Time integration

DVI and HHT time stepping methods in Chrono

Time Integration in Chrono

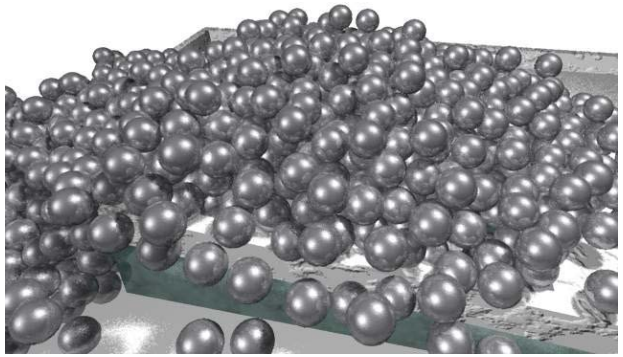
- Two classes of time stepping methods in Chrono
  - Time steppers for smooth dynamics
    - Classical multibody dynamics – rigid and flexible connected through joints
    - FEA
    - Fluid solid interaction problems
  - Time steppers for non-smooth dynamics
    - Scenarios w/ friction and contact

Time Integration – Smooth Dynamics

- Smooth dynamics:
  - Equations of Motion: formulated as Differential Algebraic Equations (DAE)
  - Time-stepping methods:
    - **HHT**
    - **Euler implicit**
    - **Euler semi-implicit linearized**
    - **Newmark**
  - Require solution of a linear system at each time step
    - **MINRES**
    - **MKL**
    - **MUMPS**, etc.
  - Discontinuous forces if any, are regularized via penalty
    - Can still have friction and contact, but is “smoothed”

Time Integration – Non-smooth Dynamics

- Non-smooth dynamics:
  - Equations of motion formulated as Differential Variational Inequality (DVI) Problems
  - Time-stepping method:
    - **Euler implicit linearized** (Anitescu-Trinkle)
  - Required solver at each time step: **Cone Complementarity Problem**
    - **SOR**
    - **Barzilai-Borwein**
    - **APGD**
  - Set-valued and discontinuous forces: no need to be “smoothed”
  -
  - No support for FEA yet



Smooth dynamics - DAE

The HHT Time Stepper

Linear Solvers

Differential problems

- An Ordinary Differential Equation (**ODE**):

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, t)$$

- A Differential Algebraic Equation (**DAE**)
  - In implicit form:

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, t)$$
$$\boldsymbol{g}(\boldsymbol{x}, t) = \boldsymbol{0}$$

  - Introduces constraints ***g***

The Chrono case

$$M\frac{d\boldsymbol{v}}{dt} = \boldsymbol{f}(\boldsymbol{q}, \boldsymbol{v}, t) + D_{\mathcal{B}}\widehat{\gamma}_{\mathcal{B}}(t)$$
$$\boldsymbol{C}(\boldsymbol{q}, t) = \boldsymbol{0}$$

DAE Explicit Integrators

- **Explicit** integrators:

$$\boldsymbol{x}(t + \Delta t) = \boldsymbol{F}(\boldsymbol{x}(t))$$

  - Very straightforward - they do not require solving linear systems
  - Require very small time steps, due to stability reasons
  - The stiffer the problem, the smaller the time step
  - Lead to numerical drift when handling DAEs
  - Used by traditional DEM granular dynamics simulators

DAE Implicit Integrators

- **Implicit** integrators:

$$\boldsymbol{G}\left(\boldsymbol{x}(t+\Delta t)\,,\boldsymbol{x}(t)\right) = \mathbf{0}$$

  - Can use large time steps
  - More complex: they find $\boldsymbol{x}(t+\Delta t)$ by solving a nonlinear system $\boldsymbol{G} = \mathbf{0}$ with Newton Raphson
    - Jacobians matrices of $\boldsymbol{G}$ are needed (ex. stiffness matrices, etc.)
    - Require solution of one or more linear systems at each time step
  - Useful both for ODEs and DAEs – for the latter, they treat the constraints well
  - Used in FEA problems, handle stiffness well

DAE Implicit Integrators in Chrono

- Classical Euler implicit
  - First order accurate, large numerical damping

- **Euler semi-implicit linearized (1 step)**
  - **First order accurate, large numerical damping**
  - **Same time-stepping used for DVI non-smooth dynamics, it can use complementarity solvers**

- Trapezoidal
  - Second order accurate, no numerical damping
  - Doesn’t work well with joints (kinematic constraints)

- Newmark
  - Adjustable numerical damping, first order (except in particular case)

- **HHT**
  - **Second order accurate, adjustable numerical damping**
  - **Most used integrator for FEA problems in Chrono**

The HHT integrator

- The DAE problem is:

$$M\frac{d\boldsymbol{v}}{dt} = \boldsymbol{f}(\boldsymbol{q},\boldsymbol{v},t) + D_{\mathcal{B}}\widehat{\gamma}_{\mathcal{B}}(t)$$
$$\boldsymbol{C}(\boldsymbol{q},t) = \boldsymbol{0}$$

- The HHT time discretization is:

$$\boldsymbol{q}^{l+1} - \boldsymbol{q}^{l} - h\boldsymbol{v}^{l} - \frac{h^2}{2}\left[(1-2\beta)\boldsymbol{a}^{l} + 2\beta\boldsymbol{a}^{l+1}\right] = 0$$
$$\boldsymbol{v}^{l+1} - \boldsymbol{v}^{l} - h\left[(1-\gamma)\boldsymbol{a}^{l} + \gamma\boldsymbol{a}^{l+1}\right] = 0$$
$$M\boldsymbol{a}^{l+1} + (1+\alpha)(C_q^T\boldsymbol{\lambda} - \boldsymbol{f})^{l+1} - \alpha(C_q^T\boldsymbol{\lambda} - \boldsymbol{f})^{l} = 0$$

- A-stable for: $\gamma = \dfrac{1-2\alpha}{2} \quad \beta = \dfrac{(1-\alpha)^2}{4}$
- 2-nd order accurate
- An alternative formulation exis $\alpha \in [-\frac{1}{3}, 0]$ ion-level HHT
- Adjustable parameter α : from 0 (no numerical damping; i.e., trapezoidal) to -1/3 (max numerical damping)

The HHT integrator

- The $\boldsymbol{G}\left(\boldsymbol{x}(t+\Delta t)\,,\,\boldsymbol{x}(t)\right)=\mathbf{0}$ non-linear problem to solve is:

$$\begin{aligned}
\boldsymbol{q}^{l+1}-\boldsymbol{q}^{l}-h\boldsymbol{v}^{l}-\frac{h^2}{2}\left[(1-2\beta)\boldsymbol{a}^{l}+2\beta\boldsymbol{a}^{l+1}\right]&=0\\
\boldsymbol{v}^{l+1}-\boldsymbol{v}^{l}-h\left[(1-\gamma)\boldsymbol{a}^{l}+\gamma\boldsymbol{a}^{l+1}\right]&=0\\
M\boldsymbol{a}^{l+1}+(1+\alpha)(C_q^T\boldsymbol{\lambda}-\boldsymbol{f})^{l+1}-\alpha(C_q^T\boldsymbol{\lambda}-\boldsymbol{f})^{l}&=0
\end{aligned}$$

- Its Newton-Raphson step requires solving this **linear system**:

$$\begin{bmatrix} H & \overline{C}_q^T \\ \overline{C}_q & 0 \end{bmatrix}\begin{Bmatrix} \Delta\boldsymbol{a}^{l+1} \\ \Delta\boldsymbol{\lambda}^{l+1} \end{Bmatrix}=$$

$$\begin{Bmatrix} \frac{1}{1+\alpha}(M\boldsymbol{a}^{l+1})+(C_q^T\boldsymbol{\lambda}-\boldsymbol{f})^{l+1}-\frac{\alpha}{1+\alpha}(C_q^T\boldsymbol{\lambda}-\boldsymbol{f})^{l} \\ \frac{1}{\beta h^2}\boldsymbol{C}^{l+1} \end{Bmatrix}$$

$$\begin{aligned}
\boldsymbol{a}_{n+1}^{l+1}&=\boldsymbol{a}_{n}^{l+1}+\Delta\boldsymbol{a}^{l+1}\\
\boldsymbol{\lambda}_{n+1}^{l+1}&=\boldsymbol{\lambda}_{n}^{l+1}+\Delta\boldsymbol{\lambda}^{l+1}\\
\boldsymbol{v}^{l+1}&=\boldsymbol{v}^{l}+h\left[(1-\gamma)\boldsymbol{a}^{l}+\gamma\boldsymbol{a}^{l+1}\right]\\
\boldsymbol{q}^{l+1}&=\boldsymbol{q}^{l}+h\boldsymbol{v}^{l}+\frac{h^2}{2}\left[(1-2\beta)\boldsymbol{a}^{l}+2\beta\boldsymbol{a}^{l+1}\right]
\end{aligned}$$

$$H=\left[M-\gamma h\nabla_v\boldsymbol{f}^{l+1}-\beta h^2\nabla_q\boldsymbol{f}^{l+1}+\beta h^2\left[(M\boldsymbol{a})_q+(C_q{}^T\boldsymbol{\lambda})_q\right]\right]$$

Configuring the Integrator in Chrono

- It can be changed with `SetTimesteppertype()`
- Additional parameters via `std::static_pointer_cast<...>(my_system.GetTimestepper())`

```
// change the time integration to Euler, also suitable for NSC too (this is the default)
my_system.SetTimesteppertype(ChTimestepper::Type::EULER_IMPLICIT_LINEARIZED);
```

```
// change the time integration to HHT:
my_system.SetTimesteppertype(ChTimestepper::Type::HHT);
auto integrator = std::static_pointer_cast<ChTimestepperHHT>(my_system.GetTimestepper());
integrator->SetAlpha(-0.2);
integrator->SetMaxiters(8);
integrator->SetAbsTolerances(5e-05, 1.8e00);
integrator->SetMode(ChTimestepperHHT::POSITION);
integrator->SetModifiedNewton(false);
integrator->SetScaling(true);
integrator->SetVerbose(true);
```

