Documentation

Alex Elzenaar

December 17, 2019

Contents

1 Overview

The purpose of this MATLAB software package is to generate putatively optimal spherical (t, t)-designs [Wal18].

The package itself will only work on newer versions of MATLAB (it has been tested on R2018b and above); it depends on the following packages beyond those that are part of the 'standard' MATLAB release:-

- Statistics and Machine Learning Toolbox

In addition, should the user wish to use the `generate.py` Python script to create a nice index for their generated designs, they will require:

- The MATLAB Engine API for Python[1]

[1] This is included by default with MATLAB but must be specifically installed. Documentation may be found at the following URL: `https://au.mathworks.com/help/matlab/matlab-engine-for-python.html`

2 Software usage

In this section, we describe how the user can use the algorithm implementation to generate putatively optimal spherical designs.

MATLAB files are located in the `matlab/` subdirectory; all the scripts and procedures are implemented in MATLAB unless otherwise stated.

2.1 The tightframe method

The main method available is TIGHTFRAME; this method takes the parameters listed in table 1 and produces the tuple of return values listed in table 2. Broadly speaking, the purpose of this method is to produce a putatively optimal spherical (t, t)-design of n vectors in $\mathbb{C}^d$; the three parameters here (d, n, and t) are the most likely ones to be changed. The fourth parameter, k (the number of iterations) is also one that most users might wish to manipulate. The final parameters listed in the table allow the caller to modify and fine-tune specific parts of the algorithm, and will be described in a later section.

Example 1. The following command computes a putatively optimal $(2, 2)$-design consisting of four vectors in $\mathbb{C}^2$:-

```
[result, errors, totalBadness] =...
  tightframe(2, 4, 2, 5000, 100, 10, 5000, 1, 1);
```

The errors obtained over time may be plotted by MATLAB using the following commands:

```
t = 1:length(errors);
plot(t,errors);
hold on;
set(gca, 'YScale', 'log');
```

An example plot corresponding to example 1 is included as figure 1. The plot produced should resemble an exponential decay curve, because as the number of iterations increases the step size of the algorithm towards the variety of designs decreases proportionally.

The only return value of TIGHTFRAME which is obscure to understand is *totalBadness*; when this value is large compared to the total number of iterations, it means that the algorithm had trouble finding a design (in the sense that it took many tries to get a single step closer to the variety of designs). We shall make this more precise in a later section.

Example 2. Figure 2 shows a error plot of an example with extremely high badness (0.913); and figure 3 shows an example with medium badness (0.706).[2]

Note that in general a high badness corresponds to two characteristics of the error curve:

[2]Badness numbers will usually be quoted as proportions:- $totalBadness/k$ where k is the number of iterations completed.

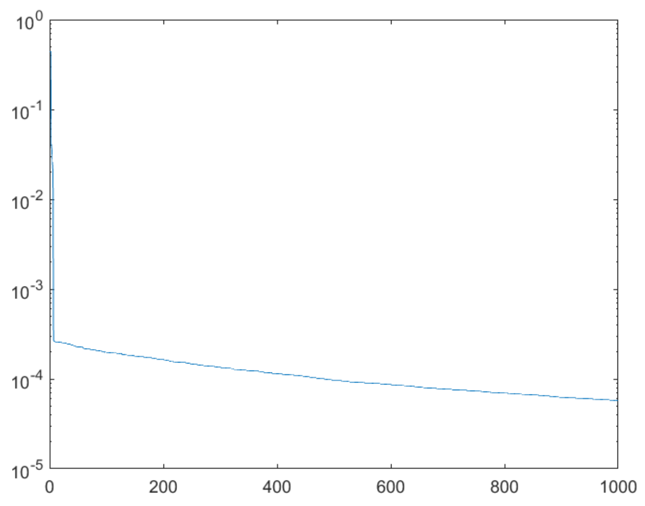

Figure 1: The error plot of a run with low badness.

