Herzlich Willkommen

Computational Aspects of Electrical Impedance Tomography
Prof. math. Andreas Stahel

Computational Aspects of Electrical Impedance Tomography, EIT

Dr. Andreas Stahel, BFH TI, HUCE

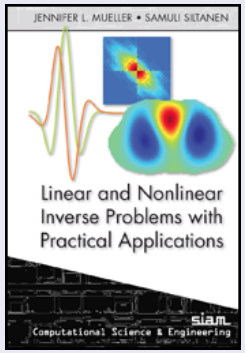
researchXchange Seminar

October 23, 2020

Collaborators and Resources

Collaborators:

- Jennifer Mueller, Colorado State University
- Melody Alsaker (Dodd), Gonzaga University, Washington



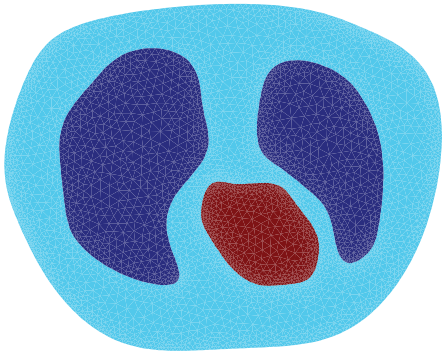
Resources:

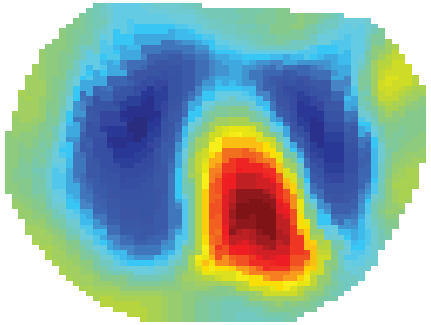
- Jennifer Mueller, Samuli Siltanen, *Linear and Nonlinear Inverse Problems with Practical Applications*, SIAM 2012
- Dodd, Mueller, *A Real-time D-bar Algorithm for 2-D Electrical Impedance Tomography Data*, Inverse Prob Imag, 8, 4, 2014
- K. Knudsen, J. Mueller, S. Siltanen, *Numerical Solution Method for the dbar-equation in the plane*, J. Comput. Phys. 198, 2004
- J. Mueller, S. Siltanen, *The D-bar method for electrical impedance tomography–demystified*, Inverse Problems, 36, 2020

What is EIT, Electrical Impedance Tomography

- Measure in the lab or hospital.

- Gain knowledge about the interior of the chest, e.g. to observe breathing in real time.





- EIT is an **illposed problem**, i.e. large change of conductivity might have very little influence on the boundary measurements.

EIT, the Dbar Method I

For a conductivity σ on a bounded domain $\Omega \subset \mathbb{R}^2$ consider the PDE

$$\nabla \cdot (\sigma \, \nabla \, u) = 0 \quad \text{in} \quad \Omega \subset \mathbb{R}^2$$

with the current density J given by Ohm's law $J = \sigma \cdot \nabla u$.
Apply a voltage u on the boundary and measure the resulting normal current density J on the boundary

$$J(z) = \sigma(z) \, \frac{\partial \, u(z)}{\partial n} \quad \text{for} \quad z \in \partial\Omega$$

to obtain the **Dirichlet to Neumann** map

$$\Lambda_\sigma \; : \; u \; \to \; \sigma \, \frac{\partial \, u}{\partial n} \quad \text{on} \quad \partial\Omega$$

also called **voltage to current density** map.

EIT, the Dbar Method II

The goal of EIT is to recover the conductivity σ from Λ_σ

- Good news: In 1980 Alberto Calderon proved that σ is completely determined by Λ_σ for the linearized case.
- Good news: In 1996 Adrian Nachman gave a constructive proof for the general case.
- Bad news: The EIT problem is extremely ill-posed, i.e. large variations in the conductivity σ might only have a minimal influence on Λ_σ. There is a simple analytical example by Giovanni Alessandrini.
- Good news: the problem can be overcome by regularization using scattering transforms, based on **CGO** solutions (Complex Geometrical Optics). This leads to the **Dbar** method for EIT.

The Equations (!with too many shortcuts!) I

Using the transformation $q = \frac{\Delta\sqrt{\sigma}}{\sqrt{\sigma}}$ and $\tilde{u} = \sqrt{\sigma} \cdot u$ one verifies

$$\nabla \cdot (\sigma \, \nabla u) = 0 \qquad \Longleftrightarrow \qquad \Delta \tilde{u} = q \cdot \tilde{u}$$

with the Schrödinger potential q .
With $k \in \mathbb{C} = \mathbb{R}^2$ a **CGO** solution $\psi(\cdot, k)$ solves the PDE

$$\Delta \, \psi(\cdot, k) = q(\cdot) \, \psi(\cdot, k)$$

with the asymptotic growth condition $\psi(z, k) \approx e^{i\,k\,z}$ as $|z| \to \infty$.
The function $m(z, k) = e^{-i\,k\,z} \, \psi(z, k)$ is bounded and solves

$$\left(-\Delta - 4\,i\,k\,\bar{\partial}\right) \, m(z, k) = -q(z) \, m(z, k)$$

Then $\sqrt{\sigma(z)} = m(z, 0)$, i.e. we have the conductivity.