Linear System Solvers

- All DAE solvers require solving a linear system
- Linear system solvers are independent from the time integrator
  - One can mix and match
- Available linear system solvers
  - MINRES (iterative solver, free)
  - MKL (direct solver, requires license)
  - MUMPS (direct solver, free)
- Moving forward:
  - MUMPS with OpenBLAS since they are both free and licensed under BSD

Linear System Solvers: MINRES

- Available in the main Chrono unit
- A Krylov-type iterative solver
- Convergence might slow down when large mass or stiffness ratios are used
- Robust in case of redundant constraints
- Warm starting can be used to reuse last solution (faster solution)

```
// Change solver settings
my_system.SetSolverType(ChSolver::Type::MINRES);
my_system.SetSolverWarmStarting(true);
my_system.SetMaxItersSolverSpeed(200); // Max number of iterations for main solver
my_system.SetMaxItersSolverStab(200);  // Used only by few time-integrators
my_system.SetTolForce(1e-13);
```

Linear System Solvers: MKL

- MKL Intel libraries must be licensed and installed on your system,
- Available in the optional Chrono::MKL unit (enable it in Cmake)
- Direct parallel solver: no iterations are needed
- Not robust in case of redundant constraints – avoid them!
- Cannot use `SetSolverType()`, you must create a solver and plug it in the ChSystem:

```
#include "chrono_mkl/ChSolverMKL.h"
...
// change the solver to MKL:
auto mkl_solver = std::make_shared<ChSolverMKL<>>();
my_system.SetSolver(mkl_solver);
mkl_solver->SetSparsityPatternLock(true);
mkl_solver->SetVerbose(true);
```

Linear System Solvers: MUMPS

- Work in progress to be wrapped up by mid January
- Direct parallel solver
- Developed in France/UK, relies on OpenBLAS, which developed in China
- Free solution, source code available for MUMPS & OpenBLAS

```
#include "chrono_mkl/ChSolverMUMPS.h"
...
// change the solver to MUMPS:
auto mumps_solver = std::make_shared<ChSolverMUMPS<>>();
my_system.SetSolver(mumps_solver);
mumps_solver->SetSparsityPatternLock(true);
```

Non-Smooth dynamics - NSC

The DVI time-stepper

The CCP solvers

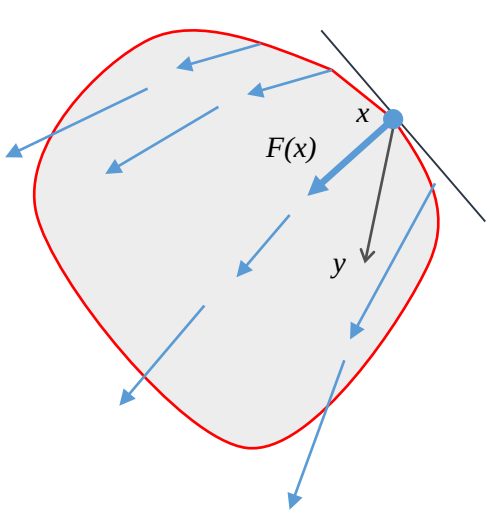
Variational Inequalities

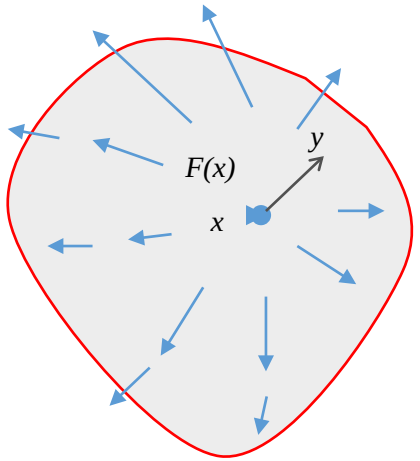
- Definition of Variational Inequality (VI):

$$\boldsymbol{x} \in \mathbb{K} \quad : \quad \langle F(\boldsymbol{x}), \boldsymbol{y} - \boldsymbol{x} \rangle \geq 0 \quad \forall \boldsymbol{y} \in \mathbb{K}$$

  - for continuous $F(\boldsymbol{x}) : \mathbb{K} \to \mathbb{R}^n$
  - with closed and convex $\mathbb{K}$

(see Kinderleher and Stampacchia ,1980)





- Alternative formulation:

$$\mathbf{x} \in \mathscr{K}\,,\ \mathbf{g}(\mathbf{x}) \in -\mathsf{N}_{\mathscr{K}}(\mathbf{x})$$

$$\mathsf{N}_{\mathscr{K}}(\mathbf{x}) = \{\mathbf{y} \in \mathbb{R}^n : \langle \mathbf{y}, \mathbf{x} - \mathbf{z} \rangle \geq 0, \forall \mathbf{z} \in \mathscr{K}\}$$

Differential Variational Inequality

- Differential Variational Inequality (DVI)

$$\frac{d\boldsymbol{x}}{dt} = f(t, \boldsymbol{x}, \boldsymbol{u})$$
$$\boldsymbol{u} \in \mathrm{SOL}\left(\mathbb{K}, F(t, \boldsymbol{x}(t), \cdot)\right)$$
$$\Xi(\boldsymbol{x}(0), \boldsymbol{x}(T)) = 0$$

where $\mathrm{SOL}\left(\mathbb{K}, F(t, \boldsymbol{x}(t), \cdot)\right)$ is the set of solutions to the VI $(\mathbb{K}, F(t, \boldsymbol{x}(t), \cdot))$

- It is also a special class of Differential Inclusion (DI), $dx/dt \in f(x,t)$

Differential Inclusions: motivation

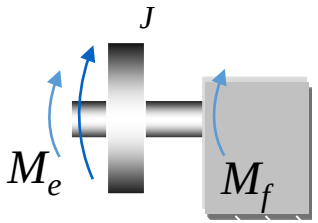
- Most differential problems can be posed as **equalities** like:

  $dx/dt = f(x,t)$ → ODE, DAE , ok

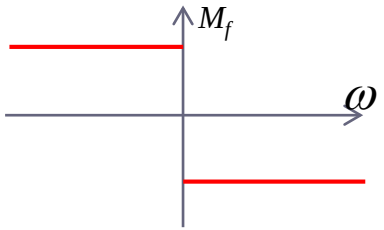
- But some problems require **inequalities** or **inclusions** like

  $dx/dt \in f(x,t)$ → Differential Inclusion! (DI)

- *Example: a flywheel with brake torque and applied torque (looks simple?!)*

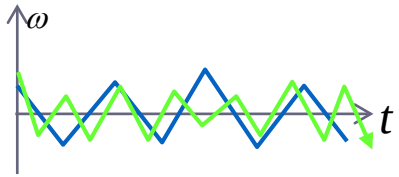


- $J\, d\omega/dt = M_f(\omega) + M_e(t)$ where $M_f = -M_{f\,max}$ if $\omega > 0$
  and $M_f = M_{f\,max}$ if $\omega < 0$



- All ODE integrator would never stop in $\omega = 0$!
  It would just ripple about $\omega = 0$ ..

- Reducing Δt in ODE integrator may reduce the ripple,
  But what if low J ? Divergence!



- Regularization methods? A) Numerical stiffness!
  B) Approximation! C) The brake would never stick! …

- Also, if ever $\omega = 0$, which M_f ? Not computable!

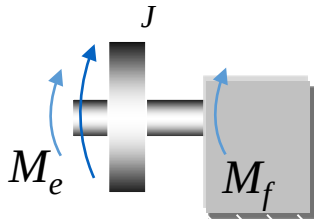
Differential Inclusions: motivation

- Most differential problems can be posed as **equalities** like:

  $dx/dt = f(x,t)$ → ODE, DAE , ok

- But some problems require **inequalities** or **inclusions** like

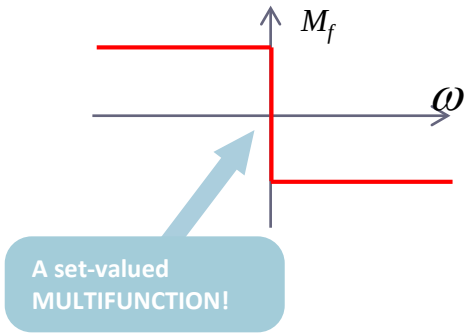
  $dx/dt \in f(x,t)$ → Differential Inclusion! (DI)



- *Example: a flywheel with brake torque and applied torque (simple?!)*
- *Improved model!*
- $J\, d\omega/dt = M_f(\omega) + M_e(t)$ where $M_f = -M_{f\,max}$ for $\omega > 0$

  and $M_f = M_{f\,max}$ for $\omega < 0$

  and $-M_{f\,max} < M_f < M_{f\,max}$ for $\omega = 0$



- This could handle also $\omega = 0$ case, ex. brake sticking

- But now we have a **differential inclusion** $d\omega/dt \in f(\omega,t)$. It requires special solvers.