- A very long, shallow tail (i.e. for large x-ordinates on the graph, $\frac{\mathrm{d}y}{\mathrm{d}x} \approx 0$). This is particularly noticable in figure 2.
- A very 'blocky' curve shape consisting of stretches of zero gradient interspersed with periods of extreme decline. This can be seen in figure 3.

The shallow tail corresponds to the fact that the algorithm decreases its step size according to the total badness (we shall discuss this in detail in a later section); the blockiness occurs because if it is 'hard' for the algorithm to find a better candidate for the design then the error will remain constant over long stretches.

Both of the graphs in this example were generated with $(d, n, t) = (3, 5, 3)$, and it is known that there is no such design (see, e.g. the list given in [Wal18, §6.16]); this explains why the asymptote of the error graph is non-zero (in this case, it appears to be ≈ 26).

2.2 The `runtf` script

The TIGHTFRAME method is useful if the user wants to script the algorithm (e.g. try to run it for lots of values of n to find the minimal n such that a (t, t)–design in $\mathbb{C}^d$ exists). However, if the user does not need this power they will probably want to use the wrapper script in `runtf.m`.

In order to change the design parameters (the same as those listed in table 1, the user should change the first few lines of `runtf.m`; then the script (when run) will produce the following:

- The console output from TIGHTFRAME in the console verbatim.

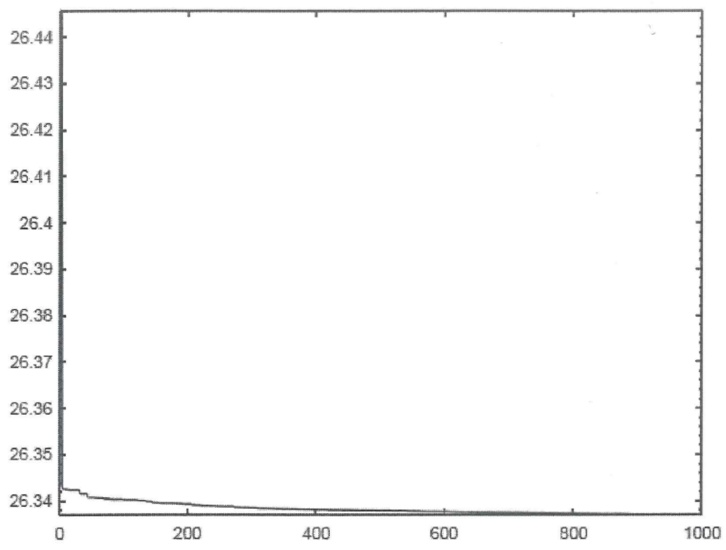

Figure 2: The error plot of a run with high badness.

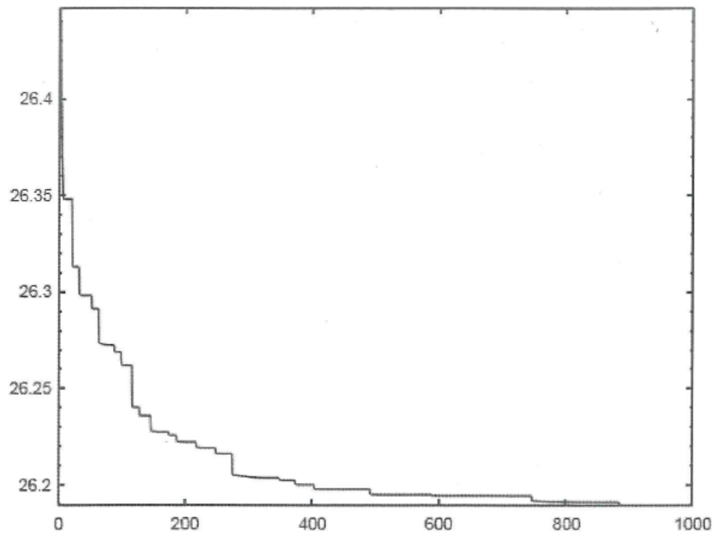

Figure 3: The error plot of a run with medium badness.

