A framework for data exchange and benchmarking of frictional contact solvers in multibody dynamics

Claude Lacoursière[*], **Mattias Linde**[×], **Ying Lu**[#], **Jeff Trinkle**[#]

[*] HPC2N/UMIT Research Lab
Umeå University
901 87 Umeå, Sweden
claude@hpc2n.umu.se

[×] Dept. of Computing Science
Umeå University
901 87 Umeå, Sweden
linde@cs.umu.se

[#] Dept. of Computer Science
Rensselaer Polytechnic Institute
110 8th Street
Troy, NY 12180, U.S.A.
rosebudflyaway@gmail.com
trink@cs.rpi.edu

ABSTRACT

We present an HDF5 layout specification to store and exchange the run-time kinematic data of simulations of contacting multibody systems subject to dry friction in descriptor form. This is intended to be used to test solvers on the incremental problem defined by one single configuration, allowing any mathematical formulation and friction law, and compatible with any numerical method. We also introduce metrics to measure the quality of a solution comprehensively. We started a public web site with a collection of problems, from simple stacking to full vehicle dynamics as well as grasping robots. We provide software to manipulate the data, assemble matrices needed for different formulations, interface with existing solvers, and to compute our metrics. As this is written in MATLAB and OCTAVE we believe our contributions will allow anyone wanting to write or test a solver to work with "real life" examples without any difficulty, and concentrate on the numerical methods themselves, and get results before writing a code suitable for integration with a full featured software package.

Keywords: multibody dynamics, benchmark, solvers, friction, complementarity, contact mechanics, data representation.

1 INTRODUCTION

There is currently no entirely satisfactory solver for the computation of contact forces for multibody systems subject to dry friction laws. Issues include performance, stability, robustness, and accuracy [see 2, implicitly stated]. There are also multiple physical models based either on points or contacts for instance, multiple formulations of the mathematical problem [2] as Nonlinear or Linear Complementarity Problems, Variational Inequalities [11], and optimization problems [10]. There are then numerous numerical methods such as stationary iterative ones, Krylov subspace ones, pivoting ones, and non-smooth Newton ones [see 2, for a detailed list and references]. This makes comparison difficult. As for testing a solver during development, reference cases are necessary and should be easily accessible from a prototyping environment, where one can perform tests on the incremental problem defined in discussed in Sec. 2 first. These issues have been discussed previously [7, 9, 15]. But since there isn't an exchange format yet, one is restricted to work on simplistic examples, or random problems [9, 15], which is hardly representative of real scenarios.

Benchmarking of multibody dynamics simulation library is already being addressed [13, 19]. But the focus in the latter studies and benchmark frameworks is to evaluate the global performance of a numerical method on given problems defined by configuration, initial values, and scenario. The test results are then affected by the overall formulation, the time integration algorithms, and the numerical methods used for solving forces and accelerations as well. As useful as this is, this is, we are consider with a different aspect of benchmarking.

Solver evaluation should be focused on the computation of forces and accelerations for the incremental problem, needed at any one stage of a time integration method. It is thus useful to have access to numerical kinematic data instead of having to write a simulation library or linking to one to access realistic scenarios. This is a radical difference from the exchange mechanism described in the last paragraph. With raw data, one can work with scripting languages and their libraries, such as MATLAB, OCTAVE, or PYTHON to name but three. This is the strategy we present here. Our datasets contain numerical values of mass matrices, Jacobians, velocities, forces, etc., for the descriptor form of the equations of motion. This data can then be processed easily by the aforementioned software tools without writing any interfacing code.

The interfaces to solver involve only matrices, vectors, and scalars, which are all naturally supported. This data can be rearranged and adapted to the need of any solver, existing ones in particular, which can then be used via plugins for the tools mentioned and others. Rearrangement includes transforming to dense matrices from sparse ones, using row or column major matrices, using saddle point formulations or computing the Delassus operator, for instance. The exchange mechanism we chose is HDF5 since it is natively supported by a plethora of data analysis and scripting software suitable for numerical computations.

A large collection of reference problem sets can then be generated easily with only a few simulation libraries which can write in this format, or whose output files can be converted to same if all the data is available. This is more general than declarative exchange formats at least for the incremental problem, since it is irrelevant at the numerical level whether a given Jacobian corresponds to a screw or tripode joint. Only numerical values are relevant for a solver.

