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):**
  \[
  \frac{dx}{dt} = f(x, t)
  \]

- **A Differential Algebraic Equation (DAE)**
  - In implicit form:
    \[
    \frac{dx}{dt} = f(x, t) \\
    g(x, t) = 0
    \]
  - Introduces constraints \( g \)
  
  The Chrono case
  
  \[
  M\frac{dv}{dt} = f(q, v, t) + D_B\hat{\gamma}_B(t) \\
  C(q, t) = 0
  \]
DAE Explicit Integrators

- **Explicit** integrators:

\[ x(t + \Delta t) = F(x(t)) \]

- 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:

\[ G(x(t + \Delta t), x(t)) = 0 \]

- Can use large time steps
- More complex: they find \( x(t + \Delta t) \) by solving a nonlinear system \( G = 0 \) with Newton Raphson
  - Jacobians matrices of \( G \) 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:

\[
M \frac{dv}{dt} = f(q, v, t) + D_B \dot{\gamma}_B(t)
\]
\[
C(q, t) = 0
\]

• The HHT time discretization is:

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

• A-stable for:
  - 2-nd order accurate
  - An alternative formulation exists for position-level HHT
  - Adjustable parameter \( \alpha \): from 0 (no numerical damping; i.e., trapezoidal) to \(-1/3\) (max numerical damping)
The HHT integrator

- The \( G (x(t+\Delta t) \ , \ x(t)) = 0 \) non-linear problem to solve is:

\[
\begin{align*}
q^{l+1} - q^l - hv^l - \frac{h^2}{2} [(1-2\beta)a^l + 2\beta a^{l+1}] &= 0 \\
v^{l+1} - v^l - h [(1-\gamma)a^l + \gamma a^{l+1}] &= 0 \\
M\alpha^{l+1} + (1+\alpha)(C_q^T\lambda - f)^{l+1} - \alpha(C_q^T\lambda - f) &= 0
\end{align*}
\]

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

\[
\begin{bmatrix}
H & C_q^T \\
C_q & 0
\end{bmatrix}
\begin{bmatrix}
\Delta a^{l+1} \\
\Delta \lambda^{l+1}
\end{bmatrix} =
\begin{bmatrix}
\frac{1}{1+\alpha}(Ma^{l+1}) + (C_q^T\lambda - f)^{l+1} - \frac{\alpha}{1+\alpha}(C_q^T\lambda - f) \\
\frac{1}{\beta h^2}C^{l+1}
\end{bmatrix}
\]

\[
\begin{align*}
\alpha_{n+1}^{l+1} &= \alpha_n^{l+1} + \Delta a^{l+1} \\
\lambda_{n+1}^{l+1} &= \lambda_n^{l+1} + \Delta \lambda^{l+1} \\
v^{l+1} &= v^l + h [(1-\gamma)a^l + \gamma a^{l+1}] \\
q^{l+1} &= q^l + hv^l + \frac{h^2}{2} [(1-2\beta)a^l + 2\beta a^{l+1}]
\end{align*}
\]

\[
H = [M - \gamma h \nabla_v f^{l+1} - \beta h^2 \nabla_q f^{l+1} + \beta h^2 [(M\alpha)_q + (C_q^T\lambda)_q]]^{11}
\]
Configuring the Integrator in Chrono

• It can be changed with SetTimestepperType()
• Additional parameters via std::static_pointer_cast<...>(my_system.GetTimestepper())

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

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

```cpp
// 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:

