Appendix A - Manual
HERKosiDAE

Implementation
Half-explicit Runge-Kutta methods for overdetermined
semi-implicit differential-algebraic equations

Version 1.0

April 13, 2017

1 Structure

The files are organized in the following way. The main routine is `herkosidae.m` and uses the subroutines in the folder `Helpers`. The program is executed with `run.m` contained in the subfolders of the folder `Examples`. A new problem can be set up with the files in the folder `Mask for new Problem`.

```
HERKosiDAE
├─ Examples
│  ├─ ...
│  ├─ Chemakzo
│  ├─ Linear Circuit with one CV Loop
│  ├─ Mask for new Problem
│  ├─ Mathematical Pendulum
│  └─ ...
├─ Helpers
│  ├─ divnewton.m
│  ├─ getRKmethod.m
│  ├─ lusp.m
│  ├─ numjabion.m
│  ├─ selectors.m
│  ├─ simpledivnewton.m
│  └─ solveExf.m
├─ Manual
│  ├─ Manual.pdf
│  └─ HERKosiDAE.pdf
└─ herkosidae.m
```

2 Setup and Execution

To get to know the program, we recommend to get used to the functionality by fiddling with the given examples in the folder `Examples`. The program is executed with the `run.m` file within the subfolders, which defines the given system, variables and options. To start a computation simply click `Run` in the MATLAB Editor Window as shown below.

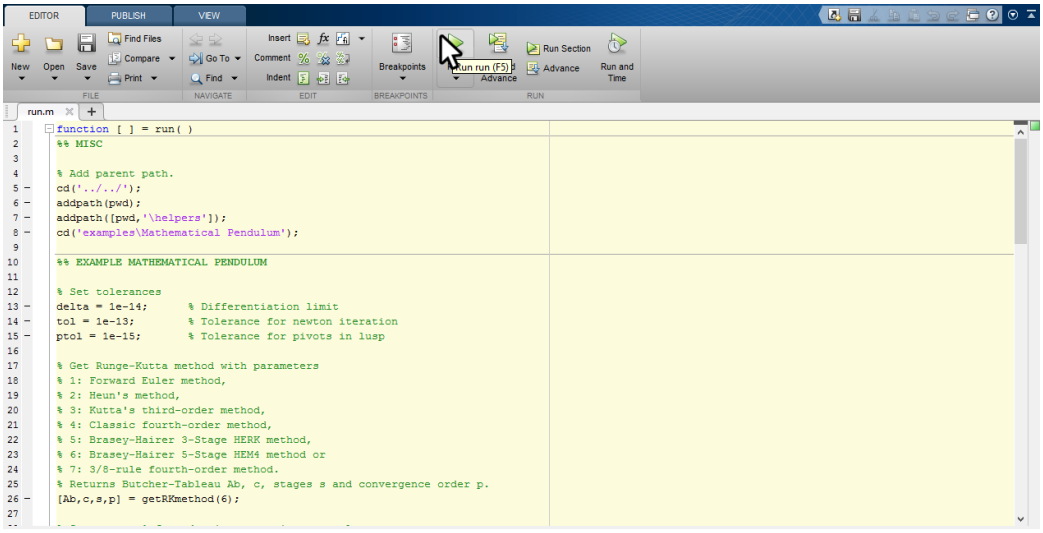

Figure 1: Execution of a predefined example

The easiest way to set up a new problem is to use the mask available in the folder `'../Examples/Mask for new Problem/'`, which contains the files

`func_E.m`	$\hat{=}$	defines $E(x,t)$,
`func_f.m`	$\hat{=}$	defines $f(x,t)$,
`func_g.m`	$\hat{=}$	defines $g(x,t)$,
`func_J.m`	$\hat{=}$	optionally defines Jacobian of $g(x,t)$,
`run.m`	$\hat{=}$	defines variables, settings and executes algorithm.

