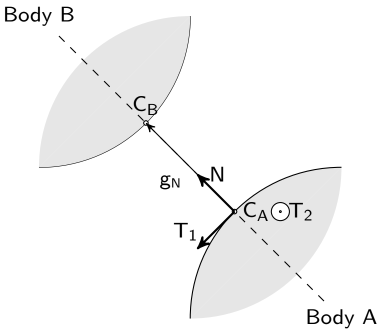
Siconos/numerics and FCLIB: a collection of solvers and benchmarks for solving frictional contact problems

CMIS, May 2024

Vincent Acary

Inria - Centre de l'Université Grenoble Alpes - Laboratoire Jean Kuntzmann

Signorini's condition and Coulomb's friction



- gap function $g_N = (C_B - C_A)N$.
- reaction forces velocities

$$r = r_N N + r_T, \quad \text{with } r_N \in \mathbf{R} \text{ and } r_T \in \mathbf{R}^2.$$

$$u = u_N N + u_T, \quad \text{with } u_N \in \mathbf{R} \text{ and } u_T \in \mathbf{R}^2.$$

- Signorini conditions

$$\text{position level :} 0 \leqslant g_N \perp r_N \geqslant 0.$$

$$\text{velocity level :} \begin{cases} 0 \leqslant u_N \perp r_N \geqslant 0 & \text{if } g_N \leqslant 0 \\ r_N = 0 & \text{otherwise.} \end{cases}$$

Signorini's condition and Coulomb's friction

Modeling assumption

Let μ be the coefficient of friction. Let us define the Coulomb friction cone K which is chosen as the isotropic second order cone

$$K = \{r \in \mathbb{R}^3 \mid \|r_{\mathrm{T}}\| \leqslant \mu r_n\}. \tag{1}$$

Coulomb friction postulates

- for the sticking case that
$$u_{\mathrm{T}} = 0, \quad r \in K, \tag{2}$$
- and for the sliding case that
$$u_{\mathrm{T}} \neq 0, \quad r \in \partial K, \frac{r_{\mathrm{T}}}{\|r_{\mathrm{T}}\|} = -\frac{u_{\mathrm{T}}}{\|u_{\mathrm{T}}\|}. \tag{3}$$

Disjunctive formulation of the frictional contact behavior

$$\begin{cases} r = 0 & \text{if } g_{\mathrm{N}} > 0 \quad \text{(no contact)} \\ r = 0, u_{\mathrm{N}} \geqslant 0 & \text{if } g_{\mathrm{N}} \leqslant 0 \quad \text{(take–off)} \\ r \in K, u = 0 & \text{if } g_{\mathrm{N}} \leqslant 0 \quad \text{(sticking)} \\ r \in \partial K, u_{\mathrm{N}} = 0, \dfrac{r_{\mathrm{T}}}{\|r_{\mathrm{T}}\|} = -\dfrac{u_{\mathrm{T}}}{\|u_{\mathrm{T}}\|} & \text{if } g_{\mathrm{N}} \leqslant 0 \quad \text{(sliding)} \end{cases} \tag{4}$$

Signorini's condition and Coulomb's friction

Second Order Cone Complementarity (SOCCP) formulation

- Modified relative velocity $\hat{u} \in \boldsymbol{R}^3$ defined by (**DeSaxce92**)

$$\hat{u} = u + \mu \| u_{\mathrm{T}} \| \mathrm{N}. \tag{5}$$

- Second-Order Cone Complementarity Problem (SOCCP)

$$K^\star \ni \hat{u} \perp r \in K \tag{6}$$

if $g_{\mathrm{N}} \leqslant 0$ and $r = 0$ otherwise.
The set $K^\star$ is the dual convex cone to K defined by

$$K^\star = \{ u \in \boldsymbol{R}^3 \mid r^\top u \geqslant 0, \quad \text{for all } r \in K \}. \tag{7}$$

(**Acary.Brogliato2008**; **Acary.ea˙ZAMM2011**)

