The first part of my internship included mainly developing routines in Matlab that were supposed to be used for demo purposes and for testing personal ideas but also those suggested by professor Hans Georg Feichtinger.

Because, in some cases, memory allocation was a delicate issue, the routines were developed in such a way that the running time would not be more than a few seconds.

But this was not possible in all cases for some routines mentioned in Chapter 3 such as bestnlatt.m, the running time for an input lattice of size 360 was close to 5 minutes. Also there were extreme cases in which the run time was over 3 and a half hours ( odermat.m for an input default sequence of 34 numbers between 200 and 1000).

This were isolated cases and, in most situations, the speed could be improved.

This report will be structured in 3 chapters each corresponding to a specific field in which I have worked under strict observation and close collaboration with professor Feichtinger for the past 4/5 weeks.

Chapter 1 - Function Spaces Visualization

The first week was allocated to finding new and smart ways of representing function spaces using MATLAB. This was one of topics presented by Professor Feichtinger in his lecture called Harmonic Analysis. Also some interesting topics related to Gabor Analysis, Short-Time Fourier Transform, Banach spaces, Banach-Gelfand- Triple, inclusions of Wiener spaces and Hausdorff- Young theorem were mentioned by the professor in this lecture for master students.

The first task was to develop some routine to create a representation for the unit Lp-ball.

The Lp spaces are function spaces defined using natural generalizations of p-norms for finite-dimensional vector spaces.

The p-norm of the vector \( x = [x_1, x_2, \ldots, x_n] \) is

\[
\|x\|_p := \sum_{i=0}^{\infty} |x_i|^p < \infty
\]

In order that a point \( z = x + iy \) (of coordinates \((x,y)\)) in the complex plane to be included in the Lp-ball, we must have that \( \|x\|_p := \sum_{i=0}^{\infty} |x_i|^p \leq 1 \) (where \( p \) is greater than 0).

L1 is a rotated square, L2 is the unit circle and Linf is a bigger square that includes in it the rotation of L1 (Hausdorff-Young theorem for \( p = 1 \)).

The routine lpball.m creates such a representation of a Lp space for a given \( p \) and also a given \( n \) number of points that compose the contour of the ball. With bigger \( n \), the shape of the ball is improving to the actual form (\( n \to \infty \)). This routine
creates the surface using points that are equidistant on the Lp ball (in sense of complex arguments).

The routine has a negative side: it is very slow for large values of n:  
\>
\texttt{tic \ lpball(2,1024); toc}

Elapsed time is 9.536579 seconds.

The faster version of this routine is \texttt{lpb.m} that creates the surfaces with points that are not equidistant. But this is not so important for demo purposes (for large n the difference between the actual plots produced by \texttt{lpb(p,n)} and \texttt{lpball(p,n)} cannot be seen).

The speed factor is very important (\texttt{lpb.m} is 1000 times faster than \texttt{lpball.m})

\>
\texttt{tic \ lpb(2,1024); toc}

Elapsed time is 0.008493 seconds.

This 2 routines (especially \texttt{lpb.m}) were used to develop further complicated routines used to display some interesting results from Harmonic Analysis:

1. Hausdorff-Young Theorem that states that the Fourier transform of the functions from Lp are included in Lp for see demolpf12.m
2. A Wiener space characterized by $p$ and $q$ is included in another Wiener space characterized by $p'$ and $q'$ iff $p > p'$ and $q > q'$. ($p, p', q, q'$ are from $1, 2, \infty$) see demowiener.m

3. The rotated Wiener space characterized by $p$ and $q$ is included in the Wiener space characterized by $q$ and $p$ where $\frac{1}{p} + \frac{1}{p'} = 1$ and $\frac{1}{q} + \frac{1}{q'} = 1$ for $p, q, p', q'$ from $1, 2, \infty$ see demowhy.m
Another interesting application was the improvement of the sospc.m ( sospcnew.m ) routine developed by professor Feichtinger into the more complete routine allspace.m that displays a variety of functional spaces (24 in total).