```cpp
#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

```cpp
#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):

\[ x \in K \quad : \quad \langle F(x), y - x \rangle \geq 0 \quad \forall y \in K \]

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

(see Kinderlehrer and Stampacchia, 1980)

• Alternative formulation:

\[ x \in \mathcal{H}, \ g(x) \in -N_{\mathcal{H}}(x) \]

\[ N_{\mathcal{H}}(x) = \{ y \in \mathbb{R}^n : \langle y, x - z \rangle \geq 0, \forall z \in \mathcal{H} \} \]
Differential Variational Inequality

- Differential Variational Inequality (DVI)

\[
\frac{dx}{dt} = f(t, x, u) \\
u \in \text{SOL}\left(\mathbb{R}, F(t, x(t), \cdot)\right) \\
\Xi(x(0), x(T)) = 0
\]

where \( \text{SOL}\left(\mathbb{R}, F(t, x(t), \cdot)\right) \) is the set of solutions to the VI \( (\mathbb{R}, F(t, x(t), \cdot)) \)

- 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:
  \[ \frac{dx}{dt} = f(x,t) \rightarrow \text{ODE, DAE, ok} \]

• But some problems require **inequalities** or **inclusions** like
  \[ \frac{dx}{dt} \in f(x,t) \rightarrow \text{Differential Inclusion! (DI)} \]

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

\[ J \frac{d\omega}{dt} = M_f(\omega) + M_e(t) \]

where \( M_f = -M_{f_{\text{max}}} \) if \( \omega > 0 \)

and \( M_f = M_{f_{\text{max}}} \) if \( \omega < 0 \)

• All ODE integrator would never stop in \( \omega = 0 \)!
  It would just ripple about \( \omega = 0 \) ..

• Reducing \( \Delta t \) 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:

\[ \frac{dx}{dt} = f(x,t) \]  \(\rightarrow\) ODE, DAE, ok

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

\[ \frac{dx}{dt} \in f(x,t) \]  \(\rightarrow\) Differential Inclusion! (DI)

• **Example**: a flywheel with brake torque and applied torque (simple?!)  
  **Improved model!!**

\[ J \frac{d\omega}{dt} = M_f(\omega) + M_e(t) \]

where  
- \(M_f = -M_{f_{\text{max}}} \) for \(\omega > 0\)
- \(M_f = M_{f_{\text{max}}} \) for \(\omega < 0\)
- \(-M_{f_{\text{max}}} < M_f < M_{f_{\text{max}}} \) 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) \( \rightarrow \) measure distributions
  
  The velocity has ‘jumps’ \( \rightarrow \) function of bounded variation

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

\[
\frac{d\mu}{d\nu(t)} \in K(t)
\]

For singular decomposition of Borel measure \( \mu = m\nu + \mu_s \)

\( m(t) \in K(t) \quad \frac{d\mu_s}{d|\mu_s|}(t) \in K(t)\infty \)
Our DVI model

• Mechanical system with

  • Set $G_B$ of bilateral joints
  • Set $G_A$ of point contacts
  • External forces

\[ \dot{q} = \Gamma(q) \nu \]

\[ M(q) \frac{d\nu}{dt} = \sum_{i \in G_B} \hat{\gamma}_B^i \nabla \Psi^i + \sum_{i \in G_A} \hat{\gamma}_A^i D^i + f(t, q, \nu) \]

\[ \Psi^i(q, t) \in \emptyset, \quad i \in G_B \]

\[ \hat{\gamma}_A^i \in \text{SOL} (\gamma_i^i, F(t, q(t), \nu(t), \cdot)), \quad i \in G_A. \]
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

\[
M(v^{(l+1)}) - v^l = \sum_{i \in A(q^{(l)}, \epsilon)} (\gamma_n^i D_n^i + \gamma_u^i D_u^i + \gamma_v^i D_v^i) + \sum_{i \in G_B} \left( \gamma_b^i \nabla \Psi^i \right) + h f_i(t^{(l)}, q^{(l)}, v^{(l)})
\]

\[
0 = \frac{1}{h} \Psi^i(q^{(l)}) + \nabla \Psi^i T v^{(l+1)} + \frac{\partial \Psi^i}{\partial t}, \quad i \in G_B
\]

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

\[
(\gamma^i_n, \gamma^i_v) = \arg\min_{\mu^i_n, \gamma^i_v \geq 0} \mu^i_n \gamma^i_n \geq \sqrt{\gamma^i_u \gamma^i_u + \gamma^i_v \gamma^i_v} \quad i \in A(q^{(l)}, \epsilon)
\]

\[
q^{(l+1)} = q^{(l)} + h v^{(l+1)}
\]
Cone complementarity

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

\[ M(v^{(l+1)} - v^l) = \sum_{i \in A(q^{(l)}, \epsilon)} \left( \gamma^i_n D^i_n + \gamma^i_u D^i_u + \gamma^i_v D^i_v \right). \]

