeNum stepSizeNum2)
                         (StepSizeDenom stepSizeDenom2)
                         rangeU [Complex Double]) where
ft (Discretization1D x) = (Discretization1D $ fft x)
invFt (Discretization1D x) = (Discretization1D $ ifft x)
```

Where fft and ifft come from pure−fft package. Two parameters of that type class represent a discretized function and its Fourier transform. Using the multi-parameter type classes together with functional dependency, it is possible to infer the type of the FT of a discretized function if it knows the type of the discretized function itself and vice versa. In the instance definition, there are some important constraints on two parameters of FT class which are related to FT properties. First, the number of samples in those two parameters should be the same, which is encoded by clarifying the same numSamp for those discretized parameters. Second, the discretized functions (class parameters) are related to two dual frames, and this duality should be asserted by user. The most subtle constraint is mentioned by the MultDNonZero class. This constraint specifies that the product of the first two parameters should be equal to the third parameter, meaning that:

$$\frac{\text{stepSizeNum1}}{\text{stepSizeDenom1}} \times \frac{\text{numSamp}}{\text{SIZE3 D0 D0 D1}} = \frac{\text{stepSizeDenom2}}{\text{stepSizeNum2}}$$

Which is the same as:

$$\text{stepSize1} \times \text{stepSize2} \times \text{numSamp} = 1$$

This is exactly what we got in equation (2.2.9).

2.2.4 Detailed Type-Level Implementation

Now, we explain the interesting implementation details of our type-system. First of all, we present the type-level numbers encoded into our type-system. There are several encodings of the type-level number including Peano numbers, binary encoding, and so on. We used a phantom type representation of a sequence of decimal digits because decimal encoding makes error messages more comprehensible. Since we are using the decimal notation, we need the types for all ten digits:

```
data D0
data D1
data D2
```

```
...
data D9
```

There is a class called Digit of which all 10 type level digits are instances. It has a method to convert a type level single digit to its corresponding data level number.

```
class Digit a where
  digit :: a -> Int
```

We defined a phantom data type called SIZE to implement a 10-digit length (fixed precision) type level number. We also implemented a smaller version named SIZE3, which is a 3-digit length type level number to make the examples shorter. We made our 10-digit and 3-digits numbers an instance of class Size, which has a method **toInt** to convert a type level number into the corresponding data level (i.e. run-time) number.

```
data SIZE d9 d8 d7 d6 d5 d4 d3 d2 d1 d0

data SIZE3 d2 d1 d0

class Size a where
  toInt :: a -> Int
```

For 10-digit numbers we did the same method as 3-digits to make it an instance of the class Size.

```
instance forall d2 d1 d0 . (Digit d2, Digit d1, Digit d0)
         => Size (SIZE3 d2 d1 d0 ) where
  toInt _ = digit d0 + 10 *(digit d1) + 100 *(digit d2)
    where
      d0 = undefined :: d0
      d1 = undefined :: d1
      d2 = undefined :: d2
```

In the instance definition, d0, d1, and d3 are all types, but we need terms with these types in the definition of **toInt** function. This is why we defined several terms with an undefined value and these types in the **where** expression. Then we implemented the required arithmetic operations on our type level number. There is a class called Times for multiplying two single digits with each other.

```
class Times da db high low | da db -> high , da db -> low where
```

Then we specified all of its instances w.r.t different combinations of any two digits. For example:

```
instance Times D0 D0 D0 D0 where

⋮

instance Times D1 D0 D0 D0 where
instance Times D1 D1 D0 D1 where

⋮

instance Times D9 D1 D0 D9 where
```

⋮

```
instance Times D9 D9 D8 D1 where
```

For multiplying two type level numbers, e.g two number of type Size, we needed to implement the type level addition of 3 to 20 digits. Here, we present such additional classes for adding 3 and 4 digits. Other classes for adding more number of digits have the same implementation. For adding 3 digits, Add3 is implemented as:

```
class Add3 a1 a2 a3 sh sl | a1 a2 a3 -> sh, a1 a2 a3 -> sl where

instance (Add2 a1 a2 a1a2h a1a2l, Add2 a1a2l a3 a1a2la3h a1a2a3l,
          Add2 a1a2h a1a2la3h D0 a1a2a3h) => Add3 a1 a2 a3 a1a2a3h a1a2a3l where
```

For adding 4 digits, Add4 is implemented as:

```
class Add4 a1 a2 a3 a4 sh sl | a1 a2 a3 a4 -> sh, a1 a2 a3 a4 -> sl where
instance (Add3 a1 a2 a3 a1a2a3h a1a2a3l, Add2 a1a2a3l a4 a1a2a3la4h a1a2a3a4l,
          Add2 a1a2a3h a1a2a3la4h  D0 a1a2a3a4h) =>
          Add4 a1 a2 a3 a4 a1a2a3a4h a1a2a3a4l where
```

We used Multi-parameter type classes together with functional dependencies in the implementation of those type-level addition meaning that by knowing all input digits (a1 a2 a3 a4) it can infer the type of both low and high digits of the result.

To present the implementation of a class for multiplying two type level numbers (of type SIZE), we present the smaller version that is responsible for multiplying two SIZE3 numbers, which captures all of the important ideas. We start by sketching the multiplication of 3 digits by 3 digits to show the required constraints for their multiplication class.

			f_2	f_1	f_0
			e_2	e_1	e_0
–	–	–	–	$[e_0 * f_0]_h$	$[e_0 * f_0]_l$
–	–	–	$[e_0 * f_1]_h$	$[e_0 * f_1]_l$	–
–	–	$[e_0 * f_2]_h$	$[e_0 * f_2]_l$		–
–	–	–	$[e_1 * f_0]_h$	$[e_1 * f_0]_l$	–
–	–	$[e_1 * f_1]_h$	$[e_1 * f_1]_l$	–	–
–	$[e_1 * f_2]_h$	$[e_1 * f_2]_l$	–	–	–
–	–	$[e_2 * f_0]_h$	$[e_2 * f_0]_l$	–	–
–	$[e_2 * f_1]_h$	$[e_2 * f_1]_l$	–	–	–
$[e_2 * f_2]_h$	$[e_2 * f_2]_l$	–	–	–	–
			g_2	g_1	g_0

For the first digit of result 'g_0', there is just one term to be counted, which is the low digit of '$e_0 \times f_0$', and this can be implemented using the Times class. The second digit of result 'g_1' is obtained by adding the high digit of '$e_0 \times f_0$', the low digit of

'$e_0 \times f_1$', and the low digit of '$e_1 \times f_0$' which can be implemented using the 'Add3' class. The third digit is the result of adding 5 different known terms which can be implemented using Add5 class. We wanted to have a class for multiplying 2 type level numbers and represent the result as another type level number with the same size because our decimal type level numbers have fixed precision and the resulting typed number should have the same number of digits as the arguments have. It means that for multiplying 3 digits by 3 digits, all higher order digits (higher than 3) should be zero. It turns out that any term in the forth, fifth, and sixth columns should be zero, and can also be implemented as a constraint using the Times class. Therefore, the MultD3 class is defined as:

```
class MultD3  f2 f1 f0  e2 e1 e0  g2 g1 g0 |
              f2 f1 f0 e2 e1 e0 -> g2,
              f2 f1 f0 e2 e1 e0 -> g1,
              f2 f1 f0 e2 e1 e0 -> g0 where
instance (  Times f0 e0 p00h p00l, Times f1 e0 p10h p10l, Times f2 e0 D0 p20l,
            Times f0 e1 p01h p01l,Times f1 e1 D0 p11l, Times f2 e1 D0 D0,
            Times f0 e2 D0 p02l, Times f1 e2 D0 D0,Times f2 e2 D0 D0,
            Add3 p00h p10l p01l c1h c1l, Add5 p20l p01h p11l p02l c1h D0 c2l)
          => MultD3 f2 f1 f0 e2 e1 e0 c2l c1l p00l where
```

Implementing MultD10 for multiplying 10 digits by 10 digits have the same procedure. In order to unify all different size numbers, we defined a MultD class such that MultD3 and MultD10 are the instances of this one:

```
class MultD f e g | f e -> g where

instance (MultD10 f9 f8 f7 f6 f5 f4 f3 f2 f1 f0
                  e9 e8 e7 e6 e5 e4 e3 e2 e1 e0
                  g9 g8 g7 g6 g5 g4 g3 g2 g1 g0)
  => MultD (SIZE f9 f8 f7 f6 f5 f4 f3 f2 f1 f0)
           (SIZE e9 e8 e7 e6 e5 e4 e3 e2 e1 e0)
           (SIZE g9 g8 g7 g6 g5 g4 g3 g2 g1 g0) where

instance (MultD3 f2 f1 f0  e2 e1 e0  g2 g1 g0) =>
              MultD (SIZE3 f2 f1 f0) (SIZE3 e2 e1 e0) (SIZE3 g2 g1 g0) where
```

In addition, we defined an extra multiplying class called MultDNonZero with extra constraints such that functional dependency works in all directions, meaning that knowing the type of any two elements of the triple (f, e, g) gives the type of third one. This property helped us in the FT definition to implement the equation we got from the sampling theorem as a constraint. Type inference would not work if some sizes were allowed to be zero, since any x satisfies $x \cdot 0 = 0$.

```
class MultDNonZero f e g | f e -> g, f g -> e, e g -> f where
instance  (NonZero f,  NonZero e, NonZero g, MultD e f g)  =>
                       MultDNonZero e f g where
```

Type level numbers are needed for both sizes and dimensions, but dimensions are never big numbers, so it makes sense to create small numbers using phantom data types to represent different powers for each basic unit. For each required basic

unit w.r.t the different powers (we do not need all basic units for medical imaging purpose), we created a data type. For example:

```
data M0   -- means the dimension length to the power of 0 (dimensionless)
data M1   -- means the dimension length to the power of 1
data M2
...
data M_1  -- means the dimension length to the power of -1
data M_2
```

For every basic unit, we needed a class with a method to convert type level dimensional numbers (from -5 to 5 is enough for our application) into its corresponding data level number. For example for length, we have a class called UnitM:

```
class UnitM a where
  toIntM :: a -> Int
  instance UnitM M0 where
  toIntM _ = 0
instance UnitM M1 where
  toIntM _ = 1
...
instance UnitM M_5 where
  toIntM _ = -5
```

There is also another class for multiplying two powers of the same basic dimensions, which is responsible for simplifying the resulting dimensions by adding their powers.