The Equations (!with too many shortcuts!) II

With a fundamental solution g_k of (use $\bar\partial = \bar\partial_z = \frac{1}{2}\left(\frac{\partial}{\partial x} + i\,\frac{\partial}{\partial y}\right)$)

$$\left(-\Delta - 4\,i\,k\,\bar\partial\right) g_k(z) = \delta(z)$$

$m(z)$ is a solution of a Fredholm integral equation

$$m - 1 = -g_k * (q\,m) = -\iint\limits_{\mathbb{R}^2} g_k(z-w)\,q(w)\,m(w)\;dw\,.$$

Define the **scattering transform** by

$$\mathbf{t}(k) := \iint\limits_{\mathbb{R}^2} e^{-i\,2\,(-k_1 x + k_2 y)}\,q(z)\,m(z,k)\;dx\,dy\,.$$

Since m is asymptotically close to 1 this scattering transform is approximately the Fourier transform of q, evaluated at $(-2\,k_1, 2k_2)$.

$$\mathbf{t}(k) \approx \iint\limits_{\mathbb{R}^2} e^{-i\,2\,(-k_1 x + k_2 y)}\,q(z)\;dx\,dy$$

The Equations (!with too many shortcuts!) III

For differential images one (A. Nachmann 1996) can show

$$\begin{aligned}\mathbf{t}(k) &= \oint_{\partial\Omega} e^{+i\,\bar{k}\,\bar{z}}(\Lambda_\sigma - \Lambda_{\sigma_0})\,\psi(z,k)\;ds \\ &\approx \oint_{\partial\Omega} e^{+i\,\bar{k}\,\bar{z}}(\Lambda_\sigma - \Lambda_{\sigma_0})\,e^{+i\,k\,z}\;ds\,.\end{aligned}$$

As a **regularization method** the scattering transform $\mathbf{t}(k)$ is then restricted to $|k| \leq R$, i.e. a low pass filter.

- For very low noise levels and aiming for high resolution, choose R large.
- For high noise levels work with small values of R, suppressing noise, but giving up resolution.

The Equations (!with too many shortcuts!) IV

The function $m(z,k)$ solves the Dbar ($\bar\partial$) equation

$$\bar\partial_k\, m(z,k) = \frac{\mathbf{t}(k)}{4\,\pi\,\bar k}\; e^{-2\,i\,(-k_1,k_2)\,z}\;\overline{m(z,k)}$$

with the fundamental solution $\frac{1}{\pi\,k}$. For each point $z = x + i\,y \in \Omega$ the corresponding (complex) integral equation is given by

$$m(z,k) = 1 + \frac{1}{4\,\pi^2}\iint\limits_{|k'|<R} \frac{\mathbf{t}(k')}{(k-k')\,\overline{k'}}\; e^{i\,2\,(k_1' x - k_2' y)}\;\overline{m(z,k')}\;dk'\,.$$

Using a computational grid in $\mathbb{C} = \mathbb{R}^2$ this translates to a real, linear system.

$$(\mathbb{I} - \mathbf{A})\,\vec m = \vec 1$$

The Equations (!with too many shortcuts!) V

Use a $N \times N$ grid of data points in $\mathbb{R}^2 = \mathbb{C}$, here with $N = 64$. The 2D convolution has to be evaluated using 2D FFT and IFFT, not by constructing the (real) matrix $\mathbf{A} \in \mathbb{R}^{2N^2 \times 2N^2}$ and then a matrix multiplication.

flops for one matrix multiplication: $4\,N^4 \xrightarrow{\text{FFT}} N^2 \log N$

Thus we have to solve a system of the form

$$(\mathbb{I} - \mathbf{A})\,\vec{m} = \vec{1}$$

Then use $m(z,0) = \sqrt{\sigma(z)} - \sqrt{\sigma_0(z)}$, i.e. we have the conductivity at one point z.

The Equations (!with too many shortcuts!) VI

There is no need for the entries in the matrix $\mathbf{A}$, only multiplying a vector by the matrix is required. This can be performed by one 2D FFT and one IFFT on a $N \times N$ grid, $N = 64$, and some other, elementary operations.

Options to solve the linear system:

- not using 2D FFT would lead to horrendous computation times!
- direct solvers for $2N^2 \times 2N^2$ matrices require N^6 flops: too slow
- an iterative solver, e.g. GMRES: possible, used with success by Dodd&Mueller in 2014.

Observation: for differential images often only one or two GMRES iterations are required.