\[ + \sum_{i \in G_B} \left( \gamma^i_b \nabla \Psi^i \right) + hf \left( t^{(l)}, q^{(l)}, v^{(l)} \right) \]

\[ 0 = \frac{1}{h} \Psi^i(q^{(l)}) + \nabla \Psi^i T v^{(l+1)} + \frac{\partial \Psi^i}{\partial t}, \quad i \in G_B \]

\[ 0 \leq \frac{1}{h} \Phi^i(q^{(l)}) + \nabla \Phi^i T v^{(l+1)} - \mu^i \sqrt{(D^i_u T v)^2 + (D^i_v T v)^2} \]

\[ \perp \quad \gamma^i_n \geq 0, \quad i \in A(q^{(l)}, \epsilon) \]

\[ (\gamma^i_u, \gamma^i_v) = \arg\min_{\mu^i \geq \sqrt{(\gamma^i_n)^2 + (\gamma^i_v)^2}, \ i \in A(q^{(l)}, \epsilon)} \left[ v^T \left( \gamma^i_u D^i_u + \gamma^i_v D^i_v \right) \right] \]

\[ q^{(l+1)} = q^{(l)} + h v^{(l+1)}, \]

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:

\[ b_A = \left\{ \frac{1}{h} \Phi^{i_1}, 0, 0, \frac{1}{h} \Phi^{i_2}, 0, 0, \ldots, \frac{1}{h} \Phi^{i_{n_A}}, 0, 0 \right\} \]

\[ \gamma_A = \left\{ \gamma_{i_1}^{i_1}, \gamma_{u_1}^{i_1}, \gamma_{v_1}^{i_1}, \gamma_{i_2}^{i_1}, \gamma_{u_2}^{i_1}, \gamma_{v_2}^{i_1}, \ldots, \gamma_{i_{n_A}}^{i_1}, \gamma_{u_{n_A}}^{i_1}, \gamma_{v_{n_A}}^{i_1} \right\} \]

\[ b_B = \left\{ \frac{1}{h} \Psi^1 + \frac{\partial \Psi^1}{\partial t}, \frac{1}{h} \Psi^2 + \frac{\partial \Psi^2}{\partial t}, \ldots, \frac{1}{h} \Psi^{n_B} + \frac{\partial \Psi^{n_B}}{\partial t} \right\} \]

\[ \gamma_B = \{ \gamma_b^1, \gamma_b^2, \ldots, \gamma_b^{n_B} \} \]

\[ D_A = [D_i^1 | D_i^2 | \ldots | D_i^{n_A}], \quad i \in A(q^l, \epsilon) \]

\[ D^i = [D_n^i | D_u^i | D_v^i] \]

\[ D_B = [\nabla \Psi_1 | \nabla \Psi_2 | \ldots | \nabla \Psi_{n_B}], \quad i \in B \]

\[ b_\epsilon \in \mathbb{R}^{n_\epsilon} = \{ b_A, b_B \} \]

\[ \gamma_\epsilon \in \mathbb{R}^{n_\epsilon} = \{ \gamma_A, \gamma_B \} \]

\[ D_\epsilon = [D_A | D_B] \]

• We also introduce the convex cone

\[ \Upsilon = \left( \bigoplus_{i \in A(q^l, \epsilon)} \mathcal{F}^i \right) \bigoplus \left( \bigoplus_{i \in B} \mathcal{B}^i \right) \]

• ..and its polar cone:

\[ \Upsilon^\circ = \left( \bigoplus_{i \in A(q^l, \epsilon)} \mathcal{F}^{i \circ} \right) \bigoplus \left( \bigoplus_{i \in B} \mathcal{B}^{i \circ} \right) \]

\[ \mathcal{F}^i \quad \text{is } i\text{-th friction cone} \quad \mathcal{B}^i \quad \text{is } \mathbb{R}^{26} \]
Cone complementarity

• We introduce the Delassus operator $N$

$$N = D_T^e M^{-1} D_e$$

$$r = D_T^e M^{-1} \tilde{k} + b_e \quad \tilde{k}^{(l)} = Mv^{(l)} + h f_t(t^{(l)}, q^{(l)}, v^{(l)})$$

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

$$(N \gamma_e + r) \in -\gamma^o \perp \gamma_e \in \gamma$$
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:

Rigid, with plastic cohesion

Es:
- Lennard-Jones
- Johnson-Kendall-Roberts
- ...
In general, DVI are useful for various reasons that are difficult to handle in DAE:

- very stiff or rigid contacts $\rightarrow$ set valued force laws $\rightarrow$ VI
- plasticity in contacts $\rightarrow$ yield surfaces $\rightarrow$ VI
- friction $\rightarrow$ set valued force laws $\rightarrow$ 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 $(N\gamma_E + r) \in -\mathcal{Y}^c \perp \gamma_E \in \mathcal{Y}$

- Fixed point iteration with projection on cones:
  \[
  \gamma^{r+1} = \lambda \Pi_{\mathcal{Y}} \left( \gamma^r - \omega B^r \left( N\gamma^r + r + K^r (\gamma^{r+1} - \gamma^r) \right) \right) + (1 - \lambda) \gamma^r
  \]

  With \( N = D^T M^{-1} D \)

  At each \( r \)-th iteration:
  \[
  \delta^{i,r+1}_b = \gamma^{i,r}_b - \omega \eta_i \left( D^{i,T} M^{-1} \left( \sum_{z=1}^{i-1} D^{z,T} \gamma^{z,r+1} + \sum_{z=1}^{n_A} D^{z,T} \gamma^{z,r} + k^i \right) + b^i \right)
  \]

  \[
  \gamma^{i,r+1} = \lambda \Pi_{\mathcal{Y}_i} \left( \delta^{i,r+1}_i \right) + (1 - \lambda) \gamma^{i,r}
