An Algebraic Framework for the Real-Time Solution of Inverse Problems on Embedded Systems

Christoph Gugg, Matthew Harker, Paul O'Leary and Gerhard Rath

This article presents a new approach to the real-time solution of inverse problems on embedded systems. The class of problems addressed corresponds to ordinary differential equations (ODEs) with generalized linear constraints, whereby the data from an array of sensors forms the forcing function. The algebraic discretization of the problem enables a one-to-one mapping of the ODE to its discrete equivalent linear differential operator, together with an additional matrix equation representing the constraints. The solution of the equation is formulated as a least squares (LS) problem with linear constraints. The LS approach makes the method suitable for the explicit solution of inverse problems where the forcing function is perturbed by noise. The algebraic computation is partitioned into a initial preparatory step, which precomputes the matrices required for the run-time computation; and the cyclic run-time computation, which is repeated with each acquisition of sensor data. The cyclic computation consists of a single matrix-vector multiplication, in this manner computation complexity is known a-priori, fulfilling the definition of a real-time computation. Numerical testing of the new method is presented on perturbed as well as unperturbed problems; the results are compared with known analytic solutions and solutions acquired from state-of-the-art implicit solvers. In all performed numerical tests the new method was both faster and more accurate for repeated solutions of the same ODE. The solution is implemented with model based design and uses only fundamental linear algebra; consequently, this approach supports automatic code generation for deployment on embedded systems. The targeting concept was tested via software- and processor-in-the-loop verification on two systems with different processor architectures. Finally, the method was tested on a laboratory prototype with real measurement data for the monitoring of flexible structures. The measurement arrangement consists of an embedded system with a chain of 14 inclinometer sensors connected to it, two additional nodes implement a total of four constraints. The problem solved is: the real-time overconstrained reconstruction of a curve from measured gradients. Such systems are commonly encountered in the monitoring of structures and/or ground subsidence.

1. MOTIVATION AND PROBLEM STATEMENT

The original motivation for this work was the development of a large scale cyber-physical system (CPS) to monitor ground subsidence and possible deformation of structures during the construction of the new City-Circle Line subway in Copenhagen, Denmark. Very stringent geo-mechanical monitoring requirements have been established

for underground construction projects in urban areas following an accident on March 3, 2009 in Cologne, Germany: the building of the city's archive collapsed into a Stadtbahn tunnel under construction on the Severinstraße, killing two people[1]. The monitoring concept consists of a large number of vertical holes sunk along the planned path of the tunnel distributed over a distance of approximately 15 [km]. Each of these holes is equipped with a series of rods, and each rod is equipped with a pair of inclinometers, effectively forming a chain of inclinometers. Chains of inclinometers are used in the monitoring of ground subsidence [Machan and Bennett 2008] and for measuring the deformation of structures [O'Leary and Harker 2012]. Determining the ground movement from the orientation of the rods is an inverse problem. Additionally, there are points where constraints are placed on the construction, for example pillars, which in turn define initial-, inner- or boundary values for the inverse problems. Reconstructing the deformation under these circumstances requires the solution of an inverse boundary value problem for each chain of rods. Consequently, it is necessary to solve a large number of inverse initial-, inner- or boundary value problems in real-time for different sets of measurement data. Each chain of inclinometers is equipped with an embedded system that acquires and processes the data from the sensors, forming an independent sensor node. The individual sensor nodes are part of a larger sensor network. Decentralized processing of measurement data introduces an implicit form of parallelism thanks to distributed computing. The network's bandwidth demands are lowered due to the higher information density.

Necsulescu also identified the necessity of solving inverse problems in critical infrastructure monitoring [Necsulescu and Ganapathy 2005]. Lee [Lee et al. 2012] identified that predictable real-time solutions of complex systems, with an understandable concurrency, are a key issue for future developments of CPS. He points out that this issue was inadequately dealt with in the past. There are numerous engineering and scientific applications which require the real-time solution of inverse problems, e.g. [Loh and Dickin 1996]. Therefore, this is clearly an area of research which is of significance.

2. SCOPE OF THE ARTICLE

This article develops a new method for the numerical solution of inverse problems based on a matrix algebraic approach. It provides global least squares solutions to inverse initial-, inner- or boundary value problems. The method has been developed specifically with the aim of solving inverse problems associated with measurement systems in an efficient manner, whereby multiple measurements are performed over time and repeated solutions of the same equation are required. The goal is to directly embed the solver onto the sensor node's hardware. The main contributions of the article are:

(1) A new algebraic approach to the numerical solution of inverse problems is derived. The method splits the calculations into two portions: a preparatory (offline) computation and a run-time (online) computation. The run-time computation is repeatedly performed with each new measurement. Solving the inverse problem at run-time is reduced to one matrix multiplication and one vector addition. In this manner, the exact number of floating point operations (FLOPs) is known a-priori, $W(n) = 2\,n^2$, where n is the number of measurement points. Additionally, the memory requirements are known in advance. Consequently, a strict upper-bound $\mathcal{O}(n^2)$ can be determined for the execution time on a given processor. This makes the method, by definition, suitable for real-time applications. Furthermore, the covariance propagation for perturbations of the sensor inputs to the solution is derived.

---

[1]An article relating to the incident can be found at http://www.ksta.de/html/artikel/1266930835566.shtml

This enables the computation of a confidence interval for the solution. The run-time computational complexity of estimating the confidence is $\mathcal{O}(n)$.

Extensive model-in-the-loop (MIL) testing of the method on a personal computer (PC) is presented to validate the method. The results of a classical Runge-Kutta type approach are compared with those obtained using the new approach. The results demonstrate the accuracy of the method and its numerical efficiency.

(2) A model based design (MBD) approach is presented which enables the system formulation at an abstract level. The presented model only utilizes fundamental linear algebra operations such as matrix multiplication and vector addition; consequently, automatic generation of C code becomes possible. Software-in-the-loop (SIL) verification is used to proof the functional equivalence of the model and the generated code. Embedded targeting enables the deployment of the code directly onto a microcontroller. The results computed by the embedded processor are compared to the results computed by the model running on a PC via processor-in-the-loop (PIL) verification. The viability of the model is demonstrated on a very limited, yet cheap and available, 8-bit microcontroller. Furthermore, a laboratory setup with a chain of inclinometers mounted on a flexible structure demonstrates the applicability of the model for real measurement data.

3. CONTINUOUS MEASUREMENT MODEL

The measurement model is central to this article: it defines the class of problem which is being solved. Furthermore, it defines the requirements for the MBD environment. The aim is to use MBD to automatically generate the functionally equivalent code which is capable of solving any example of this problem on an embedded system in real-time.

The class of inverse problems being considered in this article consist of an ordinary differential equation (ODE) of degree m of the form

$$a_m(x)\,y^{(m)} + a_{m-1}(x)\,y^{(m-1)} + \ldots + a_1(x)\,y' + a_0(x)\,y = g(x), \tag{1}$$

where y is a function of x, $y^{(i)}$ is the notation for the ith derivative of y with respect to x, $a_i(x)$ are the coefficient functions and $g(x)$ is the forcing function. Additionally, a minimum of m independent initial-, inner- or boundary values are required to ensure that there is a unique solution to the equation. The n measurements, forming the vector $\boldsymbol{g}$, correspond to discrete samples of the forcing function $g(x)$. The n measurements may emanate from n sensors forming a spatial array or from a time sequence of n measurements from one single sensor. In this class of problems, the forcing function $g(x)$, the input, is considered to be perturbed, since it is formed from measurements which are subject to noise. Only the forcing function $g(x)$ changes from one measurement to the next. The task is to recompute $y(x)$ for each new measurement $g(x)$. This type of problem occurs, for example, in the monitoring of structures [Burdet and Zanella 2002; Golser 2010; Harker and O'Leary 2013a].

The new method can, however, deal with overconstrained systems, i.e., there are p independent constraints whereby $p > m$. The initial-, inner- or boundary values correspond to constraints on the function value $y(x)$ or its derivatives $y^{(i)}(x)$ at specific x locations. Both Dirichlet and Neumann boundary conditions are special cases of such constraints. The nature of the constraints determines if the system is considered to be an initial value (IVP) or boundary value (BVP) or inner value problem.

One peculiarity of this class of inverse problems is: that the abscissae, i.e., the positions where the solutions are required, is determined by the measurements; these positions are called the nodes. In the case of a chain of sensors, the physical position of the sensor corresponds to the abscissae x. In temporal sequences, it is the time points

of the individual measurements which define the abscissae. Consequently, we are not free to select the positions of where the ODE is to be solved. This precludes the use of variable step size algorithms. A further consequence is that a general framework for this type of inverse problem has to be capable of computing the solution for arbitrary nodes.