The Basic Algorithm

$$m(z,k) = 1 + \frac{1}{4\,\pi^2} \iint\limits_{|k'|<R} \frac{\mathbf{t}(k')}{(k-k')\,\overline{k'}}\, e^{i\,2\,(k_1' x - k_2' y)}\, \overline{m(z,k')}\, dk'$$

Set up the computational grid, e.g. select z points and k grid

Precompute values of $\frac{\exp(i\,2\,(k_1' x - k_2' y))}{\overline{k'}}$ for all $z = x + i\,y \in \Omega$ and k

for *each frame* **do**

  compute $\mathbf{t}(k')$, using $\Lambda_\sigma - \Lambda_{\sigma_0}$

  for *each value of* $z = x + i\,y$ **do**

    solve the integral equation $(\mathbb{I} - \mathbf{A})\,\vec{m} = \vec{1}$

    $\sqrt{\sigma(z)} - \sqrt{\sigma_0(z)} = m(z,0)$

  end

end

First Timing Results by using MATLAB

For this project many hosts were used, only two shown here:

- `hilbert`: Intel Xeon E5-1650, 6 cores, 12 threads, 3.5 GHz
- `karman`: AMD Ryzen 3950X, 16 cores, 32 threads, 3.5 GHz

A MATLAB implementation of the above algorithm was timed on `hilbert`. 600 frames with 6034 z values are used. The **for** loops are parallelized over frames. Using parallel threads over z values is slower.

Setup	time [sec]
MATLAB, with `for`, 1 thread	2770
MATLAB, with `parfor`, 6 threads	603
MATLAB, improved, with `parfor`, 6 threads	267

The starting point for computing time is thus 267 sec on `hilbert`.

Goal of this Project

- Develop a faster implementation of the Dbar algorithm.
- Real time evaluations asks for 10 to 20 frames per second. A moderate resolution is acceptable, e.g. 2000 z values.
- Run the parallel loop over the z points, not the frames.
- Take advantage of new, faster hardware:
  - Modern GPUs are very fast.
  - New CPUs can have many cores, at moderate cost.
- Use "standard" desktop hardware, i.e. no (expensive) servers.

Speedup by using a GPU? I

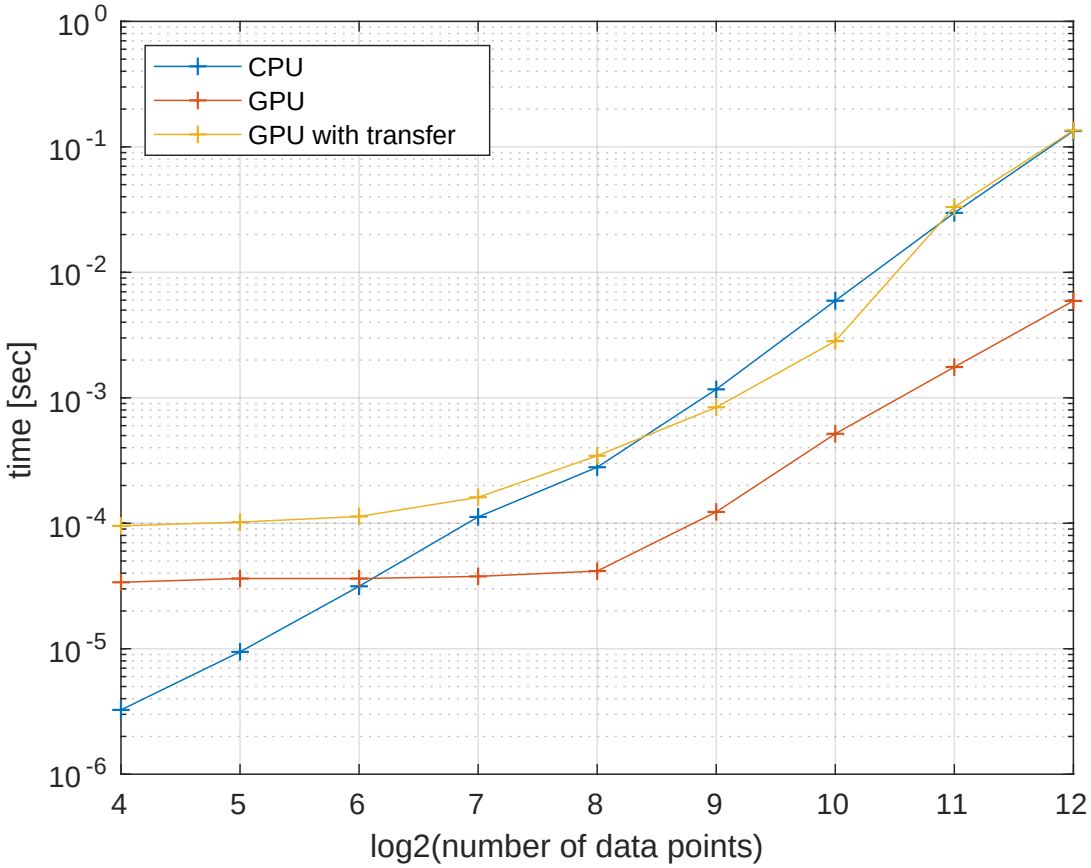
A good idea: use the huge computing power of a GPU.

