Systemmatrices in MFS and OctoView

Version 15.06.2025

Author: Patrick Vogel

Short introduction

In **Magnetic Particle Imaging (MPI)**, the **system matrix** refers to a mathematical model that relates the response of magnetic nanoparticles (MNPs) to the applied magnetic fields, which helps in reconstructing an image from the MPI measurements [5, 6].

Magnetic Particle Imaging (MPI) Overview:

MPI is an imaging technique that uses superparamagnetic nanoparticles as tracers and exploits their nonlinear response to a time-varying magnetic field. The key advantage of MPI is its ability to directly image the magnetic particles with high sensitivity and resolution, without background signals from tissues or other materials. It is widely used in medical imaging, drug delivery tracking, and other biomedical applications.

What is the System Matrix in MPI?

In MPI, the **system matrix** is a critical component of the image reconstruction process. It encapsulates the relationship between the **spatial distribution of the magnetic nanoparticles** and the corresponding **measured signal** detected by the system. Specifically, the system matrix is a linear transformation that maps the spatial arrangement of the magnetic particles within the imaging field to the voltage signals recorded by the detection coils.
Mathematically, this can be described as:

$$\boldsymbol{b} = \boldsymbol{A} \cdot \boldsymbol{x}$$

Where:

1. $\boldsymbol{b}$ is the vector of measured signals (the output of the MPI scanner).
2. $\boldsymbol{A}$ is the **system matrix**, which represents the response of the nanoparticles at different spatial locations to the applied magnetic fields.
3. $\boldsymbol{x}$ is the vector representing the unknown spatial distribution of the magnetic nanoparticles (the image to be reconstructed).

Key Points about the System Matrix in MPI

1. **Calibration of the System Matrix**: The system matrix is typically obtained through a **calibration process**. A known quantity of magnetic nanoparticles is scanned at different spatial positions within the field of view. The system records the response of the nanoparticles at each position, and these responses are stored in the system matrix. In essence, the system matrix is a comprehensive map that relates the magnetic particle distribution at various locations to the MPI signal.

2. **Size and Complexity**: The system matrix in MPI can be very large because it must account for the system's response at every spatial position (voxel) and across multiple signal channels (e.g., different frequency components of the received signal). For high-resolution images, this can lead to a system matrix with millions of elements, which poses challenges for data storage and computational efficiency.

3. **Image Reconstruction**: Once the system matrix is calibrated, it is used to reconstruct images. The MPI measurement vector $\boldsymbol{b}$ is obtained during an imaging session, and the inverse problem of finding $\boldsymbol{x}$ (the spatial distribution of nanoparticles) from $\boldsymbol{b}$ is solved using the system matrix $\boldsymbol{A}$. This typically involves

solving a large-scale linear system or using regularization techniques to handle noise and ill-posedness in the inversion process.

4. **Forward Model**: In MPI, the system matrix embodies the forward model, meaning it predicts how a given distribution of magnetic nanoparticles would produce a measured signal. In practice, the system matrix can account for several physical effects, including:
    - The spatial sensitivity of the detection coils.
    - Nonlinear responses of magnetic nanoparticles.
    - The geometry of the applied magnetic field and its gradient.

5. **Alternatives to the System Matrix Approach**: While the system matrix method is widely used, it is not the only approach to image reconstruction in MPI. Another common method is the **X-space reconstruction** technique [7], which relies on real-time processing of the MPI signal without the need for a pre-calibrated system matrix. X-space methods trade off some flexibility and accuracy for computational efficiency compared to system matrix-based reconstructions.
   There is a hybrid method called **image-based system matrix** approach [1-4], which combines both reconstruction methods by utilizing x-space input data for the system matrix instead of spectral information. The advantage is a more flexible reconstruction approach, which is independent from hardware influences.

Challenges and Considerations

- **Calibration Time**: Acquiring a system matrix can be time-consuming because it requires scanning a calibration object at many locations.
- **Memory and Computational Demand**: Storing and manipulating large system matrices, especially for high-resolution imaging, requires significant computational resources.
- **Regularization**: The inversion of the system matrix is typically ill-posed, meaning the reconstruction process can be sensitive to noise, so regularization techniques (like Tikhonov regularization) are often used to stabilize the image reconstruction.
- **Estimate Moore-Penrose inverse of the given matrix**: another approach for solving the equation is to calculate the inverse of the given matrix. A singular value decomposition (SVD) is a prominent approach for that.

Summary

In Magnetic Particle Imaging (MPI), the **system matrix** is a critical tool for image reconstruction, representing the relationship between the distribution of magnetic nanoparticles and the signals measured by the system. It is obtained through calibration and used to solve the inverse problem of reconstructing images from the measured signals, enabling the visualization of nanoparticle distributions within the body or other environments.

Understanding the concept of system matrix reconstruction

The principal idea behind a system matrix reconstruction approach is to handle parameters, effects and influences from hardware, sequences and particle systems, which cannot be easily described by formalism.
In short, a system matrix reconstruct approach is a black-box approach. For that, the particle system, which is used in upcoming measurements is placed as a small point-like sample at each position in space within the scanners' field of view (FOV) (see Fig. 1). At each position (1..4), a full dataset is acquired with a specific transfer function for each position (**spatial dependent point-spread function – sdPSF**). These input data sets are stored in the system matrix as column vectors (Fig. 2).
It has to be mentioned that as input information all possible parameters can be used.
With the given system matrix, which is now unique for the used scanner, the sequence and the MNPs, arbitrary distributions of the MNP measured with the same sequence can be reconstructed. Two major reconstruction

processes are available: iterative regularization methods (e.g. Kaczmarz in combination with Tikhonov) or pre-computed pseudo inverse (e.g. using SVD).

