HAW HAMBURG

Bachelorarbeit

Tobias Wulf

Winkelmessung durch magnetische Sensor-Arrays und Toleranzkompensation mittels Gauß-Prozess

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Computer Science and Engineering
Department of Information and Electrical Engineering

Tobias Wulf

Winkelmessung durch magnetische Sensor-Arrays und Toleranzkompensation mittels Gauß-Prozess

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Elektro- und Informationstechnik*
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Karl-Ragmar Riemschneider
Zweitgutachter: Prof. Dr. Klaus Jünemann

Eingereicht am: TT. Monat Jahr

Tobias Wulf

Thema der Arbeit

Winkelmessung durch magnetische Sensor-Arrays und Toleranzkompensation mittels Gauß-Prozess

Stichworte

Sensor-Array Simulation, Dipol, Magnetfeld, Kugelmagnetapproximation, TMR, TDK TAS2141, AMR, NXP KMZ60, Toleranzkompensation, Gauß-Prozess, Kovarianzmatrix, Regression, Winkelvorhersage

Kurzzusammenfassung

...

Tobias Wulf

Title of Thesis

Angular Measurement by Magnetic Sensor Arrays and Tolerance Compensation by Gaussian Process

Keywords

Sensor Array Simulation, Dipole, Magnetic Field, Sperical Magnet Approximation, TMR, TDK TAS2141, AMR, NXP KMZ60, Tolerance Compensation, Gaussian Process, Covariance Matrix, Regression, Angular Prediction

Abstract

...

Inhaltsverzeichnis

1 Motivation 0.0.1 14.12.2020

2 Grundlagen 0.0.1 14.12.2020

3 Entwicklung von Software für die Optimierungs-Experimente 0.0.1 14.12.2020

4 Erprobungs- und Optimierungs-Experimente 0.0.1 14.12.2020

5 Auswertung 0.0.1 14.12.2020

6 Zusammenfassung und Bewertung 0.0.1 14.12.2020

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

CPU Prozessorkern.

HDD Festplattenlaufwerk.

OS Betriebssystem.

RAM Arbeitsspeicher.

SW Software.

Literatur

Paper

[1] T. Schüthe, A. Albounyan und K. Riemschneider. „Two-Dimensional Characterization and Simplified Simulation Procedure for Tunnel Magnetoresistive Angle Sensors“. In: *Sensors Applications Symposium (SAS)*. Sensors Applications Symposium (SAS). (13. März 2019). IEEE, 2019, S. 1–6. DOI: 10.1109/SAS.2019.8706125. URL: https://ieeexplore.ieee.org/document/8706125 (besucht am 05. 10. 2020). Online.

Manual

[2] R. Johnson. *MATLAB Style Guidlines 2.0*. Version 2. MATLAB Central File Exchange, 2014. 43 S. URL: https://de.mathworks.com/matlabcentral/fileexchange/46056-matlab-style-guidelines-2-0 (besucht am 21. 09. 2020). Online.

Web-Recherche

[3] Bitbucket. *Feature Branch Workflow in Git*. Hrsg. von ATLASSIAN. 2020. URL: https://www.atlassian.com/de/git/tutorials/comparing-workflows/feature-branch-workflow (besucht am 10. 09. 2020). Online.

Anhang

A Genutzte Software 0.0.3 08.01.2021

Für die Nachvollziehbarkeit der getätigten Entwicklungsarbeiten und die Erstellung der Bachelor-Thesis, ist das dafür jeweilige Betriebssystem (OS) und die verwendete Software (SW) tabellarisch aufgeführt. Es finden sich genutzte Versionen der SW und Angaben zur Minimalanforderung für deren Nutzung. Die Anforderungen sind für Prozessorkern (CPU), Arbeitsspeicher (RAM), Festplattenlaufwerk (HDD) näher aufgeschlüsselt. Die Programmierarbeiten mit MATLAB sind jeweils mit Windows und Linux geschrieben bzw. getestet worden.

Software	**Verwendungszweck (Typ)**	**Min.-Anforderung**	**Version**	**Erscheinungstag**
Ubunut Budgie	Linux-Betriebssystem (Laptop OS)	2 GHz Dual-Core-CPU 4 GB RAM 25 GB freier HDD-Speicher	18.04 LTS	26.04.2018
Windows 10 Enterprise	Windows-Betriebssystem (Laptop OS)	1 GHz Core-CPU 1 GB RAM 32 GB freier HDD-Speicher	1909	12.11.2020
MATLAB	Simulationssoftware (Multi-Paradigmen Programmier- Sprache, IDE)	Intel/ AMD x86-64 CPU 4 GB RAM 3.5 GB freier HDD-Speicher	2020b	17.09.2020
Git	Versionierung (Kommandozeilenprogramm)	- - -	2.29	29.10.2020
Inkscape	Vektorgrafikzeichenprogramm (Grafikaufbereitung)	1 GHz CPU 256 MB RAM 302 MB freier HDD-Speicher	0.92.3	11.03.2018
Texstudio	Textbearbeitung f. LaTeX Dokumente (Editor)	- - 24.7 MB freier HDD Speicher	2.12.6	25.07.2020
wkhtmltopdf	HTML- zu Pdf-Konvertierung	- - -	0.12.6	11.06.2020
JabRef	Literaturverwaltungsprogramm f.BibLaTeX (Editor)	- - -	5.1	30.08.2020

Tabelle A.1: Genutzte Software zur Erstellung der Thesis und Dokumentation der Ergebnisse, Entwicklungsumgebung für die geschriebene Simulationssoftware zur Generierung und Auswertung der Sensor-Array-Simulation.

B Software-Dokumentation 0.0.2 08.01.2021

Die Software-Dokumentation ist automatisiert mit MATLAB-Skripten erstellt worden. Es ist dafür ein zweistufiger Prozess implementiert, der im ersten Schritt eine in MATLAB integrierte HTML-Dokumentation erstellt und im Anschluss diese zu eigenständigen PDF-Dateien exportiert. Als letzter Schritt sind diese zu einem LaTeX-Manual zusammengefasst im Anhang eingebunden. Mit diesem Verfahren ist es möglich, eine Dokumentation direkt aus geschriebenen M-Dateien zu generieren. Allerdings ist es dafür nötig, eine spezielle Formatierung und einen gewissen Programmierstil einzuhalten [2]. Die Dokumentation enthält neben dem erstellten Quellcode eine Reihe von Arbeitsanweisungen, wie mit der Software umzugehen ist. Zusätzlich sind Beschreibungen für die Erstellung und Pflege des Software-Projektes mit beigefügt. Die geschriebene Software ist mithilfe des Software-Versionierungsprogramms Git erstellt worden, was eine genaue Nachvollziehbarkeit in Bezug auf die einzelnen Arbeitsschritte ermöglicht. Zur Versionierung ist der Git-Feature-Branch-Workflow [3] angewandt worden. Aus stilistischen Gründen ist die gesamte Software-Dokumentation in Englisch verfasst.

GaussianProcessDipoleSimulation

The project of sensor array simulation and Gaussian Processes for angle predictions on simulation datasets starts in

May 06. 2019

with IEEE paper by Thorben Schüthe which is a base investigation of "Two-Dimensional Characterization and Simplified Simualtion Procedure for Tunnel Magnetorersistive Angle Sensors". What produces characterization datasets of different current available angular sensors on the market.

June 11. 2019

Thorben Schüthe came up with a high experimental scripting for abstracting sensor characterization fields to an array of sensor fields which was stimulated by magnetic dipole field eqautions to approximate a sperical magnet.

November 06. 2019

Prof. Dr. Klaus Jünemann supports the team around Prof. Dr.-Ing. Karl-Ragmar Riemschneider and Thorben Schüthe with an apply of Gaussian Process learning to investigate on angle predictions for sensor array simualtion results. The attemp of the solution was working for tigth set of parameter and was highly experimental with rare documentation and few set of functions and script. The math of this very solution based the standard book for Gaussian Process by Williams and Rasmussen. The algorithm is related to the guidline for linear regression model which worked fine for a setup of standard use cases but needed further investigation for a wider set of parameters and functions to identify general and relevant parameter settings to provide an applicable angular prediction.

September 21. 2020

Tobias Wulf establish a Matlab project structur and programming guidance and flows to document the source code integrated in the Matlab project architecture. That includes templating for scripts and functions and general descriptions of project structure and guidance for testing and documenting project results or new source code including automation for publishing html in Matlab integrated fashion.

October 22. 2020

Tobias Wulf added TDK TAS2141 TMR characterization to the project. Thorben Schüthe provided a raw dataset which was manually modified by Tobias Wulf to dataset which is plotable and reconstructable in stimulus and characterization field investigations.

October 31. 2020

Tobias Wulf establish a general configuration flow to control part of software via config file which is part loaded as needed into workspace.

November 29. 2020

Tobias Wulf finished the implementation of sensor array simulation which uses TDK TAS2141 as base of simualtion. The software includes now simualtion for situmuls magnet (dipole sphere) and automated way fast generate training and test datasets by set configuration. Various plots and animation for datasets and a best practice workflow for simulation. Also included are unittest and Matlab integrated documentation in html files. A full description of generated datasets is included too.

December 05. 2020

Tobias Wulf integrated a second characterization dataset for NXP KMZ60 into the sensor array simulation software. The

dataset was manually modified in the same way as the TDK TAS2141 dataset. The KMZ60 raw data was provided from Thorben Schüthe. The simulation software was adjusted to run with both datasets now. Additional plots for transfer curves are included for both and same plots for characterization view of KMZ60 as for TAS2141 too.

Created on September 21. 2020 by Tobias Wulf.

Published with MATLAB® R2020b

Workflows

Developing software needs conventions to produce common results and good working software. There are certain points which matches good written software:

1. The reuse factor of the wirtten souce code.
2. Good source code structure or hierachy to expand.
3. Testing with aprobat frameworks e.g. Unittest.
4. Source code versioning.
5. Source code readability and detailed commenting and documentation

The last point can be split into two points but Matlab provides a publish porcess with in source code comments can be used for documentation. What is probably not detailed enough and needs further documents in completition. Ongoing on that to provide support in guidance for current or upcoming project work it is recommended to declare common workflows for those points.

Coding conventions are used from MATLAB Style Guidlines 2.0 by Richard Johnson.

Contents

See Also

- MATLAB Stye Guidlines 2.0

Project Preparation

How to setup a Matlab project with Git support and simple backup plan.

Project Structure

Directory structure, associated task and how to add new elements.

Git Feature Branch Workflow

How to work in the project with Git support in feature driven way.

Documentation Workflow

How to document the project work in progress and introduce new project elements to publishing process.

Simulation Workflow

Best practice simulation workflow for sensor array simulations to generate training and test datasets.

Published with MATLAB® R2020b

Project Preparation

The first steps to setup a scalable software project are none trival and need a good strcuture for later project expands. Either to setup further new projects a well known scalable project structure helps to combine different software parts to bigger environment packages. Therefore a project preparation flow needs to be documented. It unifies the outcome of software projects and part guarantee certain quality aspects.

The following steps can be used as guidance to establish a propper Matlab project structure in general. Each step is documented with screenshots to give a comprehensible explanation.

Contents

See Also

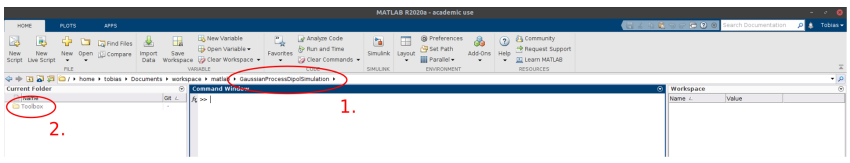
- Create a New Project From a Folder
- Add a Project to Source Control
- Setup Git Source Control
- Use Source Control with Projects
- Git Attributes
- Git Ignores
- Add Files to the Project
- Commit Modified Files to Source Control
- Clone Git Repository

Create Main Project Directory

The main project directory contains only two subfolders. The first one is the Toolbox folder where the project, m-files and other project files like documentation are placed. The folder is also called sandbox folder in Matlab project creation flows which is just another description for a project folder where the coding takes place. The second folder is a hidden Git repository folder which keeps the versionation in final. It is respectively seen a remote repository that establish basics to setup backup plans via Git clone or can be laterly replaced by remote repository on a server or a GitHub repository to work in common on the project.

First step:

1. Create an empty project folder, open Matlab navigate to folder path.
2. Right click in the Current Folder pane and create New> Folder "Toolbox".
3. Open a Git terminal and in the project directory and initialize an empty Git repository.



```
tobias@bean42:~/Documents/workspace/matlab$ cd GaussianProcessDipolSimulation/
tobias@bean42:~/Documents/workspace/matlab/GaussianProcessDipolSimulation$ ls
Toolbox
tobias@bean42:~/Documents/workspace/matlab/GaussianProcessDipolSimulation$ git init
Initialized empty Git repository in /home/tobias/Documents/workspace/matlab/GaussianProcessDipolSimulation/.git/
tobias@bean42:~/Documents/workspace/matlab/GaussianProcessDipolSimulation$ ls -a
.  ..  .git  Toolbox
tobias@bean42:~/Documents/workspace/matlab/GaussianProcessDipolSimulation$ git config -l
user.name=Tobias Wulf
user.email=46107549+TobiasWulf@users.noreply.github.com
core.editor=gedit
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
tobias@bean42:~/Documents/workspace/matlab/GaussianProcessDipolSimulation$
```

3.

Create Matlab Project with Git Support

In second it is needed to create the Matlab project files in a certain way to get full Git support and support for the Matlab help browser environment. In this use case the before created local Git repository is used as remote origin. So several settings are automatacally made during the creation process by Matlab and as mentioned before the "local remote" repository can be replaced later by a remote origin located on a server or GitHub. The Toolbox folder must be empty to process the following steps.

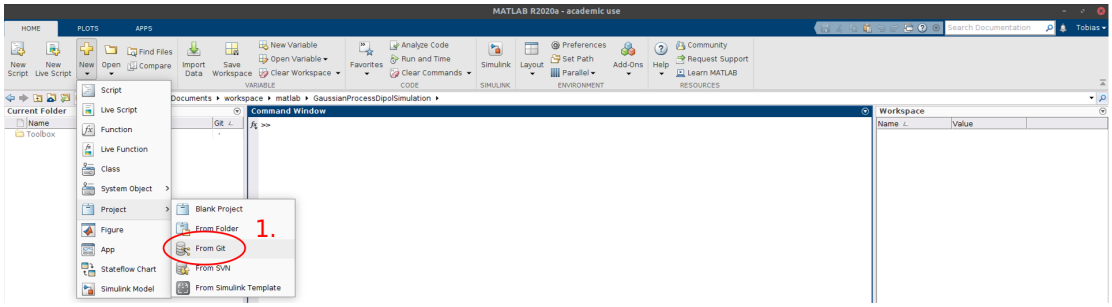
It is recommend to do no further Git actions on the created Git repository via Git terminal!

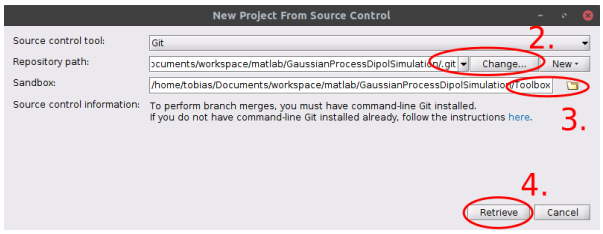
These steps only proceed the project setup further Matlab framework functionality is added later.

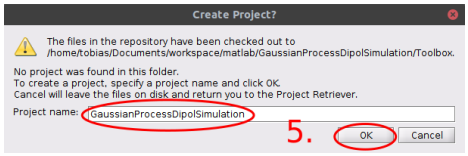
Second step:

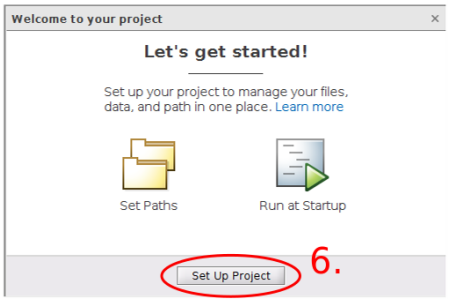
1. In the created main project directory create a New > Project > From Git.
2. Change the repository path to the hidden Git repository path in the main project directory.
3. Change the sandbox path to the Toolbox path in the main project directory.
4. Click Retrieve.
5. Enter the project name given by the main project directory name and click OK.
6. Click on Set Up Project and skip the two follwing steps via Next and Finish.
7. Switch to Toolbox directory by double click on the folder in the Current Folder pane, open the created Matlab project file with a double click and check source control information under PROJECT tab by clicking Git Details.
8. Add a short project summary by click on Details under the ENVIRONMENT section of the PROJECT tab.
9. Click Apply.
10. Click OK.

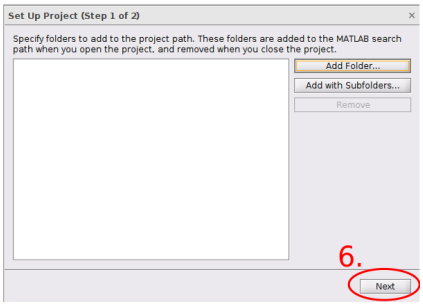
The project itself is under source control now.

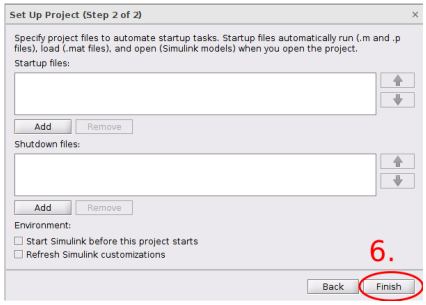


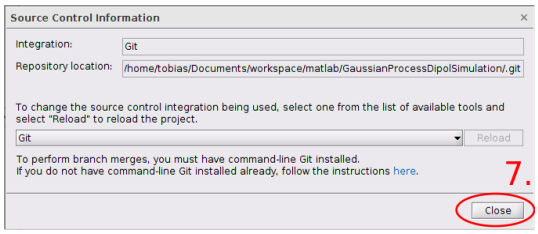

New Project From Source Control
Source control tool:
Git
Repository path:
Change...
New
Sandbox:
/home/tobias/Documents/workspace/matlab/GaussianProcessDipolSimulation/Toolbox
Source control information:
To perform branch merges, you must have command-line Git installed.
If you do not have command-line Git installed already, follow the instructions here.
2.
3.
4.
Retrieve
Cancel

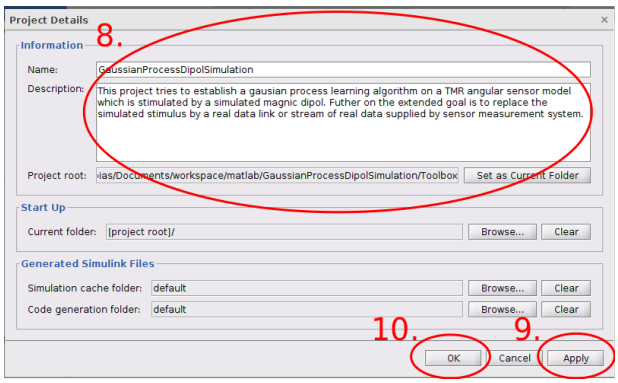

Create Project?
The files in the repository have been checked out to
/home/tobias/Documents/workspace/matlab/GaussianProcessDipolSimulation/Toolbox.
No project was found in this folder.
To create a project, specify a project name and click OK.
Cancel will leave the files on disk and return you to the Project Retriever.
Project name:
GaussianProcessDipolSimulation
5.
OK
Cancel


Welcome to your project
Let's get started!
Set up your project to manage your files,
data, and path in one place. Learn more
Set Paths
Run at Startup
Set Up Project
6.


Set Up Project (Step 1 of 2)
Specify folders to add to the project path. These folders are added to the MATLAB search
path when you open the project, and removed when you close the project.
Add Folder...
Add with Subfolders...
Remove
6.
Next


Set Up Project (Step 2 of 2)
Specify project files to automate startup tasks. Startup files automatically run (.m and .p
files), load (.mat files), and open (Simulink models) when you open the project.
Startup files:
Add
Remove
Shutdown files:
Add
Remove
Environment:
Start Simulink before this project starts
Refresh Simulink customizations
6.
Back
Finish





Registrate Binaries to Git and Prepare Git Ignore Cases

The root of Git is to work as text file versioner. Source code files are just text files. So git versionates, tags and merges them in various ways in work flow process. That means git edits files. This point can be critical if git does edit a binary file and corrupts it, so that is not executable any more. Therefore binary files must be registrated to Git. Another good reason is to registrated binary or other none text files beccause Git performs no automatic merges on file if they are not known text files. To keep the versionating Git makes a taged copy of that file every time the file changed. That can be a very junk of memory and lets repository expands to wide.

To prevent Git for misshandling binaries it is able to regestrate them in a certain file and mark the file types how to handle them in progress. The file is called .gitattributes must be placed in the Git assinged working directory which is the sandbox folder for Matlab projects. The .gitattributes file itself is hidden.

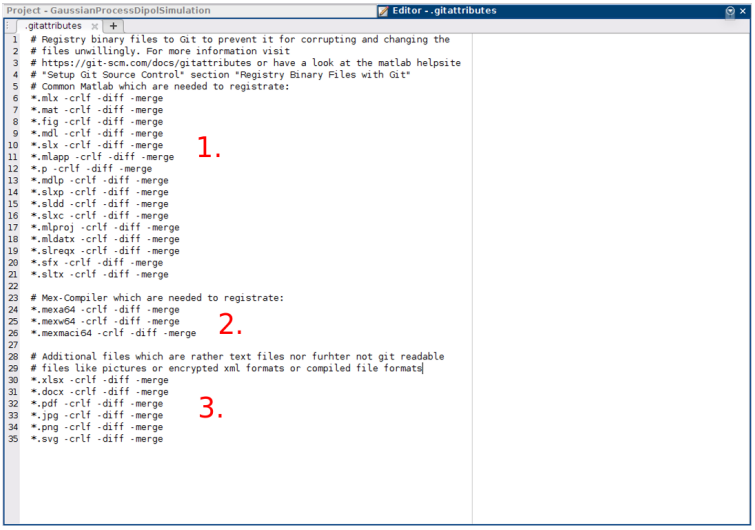
Three options are needed to mark a file type as binary. The -crlf option disables end of line conversion and the -diff option in combination with the -merge option to mark the file as binary.

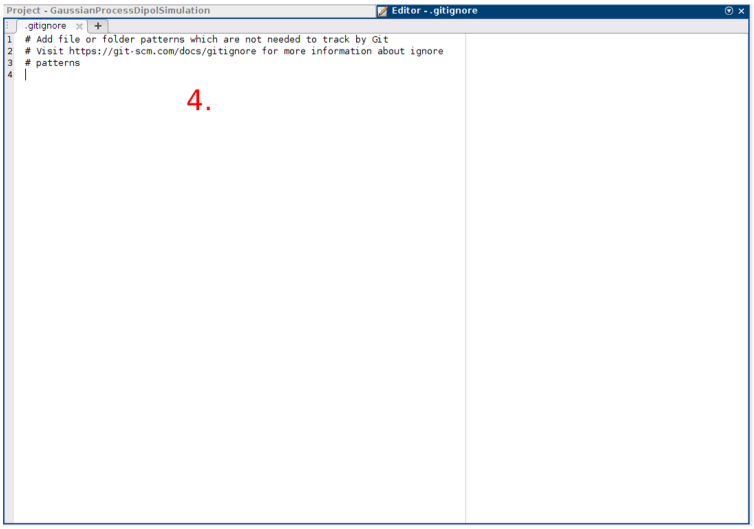
In addition to that it is possible to delclare several ignore cases to Git. So certain directories or file types are not touched or are left out from source control. This is done in .gitignore file. The must be placed in the sandbox folder too.

From the sandbox directory enter in the Matlab command prompt edit .gittatributes and edit .gitignore and save both files. The files are not shown in Current Folder pane (hidden files). Edit both files in the Matlab editor and save the files.

Third step:

1. Add common Matlab file types to .gitattributes.
2. Add Matlab compiler file types to .gitattributes.
3. Add other file types which can be appear during the work to .gitattributes.
4. Add ignore cases to .gitignore if needed.





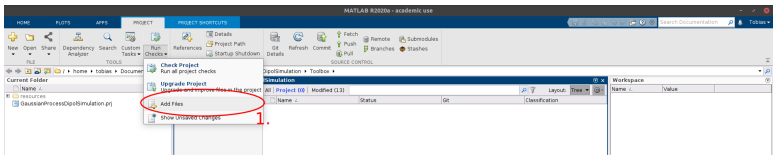
Checkout Project State and Do an Initial Commit

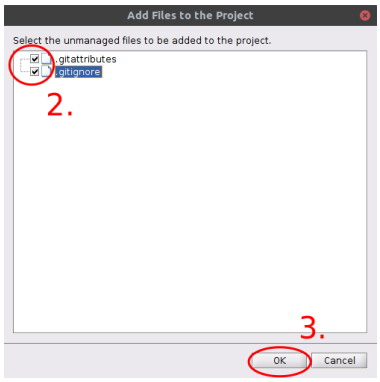
The main part is done. It just needs a few further step to save the work and add the created files to the project.

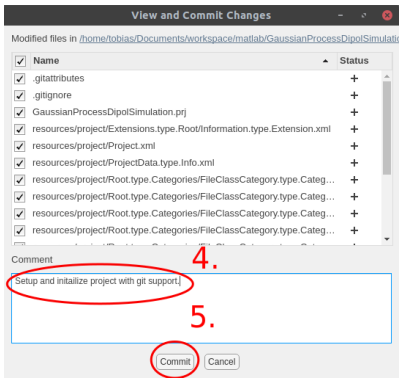
Fourth step:

1. Add created files to the project. In the PROJECT tab under TOOLS section click Run Checks > Add Files.
2. Check the files to add to the project.
3. Click OK.
4. Right click in the white space of Current Folder pane and click Source Control> View and Commit Changes... and add commemt to the commit.
5. Click Commit.

The project is now initialized.



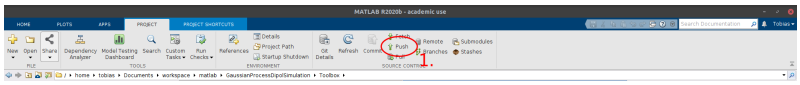


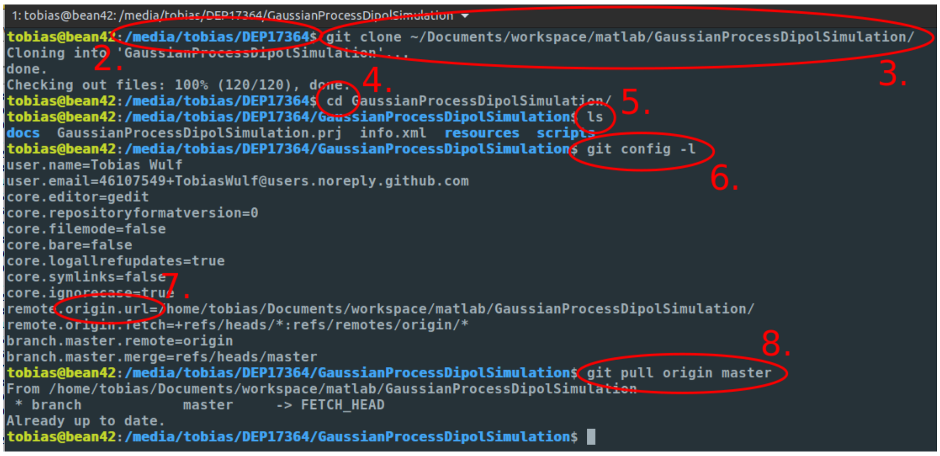


Push to Remote and Backup

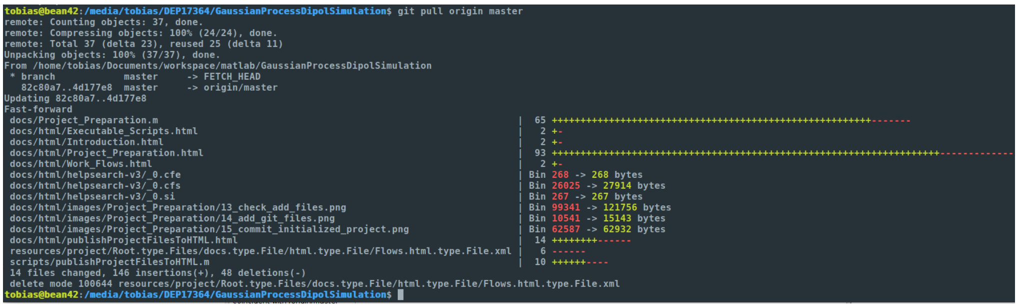
The project is ready to work with. Finally it needs a backup meachnism to save the done work after closing the Matlab session. Git and how the project is build up provide an easy way to make backups.

1. Push the committed changes to remote repository.
2. Insert a backup medium e.g. USB stick and open a git terminal there.
3. Clone the project remote repository from project directory.
4. Change the directory to cloned project.
5. Check if everything was cloned.
6. Check if the remote url fits to origin.
7. Pull from remote to check if everything is up to date.





If further changes are comitted to the project push again to the remote from Matlab environment and repeat update the backup from time to time by inserting your medium and make a fresh pull. Change the directory to the folder and just pull again. See below as an example how does it look like.



Port Remote Repository to GitHub

The remote repository is ported to GitHub laterly. Therfore some minimal changes are made manually to the local repository.

1. According to new rules on GitHub the master branch is renamed to main.
2. Due to that a new upstream is set to origin/main from origin/master
3. To fetch all casualties a merge was needed from origin/main on local main. The orign/master reference was included.
4. Change remote repository to GitHub URL https://github.com/TobiasWulf/GuassianProcessDipolSimulation.git
5. At the moment the GitHub repository is private and not visible in the web. After finishing the general work the repository will be set to publish in consultation with HAW TMR research project and team.
6. After publish on GitHub, clone or fork to work with.
7. The source code is hosted under MIT license.
8. Use GitHub flows to clone or fork and push changes to backup done work.
9. Toolbox folder is not needed anymore because remote is elswhere now
10. Re clone from remote to get new structurew without Toolbox folder

Project Structure

A good project directory structure is the key to build scalable and expandable software projects. Therfore each project folder has to fulfill an associated task. Additionally, a good structure facilitates project navigation and the retrieval and reuse of project content. Further on Matlab provides stratgies to add content to existing project structures and label it for script based execution of project task to manage project files. To add new content have a look at the links below.

Contents

See Also

- Specify Project Path
- Add Files to the Project
- Add Labels to Files

Directory Overview

```
GaussianProcessDipoleSimulation
├── [4.0K]  data
│   ├── [4.0K]  test
│   └── [4.0K]  training
├── [4.0K]  docs
│   ├── [ 12K]  html
│   │   ├── [4.0K]  figures
│   │   ├── [4.0K]  helpsearch-v3
│   │   └── [4.0K]  images
│   │       ├── [4.0K]  avi
│   │       ├── [4.0K]  eps
│   │       ├── [4.0K]  pdf
│   │       └── [4.0K]  svg
│   └── [4.0K]  latex
│       ├── [4.0K]  BA_Thesis_Tobias_Wulf
│       └── [4.0K]  Manual
├── [4.0K]  resources
├── [4.0K]  scripts
├── [4.0K]  src
│   ├── [4.0K]  sensorArraySimulation
│   └── [4.0K]  util
│       └── [4.0K]  plotFunctions
├── [4.0K]  temp
└── [4.0K]  tests

23 directories
```

Generated with linux shell command from on directory above the main project directory.

```
tree -dhn GaussianProcessDipoleSimulation ...
    -o GaussianProcessDipoleSimulation/docs/html/Directory_Tree.txt -I ...
    "project|Project_*|thesis|images"
```

Directory Tasks

Directory	Task
./	Main project directory which contains the Matlab project sandbox files and the hidden repository files. Matlab project sandbox directory. Project root directory which contains the Matlab project file, the info.xml, .gitignore, .gitattributes files and all other project related subdirectories. Startup directory.
.git	Hidden repository for local statndalone work. Saves daily working results. Provide a Git clonable instance of sandbox the directory. Replacable. Not Matlab driven, simulates remote repository.
./resources	Autogenerated directory from Matlab project. Contains the local project versionation and project xml-files.
./data	Contains all project related datasets e.g. mat-files.
./data/trainig	Contains mat-files from sensor array simulation for training cases of the gaussian process.
./data/test	Contains mat-files from sensor array simulation for test cases of the gaussian process.
./docs	Documentation directory which contains m-files only for documtation use and the direcoty where all project remarked files are published into HTML output files.
./docs/html	Publish directory where published m-files are collected and bind to a Matlab help browser readable documentation. It contains html-files and subdirectory for images and figures which are used in the documentaion. The help browser search database is placed here too. Much more important the directory contains the helptoc.xml which pointed by the info.xml from root project directory.
./docs/html/figures	Contains all needed fig-files which are used in the documentation.
./docs/html/helpsearch-v3	Contains autogenerated help search database entries. The directory is rewritten during the publish documentation process.
./docs/html/images	Contains all needed image files like png-files which are used in the documentation.
./docs/html/images/avi	Contains video avi-files.
./docs/html/images/eps	Contains saved figures as eps-files.
./docs/html/images/pdf	Contains saved figures as pdf-files.
./docs/html/images/svg	Contains saved figures as svg-files.
./docs/latex	Documentation directory which LaTeX documentation of the project including subfolders for Thesis of each project participant.
./docs/latex/BA_Thesis_Tobias_Wulf	Bachelor Thesis directory of Tobias Wulf.
./docs/latex/Manual	Export directory for documentation written in Matlab as pdf export.

Directory	Task
./scripts	The sripts directory contains all executable script m-files to solve certain tasks in the project, to generate datasets or execute parts of the toolbox source code.
./src	Source code directory which contains reusable source code clustered in submodule direcotries. The code can be function oriented or class oriented or a mix of both. Contains no bare script files.
./src/sensorArraySimulation	Sensor Array Simulation function and class. Contains functions, mathematical functions and classes to simulate an N x N sensor array on base of the TDK TAS2141 characterization dataset.
./src/util	Util function and class space. Function and class source code to slove upcomming help tasks e.g. to manage project content, to support plot framework or reporting or publishing processes.
./src/util/plotFunctions	Contain plot functions for reuse.
./tests	For test driven development each function or class needs a own test space or file. The directory contains these test.
./temp	Temporally working directory to save intermediate results or the last software state from session before or scratch files which flies arround.

Add New Elements

Add new folder to project:

1. Create a new folder and add to Project Path after Matlab flow.
2. Run Checks > Add Files.
3. Run tree command from shell to update directory for the documentation (optional).
4. Update directorry task table of this document.

Add new file to project:

1. Create new File and edit the file after Documentation Workflow. and Conventions.
2. Run Checks > Add Files.
3. Label the new file from project pane.
4. Commit file into active branch.
5. Registrate to the documentation if needed (publish, toc and listings docs).

Created on October 10. 2020 by Tobias Wulf. Copyright Tobias Wulf 2020.

Published with MATLAB® R2020b

Git Feature Branch Workflow

The project work with Git requires a consistent workflow to apply changes to the Matlab project in a way that no broken source code affects the current state of the project. Therefore Git has the ability to work on new features, issues or bugs in the certain workflow which matches those requirements. This workflow is called Feature Branch Workflow. The workflow describes that for every change in the source code a new branch must be opened in the Git tree. The following changes are committed to the new branch and so that changing commits are not listed in the master branch of the Git tree and have no effect on the made work until the branch is merged back into the master branch. That makes it possible to work on several new features at a time and guarantees a functional working version of the project.

For a deeper understanding in example have a look at the description of Atlassian tutorial page of the Feature Branch Workflow. The listed Matlab help pages describe to use the embedded Matlab Git tooling to apply changes with branching merging.

Contents

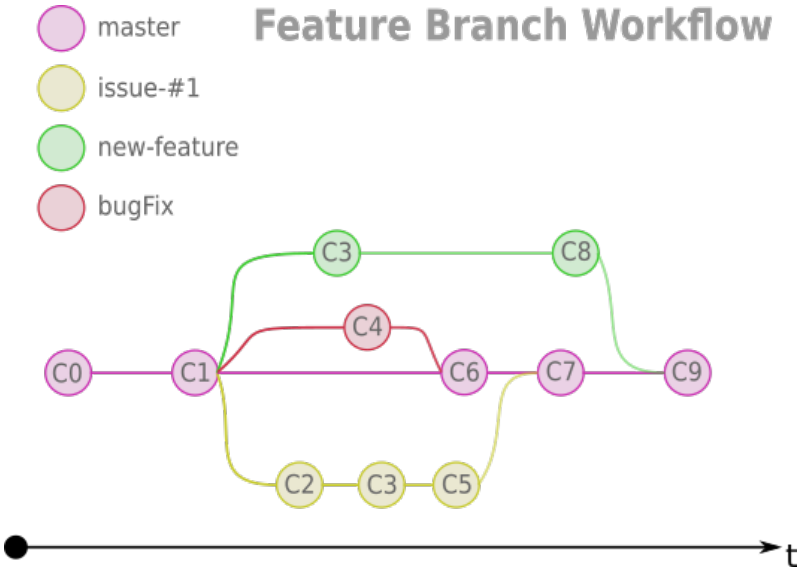
See Also

- Feature Branch Workflow
- Branch and Merge with Git
- Pull, Push and Fetch Files with Git
- Update Git File Status and Revision

Examples

1. The master branch is created. Project starts with commit C0.
2. One commit C1 is added to the master branch e.g. by adding .gitattributes.
3. But there was an issue with that attributes declaration so a new branch is opened to solve that issue.
4. On the same time a new feature must be established e.g. a new script or function. So a second branch is opened.
5. Also a third for a small bug fix.
6. Now the work at those three different task can be done in parrallel without affecting each other.
7. Switch between the different branches by checkout the branch and commit the ongoing work into each branch for itself.
8. If the work is done in a branch, the branch must be merged on the master branch. Git makes automated merge commits (C6, C7, C9) where the changes from the branches are integrated in master branch files.
9. At this point it is possible that merging conflicts are raised. Those conflicts in the files must be solved manually.
10. Just open a new branch for the next change, switch to it and commit the work until its done and the branch is ready to merge back into master

It is best practice to push all created local branches to a remote repository too! It completes the backup on the one hand and on the other it makes the ongoing work accessable to third.



Created on October 07. 2020 by Tobias Wulf. Copyright Tobias 2020.

Published with MATLAB® R2020b

Documentation Workflow

The documentation workflow describes how to document new m-file scripts or functions and where they must be registrated into the publishing process of the documentation. So the published m-file is available in the Matlab help browser of this project.

1. Create a new m-file in the project structure
2. Use the script or function template for initial edit and fill the template with new content.
3. Make introducing documentation entries. If it is a new module, so introduce the module with its own doc where all scripts, functions and classes are listed. If this document already exist, make a new entry.
4. Make help entry in the helptoc.xml via tocitem tag. List all sections of the doc comment as sub tocitems.
5. Introduce the new file to the publish script and make an entry under a fitting section or make a new one if it is a new module or folder.
6. Introduce the new file to export published files script and do toc entries into script file generate pdf-manual
7. Commit the done work.

Contents

See Also

- Project Structure.
- Display Costum Documentation
- publishProjectFilesToHTML
- exportPublishedToPdf

Script Template

```
%% scriptName
% Detailed description of the script task and summary description of
% underlaying script sections.
%
%
%% Requirements
% * Other m-files required: None
% * Subfunctions: None
% * MAT-files required: None
%
%
%% See Also
% * Reference1
% * Reference2
% * Reference3
%
%
% Created on Month DD. YYYY by Creator. Copyright Creator YYYY.
%
% <html>
% <!--
% Hidden Clutter.
% Edited on Month DD. YYYY by Editor: Single line description.
```

```
% -->
% </html>
%
%
%% First Script Section
% Detailed section description of step by step executed script code.
disp("Prompt current step or meaningful information of variables.")
Enter section source code

%% Second Script Section
% Detailed section description of step by step executed script code.
disp("Prompt current step or meaningful information of variables.")
Enter section source code
```

Function Template

```
%% functionName
% Single line summary.
%
%% Syntax
%   outputArg = functionName(positionalArg)
%   outputArg = functionName(positionalArg, optionalArg)
%
%
%% Description
% *outputArg = functionName(positionalArg)* detailed use case description.
%
% *outputArg = functionName(positionalArg, optionalArg)* detailed use case
% description.
%
%
%% Examples
%   Enter example matlab code for each use case.
%
%
%% Input Argurments
% *positionalArg* argurment description.
%
% *optionalArg* argurment description.
%
%
%% Output Argurments
% *outputArg* argurment description.
%
%
%% Requirements
% * Other m-files required: None
% * Subfunctions: None
% * MAT-files required: None
%
%
%% See Also
% * Reference1
% * Reference2
% * Reference3
%
%
% Created on Month DD. YYYY by Creator. Copyright Creator YYYY.
%
% <html>
```

```
% <!--
% Hidden Clutter.
% Edited on Month DD. YYYY by Editor: Single line description.
% -->
% </html>
%
function [outputArg] = functionName(possitionalArg, optionalArg)
    arguments
        % validate possitionalArg: dim class {validator}
        possitionalArg (1,:) double {mustBeNumeric}
        % validate optionalArg: dim class {validator} = defaultValue
        optionalArg (1,:) doubel {mustBeNumeric, mustBeEqualSize(positionalArg, optionalArg)} = 4
    end
    outputArg = positionalArg + optionalArg;
end

% Custom validation function
function mustBeEqualSize(a,b)
    % Test for equal size
    if ~isequal(size(a),size(b))
        eid = 'Size:notEqual';
        msg = 'Size of first input must equal size of second input.';
        throwAsCaller(MException(eid,msg))
    end
end
```


Published with MATLAB® R2020b

Simulation Workflow

That workflow describes a best practice way to simulate a sensor array with dipole (spherical magnet).

1. Clean up old simuation datasets and plots of by executing deleteSimulationDatasets and deleteSimulationPlots.
2. Edit generateConfigMat to needed specifications for simualtion and generate or regenerate config.mat by executing the script.
3. Execute generateSimulationDatasets to generate configure training and test datasets.
4. Execute the needed plots to describe the simulation as wished.
5. Execute other parts of the software to work with current setup of simualtion datasets.
6. Rename plots or move them to a subfolder to save them.
7. Move or rename Datasets if it is needed to keep them after done work.
8. Restart workflow for a next configuration to investigate on.

Contents

See Also

- generateConfigMat
- deleteSimulationDatasets
- generateSimulationDatasets
- deleteSimulationPlots

Published with MATLAB® R2020b

Executable Scripts

Executable scripts of the project to solve various actions or prject tasks. The main approach of project scripts is an automated way to collect and execute certain actions in an example to run project documentation at once or generate project configuration file which are used by other scripts or loaded by functions to control and execute task in a unified project structure.

Contents

exportPublishedToPdf

Export published HTML files to a pdf manual.

deleteSimulationPlots

Delete simulation training and test dataset plots from figures and images path with training and test filename pattern.

deleteSimulationDatasets

Delete generated simulation datasets from data path.

generateSimulationDatasets

Generate simulation datasets from sensor array simulation configuration.

publishProjectFilesHTML

Publish Matlab help browser integrated HTML documentation.

generateConfigMat

Generate configuration for generic use or part use in different program layers.

Created on September 21. 2020 by Tobias Wulf. Copyright Tobias Wulf 2020.

Published with MATLAB® R2020b

publishProjectFilesToHTML

The is script is used to publish all toolbox included files to HTML documentation folder docs/html. The script runs a section with certain options for each project part and uses the built-in function to generate the documentation files. For a complete documentation support each generated html document needs to get listed in the project helptoc file with toc entry.

Contents

Requirements

- Other m-files required: src/util/removeFilesFromDir.m
- Subfunctions: None
- MAT-files required: data/config.mat

See Also

- generateConfigMat
- publishFilesFromDir
- builddocsearchdb
- removeFilesFromDir
- Documentation Workflow

Created on September 21. 2020 by Tobias Wulf. Copyright Tobias Wulf 2020.

Start Publishing Script, Clean Up and Load Config

At first clean up junk from workspace and clear prompt for new output. Set project root path to create absolute file path with fullfile function. Load absolute path variables and publishing options from config.mat

```
disp('Workspace cleaned up ...');
clearvars;
clc;
disp('Load configuration ...');
try
    load('config.mat', 'PathVariables', 'PublishOptions');
catch ME
    rethrow(ME);
end
```

Remove Equation PNG Files

Remove equation png file from HTML output folder before create or recreate publishing files. To prevent the directory expanse of old or edited equation files.

```
yesno = input('Renew eqautions in docs [y/n]: ', 's');
if strcmp(yesno, 'y')
    removeFilesFromDir(PublishOptions.outputDir, '*_eq*.png');
end
```

Project Documentation Files

In this section of the publish script every bare documentation script should be handled and executed to publish. These are m-files without any executeable code so they exist just to transport the documentation content into html output. Dir all m-files from docs path. Not recursively but verbose. No expected directory tree search for m-files.

```
disp('Publish project documentation files ...');
publishFilesFromDir(PathVariables.docsPath, PublishOptions, false, true);
```

Executable Script Files

The section collects all ready to execute scripts from project scripts folder and publish them to html documentation folder. Every script must be notice in in Executable_Scripts.m file with one line description. That is very important to not execute the scripts during publishing. If a script contains critical or loop gaining code. In example the publishProjectFilesToHTML.m script such loop gaining code. If eval code during publishing is enabled the script starts publishing itself over and over again because it contains the loop entry via the publish function. So routine is minmal adjusted by evalCode parameter in PublishOptions struct. No expected directory to search for m-files so no recursively but verbose

```
disp('Publish executable scripts ...');
PublishOptions.evalCode = false;
publishFilesFromDir(PathVariables.scriptsPath, PublishOptions, false, true);
```

Source Code Functions and Classes

That part of the publish script collects function and class m-files from the util section of the source code located in src/. Introcude every new m-file to the source code related documentation m-file and add a description. In general functions and class files are not executed on publishing execution so set evalCode option to false in PublishOptions struct. In addition to that the source code itself should not be in the published document, so the showCode option is switched to false. Publish recursively from underlaying directory tree, verbose.

```
disp('Publish source code functions and classes ...');
PublishOptions.evalCode = false;
publishFilesFromDir(PathVariables.srcPath, PublishOptions, true, true);
```

Unit Test Scripts

Publish unit tests scripts and run test suite to include unit test results. It is the only section of whole publish process which executes script code. The test files are closed loop and do not harm other sections of source code.

```
disp('Publish unit tests scripts ...');
PublishOptions.evalCode = true;
publishFilesFromDir(PathVariables.unittestPath, ...
```

```
    PublishOptions, true, true);
```

Build Documentation Database for Matlab Help Browser

To support Matlabs help browser it is needed build searchable help browser entries including a searchable database backend. Matlabs built-in function builddocsearchdb does the trick. The function just needs the output directory of builded html documentation and it creates a subfolder which includes the database. About the info.xml from the project root and the helptoc.xml file the html documentation folder all listet documentation is accessable. At first remove old database before build the new reference database. Remove autogenerated directory helpsearch-v3. At first get folder content and remove first two relative directory entries from struct. Then delete files and check if files do not exist any more. At least build up new search database entries to Matlab help.

```
disp('Remove old search entries ...');
clearvars;
close all;
clc;
disp('Reload configuration after unit test execution ...');
try
    load('config.mat', 'PathVariables', 'PublishOptions');
catch ME
    rethrow(ME);
end
if removeFilesFromDir(PathVariables.helpsearchPath)
    builddocsearchdb(PublishOptions.outputDir);
else
    disp('Could not remove old search entries ...');
end
```

Open Generated Documentation.