With `MATLAB` determine the time required to compute a 2D FFT of a $N \times N$ matrix, using three different methods:

- Use the CPU only, i.e. all 6 cores on `hilbert`.
- Compute on the GPU only, i.e. do not move data between CPU and GPU.
- Compute on the GPU and move data between CPU and GPU.

The GPU used (GeForce GTX TITAN Black, 1.8 TFLOPS FP64) is rather powerful. The call of 2D FFT is examined, more coding will move the advantage towards the CPU only solution.

Speedup by using a GPU? II



Since we will have to use $N = 64 = 2^6$, there will be **no speedup by using the GPU**, tough luck.

General Goal of the New Implementation

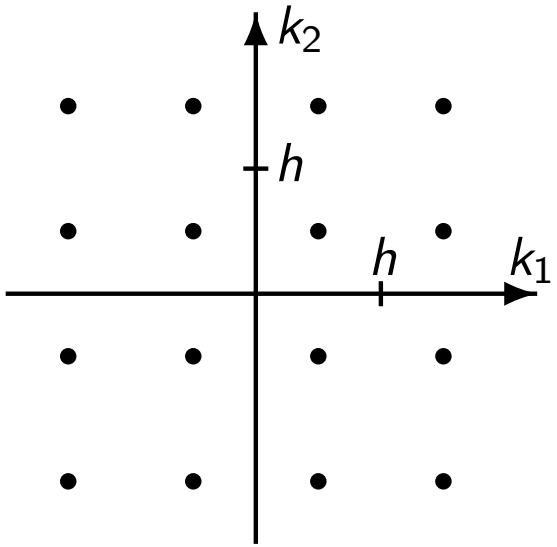
- Keep *Octave*/`MATLAB` for as many parts as possible.
- Select faster algorithms for some steps, if possible.
- Implement time critical parts in C, with (hopefully) faster code.
- Take advantage of the many cores available on new CPUs, e.g. the AMD Ryzen 3950X on the host `karman` has 16 true cores. With hyperthreading Linux sees 32 computing cores.
- To exchange data between *Octave*/`MATLAB` and C code use either files (slow) or shared memory (fast).
- KISS

2D Interpolation, Fast Implementation

- Since the Green's function contains $\frac{1}{|k|}$ the origin $(0,0)$ can not be part of the k grid.
- $\sigma(z) = m(z, \vec{0})$ requires an interpolation at $k = \vec{0}$.
- The calls of `interp2()` consumed almost half of the CPU time!
- Replace the call to `interp2()` by a weighted average.

Consequence: huge speed gain, i.e. 267 sec instead of 603 sec.

$$\frac{1}{256}\begin{bmatrix} 1 & -9 & -9 & 1 \\ -9 & 81 & 81 & -9 \\ -9 & 81 & 81 & -9 \\ 1 & -9 & -9 & 1 \end{bmatrix}$$



GMRES or Neumann Approximation

Solving $(\mathbb{I} - \mathbf{A})\,\vec{m} = \vec{1}$:

- Observation using GMRES: for differential images often only one or two iterations are required.
- The suspicion $\|\mathbf{A}\| \ll 1$ is verified.
- Thus consider the converging Neumann series

$$(\mathbb{I} - \mathbf{A})^{-1} = \mathbb{I} + \mathbf{A} + \mathbf{A}^2 + \mathbf{A}^3 + \mathbf{A}^4 + \ldots$$

  It turns out that

$$(\mathbb{I} - \mathbf{A})^{-1} \approx \mathbb{I} + \mathbf{A} + \mathbf{A}^2$$

  is usually good enough. And it is a bit faster than GMRES.
- A good example of KISS.

New Implementation, Using POSIX Threads

- The setup and postprocessing is done with `MATLAB`/*Octave*.
- The values of $\frac{\exp(i\,2\,(k_1' x - k_2' y))}{\overline{k'}}$ are computed with C code and stored in **shared memory**, for faster access by the inner loops.
- The inner loop over z values is implemented in C, using **POSIX threads**.
  - The binary `RunMultipleTH_z_Neumann` accepts a few parameters: `NT frame index_low index_high NumIter`
  - The optimized FFTW library is used for the 2D FFT computations, using a single core. Only the setup of FFTW is not thread safe, i.e. a Mutex (mutually exclusive) lock/unlock has to be used to initialize the library in each thread.
  - OPENBLAS has to be restricted to use one thread only.
- The required parameters are passed as arguments of the `system()` command in `MATLAB`/*Octave*. Results are returned to *Octave*/`MATLAB` in files.