The definition of the model functions `E`, `f`, `g` and `J` is straight-forward. We use the additional input parameter `var` to pass constants from the executing file to the functions. For the execution of the main routine `herkosidae` in `run.m` we will need the following input parameters.

`delta`	$\hat{=}$	Differentiation limit for numerical differentiation
`tol`	$\hat{=}$	Tolerance for Newton iteration
`ptol`	$\hat{=}$	Tolerance for pivots
`Ab,c,s,p`	$\hat{=}$	Definition and parameters of Runge-Kutta method
`ssc`	$\hat{=}$	Boolean option 1 Adaptive step size control 0 Constant step size
`eps0`	$\hat{=}$	Desired accuracy for adaptive step size control
`beta`	$\hat{=}$	Safety factor for adaptive step size control
`Jopt`	$\hat{=}$	Boolean option 1 Jacobian of $g(x,t)$ analytically defined in `func_J.m` 0 Use numerical differentiation of $g(x,t)$
`Estat`	$\hat{=}$	Boolean option 1 Leading matrix $E(x,t)$ is invariant 0 Leading matrix $E(x,t)$ is not invariant
`Nopt`	$\hat{=}$	Boolean option 1 Use simplified Newton method 0 Use classical Newton method
`func`	$\hat{=}$	String of employed functions
`var`	$\hat{=}$	Container for constants
`x0`	$\hat{=}$	Initial value
`t0,tf`	$\hat{=}$	Start and finish time
`h0`	$\hat{=}$	Initial step size

Please note that if the user wishes to rename the folder, the executed path in the `MISC` section of `run.m` has to be adjusted. The program was written in MATLAB R2016b. Errors may occur due to version differences.

3 HERKosiDAE and Subroutines

3.1 HERKosiDAE

This is the main routine. Its core function iteratively calculates the numerical solution of the given overdetermined semi-implicit differential-algebraic equation with

$$\left.\begin{aligned} \mathcal{X}_i^d &= x_0^d + h \cdot \textstyle\sum_{j=1}^{i-1} a_{i,j}\ \dot{\mathcal{X}}_j^d, \\ 0 &= g(\mathcal{X}_i^d \oplus \mathcal{X}_i^a\,, t_0 + c_i h), \\ E(\mathcal{X}_i\,, t_0 + c_i h)\,\dot{\mathcal{X}}_i &= f(\mathcal{X}_i, t_0 + c_i h), \end{aligned}\right\} \quad \text{for } i = 1, ..., s$$

$$\begin{aligned} x_1^d &= x_0^d + h \cdot \textstyle\sum_{j=1}^{s} b_j\ \dot{\mathcal{X}}_j^d, \\ 0 &= g(x_1^d \oplus x_1^a, t_0 + h). \end{aligned}$$

For more details refer to Chapter 3 in the document `HERKosiDAE.pdf`. It also contains a wrapper function to reflect the functionality of the adaptive step size control, if the option `scc` is set to 1. For more details refer to Section 4.1.5 in the document `HERKosiDAE.pdf`.

Input:

As described above

Output:

1. Approximation to the solution `APPROX`
2. Vector of time points `T`
3. Vector of step sizes `H`

3.2 solveExf

Solves

$$E(x,t)\,\dot{x} = f(x,t)$$

for $\dot{x}$ with a LU decomposition. Also, finds indicator `FV` of required algebraic components while checking inhomogeneities of f.

Input:

1. Leading matrix function E
2. Preallocated memory `LE`, `UE`, `PE`, `QE`, `uE` for the decomposition of E
3. Right-hand side f
4. State x
5. Independent variable t
6. Bundle of static variables `var` used in given functions
7. Bundle `dim` that holds the dimensions of the system
8. Desired accuracy `eps0`
9. Tolerance for Newton iteration `tol`
10. Option that determines if E is invariant `Estat`

Output:

1. Solution $\dot{x}$
2. Indicator `FV` for required algebraic components
3. Information of decomposition `L`, `U`, `P`, `Q`, `u` of E

3.3 selectors

Determines algebraic and differential selectors. For more details refer to Section 3.2 and Section 4.1.3 in the document `HERKosiDAE.pdf`.