Open generated HTML documentation from documentation root HTML file which should be a project introduction or project roadmap page. Comment out if this script is added to project shutdown tasks.

```
open(fullfile(PublishOptions.outputDir, ...
    'GaussianProcessDipoleSimulation.html'));
disp('Done ...');
```

Published with MATLAB® R2020b

generateConfigMat

Generate configuration mat-file wich contains reusable configuration to control the software or certain function parameters. Centralized collection of configuration. If it is certain configuration needed place it here.

Contents

Requirements

- Other m-files required: None
- Subfunctions: None
- MAT-files required: None

See Also

- save
- load
- matfile

Created on October 29. 2020 by Tobias Wulf. Copyright Tobias Wulf 2020.

Clean Up

Clear variables from workspace to build up a fresh new configuration workspace.

```
disp('Clean up workspace ...');
clearvars;
clc;
```

GeneralOptions

General options like formats for strings or date or anything else what has no special relation to a theme complex. Fix parameters.

```
disp('Set general options ...');
GeneralOptions = struct;
GeneralOptions.dateFormat = 'yyyy-mm-dd_HH-MM-SS-FFF';
```

Path Variables

Key path variables and directories, often used in functions or scripts. Collet the path in a struct for easier save the struct fields as variables to config.mat via -struct flag. Fix parameters.

```
disp('Create current project instance to gather information ...');

% create current project instance to retrieve root information
projectInstance = matlab.project.currentProject;

disp('Set path variables ...');
PathVariables =struct;

% project root path, needs to be recreated generic to work on
% different machines
PathVariables.rootPath = projectInstance.RootFolder;

% path to data folder, which contains datasets and config.mat
PathVariables.dataPath = fullfile(PathVariables.rootPath, 'data');

% path to TDK TAS2141 TMR angular sensor characterization dataset
PathVariables.tdkDatasetPath = fullfile(PathVariables.dataPath, ...
    'TDK_TAS2141_Characterization_2020-10-22_18-12-16-827.mat');

% path to TDK TAS2141 TMR angular sensor characterization dataset
PathVariables.kmz60DatasetPath = fullfile(PathVariables.dataPath, ...
    'NXP_KMZ60_Characterization_2020-12-03_16-53-16-721.mat');

% path to config file dataset
PathVariables.configPath = fullfile(PathVariables.dataPath, ...
    'config.mat');

% path to training dataset folder
PathVariables.trainingDataPath = fullfile(PathVariables.dataPath, ...
    'training');

% path to test dataset folder
PathVariables.testDataPath = fullfile(PathVariables.dataPath, ...
    'test');

% path to documentation and m-files only for documentation
PathVariables.docsPath = fullfile(PathVariables.rootPath, ...
    'docs');

% path to publish html documentation output directory, helptoc.xml location
PathVariables.publishHtmlPath = fullfile(PathVariables.docsPath, 'html');

% path to save plots as images svg, eps, png, etc.
PathVariables.saveImagesPath = fullfile(PathVariables.publishHtmlPath, ...
    'images');

% path to save matlab figures
PathVariables.saveFiguresPath = fullfile(PathVariables.publishHtmlPath, ...
    'figures');

% path to latex docs folder
PathVariables.latexDocsPath = fullfile(PathVariables.docsPath, ...
```

```
    'latex');

% path to latex Thesis Tobias Wulf (take care if comment in)
% PathVariables.thesisTobiasWulf = fullfile(PathVariables.latexDocsPath, ...
%     'BA_Thesis_Tobias_Wulf');

% path to docs export folder for Manual
PathVariables.exportPublishPath = fullfile(PathVariables.latexDocsPath, ...
    'Manual');

% path to style sheet for html documentation, Matlab provided style sheet
PathVariables.publishStyleSheetPath = fullfile(PathVariables.publishHtmlPath, ...
    'docsHtmlStyleSheet.xsl');

% path to documentation search database entries for Matlab help browser support
PathVariables.helpsearchPath = fullfile(PathVariables.publishHtmlPath, ...
    'helpsearch-v3');

% path to executable m-file scripts of the project
PathVariables.scriptsPath = fullfile(PathVariables.rootPath, 'scripts');

% path to source code files, function and class files
PathVariables.srcPath = fullfile(PathVariables.rootPath, 'src');

% path to unittest files, scripts and script suite
PathVariables.unittestPath = fullfile(PathVariables.rootPath, 'tests');
```

Publish Options

These are general options for documents to publish. They are passed to the matlab publish function via a struct where each option gets its own field. The option struct can be copied and adjusted for differing publish conditions in example for scripts, functions, and bare document m-files. Initialize the option struct with output format field name and field value and add further fields (options) with point value. Fix parameters.

```
disp('Set publish options struct for publish function ...');
PublishOptions = struct('format', 'html');
PublishOptions.outputDir = PathVariables.publishHtmlPath;
PublishOptions.stylesheet = PathVariables.publishStyleSheetPath;
PublishOptions.createThumbnail = false;
PublishOptions.figureSnapMethod = 'entireFigureWindow';
PublishOptions.imageFormat = 'png';
PublishOptions.maxHeight = [];
PublishOptions.maxWidth = [];
PublishOptions.useNewFigure = false;
PublishOptions.evalCode = false;
PublishOptions.catchError = true;
PublishOptions.codeToEvaluate = [];
PublishOptions.maxOutputLines = Inf;
PublishOptions.showCode = true;
```

Sensor Array Options

The options control the build up of the sensor array in geometry and techincal behavior. This means number of sensors in the array and its size in mm. The supply and offset voltage of each sensor which is needed for using the characterization which is normed in mV/V. These parameters should be fix during generation a pulk of training or test data sets. The simulation function does not covers vectors yet.

```
disp('Set sensor array option for geometry and behavior ...');
SensorArrayOptions = struct;

% Geometry of the sensor array current sensor array can be. Fix parameter.
% square - square sensor array with even distances to each sensor point
SensorArrayOptions.geometry = 'square';

% Sensor array square dimension. Fix parameter.
SensorArrayOptions.dimension = 8;

% Sensor array edge length in mm. Fix parameter.
SensorArrayOptions.edge = 2;

% Sensor array simulated supply voltage in volts. Fix parameter.
SensorArrayOptions.Vcc = 5;

% Sensor array simulated offset voltage for bridge outputs in volts. Fix
% paramter.
SensorArrayOptions.Voff = 2.5;

% Senor array voltage norm factor to recalculate norm bridge outputs to
% given supply voltage and offset voltage, current normin is mV/V which
% implements factor of 1e3. Fix paramter.
SensorArrayOptions.Vnorm = 1e3;
```

Dipole Options

Dipole options to calculate the magnetic field which stimulate the sensor array. The dipole is gained to sphere with additional z distance to the array by sphere radius. These parameters should be fix during generation a pulk of training or test data sets. The simulation function does not covers vectors yet.

```
disp('Set dipole options to calculate magnetic stimulus ...');
DipoleOptions = struct;

% Radius in mm of magnetic sphere in which the magnetic dipole is centered.
% So it can be seen as z-offset to the sensor array. Fix parameter.
DipoleOptions.sphereRadius = 2;

% H-field magnitude to multiply of generated and relative normed dipole
% H-fields, the norming is done in zero position of [0 0 z0 + sphere radius] for
% 0° due to the position of the magnetic moment [-1 0 0] x and y components
% are not relevant, norming without tilt. Magnitude in kA/m. The magnitude
% refers that the sphere magnet has this H-field magnitude in a certain distance
% z0 in example sphere with 2mm sphere radius has a H magnitude of 200kA/m in
% 5mm distance. Standard field strength for ferrite sphere magnets are between
% 180 and 200kA/m. Fix parameter.
DipoleOptions.H0mag = 200;

% Distance in zero position of the spherical magnet in which the imprinted
% H-field strength magnitude takes effect. Together with the sphere radius and
% and the imprinted field strength magnitude the distance in rest position
% characterizes the spherical magnet to later relative positions of the sensor
% array and generated dipole H-fields in rotation simulation. In mm. Fix
% parameter.
DipoleOptions.z0 = 1;

% Magnetic moment magnitude attach rotation to the dipole field at a
% certain position with x, y and z components. Choose a huge value to
```

```
% prevent numeric failures, by norming the factor is eleminated later. Fix
% parameter.
DipoleOptions.M0mag = 1e6;
```

Traning Options

Training options gives the software the needed information to generate training datasets by the sensor array simulation with a dipole manget as stimulus which pushed with an z offeset to a sphere.

```
disp('Set training options to generate dataset ...');
TrainingOptions = struct;

% Use case of options define what dataset it is and where to save resulting
% datasets by simulation function. Fix parameter.
TrainingOptions.useCase = 'Training';

% Sensor array relative position to dipole magnet as position vector with
% x, y and z posiotn in mm. Negative x for left shift, negative y for up
% shift and negative z to place the layer under the dipole decrease z to
% increase the distance. The z-position will be subtracted by dipole sphere
% radius in simulation. So there is an offset given by the sphere radius.
% Loop parameters.
TrainingOptions.xPos = [0,];
TrainingOptions.yPos = [0,];
TrainingOptions.zPos = [7,];

% Dipole tilt in z-axes in degree. Fix parameter.
TrainingOptions.tilt = 0;

% Resolution of rotaion in degree, use same resoultion in training and test
% datasets to have the ability to back reference the index to fullscale
% test data sets. In degree. Fix parameter.
TrainingOptions.angleRes = 0.5;

% Phase index applies a phase offset in the rotation, it is used as phase index
% to a down sampling to generate even distributed angles of a full scale
% rotation. Offset index of full rotation. In example a full scale rotation from
% 0° to 360° - angleRes returns 720 angles, if nAngles is set to 7 it returns 7
% angles [0, 51.5, 103, 154.5, 206, 257.5, 309]. To get a phase shift of 11° set
% phaseIndex to 22 a multiple of the resolution angleRes and get
% [11, 62.5, 114, 165.5, 217, 268.5, 320]. Must be positive integer. Fix
% parameter.
TrainingOptions.phaseIndex = 0;

% Number rotaion angles, even distribute between 0° and 360° with respect
% to the resolution, even down sampling. To generate full scale the number
% relatead to the resolution or fast generate but wrong number set it to 0 to
% generate full scale rotation too. Fix Parameter.
TrainingOptions.nAngles = 16;

% Charcterization datset to use in simulation. Current available datasets are
% TDK - for characterization dataset of TDK TAS2141 TMR sensor
% KMZ60 - for characterization dataset of NXP KMZ60 AMR sensor
TrainingOptions.BaseReference = 'TDK';

% Characteraztion field which should be load as refernce image from
% characterization data set, in TDK dataset are following fields. In the
% current dataset Rise has the widest linear plateau with a radius of ca.
```

```
% 8.5 kA/m. Fix parameter.
% Rise - Bridge outputs for rising stimulus amplituded
% Fall - Bridge outputs for falling stimulus amplitude
% All  - Superimposed bridge outputs
% Diff - Differentiated bridge outputs
TrainingOptions.BridgeReference = 'Rise';
```

Test Options

Test options gives the software the needed information to generate test datasets by the sensor array simulation with a dipole manget as stimulus which pushed with an z offeset to a sphere.

```
disp('Set test options to generate dataset ...');
TestOptions = struct;

% Use case of options define what dataset it is and where to save resulting
% datasets by simulation function. Fix Parameter.
TestOptions.useCase = 'Test';

% Sensor array relative position to dipole magnet as position vector with
% x, y and z posiotn in mm. Negative x for left shift, negative y for up
% shift and negative z to place the layer under the dipole decrease z to
% increase the distance. The z-position will be subtracted by dipole sphere
% radius in simulation. So there is an offset given by the sphere radius.
% Loop parameter.
TestOptions.xPos = [0, 1];
TestOptions.yPos = [0, 1];
TestOptions.zPos = [7,];

% Dipole tilt in z-axes in degree. Fix parameter.
TestOptions.tilt = 0;

% Resolution of rotaion in degree, use same resoultion in training and test
% datasets to have the ability to back reference the index to fullscale
% test data sets. In degree. Fix parameter.
TestOptions.angleRes = 0.5;

% Phase index applies a phase offset in the rotation, it is used as phase index
% to a down sampling to generate even distributed angles of a full scale
% rotation. Offset index of full rotation. In example a full scale rotation from
% 0° to 360° - angleRes returns 720 angles, if nAngles is set to 7 it returns 7
% angles [0, 51.5, 103, 154.5, 206, 257.5, 309]. To get a phase shift of 11° set
% phaseIndex to 22 a multiple of the resolution angleRes and get
% [11, 62.5, 114, 165.5, 217, 268.5, 320]. Must be positive integer. Fix
% parameter.
TestOptions.phaseIndex = 0;

% Number rotaion angles, even distribute between 0° and 360° with respect
% to the resolution, even down sampling. To generate full scale the number
% relatead to the resolution or fast generate but wrong number to 0 to
% generate full scale rotation. Fix parameter.
TestOptions.nAngles = 720;

% Charcterization datset to use in simulation. Current available datasets are
% TDK - for characterization dataset of TDK TAS2141 TMR sensor
% KMZ60 - for characterization dataset of NXP KMZ60 AMR sensor
TestOptions.BaseReference = 'TDK';
```

```
% Characteraztion field which should be load as refernce image from
% characterization data set, in TDK dataset are following fields. In the
% current dataset Rise has the widest linear plateau with a radius of ca.
% 8.5 kA/m. Fix parameter.
% Rise - Bridge outputs for rising stimulus amplituded
% Fall - Bridge outputs for falling stimulus amplitude
% All  - Superimposed bridge outputs
% Diff - Differentiated bridge outputs
TestOptions.BridgeReference = 'Rise';
```

Save Configuration

Save section wise each config part as struct to standalone variables in config.mat use newest save format with no compression. create config.mat with timestamp of creation

```
disp('Create config.mat ...');
timestamp = datestr(now, GeneralOptions.dateFormat);
save(PathVariables.configPath, ...
    'timestamp', ...
    'GeneralOptions', ...
    'PathVariables', ...
    'PublishOptions', ...
    'SensorArrayOptions', ...
    'DipoleOptions', ...
    'TrainingOptions', ...
    'TestOptions', ...
    '-v7.3', '-nocompression');
```

Published with MATLAB® R2020b

generateSimulationDatasets

Generate sensor array simulation datasets for training and test applications. Loads needed configurations from config.mat and characterization data from defined characterization dataset (current: PathVariables.tdkDatasetPath). Simulated dataset are saved to data/training and data/test path. Generate dataset for a predefined configuration at once. Best use is to generate simulation data, do wish application or evaluation on it and save results. Delete datasets, edit configuration and rerun for a new set of datasets.

Contents

Requirements

- Other m-files required: simulateDipoleSquareSensorArray.m
- Subfunctions: None
- MAT-files required: config.mat, TDK_TAS2141_Characterization_2020-10-22_18-12-16-827.mat

See Also

- sensorArraySimulation
- simulateDipoleSquareSensorArray
- generateConfigMat

Created on November 25. 2020 by Tobias Wulf. Copyright Tobias Wulf 2020.

Load Configuration and Characterization Dataset

Load configuration to generate dataset from config.mat and defined characterization dataset.

```
try
    clearvars;
    close all;
    disp('Load configuration ...');
    load('config.mat', 'GeneralOptions', 'PathVariables', ...
        'SensorArrayOptions', 'DipoleOptions', ...
        'TrainingOptions', 'TestOptions');
    disp('Load characterization dataset ...');
    switch TrainingOptions.BaseReference
        case 'TDK'
            TrainingCharDataset = load(PathVariables.tdkDatasetPath);
        case 'KMZ60'
            TrainingCharDataset = load(PathVariables.kmz60DatasetPath);
        otherwise
            error('Unknow characterization dataset in config.');
    end

    switch TestOptions.BaseReference
        case 'TDK'
```

```
            TestCharDataset = load(PathVariables.tdkDatasetPath);
        case 'KMZ60'
            TestCharDataset = load(PathVariables.kmz60DatasetPath);
        otherwise
            error('Unknow characterization dataset in config.');
    end

catch ME
    rethrow(ME)
end
```

Generate Training Datasets

Generate training dataset from configuration and characterization dataset.

```
disp('Generate training datasets ...');
simulateDipoleSquareSensorArray(GeneralOptions, PathVariables, ...
    SensorArrayOptions, DipoleOptions, TrainingOptions, TrainingCharDataset)
```

Generate Test Datasets

Generate test dataset from configuration and characterization dataset.

```
disp('Generate test datasets ...');
simulateDipoleSquareSensorArray(GeneralOptions, PathVariables, ...
    SensorArrayOptions, DipoleOptions, TestOptions, TestCharDataset)
```

Published with MATLAB® R2020b

deleteSimulationDatasets

Delete simulation dataset from data/training and data/test path at once.

Contents

Requirements

- Other m-files required: removeFilesFromDir.m
- Subfunctions: None
- MAT-files required: config.mat

See Also

- removeFilesFromDir
- gernerateConfigMat
- Project_Structure

Created on November 25. 2020 by Tobias Wulf. Copyright Tobias Wulf 2020.

Load Path to Clean Up

Load path from config.mat and where to find training and test datasets.

```
try
    clearvars;
    close all;
    load('config.mat', 'PathVariables')
    disp('Delete from ...')
    disp(PathVariables.trainingDataPath);
    disp(PathVariables.testDataPath);
catch ME
    rethrow(ME)
end
```

Delete Datasets

Delete datasets from training dataset path and test dataset path with certain file pattern.

```
answer = removeFilesFromDir(PathVariables.trainingDataPath, '*.mat');
fprintf('Delete training datasets: %s\n', string(answer));
answer = removeFilesFromDir(PathVariables.testDataPath, '*.mat');
fprintf('Delete test datasets: %s\n', string(answer));
```

Published with MATLAB® R2020b

deleteSimulationPlots

Delete plots of simulation dataset from figure and image path at once.

Contents

Requirements

- Other m-files required: removeFilesFromDir.m
- Subfunctions: None
- MAT-files required: config.mat

See Also

- removeFilesFromDir
- gernerateConfigMat
- Project_Structure

Created on November 02. 2020 by Tobias Wulf. Copyright Tobias Wulf 2020.

Load Path to Clean Up

Load path from config.mat and where to find training and test datasets.

```
try
    clearvars;
    close all;
    load('config.mat', 'PathVariables')
    disp('Delete from ...')
    fig = PathVariables.saveFiguresPath;
    svg = fullfile(PathVariables.saveImagesPath, 'svg');
    eps = fullfile(PathVariables.saveImagesPath, 'eps');
    pdf = fullfile(PathVariables.saveImagesPath, 'pdf');
    avi = fullfile(PathVariables.saveImagesPath, 'avi');
catch ME
    rethrow(ME)
end
```

Delete Dataset Plots

Delete datasets plots from image path and figure path with certain file pattern. path

```
pth = [fig svg eps pdf avi];
% extension
ext = ["fig" "svg" "eps" "pdf" "avi"];
% file patterns
pat = ["Test_*" "Training_*"];
```

```
for i = 1:length(pth)
    disp(pth(i));
    for p = pat
        asw = removeFilesFromDir(pth(i), join([p, ext(i)], "."));
        fprintf('Deleted pattern %s.%s %s\n', p, ext(i), string(asw));
    end
end
```

Published with MATLAB® R2020b

exportPublishedToPdf

Export Matlab generated HTML documentation (publish) to pdf-files and combine them into a latex index file ready compile to pdf manual. This script works on unix sytems only or needs to be adjusted for windows systems for library path and wkhtmltopdf binary path.

Contents

Requirements

- Other m-files required: src/util/removeFilesFromDir.m
- Subfunctions: wkhtmltopdf (shell), pdflatex (shell)
- MAT-files required: data/config.mat

See Also

- generateConfigMat
- system
- wkhtmltopdf
- publishProjectFilesToHTML
- Documentation Workflow

Created on December 10. 2020 by Tobias Wulf. Copyright Tobias Wulf 2020.

Start Exporting Script, Clean Up and Load Config

At first clean up junk from workspace and clear prompt for new output. Set project root path to create absolute file path with fullfile function. Load absolute path variables and publishing options from config.mat

```
disp('Workspace cleaned up ...');
clearvars;
clc;
disp('Load configuration ...');
try
    load('config.mat', 'PathVariables');
catch ME
    rethrow(ME);
end
```

Define Manual TOC

The maual toc must be in the same order as in helptoc.xml in the publish html folder. The toc is used to generate a latex file to

include for appendices.

```
toc = ["section",       "GaussianProcessDipoleSimulation.pdf";
       "subsection"     "Workflows.pdf";
       "subsubsection", "Project_Preparation.pdf";
       "subsubsection", "Project_Structure.pdf";
       "subsubsection", "Git_Feature_Branch_Workflow.pdf";
       "subsubsection", "Documentation_Workflow.pdf";
       "subsubsection", "Simulation_Workflow.pdf";
       "subsection",    "Executable_Scripts.pdf";
       "subsubsection", "publishProjectFilesToHTML.pdf";
       "subsubsection", "generateConfigMat.pdf";
       "subsubsection", "generateSimulationDatasets.pdf";
       "subsubsection", "deleteSimulationDatasets.pdf";
       "subsubsection", "deleteSimulationPlots.pdf";
       "subsubsection", "exportPublishedToPdf";
       "subsection",    "Source_Code.pdf";
       "subsubsection", "sensorArraySimulation.pdf";
       "paragraph",     "rotate3DVector.pdf";
       "paragraph",     "generateDipoleRotationMoments.pdf";
       "paragraph",     "generateSensorArraySquareGrid.pdf";
       "paragraph",     "computeDipoleH0Norm.pdf";
       "paragraph",     "computeDipoleHField.pdf";
       "paragraph",     "simulateDipoleSquareSensorArray.pdf";
       "subsubsection", "util.pdf";
       "paragraph",     "removeFilesFromDir.pdf";
       "paragraph",     "publishFilesFromDir.pdf";
       "paragraph",     "plotFunctions.pdf";
       "subparagraph",  "plotTDKCharDataset.pdf";
       "subparagraph",  "plotTDKCharField.pdf";
       "subparagraph",  "plotTDKTransferCurves.pdf";
       "subparagraph",  "plotKMZ60CharDataset.pdf";
       "subparagraph",  "plotKMZ60CharField.pdf";
       "subparagraph",  "plotKMZ60TransferCurves.pdf";
       "subparagraph",  "plotDipoleMagnet.pdf";
       "subparagraph",  "plotSimulationDataset.pdf";
       "subparagraph",  "plotSingleSimulationAngle.pdf";
       "subparagraph",  "plotSimulationSubset.pdf";
       "subparagraph",  "plotSimulationCosSinStats.pdf"
       "subparagraph",  "plotSimulationDatasetCircle.pdf";
       "subsection",    "Datasets.pdf";
       "subsubsection", "TDK_TAS2141_Characterization.pdf";
       "subsubsection", "NXP_KMZ60_Characterization.pdf";
       "subsubsection", "Config_Mat.pdf";
       "subsubsection", "Training_and_Test_Datasets.pdf";
       "subsection",    "Unit_Tests.pdf";
       "subsubsection", "runTests.pdf";
       "subsubsection", "removeFilesFromDirTest.pdf";
       "subsubsection", "rotate3DVectorTest.pdf";
       "subsubsection", "generateDipoleRotationMomentsTest.pdf";
       "subsubsection", "generateSensorArraySquareGridTest.pdf";
       "subsubsection", "computeDipoleH0NormTest.pdf";
       "subsubsection", "computeDipoleHFieldTest.pdf";
       "subsubsection", "tiltRotationTest.pdf";];

nToc = length(toc);
fprintf("%d toc entries remarked ...\n", nToc);
```

Write TOC to LaTeX File

Wirete TOC to latex file and generate for each pdf to include a toc content line with marked toc depth.

```
disp('Write TOC to Manual.tex ...');
%addPdfStr = "\\includepdf[page=-, pagecommand={\\phantomsection" + ...
%    "\\addcontentsline{toc}{%s}{%s}}]{%s}\n";
fileID = fopen(fullfile(...
    PathVariables.exportPublishPath, 'Manual.tex'), 'w');
% fprintf(fileID, "%% !TEX root = ../thesis.tex\n");
fprintf(fileID, "%% appendix software documentation\n");
fprintf(fileID, "%% @author Tobias Wulf\n");
fprintf(fileID, ...
    "%% Autogenerated LaTeX file. Generated by exportPublishedToPdf.\n");
fprintf(fileID, ...
    "%% Software manual with TOC generated in the same script.\n");
fprintf(fileID, "%% Generated on %s.\n\n", datestr(datetime('now')));
for i = 1:nToc
    level = toc(i);
    fName = toc(i,2);
    [~, titleStr, ~] = fileparts(fName);
    titleStr = strrep(titleStr, '_', ' ');
    fprintf(fileID, addPdfStr, level, titleStr, fName);
end
fclose(fileID);
```

Scan for HTML Files

Scan for all published HTML files in the project publish directory.

```
disp('Scan for published files ...');
HTML = dir(fullfile(PathVariables.publishHtmlPath, '*.html'));
if nToc ~= length(HTML)
    warning(...
        'TOC (%d) length and found HTML (%d) files are diverging.', ...
        nToc, length(HTML));
end
```

Export HTML to Pdf

Export found HTML files to Pdf files. Each file gets its own Pdf represenstation. Filename is kept with pdf extension. Write files into Manual folder under latex subdirectory in docs path. Using wkhtmltopdf shell application. Get filename, add pdf extension new path to file. Create shell string to execute with system command. Get current library path (Matlab) and change it to system library path to execute wkhtmltopdf after that restor library back to Matlab.

```
disp('Change local library path to system path ...');
matlabLibPath = getenv('LD_LIBRARY_PATH');
systemLibPath = '/usr/lib/x86_64-linux-gnu';
setenv('LD_LIBRARY_PATH', systemLibPath);

disp('Export published HTML to Pdf ...');
fprintf('Source: %s\n', HTML(1).folder);
fprintf('Destination: %s\n', PathVariables.exportPublishPath);
for fhtml = HTML'
    disp(fhtml.name);
    [~, fName, ~] = fileparts(fhtml.name);
```

```
    sourcePath = fullfile(fhtml.folder, fhtml.name);
    destinationPath = fullfile(...
        PathVariables.exportPublishPath, [fName '.pdf']);

    cmdStr = join(["wkhtmltopdf", ...
        ..."-B 57mm", ...
        ..."-L 30mm", ...
        ..."-R 30mm", ...
        ..."-T 37mm", ...
        "--minimum-font-size 11", ...
        "--enable-local-file-access", ...
        "--disable-external-links", ...
        "--disable-internal-links", ...
        ..."--disable-smart-shrinking", ...
        "--window-status finished", ...
        "--no-stop-slow-scripts", ...
        "--javascript-delay 2000", ...
        "%s %s"]);
    shellStr = sprintf(cmdStr, sourcePath, destinationPath);

    try
        [status, cmdout] = system(shellStr);
        % disp(cmdout);
        if status ~= 0
            error('Export failure.');
        end
    catch ME
        %setenv('LD_LIBRARY_PATH', matlabLibPath);
        disp(cmdout);
        rethrow(ME)
    end
end

disp('Restore local library path ...');
setenv('LD_LIBRARY_PATH', matlabLibPath);
```

Published with MATLAB® R2020b

Source Code

The project source code is clustered in modules where every subdirectory represents one certain module. Each module gathers functions and classes which are related to module specific themes or task fields. So the basic structured source code is located here. The combination of module functionality takes place in executable area of the project. So use the functions and classes in scripts and further on compiled binaries. Do not write bare executable source code here. For reproducable results and source code tracebillity each module has its own documentation entry where all underlaying functions and classses are listed. The best practice to develop new source code or modules is to do it in test driven way. This means write a test m-file for every new function or class m-file and test the functionality of the source code with assertion. This test driven development is called unittest and provides in combination with detailed documentation a high percentage of reusable source code.

Contents

sensorArraySimulation

Function space to solve sensor array simulation with a certain magnetic stimulus. The Array simulation is based on the TDK TAS2141 characterization dataset. A magnetic dipole is used as basic magnetic stimulus and moved as imaginary sphere magnet with an certain radius tained as dipole with offset radius. The magnet rotate in z-direction counterclockwise.

util

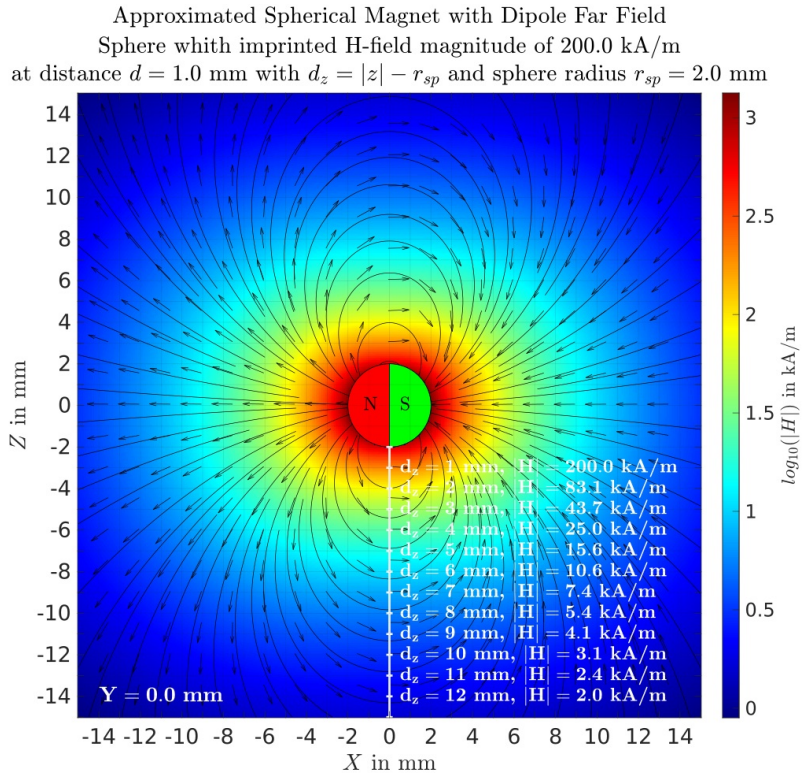
Util function and classes to provide reuse for often upcommings tasks and functionality besides project kernel and module source code. Located under source code directory: **./src/util**.

Published with MATLAB® R2020b

sensorArraySimulation

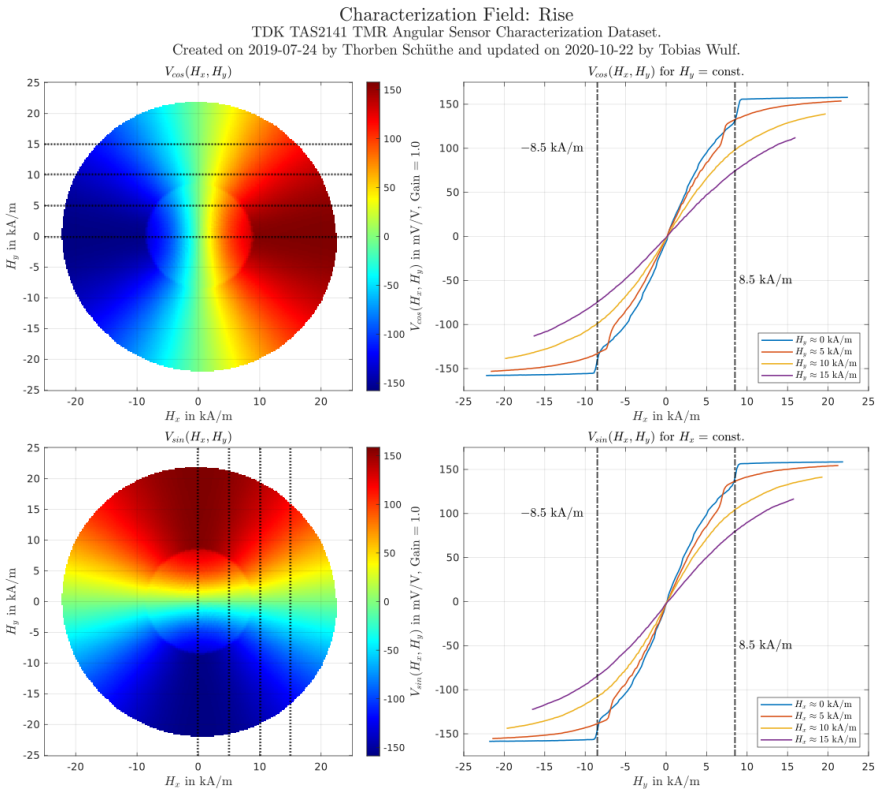
A spherical magnet is assumed to be used for stimulation of the sensor array. The far field of a spherical magnet can be approximately described by the magnetic field of a magnetic dipole. The magnetization of the sphere is assumed to be in y direction and the magnetic moment in rest position for 0° points in x direction. The magnet must be defined in a way that its field lines or field strengths own gradients sufficiently strong enough in the distance to the sensor array and so the rotation of the magnet generates a small scattering of the bridge outputs in the individual sensor points in the array. That all sensors in the array approximately perceive the same magnetic field gradients of the current rotation step and the sensors in the array run through approximately equal circular paths in the characterization field. This means the sperical magnet is characterized by a favorable mating of sphere radius and a certain distance in rest position in which a sufficiently high field strength takes effect. Here are neglected small necessary distances which are demanded in standard automotive applications. The focus here is on to generate simulation datasets, which are uniform and valid for angle detection. The modelling of suitable small magnets is not taking place of the work.

A good working magnet is found emperical for H-field magnitudes of 200 kA/m and a distance from surface of 1 mm. See below figure of used magnet.

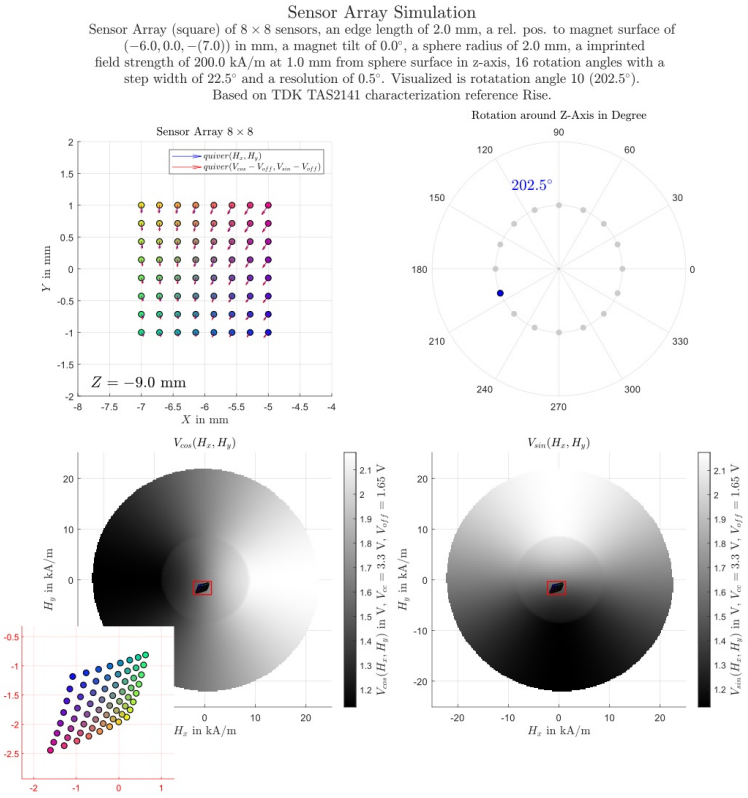


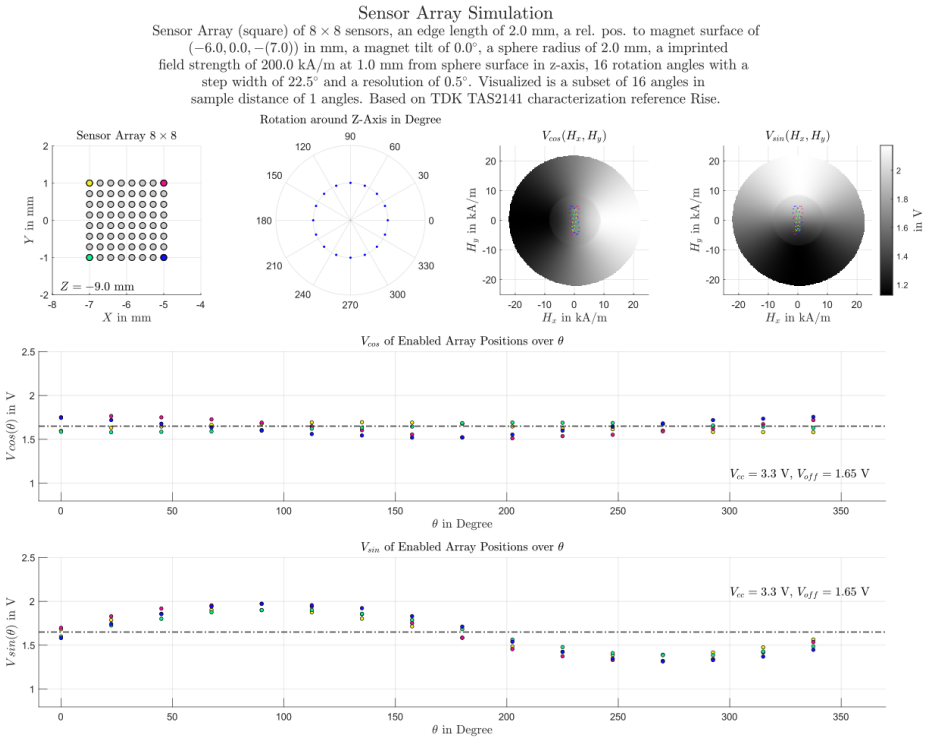
To change settings for simulation edit the config script and rerun it. To generate trainging and test data set use simulation script. It generates dataset for all position are known to TrainingOptions and TestOptions in config. Generate a set of dataset for one evaluation case. Evaluate datasets, save results for later clustering, edit config for next use case and rerun simulation.

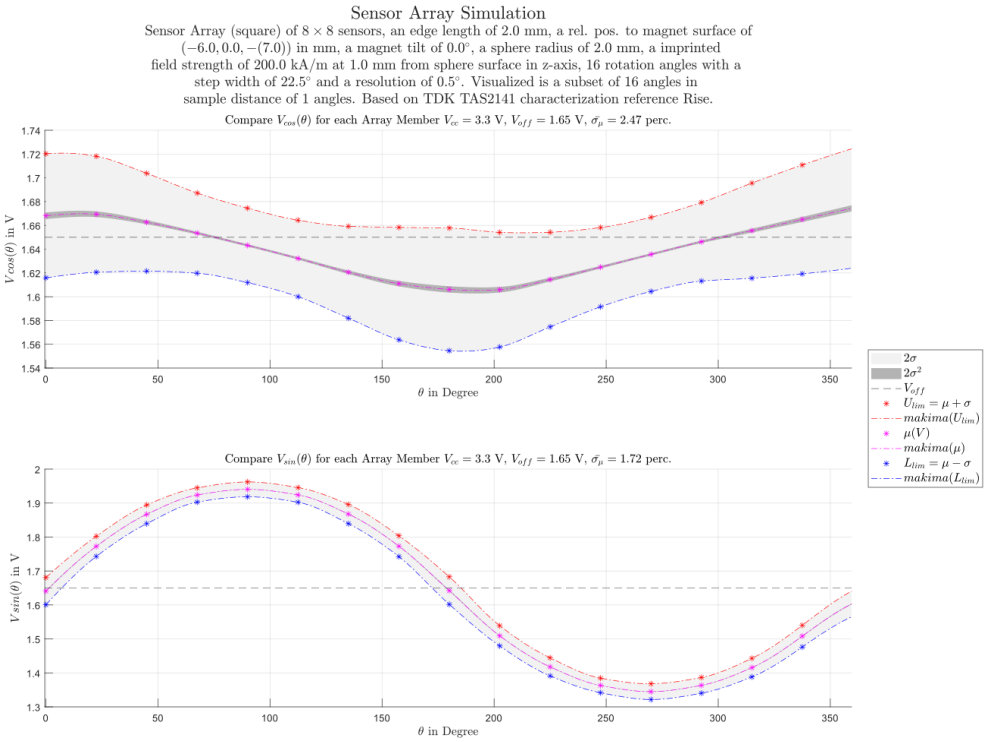
The simulation bases on TDK TAS2141 "Rise" chracterization field. It has the widest linear plateau for corresponding Hx and Hy field strengths.

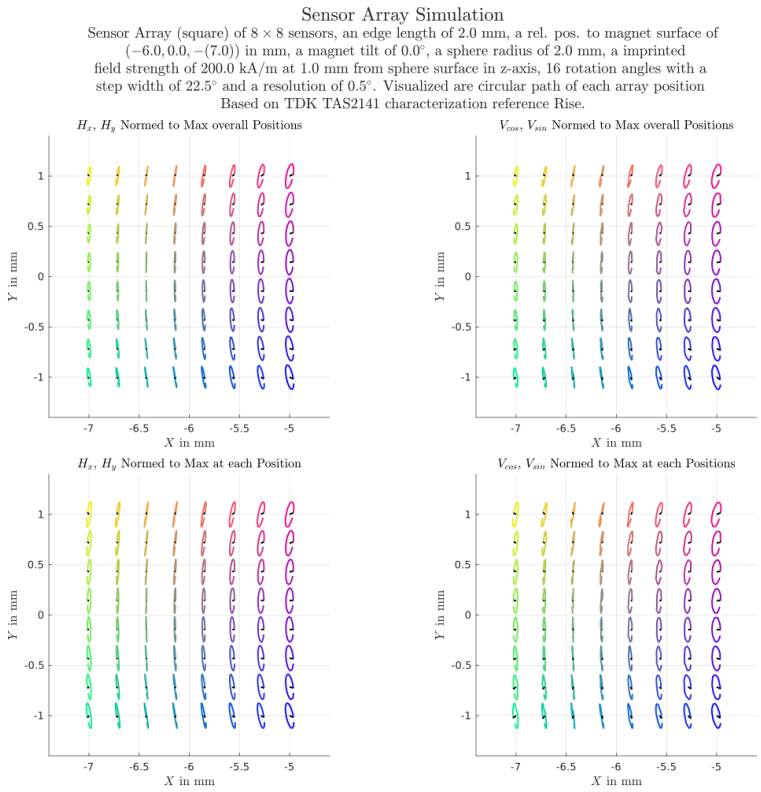


Here are some example plots of sensor array simulation with described magnet configuration from above and TDK Rise characterzation field as behavior base.


Sensor Array Simulation
Sensor Array (square) of 8 × 8 sensors, an edge length of 2.0 mm, a rel. pos. to magnet surface of (−6.0, 0.0, −(7.0)) in mm, a magnet tilt of 0.0°, a sphere radius of 2.0 mm, a imprinted field strength of 200.0 kA/m at 1.0 mm from sphere surface in z-axis, 16 rotation angles with a step width of 22.5° and a resolution of 0.5°. Visualized is rotatation angle 10 (202.5°).
Based on TDK TAS2141 characterization reference Rise.
Sensor Array 8 × 8
Z = −9.0 mm
X in mm
Y in mm
Rotation around Z-Axis in Degree
202.5°
$V_{cos}(H_x, H_y)$
$V_{sin}(H_x, H_y)$
H_x in kA/m
H_y in kA/m
$V_{cos}(H_x, H_y)$ in V, $V_{cc} = 3.3$ V, $V_{off} = 1.65$ V
$V_{sin}(H_x, H_y)$ in V, $V_{cc} = 3.3$ V, $V_{off} = 1.65$ V


Sensor Array Simulation
Sensor Array (square) of 8 × 8 sensors, an edge length of 2.0 mm, a rel. pos. to magnet surface of (−6.0, 0.0, −(7.0)) in mm, a magnet tilt of 0.0°, a sphere radius of 2.0 mm, a imprinted field strength of 200.0 kA/m at 1.0 mm from sphere surface in z-axis, 16 rotation angles with a step width of 22.5° and a resolution of 0.5°. Visualized is a subset of 16 angles in sample distance of 1 angles. Based on TDK TAS2141 characterization reference Rise.
Rotation around Z-Axis in Degree
Sensor Array 8 × 8
Y in mm
X in mm
Z = −9.0 mm
$V_{cos}(H_x, H_y)$
$V_{sin}(H_x, H_y)$
H_y in kA/m
H_x in kA/m
in V
V_{cos} of Enabled Array Positions over θ
$V cos(\theta)$ in V
$V_{cc} = 3.3$ V, $V_{off} = 1.65$ V
θ in Degree
V_{sin} of Enabled Array Positions over θ
$V sin(\theta)$ in V
$V_{cc} = 3.3$ V, $V_{off} = 1.65$ V
θ in Degree


Sensor Array Simulation
Sensor Array (square) of 8 × 8 sensors, an edge length of 2.0 mm, a rel. pos. to magnet surface of (−6.0, 0.0, −(7.0)) in mm, a magnet tilt of 0.0°, a sphere radius of 2.0 mm, a imprinted field strength of 200.0 kA/m at 1.0 mm from sphere surface in z-axis, 16 rotation angles with a step width of 22.5° and a resolution of 0.5°. Visualized is a subset of 16 angles in sample distance of 1 angles. Based on TDK TAS2141 characterization reference Rise.
Compare $V_{cos}(\theta)$ for each Array Member $V_{cc} = 3.3$ V, $V_{off} = 1.65$ V, $\bar{\sigma_\mu} = 2.47$ perc.
$V\,cos(\theta)$ in V
θ in Degree
Compare $V_{sin}(\theta)$ for each Array Member $V_{cc} = 3.3$ V, $V_{off} = 1.65$ V, $\bar{\sigma_\mu} = 1.72$ perc.
$V\,sin(\theta)$ in V
θ in Degree
2σ
$2\sigma^2$
V_{off}
$U_{lim} = \mu + \sigma$
$makima(U_{lim})$
$\mu(V)$
$makima(\mu)$
$L_{lim} = \mu - \sigma$
$makima(L_{lim})$



Contents

See Also

- generateConfigMat
- generateSimulationDatasets
- deleteSimulationDatasets

simulateDipoleSquareSenorArray

Simulate a square sensor array with dipole magnet as stimulus for a certain setup of training or test options. Saves generated dataset to data/training or data/test.

computeDipoleHField

Compute dipole field strength for meshgrids with additional abillity to imprint a certain field strength in defined radius on

resulting field.

computeDipoleH0Norm

Compute a norm factor to imprint a magnetic field strength to magnetic dipole fields with same magnetic moment magnitude and constant dipole sphere radius on which the imprinted field strengt takes effect.

generateSensorArraySquareGrid

Generat a square sensor array grid in a 3D coordinate system with relative position to center of the system and an additional offset in z direction.

generateDipoleRotationMoments

Generate magnetic rotation moments to rotate a magnetic dipol in its z-axes with a certain tilt.

rotate3DVector

Rotate a vector with x-, y- and z-components in a 3D-coordinate system. Rotate one step of certain angles.