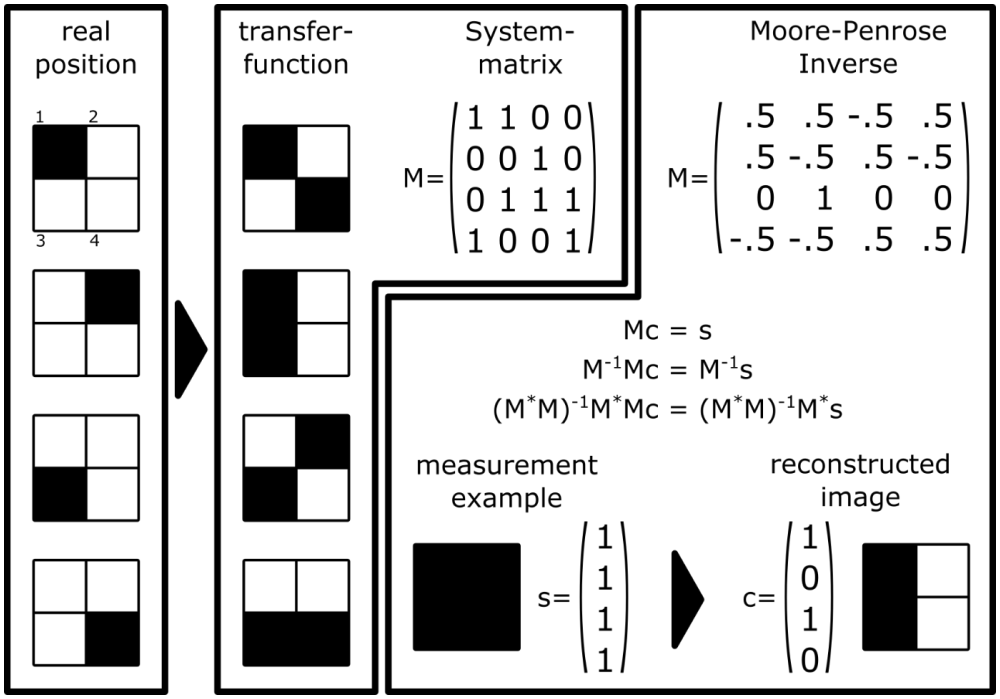

Figure 1: Sketch of an example for a system matrix reconstruction: at each positions in the scanner a point-like sample is measured and their transfer-function is stored within a system matrix as column vectors. After inverting the matrix, arbitrary measurement data can be reconstructed.

The system matrix approach overcomes all issues coming up with hardware distortions, MNP relaxation effects and other possible recurring effects during the imaging process.
The disadvantage is the fact, that the acquired system matrix is only valid for this specific parameter space, which means by changing one parameter, the entire system matrix has to be acquired again. Since the process of acquisition can take a long time, especially when using 3D system matrices, this approach is quite unflexible.

As mentioned above, the input information can be different. In fig. 2, raw-images (similar to convolved x-space images) are used as input information. The advantage by using the raw-image instead of direct k-space parameters is the fact, that raw-images are independent from the scanners' hardware, because all distortions, e.g. coming from the amplifiers or receive chain, is corrected. The only influence is the particle relaxation itself, which can influence the PSF symmetry (it has to mentioned, that relaxation effects can partially be compensated by using exponential deconvolution kernels [9]).
Since now the raw-image, which is the input information for the system matrix, is independent from the scanners' hardware and partially independent from the particles hardware, these system matrices can be pre-calculated by a simulation tool, which is able to emulate the scanners' hardware [8]. This process is much faster and more flexible to adapt different parameters, especially from the sequence site.

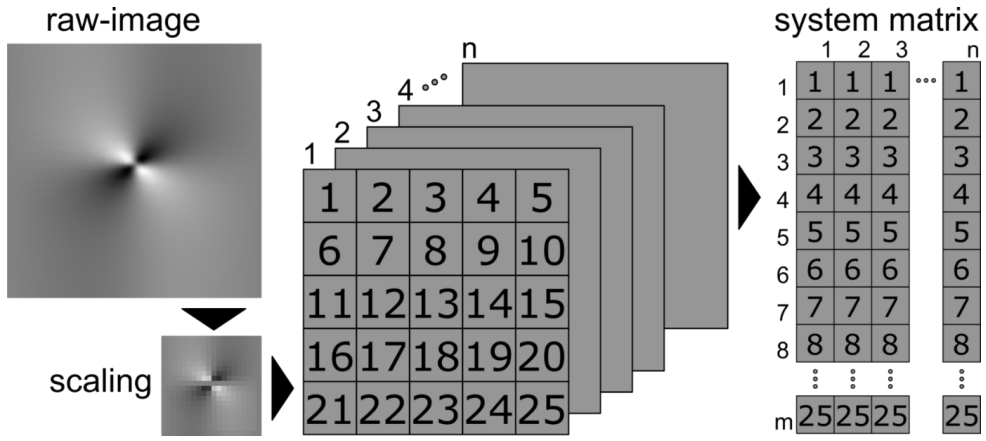

Figure 2: Image-based system matrix approach: This hybrid reconstruction method raw-images used as input information for system matrices. For each position in space, a point-like sample is used.

For 3D space, the acquisition for each voxel follows a defined trajectory, which is stored in the order of the columns within the system matrix (Fig. 3).

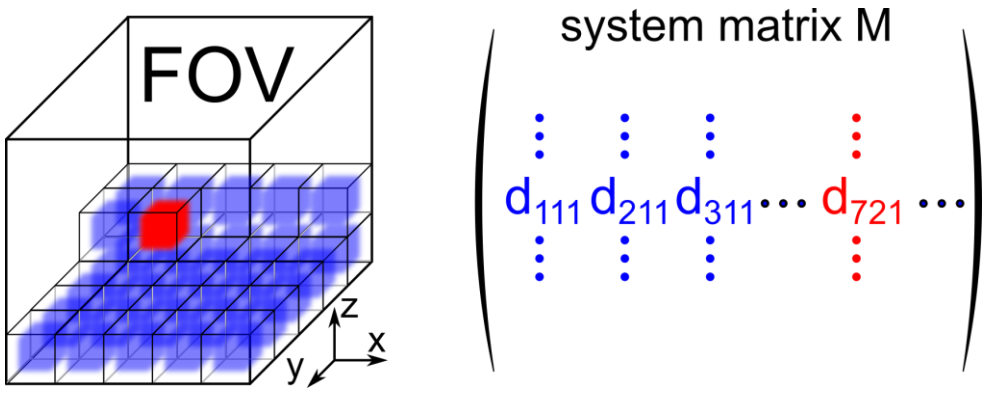

Figure 3: For the storage of the system matrix input information, a given order has to be defined.

