FELIS-MOR Documentation

Frederik Quetscher

I. MOR Stage 2

1) Affine decomposition for volumetric matrices+ports: The code is available in the commit 'MOR Stage 2+documentation' of 5.12.22. The code exploits the affine decomposition

$$A_{closed} = CC + \theta_1(f)M_E + \theta_2(f)M_C, \tag{1}$$

for the closed problem and the decomposition

$$P_{i,red} = U^\top P_i U = U^\top p_i p_i^\top U = (U^\top p_i)(U^\top p_i)^\top \tag{2}$$

for the ports. This reduces the number of operations from two matrix-matrix products to just two matrix-vector products and one outer vector-vector product with low dimensions

The reduced system matrix A_{red} is assembled as $A_{red} = A_{closed,red} + P_{red}$..

2) Affine decomposition for SibcE: Currently the computation of the SibcE is computed for each frequency point as follows:

1: Select a surface element
2: Calculate impedance on this surface (material s piecewise constant)
3: Calculate local matrix (by using the calculated impedance)
4: Transform local matrix to a global one
5: Add global element matrix to the global Sibc-Matrix

Instead I propose to create and store for each surface a global matrix for each surface. Then for each frequency, the surface matrices have to be multiplied with their corresponding impedance and added together. To me this seems more efficient and since the Sibc matrix is much sparser than the system-matrix, it should not be problematic for the memory. If only one resistive material and pec is used, then this decomposition consists of only one matrix and impedance. So it can be used for model problems without modifications.

3) other infos: The right hand side is taken directly from FELIS, this has to be improved in stage 3. The computation of the residual also exploits the shown decomposition of the port matrices.

The snapshots are placed equidistantly.

A. results

1) Square Cavity: The convergence is monitored for the residual, but also for the absolute error in the solution against a test set consisting of 250 equidistantly spaced solutions. The evaluation points are spaced according to a 1D latin hypercube (latin line?) to not be equal the training data for $m = 50$.

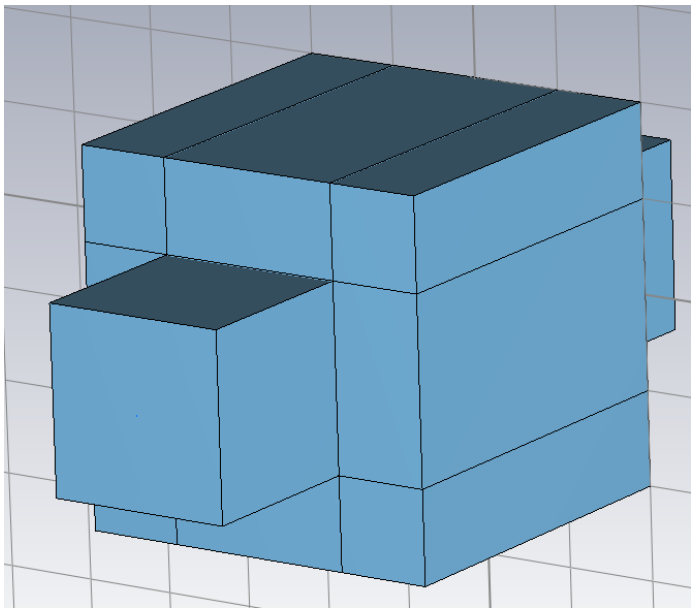

Fig. 1. Geometry of the square cavity.

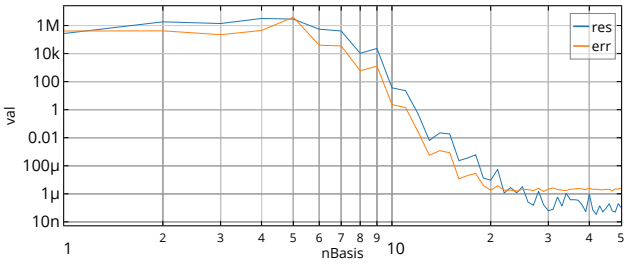

Fig. 2. The required number of snapshots is in the range of 5 to 20 for this example. There is a perfect agreement between the RHS residual and the error against the test-data. Hence both error metrics are viable.