Created on November 04. 2020 by Tobias Wulf. Copyright Tobias Wulf 2020.

Published with MATLAB® R2020b

rotate3DVector

Rotate a 3 dimensional vector with x-, y- and z-components in 3 dimensional coordinate system along the x-, y- and z-axes. Using rotation matrix for x-, y- and z-axes. Angle must be served in degree. Vector must be a column vector 3 x 1 or matrix related x-, y-, z-components 3 x N.

This function was originally created by Thorben Schüthe is ported into source code under improvements and including Matlab built-in functions. Function rewritten.

Contents

Syntax

```
rotated = rotate3DVector(vector, alphaX, betaY, gammaZ)
```

Description

rotated = rotate3DVector(vector, alphaX, betaY, gammaZ) returns a rotated vector which is rotated by given angles on related axes. alphaX rotates along the x-axes, betaY along the y-axes and gammaZ along the z-axes. Therfore each rotations is described by belonging rotation matrix. The resulting rotation of the vector is computed by the matrix and vector multiplacation of the rotation matrices and the input vecotor.

$$v' = Av = R_z(\gamma)R_y(\beta)R_x(\alpha)v$$

Examples

```
% rotate a vector along z-axes by 45°
vector = [1; 0; 0]
rotated = rotate3DVector(vector, 0, 0, 45)

% rotate a vector along z-axes by 35° with a tilt in x-axes by 1°
vector = [1; 0; 0]
rotated = rotate3DVector(vector, 1, 0, 35)

% rotate a vector along z-axes by 35° with a tilt in x-axes by 1° and a
% tilt in y-axes by 5°
vector = [1; 0; 0]
rotated = rotate3DVector(vector, 1, 5, 35)
```

Input Arguments

vector is a 3 x N column vector of real numbers which representates the a vector in a 3D coordinate system with x-, y- and z-components.

alphaX is a scalar angular value in degree and rotates the vector in the x-axes.

betaY is a scalar angular value in degree and rotates the vector in the y-axes.

gammaZ is a scalar angular value in degree and rotates the vector in the z-axes.

Output Arguments

rotated is rotation of vector by passed axes related angles.

Requirements

- Other m-files required: None
- Subfunctions: rotx, roty, rotz
- MAT-files required: None

See Also

- rotx
- roty
- rotz
- Wikipedia Drehmatrix

Created on August 03. 2016 by Thorben Schüthe. Copyright Thorben Schüthe 2016.

```
function [rotated] = rotate3DVector(vector, alphaX, betaY, gammaZ)
    arguments
        % validate as vecotor or matrix of size 3 x N
        vector (3,:) double {mustBeReal}
        % validate angles as scalar
        alphaX (1,1) double {mustBeReal}
        betaY (1,1) double {mustBeReal}
        gammaZ (1,1) double {mustBeReal}
    end

    % rotate vector or vector field as 3 x N matrix counterclockwise by given
    % angles along axes, calculate rotation matrices for each axes and
    % multiplicate with input vector
    rotated = rotz(gammaZ) * roty(betaY) * rotx(alphaX) * vector(:, 1:end);
end
```

Published with MATLAB® R2020b

generateDipoleRotationMoments

Generate magnetic moments to perform a full rotation of a magnetic dipole in the z-axes with a certain tilt. The moments covers a rotation from 0° to 360° and are eqaul distributed between 0° and 360°. 0° and 360° are related to the first moment which is representated by the start vector of

$$\overrightarrow{m_0} = |m_0| \cdot \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix}$$

Due to the start vecotor position the tilt of z-axes must be applied with a tilt angle in y-axes. So the rotated vector of the start moment is described by

$$\overrightarrow{m_i} = R_z(\theta_i) R_y(\phi) R_x(0^\circ) \overrightarrow{m_0}$$

The returning Moments matrix is 3 x N matrix where each moment vector

$$\vec{M} = \left[\overrightarrow{m_i} \cdots \overrightarrow{m_N} \right]$$

corresponds to a i-th angle in 1 x N thetas vector.

$$\vec{\theta} = [\theta_i \cdots \theta_N]$$

for

$$i = 1 \cdots N$$

The resolution of the angles can be modified additionally. At first the full angle vector theta is fully generated with given resolution and downsampled afterwards to the defined number of angles. On the resultating theta vector is base of magnetical moments.

Contents

Syntax

```
M = generateDipoleRotationMoments(m0, nTheta)
[M, thetas] = generateDipoleRotationMoments(m0, nTheta)
[M, thetas] = generateDipoleRotationMoments(m0, nTheta, phi)
[M, thetas] = generateDipoleRotationMoments(m0, nTheta, phi, resolution)
[M, thetas, index] = generateDipoleRotationMoments(m0, nTheta, phi, resolution, phaseIndex)
```

Description

M = generateDipoleRotationMoments(m0, nTheta) generate magnetic moments for N numbers of rotation angles theta in 3 x N sized matrix. With a default angle resoulution of 1° and a start angle of 0°.

[M, theta] = generateDipoleRotationMoments(m0, nTheta) returns so magnetic moments as before and related angles theta as 1 x N vector.

[M, theta] = generateDipoleRotationMoments(m0, nTheta, phi) generate magnetic moments for a rotation with a tilt angle phi.

[M, theta] = generateDipoleRotationMoments(m0, nTheta, phi, resolution) return moments and angles like described above but with given resolution in degree. The resolution is used in generation of full scale rotation angle base and sometime not visible in the output caused by the number of angles. So which angle are even picked from full scale rotation to compute a down sampled set of angles.

[M, theta, index] = generateDipoleRotationMoments(m0, nTheta, phi, resolution, phaseIndex)returns the moments, the angles and index reprensetation of down sampled angles in the full scale rotation vectort.

Examples

```
% choose a huge moment amplitude to withdraw numeric erros in later H-field
% strength calculations
m0 = 1e6;

% get a full scale (FS) rotation of with 0.5° resolution and no tilt
[MFS, thetaFS] = generateDipolRotationMoments(m0, 0, 0, 0.5);

% get down sampled (DS) rotation with equal distanced angles of the same full
% scale and refered index to the full scale. 8 angles.
[MDS, thetaDS, iFS] = generateDipolRotationMoments(m0, 8, 0, 0.5);

% check distribution to full scale must be true if distribution is correct
all(MFS(iFS) == MDS)
all(thetaFS(iFS) == thetaDS)

% now shift the sample pick by 22 samples (11° with resolution of 0.5°)
[MDSS, thetaDSS] = generateDipolRotationMoments(m0, 8, 0, 0.5, 22);

% check with index shift by 22 in iFS index
all(MFS(iFS + 22) == MDSS)
all(thetaFS(iFS + 22) == thetaDSS)
```

Input Arguments

m0 scalar value of magnetic moment magnitude. Choose huge value to prevent numeric failures in later field strength calculation. 1e6 is a proven value. The is later normated in the field calculation process. Can be any real number.

nTheat scalar value and number of angles which are even picked from the full rotation to produce smaller rotatation datasets. Must be a positive integer or zero. If zero the full scale rotation is returned.

phi scalar angule in degree to tilt the z-axes of the rotation. Can be any real number. Default is 0°.

resolution scalar angle resolution must be real positive number and probably smaller than 360°. Default is 1°.

phaseIndex scalar integer number to shift the start index of down sampling the full scale rotation. Therfore nTheta must be greater than 0. Default is 0.

Output Arguments

M matrix of magnetic moments related to vector theta. Matrix of size 3 x N.

theta related angles to calculated magnetic moments in a row vector of size 1 x N.

index reference to full scale angle vector. Empty if nTheta is zero and theta is the full scale vector.

Requirements

- Other m-files required: rotate3DVector.m
- Subfunctions: length, downsample, ismember, find
- MAT-files required: None

See Also

- rotate3DVector
- downsample
- ismember
- find

Created on November 06. 2020 by Tobias Wulf. Copyright Tobias Wulf 2020.

```
function [M, theta, index] = generateDipoleRotationMoments(m0, nTheta, ...
    phi, resolution, phaseIndex)
    arguments
        % validate amplitude of magnetic moment as real scalar value
        m0 (1,1) double {mustBeReal}
        % validate number of used angulars as positive integer, for 0 return all
        nTheta (1,1) double {mustBeNonnegative, mustBeInteger}
        % validate tilt angle as real value with default 0°
        phi (1,1) double {mustBeReal} = 0
        % validate angle resolution as real positive value
        resolution (1,1) double {mustBePositive} = 1
        % validate downsample phase as positive integer with default 0, no shift
        phaseIndex (1,1) double {mustBeNonnegative, mustBeInteger} = 0
    end

    % scale full rotation angle vector with given resolution from 0° to 360°
    % so run to 360°-resolution because 0° == 360°, its a circle
    fullScale = 0:resolution:(360 - resolution);

    % if nThetas is greater than 0 downsample to nTheta else use full scale
    if nTheta
        % get equal distribute distance of samples in thetas for nThetas
        sampleDistance = length(downsample(fullScale, nTheta));

        % downsample with equal sample distance and passed sample phase to shift
        % first sample in downsample vector from 1 to phaseIndex
        theta = downsample(fullScale, sampleDistance, phaseIndex);

        % find index members of down sampled angles in full scale vector
        members = ismember(fullScale, theta);
        index = find(members);

    else
```

```
        % 0 is given for number of theta so it returns the full scale rotation
        % no index relations if full scale is returned
        nTheta = length(fullScale);
        theta = fullScale;
        index = [];
    end

    % create start moment with given magnetic moment amplitude basic moment to
    % produce rotate moments
    m0 = m0 * [-1; 0; 0];

    % allocate memory for the moments Matrix of rotated basic moments by i-th
    % theta and fixed tilt of phi and rotate of theta angulars
    M = zeros(3, nTheta);
    for i = 1:nTheta
        M(:,i) = rotate3DVector(m0, 0, phi, theta(i));
    end
end
```

Published with MATLAB® R2020b

generateSensorSquareArrayGrid

Generate a position grid of sensors in x, y and z dimension. So the function returns a grid in shape of a square in which all sensors have even distances to each and another in x and y direction z is constant due to that all sensor are in the same distance to the .

The size of the sensor array is described by its edge length a

$$A = a^2$$

and the distance d of each coordinate to the next point in x and y direction

$$d = \frac{a}{N-1}$$

The coordinates of the array are scale from center of the square. So for the upper left corner position is described by

$$x_{1,1} = -\frac{a}{2} \quad y_{1,1} = \frac{a}{2} \quad z = const.$$

The coodinates of each dimension are placed in matrices of size N x N related to the number of sensors at one edge of the square Array. So position pattern in x dimension are returned as

$$X_0 = \begin{bmatrix} x_{1,1} & \cdots & x_{1,N} \\ \vdots & \ddots & \vdots \\ x_{N,1} & \cdots & x_{N,N} \end{bmatrix}$$

$$x_{i,j} = x_{1,1} + j \cdot d - d$$

same wise for y dimension but transposed

$$Y_0 = \begin{bmatrix} y_{1,1} & \cdots & y_{1,N} \\ \vdots & \ddots & \vdots \\ y_{N,1} & \cdots & y_{N,N} \end{bmatrix}$$

$$y_{i,j} = y_{1,1} - i \cdot d + d$$

$$Y_0 = -X_0^T$$

and z dimension

$$Z_0 = \begin{bmatrix} z_{1,1} & \cdots & z_{1,N} \\ \vdots & \ddots & \vdots \\ z_{N,1} & \cdots & z_{N,N} \end{bmatrix}$$

$$z_{i,j} = 0$$

for

$i = 1,2,\ldots,N \quad j = 1,2,\ldots,N$

A relative position shift can be performed by pass a postion vector p with relativ position to center

$$\vec{p} = \begin{bmatrix} x_p \\ y_p \\ z_p \end{bmatrix}$$

So that a left shift in x direction relative to the magnet in the center of the coordinate system is done by negative values for p(1) and a up in shift in y direction is performed by positive values for p(2). To gain distance in z from center point so the magnet is above the z layer of the sensor array increase the z positive. In addition to the z shift an offset r sphere can be set. The offset represents the radius of a sphere magnet in which center the dipole is placed. The dipole is placed in the center of the coordinate system and sensor array position is relative to the dipole or center. So shifts are described by

$$X = X_0 + x_p \quad Y = Y_0 + y_p \quad Z = Z_0 - (z_p + r_{sp})$$

Contents

Syntax

```
[X, Y, Z] = generateSensorArrayGrid(N, a, p, r)
```

Description

[X, Y, Z] = generateSensorArrayGrid(N, a, p, r) returns a sensor array grid of size N x N with grid position matrices for x, y and z positions of each sensor in the array.

Examples

```
% generate a grid of 8 x 8 sensors with no shift in x or y direction
and a static position of 4mm under the center in z dimension with a
z offset of 2mm so (2 + 2)mm
N = 8;
p = [0, 0, 2]
r = 2;
[X, Y, Z] = generateSensorArrayGrid(N, a, p, r);

% same layer but left shift by 2mm and down shift in y by 1mm
p = [-2, 1, 2]
r = 2;
[X, Y, Z] = generateSensorArrayGrid(N, a, p, r);
```

Input Argurments

N positive integer scalar number of sensors at one edge of the square grid. So the resulting grid has dimensions N x N.

a positive real scalar value of sensor array edge length.

p relative position vector, relative sensor array postion to center of the array. Place the array in 3D coorodinate system relative to the center of system.

r positive real scalar is offset in z dimension and represents the sphere radius in which center the magnetic dipole is placed.

Output Argurments

X x coordinates for each sensor in N x N matrix where each point has the same orientation as in y and z dimension.

Y y coordinates for each sensor in N x N matrix where each point has the same orientation as in x and z dimension.

Z z coordinates for each sensor in N x N matrix where each point has the same orientation as in x and y dimension.

Requirements

- Other m-files required: None
- Subfunctions: meshgrid
- MAT-files required: None

See Also

- meshgrid

Created on November 10. 2020 by Tobias Wulf. Copyright Tobias Wulf 2020.

```
function [X, Y, Z] = generateSensorArraySquareGrid(N, a, p, r)
    arguments
        % validate N as positive integer
        N (1,1) double {mustBePositive, mustBeInteger}
        % validate array edge length as positive scalar
        a (1,1) double {mustBeReal, mustBePositive}
        % validate p as column vector of real scalars
        p (3,1) double {mustBeReal, mustBeVector}
        % validate r as real scalar
        r (1,1) double {mustBeReal}
    end

    % half edge length for square corners
    aHalf = a / 2;

    % distance in x and y direction of each coordinate to next point
    d = a / (N - 1);

    % grid vector for x and y coordinates z is constant layer with shifts
    x = (-aHalf:d:aHalf) + p(1);
    y = (aHalf:-d:-aHalf) + p(2);
    z = -(p(3) + r);

    % scale grid in x, y dimension with constant z dimension
    [X, Y, Z] = meshgrid(x, y, z);
end
```

Published with MATLAB® R2020b

computeDipoleH0Norm

Compute the norm factor for magnetic field generated by an Dipole in its zero position. That means the maximum H-field magnitude in zero position with no position shifts in x or y direction. So that norm factor is related to the center point of coordinate system in x and y direction and to the dipoles initial z position. Which can be seen as sphere magnet for far field of the sphere. The norm relates that a dipole magnet in center of a sphere with a radius has certain field strength in related distance. In example a sphere of 2mm radius has in 5mm distance a field strength of 200kA/m

It is simplified computation for the dipole equation for one position in inital state without tilt in z-axes tor bring on a free choosen field strength to define the magnet. Because far field of sphere can be seen as dipole.

$$\vec{H_0}(\vec{r_0}) = \frac{1}{4\pi} \cdot \left(\frac{3\vec{r_0}\left(\vec{m_0}^T \vec{r_0}\right)}{|\vec{r_0}|^5} - \frac{\vec{m_0}}{|\vec{r_0}|^3} \right)$$

$$H_{0norm} = \frac{H_{mag}}{|H_0(r_0)|}$$

Contents

Syntax

```
H0norm = functionName(Hmag, m0, r0)
```

Description

H0norm = functionName(Hmag, m0, r0) compute scalar norm factor related to dipole rest position. Multiply that factor to dipole generated fields which are computed with the same magnetic moment magnitude to imprint a choosen magnetic field strength magnitude on the dipole field rotation.

Examples

```
% distance where the magnetic field strength is the value of wished
% magnitude, in mm
r0 = [0; 0; -5]
% field strength to imprint in norm factor in kA/m
Hmag = 200
% magnetic moment magnitude which is used generate rotation moments
m0 = [-1e6; 0; 0]
% compute norm factor for dipole rest position
H0norm = computeDipoleH0Norm(Hmag, m0, r0)
```

Input Argurments

Hmag real scalar of H-field strength magnitude to imprint in norm factor to define a dipole sphere with constant radius and field strength at this radius.

m0 vector of magnetic moment magnitude which must be same as for later roatition of the dipole.

r0 vector of distance in rest position of magnet center.

Output Argurments

H0norm real scalar of norm factor which relates to the zero position of the dipole sphere and can be multiplied to generated dipole H-field to imprint a magnetic field strength relative to the position of sensor array. The imprinted field strength magnitude relates to the rest position z0 + rsp.

Requirements

- Other m-files required: None
- Subfunctions: None
- MAT-files required: None

See Also

- rotate3DVector
- generateDipoleRotationMoments
- Wikipedia Magnetic Dipole

Created on November 11. 2020 by Tobias Wulf. Copyright Tobias Wulf 2020.

```
function [H0norm] = computeDipoleH0Norm(Hmag, m0, r0)
    arguments
        % validate inputs as real scalars
        Hmag (1,1) double {mustBeReal}
        m0 (3,1) double {mustBeReal, mustBeVector}
        r0 (3,1) double {mustBeReal, mustBeVector}
    end

    % calculate the magnitude of all positions
    r0abs = sqrt(sum(r0.^2, 1));

    % calculate the the unit vector of all positions
    r0hat = r0 ./ r0abs;

    % calculate field strength and magnitude at position
    H0 = (3 * r0hat .* (m0' * r0hat) - m0) ./ (4 * pi *r0abs.^3);
    H0abs = sqrt(sum(H0.^2, 1));

    % compute the norm factor like described in the equations
    H0norm = Hmag / H0abs;
end
```

Published with MATLAB® R2020b

computeDipoleHField

Computes the magnetic field strength H of a dipole magnet dependent of position and magnetic moment and imprint a field strength magnitude on the resultating field by passing a norm factor which relates to the rest position of the dipole magnet. The resultating field strength has field components in x, y and z direction.

The magnetic dipole moment w must be a column vector or shape

$$\vec{m} = \begin{bmatrix} m_x \\ m_y \\ m_z \end{bmatrix}$$

so that the magnetic moment corresponds to a position vector

$$\vec{r} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

with coordinates for x, y and z in 3D coordinate system which can be taken part of its unit vector and its magnitude.

$$\vec{r} = \hat{r} \cdot |\vec{r}|$$

It computes the field strenght at this position with the current magnetic moment for field compents in the same orientation.

$$\vec{H}(\vec{r}) = \begin{bmatrix} H_x \\ H_y \\ H_z \end{bmatrix}$$

The originally equation of the magnetic dipole is known as

$$\vec{H}(\vec{r}) = \frac{\vec{B}(\vec{r})}{\mu_0}$$

$$\vec{H}(\vec{r}) = \frac{1}{4\pi} \cdot \frac{3\vec{r} \cdot (\vec{m}^T \cdot \vec{r}) - \vec{m} \cdot |\vec{r}|^2}{|\vec{r}|^5}$$

which can be simplified by putting in the unit vector of the position in into the equation.

$$\vec{H}(\vec{r}) = \frac{1}{4\pi |\vec{r}|^3} \cdot \left(3\hat{r} \cdot (\vec{m}^T \cdot \hat{r}) - \vec{m}\right)$$

To imprint a certain field strength related to a rest position of the dipole the resulting field strength is multiplied with a norming factor. The factor must be computed with same magnitude of the magnetic dipole moments which is passed to this computation to get correct field strengths. To get fields without imprinting set the norming factor to 1.

$$\vec{H}(\vec{r}) \cdot H_{0norm}$$

Contents

Syntax

```
H = computeDipoleHField(x, y, z, m, H0norm)
```

Description

H = computeDipoleHField(x, y, z, m, H0norm)compute dipole field strength at passed position (x,y,z) with the magnetic dipole moment m. The resulting field strenght is a vector with components in x, y and z direction. A field strength norming is imprinted on a rest position computation and multiplied on the result by multiplying a norm factor to the field. The normfactor must be relate to the same magnitude of the magnetic dipole moment which is used here and correspond to the magnets rest position in defined distance of the magnets surface.

Examples

```
% compute a single point without norming
H = computeDipoleHField(1, 2, 3, [1; 0; 0], 1)

% compute a 3D grid of positions
x = linspace(-10, 10, 40);
y = linspace(10, -10, 40);
z = linspace(10, -10, 40);
[X, Y, Z] = meshgrid(x, y, z);

% allocate memory for field components in x,y,z
Hx = zeros(40, 40, 40);
Hy = zeros(40, 40, 40);
Hz = zeros(40, 40, 40);

% compute without norming for each z layer and reshape results into layer
% magnetic moments points in -x direction which implies north and south pole
% is in x direction and rotation axes in z
for i=1:40
    H = computeDipoleHField(X(:,:,i),Y(:,:,i),Z(:,:,i),[-1;0;0],1);
    Hx(:,:,i) = reshape(H(1,:),40,40);
    Hy(:,:,i) = reshape(H(2,:),40,40);
    Hz(:,:,i) = reshape(H(3,:),40,40);
end

% calculate magnitude in each point for better view the results
Habs = sqrt(Hx.^2+Hy.^2+Hz.^2);

% define a index to view only every 4th point for not overcrowded plot
idx = 1:4:40;

% downsample and norm
Xds = X(idx,idx,idx);
Yds = Y(idx,idx,idx);
Zds = Z(idx,idx,idx);
Hxds = Hx(idx,idx,idx) ./ Habs(idx,idx,idx);
```

```
Hyds = Hy(idx,idx,idx) ./ Habs(idx,idx,idx);
Hzds = Hz(idx,idx,idx) ./ Habs(idx,idx,idx);

% show results
quiver3(Xds, Yds, Zds, Hxds, Hyds, Hzds);
axis equal;
```

Input Argurments

x coordinates of positions at the field strength is calculated can be scalar, vector or matrix of coordinates. Must be same size as y and z.

y coordinates of positions at the field strength is calculated can be scalar, vector or matrix of coordinates. Must be same size as x and z.

z coordinates of positions at the field strength is calculated can be scalar, vector or matrix of coordinates. Must be same size as x and y.

m magnetic dipole moment as 3 x 1 vector. The magnetic field strength is calculated with the same moment for all passed positions.

H0norm scalar factor to imprint a field strength to the dipole field. Must be computed with the same magnitude of passed magnetic moment vector. Set 1 to disable imprinting.

Output Argurments

H computed magnetic field strength at passed positions with related magnetic moment. If passed position is scalar H has size of 3 X 1 with its components in x, y and z dircetion. H(1) -> x, H(2) -> y and H(3) -> z. If passed positions are not scalar H has size of 3 x numel(x) with position relations in columns. So reshape rows to shapes of positions to keep orientation as origin.

Requirements

- Other m-files required: None
- Subfunctions: mustBeEqualSize
- MAT-files required: None

See Also

- generateDipoleRotationMoments
- generateSensorArraySquareGrid
- computeDipoleH0Norm

Created on June 11. 2019 by Thorben Schüthe. Copyright Thorben Schüthe 2019.

```
function [H] = computeDipoleHField(x, y, z, m, H0norm)
    arguments
        % validate position, can be any size but must be same size of
        x (:,:,:) double {mustBeReal}
        y (:,:,:) double {mustBeReal, mustBeEqualSize(x, y)}
        z (:,:,:) double {mustBeNumeric, mustBeReal, mustBeEqualSize(y, z)}
        % validate magnetic moment as 3 x 1 vector
        m (3,1) double {mustBeReal, mustBeVector}
        % validate norm factor as scalar
        H0norm (1,1) double {mustBeReal}
    end
```

```
    % unify positions to column vector or matrix of column vectors if positions
    % are not passed as column vectors or scalar, resulting size of position R
    % is 3 x length(X), a indication if is column vector is not needed because
    % x(:) is returning all content as column vector. Transpose to match shape.
    r = [x(:), y(:), z(:)]';

    % calculate the magnitude of all positions
    rabs = sqrt(sum(r.^2, 1));

    % calculate the the unit vector of all positions
    rhat = r ./ rabs;

    % calculate H-field of current magnetic moment for all passed positions
    % calculate constants in eqution once in the first bracket term, all vector
    % products in the second term and finially divide by related magnitude ^3
    H = (H0norm / 4 / pi) * (3 * rhat .* (m' * rhat) - m) ./ rabs.^3;
end

% Custom validation function
function mustBeEqualSize(a,b)
    % Test for equal size
    if ~isequal(size(a),size(b))
        eid = 'Size:notEqual';
        msg = 'X Y Z positions must be the same size and orientation.';
        throwAsCaller(MException(eid,msg))
    end
end
```

Published with MATLAB® R2020b

simulateDipoleSquareSensorArray

Simulate a sensor array of square shape with dipole magnet as stimulus. Needs options loaded from config file or generated from config generation script. Characterization data must be loaded befor and served as CharData struct. Loops throught positions saves a data set for every supported position of UseOptions which is called TrainingOptions or TestOptions in config.

Contents

Syntax

```
simulateDipoleSquareSensorArray(GeneralOptions, PathVariables, ...
    SensorArrayOptions, DipoleOptions, UseOptions, CharData)
```

Description

simulateDipoleSquareSensorArray(GeneralOptions, PathVariables, ... SensorArrayOptions, DipoleOptions, UseOptions, CharData) saves simulation datasets to data path specified in PathVariables and UseOptions.

Examples

```
% load config from mat-file
load('config.mat', 'GeneralOptions', 'PathVariables', 'SensorArrayOptions',
    'DipoleOptions', 'TrainingOptions', 'TestOptions');

% load characteriazation dataset
TDK = load(PathVariables.tdkDatasetPath);

% generate training dataset(s)
simulateDipoleSquareSensorArray(GeneralOptions, PathVariables, ...
    SensorArrayOptions, DipoleOptions, TrainingOptions, TDK)

% generate test dataset(s)
simulateDipoleSquareSensorArray(GeneralOptions, PathVariables, ...
    SensorArrayOptions, DipoleOptions, TestOptions, TDK)
```

Input Argurments

GeneralOptions struct of general options generate by config script, includes date format and so on.

PathVariables struct of project path generated by config script, includes data path for save and load data.

SensorArrayOptions struct of sensor array shape and behavior generated by congig script.

DipoleOptions struct of dipole specification, defines magnet and stimulus, generated by config script.

UseOptions struct of implemetnation of use case, defines which kind of datset will be generated. At current state test and training dataset are available options in config. In config generated structs are TestOptions and TrainingOptions.

CharData struct of characteriazation data. Therfore load characterization dataset as shown in examples into a struct.

Output Argurments

None

Requirements

- Other m-files required: computeDipoleH0Norm.m, computeDipoleHField.m, genertateDipoleRotationMoments.m, generateSensorArraySquareGrid.m, rotate3DVector.m
- Subfunctions: reshape, interp2, sum
- MAT-files required: config.mat, TDK_TAS2141_Characterization_2020-10-22_18-12-16-827.mat

See Also

- computeDipoleH0Norm
- computeDipoleHField
- genertateDipoleRotationMoments
- generateSensorArraySquareGrid
- rotate3DVector

Created on June 11. 2019 by Thorben Schüthe. Copyright Thorben Schüthe 2019.

```
function simulateDipoleSquareSensorArray(GeneralOptions, PathVariables, ...
    SensorArrayOptions, DipoleOptions, UseOptions, CharData)
    arguments
        % validate inputs as struct, structs generated in config.mat
        GeneralOptions struct {mustBeA(GeneralOptions, 'struct')}
        PathVariables struct {mustBeA(PathVariables, 'struct')}
        SensorArrayOptions struct {mustBeA(SensorArrayOptions, 'struct')}
        DipoleOptions struct {mustBeA(DipoleOptions, 'struct')}
        UseOptions struct {mustBeA(UseOptions, 'struct')}
        CharData struct {mustBeA(CharData, 'struct')}
    end

    % try to load relavant values in local variable space for better
    % handling and short names second check if struct fields are reachable
    try
        % general options needed to create filenames etc.
        dfStr = GeneralOptions.dateFormat;

        % number of sensors at edge of square array, dimension N x N
        N = SensorArrayOptions.dimension;
        % sensor array edge length, square edge a
        a = SensorArrayOptions.edge;
        % sensor array supply voltage used to generate bridge outputs from
        % characterization data in combination with bridge offset voltage
        % characterization data should be in mv/V so check norm factor
        Vcc = SensorArrayOptions.Vcc;
        Voff = SensorArrayOptions.Voff;
        Vnorm = SensorArrayOptions.Vnorm;
        switch CharData.Info.Units.SensorOutputVoltage
```

```
        case 'mV/V'
            if Vnorm ~= 1e3
                error('Wrong norming mV/V: %e', Vnorm);
            end
        otherwise
            error('Unknown norm voltage: %s', ...
                CharData.Info.Units.SensorOutputVoltage)
    end

    % sphere radius for dipole approximation of spherical magnet
    rsp = DipoleOptions.sphereRadius;
    % H-field magnitude to imprint in certain distance from magnet
    % surface which sphere radius rsp plus distance z0
    H0mag = DipoleOptions.H0mag;
    % distance from magnet surface where to imprint the H0mag
    z0 = DipoleOptions.z0;
    % magnetic dipole moment magnitude which define orgin moment of the
    % magnet in rest position
    M0mag = DipoleOptions.M0mag;

    % dataset type or use case in which later it is use in application
    useCase = UseOptions.useCase;
    % destination path and filename to save gernerated data sets with
    % timestamps in filename, place timestamps with sprintf
    switch useCase
        case 'Training'
            fPath = PathVariables.trainingDataPath;
            fNameFmt = 'Training_%s.mat';
        case 'Test'
            fPath = PathVariables.testDataPath;
            fNameFmt = 'Test_%s.mat';
        otherwise
            error('Unknown use case: %s', UseOptions.useCase);
    end
    % x, y and z positions in which pairing the datasets are generated
    % position vectors are run through in all combinations with tilt
    % and number of angles
    xPos = UseOptions.xPos;
    yPos = UseOptions.yPos;
    zPos = UseOptions.zPos;
    tilt = UseOptions.tilt;
    nAngles = UseOptions.nAngles;
    % constants for generated use case, angle resolution for generated
    % rotation angles, phase index for a phase shift in generation of
    % rotation angles
    angleRes = UseOptions.angleRes;
    phaseIndex = UseOptions.phaseIndex;
    % which characterization reference should be load from CharData
    % sensor output bridge fields (cos/sin)
    refImage = UseOptions.BridgeReference;

    % load values from characterization dataset
    % scales of driven Hx and Hy amplitudes in characteriazation
    % stimulus in kA/m
    if ~strcmp(CharData.Info.Units.MagneticFieldStrength, 'kA/m')
        error('Wrong H-field unit: %s', ...
            CharData.Info.Units.MagneticFieldStrength);
    end
    HxScale = CharData.Data.MagneticField.hx;
    HyScale = CharData.Data.MagneticField.hy;
```

```
        % cosinus and sinus characterization images for corresponding field
        % amplitudes, load and norm to Vcc and Voff, references of
        % simulation, adjust reference to bridge gain for output volgates
        gain = CharData.Info.SensorOutput.BridgeGain;
        VcosRef = CharData.Data.SensorOutput.CosinusBridge.(refImage) ...
            .* (gain * Vcc / Vnorm) + Voff;
        VsinRef = CharData.Data.SensorOutput.SinusBridge.(refImage) ...
            .* (gain * Vcc / Vnorm) + Voff;
    catch ME
        rethrow(ME)
    end

    % now everything is successfully loaded, execute further constants
    % which are needs to be generated once for all following operations
    % meshgrids for refernece images to query bridge reference with interp2
    [HxScaleGrid, HyScaleGrid] = meshgrid(HxScale, HyScale);
    % allocate memory for results of on setup run, speed up compute by 10
    % fix allocations which are not changing by varring parameters like
    % number of angles or positon, for all parameter depended memory size
    % allocalte matlab automatically by function call or need reallocation
    % in for loops
    % H-field components for each rotation step
    Hx = zeros(N, N, nAngles);
    Hy = zeros(N, N, nAngles);
    Hz = zeros(N, N, nAngles);
    % H-field abs for each rotation setp
    Habs = zeros(N, N, nAngles);
    % Bridge output voltages for each sensor in grid, H-fields, sensor
    % grid, voltages all same orientation
    Vcos = zeros(N, N, nAngles);
    Vsin = zeros(N, N, nAngles);


    % compute values which not changing by loop parameters
    % magnetic dipole moments for each rotation step
    % rotation angles to compute
    % index corresponding to full scale rotation with angleRes
    [m, angles, angleRefIndex] = generateDipoleRotationMoments(M0mag, ...
        nAngles, tilt, angleRes, phaseIndex);

    % rotation angle step width on full rotation 360° with subset of angles
    if length(angles) > 1
        angleStep =  angles(2) - angles(1);
    else
        angleStep = 0;
    end

    % compute dipole rest position norm to imprint a certain field
    % strength magnitude with respect of tilt in y axes and magnetization
    % in x direction as in generate Dipole rotation moments
    r0 = rotate3DVector([0; 0; -(z0 + rsp)], 0, tilt, 0);
    m0 = rotate3DVector([-M0mag; 0; 0], 0, tilt, 0);
    H0norm = computeDipoleH0Norm(H0mag, m0, r0);

    % prepare file header Info struct, overwrite certain fields in loop like x,
    % y, z positions
    Info = struct;
    Info.SensorArrayOptions = SensorArrayOptions;
    Info.SensorArrayOptions.SensorCount = N^2;
    Info.DipoleOptions = DipoleOptions;
```

```
    Info.UseOptions = UseOptions;
    Info.CharData = join( ...
        [CharData.Info.SensorManufacturer, CharData.Info.Sensor]);
    Info.Units.SensorOutputVoltage = 'V';
    Info.Units.MagneticFieldStrength = 'kA/m';
    Info.Units.Angles = 'degree';
    Info.Units.Length = 'mm';

    % collect relevant to Data struct for save to file with file
    % header Info struct, overwrite position depended fields in loop before save
    Data = struct;
    Data.HxScale = HxScale;
    Data.HyScale = HyScale;
    Data.VcosRef = VcosRef;
    Data.VsinRef = VsinRef;
    Data.Gain = gain;
    Data.r0 = r0;
    Data.m0 = m0;
    Data.H0norm = H0norm;
    Data.m = m;
    Data.angles = angles;
    Data.angleStep = angleStep;
    Data.angleRefIndex = angleRefIndex;

    % generate dataset for all use case setup pairs in for loop and append
    % generated dataset path to path struct for result
    % outer to inner loop is positions to angles
    % generate z layer wise
    for z = zPos
        for x = xPos
            for y = yPos
                % generate sensor array grid according to current position
                % current position vector of sensor array relative to
                % magnet surface
                p = [x; y; z];
                % write current position in file header
                Info.UseOptions.xPos = x;
                Info.UseOptions.yPos = y;
                Info.UseOptions.zPos = z;
                % sensor array grid coordinates
                [X, Y, Z] = generateSensorArraySquareGrid(N, a, p, rsp);
                % save current sensor gird to Data struct
                Data.X = X;
                Data.Y = Y;
                Data.Z = Z;
                for i = 1:nAngles
                    % calculate H-field of one rotation step for all
                    % positions, the field is normed to zero position
                    H = computeDipoleHField(X, Y, Z, m(:,i), H0norm);
                    % separate parts or field in axes direction/ components
                    Hx(:,:,i) = reshape(H(1,:), N, N);
                    Hy(:,:,i) = reshape(H(2,:), N, N);
                    Hz(:,:,i) = reshape(H(2,:), N, N);
                    Habs(:,:,i) = reshape(sqrt(sum(H.^2, 1)), N, N);
                    % get bridge outputs from references by cross pick
                    % references from grid, the Hx and Hy queries can be
                    % served as matrix as long they have same size and
                    % orientation the nearest neigbhor interpolation
                    % returns of same size and related to orientation, for
                    % outlayers return NaN, do this for every angle
```

```
                    Vcos(:,:,i) = interp2(HxScaleGrid, HyScaleGrid, VcosRef, ...
                        Hx(:,:,i), Hy(:,:,i), 'nearest', NaN);
                    Vsin(:,:,i) = interp2(HxScaleGrid, HyScaleGrid, VsinRef, ...
                        Hx(:,:,i), Hy(:,:,i), 'nearest', NaN);
                end % angles
                % save rotation results to Data struct
                Data.Hx = Hx;
                Data.Hy = Hy;
                Data.Hz = Hz;
                Data.Habs = Habs;
                Data.Vcos = Vcos;
                Data.Vsin = Vsin;
                % save results to file
                fName = sprintf(fNameFmt, datestr(now, dfStr));
                Info.filePath = fullfile(fPath, fName);
                disp(Info.filePath)
                save(Info.filePath, 'Info', 'Data', '-v7.3', '-nocompression');
            end % y
        end % x
    end % z
end
```

Published with MATLAB® R2020b

util

The main property of this module is to contain functions and classes which are used in different scenarios or reused in different modules. So they are providing a more general use case and not a specific one e.g. like a certain algebra function that almost or allways computes the same use case. The util classifficated source code solve module unrelated tasks.

Contents

removeFilesFromDir

Remove files from passed directory and identifier. Return a operation status if files are removed successful or not.

publishFilesFromDir

Publish m-files with Matlab built-in publish mechanism, scan m-files from directory recursively.

plotFunctions

A submodule to contain plot functions for general and specific use on data or datasets.

Created on October 10. 2020 by Tobias Wulf.

Published with MATLAB® R2020b

removeFilesFromDir

Remove files from passed direcotory.

Contents

Syntax

```
removeStatus = removeFilesFromDir(directory)
removeStatus = removeFilesFromDir(directory, filePattern)
```

Description

removeStatus = removeFilesFromDir(directory) removes all files that are located in the passed directory and returns a logical 1 if the operation was successful or 0 if not. The directory argument must be char vector of 1xN and valid path to a existing directory.

removeStatus = removeFilesFromDir(directory, filePattern) removes all files in the located directory which matches the passed file pattern. The filePattern argument must be be char vector of 1xN. It is an optional argument with a default value of '*.*', valid file patterns can be filenames which part replace names by * character before the dot and exisiting file extensions e.g. myfile_*.m or *.txt and so on.

Examples

```
d = fullfile('rootPath', 'subfolder')
rs = removeFileFromDir(d)

d = fullfile('rootPath', 'subfolder')
rs = removeFileFromDir(d, '*.mat')
```

Input Arguments

directory char vector, path directory in which to scan for files with file pattern and to delete found files.

filePattern char vector of file pattern with extension. Default is**.** to delete all files. Possible patterns can be e.g.**.m or *part_name_**.* or ***part_name***.png.

Output Arguments

removeStatus locgical scalar which is true if all files wich matches the file pattern are deleted successfully from passed directory path.

Requirements

- Other m-files required: None
- Subfunctions: None

- MAT-files required: None

See Also

- fullfile
- dir
- delete
- isfile
- isempty
- ismember
- mustBeFolder
- mustBeText

Created on October 10. 2020 by Tobias Wulf. Copyright Tobias Wulf 2020.

```
function [removeStatus] = removeFilesFromDir(directory, filePattern)
    arguments
        % validate directory
        directory (1,:) char {mustBeFolder}
        % validate filePattern
        filePattern (1,:) char {mustBeText} = '*.*'
    end
    % parse pattern for dir
    parsePattern = fullfile(directory, filePattern);
    % parse directory, returns struct
    filesToRemove = dir(parsePattern);
    % delete files, tranpose to loop through struct
    for file = filesToRemove'
        % check before delete
        filePath = fullfile(file.folder, file.name);
        if isfile(filePath)
            delete(filePath);
        end
    end
    % check if dir returns an empty struct now
    check = dir(parsePattern);
    removeStatus = isempty(check(~ismember({check.name}, {'.', '..'})));
end
```

Published with MATLAB® R2020b

publishFilesFromDir

Publish m-files from given directory with passed publishing options.

Contents

Syntax

```
publishFilesFromDir(directory, PublishOptions)
publishFilesFromDir(directory, PublishOptions, recursivly)
publishFilesFromDir(directory, PublishOptions, recursivly, verbose)
```

Description

publishFilesFromDir(directory, PublishOptions) publish m-files wich are located in the passed directory with options from passed publishing options struct which is must be strict formatted after given example from Matlab documentation.

publishFilesFromDir(directory, PublishOptions, recursive) publishing like described before but scan the directory recursively for m-files. Default is false for do not recursively.

publishFilesFromDir(directory, PublishOptions, recursive, verbose) with optional verbose set to true the published html files will be displayed in the prompt. Default is false.

Examples

```
directory = 'src';
PublishOptions = struct;
PublishOptions.outputDir = 'src/html';
PublishOptions.evalCode = false;
publishFilesFromDir(directory, PublishOptions, true)

load('config.mat', 'srcPath', 'PublishOptions')
publishFilesFromDir(srcPath, PublishOptions, true, true)
```

Input Arguments

directory char vector, path to directory where m-files are located to publish.

PublishOptions struct which contains publishing options for the Matlab publish function.

recursive logical scalar which directs the function to scan recursively for m-files in passed directory if true. Default is false.

verbose logical scalar which determines to display the filenames and path to published file if true. Defaults is false.

Output Arguments

None

Requirements

- Other m-files required: None
- Subfunctions: None
- MAT-files required: None

See Also

- dir
- fullfile
- publish

Created on October 31. 2020 by Tobias Wulf. Copyright Tobias Wulf 2020.

```
function publishFilesFromDir(directory, PublishOptions, recursive, verbose)
    arguments
        % validate directory if exist
        directory (1,:) char {mustBeFolder}
        % validate Publish options if it is a struct
        PublishOptions (1,1) struct {mustBeA(PublishOptions, ["struct"])}
        % validate recursive if logical
        recursive (1,1) logical {mustBeNumericOrLogical} = false
        verbose (1,1) logical {mustBeNumericOrLogical} = false
    end

    % file extension is alway m-file
    ext = '*.m';

    % reursive parsing for files with dir function needs a certain regex
    if recursive
        pathPattern = fullfile(directory, '**', ext);
    else
        pathPattern = fullfile(directory, ext);
    end

    % scan directory for files returns a column struct with path fields
    files = dir(pathPattern);

    % transpose files struct and loop through for publish
    for file = files'
        %  if not dir must be file
        if ~file.isdir
            % build path by struct field for recursive tree
            written = publish(fullfile(file.folder, file.name), PublishOptions);
            if verbose
                disp(written);
            end
        end
    end
end
```

Published with MATLAB® R2020b

plotFunctions

Project related reusable plots for datasets and results.

Contents

plotTDKTransferCurves

Plot transfer curves for bridge output voltages of TDK TAS2141.

plotKMZ60TransferCurves

Plot transfer curves for bridge output voltages of NXP KMZ60.

plotKMZ60CharField

Plot NXP KMZ60 characterization field and slice around 0, 5, 10 and 15 kA/m.

plotKMZ60CharDataset

Explore the basic dataset of characterized NXP AMR sensor KMZ60 and plot the dataset content to visualize the base of dipol simulations.

plotSimulationDatasetCircle

Plot circular path of Hx, Hy and Vcos, Vsin at each sensor array position. Normed to max overall array positions and normed to max at each array position.

plotSimulationCosSinStats

Statistical compare plot of Vcos and Vsin output voltages for each sensor array members.

plotSimulationSubset

Plot subset of angles and sensor array position from training or test dataset.

plotSingleSimulationAngle

Plot single rotation step of test or training dataset.

plotSimulationDataset

Plot simulation test or training dataset created by sensor array simulation.

plotTDKCharField

Plot TDK TAS2141 characterization field and slice around 0, 5, 10 and 15 kA/m.

plotTDKCharDataset

Explore the basic dataset of characterized TDK TMR Sensor TAS2141 and plot the dataset content to visualize the base of dipol simulations.

plotDipoleMagnet

Plot dipole magnet and its approximation as spherical magnet from constants set in config file. Plot manget in rest position.

Created on October 24. 2020 by Tobias Wulf. Copyright Tobias Wulf 2020.

Published with MATLAB® R2020b

plotTDKCharDataset

Explore TDK TAS2141 characterization dataset and plot its content.

Contents

Syntax

```
plotTDKCharDataset()
```

Description

plotTDKCharDataset() explores the dataset and plot its content in three docked figure windows. Loads dataset location from config.mat.