4. THEORY OF ORDINARY DIFFERENTIAL EQUATIONS

Some preliminary theory is required if an objective evaluation of previous work is to be performed. The numerical solution of an inverse problem requires the discrete approximation of a continuous system. Consequently, we can derive properties of the continuous operations which must be fulfilled by the corresponding discrete operators. We first define the continuous domain differential operator D such that, $D^{(i)} y \equiv y^{(i)}$. Most commonly, the discrete implementation of the differentiating matrix is implemented using polynomial interpolation. The properties of D with respect to a polynomial are essential to the desired behavior of numerical differentiation. Defining a power series approximation for y with coefficients c_i,

$$y = \sum_{i=0}^{m} c_i\, x^i. \tag{2}$$

Applying the differential operator D yields,

$$y' = D\, y = \sum_{i=0}^{m} i\, c_i\, x^{i-1} = \sum_{i=1}^{m} i\, c_i\, x^{i-1}. \tag{3}$$

By definition of the derivative, the constant portion of the polynomial differentiates to zero, hence the constant coefficient c_0 vanishes. We assume that D is composed of formulae which are consistent, in the sense that in the limit they define a derivative. If this is the case then the matrix D should satisfy the following properties, such that D is a consistent discrete approximation to the continuous operator D:

(1) The matrix D must be rank-1 deficient; i.e, its null space is of dimension one.
(2) The null space of D must be spanned by the constant vector $\boldsymbol{1}\,\alpha$; equivalently, the row-sums of D are all zero,

$$\mathsf{D}\,\boldsymbol{1}\,\alpha = \boldsymbol{0}. \tag{4}$$

These conditions ensure that the differentiating matrix D is consistent with the continuous domain definition of the derivative. Given that, interpolating polynomials are unique, the formula for the derivative should be independent of the particular polynomials chosen for interpolation. However, differences do lie in the numerical behavior of different formulas; regardless, a given set of nodes, $\boldsymbol{x}$, should uniquely define the differentiating matrix of a given polynomial degree of accuracy.

For the purpose of treating ODEs, we use the general notion of a linear differential operator [Lanczos 1997]. Specifically, by substituting the continuous differential operator D for the differentials $y^{(i)}$ in Eqn. (1) yields,

$$a_m(x)\, D^{(m)}\, y + a_{m-1}(x)\, D^{(m-1)}\, y + \ldots + a_1(x)\, D\, y + a_0(x)\, y = g(x). \tag{5}$$

Factoring y to the right yields,

$$\left\{ a_m(x)\, D^{(m)} + a_{m-1}(x)\, D^{(m-1)} + \ldots + a_1(x)\, D + a_0(x) \right\} y = g(x). \tag{6}$$

The linear differential operator L for the continuous equation can now be defined as,

$$L \triangleq a_m(x)\, D^{(m)} + a_{m-1}(x)\, D^{(m-1)} + \ldots + a_1(x)\, D + a_0(x). \tag{7}$$

Consequently, Eqn. (1) is written as,

$$\boxed{L\,y = g(x).} \tag{8}$$

5. OVERVIEW OF NUMERICAL ODE SOLVERS

The *Taylor matrix* uses the known analytical relationship between the coefficients, s, of a Taylor polynomial and those of its derivatives, $\dot{s}$, to compute a differentiating matrix D for the solution of ODEs [Kurt and Cevik 2008]. The matrix D together with the matrix of basis functions arranged as the columns of the matrix B are used to compute numerical solutions to the differential equations. The method of the Taylor matrix was extended to the computation of fractional derivatives [Keskyn et al. 2011]. The most serious problem associated with the Taylor matrix approach is that it requires the inversion of the Vandermonde matrix, a process which is numerically unstable. The errors in the differentiating matrix are strongly dependent on the degree of the polynomial, i.e., the number of nodes and the node placement.

A *Chebyshev matrix* approach was presented by Sezer [Sezer and Kaynak 1996] and others [Welfert 1997; Weideman and Reddy 2000; Driscoll et al. 2008; Jewell 2013]. The approach is fundamentally the same as for the Taylor matrix, whereby the Chebyshev polynomials are used as an alternative to geometric polynomials. The advantage of defining polynomials on the Chebyshev points is that they deliver stable polynomials and differentials. The main disadvantage, however, is that the numerical solution to the differential equations is restricted to the locations of the Chebyshev points; this lacks the generality needed for inverse problems[2] being considered here.

Synthesizing differentiating matrices for arbitrary nodes is an issue one might assume has been sufficiently dealt with in literature. However, a closer examination of literature and textbooks shows that some clarification is still necessary. Most books on *spectral and pseudo-spectral techniques*, e.g., [Fornberg 1998], approach differentiation matrices from the view point of simulation and do not consider the connotations of inverse problems. In a simulation, it is in general possible to select the position of the nodes, so that they are well suited to the solution method, e.g., it is possible to use either the Chebyshev or Legendre collocation nodes. This luxury is not given with inverse problems; the placement of the sensors may be arbitrary and or the time points for which solutions are required are evenly spaced. Consequently, it is necessary to generate differentiating matrices for truly arbitrary nodes.

There are a number of papers [Welfert 1997; Weideman and Reddy 2000] which explicitly claim to compute *differentiating matrices using global methods for arbitrary nodes* and there are some toolboxes which suggest this is possible [Jewell 2013]. The published code for all these methods generate degenerate differentiating matrices with null spaces of dimensions higher than one. That is, they do not fulfill the prerequisites defined in Section 4. In contrast, the *local polynomial approximation to differentiation* [Savitzky and Golay 1964] with correct end-point formulas [Burden and Faires 2005] generates a consistent matrix. The poor behavior of high order polynomial interpolation and differentiation is due to Runge's phenomenon, which will be always be present due to the uniqueness of interpolating polynomials; hence, approximations of relatively low degree are preferable to global approaches. The published methods [Welfert 1997; Weideman and Reddy 2000] work reliably only for very small

---

[2]This is not dismissing the Chebyshev methods, it simply points out that they are limited in their applications. Furthermore, the methods in this article work for truly arbitrary node placements. Consequently, the Chebyshev polynomials are only a special case.

problems[3], $n \leq 10$; this is not sufficient to address most real inverse problems encountered in engineering applications. We conclude that global techniques for computing differentiating matrices are not applicable to large scale inverse problems.

Finite difference methods [Strikwerda 2004; Smith 1985] rarely deal with higher degree approximations, typically 3 or 5 point formulas are used. The issue of correct end point formulas is sacrificed for the advantage of band diagonal matrices. In general, these techniques deal with Dirichlet and possibly Neumann boundary conditions. However, they provide no method of implementing general boundary conditions of the form

$$D^{(i)} y(x_j) = d, \tag{9}$$

where $D^{(i)}$ represents the ith derivation of y evaluated at the point $x = x_j$ with the value d. There may be $p \geq m$ such constraints.

A *new matrix approach for the solution of inverse problems*, associated with monitoring of structures using inclinometers, was presented [O'Leary and Harker 2012] and generalized in [Harker and O'Leary 2013a]. It was proven that ODEs can be formulated as a least squares problem with linear constraints, of the form:

$$\mathsf{L}\, \boldsymbol{y} = \boldsymbol{g} \quad \text{subject to} \quad \mathsf{C}^{\mathrm{T}}\, \boldsymbol{y} = \boldsymbol{d}, \tag{10}$$

whereby L is the discretized linear differential operator, $\boldsymbol{y}$ is the solution vector sought (function values), $\boldsymbol{g}$ is the discrete forcing function (measurement values), C defines the type of constraints and $\boldsymbol{d}$ are the values of the constraints. The least squares solution makes the method suitable for problems where the forcing function $g(x)$ is perturbed.

The continuous linear differential operator L in Eqn. (7) is discretized as the matrix L, such that

$$\mathsf{L} \triangleq \mathsf{A}_m\, \mathsf{D}_m + \mathsf{A}_{m-1}\, \mathsf{D}_{m-1} + \ldots + \mathsf{A}_1\, \mathsf{D} + \mathsf{A}_0, \tag{11}$$

where $\mathsf{A}_i = \mathrm{diag}(a_i(\boldsymbol{x}))$, the matrix D_i is a local discrete approximation with support length l_s to the continuous differential operator $D^{(i)}$. Care is taken to implement the correct end-point formulas, ensuring the degree of approximation is constant for the complete support. The details of generating these matrices can be found in [Harker and O'Leary 2013a], as can the explanation for the generation of the constraints $\mathsf{C}^{\mathrm{T}}\, \boldsymbol{y} = \boldsymbol{d}$. Furthermore, MATLAB toolboxes are available [Harker and O'Leary 2013b; Harker and O'Leary 2013c] for all the functions required in this article.