Input:

1. Given function g
2. Function to determine Jacobian $\mathcal{J}$
3. Preallocated memory for Jacobian of current state and time `Jac`
4. Boolean option for analytical determination of Jacobian `Jopt`
5. State x
6. Independent variable t
7. Increment `delta`
8. Tolerance for pivots `ptol`
9. Indicator for required algebraic components `FV`
10. Bundle of static variables `var` used in given functions
11. Bundle `dim` that holds the dimensions of the system

Output:

1. Selector for algebraic components `sa`
2. Inflated selector for algebraic components `SA`
3. Inflated selector for differential components `SD`
4. Vector with positions of selected algebraic components `q`

3.4 lusp

LU factorization with special pivoting such that

$$L\,U = P\,A\,Q.$$

Modification of LU factorization with complete pivoting (Matrix Computations Algorithm 3.4.2) implemented by Nick Henderson (c) 2010. See MathWorks File Exchange `27249`. For more details refer to Section 3.2 in the document `HERKosiDAE.pdf`.

Input:

1. Matrix A
2. Required algebraic components `FV`
3. Pivot tolerance `tol`

Output:

1. Permutation matrix Q
2. Vector with permutation order `q`

3.5 numjacobian

Simple function to determine entries of Jacobian of given function.

Input:

1. Given function f
2. Preallocated memory for Jacobian of current state and time `Jac`
3. Vector with positions of selected algebraic components `q`
4. State x
5. Independent variable t
6. Increment `delta`
7. Bundle of static variables `var` used in given functions

Output:

1. Jacobian $\mathcal{J}$ of given function

3.6 divnewton

Implements Divided Newton method. For more details refer to Section 3.3 in the document `HERKosiDAE.pdf`.

Input:

1. Given function g
2. Prior state X_i (technically X_{i-1})
3. Updated differential components X_i^d
4. Function to determine Jacobian $\mathcal{J}$
5. Preallocated memory for Jacobian of current state and time `Jac`
6. Boolean option for analytical determination of Jacobian `Jopt`
7. Vector with positions of selected algebraic components `q`
8. Selector for algebraic components `sa`
9. Inflated selector for algebraic components `SA`
10. Independent variable t
11. Increment for Jacobian `delta`
12. Tolerance `tol`
13. Bundle of static variables `var` used in given functions

Output:

1. Updated algebraic components X_i^a

3.7 simpledivnewton

Implements Simple Divided Newton method. For more details refer to Section 4.1.4 in the document `HERKosiDAE.pdf`.

Input:

1. Given function g
2. Prior state X_i (technically X_{i-1})
3. Updated differential components X_i^d
4. Function to determine Jacobian $\mathcal{J}$
5. Preallocated memory for Jacobian of current state and time `Jac`
6. Boolean option for analytical determination of Jacobian `Jopt`
7. Vector with positions of selected algebraic components `q`
8. Selector for algebraic components `sa`
9. Inflated selector for algebraic components `SA`
10. Independent variable t
11. Increment for Jacobian `delta`
12. Tolerance `tol`
13. Bundle of static variables `var` used in given functions

Output:

1. Updated algebraic components X_i^a

3.8 getRKmethod

Predefines coefficients, stage and order of Runge-Kutta methods. The program is shipped with the following options

1. Explicit Euler method
2. Heun's method
3. Kutta's third-order method
4. Classic fourth-order method
5. Brasey-Hairer 3-Stage HERK
6. Brasey-Hairer 5-Stage HEM4
7. 3/8-rule fourth-order method

Input:

1. Option `run`

Output:

1. Coefficients of selected Runge-Kutta method `Ab` and `c`
2. Stages `s`
3. Convergence order `p`