1 solveSingular.m

```
function [H_e, allRestrictions, WMatrix, DiracBracketsInd,
    eqMotion] = ...
              SolveSingular(L, H_proposed, gaugeFix, varargin)
```

Brief description `SolveSingular` performs the following tasks:

- Calculates the Hessian matrix of the Lagrangian.
- Computes the canonical momenta.
- Determines primary constraints strictly (null vectors of the Hessian).
- Builds both the canonical and the extended Hamiltonians.
- Finds secondary constraints.
- Classifies constraints and reorganises their structure (null vectors of the Poisson-bracket matrix of the constraints).
- Computes the Poisson-bracket matrix of the constraints and its inverse.
- Derives canonical Dirac brackets.
- Generates equations of motion for first-class systems (Poisson brackets with the extended Hamiltonian).
- Generates equations of motion for second-class systems (Dirac brackets with the canonical Hamiltonian).
- Generates equations of motion for mixed systems (Dirac brackets using only second-class constraints).
- Allows gauge fixing to fully determine first-class or mixed systems.

All intermediate results (matrices, brackets, messages, and equations) are *printed* to the MATLAB console for the user.

Input arguments

Name	Type	Meaning
`L`	*sym*	Symbolic Lagrangian $L(q_i, \dot{q}_i)$.
`H_proposed`	*sym*	(Optional) symbolic canonical Hamiltonian guess. If omitted, non-symbolic, or `0`, an empty symbolic object is used and the routine computes H_{can} internally.
`gaugeFix`	*cell* of *sym*	(Optional) cell array of symbolic gauge conditions $\chi_j(q_i, p_i, t) = 0$. Leave empty to skip gauge fixing.
`varargin`	*char* ...	List of the generalized coordinates $q_1, q_2, \ldots, q_N$. Their conjugate momenta p_i are inferred automatically as P_{q_i}.

Table 1 `SolveSingular`: input arguments

Input Conventions

- **Coordinate and velocity names**
  Use plain letters for the coordinates (`x`, `y`, `z`) and append `_dot` to denote their time derivatives:

```
L = 1/2*(x_dot^2 + y_dot^2 + z_dot^2) + V(x,y,z);
```

- **Canonical momenta in Hamiltonians**
  Represent momenta with an uppercase `P` followed by an underscore and the corresponding coordinate:

```
H = 1/2*(P_x^2 + P_y^2 + P_z^2) - V(x,y,z);
```

- **Gauge fixing**
  Provide `gaugeFix` as a *cell array* of symbolic *equations*, e.g. $\{\xi,\ \eta\ ,\ldots\}$. :

```
gauge = {x + y, z - h}
```

- **Dynamic variables**
  Provide the generalized coordinates as a list of *names* (character vectors or strings), e.g. `'x'`, `'y'`, `'z'`. Do *not* pass symbolic variables themselves:
- **Symbolic declarations**
  Declare *all* symbols as real to ensure consistency and prevent MATLAB from introducing unwanted complex conjugates:

```
syms x y z x_dot y_dot z_dot P_x P_y P_z real
```

- **Explicit potential definitions**
  Although the code recognises a generic potential `V(x,y,z)`, it is safer to supply an explicit symbolic expression to avoid convoluted or conjugated results during simplification. Example:

```
syms k real
V  = k/2*(x^2 + y^2 + z^2);   % harmonic potential
L  = 1/2*(x_dot^2 + y_dot^2 + z_dot^2) + V;
H  = 1/2*(P_x^2 + P_y^2 + P_z^2) - V;
```

Output arguments

Name	Content
`H_e`	Symbolic *extended* Hamiltonian (including all primary constraints with multipliers).
`allRestrictions`	Cell array with every constraint detected (primary + secondary).
`WMatrix`	Global $W_{ab} = \{\phi_a, \phi_b\}$ evaluated on all constraints.
`DiracBracketsInd`	Symbolic matrix of canonical Dirac brackets $\{z_i, z_j\}_{DB}$ (empty if only first-class constraints).
`eqMotion`	Column vector containing $\dot{q}_i$ and $\dot{p}_i$ in symbolic form, already simplified on the final constraint surface.

Typical call pattern

```
% Example with coordinates x, y:
[H_e, phi, W, D, EOM] = SolveSingular(L, H_can, {x*y}, 'x', 'y');
```

1.1 Available `solveSingular` Versions

All variants share the *same* input arguments and return structure; only their internal algorithms differ.

`solveSingular`

The safest and most thoroughly tested implementation. It applies the full Dirac–Bergmann formalism *without* introducing any algebraic shortcuts. Use this variant for production runs; it has been validated on multiple benchmark systems and is considered the most robust.

`solveSingularS`

A streamlined variant that simplifies each constraint as soon as it is detected. Although extensively tested, these simplifications do not always preserve the correct dynamics. Run it *after* the standard `solveSingular` and compare outputs to ensure the results remain equivalent.

`solveSingularH`

A version equipped with an enhanced routine for constructing Hamiltonians. Unlike the other two variants—which request a user-supplied Hamiltonian when automatic construction fails—this implementation attempts to generate one automatically, recognising that multiple valid choices may exist. Preliminary tests confirm it yields correct but often highly intricate expressions, so treat its output as a guide rather than a final, user-friendly Hamiltonian.

2 Auxiliar functions

`H_singular`

```
function [H_e, allRestrictions] = H_singular(L, H_guess, varargin
    )
```

Brief description

Runs the full Dirac–Bergmann consistency algorithm for a (possibly singular) Lagrangian L.