6. SOLVING THE INVERSE PROBLEM

Previously the problem in Eqn. (10) was solved using an efficient and accurate solution which is found in [Golub and Van Loan 1996, Chapter 12]. In this paper we take a different approach to partitioning the numerical computations, which takes advantage of the fact that the inverse problem is to be solved repeatedly. Fundamentally, the new approach delivers exactly the same explicit solution; however, through the new partitioning of the computation it is possible to ensure that the numerical work $W(n)$ and the memory required are run-time are known exactly in advance. Consequently, an exact upper-bound for the execution time can be determined, this by definition makes the solution suitable for real-time applications[4].

The computation of the solution is separated into two portions:

---

[3]This can be verified by running the available code with $n = 20$. Testing the resulting D matrix or its singular values reveals that a null space of higher dimension is present. As a consequence, the matrix does not fulfill the necessary prerequisites.

[4]A *real-time system* is defined as any information processing activity or system which has to respond to externally generated input stimuli within a finite and specified period [Young 1982].

(1) The preparatory computations which can be performed offline. They are characteristic for the equation being solved and change neither with the acquisition of new measurement data, nor with new values for the boundary conditions. These computations need not be performed on the embedded system and may be computed with higher precision arithmetic on a host system if necessary.
(2) The online computation, which must be performed repeatedly with each new set of sensor data. This is the solution which is computed explicitly on the embedded system in real-time.

6.1. Preparatory Computations

The constraints on the solution are defined by,

$$\mathsf{C}^{\mathrm{T}}\,\boldsymbol{y} = \boldsymbol{d}. \tag{12}$$

Each column of C, together with the corresponding row of $\boldsymbol{d}$, defines a constraint. Consequently, $p = \mathrm{rank}\left\{\mathsf{C}\right\}$ is the number of linearly independent constraints. Additionally, the constraints must be consistent, i.e., $\boldsymbol{d} \in \mathrm{range}\left\{\mathsf{C}^{\mathrm{T}}\right\}$. A minimum of $p \geq m$ constraints are required to ensure a unique solution to an ODE of degree m. We now define the matrices: P, such that $\mathrm{range}\left\{\mathsf{P}\right\} = \mathrm{range}\left\{\mathsf{C}\right\}$, i.e., P is the Moore-Penrose pseudo inverse of C^{T}, hence $\mathsf{P} = \{\mathsf{C}^{\mathrm{T}}\}^{+}$; F, an orthonormal basis function set for the null-space of C^{T}, i.e., $\mathsf{F}^{\mathrm{T}}\mathsf{F} = \mathsf{I}$ and $\mathrm{range}\left\{\mathsf{F}\right\} = \mathrm{null}\left\{\mathsf{C}^{\mathrm{T}}\right\}$; and $\mathsf{H} \triangleq \mathsf{P} + \mathsf{F}\mathsf{R}$, where R is an arbitrary matrix. In this manner the solution for $\boldsymbol{y}$ can be parameterized as,

$$\boldsymbol{y} = \mathsf{H}\,\boldsymbol{d} + \mathsf{F}\,\boldsymbol{\beta}, \tag{13}$$

where $\boldsymbol{\beta}$ is the parameter vector. It is important to realize that neither H nor F are unique. Any function which fulfills the constraints is a valid selection for $\boldsymbol{y}_c$. A function $\boldsymbol{y}_c$ which fulfills the constraints can be defined as,

$$\boldsymbol{y}_c \triangleq \mathsf{H}\,\boldsymbol{d}, \tag{14}$$
$$= \{\mathsf{P} + \mathsf{F}\mathsf{R}\}\,\boldsymbol{d}. \tag{15}$$

The matrix R is arbitrary, consequently the values can be selected so that $\boldsymbol{y}_c$ fulfills additional conditions without altering the solution for $\boldsymbol{y}$. In Fig. 1 three different solutions for the constraints $y(0) = 1$ and $y(1) = 0$ are shown, to demonstrate this fact. It may be advantageous for a specific problem to select a particular solution for $\boldsymbol{y}_c$ which has desirable properties; for example, when solving the ODE for a cantilever it may be appropriate to select a polynomial solution for $\boldsymbol{y}_c$, since the solution to the ODE is known to be a polynomial. More formally: the matrix $\mathsf{H} = \left\{\mathsf{C}^{\mathrm{T}}\right\}^{-}$ is a generalized inverse [Ben-Israel and Greville 2003] of C^{T}. A generalized inverse A^{-} of a matrix A fulfills the condition,

$$\mathsf{A}\,\mathsf{A}^{-}\,\mathsf{A} = \mathsf{A}. \tag{16}$$

The Moore-Penrose pseudo inverse is the particular generalized inverse, where $\mathsf{R} = 0$; it yields an inverse which minimizes the 2-norm of the solution vector; alternatively, a QR decomposition can be used to compute a generalized inverse which leads to a solution vector with a minimum number of nonzero entries. The selection of an appropriate solution for $\boldsymbol{y}_c$ is more important when solving inverse problems, since it has implications for the implementation of regularization.

The orthonormal basis functions F for the null-space of C^{T} are also not unique. They can be obtained directly from C^{T} by applying QR decomposition and partitioning Q according to the $\mathrm{rank}\left\{\mathsf{R}\right\}$. Alternatively, constrained basis functions, e.g. constrained polynomials, can be used to implement a set of orthogonal basis functions F, Fig. 2 shows an example of such admissible functions for the constraints $y(0) = 1$, and

Fig. 1: Three different solutions to the constraints $y(0) = 1$ and $y(1) = 0$.

Fig. 2: The first four discrete orthogonal constrained polynomials for the constraints $y(0) = 1$ and $y(1) = 0$, i.e., the first 4 basis functions from F.

$y(1) = 0$. In the case of inverse problems constrained basis functions offer a method of implementing spectral regularization [O'Leary and Harker 2012]. The MATLAB library required to generate discrete orthogonal constrained polynomials is available at [Harker and O'Leary 2013b].

$$\mathsf{C}^{\mathsf{T}}\mathsf{F} = 0 \quad \text{and hence} \quad \mathsf{C}^{\mathsf{T}}\mathsf{F}\boldsymbol{\beta} = \boldsymbol{0}. \tag{17}$$

Substituting Eqn. (13) for y in $\mathsf{L}\,y = g$ now yields an unconstrained algebraic equation for the ODE,

$$\mathsf{L}\{\mathsf{H}\,d + \mathsf{F}\,\boldsymbol{\beta}\} = g. \tag{18}$$

In the class of inverse problems being considered in this article, the forcing function g is formed from the measurement values which are perturbed, i.e., is subject to noise. Consequently, the solution of Eqn. (18) is formulated as a least squares problem to obtain the unique global minimum of

$$\min_{\boldsymbol{\beta}} \|\mathsf{L}\,\mathsf{H}\,d + \mathsf{L}\,\mathsf{F}\,\boldsymbol{\beta} - g\|_2^2. \tag{19}$$

The least squares approach has been selected since it delivers a maximum likelihood solution in the case that g is perturbed by Gaussian noise. A further advantage of the global least squares formulation is that the solution has no implicit direction of integration. Avoiding a direction of integration eliminates the problem of accumulation of errors, as are typical with IVP approaches such as Runge-Kutta. Additionally, the least squares approach yields a solution which is globally minimum with respect to all errors in Eqn. (19), i.e., it is also minimizing the consequences of the errors in the numerical computations. Consequently, the method is suitable for solving both perturbed and unperturbed problems. Now solving the minimization problem defined by Eqn. (19) yields,

$$\boldsymbol{\beta} = \{\mathsf{L}\,\mathsf{F}\}^+ \{g - \mathsf{L}\,\mathsf{H}\,d\} + \mathsf{K}\,\boldsymbol{\gamma}, \tag{20}$$

where K is an orthonormal vector basis set for the null-space of $\mathsf{L}\,\mathsf{F}$, i.e., $\mathsf{K}^{\mathsf{T}}\mathsf{K} = \mathsf{I}$ and $\operatorname{span}\{\mathsf{K}\} = \operatorname{null}\{\mathsf{L}\,\mathsf{F}\}$. This equation is now expanded into three relevant terms,

$$\boldsymbol{\beta} = \{\mathsf{L}\,\mathsf{F}\}^+ \, g - \{\mathsf{L}\,\mathsf{F}\}^+ \mathsf{L}\,\mathsf{H}\,d + \mathsf{K}\,\boldsymbol{\gamma}. \tag{21}$$