Also for demo purposes, I have developed some useful routine for displaying the intersection/union of the following geometrical shapes:

- ellipses (deformed and rotated circles) see interell.m
- rhombus (deformed squares) see lifli.m and ulifli.m
- hexagones see intoctwr.m and unoctwr.m

In creating the Wiener space representations for \( W(\text{LI}, \text{lt}) \), \( W(\text{LT}, \text{co}) \), \( W(\text{LT}, \text{li}) \) and \( W(\text{CO}, \text{lt}) \) I have developed a routine that can output the convex hull for the surfaces described by complex vectors see convhullc.m

Example: \( W(\text{LI}, \text{lt}) \) is the convex hull of the unit \( L_p \) balls for \( p = 1 \) (LI) and \( p=2 \) (LT) and \( W(\text{LT}, \text{li}) \) is the rotated version of \( W(\text{LI}, \text{lt}) \).

The construction idea for the Wiener spaces \( W(\ldots , \ldots) \) is the following:
Let \( W(\text{LP1}, \text{LP2}) \) be the space constructed from the 2 unit balls. It is locally as the \( L_{P1} \) space and globally as the \( L_{P2} \) space. The following picture shows the 9 Wiener spaces:
One fact that I think should be mentioned in the ending of this chapter is that in order to show multiple inclusions of the spaces involved (over 24 in total produced by routine such as allspace.m, sospcnew.m, wienspace.m and wienspace1.m), I have used deformed versions of the spaces such that a maximal number of inclusions were held.

The deformations were done using routines as dilcp.m and dilcompl.m which amplify/rotate the complex input vector with some coefficients and create a deformed version of it.

A downside is that I couldn't find the perfect coefficients such that all inclusions are satisfied.

Chapter 2 - Lattice approximation

I have found out that lattice manipulation is very important in this particular field of Gabor analysis using stft (Short-Time Fourier Transform) of a gaussian atom which represents the actual window for the transform which is moved in each point of the lattice (each non-zero entry of 0/1 matrix).

Professor put a strong emphasize on the lattice representations. He wanted an alternative representation to the one created by lattp.m lattp(n,a,b) is a nxn matrix of 0/1 that has 1 entries on the parallel time axes (Ox) with step size a and 1 entries on the parallel frequency axes (Oy) with step size b.

A natural alternative way to represent a lattice is using complex numbers represent the lattice as a matrix nxn with complex entries or as a 2x n matrix with real entries having the x coordinates on the first line and the y coordinates on the second line.

I have developed the routine lattnbh.m that can generate a lattice in both forms.
mentioned above and outputs the points in the standard way. A similar version of this routine is lattgen.m that can output the lattice points in the Matlab way: suppose we want to display a 3 x 3 lattice centered in 0 and composed out of (0,1,-1,1+i,-1+i,-1-i,1-i,-i) , the routine lattgen(1,1,1) will output the sequence (0,-i,i,-1,-1-i,-1+i,1,1-i,1+i). ( the input for lattgen/lattnbh permits us to choose the number of points on the vertical/horizontal direction as well as the step size i.e the distance between the points ).

The actual problem can be formulated in the following way:

We don’t want to use a regular nice lattice for the Short-Time Fourier Transform. Maybe we have as a input a sequence of points in the complex plane resulted from applying a deforming vector field to a regular lattice.

We want to have the best lattice approximation ( even if this is not a regular lattice ) for the jittered sequence of points resulted as mentioned above.

For this I suggested to produce a large number (?!lets say 200 ) of slightly deformed lattices from the initial regular lattice using dilcompl.m routine developed by my colleague Radu Frunza.

Because the vector field used should not jitter the lattice drastically , I figured that the optimal lattice approximation should be found among this 200 slightly jittered lattices.