Signorini's condition and Coulomb's friction

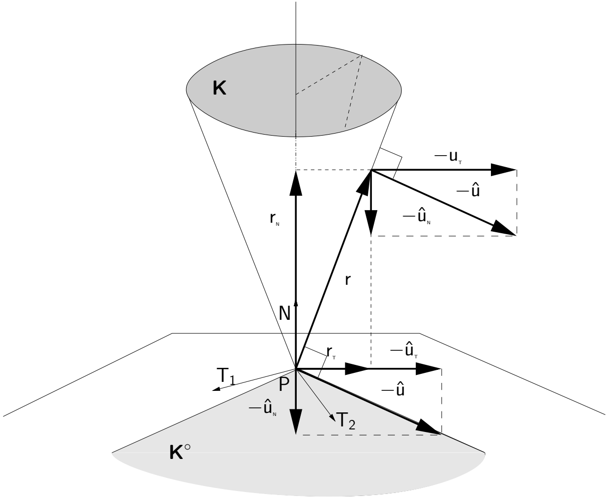


Figure: Coulomb's friction and the modified velocity $\hat{u}$. The sliding case.

3D frictional contact problems

Problem 1 (General discrete frictional contact problem)

Given

- *a symmetric positive definite matrix* $M \in \mathbb{R}^{n\times n}$,
- *a vector* $f \in \mathbb{R}^n$,
- *a matrix* $H \in \mathbb{R}^{n\times m}$,
- *a vector* $w \in \boldsymbol{R}^m$,
- *a vector of coefficients of friction* $\mu \in \boldsymbol{R}^{n_c}$,

find three vectors $v \in \mathbb{R}^n$, $u \in \boldsymbol{R}^m$ *and* $r \in \boldsymbol{R}^m$, *denoted by* $\mathrm{FC/I}(M, H, f, w, \mu)$ *such that*

$$\begin{cases} Mv = Hr + f \\ u = H^\top v + w \\ \hat{u} = u + g(u) \\ K^\star \ni \hat{u} \perp r \in K \end{cases} \tag{8}$$

with $g(u) = [[\mu^\alpha \| u_T^\alpha \| \mathrm{N}^\alpha]^\top, \alpha = 1 \dots n_c]^\top$. □

3D frictional contact problems

Problem 2 (Reduced discrete frictional contact problem)

Given

- *a symmetric positive semi–definite matrix* $W \in \mathbb{R}^{m\times m}$,
- *a vector* $q \in \mathbb{R}^{m}$,
- *a vector* $\mu \in \boldsymbol{R}^{n_c}$ *of coefficients of friction,*

find two vectors $u \in \boldsymbol{R}^m$ *and* $r \in \boldsymbol{R}^m$, *denoted by* $\mathrm{FC/II}(W, q, \mu)$ *such that*

$$\begin{cases} u = Wr + q \\ \hat{u} = u + g(u) \\ K^\star \ni \hat{u} \perp r \in K \end{cases} \tag{9}$$

with $g(u) = [[\mu^\alpha \| u^\alpha_T \| \mathrm{N}^\alpha]^\top, \alpha = 1 \ldots n_c]^\top$. □

Relation with the general problem

$W = H^\top M^{-1} H$ and $q = H^\top M^{-1} f + w$.

VI based methods

VI reformulation

$$F(r) = Wr + q + g(Wr + q), \qquad -F(r) \in N_K(r) \tag{10}$$

Standard methods

- Basic fixed point iterations with projection [FP-VI]

$$\mathrm{r_{k+1} \leftarrow P_K(r_k - \rho_k\, F(r_k))}$$

- Extragradient method [EG-VI]

$$\mathrm{r_{k+1} \leftarrow P_K(r_k - \rho_k\, F(P_K(r_k - \rho_k F(r_k))))}$$

with fixed $\rho_k = \rho$, we get the Uzawa Algorithm of De Saxcé-Feng [FP-DS]

Self-adaptive procedure for ρ_k

[UPK]

$$\text{Armijo-like}: \; m_k \in \boldsymbol{N} \quad \text{such that} \begin{cases} \rho_k = \rho 2^{m_k}, \\ \rho_k \|F(r_k) - F(\bar{r}_k)\| \leqslant \|r_k - \bar{r}_k\| \end{cases}$$

