~~Lights~~, ~~Camera~~,

Data, Lagrangian, Action!

Simulating mechanisms direct from a text file

John Pryce

Cardiff University

Joint work with Ned Nedialkov, McMaster University, Canada

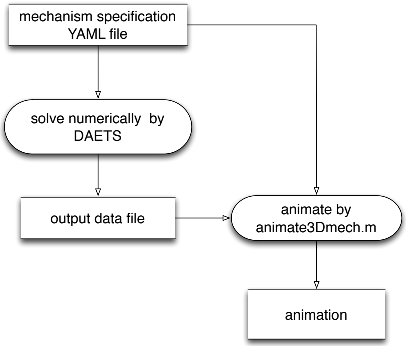
Summer Workshop on Interval Methods (SWIM2018)
University of Rostock, Germany
25–27 July 2018

Summary

- We solve a mechanism directly from a Lagrangian form
  - We don't ~~explicitly derive~~ equations of motion
  - No ~~symbolic algebra~~ to manipulate equations
  - We work in cartesian coordinates
- Lagrangian derived from a text file description—the Data
- Action means the resulting animation
- Our technology is based on
  - Automatic differentiation (AD, aka algorithmic differentiation)
  - DAETS (NN, JP, 2009–), a C++ solver for high-index differential-algebraic equations (DAEs)
    Based on Pryce's structural analysis, and Taylor series

System

- We have built a system that
  - Data: reads a text-file specification of a mechanism, initial conditions etc.
  - Creates Lagrangian; calls DAETS to solve and write output file
  - Action: visualizes by our MATLAB code animate3Dmech.m

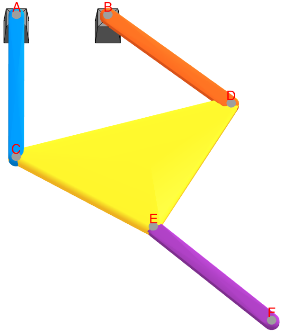


- Text-file is in YAML, a human-readable data serialization language

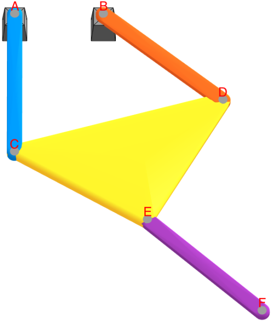
Outline

Example: Mechanism1

- 3 uniform thin rods AC, BD, EF of mass m and length ℓ
- A uniform triangular $(45°45°90°)$ plate CDE of mass M and short side ℓ
- Pin-jointed at C, D, E and at fixed points A, B on same horizontal level, distance L apart, from which system hangs
- Moves under gravity
- Animation



Dramatis Personae (Mechanism 1 specification in YAML)




```
Title: Mechanism1
Dimension: 2
PhysicalParams:
  l: 1      # length ℓ of rods
  L: 0.58   # distance of top pivots A, B
  m: 2      # mass of rods
  M: 5      # mass of plate
PartData:
  # coordinates of fixed points.
  Fixed: {A: [-L/2], B: [L/2]}
  Rigids:
   # Geom: local frame geometry
                                            #Dyna: centroid, mass, moment of inertia
   AC,BD,EF: {Geom: [ [l] ], Dyna: [ [l/2], m, m*l**2/12 ]}
   CDE:
    Geom: [ [l*sqrt(2)], [l/sqrt(2), -l/sqrt(2)] ]
    Dyna: [ [l/sqrt(2), -l/(3*sqrt(2))], M, M*l**2/9 ]
AppliedForces:
  Gravity:                                  #turns it on with default value in SI units
```

Act 1 Scene 1 + Stage Directions (IVs; solver & animation settings)

```
ProblemData:  # integration interval, etc.
   t0:        0
   tend:      60
   # Guesses for C, Ċ etc. where "fixed" means IV not guess - don't change it
   positions: {C: [[-l/sqrt(2),fixed],-l], D: [l/sqrt(2),-l],
     F: [[0,fixed], -l*(1+1/sqrt(2))]}
   velocities: {C: [[-6,fixed],0], D: [2,0], F: [[3,fixed],0]}
   #That starts it in equilibrium position & gives a sideways "kick"
SolverParams : # to guide DAETS
  Integration:
    tol:     1e-12
    order:   20
  OutFile: #says output 0th and & 1st derivative of each moving point
    points: [C: 2, D: 2, E: 2, F: 2]
    tformat: '% .17e'  #to 17 sig figs
    qformat: '% .17e'
Animation: #this guides animate3Dmech.m
  view:  [0, 90] # camera azimuth, elevation
  Skeleton:
    zscale:          0.02 #vertical scale, e.g. of fixed pivots
    fleshoutwid:     0.05 #says how wide "thin" things are drawn
```

Lagrangian mechanics theory

The Lagrangian function

$$L = T - V$$

is a powerful way to describe a mechanical system

- $T =$ total kinetic energy, in terms of velocities and possibly positions
- $V =$ total potential energy, caused by conservative (energy preserving) forces depending only on system position
- May also have holonomic (not velocity-dependent) constraints on motion, and/or external applied forces
- Simplifies modelling!