Coding Details I

Initialization

```
#include <fftw3.h>      // using the FFTW library
#include <pthread.h>    // using pthreads
#include <sys/shm.h>    // using shared memory

...

// check for the optimal FFTW configuration
// on the current host, load if available
if ((inFile = fopen("FFTW.wis","r"))){
    fclose(inFile);
    fftw_import_wisdom_from_filename("FFTW.wis");
}
```

Coding Details II

Setting up parameters and calling multiple POSIX threads

```
for (th_id = 0; th_id<NT; th_id++){
RM_z[th_id].M          = M;
RM_z[th_id].NumIter    = NumIter;
RM_z[th_id].fft_beta   = fft_beta;
RM_z[th_id].texp       = texp;
RM_z[th_id].EXP        = EXP;
RM_z[th_id].gamma_ptr  = gamma;
RM_z[th_id].index_low  = index_low+th_id*index_step;
RM_z[th_id].index_high=
  ((index_low+(th_id+1)*index_step)<index_high)?
   (index_low+(th_id+1)*index_step-1):index_high;
pthread_create(&(my_threads[th_id]),NULL,
        RunMultiple_z_th,&RM_z[th_id]);
}
```

Coding Details III

Reading the parameters in the thread and starting the thread

```
void * RunMultiple_z_th(void * info_ptr){
  int index, index_low, index_high, M, NumIter;
  struct RM_z_struct *info = info_ptr;
  index_low  = info->index_low;
  index_high = info->index_high;
  M          = info->M;
  NumIter    = info->NumIter;
  gamma_ptr  = info->gamma_ptr;
  fft_beta   = info->fft_beta;
  texp       = info->texp;
  EXP        = info->EXP;
  ...
  }
```

Coding Details IV

FFTW initialization, using a mutex lock and saving the configuration.

```
pthread_mutex_lock(&lock_plan);
fftw_plan plan_forward = fftw_plan_dft_2d(2*M,2*M,
                    ft_Inp, ft_Res,
                    FFTW_FORWARD,FFTW_EXHAUSTIVE);
fftw_plan plan_backward = fftw_plan_dft_2d(2*M,2*M,
                    ft_Res, ft_Big,
                    FFTW_BACKWARD,FFTW_EXHAUSTIVE);

if ((inFile = fopen("FFTW.wis","r"))){
  fclose(inFile);
}
else {
  fftw_export_wisdom_to_filename("FFTW.wis");
}
pthread_mutex_unlock(&lock_plan);
```

Coding Details V

Apply the convolution by 2D FFT, multiplication and 2D IFFT. Then work on zero padding.

```
fftw_execute(plan_forward);
for(ii=0;ii<4*numk;ii++) ft_Res[ii] *= fft_beta[ii]
fftw_execute(plan_backward);

// Remove zero padding and update
for(ii=0; ii<M; ii++){
  for(jj=0; jj<M; jj++){
    tmp[ii*M+jj] = ft_Big[(DOM_min+ii)*
                          2*M+DOM_min+jj]*FFT_factor;
  }
}
for (ii=0;ii<numk;ii++){ f[ii] += tmp[ii];}
```

Timing for Multithreaded Code

Determine the computation time [ms] for 1 frame:

		NT, number of threads					
host	`numz`	2	4	6	8	12	24
hilbert	967	72.1	45.5	33.3	36.3	27.8	32.6
`MATLAB`	6034	326.4	177.5	129.6	152.9	105.6	119.2
hilbert	967	67.2	40.7	41.1	33.9	24.7	26.0
Octave	6034	288.1	158.1	154.9	136.9	93.9	103.9
karman	967	43.3	31.2	26.7	26.5	24.1	21.4
`MATLAB`	6034	176	99.5	76.3	80.2	66.4	56.1
karman	967	38.9	27.6	23.5	19.6	18.8	16.0
Octave	6034	163	94.0	69.4	55.8	57.1	39.2

Using the 600 frames with 6034 z values this leads to a minimal total time of 56.3 sec on `hilbert` and 23.5 sec on `karman`.
The starting point was 267 sec (or even 603 sec) on `hilbert`.

Analysis by Regression I

The time required to compute one frame depends on multiple parameters

- `numz`: the number of z points in one frame
- `NT`: the number of threads used

For each frame we expect contributions to the computation time:

- A constant time c_1 to determine the scattering transform and setup the iteration over the z values.
- The integral equations for each of the `numz` different z values are solved in parallel, using multiple threads.