Measure Differential Inclusions

- What if the velocity must have discontinuities?
  - ..because of impulses,
  - ..because of impacts,
  - ..because of friction effects such as in Painlevé paradox

- The RHS has ‘peaks’ (impulses) → *measure distributions*
  The velocity has ‘jumps’ → *function of bounded variation*

- Measure Differential Inclusion (MDI): *strong* definition [Moreau]

$$\mathrm{d}\mu/\mathrm{d}v(t) \in K(t)$$

For singular decomposition of Borel **measure** $\mu = m\,v + \mu_s$

$$m(t) \in K(t) \qquad \frac{\mathrm{d}\mu_s}{\mathrm{d}|\mu_s|}(t) \in K(t)_\infty$$

Our DVI model

- Mechanical system with
  - Set $\mathcal{G}_{\mathcal{B}}$ of bilateral joints
  - Set $\mathcal{G}_{\mathcal{A}}$ of point contacts
  - External forces

$$\dot{\boldsymbol{q}} = \Gamma(\boldsymbol{q})\boldsymbol{v}$$

$$M(\boldsymbol{q})\frac{d\boldsymbol{v}}{dt} = \sum_{i\in\mathcal{G}_{\mathcal{B}}} \widehat{\gamma}^i_{\mathcal{B}} \nabla\Psi^i + \sum_{i\in\mathcal{G}_{\mathcal{A}}} \widehat{\gamma}^i_{\mathcal{A}} D^i + \boldsymbol{f}_t(t,\boldsymbol{q},\boldsymbol{v})$$

$$\Psi^i(\boldsymbol{q},t) \in \emptyset, \quad i \in \mathcal{G}_{\mathcal{B}}$$

$$\widehat{\gamma}^i_{\mathcal{A}} \in \mathrm{SOL}\left(\Upsilon^i, F(t,\boldsymbol{q}(t),\boldsymbol{v}(t),\cdot)\right), \quad i \in \mathcal{G}_{\mathcal{A}}.$$

Bilateral constraint equations

Contact forces VIs

Our DVI model

- Do DVI time-step discretization, as a Measure Differential Inclusion MDI

- It leads to a Nonlinear Complementarity Problem (NCP), also a Variational Inequality (VI)

- Solve VI at each time step for
  - **unknown speeds**
  - **unknown reaction impulses**

$$M(\boldsymbol{v}^{(l+1)} - \boldsymbol{v}^{l}) = \sum_{i \in \mathcal{A}(q^{(l)}, \epsilon)} (\gamma_n^i \boldsymbol{D}_n^i + \gamma_u^i \boldsymbol{D}_u^i + \gamma_v^i \boldsymbol{D}_v^i) + \sum_{i \in \mathcal{G}_\mathcal{B}} (\gamma_b^i \nabla \Psi^i) + h \boldsymbol{f}_t(t^{(l)}, \boldsymbol{q}^{(l)}, \boldsymbol{v}^{(l)})$$

$$0 = \frac{1}{h}\Psi^i(\boldsymbol{q}^{(l)}) + \nabla \Psi^{i^T} \boldsymbol{v}^{(l+1)} + \frac{\partial \Psi^i}{\partial t}, \quad i \in \mathcal{G}_\mathcal{B}$$

$$0 \leq \frac{1}{h}\Phi^i(\boldsymbol{q}^{(l)}) + \nabla \Phi^{i^T} \boldsymbol{v}^{(l+1)}$$

$$\perp \quad \gamma_n^i \geq 0, \ i \in \mathcal{A}(q^{(l)}, \epsilon)$$

$$(\gamma_u^i, \gamma_v^i) = \operatorname{argmin}_{\mu^i \gamma_n^i \geq \sqrt{(\gamma_u^i)^2 + (\gamma_v^i)^2}} \quad i \in \mathcal{A}(q^{(l)}, \epsilon)$$

$$\left[\boldsymbol{v}^T(\gamma_u \boldsymbol{D}_u^i + \gamma_v \boldsymbol{D}_v^i)\right]$$

$$\boldsymbol{q}^{(l+1)} = \boldsymbol{q}^{(l)} + h\boldsymbol{v}^{(l+1)},$$

Speeds

Reaction impulses

Forces

Stabilization terms

Bilateral constraint equations

Contact constraint equations

COMPLEMENTARITY!

Coulomb 3D friction model

Cone complementarity

- A modification (relaxation, to get a convex problem):

$$
\begin{aligned}
M(\boldsymbol{v}^{(l+1)} - \boldsymbol{v}^{l}) = & \sum_{i \in \mathcal{A}(q^{(l)}, \epsilon)} \left(\gamma_n^i \boldsymbol{D}_n^i + \gamma_u^i \boldsymbol{D}_u^i + \gamma_v^i \boldsymbol{D}_v^i\right) \cdot \\
& + \sum_{i \in \mathcal{G}_{\mathcal{B}}} \left(\gamma_b^i \nabla \Psi^i\right) + h \boldsymbol{f}_t(t^{(l)}, \boldsymbol{q}^{(l)}, \boldsymbol{v}^{(l)}) \\
0 = & \frac{1}{h} \Psi^i(\boldsymbol{q}^{(l)}) + \nabla \Psi^{i^T} \boldsymbol{v}^{(l+1)} + \frac{\partial \Psi^i}{\partial t}, \quad i \in \mathcal{G}_{\mathcal{B}} \\
0 \leq & \frac{1}{h} \Phi^i(\boldsymbol{q}^{(l)}) + \nabla \Phi^{i^T} \boldsymbol{v}^{(l+1)} \boxed{- \mu^i \sqrt{(\boldsymbol{D}_u^{i,T} \boldsymbol{v})^2 + (\boldsymbol{D}_v^{i,T} \boldsymbol{v})^2}} \\
& \perp \quad \gamma_n^i \geq 0, \ i \in \mathcal{A}(q^{(l)}, \epsilon) \\
(\gamma_u^i, \gamma_v^i) = & \operatorname{argmin}_{\mu^i \gamma_n^i \geq \sqrt{(\gamma_u^i)^2 + (\gamma_v^i)^2}} \quad i \in \mathcal{A}(q^{(l)}, \epsilon) \\
& \left[\boldsymbol{v}^T (\gamma_u \boldsymbol{D}_u^i + \gamma_v \boldsymbol{D}_v^i)\right] \\
\boldsymbol{q}^{(l+1)} = & \boldsymbol{q}^{(l)} + h \boldsymbol{v}^{(l+1)},
\end{aligned}
$$

For small h and/or small speeds and/or small friction, almost no differences from the Coulomb theory.
Also, convergence proved as in the original scheme.

[see M.Anitescu, "Optimization Based Simulation of Nonsmooth Rigid Body Dynamics"]

Cone complementarity

- Aiming at a more compact formulation:

$$\boldsymbol{b}_{\mathcal{A}} = \left\{ \frac{1}{h}\Phi^{i_1}, 0, 0, \frac{1}{h}\Phi^{i_2}, 0, 0, \ldots, \frac{1}{h}\Phi^{i_{n_{\mathcal{A}}}}, 0, 0 \right\}$$

$$\boldsymbol{\gamma}_{\mathcal{A}} = \left\{ \gamma_n^{i_1}, \gamma_u^{i_1}, \gamma_v^{i_1}, \gamma_n^{i_2}, \gamma_u^{i_2}, \gamma_v^{i_2}, \ldots, \gamma_n^{i_{n_{\mathcal{A}}}}, \gamma_u^{i_{n_{\mathcal{A}}}}, \gamma_v^{i_{n_{\mathcal{A}}}} \right\}$$

$$\boldsymbol{b}_{\mathcal{B}} = \left\{ \frac{1}{h}\Psi^1 + \frac{\partial \Psi^1}{\partial t}, \frac{1}{h}\Psi^2 + \frac{\partial \Psi^2}{\partial t}, \ldots, \frac{1}{h}\Psi^{n_{\mathcal{B}}} + \frac{\partial \Psi^{n_{\mathcal{B}}}}{\partial t} \right\}$$

$$\boldsymbol{\gamma}_{\mathcal{B}} = \left\{ \gamma_b^1, \gamma_b^2, \ldots, \gamma_b^{n_{\mathcal{B}}} \right\}$$

$$D_{\mathcal{A}} = \left[D^{i_1} | D^{i_2} | \ldots | D^{i_{n_{\mathcal{A}}}} \right], \quad i \in \mathcal{A}(\boldsymbol{q}^l, \epsilon) \qquad D^i = \left[\boldsymbol{D}_n^i | \boldsymbol{D}_u^i | \boldsymbol{D}_v^i \right]$$

$$D_{\mathcal{B}} = \left[\nabla\Psi^{i_1} | \nabla\Psi^{i_2} | \ldots | \nabla\Psi^{i_{n_{\mathcal{B}}}} \right], \quad i \in \mathcal{G}_{\mathcal{B}}$$

$$\boldsymbol{b}_{\mathcal{E}} \in \mathbb{R}^{n_{\mathcal{E}}} = \{ \boldsymbol{b}_{\mathcal{A}}, \boldsymbol{b}_{\mathcal{B}} \}$$

$$\boldsymbol{\gamma}_{\mathcal{E}} \in \mathbb{R}^{n_{\mathcal{E}}} = \{ \boldsymbol{\gamma}_{\mathcal{A}}, \boldsymbol{\gamma}_{\mathcal{B}} \}$$

$$D_{\mathcal{E}} = [D_{\mathcal{A}} | D_{\mathcal{B}}]$$

- We also introduce the convex cone

$$\Upsilon = \left(\bigoplus_{i \in \mathcal{A}(\boldsymbol{q}^l, \epsilon)} \mathcal{FC}^i \right) \bigoplus \left(\bigoplus_{i \in \mathcal{G}_{\mathcal{B}}} \mathcal{BC}^i \right)$$

- ..and its polar cone:

$$\Upsilon^{\circ} = \left(\bigoplus_{i \in \mathcal{A}(\boldsymbol{q}^l, \epsilon)} \mathcal{FC}^{i\circ} \right) \bigoplus \left(\bigoplus_{i \in \mathcal{G}_{\mathcal{B}}} \mathcal{BC}^{i\circ} \right)$$