The SVD algorithm

The sketch in figure 1 shows the common way for solving a system matrix reconstruction or in other words solving the linear equation $Ax = b$.
To get there, one prominent approach is to calculate the inverse of the matrix A.
The **pseudo-inverse** (or Moore-Penrose pseudo-inverse) is a generalization of the inverse for matrices that are not square or are singular. It is widely used in solving linear systems, especially in least-squares problems.

Definition:
For a matrix A, the pseudo-inverse A^+ satisfies:

1. $AA^+A = A$
2. $A^+AA^+ = A^+$
3. $(AA^+)^T = AA^+$
4. $(A^+A)^T = A^+A$

Applications:

- Solving linear systems $Ax = b$ when A is not invertible.
  - **Exact solution**: $x = A^+b$, if one exists.
  - **Least-squares solution**: Finds x that minimizes $\|Ax - b\|$.
- In machine learning and statistics for linear regression.
- Dimensionality reduction, e.g., Principal Component Analysis (PCA).

One method to compute A^+ is the SVD (Singular Value Decomposition) method, the most robust and common method:

$$A = U\Sigma V^T,$$

where U and V are orthogonal, and Σ is a diagonal matrix of singular values.
The pseudo-inverse is computed as:

$$A^{+} = V\Sigma^{+}U^{T},$$

where Σ^{+} is the pseudo-inverse of Σ, obtained by taking the reciprocal of non-zero singular values and transposing.
There are other methods like Normal Equation (for over-determined systems), Gram-Schmidt or QR Decomposition (for under-determined systems) and iterative methods. The latter approach is useful for large, sparse matrices and can approximate the pseudo-inverse without direct computing. Algorithms are gradient descent algorithm or Kaczmarz method.

Pseudocode:

```
// ************************************************************************
// desc.: perform a SVD and calc the inverse matrix.
// finally perform a vector multiplication
// input: systemmatrix A --> SVD(W,U,VT)
//        s --> signal-vector
//        lambda --> regularization value
//        trunc_k --> truncation
// out:   1D vector consisting the reconstructed information
// ************************************************************************
function Calc_InvMatrix_MulVector ( A: TReal2DArray; s: TReal1DArray; lambda:
Real = -1; trunc_k: Integer = -1 ): TReal1DArray;
var
  Wl, W: TReal1DArray;
  U, VT, inv_a: TReal2DArray;
  m, n, i, j, k: Integer;
begin
  m := Length ( _A );
  n := Length ( _A [ 0 ] );

  // test
  if Length ( _s ) <> m then exit;

  // calculate the single value decomposition of the matrix
  RMatrixSVD ( _A, m, n, 2, 2, 2, W, U, VT ); //e.g. ALGLib or LAPACK/BLAS

  SetLength ( inv_a, n, m );
  SetLength ( result, n );
  for i := 0 to n - 1 do begin
    result [ i ] := 0;
    for j := 0 to m - 1 do begin
      for k := 0 to n - 1 do
        if W [ k ] <> 0 then
          inv_a [ i,j ] := inv_a [ i,j ] + VT [ k,i ] * U [ j,k ] / W [ k ];

      result [ i ] := result [ i ] + inv_a [ i,j ] * _s [ j ];
    end;
  end;

  // free memory
  W := nil;
  Wl := nil;
  U := nil;
  VT := nil;
  inv_a := nil;
end; // <- Calc_InvMatrix_MulVector
```

Usage of the SVD algorithm in Octoview

TODO

The Kaczmarz algorithm

An iterative algorithm for solving large, sparse linear systems $Ax = b$. It works by successively projecting the solution estimate onto the hyperplanes defined by the rows of A. Each step adjusts the solution to better satisfy one equation of the system. The method is efficient for large and sparse systems and finds applications in image reconstruction and tomography.

Sparsity Constraints (L1 regularization)
Assumes that the tracer distribution is sparse (i.e., only a few locations contain the tracer particles). L1 regularization promotes sparsity, helping isolate specific areas of interest and reduce noise in the reconstructed image.

Tikhonov Regularization (L2 regularization):
A technique to stabilize the solution of ill-posed or noisy problems, especially when A is poorly conditioned or singular. It modifies the objective by adding a regularization term to the least-squares problem:

$$\|Ax - b\|^2 + \lambda\|x\|^2$$

where λ controls the balance between fitting the data and smoothness of the solution.
It is also known as **ridge regularization**, which minimizes the sum of squared differences between the observed signal and the reconstructed image, while also penalizing large values in the solution. It smooths the reconstruction and helps reduce the impact of noise, though it may blur boundaries.

Relationship:

- The Kaczmarz method may struggle with ill-posed systems where solutions are unstable or undefined.
- Tikhonov regularization (L2 regularization) can be incorporated into iterative methods like Kaczmarz to stabilize the process and ensure convergence to a meaningful solution.
- Iterative approaches to solve Tikhonov-regularized systems can be based on modified Kaczmarz iterations, making them computationally efficient for large-scale problems.

In summary, the Kaczmarz method efficiently handles linear systems iteratively, while Tikhonov regularization ensures stability and robustness in challenging cases. They complement each other when applied together.

Usage of the Kaczmarz algorithm in OctoView

The Kaczmarz algorithm used in OctoView is an optimized iterative solver for linear systems with following parameters (fig. 4):

- **iterations** Defines the number of passes the algorithm is used for one reconstruction. Useful values are between 5 and 100. A high value causes a high calculation time.
- **lambda** Tikhonov regularization parameter. Useful values are between 1 and 1e-5.
  The higher the value the higher the filtering effect (less higher system functions are used for reconstruction)
- **threshold** System matrices can be ordered by the energy of their system functions
  This default setting if given by model-based system matrix generation in MFS5. Useful values are 25%-50% of the number of system functions given in the system matrix. A reduced number causes higher efficiency and short calculation times.
  HINT: when using non-sorted system matrices, this value has to be set to -1 to ensure that all relevant system functions are used.
- **softTH** This parameter sets a soft threshold: Below this value all intermediate results are set to zero. This helps isolate specific areas of interest and reduce noise.
- **force real** Avoid negative values for all intermediate results during the reconstruction process.