Expect a contribution of the form

$$\texttt{Time} \approx c_1 + \texttt{numz}\left(c_2 + c_3\,\frac{1}{\texttt{NT}}\right)$$

Analysis by Regression II

Run the code with *Octave* on `karman` for many different values of `numz` and `NT`, then use linear regression to find

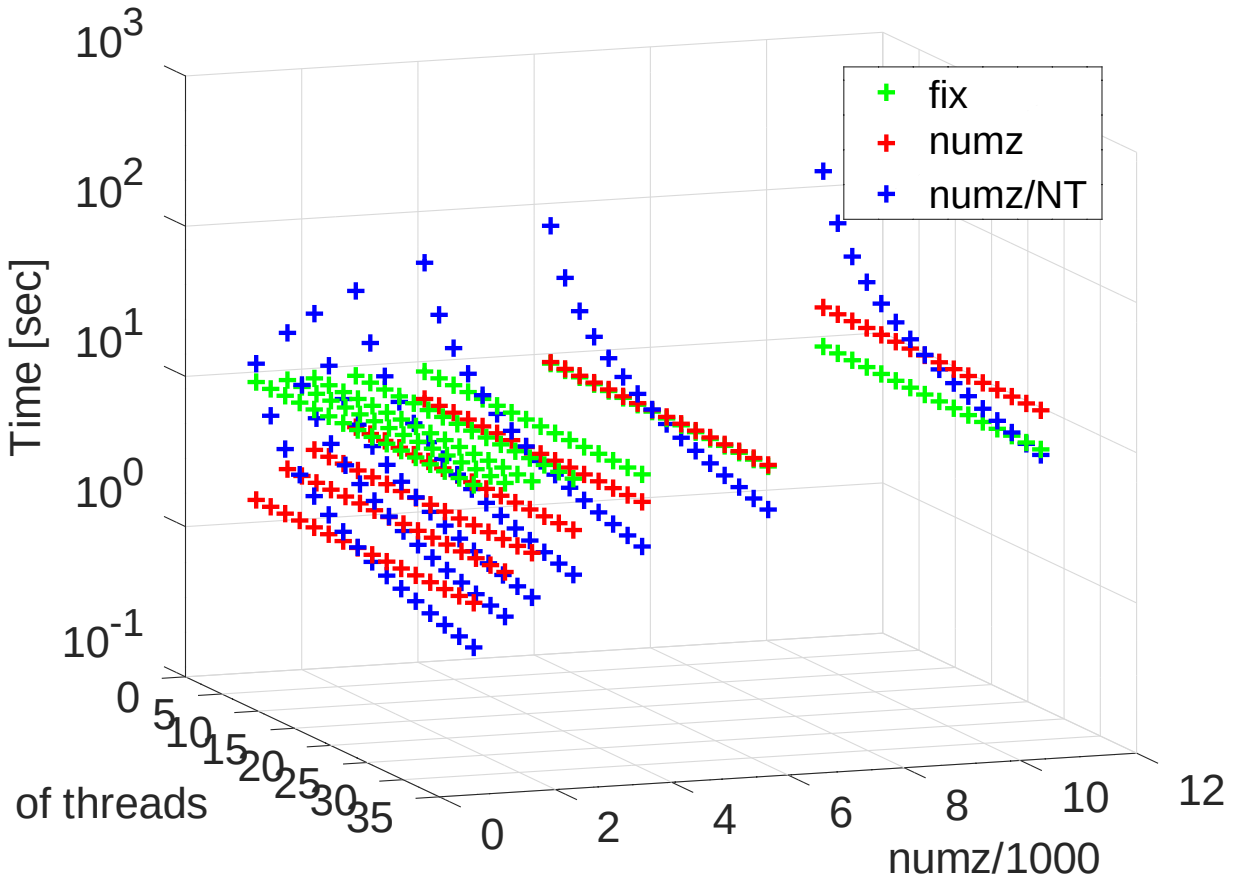
$$\begin{aligned}\texttt{Time} &\approx c_1 + c_2\,\texttt{numz} + c_3\,\texttt{numz}\cdot\frac{1}{\texttt{NT}}\\ &\approx 9.65 + 1.63\,\frac{\texttt{numz}}{1000} + 26.4\,\frac{\texttt{numz}}{1000}\cdot\frac{1}{\texttt{NT}}\end{aligned}$$

Using `MATLAB` on `karman` leads to

$$\texttt{Time} \approx 10.48 + 1.79\,\frac{\texttt{numz}}{1000} + 26.8\,\frac{\texttt{numz}}{1000}\cdot\frac{1}{\texttt{NT}}$$