.

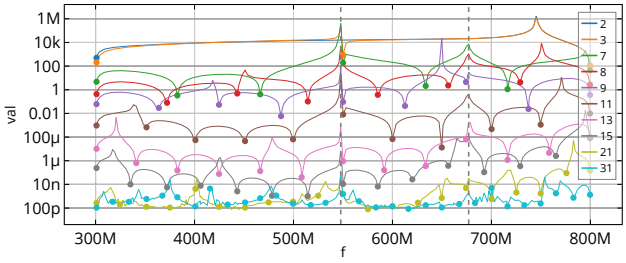

Fig. 3. The residual is almost frequency independent, hence an adaptive sweep would not improve convergence. The use of nested points would be more advantageous. An exception to this is the accuracy close to the resonances before convergence starts, there adaptive points may be beneficial.

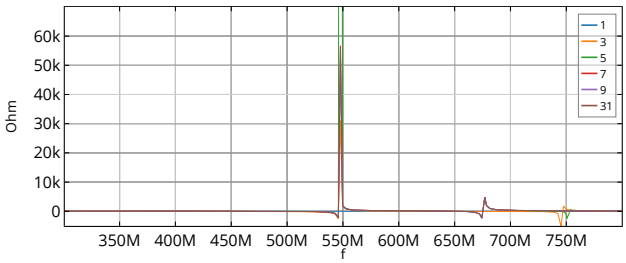

Fig. 4. Imaginary part of the impedance. The first resonance frequency is met to full visual accuracy at nB=3, the second at nB= 7. At nB=7 no further convergence is visible.

2) thoughts about stage3: Stage3 consists of the affine decomposition of the rhs. This serves the reason to calculate the residual faster. In python this may reduce the computational cost quite much, since it would make the calls of multiplyVecPortMat obsolete, which take up 11.5s of the total online time of 31s in Fig.8. A good part (6s) of this task are transpositions, which should not be costly in c++ . At this point, I assume that the residual computation should not be that costly anyways. Especially if only the beam is used as excitation the sparsity of the RHS could be exploited in the products, instead of searching for an affine decomposition.

update: The transpositions are basically free now, since they can be precomputed. Now the large number of matrix vector multiplications in the port-mode functions are the most costly, but this should be cheap in c++. Another expensive task is the computation of gamma, since the root has to be calculated. This could also be solved by precomputations.

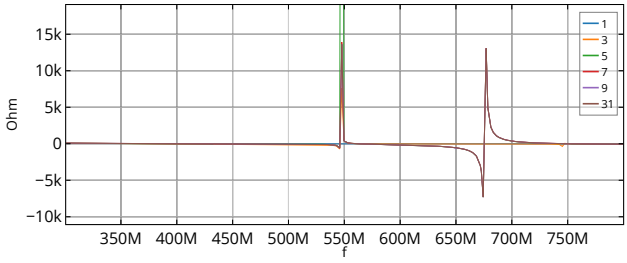

Fig. 5. Real part of the impedance. About the same observations regarding the convergence. Main observation: The real part does not agree with the FELIS code (Zre=0), but agrees with CST (if I remember correctly). Also if I calculate the impedance in Matlab using the field solution and current density exported from FELIS, I get this result. Is there maybe a mistake in FELIS? Update: Im python code war noch ein Fehler, jetzt könnten Felis und python übereinstimmen

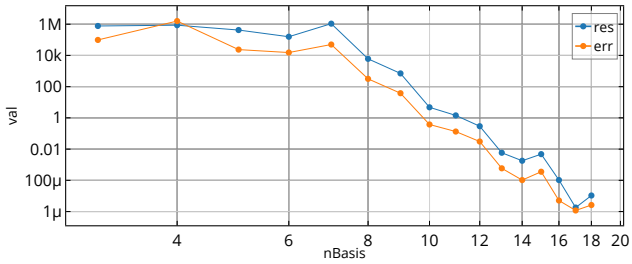

Fig. 6. Convergence if always the frequency with largest residual gets added. The convergence is slightly faster. Compare to linearly spaced problem, looking at err: For nB=8 both samplings lead to the same error. The adaptive sampling reaches at nB=16 the full accuracy, the linear one at nB=19. The adaptive sampling breaks down, because a previously chosen frequency point is chosen again.