Examples

```
plotTDKCharDataset();
```

Input Arguments

None

Output Arguments

None

Requirements

- Other m-files: none
- Subfunctions: none
- MAT-files required: data/TDK_TAS2141_Characterization_2020-10-22_18-12-16-827.mat, data/config.mat

See Also

- plot
- imagesc
- polarplot

Created on October 24. 2020 by Tobias Wulf. Copyright Tobias Wulf 2020.

```
function plotTDKCharDataset()
    try
        % load dataset path and dataset content into function workspace
```

```
        load('config.mat', 'PathVariables');
        load(PathVariables.tdkDatasetPath, 'Data', 'Info');
        close all;
    catch ME
        rethrow(ME)
    end

    % figure save path for different formats
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    fig1Filename = 'tdk_magnetic_stimulus';
    fig1Path = fullfile(PathVariables.saveFiguresPath, fig1Filename);
    fig1SvgPath = fullfile(PathVariables.saveImagesPath, 'svg', fig1Filename);
    fig1EpsPath = fullfile(PathVariables.saveImagesPath, 'eps', fig1Filename);
    fig1PdfPath = fullfile(PathVariables.saveImagesPath, 'pdf', fig1Filename);

    fig2Filename = 'tdk_cosinus_bridge';
    fig2Path = fullfile(PathVariables.saveFiguresPath, fig2Filename);
    fig2SvgPath = fullfile(PathVariables.saveImagesPath, 'svg', fig2Filename);
    fig2EpsPath = fullfile(PathVariables.saveImagesPath, 'eps', fig2Filename);
    fig2PdfPath = fullfile(PathVariables.saveImagesPath, 'pdf', fig2Filename);

    fig3Filename = 'tdk_sinus_bridge';
    fig3Path = fullfile(PathVariables.saveFiguresPath, fig3Filename);
    fig3SvgPath = fullfile(PathVariables.saveImagesPath, 'svg', fig3Filename);
    fig3EpsPath = fullfile(PathVariables.saveImagesPath, 'eps', fig3Filename);
    fig3PdfPath = fullfile(PathVariables.saveImagesPath, 'pdf', fig3Filename);

    % load needed data from dataset in to local variables for better handling
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % check if modulation fits to following reconstructioning
    if ~strcmp("triang", Info.MagneticField.Modulation)
        error("Modulation function is not triang.");
    end
    if ~(strcmp("cos", Info.MagneticField.CarrierHx) && ...
            strcmp("sin", Info.MagneticField.CarrierHy))
        error("Carrier functions are not cos or sin.");
    end

    % modulation frequency
    fm = Info.MagneticField.ModulationFrequency;
    % carrier frequency
    fc = Info.MagneticField.CarrierFrequency;
    % max and min amplitude
    Hmax = Info.MagneticField.MaxAmplitude;
    Hmin = Info.MagneticField.MinAmplitude;
    % step range or window size for output picking
    Hsteps = Info.MagneticField.Steps;
    % resoulution of H steps
    Hres = Info.MagneticField.Resolution;
    % get unit strings from
    kApm = Info.Units.MagneticFieldStrength;
    Hz = Info.Units.Frequency;
    mV = Info.Units.SensorOutputVoltage;

    % get dataset infos and format strings to place in figures
    % subtitle string for all figures
    infoStr = join([Info.SensorManufacturer, Info.Sensor, ...
        Info.SensorTechnology, ...
```

```
    Info.SensorType, "Sensor Characterization Dataset."]);
dateStr = join(["Created on", Info.Created, "by", 'Thorben Sch\"uthe', ...
    "and updated on", Info.Edited, "by", Info.Editor + "."]);

% load characterization data
Vcos = Data.SensorOutput.CosinusBridge;
Vsin = Data.SensorOutput.SinusBridge;
gain = Info.SensorOutput.BridgeGain;

% clear dataset all loaded
clear Data Info;

% reconstruct magnetic stimulus and reduce the view for example plot by 10
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% number of periods reduced by factor 10
reduced = 10;
nPeriods = fc / fm / reduced;
% number of samples for good looking 40 times nPeriods
nSamples = nPeriods * 400;
% half number of samples
nHalf = round(nSamples / 2);
% generate angle base
phi = linspace(0, nPeriods * 2 * pi, nSamples);
% calculate modulated amplitude, triang returns a column vector, transpose
Hmag = Hmax * triang(nSamples)';
% calculate Hx and Hy stimulus
Hx = Hmag .* cos(phi);
Hy = Hmag .* sin(phi);
% index for rising and falling stimulus
idxR = 1:nHalf;
idxF = nHalf:nSamples;
% find absolute min and max values in bridge outputs for uniform colormap
A = cat(3, Vcos.Rise, Vcos.Fall, Vcos.All, Vcos.Diff, Vsin.Rise, ...
    Vsin.Fall, Vsin.All, Vsin.Diff);
Vmax = max(A, [], 'all');
Vmin = min(A, [], 'all');
clear A;

% figure 1 magnetic stimulus
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fig1 = figure('Name', 'Magnetic Stimulus', ...
    'NumberTitle' , 'off', ...
    'WindowStyle', 'normal', ...
    'MenuBar', 'none', ...
    'ToolBar', 'none', ...
    'Units', 'centimeters', ...
    'OuterPosition', [0 0 30 30], ...
    'PaperType', 'a4', ...
    'PaperUnits', 'centimeters', ...
    'PaperOrientation', 'landscape', ...
    'PaperPositionMode', 'auto', ...
    'DoubleBuffer', 'on', ...
    'RendererMode', 'manual', ...
    'Renderer', 'painters');

tdl = tiledlayout(fig1, 2, 2, ...
    'Padding', 'compact', ...
    'TileSpacing' , 'compact');
```

```
title(tdl, 'Reconstructed $H_x$-/ $H_y$-Stimulus in Reduced View', ...
    'FontWeight', 'normal', ...
    'FontSize', 18, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

subtitle(tdl, [infoStr; dateStr], ...
    'FontWeight', 'normal', ...
    'FontSize', 14, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

xlabel(tdl, sprintf('$\\phi$ in rad, %d periods, reduced by factor %d', ...
    nPeriods*reduced, reduced), ...
    'FontWeight', 'normal', ...
    'FontSize', 16, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

ylabel(tdl, sprintf('$H_x$, $H_y$, $|H|$ in %s', kApm), ...
    'FontWeight', 'normal', ...
    'FontSize', 16, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

% Hx stimulus
nexttile;
p = plot(phi, Hmag, phi, -Hmag, phi(idxR), Hx(idxR), phi(idxF), Hx(idxF));
set(p, {'Color'}, {'k', 'k', 'b', 'r'}');
legend([p(1) p(3) p(4)], {'mod', 'rise', 'fall'},...
    'FontWeight', 'normal', ...
    'FontSize', 9, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex', ...
    'Location', 'NorthEast');

xticks((0:0.25*pi:2*pi) * nPeriods);
xticklabels({'0', '8\pi', '16\pi', '24\pi', '32\pi', '40\pi', '48\pi', ...
    '56\pi', '64\pi'});
xlim([0 phi(end)]);
ylim([Hmin Hmax]);

xlabel('$\phi$', ...
    'FontWeight', 'normal', ...
    'FontSize', 12, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

ylabel('$H_x(\phi)$', ...
    'FontWeight', 'normal', ...
    'FontSize', 12, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

title(sprintf(...
    'Modulation $f_m = %1.2f$ %s, Cos-Carrier $f_c = %1.2f$ %s', ...
    fm, Hz, fc, Hz), ...
    'FontWeight', 'normal', ...
    'FontSize', 12, ...
```

```
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % Hy stimulus
    nexttile;
    p = plot(phi, Hmag, phi, -Hmag, phi(idxR), Hy(idxR), phi(idxF), Hy(idxF));
    set(p, {'Color'}, {'k', 'k', 'b', 'r'}');
    legend([p(1) p(3) p(4)], {'mod', 'rise', 'fall'},...
        'FontWeight', 'normal', ...
        'FontSize', 9, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex', ...
        'Location', 'NorthEast');

    xticks((0:0.25*pi:2*pi) * nPeriods);
    xticklabels({'0', '8\pi', '16\pi', '24\pi', '32\pi', '40\pi', '48\pi',...
        '56\pi', '64\pi'});
    xlim([0 phi(end)]);
    ylim([Hmin Hmax]);

    xlabel('$\phi$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$H_y(\phi)$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title(sprintf(...
        'Modulation $f_m = %1.2f$ %s, Sin-Carrier $f_c = %1.2f$ %s', ...
        fm, Hz, fc, Hz), ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % polar for rising modulation
    nexttile;
    polarplot(phi(idxR), Hmag(idxR), 'b');
    title('$|H(\phi)| \cdot e^{-j\phi}$ f. $0 \le \phi \le 32\pi$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % polar for rising modulation
    nexttile;
    polarplot(phi(idxF), Hmag(idxF), 'r');
    title('$|H(\phi)| \cdot e^{-j\phi}$ f. $32\pi \le \phi \le 64\pi$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % figure 2 cosinus bridge outputs
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fig2 = figure('Name', 'Cosinus Bridge', ...
    'NumberTitle' , 'off', ...
    'WindowStyle', 'normal', ...
    'MenuBar', 'none', ...
    'ToolBar', 'none', ...
    'Units', 'centimeters', ...
    'OuterPosition', [0.0 0.0 30.0 30.0], ...
    'PaperType', 'a4', ...
    'PaperUnits', 'centimeters', ...
    'PaperOrientation', 'landscape', ...
    'PaperPositionMode', 'auto', ...
    'DoubleBuffer', 'on', ...
    'RendererMode', 'manual', ...
    'Renderer', 'painters');

tdl = tiledlayout(fig2, 2, 2, ...
    'Padding', 'normal', ...
    'TileSpacing' , 'compact');

title(tdl, ...
    'Measured Cosinus Bridge Outputs of Corresponding $H_x$-/ $H_y$-Amplitudes', ...
    'FontWeight', 'normal', ...
    'FontSize', 18, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

subtitle(tdl, [infoStr; dateStr], ...
    'FontWeight', 'normal', ...
    'FontSize', 14, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

xlabel(tdl, sprintf('$H_x$, $H_y$ in %s, %d Steps in %.4f %s', ...
    kApm, Hsteps, Hres, kApm), ...
    'FontWeight', 'normal', ...
    'FontSize', 16, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

colormap('jet');

% cosinus bridge recorded during rising stimulus
nexttile;
im = imagesc([Hmin Hmax], [Hmin Hmax], Vcos.Rise);
set(gca, 'YDir', 'normal');
set(im, 'AlphaData', ~isnan(Vcos.Rise));
caxis([Vmin, Vmax]);
xlim([Hmin Hmax]);
ylim([Hmin Hmax]);
yticks(xticks);
axis square xy;
grid on;

xlabel('$H_x$', ...
    'FontWeight', 'normal', ...
    'FontSize', 12, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');
```

```
    ylabel('$H_y$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('Rising $H$-Amplitudes', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % cosinus bridge recorded during falling stimulus
    nexttile;
    im = imagesc([Hmin Hmax], [Hmin Hmax], Vcos.Fall);
    set(gca, 'YDir', 'normal');
    set(im, 'AlphaData', ~isnan(Vcos.Fall));
    caxis([Vmin, Vmax]);
    xlim([Hmin Hmax]);
    ylim([Hmin Hmax]);
    yticks(xticks);
    axis square xy;
    grid on;

    xlabel('$H_x$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$H_y$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('Falling $H$-Amplitudes', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % cosinus bridge recorded during superimposed stimulus
    nexttile;
    im = imagesc([Hmin Hmax], [Hmin Hmax], Vcos.All);
    set(gca, 'YDir', 'normal');
    set(im, 'AlphaData', ~(~Vcos.All));
    caxis([Vmin, Vmax]);
    xlim([Hmin Hmax]);
    ylim([Hmin Hmax]);
    yticks(xticks);
    axis square xy;
    grid on;

    xlabel('$H_x$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');
```

```
    ylabel('$H_y$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('Superimposed $H$-Amplitudes', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % cosinus bridge recorded during differentiated stimulus
    nexttile;
    im = imagesc([Hmin Hmax], [Hmin Hmax], Vcos.Diff);
    set(gca, 'YDir', 'normal');
    set(im, 'AlphaData', ~isnan(Vcos.Diff));
    caxis([Vmin, Vmax]);
    xlim([Hmin Hmax]);
    ylim([Hmin Hmax]);
    yticks(xticks);
    axis square xy;
    grid on;

    xlabel('$H_x$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$H_y$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('Differentiated $H$-Amplitudes', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % add colorbar and place it overall plots
    cb = colorbar;
    cb.Layout.Tile = 'east';
    cb.Label.String = sprintf(...
        '$V_{cos}(H_x, H_y)$ in %s, Gain $ = %.1f$', mV, gain);
    cb.Label.Interpreter = 'latex';
    cb.Label.FontSize = 16;

    % figure 3 sinus bridge outputs
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    fig3 = figure('Name', 'Sinus Bridge', ...
        'NumberTitle' , 'off', ...
        'WindowStyle', 'normal', ...
        'MenuBar', 'none', ...
        'ToolBar', 'none', ...
        'Units', 'centimeters', ...
        'OuterPosition', [0.0 0.0 30.0 30.0], ...
```

```
        'PaperType', 'a4', ...
        'PaperUnits', 'centimeters', ...
        'PaperOrientation', 'landscape', ...
        'PaperPositionMode', 'auto', ...
        'DoubleBuffer', 'on', ...
        'RendererMode', 'manual', ...
        'Renderer', 'painters');

    tdl = tiledlayout(fig3, 2, 2, ...
        'Padding', 'normal', ...
        'TileSpacing' , 'compact');

    title(tdl,...
        'Measured Sinus Bridge Outputs of Corresponding $H_x$-/ $H_y$-Amplitudes', ...
        'FontWeight', 'normal', ...
        'FontSize', 18, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    subtitle(tdl, [infoStr; dateStr], ...
        'FontWeight', 'normal', ...
        'FontSize', 14, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    xlabel(tdl, sprintf('$H_x$, $H_y$ in %s, %d Steps in %.4f %s', ...
        kApm, Hsteps, Hres, kApm), ...
        'FontWeight', 'normal', ...
        'FontSize', 16, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    colormap('jet');

    % sinus bridge recorded during rising stimulus
    nexttile;
    im = imagesc([Hmin Hmax], [Hmin Hmax], Vsin.Rise);
    set(gca, 'YDir', 'normal');
    set(im, 'AlphaData', ~isnan(Vsin.Rise));
    caxis([Vmin, Vmax]);
    xlim([Hmin Hmax]);
    ylim([Hmin Hmax]);
    yticks(xticks);
    axis square xy;
    grid on;

    xlabel('$H_x$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$H_y$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('Rising $H$-Amplitudes', ...
        'FontWeight', 'normal', ...
```

```
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % sinus bridge recorded during falling stimulus
    nexttile;
    im = imagesc([Hmin Hmax], [Hmin Hmax], Vsin.Fall);
    set(gca, 'YDir', 'normal');
    set(im, 'AlphaData', ~isnan(Vsin.Fall));
    caxis([Vmin, Vmax]);
    xlim([Hmin Hmax]);
    ylim([Hmin Hmax]);
    yticks(xticks);
    axis square xy;
    grid on;

    xlabel('$H_x$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$H_y$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('Falling $H$-Amplitudes', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % sinus bridge recorded during superimposed stimulus
    nexttile;
    im = imagesc([Hmin Hmax], [Hmin Hmax], Vsin.All);
    set(gca, 'YDir', 'normal');
    set(im, 'AlphaData', ~(~Vsin.All));
    caxis([Vmin, Vmax]);
    xlim([Hmin Hmax]);
    ylim([Hmin Hmax]);
    yticks(xticks);
    axis square xy;
    grid on;

    xlabel('$H_x$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$H_y$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('Superimposed $H$-Amplitudes', ...
        'FontWeight', 'normal', ...
```

```
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % sinus bridge recorded during differentiated stimulus
    nexttile;
    im = imagesc([Hmin Hmax], [Hmin Hmax], Vsin.Diff);
    set(gca, 'YDir', 'normal');
    set(im, 'AlphaData', ~isnan(Vsin.Diff));
    caxis([Vmin, Vmax]);
    xlim([Hmin Hmax]);
    ylim([Hmin Hmax]);
    yticks(xticks);
    axis square xy;
    grid on;

    xlabel('$H_x$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$H_y$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('Differentiated $H$-Amplitudes', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % add colorbar and place it overall plots
    cb = colorbar;
    cb.Layout.Tile = 'east';
    cb.Label.String = sprintf(...
        '$V_{sin}(H_x, H_y)$ in %s, Gain $ = %.1f$', mV, gain);
    cb.Label.Interpreter = 'latex';
    cb.Label.FontSize = 16;

    yesno = input('Save? [y/n]: ', 's');
    if strcmp(yesno, 'y')
        % save results of figure 1
        savefig(fig1, fig1Path);
        print(fig1, fig1SvgPath, '-dsvg');
        print(fig1, fig1EpsPath, '-depsc', '-tiff', '-loose');
        print(fig1, fig1PdfPath, '-dpdf', '-loose', '-fillpage');

        % save results of figure 2
        savefig(fig2, fig2Path);
        print(fig2, fig2SvgPath, '-dsvg');
        print(fig2, fig2EpsPath, '-depsc', '-tiff', '-loose');
        print(fig2, fig2PdfPath, '-dpdf', '-loose', '-fillpage');

        % save results of figure 3
        savefig(fig3, fig3Path);
        print(fig3, fig3SvgPath, '-dsvg');
        print(fig3, fig3EpsPath, '-depsc', '-tiff', '-loose');
```

```
        print(fig3, fig3PdfPath, '-dpdf', '-loose', '-fillpage');
    end
    close(fig1)
    close(fig2)
    close(fig3)
end
```

Published with MATLAB® R2020b

plotTDKCharField

Explore TDK TAS2141 characterization field.

Contents

Syntax

```
plotTDKCharField()
```

Description

plotTDKCharField() explore characterization field of TDK sensor.

Examples

```
plotTDKCharField();
```

Input Arguments

None

Output Arguments

None

Requirements

- Other m-files: none
- Subfunctions: none
- MAT-files required: data/TDK_TAS2141_Characterization_2020-10-22_18-12-16-827.mat, data/config.mat

See Also

- plotTDKCharDataset

Created on October 28. 2020 by Tobias Wulf. Copyright Tobias Wulf 2020.

```
function plotTDKCharField()
    try
        % load dataset path and dataset content into function workspace
        load('config.mat', 'PathVariables');
        load(PathVariables.tdkDatasetPath, 'Data', 'Info');
        close all;
    catch ME
```

```
        rethrow(ME)
    end

    % load needed data from dataset in to local variables for better handling %%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % get from user which field to investigate and limits for plateau
    fields = Info.SensorOutput.CosinusBridge.Determination;
    nFields = length(fields);
    fprintf('Choose 1 of %d fields ...\n', nFields);
    for i = 1:nFields
        fprintf('%s\t:\t(%d)\n', fields{i}, i);
    end

    iField = input('Choice: ');
    field = fields{iField};
    pl = input('Plateu limit in kA/m: ');

    Vcos = Data.SensorOutput.CosinusBridge.(field);
    Vsin = Data.SensorOutput.SinusBridge.(field);
    gain = Info.SensorOutput.BridgeGain;
    HxScale = Data.MagneticField.hx;
    HyScale = Data.MagneticField.hy;
    Hmin = Info.MagneticField.MinAmplitude;
    Hmax = Info.MagneticField.MaxAmplitude;

    % get unit strings from
    kApm = Info.Units.MagneticFieldStrength;
    mV = Info.Units.SensorOutputVoltage;

    % get dataset infos and format strings to place in figures
    % subtitle string for all figures
    infoStr = join([Info.SensorManufacturer, ...
        Info.Sensor, Info.SensorTechnology, ...
        Info.SensorType, "Sensor Characterization Dataset."]);
    dateStr = join(["Created on", Info.Created, "by", 'Thorben Sch\"uthe', ...
        "and updated on", Info.Edited, "by", Info.Editor + "."]);

    % clear dataset all loaded
    clear Data Info;

    % figure save path for different formats %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    fName = sprintf("tdk_char_field_%s", field);
    fPath = fullfile(PathVariables.saveFiguresPath, fName);
    fSvgPath = fullfile(PathVariables.saveImagesPath, 'svg', fName);
    fEpsPath = fullfile(PathVariables.saveImagesPath, 'eps', fName);
    fPdfPath = fullfile(PathVariables.saveImagesPath, 'pdf', fName);

    % define slices and limits to plot %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    Hslice = [128 154 180 205]; % hit ca. 0, 5, 10, 15 kA/m
    Hlims = [-pl pl];
    mVpVlims = [-175 175];

    % create figure for plots %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    fig = figure('Name', 'Char Field', ...
        'NumberTitle' , 'off', ...
        'WindowStyle', 'normal', ...
        'MenuBar', 'none', ...
```

```
        'ToolBar', 'none', ...
        'Units', 'centimeters', ...
        'OuterPosition', [0 0 33 30], ...
        'PaperType', 'a4', ...
        'PaperUnits', 'centimeters', ...
        'PaperOrientation', 'landscape', ...
        'PaperPositionMode', 'auto', ...
        'DoubleBuffer', 'on', ...
        'RendererMode', 'manual', ...
        'Renderer', 'painters');

    tdl = tiledlayout(fig, 2, 2, ...
        'Padding', 'compact', ...
        'TileSpacing' , 'compact');

    title(tdl, sprintf('Characterization Field: %s', field), ...
        'FontWeight', 'normal', ...
        'FontSize', 18, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    subtitle(tdl, [infoStr; dateStr], ...
        'FontWeight', 'normal', ...
        'FontSize', 14, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % set colormap
    colormap('jet');

    % cosinus bridge %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    nexttile(1);
    im = imagesc(HxScale, HyScale, Vcos);
    set(gca, 'YDir', 'normal');
    set(im, 'AlphaData', ~isnan(Vcos));
    yticks(xticks);
    axis square xy;
    grid on;

    % plot lines for slice to investigate
    hold on;
    for i = Hslice
        yline(HyScale(i), 'k:', 'LineWidth', 2);
    end
    hold off;

    xlabel(sprintf('$H_x$ in %s', kApm), ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel(sprintf('$H_y$ in %s', kApm), ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('$V_{cos}(H_x,H_y)$', ...
```

```
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    cb = colorbar;
    cb.Label.String = sprintf(...
        '$V_{cos}(H_x, H_y)$ in %s, Gain $ = %.1f$', mV, gain);
    cb.Label.Interpreter = 'latex';
    cb.Label.FontSize = 12;

    % cosinus bridge sclices %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    nexttile(2);
    % slices
    p = plot(HxScale, Vcos(Hslice,:), 'LineWidth', 1.2);

    % plateau limits
    if pl > 0
        hold on;
        xline(Hlims(1), 'k-.', 'LineWidth', 1.5);
        xline(Hlims(2), 'k-.', 'LineWidth', 1.5);
        hold off;

        text(Hlims(1)-9.5, 100, ...
            sprintf('$%.1f$ %s', Hlims(1), kApm), ...
            'Color', 'k', ...
            'FontSize', 12, ...
            'FontName', 'Times', ...
            'Interpreter', 'latex');

        text(Hlims(2)+0.5, -50, ...
            sprintf('$%.1f$ %s', Hlims(2), kApm), ...
            'Color', 'k', ...
            'FontSize', 12, ...
            'FontName', 'Times', ...
            'Interpreter', 'latex');
    end

    legend(p, {'$H_y \approx 0$ kA/m', ...
               '$H_y \approx 5$ kA/m', ...
               '$H_y \approx 10$ kA/m', ...
               '$H_y \approx 15$ kA/m'},...
            'FontWeight', 'normal', ...
            'FontSize', 9, ...
            'FontName', 'Times', ...
            'Interpreter', 'latex', ...
            'Location', 'SouthEast');

    xlabel(sprintf('$H_x$ in %s', kApm), ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('$V_{cos}(H_x,H_y)$ for $H_y = $ const.', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');
```

```
grid on;
ylim(mVpVlims);
xlim([Hmin Hmax])

% sinus bridge %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
nexttile(3);
im = imagesc(HxScale, HyScale, Vsin);
set(gca, 'YDir', 'normal');
set(im, 'AlphaData', ~isnan(Vsin));
yticks(xticks);
axis square xy;
grid on;

% plot lines for slice to investigate
hold on;
for i = Hslice
    xline(HxScale(i), 'k:', 'LineWidth', 2);
end
hold off;

xlabel(sprintf('$H_x$ in %s', kApm), ...
    'FontWeight', 'normal', ...
    'FontSize', 12, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

ylabel(sprintf('$H_y$ in %s', kApm), ...
    'FontWeight', 'normal', ...
    'FontSize', 12, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

title('$V_{sin}(H_x,H_y)$', ...
    'FontWeight', 'normal', ...
    'FontSize', 12, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

cb = colorbar;
cb.Label.String = sprintf(...
    '$V_{sin}(H_x, H_y)$ in %s, Gain $ = %.1f$', mV, gain);
cb.Label.Interpreter = 'latex';
cb.Label.FontSize = 12;

% sinus bridge sclices %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
nexttile(4);
% slices
p = plot(HxScale, Vsin(:,Hslice), 'LineWidth', 1.2);

% plateau limits
if pl > 0
    hold on;
    xline(Hlims(1), 'k-.', 'LineWidth', 1.5);
    xline(Hlims(2), 'k-.', 'LineWidth', 1.5);
    hold off;

    text(Hlims(1)-9.5, 100, ...
```

```
            sprintf('$%.1f$ %s', Hlims(1), kApm), ...
            'Color', 'k', ...
            'FontSize', 12, ...
            'FontName', 'Times', ...
            'Interpreter', 'latex');

        text(Hlims(2)+0.5, -50, ...
            sprintf('$%.1f$ %s', Hlims(2), kApm), ...
            'Color', 'k', ...
            'FontSize', 12, ...
            'FontName', 'Times', ...
            'Interpreter', 'latex');
    end

    legend(p, {'$H_x \approx 0$ kA/m', ...
               '$H_x \approx 5$ kA/m', ...
               '$H_x \approx 10$ kA/m', ...
               '$H_x \approx 15$ kA/m'},...
            'FontWeight', 'normal', ...
            'FontSize', 9, ...
            'FontName', 'Times', ...
            'Interpreter', 'latex', ...
            'Location', 'SouthEast');

    xlabel(sprintf('$H_y$ in %s', kApm), ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('$V_{sin}(H_x,H_y)$ for $H_x = $ const.', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    grid on;
    ylim(mVpVlims);
    xlim([Hmin Hmax])

    % save results of figure %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    yesno = input('Save? [y/n]: ', 's');
    if strcmp(yesno, 'y')
        savefig(fig, fPath);
        print(fig, fSvgPath, '-dsvg');
        print(fig, fEpsPath, '-depsc', '-tiff', '-loose');
        print(fig, fPdfPath, '-dpdf', '-loose', '-fillpage');
    end
    close(fig)

end
```

Published with MATLAB® R2020b

plotTDKTransferCurves

Plot TDK TAS2141 characterization field transfer curves.

Contents

Syntax

```
plotTDKTransferCurves()
```

Description

plotTDKTransferCurves() plot characterization field of TDK sensor.

Examples

```
plotTDKTransferCurves();
```

Input Arguments

None

Output Arguments

None

Requirements

- Other m-files: none
- Subfunctions: none
- MAT-files required: data/TDK_TAS2141_Characterization_2020-10-22_18-12-16-827.mat, data/config.mat

See Also

- plotTDKCharField

Created on December 05. 2020 by Tobias Wulf. Copyright Tobias Wulf 2020.

```
function plotTDKTransferCurves()
    try
        % load dataset path and dataset content into function workspace
        load('config.mat', 'PathVariables');
        load(PathVariables.tdkDatasetPath, 'Data', 'Info');
        close all;
    catch ME
```

```
        rethrow(ME)
    end

    % load needed data from dataset in to local variables for better handling %%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % get from user which field to investigate and limits for plateau
    fields = Info.SensorOutput.CosinusBridge.Determination;
    nFields = length(fields);
    fprintf('Choose 1 of %d fields ...\n', nFields);
    for i = 1:nFields
        fprintf('%s\t:\t(%d)\n', fields{i}, i);
    end

    iField = input('Choice: ');
    field = fields{iField};
    pl = input('Plateu limit in kA/m: ');

    Vcos = Data.SensorOutput.CosinusBridge.(field);
    Vsin = Data.SensorOutput.SinusBridge.(field);
    gain = Info.SensorOutput.BridgeGain;
    HxScale = Data.MagneticField.hx;
    HyScale = Data.MagneticField.hy;
    Hmin = Info.MagneticField.MinAmplitude;
    Hmax = Info.MagneticField.MaxAmplitude;

    % get unit strings from
    kApm = Info.Units.MagneticFieldStrength;
    mV = Info.Units.SensorOutputVoltage;

    % get dataset infos and format strings to place in figures
    % subtitle string for all figures
    infoStr = join([Info.SensorManufacturer, ...
        Info.Sensor, Info.SensorTechnology, ...
        Info.SensorType, "Sensor Characterization Dataset."]);
    dateStr = join(["Created on", Info.Created, "by", 'Thorben Sch\"uthe', ...
        "and updated on", Info.Edited, "by", Info.Editor + "."]);

    % clear dataset all loaded
    clear Data Info;

    % figure save path for different formats %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    fName = sprintf("tdk_transfer_curves_%s", field);
    fPath = fullfile(PathVariables.saveFiguresPath, fName);
    fSvgPath = fullfile(PathVariables.saveImagesPath, 'svg', fName);
    fEpsPath = fullfile(PathVariables.saveImagesPath, 'eps', fName);
    fPdfPath = fullfile(PathVariables.saveImagesPath, 'pdf', fName);

    % define slices and limits to plot %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    Hslice = 128; % hit ca. 0 kA/m
    Hlims = [-pl pl];
    mVpVlims = [-175 175];

    % create figure for plots %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    fig = figure('Name', 'Transfer Curves', ...
        'NumberTitle' , 'off', ...
        'WindowStyle', 'normal', ...
        'MenuBar', 'none', ...
```

```
        'ToolBar', 'none', ...
        'Units', 'centimeters', ...
        'OuterPosition', [0 0 33 30], ...
        'PaperType', 'a4', ...
        'PaperUnits', 'centimeters', ...
        'PaperOrientation', 'landscape', ...
        'PaperPositionMode', 'auto', ...
        'DoubleBuffer', 'on', ...
        'RendererMode', 'manual', ...
        'Renderer', 'painters');

    tdl = tiledlayout(fig, 2, 2, ...
        'Padding', 'compact', ...
        'TileSpacing' , 'compact');

    title(tdl, sprintf('Transfer Curves: %s', field), ...
        'FontWeight', 'normal', ...
        'FontSize', 18, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    subtitle(tdl, [infoStr; dateStr], ...
        'FontWeight', 'normal', ...
        'FontSize', 14, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % set colormap
    colormap('jet');

    % cosinus bridge %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    nexttile(1);
    im = imagesc(HxScale, HyScale, Vcos);
    set(gca, 'YDir', 'normal');
    set(im, 'AlphaData', ~isnan(Vcos));
    yticks(xticks);
    axis square xy;
    grid on;

    % plot lines for slice to investigate
    hold on;
    yline(HyScale(Hslice), 'k:', 'LineWidth', 3);
    hold off;

    xlabel(sprintf('$H_x$ in %s', kApm), ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel(sprintf('$H_y$ in %s', kApm), ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('$V_{cos}(H_x,H_y)$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
```

```
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % sinus bridge %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    nexttile(2);
    im = imagesc(HxScale, HyScale, Vsin);
    set(gca, 'YDir', 'normal');
    set(im, 'AlphaData', ~isnan(Vsin));
    yticks(xticks);
    axis square xy;
    grid on;

    % plot lines for slice to investigate
    hold on;
    xline(HxScale(Hslice), 'k:', 'LineWidth', 3);
    hold off;

    xlabel(sprintf('$H_x$ in %s', kApm), ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel(sprintf('$H_y$ in %s', kApm), ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('$V_{sin}(H_x,H_y)$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % colorbar for both %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    cb = colorbar;
    cb.Label.String = sprintf(...
        '$V_{out}(H_x, H_y)$ in %s, Gain $ = %.1f$', mV, gain);
    cb.Label.Interpreter = 'latex';
    cb.Label.FontSize = 12;

    % cosinus bridge sclices %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    nexttile([1 2]);
    % slices
    p = plot(HxScale, Vcos(Hslice,:), ...
        HyScale, Vsin(:,Hslice)', 'LineWidth', 1.2);

    % plateau limits
    if pl > 0
        hold on;
        xline(Hlims(1), 'k-.', 'LineWidth', 1.5);
        xline(Hlims(2), 'k-.', 'LineWidth', 1.5);
        hold off;

        text(Hlims(1)+0.5, 4, ...
```

```
            sprintf('$%.1f$ %s', Hlims(1), kApm), ...
            'Color', 'k', ...
            'FontSize', 12, ...
            'FontName', 'Times', ...
            'Interpreter', 'latex');

        text(Hlims(2)+0.5, 4, ...
            sprintf('$%.1f$ %s', Hlims(2), kApm), ...
            'Color', 'k', ...
            'FontSize', 12, ...
            'FontName', 'Times', ...
            'Interpreter', 'latex');
    end

    legend(p, {sprintf('$V_{cos}(H_x,H_y)$ $H_y \\approx 0$ %s', kApm), ...
               sprintf('$V_{sin}(H_x,H_y)$ $H_x \\approx 0$ %s', kApm)},...
            'FontWeight', 'normal', ...
            'FontSize', 9, ...
            'FontName', 'Times', ...
            'Interpreter', 'latex', ...
            'Location', 'SouthEast');

    ylabel(sprintf('$V_{out}$ in %s', mV), ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    xlabel(sprintf('$H$ in %s', kApm), ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('$V_{out}(H_x,H_y)$, Cosinus and Sinus Transfer Curves', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    grid on;
    ylim(mVpVlims);
    xlim([Hmin Hmax])

    % save results of figure %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    yesno = input('Save? [y/n]: ', 's');
    if strcmp(yesno, 'y')
        savefig(fig, fPath);
        print(fig, fSvgPath, '-dsvg');
        print(fig, fEpsPath, '-depsc', '-tiff', '-loose');
        print(fig, fPdfPath, '-dpdf', '-loose', '-fillpage');
    end
    close(fig)

end
```

Published with MATLAB® R2020b

plotKMZ60CharDataset

Explore NXP KMZ60 characterization dataset and plot its content.

Contents

Syntax

```
plotKMZ60CharDataset()
```

Description

plotKMZ60CharDataset() explores the dataset and plot its content in three docked figure windows. Loads dataset location from config.mat.

Examples

```
plotKMZ60CharDataset();
```

Input Arguments

None

Output Arguments

None

Requirements

- Other m-files: none
- Subfunctions: none
- MAT-files required: data/NXP_KMZ60_Characterization_2020-12-03_16-53-16-721.mat, data/config.mat

See Also

- plotTDKCharDataset

Created on December 05. 2020 by Tobias Wulf. Copyright Tobias Wulf 2020.

```
function plotKMZ60CharDataset()
    try
        % load dataset path and dataset content into function workspace
        load('config.mat', 'PathVariables');
        load(PathVariables.kmz60DatasetPath, 'Data', 'Info');
        close all;
```

```
    catch ME
        rethrow(ME)
    end

    % figure save path for different formats
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    fig1Filename = 'kmz60_magnetic_stimulus';
    fig1Path = fullfile(PathVariables.saveFiguresPath, fig1Filename);
    fig1SvgPath = fullfile(PathVariables.saveImagesPath, 'svg', fig1Filename);
    fig1EpsPath = fullfile(PathVariables.saveImagesPath, 'eps', fig1Filename);
    fig1PdfPath = fullfile(PathVariables.saveImagesPath, 'pdf', fig1Filename);

    fig2Filename = 'kmz60_cosinus_bridge';
    fig2Path = fullfile(PathVariables.saveFiguresPath, fig2Filename);
    fig2SvgPath = fullfile(PathVariables.saveImagesPath, 'svg', fig2Filename);
    fig2EpsPath = fullfile(PathVariables.saveImagesPath, 'eps', fig2Filename);
    fig2PdfPath = fullfile(PathVariables.saveImagesPath, 'pdf', fig2Filename);

    fig3Filename = 'kmz60_sinus_bridge';
    fig3Path = fullfile(PathVariables.saveFiguresPath, fig3Filename);
    fig3SvgPath = fullfile(PathVariables.saveImagesPath, 'svg', fig3Filename);
    fig3EpsPath = fullfile(PathVariables.saveImagesPath, 'eps', fig3Filename);
    fig3PdfPath = fullfile(PathVariables.saveImagesPath, 'pdf', fig3Filename);

    % load needed data from dataset in to local variables for better handling
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % check if modulation fits to following reconstructioning
    if ~strcmp("triang", Info.MagneticField.Modulation)
        error("Modulation function is not triang.");
    end
    if ~(strcmp("cos", Info.MagneticField.CarrierHx) && ...
            strcmp("sin", Info.MagneticField.CarrierHy))
        error("Carrier functions are not cos or sin.");
    end

    % modulation frequency
    fm = Info.MagneticField.ModulationFrequency;
    % carrier frequency
    fc = Info.MagneticField.CarrierFrequency;
    % max and min amplitude
    Hmax = Info.MagneticField.MaxAmplitude;
    Hmin = Info.MagneticField.MinAmplitude;
    % step range or window size for output picking
    Hsteps = Info.MagneticField.Steps;
    % resoulution of H steps
    Hres = Info.MagneticField.Resolution;
    % get unit strings from
    kApm = Info.Units.MagneticFieldStrength;
    Hz = Info.Units.Frequency;
    mV = Info.Units.SensorOutputVoltage;

    % get dataset infos and format strings to place in figures
    % subtitle string for all figures
    infoStr = join([Info.SensorManufacturer, ...
        Info.Sensor, Info.SensorTechnology, ...
        Info.SensorType, "Sensor Characterization Dataset."]);
    dateStr = join(["Created on", Info.Created, "by", 'Thorben Sch\"uthe', ...
        "and updated on", Info.Edited, "by", Info.Editor + "."]);
```

```
% load characterization data
Vcos = Data.SensorOutput.CosinusBridge;
Vsin = Data.SensorOutput.SinusBridge;
gain = Info.SensorOutput.BridgeGain;

% clear dataset all loaded
clear Data Info;

% reconstruct magnetic stimulus and reduce the view for example plot by 10
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% number of periods reduced by factor 10
reduced = 10;
nPeriods = fc / fm / reduced;
% number of samples for good looking 40 times nPeriods
nSamples = nPeriods * 400;
% half number of samples
nHalf = round(nSamples / 2);
% generate angle base
phi = linspace(0, nPeriods * 2 * pi, nSamples);
% calculate modulated amplitude, triang returns a column vector, transpose
Hmag = Hmax * triang(nSamples)';
% calculate Hx and Hy stimulus
Hx = Hmag .* cos(phi);
Hy = Hmag .* sin(phi);
% index for rising and falling stimulus
idxR = 1:nHalf;
idxF = nHalf:nSamples;
% find absolute min and max values in bridge outputs for uniform colormap
A = cat(3, Vcos.Rise, Vcos.Fall, Vcos.All, Vcos.Diff, Vsin.Rise, ...
    Vsin.Fall, Vsin.All, Vsin.Diff);
Vmax = max(A, [], 'all');
Vmin = min(A, [], 'all');
clear A;

% figure 1 magnetic stimulus
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fig1 = figure('Name', 'Magnetic Stimulus', ...
    'NumberTitle' , 'off', ...
    'WindowStyle', 'normal', ...
    'MenuBar', 'none', ...
    'ToolBar', 'none', ...
    'Units', 'centimeters', ...
    'OuterPosition', [0 0 30 30], ...
    'PaperType', 'a4', ...
    'PaperUnits', 'centimeters', ...
    'PaperOrientation', 'landscape', ...
    'PaperPositionMode', 'auto', ...
    'DoubleBuffer', 'on', ...
    'RendererMode', 'manual', ...
    'Renderer', 'painters');

tdl = tiledlayout(fig1, 2, 2, ...
    'Padding', 'compact', ...
    'TileSpacing' , 'compact');

title(tdl, 'Reconstructed $H_x$-/ $H_y$-Stimulus in Reduced View', ...
    'FontWeight', 'normal', ...
```

```
        'FontSize', 18, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    subtitle(tdl, [infoStr; dateStr], ...
        'FontWeight', 'normal', ...
        'FontSize', 14, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    xlabel(tdl, sprintf(...
        '$\\phi$ in rad, %d periods, reduced by factor %d', ...
        nPeriods*reduced, reduced), ...
        'FontWeight', 'normal', ...
        'FontSize', 16, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel(tdl, sprintf('$H_x$, $H_y$, $|H|$ in %s', kApm), ...
        'FontWeight', 'normal', ...
        'FontSize', 16, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % Hx stimulus
    nexttile;
    p = plot(phi, Hmag, phi, -Hmag, phi(idxR), Hx(idxR), phi(idxF), Hx(idxF));
    set(p, {'Color'}, {'k', 'k', 'b', 'r'}');
    legend([p(1) p(3) p(4)], {'mod', 'rise', 'fall'},...
        'FontWeight', 'normal', ...
        'FontSize', 9, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex', ...
        'Location', 'NorthEast');

    xticks((0:0.25*pi:2*pi) * nPeriods);
    xticklabels({'0', '8\pi', '16\pi', '24\pi', '32\pi', '40\pi', ...
        '48\pi', '56\pi', '64\pi'});
    xlim([0 phi(end)]);
    ylim([Hmin Hmax]);

    xlabel('$\phi$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$H_x(\phi)$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title(sprintf(...
        'Modulation $f_m = %1.2f$ %s, Cos-Carrier $f_c = %1.2f$ %s',...
        fm, Hz, fc, Hz), ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');
```

```
% Hy stimulus
nexttile;
p = plot(phi, Hmag, phi, -Hmag, phi(idxR), Hy(idxR), phi(idxF), Hy(idxF));
set(p, {'Color'}, {'k', 'k', 'b', 'r'}');
legend([p(1) p(3) p(4)], {'mod', 'rise', 'fall'},...
    'FontWeight', 'normal', ...
    'FontSize', 9, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex', ...
    'Location', 'NorthEast');

xticks((0:0.25*pi:2*pi) * nPeriods);
xticklabels({'0', '8\pi', '16\pi', '24\pi', '32\pi', '40\pi', ...
    '48\pi', '56\pi', '64\pi'});
xlim([0 phi(end)]);
ylim([Hmin Hmax]);

xlabel('$\phi$', ...
    'FontWeight', 'normal', ...
    'FontSize', 12, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

ylabel('$H_y(\phi)$', ...
    'FontWeight', 'normal', ...
    'FontSize', 12, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

title(sprintf(...
    'Modulation $f_m = %1.2f$ %s, Sin-Carrier $f_c = %1.2f$ %s', ...
    fm, Hz, fc, Hz), ...
    'FontWeight', 'normal', ...
    'FontSize', 12, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

% polar for rising modulation
nexttile;
polarplot(phi(idxR), Hmag(idxR), 'b');
title('$|H(\phi)| \cdot e^{-j\phi}$ f. $0 \le \phi \le 32\pi$', ...
    'FontWeight', 'normal', ...
    'FontSize', 12, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

% polar for rising modulation
nexttile;
polarplot(phi(idxF), Hmag(idxF), 'r');
title('$|H(\phi)| \cdot e^{-j\phi}$ f. $32\pi \le \phi \le 64\pi$', ...
    'FontWeight', 'normal', ...
    'FontSize', 12, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

% figure 2 cosinus bridge outputs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fig2 = figure('Name', 'Cosinus Bridge', ...
```

```
        'NumberTitle' , 'off', ...
        'WindowStyle', 'normal', ...
        'MenuBar', 'none', ...
        'ToolBar', 'none', ...
        'Units', 'centimeters', ...
        'OuterPosition', [0.0 0.0 30.0 30.0], ...
        'PaperType', 'a4', ...
        'PaperUnits', 'centimeters', ...
        'PaperOrientation', 'landscape', ...
        'PaperPositionMode', 'auto', ...
        'DoubleBuffer', 'on', ...
        'RendererMode', 'manual', ...
        'Renderer', 'painters');

    tdl = tiledlayout(fig2, 2, 2, ...
        'Padding', 'normal', ...
        'TileSpacing' , 'compact');

    title(tdl,...
        'Measured Cosinus Bridge Outputs of Corresponding $H_x$-/ $H_y$-Amplitudes', ...
        'FontWeight', 'normal', ...
        'FontSize', 18, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    subtitle(tdl, [infoStr; dateStr], ...
        'FontWeight', 'normal', ...
        'FontSize', 14, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    xlabel(tdl, sprintf(...
        '$H_x$, $H_y$ in %s, %d Steps in %.4f %s',...
        kApm, Hsteps, Hres, kApm), ...
        'FontWeight', 'normal', ...
        'FontSize', 16, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    colormap('jet');

    % cosinus bridge recorded during rising stimulus
    nexttile;
    im = imagesc([Hmin Hmax], [Hmin Hmax], Vcos.Rise);
    set(gca, 'YDir', 'normal');
    set(im, 'AlphaData', ~isnan(Vcos.Rise));
    caxis([Vmin, Vmax]);
    xlim([Hmin Hmax]);
    ylim([Hmin Hmax]);
    yticks(xticks);
    axis square xy;
    grid on;

    xlabel('$H_x$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$H_y$', ...
```

```
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('Rising $H$-Amplitudes', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % cosinus bridge recorded during falling stimulus
    nexttile;
    im = imagesc([Hmin Hmax], [Hmin Hmax], Vcos.Fall);
    set(gca, 'YDir', 'normal');
    set(im, 'AlphaData', ~isnan(Vcos.Fall));
    caxis([Vmin, Vmax]);
    xlim([Hmin Hmax]);
    ylim([Hmin Hmax]);
    yticks(xticks);
    axis square xy;
    grid on;

    xlabel('$H_x$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$H_y$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('Falling $H$-Amplitudes', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % cosinus bridge recorded during superimposed stimulus
    nexttile;
    im = imagesc([Hmin Hmax], [Hmin Hmax], Vcos.All);
    set(gca, 'YDir', 'normal');
    set(im, 'AlphaData', ~(~Vcos.All));
    caxis([Vmin, Vmax]);
    xlim([Hmin Hmax]);
    ylim([Hmin Hmax]);
    yticks(xticks);
    axis square xy;
    grid on;

    xlabel('$H_x$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$H_y$', ...
```

```
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('Superimposed $H$-Amplitudes', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % cosinus bridge recorded during differentiated stimulus
    nexttile;
    im = imagesc([Hmin Hmax], [Hmin Hmax], Vcos.Diff);
    set(gca, 'YDir', 'normal');
    set(im, 'AlphaData', ~isnan(Vcos.Diff));
    caxis([Vmin, Vmax]);
    xlim([Hmin Hmax]);
    ylim([Hmin Hmax]);
    yticks(xticks);
    axis square xy;
    grid on;

    xlabel('$H_x$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$H_y$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('Differentiated $H$-Amplitudes', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % add colorbar and place it overall plots
    cb = colorbar;
    cb.Layout.Tile = 'east';
    cb.Label.String = sprintf(...
        '$V_{cos}(H_x, H_y)$ in %s, Gain $ = %.1f$', mV, gain);
    cb.Label.Interpreter = 'latex';
    cb.Label.FontSize = 16;

    % figure 3 sinus bridge outputs
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    fig3 = figure('Name', 'Sinus Bridge', ...
        'NumberTitle' , 'off', ...
        'WindowStyle', 'normal', ...
        'MenuBar', 'none', ...
        'ToolBar', 'none', ...
        'Units', 'centimeters', ...
        'OuterPosition', [0.0 0.0 30.0 30.0], ...
        'PaperType', 'a4', ...
```

```
        'PaperUnits', 'centimeters', ...
        'PaperOrientation', 'landscape', ...
        'PaperPositionMode', 'auto', ...
        'DoubleBuffer', 'on', ...
        'RendererMode', 'manual', ...
        'Renderer', 'painters');

    tdl = tiledlayout(fig3, 2, 2, ...
        'Padding', 'normal', ...
        'TileSpacing' , 'compact');

    title(tdl,...
        'Measured Sinus Bridge Outputs of Corresponding $H_x$-/ $H_y$-Amplitudes', ...
        'FontWeight', 'normal', ...
        'FontSize', 18, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    subtitle(tdl, [infoStr; dateStr], ...
        'FontWeight', 'normal', ...
        'FontSize', 14, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    xlabel(tdl, sprintf(...
        '$H_x$, $H_y$ in %s, %d Steps in %.4f %s', ...
        kApm, Hsteps, Hres, kApm), ...
        'FontWeight', 'normal', ...
        'FontSize', 16, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    colormap('jet');

    % sinus bridge recorded during rising stimulus
    nexttile;
    im = imagesc([Hmin Hmax], [Hmin Hmax], Vsin.Rise);
    set(gca, 'YDir', 'normal');
    set(im, 'AlphaData', ~isnan(Vsin.Rise));
    caxis([Vmin, Vmax]);
    xlim([Hmin Hmax]);
    ylim([Hmin Hmax]);
    yticks(xticks);
    axis square xy;
    grid on;

    xlabel('$H_x$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$H_y$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('Rising $H$-Amplitudes', ...
        'FontWeight', 'normal', ...
```

```
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % sinus bridge recorded during falling stimulus
    nexttile;
    im = imagesc([Hmin Hmax], [Hmin Hmax], Vsin.Fall);
    set(gca, 'YDir', 'normal');
    set(im, 'AlphaData', ~isnan(Vsin.Fall));
    caxis([Vmin, Vmax]);
    xlim([Hmin Hmax]);
    ylim([Hmin Hmax]);
    yticks(xticks);
    axis square xy;
    grid on;

    xlabel('$H_x$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$H_y$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('Falling $H$-Amplitudes', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % sinus bridge recorded during superimposed stimulus
    nexttile;
    im = imagesc([Hmin Hmax], [Hmin Hmax], Vsin.All);
    set(gca, 'YDir', 'normal');
    set(im, 'AlphaData', ~(~Vsin.All));
    caxis([Vmin, Vmax]);
    xlim([Hmin Hmax]);
    ylim([Hmin Hmax]);
    yticks(xticks);
    axis square xy;
    grid on;

    xlabel('$H_x$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$H_y$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('Superimposed $H$-Amplitudes', ...
        'FontWeight', 'normal', ...
```

```
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % sinus bridge recorded during differentiated stimulus
    nexttile;
    im = imagesc([Hmin Hmax], [Hmin Hmax], Vsin.Diff);
    set(gca, 'YDir', 'normal');
    set(im, 'AlphaData', ~isnan(Vsin.Diff));
    caxis([Vmin, Vmax]);
    xlim([Hmin Hmax]);
    ylim([Hmin Hmax]);
    yticks(xticks);
    axis square xy;
    grid on;

    xlabel('$H_x$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$H_y$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('Differentiated $H$-Amplitudes', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % add colorbar and place it overall plots
    cb = colorbar;
    cb.Layout.Tile = 'east';
    cb.Label.String = sprintf(...
        '$V_{sin}(H_x, H_y)$ in %s, Gain $ = %.1f$', mV, gain);
    cb.Label.Interpreter = 'latex';
    cb.Label.FontSize = 16;

    yesno = input('Save? [y/n]: ', 's');
    if strcmp(yesno, 'y')
        % save results of figure 1
        savefig(fig1, fig1Path);
        print(fig1, fig1SvgPath, '-dsvg');
        print(fig1, fig1EpsPath, '-depsc', '-tiff', '-loose');
        print(fig1, fig1PdfPath, '-dpdf', '-loose', '-fillpage');

        % save results of figure 2
        savefig(fig2, fig2Path);
        print(fig2, fig2SvgPath, '-dsvg');
        print(fig2, fig2EpsPath, '-depsc', '-tiff', '-loose');
        print(fig2, fig2PdfPath, '-dpdf', '-loose', '-fillpage');

        % save results of figure 3
        savefig(fig3, fig3Path);
        print(fig3, fig3SvgPath, '-dsvg');
        print(fig3, fig3EpsPath, '-depsc', '-tiff', '-loose');
```

```
        print(fig3, fig3PdfPath, '-dpdf', '-loose', '-fillpage');
    end
    close(fig1)
    close(fig2)
    close(fig3)
end
```

Published with MATLAB® R2020b

plotKMZ60CharField

Explore NXP KMZ60 characterization field.