Lagrangian cont.

- Describe configuration at time t by vector $\mathbf{q} = (q_1, \ldots, q_{n_q})$ of generalised position coordinates
- Vector $\dot{\mathbf{q}}$ is generalised velocities
- Assumptions from previous slide imply

$$L = T - V, \qquad \text{with } T = T(\mathbf{q}, \dot{\mathbf{q}}),\ V = V(\mathbf{q})$$

plus any constraints on motion:

$$0 = C_j(t, \mathbf{q}), \qquad j = 1 : n_c$$

Lagrangian cont.

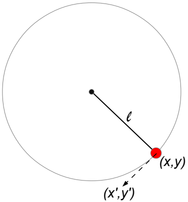
- Whatever coordinates chosen, variational "stationary action" principle gives (n_q+n_c) Euler–Lagrange equations of motion:

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} + \sum_{j=1}^{n_c} \lambda_j \frac{\partial C_j}{\partial q_i} = Q_i(t), \qquad i = 1:n_q \tag{1}$$

$$C_j(t, \mathbf{q}) = 0, \qquad j = 1:n_c \tag{2}$$

- λ_j are *Lagrange multipliers* for the constraints
  $Q_i(t)$ are *generalised external force* components, if any (whose definition also involves $\partial/\partial q_i$)
- If $n_c > 0$ the system is of first kind and is an index 3 DAE
- If $n_c = 0$ the system is of second kind, reducible to an ODE

Example: free motion of simple pendulum



Taking $\mathbf{q} = (x, y)$ = cartesian coordinates of pendulum bob (of mass m) with y downward, gives

$$T = \tfrac{1}{2}m(\dot{x}^2 + \dot{y}^2), \qquad V = -mgy$$

$$L = \tfrac{1}{2}m(\dot{x}^2 + \dot{y}^2) + mgy$$

with one constraint that we write

$$0 = C = \tfrac{1}{2}(x^2 + y^2 - \ell^2)$$

Euler–Lagrange, on dividing through by m, give pendulum DAE

$$\left.\begin{aligned} 0 = A &= \ddot{x} + x\lambda && \text{from } 0 = \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{x}} - \frac{\partial L}{\partial x} + \lambda\frac{\partial C}{\partial x} \\ 0 = B &= \ddot{y} + y\lambda - g && \text{from } 0 = \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{y}} - \frac{\partial L}{\partial y} + \lambda\frac{\partial C}{\partial y} \\ 0 = 2C &= x^2 + y^2 - \ell^2 && \end{aligned}\right\} \quad (3)$$

Pendulum cont.

Alternatively, taking $\mathbf{q} = (\theta) =$ angle of pendulum from downward vertical, gives

$$T = \tfrac{1}{2}m(\ell\dot{\theta})^2, \qquad V = -mg\ell\cos\theta$$
$$L = \tfrac{1}{2}m(\ell\dot{\theta})^2 + mg\ell\cos\theta$$

with no constraints. Then Euler–Lagrange lead to an ODE form

$$\ddot{\theta} = -\frac{g}{\ell}\sin\theta \qquad \text{from } 0 = \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial\dot{\theta}} - \frac{\partial L}{\partial\theta}$$

which is equivalent to the DAE

For one pendulum the angle model wins, but for $n > 1$ pendula (in a chain) the cartesian model is **much** simpler . . .