A non-empty vector basis set K indicates that the linear differential operator L is not sufficiently constrained to ensure a unique solution, i.e., there is no unique solution to

the problem being posed. The requirement for a unique solution is

$$\text{rank}\begin{bmatrix} \mathsf{L} \\ \mathsf{C}^\mathsf{T} \end{bmatrix} = n, \tag{22}$$

where n is the number of nodes. This is a method of determining if the problem is well defined. Alternately, the singular values of the matrix can be used to determine if the problem is numerically well posed. We will now assume that the problem is well posed: with this, the term involving γ vanishes. Now back-substituting for β in Eqn. (13), yields,

$$\boldsymbol{y} = \mathsf{F} \left\{ \mathsf{L}\,\mathsf{F} \right\}^+ \boldsymbol{g} + \mathsf{H}\,\boldsymbol{d} - \mathsf{F} \left\{ \mathsf{L}\,\mathsf{F} \right\}^+ \mathsf{L}\,\mathsf{H}\,\boldsymbol{d}. \tag{23}$$

Defining the following abbreviations:

$$\mathsf{M} \triangleq \mathsf{F} \left\{ \mathsf{L}\,\mathsf{F} \right\}^+ \qquad \text{and} \qquad \mathsf{N} \triangleq \left\{ \mathsf{I} - \mathsf{M}\,\mathsf{L} \right\} \mathsf{H}, \tag{24}$$

yields,

$$\boldsymbol{y} = \mathsf{N}\,\boldsymbol{d} + \mathsf{M}\,\boldsymbol{g} \tag{25}$$

$$= \boldsymbol{y}_h + \boldsymbol{y}_p. \tag{26}$$

The homogeneous portion of the solution $\boldsymbol{y}_h = \mathsf{N}\,\boldsymbol{d}$ is only dependent of the constraint values and the particular solution $\boldsymbol{y}_p = \mathsf{M}\,\boldsymbol{g}$ is only dependent on the forcing function, i.e., the measurement values. In the problems considered in this paper the constraint values do not change from one measurement to the next. Consequently, $\boldsymbol{y}_h$ can be computed a-priori and made available as a vector of constraints for the run-time computation.

6.2. Run-Time Computation

Both M and $\boldsymbol{y}_h$ are computed a-priori. A standard PC with higher precision arithmetic can be used for these computations. In this manner, the final errors in M and $\boldsymbol{y}_h$ are dominated by the rounding effects of converting the double precision values to single precision for the embedded computation, should the embedded system not support double precision arithmetic. Substituting M and $\boldsymbol{y}_h$ into Eqn. (23) yields,

$$\boxed{\boldsymbol{y} = \mathsf{M}\,\boldsymbol{g} + \boldsymbol{y}_h.} \tag{27}$$

Only the vector of sensor data $\boldsymbol{g}$ changes with each measurement. Consequently, the run-time solution of the inverse problem is reduced to a single matrix multiplication and a vector addition. This makes the repeated computation of the solution very efficient. Given n measurement values, the computational cost $W(n)$ is,

$$W(n) = 2n^2. \tag{28}$$

For example, a sensor chain with $n = 21$ inclinometers would require $W(n) = 882$ FLOPs to solve the inverse problem. The computation effort reduces to

$$W(n) = n^2. \tag{29}$$

if the processor architecture being used supports a multiply-accumulate[5] operation. Both the exact number of FLOPs and memory required are known prior to the run-time computation. This enables the computation of a strict upper bound for the execution time of the equation. Consequently, the method is, by definition, suitable for real-time

---

[5]See for example the specifications for the ARM Cortex Microcontroller Software Interface Standard (CMSIS) at http://www.arm.com/products/processors/cortex-m/.

applications. The computational complexity, $\mathcal{O}(n^2)$, is independent of the placement of the nodes, the equation being solved and the support length selected.

6.3. Error Estimation and Confidence Interval

There is uncertainty associated with the solution of any inverse problem. Regularization is used to control this uncertainty. The aim now is to quantify the uncertainty associated with the solution presented here. The are two primary sources of possible errors involved in the computation of y in Eqn. (27):

(1) Errors in the values contained in M and y_h. The numerical testing (see Section 7) demonstrates that these errors are negligible, when computed in double precision, in comparison to realistic perturbations of g. For some embedded systems it is necessary to reduce the values from double to single precision. Software-in-the-loop (SIL) and processor-in-the-loop (PIL) testing are used to quantify these errors. In Section 9 it is experimentally verified that these errors can be ignored.
(2) The errors at run-time are dominated by the perturbations of g, these errors are orders of magnitude larger than the residual numerical errors in M and y_h. Consequently, only errors in g are considered for the covariance propagation. There is also an approximation error in M based on the choice of the interpolating functions. These may not be insignificant depending on the nature of the solution y.

The following computation assumes that only the forcing function g is subject to Gaussian perturbation. The covariance Λ_y associated with the computation of y using Eqn. (27), can be explicitly [Brandt 1998] calculated as

$$\Lambda_y = M \Lambda_g M^T, \tag{30}$$

where Λ_g is the covariance of the forcing function. In practical applications we determine the magnitude of the noise component for each sensor, using dedicated noise measurements. In this case, and assuming that the noise is independent identically distributed (i.i.d.) Gaussian noise with standard deviation σ_g, then,

$$\Lambda_g = \sigma_g^2 \, I, \tag{31}$$

where σ_g is a measured value. Substituting this into Eqn. (30) yields,

$$\Lambda_y = \sigma_g^2 \, M \, M^T. \tag{32}$$

An upper-bound estimate within a given confidence interval for the vector of standard deviations for y, is computed as,

$$\boxed{\sigma_y = \sigma_g \, s,} \tag{33}$$

where $M = (m_{ij})$ and the individual elements of the unscaled standard deviation s are $s_i = (\sum_{j=1}^n m_{ij}^2)^{1/2}$, i.e. the square root of the diagonal elements of $(M \, M^T)$. This term can be computed a-priori; consequently, the run-time computational complexity for determining the standard deviation of each solution point is $\mathcal{O}(n)$.

Alternatively, the error vector ϵ can be computed for each measurement as the difference between the forward and inverse problem, i.e.,

$$\epsilon = g - L \, (M \, g + y_h) \,. \tag{34}$$

A Kolmogorov-Smirnov test can be applied to ϵ to determine if it is Gaussian. This yields additional information on the suitability of the model for the specific measurement. Given the standard deviation, the confidence interval with a specific degree of certainty is computable via the inverse Student-t distribution [Brandt 1998].

7. MODEL-IN-THE-LOOP TESTING

The aim of this section is to verify the numerical accuracy and efficiency of the new method on a PC, embedded testing is presented later. We have chosen to solve unperturbed problems for the first tests, since these enable the comparison with analytical solutions and with standard engineering approaches such as Runge-Kutta methods[6]. The unperturbed tests enable the separation of the errors involved in computing M and y_h from those resulting from the perturbation of g. It is difficult to define a truly objective method of comparing solution approaches which are fundamentally different[7]. Each approach has its own weaknesses and strengths. The tests have been devised to reflect, as close as possible, the conditions which are to be expected from the application specific method.