By limiting the contact descriptions to points and normals, the user is free to choose the representation of contact manifolds. We believe that our datasets represent the lowest common denominator from which any formulation of the contact problem and any friction law can be expressed, and the data layout can be adapted to any type of solver designed for the descriptor form of the equations of motion. This contrast with a similar effort FCLIB [1] which represents one specific problem formulation in one specific matrix form. This is ill-suited for stationary iterative methods for instance, as they need original contact information unavailable FCLIB.

Performance metrics are needed to develop better solvers but are currently deficient. These must match computed solutions against the chosen physical model in a detailed and comprehensive way. Metrics are also needed provide a global statistical picture of the quality and performance of a solver on a large number of problems, and visualization tools or templates are needed to present this information.

The rest of the paper covers the definition of the incremental problem in Sec. 2, and the specification of the datasets in Sec. 3. The analysis pipeline is introduced in Sec. 4, with simple usage examples. Quality metrics are defined in Sec. 5. We present experimental results using our tools in Sec. 6 with discussion in Sec. 7 and conclusions in Sec. 8.

2 Problem definition

We consider the incremental problem of computing constraints and contact forces for a planar or spatial multibody system subject to dry frictional contacts. This is solved at any stage of a numerical time integration method. We concentrate on the spatial case in what follows for brevity.

In descriptor form, this can be formulated as follows:

$$\begin{aligned}
\mathbf{M}\bar{\mathbf{v}} &= \mathbf{G}^T\lambda + \mathbf{N}^T\nu + \mathbf{D}^T\beta + \mathbf{a}\\
\mathbf{G}\bar{\mathbf{v}} &= \mathbf{b},\\
\mathbf{N}\bar{\mathbf{v}} &\geq \mathbf{c}, \quad \nu \geq 0, \quad \text{and } \nu\cdot(\mathbf{N}\bar{\mathbf{v}} - \mathbf{c}) = 0,\\
\mathscr{F}(\bar{\mathbf{v}}, \nu, \beta) &= \text{true}.
\end{aligned} \tag{1}$$

Bold face is used for vectors and matrices, the latter in upper case, $(\cdot)^T$ is the transpose operator. Definitions follow.

The unknowns are the increment velocity vector $\bar{\mathbf{v}}$, the multipliers λ, ν, and β correspond to constraint, normal and tangent forces, respectively, according to $\mathbf{G}^T\lambda$, $\mathbf{N}^T\nu$ and $\mathbf{D}^T\beta$, respectively. The unknowns are to be computed so they satisfy a chosen friction law, $\mathscr{F}(\bar{\mathbf{v}}, \nu, \beta) = \text{true}$, with any numerical method chosen by the user of the dataset.

Quantities contained in or derived from the datasets are as follows.

The system's mass matrix $\mathbf{M}(\mathbf{q})$ is in represented in inertial frame, including inertia tensors, and depends on the generalized coordinates $\mathbf{q}$ which include quaternions for rotations. They are related nonlinearly to the generalized velocities $\mathbf{v}$ via a matrix $\mathbf{E}(\mathbf{q})$ so that $\dot{\mathbf{q}} = \mathbf{E}(\mathbf{q})\mathbf{v}$. The generalized velocity $\mathbf{v}$ is expressed in Cartesian coordinates, and includes translational and angular components.

The Jacobian matrix $\mathbf{G}$ corresponds to linearized equality constraints $\mathbf{g}(\mathbf{q}) = 0$ so that $\dot{\mathbf{g}}(\mathbf{q}) = \mathbf{G}\mathbf{v}$. The constraint definition is irrelevant for the computation of the incremental problem, but we do provide the value of $\mathbf{g}(\mathbf{q})$ in the data file, allowing for constraint violation.

Vectors $\mathbf{a}, \mathbf{b}$, and $\mathbf{c}$ depend on the chosen time discretization and stepping model, and are computed from the stored velocities, gaps and other information in the dataset according to the chosen model.