Contents

Syntax

```
plotKMZ60CharField()
```

Description

plotKMZ60CharField() explore characterization field of KMZ60 sensor.

Examples

```
plotKMZ60CharField();
```

Input Arguments

None

Output Arguments

None

Requirements

- Other m-files: none
- Subfunctions: none
- MAT-files required: data/NXP_KMZ60_Characterization_2020-12-03_16-53-16-721.mat, data/config.mat

See Also

- plotTDKCharField

Created on December 05. 2020 by Tobias Wulf. Copyright Tobias Wulf 2020.

```
function plotKMZ60CharField()
    try
        % load dataset path and dataset content into function workspace
        load('config.mat', 'PathVariables');
        load(PathVariables.kmz60DatasetPath, 'Data', 'Info');
        close all;
    catch ME
```

```
        rethrow(ME)
    end

    % load needed data from dataset in to local variables for better handling %%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % get from user which field to investigate and limits for plateau
    fields = Info.SensorOutput.CosinusBridge.Determination;
    nFields = length(fields);
    fprintf('Choose 1 of %d fields ...\n', nFields);
    for i = 1:nFields
        fprintf('%s\t:\t(%d)\n', fields{i}, i);
    end

    iField = input('Choice: ');
    field = fields{iField};
    pl = input('Plateu limit in kA/m: ');

    Vcos = Data.SensorOutput.CosinusBridge.(field);
    Vsin = Data.SensorOutput.SinusBridge.(field);
    gain = Info.SensorOutput.BridgeGain;
    HxScale = Data.MagneticField.hx;
    HyScale = Data.MagneticField.hy;
    Hmin = Info.MagneticField.MinAmplitude;
    Hmax = Info.MagneticField.MaxAmplitude;

    % get unit strings from
    kApm = Info.Units.MagneticFieldStrength;
    mV = Info.Units.SensorOutputVoltage;

    % get dataset infos and format strings to place in figures
    % subtitle string for all figures
    infoStr = join([Info.SensorManufacturer, ...
        Info.Sensor, Info.SensorTechnology, ...
        Info.SensorType, "Sensor Characterization Dataset."]);
    dateStr = join(["Created on", Info.Created, "by", 'Thorben Sch\"uthe', ...
        "and updated on", Info.Edited, "by", Info.Editor + "."]);

    % clear dataset all loaded
    clear Data Info;

    % figure save path for different formats %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    fName = sprintf("kmz60_char_field_%s", field);
    fPath = fullfile(PathVariables.saveFiguresPath, fName);
    fSvgPath = fullfile(PathVariables.saveImagesPath, 'svg', fName);
    fEpsPath = fullfile(PathVariables.saveImagesPath, 'eps', fName);
    fPdfPath = fullfile(PathVariables.saveImagesPath, 'pdf', fName);

    % define slices and limits to plot %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    Hslice = [128 154 180 205]; % hit ca. 0, 5, 10, 15 kA/m
    Hlims = [-pl pl];
    mVpVlims = [-8 8];

    % create figure for plots %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    fig = figure('Name', 'Char Field', ...
        'NumberTitle' , 'off', ...
        'WindowStyle', 'normal', ...
        'MenuBar', 'none', ...
```

```
        'ToolBar', 'none', ...
        'Units', 'centimeters', ...
        'OuterPosition', [0 0 33 30], ...
        'PaperType', 'a4', ...
        'PaperUnits', 'centimeters', ...
        'PaperOrientation', 'landscape', ...
        'PaperPositionMode', 'auto', ...
        'DoubleBuffer', 'on', ...
        'RendererMode', 'manual', ...
        'Renderer', 'painters');

    tdl = tiledlayout(fig, 2, 2, ...
        'Padding', 'compact', ...
        'TileSpacing' , 'compact');

    title(tdl, sprintf('Characterization Field: %s', field), ...
        'FontWeight', 'normal', ...
        'FontSize', 18, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    subtitle(tdl, [infoStr; dateStr], ...
        'FontWeight', 'normal', ...
        'FontSize', 14, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % set colormap
    colormap('jet');

    % cosinus bridge %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    nexttile(1);
    im = imagesc(HxScale, HyScale, Vcos);
    set(gca, 'YDir', 'normal');
    set(im, 'AlphaData', ~isnan(Vcos));
    yticks(xticks);
    axis square xy;
    grid on;

    % plot lines for slice to investigate
    hold on;
    for i = Hslice
        yline(HyScale(i), 'k:', 'LineWidth', 2);
    end
    hold off;

    xlabel(sprintf('$H_x$ in %s', kApm), ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel(sprintf('$H_y$ in %s', kApm), ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('$V_{cos}(H_x,H_y)$', ...
```

```
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    cb = colorbar;
    cb.Label.String = sprintf(...
        '$V_{cos}(H_x, H_y)$ in %s, Gain $ = %.1f$', mV, gain);
    cb.Label.Interpreter = 'latex';
    cb.Label.FontSize = 12;

    % cosinus bridge sclices %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    nexttile(2);
    % slices
    p = plot(HxScale, Vcos(Hslice,:), 'LineWidth', 1.2);

    % plateau limits
    if pl > 0
        hold on;
        xline(Hlims(1), 'k-.', 'LineWidth', 1.5);
        xline(Hlims(2), 'k-.', 'LineWidth', 1.5);
        hold off;

        text(Hlims(1)-9.5, 4, ...
            sprintf('$%.1f$ %s', Hlims(1), kApm), ...
            'Color', 'k', ...
            'FontSize', 12, ...
            'FontName', 'Times', ...
            'Interpreter', 'latex');

        text(Hlims(2)+0.5, 4, ...
            sprintf('$%.1f$ %s', Hlims(2), kApm), ...
            'Color', 'k', ...
            'FontSize', 12, ...
            'FontName', 'Times', ...
            'Interpreter', 'latex');
    end

    legend(p, {'$H_y \approx 0$ kA/m', ...
               '$H_y \approx 5$ kA/m', ...
               '$H_y \approx 10$ kA/m', ...
               '$H_y \approx 15$ kA/m'},...
            'FontWeight', 'normal', ...
            'FontSize', 9, ...
            'FontName', 'Times', ...
            'Interpreter', 'latex', ...
            'Location', 'SouthEast');

    xlabel(sprintf('$H_x$ in %s', kApm), ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('$V_{cos}(H_x,H_y)$ for $H_y = $ const.', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');
```

```
grid on;
ylim(mVpVlims);
xlim([Hmin Hmax])

% sinus bridge %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
nexttile(3);
im = imagesc(HxScale, HyScale, Vsin);
set(gca, 'YDir', 'normal');
set(im, 'AlphaData', ~isnan(Vsin));
yticks(xticks);
axis square xy;
grid on;

% plot lines for slice to investigate
hold on;
for i = Hslice
    xline(HxScale(i), 'k:', 'LineWidth', 2);
end
hold off;

xlabel(sprintf('$H_x$ in %s', kApm), ...
    'FontWeight', 'normal', ...
    'FontSize', 12, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

ylabel(sprintf('$H_y$ in %s', kApm), ...
    'FontWeight', 'normal', ...
    'FontSize', 12, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

title('$V_{sin}(H_x,H_y)$', ...
    'FontWeight', 'normal', ...
    'FontSize', 12, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

cb = colorbar;
cb.Label.String = sprintf(...
    '$V_{sin}(H_x, H_y)$ in %s, Gain $ = %.1f$', mV, gain);
cb.Label.Interpreter = 'latex';
cb.Label.FontSize = 12;

% sinus bridge sclices %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
nexttile(4);
% slices
p = plot(HxScale, Vsin(:,Hslice), 'LineWidth', 1.2);

% plateau limits
if pl > 0
    hold on;
    xline(Hlims(1), 'k-.', 'LineWidth', 1.5);
    xline(Hlims(2), 'k-.', 'LineWidth', 1.5);
    hold off;

    text(Hlims(1)-9.5, 4, ...
```

```
            sprintf('$%.1f$ %s', Hlims(1), kApm), ...
            'Color', 'k', ...
            'FontSize', 12, ...
            'FontName', 'Times', ...
            'Interpreter', 'latex');

        text(Hlims(2)+0.5, 4, ...
            sprintf('$%.1f$ %s', Hlims(2), kApm), ...
            'Color', 'k', ...
            'FontSize', 12, ...
            'FontName', 'Times', ...
            'Interpreter', 'latex');
    end

    legend(p, {'$H_x \approx 0$ kA/m', ...
               '$H_x \approx 5$ kA/m', ...
               '$H_x \approx 10$ kA/m', ...
               '$H_x \approx 15$ kA/m'},...
            'FontWeight', 'normal', ...
            'FontSize', 9, ...
            'FontName', 'Times', ...
            'Interpreter', 'latex', ...
            'Location', 'SouthEast');

    xlabel(sprintf('$H_y$ in %s', kApm), ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('$V_{sin}(H_x,H_y)$ for $H_x = $ const.', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    grid on;
    ylim(mVpVlims);
    xlim([Hmin Hmax])

    % save results of figure %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    yesno = input('Save? [y/n]: ', 's');
    if strcmp(yesno, 'y')
        savefig(fig, fPath);
        print(fig, fSvgPath, '-dsvg');
        print(fig, fEpsPath, '-depsc', '-tiff', '-loose');
        print(fig, fPdfPath, '-dpdf', '-loose', '-fillpage');
    end
    close(fig)

end
```

Published with MATLAB® R2020b

plotKMZ60TransferCurves

Plot NXP KMZ60 characterization field transfer curves.

Contents

Syntax

```
plotKMZ60TransferCurves()
```

Description

plotKMZ60TransferCurves() plot characterization field of KMZ 60 sensor.

Examples

```
plotKMZ60TransferCurves();
```

Input Arguments

None

Output Arguments

None

Requirements

- Other m-files: none
- Subfunctions: none
- MAT-files required: data/NXP_KMZ60_Characterization_2020-12-03_16-53-16-721.mat, data/config.mat

See Also

- plotKMZ60CharField

Created on December 05. 2020 by Tobias Wulf. Copyright Tobias Wulf 2020.

```
function plotKMZ60TransferCurves()
    try
        % load dataset path and dataset content into function workspace
        load('config.mat', 'PathVariables');
        load(PathVariables.kmz60DatasetPath, 'Data', 'Info');
        close all;
    catch ME
```

```
        rethrow(ME)
    end

    % load needed data from dataset in to local variables for better handling %%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % get from user which field to investigate and limits for plateau
    fields = Info.SensorOutput.CosinusBridge.Determination;
    nFields = length(fields);
    fprintf('Choose 1 of %d fields ...\n', nFields);
    for i = 1:nFields
        fprintf('%s\t:\t(%d)\n', fields{i}, i);
    end

    iField = input('Choice: ');
    field = fields{iField};
    pl = input('Plateu limit in kA/m: ');

    Vcos = Data.SensorOutput.CosinusBridge.(field);
    Vsin = Data.SensorOutput.SinusBridge.(field);
    gain = Info.SensorOutput.BridgeGain;
    HxScale = Data.MagneticField.hx;
    HyScale = Data.MagneticField.hy;
    Hmin = Info.MagneticField.MinAmplitude;
    Hmax = Info.MagneticField.MaxAmplitude;

    % get unit strings from
    kApm = Info.Units.MagneticFieldStrength;
    mV = Info.Units.SensorOutputVoltage;

    % get dataset infos and format strings to place in figures
    % subtitle string for all figures
    infoStr = join([Info.SensorManufacturer, ...
        Info.Sensor, Info.SensorTechnology, ...
        Info.SensorType, "Sensor Characterization Dataset."]);
    dateStr = join(["Created on", Info.Created, "by", 'Thorben Sch\"uthe', ...
        "and updated on", Info.Edited, "by", Info.Editor + "."]);

    % clear dataset all loaded
    clear Data Info;

    % figure save path for different formats %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    fName = sprintf("kmz60_transfer_curves_%s", field);
    fPath = fullfile(PathVariables.saveFiguresPath, fName);
    fSvgPath = fullfile(PathVariables.saveImagesPath, 'svg', fName);
    fEpsPath = fullfile(PathVariables.saveImagesPath, 'eps', fName);
    fPdfPath = fullfile(PathVariables.saveImagesPath, 'pdf', fName);

    % define slices and limits to plot %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    Hslice = 128; % hit ca. 0 kA/m
    Hlims = [-pl pl];
    mVpVlims = [-8 8];

    % create figure for plots %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    fig = figure('Name', 'Transfer Curves', ...
        'NumberTitle' , 'off', ...
        'WindowStyle', 'normal', ...
        'MenuBar', 'none', ...
```

```
        'ToolBar', 'none', ...
        'Units', 'centimeters', ...
        'OuterPosition', [0 0 33 30], ...
        'PaperType', 'a4', ...
        'PaperUnits', 'centimeters', ...
        'PaperOrientation', 'landscape', ...
        'PaperPositionMode', 'auto', ...
        'DoubleBuffer', 'on', ...
        'RendererMode', 'manual', ...
        'Renderer', 'painters');

    tdl = tiledlayout(fig, 2, 2, ...
        'Padding', 'compact', ...
        'TileSpacing' , 'compact');

    title(tdl, sprintf('Transfer Curves: %s', field), ...
        'FontWeight', 'normal', ...
        'FontSize', 18, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    subtitle(tdl, [infoStr; dateStr], ...
        'FontWeight', 'normal', ...
        'FontSize', 14, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % set colormap
    colormap('jet');

    % cosinus bridge %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    nexttile(1);
    im = imagesc(HxScale, HyScale, Vcos);
    set(gca, 'YDir', 'normal');
    set(im, 'AlphaData', ~isnan(Vcos));
    yticks(xticks);
    axis square xy;
    grid on;

    % plot lines for slice to investigate
    hold on;
    yline(HyScale(Hslice), 'k:', 'LineWidth', 3);
    hold off;

    xlabel(sprintf('$H_x$ in %s', kApm), ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel(sprintf('$H_y$ in %s', kApm), ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('$V_{cos}(H_x,H_y)$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
```

```
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % sinus bridge %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    nexttile(2);
    im = imagesc(HxScale, HyScale, Vsin);
    set(gca, 'YDir', 'normal');
    set(im, 'AlphaData', ~isnan(Vsin));
    yticks(xticks);
    axis square xy;
    grid on;

    % plot lines for slice to investigate
    hold on;
    xline(HxScale(Hslice), 'k:', 'LineWidth', 3);
    hold off;

    xlabel(sprintf('$H_x$ in %s', kApm), ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel(sprintf('$H_y$ in %s', kApm), ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('$V_{sin}(H_x,H_y)$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % colorbar for both %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    cb = colorbar;
    cb.Label.String = sprintf(...
        '$V_{out}(H_x, H_y)$ in %s, Gain $ = %.1f$', mV, gain);
    cb.Label.Interpreter = 'latex';
    cb.Label.FontSize = 12;

    % cosinus bridge sclices %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    nexttile([1 2]);
    % slices
    p = plot(HxScale, Vcos(Hslice,:),...
        HyScale, Vsin(:, Hslice)', 'LineWidth', 1.2);

    % plateau limits
    if pl > 0
        hold on;
        xline(Hlims(1), 'k-.', 'LineWidth', 1.5);
        xline(Hlims(2), 'k-.', 'LineWidth', 1.5);
        hold off;

        text(Hlims(1)+0.5, 4, ...
```

```
            sprintf('$%.1f$ %s', Hlims(1), kApm), ...
            'Color', 'k', ...
            'FontSize', 12, ...
            'FontName', 'Times', ...
            'Interpreter', 'latex');

        text(Hlims(2)+0.5, 4, ...
            sprintf('$%.1f$ %s', Hlims(2), kApm), ...
            'Color', 'k', ...
            'FontSize', 12, ...
            'FontName', 'Times', ...
            'Interpreter', 'latex');
    end

    legend(p, {sprintf('$V_{cos}(H_x,H_y)$ $H_y \\approx 0$ %s', kApm), ...
               sprintf('$V_{sin}(H_x,H_y)$ $H_x \\approx 0$ %s', kApm)},...
            'FontWeight', 'normal', ...
            'FontSize', 9, ...
            'FontName', 'Times', ...
            'Interpreter', 'latex', ...
            'Location', 'SouthEast');

    ylabel(sprintf('$V_{out}$ in %s', mV), ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    xlabel(sprintf('$H$ in %s', kApm), ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('$V_{out}(H_x,H_y)$, Cosinus and Sinus Transfer Curves', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    grid on;
    ylim(mVpVlims);
    xlim([Hmin Hmax])

    % save results of figure %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    yesno = input('Save? [y/n]: ', 's');
    if strcmp(yesno, 'y')
        savefig(fig, fPath);
        print(fig, fSvgPath, '-dsvg');
        print(fig, fEpsPath, '-depsc', '-tiff', '-loose');
        print(fig, fPdfPath, '-dpdf', '-loose', '-fillpage');
    end
    close(fig)

end
```

Published with MATLAB® R2020b

plotDipoleMagnet

Plot dipole magnet which approximate a spherical magnet in its far field.

Contents

Syntax

```
plotDipoleMagnet()
```

Description

plotDipoleMagnet() load dipole constants from config.mat and construct magnet in its rest position in x and z layer for y = 0.

Examples

```
plotDipoleMagnet();
```

Input Arguments

None

Output Arguments

None

Requirements

- Other m-files: generateDipoleRotationMoments.m, computeDipoleH0Norm.m, computeDipoleHField
- Subfunctions: none
- MAT-files required: data/config.mat

See Also

- quiver
- imagesc
- streamslice

Created on November 20. 2020 by Tobias Wulf. Copyright Tobias Wulf 2020.

```
function plotDipoleMagnet()
    try
        % load dataset path and dataset content into function workspace
        load('config.mat', 'PathVariables', 'DipoleOptions');
```

```
    close all;
catch ME
    rethrow(ME)
end

% figure save path for different formats
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figFilename = 'dipole_magnet';
figPath = fullfile(PathVariables.saveFiguresPath, figFilename);
figSvgPath = fullfile(PathVariables.saveImagesPath, 'svg', figFilename);
figEpsPath = fullfile(PathVariables.saveImagesPath, 'eps', figFilename);
figPdfPath = fullfile(PathVariables.saveImagesPath, 'pdf', figFilename);

% load needed data from dataset in to local variables for better handling
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Radius in mm of magnetic sphere in which the magnetic dipole is centered.
% So it can be seen as z-offset to the sensor array.
rsp = DipoleOptions.sphereRadius;

% H-field magnitude to multiply of generated and relative normed dipole
Hmag = DipoleOptions.H0mag;

% Distance in zero position of the spherical magnet in which is imprinted
z0 = DipoleOptions.z0;

% Magnetic moment magnitude attach rotation to the dipole field
m0 = DipoleOptions.M0mag;

% clear dataset all loaded
clear DipoleOptions;

% set construction dipole magnet, all length in mm and areas mm^2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% number of samples for good looking
nSamples = 501;
% slice in view for quiver, every 25th point
slice = 25:25:nSamples-25;
% grid edge of meshgrid, square grid
xz = 15;
% y layer in coordinate system
y = 0;
% orientat of magnet along z axes
pz = pi/2:0.01:3*pi/2;
% distances magnet surface to display in plot
zd = -rsp:-z0:-xz;
xd = zeros(1, length(zd));
% scale grid to simulate
x = linspace(-xz, xz, nSamples);
z = linspace(xz, -xz, nSamples);
[X, Z, Y] = meshgrid(x, z, y);

% compute dipole and fetch to far field to approximate a sperical magnet
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% generate dipole moment for 0°
m = generateDipoleRotationMoments(m0, 1);
% compute H-field norm factor imprieng H magnitude on dipole, rest position
```

```
H0norm = computeDipoleH0Norm(Hmag, m, [0; 0 ;-(z0 + rsp)]);
% compute dipole H-field for rest position in y = 0 layer
H = computeDipoleHField(X, Y, Z, m, H0norm);
% calculate magnitudes for each point in the grid
Habs = reshape(sqrt(sum(H.^2, 1)), nSamples, nSamples);
% split H-field in componets and reshape to meshgrid
Hx = reshape(H(1,:), nSamples, nSamples) ./ Habs;
Hy = reshape(H(2,:), nSamples, nSamples) ./ Habs;
Hz = reshape(H(3,:), nSamples, nSamples) ./ Habs;
% exculde value within the spherical magnet, < rsp
innerField = X.^2 + Z.^2 <= rsp.^2;
Habs(innerField) = NaN;
% find relevant magnitudes at anounced distances
Hd = interp2(X, Z, Habs, xd, zd, 'nearest', NaN);


% figure dipole magnet
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fig = figure('Name', 'Dipole Magnet', ...
    'NumberTitle' , 'off', ...
    'WindowStyle', 'normal', ...
    'MenuBar', 'none', ...
    'ToolBar', 'none', ...
    'Units', 'centimeters', ...
    'OuterPosition', [0 0 30 30], ...
    'PaperType', 'a4', ...
    'PaperUnits', 'centimeters', ...
    'PaperOrientation', 'landscape', ...
    'PaperPositionMode', 'auto', ...
    'DoubleBuffer', 'on', ...
    'RendererMode', 'manual', ...
    'Renderer', 'painters');

% plot magnitude as colormap
imagesc(x, z, log10(Habs), 'AlphaData', 1);
set(gca, 'YDir', 'normal');
colormap('jet');
shading flat;

% set colorbar to log10 scaling of map
cb = colorbar;
cb.Label.String = '$log_{10}(|H|)$ in kA/m';
cb.Label.Interpreter = 'latex';
%cb.Label.FontSize = 16;

hold on;
grid on;

% plot field lines
st = streamslice(X, Z, Hx, Hz, 'noarrows', 'cubic');
set(st, 'Color', 'k');

% plot field vectors
quiver(X(slice, slice), Z(slice, slice), Hx(slice, slice), ...
    Hz(slice, slice), 0.5, 'k');

% plot magnet with north and south pole
rectangle('Position', [-rsp -rsp 2*rsp 2*rsp], 'Curvature', [1 1]);
semicrc = rsp.*[cos(pz); sin(pz)];
```

```
patch(semicrc(1,:), semicrc(2,:),'r');
patch(-semicrc(1,:), -semicrc(2,:),'g');
text(-1.25, 0, 'N', 'FontSize', 18);
text(0.5, 0, 'S', 'FontSize', 18);

% additional figure text and lines
text(-(xz-1), -(xz-1), ...
    sprintf('$\\mathbf{Y = %.1f}$ \\textbf{mm}', y), ...
    'Color', 'w', ...
    'FontSize', 20, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

% distance scale in -z direction for x =0, distance from magnet surface
line(xd, zd, 'Marker', '_', 'LineStyle', '-',...
    'Color', 'w', 'LineWidth', 2.0);

% place text along marker
for i = 2:length(zd)-1
    text(0.5, zd(i), ...
        sprintf(...
        '\\textbf{$\\mathbf{d_z = %d}$ mm, $\\mathbf{|H| = %.1f}$ kA/m}', ...
        abs(zd(i))-rsp, Hd(i)), ...
        'Color', 'w', ...
        'FontSize', 18, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');
end

% limits ticks and labels
xlim([-xz xz]);
ylim([-xz xz]);

xticks(-xz:xz);
yticks(-xz:xz);

labels = string(xticks);
labels(1:2:end) = "";
xticklabels(labels)
yticklabels(labels)

% axis shape set
axis equal;
axis tight;

% title and figure labels
title('Approximated Spherical Magnet with Dipole Far Field', ...
    'FontWeight', 'normal', ...
    ...'FontSize', 18, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

subtitle(...
    [sprintf(...
    "Sphere whith imprinted H-field magnitude of $%.1f$ kA/m", Hmag); ...
    sprintf("at distance $d = %.1f$ mm with $d_z = |z| - r_{sp}$", z0) + ...
    sprintf(" and sphere radius $r_{sp} = %.1f$ mm", rsp)], ...
    'FontWeight', 'normal', ...
    ...'FontSize', 14, ...
    'FontName', 'Times', ...
```

```
        'Interpreter', 'latex');

    xlabel('$X$ in mm', ...
        'FontWeight', 'normal', ...
        ...'FontSize', 16, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$Z$ in mm', ...
        'FontWeight', 'normal', ...
        ...'FontSize', 16, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    set(0, 'defaultaxesfontsize', 20);
    set(0, 'defaulttextinterpreter','latex');

    % save results of figure
    yesno = input('Save? [y/n]: ', 's');
    if strcmp(yesno, 'y')
        savefig(fig, figPath);
        print(fig, figSvgPath, '-dsvg');
        print(fig, figEpsPath, '-depsc', '-tiff', '-loose');
        print(fig, figPdfPath, '-dpdf', '-loose', '-fillpage');
    end
    close(fig)
end
```

Published with MATLAB® R2020b

plotSimulationDataset

Search for available trainings or test dataset and plot dataset. Follow user input dialog to choose which dataset and decide how many angles to plot. Save dataset content redered to an avi-file. Filename same as dataset.

Contents

Syntax

```
plotSimulationDataset()
```

Description

plotSimulationDataset() plot training or test dataset which are loacated in data/test or data/training. The function list all datasets and the user must decide during user input dialog which dataset to plot and how many angles to to visualize. It loads path from config.mat and scans for file automatically.

Examples

```
plotSimulationDataset()
```

Input Argurments

None

Output Argurments

None

Requirements

- Other m-files required: None
- Subfunctions: None
- MAT-files required: config.mat

See Also

- generateSimulationDatasets
- sensorArraySimulation
- generateConfigMat

Created on November 25. 2020 by Tobias Wulf. Copyright Tobias Wulf 2020.

```
function plotSimulationDataset()
```

```
% scan for datasets and load needed configurations %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
try
    disp('Plot simulation dataset ...');
    close all;
    % load path variables
    load('config.mat', 'PathVariables');
    % scan for datasets
    TrainingDatasets = dir(fullfile(PathVariables.trainingDataPath, ...
        'Training_*.mat'));
    TestDatasets = dir(fullfile(PathVariables.testDataPath, 'Test_*.mat'));
    allDatasets = [TrainingDatasets; TestDatasets];
    % check if files available
    if isempty(allDatasets)
        error('No training or test datasets found.');
    end
catch ME
    rethrow(ME)
end

% display availabe datasets to user, decide which to plot %%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% number of datasets
nDatasets = length(allDatasets);
fprintf('Found %d datasets:\n', nDatasets)
for i = 1:nDatasets
    fprintf('%s\t:\t(%d)\n', allDatasets(i).name, i)
end
% get numeric user input to indicate which dataset to plot
iDataset = input('Type number to choose dataset to plot to: ');
% iDataset = 2;

% load dataset and ask user which one and how many angles %%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
try
    ds = load(fullfile(allDatasets(iDataset).folder, ...
        allDatasets(iDataset).name));
    % check how many angles in dataset and let user decide how many to
    % render in polt
    fprintf('Detect %d angles in dataset ...\n',...
        ds.Info.UseOptions.nAngles);
    nSubAngles = input('How many angles to you wish to plot: ');
    % nSubAngles = 120;
    % indices for data to plot, get sample distance for even distance
    sampleDistance = length(downsample(ds.Data.angles, nSubAngles));
    % get subset of angles
    subAngles = downsample(ds.Data.angles, sampleDistance);
    nSubAngles = length(subAngles); % just ensure
    % get indices for subset data
    indices = find(ismember(ds.Data.angles, subAngles));
catch ME
    rethrow(ME)
end

% create dataset figure for a subset or all angle %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fig = figure('Name', 'Sensor Array', ...
    'NumberTitle' , 'off', ...
    'WindowStyle', 'normal', ...
```

```
        'MenuBar', 'none', ...
        'ToolBar', 'none', ...
        'Units', 'centimeters', ...
        'OuterPosition', [0 0 30 30], ...
        'PaperType', 'a4', ...
        'PaperUnits', 'centimeters', ...
        'PaperOrientation', 'landscape', ...
        'PaperPositionMode', 'auto', ...
        'DoubleBuffer', 'on', ...
        'RendererMode', 'manual', ...
        'Renderer', 'painters');

    tdl = tiledlayout(fig, 2, 2, ...
        'Padding', 'normal', ...
        'TileSpacing' , 'compact');


    title(tdl, 'Sensor Array Simulation', ...
        'FontWeight', 'normal', ...
        'FontSize', 18, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    subline1 = "Sensor Array (%s) of $%d\\times%d$ sensors, an edge" + ...
        " length of $%.1f$ mm, a rel. pos. to magnet surface of";
    subline2 = " $(%.1f, %.1f, -(%.1f))$ in mm, a magnet" + ...
        " tilt of $%.1f^\\circ$, a sphere radius of $%.1f$ mm, a imprinted";
    subline3 = "field strength of $%.1f$ kA/m at $%.1f$ mm" + ...
        " from sphere surface in z-axis, $%d$ rotation angles with a ";
    subline4 = "step width of $%.1f^\\circ$ and a resolution" + ...
        " of $%.1f^\\circ$. Visualized is a subset of $%d$ angles in ";
    subline5 = "sample distance of $%d$ angles. Based on %s" + ...
        " characterization reference %s.";
    sub = [sprintf(subline1, ...
                    ds.Info.SensorArrayOptions.geometry, ...
                    ds.Info.SensorArrayOptions.dimension, ...
                    ds.Info.SensorArrayOptions.dimension, ...
                    ds.Info.SensorArrayOptions.edge); ...
           sprintf(subline2, ...
                    ds.Info.UseOptions.xPos, ...
                    ds.Info.UseOptions.yPos, ...
                    ds.Info.UseOptions.zPos, ...
                    ds.Info.UseOptions.tilt, ...
                    ds.Info.DipoleOptions.sphereRadius); ...
           sprintf(subline3, ...
                    ds.Info.DipoleOptions.H0mag, ...
                    ds.Info.DipoleOptions.z0, ...
                    ds.Info.UseOptions.nAngles); ...
           sprintf(subline4, ...
                    ds.Data.angleStep, ...
                    ds.Info.UseOptions.angleRes, ...
                    nSubAngles)
           sprintf(subline5, ...
                    sampleDistance, ...
                    ds.Info.CharData, ...
                    ds.Info.UseOptions.BridgeReference)];

    subtitle(tdl, sub, ...
        'FontWeight', 'normal', ...
        'FontSize', 14, ...
```

```
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % get subset of needed data to plot, only one load %%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    N = ds.Info.SensorArrayOptions.dimension;
    X = ds.Data.X;
    Y = ds.Data.Y;
    Z = ds.Data.Z;

    % calc limits of plot 1
    maxX = ds.Info.UseOptions.xPos + ds.Info.SensorArrayOptions.edge;
    maxY = ds.Info.UseOptions.yPos + ds.Info.SensorArrayOptions.edge;
    minX = ds.Info.UseOptions.xPos - ds.Info.SensorArrayOptions.edge;
    minY = ds.Info.UseOptions.yPos - ds.Info.SensorArrayOptions.edge;

    % calculate colormap to identify scatter points
    c=zeros(N,N,3);
    for i = 1:N
        for j = 1:N
            c(i,j,:) = [(2*N+1-2*i), (2*N+1-2*j), (i+j)]/2/N;
        end
    end
    c = squeeze(reshape(c, N^2, 1, 3));

    % load offset voltage to subtract from cosinus, sinus voltage
    Voff = ds.Info.SensorArrayOptions.Voff;

    % plot sensor grid in x and y coordinates and constant z layer %%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    ax1 = nexttile(1);
    % plot each cooredinate in loop to create a special shading constant
    % reliable to orientation for all matrice
    hold on;
    scatter(X(:), Y(:), [], c, 'filled', 'MarkerEdgeColor', 'k', ...
        'LineWidth', 0.8);

    % axis shape and ticks
    axis square xy;
    axis tight;
    grid on;
    xlim([minX maxX]);
    ylim([minY maxY]);

    % text and labels
    text(minX+0.2, minY+0.2, ...
        sprintf('$Z = %.1f$ mm', Z(1)), ...
        'Color', 'k', ...
        'FontSize', 16, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    xlabel('$X$ in mm', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$Y$ in mm', ...
        'FontWeight', 'normal', ...
```

```
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title(sprintf('Sensor Array $%d\\times%d$', N, N), ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    hold off;

    % plot rotation angles in polar view %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    nexttile(2);
    % plot all angles grayed out
    polarscatter(ds.Data.angles/180*pi, ...
        ones(1, ds.Info.UseOptions.nAngles), ...
        [], [0.8 0.8 0.8], 'filled');

    % radius ticks and label
    rticks(1);
    rticklabels("");
    hold on;

    % plot subset of angles
    % polarscatter(subAngles/180*pi, ones(1, nSubAngles), ...
    %     'k', 'LineWidth', 0.8);
    ax2 = gca;

    % axis shape
    axis tight;

    % text an labels
    % init first rotation step label
    tA = text(2/3*pi, 1.5, ...
        '$\\theta$', ...
        'Color', 'b', ...
        'FontSize', 16, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('Rotation around Z-Axis in Degree', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    hold off;

    % Cosinus bridge outputs for rotation step %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    ax3 = nexttile(3);
    hold on;

    % set colormap
    colormap('gray');

    % plot cosinus reference, set NaN values to white color, orient Y to normal
    imC = imagesc(ds.Data.HxScale, ds.Data.HyScale, ds.Data.VcosRef);
```

```
    set(imC, 'AlphaData', ~isnan(ds.Data.VcosRef));
    set(gca, 'YDir', 'normal')

    % axis shape and ticks
    axis square xy;
    axis tight;
    yticks(xticks);
    grid on;

    % test and labels
    xlabel('$H_x$ in kA/m', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$H_y$ in kA/m', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('$V_{cos}(H_x, H_y)$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % add colorbar and place it
    cb1 = colorbar;
    cb1.Label.String = sprintf(...
        '$V_{cos}(H_x, H_y)$ in V, $V_{cc} = %1.1f$ V, $V_{off} = %1.2f$ V', ...
        ds.Info.SensorArrayOptions.Vcc, ds.Info.SensorArrayOptions.Voff);
    cb1.Label.Interpreter = 'latex';
    cb1.Label.FontSize = 12;

    hold off;

    % Sinus bridge outputs for rotation step %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    ax4 = nexttile(4);
    hold on;

    % set colormap
    colormap('gray');

    % plot sinus reference, set NaN values to white color, orient Y to normal
    imS = imagesc(ds.Data.HxScale, ds.Data.HyScale, ds.Data.VsinRef);
    set(imS, 'AlphaData', ~isnan(ds.Data.VsinRef));
    set(gca, 'YDir', 'normal')

    % axis shape and ticks
    axis square xy;
    axis tight;
    yticks(xticks);
    grid on;

    % test and labels
    xlabel('$H_x$ in kA/m', ...
        'FontWeight', 'normal', ...
```

```
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$H_y$ in kA/m', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('$V_{sin}(H_x, H_y)$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % add colorbar and place it
    cb2 = colorbar;
    cb2.Label.String = sprintf(...
        '$V_{sin}(H_x, H_y)$ in V, $V_{cc} = %1.1f$ V, $V_{off} = %1.2f$ V', ...
        ds.Info.SensorArrayOptions.Vcc, ds.Info.SensorArrayOptions.Voff);
    cb2.Label.Interpreter = 'latex';
    cb2.Label.FontSize = 12;

    hold off;

    % zoom axes for scatter on cosinuns reference images %%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    nexttile(3);
    ax5 = axes('Position', [0.07 0.02 0.19 0.19], 'XColor', 'r', 'YColor', 'r');
    hold on;
    axis square xy;
    grid on;
    hold off;

    % draw everything prepared before start renewing frame wise and prepare for
    % recording frames to video file %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % draw frame
    drawnow;

    % get file path and change extension
    [~, fName, ~] = fileparts(ds.Info.filePath);
    fPath = PathVariables.saveImagesPath;

    % string allows simple cat ops
    VW = VideoWriter(fullfile(fPath, 'avi', fName + ".avi"), ...
        "Uncompressed AVI");

    % scale frame rate on 10 second movies, ensure at least 1 fps
    fr = floor(nSubAngles / 10) + 1;
    VW.FrameRate = fr;

    % open video file, ready to record frames
    open(VW)


    % loop through subset angle dataset and renew plots %%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i = indices
```

```
% H load subset
Hx = ds.Data.Hx(:,:,i);
Hy = ds.Data.Hy(:,:,i);
% get min max
maxHx = max(Hx, [], 'all');
maxHy = max(Hy, [], 'all');
minHx = min(Hx, [], 'all');
minHy = min(Hy, [], 'all');
dHx = abs(maxHx - minHx);
dHy = abs(maxHy - minHy);

% load V subset
Vcos = ds.Data.Vcos(:,:,i) - Voff;
Vsin = ds.Data.Vsin(:,:,i) - Voff;
angle = ds.Data.angles(i);

% lock plots
hold(ax1, 'on');
hold(ax2, 'on');
hold(ax3, 'on');
hold(ax4, 'on');
hold(ax5, 'on');

% update plot 1
qH = quiver(ax1, X, Y, Hx, Hy, 0.5, 'b');
qV = quiver(ax1, X, Y, Vcos, Vsin, 0.5, 'r');
legend([qH qV], {'$quiver(H_x,H_y)$', ...
    '$quiver(V_{cos}-V_{off},V_{sin}-V_{off})$'},...
    'FontWeight', 'normal', ...
    'FontSize', 9, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex', ...
    'Location', 'NorthEast');

% update plot 2
tA.String = sprintf('$%.1f^\\circ$', angle);
pA = polarscatter(ax2, angle/180*pi, 1, 'b', 'filled', ...
    'MarkerEdgeColor', 'k', 'LineWidth', 0.8);

% update plot 3 and 4
sC = scatter(ax3, Hx(:), Hy(:), 5, c, 'filled', ...
    'MarkerEdgeColor', 'k', ...
    'LineWidth', 0.8);
sS = scatter(ax4, Hx(:), Hy(:), 5, c, 'filled', ...
    'MarkerEdgeColor', 'k', ...
    'LineWidth', 0.8);

% calc position of scatter area frame and reframe
pos = [minHx - 0.3 * dHx, minHy - 0.3 * dHy, 1.6 * dHx, 1.6 * dHy];
rtC = rectangle(ax3, 'Position', pos, 'LineWidth', 1,...
    'EdgeColor', 'r');
rtS = rectangle(ax4, 'Position', pos, 'LineWidth', 1,...
    'EdgeColor', 'r');

% update plot 5 (zoom)
sZ = scatter(ax5, Hx(:), Hy(:), [], c, 'filled', ...
    'MarkerEdgeColor', 'k', ...
    'LineWidth', 0.8);
xlim(ax5, [pos(1) maxHx + 0.3 * dHx])
ylim(ax5, [pos(2) maxHy + 0.3 * dHy])
```

```
        % release plots
        hold(ax1, 'off');
        hold(ax2, 'off');
        hold(ax3, 'off');
        hold(ax4, 'off');
        hold(ax5, 'off');

        % draw frame
        drawnow;

        % record frame to file
        frame = getframe(fig);
        writeVideo(VW, frame);

        % delete part of plots to renew for current angle, delete but last
        if i ~= indices(end)
            delete(qH);
            delete(qV);
            delete(pA);
            delete(rtC);
            delete(rtS);
            delete(sC);
            delete(sS);
            delete(sZ);
        end
    end
    % close video file
    close(VW)
    close(fig)
end
```

Published with MATLAB® R2020b

plotSingleSimulationAngle

Search for available trainings or test dataset and plot dataset. Follow user input dialog to choose which dataset and decide how many angles to plot. Plot single Angle and save figure to file. File name same as dataset with attach angle index.

Contents

Syntax

```
plotSingleSimulationAngle()
```

Description

plotSingleSimulationAngle() plot training or test dataset which are loacated in data/test or data/training. The function list all datasets and the user must decide during user input dialog which dataset to plot and which angle to visualize to. It loads path from config.mat and scans for file automatically.