  \]

  Loop on all \( i \)-th constraints

If \( i \)-th is a contact constraint:
\[
D^{i,T} \quad \text{Jacobian for body A} \quad \text{Jacobian for body B}
\]
\[
\gamma^{i}_a \quad b^i_a
\]
\[
\eta^i_a = \frac{3}{\text{Trace}(g^i_a)} \quad g^i_a = D^{i,T} M^{-1} D^i
\]

If \( i \)-th is a scalar bilateral constraint:
\[
D^{i,T} \quad \nabla \Psi^{i,T} \quad \text{Jacobian for body A} \quad \text{Jacobian for body B}
\]
\[
\gamma^{i}_b \quad b^i_b
\]
\[
\eta^i_b = \frac{1}{g^i_b} \quad g^i_b = D^{i,T} M^{-1} D^i
\]
P-SOR solver for CCP

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

• For each frictional contact constraint:

\[
\Pi_T = \left\{ \Pi_{T_1}(\gamma_1)^T, \ldots, \Pi_{T_{n_A}}(\gamma_{n_A})^T, \Pi_{b_1}(\gamma_{b_1}), \ldots, \Pi_{b_{n_B}}(\gamma_{b_{n_B}}) \right\}^T
\]

• For each bilateral constraint, simply do nothing.

• The complete operator:

$\forall i \in A(q^{(i)}, e)$

\[
\begin{align*}
\gamma_r < \mu_i \gamma_n & \quad \Rightarrow \quad \Pi_i = \gamma_i \\
\gamma_r < -\frac{1}{\mu_i} \gamma_n & \quad \Rightarrow \quad \Pi_i = \{0, 0, 0\} \\
\gamma_r > \mu_i \gamma_n \land \gamma_r > -\frac{1}{\mu_i} \gamma_n & \quad \Rightarrow \quad \begin{cases} 
\Pi_{i,n} = \frac{\gamma_r \mu_i + \gamma_n}{\mu_i^2 + 1}, & \\
\Pi_{i,u} = \gamma_u \frac{\mu_i \Pi_{i,n}}{\gamma_r}, & \\
\Pi_{i,v} = \gamma_v \frac{\mu_i \Pi_{i,n}}{\gamma_r}.
\end{cases}
\end{align*}
\]
P-SOR solver for CCP

- P-SOR in incremental efficient form

\[
\delta_{i,r+1}^{i,r} = \gamma_{i,r}^{i,r} - \omega \eta^{i} \left( D_{i}^{i,T} M^{-1} \left( \sum_{z=1}^{i-1} D_{z}^{z,z,\gamma_{z,\gamma_{z}}^{r}}^{r} + \sum_{z=i}^{nA} D_{z}^{z,z,\gamma_{z,\gamma_{z}}^{r}}^{r} + k_{i}^{i} \right) + b_{i}^{i} \right)
\]

\[
\gamma_{i,r+1}^{i,r} = \lambda \Pi_{\gamma_{i}} \left( \delta_{i,r+1}^{i,r} \right) + (1 - \lambda) \gamma_{i,r}^{i,r}
\]

We know that: \( v = M^{-1}D \gamma + M^{-1} \tilde{k} \) .

So we rewrite:

\[
\delta_{i,r+1}^{i,r} = \left( \gamma_{i,r}^{i,r} - \omega \eta^{i} \left( D_{i}^{i,T} v^{r} + b_{i}^{i} \right) \right);
\]

\[
\gamma_{i,r+1}^{i,r} = \lambda \Pi_{\gamma} \left( \delta_{i,r+1}^{i,r} \right) + (1 - \lambda) \gamma_{i,r}^{i,r};
\]

\[
\Delta \gamma_{i,r+1}^{i,r} = \gamma_{i,r+1}^{i,r} - \gamma_{i,r}^{i,r};
\]

\[v := v + M^{-1}D \Delta \gamma_{i,r+1}^{i,r}\]

Avoid these loops, otherwise each iteration would be \( O(n^2) \).

Only one of these multiplier changes at each iteration...

Loop on all i-th constraints
P-SOR solver for CCP

- Pseudocode

\begin{align}
&S_i^a = M^{-1} D^i \\
g^a = D^{i,T} s^i \\
&\eta^a = \frac{1}{\text{trace}(g^a)} \\
&S_j^b = M^{-1} \nabla \Psi^j \\
g^b = \nabla \Psi^{i,T} s^i \\
&\eta^b = \frac{1}{s^i} \\
&S_k^c = \text{Initialization impulses} \\
&\gamma^0_c = 0 \\
&v = \sum_{i=1}^{n_A} S_i^a \gamma^0_a + \sum_{i=1}^{n_B} S_i^b \gamma^0_b + M^{-1} \bar{k} \\
&\delta^{i,r} = \left( \gamma^{i,r} - \omega \eta^a \left( D^{i,T} v^r + b^a \right) \right) \\
&\gamma^{i,r+1} = \lambda \Pi_T \left( \delta^{i,r} \right) + (1 - \lambda) \gamma^{i,r} \\
&\Delta^{i,r+1} = \gamma^{i,r+1} - \gamma^{i,r} \\
&v = v + \hat{s}^i_a \Delta \gamma^{i,r+1} \\
&S_k^c = \text{Initialization speeds} \\
&\gamma^0_c = 0 \\
&v = \sum_{i=1}^{n_A} S_i^a \gamma^0_a + \sum_{i=1}^{n_B} S_i^b \gamma^0_b + M^{-1} \bar{k} \\
&\delta^{i,r} = \left( \gamma^{i,r} - \omega \eta^b \left( \nabla \Psi^{i,T} v^r + b^b \right) \right) \\
&\gamma^{i,r+1} = \lambda \Pi_T \left( \delta^{i,r} \right) + (1 - \lambda) \gamma^{i,r} \\
&\Delta^{i,r+1} = \gamma^{i,r+1} - \gamma^{i,r} \\
&v = v + \hat{s}^i_b \Delta \gamma^{i,r+1} \\
&\text{return } \gamma_c, v
\end{align}
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:

```cpp
// 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:

\[
\min \quad f(x) = \frac{1}{2} x^T A x + x^T b \\
\text{s.t.} \quad x \in \mathcal{H}
\]

• 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 isotropic cone scaling)
- Performs projection onto Lorentz cones
P-SPG-FB solver for CCP

- Comparison with other Krylov solvers for simple linear case
- (only bilateral constraints):
P-SPG-FB solver for CCP

- Comparison with other solvers for complementarity 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:

```cpp
// 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:

\[
\min f(x) = \frac{1}{2} x^T A x + x^T b \\
\text{s.t.} \quad x \in \mathcal{H}
\]

• Properties of convergence are very similar to P-SPG-FB just presented.
APGD solver for CCP

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

```cpp
// 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

<table>
<thead>
<tr>
<th>LINEAR SYSTEM (DAE)</th>
<th>CCP (DVI)</th>
<th>Iterative</th>
<th>Redundant constraints</th>
<th>Optional Chrono module</th>
<th>Large systems</th>
<th>Time integrator compatibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEA*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FEA*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Iterative</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Redundant constraints</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optional Chrono module</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Large systems</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time integrator compatibility</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| SOR                  | *         |   ●●●   | *                     |                       |              | 2 DAE, DVI                   |
| BARZILAIRBOWEURIN    | *         |   ●●●   | *                     |                       |              |                             |
| APGD                 | *         |   ●●●   | *                     |                       |              |                             |
| MINRES               |           | 1        | *                     |                       |              | 2 DAE, DVI                   |
| MKL                  |           |   ●●●   | *                     |                       |              | 2 DAE, DVI                   |
| MUMPS                |           |   ●●●   | *                     |                       |              | 2 DAE, DVI                   |

* 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