Matrix $\mathbf{N}$ projects the generalized velocities onto the normal bundle of the manifold $\chi(\mathbf{q}) = 0$, and $\mathbf{D}$ projects to the tangent bundle of same. The definition of the manifold is left to the user using the point-wise contact information found in the dataset. This means that the two matrices, $\mathbf{N}$ and $\mathbf{D}$ are not included in the files. The definition of $\mathbf{N}$ must satisfy $\dot{\chi} = \mathbf{N}\mathbf{v}$, meaning that the incremented gap would then be $\bar{\chi} = \chi(\mathbf{q}) + hN\bar{\mathbf{v}}$, where h is the time increment.

The complementarity condition on the third line of Eqn. (1), where inequalities are understood component-wise, means that separating contacts corresponds to components $\chi^{(j)}(\mathbf{q}) > 0, j = 1, 2, \ldots, n_{\text{contacts}}$ have zero normal forces. Other contacts have positive normals. Since penetration $\chi^{(j)}(\mathbf{q})$ can occur due to numerical errors, the vector $\mathbf{c}$ is included for generality, and would be computed and used to adjust for penetration error according to a given stepping scheme.

The friction law is represented by a function $\mathscr{F}(\bar{\mathbf{v}}, \nu, \beta)$ and its definition is left to the user.

We believe that the incremental problem defined in Eqn. (1) covers all models expressed in the descriptor form, for linearized constraints.

We left out joint friction and rheonomic constraints in Eqn. (1) for brevity but these are supported by the dataset specification.

Friction models based on position projection such as the Paoli-Schatzman scheme [22, 23, 24] are based on three step recurrences

$$\mathbf{q}_{k+1} \leftarrow \Phi(\mathbf{q}_k, \mathbf{q}_{k-1}), \tag{2}$$

where k is the discrete time and Φ is a mapping. This is also compatible as long as it is possible to invert the mapping Φ using the coordinates $\mathbf{q}_k$ and velocities $\mathbf{v}_k$. The datasets contain enough information to represent these as well.

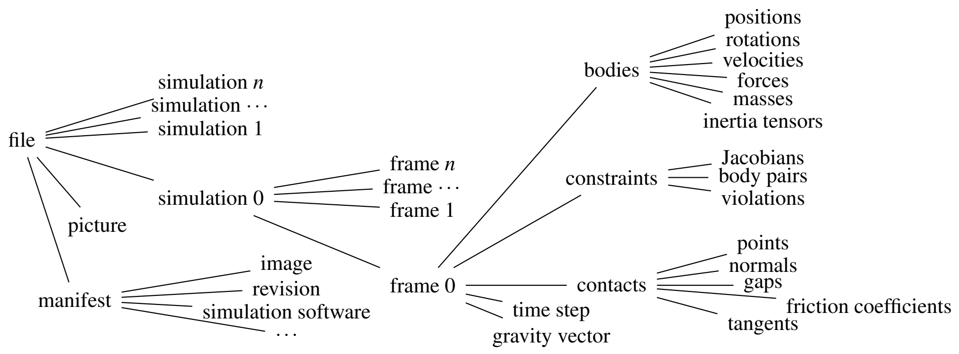

Figure 1: The layout of the HDF5 specification, excluding optional data.

3 DATASET SPECIFICATION AND HANDLING

We previously described the layout of the required kinematic data in the Hierarchical Data Format 5 (HDF5) [18] and started to collect problem sets on our webpage [20]. An almost complete description of the hierarchical specification implemented with HDF5 is described in Fig. 1, excluding optional, non-essential data. A more technical description is published on our website.

The Jacobian matrix **G** is stored explicitly in block-packed format, with two blocks per constraint, one block per constrained body. The number of rows is the number of individual equations, e.g., three for a ball joint, five for a hinge. An extension in preparation will cover constraints with more than two bodies. This contains no reference to the type of joint it is associated with.

Body variables such as mass, inertia tensor, position, velocity, angular velocities and forces are stored body-wise. This means that assembly is necessary if one wants the corresponding system-wide variables. The reason is that stationary iterative methods as well as Krylov subspace ones operate directly on individual bodies and individual constraint Jacobian blocks or individual rows. There lies one of the advantages of this specification when compared to that of FCLIB [1].

The normal and tangent Jacobians, **N** and **D**, are not stored at all in the data file as illustrated in Fig. 1. Instead, the user constructs these from contact points and normals according to the chosen representation of the contact manifold. This allows for point-wise or surface-wise models. One can also polygonize the Coulomb friction cone[26], or include the Contensou effects [16]. The contact data also allows for geometric analysis, for contact reduction for instance.