```
class AddDim a b c | a b -> c, a c -> b, b c -> a where
```

We added all of its reasonable instances to our type-system. For example:

```
...
instance AddDim M_4 M3 M_1 where
...
```

The above instance means that $m^{-4} \times m^3$ is equal to m^{-1}.

For composite units, we defined a data type constructor called SIUnit which has 5 type arguments to represent the basic units, and another type variable to represent the value which has this unit, for example a double or even an expression.

```
data SIUnit u1 u2 u3 u4 u5 val = SIUnit val
```

Type synonyms were used for more general composite units:

```
type Unitless val  = SIUnit M0 Kg0 S0 A0 Mol0 val
type Meter val     = SIUnit M1 Kg0 S0 A0 Mol0 val
type PerM val      = SIUnit M_1 Kg0 S0 A0 Mol0 val
type MPerS val     = SIUnit M1 Kg0 S_1 A0 Mol0 val
...
```

We implemented a class, Unit for all possible units which accepts only composite units which have the basic units in the specific order.

```
class Unit a where

instance  (UnitM m, Show m, UnitKg kg, Show kg, UnitS s, Show s,
           UnitA a, Show a, UnitMol mol, Show mol, Show val)
           => Unit (SIUnit m kg s a mol val) where
```

We also needed to implement the required arithmetic operation with respect to the physical quantities. Such operations are more restricted than their default implementation in the **Prelude**. For example, adding or subtracting two physical quantities only makes sense when both have the same dimension. We implemented the Mult class with the '$(*)$' method and the Add class with the '$(+)$', **negate** and '$(-)$' methods:

```
class Mult a b c | a b -> c, a c -> b, b c -> a where
  (*) :: a -> b -> c

class Add a where
  (+) :: a -> a -> a
  negate :: a -> a
  (-) :: a -> a -> a
```

Then we made the SIUnit data an instance of those classes:

```
instance  (UnitM u1, UnitKg u2, UnitS u3, UnitA u4, UnitMol u5, Add val)
          => Add (SIUnit u1 u2 u3 u4 u5 val) where
  (SIUnit x) + (SIUnit y) = SIUnit (x + y)
  negate (SIUnit x) = SIUnit (negate x)
  (SIUnit x) - (SIUnit y) = SIUnit (x - y)
```

Which states that addition and subtraction are only valid when both quantities have the same dimension.

```
instance  (AddDim u1 v1 w1, AddDim u2 v2 w2, AddDim u3 v3 w3,
           AddDim u4 v4 w4, AddDim u5 v5 w5, Mult val val val)
    => Mult  (SIUnit u1 u2 u3 u4 u5 val)
             (SIUnit v1 v2 v3 v4 v5 val)
             (SIUnit w1 w2 w3 w4 w5 val) where
  (SIUnit a) * (SIUnit b) = SIUnit (a * b)
```

Where the power of each basic unit in the first arguments is added to the power of the same basic unit in the second argument to make the physical dimension of the result. The functional dependency in all directions makes it possible to infer the physical unit of any argument by knowing the two others.
At this state, we can show an example using the physical dimensions in our type-system. Suppose we have three physical quantities representing distance, time and velocity defined as:

```
distance = SIUnit 2 ::Meter Double
time = SIUnit 10 :: Second Double
velocity = SIUnit 0.2 :: MPerS Double
```

If we multiply velocity and time, the resulting quantity has the distance dimension which makes sense.

```
>  velocity * time
2.0 m
```

But if we try to add time with distance it will cause a compile time error:

```
> time + distance
  Couldn't match expected type 'M0' with actual type 'M1'
    Expected type: Second Double
      Actual type: Meter Double
```