Examples

```
plotSingleSimulationAngle()
```

Input Argurments

None

Output Argurments

None

Requirements

- Other m-files required: None
- Subfunctions: None
- MAT-files required: config.mat

See Also

- generateSimulationDatasets
- sensorArraySimulation
- generateConfigMat

Created on November 28. 2020 by Tobias Wulf. Copyright Tobias Wulf 2020.

```
function plotSingleSimulationAngle()
```

```
% scan for datasets and load needed configurations %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
try
    disp('Plot single simulation angle ...');
    close all;
    % load path variables
    load('config.mat', 'PathVariables');
    % scan for datasets
    TrainingDatasets = dir(fullfile(PathVariables.trainingDataPath, ...
        'Training_*.mat'));
    TestDatasets = dir(fullfile(PathVariables.testDataPath, 'Test_*.mat'));
    allDatasets = [TrainingDatasets; TestDatasets];
    % check if files available
    if isempty(allDatasets)
        error('No training or test datasets found.');
    end
catch ME
    rethrow(ME)
end

% display availabe datasets to user, decide which to plot %%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% number of datasets
nDatasets = length(allDatasets);
fprintf('Found %d datasets:\n', nDatasets)
for i = 1:nDatasets
    fprintf('%s\t:\t(%d)\n', allDatasets(i).name, i)
end
% get numeric user input to indicate which dataset to plot
iDataset = input('Type number to choose dataset to plot to: ');
% iDataset = 2;

% load dataset and ask user which one and how many angles %%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
try
    ds = load(fullfile(allDatasets(iDataset).folder, ...
        allDatasets(iDataset).name));
    % check how many angles in dataset and let user decide how many to
    % render in polt
    fprintf('Detect %d angles ([1:%d]) in dataset ...\n', ...
        ds.Info.UseOptions.nAngles, ds.Info.UseOptions.nAngles);
    fprintf('Resolution\t:\t%.1f\n', ds.Info.UseOptions.angleRes);
    fprintf('Step width\t:\t%.1f\n', ds.Data.angleStep);
    fprintf('Start angle\t:\t%.1f\n', ds.Data.angles(1))
    idx = input('Which angle do you wish to plot (enter index): ');
    angle = interp1(ds.Data.angles, idx, 'nearest');
catch ME
    rethrow(ME)
end

% figure save path for different formats %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fPath = fullfile(PathVariables.saveFiguresPath);
fSvgPath = fullfile(PathVariables.saveImagesPath, 'svg');
fEpsPath = fullfile(PathVariables.saveImagesPath, 'eps');
fPdfPath = fullfile(PathVariables.saveImagesPath, 'pdf');

% create dataset figure for a subset or all angle %%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
    fig = figure('Name', 'Sensor Array', ...
        'NumberTitle' , 'off', ...
        'WindowStyle', 'normal', ...
        'MenuBar', 'none', ...
        'ToolBar', 'none', ...
        'Units', 'centimeters', ...
        'OuterPosition', [0 0 30 30], ...
        'PaperType', 'a4', ...
        'PaperUnits', 'centimeters', ...
        'PaperOrientation', 'landscape', ...
        'PaperPositionMode', 'auto', ...
        'DoubleBuffer', 'on', ...
        'RendererMode', 'manual', ...
        'Renderer', 'painters');

    tdl = tiledlayout(fig, 2, 2, ...
        'Padding', 'normal', ...
        'TileSpacing' , 'compact');


    title(tdl, 'Sensor Array Simulation', ...
        'FontWeight', 'normal', ...
        'FontSize', 18, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    subline1 = "Sensor Array (%s) of $%d\\times%d$ sensors, " + ...
        "an edge length of $%.1f$ mm, a rel. pos. to magnet surface of";
    subline2 = " $(%.1f, %.1f, -(%.1f))$ in mm, a magnet tilt" + ...
        " of $%.1f^\\circ$, a sphere radius of $%.1f$ mm, a imprinted";
    subline3 = "field strength of $%.1f$ kA/m at $%.1f$ mm from" + ...
        " sphere surface in z-axis, $%d$ rotation angles with a ";
    subline4 = "step width of $%.1f^\\circ$ and a resolution of" + ...
        " $%.1f^\\circ$. Visualized is rotatation angle %d $(%.1f^\\circ)$.";
    subline5 = "Based on %s characterization reference %s.";
    sub = [sprintf(subline1, ...
                    ds.Info.SensorArrayOptions.geometry, ...
                    ds.Info.SensorArrayOptions.dimension, ...
                    ds.Info.SensorArrayOptions.dimension, ...
                    ds.Info.SensorArrayOptions.edge); ...
           sprintf(subline2, ...
                    ds.Info.UseOptions.xPos, ...
                    ds.Info.UseOptions.yPos, ...
                    ds.Info.UseOptions.zPos, ...
                    ds.Info.UseOptions.tilt, ...
                    ds.Info.DipoleOptions.sphereRadius); ...
           sprintf(subline3, ...
                    ds.Info.DipoleOptions.H0mag, ...
                    ds.Info.DipoleOptions.z0, ...
                    ds.Info.UseOptions.nAngles); ...
           sprintf(subline4, ...
                    ds.Data.angleStep, ...
                    ds.Info.UseOptions.angleRes, ...
                    idx, angle)
           sprintf(subline5, ...
                    ds.Info.CharData, ...
                    ds.Info.UseOptions.BridgeReference)];

    subtitle(tdl, sub, ...
        'FontWeight', 'normal', ...
```

```
        'FontSize', 14, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % get subset of needed data to plot, only one load %%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    N = ds.Info.SensorArrayOptions.dimension;
    X = ds.Data.X;
    Y = ds.Data.Y;
    Z = ds.Data.Z;

    % calc limits of plot 1
    maxX = ds.Info.UseOptions.xPos + ds.Info.SensorArrayOptions.edge;
    maxY = ds.Info.UseOptions.yPos + ds.Info.SensorArrayOptions.edge;
    minX = ds.Info.UseOptions.xPos - ds.Info.SensorArrayOptions.edge;
    minY = ds.Info.UseOptions.yPos - ds.Info.SensorArrayOptions.edge;

    % calculate colormap to identify scatter points
    c=zeros(N,N,3);
    for i = 1:N
        for j = 1:N
            c(i,j,:) = [(2*N+1-2*i), (2*N+1-2*j), (i+j)]/2/N;
        end
    end
    c = squeeze(reshape(c, N^2, 1, 3));

    % load offset voltage to subtract from cosinus, sinus voltage
    Voff = ds.Info.SensorArrayOptions.Voff;

    % plot sensor grid in x and y coordinates and constant z layer %%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    ax1 = nexttile(1);
    % plot each cooredinate in loop to create a special shading constant
    % reliable to orientation for all matrice
    hold on;
    scatter(X(:), Y(:), [], c, 'filled', 'MarkerEdgeColor', 'k', ...
        'LineWidth', 0.8);

    % axis shape and ticks
    axis square xy;
    axis tight;
    grid on;
    xlim([minX maxX]);
    ylim([minY maxY]);

    % text and labels
    text(minX+0.2, minY+0.2, ...
        sprintf('$Z = %.1f$ mm', Z(1)), ...
        'Color', 'k', ...
        'FontSize', 16, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    xlabel('$X$ in mm', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$Y$ in mm', ...
```

```
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title(sprintf('Sensor Array $%d\\times%d$', N, N), ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    hold off;

    % plot rotation angles in polar view %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    nexttile(2);
    % plot all angles grayed out
    polarscatter(ds.Data.angles/180*pi, ones(1, ds.Info.UseOptions.nAngles), ...
        [], [0.8 0.8 0.8], 'filled');

    % radius ticks and label
    rticks(1);
    rticklabels("");
    hold on;

    % plot subset of angles
    % polarscatter(subAngles/180*pi, ones(1, nSubAngles),...
    %     'k', 'LineWidth', 0.8);
    ax2 = gca;

    % axis shape
    axis tight;

    % text an labels
    % init first rotation step label
    tA = text(2/3*pi, 1.5, ...
        '$\\theta$', ...
        'Color', 'b', ...
        'FontSize', 16, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('Rotation around Z-Axis in Degree', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    hold off;

    % Cosinus bridge outputs for rotation step %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    ax3 = nexttile(3);
    hold on;

    % set colormap
    colormap('gray');

    % plot cosinus reference, set NaN values to white color, orient Y to normal
    imC = imagesc(ds.Data.HxScale, ds.Data.HyScale, ds.Data.VcosRef);
```

```
    set(imC, 'AlphaData', ~isnan(ds.Data.VcosRef));
    set(gca, 'YDir', 'normal')

    % axis shape and ticks
    axis square xy;
    axis tight;
    yticks(xticks);
    grid on;

    % test and labels
    xlabel('$H_x$ in kA/m', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$H_y$ in kA/m', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('$V_{cos}(H_x, H_y)$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % add colorbar and place it
    cb1 = colorbar;
    cb1.Label.String = sprintf(...
        '$V_{cos}(H_x, H_y)$ in V, $V_{cc} = %1.1f$ V, $V_{off} = %1.2f$ V', ...
        ds.Info.SensorArrayOptions.Vcc, ds.Info.SensorArrayOptions.Voff);
    cb1.Label.Interpreter = 'latex';
    cb1.Label.FontSize = 12;

    hold off;

    % Sinus bridge outputs for rotation step %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    ax4 = nexttile(4);
    hold on;

    % set colormap
    colormap('gray');

    % plot sinus reference, set NaN values to white color, orient Y to normal
    imS = imagesc(ds.Data.HxScale, ds.Data.HyScale, ds.Data.VsinRef);
    set(imS, 'AlphaData', ~isnan(ds.Data.VsinRef));
    set(gca, 'YDir', 'normal')

    % axis shape and ticks
    axis square xy;
    axis tight;
    yticks(xticks);
    grid on;

    % test and labels
    xlabel('$H_x$ in kA/m', ...
        'FontWeight', 'normal', ...
```

```
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$H_y$ in kA/m', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('$V_{sin}(H_x, H_y)$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % add colorbar and place it
    cb2 = colorbar;
    cb2.Label.String = sprintf(...
        '$V_{sin}(H_x, H_y)$ in V, $V_{cc} = %1.1f$ V, $V_{off} = %1.2f$ V', ...
        ds.Info.SensorArrayOptions.Vcc, ds.Info.SensorArrayOptions.Voff);
    cb2.Label.Interpreter = 'latex';
    cb2.Label.FontSize = 12;

    hold off;

    % zoom axes for scatter on cosinuns reference images %%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    nexttile(3);
    ax5 = axes('Position', [0.07 0.02 0.19 0.19], 'XColor', 'r', 'YColor', 'r');
    hold on;
    axis square xy;
    grid on;
    hold off;


    % plot angle into plots %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % H load subset
    Hx = ds.Data.Hx(:,:,idx);
    Hy = ds.Data.Hy(:,:,idx);
    % get min max
    maxHx = max(Hx, [], 'all');
    maxHy = max(Hy, [], 'all');
    minHx = min(Hx, [], 'all');
    minHy = min(Hy, [], 'all');
    dHx = abs(maxHx - minHx);
    dHy = abs(maxHy - minHy);

    % load V subset
    Vcos = ds.Data.Vcos(:,:,idx) - Voff;
    Vsin = ds.Data.Vsin(:,:,idx) - Voff;
    angle = ds.Data.angles(idx);

    % lock plots
    hold(ax1, 'on');
    hold(ax2, 'on');
    hold(ax3, 'on');
    hold(ax4, 'on');
    hold(ax5, 'on');
```

```
    % update plot 1
    qH = quiver(ax1, X, Y, Hx, Hy, 0.5, 'b');
    qV = quiver(ax1, X, Y, Vcos, Vsin, 0.5, 'r');
    legend([qH qV], {'$quiver(H_x,H_y)$', ...
        '$quiver(V_{cos}-V_{off},V_{sin}-V_{off})$'},...
        'FontWeight', 'normal', ...
        'FontSize', 9, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex', ...
        'Location', 'NorthEast');

    % update plot 2
    tA.String = sprintf('$%.1f^\\circ$', angle);
    polarscatter(ax2, angle/180*pi, 1, 'b', 'filled', ...
        'MarkerEdgeColor', 'k', 'LineWidth', 0.8);

    % update plot 3 and 4
    scatter(ax3, Hx(:), Hy(:), 5, c, 'filled', 'MarkerEdgeColor', 'k', ...
        'LineWidth', 0.8);
    scatter(ax4, Hx(:), Hy(:), 5, c, 'filled', 'MarkerEdgeColor', 'k', ...
        'LineWidth', 0.8);

    % calc position of scatter area frame and reframe
    pos = [minHx - 0.3 * dHx, minHy - 0.3 * dHy, 1.6 * dHx, 1.6 * dHy];
    rectangle(ax3, 'Position', pos, 'LineWidth', 1,  'EdgeColor', 'r');
    rectangle(ax4, 'Position', pos, 'LineWidth', 1,  'EdgeColor', 'r');

    % update plot 5 (zoom)
    scatter(ax5, Hx(:), Hy(:), [], c, 'filled', 'MarkerEdgeColor', 'k', ...
        'LineWidth', 0.8);
    xlim(ax5, [pos(1) maxHx + 0.3 * dHx])
    ylim(ax5, [pos(2) maxHy + 0.3 * dHy])

    % release plots
    hold(ax1, 'off');
    hold(ax2, 'off');
    hold(ax3, 'off');
    hold(ax4, 'off');
    hold(ax5, 'off');

    % save figure to file %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % get file path to save figure with angle index
    [~, fName, ~] = fileparts(ds.Info.filePath);

    % save to various formats
    yesno = input('Save? [y/n]: ', 's');
    if strcmp(yesno, 'y')
        fLabel = input('Enter file label: ', 's');
        fName = fName + sprintf("_AnglePlot_%d_", idx) + fLabel;
        savefig(fig, fullfile(fPath, fName));
        print(fig, fullfile(fSvgPath, fName), '-dsvg');
        print(fig, fullfile(fEpsPath, fName), '-depsc', '-tiff', '-loose');
        print(fig, fullfile(fPdfPath, fName), '-dpdf', '-loose', '-fillpage');
    end
    close(fig);
end
```

Published with MATLAB® R2020b

plotSimulationSubset

Search for available trainings or test dataset and plot dataset. Follow user input dialog to choose which dataset and decide which array elements to plot. Save created plot to file. Filename same as dataset with attached info.

Contents

Syntax

```
plotSimulationSubset()
```

Description

plotSimulationSubset() plot training or test dataset which are loacated in data/test or data/training. The function list all datasets and the user must decide during user input dialog which dataset to plot and how many angles to to visualize. It loads path from config.mat and scans for file automatically.

Examples

```
plotSimulationSubset()
```

Input Argurments

None

Output Argurments

None

Requirements

- Other m-files required: None
- Subfunctions: None
- MAT-files required: config.mat

See Also

- generateSimulationDatasets
- sensorArraySimulation
- generateConfigMat

Created on November 29. 2020 by Tobias Wulf. Copyright Tobias Wulf 2020.

```
function plotSimulationSubset()
```

```
% scan for datasets and load needed configurations %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
try
    disp('Plot simulation dataset ...');
    close all;
    % load path variables
    load('config.mat', 'PathVariables');
    % scan for datasets
    TrainingDatasets = dir(fullfile(PathVariables.trainingDataPath, ...
        'Training_*.mat'));
    TestDatasets = dir(fullfile(PathVariables.testDataPath, 'Test_*.mat'));
    allDatasets = [TrainingDatasets; TestDatasets];
    % check if files available
    if isempty(allDatasets)
        error('No training or test datasets found.');
    end
catch ME
    rethrow(ME)
end

% display availabe datasets to user, decide which to plot %%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% number of datasets
nDatasets = length(allDatasets);
fprintf('Found %d datasets:\n', nDatasets)
for i = 1:nDatasets
    fprintf('%s\t:\t(%d)\n', allDatasets(i).name, i)
end
% get numeric user input to indicate which dataset to plot
iDataset = input('Type number to choose dataset to plot to: ');

% load dataset and ask user which one and how many angles %%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
try
    ds = load(fullfile(allDatasets(iDataset).folder, ...
        allDatasets(iDataset).name));
    % check how many angles in dataset and let user decide how many to
    % render in polt
    fprintf('Detect %d x %d sensors in dataset ...\n', ...
        ds.Info.SensorArrayOptions.dimension, ...
        ds.Info.SensorArrayOptions.dimension);
    xIdx = input("Enter x indices in []: ");
    yIdx = input("Enter y indices in []: ");
    if length(xIdx) ~= length(yIdx)
        error('Indices must have the same length!')
    end
    fprintf('Detect %d angles in dataset ...\n', ...
        ds.Info.UseOptions.nAngles);
    nSubAngles = input('How many angles to you wish to plot: ');
    % indices for data to plot, get sample distance for even distance
    sampleDistance = length(downsample(ds.Data.angles, nSubAngles));
    % get subset of angles
    subAngles = downsample(ds.Data.angles, sampleDistance);
    nSubAngles = length(subAngles); % just ensure
    % get indices for subset data
    angleIdx = find(ismember(ds.Data.angles, subAngles));
catch ME
    rethrow(ME)
end
```

```
% figure save path for different formats %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fPath = fullfile(PathVariables.saveFiguresPath);
fSvgPath = fullfile(PathVariables.saveImagesPath, 'svg');
fEpsPath = fullfile(PathVariables.saveImagesPath, 'eps');
fPdfPath = fullfile(PathVariables.saveImagesPath, 'pdf');

% create dataset figure for a subset or all angle %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fig = figure('Name', 'Sensor Array', ...
    'NumberTitle' , 'off', ...
    'WindowStyle', 'normal', ...
    'MenuBar', 'none', ...
    'ToolBar', 'none', ...
    'Units', 'centimeters', ...
    'OuterPosition', [0 0 37 29], ...
    'PaperType', 'a4', ...
    'PaperUnits', 'centimeters', ...
    'PaperOrientation', 'landscape', ...
    'PaperPositionMode', 'auto', ...
    'DoubleBuffer', 'on', ...
    'RendererMode', 'manual', ...
    'Renderer', 'painters');

tdl = tiledlayout(fig, 3, 4, ...
    'Padding', 'compact', ...
    'TileSpacing' , 'compact');


title(tdl, 'Sensor Array Simulation', ...
    'FontWeight', 'normal', ...
    'FontSize', 18, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

subline1 = "Sensor Array (%s) of $%d\\times%d$ sensors," + ...
    " an edge length of $%.1f$ mm, a rel. pos. to magnet surface of";
subline2 = " $(%.1f, %.1f, -(%.1f))$ in mm, a magnet tilt" + ...
    " of $%.1f^\\circ$, a sphere radius of $%.1f$ mm, a imprinted";
subline3 = "field strength of $%.1f$ kA/m at $%.1f$ mm from" + ...
    " sphere surface in z-axis, $%d$ rotation angles with a ";
subline4 = "step width of $%.1f^\\circ$ and a resolution" + ...
    " of $%.1f^\\circ$. Visualized is a subset of $%d$ angles in ";
subline5 = "sample distance of $%d$ angles. Based on %s" + ...
    " characterization reference %s.";
sub = [sprintf(subline1, ...
                ds.Info.SensorArrayOptions.geometry, ...
                ds.Info.SensorArrayOptions.dimension, ...
                ds.Info.SensorArrayOptions.dimension, ...
                ds.Info.SensorArrayOptions.edge); ...
       sprintf(subline2, ...
                ds.Info.UseOptions.xPos, ...
                ds.Info.UseOptions.yPos, ...
                ds.Info.UseOptions.zPos, ...
                ds.Info.UseOptions.tilt, ...
                ds.Info.DipoleOptions.sphereRadius); ...
       sprintf(subline3, ...
                ds.Info.DipoleOptions.H0mag, ...
                ds.Info.DipoleOptions.z0, ...
```

```
                ds.Info.UseOptions.nAngles); ...
        sprintf(subline4, ...
                ds.Data.angleStep, ...
                ds.Info.UseOptions.angleRes, ...
                nSubAngles)
        sprintf(subline5, ...
                sampleDistance, ...
                ds.Info.CharData, ...
                ds.Info.UseOptions.BridgeReference)];

subtitle(tdl, sub, ...
    'FontWeight', 'normal', ...
    'FontSize', 14, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

% get subset of needed data to plot, only one load %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
N = ds.Info.SensorArrayOptions.dimension;
X = ds.Data.X;
Y = ds.Data.Y;
Z = ds.Data.Z;

% calc limits of plot 1
maxX = ds.Info.UseOptions.xPos + ds.Info.SensorArrayOptions.edge;
maxY = ds.Info.UseOptions.yPos + ds.Info.SensorArrayOptions.edge;
minX = ds.Info.UseOptions.xPos - ds.Info.SensorArrayOptions.edge;
minY = ds.Info.UseOptions.yPos - ds.Info.SensorArrayOptions.edge;

% calculate colormap to identify scatter points
c=zeros(N,N,3);
for i = 1:N
    for j = 1:N
        c(i,j,:) = [(2*N+1-2*i), (2*N+1-2*j), (i+j)]/2/N;
    end
end
c = squeeze(reshape(c, N^2, 1, 3));
% reshape RGB for picking single sensors
R = reshape(c(:,1), N, N);
G = reshape(c(:,2), N, N);
B = reshape(c(:,3), N, N);

% load offset voltage to subtract from cosinus, sinus voltage
Voff = ds.Info.SensorArrayOptions.Voff;
Vcc = ds.Info.SensorArrayOptions.Vcc;

% plot sensor grid in x and y coordinates and constant z layer %%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ax1 = nexttile(1);
% plot each cooredinate in loop to create a special shading constant
% reliable to orientation for all matrice
hold on;

scatter(X(:), Y(:), [], [0.8 0.8 0.8], 'filled', ...
    'MarkerEdgeColor', 'k', 'LineWidth', 0.8);

for k = 1:length(xIdx)
    i = xIdx(k); j = yIdx(k);
    scatter(X(i,j), Y(i,j), [], [R(i,j), G(i,j), B(i,j)], 'filled', ...
        'MarkerEdgeColor', 'k', 'LineWidth', 0.8);
```

```
    end

    % axis shape and ticks
    axis square xy;
    axis tight;
    grid on;
    xlim([minX maxX]);
    ylim([minY maxY]);

    % text and labels
    text(minX+0.2, minY+0.2, ...
        sprintf('$Z = %.1f$ mm', Z(1)), ...
        'Color', 'k', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    xlabel('$X$ in mm', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$Y$ in mm', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title(sprintf('Sensor Array $%d\\times%d$', N, N), ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    hold off;

    % plot rotation angles in polar view %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    nexttile(2);
    % plot all angles grayed out
    polarscatter(ds.Data.angles/180*pi, ones(1, ds.Info.UseOptions.nAngles), ...
        5, [0.8 0.8 0.8], 'filled');

    % radius ticks and label
    rticks(1);
    rticklabels("");
    hold on;

    % plot subset of angles
    polarscatter(subAngles/180*pi, ones(1, nSubAngles), 5, 'b', 'filled');
    ax2 = gca;

    % axis shape
    axis tight;

    % text an labels
    title('Rotation around Z-Axis in Degree', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
```

```
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    hold off;

    % Cosinus bridge outputs for rotation step %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    ax3 = nexttile(3);
    hold on;

    % set colormap
    colormap('gray');

    % plot cosinus reference, set NaN values to white color, orient Y to normal
    imC = imagesc(ds.Data.HxScale, ds.Data.HyScale, ds.Data.VcosRef);
    set(imC, 'AlphaData', ~isnan(ds.Data.VcosRef));
    set(gca, 'YDir', 'normal')

    % axis shape and ticks
    axis square xy;
    axis tight;
    yticks(xticks);
    grid on;

    % test and labels
    xlabel('$H_x$ in kA/m', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$H_y$ in kA/m', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('$V_{cos}(H_x, H_y)$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    hold off;

    % Sinus bridge outputs for rotation step %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    ax4 = nexttile(4);
    hold on;

    % set colormap
    colormap('gray');

    % plot sinus reference, set NaN values to white color, orient Y to normal
    imS = imagesc(ds.Data.HxScale, ds.Data.HyScale, ds.Data.VsinRef);
    set(imS, 'AlphaData', ~isnan(ds.Data.VsinRef));
    set(gca, 'YDir', 'normal')

    % axis shape and ticks
    axis square xy;
```

```
    axis tight;
    yticks(xticks);
    grid on;

    % test and labels
    xlabel('$H_x$ in kA/m', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$H_y$ in kA/m', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('$V_{sin}(H_x, H_y)$', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % add colorbar and place it
    cb2 = colorbar;
    cb2.Label.String = 'in V';
    cb2.Label.Interpreter = 'latex';
    cb2.Label.FontSize = 12;

    hold off;

    % plot Vcos Vsin over angles %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % axes limits
    xlimits = [-10 370];
    ylimits = [min(cat(...
        3, ds.Data.VsinRef, ds.Data.VcosRef), [], 'all') - 0.1*Vcc, ...
        max(cat(3, ds.Data.VsinRef, ds.Data.VcosRef), [], 'all') + 0.1*Vcc];

    % Vcos
    ax5 = nexttile([1 4]);
    yline(Voff, 'k-.', 'LineWidth', 1.2);
    xlim(xlimits);
    ylim(ylimits);
    grid on;

    xlabel('$\theta$ in Degree', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$V{cos}(\theta)$ in V', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title(sprintf(...
        "$V_{cos}$ of Enabled Array Positions over $\\theta$," + ...
```

```
        " $V_{cc} = %.1f$ V, $V_{off} = %.2f$ V", Vcc, Voff), ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % Vsin
    ax6 = nexttile([1 4]);
    yline(Voff, 'k-.', 'LineWidth', 1.2);
    xlim(xlimits);
    ylim(ylimits);
    grid on;

    xlabel('$\theta$ in Degree', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$V{sin}(\theta)$ in V', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title(sprintf("$V_{sin}$ of Enabled Array Positions over" + ...
        " $\\theta$, $V_{cc} = %.1f$ V, $V_{off} = %.2f$ V", Vcc, Voff), ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % loop through subset of dataset and renew plots %%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % lock plots
    hold(ax3, 'on');
    hold(ax4, 'on');
    hold(ax5, 'on');
    hold(ax6, 'on');

    % loop over indices
    for k = 1:length(xIdx)
        i = xIdx(k); j = yIdx(k);
        % H load subset
        Hx = squeeze(ds.Data.Hx(i,j,angleIdx));
        Hy = squeeze(ds.Data.Hy(i,j,angleIdx));
        % get min max

        % load V subset
        Vcos = squeeze(ds.Data.Vcos(i,j,angleIdx));
        Vsin = squeeze(ds.Data.Vsin(i,j,angleIdx));

        % update plot 3, 4, 5 and 6
        scatter(ax3, Hx, Hy, 1, [R(i,j), G(i,j), B(i,j)] , 'filled');
        scatter(ax4, Hx, Hy, 1, [R(i,j), G(i,j), B(i,j)], 'filled');
        scatter(ax5, subAngles, Vcos, 12, [R(i,j), G(i,j), B(i,j)] , ...
            'filled', 'MarkerEdgeColor', 'k', 'LineWidth', 0.5);
        scatter(ax6, subAngles, Vsin, 12, [R(i,j), G(i,j), B(i,j)] , ...
            'filled', 'MarkerEdgeColor', 'k', 'LineWidth', 0.5);
    end
```

```
    % release plots
    hold(ax3, 'off');
    hold(ax4, 'off');
    hold(ax5, 'off');
    hold(ax6, 'off');

    % save figure to file %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % get file path to save figure with angle index
    [~, fName, ~] = fileparts(ds.Info.filePath);

    % save to various formats
    yesno = input('Save? [y/n]: ', 's');
    if strcmp(yesno, 'y')
        fLabel = input('Enter file label: ', 's');
        fName = fName + "_SubsetPlot_" + fLabel;
        savefig(fig, fullfile(fPath, fName));
        print(fig, fullfile(fSvgPath, fName), '-dsvg');
        print(fig, fullfile(fEpsPath, fName), '-depsc', '-tiff', '-loose');
        print(fig, fullfile(fPdfPath, fName), '-dpdf', '-loose', '-fillpage');
    end
    close(fig);
end
```

Published with MATLAB® R2020b

plotSimulationCosSinStats

Search for available trainings or test dataset and plot dataset. Follow user input dialog to choose which dataset to plot and statistics of cos sin. Save created plot to file. Filename same as dataset with attached info.

Contents

Syntax

```
plotSimulationCosSinStats()
```

Description

plotSimulationCosSinStats() plot training or test dataset which are loacated in data/test or data/training. The function list all datasets and the user must decide during user input dialog which dataset to plot. It loads path from config.mat and scans for file automatically.

Examples

```
plotSimulationCosSinStats()
```

Input Argurments

None

Output Argurments

None

Requirements

- Other m-files required: None
- Subfunctions: None
- MAT-files required: config.mat

See Also

- generateSimulationDatasets
- sensorArraySimulation
- generateConfigMat

Created on November 30. 2020 by Tobias Wulf. Copyright Tobias Wulf 2020.

```
function plotSimulationCosSinStats()
```

```
% scan for datasets and load needed configurations %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
try
    disp('Plot simulation dataset ...');
    close all;
    % load path variables
    load('config.mat', 'PathVariables');
    % scan for datasets
    TrainingDatasets = dir(fullfile(PathVariables.trainingDataPath, ...
        'Training_*.mat'));
    TestDatasets = dir(fullfile(PathVariables.testDataPath, 'Test_*.mat'));
    allDatasets = [TrainingDatasets; TestDatasets];
    % check if files available
    if isempty(allDatasets)
        error('No training or test datasets found.');
    end
catch ME
    rethrow(ME)
end

% display availabe datasets to user, decide which to plot %%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% number of datasets
nDatasets = length(allDatasets);
fprintf('Found %d datasets:\n', nDatasets)
for i = 1:nDatasets
    fprintf('%s\t:\t(%d)\n', allDatasets(i).name, i)
end
% get numeric user input to indicate which dataset to plot
iDataset = input('Type number to choose dataset to plot to: ');

% load dataset and ask user which one and how many angles %%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
try
    ds = load(fullfile(allDatasets(iDataset).folder, ...
        allDatasets(iDataset).name));
    % check how many angles in dataset and let user decide how many to
    % render in polt
    fprintf('Detect %d angles in dataset ...\n', ...
        ds.Info.UseOptions.nAngles);
    nSubAngles = input('How many angles to you wish to plot: ');
    % nSubAngles = 120;
    % indices for data to plot, get sample distance for even distance
    sampleDistance = length(downsample(ds.Data.angles, nSubAngles));
    % get subset of angles
    subAngles = downsample(ds.Data.angles, sampleDistance);
    nSubAngles = length(subAngles); % just ensure
    % get indices for subset data
    indices = find(ismember(ds.Data.angles, subAngles));
catch ME
    rethrow(ME)
end

% figure save path for different formats %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fPath = fullfile(PathVariables.saveFiguresPath);
fSvgPath = fullfile(PathVariables.saveImagesPath, 'svg');
fEpsPath = fullfile(PathVariables.saveImagesPath, 'eps');
fPdfPath = fullfile(PathVariables.saveImagesPath, 'pdf');
```

```
% create dataset figure for a subset or all angle %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fig = figure('Name', 'Sensor Array', ...
    'NumberTitle' , 'off', ...
    'WindowStyle', 'normal', ...
    'MenuBar', 'none', ...
    'ToolBar', 'none', ...
    'Units', 'centimeters', ...
    'OuterPosition', [0 0 37 29], ...
    'PaperType', 'a4', ...
    'PaperUnits', 'centimeters', ...
    'PaperOrientation', 'landscape', ...
    'PaperPositionMode', 'auto', ...
    'DoubleBuffer', 'on', ...
    'RendererMode', 'manual', ...
    'Renderer', 'painters');

tdl = tiledlayout(fig, 2, 1, ...
    'Padding', 'compact', ...
    'TileSpacing' , 'compact');


title(tdl, 'Sensor Array Simulation', ...
    'FontWeight', 'normal', ...
    'FontSize', 18, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

subline1 = "Sensor Array (%s) of $%d\\times%d$ sensors, " + ...
    "an edge length of $%.1f$ mm, a rel. pos. to magnet surface of";
subline2 = " $(%.1f, %.1f, -(%.1f))$ in mm, a magnet tilt" + ...
    " of $%.1f^\\circ$, a sphere radius of $%.1f$ mm, a imprinted";
subline3 = "field strength of $%.1f$ kA/m at $%.1f$ mm from" + ...
    " sphere surface in z-axis, $%d$ rotation angles with a ";
subline4 = "step width of $%.1f^\\circ$ and a resolution of" + ...
    " $%.1f^\\circ$. Visualized is a subset of $%d$ angles in ";
subline5 = "sample distance of $%d$ angles. Based on %s" + ...
    " characterization reference %s.";
sub = [sprintf(subline1, ...
                ds.Info.SensorArrayOptions.geometry, ...
                ds.Info.SensorArrayOptions.dimension, ...
                ds.Info.SensorArrayOptions.dimension, ...
                ds.Info.SensorArrayOptions.edge); ...
       sprintf(subline2, ...
                ds.Info.UseOptions.xPos, ...
                ds.Info.UseOptions.yPos, ...
                ds.Info.UseOptions.zPos, ...
                ds.Info.UseOptions.tilt, ...
                ds.Info.DipoleOptions.sphereRadius); ...
       sprintf(subline3, ...
                ds.Info.DipoleOptions.H0mag, ...
                ds.Info.DipoleOptions.z0, ...
                ds.Info.UseOptions.nAngles); ...
       sprintf(subline4, ...
                ds.Data.angleStep, ...
                ds.Info.UseOptions.angleRes, ...
                nSubAngles)
       sprintf(subline5, ...
                sampleDistance, ...
```

```
                ds.Info.CharData, ...
                ds.Info.UseOptions.BridgeReference)];

subtitle(tdl, sub, ...
    'FontWeight', 'normal', ...
    'FontSize', 14, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

% get subset of needed data to plot, only one load %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
M = ds.Info.SensorArrayOptions.dimension^2;
% N = ds.Info.UseOptions.nAngles;
res = ds.Info.UseOptions.angleRes;
%angles = ds.Data.angles;
anglesIP = 0:res:360-res;

% load V subset and reshape for easier computing statistics
Vcos = squeeze(reshape(ds.Data.Vcos(:,:,indices), 1, M, nSubAngles));
Vsin = squeeze(reshape(ds.Data.Vsin(:,:,indices), 1, M, nSubAngles));

% load offset voltage to subtract from cosinus, sinus voltage
Voff = ds.Info.SensorArrayOptions.Voff;
Vcc = ds.Info.SensorArrayOptions.Vcc;

% compute statistics of Vcos Vsin %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% interpolate with makima makes best results, ensure to kill nans for
% fill otherwise fill strokes, use linstyle none for fill without frame
interpM = 'makima';
VcosMean = mean(Vcos, 1);
VcosMeanIP = interp1(subAngles, VcosMean, anglesIP, interpM);

VcosStd = std(Vcos, 1, 1);
VcosVar = var(Vcos, 1, 1); % std^2

% meanvariation coefficient in percent
VcosMVCP = mean(VcosStd ./ VcosMean) * 100;

VcosUpper1 = VcosMean + VcosStd;
VcosUpper2 = VcosMean + VcosVar;
VcosLower1 = VcosMean - VcosStd;
VcosLower2 = VcosMean - VcosVar;

VcosUpper1IP = interp1(subAngles, VcosUpper1, anglesIP, interpM);
VcosUpper1IP = fillmissing(VcosUpper1IP, 'previous');

VcosLower1IP = interp1(subAngles, VcosLower1, anglesIP, interpM);
VcosLower1IP = fillmissing(VcosLower1IP, 'previous');

VcosUpper2IP = interp1(subAngles, VcosUpper2, anglesIP, interpM);
VcosUpper2IP = fillmissing(VcosUpper2IP, 'previous');

VcosLower2IP = interp1(subAngles, VcosLower2, anglesIP, interpM);
VcosLower2IP = fillmissing(VcosLower2IP, 'previous');

VsinMean = mean(Vsin, 1);
VsinMeanIP = interp1(subAngles, VsinMean, anglesIP, interpM);

VsinStd = std(Vsin, 1, 1);
```

```
    VsinVar = var(Vsin, 1, 1); % std^2

    % meanvariation coefficient in percent
    VsinMVCP = mean(VsinStd ./ VsinMean) * 100;

    VsinUpper1 = VsinMean + VsinStd;
    VsinUpper2 = VsinMean + VsinVar;
    VsinLower1 = VsinMean - VsinStd;
    VsinLower2 = VsinMean - VsinVar;

    VsinUpper1IP = interp1(subAngles, VsinUpper1, anglesIP, interpM);
    VsinUpper1IP = fillmissing(VsinUpper1IP, 'previous');

    VsinLower1IP = interp1(subAngles, VsinLower1, anglesIP, interpM);
    VsinLower1IP = fillmissing(VsinLower1IP, 'previous');

    VsinUpper2IP = interp1(subAngles, VsinUpper2, anglesIP, interpM);
    VsinUpper2IP = fillmissing(VsinUpper2IP, 'previous');

    VsinLower2IP = interp1(subAngles, VsinLower2, anglesIP, interpM);
    VsinLower2IP = fillmissing(VsinLower2IP, 'previous');

    % plot Vcos Vsin over angles %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % Vcos
    nexttile;
    hold on;

    fillStdX = [anglesIP, fliplr(anglesIP)];
    fillStdY = [VcosLower1IP, fliplr(VcosUpper1IP)];
    fill(fillStdX, fillStdY, [0.95 0.95 0.95], 'LineStyle', 'none');

    fillVarX = [anglesIP, fliplr(anglesIP)];
    fillVarY = [VcosLower2IP, fliplr(VcosUpper2IP)];
    fill(fillVarX, fillVarY, [0.7 0.7 0.7], 'LineStyle', 'none');

    yline(Voff, 'k--');
    scatter(subAngles, VcosUpper1, [], 'r*');
    plot(anglesIP, VcosUpper1IP, 'r-.');
    scatter(subAngles, VcosMean, [], 'm*');
    plot(anglesIP, VcosMeanIP, 'm-.');
    scatter(subAngles, VcosLower1, [], 'b*');
    plot(anglesIP, VcosLower1IP, 'b-.');


    hold off;
    xlim([-res 360-res]);
    %ylim(ylimits);
    grid on;

    xlabel('$\theta$ in Degree', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$V{cos}(\theta)$ in V', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
```

```
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title(sprintf(...
        "Compare $V_{cos}(\\theta)$ for each Array Member $V_{cc} = %.1f$" + ...
        "V, $V_{off} = %.2f$ V, $\\bar{\\sigma_\\mu} = %.2f$ perc.", ...
        Vcc, Voff, VcosMVCP), ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % Vsin
    nexttile;
    hold on;

    fillStdX = [anglesIP, fliplr(anglesIP)];
    fillStdY = [VsinLower1IP, fliplr(VsinUpper1IP)];
    l1 = fill(fillStdX, fillStdY, [0.95 0.95 0.95], 'LineStyle', 'none');

    fillVarX = [anglesIP, fliplr(anglesIP)];
    fillVarY = [VsinLower2IP, fliplr(VsinUpper2IP)];
    l2 = fill(fillVarX, fillVarY, [0.7 0.7 0.7], 'LineStyle', 'none');

    l3 = yline(Voff, 'k--');
    l4 = scatter(subAngles, VsinUpper1, [], 'r*');
    l5 = plot(anglesIP, VsinUpper1IP, 'r-.');
    l6 = scatter(subAngles, VsinMean, [], 'm*');
    l7 = plot(anglesIP, VsinMeanIP, 'm-.');
    l8 = scatter(subAngles, VsinLower1, [], 'b*');
    l9 = plot(anglesIP, VsinLower1IP, 'b-.');

    hold off;
    xlim([-res 360-res]);
    grid on;

    xlabel('$\theta$ in Degree', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$V{sin}(\theta)$ in V', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');
    title(sprintf(...
        "Compare $V_{sin}(\\theta)$ for each Array Member $V_{cc} = %.1f$" + ...
        " V, $V_{off} = %.2f$ V, $\\bar{\\sigma_\\mu} = %.2f$ perc.", ...
        Vcc, Voff, VsinMVCP), ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % plot legend %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    l = [l1 l2 l3 l4 l5 l6 l7 l8 l9];
    L = legend(l, {'$2\sigma$', ...
```

```
                    '$2\sigma^2$', ...
                    '$V_{off}$', ...
                    '$U_{lim} = \mu + \sigma$', ...
                    sprintf('$%s(U_{lim})$', interpM), ...
                    '$\mu(V)$', ...
                    sprintf('$%s(\\mu)$', interpM), ...
                    '$L_{lim} = \mu - \sigma$', ...
                    sprintf('$%s(L_{lim})$', interpM)}, ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');
    L.Layout.Tile = 'east';

    % save figure to file %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % get file path to save figure with angle index
    [~, fName, ~] = fileparts(ds.Info.filePath);

    % save to various formats
    yesno = input('Save? [y/n]: ', 's');
    if strcmp(yesno, 'y')
        fLabel = input('Enter file label: ', 's');
        fName =  fName + "_StatsPlot_" + fLabel;
        savefig(fig, fullfile(fPath, fName));
        print(fig, fullfile(fSvgPath, fName), '-dsvg');
        print(fig, fullfile(fEpsPath, fName), '-depsc', '-tiff', '-loose');
        print(fig, fullfile(fPdfPath, fName), '-dpdf', '-loose', '-fillpage');
    end
    close(fig);
end
```

Published with MATLAB® R2020b

plotSimulationDatasetCircle

Search for available trainings or test dataset and plot dataset. Follow user input dialog to choose which dataset to plot. Save created plot to file. Filename same as dataset with attached info.

Contents

Syntax

```
plotSimulationDatasetCircle()
```

Description

plotSimulationDatasetCircle() plot training or test dataset which are loacated in data/test or data/training. The function list all datasets and the user must decide during user input dialog which dataset to plot. It loads path from config.mat and scans for file automatically.

Examples

```
plotSimulationDatasetCircle()
```

Input Argurments

None

Output Argurments

None

Requirements

- Other m-files required: None
- Subfunctions: None
- MAT-files required: config.mat

See Also

- generateSimulationDatasets
- sensorArraySimulation
- generateConfigMat

Created on December 02. 2020 by Tobias Wulf. Copyright Tobias Wulf 2020.

```
function plotSimulationDatasetCircle()
```

```
% scan for datasets and load needed configurations %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
try
    disp('Plot simulation dataset ...');
    close all;
    % load path variables
    load('config.mat', 'PathVariables');
    % scan for datasets
    TrainingDatasets = dir(fullfile(PathVariables.trainingDataPath, ...
        'Training_*.mat'));
    TestDatasets = dir(fullfile(PathVariables.testDataPath, 'Test_*.mat'));
    allDatasets = [TrainingDatasets; TestDatasets];
    % check if files available
    if isempty(allDatasets)
        error('No training or test datasets found.');
    end
catch ME
    rethrow(ME)
end

% display availabe datasets to user, decide which to plot %%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% number of datasets
nDatasets = length(allDatasets);
fprintf('Found %d datasets:\n', nDatasets)
for i = 1:nDatasets
    fprintf('%s\t:\t(%d)\n', allDatasets(i).name, i)
end
% get numeric user input to indicate which dataset to plot
iDataset = input('Type number to choose dataset to plot to: ');

% load dataset and ask user which one and how many angles %%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
try
    ds = load(fullfile(allDatasets(iDataset).folder, ...
        allDatasets(iDataset).name));
catch ME
    rethrow(ME)
end

% figure save path for different formats %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fPath = fullfile(PathVariables.saveFiguresPath);
fSvgPath = fullfile(PathVariables.saveImagesPath, 'svg');
fEpsPath = fullfile(PathVariables.saveImagesPath, 'eps');
fPdfPath = fullfile(PathVariables.saveImagesPath, 'pdf');

% create dataset figure for a subset or all angle %%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fig = figure('Name', 'Sensor Array', ...
    'NumberTitle' , 'off', ...
    'WindowStyle', 'normal', ...
    'MenuBar', 'none', ...
    'ToolBar', 'none', ...
    'Units', 'centimeters', ...
    'OuterPosition', [0 0 30 30], ...
    'PaperType', 'a4', ...
    'PaperUnits', 'centimeters', ...
    'PaperOrientation', 'landscape', ...
```

```
        'PaperPositionMode', 'auto', ...
        'DoubleBuffer', 'on', ...
        'RendererMode', 'manual', ...
        'Renderer', 'painters');

    tdl = tiledlayout(fig, 2, 2, ...
        'Padding', 'compact', ...
        'TileSpacing' , 'compact');

    title(tdl, 'Sensor Array Simulation', ...
        'FontWeight', 'normal', ...
        'FontSize', 18, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    subline1 = "Sensor Array (%s) of $%d\\times%d$ sensors," + ...
        " an edge length of $%.1f$ mm, a rel. pos. to magnet surface of";
    subline2 = " $(%.1f, %.1f, -(%.1f))$ in mm, a magnet tilt" + ...
        " of $%.1f^\\circ$, a sphere radius of $%.1f$ mm, a imprinted";
    subline3 = "field strength of $%.1f$ kA/m at $%.1f$ mm from" + ...
        " sphere surface in z-axis, $%d$ rotation angles with a ";
    subline4 = "step width of $%.1f^\\circ$ and a resolution of" + ...
        " $%.1f^\\circ$. Visualized are circular path of each array position  ";
    subline5 = "Based on %s characterization reference %s.";
    sub = [sprintf(subline1, ...
                    ds.Info.SensorArrayOptions.geometry, ...
                    ds.Info.SensorArrayOptions.dimension, ...
                    ds.Info.SensorArrayOptions.dimension, ...
                    ds.Info.SensorArrayOptions.edge); ...
           sprintf(subline2, ...
                    ds.Info.UseOptions.xPos, ...
                    ds.Info.UseOptions.yPos, ...
                    ds.Info.UseOptions.zPos, ...
                    ds.Info.UseOptions.tilt, ...
                    ds.Info.DipoleOptions.sphereRadius); ...
           sprintf(subline3, ...
                    ds.Info.DipoleOptions.H0mag, ...
                    ds.Info.DipoleOptions.z0, ...
                    ds.Info.UseOptions.nAngles); ...
           sprintf(subline4, ...
                    ds.Data.angleStep, ...
                    ds.Info.UseOptions.angleRes)
           sprintf(subline5, ...
                    ds.Info.CharData, ...
                    ds.Info.UseOptions.BridgeReference)];

    subtitle(tdl, sub, ...
        'FontWeight', 'normal', ...
        'FontSize', 14, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % get subset of needed data to plot, only one load %%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    N = ds.Info.SensorArrayOptions.dimension;
    M = ds.Info.UseOptions.nAngles;
    Voff = ds.Info.SensorArrayOptions.Voff;
    Vcos = ds.Data.Vcos - Voff;
    Vsin = ds.Data.Vsin - Voff;
    Hx = ds.Data.Hx;
```

```
Hy = ds.Data.Hy;

% calulate norm values to align circles around position only for x,y
% directition for each sensor dot over all angles.
Vmag = sqrt(Vcos.^2 + Vsin.^2);
Hmag = sqrt(Hx.^2 + Hy.^2);
%Hmag = ds.Data.Habs;

% related to position, multiply scale factor for circle diameter
diameterFactor = 2 * N / ds.Info.SensorArrayOptions.edge;
MaxVmagPos = max(Vmag, [], 3) * diameterFactor;
MaxHmagPos = max(Hmag, [], 3) * diameterFactor;

% Overall maxima, scalar, multiply scale factor for circle diameter
MaxVmagOA =  max(Vmag, [], 'all') * diameterFactor;
MaxHmagOA =  max(Hmag, [], 'all') * diameterFactor;

% norm and scale volatages and filed strengths
VcosNorm = Vcos ./ MaxVmagPos;
VcosScaled = Vcos / MaxVmagOA;
VsinNorm = Vsin ./ MaxVmagPos;
VsinScaled = Vsin / MaxVmagOA;

HxNorm = Hx ./ MaxHmagPos;
HxScaled = Hx / MaxHmagOA;
HyNorm = Hy ./ MaxHmagPos;
HyScaled = Hy / MaxHmagOA;

% sensor array grid
X = ds.Data.X;
Y = ds.Data.Y;
Z = ds.Data.Z;

% calc limits of plot 1
maxX = ds.Info.UseOptions.xPos + 0.7 * ds.Info.SensorArrayOptions.edge;
maxY = ds.Info.UseOptions.yPos + 0.7 * ds.Info.SensorArrayOptions.edge;
minX = ds.Info.UseOptions.xPos - 0.7 * ds.Info.SensorArrayOptions.edge;
minY = ds.Info.UseOptions.yPos - 0.7 * ds.Info.SensorArrayOptions.edge;

% plot sensor grid in x and y coordinates %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% plot each cooredinate in loop to create a special shading constant
% reliable to orientation for all matrice
 % calculate colormap to identify scatter points
c=zeros(N,N,3);
for i = 1:N
    for j = 1:N
        c(i,j,:) = [(2*N+1-2*i), (2*N+1-2*j), (i+j)]/2/N;
    end
end
c = squeeze(reshape(c, N^2, 1, 3));
% reshape RGB for picking single sensors
R = reshape(c(:,1), N, N);
G = reshape(c(:,2), N, N);
B = reshape(c(:,3), N, N);

% Field strength scaled to overall maxima %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
nexttile;
hold on;
```

```
    for i = 1:N
        for j = 1:N
            plot(squeeze(HxScaled(i, j, :)) + X(i,j), ...
                 squeeze(HyScaled(i, j, :)) + Y(i,j), ...
                 'Color', [R(i,j) G(i,j) B(i,j)], ...
                 'LineWidth' , 1.5)
            line([X(i,j), HxScaled(i,j,1) + X(i,j)], ...
                 [Y(i,j), HyScaled(i,j,1)  + Y(i,j)], ...
                'Color','k','LineWidth',1.5)
        end
    end

    % scatter magnet x,y position (0,0,z)
    scatter(0, 0, 32, 'r', 'filled');

    hold off;

    % axis shape and ticks
    axis square xy;
    axis tight;
    grid on;
    xlim([minX maxX]);
    ylim([minY maxY]);

    xlabel('$X$ in mm', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$Y$ in mm', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('$H_x$, $H_y$ Normed to Max overall Positions', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % Cosinus, sinus voltage scaled to overall maxima %%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    nexttile;
    hold on;
    for i = 1:N
        for j = 1:N
            plot(squeeze(VcosScaled(i, j, :)) + X(i,j), ...
                 squeeze(VsinScaled(i, j, :)) + Y(i,j), ...
                 'Color', [R(i,j) G(i,j) B(i,j)], ...
                 'LineWidth' , 1.5)
            line([X(i,j), VcosScaled(i,j,1) + X(i,j)], ...
                 [Y(i,j), VsinScaled(i,j,1)  + Y(i,j)], ...
                'Color','k','LineWidth',1.5)
        end
    end

    % scatter magnet x,y position (0,0,z)
    scatter(0, 0, 32, 'r', 'filled');
```

```
hold off;

% axis shape and ticks
axis square xy;
axis tight;
grid on;
xlim([minX maxX]);
ylim([minY maxY]);

xlabel('$X$ in mm', ...
    'FontWeight', 'normal', ...
    'FontSize', 12, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

ylabel('$Y$ in mm', ...
    'FontWeight', 'normal', ...
    'FontSize', 12, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

title('$V_{cos}$, $V_{sin}$ Normed to Max overall Positions', ...
    'FontWeight', 'normal', ...
    'FontSize', 12, ...
    'FontName', 'Times', ...
    'Interpreter', 'latex');

% Field strength normed each maxima at position %%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
nexttile;
hold on;
for i = 1:N
    for j = 1:N
        plot(squeeze(HxNorm(i, j, :)) + X(i,j), ...
             squeeze(HyNorm(i, j, :)) + Y(i,j), ...
             'Color', [R(i,j) G(i,j) B(i,j)], ...
             'LineWidth' , 1.5)
        line([X(i,j), HxNorm(i,j,1) + X(i,j)], ...
             [Y(i,j), HyNorm(i,j,1)  + Y(i,j)], ...
            'Color','k','LineWidth',1.5)
    end
end

% scatter magnet x,y position (0,0,z)
scatter(0, 0, 32, 'r', 'filled');

hold off;

% axis shape and ticks
axis square xy;
axis tight;
grid on;
xlim([minX maxX]);
ylim([minY maxY]);

xlabel('$X$ in mm', ...
    'FontWeight', 'normal', ...
    'FontSize', 12, ...
    'FontName', 'Times', ...
```

```
        'Interpreter', 'latex');

    ylabel('$Y$ in mm', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('$H_x$, $H_y$ Normed to Max at each Position', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    % Cosinus, sinus voltage normed to each maxima at position %%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    nexttile;
    hold on;
    for i = 1:N
        for j = 1:N
            plot(squeeze(VcosNorm(i, j, :)) + X(i,j), ...
                 squeeze(VsinNorm(i, j, :)) + Y(i,j), ...
                 'Color', [R(i,j) G(i,j) B(i,j)], ...
                 'LineWidth' , 1.5)
            line([X(i,j), VcosNorm(i,j,1) + X(i,j)], ...
                 [Y(i,j), VsinNorm(i,j,1)  + Y(i,j)], ...
                'Color','k','LineWidth',1.5)
        end
    end

    % scatter magnet x,y position (0,0,z)
    scatter(0, 0, 32, 'r', 'filled');

    hold off;

    % axis shape and ticks
    axis square xy;
    axis tight;
    grid on;
    xlim([minX maxX]);
    ylim([minY maxY]);

    xlabel('$X$ in mm', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    ylabel('$Y$ in mm', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');

    title('$V_{cos}$, $V_{sin}$ Normed to Max at each Positions', ...
        'FontWeight', 'normal', ...
        'FontSize', 12, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');
```

```
    % save figure to file %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % get file path to save figure with angle index
    [~, fName, ~] = fileparts(ds.Info.filePath);

    % save to various formats
    yesno = input('Save? [y/n]: ', 's');
    if strcmp(yesno, 'y')
        fLabel = input('Enter file label: ', 's');
        fName = fName + "_CirclePlot_" + fLabel;
        savefig(fig, fullfile(fPath, fName));
        print(fig, fullfile(fSvgPath, fName), '-dsvg');
        print(fig, fullfile(fEpsPath, fName), '-depsc', '-tiff', '-loose');
        print(fig, fullfile(fPdfPath, fName), '-dpdf', '-loose', '-fillpage');
    end
    close(fig);
end
```

Published with MATLAB® R2020b

Datasets

Datasets are an appreciated way to save and reach done work and reuse it in progress. The easiest way to build and to use propper datasets in matlab are mat-files. They are easy to load and can be build by an script or function it just needs to save the variables from workspace. So latery save datasets can be used for futher calculations or to load certain configuration in to workspace and to solve task in a unified way.