The speed factor is not that great for this routine: > tic; testapp; toc Elapsed time is 4.913545 seconds.

Also the routine outputs a lattice that indeed approximates best the jittered sequence but only amongst the 200( 500,1000) possibilities. So the routine is also limited and slow.

I obtained an error around 0.3 for mild deformation which is not so great looking at the errors obtained by the routine lattpinv.m : 0.02 to 0.08 for mild deformation. The following picture shows the exact errors obtained:
Also the speed improvement needs to be mentioned: 0.005 to 0.01 seconds (so 500 times faster than lattapp.m).

This new better and faster routine was developed using an idea of professors Feichtinger: use the pinv (pseudo Inverse in the Moore-Penrose sense) method to find the best approximation: \( W = (Y \ast \text{pinv}(Z)) \ast Z \) where \( Y \) is the input sequence, \( Z \) is the regular lattice produced with lattnbh routine and \( W \) is the best approximation lattice.

I have used lattpinnv.m routine in developing a demo test routine - see testlatt.m This does not take any input it generates internally a jittered version of a regular lattice of size 5 x 5 (default value).

It uses lattjiter.m to create this jittered sequence of points (it simulates the action of the vector field mentioned above) and then it calls both approximating routines lattapp.m and lattpinnv.m and compares the results obtained (the errors obtained).

This test routine demonstrates the efficiency of the pinv method.

And finally, the most complete routine (that offers more input liberty) is bestapp.m that takes as an input a sequence of points in the matrix n x n form (n should be odd) and outputs the best lattice approximation using the lattpinnv.m routine also in the n x n form.
Chapter 3 - Generating Lattices

In this section of my internship, I focused mostly on producing new lattices from an initial regular lattice created by the command `lattp(n,a,b)`.

The actual procedure is not very complicated: just apply `sidedigm.m` routine over and over again to the initial lattice and thus obtain a (finite) group of lattices having the order an integer number that represents the least number $n$ such that applying `sidedigm.m` $n$ times we obtain the initial regular lattice.

The idea behind the `sidedigm.m` routine is the following: it creates a new lattice from the initial one $xpo = lattp(n,a,b)$ by placing the rows on the diagonals, the first row becomes the new main diagonal, the two parts of the second row that are above and below the initial main diagonal are now the next main diagonals and so on.

I have created the `testsdgm.m` routine that outputs the group of lattices produced as described above. The speed factor is not great because `sidedigm.m` turns up to be a not so fast routine for large input lattices we get that `testsdgm.m` is taking a lot more than expected:

```
> tic; testsdgxm(360, 18, 15); toc
Elapsed time is 8.726175 seconds.
```

So that's the reason for developing a new faster routine named `fastlatt.m` that produces the same output as `testsdgm.m` the idea was given by professor Feichtinger.

Instead of using `sidedigm.m` over and over again, I am using the indices of the nonzero entries of `sidedigm(xpo)` (using `find` command) and then apply once more `sidedigm` for the matrix `testmat(n)` thus only 2 uses of the `sidedigm.m` routine which turns up to be a not so spectacular improvement but still:
> tic; fastlatt(360, 18, 15); toc Elapsed time is 4.824583 seconds.

I have not figured out what is the relation between the number representing the order of the group of lattices created with testsdgm.m (fastlatt.m) and the actual size and redundancy of the initial lattice given by parameters n, a and b.

Other results that need to be mentioned are the routine(s) that calculate the merit numbers (conditional numbers) of all lattices created from initial regular lattices with a given redundancy: red = \( \frac{a}{ab} \) and it tells us how sparse is the initial lattice.

We are mainly interested in lattices with red in (1, 1.5) so that is why I have chosen for most routines as default lattice the lattice lattp(360, 18, 15) with red = 1.33.