3) Resistive Square Cavity: The entire surface has an impedance of 5.8e3, so the Sibc matrix comes in affine form. The convergence is about the same as for the PEC case.

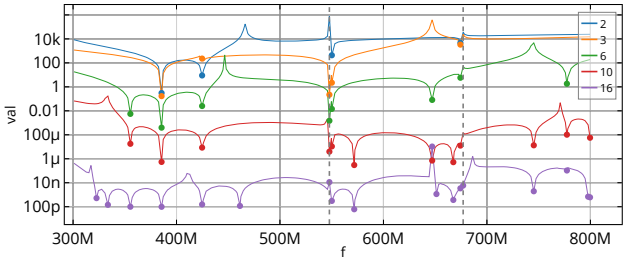

Fig. 7. Convergence of the residual over the frequency axis. The dots show the frequencies where the snapshots are taken. The algorithm breaks down, because the sample at around 660MHz is chosen again.
As visible: 1. the convergence is not uniform; 2. the spacing is surprisingly close to uniform; 3. almost always the residual is the lowest at the snapshots. Interestingly the technique of sampling at the highest residual is similar to an adaptive Gram-Schmidt method. A faster and more uniform convergence may be achieved, if the samples would be taken at the frequencies of highest expected improvement. This is actually the idea behind the SVD.

<lambda>	1.76 min	533
morOnlineSt2	31.33 s	16
getReducedPortMat	17.14 s	8000
multiplyVecPortMat	11.49 s	8000
__matmul__	4.03 s	172000
<method 'conj' of 'numpy....	2.46 s	164048
<built-in method numpy.z...	927.22 ms	429697
<built-in method builtins.l...	300.94 ms	2286076
norm	248.14 ms	12016
<method 'astype' of 'num...	182.65 ms	165049
solve	172.72 ms	4000
impedance	87.22 ms	4000
__rmatmul__	55.79 ms	48
<method 'append' of 'list' ...	28.10 ms	210135
<lambda>	22.65 ms	8000
<lambda>	8.93 ms	8000
<lambda>	1.74 ms	8000
toarray	10.64 s	80506
_find_and_load	2.62 s	1588
readModes	2.59 s	2
__init__	1.54 s	2
<built-in method numpy.zeros>	927.22 ms	429697
readMaps	456.64 ms	2
<built-in method builtins.len>	300.94 ms	2286076
norm	248.14 ms	12016
<method 'astype' of 'numpy.n...	182.65 ms	165049
svd	30.12 ms	16

Fig. 8. The total runtime of adaptive version. The meanings are:
1, lambda, reading the data from the drive, mainly the right hand sides;
2. morOnline, the major part of the code;
2.1+2.2, obviously the major parts of the runtime comes from operations involving the ports. This first due to the large number of calls (8 000) and their subfunctions (80 000), second, because in each call a sparse vector is cast into a dense one, and third, because surprisingly the transpositions are costly. I assume this should be much faster in c++;
2.3, matmul, part of 2.2;
3, toarray, part of 2.1;
svd: calculating the svds is super cheap, just 30ms in total.

.