$\mathcal{FC}^i$ is *i*-th friction cone $\mathcal{BC}^i$ is R

Cone complementarity

- We introduce the Delassus operator N

$$N = D_{\mathcal{E}}^T M^{-1} D_{\mathcal{E}}$$

$$\boldsymbol{r} = D_{\mathcal{E}}^T M^{-1} \tilde{\boldsymbol{k}} + \boldsymbol{b}_{\mathcal{E}} \quad \tilde{\boldsymbol{k}}^{(l)} = M\boldsymbol{v}^{(l)} + h\boldsymbol{f}_t(t^{(l)}, \boldsymbol{q}^{(l)}, \boldsymbol{v}^{(l)})$$

- Finally we formulate everything as a Cone Complementarity Problem (CCP):

$$\begin{aligned}
M(\boldsymbol{v}^{(l+1)} - \boldsymbol{v}^l) &= \sum_{i \in \mathcal{A}(q^{(l)}, \epsilon)} (\gamma_n^i \boldsymbol{D}_n^i + \gamma_u^i \boldsymbol{D}_u^i + \gamma_v^i \boldsymbol{D}_v^i) + \\
&+ \sum_{i \in \mathcal{G}_{\mathcal{B}}} (\gamma_b^i \nabla \Psi^i) + h\boldsymbol{f}_t(t^{(l)}, \boldsymbol{q}^{(l)}, \boldsymbol{v}^{(l)}) \\
0 &= \frac{1}{h}\Psi^i(\boldsymbol{q}^{(l)}) + \nabla \Psi^{i^T} \boldsymbol{v}^{(l+1)} + \frac{\partial \Psi^i}{\partial t}, \quad i \in \mathcal{G}_{\mathcal{B}} \\
0 &\leq \frac{1}{h}\Phi^i(\boldsymbol{q}^{(l)}) + \nabla \Phi^{i^T} \boldsymbol{v}^{(l+1)} \\
&\perp \quad \gamma_n^i \geq 0, \; i \in \mathcal{A}(q^{(l)}, \epsilon) \\
(\gamma_u^i, \gamma_v^i) &= \operatorname{argmin}_{\mu^i \gamma_n^i \geq \sqrt{(\gamma_u^i)^2 + (\gamma_v^i)^2}} \quad i \in \mathcal{A}(q^{(l)}, \epsilon) \\
&\quad [\boldsymbol{v}^T(\gamma_u \boldsymbol{D}_u^i + \gamma_v \boldsymbol{D}_v^i)]
\end{aligned}$$

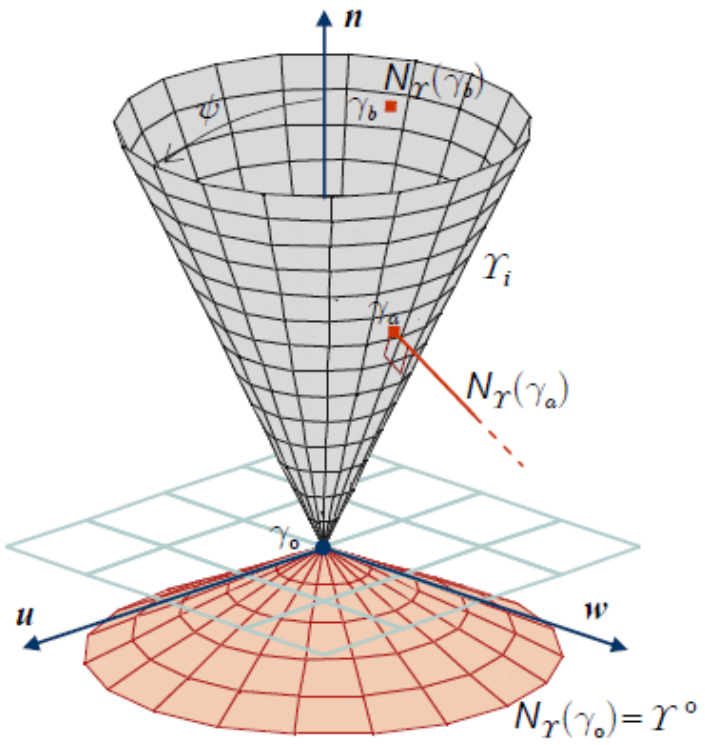
becomes..

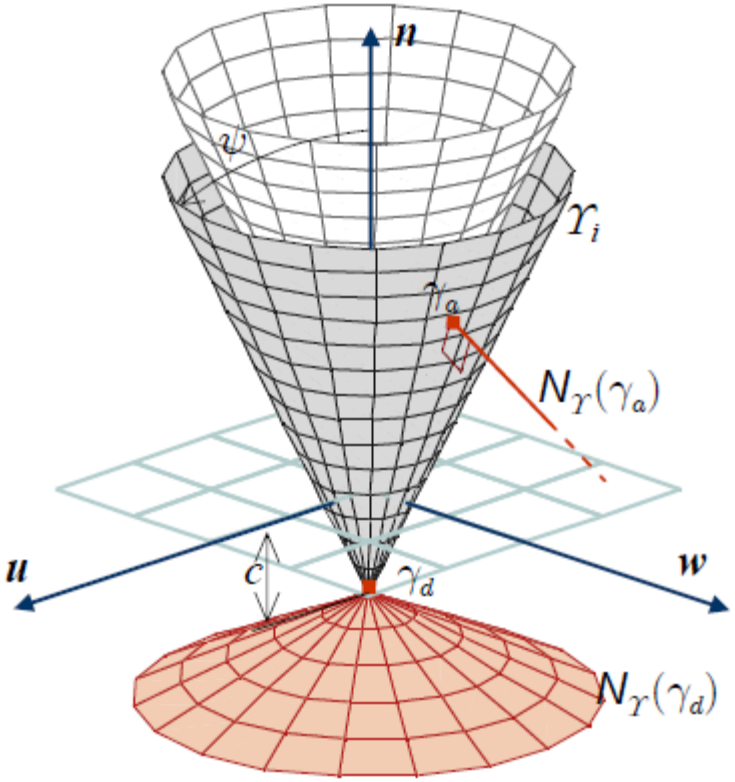
CONE COMPLEMENTARITY PROBLEM

$$(N\boldsymbol{\gamma}_{\mathcal{E}} + \boldsymbol{r}) \in -\Upsilon^{\circ} \quad \perp \quad \boldsymbol{\gamma}_{\mathcal{E}} \in \Upsilon$$

DVI Elasto-Plastic contact

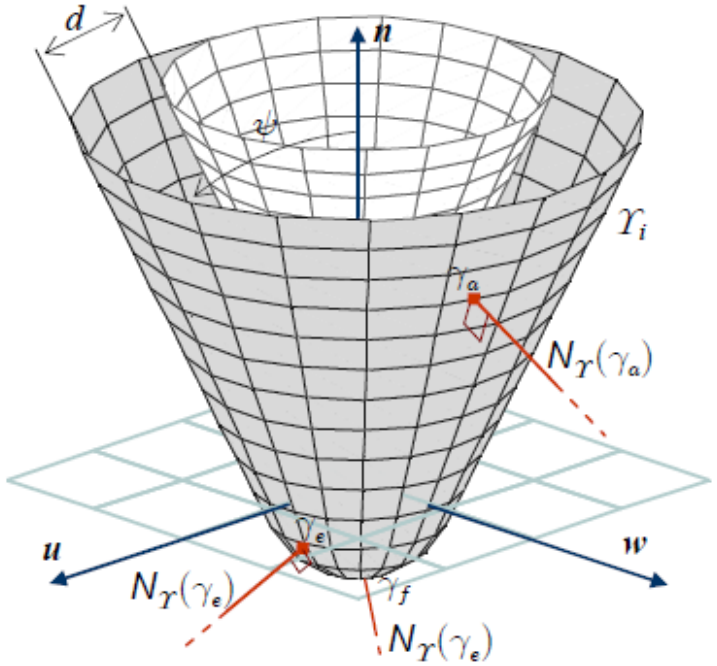
- DVI formulation can be extended to more general friction/contact laws

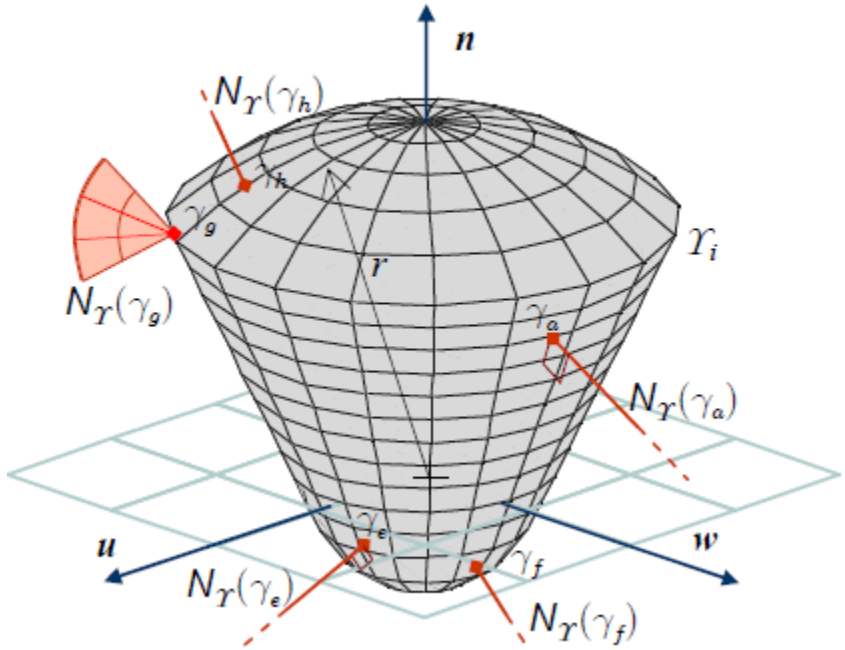




DVI Elasto-Plastic contact

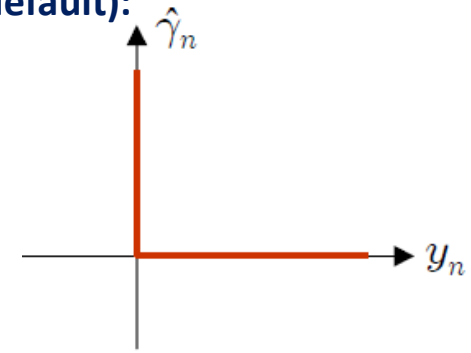
- DVI formulation can be extended to more general friction/contact laws