Optional contact information includes tangent vectors t_u, t_v which span the local contact surface which can be used to model anisotropic friction, in which case additional friction coefficients can be stored.

The generality of this specification is discussed further in Sec. 4.

The datasets comes from simulations which write "frames" – kinematic data – at selected time steps. A file can contain datasets from any number simulations, and each simulation containing any number of frames, each frame containing the data for one time step. The datasets are then loaded in the analysis environment and assembled in suitable form to be processed by a solver. Also included is a "manifest" for each simulation which describes the origin of said, along with any information relevant to reproduce same, though only a few descriptive words are required. An image can and should also be inserted. We are now defining keywords which should be present for use in plotting. This layout makes it easy to curate datasets since few files are involved and since they are self-describing. The format allows for any extension so that one could, for instance, include solutions in the datasets themselves so that with good management, these files could accumulate a wealth of information.

Instrumenting an existing simulation library is relatively simple. We provide a minimal C++ interface to HDF5 which hides the complexity of the standard HDF5 API for the needs at hand. We also provide a plugin for ODE [25]. Since ODE is integrated in Gazebo [12], widely used for robotics simulations, and is also integrated with V-Rep, also designed for robotics, a multitude of datasets can be generated with very little effort. Datasets can also be produced with AgX, a commercial multibody dynamics code[5], as well as the RPI-Matlab Simulator [27]. Other libraries can presumably store this kinematic data in which case conversion to our format requires straight forward scripting. We understand that introducing HDF5 write functionality in an existing library is intrusive, be it easy. In the worst case scenario, one would use an sufficiently expressive instrumented simulation library to obtain data for an example of interest, and this connects with the other benchmark efforts [13].

4 ANALYSIS PIPELINE

Using the information in the dataset described in Sec. 3 it is possible to assemble the matrices needed to solve any of the formulations of the dry friction problem [3]. We provide scripts to construct the most commonly used ones. From that point, any solver which has an interface for or is written in MATLAB can be immediately tested using our toolchain. The same goes for OCTAVE. For instance, we wrote scripts to produce GAMS files using the GDXMRW utilities, and this alone opens the door to a large number of solvers designed for general problems in mathematical programming. We wrote plugin code to link solvers found in the Siconos platform `http://tinyurl.com/siconos`, as well as for the Lemke [17] solver found in MOBY [8]. We also provide script versions of solvers we wrote ourselves. All the software we wrote for this project is posted on our website. Python offers extensive HDF5 support so it should be easy to do the same.

As a usage example, one can, for instance, construct the LCP corresponding to the polygonized friction cone [26] from the dataset found in a `frame` shown in Fig. 1, with a simple command:

```
[A, d] = h5_build_stap_reduced (frame, n_polygon_sides);
```

yielding a matrix **A** and a vector **d** corresponding to the LCP

$$0 \leq \mathbf{z} \perp \mathbf{A}\mathbf{z} + \mathbf{b} \geq 0. \tag{3}$$

What the function does is construct all matrices in Eqn. (1) but with a special definition of **N** and **D** which is related to the polygonization, compute Schur complements, and assemble **A** in a suitable form.

A complete test an implementation of Lemke's algorithm [17] on a simulation dataset would read as follows.

```
pivots = []; metrics = cell();
p = load('problems.h5');   %% file problems.h5 contains a dataset
for [ frame, key ] = p.simulation_00001;
  [A, q] = h5_build_stap_reduced (frame, n_polygon_sides);
  %% sol contains number of pivots and complementarity error
  [sol] = my_lemke(A, q);
  pivots = [pivots; sol.pivots];
  metric = coulomb_metrics(frame, sol, n_polygon_sides);
  metrics{end+1} = metric;
end
```

If that implementation came from an existing library, the sparse matrix **A** would be processed in the `mex` file `my_lemke` and formatted suitably. After this, the data goes back through the `mex` file, information is packed into a `struct`. The `coulom_metric` script will then analyze the solution and compute the different errors mentioned below in Sec. 5, as they apply this friction law. Other statistics would be presumably extracted. This simple script would then provide data for a large set of problems and potentially reveal anomalies not present in simple or random problems, but which would appear later in situ, at which point they are hard to isolate and understand.