4) Resistive Square Cavity Broadband:

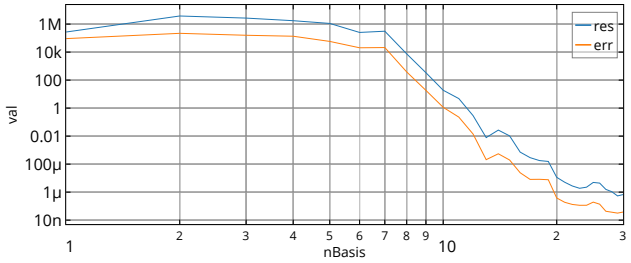

Fig. 9. Convergence for the resistive square cavity with linearly spaced points.

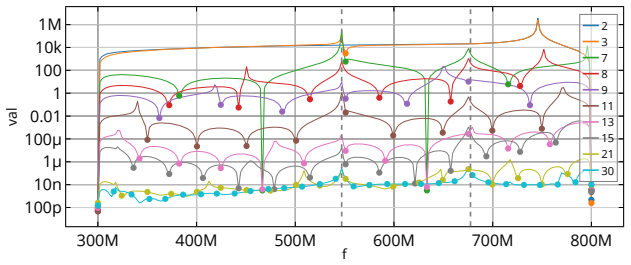

Fig. 10. Convergence for the resistive square cavity over frequency.

.

5) Constriction:

6) CubeWire: Convergence starts at 2nd basis function Convergence reaches arithmetic accuracy at 11-13 samples with 13 decimals between largest and smallest error/res fRange=0.2...1GHz

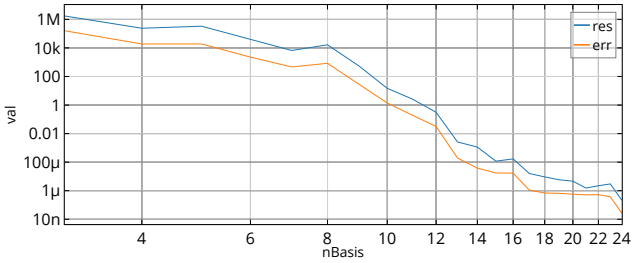

Fig. 11. Convergence for the resistive square cavity with adaptive points. The convergence is slightly faster.

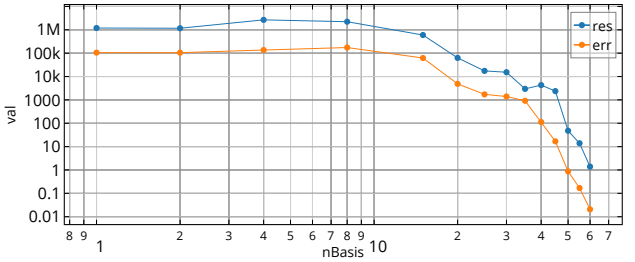

Fig. 12. Convergence for the resistive square cavity with linearly spaced points over a broad frequency range.

II. TIME ANALYSIS

This section is based on the commit ”Interactive with time measurement”.

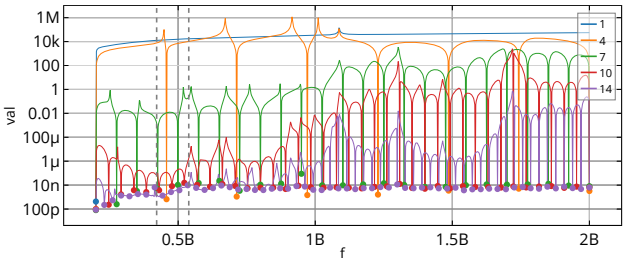

Fig. 13. Convergence for the resistive square cavity over frequency. Compared to the previous examples, this time the convergence is not uniform and significantly quicker in the lower frequencies. The numbers don’t equal the number of basis functions: 14=60

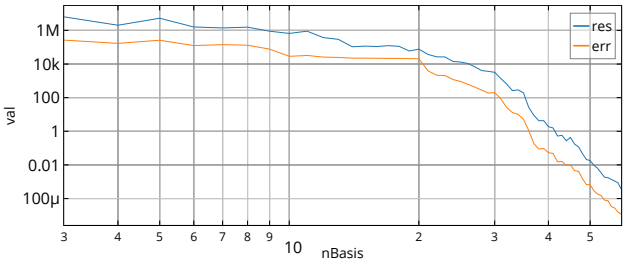

Fig. 14. Convergence for the resistive square cavity with adaptive points.