An usual way to calculate the merit number is the following:

\[
\text{Merit} = \sum \left( \sum (\text{adjlat}(xpo) \ast gg) \right) \quad \text{where} \quad gg = g(:) \ast g \quad \text{and} \quad g \quad \text{is the gaussian of size} \quad n \quad (g = \text{gaussnk}(n)) \quad \text{also adjlat(xpo) represents the adjoint lattice of xpo.}
\]

As an example, \( \text{adjlat} \ (\text{lattp}(n,a,b)) = \text{lattp}(n,n/b,n/a) \).

This method was used in the routines sdgmmerit.m, plotmerit1.m, meritmat1.m and also bestlatt.m and bestnlatt.m.

The following image shows the merit numbers calculated using the plotmerit1 approach for the lattices that form the group (360, 18, 15):

But professor Feichtinger suggested also other methods for calculating the merit numbers and I have tried to implement some of his ideas:

The alternatives for plotmerit1.m are plotmerit2.m and plotmerit3.m. The actual calculation methods do not differ very much and the results are comparable up to a scaling factor. Please see the picture below produced using the comparem.m and plotord1.m routines:
Also the errors were calculated and we obtain that the plotmerit2.m and plotmerit3.m are the most “similar”:

Also, for each plotmerit.m routine, I have developed a so called meritmat.m routine that can output a matrix of merit numbers. More exactly, for a given n, we take all possible combinations \((d_1,d_2)\) of divisors of \(n\) (without 1 and \(n\)), and we form all groups generated from the regular lattice \((n,d_1,d_2)\).

For each lattice in a certain group, we calculate the merit numbers and then take the minimal number from the set created and so create a matrix of size \(\text{length(alldiv(n))} * \text{length(alldiv(n))}\) filled with minimal merit numbers. We display the output as a scaled image. The output is comparable for methods plotmerit1 and plotmerit2 but completely reversed for plotmerit3 as you can see in the following images:

Minimal merit numbers displayed using meritmat1.m:
Minimal merit numbers displayed using meritmat2.m:

Minimal merit numbers displayed using meritmat3.m:
The condmat.m routine does the same thing but with conditional numbers:

As a similar routine, I want to mention the ordermat.m routine that takes as an input a sequence of numbers and for each number it creates a matrix similar with the one described above, but instead of filling it with minimal merit numbers, we fill it with the orders of the groups created. The goal is to find groups with high orders. The record so far is order of 540 for the group (810,81,10). The picture below shows the distribution of order numbers for 6 values of n in the range (200,500):
I have used the plotord1.m routine to obtain this picture. It has a flexible input that allows displaying a variable number of subplots per plot. For this case I have chosen the format 2 plots per column and 3 per row. It could be case that only one colorbar is used for all subplots but in this case each subplot has its own colorbar just to demonstrate better the distribution of group orders.

In addition to the 3 plotmerit routines, I have tried 3 other methods that have one thing in common: they represent the errors obtained with different norm calculations (the regular norm and the sonorm) between the value of the standard gaussian dual athom calculated using the pinv method and the non-standard dual gaussian athom calculated with the gabdatr.m routine.

I have renamed these methods as the sonorm, norm and eye methods.

The results show that the norm and eye methods produce the most similar results. I have compared them using the gab3met.m routine. Please check the following images:
Also the errors were calculated:

There exists a standard method for calculating the conditional number which is cond.m:

- cond(G) where G represents the Gabor family starting with an atom g = gaussnk(n) and placing it in the non zero entries of the lattice xpo : G = gabbas(g,xpo).

Based on this procedures, I have developed the routines sdgmmerit.m and sdgmcnd.m that outputs a cell matrix that have as entries vectors of merit/conditional numbers for each group of lattices produced starting with all combinations of regular lattice that have a given redundancy red.

Also plotmerit1.m and plotcond1.m output and plots a vector of merit/conditional numbers from lattices created from one specific regular lattice.

Our goal is to search for lattices that have minimal merit/conditional number close to 1 such that the covering of the plane with circles (ellipses) is optimal; the circles are produced with imgc.m routine and represent centered versions of the stft's for each window. So that is why we are looking mainly for hexagonal lattices that, of course, have a small merit/conditional number.