7.1. Test A: Initial Value Problem 1

The ODE (details can be found in [Adams 2006]) is a third order ($m = 3$) non-homogeneous ODE with constant coefficients a_i and $p = 3$ constraints. The equation is

$$y^{(3)} + 3\,y'' + 3y' + y = 30\,\mathrm{e}^{-x} \quad \text{given} \tag{35}$$
$$y(0) = 3, \quad y'(0) = -3, \quad y''(0) = -47$$

in the interval $0 \le x \le 8$. The analytical solution to this equation is

$$y(x) = (3 - 25\,x^2 + 5\,x^3)\,\mathrm{e}^{-x}. \tag{36}$$

In the case of the inverse problems being addressed, the position of the solution points is determined by the measurement. To simulate this condition, the `ode45` solver [Shampine and Reichelt 1997] in MATLAB has been used to solve this differential equation. This is a variable step size method which yields both a vector of abscissae x consisting of $n = 77$ points and the solution vector y. Exactly those $n = 77$ points on the abscissae and a support length $l_s = 9$ was used for the test of the new method. Using the `ode45` solver in addition to the analytical solution enables the comparison of the new method with well established techniques. The results of the three computations are shown in Fig. 3. The residual errors, i.e., the difference between the analytical solution, the new method and the `ode45` solutions are shown in Fig. 4. The 2-norm of the residual errors $|\epsilon|_2$ and the computation time for $k = 10000$ iterations[8] for the solution of the ODE are given in Table I for the `ode45` method as well as the new method. The first observation is that the residual numerical errors for the new method

Table I: The 2-norm of the residual errors $|\epsilon|_2$ and the computation time for $k = 10000$ iterations for the solution of the ODE, for the `ode45` method and new method (New). These computations were performed with an Intel Core 2 Duo CPU P8600 at 2.4 [GHz] with 2.9 [GB] RAM.

method	$|\epsilon|_2$	time ($k = 10000$)
`ode45`	$1.79\,10^{-3}$	29.823728 [s]
New	$1.14\,10^{-7}$	0.061681 [s]

---

[6]The MATLAB `ode45` implementation of a Runge-Kutta method was used for this purpose.

[7]To support independent verification of our results, we have made the MATLAB code available which we used to generate all the results presented in this section, see http://www.mathworks.com/matlabcentral/fileexchange/45947.

[8]It is not the absolute times which are important, since they will change from one platform to another. It is the relative speed which shows the potential performance of the new method.

Fig. 3: The plot shows the analytic solution from Eqn. (36), the solution with the new method with $l_s = 9$ as well as the solution by a Runge-Kutta `ode45` method. All solutions are evaluated at exactly the $n = 77$ points provided by the `ode45` method.

Fig. 4: Residual errors: difference of the new numerical solution vs. the analytical solution and the solution by a Runge-Kutta `ode45` method vs. the analytical solution. The new method is approximately 4-orders of magnitude more accurate than the `ode45` method, for exactly the same abscissae.

are approximately 4-orders of magnitude smaller than with the `ode45` method and the computation is almost 500 times faster. Reducing the error bound for the Runge-Kutta method will improve the numerical accuracy, but at the expense of computational effort. In order to reduce the error by 4-orders of magnitude in the Runge-Kutta solution, an even higher degree Runge-Kutta method must be formulated, which in turn would require unreasonably high computational effort. The very small errors are significant: since, when computing the confidence interval for the solution, they can be neglected when they are small in comparison with the perturbations of the forcing function.

The comparison of speed is somewhat subjective, since we have no insights into how much function-call-overhead is involved in the MATLAB implementation; nevertheless, it does show the potential speed of the new approach. This test demonstrates the ability of the new method to compute solutions to the ODE at arbitrary given nodes with a very high accuracy.

7.2. Test B: Alternative Node Placement for Initial Value Problem 1

In this test the same ODE is solved as in Test A, however, a reduced number of $n = 20$ evenly spaced nodes has been selected for the new method since systems sampling in time in general use even spacing. The results are shown in Fig. 5 and 6. The new method achieves the same solution quality, in terms of accuracy, as the `ode45` method, however with a significantly reduced number of nodes. This corresponds to an accurate solution of the inverse problem with a small number of sensors.

7.3. Test C: Initial Value Problem 2

The second example is a second order ($m = 2$) ODE with variable coefficients $a_i(x)$ and $p = 2$ constraints. This demonstrates the ability of the method to deal with variable coefficients and with solutions which are irrational functions. The equation is,

$$2\,x^2\,y'' - x\,y' - 2y = 0 \quad \text{given} \tag{37}$$
$$y(1) = 5, \quad y'(1) = 0$$

Fig. 5: The plot shows the analytic solution from Eqn. (36), the solution with the new method with $l_s = 9$ and $n = 20$ as well as the solution by a Runge-Kutta `ode45` method with $n = 77$.

Fig. 6: Residual errors: difference of the new numerical solution with $n = 20$ vs. the analytical solution and the solution by a Runge-Kutta `ode45` method with $n = 77$ vs. the analytical solution.

Fig. 7: The plot shows the analytic solution from Eqn. (38), the solution with the new method with $l_s = 15$ and $n = 69$ evenly placed nodes as well as the solution by a Runge-Kutta `ode45` method with $n = 69$ variably placed nodes.

Fig. 8: Residual errors: difference of the new numerical solution vs. the analytical solution and the solution by a Runge-Kutta `ode45` method vs. the analytical solution.

in the interval $1 \le x \le 10$. The analytical solution to this equations is

$$y(x) = x^2 + \frac{4}{\sqrt{x}}. \tag{38}$$

The solution's appearance would not suggest that this is a demanding problem. However, the analytical solution is the sum of a polynomial and an irrational function. Computing good estimates for the derivatives of such functions can require a high degree of polynomial approximation. The solution obtained using the new method, the analytical solution and the result of the `ode45` solver are shown in Fig. 7. The high density of nodes at the start of the interval produced by the `ode45` method indicates that the method required disproportionately many steps for finding a solution with sufficient accuracy. The new method is once again more accurate than the `ode45` solver.

7.4. Test D: Selecting a Support Length for Initial Value Problem 2

In this test, the same ODE is solved as in Test C. As pointed out in Section 5, there is an issue in selecting the support length l_s (or degree) of the local approximation for

the differentiating matrix D. This matrix and the linear differential operator L have been implemented for all odd support lengths in the range $3 \leq l_s \leq 25$ and the IVP was solved for each of these implementations. The relative error was computed for the corresponding solutions as

$$\epsilon(l_s) = \frac{|\boldsymbol{y}_a - \boldsymbol{y}|_2}{|\boldsymbol{y}_a|_2}, \tag{39}$$

where $\boldsymbol{y}_a$ is the sampled analytical solution and $\boldsymbol{y}$ is the solution computed with the new method. The $\log_{10}(\epsilon)$ vs. l_s is shown in Fig. 9. This result shows that there is a minimum in the relative error for $l_s = 15$, indicating that there is a justification for implementing local approximation to derivatives for specific problems with high numbers of nodes[9]. The dependence of ϵ on l_s is a function of the equation being solved. There will be no solution that is optimal for all cases. With the proposed method the necessary l_s is determined during the preparatory computations and not at run-time.

Fig. 9: The relative error ϵ has a minimum for $l_s = 15$.

7.5. Test E: Inverse 3-Point Boundary Value Problem

The following test is an over constrained first order ($m = 1$) 3-point inverse BVP[10]; however, with $p = 4$ constraints at 3 locations on the abscissae. It belongs to the class of inverse multi-point BVPs[11]. The constraints implemented are both: homogeneous and non-homogeneous; as well as Dirichlet and Neumann boundary conditions. This example has been chosen to demonstrate the numerical efficiency and behavior of the method with respect to a perturbed inverse BVP. Furthermore, it demonstrates the ability of the algebraic framework to deal with generalized constraints. Synthetic data is produced for a function and its analytic derivatives, in this manner the result of the reconstruction can be compared with the function from which the data was derived.

The problem being considered is to reconstruct a curve $\boldsymbol{y}$ from multiple local measurements of the curve's gradients $\boldsymbol{g}$ while fulfilling a set of constraints $\mathsf{C}^{\mathsf{T}}\boldsymbol{y} = \boldsymbol{d}$, which are not restricted to the ends of the support, i.e., inner constraints are also present.

---

[9]Many books [Burden and Faires 2005; Lapidus and Pinder 1999; Strikwerda 2004] discuss the possibility of implementing approximations of higher degree; however, they never actually show comparative numerical results for practical problems.

[10]At least one of the constraints is interior, since the constraints are not restricted to the boundaries.

[11]Although we have been able to find a number of publications on methods relating to multi-point BVPs [Welsh and Ojika 1980; Agarwal et al. 2003], there is very little literature available on inverse multi-point BVPs, e.g. [Kurylev 1993]. There are no general approaches available at the present time.

Fig. 10: First order ($m = 1$) constrained test function and its analytical derivatives: this function fulfills all $p = 4$ constraints specified in Eqn. (41) and (42). The homogeneous and non-homogeneous Dirichlet and Neumann constraints are marked at the three locations $[0, 0.7895, 1]$ in the top and bottom plots respectively.