```
In the second argument of '(+)', namely 'distance'
In the expression: time + distance
```

In the error message, it explicitly mentions that the expected type is Second **Double** and the actual type is Meter **Double** which is easy enough for user to understand.

Our main intention to implement a type-system for scientific computation was to verify the correctness of medical imaging computations specially for MRI. In such cases, we have to deal with inverse problems, which are explained in detail in the next chapter. We discussed earlier that the tissue density is not directly measurable, and in most physical/medical experiments, there is the same situation meaning that we rarely can measure the quantity of interest directly. The process of extracting the quantity of interest from the observable data in such experiments is called inverse problem. To solve an inverse problem, we need to use regularization which is discussed in the following chapter. Hence, we have to verify the correctness of both the model and the regularizer to have a complete verification for that resulting software. Most models are (or can be approximated by) linear transformations like the Fourier transformation, and they can be encoded into our type-system analogously. The important part is to formalize the regularization term. The problem is that we could not do this for the leading regularizers. For example, we could not fit $\ell_1 - norm$ and Total Variation regularizations into our type system. We believe this is because they were not developed from physics but from abstract mathematical arguments. Therefore, the second focus of this thesis is to introduce and review inverse problems and regularizations so that we can propose a reasonable Bayesian regularization method with sufficient properties that we believe can be encoded in our type-system.

Chapter 3

Inverse Problems And Regularization

3.1 Inverse Problem

An inverse problem means converting observed data (measurements) into information about an unknown quantity that we want to determine. This unknown quantity depends on the measured data via a system or model.

Inverse problems appear in many fields of science, including medical imaging (such as MRI), computer vision, machine learning, statistical inference, geophysics, ocean acoustic tomography, astronomy, physics, and many other branches. In most of these fields, the quantity that we are interested in determining is different from the observable quantity. When the measured data depends on the unknown quantities of interest in terms of an explicit model, then reconstructing the unknown quantities of interest is called an inverse problem. For example in MRI, the observable quantity is the electromagnetic field related to protons in the tissue. But, the quantity of interest is the tissue density.

In a typical inverse problem, the idea is to determine an estimate for the unknown quantity f based on a noisy measurement g that is related to f according to a known model K such that $g = K(f)$. Inverse problems are classified by whether or not they satisfy Hadamard's principles of well-posedness. Well-posed problems can be solved unambiguously. Ill-posed problems may not have a solution, or solutions may not be unique or depend continuously on the data. Regularization was introduced to deal with ill-posed problems.

3.2 Ill-posed and ill-conditioned problems

The basic idea of a well-posed problem was introduced by the French mathematician Jacques Hadamard. In most cases, forward problems (also called simulations) are well-posed and easy to compute, whereas the inverse problems are ill-posed or ill-conditioned in the sense of Hadamard's conditions. Hadamard, in his lectures published in [Han98], claims that a mathematical model for a physical problem is well-posed if it has the following properties:

1. There exists a solution of the problem (existence).
2. There is at most one solution of the problem (uniqueness).
3. The solution depends continuously on the data (stability).

Mathematically, the existence of a solution can be imposed by extending the solution space. The non-uniqueness problem can be solved by adding a priori information into the model. The third condition is the most crucial one. The continuity condition is related to the stability or robustness of the solution. Continuity, however, is a necessary but not sufficient condition for stability. A well-posed problem can still be ill-conditioned, meaning that even small changes in the measurement space, e.g. due to noise, result in large changes in the model space. Ill conditioning can be measured by the condition number which is the ratio of the the largest to the smallest singular values of the Jacobian matrix of partial derivatives. A well-posed problem, in order to have solutions that are robust against noise, must also be well-conditioned. Many ill-conditioned problems are improved by adding regularization.

Historically, mathematicians were motivated by systems like Fredholm integral equations, which were often ill-posed:

Example : Fredholm integral equations of the first kind [Idi08].
This is a equation of the form:

$$y(s) = \int_a^b K(s,t)x(t)\,\mathrm{d}t \tag{3.2.1}$$

In this equation y is a given function which is usually the known data or measurement, $K(,)$ is the kernel of the equation and x is the solution of interest which is an unknown function. The existence of solutions is not guaranteed (or obvious) and depends on the properties of K. Another challenging question is whether the solution is unique. For example, if $K(s,t) = s\sin t$, the function $x(t) = 1/2$ is a solution of :

$$s = \int_0^\pi s\sin(t)x(t)\,\mathrm{d}t \tag{3.2.2}$$

But the crucial point is that each of the functions $x_n(t) = 1/2 + \sin(nt)$, for $n = 1, 2, 3, \ldots$ is a solution to that equation too.

According to the Riemann-Lebesgue lemma which states that for any square integrable kernel, $K(.,.)$,

$$\int_0^\pi K(s,t)\sin(nt)\,\mathrm{d}t \to 0 \quad \text{as} \quad n \to 0 \ ,$$

if x is a solution of Eq.(1) and A is an arbitrary value, then

$$\int_0^\pi K(s,t)(x(t) + (A\sin(nt))\,\mathrm{d}t \to y(s) \quad \text{as} \quad n \to 0 \ ,$$

So $\tilde{x}(t) = x(t) + A\sin(nt)$ is the solution of

$$\tilde{y}(s) = y(s) + A\int_0^\pi K(s,t)\sin(nt)\,\mathrm{d}t$$

for large values of n, $\tilde{y}(s) \to y(s)$ but $\tilde{x}(t)$ differs noticeably from $x(t)$.
Therefore, for Fredholm equations of the first kind, solutions usually do not depend continuously on the data.

3.3 Regularization

Regularization methods are the most commonly used techniques when solving ill-posed inverse problems. The goal of regularization is to find a well-posed estimation for the original ill-posed problem such that the approximate solution would be as close as possible to the exact solution. Therefore, regularization is a standard method to transform an ill-posed problem into a well-posed one. The main idea of the regularization methods is to employ the additional information explicitly in order to make a bunch of approximate solutions. Regularization methods are divided into two categories, classical (deterministic) and statistical.

In the following, we first explain the motivation for adding regularization terms. Secondly, we introduce two classical methods including TSVD and Tikhonov methods. Then, we discuss both ℓ_2-norm and TV regularizations, popular in imaging, which stem from Tikhonov methods. We also introduce compressive sensing which is one of the most recent methods introduced for under sampling problems. Finally, we will formally explain statistical and Bayesian methods and discuss hidden Markov random fields as one of the most popular Bayesian methods in signal and image restoration.

3.4 Classical Approach

Any practical or physical system can be represented by three main parameters: input, system parameters (forward model of the system), and output. Two possible inverse problems related to such a physical system are:

- Given the system parameters (forward model) and the output, determine the input of the system.
- Given the input and the output of the system, determine the system parameters (forward model of the system).

The first one has more application in experimental situations, especially in imaging fields. When the second problem arises, it is often called a calibration problem. To formalize the first situation, the system can be introduced by the mapping $h : \mathcal{X} \mapsto \mathcal{Y}$, which is the forward model. $\mathcal{X}$ is called the model space which contains all possible values of the unknown quantity of interest, x. Likewise $\mathcal{Y}$ is called data space which includes all possible values of data (measurement), y. Therefore a nonlinear forward problem can be represented by:

$$y = h(x) \tag{3.4.1}$$

If the problem is linear, then, the forward system can be represented as:

$$y = Hx \tag{3.4.2}$$

In most practical cases there are measurement errors, so the measured data may not be a member of the range of h, meaning there is not an exact solution for:

$$h(x) - y = 0 \tag{3.4.3}$$

which violates Hadamard's first condition as explained before. To handle the non-existent solution problem, a common approach is calculating an approximate solution using the least squares (LS) method:

$$x_{LS} = \arg\ \min_x \|h(x) - y\|^2 \tag{3.4.4}$$

But there is no guarantee about the uniqueness and stability of such estimates. So in general we have to deal with an ill-posed problem.

The null space of h, $\mathcal{N}(h) = \{x \in \mathcal{X} : h(x) = 0\}$, has an important role in the ill-posedness analyses of (3.4.1). For example, if $x_{\mathcal{N}} \in \mathcal{N}(h)$ and x_{LS} is a solution of (3.4.1) then, $x_{LS} + x_{\mathcal{N}}$ is a solution too. So if the null space of h is a non-zero set, then the uniqueness will be violated. For linear cases, the null space analysis is

possible by singular value decomposition (SVD), which will be discussed. Nonlinear cases are usually handled by linearization.

3.4.1 TSVD

Any linear forward operator $H : \mathcal{X} \mapsto \mathcal{Y}$ is stated by the matrix $H \in \mathbb{R}^{D\times M}$. The least square equation (3.4.4) can be rewritten by:

$$x_{LS} = \arg\min_x \|Hx - y\|^2 \tag{3.4.5}$$

The singular value decomposition of H has the following form:

$$H = U\Sigma V^T = \sum_{i=1}^{q} u_i \lambda_i v_i^T$$

Where $q = \min(M, D)$, U and V are orthonormal matrixes,

$$U = (u_1, u_2, ..., u_D) \in \mathbb{R}^{D\times D}, \quad V = (v_1, v_2, ..., v_M) \in \mathbb{R}^{M\times M},$$

and $\Sigma = \text{diag}(\lambda_1, \lambda_2, ..., \lambda_q) \in \mathbb{R}^{D\times M}$ is a diagonal matrix with non-negative diagonal elements λ_i, which are called singular values of h and are ordered in the following way:

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_q \geq 0 \tag{3.4.6}$$

