# *Homework 3*

1. Create problems and solutions on the course training wiki: `http://gragg.math.kth.se/sf2524/merge_group_pages2.php?name=97359`. At least one problem and solution has to be in block 3. Further problems and solutions can be in any block of the course. This task is optional but can increase your bonus. Individual task (do work in groups). See how the work influences your bonus points under `http://kth.instructure.com/courses/17791/pages/homework-slash-bonus-points-rules`.

   Course training wiki: `http://gragg.math.kth.se/sf2524/merge_group_pages2.php?name=97359`

2. FFT algorithm

   a) Carry out FFT (by hand) on the vector.

   In a), you may want to keep an eye out for any structures that can simplify the computation. It will be needed in problem d).

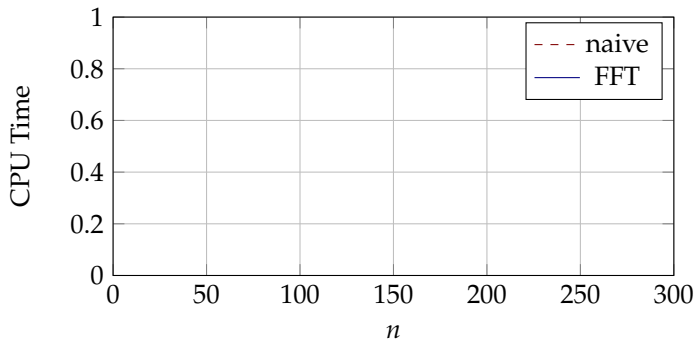
$$b = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 2 \end{bmatrix}$$

   b) Implement the recursive FFT algorithm (Algorithm 1 in block3.pdf or Algorithm 1.8.1 in Bjrck) and apply it to the vector

```
b=tan(2*(1:16)').
```

   c) Make a CPU-time comparison between your FFT-implementation and the naive matrix vector multiplication consisting of forming the entire $F_N$-matrix and computing `FN*b`. (Creating the matrix should not seen as a part of the computation.) Provide a figure like below. Use sizes `nv=2.^(1:p)` and increase $p$ until you see what you expect. Use a random $b$-vector.

   It is difficult to do good CPU-timing in MATLAB. Use the technique in: CANVAS → SF2526 → Pages → View all pages → CPU comparisons




d) Improve the FFT algorithm for vectors of the following form using what you may have observed in a):

$$b = \begin{bmatrix} y_1 \\ y_1 \\ y_2 \\ y_2 \\ \vdots \\ y_s \\ y_s \end{bmatrix}$$

where $y \in \mathbb{R}^s$ and $s = 2^{p-1}$. How much better is it in theory? Present one comparison simulation (or add another plot in figure in c).

3. Download the audio file from CANVAS course page

   `hw3_terrible_sound_with_hidden_message.ogg`.

   (Turn down the volume of your speaker before listening.) Use MATLAB commands `audioread` and `sound` to load and play the sound.

   (a) Use FFT and inverse FFT to remove (at least some) of the annoying sound, by setting some frequencies to zero. Hint: The noice frequencies are equidistant in the frequency (with distance 100 units). What is the hidden message?

   (b) With the naive DFT, how large problems can you solve on your computer (with say approx 30 min computation time)? Is it possible to solve the problem in this case?

4. Do "Homework 3 quiz: structured matrices" on CANVAS. This is mandatory and should be done individually (not in a group).

5. Download the files `build_bisection_tree.m` and `smatrix_skeleton.m`. from

   CANVAS SF2526 → Files → homework material

You will use these to construct your own matrix vector product.

(a) In the first lines of code in `smatrix_skeleton.m` a matrix is created. Why is this a semiseparable matrix?

S-matrix is the name of a special HODLR-matrix

In `smatrix_skeleton.m` there are some pieces of code missing. Fill this out. Convince yourself that you did it correctly by checking the naive matrix vector product computes the same.

(b) Move the first method to compute the matrix vector product and create a separate function in the file `matvec.m`:

```
function b=matvec(x,A,child_index,node_index_vector)
```

The file `matvec.m` should not include the generation of the index vectors, but take it as a parameter. How large is the difference between the naive matrix vector product and a call to `matvec`. Use the same vector `smatrix_skeleton.m`

In (c) we use the function eigs, which is an efficient implementation of Arnoldi's method, not covered in this course (see SF2524)

(c) When you use `eig`-function to compute the eigenvalues with smallest real part of a symmetric matrix, you can do so by only providing a matrix vector product procedure. The naive method to compute the

```
>> f_naive=@(x) A*x; % The matrix vector product
>> [V,D]=eigs(f_naive,n,3,'sr'); # compute three eigenvalues
```

The index vector (HODLR-matrix) generation in `build_bisection_tree` is not optimized and the timings should therefore not include this generation. In a practical situation the generation is often not dominating.

Make a matrix vector product function `f_smat`, based on a call to `atvec` (b) which only takes a vector `x` as input. Compare the timings of the naive matrix vector product and the improved matrix vector product. Generate the table

| $n$ | naive | smatrix |
|---|---|---|
| 600 | ?? | ?? |
| 800 | ?? | ?? |
| 1000 | ?? | ?? |

Depending on your computer you may need to adjust the $n$ values in the table. Make $n$ sufficiently large so you can make an empirical conclusion for when $n$ increases.