Nonsmooth Equations based methods

Nonsmooth Newton on $G(z) = 0$

$$z_{k+1} = z_k - \Phi^{-1}(z_k)(G(z_k)), \qquad \Phi(z_k) \in \partial G(z_k)$$

- Alart–Curnier Formulation (**Alart.Curnier1991**) [NSN-AC]

$$\begin{cases} r_{\mathrm{N}} - P_{\boldsymbol{R}^{n_c}_{+}}(r_{\mathrm{N}} - \rho_{\mathrm{N}} u_{\mathrm{N}}) = 0, \\ r_{\mathrm{T}} - P_{D(\mu, r_{\mathrm{N},+} + \rho u_{\mathrm{N}})}(r_{\mathrm{T}} - \rho_{\mathrm{T}} u_{\mathrm{T}}) = 0, \end{cases}$$

- Jean–Moreau Formulation [NSN-MJ]

$$\begin{cases} r_{\mathrm{N}} - P_{\boldsymbol{R}^{n_c}_{+}}(r_{\mathrm{N}} - \rho_{\mathrm{N}} u_{\mathrm{N}}) = 0, \\ r_{\mathrm{T}} - P_{D(\mu, r_{\mathrm{N},+})}(r_{\mathrm{T}} - \rho_{\mathrm{T}} u_{\mathrm{T}}) = 0, \end{cases}$$

- Direct normal map reformulation [NSN-NM]

$$r - P_K\,(r - \rho(u + g(u))) = 0$$

- Extension of Fischer-Burmeister function to SOCCP [NSN-FB]

$$\phi_{\mathrm{FB}}(x, y) = x + y - (x^2 + y^2)^{1/2}$$

Matrix block-splitting and projection based algorithms (**Moreau1994**; **Jean.Touzot1988**)

Block splitting algorithm with $W^{\alpha\alpha} \in \boldsymbol{R}^3$ [NSGS-*]

$$\begin{cases} u^{\alpha}_{i+1} - W^{\alpha\alpha} P^{\alpha}_{i+1} = q^{\alpha} + \sum\limits_{\beta<\alpha} W^{\alpha\beta} r^{\beta}_{i+1} + \sum\limits_{\beta>\alpha} W^{\alpha\beta} r^{\beta}_{i} \\[2ex] \widehat{u}^{\alpha}_{i+1} = \left[u^{\alpha}_{\mathrm{N},i+1} + \mu^{\alpha}\, ||u^{\alpha}_{\mathrm{T},i+1}||, u^{\alpha}_{\mathrm{T},i+1} \right]^T \\[2ex] \mathbf{K}^{\alpha,*} \ni \widehat{u}^{\alpha}_{i+1} \perp r^{\alpha}_{i+1} \in \mathbf{K}^{\alpha} \end{cases} \tag{11}$$

for all $\alpha \in \{1 \dots m\}$.

Over-Relaxation [PSOR-*]

One contact point problem

- closed form solutions
- Any solver listed before.

Optimization based methods

- Alternating optimization problems (Panagiotopoulos et al.) [PANA-*]
- Successive approximation with Tresca friction (Haslinger et al.) [TRESCA-*]

$$\left\{\begin{array}{l} \theta = h(r_{\mathrm{N}}) \\ \min \ \frac{1}{2} r^\top W r + r^\top q \\ \quad \text{s.t.} \qquad r \in C(\mu, \theta) \end{array}\right. \tag{12}$$

where $C(\mu, \theta)$ is the cylinder of radius $\mu\theta$.

- Fixed point on the norm of the tangential velocity [A., Cadoux, Lemaréchal, Malick(2011)] [ACLM-*].

$$\left\{\begin{array}{l} s = \|u_{\mathrm{T}}\| \\ \min \ \frac{1}{2} r^\top W r + r^\top (q + \alpha s) \\ \quad \text{s.t.} \qquad r \in K \end{array}\right. \tag{13}$$

Fixed point or Newton Method on $F(s) = s$

Interior Point Methods

Presentation of Hoang Minh Nguyen.