Parameter	Description	Sane value
d	Dimension of the space to embed the design into.	
n	Number of vectors in the design.	
t	Parameter of the (t, t)-design.	
k	Number of iterations of the algorithm to run before returning.	
s	Number of inital seed matrices to check.	1×10^6
b	Number of attempts at walking down the line of the gradient before falling back to looking at a ball.	10
ap	Number of times to run alternating projection on each iteration.	10 [a]
$errorMultiplier$	Scale factor for the walking distance at each iteration.	1×10^{-4} [b]
fd	MATLAB file descriptor for progress output.	1 [c]

[a] According to [Tro04, §7.2.7], increasing *ap* beyond around 5000 does not significantly change the results obtained. However, note that in that thesis the projection algorithm itself is the convergence procedure (as opposed to the current project, in which a separate convergence step is performed). Thus we only need to run the projection a small number of times (and in fact larger numbers here will cause the Gram matrix to drift away from the line of descent towards a low value of FP).

[b] If n is large, *errorMultiplier* should be made very small. For $n \leq 4$, *errorMultiplier* ≈ 1 is OK; then decrease by powers of 10 from there. If the number is too large, it is more likely that the algorithm will go too far past the variety of (t, t)-designs. If in doubt, set this small (especially if the badness proportion ends up high on test runs) and increase k.

[c] Set to '1' for console output; for file output set to `fopen('<filename>', 'w')`.

Table 1: Parameters for the TIGHTFRAME method.

Return value	Description
$result$	A $d \times n$ matrix consisting of n column vectors in $\mathbb{C}^d$ which is putatively optimal according to the error function.
$errors$	A $k \times 1$ matrix where the ith value is the minimum error of the best design found by iteration i.
$totalBadness$	The total number of times that the algorithm failed to improve the estimate by walking down the gradient. A more 'efficient' run minimises $totalBadness/k$ (where k is the number of iterations).

Table 2: Return values for the TIGHTFRAME method.

- A printout (to the default number of decimal places) of the returned $d \times n$ matrix *result*.
- A printout of the final error of that design (i.e. $\mathrm{FP}_{\mathbb{C}^d,n,t}(result)$).
- A printout of the overall badness proportion *totalBadness*/*k*.
- The parameters used for the generation of the design, together with the *result* matrix, the *totalBadness*, and the list of *errors* in the file `tf_run_YYYY-MM-DD-HH-MM-SS.mat` (in the current directory).
- A plot of the best error over time (displayed on screen as a MATLAB figure).

2.3 The compute3Products method

This method takes a single argument — a $d \times n$ matrix A — and computes the set of all n^3 3–products of the vectors of A, sorted according to the MATLAB SORT method.

2.4 The `generate.py` script

The purpose of this script, located in the top-level directory, is to take a directory of `.mat` files generated by the `runtf` method and produce a folder containing a nice HTML index of the generated designs.

The script takes one required argument — the directory in which to look for the `.mat` files. If the `-R` flag is specified, then the script will search recursively; otherwise it will search only the directory given and no subdirectories. For each `.mat` file it finds, the script will check whether it contains a variable called `result`. If it does, then it is copied to the output directory (default is `html/`, but this may be changed using the `-O` flag). An `index.html` file is also generated in the output directory containing a list of all the found designs, along with some parameters (d, n, t, and k from table 1, along with the error of the design found).

If the flag `-e` is set, the script will attempt to connect to a running MATLAB instance; otherwise it will try to start MATLAB itself. If connection to an existing MATLAB instance is needed, the user should note that some user interaction is required: a command needs to be run in the MATLAB console and then the Enter key needs to be pressed to make the script continue. The command needs only be run once for a given MATLAB instance.

The up-to-date usage information for the script may be viewed by running `python generate.py -h`.