Furthermore, the measurements are perturbed by white noise. The ODE is

$$y'(x) = g(x) + \epsilon, \tag{40}$$

where ϵ is the error caused by the forcing function's perturbation. This equation is subject to the homogeneous and non-homogeneous Dirichlet boundary conditions,

$$y(0.7895) = 0 \quad \text{and} \quad y(1) = -0.1 \tag{41}$$

as well as the non-homogeneous and homogeneous Neumann boundary conditions,

$$y'(0) = 1 \quad \text{and} \quad y'(1) = 0. \tag{42}$$

A synthetic test function which fulfills these conditions was generated by combining an arbitrary polynomial of 4th degree,

$$y(x) = 1.1x^4 + 0.4x^3 + 0.5x^2 - 1.2x - 0.3, \tag{43}$$

with the four constraints, this yields an 8th degree polynomial[12] which also fulfills the constraints:

$$\begin{aligned} y_c(x) = & -0.46985x^8 + 0.41127x^7 + 0.34891x^6 + 0.03827x^5 \\ & + 1.0323x^4 - 1.5886x^3 - 0.88426x^2 + x + 0.011895. \end{aligned} \tag{44}$$

The first derivative of Eqn. (44) can be computed analytically, making it a suitable test function for constrained curve reconstruction from gradients. The function $g(x)$, its analytical gradient $g'(x)$ and the constraints are shown in Fig. 10.

The analytical gradient is evaluated at $n = 21$ points[13]. Then in a Monte Carlo simulation, with $k = 10000$ iterations, the gradients are perturbed by artificial Gaussian noise with a standard deviation of 1% of the maximum value of $Dy(x)$. For each simulation, a reconstruction is performed with the appropriate M and y_h and the statistics are computed. The result of solving this problem using the proposed method is shown in Fig. 11, together with the error bars corresponding to the standard deviation of the reconstructed values observed in the Monte Carlo simulation. The error bars have been magnified by a factor of 10 to increase the visibility.

In Fig. 12, the bias of the reconstruction is shown, i.e., the difference between the analytical solution and the mean of the Monte Carlo simulations. Additionally, the

---

[12]The theory behind this computation can be found in [Harker and O'Leary 2013d].

[13]For example, these would correspond to the positions of the inclinometers on a structure being monitored.

Fig. 11: Comparison of the analytical solution and the result of the reconstruction from the perturbed gradients. The two Dirichlet boundary conditions are marked on the reconstruction, the two Neumann conditions are not shown. The error bars correspond to the standard deviations, i.e., an estimate for the 68.3% confidence interval. They are obtained from a Monte Carlo simulation with $k = 10000$ iterations. The error bars have been magnified by a factor of 10 to increase the visibility.

Fig. 12: Top: Bias of the reconstruction, i.e., the difference between the mean reconstruction from the $k = 10000$ Monte Carlo iterations and the analytical solution (the result is scaled by 10^5 to make the error visible.). Bottom: The standard deviation of the reconstruction as predicted (P) by the covariance propagation according to Eqn. (33) and the results from the Monte Carlo simulations (MC) (the result is scaled by 10^3).

standard deviation of the result as predicted by Eqn. (33) and the results of the Monte Carlo simulation are compared. These results verify that the solution is, for all intents and purposes, bias free and the predicted uncertainty is correct. The method has been successfully applied to an inverse three-point boundary value problem, with two Dirichlet and two Neumann boundary conditions. Not only is the problem solved, but in addition the uncertainty of the solution is delivered by the new method.

7.6. Summary of the Numerical Testing

The implications of the above test can be summarized as follows:

(1) The new method is capable of solving both perturbed and unperturbed inverse problems, including initial-, inner- and boundary value problems.
(2) The method enables the formulation and solution of problems with constraints on arbitrary derivatives of the solution function. Consequently, both Dirichlet and Neumann boundary conditions can be dealt with. The constraints can be both homogeneous ($d = 0$) and non-homogeneous ($d \neq 0$).
(3) The method exhibits significantly smaller numerical errors than the Runge-Kutta `ode45` approach, while being significantly faster. Reducing the error bound for the `ode45` will improve the numerical accuracy, but at the expense of computational effort. The numerical errors are so small that they can be neglected when solving inverse problems where the perturbation of the forcing function is significant.
(4) The separation of the solution into a preparatory (offline) and run-time (online) computation makes the method suitable for embedding in real-time systems.

8. AUTOMATIC CODE GENERATION

The aim of the code generation is, given the definition of specific inverse problem in terms of a measurement model (see Section 3), to automatically generate the code required to solve the problem on an embedded computing system. That is, the problem is

defined as a symbolic definition of a differential equation together with a suitable set of constraints and a source of data. The system will then solve any ODE, regardless of its nature (IVP, BVP, etc.) from this specification. A similar concept of automatic code generation (ACG) for the embedding of convex optimization was explained in literature [Mattingley and Boyd 2012]. Furthermore, it was shown, that numerical ODE solvers can be deployed on FPGAs using VHDL in [Huang et al. 2013]. During MBD, the system is designed on an abstract model level based on the system's requirements while ensuring the consistency of the system's physical representation. ACG is the task of converting the models and their algorithms to usable code, effectively automating the time-consuming and error-prone process of low-level programming. Basically, there are three low-level target languages for embedded systems: multi-purpose *ANSI-C code*, which is the focus of this article; *hardware description language (HDL)* for field programmable gate arrays (FPGA) or application specific integrated circuits (ASIC); and IEC 61131-3 compliant languages such as *structured text (ST)* for programmable logic controllers (PLC).

Code deployment is the integration of code on the embedded systems. In most cases, the architecture of the development (host) system (x86 or x64) is largely different from the embedded (targeted) system (ARM, ATmel, etc.). There are two approaches for solving this issue *compilation of code on the target system* if an OS and an appropriate compiler is present; or *cross compilation* on the host system via processor virtualization, a popular tool to perform this task is the LLVM compiler infrastructure.

After MBD is complete, so called production code is generated. The process strips out all parameters needed during testing and optimizes the code for performance (low memory usage, high computational speed) or safety (data consistency, robust algorithms).

8.1. Embedded Target Hardware

The goal is to show that even highly abstract and complex mathematical models are deployable on the simplest embedded hardware, demonstrating the scalability of the method. The open-source *Raspberry Pi* or the proprietary *BeagleBone Black* are popular entry-level embedded systems for target programming. The *WAGO PFC-200* is an IEC 61131-3 compliant industrial PLC with open source software. These three systems are based on 32-bit ARM processors and they run an embedded Linux derivative as operating system (OS); in the case of the WAGO device it's a real-time OS. The automated resource management is the main advantage of embedded systems with an OS. The low end of the systems is represented by the fully open source *Arduino Uno* platform. The utilized Atmel 8-bit AVR RISC-based ATmega328 microcontroller has no dedicated OS, the program logic is directly stored on the chip's 32 [kB] flash memory as firmware. The Arduino Uno has been chosen for experimental PIL testing, see Section 9. A laboratory experiment featuring the BeagleBone Black is presented in Section 10.

8.2. MBD Software for Code Generation

Most engineering and scientific software for designing mathematical models has the functionality to automatically generate standard ANSI-C code from its application-specific syntax, e.g., LabVIEW, Maple or Mathematica. This is usually necessary, because most industrial controllers are only programmable with C. A short survey on tools for ACG has been given by Rafique et al. [Rafique et al. 2013]. This article is focused on the usage of MATLAB and its Coder toolbox, because it is the standard software for mathematical MBD. Code generation fully supports linear algebra. Nevertheless, the presented approach is so simple, that a C code parser could be implemented manually without much effort. Two test cases confirmed the correct functionality of

algebraic functions generated by MATLAB Coder: C code was generated for QR decomposition and singular value decomposition. Compiling the code with Microsoft Visual Studio 2010 for the 32-bit host development system delivered an executable program, which successfully validated the code via SIL verification. Deploying the same code on the Arduino Uno confirmed the correct functionality via PIL verification.

model based design and in-the-loop verification for automatic code generation

Fig. 13: MBD is an iterative approach. Each step requires verification to ensure that the (sub)system's requirements are met. In this article, model-in-the-loop (MIL) verifies the correctness of the algebraic framework on model level, software-in-the-loop (SIL) verifies the functional equivalence of the generated C code on code level and processor-in-the-loop (PIL) verifies the correct computation on the employed microcontroller on binary level. This graphic shows the process for the Arduino platform.

8.3. Targeting and Verification Process

The ACG process is completely general, the illustration in Fig. 13 shows the procedure for the Arduino platform. Following steps must be carried out:

Model-in-the-loop (MIL): The system is identified, designed and simulated on abstract model level in an artificial environment, producing MATLAB model code (*.m). This is an efficient way to estimate model parameters with varying configurations. This includes the determination of the optimal support length l_s, computation of the constrained linear differential operator M and the homogeneous solution y_h. Furthermore, changes in the requirements can easily be implemented in this early design stage.