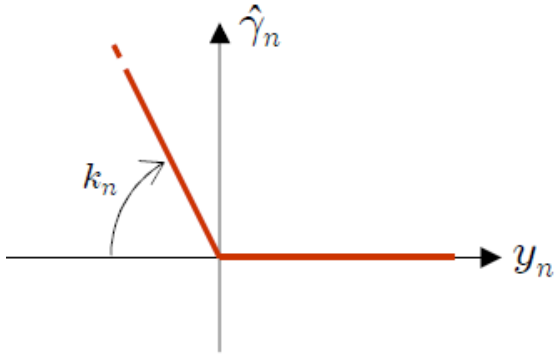


DVI advanced contact laws

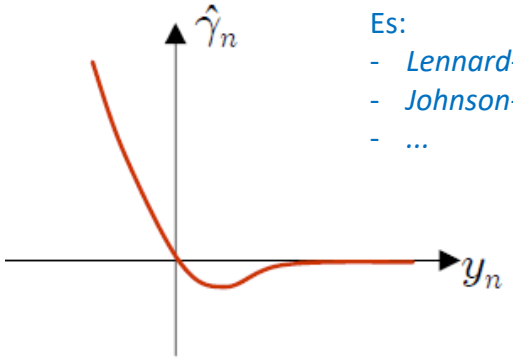
Rigid contact (default):



Compliant contact:

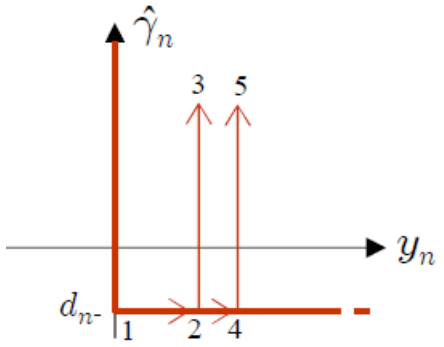


Nonlinear, with cohesion:



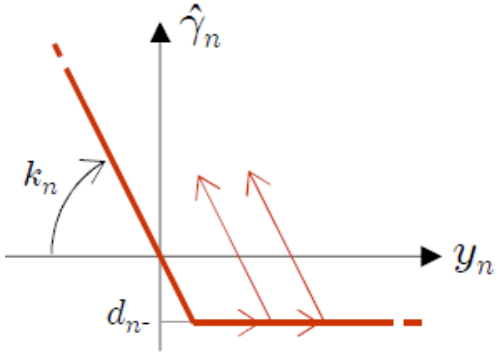
Es:
- *Lennard-Jones*
- *Johnson-Kendall-Roberts*
- *…*

Rigid, with plastic cohesion

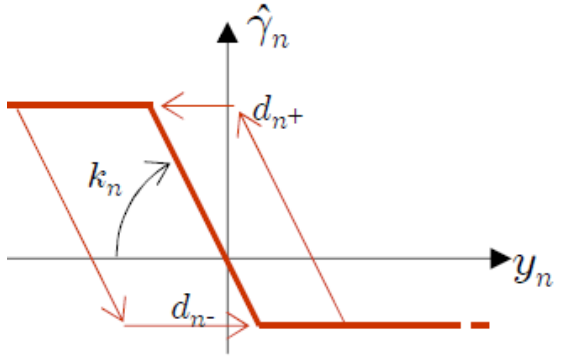


DVI advanced contact laws

Compliant, plastic cohesion



Compliant, plastic cohesion and compression



- *In general, DVI are useful for various reasons that are diffult to handle in DAE:*

- *very stiff or rigid contacts → set valued force laws → VI*
- *plasticity in contacts → yield surfaces → VI*
- *friction → set valued force laws → VI*

CCP solvers in Chrono

In the DVI-MDI time-stepper, a VI (or CCP) must be solved at each time step.

Which methods are available to solve a CCP in Chrono ?

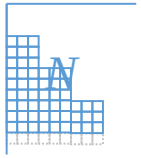
- Fixed-point solvers:
  - Projected-SOR
  - Projected-GaussSeidell
  - Projected-Symmetric-SOR

- Krylov spectral methods
  - Barzilai-Borwein
  - Nesterov Accelerated Projected Gradient Descent (APGD)

P-SOR solver for CCP $(N\gamma_{\mathcal{E}} + \boldsymbol{r}) \in -\Upsilon^{\circ} \quad \perp \quad \gamma_{\mathcal{E}} \in \Upsilon$

- Fixed point iteration with projection on cones:

$$\gamma^{r+1} = \lambda \Pi_{\Upsilon}\left(\gamma^{r} - \omega B^{r}\left(N\gamma^{r} + \boldsymbol{r} + K^{r}\left(\gamma^{r+1} - \gamma^{r}\right)\right)\right) + (1-\lambda)\gamma^{r}$$

With  $N = D^{T} M^{-1} D$

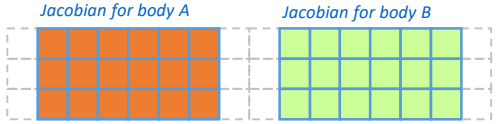
At each r-th iteration:

Loop on all i-th constraints

$$\delta^{i,r+1} = \gamma^{i,r} - \omega\eta_i \left(\boldsymbol{D}^{i,T} M^{-1} \left(\sum_{z=1}^{i-1} D^{z}\gamma^{z,r+1} + \sum_{z=i}^{n_{\mathcal{A}}} D^{z}\gamma^{z,r} + \tilde{\boldsymbol{k}}^{i} \right) + \boldsymbol{b}^{i} \right)$$

$$\gamma^{i,r+1} = \lambda \Pi_{\Upsilon^{i}}\left(\boldsymbol{\delta}^{i,r+1}\right) + (1-\lambda)\gamma^{i,r}$$

If i-th is a contact constraint:

$D^{i,T}$ 

$\boldsymbol{\gamma}_a^i$  $\boldsymbol{b}_a^i$ 

$\eta_a^i = \frac{3}{\mathrm{Trace}(g_a^i)}$ $g_a^i = D^{i,T} M^{-1} D^{i}$

If i-th is a scalar bilateral constraint

$D^{i,T}$ $\nabla\Psi^{i,T}$ 

γ_b^i  b_b^i 

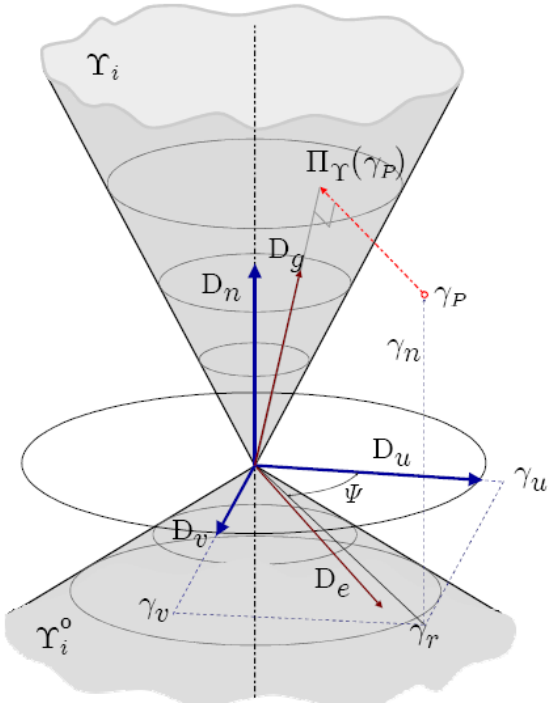
$\eta_b^i = \frac{1}{g_b^i}$ $g_b^i = D^{i,T} M^{-1} D^{i}$

P-SOR solver for CCP

- The projection operator must be non-extensive, i.e. lipschitzian with $\|f(a)\text{-}f(b)\| \leq \|a - b\|$

- *For each frictional contact constraint:*

$$\Pi_\Upsilon = \left\{ \Pi_{\Upsilon_1}(\boldsymbol{\gamma}_1)^T, \ldots \Pi_{\Upsilon_{n_\mathcal{A}}}(\boldsymbol{\gamma}^{n_\mathcal{A}})^T, \Pi_b^1(\gamma_b^1), \ldots, \Pi_b^{n_\mathcal{B}}(\gamma_b^{n_\mathcal{B}}) \right\}^T$$



- *For each bilateral constraint, simply do nothing.*

- The complete operator:

$$\begin{array}{l} \forall i \in \mathcal{A}(\boldsymbol{q}^{(l)}, \epsilon) \\ \quad \gamma_r < \mu_i \gamma_n \qquad\qquad\qquad\qquad \Pi_i = \boldsymbol{\gamma}_i \\ \quad \gamma_r < -\dfrac{1}{\mu_i}\gamma_n \qquad\qquad\qquad\qquad \Pi_i = \{0,0,0\} \\ \quad \gamma_r > \mu_i \gamma_n \wedge \gamma_r > -\dfrac{1}{\mu_i}\gamma_n \quad \Pi_{i,n} = \dfrac{\gamma_r \mu_i + \gamma_n}{\mu_i^2 + 1} \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \Pi_{i,u} = \gamma_u \dfrac{\mu_i \Pi_{i,n}}{\gamma_r} \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \Pi_{i,v} = \gamma_v \dfrac{\mu_i \Pi_{i,n}}{\gamma_r} \end{array}$$