Siconos/Numerics

Siconos

Open source software for modelling and simulation of nonsmooth systems

Siconos/Numerics

Collection of C routines to solve FC3D problems in dense, sparse or block sparse versions:

- VI solvers: Fixed point, Extra-Gradient, Uzawa
- VI based projection/splitting algorithm: NSGS, PSOR
- Semismooth Newton methods
- Optimization based solvers. Panagiotopoulos, Tresca, SOCQP, ADMM
- Interior point methods, . . .

Collection of routines for optimization and complementarity problems

- LCP solvers (iterative and pivoting (Lemke))
- Standard QP solvers (Projected Gradient (Calamai & Moré), Projected CG (Moré & Toraldo), active set technique)
- linear and nonlinear programming solvers.

Siconos/Numerics

Implementation details

- Matrix format.
  - dense (column-major)
  - sparse matrices (triplet, CSR, CSC)
- Linear algebra libraries and solvers.
  - BLAS/LAPACK, MKL
  - MUMPS, SUPERLU, UMFPACK,
  - PETSc
- Python interface (swig (pybind11 coming soon))
- Generic structure for problem, driver and options

```
int fc3d_driver(FrictionContactProblem* problem,
                double* reaction,
                double* velocity,
                SolverOptions* numerics_solver_options);
```

C structure to encode the problem

Reduced discrete frictional contact problem

```
struct FrictionContactProblem {
  /** dimension of the contact space (3D or 2D ) */
  int dimension;
  /** the number of contacts \f£ n_c \f£ */
  int numberOfContacts;
  /** \f£ {M} \in {{\mathrm{I\!R}}}^{n \times n} \f£,
  a matrix with \f£ n = d  n_c \f£ stored in NumericsMatrix structure */
  NumericsMatrix *M;
  /** \f£ {q} \in {{\mathrm{I\!R}}}^{n} \f£ */
  double *q;
  /** \f£ {\mu} \in {{\mathrm{I\!R}}}^{n_c} \f£, vector of friction coefficients
  (\f£ n_c = \f£ numberOfContacts) */
  double *mu;
};
```

C structure to encode the problem

Global discrete frictional contact problem

```
struct GlobalFrictionContactProblem {
  /** dimension \f£ d=2 \f£ or \f£ d=3 \f£ of the contact space (3D or 2D ) */
  int dimension;
  /** the number of contacts \f£ n_c \f£ */
  int numberOfContacts;
  /** \f£ M \in {\mathrm{I\!R}}^{n \times n} \f£,
  a matrix with \f£ n\f£ stored in NumericsMatrix structure */
  NumericsMatrix *M;
  /**  \f£ {H} \in {{\mathrm{I\!R}}}^{n \times m} \f£,
  a matrix with \f£ m = d  n_c\f£ stored in NumericsMatrix structure */
  NumericsMatrix *H;
  /** \f£ {q} \in {{\mathrm{I\!R}}}^{n} \f£ */
  double *q;
  /** \f£ {b} \in {{\mathrm{I\!R}}}^{m} \f£ */
  double *b;
  /** \f£ {\mu} \in {{\mathrm{I\!R}}}^{n_c} \f£, vector of friction
  coefficients
  (\f£ n_c = \f£ numberOfContacts) */
  double *mu;
};
```

A very basic example in C

```
// Problem Definition
int NC = 3;//Number of contacts
double M[81] = {1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
double q[9] = { -1, 1, 3, -1, 1, 3, -1, 1, 3};
double mu[3] = {0.1, 0.1, 0.1};

FrictionContactProblem NumericsProblem;
NumericsProblem.numberOfContacts = NC;
NumericsProblem.dimension = 3;
NumericsProblem.mu = mu;
NumericsProblem.q = q;

NumericsMatrix *MM = (NumericsMatrix*)malloc(sizeof(NumericsMatrix));
MM->storageType = NM_DENSE;
MM->matrix0 = M;
MM->size0 = 3 * NC;
MM->size1 = 3 * NC;
NumericsProblem.M = MM;
```