Code generation: The MATLAB Coder toolbox is a sophisticated parser engine. It converts the model code (*.m) into C code (*.c) and the associated header files (*.h).

Code compilation for SIL: MATLAB Coder features the ability to replace model function calls with calls for MATLAB executables (*.mex). Such functions are wrappers around compiled C code, which can be directly called from the MATLAB development environment.

Software-in-the-loop (SIL): The model and the generated C code must be functionally equivalent, i.e., a certain input must deliver the same output on all abstraction layers. This is especially relevant when the target language misses certain features of the

model language, e.g. shorter bit-lenghts of variable types or no support for floating point operations. The consistency of the model must be ensured on all levels.

Code compilation for PIL: The C code (*.c/*.h) is imported into the Arduino IDE. The header (*.h) files must be included in the Arduino project's main (*.ino) file. The C code is cross-compiled for the Arduino platform delivering a (*.hex) file, which is stored directly on the ATmega328's flash memory as firmware.

Processor-in-the-loop (PIL): The code runs on the embedded real-time system. The outcome is not necessarily the same as during simulation, because the hardware platform used during MIL and SIL is different from the PIL target.

9. SOFTWARE- AND PROCESSOR-IN-THE-LOOP TESTING

In Section 7, the viability of the new method was shown during MIL. In this section, the test cases are directly executed on the Arduino Uno for PIL verification. The microcontroller features $2\,[\text{kB}]$ SRAM and a processing power of 16 million instructions per second (MIPS). The 23 general purpose I/O lines, the 6-channel 10-bit A/D converter and the operating voltage of $1.8 - 5.5\,[\text{V}]$ makes it a well suited setup for acquiring and processing sensor data. The problem size must be scaled down in order to fit the Arduino Uno's limited system resources. An Arduino Uno double variable requires $4\,[\text{B}]$ of memory, so theoretically, the ATmega328 chip stores up to 512 double variables in its memory of $2\,[\text{kB}]$. Obviously, operations and other variables also require memory space, therefore the problem size has been shrunk to 10 input signals. This corresponds in means of problem size to the Arduino Uno's 6 analog I/O ports, which are usable to connect sensors to the device. However, the problem classes are still the same.

The constrained linear differential operator M has then a size of (10×10) and the homogeneous solution vector $\boldsymbol{y}_h$ has a size of (10×1). Both, M and $\boldsymbol{y}_h$, are computed a-priori during offline calibration. Only the measurement vector $\boldsymbol{g}$ with size (10×1) changes its values from one measurement to the next. Consequently, the result of the online computation, i.e. the solution vector $\boldsymbol{y}$, is of size (10×1).

9.1. Initial Value Problem 1

The test case in Section 7.1 has been modified to have $n = 10$ evenly spaced nodes in the interval $0 \leq x \leq 0.1$ with a support length of $l_s = 5$. The computation time on the Arduino Uno is $t = 1.788\,[\text{ms}]$, i.e., a sample rate of $> 500\,[\text{Hz}]$ is possible. The error plots of the numerical computations are shown in Fig. 14 and 15.

9.2. Initial Value Problem 2

The test case in Section 7.3 has been modified to have $n = 10$ evenly spaced nodes in the interval $1 \leq x \leq 2$ with a support length of $l_s = 5$. The computation time on the Arduino Uno is $t = 1.228\,[\text{ms}]$, i.e., a sample rate of $> 800\,[\text{Hz}]$ is possible. The error plots of the numerical computations are shown in Fig. 16 and 17

9.3. Inverse 3-Point Boundary Value Problem

The test case in Section 7.5 has been modified to have $n = 10$ evenly spaced nodes in the interval $0 \leq x \leq 0.1$ with a support length of $l_s = 5$. New constraints have been defined to conserve the test case's characteristics:

$$y(0.0556) = 0, \qquad\qquad y(0.1) = -0.1, \qquad\qquad \textbf{(45)}$$
$$D\,y(0) = 1, \qquad\qquad D\,y(0.1) = 0. \qquad\qquad \textbf{(46)}$$

The computation time on the Arduino Uno is $1.796\,[\text{ms}]$, i.e., a sample rate of $> 550\,[\text{Hz}]$ is possible. The error plots of the numerical computations are shown in Fig. 18 and 19. In contrast to the previous test cases, here the SIL verification delivered an error

Fig. 14: The plot shows the error between the analytic solution and MATLAB's solution (M), the error norm is $|\epsilon|_2 = 4.6771 \, 10^{-4}$, as well as the error between the analytic solution and Arduino's solution (A), the error norm is $|\epsilon|_2 = 4.6749 \, 10^{-4}$.

Fig. 15: PIL verification: the difference between MATLAB's solution and Arduino's solution is shown, the error norm is $|\epsilon|_2 = 3.1999 \, 10^{-7}$. The result is scaled by 10^7 to increase the visibility.

Fig. 16: The plot shows the error between the analytic solution and MATLAB's solution (M), the error norm is $|\epsilon|_2 = 4.9541 \, 10^{-4}$, as well as the error between the analytic solution and Arduino's solution (A), the error norm is $|\epsilon|_2 = 4.9568 \, 10^{-4}$.

Fig. 17: PIL verification: the difference between MATLAB's solution and Arduino's solution is shown, the error norm is $|\epsilon|_2 = 5.4520 \, 10^{-7}$. The result is scaled by 10^7 to increase the visibility.

vector with norm of $|\epsilon|_2 = 3.3866 \, 10^{-14}$, i.e., the C code's result is slightly different than the MATLAB model code's result.

10. LABORATORY TESTING

The introduced algebraic model is tested on a laboratory setup, see Fig. 20. A chain of equally spaced one-dimensional inclinometers is mounted on a $b = 1.8 \, [\text{m}]$ long flexible structure. The arrangement consists of 14 sensors with an additional 2 screw clamps, effectively forcing 2 pairs, i.e. $p = 4$, of homogeneous Dirichlet and Neumann constraints at the structure's ends. These leads to a total of $n = 16$ points for the computation, i.e., the vector of measurement data g is of size (16×1). In order to vary the structure's bending, a square metal profile with feed size $h = 20 \, [\text{mm}]$ is placed

Fig. 18: The plot shows the error between the analytic solution and MATLAB's solution (M), the error norm is $|\epsilon|_2 = 4.7958 \, 10^{-10}$, as well as the error between the analytic solution and Arduino's solution (A), the error norm is $|\epsilon|_2 = 2.3902 \, 10^{-8}$.

Fig. 19: PIL verification: the difference between MATLAB's solution and Arduino's solution is shown, the error norm is $|\epsilon|_2 = 2.3673 \, 10^{-8}$. The result is scaled by 10^8 to increase the visibility.

Fig. 20: The illustration shows the chain of inclinometers mounted on a flexible structure. Each of the 14 sensors is connected to an industrial RS-485 bus. This bus is converted to a RS-232 serial interface, which enables the connection of the BeagleBone Black. The 2 screw clamps force the homogeneous boundary values at the structure's ends.

between the structure and the supporting mounting platforms at the ξ_i positions. The results for these tests are shown in Fig. 21 to 26. Note, that the reference data has been acquired with calipers and hardly represents the true value; however, it is a good basis for comparisons. The constrained linear differential operator M is of size (16×16). M and the homogeneous solution y_h are computed in a preparatory step. The online computation is carried out by a BeagleBone Black. The hardware features a RISC processor based on the ARMv7 Cortex A8 platform with $1 \, [\mathrm{GHz}]$ (2000 MIPS) and $512 \, [\mathrm{MB}]$

Fig. 21: The reference values $\hat{y}$ and the computed curve y for the metal profile placed at $\xi_1 = 530\,[\text{mm}]$ are shown.

Fig. 22: The difference between both values $\epsilon = \hat{y} - y$ is shown, the error norm is $|\epsilon|_2 = 4.245\,[\text{mm}]$.

Fig. 23: The reference values $\hat{y}$ and the computed curve y for the metal profile placed at $\xi_2 = 890\,[\text{mm}]$ are shown.

Fig. 24: The difference between both values $\epsilon = \hat{y} - y$ is shown, the error norm is $|\epsilon|_2 = 2.636\,[\text{mm}]$.

Fig. 25: The reference values $\hat{y}$ and the computed curve y for the metal profile placed at $\xi_3 = 1160\,[\text{mm}]$ are shown.