Generating datasets does require instrumented libraries or converters and this is admittedly a limiting factor. But we have already produced very many of these covering a variety of configurations, including systems with joints and simpler stacking ones. We believe that even if only a few libraries were instrumented, the amount of available datasets would cover more than sufficiently many cases to stress test a solver, and detect any anomaly. Likewise, following a standard is always a challenge but since our goal is to maintain both the library and the supporting scripts, this should not be an issue for the end users, those who write and analyze solvers. In essence, the model we emulate here is that of the University of Florida Sparse Matrix Collection [6], a collection with far more consumers, software developers, than application focused producers.

5 QUALITY METRICS AND STATISTICAL ANALYSIS

A good solution to a frictional problem should satisfy the friction law in all its details. The global error reported by a solver is an average and provides too little information. For instance, if a contact point is reported as sticking, the tangential velocity should be zero. The error for this should be the residual tangential speed or *creep*, and should identified separately. For grasping problems for instance, this is perhaps the most significant error. For a sliding contact, one needs to know whether or not the friction force opposes the velocity. To our surprise, we found that the computed friction forces are often *aligned* with the sliding velocity, corresponding to *negative* friction. A measure of error here is the misalignment between the two. One also needs to know whether or not the transition between stick and slip modes is captured properly. Penetration is nearly inevitable and this too deserves its own measurement. This still leaves the question of which norm to use, but we leave that open and stick to the two-norm herein.

It is also important to understand the difficulty of a given problem. It is our experience that problems involving sliding contacts are more difficult to solve and so we measure both the total number of resting contacts as well as the fraction of sliding ones.

We measure the following:

- total number of resting contacts
- number of sliding contacts
- global error as defined by a given solver
- nonpenetration error: $\|\chi + h\mathbf{N}\mathbf{v}\|_2$ in the two norm, and h is the time step
- slide alignment error for each contact:$\left|\frac{\mathbf{t}^{(j)}.\tilde{\mathbf{v}}^{(j)}}{\|\tilde{\mathbf{v}}^{(j)}\|\|\mathbf{t}^{(j)}\|} + 1\right|$, and the two norm of these
- stick residual velocity error for static contacts: $\|\tilde{\mathbf{v}}^{(j)}\|$
- cone satisfaction error: $|\min(0, \mu^{(j)}\nu^{(j)} - \|\mathbf{f}^{(j)}\|)|$, and two norm of these
- anomalous friction: $\left|\max(\mathbf{f}^{(j)} \cdot \tilde{\mathbf{v}}^{(j)}\|, 0)\right|$, and two norm of these

Other metrics are related to the performance of different types of solvers as well as statistics for global evaluation on very large sets of problems are not presented here but are forthcoming.

6 EXPERIMENTS

We chose only two experiments for illustrative purposes. Comprehensive analysis is beyond the scope of the present article.

We extracted problems from a simulation in which logs are dropped and pile up on an inclined plane at a sufficient angle to cause sliding. We also include results from a simulation of a wheel loader shoveling rocks, as used in virtual reality based operator training systems. Performance is

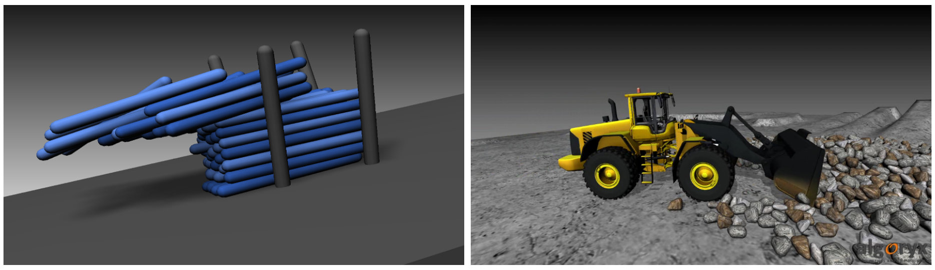
Figure 2: Slanted log pile and wheel loader simulations.

critical for the latter case because of real-time constraints, and this begs the questions of which type of errors are acceptable, which aren't. Still frames are shown in Fig 2.

The data was collected from the AgX toolkit `www.algoryx.se` first. This uses a split solver, where standard constraints and normal forces are computed with an block principal pivot LCP solver [14], and friction are computed with PGS. This procedure is repeated a number of times to approximate the Coulomb law.