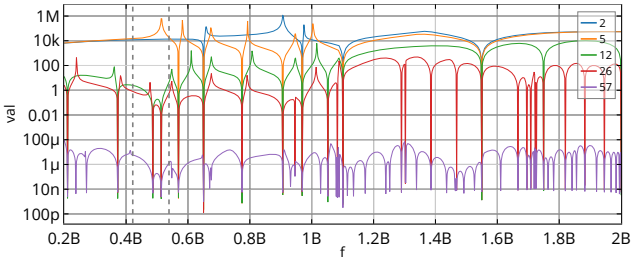

Fig. 15. Convergence for the resistive square cavity over frequency with adaptive sampling. Compared to the previous example, this time the convergence is uniform.

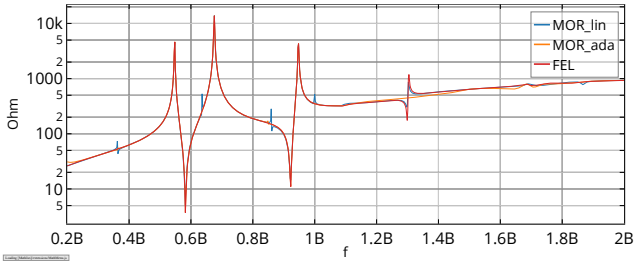

Fig. 16. Magnitude of the impedance for 18 basis functions. This number is chosen so that some error is visible. It is difficult to say whether the adaptive or non-adaptive version is better. The non-adaptive found all resonances, but has four spurious ones. The adaptive one has no spurious resonances but is missing the last resonance.