P-SOR solver for CCP

- P-SOR in incremental efficient form

$$\delta^{i,r+1} = \gamma^{i,r} - \omega\eta_i \left(\boldsymbol{D}^{i,T} M^{-1} \left(\sum_{z=1}^{i-1} D^z \gamma^{z,r+1} + \sum_{z=i}^{n_A} D^z \gamma^{z,r} + \tilde{\boldsymbol{k}}^i \right) + \boldsymbol{b}^i \right)$$

$$\gamma^{i,r+1} = \lambda \Pi_{\Upsilon^i} \left(\boldsymbol{\delta}^{i,r+1} \right) + (1-\lambda)\gamma^{i,r}$$

Avoid these loops, otherwise each iteration would be O(n²)
Only one of these multiplier changes at each iteration…

We know that: $\boldsymbol{v} = M^{-1} D \ \boldsymbol{\gamma} + M^{-1} \tilde{\boldsymbol{k}}$ *..so we rewrite:*

$$\begin{aligned}
&\boldsymbol{\delta}^{i,r+1} = \left(\boldsymbol{\gamma}^{i,r} - \omega\eta^i \left(D^{i,T} \boldsymbol{v}^r + \boldsymbol{b}^i \right) \right); \\
&\boldsymbol{\gamma}^{i,r+1} = \lambda \Pi_{\Upsilon} \left(\boldsymbol{\delta}^{i,r+1} \right) + (1-\lambda)\boldsymbol{\gamma}^{i,r}; \\
&\Delta\boldsymbol{\gamma}^{i,r+1} = \boldsymbol{\gamma}^{i,r+1} - \boldsymbol{\gamma}^{i,r}; \\
&\boldsymbol{v} := \boldsymbol{v} + M^{-1} D^i \Delta\boldsymbol{\gamma}^{i,r+1}
\end{aligned}$$

Loop on all i-th constraints

P-SOR solver for CCP

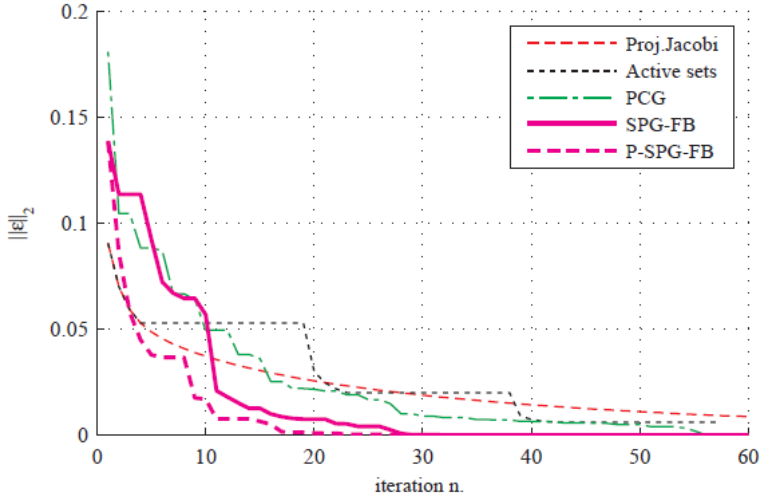
- Pseudocode

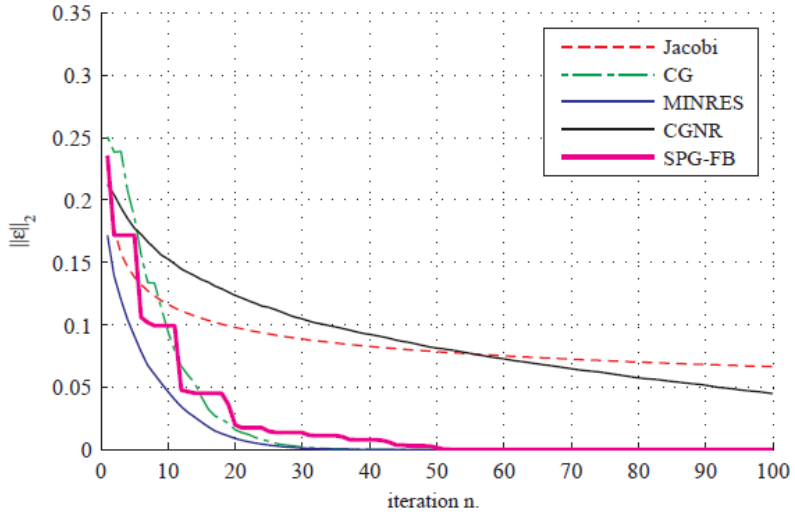
(1) *// Pre-compute some data for friction constraints*
(2) **for** $i := 1$ **to** $n_{\mathcal{A}}$
(3) $\quad \boldsymbol{s}_a^i = M^{-1} D^i$
(4) $\quad g_a^i = D^{i,T} \boldsymbol{s}_a^i$
(5) $\quad \eta_a^i = \frac{3}{\text{Trace}(g_a^i)}$
(6) *// Pre-compute some data for bilateral constraints*
(7) **for** $i := 1$ **to** $n_{\mathcal{B}}$
(8) $\quad s_b^i = M^{-1} \nabla \Psi^i$
(9) $\quad g_b^i = \nabla \Psi^{i,T} s_b^i$
(10) $\quad \eta_b^i = \frac{1}{g_b^i}$
(11)
(12) *// Initialize impulses*
(13) **if** warm start with initial guess $\boldsymbol{\gamma}_{\mathcal{E}}^*$
(14) $\quad \boldsymbol{\gamma}_{\mathcal{E}}^0 = \boldsymbol{\gamma}_{\mathcal{E}}^*$
(15) **else**
(16) $\quad \boldsymbol{\gamma}_{\mathcal{E}}^0 = \mathbf{0}$
(17)
(18) *// Initialize speeds*
(19) $\boldsymbol{v} = \sum_{i=1}^{n_{\mathcal{A}}} \boldsymbol{s}_a^i \boldsymbol{\gamma}_a^{i,0} + \sum_{i=1}^{n_{\mathcal{B}}} s_b^i \gamma_b^{i,0} + M^{-1} \tilde{\boldsymbol{k}}$

(21) *// Main iteration loop*
(22) **for** $r := 0$ **to** r_{max}
(23) $\quad$ *// Loop on frictional constraints*
(24) $\quad$ **for** $i := 1$ **to** $n_{\mathcal{A}}$
(25) $\quad\quad \boldsymbol{\delta}_a^{i,r} = \left(\boldsymbol{\gamma}_a^{i,r} - \omega \eta_a^i \left(D^{i,T} \boldsymbol{v}^r + \boldsymbol{b}_a^i\right)\right)$;
(26) $\quad\quad \boldsymbol{\gamma}_a^{i,r+1} = \lambda \Pi_{\Upsilon} \left(\boldsymbol{\delta}_a^{i,r}\right) + (1-\lambda) \boldsymbol{\gamma}_a^{i,r}$;
(27) $\quad\quad \Delta \boldsymbol{\gamma}_a^{i,r+1} = \boldsymbol{\gamma}_a^{i,r+1} - \boldsymbol{\gamma}_a^{i,r}$;
(28) $\quad\quad \boldsymbol{v} := \boldsymbol{v} + \boldsymbol{s}_a^{i^T} \Delta \boldsymbol{\gamma}_a^{i,r+1}$.
(29) $\quad$ *// Loop on bilateral constraints*
(30) $\quad$ **for** $i := 1$ **to** $n_{\mathcal{B}}$
(31) $\quad\quad \delta_b^{i,r} = \left(\gamma_b^{i,r} - \omega \eta_b^i \left(\nabla \Psi^{i,T} \boldsymbol{v}^r + b_b^i\right)\right)$;
(32) $\quad\quad \gamma_b^{i,r+1} = \lambda \Pi_{\Upsilon} \left(\delta_b^{i,r}\right) + (1-\lambda) \gamma_b^{i,r}$;
(33) $\quad\quad \Delta \gamma_b^{i,r+1} = \gamma_b^{i,r+1} - \gamma_b^{i,r}$;
(34) $\quad\quad \boldsymbol{v} := \boldsymbol{v} + s_b^{i^T} \Delta \gamma_b^{i,r+1}$.
(35)
(36) **return** $\boldsymbol{\gamma}_{\mathcal{E}}$, $\boldsymbol{v}$

P-SOR solver for CCP

- Very robust algorithm
- It supports redundant constraints
- It is very fast – good for robotics, etc.
- ...but it has slow convergence:





- Other methods, without the convergence stall, are needed when high precision is needed

P-SOR solver for CCP

- Use `SetSolverType()` to change the solver:

```
// change the solver to P-SOR:
my_system.SetSolverType(ChSystem::SOLVER_SOR);

// use high iteration number if constraints tend to ‘dismount’ or contacts interpenetrate:
my_system.SetMaxItersSolverSpeed(90);
```

P-SPG-FB solver for CCP

- In case of convexified problem (i.e. ‘associative flows’ as our CCP) one can express the VI as a constrained quadratic program:

$$\begin{aligned} \min \quad & f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} + \mathbf{x}^T \mathbf{b} \\ \text{s.t.} \quad & \mathbf{x} \in \mathcal{K} \end{aligned}$$

- One can use the Spectral Projected Gradient (SPG) method for solving it!
- It is a modified Barzilai-Borwein iteration

P-SPG-FB solver for CCP

- Our modified P-SPG-FB algorithm

- Supports premature termination with fall-back strategy (FB)