We then used our framework to experiment with four different solvers on the slanted log pile. First comes PGS, then a solver of Morales et al. [21] using a combination of subspace minimization method and PGS, the non-smooth Newton method of Alart and Curnier [4], and the AgX solver itself. These are labeled as "PGS", "Morales", "Alart-Curnier", and "Block Pivot", respectively, and appear anticlockwise from the south-east corner in Fig. 3, on which the metrics of Sec. 5 are plotted. We let the PGS solver reach a stagnation point which is around 200 iterations. For the other solvers, we chose parameters so they would perform approximately the same amount of work in terms of linear algebra. The solutions produced appear to be the best they can deliver according to our experiments.

We arranged the results in such a way as to give a global picture of the solver's performance over many frames. The bar graphs are cumulative: the height corresponds to the total number of contacts. The number of contacts with bad sliding direction appear at the bottom in black, above, in light red is the number of sliding contacts with good alignment, and in dark red at the top, the number of sticking contacts. The alignment error is the red line, creep is in pale blue, and the friction cone violation is in black. There is one column and point on each curve for each frame extracted from the simulation. Clearly, alignment errors are big and the number of cases of "negative" friction is significant for all solvers and all configurations. Given that not all contacts are sliding, this would not be noticed in the global error. Worthy of notice is that the total number of contact varies between solvers, meaning that they identify different numbers of resting and separating contacts. The Morales solver underestimates the former if the other three solvers are any indication of the correct solution. The Alart-Curnier solver exhibits a number of anomalous solutions, even though the problems are similar from frame to frame.

For the wheel loader simulation, we only present the result from the AgX solver in Fig. 4. The PGS solver failed systematically on this problem. The Morales solver is not designed to handle problems with both constraints and contacts. And the Alart-Curnier solver was too slow and had too many failures. It is not clear if this is due to an implementation error, or because of numerical issues such as ill-conditioning. What the graph demonstrates is that errors are consistent throughout the simulation, a good thing for an operator training application, but the amount of creep is significant, and so is the number of contacts with negative friction. This is problematic for a vehicle as this causes the wheels to slip. This information can be used then to improve the situation by testing parameters in the analysis environment, i.e., without running the simulation again. This also begs the question of whether or not one should focus on producing a solver with "hard" stiction, perhaps at the cost of introducing larger errors somewhere else.