3 Mathematical background

Let $\mathscr{H}$ be a finite-dimensional Hilbert space of dimension d (so, as a vector space, $\mathscr{H} \simeq K^d$ for some field K), and let $\mathscr{S}_{\mathscr{H}}(n)$ be the set of finite sequences of n elements of V with unit norm:

$$\mathscr{S}_{\mathscr{H}}(n) := \{(v_i)_{i=1}^n : \forall_i (v_i \in V \text{ and } \|v_i\| = 1)\}.$$

Suppose S is the unit sphere in $\mathscr{H}$, let σ be a normalised measure on S, and suppose we have picked an orthonormal basis $(w_i)_{i=1}^d$ on $\mathscr{H}$. Let π be the projection map from $\mathscr{H}$ onto the subspace spanned by w_1, and for each $t \in \mathbb{N}$ define the *bound weighting* $c_t(\mathscr{H})$ to be the (real) quantity

$$c_t(\mathscr{H}) = \left(\int_S \|\pi v\|^{2t} \, \mathrm{d}\sigma(v) \right)^{-1}.$$

(Remark: the quantity $c_t(\mathscr{H})$ is independent of the choice of basis.)

Finally, define the functions $\mathrm{FP}_{\mathscr{H},n,t} : \mathscr{S}_{\mathscr{H}}(n) \to \mathbb{R}$ by the rule

$$\mathrm{FP}_{\mathscr{H},n,t} : (v_i)_{i=1}^n \mapsto c_t(\mathscr{H}) \sum_{i=1}^n \sum_{j=1}^n \left| \langle v_i, v_j \rangle \right|^{2t} - n^2. \tag{1}$$

The value of FP for a given design is called the *error* of that design.

Define the set $\mathsf{SD}_{\mathscr{H}}(n,t) := \mathrm{FP}^{-1}_{\mathscr{H},n,t}(0)$; the elements of this set are called *spherical* (t,t)*-designs* of order n, embedded in $\mathscr{H}$. (These designs obviously also depend on σ, the spherical measure — but usually this comes naturally with $\mathscr{H}$.)

If $\mathscr{H} = \mathbb{R}^d$ or $\mathscr{H} = \mathbb{C}^d$ (with the usual inner product and with σ the normalised surface area measure of the sphere) then one can precalculate the following values [Wal18, p. 122]:-

$$c_t(\mathbb{R}^d) = \frac{d(d+2)\cdots(d+2(t-1))}{1 \cdot 3 \cdot 5 \cdots (2t-1)}, \qquad c_t(\mathbb{C}^d) = \binom{d+t-1}{t}.$$

The data of a design may be summarised by a $d \times n$ matrix V whose columns form the n column vectors of the design with respect to the basis (w_i). A frequently more useful representation, however, is the $n \times n$ *Gram matrix* of the design:

$$\Gamma((v_i)_{i=1}^d) = [V^* V] = [\langle v_j, v_i \rangle_{i,j}].$$

It can be shown that the Gram matrix Γ has the following properties:

(G1) Γ is Hermitian;

(G2) Γ has unit diagonal;

(G3) Γ is positive definite;

(G4) Γ has rank d;

(G5) Γ has trace n.

Note that (1) and (2) are conditions on the *entries* of Γ, while (3), (4), and (5) are conditions on the *spectrum* of Γ.

We may also find the gradient of $\mathrm{FP}_{\mathscr{H},n,t}$ with respect to the entries of the Gram matrix: we may rewrite FP in terms of Γ as

$$\mathrm{FP}_{(}\mathscr{H},n,t) : \Gamma \mapsto c_t(\mathscr{H}) \sum_{i=1}^{n} \sum_{j=1}^{n} \left|\Gamma_{j,i}\right|^{2t} - n^2$$

and taking the partial derivatives of this, we obtain

$$\frac{\partial\,\mathrm{FP}}{\partial \Gamma_{i,j}} = t\, c_t(\mathscr{H}) \left|\Gamma_{i,j}\right|^{2(t-1)} \overline{\Gamma_{i,j}};$$

thus to decrease the frame potential we must perturb the Gram matrix in the direction $-\nabla\,\mathrm{FP}$ where $(\nabla\,\mathrm{FP})_{i,j} = \frac{\partial\,\mathrm{FP}}{\partial \Gamma_{i,j}}$.