In order to trace this optimal lattices from the big collection generated with tests-dgm.m or fastlatt.m, I have developed the routines bestlatt.m and bestnlatt.m.

The output is the first best or the best N (where N is an input argument) lattice(s) in the sense of minimal merit numbers from the collection of lattices that are produced from all initial ones of size n with a given redundancy red. The routines display the original version of the lattice(s) and also a smeared version that allows observing the tilt/pattern of the lattice(s).

In order to take a closer look to larger lattices (let's say size 360), I followed professors Feichtinger idea and developed a routine that displays one after the other each lattice produced with fastlatt.m accompanied by a plot representing a cut version of it.
The cutting is produced using a filter routine named fltc.m that takes the non-zero entries from a lattice within a given range determined by a fixed radius. I have used the kron.m routine to form a larger lattice using 4 copies of the initial lattice, and then look at the points from the center situated inside a circle of a radius that depends on the a and b coefficients: \( R = \text{round}(4 \cdot \text{coeff} \cdot \sqrt{ab}) \) - the value of the coeff variable is also not so clear. If the points form a regular hexagon, then we have an optimal lattice.

Tests were made using kronsix.m and they show that the optimal value of coeff varies from 1 to 3. Still it was not enough only to find this coefficient because this value offered the certainty that the number of filtered lattices with exactly 6 six points (7 with the origin). We had also extreme cases when these points were collinear.

In order to filter out these cases, I have designed the convcomp.m routine that can convert a index sequence representing the non-zero entries of a 0/1 lattice to a sequence of complex numbers that represents the exact copy of the lattice in the complex plane.

Having the complex numbers, we would just look at the arguments and select only the cases when all 6 points have different values of arguments (without the origin).

This procedure is done by the routine kronhex.m. Wanting to extend it to regular hexagon patterns, I developed the routine kronreg.m that selects only the cases when the points form a (close to) regular hexagon.

Another task was to find the regular lattices (of the form lattp(n,a,b)) within a certain group of size n. I have developed the lattreg.m routine that can output the regular lattices in 0/1 form, the a/b coefficients for these lattices and also whether or not the lattice lattp(n,b,a) is part of the group generated by lattp(n,a,b).

So far tests have proved that this is always true.

An important feature of a routine, apart from the run time, is the size of the
output. Because we were dealing with large values of $n$ (between 200 and 1000) it is obvious that storing hundreds of matrixes $n \times n$ with such values of $n$ was not the most optimal thing to do.

So, instead of storing the lattice as a 0/1 matrix, we used an alternative form of the lattice: just the sequence of indexes of the non-zero entries. For example, the length of this sequence for $\text{lattp}(n,a,b)$ is actually $\frac{n^2}{ab}$. Because we were interested in storing lattices with redundancy bigger than 1 (but smaller than 1.5) meaning that $\frac{n}{ab} > 1$, it was even profitable to store the indexes of the non-zero entries of the adjoint lattice instead of the lattice itself (ab entries instead of $\frac{n^2}{ab}$).

The routines fastlattnd.m and lattregnd.m are exact copies of fastlatt.m and lattreg.m, but they store groups of lattices not as collection of 0/1 matrixes but as one matrix that has on each row the non-zero entries of each lattice.

Because we are interested in specific redundancies for lattices (in the range $(1,1.5)$), I have adapted the bestnlatt.m routine into findred.m. It takes as an input the value of $n$, the minimal and maximal redundancy accepted and the desired number of first best lattice outputed. The output is a collection of best lattices with redundancies between redmin and redmax.

These few pages summarized my work at the NUHAG institute in the Summer of 2010. Most of the routines presented in this paper are uploaded to the Matlab Modules section in the database of the NUHAGs website. Also, some of the picture were as well uploaded to the Media section.