Example: $n > 1$ pendula, in 3D cartesians

- $\mathbf{r}_i = (x_i, y_i, z_i)$ position of ith bob (with z downward)
- Generalized coordinates
  $\mathbf{q} = (\mathbf{r}_1, \ldots, \mathbf{r}_n) = (x_1, y_1, z_1, \ldots, x_n, y_n, z_n)$
- 1st kind formulation is

$$\left.\begin{array}{lr} L = \frac{1}{2} m \sum_{i=1}^{n} |\dot{\mathbf{r}}_i|^2 + mg \sum_{i=1}^{n} z_i & \\ 0 = C_j = |\mathbf{r}_j - \mathbf{r}_{j-1}|^2 - \ell^2, & j = 1 : n \end{array}\right\} \tag{4}$$

  where $\mathbf{r}_0 = \mathbf{0}$, and $|\cdot|^2$ is the squared length of a 3-vector
  - Constraints say the rods have length ℓ
  - $3n$ coordinate variables, n Lagrange multipliers
    Hence second-order DAE of size $4n$ and index 3
- DAETS with our "Lagrangian facility" solves (4) as written

Example: The same, in ODE form

- Use spherical polar coordinates (θ_i, ϕ_i) for rod i
  - θ_i is rod's angle with downward vertical
  - ϕ_i is angle of rotation from the xz plane
- With $\mathbf{q} = (\theta_1, \phi_1, \ldots, \theta_n, \phi_n)$ we can get rid of the constraints
- $2n$ coordinates, so $4n$ ODEs when reduced to first-order
- Formulation is way more complex. E.g. KE is

$$T = \frac{1}{2} m\ell^2 \sum_{k=1}^{n} \left| \sum_{i=1}^{k} \begin{pmatrix} \cos\theta_i\,\dot\theta_i\cos\phi_i - \sin\theta_i\sin\phi_i\,\dot\phi_i \\ \cos\theta_i\,\dot\theta_i\sin\phi_i + \sin\theta_i\cos\phi_i\,\dot\phi_i \\ -\sin\phi_i\,\dot\phi_i \end{pmatrix} \right|^2$$

  and you still have the $\partial/\partial q_i, \partial/\partial \dot{q}_i$ stuff to do
- It seems any other way to remove the constraints will use angles in some form

Summary advantages of a high-index DAE code

- Index measures how difficult is to solve a DAE compared to an ODE (index 0)
- ODEs and index-1 DAEs seen as "easy" to solve and high-index DAEs as "hard"
- So considerable effort is spent to find coordinates giving a 2nd-kind Lagrangian and deriving equations of motion
- But mathematical model often simpler in cartesian coordinates
- DAETS handles resulting 1st kind Lagrangian systems easily
- We streamline this by DAETS's Lagrangian facility and mechanism facility ...

First… Lagrangian facility

- On top of DAETS, this solves directly from L and the constraints
  - Builds on AD package FADBAD++ which is integral to DAETS
  - Computes derivatives “on the fly” behind the scenes
  - No symbolic algebra
- Very efficient (thanks to Xiao Li, M.Sc. McMaster U)
  - common subexpression elimination through operator overloading in AD
  - sparse algorithms in AD
  - sparse linear algebra

Next... mechanism facility

Building on the previous, our **goal** is to

1. *Theory*: Express Lagrangian of mechanism (robot arm etc.) by x, y, z coordinates of chosen reference points (RPs) on its parts
   - In 2D, a rigid body's position is fixed by 2 points on it in "general position"; in 3D, by 3 points; etc
   - Express its PE (if relevant) and KE in terms of world-positions and velocities of such RPs
   - Hence multi-body $L = L(\mathbf{q}, \dot{\mathbf{q}})$ with $\mathbf{q} =$ (suitable RPs)
   - ... entirely in cartesian coordinates
2. *Practice*: Create
   - text file syntax/semantics for describing a class of mechanisms
   - C++ API to convert this to a Lagrangian that DAETS then handles by the Lagrangian facility

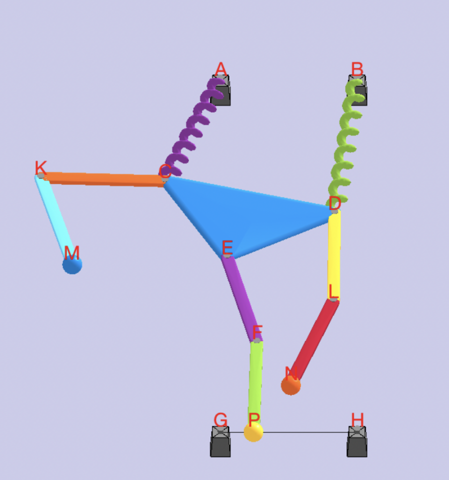
What mechanism facility currently provides

We are tidying up 2D before moving to 3D

- Named Points; same point on two parts means joined there
  - One declares some points fixed in world frame
  - All others are assumed moving
- Parts (a part's name is the list of points on it)
  - Rigid body (dynamics = mass, centroid, moment of inertia)
  - Particle (dynamics = mass only)
  - Spring (dynamics = stiffness, rest-length)
    Optionally mass, then dynamics of a stretchable uniform rod
- Forces
  - Constant (in world frame or in local frame) force
    Applied at a named point of body, fixed in local frame
  - Constant torque on a rigid part
  - Time-varying forces still to come—need compile/link stage
- Collinears
  - Constrains 3 or more points to lie on a straight line
    Useful for specifying various kinds of joint