Contents

TDK TAS2141 Characterization

The characterization dataset of the TDK TMR angular sensor as base dataset for sensor array dipol simulation. The dataset contains information about the stimulus wich was used for characterization, the magnetic resolustion or the sensor bridge outputs for Hx and Hy fields and bridge outputs corresponding to stimulus amplitudes in Hx and Hy direction.

NXP KMZ60 Characterization

The characterization dataset of the NXP AMR angular sensor is second characterization dataset which was aquirred in the same way as the TDK dataset. The dataset is integrated in the simulation software after finish for TDK and comes along with option choose between both dataset. Bridge gain is introduced to handle internal amplification of bridge outputs.

Config Mat

Configuration dataset to control the main program from centralized config file. Includes any kind of configuration and parameters to load in function or script workspaces.

Training and Test Datasets

Sensor array simulation datasets for training and test purpose for angle prediction via gaussian processes.

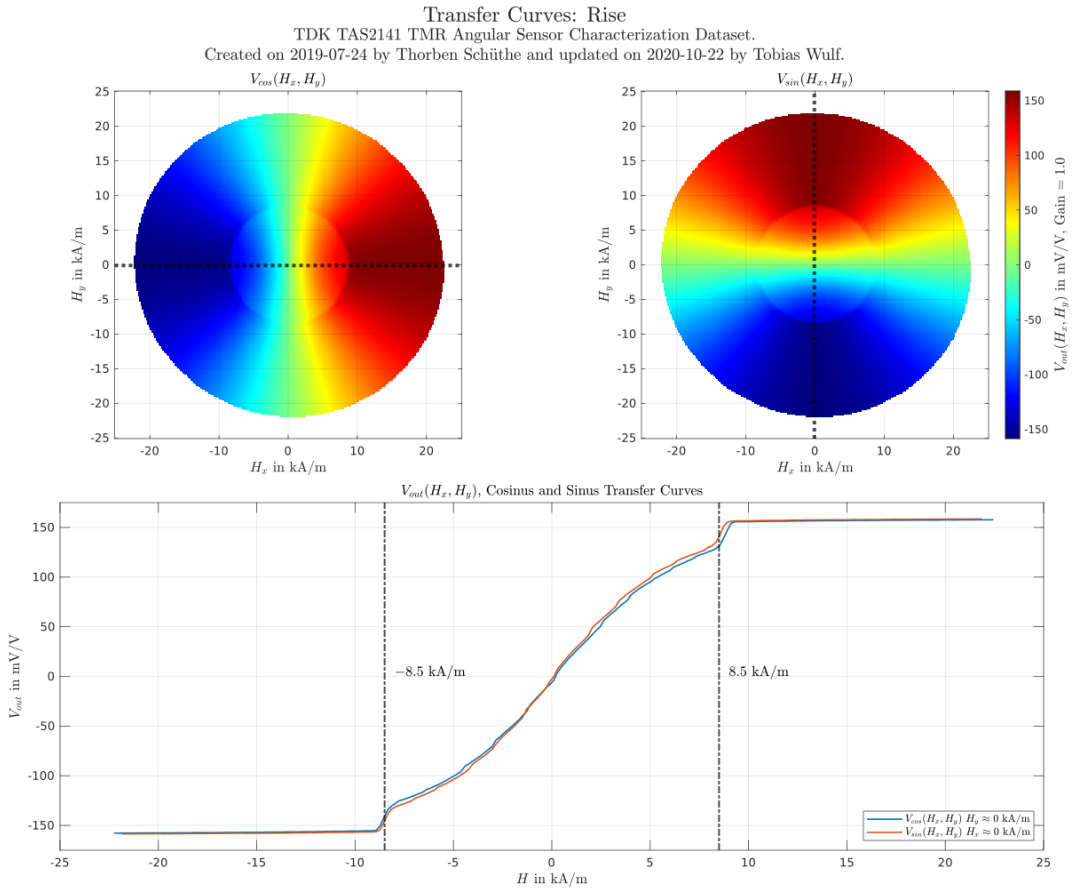
Created on October 27. 2020 Tobias Wulf.

Published with MATLAB® R2020b

TDK TAS2141 Characterization

TDK characterization as base of the sensor array simulation was done before the dataset is just modified in its structure and not in its values. An additional info struct is added which contains information about how the dataset was aquirred and a data struct which contains the magnetic field resolution and the cosinus and sinus bridge images for variable Hx and Hy fieldstrengths. The raw dataset was aquirred after the method Thorben Schüthe describe in his IEEE paper for two-dimensional characterization of TMR angular sensors. The sensor characterized for both bridges a cosinus and sinus bridge. The bridges have a phisically phase shift of 90° so the sensor as able to reference a superimposed magnetic field flud in x- and y-direction. The field was generated by an cross coil setup.

The resulting TMR characterization field abstracts a full rotation for cosinus and sinus output voltages by representing one maximum and minimun in the characterization fields. So circular path on the charctiration fields generates one sinoid output related on current angle position of stimulus magnetic field.



Contents

See Also

- Two-Dimensional Characterization (Schüthe)

Magnetic Stimulus

The right stimulus is the keynote for characterization records. It needs to have the abillity record slow enough for quasi static recordings but is not allowed to be real static so the magnetic field is not interrupted during the recording. Therefore slow sinoid carrier functions with even slower amplitude modulation is choosen to provide a quasi static stimulus.

The carrier function for the Hx-field stimulus is related to the cosinus bridge and so:

$$c_1(t) = cos(\phi(t))$$

Due to the physically phase shift the Hy-field stimulus is related to sinus:

$$c_2(t) = sin(\phi(t))$$

Both carrier runs with same carrier frequency:

$$f_c = 3.2Hz$$

so they are executed with the phase vector over time:

$$\phi(t) = 2\pi f_c t$$

The carrier functions are triangle modulated to generate rising and falling amplituded. The modulation frequency is set to:

$$f_m = 0.01Hz$$

Which generates a stimulus with 320 periods where 160 periods feeds a rising and falling record each multiplied with maximum fieldstrength amplitude:

$$m(t) = H_{max} \cdot tri(t) = H_{max} \cdot tri(2(t - t_0)f_m)$$

$$t_0 = \frac{1}{2f_m}$$

So the Hx- and Hy-field stimulus is described by:

$$H_x(t) = m(t) \cdot c_1(t)$$

$$H_y(t) = m(t) \cdot c_1(t)$$

The stimulus amplitude depending on the phase in polar coordinates can be displayed for both parts by:

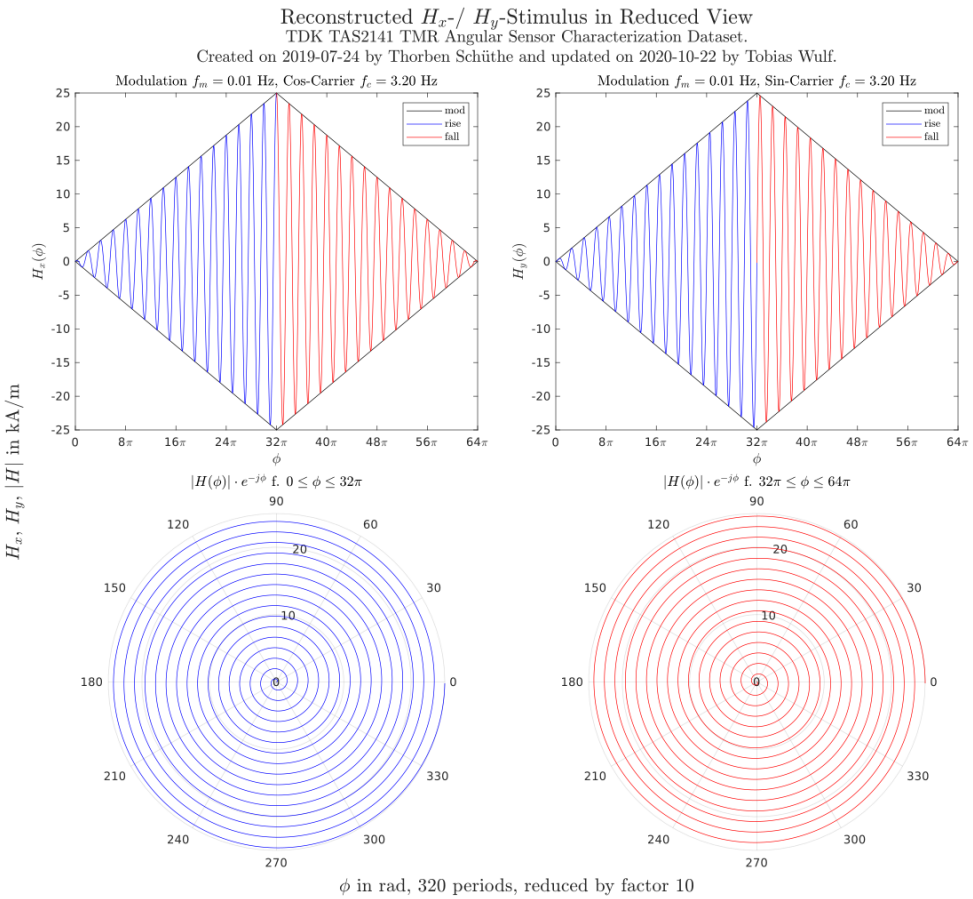
$$H_{x,y}(\phi) = |H_{x,y}(\phi)| \cdot e^{j\phi} = m(t) \cdot e^{j\phi(t)}$$

Where a rising spiral runs from center outwards for:

$$0 < t < t_0$$

And a falling spiral of amplitudes from outwards to center for:

$$t_0 < t < \frac{1}{f_m}$$

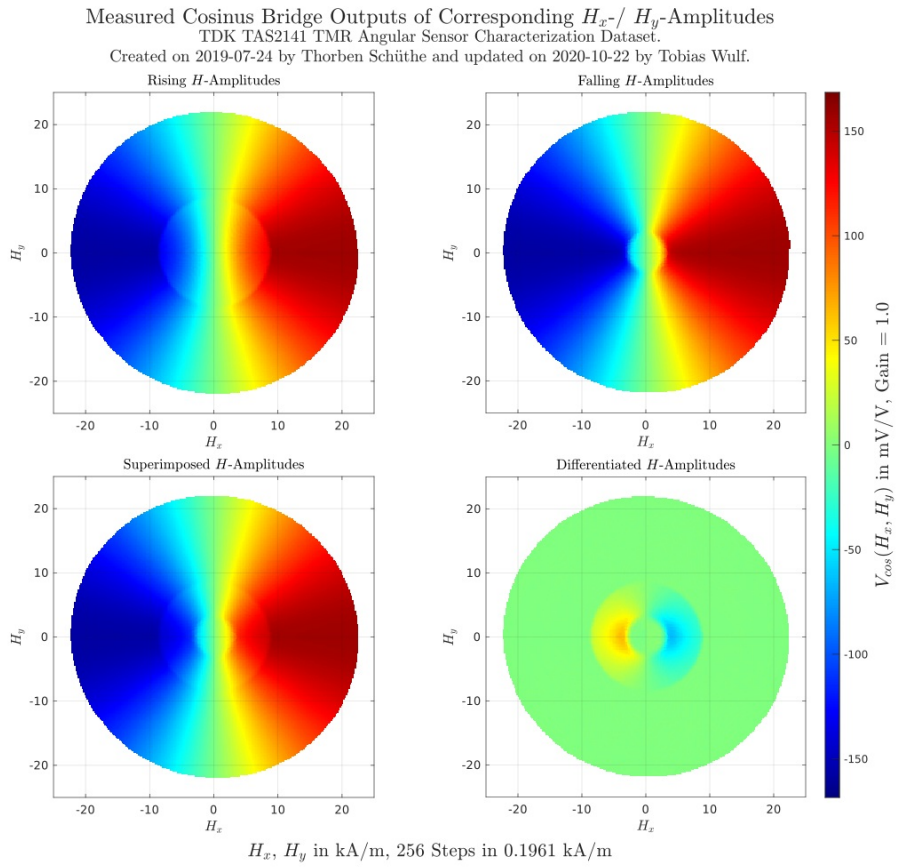


Cosinuns Bridge Output

The record characterization raw data are one dimensional time discrete vecotrs. To fieldstrength images like down below the recorded data must be referenced backwords to driven stimulus of Hx- and Hy-direction. But at first the image size of must be determined. Here fix size is set to 256px for each direction. So it spans a vector for Hx- and Hy-direction from minimum -25kA/m to maximum 25kA/m in 256 steps with a resolution of 0.1961kA/m. So it results into a 256x256 image. Now it runs for each point on the Hx- and Hy-axes and get the record index of the stimulus as backreference to the recorded bridge signal and sets the pixel. That runs for the rising modulation amplitude and falling amplitude until every pixel is hit and ended up into a dimensional function image as:

$$V_{cos}(H_x, H_y) = [mV/V]$$

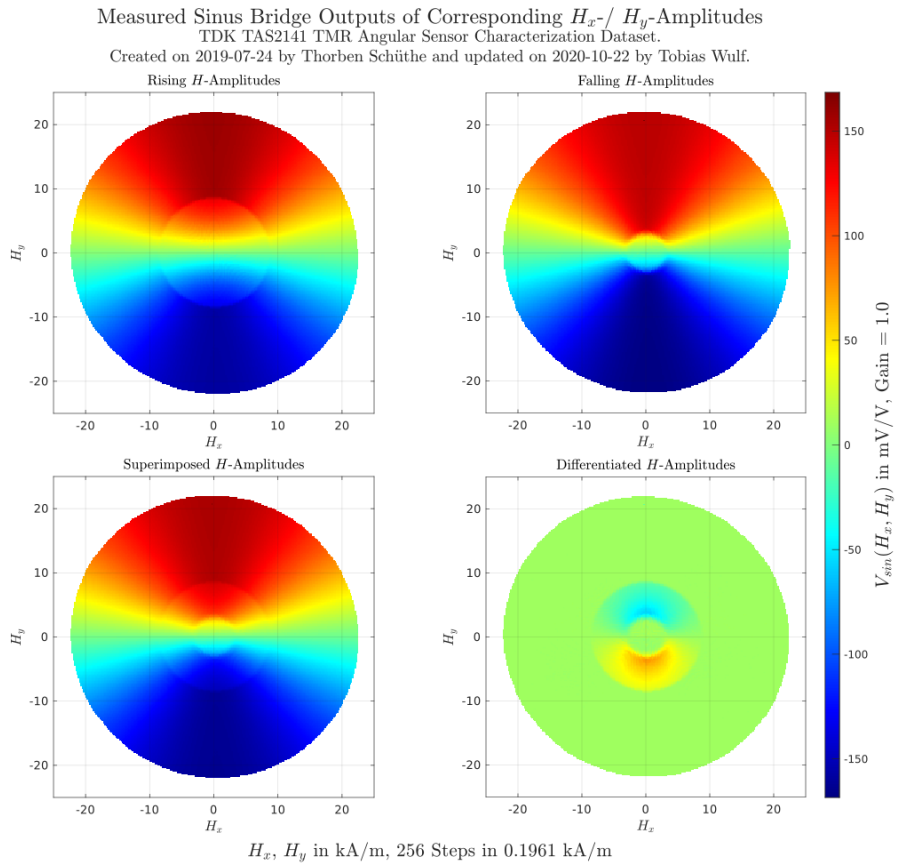
The information of the image is build up in row. Reference Hx for constant Hy in each row. The method is also comparable to a historgram of Hx matches in the recorded sensor signal for one constant Hy and so on next historgram append on the next row for the next Hy.



Sinus Bridge Output

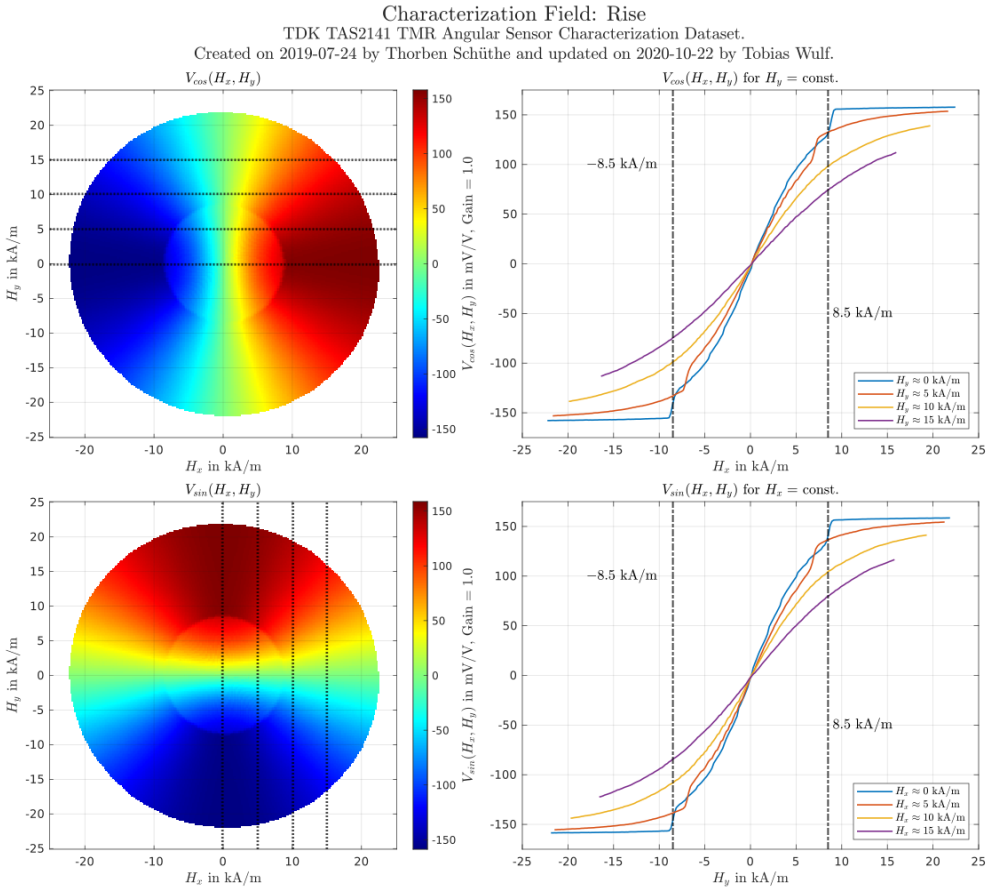
The sinus characterization field is build up similar to the cosinus images but the information lays now in the columns so the data is collected in each column for a constant Hx and variable Hy:

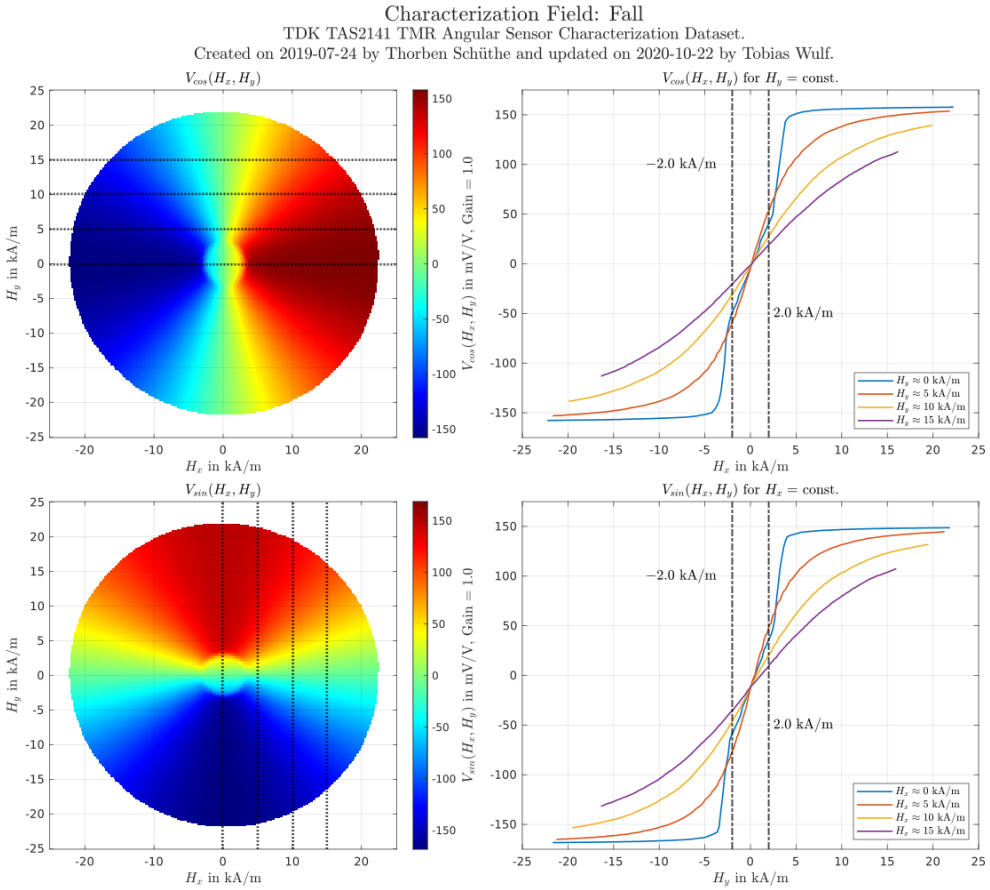
$V_{sin}(H_x, H_y) = [mV/V]$



Operating Point

To determine an operating point in sensor array simulation the characterization fields needs some further investigation in static Hy and variable Hx field strength for cosinus bridge and vice versa for sinus bridge references. The best results supports the "Rise" field because it has a wide linear plateau between -8.5 kA/m and 8.5 kA/m. So Rise characterization field is used in sesnor array simulation. It is not needed to drive the sensor in saturation.


Characterization Field: Rise
TDK TAS2141 TMR Angular Sensor Characterization Dataset.
Created on 2019-07-24 by Thorben Schüthe and updated on 2020-10-22 by Tobias Wulf.
$V_{cos}(H_x, H_y)$
$V_{cos}(H_x, H_y)$ for H_y = const.
$V_{sin}(H_x, H_y)$
$V_{sin}(H_x, H_y)$ for H_x = const.
H_x in kA/m
H_y in kA/m
$V_{cos}(H_x, H_y)$ in mV/V, Gain = 1.0
$V_{sin}(H_x, H_y)$ in mV/V, Gain = 1.0
−8.5 kA/m
8.5 kA/m
$H_y \approx 0$ kA/m
$H_y \approx 5$ kA/m
$H_y \approx 10$ kA/m
$H_y \approx 15$ kA/m
$H_x \approx 0$ kA/m
$H_x \approx 5$ kA/m
$H_x \approx 10$ kA/m
$H_x \approx 15$ kA/m



Dataset Structure

Info:

The dataset is parted in two main structs. The first one is filled with meta data. So it represents the file header. The struct is called Info and contains information about how the dataset is aquirred. So the stimulus is reconstructable from that meta data.

- Created - string, contains dataset creation date
- Creator - string, contains dataset creator
- Edited - string, contains last time edited date
- Editor - string, contains last time editor
- Senor - string, sensor identification name e.g. TAS2141
- SensorType - string, kind of sensor e.g. Angular
- SensorTechnology - string, bridge technology e.g. AMR, GMR, TMR
- SensorManufacturer - string, producer or supplier e.g. NXP, TDK
- **MagneticField** - struct, contains further information about Hx and Hy
- **SensorOutput** - struct, contains information about sensor produced output and gathered image information
- **Units** - struct, contains information about used si units in dataset

- **MagneticField:**

- Modulation - string, contains modulation equivalent Matlab function

- ModulationFrequency - double, contains frequeny of modulation in Hz
- CarrierFrequency - double, carrier frequency for both Hx and Hy carrier in Hz
- MaxAmplitude - double, maximum Hx and Hy field amplitude in kA/m
- MinAmplitude - double, minimum Hx and Hy field amplitude in kA/m
- Steps - double, Hx- and Hy-field steps to build characterization images
- Resolution - double, resolution of one step in kA/m
- CarrierHx - string, contains Hx carrier equivalent Matlab function
- CarrierHy - string, contains Hy carrier equivalent Matlab function

- **SensorOutput:**

- **CosinusBridge** - struct, contains further information about sensor cosinus bridge outpus
- **SinusBridge** - struct, contains further information about sensor sinus bridge outputs
- BridgeGain - double, scalar factor of bridge gain for output voltage

- **CosinusBridge/ SinusBridge:**

- xDimension - double, image size in x-direction
- yDimension - double, image size in y-direction
- xDirection - string, x-axis label
- yDirection - string, y-axis label
- Orientation - string, orientation of varying data, row or column
- Determination - cell, images in data {"Rise", "Fall", "All", "Diff"}

- **Units:**

- MagneticFieldStrength - string, kA/m
- Frequency - string, Hz
- SensorOutputVoltage - string, mV/V

Data:

The second struct contains the preporcessed characteriazation data of the TDK TAS2141 TMR angular Sensor. It is divided into two main structs one for the magnetic field reference points of the characterization images and one for the characteriazation sensor output images.

- **MagneticField** - struct, contain Hx- and Hy-field vectors which are the resolution references to each pixel in the characterization images of the sensors preprocessed bridge outputs
- **SensorOutput** - struct, contains structs for cosinus and sinus bridge outputs preprocessed in images of size of 256x256 pixels where each pixels references a bridge output in mV to a certain Hx- and Hy-fieldstrength amplitdue

- **MagneticField:**

- hx - array, Hx field axis of characteriazation images column vector of 1x256 double values from -25 kA/m to 25 kA/m with a resolution of 0.1961 kA/m
- hy - array, Hy field axis of characteriazation images column vector of 1x256 double values from -25 kA/m to 25 kA/m with a resolution of 0.1961 kA/m

- **SensorOutput:**

- **CosinusBridge** - struct, contains preprocessed characterization results of the sensors cosinus bridge outputs
- **SinusBridge** - struct, contains preprocessed characterization results of the sensors sinus bridge outputs

- **CosinusBridge:**

- Rise - array, double array of size 256x256 which references the cosinus bridge outputs for rising modulated stimulus amplitude to each cross reference of vectors MagneticField.hx and MagneticField.hy

- Fall - array, double array of size 256x256 which references the cosinus bridge outputs for falling modulated stimulus amplitude to each cross reference of vectors MagneticField.hx and MagneticField.hy
- All - array, double array of size 256x256 superimposed image of Rise and Fall
- Diff - array, double array of size 256x256 differentiated image of Rise and Fall

- **SinusBridge:**

- Rise - array, double array of size 256x256 which references the sinus bridge outputs for rising modulated stimulus amplitude to each cross reference of vectors MagneticField.hx and MagneticField.hy
- Fall - array, double array of size 256x256 which references the sinus bridge outputs for falling modulated stimulus amplitude to each cross reference of vectors MagneticField.hx and MagneticField.hy
- All - array, double array of size 256x256 superimposed image of Rise and Fall
- Diff - array, double array of size 256x256 differentiated image of Rise and Fall

The edited raw dataset provided from Thorben Schüthe is save with Matlabs build-in save function in a certain way to perform partial loads from the dataset.

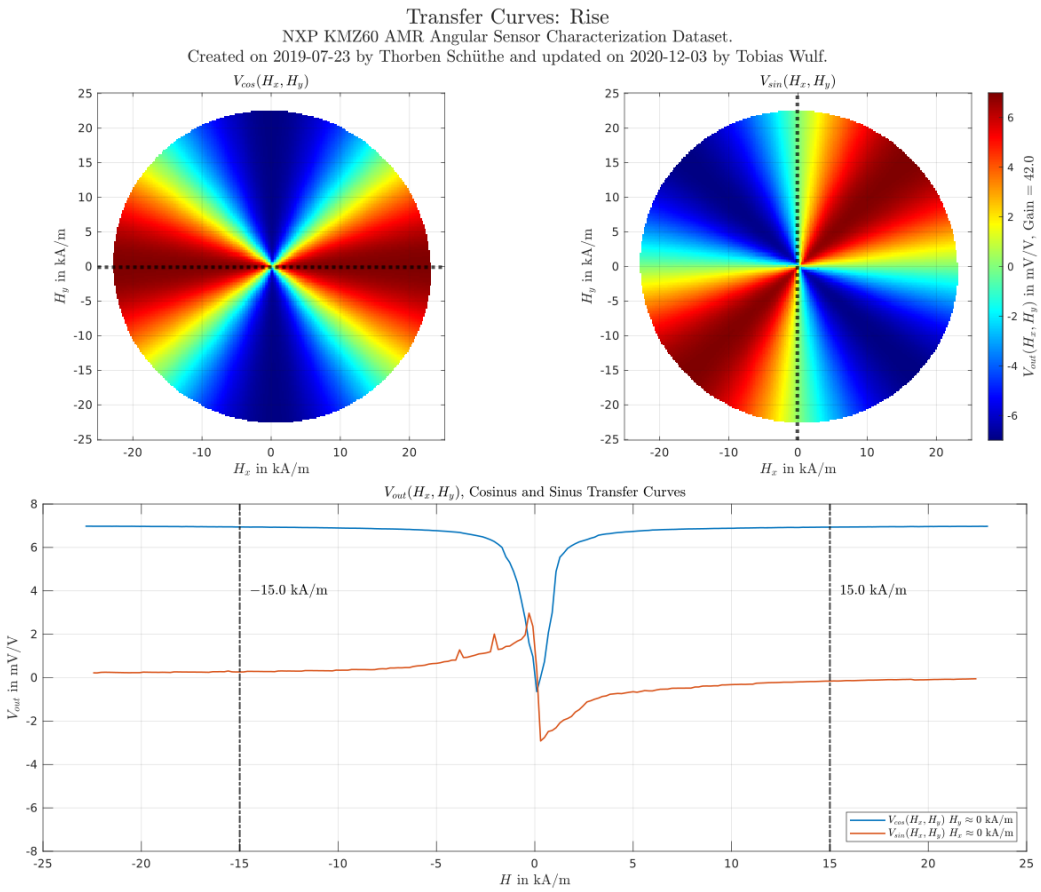
```
save('data/TDK_TAS2141_Characterization_2020-10-22_18-12-16-827.mat', ...
     'Info', 'Data', '-v7.3', '-nocompression')
```

Published with MATLAB® R2020b

NXP KMZ60 Characterization

NXP KMZ60 characterization as second base of the sensor array simulation was although done before and was manually modified to same structure as TDK dataset. With additional values for bridge gain. TDK dataset is adjusted in the same way now. The raw dataset was aquirred after the method Thorben Schüthe describe in his IEEE paper for two-dimensional characterization of TMR angular sensors. The sensor characterized for both bridges a cosinus and sinus bridge. The bridges have a phisically phase shift of 90° so the sensor as able to reference a superimposed magnetic field flud in x- and y-direction. The field was generated by an cross coil setup.

The resulting AMR characterization field abstracts a full rotation for cosinus and sinus output voltages by representing two maximum and minimun areas in the characterization fields. So circular path on the charctiration fields generates two sinoid periods related on current angle position between 0° and 180° or 180° and 360°.



Contents

See Also

- Two-Dimensional Characterization (Schüthe)

Magnetic Stimulus

The right stimulus is the keynote for characterization records. It needs to have the abillity record slow enough for quasi static recordings but is not allowed to be real static so the magnetic field is not interrupted during the recording. Therefore slow sinoid carrier functions with even slower amplitude modulation is choosen to provide a quasi static stimulus.

The carrier function for the Hx-field stimulus is related to the cosinus bridge and so:

$$c_1(t) = cos(\phi(t))$$

Due to the physically phase shift the Hy-field stimulus is related to sinus:

$$c_2(t) = sin(\phi(t))$$

Both carrier runs with same carrier frequency:

$$f_c = 3.2Hz$$

so they are executed with the phase vector over time:

$$\phi(t) = 2\pi f_c t$$

The carrier functions are triangle modulated to generate rising and falling amplituded. The modulation frequency is set to:

$$f_m = 0.01Hz$$

Which generates a stimulus with 320 periods where 160 periods feeds a rising and falling record each multiplied with maximum fieldstrength amplitude:

$$m(t) = H_{max} \cdot tri(t) = H_{max} \cdot tri(2(t - t_0)f_m)$$

$$t_0 = \frac{1}{2f_m}$$

So the Hx- and Hy-field stimulus is described by:

$$H_x(t) = m(t) \cdot c_1(t)$$

$$H_y(t) = m(t) \cdot c_1(t)$$

The stimulus amplitude depending on the phase in polar coordinates can be displayed for both parts by:

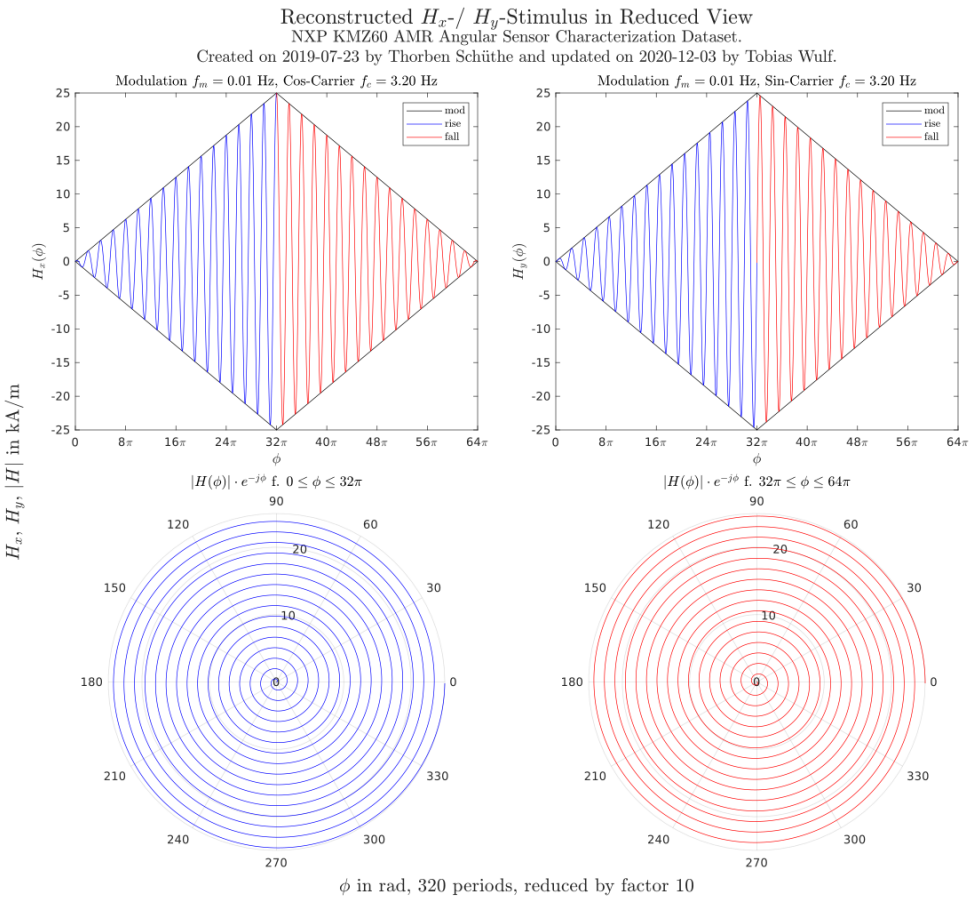
$$H_{x,y}(\phi) = |H_{x,y}(\phi)| \cdot e^{j\phi} = m(t) \cdot e^{j\phi(t)}$$

Where a rising spiral runs from center outwards for:

$$0 < t < t_0$$

And a falling spiral of amplitudes from outwards to center for:

$$t_0 < t < \frac{1}{f_m}$$

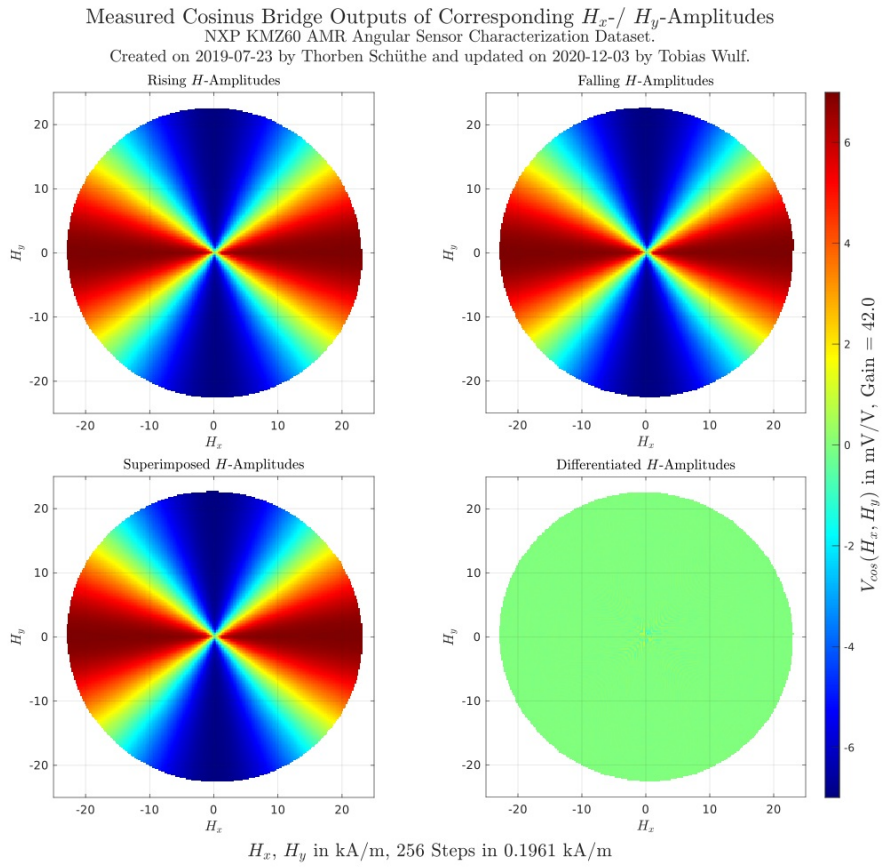


Cosinuns Bridge Output

The record characterization raw data are one dimensional time discrete vecotrs. To fieldstrength images like down below the recorded data must be referenced backwords to driven stimulus of Hx- and Hy-direction. But at first the image size of must be determined. Here fix size is set to 256px for each direction. So it spans a vector for Hx- and Hy-direction from minimum -25kA/m to maximum 25kA/m in 256 steps with a resolution of 0.1961kA/m. So it results into a 256x256 image. Now it runs for each point on the Hx- and Hy-axes and get the record index of the stimulus as backreference to the recorded bridge signal and sets the pixel. That runs for the rising modulation amplitude and falling amplitude until every pixel is hit and ended up into a dimensional function image as:

$$V_{cos}(H_x, H_y) = [mV/V]$$

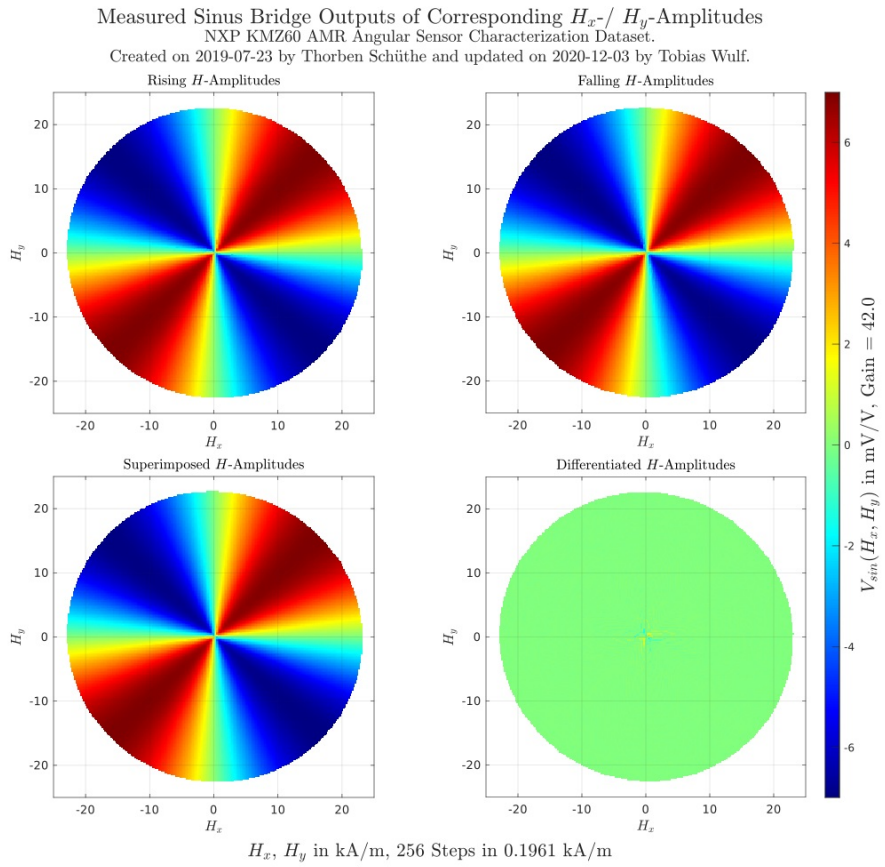
The information of the image is build up in row. Reference Hx for constant Hy in each row. The method is also comparable to a historgram of Hx matches in the recorded sensor signal for one constant Hy and so on next historgram append on the next row for the next Hy.