Figure 4: Parameter overview for Kaczmarz algorithm.

The non-negative least squares (NNLS) algorithm

The **Fast Non-Negative Least Squares (FNNLS)** algorithm solves the linear system **Ax** = **b** under the constraint that all components of the solution **x** are non-negative (**x** > 0). It minimizes the least-squares error min ||Ax - b||^2. The algorithm uses an active set approach, iteratively updating a set of active variables **P** to find the optimal solution returned as vector **d** [11].

Iteration Mechanics

Main Loop

- **Gradient Check**: Computes **w** = $\mathbf{A}^T\mathbf{b}$ - $\mathbf{A}^T\mathbf{Ad}$. Identifies the largest **w**[*i*] for $i \notin \mathbf{P}$). If **w**[i] ≤ Tol, the algorithm terminates.
- **Add Variable**: Adds the index with the largest **w**[i] to **P**.
- **Solve Least Squares**: Solves min $\|\mathbf{A} - \mathbf{b}\|^2$ for active variables in **P**, setting **s**[*i*] = 0 for $i \notin \mathbf{P}$.
- **Inner Loop (Non-Negativity)**: Ensures **s**[*i*] ≥ 0 for $i \in \mathbf{P}$:
    - If **s**[*i*] ≤ Tol, computes step size *a* = min *d*[*i*]/(*d*[*i*] - *s*[*i*]), updates d = d + a(s - d), and removes variables with d[i] ≈ 0 from **P**.
    - Re-solves the least-squares problem for updated **P**.
- **Update**: Sets *d*[*i*] = max(0, s[i]), updates *w*, and checks if **P** changed. If unchanged for multiple iterations (*no_update* limit) or *MaxIter* is reached, the algorithm stops.

Image Reconstruction Context

In image reconstruction, *x* represents pixel intensities n = width × height. High iteration counts occur because:

- **Large** *n*: Each pixel is a variable, potentially requiring an iteration to add to **P**.
- **Dense Solutions**: Images with many non-zero pixels increase the size of **P**, needing more iterations.
- **Matrix A**: Complex or ill-conditioned matrices slow convergence.
- **Small Tolerance (**Tol**)**: Strict *Tol* values demand more iterations for precision.

To reduce iterations:

- Use a better initial **P** (*PInitial*), e.g., from a coarse reconstruction.
- Increase *Tol* for faster convergence (balance with image quality).
- Exploit image sparsity or optimize **A** for efficiency.

Usage of the FNNLS algorithm in OctoView

The implementation requires for the first use a pre-calculation of the correlation matrix $\mathbf{AA^T}$, which can take a while. This step has to be repeated for each change of the input system matrix.

- **iterations** Defines the number of passes the algorithm is used for one reconstruction. Useful values are between 5 and 100. A high value causes a high calculation time.
- **lambda (Tol)** Numerical tolerance for convergence check. Useful start value is 1e-8. Increasing this value causes an increased speed of convergence.
- **threshold** System matrices can be ordered by the energy of their system functions
  This default setting if given by model-based system matrix generation in MFS5. Useful values are 25%-50% of the number of system functions given in the system matrix. A reduced number causes higher efficiency and short calculation times.
  HINT: when using non-sorted system matrices, this value has to be set to -1 to ensure that all relevant system functions are used.

Summary

To conclude: a system matrix stores the spatial dependent transfer-function of a point-like sample (**sdPSF**) for a given FOV. The recipe, which defines what kind of input information, what order in the system matrix is provided, and what position in space is used, has to be stored as well.
These recipes for different system matrix reconstruction approaches are defined in the following section.

Different system matrices in MFS and OctoView

The MFS software provides several different templates for the generation of system matrices:

Fourier-based

- **GWSM**: Fourier-based system matrix for Gleich-Weizenecker systems following their provided settings [6].
  Input: <mx> <my> <ND> <df>
- **TWSM**: Fourier-based system matrix especially for TWMPI systems following a specific peak-picking process [1].
  Input: <m1> <m2> <df> <f1> <f2> <mode> <mode2>
- **FTSM**: Fourier-based system matrix collecting *N* pairs of real and imaginary parts from a given Fourier-transformed dataset from the given frequency k_{from} up to k_{to}.
  Input: <k_{from}> <k_{to}>
- **FT1F**: Fourier-based system matrix collecting real and imaginary parts from a given Fourier-transformed dataset for a given frequency f_1 and within a given range k_{from}...k_{to}.
  Input: <k_{from}> <k_{to}> <f_1> <type>

  Parameter type is defined as:
  - type=0: collecting even harmonics: $2*i*f_1 \in \{ k_{from} \dots k_{to}\}$
  - type=1: collecting odd harmonics: $(2*i-1)*f_1 \in \{ k_{from} \dots k_{to}\}$
  - type=2: collecting all harmonics within the range $\{ k_{from} \dots k_{to}\}$
- **ARSM**: The ARSM recipe is a more flexible (**AR**bitrary) scheme than the FTSM or FT1F scheme. It uses a pre-calculated look-up table (LuT), called **peaklist**, consisting of frequency indices (*k*-index) collecting the real and imaginary parts from the Fourier transformed spectrum. This peak-list is stored along the system matrix. Furthermore, the peak-list can also be pre-ordered by the system functions' energy (**ordering feature**).
  Attention: by changing the order, the order of the 3D information has to be adjusted too.
  Input: <peaklist>

To calculate the peak-list in MFS, a specific scheme is used (`calc_ARSM_data`):

<*sizeX*> <*sizeY*> <*sizeZ*>

<formula>

<$varX_{from}$> <$varX_{fo}$> <$varY_{from}$> <$varY_{to}$> <$varZ_{from}$> <$varZ_{to}$>

(<*derivative*>)

The *sizeX*, *sizeY*, *sizeZ* parameters define the amount of input voxels, the formula defines the process the frequency information is collected, the <*var*> parameters can be seen as running variables. The derivative parameter is optional and defines the number of derivations before starting the peak-picking process.