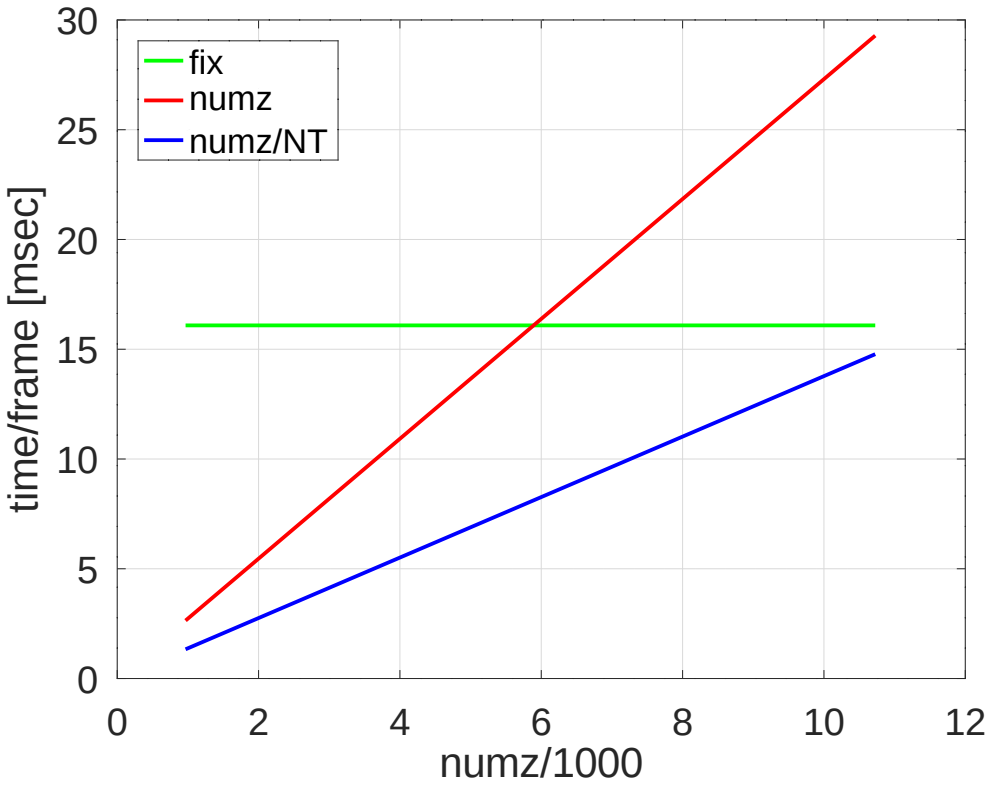
Analysis by Regression III

Examine which contribution dominates the computation time consumed to work on 600 frames.



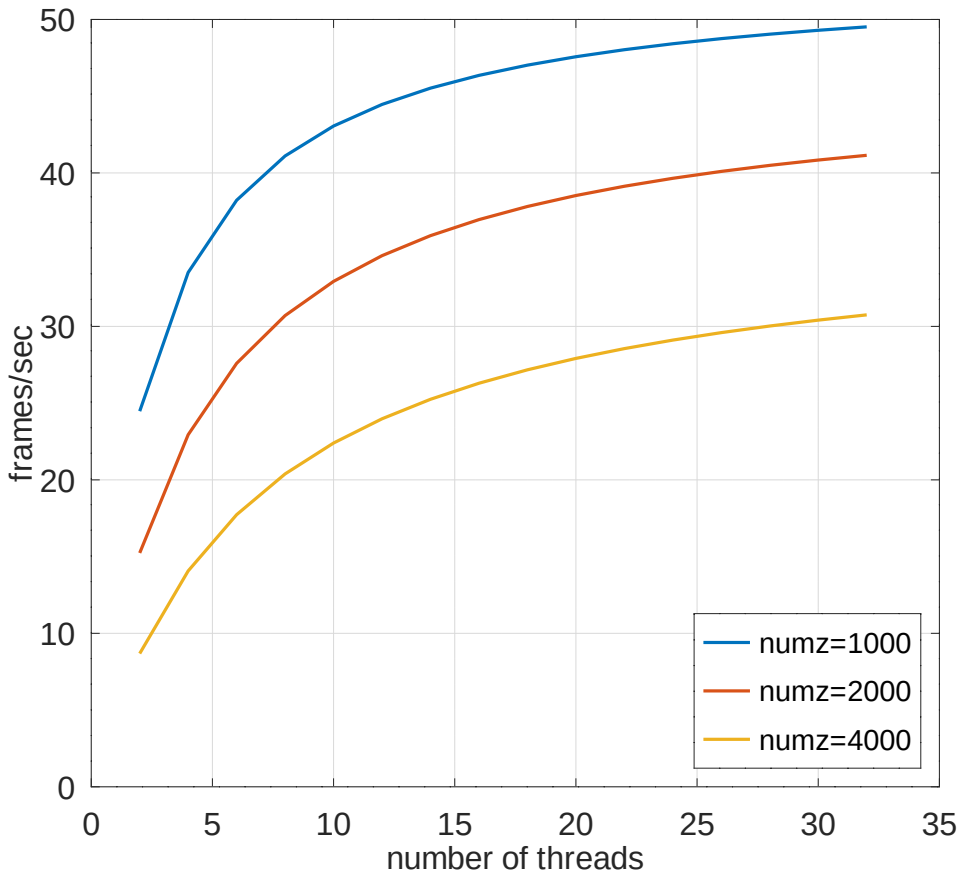
Analysis by Regression IV

Examine which contribution dominates the computation time when working with 32 threads on the host `karman`.