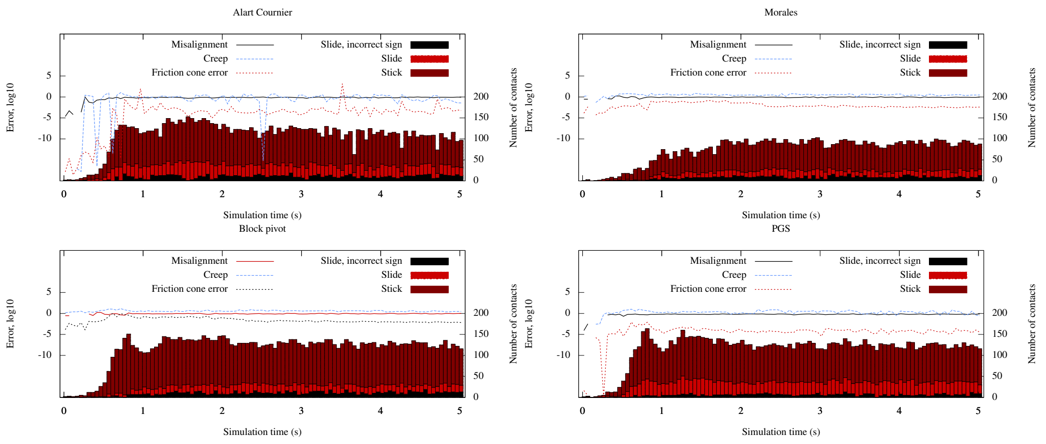

Figure 3: sdfsdfsf

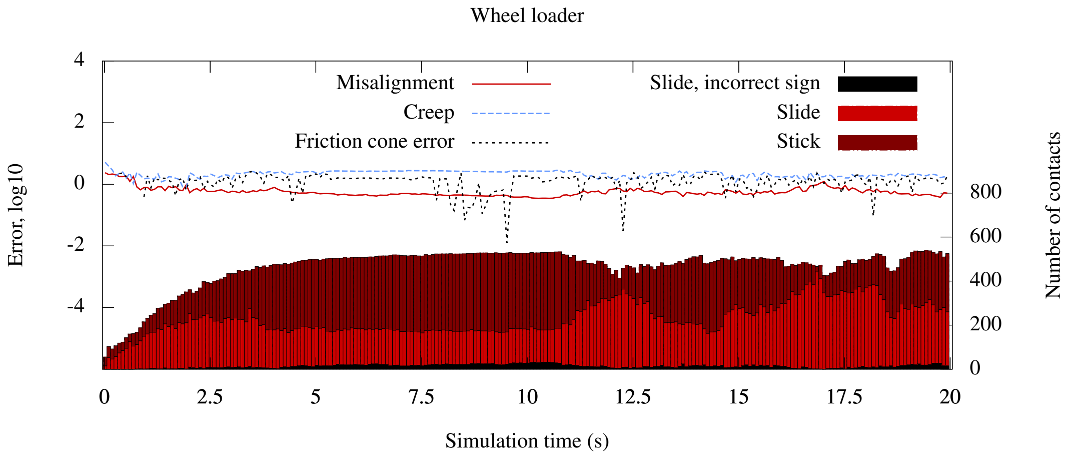

Figure 4: Performance statistics for the block pivot solver on the wheel-loader.

7 DISCUSSION

The paper is not an article about solvers and we present data only to show that the framework can give extensive quantitative information about given numerical methods. Yet the data clearly shows that at least four methods which are used routinely produce errors of the kind which, to our knowledge, has never been reported before. Yet just these two examples make our point and demonstrate our contribution. Our dataset specification and toolchain makes it easy to test and compare solvers, and the metrics we introduced provide comprehensive information allowing for deeper understanding.

8 CONCLUSION

We believe that the HDF5 layout specification, the metrics we introduced, along with the datasets and software we published on our website can and will help analyzing and constructing new, better and faster solvers, concentrating on algorithms instead of software development. From our own experience in developing solvers, this type of infrastructure is in fact a necessity. Testing new ideas from within a simulation library and running series of test cases is much more difficult, and not necessarily more enlightening. There is already a lot of useful information in the solution of a single incremental problem indeed.

The next stage is to collect datasets and perform extensive testing of existing solvers in order to establish a more accurate picture of the state of the art in solving frictional contact problems.

We are developing a set of reference datasets which can be used to test the fundamental properties of a solver such as stick-slip transitions, isotropy and creep. But various issues only appear when considering objects with different shapes, length scales and mass ratios. Ill-conditioning for instance can make a solver fail. Datasets including more difficult problems must be included in a comprehensive collection. We already provide large and complex examples and will continue to produce datasets which allow to stress test solvers.

An extension to support minimal coordinates is being considered, though few solvers support that at the moment.

ACKNOWLEDGEMENTS

Part of this research was conducted using the resources of High Performance Computing Center North (HPC2N), and support from Algoryx Simulation AB.

This work was partially supported by the National Science Foundation through the National Robotics Initiative NSF CCF-1208468, DARPA W15P7T-12-1-0002, and Lockheed-Martin Advanced Technology Laboratory. Any opinions, findings, conclusions, or recommendations expressed herein are those of the authors and do not necessarily reflect the views of the funding agencies.

REFERENCES

[1] V. Acary, M. Brémond, T Koziara, and Franck Pérignon. FCLIB: a collection of discrete 3D frictional contact problem. Technical Report 444, hal-00945820v2, Inria,, 2014.

[2] Vincent Acary and Bernard Brogliato. *Numerical Methods for Nonsmooth Dynamical Systems. Applications in Mechanics and Electronics*. Springer-Verlag, 2008.

[3] Vincent Acary, Olivier Bonnefon, and Bernard Brogliato. *Nonsmooth Modeling and Simulation for Switched Circuits*. Springer-Verlag, 2011.

[4] P. Alart and A. Curnier. A mixed formulation for frictional contact problems prone to newton like solution methods. *Computer Methods in Applied Mechanics and Engineering*, 92(3):353 – 375, 1991.

[5] Algoryx Simulations AB. AgX multiphysics simulation toolkit, Sep 2015. URL `www.algoryx.se`.