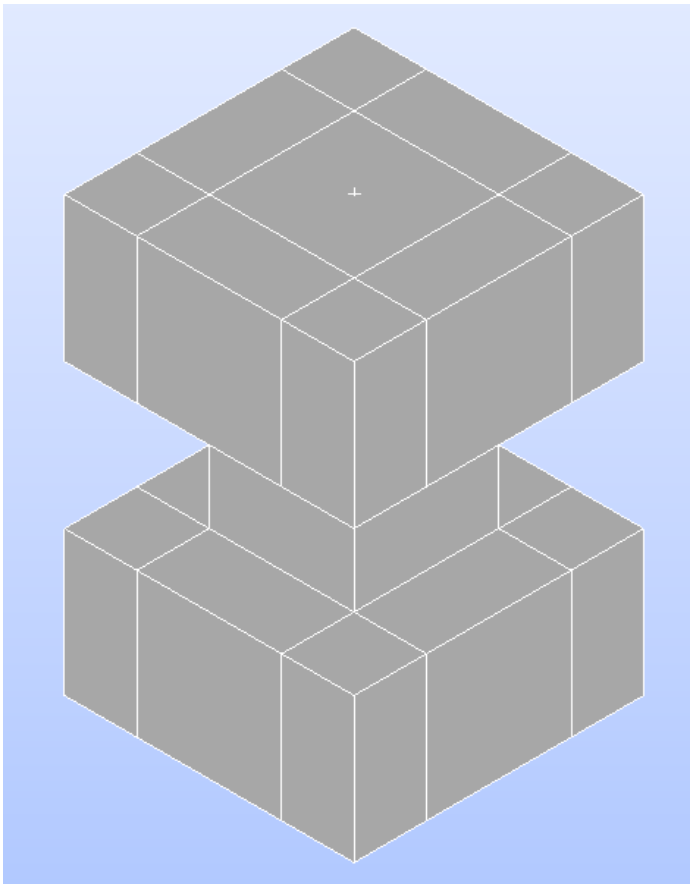

Fig. 17. Geometry of the constriction model. Surface impedance=5.8e6 S/m

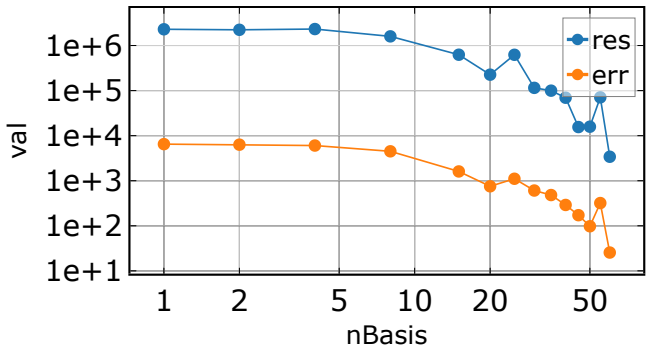

Fig. 18. Convergence for the non-adaptive code

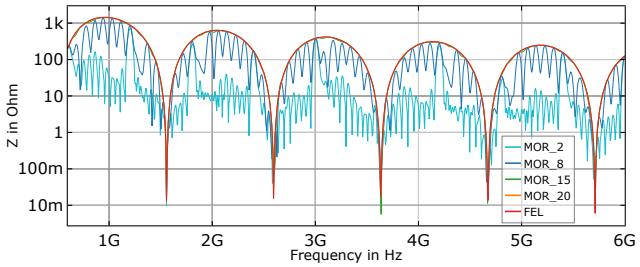

Fig. 19. Impedances.

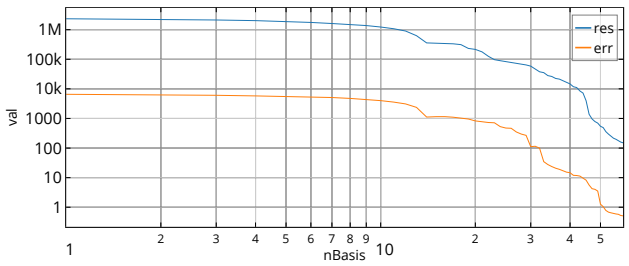

Fig. 20. Convergence for the adaptive code. The convergence is a siginificantly faster. The accuracy of the linear case with nB=50 is reached here with nB=30...35.

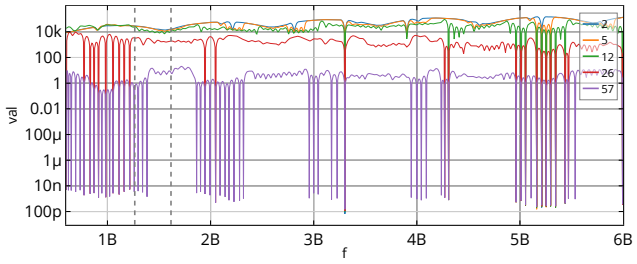

Fig. 21. convergence over frequency. Even though there are no samples where the impedance is low, there is also a good accuracy.