Fig. 26: The difference between both values $\epsilon = \hat{y} - y$ is shown, the error norm is $|\epsilon|_2 = 3.114\,[\text{mm}]$.

memory. The measurement data g is acquired via the built-in RS232 serial interface, the results y are transmitted to a centralized database. The WAGO PFC-200 would be an industrial-ready alternative hardware solution with similar architecture for this application. It features a variety of bus interfaces such as Modbus, Profibus and CAN bus.

11. CONCLUSION AND OUTLOOK

It can be concluded, from the numerical and experimental tests, that the newly proposed algebraic method outperforms previous solutions, both in accuracy and speed, for the class of problems being considered. The separation of the computation into an initial preparatory and a cyclic run-time portion yields a highly efficient numeric solution. The computation complexity of the explicit solution is only a function of the number of nodes (sensors) used. The automatic generation of C code, and the verification of its correct functionality on multiple embedded architectures has been demonstrated. The generation of C code also facilitates the use of the method in conjunction with commercial programmable logic controllers (PLCs), for the control of industrial plants and machinery. Here, the method was applied to a linear array of sensors. Presently, the tools are being extended to two-dimensional arrays and the resulting two-dimensional fields of data.

REFERENCES

R.A. Adams. 2006. *Calculus: Several Variables* (sixth ed.). Pearson Addison Wesley.

R.P Agarwal, H.B Thompson, and C.C Tisdell. 2003. Three-point boundary value problems for second-order discrete equations. *Computers and Mathematics with Applications* 45, 69 (2003), 1429 – 1435.

A. Ben-Israel and T.N.E. Greville. 2003. *Generalized Inverses: Theory and Applications*. Springer.

Siegmund Brandt. 1998. *Data Analysis* (third ed.). Springer, New York, NY, USA.

R.L. Burden and J.D. Faires. 2005. *Numerical Analysis*. Thomson Brooks/Cole.

O. Burdet and L. Zanella. 2002. Automatic Monitoring of the Riddes Bridges using Electronic Inclinometers. In *IABMAS, First International Conference on Bridge Maintenance, Safety and Management*.

T. A. Driscoll, F. Bornemann, and L. N. Trefethen. 2008. The chebop system for automatic solution of differential equations. *BIT* 48 (2008), 701–723.

Bengt Fornberg. 1998. *A practical guide to pseudospectral methods*. Vol. 1. Cambridge university press.

J. Golser. 2010. Fallbeispiel zur Bauwerksberwachung mittels online Neigungssensoren. In *25. Cristian Veder Kolloquium*.

G.H. Golub and C.F. Van Loan. 1996. *Matrix Computations* (third ed.). The Johns Hopkins University Press, Baltimore.

M. Harker and P. O'Leary. 06 May 2013 (Updated 11 Jul 2013)d. Polynomial Toolbox. (06 May 2013 (Updated 11 Jul 2013)). http://www.mathworks.com/matlabcentral/fileexchange/41658

M. Harker and P. O'Leary. 11 Apr 2013 (Updated 19 Aug 2013)b. Discrete Orthogonal Polynomial Toolbox. (11 Apr 2013 (Updated 19 Aug 2013)). http://www.mathworks.com/matlabcentral/fileexchange/41250

M. Harker and P. O'Leary. 18 Apr 2013 (Updated 17 Jun 2013)c. Ordinary Differential Equation Toolbox. (18 Apr 2013 (Updated 17 Jun 2013)). http://www.mathworks.com/matlabcentral/fileexchange/41354

M. Harker and P. O'Leary. 2013a. A Matrix Framework for the Solution of ODEs: Initial-, Boundary-, and Inner-Value Problems. *ArXiv e-prints* (April 2013).

Chen Huang, Frank Vahid, and Tony Givargis. 2013. Automatic Synthesis of Physical System Differential Equation Models to a Custom Network of General Processing Elements on FPGAs. *ACM Trans. Embed. Comput. Syst.* 13, 2, Article 23 (Sept. 2013), 27 pages. DOI:http://dx.doi.org/10.1145/2514641.2514650

Nathaniel Jewell. 29 Mar 2013 (Updated 01 Apr 2013). Collocation-based spectral-element toolbox. (29 Mar 2013 (Updated 01 Apr 2013)). http://www.mathworks.com/matlabcentral/fileexchange/41011

Yyldyray Keskyn, Onur Karaoglu, Sema Servy, and OturanÇ Galip. 2011. The Approximate Solution of High-Order Linear Fractional Differential Equations with Variable Coefficients in Terms of Generalized Taylor Polynomials. *Mathematical and Computational Applications* Vol. 16, No. 3 (2011), 617–629.

Nurcan Kurt and Mehmet Cevik. 2008. Polynomial solution of the single degree of freedom system by Taylor matrix method. *Mechanics Research Communications* 35, 8 (2008), 530 – 536.

Y.V. Kurylev. 1993. Multi-dimensional inverse boundary problems by BC-method: Groups of transformations and uniqueness results. *Mathematical and Computer Modelling* 18, 1 (1993), 33 – 45.

C. Lanczos. 1997. *Linear Differential Operators*. Dover Publications.

L. Lapidus and G.F. Pinder. 1999. *Numerical Solution of Partial Differential Equations in Science and Engineering*. Wiley.

Insup Lee, O. Sokolsky, Sanjian Chen, J. Hatcliff, Eunkyoung Jee, BaekGyu Kim, A. King, M. Mullen-Fortino, Soojin Park, A. Roederer, and K.K. Venkatasubramanian. 2012. Challenges and Research Directions in Medical Cyber Physical Systems. *Proc. IEEE* 100, 1 (jan 2012), 75 –90.

W. W. Loh and F. J. Dickin. 1996. A novel computer architecture for real-time solution of inverse problems [electric impedance tomography]. In *Advances in Electrical Tomography (Digest No: 1196/143), IEE Colloquium on*. 22/1–22/3. DOI:http://dx.doi.org/10.1049/ic:19960850

George Machan and Victoria Gene Bennett. 2008. Use of Inclinometers for Geotechnical Instrumentation on Transportation Projects. *Transportation Research E-Circular* E-C129 (2008).

Jacob Mattingley and Stephen Boyd. 2012. CVXGEN: a code generator for embedded convex optimization. *Optimization and Engineering* 13, 1 (1 March 2012), 1–27.

D. Necsulescu and G. Ganapathy. 2005. Online Solving of Inverse Problems in Critical Infrastructure Monitoring. In *Instrumentation and Measurement Technology Conference, 2005. IMTC 2005. Proceedings of the IEEE*, Vol. 2. 1585–1589.

Paul O'Leary and Matthew Harker. 2012. A Framework for the Evaluation of Inclinometer Data in the Measurement of Structures. *IEEE T. Instrumentation and Measurement* 61, 5 (2012), 1237–1251.

Omair Rafique, Manuel Gesell, and Klaus Schneider. 2013. Targeting Different Abstraction Layers by Model-Based Design Methods for Embedded Systems: A Case Study. *Real-Time Computing Systems and Applications (RTCSA), IEEE Computer Society* (2013).

Abraham Savitzky and Marcel JE Golay. 1964. Smoothing and differentiation of data by simplified least squares procedures. *Analytical chemistry* 36, 8 (1964), 1627–1639.

Mehmet Sezer and Mehmet Kaynak. 1996. Chebyshev polynomial solutions of linear differential equations. *International Journal of Mathematical Education in Science and Technology* 27, 4 (1996), 607–618.

Lawrence F. Shampine and Mark W. Reichelt. 1997. The MATLAB ODE Suite. *SIAM J. Sci. Comput.* 18, 1 (Jan. 1997), 1–22.

G.D. Smith. 1985. *Numerical Solution of Partial Differential Equations: Finite Difference Methods*. Clarendon Press.

J.C. Strikwerda. 2004. *Finite Difference Schemes and Partial Differential Equations*. Society for Industrial and Applied Mathematics.

J. A. Weideman and S. C. Reddy. 2000. A MATLAB differentiation matrix suite. *ACM Trans. Math. Softw.* 26, 4 (Dec. 2000), 465–519.

B. Welfert. 1997. Generation of Pseudospectral Differentiation Matrices I. *SIAM J. Numer. Anal.* 34, 4 (1997), 1640–1657.

Wayne Welsh and Takeo Ojika. 1980. Multipoint boundary value problems with discontinuities I. Algorithms and applications. *J. Comput. Appl. Math.* 6, 2 (1980), 133 – 143.

S. Young. 1982. *Real Time Languages: Design and Development*. Ellis Horwood Publishers, Chichester.