A basic example in C

```
// Variable declaration
double *reaction = (double*)calloc(3 * NC, sizeof(double));
double *velocity = (double*)calloc(3 * NC, sizeof(double));

// Numerics and Solver Options
SolverOptions *numerics_solver_options = solver_options_create(SICONOS_FRICTION_3D_NSGS);
numerics_solver_options->iparam[SICONOS_IPARAM_MAX_ITER] = 1000;
numerics_solver_options->dparam[SICONOS_DPARAM_TOL] = 100*DBL_EPSILON;
// numerics_set_verbose(2);

// Driver call
fc3d_driver(&NumericsProblem,
            reaction, velocity,
            numerics_solver_options);
```

A basic example in Python

```
import numpy as np
import siconos.numerics as sn

NC = 1
M = np.eye(3 * NC)
q = np.array([-1.0, 1.0, 3.0])
mu = np.array([0.1])
FCP = sn.FrictionContactProblem(3, M, q, mu)

reactions = np.array([0.0, 0.0, 0.0])
velocities = np.array([0.0, 0.0, 0.0])
sn.numerics_set_verbose(1)
```

A basic example in Python

```
def solve(problem, solver, options):
    """Solve problem for a given solver"""
    reactions[...] = 0.0
    velocities[...] = 0.0
    r = solver(problem, reactions, velocities, options)
    assert options.dparam[sn.SICONOS_DPARAM_RESIDU] < options.dparam[sn.SICONOS_DPARAM_TOL]
    assert not r

def test_fc3dnsgs():
    """Non-smooth Gauss Seidel, default"""
    SO = sn.SolverOptions(sn.SICONOS_FRICTION_3D_NSGS)
    solve(FCP, sn.fc3d_nsgs, SO)

def test_fc3dlocalac():
    """Non-smooth Gauss Seidel, Alart-Curnier as local solver."""
    SO = sn.SolverOptions(sn.SICONOS_FRICTION_3D_NSN_AC)
    solve(FCP, sn.fc3d_nonsmooth_Newton_AlartCurnier, SO)

def test_fc3dfischer():
    """Non-smooth Newton, Fischer-Burmeister."""
    SO = sn.SolverOptions(sn.SICONOS_FRICTION_3D_NSN_FB)
    solve(FCP, sn.fc3d_nonsmooth_Newton_FischerBurmeister, SO)

if __name__ == "__main__":
    test_fc3dnsgs()
    test_fc3dlocalac()
    test_fc3dfischer()
```

FCLIB : a collection of discrete 3D Frictional Contact (FC) problems

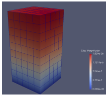
Our inspiration: MCPLIB or CUTEst

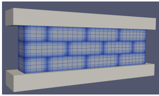
What is FCLIB ?

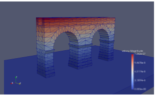
- A open source collection of Frictional Contact (FC) problems stored in a specific HDF5 format
- A open source light implementation of Input/Output functions in C Language to read and write problems (Python and Matlab coming soon)

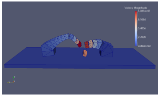
Goals of the project

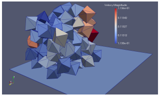
Provide a standard framework for testing available and new algorithms for solving discrete frictional contact problems share common formulations of problems in order to exchange data

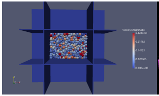

(a) Cubes_H8

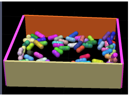

(b) LowWall_FEM


(c) Aqueduct_PR

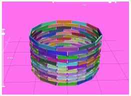

(d) Bridge_PR

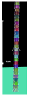

(e) 100_PR_Periobox


(f) 945_SP_Box_PL


(g) Capsules


(h) Chain


(i) KaplasTower


(j) BoxesStack

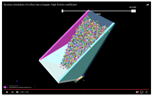

(k) Chute_1000, Chute_4000, Chute_local_problems

Figure: Illustrations of the FClib test problems

Conclusions & Perspectives

Conclusions

- Siconos/Numerics. A open source collection of solvers.
  `https://github.com/siconos/siconos` use and contribute ...
- FCLIB: a collection of discrete 3D Frictional Contact (FC) problems
  https://github.com/FrictionalContactLibrary use and contribute ...

Thank you for your attention.

Thanks to the collaborators for stimulating discussions and developments:

Pierre Alart, Paul Armand, Florent Cadoux, Frédéric Dubois,
Claude Lémaréchal, Jérôme Malick and Mathieu Renouf