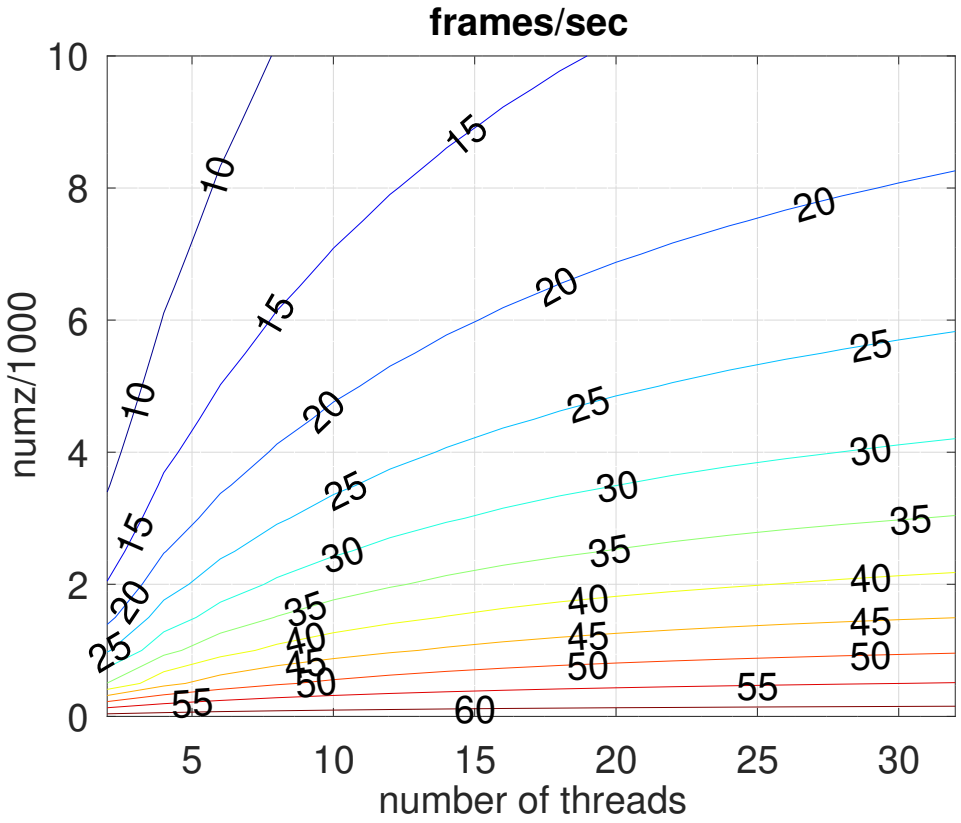
Analysis by Regression V

Evaluate how many frames can be computed per second.



Analysis by Regression VI

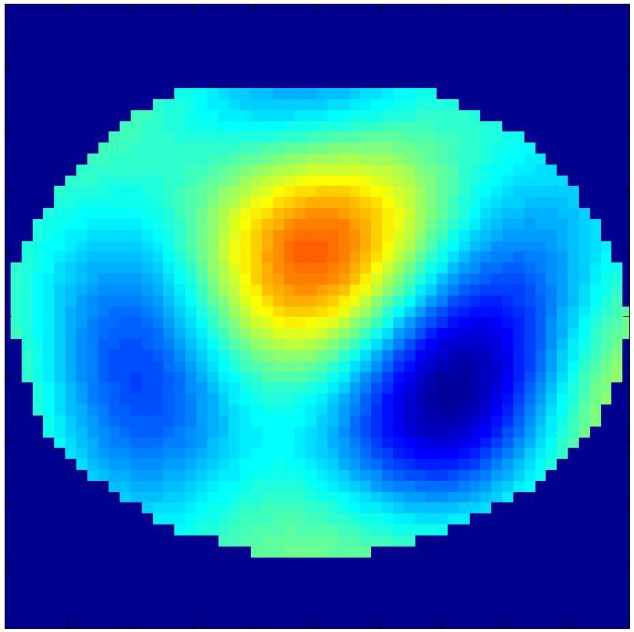
Evaluate how many frames can be computed on `karman` per second, depending on the number of threads and the number of z points.



Thank you

Goal achieved

but more to go



That's all folks

Nächste Termine

Biel

- **06.11** Die Lithium-Ionen-Batterie Grundlagen, Anwendungen und Zukunftsperspektiven
- **20.11** Analysis of Terms & Conditions
- **04.12** Special optical fibers with elevated aluminum content for high temperature or medical applications
- **18.12** Risiken bei Unfällen mit Elektrofahrzeugen
- **22.01** Veränderungen der Aderhaut und Retinadicke früh erkennen

Burgdorf

- **30.10** High-Speed Rotation in der Brennstoffzellentechnik
- **13.11** Das Euresearch Office der BFH
- **27.11** Harmonisierung der Funknutzung/Ergebnisse der WRC-19
- **11.12** Road to Cybathlon 2020
- **15.01** Wie viel Energie wird Ihre Photovoltaik-Anlage erzeugen?