[6] Timothy A. Davis and Yifan Hu. The University of Florida Sparse Matrix Collection. *ACM Transactions on Mathematical Software*, 38(1), Nov 2011.

[7] E. Drumwright and D.A. Shell. An evaluation of methods for modeling contact in multibody simulation. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1695–1701, May 2011.

[8] Evan Drumwright. Moby, Sep 2015. URL `http://physsim.sourceforge.net/`.

[9] Evan Drumwright and Dylan Shell. Extensive analysis of linear complementarity problem (LCP) solver performance on randomly generated rigid body contact problems. In *Proc. IEEE/RSJ Intl. Conference on Intelligent Robots and Systems (IROS)*, Portugal, 2012.

[10] Evan Drumwright and Dylan A. Shell. Modeling contact friction and joint friction in dynamic robotic simulation using the principle of maximum dissipation. In *Algorithmic Foundations of Robotics IX*, pages 249–266, 2010.

[11] Francisco Facchinei and Jong-Shi Pang. *Finite-dimensional variational inequalities and complementarity problems. Vol. I.* Springer-Verlag, New York, 2003.

[12] Gazebo, 2015. URL `www.gazebosim.org`.

[13] Manuel González, Francisco González, Alberto Luaces, and Javier Cuadrado. A collaborative benchmarking framework for multibody system dynamics. *Engineering with Computers*, 26(1):1–9, 2010.

[14] J. J. Júdice and F. M. Pires. A block principal pivoting algorithm for large-scale strictly monotone linear complementarity problems. *Computers Ops Res.*, 21(5):587–596, 1994.

[15] Claude Lacoursière. Splitting methods for dry frictional contact problems in rigid multibody systems: Preliminary performance results. In Mark Ollila, editor, *Conference Proceedings from SIGRAD2003, November 20–21, 2003, Umeå University, Umeå, Sweden*, pages 11–16, November 2003.

[16] R.I Leine and Ch Glocker. A set-valued force law for spatial Coulomb-Contensou friction. *European Journal of Mechanics - A/Solids*, 22(2):193 – 216, 2003.

[17] C. E. Lemke. Bimatrix equilibrium points and mathematical programming. *Management Science*, 11:681–689, 1965.

[18] Y. Lu, C. Lacoursière, J. Williams, and J.C. Trinkle. Standard interface for data analysis of solvers in multibody dynamics. In *Canadian Conference on Nonlinear Solid Mechanics (CanCNSM)*, 2013.

[19] Y. Lu, J. Williams, J.C.Trinkle, and C. Lacoursière. A framework for problem standardization and algorithm comparison in multibody system. In *ASME Intl. Design and Engineering Technical Conferences and Computers and Information in Engineering*, 2014.

[20] Y. Lu, , , C. Lacoursière, M. Linde, and J.C.Trinkle. Benchmarks problems for multibody dynamimcs with dry frictional contacts (BPMD), 2015. URL `http://tinyurl.com/rpih5`.

[21] Jose Luis Morales, Jorge Nocedal, and Mikhail Smelyanskiy. An algorithm for the fast solution of symmetric linear complementarity problems. *Numer. Math.*, 111(2):251–266, Dec 2008.

[22] Laetitia Paoli and Michelle Schatzman. Penalty approximation for dynamical systems submitted to multiple non-smooth constraints. *Multibody System Dynamics*, 8(3):345–364, 2002.

[23] Laetitia Paoli and Michelle Schatzman. A numerical scheme for impact problems i: The one-dimensional case. *SIAM Journal on Numerical Analysis*, 40(2):pp. 702–733, 2003.

[24] Laetitia Paoli and Michelle Schatzman. A numerical scheme for impact problems ii: The multidimensional case. *SIAM Journal on Numerical Analysis*, 40(2):pp. 734–768, 2003.

[25] Russel Smith, 2015. URL `www.ode.org`.

[26] D. E. Stewart and J. C. Trinkle. An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and Coulomb friction. *Int. J. Numer. Meth. Engng.*, 39:2673–2691, 1996.

[27] J. Williams, Y. Lu, S. Niebe, M. Andersen, K. Erleben, and J.C. Trinkle. RPI-MATLAB-Simulator: A tool for efficient research and practical teaching in multibody dynamics. In *Workshop on Virtual Reality Interaction and Physical Simulation (VRIPHYS)*, 2013.