Example for a TWMPI-FFL-type0: "ellipsoid 11 11 19 3 3 3 15 15 1000" and frequencies 12150Hz, 1150Hz, 150Hz with df=50Hz:

89 19 1 243*(2*x-1)+23*2*y+3*2*z;243*2*x+23*(2*y-1)+3*(2*z-1) 1 15 -5 5 -5 5

Signal-based

- **ASM2**: This system matrix scheme is a specific scheme to extract information from time signal data sets with a sliding window method.

Input: <sizeX> <sizeY> <delay> <pdl> <dx> <k_from> <k_to> (<derivation>)

Example for a TWMPI-FFL-type0: "ellipsoid 11 11 19 3 3 3 15 15 1000" and frequencies 12150Hz, 1150Hz, 150Hz with df=50Hz:

89 19 1 243*(2*x-1)+23*2*y+3*2*z;243*2*x+23*(2*y-1)+3*(2*z-1) 1 15 -5 5 -5 5

Image-based

- **TWI1/TWI2/TWI3**: The image-base system matrix schemes are divided into 3 different approaches [1]:
  - **TWI1**: After raw-image generation, the image is cropped and scaled down before stored as column vector in the system matrix.
  - **TWI2**: After raw-image generation, the image is 2D Fourier transformed. A defined inner area (low frequencies) of the complex image is used a input information.
  - **TWI3**: Same as TWI2 but instead of taking the complex information, the cropped complex image is Fourier transformed again (inverse FT) and the resulting image is used as input information.

This approach can be quite powerful but requires lots of parameters since also the entire raw-image generation process has to be defined. The input information for the system matrix generation is similar for all **TWIx** schemes:

Input: <imgX> <imgY> <sx> <sy> <crop> <x1> <y1> <x2> <y2> <mode> ...

<rawIMGx> <rawIMGy> <f1> <f2> <dx> <dy> <from> <to> ...

<fillX> <fY> <foldX> <fy> <sgn1> <sgn2> <sgn3> <sgn4> ...

(<derivation>) ...

(<formula(;)> <x_min> <x_max> <y_min> <y_max> <z_min> <z_max>) ...

(<k3rem>)

Example 1: for a TWMPI-FFL-type0(3D): "ELLIPSOID 13 13 19 3 3 3 20 20 1000" with frequencies 50Hz & 8650Hz and derivation=2

19 137 100 200 0 0 0 0 0 0 200 400 50 8650 0,25 0,25 5 49995 1 0 0 0 1 -1 1 -1 2

Example 2: for a iMPI-type6 (2D): "CUBOID 35 1 75 4 4 4"

with cropping:

75 35 60 80 1 19 0 138 159 0 320 320 60 2420 0,25 0,25 10 124990 1 0 1 1 1 -1 1 -1 2

without cropping:

75 35 80 80 0 0 0 0 0 0 320 320 60 2420 0,25 0,25 10 124990 1 0 1 1 1 -1 1 -1 2

System matrix sub-structure

All data required for a system matrix reconstruction scheme is stored in a sub-folder with a unique naming.

HINT: Do not store multiple system matrices within the same sub-folder.

```
- sub-folder <SM>\
      - <SMname>.txt            → system matrix info
      - <SMname>.sm             → system matrix values
      - <SMname>.3dt            → 3D mapping info

      // for ARMS, the peaklist table file is required:
      - <SMname>.int            → system value ordering
      - <SMname>_order.int      → orig. system value ordering
```

The data types for the generated files are defined as follow:

System matrix files (*.sm; *.3dt; *.int)
For managing the system matrix data, different types are used:

- Info-file (*.txt) – ***ASCII format***
- System matrix values (*.sm) – ***Binary file***
  header=$2\times$`Integer(4Byte)` data=$M\times N\times$`double(8Byte)`
- 3D mapping file (*.3dt) – **Binary file**
  header=`Integer(4Byte)` data=$M\times 3\times$`double(8Byte)`
- Index file (*.int) – ***Binary file***, no header $N\times$`integer(4Byte)`

For more information: see the document: ***datatype.pdf***

Generating system matrices

There are multiple ways to create system matrices:

- **Model-based:** Use a simulation framework to generate simulated system matrices within a virtual scanner. For a given discrete pattern, at each point in space a data set is generated using point-like sample and further processed depending on the desired system matrix (SM types).
- **Measurement-based:** At each point within a real scanner, a point-like sample is placed and measured.

Both ways are using the same processing steps to create a final system matrix as described:

1. Position a point-like sample at each point in space with coords (x,y,z)
2. Perform a simulation (model-based) or measurement (measurement-based) at each point
3. Prepare the data set depending on the desired system matrix to get the required encoding information and put them as column into the system matrix (*.sm)

4. Store the coords in the *.3dt file in the correct order
5. Optional: store the peak-list data (*.int) (order of the encoding information, e.g., in k-space (ARSM))

Example 1: rapid generation using MFS

The first approach for system matrix generation is to use the internal template structure of the MFS framework. For that, load the example project "`2024_TWMPI_20mm`" from the folder "`2024_TWMPI_sm_gen1`". This project loads a TWMPI scanner with pre-defined settings (f_1=50 Hz; f_2=650 Hz; f_3=9,450 Hz) with a sequence type6.

1. Go to the "MNP" tab and enable MNP0 and disable MNP1 (set checkbox used). Select MNP0.

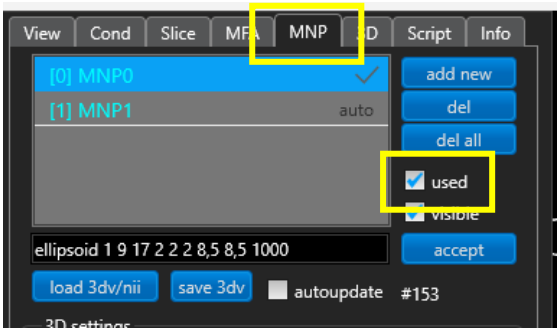

Figure 5: Screenshot from MFS5.

2. Go to the "Cond" tab and select the desired receive coil, here index 7.

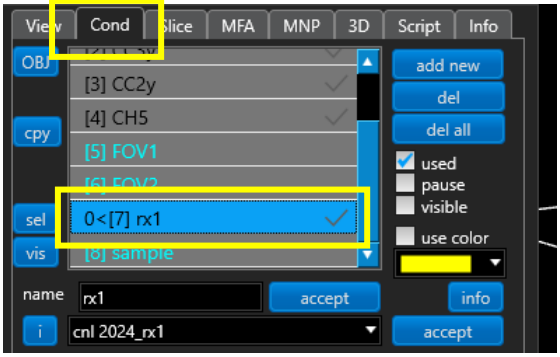

Figure 6: Screenshot from MFS5.