We represent the rank of H by r. If H is a full rank matrix then the solution of (3.4.2) is achieved by the normal equation [GVL96].

$$H^T Hx = H^T y \ ,$$

which can be represented by:

$$x_{LS} = (H^T H)^{-1} H^T y \ ,$$

If $r < q$, then the last $q-r$ singular values are zero and, respectively, the last $q-r$ singular vectors v_i are in the null space of H. This is one of the reasons which makes the (3.4.2) ill-posed as we explained before. Therefore if H is rank-deficient then equation (3.4.5) does not have a unique solution. Another reason which makes (3.4.2) ill-conditioned is that in the singular spectrum of H, there are some small λ_i values which make the unknown quantity very sensitive to the noise in the measurement.

TSVD is a common solution to both of these problems. The idea of TSVD like other standard regularizers is to specify a well-posed problem subject to the ill-posed one such that it can reduce properly the sensitivity of the solution to the measurement

noise. Using TSVD, it substitutes matrix H with one H_k of lower rank k which ignores components of the right hand side corresponding to the last $r-k$ small singular values. The matrix H_k is a rank-k matrix is defined by:

$$H_k = U\Sigma_k V^T, \quad \Sigma_k = \text{diag}(\lambda_1, \lambda_2..., \lambda_k, 0, ..., 0) \tag{3.4.7}$$

Where $k < r$ and Σ_k is like Σ in which the last (smallest) $q - k$ singular values are replaced with zero. The benefit of replacing H with H_k is that the condition number of H_k, λ_1/λ_k, will be appropriately smaller than that of H if the number k is chosen properly. [Han87] The TSVD solution to (3.4.5) is represented by:

$$x_k = H_k^+ y \tag{3.4.8}$$

Where the H_k^+ is the pseudo-inverse of H_k, which is defined by:

$$H_k^+ = V\Sigma_k^+ U^T, \quad \Sigma_k^+ = \text{diag}(\lambda_1^{-1}, \lambda_2^{-2}, ..., \lambda_k^{-1}, 0, ..., 0) \in \mathbb{R}^{M\times D}$$

3.4.2 Tikhonov Method

Tikhonov was the first person who solved ill-posed problems numerically and introduced the idea of regularization [Tik63] [TAJ77]. Tikhonov regularization is used for solving inverse problems in many fields. TSVD reduces the instability of the reconstructed quantity by explicitly truncating, while Tikhonov performs it implicitly by adding a penalty term to the objective function. Tikhonov methods incorporate a priori information about both size and smoothness of the preferred solution which has the form of the norm $\|Lx\|_2$ for discrete inverse problems.

Tikhonov methods revised the least squares equation by adding an additional so-called penalty term, [Mon05] [Kir96] and regularization parameter λ, which manages the balance between the data fit and the amount of regularization term that normally constrains the magnitude/smoothness of the solution. Hence it handles the level of smoothing. In general it is formulated by:

$$\min\{\|A(x) - y\|_2^2 + \lambda^2 \|L(x)\|_2^2\} \tag{3.4.9}$$

For discrete linear ill-posed problems, A and L are matrixes. Therefore it can be defined as:

$$\min\{\|Ax - y\|_2^2 + \lambda^2 \|Lx\|_2^2\} \tag{3.4.10}$$

Where L is often a discrete approximation of a derivative operator. Three common choices of L are I, L_1, L_2 where L_1, L_2 are respectively the first and second order

derivative operators defined by: [Han10]

$$L_1 = \begin{pmatrix} -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{pmatrix} \tag{3.4.11}$$

and,

$$L_2 = \begin{pmatrix} 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \end{pmatrix} \tag{3.4.12}$$

Therefore, the discrete norm $\|Lx\|_2$ with L equal to either I, L_1 or L_2 represents the continuous norms $\|f\|_2$, $\|f'\|$ and $\|f''\|$ respectively.

ℓ_2-norm regularization, which is used in image reconstruction, stemmed from Tikhonov regularization where L is equal to L_1, so the inverse problem can be defined by:

$$\min\{\|Ax - y\|_2^2 + \lambda^2 \|L_1 x\|_2^2\} \tag{3.4.13}$$

For the nonlinear case, nonlinear optimization methods are used to obtain the solution, while in linear case, the solution is obtained by:

$$\hat{x}_\lambda = (A^T A + \lambda^2 L L^T)^{-1} A^T y$$

The limitation of such standard regularization methods like Tikhonov regularization and Truncated Singular Value Decomposition (TSVD) is that they suppose the data set to be smooth and continuous. So they produce results that are smooth. In contrast the Total Variation (TV) regularization, introduced by Rudin, Osher, and Fatemi in [ROF92], does not impose smoothess. It can produce a discontinuous solution. Hence Tikhonov regularization produces the smooth solution while TV regularization makes the solution piecewise smooth. Hence TV regularization can preserve the edges while Tikhonov one can not.

Total Variation regularization replaces Tikhonov's ℓ_2-norm with the ℓ_1-norm of the first derivative. Hence the optimization problem for discrete inverse problem using the TV regularization is defined by: [Han10]

$$\min\{\|Ax - y\|_2^2 + \lambda^2 \|L_1 x\|_1\} \tag{3.4.14}$$

Where L_1 is defined by (3.4.11).
Therefore the $\|L_1 x\|_1$ resembles the total variation $\|f'\|_1$. To deal with sharp edges in

the source of interest, total variation regularization is an adequate method to preserves the discontinuities. However, it is more expensive because of the non-differentiability of the regularization term.

3.5 Compressive Sensing

Compressive Sensing (CS) is a method to recover signals from far fewer measurements than what is usually believed necessary by traditional methods. (Nyquist/Shannon sampling theory). The basics of the CS theory emerged in the works [CRT06, Don06]

CS introduces the fact that many signals can be represented using only a few non-zero coefficients in an appropriate basis. Reconstruction of such signals from very few measurements is possible using nonlinear optimization. Therefore, CS is an effective protocol for signal acquisition and reconstruction.

Suppose we are interested in measuring signal X which is a $N \times 1$ vector or similarly we can say $X \in \mathbb{R}^N$. Any vector in $\mathbb{R}^N$ can be represented in terms of a basis Ψ which contains $N \times 1$ vectors ψ_i. An orthonormal basis, such as wavelet basis, is chosen. The expansion of X in such a basis can be represented by [CW08]:

$$X = \sum_{i=1}^{N} s_i \psi_i \quad \text{or} \quad X = \Psi S \tag{3.5.1}$$

Where Ψ is the $N \times N$ basis matrix whose columns are the vectors $\{\psi_i\}$, and S is an $N \times 1$ vector. X is K-sparse if only K of the coefficients are nonzero and the other $(N - K)$ coefficients are zero. For compressive sensing, it is important to find a basis for the interested signal such that in that basis the signal is sparse and just a small number of its coefficients are nonzero. In that case the signal is called compressible.

The main idea of CS is that it acquires the compressed signal directly and does not go through the middle step for acquiring and saving N samples. Hence CS is supposed to recover signal $X \in \mathbb{R}^N$ from M measurements where $M << N$ and N is the original sample numbers. Assume a set of N linear combinations of the signal X as the measurement vector y which is represented by:

$$Y = \Phi X \tag{3.5.2}$$

where Φ is the $M \times N$ sensing matrix. If the sparsity basis is known, then by choosing sufficient values for M and Φ, it is possible to recover X from Y properly. One important issue is that the coherence between Φ and Ψ should be small.

By definition, the coherence between sensing basis Φ and representation basis Ψ

is denoted by:

$$\mu(\Phi, \Psi) = \sqrt{N} \cdot \max_{1<i,j<N} \| < \phi_i, \psi_j > \| \tag{3.5.3}$$

Therefore, in CS, both sparsity and incoherency are important to reconstruct the signal properly. It was shown in [CW08] that if M fulfills the following criteria then signal X can be reconstructed from Y properly with probability close to 1:

$$M \geq cK \log(N/K)$$

where c is a constant and K is the sparsity grade. It was shown that Φ could be chosen randomly.

From (0.16) and (0.17):

$$Y = \Phi\Psi S \tag{3.5.4}$$

It was mentioned that Φ is an $M \times N$ matrix and Ψ is an $N \times N$ matrix so, $\Phi\Psi$ is an $M \times N$ matrix and equation (3.5.2) is ill-posed. The value of S can be recovered from measurement vector Y using the ℓ_1-norm minimization:

$$\hat{S} = \arg\min_S \|S\|_1, \quad \text{subject to} \quad Y = \Phi\Psi S \tag{3.5.5}$$

Finally the original signal X can be recovered from equation (3.5.1).

3.6 Statistical Approach

The statistical approach reformulates inverse problems as problems of statistical inference using Bayesian statistics in which all quantities are specified as random variables. There are two groups of quantities: directly observable quantities, and unobservable quantities. These two quantities are dependent via the models. The purpose of the statistical approach is to take out information and evaluate the uncertainty about the variables (second group) based on the measurement information (first group) as well as information and models of the unknown quantities (second group) that are available prior to the measurement.

The uncertainty of observation is coded in the probability distributions of the quantities. In this approach, the solution to an inverse problem is the probability distribution of the quantity of interest, which is called the posterior probability distribution. This makes the statistical approach quite different from the classical approach discussed before. As we mentioned in the previous section, classical regularizers are used to produce a valid estimate of the quantities of interest according to the available data, but in the statistical approach the solution to an inverse problem is a probability distribution rather than a single estimate. Moreover, in the standard

regularization, a priori information restricts the solution space while in the statistical method, the a priori information is described in terms of the probability distribution [Idi08, KS05].

The fundamental rules of the statistical inversion approach are as follows:

- All variables contained in the model are represented by random variables.
- The amount of information relating to these variables is coded into their distributions.
- The solution of the inverse problem is represented by the posterior probability distribution.

Similar to the classical inversion methods, suppose we are interested in specifying an unobservable quantity $x \in \mathbb{R}^n$ which depends on an observable quantity $y \in \mathbb{R}^m$ in terms of a model f. We can define such system by:

$$y = f(x, e) \tag{3.6.1}$$

Where $f : \mathbb{R}^n \times \mathbb{R}^k \to \mathbb{R}^m$ and $e \in \mathbb{R}^k$ represents noise. According to the first rule of statistical inversion, all variables should be modelled as random variables. X, Y and E are the random variable representations of x, y and e respectively. Therefore, the relation between these random variables can be defined with respect to the equation (3.6.1) by:

$$Y = F(X, E) \tag{3.6.2}$$

The random variable Y is called the measurement, which is directly observable. The the random variable X, which is of primary interest but non-observable, is called the unknown. E represents the noise of the measurement and model.

In most practical cases like medical imaging, there is some information available about the unknown quantity of interest (e.g., we know the approximate location, size and shape of organs) before making the measurement. In the Bayesian approach this kind of information is coded into the probability density of the unknown quantity X, which is called the prior probability density and illustrated by $P(x)$.

From the product rule, the joint probability density and the conditional probability densities are related by:

$$p(x, y) = p(y|x)p(x) = p(x|y)p(y) \tag{3.6.3}$$

The conditional probability $p(y|x)$ is called the likelihood function which states the likelihood of different measurement resulting from $X = x$.

Y is the observable random variable so, actually, after performing the measurement we have $Y = y_{observed}$. The conditional probability distribution $p(x|y_{observed})$ is

the solution of the inverse problem and is called posterior distribution of X presented by:

$$p(x|y_{observed}) = \frac{p(x, y_{observed})}{p(y_{observed})} \tag{3.6.4}$$

The Bayes' theorem of inverse problem expresses that the posterior probability distribution of X, given the measurement $y_{observed}$ is:

$$p_{post}(x) = p(x|y_{observed}) = \frac{p(x)p(y_{observed}|x)}{p(y_{observed})} \tag{3.6.5}$$

In the following we substitute $y_{observed}$ by simply y because $y_{observed}$ is used to emphasize that for evaluating posterior probability density, the observed value of y is used.

In equation (3.6.3) the marginal density $p(y)$ acts like a normalizing constant.

$$p(y) = \int_{\mathbb{R}^n} p(x, y)dx \tag{3.6.6}$$

One of the challenges in this approach is to assign the a priori density $p(x)$ and the conditional density $p(y|x)$ such that they can reflect entirely the a priori information about the unknown quantity of interest and the interrelation between the unknown quantity and measurement respectively.

Making the likelihood function is usually more obvious compared to the construction of a prior function. One of the most frequent cases for likelihood construction is when the noise in the system can be modelled as additive and independent from the unknown quantity X. Hence the statistical model is

$$Y = f(X) + E \tag{3.6.7}$$

Where $X \in \mathbb{R}^n$, $Y, E \in \mathbb{R}^m$, and X and E are independent from each other. If we know the probability distribution of the noise E then, it is possible to calculate the likelihood function directly. When X and E are independent, the probability density of E does not change by fixing $X = x$. Therefore Y conditioned on $X = x$ has the same distribution as E so the likelihood function is:

$$p(y|x) = p_{noise}(y - f(x)) \tag{3.6.8}$$

When the unknown X and the noise E are dependent, it is a little bit more challenging because we need to have more information about the conditional densities.

The most challenging part in constructing the statistical model is making the a priori density. The reason is that generally we have qualitative a priori information

about the unknown while we need quantitative information to be coded into the a priori density. Gaussian priors are one of the most frequent probability densities used in practical situations.

When we are modelling the system, there are some variables which are neither observable, nor model variables of direct interest, but they usually give additional information about the model variables or the confidence we should consider in the estimates. They are conventionally called hyper parameters in most documents and denoted by θ. For example, if the signal production model includes multiple sources of noise, then the variance of each source of noise may be needed in the Bayesian approach, and knowing them tells us something about the predictability of the system, although not about the model variables we are trying to estimate.

Therefore it is necessary to generalize Bayes' theorem of inverse problem (3.6.5) as follows [Idi08]:

$$p(x|y,\theta) = \frac{p(x|\theta)p(y|x,\theta)}{p(y|\theta)} \tag{3.6.9}$$

And the normalizing constant is:

$$p(y|\theta) = \int p(y|x,\theta)p(x|\theta)dx$$

Another challenge in the Bayesian approach is how to determine the hyperparameters.

As mentioned, the solution of the inverse problem is defined by the posterior distribution. When the posterior distribution is known, it is possible to calculate both point estimates and interval estimates. The point estimates specify the most probable value of the unknown X when the observable data y and the prior information are given. Likewise, The interval estimate states an interval in which the value of the unknown X has a specific probability (95%) when the observable data Y and the a priori information are given.

One of the common point estimates is the maximum a posteriori estimate (MAP) which is defined by:

$$x_{Map} = \arg\max_{x\in\mathbb{R}^n} p(x|y) \tag{3.6.10}$$

A solution to this optimization problem may not exist or, if it exists, it could be non-unique. So the point estimate approach to inverse problem can be inadequate.

Another common point estimate method is the conditional mean estimate (CM) which is defined as:

$$x_{CM} = E\{x|y\} = \int_{\mathbb{R}^n} xp(x|y)\mathrm{d}x \tag{3.6.11}$$

The advantage of CM compared to MAP is that smoothness features of the posterior distribution are not so critical, especially when gradient-based optimization are used

to solve the optimization problem of MAP. The disadvantage of CM is that integrating over high dimensional space is more complicated than just applying usual quadrature methods.[KS05]

Maximum likelihood (ML) is the most popular point estimate method. The ML method estimates the value of the unknown which most likely produces the measured data. This is a statistical but non-Bayesian method and defined as:

$$x_{ML} = \arg\max_{x \in \mathbb{R}^n} p(y|x) \tag{3.6.12}$$

3.6.1 Hidden Markov Random Model

As our Bayesian regularizer was implemented for imaging problems in general and MR Imaging in particular, it is worthwhile to review the contribution of the Bayesian approach to signal and image reconstruction. Therefore, in this part, we want to introduce the Hidden Markov Random Field, which is the underlying probabilistic method in the statistical approach to signal and image restoration. It has also been used in other high-level vision processing such as object matching and recognition.

In MRF theory, it is possible to encode the spatial properties of an image using the relation between neighbouring pixels. For example in piecewise constant image, the neighbouring pixels should have the same intensities. This property can be achieved by mutual impact between neighbouring pixels using conditional distributions.

MRF theory provides a method to model the a priori probability of context-dependent patterns, such as the relationship between neighbouring pixels. MRF theory is commonly used together with statistical decision methods in order to express objective functions. The maximum a posteriori (MAP) probability criterion is often mixed with MRF. The MAP-MRF framework was recommended first by Geman and Geman (1984). [GG84]

In the MAP-MRF framework, the objective function is the joint posterior probability density related to the unknown quantity. There are three fundamental parts in MAP-MRF modelling which are:

- deriving the posterior probability distribution
- specifying the parameters in the posterior distribution
- designing optimization methods to find the maximum of the posterior distribution

Suppose $\mathcal{S} = \{1, 2, ..., N\}$ is the set of indices and X, Y are two random fields such that $\mathcal{L}$ and $\mathcal{D}$ are their state spaces respectively.

$$\forall i \in \mathcal{S} \Rightarrow X_i \in \mathcal{L} \text{ and } Y_i \in \mathcal{D}$$

Assume $\mathbf{x}$ and $\mathbf{y}$ specify configurations of X and Y respectively, and $\mathcal{X}, \mathcal{Y}$ are the set of all configurations such that

$$\mathcal{X} = \{\mathbf{x} = (x_1, ..., x_N) | x_i \in \mathcal{L}, i \in \mathcal{S}\} \text{ and } \mathcal{Y} = \{\mathbf{y} = (y_1, ..., y_N) | y_i \in \mathcal{D}, i \in \mathcal{S}\}$$

The conditional probability distribution of Y_i for a given value of $X_i = \ell$ follows:

$$p(y_i|\ell) = f(y_i; \theta_\ell), \quad \forall \ell \in \mathcal{L} \tag{3.6.13}$$

θ_l in the above equation is the set of all parameters. The important principle in MRF is that the indices in $\mathcal{S}$ are associated to each other through a neighbourhood system $\mathcal{N}$

$$\mathcal{N} = \{\mathcal{N}_i, i \in \mathcal{S}\}$$

where, $\mathcal{N}_i$ is the collection of all indices located in the neighbourhood of i with the following property:

$$i \notin \mathcal{N}_i \quad \text{and,} \tag{3.6.14}$$

$$i \in \mathcal{N}_j \Leftrightarrow j \in \mathcal{N}_i \tag{3.6.15}$$

Let X be a random field. If it satisfies the following criteria then it will be a MRF on $\mathcal{S}$ according to neighbourhood model $\mathcal{N}$ [ZBS01]

$$P(x) > 0, \forall x \in \mathcal{X}$$
$$P(x_i|x_{\mathcal{S}-\{i\}}) = P(x_i|x_{\mathcal{N}_i})$$

It is remarkable that a multidimensional neighbouring system can be defined using the multidimensional indices. This is why in the labelling problems $\mathcal{S}$ is defined as a set of sites.

Hammersley-Clifford theorem states that Markov random fields are equivalent to Gibbs fields, so they can be modelled by Gibbs distribution. Therefore

$$P(\mathbf{x}) = Z^{-1} \exp(-U(\mathbf{x})) \tag{3.6.16}$$

where $U(x)$ is called the energy function and Z is a constant for normalizing. The energy function is defined in terms of clique potentials where a clique $\mathcal{C}$ is a subset of $\mathcal{S}$ such that any pair of distinct indices in it are neighbours.

$$U(x) = \sum_{c \in \mathcal{C}} V_c(x) \tag{3.6.17}$$

The values of $V_c(x)$ rely on the arrangement of the cliques.

The HMRF has two main components: the hidden random field, and the observable random field.

The random field $X = \{X_i : i \in \mathcal{S}\}$ with state space $\mathcal{L}$ is the hidden random field which means its state is unobservable and it has a Gibbs distribution of the form (3.6.16). $Y = \{Y_i : i \in \mathcal{S}\}$ is the observable random field whose state space is $\mathcal{D}$.

For any configuration $\mathbf{x} \in \mathcal{X}$, the conditional probability for every Y_i has the form:

$$p(y_i|x_i) = f(y_i; \theta_{x_i}) \tag{3.6.18}$$

which is called the emission probability function.