4 Algorithms implemented

In this section we will give pseudocode for the operations that are implemented by this package, along with explaining various design choices and limitations of the approach.

4.1 The high-level method

The high-level method is implemented in TIGHTFRAME, listed in algorithm 1. The algorithm as described here does not take the fd parameter listed in table 1, as this is only used to print status updates.

We begin by choosing a good starting Gram matrix (line 2). We generate s different matrices with the properties (G1)–(G5) (the 'seed' matrices), and pick the one which minimises the potential function to begin the iteration with.

The actual iteration begins on line 8; the variable $badCount$ holds the number of iterations since the current best matrix A was found, and if $badCount < b$ we attempt to walk down the gradient a small random amount. The mean distance to walk, $\frac{error \times errorMultiplier}{totalBadness+1}$, is proportional to $error$ (so when we are 'close' the distance we walk decreases), and inversely proportional to $totalBadness$ (so if we keep failing to find a good choice the step size decreases).

If $badCount$ exceeds the parameter b, we stop trying to walk down the gradient and instead we pick random matrices from the 'ball' B around A of mean distance $error \times errorMultiplier$. The idea here is that if the badCount is high then we keep trying and failing to walk along the line of steepest descent towards the variety of (t,t)–designs, and so the path of steepest descent must be twisting sharply away from the path it has been taking (figure 4). The amount we try to walk along the path of steepest descent keeps decreasing as the badness (i.e. failure rate) increases, but if the path is very twisty then this permanently makes the walk distance tiny and so the convergence rate will become very slow. Instead, if the $badCount$ becomes large, we instead look at the ball B (dotted

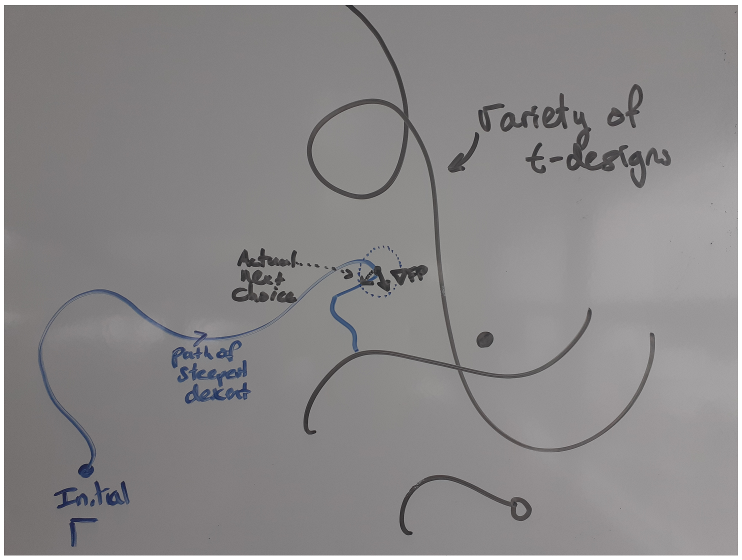

Figure 4: An intuitive depiction of the way counting 'badness' allows the algorithm to skip twists in the path of steepest descent.

blue) of (relatively) large radius, in the hope that we will hit upon the path of steepest descent again after it has rounded the corner, we will tend to skip part of the path and rejoin it on a straighter section (the dotted arrow on the figure) without having to drop our rate of walking too far. This can be seen in the console output: long runs of `*** Better found, ...` tend to accompany very small decreases of the error, while occasional blocks of `Nothing better` tend to be followed by a sharp decrease in the error. In order to take advantage of this, a healthy `badnessProportion` should be around 0.7.