3. Open the system matrix calculation window by pressing the button "calc SF/SM".

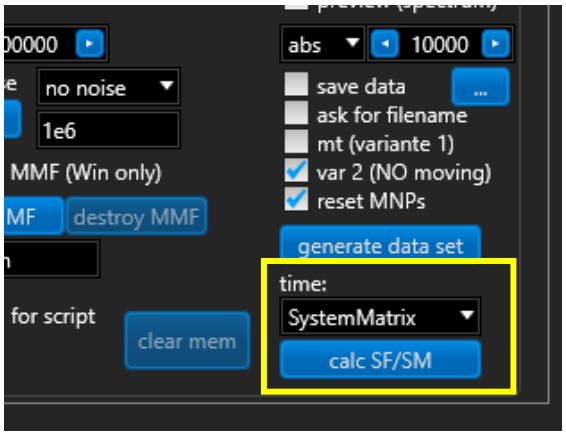

Figure 7: Screenshot from MFS5.

4. Set up the settings for the system matrix, here type ARSM.

   The settings for this system matrix type is set with one line consisting of multiple parameters (#=10) defining the peak-picking process. To get the correct size of the three first parameters defining the

numbers of points/voxels, a “get SM settings” button can provide a correct number when selected the correct MNP.

With the “accept” button, a system matrix with all required data is calculated and stored within the project-folder defined with “subdirectory”.

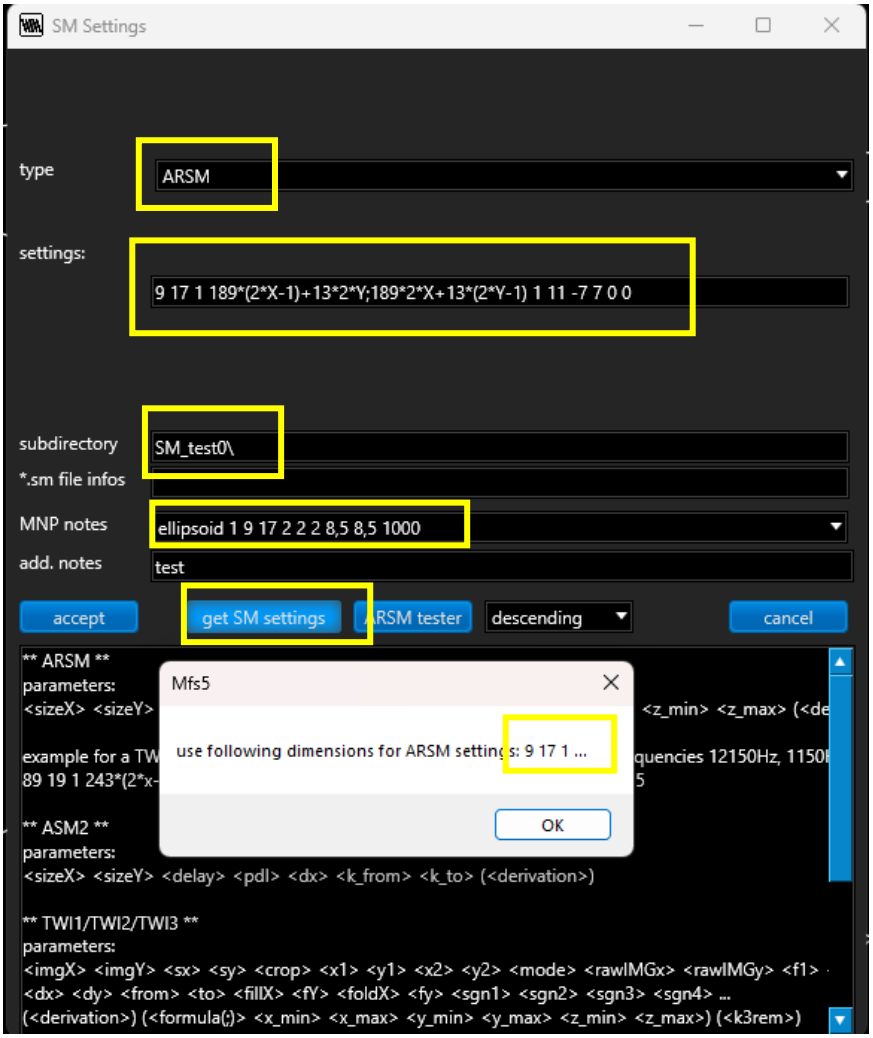

Figure 8: Screenshot from MFS5.

Example 2: point-by-point generation using MFS scripting language

Another approach for calculating a system matrix is the use of a script for the MFS framework. For that, load the example project “`2024_TWMPI_20mm`” from the folder “`2024_TWMPI_sm_gen1`”. This project loads a TWMPI scanner with pre-defined settings (f_1=50 Hz; f_2=650 Hz; f_3=9,450 Hz) with a sequence type6.

Go to the “Script” tab and select “script2”: this script allows to run point-by-point through a defined MNP structure defining the points in space for system matrix (`param mnpSM 0`).

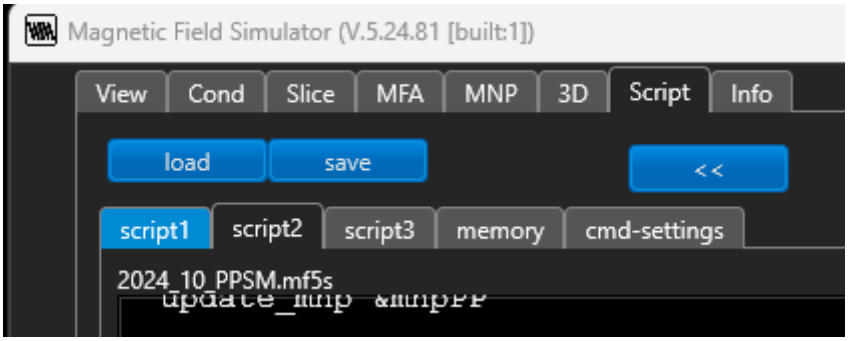

Figure 9: Screenshot from MFS5.