- Diagnoses the Hessian of the Lagrangian to determine whether it is singular or regular.
- Computes the canonical momenta $P_i = \partial L / \partial \dot{q}_i$.

- Derives primary constraints and follows the Dirac–Bergmann algorithm to obtain all secondary, tertiary, etc. constraints.
- Classifies constraints (first-/second-class) and arranges them in a consistent structure.
- Builds the canonical Hamiltonian automatically (via `H_regular`) if the user does not supply one.
- Constructs the extended Hamiltonian $H_e = H_p + \sum u_i \phi_i$.
- Iteratively enforces consistency, fixing Lagrange multipliers or discovering new constraints until closure.
- Prints every intermediate matrix, bracket, and diagnostic message to the MATLAB console.

`PoissonBrackets`

```
function PoissonBracket = PoissonBrackets(F, G, varargin)
```

Brief description

- Accepts two symbolic expressions F and G together with the names of the generalized coordinates.
- Automatically constructs the corresponding canonical momenta P_{q_i} for every coordinate q_i supplied in `varargin`.
- Computes the Poisson bracket

$$\{F, G\} = \sum_i \left(\frac{\partial F}{\partial q_i} \frac{\partial G}{\partial P_{q_i}} - \frac{\partial F}{\partial P_{q_i}} \frac{\partial G}{\partial q_i} \right)$$

  in a straightforward loop and returns the expanded result.

Name	Type	Meaning
`F`	*sym*	First symbolic expression, depending on coordinates and/or canonical momenta.
`G`	*sym*	Second symbolic expression.
`varargin`	*char*	Names of the generalized coordinates $q_1, q_2, \ldots, q_N$. For each name the routine internally defines the corresponding canonical momentum P_{q_i}.

Table 2 `PoissonBrackets`: input arguments

`calculateWMatrix`

```
function [WMatrix, WMatrixInv, NewRestrictions,
    FirtsClassRestrictions, ...
```

Name	Type	Meaning
`PoissonBracket`	*sym*	Symbolic expression of the Poisson bracket $\{F, G\}$.

Table 3 `PoissonBrackets`: output

```
2          SecondClassRestrictions] = calculateWMatrix(
               allRestrictions, varargin)
```

Brief description

- Builds the constraint matrix $W_{ab} = \{\phi_a, \phi_b\}$ from an input list of symbolic constraints.
- Detects non-trivial null vectors of W; if present, contracts the original constraints into a new set of *true* first-class and second-class constraints.
- Classifies every constraint and computes the system's degrees of freedom $G_L = \frac{1}{2}\big(2N_q - N_2 - 2N_1\big)$.
- Returns W^{-1} automatically when *no* first-class constraints remain, enabling direct construction of Dirac brackets.
- Prints detailed diagnostics (null vectors, true constraints, W^{-1}, etc.) to the MATLAB console.

Name	Type	Meaning
`allRestrictions`	*cell*	Row or column cell array of symbolic constraint expressions $\phi_i(q, p)$.
`varargin`	*char*	Generalized coordinate names passed to `PoissonBrackets`; for every name q_i the routine assumes an associated momentum P_{q_i}.

Table 4 `calculateWMatrix`: input arguments

Name	Type	Meaning
`WMatrix`	*sym*	Constraint matrix built from the (possibly contracted) set of constraints.
`WMatrixInv`	*sym*	Inverse of W if no first-class constraints exist; empty otherwise.
`NewRestrictions`	*cell*	Cell array containing all *true* constraints when the null space of W is non-trivial; empty if the null space is trivial.
`FirtsClassRestrictions`	*cell*	Cell array of the resulting first-class constraints $\phi^{(1)}$.
`SecondClassRestrictions`	*cell*	Cell array of the resulting second-class constraints $\phi^{(2)}$.

Table 5 `calculateWMatrix`: output variables

`DiracBrackets`

```
function DiracBracket = DiracBrackets(q_i, P_j, phi, Minv,
    varargin)
```

Brief description

- Computes the canonical Poisson bracket $\{q_i, P_j\}$.
- If the inverse constraint matrix M^{-1} is unavailable (empty, non-finite, or symbolic with undefined entries), the routine stops here and returns the basic Poisson bracket.
- Otherwise evaluates the Dirac-bracket correction

$$\{q_i, P_j\}_D \;=\; \{q_i, P_j\} - \sum_{a,b} \{q_i, \phi_a\}\, M^{-1}_{ab}\, \{\phi_b, P_j\},$$

  using the set of second-class constraints ϕ_a.
- All intermediate derivatives are handled by `PoissonBrackets`; every symbol is treated as real to avoid unintended complex conjugates.

Name	Type	Meaning
`q_i`	*sym*	Coordinate whose Dirac bracket is desired.
`P_j`	*sym*	Canonical momentum to pair with `q_i`.
`phi`	*sym* vector or *cell*	Collection of second-class constraints ϕ_a.
`Minv`	*sym* or numeric	Inverse of the constraint-matrix block $M_{ab} = \{\phi_a, \phi_b\}$. If empty or non-finite, only the Poisson bracket is returned.
`varargin`	*char* ...	Names of the generalized coordinates $q_1, q_2, \ldots$ needed by `PoissonBrackets`.

Table 6 `DiracBrackets`: input arguments

Name	Type	Meaning
`DiracBracket`	*sym*	Symbolic expression of the Dirac bracket $\{q_i, P_j\}_D$ (or the plain Poisson bracket if M^{-1} is unavailable).

Table 7 `DiracBrackets`: output