The HMRF also has the conditional independence property. The conditional independence property states that for every configuration $\mathbf{x}$:

$$P(\mathbf{y}|\mathbf{x}) = \prod_{i \in \mathcal{S}} P(y_i|x_i) \tag{3.6.19}$$

Regarding the conditional independence property, the joint probability of (X, Y) can be written as

$$P(\mathbf{y}, \mathbf{x}) = P(\mathbf{y}|\mathbf{x})P(\mathbf{x}) = P(\mathbf{x}) \prod_{i \in \mathcal{S}} P(y_i|x_i) \tag{3.6.20}$$

The joint probability for any pair of (X_i, Y_i) for given $X_{\mathcal{N}_i}$ can be simplified based on the local characteristics of MRFs:

$$P(y_i, x_i|x_{\mathcal{N}_i}) = P(y_i|x_i)P(x_i|x_{\mathcal{N}_i}) \tag{3.6.21}$$

The parameter set θ is $\theta = \{\theta_\ell, \ell \in \mathcal{L}\}$. Hence, the conditional probability of Y_i on the parameter set θ and $X_{\mathcal{N}_i}$ is

$$p(y_i|x_{\mathcal{N}_i}, \theta) = \sum_{\ell \in \mathcal{L}} p(y_i, \ell|x_{\mathcal{N}_i}, \theta) = \sum_{\ell \in \mathcal{L}} f(y_i; \theta_l) p(\ell|x_{\mathcal{N}_i}) \tag{3.6.22}$$

If we assume the random variables X_i are independent of each other then for every $\ell \in \mathcal{L}$ and $i \in \mathcal{S}$, we have

$$P(\ell|x_{N_i}) = P(\ell) = w_\ell \tag{3.6.23}$$

Which doesn't depend on the index $i \in \mathcal{S}$ [ZBS01]. We also define ϕ as the model parameter set by:

$$\phi = \{w_l; \theta_l | \ell \in \mathcal{L}\} \tag{3.6.24}$$

One of the most common choice of the emission distribution is the Gaussian distribution, in which the observable random variables have the following probability

density:

$$p(Y_i = y|\phi) = \sum_{\ell \in \mathcal{L}} w_\ell \cdot g(y; \theta_\ell) \tag{3.6.25}$$

where $\theta_\ell = (\mu_\ell, \sigma_\ell)^T$ and

$$g(y, \theta_\ell) = \frac{1}{\sqrt{2\pi {\sigma_\ell}^2}} \exp\left(-\frac{(y-\mu_\ell)^2}{2{\sigma_\ell}^2}\right) \tag{3.6.26}$$

Therefore the Gaussian hidden Markov random field (GHMRF) can be clarified as

$$p(y_i|x_{\mathcal{N}_i}, \theta) = \sum_{\ell \in \mathcal{L}} g(y_i; \theta_\ell) p(\ell|x_{\mathcal{N}_i}) \tag{3.6.27}$$

The MRF-MAP method can be used for image classification [ZBS01] which means assigning a label from set $\mathcal{L}$ to each pixel (x_i) when the pixels are indexed over lattice $\mathcal{S}$ and each pixel is presented by an intensity value y_i from the set $\mathcal{D}$. Hence the solution $\hat{\mathbf{x}}$ is estimated by:

$$\hat{\mathbf{x}} = \arg\ \max_{x \in \mathcal{X}} \{P(\mathbf{y}|\mathbf{x}) P(\mathbf{x})\} \tag{3.6.28}$$

Since $\mathbf{x}$ is a realization of a HMRF, the prior probability has the Gibbs distribution. Also it was assumed that the emission distribution is a Gaussian one with parameter $\theta_i = \{\mu_\ell, \sigma_\ell\}$. For $x_i = \ell$,

$$p(y_i|x_i) = g(y_i; \theta_\ell) = \frac{1}{\sqrt{2\pi {\sigma_\ell}^2}} \exp\left(-\frac{(y_i-\mu_\ell)^2}{2{\sigma_\ell}^2}\right) \tag{3.6.29}$$

Therefore the joint likelihood probability with respect to (3.6.19) can be written as:

$$P(\mathbf{y}|\mathbf{x}) = \prod_{i \in \mathcal{S}} p(y_i|x_i) = \prod_{i \in \mathcal{S}} \left[\frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(y_i-\mu_{x_i})^2}{2{\sigma_{x_i}}^2} - \log \sigma_{x_i}\right)\right] \tag{3.6.30}$$

The joint likelihood function can be rewritten as:

$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z'} \exp\left(-U(\mathbf{y}|\mathbf{x})\right) \tag{3.6.31}$$

where the likelihood energy is

$$U(\mathbf{y}|\mathbf{x}) = \sum U(y_i|x_i) = \sum_{i \in \mathcal{S}} \left[\frac{(y_i-\mu_{x_i})^2}{2{\sigma_{x_i}}^2} + \log\left(\sigma_{x_i}\right)\right] \tag{3.6.32}$$

It was shown that the MAP estimation can be written as a minimization of the posterior energy [ZBS01]

$$\hat{\mathbf{x}} = \arg\ \min_{x \in \mathcal{X}} \{U(\mathbf{y}|\mathbf{x}) + U(\mathbf{x})\} \tag{3.6.33}$$

In addition to the unknown class labels $\mathbf{x}$, the parameter set $\theta = \{\theta_\ell : \ell \in \mathcal{L}\}$ is to be determined. The difficulty arises from the fact that these two unknowns are dependent. The Expectation Maximization (EM) algorithm is usually used to deal with this problem. The details of EM method is described in [ZBS01].

Finally, the iterative optimization methods (i.e. Expectation Maximization method) are used to find the optimal solution of equation (3.6.33).

3.7 Statistical Methods Versus Classical Methods

It was mentioned that in the statistical approach, the a priori information is represented in terms of a proper probability distribution, while in standard methods, the a priori information regulates the solution space. The solution of the statistical method, the posterior distribution, represents the likelihood of an estimated solution given the measurements (observations). Different estimators like MAP or CM can be used to solve the reconstruction problem. Therefore, a large class of solutions can be obtained using the statistical methods. In contrast, the statistical methods have an expensive computational cost compared to non-statistical methods. One reason for such complexity is the hyperparameter estimation.

Chapter 4

The New Regularization Method

As mentioned in the introduction, we have implemented a type system to capture the correctness of the user-defined forward problem. The forward problems of interest involve reconstructing unobservable object properties based on physical measurements, e.g., determining tissue density/composition from MRI or x-ray data. Because the data are physical measurements, there are equations from physics and chemistry describing the signal generation processes, and these define the forward model, with physical units attached to all parts of the system. Including the units explicitly in our modelling tool allows for automatic consistency checking, which can be surprisingly powerful, including the complete elimination of some classes of modelling error.

Adding regularization is necessary for many problems to overcome noisy data or ill-conditioned systems. Since regularizers were developed in an abstract mathematical context, they are not formulated with physical units or other type attributes which ensure correctness properties. Some models are derived entirely using probabilistic arguments, which seem to be the most promising method of either automatically deriving regularizers or being able to check for validity in some sense. As should be clear from the previous presentation, most regularization methods contain some dimensionless parameters, the interpretation of which depends on computational heuristics and not model properties. This is at odds with the general understanding that regularization is a way of incorporating a priori information about the model. We feel there are two things that need to be done to obtain meaningful correctness guarantees by type checking:

1. An algebraic encoding of the allowed steps in reasoning about probabilities (including approximations, use of the central limit theorem, etc.) and properties;

2. A filtering and modification of existing regularization methods to exclude methods that cannot be derived using the rules of probability we are able to encode.

After some thought, we expect that many classical regularizations will be filtered out by this process, but that most Bayesian (including HMM) models will be consistent with such a system. As an exercise in the feasibility of this approach, we develop a new regularizer based on observed tissue segmentation in medical images, together with an informal version of the type of argument we hope to have automatically applied in our future type system. We will also give examples of common regularizers whose statistical derivation is likely to be very difficult, or to result in "a priori" knowledge that would not be acceptable to end users (e.g., the radiologist) if presented in a form understandable to them.

4.1 Statistical Derivation

To test our regularization method independently of the forward model, we consider the identity problem defined by

$$\mathbf{g} = \mathbf{f} + \mathbf{e}, \tag{4.1.1}$$

where $\mathbf{g}$ is the set of measurements, i.e., $\{g_i\}$ represents the pixel intensities of the image contaminated by noise; $\mathbf{f}$ is the set of model variables $\{f_i\}$ representing the de-noised pixel intensities; and $\mathbf{e}$ represents the noise. In the statistical interpretation, G, F and E are vectors of random variables the realizations of which are $\mathbf{g}$, $\mathbf{f}$ and $\mathbf{e}$ respectively. Every element of those vectors is related to one pixel in the image.

According to the Bayes' theorem mentioned in the previous section, the posterior function, $P(\mathbf{f}|\mathbf{g})$ is

$$P(\mathbf{f}|\mathbf{g}) = \frac{P(\mathbf{f})P(\mathbf{g}|\mathbf{f})}{P(\mathbf{g})} \tag{4.1.2}$$

Using the MAP method, the estimated solution can be clarified by:

$$\hat{\mathbf{f}} = \arg\max_{\mathbf{f}} P(\mathbf{f}|\mathbf{g})$$

To find the estimated solution $\hat{\mathbf{f}}$, we first need to calculate both the a priori probability function $P(\mathbf{f})$ and the likelihood function $P(\mathbf{g}|\mathbf{f})$.

The likelihood function is obtained from the fit-to-data term and does not have anything to do with the regularization method. Assuming that the measurements g_i are conditionally independent, the likelihood function can be written as:

$$P(\mathbf{g}|\mathbf{f}) = \prod_i P(g_i|f_i) \tag{4.1.3}$$

In the MRI experiments, the noise **e** has the Gaussian distribution. Assuming mutual independence between **f** and **e**, we conclude that **g** conditioned on **f** is distributed like **e**. Hence, the likelihood function can be represented by:

$$P(\mathbf{g}|\mathbf{f}) = \prod_i \exp(-(f_i - g_i)^2/2\sigma_n^2) \tag{4.1.4}$$

Where, f_i is the de-noised value for ith pixel, g_i is the measurement of ith pixel, and σ_n is the standard deviation of the noise.

The a priori distribution function is obtained from the segmentation properties of the image. We focus on local tissue segmentation for two reasons. First, most tissues are localized (with the exception of a few tissues that are widespread like blood) and this justifies the modelling of a few tissues in a smaller neighbourhood. Second, machine imperfection causes slow intensity variation independent of tissue type. Hence for every pixel, we consider a neighbourhood represented by S. We specify all of the partitions of S into k sets $\{S_i\}_1^k$. Then, we assign a probability to each set S_i represented by P_i. Using this decomposition of S, we want to simulate the presence of k different tissues in the S. The decomposition of S into S_i's has the following properties:

$$S_1 \cup S_2 \cup \cdots \cup S_k = S$$
$$S_i \cap S_j = \emptyset \text{ for } i \neq j$$