- Uses alternating step sizes

- Uses diagonal preconditioning (with isothropic cone scaling)

$$P = \overline{\text{diag}}(A)$$

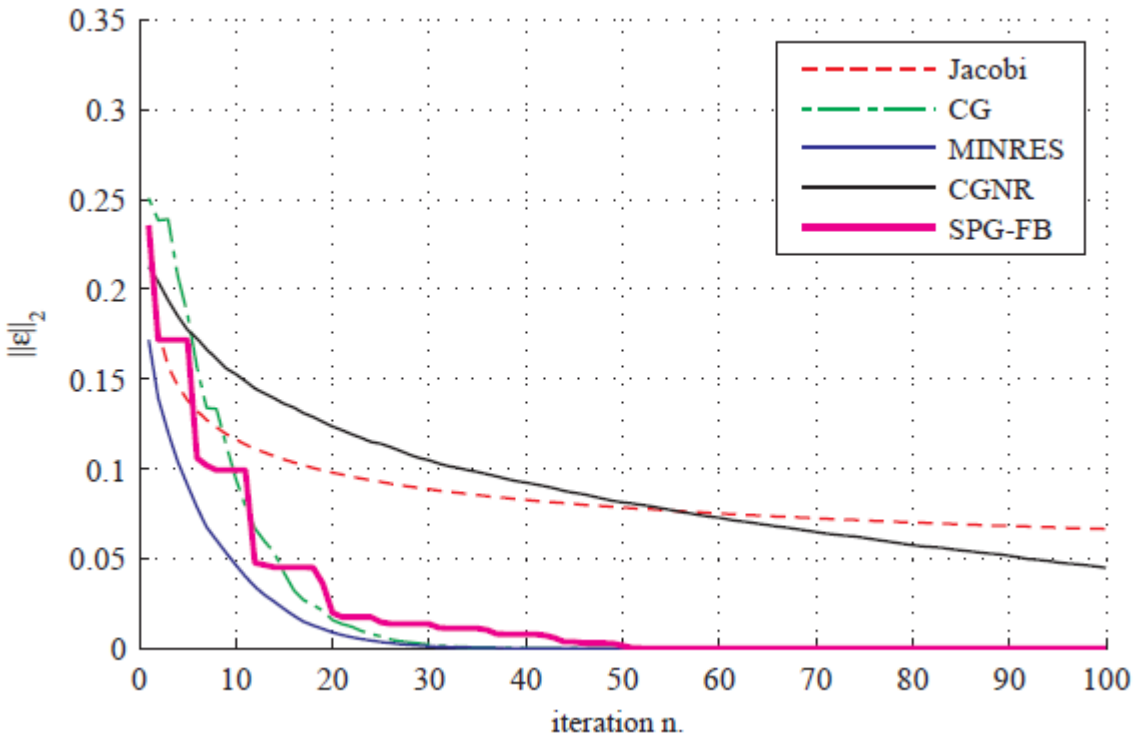
- Performs projection onto Lorentz cones

ALGORITHM P-SPG-FB(A, $\mathbf{b}$, $\mathbf{x}_0$, $\mathcal{K}$, $P \mapsto \mathbf{x}$)

$\mathbf{x}_0 := \Pi_{\mathcal{K}}(\mathbf{x}_0)$, $\mathbf{x}_{FB} = \mathbf{x}_0$,
$\hat{\alpha}_0 \in [\alpha_{min}, \alpha_{max}]$
$\mathbf{g}_0 := A\mathbf{x}_0 + \mathbf{b}$, $f(\mathbf{x}_0) = \frac{1}{2}\mathbf{x}_0^T A\mathbf{x}_0 + \mathbf{x}_0^T\mathbf{b}$, $w_0 = 10^{29}$
for $j := 0$ **to** N_{max}
  $\mathbf{p}_j = P^{-1}\mathbf{g}_j$
  $\mathbf{d}_j = \Pi_{\mathcal{K}}(\mathbf{x}_j - \hat{\alpha}_j\mathbf{p}_j) - \mathbf{x}_j$
  if $\langle \mathbf{d}_j, \mathbf{g}_j \rangle \geq 0$
    $\mathbf{d}_j = \Pi_{\mathcal{K}}(\mathbf{x}_j - \hat{\alpha}_j\mathbf{g}_j) - \mathbf{x}_j$
  $\lambda := 1$
  while line search
    $\mathbf{x}_{j+1} := \mathbf{x}_j + \lambda\mathbf{d}_j$
    $\mathbf{g}_{j+1} := A\mathbf{x}_{j+1} + \mathbf{b}$
    $f(\mathbf{x}_{j+1}) = \frac{1}{2}\mathbf{x}_{j+1}^T A\mathbf{x}_{j+1} + \mathbf{x}_{j+1}^T\mathbf{b}$
    if $f(\mathbf{x}_{j+1}) > \max_{i=0,\ldots,\min(j,N_{GLL})} f(\mathbf{x}_{j-i}) + \gamma\lambda\langle \mathbf{d}_j, \mathbf{g}_j \rangle$
      define $\lambda_{new} \in [\sigma_{\min}\lambda, \sigma_{\max}\lambda]$ and repeat line search
    else
      terminate line search
  $\mathbf{s}_j = \mathbf{x}_{j+1} - \mathbf{x}_j$
  $\mathbf{y}_j = \mathbf{g}_{j+1} - \mathbf{g}_j$
  if j is odd
    $\hat{\alpha}_{j+1} = \frac{\langle \mathbf{s}_j, P\mathbf{s}_j \rangle}{\langle \mathbf{s}_j, \mathbf{y}_j \rangle}$
  else
    $\hat{\alpha}_{j+1} = \frac{\langle \mathbf{s}_j, \mathbf{y}_j \rangle}{\langle \mathbf{y}_j, P^{-1}\mathbf{y}_j \rangle}$
  $\hat{\alpha}_{j+1} = \min(\alpha_{\max}, \max(\alpha_{\min}, \hat{\alpha}_{j+1}))$
  $w_{j+1} = \left|\left| \left[\mathbf{x}_{j+1} - \Pi_{\mathcal{K}}(\mathbf{x}_{j+1} - \tau_g\mathbf{g}_{j+1})\right] / \tau_g \right|\right|_2 = ||\varepsilon||_2$
  if $w_{j+1} \leq \min_{k=0,\ldots,j} w_k$
    $\mathbf{x}_{FB} = \mathbf{x}_{j+1}$
return $\mathbf{x}_{FB}$

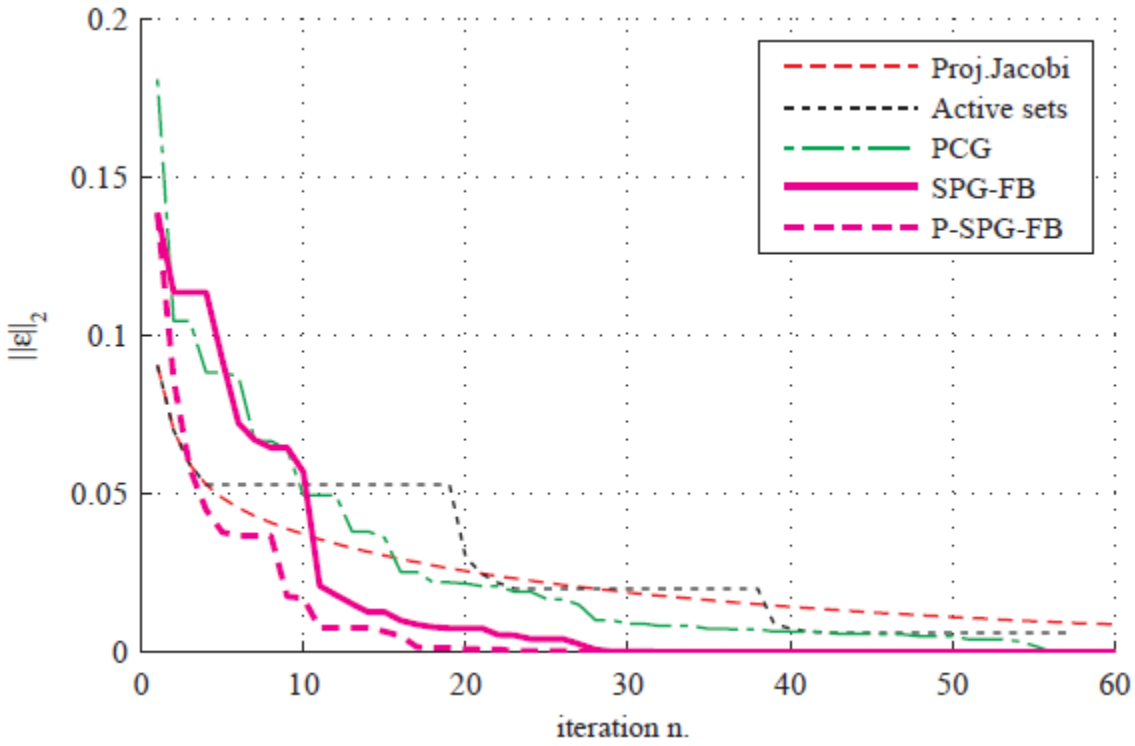
P-SPG-FB solver for CCP

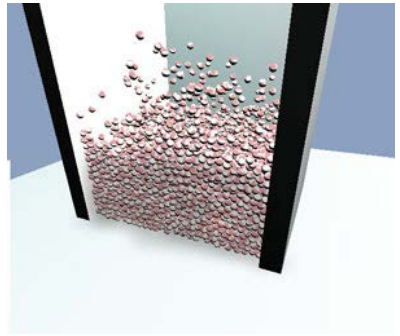
- Comparison with other Krylov solvers for simple <u>linear</u> case
- (only bilateral constraints):