Sinus Bridge Output

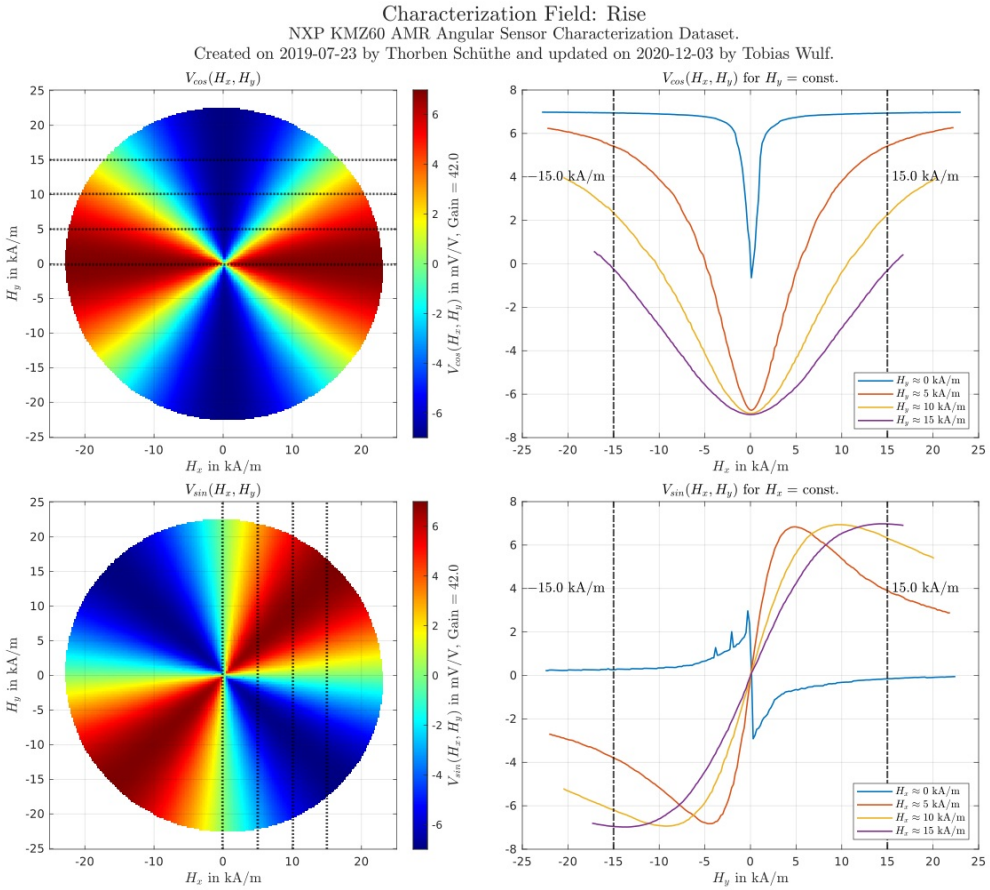
The sinus characterization field is build up similar to the cosinus images but the information lays now in the columns so the data is collected in each column for a constant Hx and variable Hy:

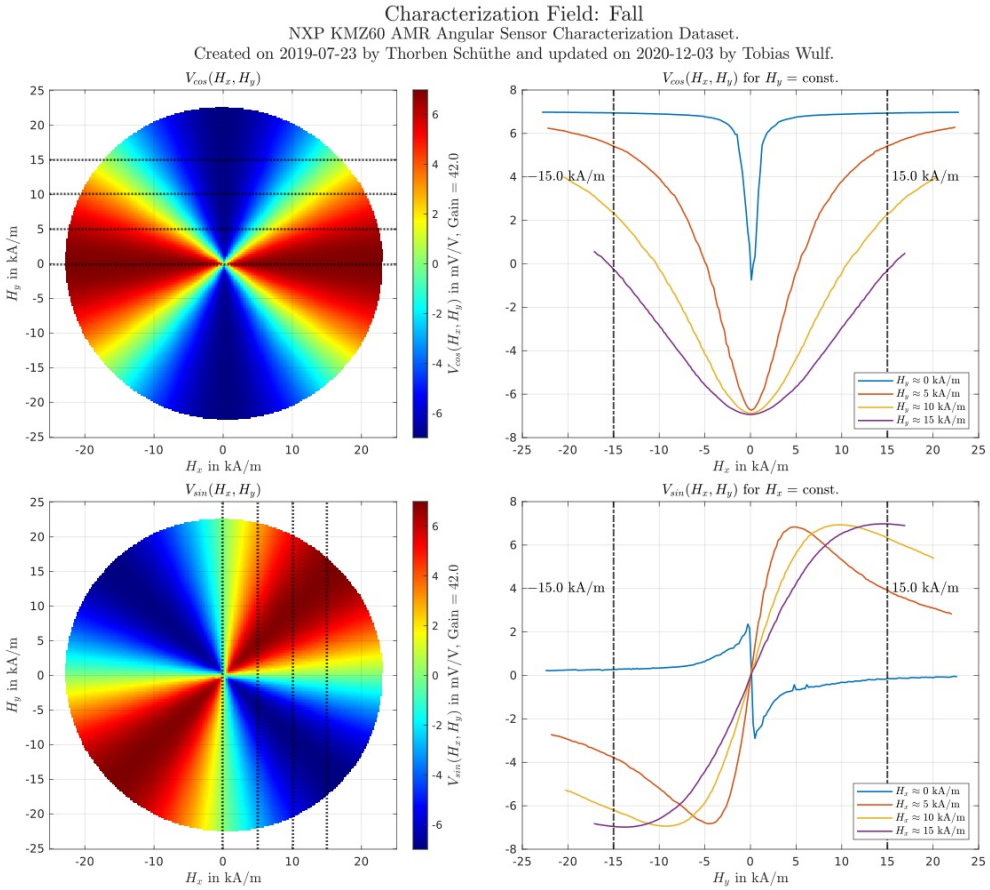
$V_{sin}(H_x, H_y) = [mV/V]$



Operating Point

To determine an operating point in sensor array simulation the characterization fields needs some further investigation in static Hy and variable Hx field strength for cosinus bridge and vice versa for sinus bridge references. In compare to the TDK TMR the NXP AMR sensor has clear linear plateu. It has a continous non-linear areas divided in two maximum and minum areas. The best results for bridge outputs is supported by an operating point in saturation of the characterization fields so circular path on the fields should be described at 20 kA/m to 25 kA/m path radius.


Characterization Field: Rise
NXP KMZ60 AMR Angular Sensor Characterization Dataset.
Created on 2019-07-23 by Thorben Schüthe and updated on 2020-12-03 by Tobias Wulf.
$V_{cos}(H_x, H_y)$
$V_{cos}(H_x, H_y)$ for H_y = const.
$V_{sin}(H_x, H_y)$
$V_{sin}(H_x, H_y)$ for H_x = const.
H_x in kA/m
H_y in kA/m
$V_{cos}(H_x, H_y)$ in mV/V, Gain = 42.0
$V_{sin}(H_x, H_y)$ in mV/V, Gain = 42.0
-15.0 kA/m
15.0 kA/m
$H_y \approx 0$ kA/m
$H_y \approx 5$ kA/m
$H_y \approx 10$ kA/m
$H_y \approx 15$ kA/m
$H_x \approx 0$ kA/m
$H_x \approx 5$ kA/m
$H_x \approx 10$ kA/m
$H_x \approx 15$ kA/m



Dataset Structure

Info:

The dataset is parted in two main structs. The first one is filled with meta data. So it represents the file header. The struct is called Info and contains information about how the dataset is aquirred. So the stimulus is reconstructable from that meta data.

- Created - string, contains dataset creation date
- Creator - string, contains dataset creator
- Edited - string, contains last time edited date
- Editor - string, contains last time editor
- Senor - string, sensor identification name e.g. TAS2141
- SensorType - string, kind of sensor e.g. Angular
- SensorTechnology - string, bridge technology e.g. AMR, GMR, TMR
- SensorManufacturer - string, producer or supplier e.g. NXP, TDK
- **MagneticField** - struct, contains further information about Hx and Hy
- **SensorOutput** - struct, contains information about sensor produced output and gathered image information
- **Units** - struct, contains information about used si units in dataset

- **MagneticField:**

- Modulation - string, contains modulation equivalent Matlab function

- ModulationFrequency - double, contains frequeny of modulation in Hz
- CarrierFrequency - double, carrier frequency for both Hx and Hy carrier in Hz
- MaxAmplitude - double, maximum Hx and Hy field amplitude in kA/m
- MinAmplitude - double, minimum Hx and Hy field amplitude in kA/m
- Steps - double, Hx- and Hy-field steps to build characterization images
- Resolution - double, resolution of one step in kA/m
- CarrierHx - string, contains Hx carrier equivalent Matlab function
- CarrierHy - string, contains Hy carrier equivalent Matlab function

- **SensorOutput:**

- **CosinusBridge** - struct, contains further information about sensor cosinus bridge outpus
- **SinusBridge** - struct, contains further information about sensor sinus bridge outputs
- BridgeGain - double, scalar factor of bridge gain for output voltage

- **CosinusBridge/ SinusBridge:**

- xDimension - double, image size in x-direction
- yDimension - double, image size in y-direction
- xDirection - string, x-axis label
- yDirection - string, y-axis label
- Orientation - string, orientation of varying data, row or column
- Determination - cell, images in data {"Rise", "Fall", "All", "Diff"}

- **Units:**

- MagneticFieldStrength - string, kA/m
- Frequency - string, Hz
- SensorOutputVoltage - string, mV/V

Data:

The second struct contains the preporcessed characteriazation data of the TDK TAS2141 TMR angular Sensor. It is divided into two main structs one for the magnetic field reference points of the characterization images and one for the characteriazation sensor output images.

- **MagneticField** - struct, contain Hx- and Hy-field vectors which are the resolution references to each pixel in the characterization images of the sensors preprocessed bridge outputs
- **SensorOutput** - struct, contains structs for cosinus and sinus bridge outputs preprocessed in images of size of 256x256 pixels where each pixels references a bridge output in mV to a certain Hx- and Hy-fieldstrength amplitdue

- **MagneticField:**

- hx - array, Hx field axis of characteriazation images column vector of 1x256 double values from -25 kA/m to 25 kA/m with a resolution of 0.1961 kA/m
- hy - array, Hy field axis of characteriazation images column vector of 1x256 double values from -25 kA/m to 25 kA/m with a resolution of 0.1961 kA/m

- **SensorOutput:**

- **CosinusBridge** - struct, contains preprocessed characterization results of the sensors cosinus bridge outputs
- **SinusBridge** - struct, contains preprocessed characterization results of the sensors sinus bridge outputs

- **CosinusBridge:**

- Rise - array, double array of size 256x256 which references the cosinus bridge outputs for rising modulated stimulus amplitude to each cross reference of vectors MagneticField.hx and MagneticField.hy

- Fall - array, double array of size 256x256 which references the cosinus bridge outputs for falling modulated stimulus amplitude to each cross reference of vectors MagneticField.hx and MagneticField.hy
- All - array, double array of size 256x256 superimposed image of Rise and Fall
- Diff - array, double array of size 256x256 differentiated image of Rise and Fall

- **SinusBridge:**

- Rise - array, double array of size 256x256 which references the sinus bridge outputs for rising modulated stimulus amplitude to each cross reference of vectors MagneticField.hx and MagneticField.hy
- Fall - array, double array of size 256x256 which references the sinus bridge outputs for falling modulated stimulus amplitude to each cross reference of vectors MagneticField.hx and MagneticField.hy
- All - array, double array of size 256x256 superimposed image of Rise and Fall
- Diff - array, double array of size 256x256 differentiated image of Rise and Fall

The edited raw dataset provided from Thorben Schüthe is save with Matlabs build-in save function in a certain way to perform partial loads from the dataset.

```
save('data/NXP_KMZ60_Characterization_2020-12-03_16-53-16-721.mat', ...
     'Info', 'Data', '-v7.3', '-nocompression')
```

Created on December 05. 2020 Tobias Wulf. Copyright Tobias Wulf 2020.

Published with MATLAB® R2020b

Config Mat

The configuration mat-file is a script generated mat-file. Generated by generateConfigMat script. The mat-files contains program and software wide useful configuration like path variables or parameter settings for program tasks or functions. It centralizes the program controlling configuration at once and can be full or partial loaded at different program stages. The key point is the configuration can only be modified by the generating script so that the config values are truly constant. Varaition should be saved to temp folder or a temp mat-file. The configuration should be generated after major changes to the program or an established regeneration flow at program startup. The config.mat file is located under data directory and to path variable. Just load in to the needed workspace.

```
load('config.mat')
```

Contents

Requirements

- Other m-files scripts/generateConfigMat
- Subfunctions: None
- MAT-files required: None

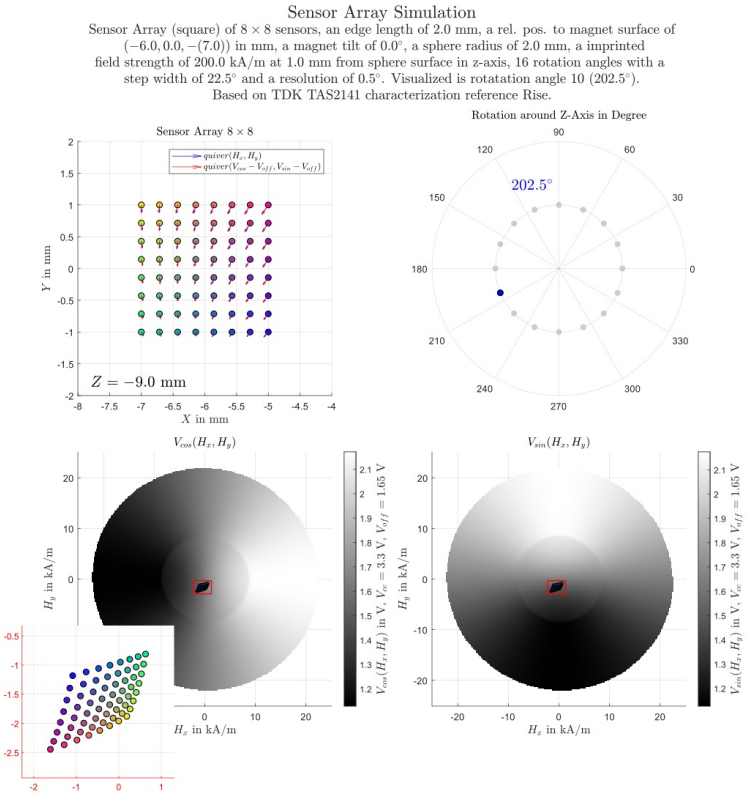
See Also

- generateConfigMat

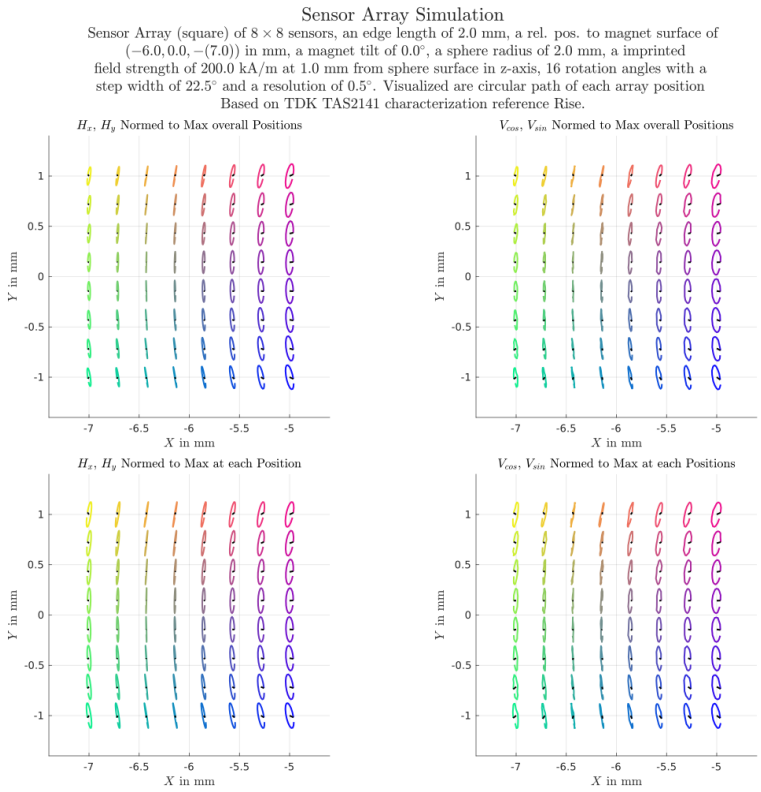
Published with MATLAB® R2020b

Training and Test Datasets

Training and test datasets are generated by sensor array simualtion part of the software. One dataset contains the simulation results generated with current configuration of used magnet in simulation and a setup of position and sensor behavior. The simulation computes for configured angles with certain angle resultion the magnetic field strength at sensor array member position for a rotation of the magnet through the configured angles. With respect to positions and angles the simulation maps the field strength for each array member to specified characterization field (current TDK Rise) and interpolates (nearest neighbor) the sensor bridge output voltages for cosinus and sinus bridge for each sensor array member. The aquirred data is save in matrices with same orientation as sensor array member matrice or corrodinate matrices of the sensor array, so it completes the rotation in related data matrices.



Above image shows the 10th rotation angle of a rotation with 16 even distributed angles between 0° and 360°.



Here a visualization of the full rotation at once with colored position related to the simulation plot of single simulation angle. The circular path are the path in in characterization field.

Training and test datasets filenames are build by a certain pattern.

[Training|Test]_YYYY-mm-dd-_HH-MM-SS-FFF.mat

They are saved under data path data/training and data/test.

A best practice can been seen in workflow topic of the documentation.

Contents

Dataset Structure

Info:

A training ot test dataset is parted into two main structures the first one the Info struct contains information about the simulation configuration and setup in which the simulation constructed the dataset.

- **SensorArrayOptions** - struct, contains setting of sensor size and behavior
- **DipoleOptions** - struct, contains setting of used magnet which was used in the simulation

- **UseOptions** - struct, contains information about use of the dataset if it is constructed for training or test use, sensor array position, number of angles, tilt of manget and so on.
- CharData - string, identifies the characteriazation data set which was used to simulate the array members.
- **Units** - struct, si units of data in datasets
- filePath - string, which points on the absolute path origin where the dataset was saved including filename.

- **SensorOptions:**

- geometry - char, indentifier string of which shape the sensor array geometry was constructed, geometry of used meshgrid in computation
- dimension - double, number of sensors at one array edge for square geometry
- edge - double, edge lenght in mm of senseor array
- Vcc - double, supply voltage of the sensor array
- Voff - double, bridge offset voltage off the sensor array
- Vnorm - double, norm value to get voltage values from characterization fields in combination with Vcc and Voff, TDK dataset is normed in mV/V.
- SensorCount, double - number of sensors in the sensor array for square geometry it is square or dimension

- **DipoleOptions:**

- sphereRadius - double, radius in mm around dipole magnet to approximate a spherical magnet in simulation with far field approximation (dipole field equation)
- H0mag - double, field strenght magnitude in kA/m which is imprinted on the compute field strength of the use magnet in a certain distance from magnet surface to consturct magnet with fitting characteristics for simualtion.
- z0 - double, distance from surface in which H0mag is imprinted on field computed field strength of the use magnet. Imprinting respects magnet tilts so the distance is always set to the magnet z-axis with no shifts in x and y direction
- M0mag - double, magnetic dipole moment magnitude which is used to define the magnetization direction of the magnet in tis rest position.

- **UseOptions:**

- useCase - string, identifies the dataset if it is for training or test purpose
- xPos - double, relative sensor array position to magnet surface
- yPos - double, relative sensor array position to magnet surface
- zPos - double, relative sensor array position to magnet surface
- tilt - double, magnet tilt in z-axis
- angleRes - double, angle resolution of rotation angles in simulation
- phaseIndex - double, start phase of rotation as index of full scale rotation angles with angleRes
- nAngles - double, number of rotation angles in datasets
- BaseReference - char, identifier which characterization dataset was loaded
- BridgeReference - char, identifier which reference from characteriazation dataset was used to generate cosinus and sinus voltages

- **Units:**

- SensorOutputVoltage - char, si unit of sensor bridge outputs
- MagneticFieldStrength - char, si unit of magnetic field strength

- Angles - char, si unit of angles
- Length - char, si unit of metric length

Data:

- HxScale - 1 x L double vector of Hx field strength amplitudes used in charcteriztion to construct sensor characterization references, x scale for characterization reference
- HyScale - 1 x L double vector of Hy field strength amplitudes used in charcteriztion to construct sensor characterization references, y scale for characterization reference
- VcosRef - L x L double matrix of cosinuns bridge characterization field corresponding to HxScale and HyScale
- VsinRef - L x L double matrix of sinuns bridge characterization field corresponding to HxScale and HyScale
- Gain - double, scalar gain factor for bridge outputs (interanl amplification)
- r0 - 3 x 1 double vector of magnet rest position from magnet surface and respect to magnet magnet tilt, used in computation of H0norm to imprint a certain field strength on magnets H-field, respects sphere radius of magnet
- m0 - 3 x 1 vector of magnetic dipole moment in magnet rest position with respect of manget tilt, used to compute H0norm to imprint a certain field strength on magnet H-field, the magnitude of this vector is equal to M0mag
- H0norm - double, scalar factor to imprint a certain field strength on magnet H-field in rest position with respect to magnet tilt in coordinate system
- m - 3 x M double vector of magnetic dipole rotation moments each 3 x 1 vector is related to i-th rotation angle
- angles - 1 x M double vector of i-th rotation angles in degree
- angleStep - double, scalar of angle step width in rotation
- angleRefIndex - 1 x M double vector of indices which refer to a full scale rotation of configure angle resuolution, so it abstracts a subset angle rotation to the same rotation with all angles given by angle resolution
- X - N x N double matrix of x coordinate positions of each sensor array member
- Y - N x N double matrix of y coordinate positions of each sensor array member
- Z - N x N double matrix of z coordinate positions of each sensor array member
- Hx - N x N x M double matrix of compute Hx-field strength at each sensor array member position for each rotation angle 1...M
- Hy - N x N x M double matrix of compute Hy-field strength at each sensor array member position for each rotation angle 1...M
- Hz - N x N x M double matrix of compute Hz-field strength at each sensor array member position for each rotation angle 1...M
- Habs - N x N x M double matrix of compute H-field strength magnitude at each sensor array member position for each rotation angle 1...M
- Vcos - N x N x M double matrix of computed cosinus bridge outputs at each sensor array member position for each rotation angle 1...M
- Vsin - N x N x M double matrix of computed sinus bridge outputs at each sensor array member position for each rotation angle 1...M

See Also

- Simulation Workflow
- sensorArraySimulation
- simulateDipoleSensorArraySquareGrid
- generateSimulationDatasets
- generateConfigMat

Published with MATLAB® R2020b

Unit Tests

Unit Tests are provideing way to test core functionallity of the written software components. Matlab supports various methods to apply Unit Tests. The designed tests are using script-based testing. So for each function or functionallity needs to be tested a own test script is written and gathered into a main test script where all standalone test scripts are combined to a test suite and executed at once.

Contents

runTests

Test suite script which executes all Unit Tests scripts at once and gathers the test results in a Matlab table.

removeFilesFromDirTest

Test of function removeFilesFromDir. Creates several files and directories and deletes them during testing.

rotate3DVectorTest

Test rotate3DVector function. Do some rotations and check results.

generateDipoleRotationMomentsTest

Test the generation of magnetic dipole moments for a full rotation between 0° and 360°.

generateSensorArraySquareGridTest

Test the meshgrid generation of the sensor array and shifting it in x and y direction.

computeDipoleH0NormTest

Test magnetic field norming function. Simple test of consitent data.

computeDipoleHFieldTest

Test the magnetic dipole equation to generate dipole fields in 3D meshgrid of data points. Test field characteristics like symmetry and so on.

tiltRotationTest

Test tilt rotation of a dipole magnetic. Tilt magnet and coordinate cross to fetch pole values during rotation.

Requirements

- Other m-files required: None
- Subfunctions: None
- MAT-files required: None

See Also

- Script-Based Unit Tests
- Write Script-Based Unit Tests
- Write Script-Based Unit Tests Using Local Functions
- Analyze Test Case Result

Created on December 14. 2020 by Tobias Wulf.

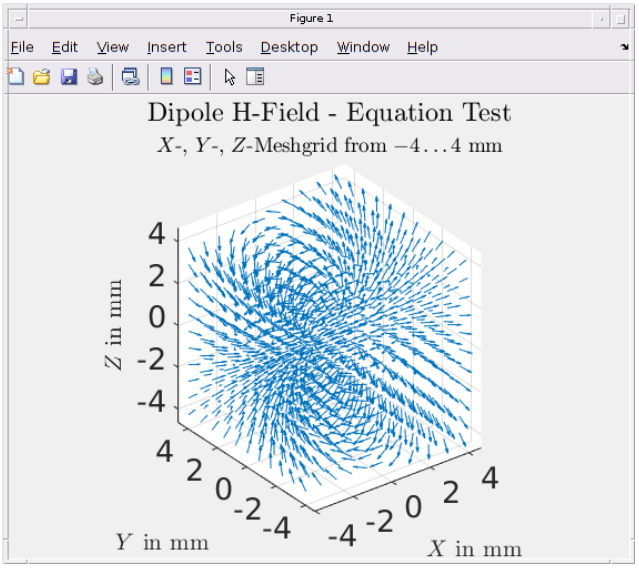
Published with MATLAB® R2020b

```
% clean workspace
clearvars;

% build suite from test files
suite = testsuite({'removeFilesFromDirTest', 'rotate3DVectorTest', ...
    'generateDipoleRotationMomentsTest', ...
    'generateSensorArraySquareGridTest', ...
    'computeDipoleH0NormTest', ...
    'computeDipoleHFieldTest', ...
    'tiltRotationTest'});

% run tests
results = run(suite);

% show results
disp(results)
disp(table(results))
cd ..
```

```
Running removeFilesFromDirTest
..
Done removeFilesFromDirTest
__________

Running rotate3DVectorTest
....
Done rotate3DVectorTest
__________

Running generateDipoleRotationMomentsTest
...
Done generateDipoleRotationMomentsTest
__________

Running generateSensorArraySquareGridTest
....
Done generateSensorArraySquareGridTest
__________

Running computeDipoleH0NormTest
.
Done computeDipoleH0NormTest
__________

Running computeDipoleHFieldTest
......
Done computeDipoleHFieldTest
__________

Running tiltRotationTest
..
Done tiltRotationTest
__________

  1×22 TestResult array with properties:

    Name
    Passed
    Failed
    Incomplete
    Duration
    Details

Totals:
   22 Passed, 0 Failed, 0 Incomplete.
   1.4049 seconds testing time.
```

Name	Passed	Failed	Incomplete	Duration	Details
{'removeFilesFromDirTest/Test1_DeleteAllFiles' }	true	false	false	0.075	{1×1 struct}
{'removeFilesFromDirTest/Test2_DeleteWithPattern' }	true	false	false	0.046689	{1×1 struct}
{'rotate3DVectorTest/Test1_OutputDimensions' }	true	false	false	0.11348	{1×1 struct}
{'rotate3DVectorTest/Test2_RotateVectorsInX_axes' }	true	false	false	0.008738	{1×1 struct}
{'rotate3DVectorTest/Test3_RotateVectorsInY_axes' }	true	false	false	0.008764	{1×1 struct}
{'rotate3DVectorTest/Test4_RotateVectorsInZ_axes' }	true	false	false	0.006091	{1×1 struct}
{'generateDipoleRotationMomentsTest/Test1_OutputDimensions' }	true	false	false	0.13145	{1×1 struct}
{'generateDipoleRotationMomentsTest/Test2_DownSampling' }	true	false	false	0.007222	{1×1 struct}
{'generateDipoleRotationMomentsTest/Test3_PhaseShift' }	true	false	false	0.008659	{1×1 struct}
{'generateSensorArraySquareGridTest/Test1_OutputDimensions' }	true	false	false	0.12205	{1×1 struct}
{'generateSensorArraySquareGridTest/Test2_EqualXAndYDistances' }	true	false	false	0.014129	{1×1 struct}
{'generateSensorArraySquareGridTest/Test3_ConstantZDistances' }	true	false	false	0.011084	{1×1 struct}
{'generateSensorArraySquareGridTest/Test3_PositionShifInXAndYDirection'}	true	false	false	0.007205	{1×1 struct}

```
    {'computeDipoleH0NormTest/Test1_PositiveScalarFactor'                }    true      false       false        0.13903    {1×1 struct}
    {'computeDipoleHFieldTest/Test1_OutputDimensions'                    }    true      false       false        0.30618    {1×1 struct}
    {'computeDipoleHFieldTest/Test2_CenterOfField'                       }    true      false       false       0.017691    {1×1 struct}
    {'computeDipoleHFieldTest/Test3_Magnetization'                       }    true      false       false       0.019976    {1×1 struct}
    {'computeDipoleHFieldTest/Test4_Imprinting'                          }    true      false       false       0.012584    {1×1 struct}
    {'computeDipoleHFieldTest/Test5_Symmetry'                            }    true      false       false        0.02123    {1×1 struct}
    {'computeDipoleHFieldTest/Test6_UnitsMilliKilo'                      }    true      false       false       0.043362    {1×1 struct}
    {'tiltRotationTest/Test1_RotationWithoutTilt'                        }    true      false       false        0.27578    {1×1 struct}
    {'tiltRotationTest/Test2_RotationWithTilt'                           }    true      false       false       0.008487    {1×1 struct}
```



Published with MATLAB® R2020b

Contents


```
% create test directory with files
cd(fileparts(which('removeFilesFromDirTest')));
mkdir('testDir');
fclose(fopen(fullfile('testDir', 'testFile1.txt'), 'w'));
fclose(fopen(fullfile('testDir', 'testFile2.txt'), 'w'));
fclose(fopen(fullfile('testDir', 'testFile3.txt'), 'w'));
```

Test 1: delete all files

```
removeStatus = removeFilesFromDir(fullfile('testDir'));
assert(removeStatus == true)

% create more files
fclose(fopen(fullfile('testDir', 'testFile1.txt'), 'w'));
fclose(fopen(fullfile('testDir', 'testFile2.txt'), 'w'));
fclose(fopen(fullfile('testDir', 'testFile3.txt'), 'w'));
```

Test 2: delete with pattern

```
removeStatus = removeFilesFromDir(fullfile('testDir'), '*.txt');
assert(removeStatus == true)

% clean up
rmdir('testDir');
```

Published with MATLAB® R2020b

Contents


```
% create column vectors with simple direction for rotations along the axes
% without tilts in other achses.
x = [-1; 0; 0];
y = [0; -1; 0];
z = [0; 0; -1];

% set angle step width in degree to rotate at choosen axes (x, y, or z)
angle = 90;
```

Test 1: output dimensions

```
rotated = rotate3DVector(x, 0, 0, angle);
assert(isequal(size(rotated), [3, 1]))
rotated = rotate3DVector([x x x x x], 0, 0, angle);
assert(isequal(size(rotated), [3, 6]))
```

Test 2: rotate vectors in x-axes

```
rotated = rotate3DVector([x y z], 0, 0, 0); % 0 degree
assert(isequal(rotated, [-1 0 0; 0 -1 0; 0 0 -1]))

rotated = rotate3DVector([x y z], angle, 0, 0); % 90 degree
assert(isequal(rotated, [-1 0 0; 0 0 1; 0 -1 0]))

rotated = rotate3DVector([x y z], 2 * angle, 0, 0); % 180 degree
assert(isequal(rotated, [-1 0 0; 0 1 0; 0 0 1]))

rotated = rotate3DVector([x y z], 3 * angle, 0, 0); % 270 degree
assert(isequal(rotated, [-1 0 0; 0 0 -1; 0 1 0]))

rotated = rotate3DVector([x y z], 4 * angle, 0, 0); % 360 degree
assert(isequal(rotated, [-1 0 0; 0 -1 0; 0 0 -1]))
```

Test 3: rotate vectors in y-axes

```
rotated = rotate3DVector([x y z], 0, 0, 0); % 0 degree
assert(isequal(rotated, [-1 0 0; 0 -1 0; 0 0 -1]))

rotated = rotate3DVector([x y z], 0, angle, 0); % 90 degree
assert(isequal(rotated, [0 0 -1; 0 -1 0; 1 0 0]))

rotated = rotate3DVector([x y z], 0, 2 * angle, 0); % 180 degree
assert(isequal(rotated, [1 0 0; 0 -1 0; 0 0 1]))

rotated = rotate3DVector([x y z], 0, 3 * angle, 0); % 270 degree
```

```
assert(isequal(rotated, [0 0 1; 0 -1 0; -1 0 0]))

rotated = rotate3DVector([x y z], 0, 4 * angle, 0); % 360 degree
assert(isequal(rotated, [-1 0 0; 0 -1 0; 0 0 -1]))
```

Test 4: rotate vectors in z-axes

```
rotated = rotate3DVector([x y z], 0, 0, 0); % 0 degree
assert(isequal(rotated, [-1 0 0; 0 -1 0; 0 0 -1]))

rotated = rotate3DVector([x y z], 0, 0, angle); % 90 degree
assert(isequal(rotated, [0 1 0; -1 0 0; 0 0 -1]))

rotated = rotate3DVector([x y z], 0, 0, 2 * angle); % 180 degree
assert(isequal(rotated, [1 0 0; 0 1 0; 0 0 -1]))

rotated = rotate3DVector([x y z], 0, 0, 3 * angle); % 270 degree
assert(isequal(rotated, [0 -1 0; 1 0 0; 0 0 -1]))

rotated = rotate3DVector([x y z], 0, 0, 4 * angle); % 360 degree
assert(isequal(rotated, [-1 0 0; 0 -1 0; 0 0 -1]))
```

Published with MATLAB® R2020b

Contents


```
% create full scale rotation with 0.5° resolution and no tilt,
% return moments
% and corressponding angles theta
amp = 1e6;
tilt = 0;
res = 0.5;
[MFS, tFS] = generateDipoleRotationMoments(amp, 0, tilt, res);

% create same rotation but only a subset of angles N = 7
% with equal distances to each and another, return additionally index which
% reference to full scale
[M, t, idx] = generateDipoleRotationMoments(amp, 7, tilt, res);

% create shifted subset, shift by 22 positions in full scale theta,
% so with 0.5° resolution it is phase shift by 11°
[MSH, tSH, idxSH] = generateDipoleRotationMoments(amp, 7, tilt, res, 22);
```

Test 1: output dimensions

```
assert(isequal(size(MFS), [3 720]))
assert(isequal(size(tFS), [1 720]))
assert(isequal(size(M), [3 7]))
assert(isequal(size(t), [1 7]))
assert(isequal(size(idx), [1 7]))
assert(isequal(size(MSH), [3 7]))
assert(isequal(size(tSH), [1 7]))
assert(isequal(size(idxSH), [1 7]))
```

Test 2: down sampling

```
assert(isequal(MFS(:,idx), M))
assert(isequal(tFS(idx), t))
assert(isequal(MFS(:,idxSH), MSH))
assert(isequal(tFS(idxSH), tSH))
```

Test 3: phase shift

```
assert(isequal(tSH(1), 11))
assert(isequal(idx, idxSH - 22))
assert(isequal(MFS(:,idx + 22), MSH))
assert(isequal(tFS(idx + 22), tSH))
```

Published with MATLAB® R2020b

Contents


```
% create sensor array infos for size and position
% number of sensors at one edge
N = 8;

% sensor array edge length in mm
a = 2;

% relative postion of the sensor array to the center of a 3D cordinate
% system (z inverse)
p = [0; 0; 2];

% z offset, later used as sphere radius of a dipole which is placed in the
% center of the coordinate system
r = 2;

% generate coordinates grid
[X, Y, Z] = generateSensorArraySquareGrid(N, a, p, r);

% create a shift in same layer
p2 = [-2; 3; 2];
[X2, Y2, Z2] = generateSensorArraySquareGrid(N, a, p2, r);
```

Test 1: output dimensions

```
assert(isequal(size(X), [N N]))
assert(isequal(size(Y), [N N]))
assert(isequal(size(Z), [N N]))
```

Test 2: equal x and y distances

```
assert(isequal(diff(Y), diff(-X, [], 2)'))
```

Test 3: constant z distances

```
assert(all(Z == -(p(3) + r), 'all'))
```

Test 3: position shif in x and y direction

```
assert(isequal(X + p2(1), X2))
assert(isequal(Y + p2(2), Y2))
assert(isequal(Z, Z2))
```

Published with MATLAB® R2020b

Contents


```
% create a dipole with constant sphere radius in rest position and relative
% to sensor array with position x=0, y=0, z=0
% sphere radius 2mm
r = 2;
% distance in which the field stength is imprinted
z = 5;
% field strengt magnitude to imprint in dipole field on sphere radius kA/m
Hmag = 8.5;
% magnetic moment magnitude which rotates the dipole without tilt
Mmag = 1e6;

% compute norm factor
H0norm = computeDipoleH0Norm(Hmag, [Mmag; 0; 0], [0; 0; -(z + r)]);
```

Test 1: positive scalar factor

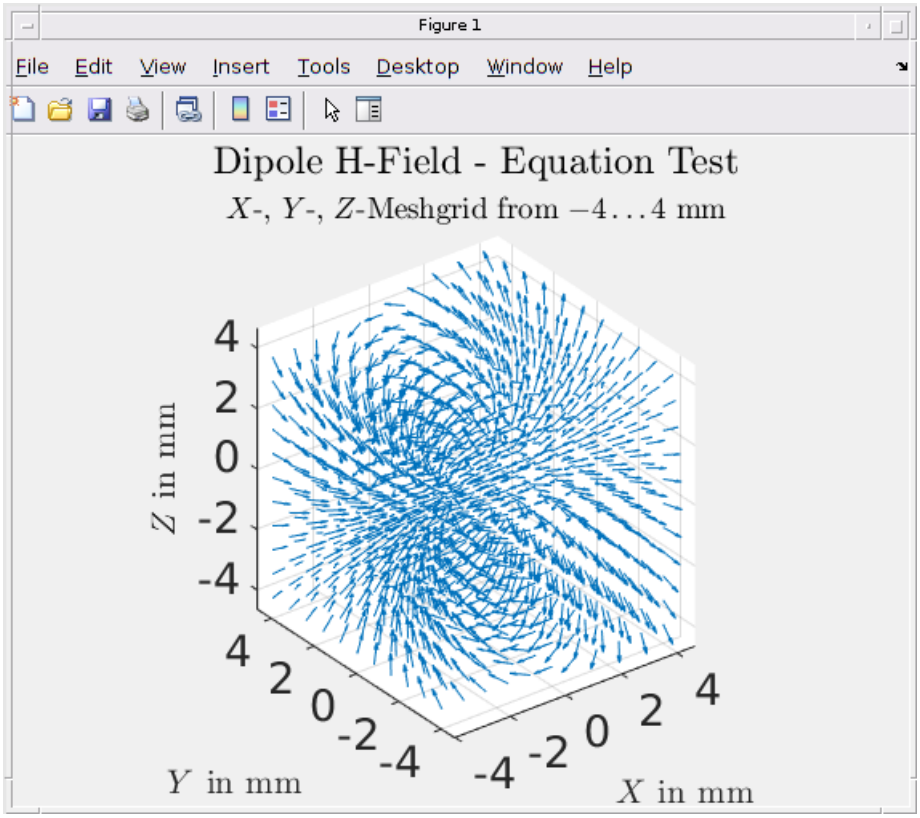
```
assert(isscalar(H0norm))
assert(H0norm > 0)
```

Published with MATLAB® R2020b

Contents


```
% compute a single point without norming
Hsingle = computeDipoleHField(1, 2, 3, [1; 0; 0], 1);

% compute a 3D grid of positions n+1 samples for even values
% in the grid and to
% include (0,0,0), in mm
x = linspace(-4, 4, 41);
y = linspace(4, -4, 41);
z = linspace(4, -4, 41);
[X, Y, Z] = meshgrid(x, y, z);

% magnetic dipole moment to define magnet orientation, no tilt
m = generateDipoleRotationMoments(-1e6, 1, 0);

% norm factor to imprint field strength in certain distance d = 1,
% r = 2 in mm,
% 200 kA/m, no tilt
r0 = rotate3DVector([0; 0; -3], 0, 0, 0);
H0norm = computeDipoleH0Norm(200, m, r0);

% allocate memory for field components in x,y,z
Hx = zeros(41, 41, 41);
Hy = zeros(41, 41, 41);
Hz = zeros(41, 41, 41);

% compute without norming for each z layer and reshape results into layer
for i=1:41
    Hgrid = computeDipoleHField(X(:,:,i),Y(:,:,i),Z(:,:,i),m,H0norm);
    Hx(:,:,i) = reshape(Hgrid(1,:),41,41);
    Hy(:,:,i) = reshape(Hgrid(2,:),41,41);
    Hz(:,:,i) = reshape(Hgrid(3,:),41,41);
end

% calculate magnitude in each point for better view the results
Habs = sqrt(Hx.^2+Hy.^2+Hz.^2);

% define a index to view only every 4th point for not overcrowded plot
idx = 1:4:41;

% downsample and norm
Xds = X(idx,idx,idx);
Yds = Y(idx,idx,idx);
Zds = Z(idx,idx,idx);
Hxds = Hx(idx,idx,idx) ./ Habs(idx,idx,idx);
Hyds = Hy(idx,idx,idx) ./ Habs(idx,idx,idx);
Hzds = Hz(idx,idx,idx) ./ Habs(idx,idx,idx);
```

```
% show results for test, comment out for regular unittest run, run suite
quiver3(Xds, Yds, Zds, Hxds, Hyds, Hzds);
xlabel('$X$ in mm', ...
        'FontWeight', 'normal', ...
        'FontSize', 16, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');
ylabel('$Y$ in mm', ...
        'FontWeight', 'normal', ...
        'FontSize', 16, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');
zlabel('$Z$ in mm', ...
        'FontWeight', 'normal', ...
        'FontSize', 16, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');
title('Dipole H-Field - Equation Test', ...
        'FontWeight', 'normal', ...
        'FontSize', 18, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');
subtitle('$X$-, $Y$-, $Z$-Meshgrid from $-4 \ldots 4$ mm', ...
        'FontWeight', 'normal', ...
        'FontSize', 14, ...
        'FontName', 'Times', ...
        'Interpreter', 'latex');
axis equal;

% pattern for logical indexing the center or opposite
p0 = false(1, 41);
p0(21) = true;
pN0 = true(41, 41, 41);
pN0(21,21, 21) = false;

% pattern for symmetry investigation
plu = [true(1, 20), false, false(1, 20)];
prl = [false(1, 20), false, true(1,  20)];

% compare values to check if fits in unit pairs of m and A/m
% and mm and kA/m
r0Apm = rotate3DVector([0; 0; -3e-3], 0, 0, 0);
H0normApm = computeDipoleH0Norm(200e3, m, r0Apm);
Xm = X * 1e-3;
Ym = Y * 1e-3;
Zm = Z * 1e-3;
HxApm = zeros(41, 41, 41);
HyApm = zeros(41, 41, 41);
HzApm = zeros(41, 41, 41);
for i=1:41
    HApm = computeDipoleHField(Xm(:,:,i),Ym(:,:,i),Zm(:,:,i),m,H0normApm);
    HxApm(:,:,i) = reshape(HApm(1,:),41,41);
    HyApm(:,:,i) = reshape(HApm(2,:),41,41);
    HzApm(:,:,i) = reshape(HApm(3,:),41,41);
end
HabsApm = sqrt(HxApm.^2+HyApm.^2+HzApm.^2);
```



Test 1: output dimensions

```
assert(isequal(size(Hsingle), [3, 1]))
assert(isequal(size(Hgrid), [3, 1681]))
```

Test 2: center of field

```
assert(X(p0,p0,p0) == 0)
assert(Y(p0,p0,p0) == 0)
assert(Z(p0,p0,p0) == 0)
assert(isnan(Hx(p0,p0,p0)))
assert(isnan(Hy(p0,p0,p0)))
assert(isnan(Hz(p0,p0,p0)))
assert(all(Hx(~p0,p0,p0) ~= 0, 'all'))
assert(all(Hx(p0,~p0,p0) ~= 0, 'all'))
assert(all(Hy(~p0,p0,p0) == 0, 'all'))
assert(all(Hy(p0,~p0,p0) == 0, 'all'))
assert(all(Hz(~p0,~p0,p0) == 0, 'all'))
```

Test 3: magnetization

```
assert(all(Hx(~p0,p0,~p0) ~= 0, 'all'))
assert(all(Hx(p0,~p0,~p0) ~= 0, 'all'))
assert(all(Hx(p0,p0,~p0) ~= 0, 'all'))
```

```
assert(all(Hy(~p0,p0,~p0) == 0, 'all'))
assert(all(Hy(p0,~p0,~p0) == 0, 'all'))
assert(all(Hy(p0,p0,~p0) == 0, 'all'))
assert(all(Hz(~p0,p0,~p0) == 0, 'all'))
assert(all(Hz(p0,~p0,~p0) ~= 0, 'all'))
assert(all(Hz(p0,p0,~p0) == 0, 'all'))
```

Test 4: imprinting

index 6 is 3mm and 36 is -3mm from surface where 200 kA/m should be imprinted

```
assert(round(abs(Hx(p0,p0,6)),6) == 200)
assert(round(abs(Hx(p0,p0,36)),6) == 200)
```

Test 5: symmetry

```
assert(all((Hx(plu,:,:) - flip(Hx(prl,:,:),1))==0, 'all'))
assert(all((Hx(:,plu,:) - flip(Hx(:,prl,:),2))==0, 'all'))
assert(all((Hy(plu,:,:) + flip(Hy(prl,:,:),1))==0, 'all'))
assert(all((Hy(:,plu,:) + flip(Hy(:,prl,:),2))==0, 'all'))
assert(all((Hz(:,:,~p0) + flip(Hz(:,:,~p0),2))==0, 'all'))
```

Test 6: units milli kilo

```
assert(all(round(HxApm(pN0) * 1e-3, 6) == round(Hx(pN0), 6), 'all'))
assert(all(round(HyApm(pN0) * 1e-3, 6) == round(Hy(pN0), 6), 'all'))
assert(all(round(HzApm(pN0) * 1e-3, 6) == round(Hz(pN0), 6), 'all'))
assert(all(round(HabsApm(pN0) * 1e-3, 6) == round(Habs(pN0), 6), 'all'))
```

Published with MATLAB® R2020b

Contents


```
% clean
clearvars;

% relevant tilt in y axes
tilt = 0.5:0.5:90;

% magnetic dipole moment to define magnet orientation, no tilt
% rotate angles theta 0°, 90°, 180°, 270°
[mNoTilt, thetaNoTilt] = generateDipoleRotationMoments(-1e6, 4, 0);

% Habs for magnetization from north to south from -x to x
HabsMust = [400 400 200 200 200 200];

% norm factor to imprint field strength in certain distance d = 1,
% r = 2 in mm,
% 200 kA/m, no tilt
r0NoTilt = [0; 0; -3];
H0normNoTilt = computeDipoleH0Norm(200, mNoTilt(:,1), r0NoTilt);

% axes with no tilt, rest position
AxesNoTilt = [-3, 3,  0, 0,  0, 0;
              0, 0, -3  3,  0, 0;
              0, 0,  0, 0, -3, 3];

% calc fields along coorditante cross and magnitudes
HNoTilt = zeros(3, 6, 4);
for i = 1:4
    % rotate axes same wise to check pole values
    RotateNoTiltAxes = rotate3DVector(AxesNoTilt, 0, 0, thetaNoTilt(i));
    HNoTilt(:,:,i) = computeDipoleHField(RotateNoTiltAxes(1,:), ...
        RotateNoTiltAxes(2,:), RotateNoTiltAxes(3,:), ...
        mNoTilt(:,i), H0normNoTilt);
end

% habs must be show imprinted field strength and double of it at poles
HabsNoTilt = sqrt(sum(HNoTilt.^2, 1));

% calc fields along tilt coorditante cross and magnitudes
HTilt = zeros(3, 6, 4, length(tilt));
for j = 1:length(tilt)
    % magnetic dipole moment to define magnet orientation, with tilt
    % rotate angles theta 0°, 90°, 180°, 270°
    [mTilt, thetaTilt] = generateDipoleRotationMoments(-1e6, 4, tilt(j));

    % norm factor to imprint field strength in certain distance d = 1,
    % r = 2 in mm,
    % 200 kA/m, no tilt
    r0Tilt = rotate3DVector(r0NoTilt, 0, tilt(j), 0);
    H0normTilt = computeDipoleH0Norm(200, mTilt(:,1), r0Tilt);

    % axes with tilt, rest position
    AxesTilt = rotate3DVector(AxesNoTilt, 0, tilt(j), 0);
```

```
    for i = 1:4
        % rotate axes same wise to check pole values
        RotateTiltAxes = rotate3DVector(AxesTilt, 0, 0, thetaTilt(i));
        HTilt(:,:,i,j) = computeDipoleHField(RotateTiltAxes(1,:), ...
            RotateTiltAxes(2,:), RotateTiltAxes(3,:), ...
            mTilt(:,i), H0normTilt);
    end
end

% habs must be show imprinted field strength and double of it at poles
HabsTilt = sqrt(sum(HTilt.^2, 1));
```

Test 1: rotation without tilt

```
assert(all(round(HabsNoTilt, 6) == round(HabsMust, 6), 'all'))
```

Test 2: rotation with tilt

```
assert(all(round(HabsTilt, 6) == round(HabsMust, 6), 'all'))
```

Published with MATLAB® R2020b

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original