where every S_i represents one tissue in the image; k corresponds to the number of tissues involved in the image. Therefore, it seems that the pixels contained in one subset S_i have the same value in the absence of noise. But there are some variations corresponding to the pixels in every S_i. We assumed that such variations have the Gaussian distribution. Hence, the probability density function for every pair of pixels (f', f'') related to one subset S_i is represented by:

$$\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{(f' - f'')^2}{2\sigma^2}\right) \tag{4.1.5}$$

Assume the random variables in F are independent. Therefore, the probability P_i which is assigned to S_i can be defined by:

$$\prod_{(f',f'')\in S_i} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{(f' - f'')^2}{2\sigma^2}\right) \tag{4.1.6}$$

Because we decomposed S into a set of subsets S_i, the probability related to S for

one decomposition represented by:

$$\prod_{i=1}^{k} \prod_{(f',f'')\in S_i} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{(f'-f'')^2}{2\sigma^2}\right) \tag{4.1.7}$$

It was mentioned that we consider all of the partitions of S into k subsets. Therefore, the whole probability related to a segment S can be represented by:

$$\sum_{\substack{\text{all}\\ \text{decompositions}\\ \text{of } S}} \prod_{i=1}^{k} \prod_{(f',f'')\in S_i} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{(f'-f'')^2}{2\sigma^2}\right) \tag{4.1.8}$$

Hence, the a priori information of the whole image can be obtained by:

$$P(\mathbf{f}) = \prod_{\substack{\text{all}\\ \text{segments}}} \sum_{\substack{\text{all}\\ \text{decompositions}\\ \text{of } S}} \prod_{i=1}^{k} \prod_{(f',f'')\in S_i} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{(f'-f'')^2}{2\sigma^2}\right) \tag{4.1.9}$$

As was mentioned before, $P(\mathbf{g})$ acts like a normalizing constant, so we can ignore it. Therefore,

$$\begin{aligned} \hat{\mathbf{f}} &= \arg\max_{\mathbf{f}} P(\mathbf{f})P(\mathbf{g}|\mathbf{f}) \\ &= \left(\prod_{i}^{n^2} \exp(-(f_i-g_i)^2/2\sigma_n^2)\right) \times \\ &\left(\prod_{\substack{\text{all}\\ \text{segments}}} \sum_{\substack{\text{all}\\ \text{decompositions}\\ \text{of } S}} \prod_{j=1}^{k} \prod_{f',f''\in S_j} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{(f'-f'')^2}{2\sigma^2}\right)\right) \end{aligned} \tag{4.1.10}$$

There are two parameters in the equation (4.1) in addition to the measurements and the unknown quantity of interest. σ_n is the standard deviation of noise in the measurements and it can be obtained from the K-space data in MRI experiments. σ is the standard deviation of the tissue variations which is smaller than the measurement noise. There are different ways to deal with σ including:

- Use hyperparameters to estimate the sigma values for different tissues
- Use average values by collecting images from people of the target group (healthy, young, children, etc.)

- Use the standard deviation of noise, because we know this is a very conservative value

4.2 Complexity of Our Model

It is worthwhile to discuss the complexity of the solution with respect to segmentation size. In equation (4.1), the first term (fit-to-data term) does not change by different segmentation styles, so there is no point in calculating its complexity, and we focus on the complexity of the second term. For the inner-most product we should consider all possible unordered pairs of one subset S_j. There are $m^2(m^2-1)/2$ potential pairs, which either appear in one of the products associated with a subset, or do not appear in any product. So the number of total pairs over which we take k products is less than $m^2(m^2-1)/2$ unless there is only one subset, in which case we have equality. The sum is responsible for all possible decompositions of a set of m^2 pixels into k subsets, which multiplies the complexity by k^{m^2}. Finally, the outer product is taken over all neighbourhood sets. At most, we can consider the set centred at each pixel sufficiently distant form the boundary. This is at most $n \times n$, the number of pixels in the image. Therefore, the overall complexity of solving equation (4.1) is at most

$$n^2 k^{m^2} \binom{m^2}{2} \tag{4.2.1}$$

times the cost of calculating the exponential of the squared difference.

In our implementation, we picked out $k = 2$ and $m = 3$ which makes sense because in a 3×3 box of pixels, it is less likely to have more than two tissue types involved.

The number of accepted partitions k depends on the segmentation size m. There are more tissue types involved in the bigger segments. We compare our 3×3 segmentation with the case that there is just one segment in the image in which case $m = n$. The ratio of complexity of $n \times n$ segmentation to the complexity of 3×3 segmentation is:

$$\frac{k^{n^2} \times n(n+1)}{(n-2)^2 \times 2^{m^2} \times m(m+1)} \tag{4.2.2}$$

Where, k is the number of tissue types included in the whole image. The complexity is dominated by the exponential term which is k^{m^2}. Therefore, the numerator is much larger than the denominator. Hence, our model has the best complexity among all other segmentation sizes.

For example, comparing the complexity of our 3×3 segmentation with another segmentation method in which $k = 2$ and $m = 4$ will show that with increasing the segmentation size by one, the complexity will be increased a lot. For 3×3 boxes, the

*	1	*
1	1	1
*	1	*

(a) pattern 1

*	0	0
1	1	1
*	1	*

(b) pattern 2

*	0	0
1	1	0
1	1	*

(c) pattern 3

1	0	1
1	1	1
1	1	1

(d) pattern 4

1	0	1
1	1	1
1	0	1

(e) pattern 5

Figure 4.1: Templates to generate allowed patterns.

total number of image segmentations is $(n-2)^2$ while, for 4×4 box, it is $(n-4)^2$. Therefore, the ratio of complexity of equation (4.1) w.r.t 4×4 segmentation and 3×3 segmentation is:

$$\frac{(n-4)^2 \times 2^{16} \times \binom{16}{2}}{(n-2)^2 \times 2^9 \times \binom{9}{2}} \simeq 5 \times 2^7 \tag{4.2.3}$$

It can be concluded that although, for bigger segmentation, there are fewer segments, the overall complexity will be increased.

4.3 Acceptable Patterns

When we decompose the 3×3 segments into 2 partitions, there are some partitions that are unlikely to happen in medical imaging. Therefore, we eliminated some of the decomposition patterns. The patterns were generated by the templates in Fig. 4.1, where the two tissues are represented by 0 and 1, $*$'s are replaced by 0 or 1, and non-symmetric templates are rotated by multiples of $pi/2$ or multiples of pi depending on the level of symmetry after replacing $*$'s.

As we mentioned earlier, we want to explain the informal version of the type of arguments required in the type system such that we can specify the correctness of the regularization part.

Going back over the reasoning and derivation, we can specify the information that the type system must carry.

In the derivation of equation (4.1.6), the type system needs to recognize whether those probabilities are independent, so multiplying those probabilities is a meaningful operation. In this case, the independence of such basic variables should be asserted by the user.

In the derivation of (4.1.7), the reasoning rule, which should be encoded in our type system, is that when we decompose a set of pixels S, the disjoint union subsets $\{S_i\}$ have independent probabilities, so the probability assigned to the set S is the product of the subsets' probabilities. Hence, the type system should derive the independence if it knows that the subsets $\{S_i\}$ are a disjoint union. Therefore, it needs to capture

set theory in our type system such that it can derive properties about probabilities of different events.

With respect to the equation (4.1.8), we can only meaningfully add probabilities if we know that the related events are mutually exclusive. Therefore, the type system should derive the mutual exclusivity property based on the fact that just one decomposition is possible at the time.

Chapter 5

Results

We implemented our method with respect to the objective function obtained in equation (4.1). We used the "AMPL" language [FGK02] to model our problem. We also wanted to test our method with a known high quality solver and we chose "Snopt" [GMS05]. The number of variables are limited in AMPL to 300. Therefore, for an $n \times n$ image with $n > 17$ we cannot solve our problem at one stage. To cope with this problem, we divided the image into some blocks and solved each block separately. We tested the performance for different block sizes, and the best performance obtained with 10×10 blocks. Hence, we divided the $n \times n$ image into 10×10 blocks and kept the middle 8×8 pixels of every block to get smooth results in the edges.

To recognize the effect of our regularization method to remove the noise from the measurement data, we used the ℓ_2 difference which is an objective numerical measurement. But the ℓ_2 difference cannot really capture one of the most interesting properties in imaging which is whether features of a certain size will be visible after de-noising. Therefore, we considered a test where there is a repeating pattern of features of the certain size.

In that test, we considered a 32×32 image which has 9 repeating features of pixels whose values are the same but different from the image background meaning that we have two different tissue types. We assigned 0 to the pixels in the background and 1 to the pixels of features. Hence all features represent the same tissue type. We considered different number of pixels per feature from 1 to 12 pixels. For each of these 12 models, we changed the values of noise and got the result with respect to different noise scales. The model corresponding to 32×32 image with 9 features of 1 and 12 pixels are represented in Fig. 5.1 and **??**

Table 5.1: Comparing the ℓ_2-norm of the remaining error in the optimized image with the ℓ_2-norm of the original noise

		Noise Scales(ns)													
		0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	1.3	1.4
	$\|g-m\|_2^2$	0.64	2.55	5.73	10.19	15.93	22.93	31.22	40.77	51.60	63.71	77.10	91.74	107.67	124.87
$\|f-m\|_2^2$	1pixel/feature	0.38	2.98	3.146	5.64	6.666	7.33	7.60	7.87	8.20	8.59	8.91	9.24	9.61	9.98
	2pixels/feature	0.48	4.97	4.75	7.41	8.55	10	13.06	14.26	14.52	14.76	14.94	15.28	15.64	15.99
	3pixels/feature	0.59	3.79	3.79	4.32	4.64	5.02	9.76	11.28	15.88	16.95	18.48	21.00	22.08	22.43
	4pixels/feature	0.02	0.08	0.17	0.31	0.48	0.70	0.96	1.30	1.83	6.61	12.75	15.94	19.01	21.85
	5pixels/feature	0.38	1.40	1.52	1.68	3.09	3.78	3.91	9.43	13.08	14.30	20.30	21.99	23.65	28.47
	6pixels/feature	0.022	0.087	0.19	0.35	0.54	0.78	1.07	1.42	1.86	3.19	11.02	21.35	23.99	26.87
	7pixels/feature	0.37	1.31	1.47	1.60	2.14	1.99	3.77	9.33	10.50	13.23	16.39	23.36	29.99	31.46
	8pixels/feature	0.02	0.08	0.18	0.32	0.50	0.72	0.99	1.31	1.84	3.19	13.28	19.24	34.18	38.44
	9pixels/feature	0.018	0.07	0.16	0.28	0.44	0.64	0.87	1.23	1.61	4.46	11.19	22.97	37.08	39.02