The second MNP structure is defined as a single point and represents the point-like sample (`param mnpPP 1`).

Additional parameters can be defined for the desired matrix structure: size, formula, variables, etc.

Pressing the "go" button starts the script and visualize the point-by-point calculation of the system matrix.
In the subfolder of the project, two additional folders are created:

SM_TEST1: consisting of the result of the point-by-point system matrix calculation

SM_TEST2: consisting of the a list of data sets (DATA00000.dbl, ...) for each point in space. Furthermore, a textfile (coords.txt) provides the coords for each point.

HINT: the data in SM_TEST2 are required for example 3.

Example 3: data set based generation using MFS scripting language

The third approach for generating a system matrix is the use of single data sets. These data sets can be generated by a simulation framework, e.g., the MFS software or directly measured with a real MPI scanner.

For the demonstration, a script has been implemented to show the workflow. For that, load the example project "`2024_TWMPI_20mm`" from the folder "`2024_TWMPI_sm_gen1`". This project loads a TWMPI scanner with pre-defined settings (f_1=50 Hz; f_2=650 Hz; f_3=9,450 Hz) with a sequence type6.

HINT: Please run the script: "`2024_10_PPSM.mf5s`" in script2 prior before running the script "`2024_10_SM_gen1.mf5s`" in script3.

The script "`2024_10_SM_gen1.mf5s`" uses external datasets with a specific naming "data00000.dbl; data00001.dbl; ..." and a textfile "`coords.txt`" consisting of the coordinates for each data set with the format:

```
X1 Y1 Z1
X2 Y2 Z2
...
```

By pressing the "go" button, the script runs through the data and generates the system matrix as defined. The result is provided in the subfolder "`SM_test2\SM\`".

Using self-generated system matrix

When generating a system matrix via example 3, the input data can come from a different source, e.g., a point-by-point acquisition with a point-like sample at different positions in the scanner.

In that case, the reconstruction parameters for using such a system matrix have to be checked and adjusted carefully depending on the processing of the data sets.
For example, a robot is used to position the point-like sample at multiple points inside the scanner. For each point, the sample data set as well as a baseline data set is acquired.
The upper script has to be adjusted to perform an additional subtraction of signal and baseline-signal data set (baseline correction).
You can do that by adding these lines for selecting the right data:

```
loop l1 0 &data_dir_count/2-1
  param c1 2*&l1counter
  param c2 &c1+1
```

```
// baseline correction
combine_data %data_dir&&c1 %data_dir&&c2 %data sub

// processing
FFT_data %data
prepare_arsm_data %data %dataout %peaklist
```

At the end, the data sets, which are used for the SM generation, does not had any other post-processing such as RCC, global phase correction, derivatives, or thresholding.
That means, that for the reconstruction process, these parameters have to be set to default values (**if not defined otherwise**):

- Do not use RCC and no derivations
- Set global phase to zero
- Set threshold to -1 (to use all system functions for reconstruction)

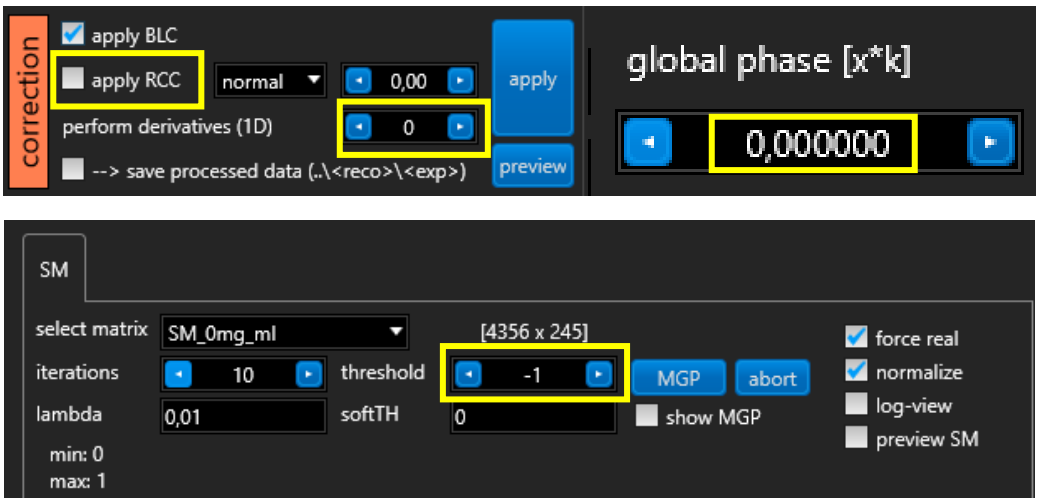

Figure 10: Screenshots from Octoview: showing the parameters which have to be set for correct reconstruction using a measured system matrix.

HINT: A good validation method to check the consistency of the system matrix is to use the same signal data sets for reconstruction.

Combining multiple system matrices

System matrices can be seen as structures, which maps the magnetic response of a point-like specific MNP system to their spatial location. For each parameter set as well as different MNP system a unique system matrix can be built.

Thus, it is possible to combine multiple system matrices, e.g., to encode within the same experiment different types of particle types. This method is also known as multi-color MPI [10].

There are multiple possible issues coming up when using multiple system matrices for the reconstruction process:

- The limitation of the amount of usable encoding information from the Fourier spectrum or image-based information: From a typical 3D spectrum for a single receive coil, approximately 3,000 components (frequencies) can be extracted. When encoding a volume with 1,000 or 2,000 voxels and for different MNP systems, the system is dramatically underdetermined.

- When stringing together multiple system matrices from different sources, a normalization of the system matrices could be mandatory.

Reasons for normalization

If you are combining two system matrices, normalization is often beneficial to ensure **consistency** and **numerical stability**. Here are some reasons and methods for normalization:

1. **Consistent Scaling**: If the two matrices have different value ranges, this could make the reconstruction algorithm unstable or give undue weight to one of the matrices.
2. **Numerical Stability**: Large differences in the value scales of the matrices can cause numerical instabilities, especially in iterative reconstruction methods.
3. **Improved Comparability and Weighting**: If the matrices represent different physical properties (e.g., different measurement ranges or frequency bands), normalization helps make them comparable.