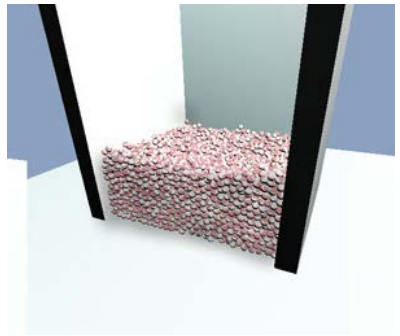


P-SPG-FB solver for CCP

- Comparison with other solvers for complemetarity problems
- (only unilateral contacts, no friction)

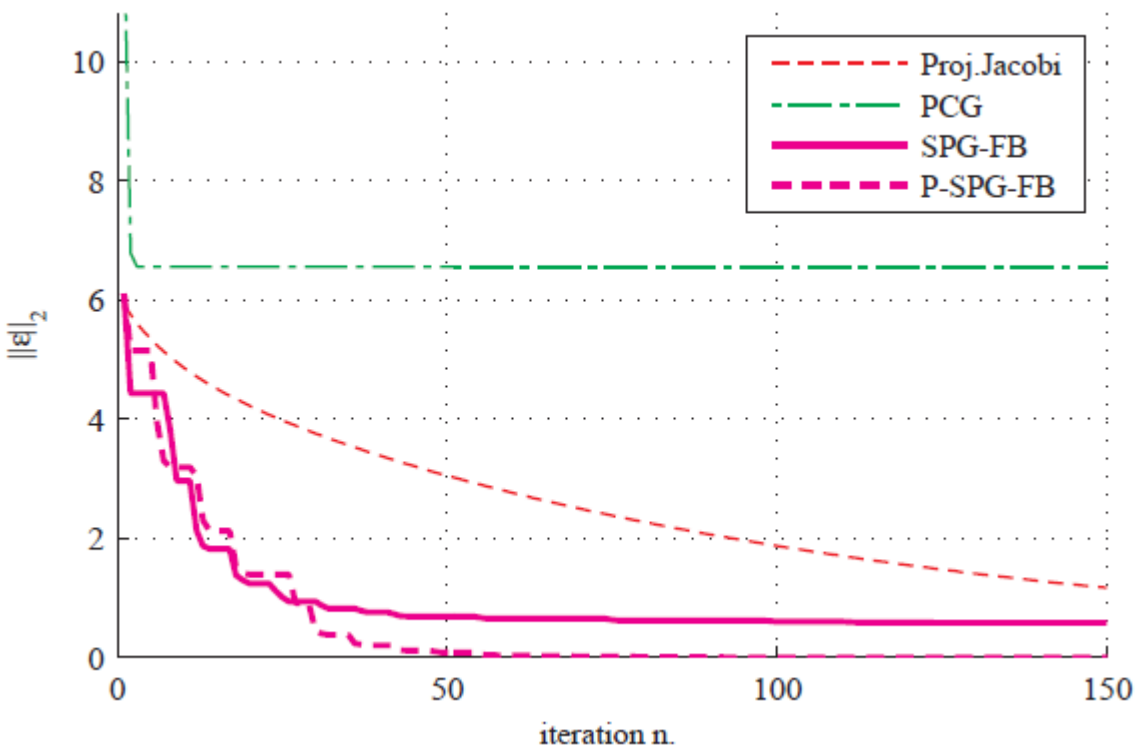


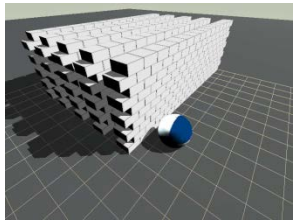


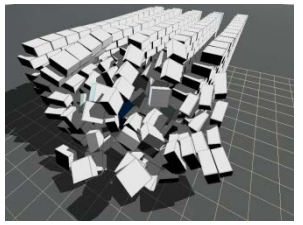


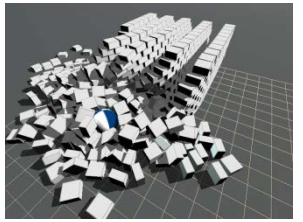
P-SPG-FB solver for CCP

- Comparison with other solvers for complemetarity problems
- (unilateral contacts AND friction - few solvers can handle it)



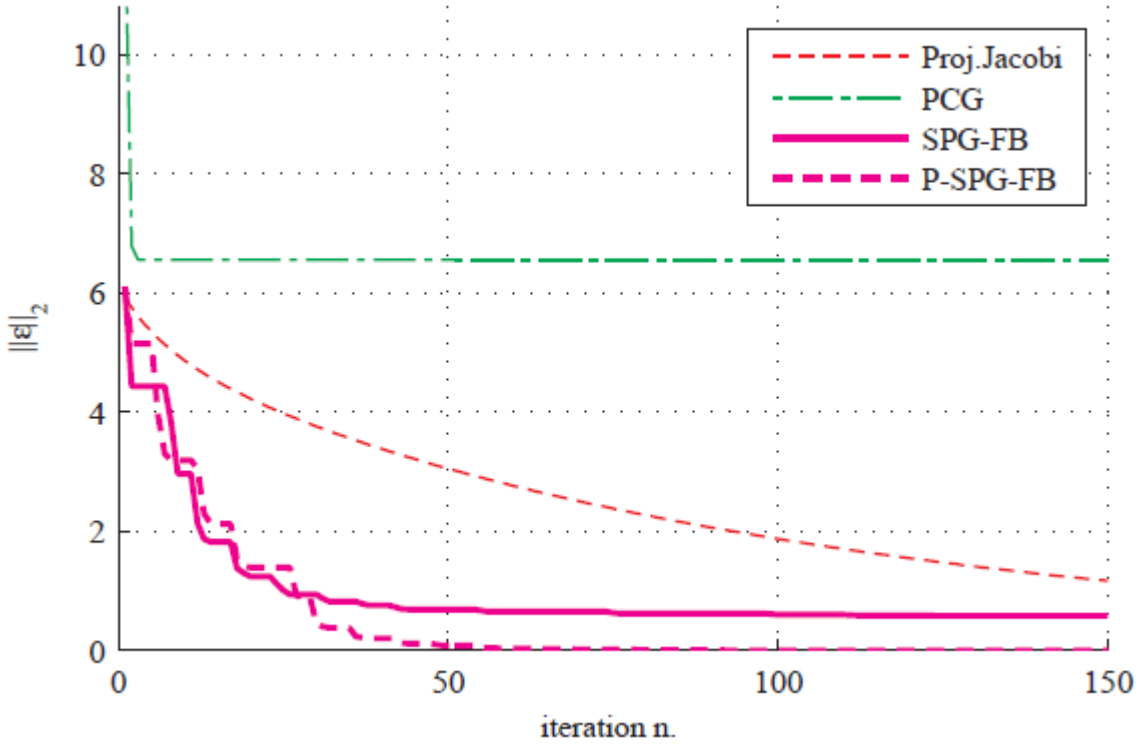


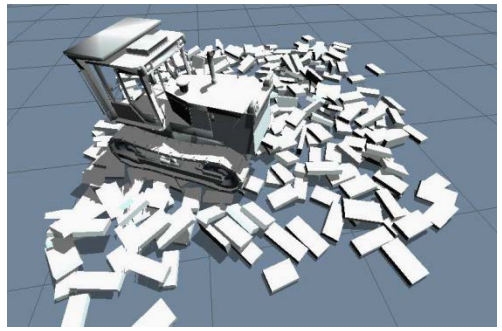




P-SPG-FB solver for CCP

- Effect of preconditioning:





P-SPG-FB solver for CCP

- Use `SetSolverType()` to change the solver:

```
// change the solver to Barzilai-Borwein P-SPG-FB:
my_system.SetSolverType(ChSystem::SOLVER_BARZILAIBORWEIN);

// will terminate iterations when this tolerance is reached:
my_system.SetTolForce(1e-7);

// use high iteration number if constraints tend to ‘dismount’ or contacts interpenetrate:
my_system.SetMaxItersSolverSpeed(110);
```

APGD solver for CCP

- Draws on the Nesterov's Accelerated Projected Gradient Descend
- It operates as a non-linear optimization for:

$$\begin{aligned} \min \quad & f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} + \mathbf{x}^T\mathbf{b} \\ \text{s.t.} \quad & \mathbf{x} \in \mathcal{K} \end{aligned}$$

- Properties of convergence are very similar to P-SPG-FB just presented.

APGD solver for CCP

- Use `SetSolverType()` to change the solver:

```
// change the solver to Nesterov' APGD:
my_system.SetSolverType(ChSystem::SOLVER_APGD);

// will terminate iterations when this tolerance is reached:
my_system.SetTolForce(1e-7);

// use high iteration number if constraints tend to 'dismount' or contacts interpenetrate:
my_system.SetMaxItersSolverSpeed(110);
```

Time-integration & solvers cheat-sheet

	LINEAR SYSTEM (DAE)		CCP (DVI)		Iterative	Redundant constraints	Optional Chrono module	Large systems	Time integrator compatibility	
		FEA*		FEA*					HHT	EULER_IMPLICIT_LINEARIZED
SOR		*	●●	*		●●		●●●	2	●●● DAE, DVI
BARZILAIBORWEIN		*	●●●	*		●●		●●●	● DAE	●●● DAE, DVI
APGD		*	●●●	*		●●		●●●	● DAE	●●● DAE, DVI
MINRES		●[1]		*		●●●		●●●	●● DAE	● DAE
MKL		●●●		*				●●	●●● DAE	● DAE
MUMPS		●●●		*		●		●●	●●● DAE	● DAE

* For FEA, the solver must support stiffness and damping matrices. Note that FEA in NSC is not yet possible at the moment.

1 The MINRES solver might converge too slow when using finite elements with ill-conditioned stiffness

2 The SOR solver is not precise enough for good HHT convergence, except for simple systems