	10pixels/feature	0.02	0.08	0.19	0.33	0.52	0.75	1.03	2.13	2.60	5.15	20.52	33.40	45.38	50.53
	11pixels/feature	0.34	1.00	1.12	1.42	2.26	1.81	3.60	5.76	8.90	18.20	33.33	44.16	55.05	55.62
	12pixels/feature	0.021	0.086	0.19	0.34	0.55	0.79	1.09	2.24	2.84	9.90	26.73	47.56	55.95	59.77

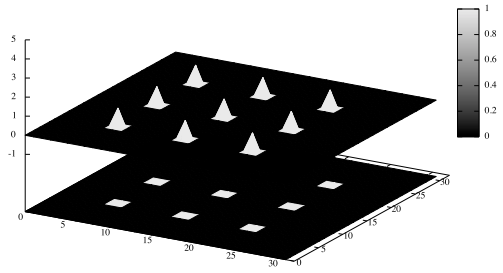

Figure 5.1: model-1pixel/feature-9features

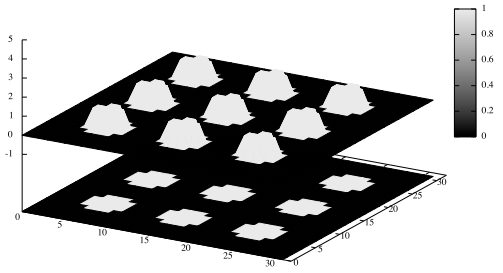

Figure 5.2: model-12pixel/feature-9features

We compared the ℓ_2-norm of the difference between the optimized result and the model with the ℓ_2-norm of the difference between the noisy measurement and the model. The model is represented by m, the noisy measurement is represented by g, and the optimized result is represented by f. In other words, $\|g-m\|_2^2$ denotes the ℓ_2-norm of the original noise, while $\|f-m\|_2^2$ denotes the ℓ_2-norm of the remaining error in the optimized image. The results are presented in Table 5.1. The noise scales are proportional to $1/SNR$ where SNR is the signal to noise ratio and $sn = 1.4$ is related to $SNR = 2.86$. Therefore, Noise Scale $\simeq 4/SNR$. The third row in that table illustrates the $\|g-m\|_2^2$ which is the same for different models (different number of pixels per feature) because it just represents the noise added to each image. Hence, for each model we can compare the corresponding ℓ_2-norm of remaining error in the optimized image, $\|f-m\|_2^2$, with the ℓ_2-norm of the original noise in the image $\|g-m\|_2^2$.

To have a better understanding of the ℓ_2-norm improvement related to the optimized images, we plotted the numerical results from Table 5.1 in Fig. 5.3. It is obvious from the ℓ_2-norm plot that most of the ℓ_2-norms of the remaining error in the

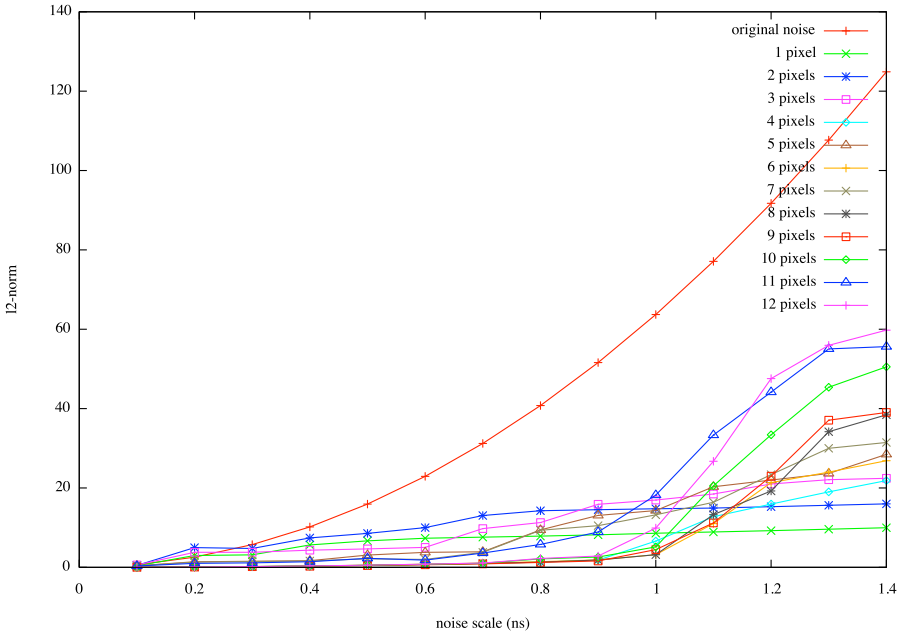

Figure 5.3: l2-norm of the remaining error and the original noise

optimized images are below the ℓ_2-norm of the original noise in the image. This illustrates the efficiency of our regularization method to remove the noise from the image data. For noise scale 0.2, the ℓ_2-norm of the error in several optimized images (e.g. 3pixels/feature) is greater than the original noise. Fig. 5.5 represents the optimized image for 3pixels/feature when $ns = 0.2$. According to Fig. 5.5, although we can distinguish 9 visible and bright spots in that image, the spots have a value less than 1 which increases the ℓ_2-norm of the error in the case of very low noise ($ns = 0.2$). It can be seen that the centre spot is brighter than those on the edges, which means centre one has a gray level close to 1 but the others have a smaller gray level. But all spots seem visible in the image.

As we mentioned earlier, just the ℓ_2-norm of error of the optimized image is not enough to specify the performance of our regularization method. Another standard factor is the visibility of some repeating features in the image which can be specified by counting the number of visible spots in our testing models. We plotted the number of visible spots for different numbers of pixels per feature with respect to different noise scales in Fig. 5.4. For counting the number of visible spots, we are strict in our interpretation of visibility meaning that in the case when 9 spots are visible, but some are considerably greyer than they should be, we do not count them. For example, for the optimized image with 2 pixels per feature and $sn = 0.7$ represented in Fig. 5.6, we just counted 1 visible spot. Although there are 9 detectable spots, 8 of them are

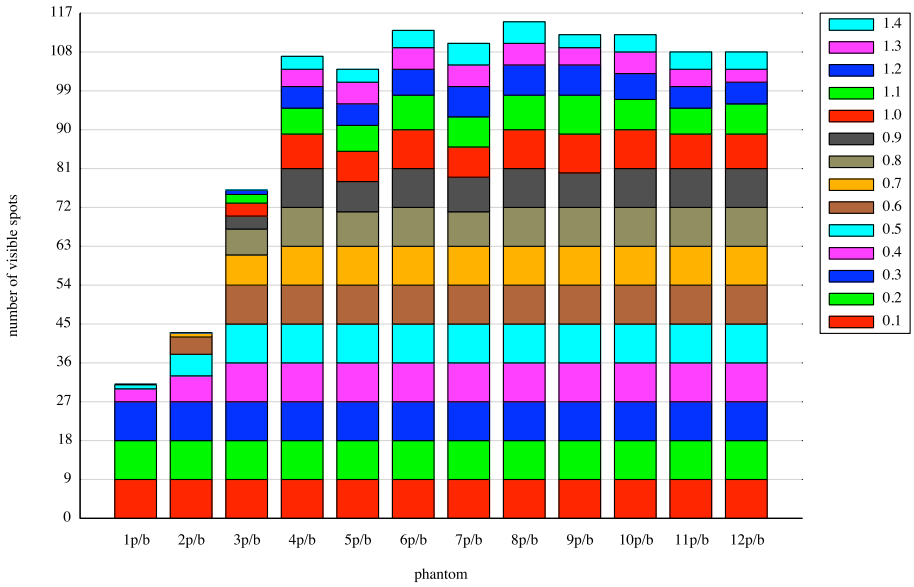

Figure 5.4: number of visible spots for different pixels/feature w.r.t noise scale

much darker than the remaining spot. We do not consider such spots in our count. This is a strict criterion, but we believe it is the correct criterion for images used in medical diagnoses by radiologists with many images to review every day.

In conclusion, we see that the ℓ_2-norm is an inadequate measure of regularization performance. In one case, it (correctly) reports a worsening of an image which is perfectly clear visually, and more seriously, it suggests that noise levels up to 1 (corresponding to a SNR of 4) are well-removed whereas the visual inspections show that a much lower threshold should be considered, with noise above 0.3 leading to inaccuracies for one or two pixels, noise above 0.6 leading to inaccuracies for three pixels, and noise above 0.7 leading to inaccuracies even for 7-pixel spots.

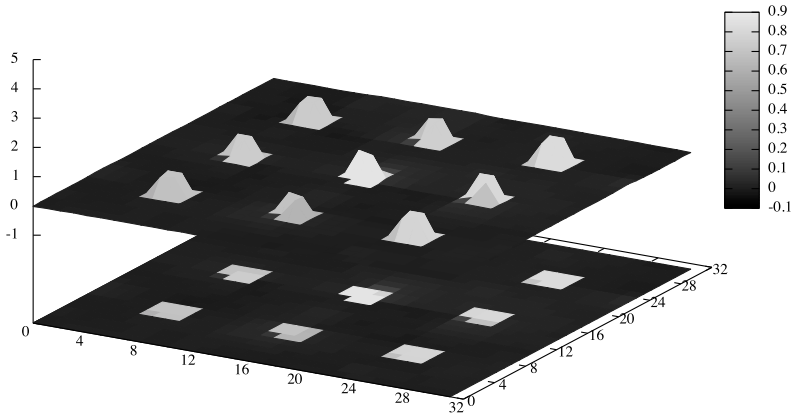

Figure 5.5: optimized image- 3pixels/feature- sn=0.2

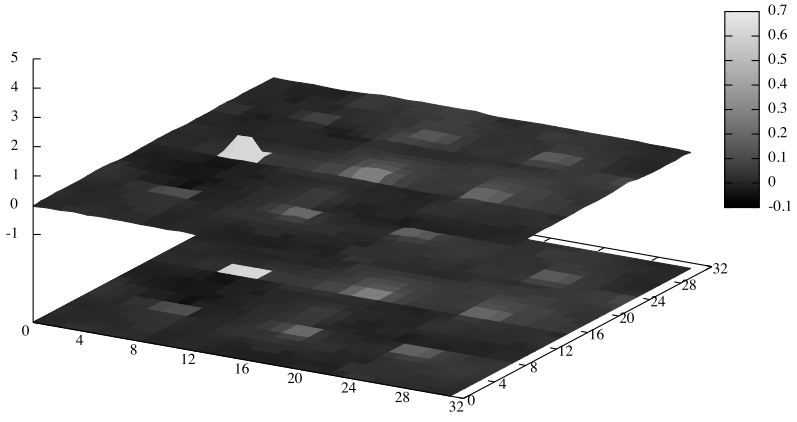

Figure 5.6: optimized image- 2pixels/feature- sn=0.7

Chapter 6

Conclusion

As we presented, we want to establish a type-system to verify the correctness of scientific software used in applications like medical imaging. We have shown that such a type system can be developed by encoding the most challenging aspects of MRI inverse problems in a system of types in the language Haskell, which is used both commercially and in academic research. Fully developing such a type system for general scientific applications is still a significant and intellectually challenging task. While it may be easier in a future more powerful programming language, based on our experience in this thesis, we would recommend this level of type safety for safety-critical scientific software with todays tools.

To formalize the frame of reference concept and Fourier Transformation properties, we extended existing approaches to type-level programming. We were able to develop a workable type system and capture all known application errors we can find in using the Discrete Fourier Transformation, which is the key concept in mathematical models of MRI reconstruction. Based on this, we have a high level of confidence that we can produce provably correct inverse solvers on the model side. Nonetheless, when we tried to formalize regularization, which is very important for advanced inverse problems, we found that the existing approaches do not seem to be compatible with the strong typing that we defined for the models. Among the existing methods, the statistical based HMM seemed the most amenable to formalization based on the probability theory. In analogy with the mathematical model which is derived from basic physical principles, we wanted to go from a small number of easy-to-justify assumptions and automatically generate the regularization from those assumptions. Hence, we have developed a new Bayesian regularization method which we think would be easier to formalize, but we have not done the formalization yet. We present a regularizer which is compatible with the theory, and is effective at denoising (the first test for regularizers in this area), but we think that the formulation can be further improved to give it competitive performance.

In the future, we hope this work will be extended to other types of inverse problems, that a formal theory of regularization be developed and encoded in a type system, and that competitive regularizers be developed which can be proven to meet transparent specifications.

Bibliography

[BDN09] Ana Bove, Peter Dybjer, and Ulf Norell. A brief overview of agda — a functional language with dependent types. In *Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics*, TPHOLs '09, pages 73–78, Berlin, Heidelberg, 2009. Springer-Verlag.

[CRT06] E.J. Candes, J. Romberg, and T. Tao. Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. *Information Theory, IEEE Transactions on*, 52(2):489 – 509, feb. 2006.

[CW08] E.J. Candes and M.B. Wakin. An introduction to compressive sampling. *Signal Processing Magazine, IEEE*, 25(2):21 –30, march 2008.

[Don06] D.L. Donoho. Compressed sensing. *Information Theory, IEEE Transactions on*, 52(4):1289 –1306, april 2006.

[FGK02] Robert Fourer, David M. Gay, and Brian W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming.* Nelson Education Ltd.; 2nd edition, 2002.

[GG84] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-6(6):721 –741, nov. 1984.

[GMS05] Philip E. Gill, Walter Murray, and Michael A. Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM Rev.*, 47(1):99–131, January 2005.

[GVL96] Gene H. Golub and Charles F. Van Loan. *Matrix computations.* Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, MD, third edition, 1996.

[Han87] Per C Hansen. The Truncated SVD as a Method for Regularization. *BIT Numerical Mathematics*, 27(4):534–553, 1987.

[Han98] P.C. Hansen. *Rank-deficient and discrete ill-posed problems: numerical aspects of linear inversion*. Number 4. Society for Industrial Mathematics, 1998.

[Han10] Per Christian Hansen. *Discrete Inverse Problems*. Society for Industrial and Applied Mathematics, Philadephia, PA, 2010.

[HHPJW07] Paul Hudak, John Hughes, Simon Peyton Jones, and Philip Wadler. A history of haskell: being lazy with class. In *Proceedings of the third ACM SIGPLAN conference on History of programming languages*, HOPL III, pages 12–1–12–55, New York, NY, USA, 2007. ACM.

[Idi08] J. Idier. *Bayesian approach to Inverse problems*. Wiley-ISTE, 2008.

[Jon00] Mark P. Jones. Type classes with functional dependencies. In *Proceedings of the 9th European Symposium on Programming Languages and Systems*, ESOP '00, pages 230–244, London, UK, UK, 2000. Springer-Verlag.

[Kir96] Andreas Kirsch. *An introduction to the mathematical theory of inverse problems*. Springer-Verlag New York, Inc., New York, NY, USA, 1996.

[Kis05] Oleg Kiselyov. Number-parameterized Types. *The Monad. Reader*, 5, 2005.

[KPcS10] Oleg Kiselyov, Simon Peyton, and Jones Chung chieh Shan. Fun with type functions, 2010.

[KS05] J. Kaipio and E. Somersalo. *Statistical and computational inverse problems*, volume 160. Springer Verlag, 2005.

[McB02] Conor McBride. Faking it simulating dependent types in haskell. *J. Funct. Program.*, 12(5):375–392, July 2002.

[Mon05] Roberto Janniel Lavarello Montero. Pulse-echo image formation using nonquadratic pulse-echo image formation using nonquadratic regularization with speckle-based images. Master's thesis, University of Illinois, 2005.

[Oka99] Chris Okasaki. From fast exponentiation to square matrices: an adventure in types. *SIGPLAN Not.*, 34(9):28–35, September 1999.

[ROF92] Leonid I. Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D*, pages 259 – 268, 1992.

[Skl88] Bernard Sklar. *Digital communications: fundamentals and applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[TAJ77] A.N. Tikhonov, V.I.A. Arsenin, and F. John. Solutions of ill-posed problems. 1977.

[Tik63] A. Tikhonov. Solution of incorrectly formulated problems and the regularization method. In *Soviet Math. Dokl.*, volume 5, page 1035, 1963.

[ZBS01] Yongyue Zhang, Michael Brady, and Stephen Smith. Segmentation of brain mr images through a hidden markov random field model and the expectation-maximization algorithm. *IEEE Transactions on Medical Imaging*, 20(1):45–57, 2001.