In either case we have picked a new random candidate for Gram matrix near A; we call it A_{new}. Since the randomisation process does not guarantee that A_{new} is Hermitian, we project it onto the space of Hermitian matrices (line 17); then we attempt to project it onto the space of matrices satisfying the properties (G1)–(G5). This is done by the method ALTERNATINGPROJECTION described below; TIGHTFRAME passes into this method the parameter ap to modify the accuracy of the alternating projection algorithm, as well as the matrix A_{new} and the desired rank d.

Finally, we compute the error of the resulting matrix; if it is lower than the current best we set the $badCount$ back to zero; otherwise we increase both $totalBadness$ and $badCount$.

When the loop is done, we are left with a Gram matrix A satisfying (G1)–(G5). The final diagonalisation process undoes the map $V \mapsto \Gamma = V * V$ which produces the Gram matrix from the initial vectors. The resulting $n \times n$ matrix will have $n - d$ zero rows (by consideration of the rank of A) and so we project down to $\mathbb{C}^d$ in the natural way to produce the final matrix $result$ which is returned.

5 Further work to be done

- Allow users to pass in an error function to be minimised.
- Add support for real designs.
- Save all generated designs in a database.
- Implement an algorithm to partition generated designs into unitary equivalence classes.

References

[Tro04] Joel Aaron Tropp. “Topics in sparse approximation”. PhD thesis. The University of Texas at Austin, 2004.

[Wal18] Shayne F. D. Waldron. *An introduction to finite tight frames.* Applied and Numerical Harmonic Analysis. Springer Science+Business Media, LLC, 2018. ISBN: 978-0-8176-4815-2.

Algorithm 1 The high-level method

```
procedure TIGHTFRAME($d, n, t, k, s, b, ap, errorMultiplier$)
    Find a good starting matrix $A$ which satisfies properties (G1)–(G5) by checking $s$ different matrices, $A_1, ..., A_s$ and calculating $\mathrm{FP}_{\mathbb{C}^d,n,t}(A_i)$ for each.

    $error \leftarrow \mathrm{FP}_{\mathbb{C}^d,n,t}(A)$
    $errors \leftarrow []$
    $badCount \leftarrow 0$
    $totalBadness \leftarrow 0$
    $h \leftarrow 1$
    while $h \leq k$ do
        Append $error$ to $errors$.
        if $badCount < b$ then
            $\delta \leftarrow$ a random positive value near $\frac{error \times errorMultiplier}{totalBadness+1}$
            $A_{\mathrm{new}} \leftarrow A - \delta \nabla \mathrm{FP}_{\mathbb{C}^d,n,t}(A)$
        else
            $\Delta \leftarrow$ an $n \times n$ matrix of random values near $(error \times errorMultiplier)$
            $A_{\mathrm{new}} \leftarrow A + \Delta$
        end if
        $A_{\mathrm{new}} \leftarrow \frac{A_{\mathrm{new}} + A^*_{\mathrm{new}}}{2}$
        Project $A_{\mathrm{new}}$ onto the space of matrices satisfying (G1)-(G5).
        $error_{\mathrm{new}} \leftarrow \mathrm{FP}_{\mathbb{C}^d,n,t}(A_{\mathrm{new}})$
        if $error_{\mathrm{new}} < error$ then
            $A \leftarrow A_{\mathrm{new}}$
            $error \leftarrow error_{\mathrm{new}}$
            $badCount \leftarrow 0$
        else
            $badCount \leftarrow badCount + 1$
            $totalBadness \leftarrow totalBadness + 1$
        end if
    end while
    $U \leftarrow$ the $n \times n$ matrix of eigenvectors of $A$
    $D \leftarrow$ the diagonal matrix of corresponding eigenvalues
    $result \leftarrow D^{1/2}U$
    Delete the zero rows of $result$, producing a $d \times n$ matrix (by rank of $A$).
    Return $(result, errors, totalBadness)$.
end procedure
```