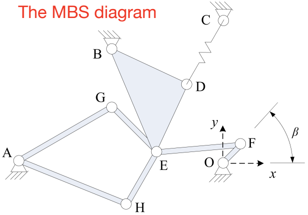
Mechanism2

- Rigid rods CK, KM, DL, LM, EF, FP and triangle plate CDE
- Springs AC, BD
- Point masses at M, N, P
- Collinear G, H, P
- Animation



Andrews squeezing mechanism

The MBS diagram



The original diagram.
K3 is star-shaped and
K5, K7 are not straight!

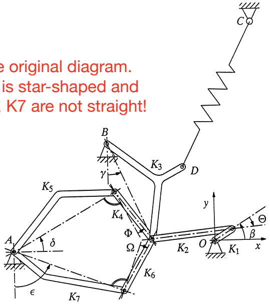


Fig. 9.1. Seven body mechanism (Schiehlen 1990, with permission)

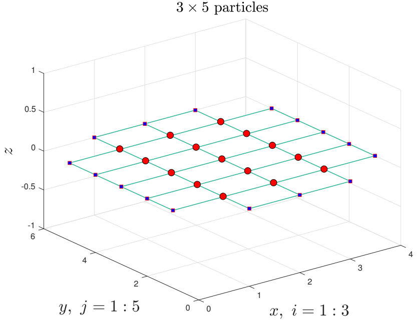
- Part of MBS Multi-Body Systems Benchmark in OpenSim
- Also in the Test Set for IVP Solvers, where . . .
  - It is formulated as an index 3 DAE in angle coordinates
  - Equations are not pretty at all
- We modeled and solved in cartesian coordinates
- Confirmed very close agreement between the two solutions
- ▸ Animation

Further examples

- These are in 3D so don't use the mechanism facility, but do use the Lagrangian facility
- John P extended the basic rigid body reference point theory to 3D and we are working out the implications
- Indeed we can do rigid body dynamics in any dimension
- QR factorization is key to the algorithm
  No quaternions
- Example: multi-pendulum made of genuinely 3D rods joined by Universal Joints (Hooke–Cardan joints)
  UJs transmit torque round a bend—permit 2 DOF of relative angular position but forbid relative rotation about rod-axes
- Animation

Larger example: Particle-spring system

- Rectangular grid of $m \times n$ particles connected by damped springs
- A test for cloth simulation in movies



- Particle (i, j) is attached to

$$(i \pm 1, j) \text{ and } (i, j \pm 1) \text{ for } i = 1\!:\!m,\ j = 1\!:\!n$$

- Index $i = 0$ or $m + 1$, resp. $j = 0$ or $n + 1$, means a fixed position

Particle-spring cont.

- Each particle (i, j)
  - coordinates $\boldsymbol{r}_{ij} = (x_{ij}, y_{ij}, z_{ij})$ full 3D motion
  - mass M
- Each spring
  - stiffness k
  - length at rest l
  - damping $k_d \times$ stretch-rate (except the boundary ones)
- Spacing Δx and Δy between particles in x and y directions
- Initially all particles at rest in xy plane, we push the middle particle upwards
- 90×90 particles, $24\,300$ second-order ODEs
- Animation

Particle-spring cont

- Lagrangian is

$$L = \tfrac{1}{2}M\sum_{i=1}^{m}\sum_{j=1}^{n}|\dot{\boldsymbol{r}}_{ij}|^2 - Mg\sum_{i=1}^{m}\sum_{j=1}^{n} z_{ij}$$
$$- \tfrac{1}{2}k\left[\sum_{i=1}^{m}\sum_{j=0}^{n}(|\boldsymbol{r}_{i,j+1}-\boldsymbol{r}_{ij}|-l)^2 + \sum_{j=1}^{n}\sum_{i=0}^{m}(|\boldsymbol{r}_{i+1,j}-\boldsymbol{r}_{ij}|-l)^2\right]$$

- We use Rayleigh's dissipative function

$$R = \tfrac{1}{2}k_d\sum_{i=1}^{m}\sum_{j=1}^{n-1}|\dot{\boldsymbol{r}}_{i,j+1}-\dot{\boldsymbol{r}}_{ij}|^2 + \tfrac{1}{2}k_d\sum_{j=1}^{n}\sum_{i=1}^{m}|\dot{\boldsymbol{r}}_{i+1,j}-\dot{\boldsymbol{r}}_{ij}|^2$$