Methods for normalization

Here are some common approaches to normalize system matrices:

1. **Max normalization**: Scale each matrix so that its maximum value is 1.

   $$A_{norm} = \frac{A}{max(|A|)}$$

   and

   $$B_{norm} = \frac{B}{max(|B|)}$$

   This scales each matrix independently to the same range.

2. **Row or Column normalization**: Normalize each row or column so that values within a row or column are brought to the same range. This is useful if rows (or columns) are intended to contribute differently to the reconstruction.

3. **Standardization (Z-score normalization)**: Subtract the mean and divide by the standard deviation of each matrix.

   $$A_{norm} = \frac{A - mean(A)}{std(A)}$$

   and

   $$B_{norm} = \frac{B - mean(B)}{std(B)}$$

   This centers and scales the distribution of values, which is especially helpful with high variability in data.

4. **Energy-Based normalization**: Calculate the energy of each matrix (e.g., the sum of the squares of all elements) and scale the matrices so that they have the same total energy.

5. **Weighted combination**: If one of the matrices is more important, you can multiply that matrix by a weight *w*:

   $$A_{combined} = w \cdot A + B$$

   This ensures that the two matrices contribute differently to the reconstruction.

When normalization is necessary

- **Different sources**: When the matrices come from different measurement sources or represent different physical units.
- **Large differences in scale**: When one matrix has a value range that strongly deviates from the other.
- **Different application domains**: In applications where the two matrices represent different frequencies or signal strengths.

Applying appropriate normalization before combining the system matrices ensures that neither matrix dominates the other and that the combination supports the desired reconstruction quality.

Combining two system matrices

After the generation of a system matrix, there are several files available in the system matrix folder.

ATTENTION: It has to be mentioned, that for combination of system matrices, the **ordering feature** of system matrices, which is required for threshold Kaczmarz reconstruction, has to be disabled.

`<SMname>.txt` **text-file**: consists of system matrix information

Here the important values are:

img_x, img_y define the size of the 2D projection space

(in the case of 2D system matrices this size represents the size of the final reconstructed image)

SM_m, SM_n define the matrix size with

SM_n number of pixels/voxels

SM_m number of encoding parameters

`<SMname>.3dt` consists of the 3D coords for the voxels in space

(#=SM_n × 3 [`Double`])

`<SMname>.int` **ONLY** required for ARSM-types: consists of the LuT for collecting the encoding information (#=SM_n / 2 [`Int`] …because of using real and imaginary part as parameters).

`<SMname>.sm` consists of the system matrix values (SM_m × SM_n [`Double`])

Step 1: extending the values img_y and SM_n (img_x and SM_m remain constant).

HINT: The extension of img_y causes a nice visualization of 2D system matrices without interleave artifacts but requires a consistency check for 3D matrices because of the 2D projection.

The new values are $$img'_y = img_y^{(1)} + img_y^{(2)}$$

$$SM'_n = SM_n^{(1)} + SM_n^{(2)}$$

Consistence check: the product of img_x and img_y must the equal to SM_n:

$$SM_n = img_x + img_y$$

Step 2: adjusting the number of 3D coords

For correct data processing and visualization, the 3D cords have to be adjusted too. Here it is important to use spatially separated areas for each of the system matrix, e.g., simply place the coordinates next to the first area.

Example: MFS-script system matrix combination

The MFS software provides the script command `combine_sm_data`, which allows the combination of two system matrices via the technique presented before.

After running the examples 1-3, the example folder should consist of the sub-folders `SM_test0`, `SM_test1`, `SM_test2`. By executing the script "`2024_11_combine_SM`", a new sub-folder "`SM_test3`" with a combined system matrix is calculated.

When using this system matrix, someone would expect that both show the same reconstruction result since both system matrices have been built with the same data. But the example 1 uses a specific ordering feature: the default setting in the MFS software is a descending ordering of the eigenvectors (energy). This ordering feature can be set directly in the system matrix calculation window or via the script command:

```
GLOBAL_PEAKLISTORDERING 1
```

Example 2 and 3 did not use a specific ordering.

References

[1] P. Vogel et al., Flexible and Dynamic Patch Reconstruction for Traveling Wave Magnetic Particle Imaging, *Int J Magn Part Imag*, 2(2):1611001, 2016.
[2] P. Vogel et al., Parallel Magnetic Particle Imaging, *Rev. Sci. Instrum.*, 91:045117, 2020.
[3] P. Vogel et al., Adjustable Hardware Lens for Traveling Wave Magnetic Particle Imaging, *IEEE Trans Magn*, 56(11):3023686, 2020.
[4] P. Vogel et al., Real-time 3D Dynamic Rotating Slice-Scanning Mode for Traveling Wave MPI, *Int J Magn Part Imag*, 3(2):1706001, 2017.
[5] A. Neumann et al., System-matrix based reconstruction in magnetic particle imaging, *Int J Magn Part Imag*, 9(1):2303005, 2023.
[6] J. Rahmer et al., Signal encoding in magnetic particle imaging: properties of the system function, *BMC Medical Imaging*, 9(4), 2009.
[7] P. Goodwill et al., The X-Space Formulation of the Magnetic Particle Imaging Process, *IEEE TMI*, 29(11):1851-1859, 2010.
[8] P. Vogel, M.A. Rückert, T. Kampf, V.C. Behr – Highly Flexible and Modular Simulation Framework for Magnetic Particle Imaging – arXiv:2208.13835, 2022. doi:10.48550/arXiv.2208.13835
[9] K. Bente et al., Electronic field free line rotation and relaxation deconvolution in magnetic particle imaging, *IEEE TMI*, 34(2):644-51, 2015.
[10] J. Rahmer et al., First experimental evidence of the feasibility of multi-color magnetic particle imaging, *Phys Med Biol*, 60:1775–1791, 2015.
[11] R. Bro, S. De Jong, A fast non-negativity-constrained least squares algorithm, *Journal of Chemometrics*. **11** (5): 393, 1997.