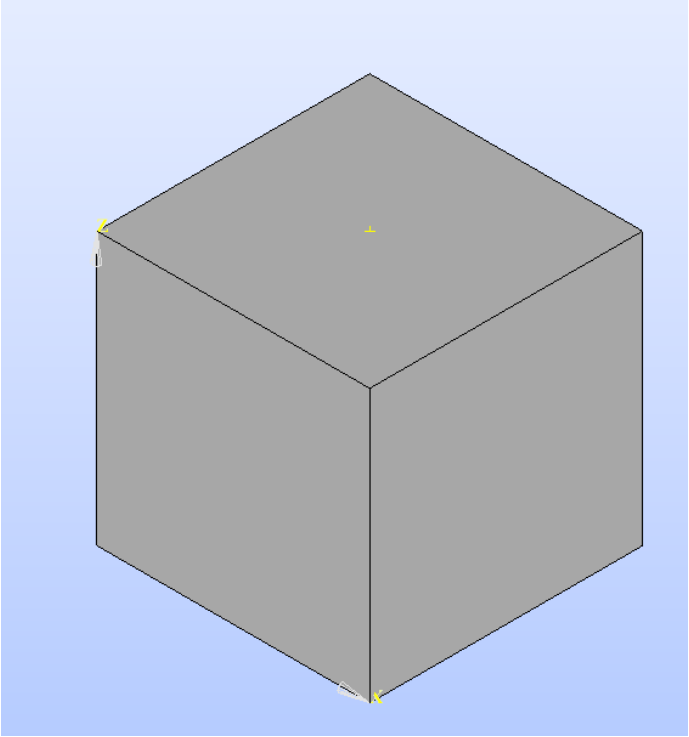

Fig. 22. Geometry. Surface conductivity=5.8e6 S/m

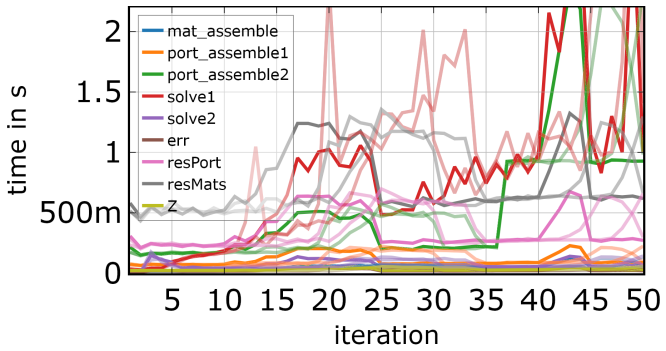

Fig. 23. Three runtimes for the Cubewire. The opaque lines show one run, the transparent ones two further ones. The increases at around iteration 20 and 40 are likely due to the cooling/turbo boost of the laptop, which can be confirmed by monitoring the task manager. Occasional spikes are distributed randomly, so likely caused by background processes. Important is the sudden increase of port-assemble2 at iteration 37 in all three runs. Also important is solve1, since it is the one which is increasing the quickest.

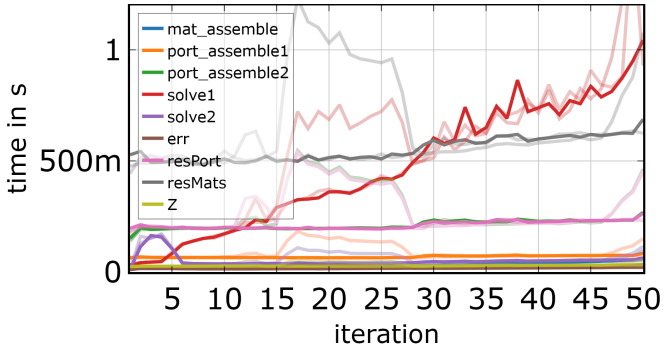

Fig. 24. Three runtimes for the square cavity. The opaque lines show one run, the transparent ones two further ones. Similar results as for the cube wire

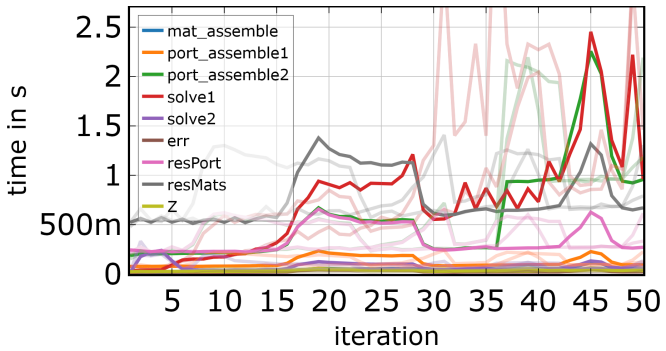

Fig. 25. Three runtimes for the constriction. The opaque lines show one run, the transparent ones two further ones. Similar results as for the cube wire. Here is also a sudden increase of port-assemble2 at iteration 37 in all three runs.