- We encode L and R—that's all
- DAETS solves a sparse, second-order ODE of size $3 \cdot m \cdot n$

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{\boldsymbol{r}}_{ij}} - \frac{\partial L}{\partial \boldsymbol{r}_{ij}} + \frac{\partial R}{\partial \dot{\boldsymbol{r}}_{ij}} = 0, \qquad i = 1:m,\ j = 1:n$$

Conclusions and further work

- "Data, Lagrangian, Action" works as a practical tool.
  We aim now to develop 3D, and improve user interface
- An implication for teaching the subject:
  - Since high-index DAEs are now as easy to solve as ODEs, a Lagrangian formulation needn't avoid constraints
  - So rigid-body mechanical systems can be modeled in cartesian coordinates, which is simpler
  - This makes the concept so easy that Lagrangian stuff can be taught at undergraduate level
    After doing some simple cases from first principles, students can experiment with a tool like the mechanism facility
- But there's a big new thing for Ned & me to learn . . .

The natural coordinates community

Ned & I have only recently learnt of work going on for ~ 30 years, on multi-body modelling by "natural coordinates"

- Their basic object is the rigid-body shift+rotate map $\mathcal{R} \mapsto \mathbf{p}(t) + Q(t)\mathcal{R}$ stored as $3 + 3 \times 3 = 12$ scalars per body
- Our basic object is point $\mathbf{x}(t)$. In 3D, three points define body position, making $3 \times 3 = 9$ scalars per body (less, as points are shared)
- They mostly have Finite Elements background so very different mind-set from ours.
  Tasks considered serious but DAETS probably handles well:
  - Finding e.g. equilibrium configuration
  - Analysis of kinematic chains

References

- JP, NN, G. Tan, X. Li. How AD can help solve differential-algebraic equations
  Optimization Methods and Software, 2018, DOI
- JP, NN. Write-up of this as a paper, same title
  We are wondering where to submit it to.
- YouTube channel Multi-body Lagrangian Simulations
  - Outer planets
  - Gravitating masses in 2D
  - Spring mass with 3 pendula
  - ...

Appendix: YAML text for Andrews squeezing mechanism

Complete description except for **PhysicalParams** values and **Title**.
The boxed text specifies the topology, geometry and dynamics.

```
Dimension: 2

PartData:
 Fixed: { O: [], A: [xa, ya ], B: [xb, yb], C: [ xc, yc] }
 Rigids:
  OF: { Geom: [ [rr] ], Dyna: [ [ ra ], m1, i1 ] }
  FE: { Geom: [ [d ] ], Dyna: [ [ da ], m2, i2 ] }
  BED: { Geom: [ [ss], [sc, sd] ], Dyna: [ [sa, sb], m3, i3 ]}
  EG: { Geom: [ [e] ], Dyna: [ [ea ], m4, i4 ] }
  AG: { Geom: [ [zt] ], Dyna: [ [ta, tb], m5, i5 ] }
  HE: { Geom: [ [zf] ], Dyna: [ [zf-fa], m6, i6 ] }
  AH: { Geom: [ [u] ], Dyna: [ [ua, -ub], m7, i7 ] }
 Springs:
  CD: [ c0, l0 ]

AppliedForces:
 ConstTorques: { OF: mom }
```

```
ProblemData:
 t0: 0.0
 tend: 0.03
 positions:
  E: [-2e-02, 1e-03]
  F: [rr*cos(beta0), [rr*sin(beta0),fixed]]
  G: [-3e-02, 1e-02]
  H: [-3e-02, -1e-02]
 velocities: # all 0's by default
```

```
SolverParams:
 Mode: solve
 Integration:
  tol: 1e-14
  order: 17

 Display:
  tableau: false
  IVs: false
  consIVs: false
  solution: false
  stats: false
  progress: false
 OutFile:
  tformat: '% .17e'
  qformat: '% .17e'
  points: [ D: 2, E: 2, F: 2, G: 2, H: 2 ]
  angles: [ OF: 1 ]

Animation:
 view: [-5, 27]
 physParamsToShow: [$beta0, $c0, $mom]
 Skeleton:
  zscale: 0.0005
  fleshoutwid: 0.0015
  Skels:
   BED: {path: XBXEXDX, newpts: [ X , [sa, sb] ] }
   AG: {path: YAYGY, newpts: [ Y , [ta, tb] ] }
   AH: {path: ZHZAZ, newpts: [ Z , [ua, -ub] ] }
```