Modeling and Simulation of Gait (part 1)

MCE 493/593 & ECE 492/592
Prosthesis Design and Control
November 20, 2014

Antonie J. (Ton) van den Bogert
Mechanical Engineering
Cleveland State University

Today

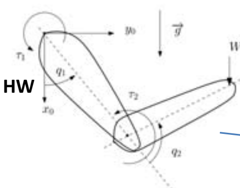
- Simple gait models
  - "four ways to use a 2-link pendulum"
- Homework assignment
- Limit cycles, root finding, stability
- Ready made musculoskeletal models
  - in Opensim
  - in Matlab
  - how to add a prosthesis
- Next week:
  - simulation and optimization of a hydraulic knee
  - what made Oscar Pistorius so fast?

Simulation is an important tool

- Like everywhere else in engineering
- Prosthetics
  - impedance controller fits human gait data, but would it actually work?
  - optimize mechanical design parameters
  - optimize control parameters

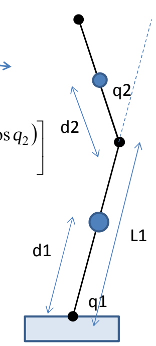
Two-link pendulum

Sep 4 HW



$$\mathbf{M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau}$$

I changed the definition of q1



W=0

$$\mathbf{M(q)} = \begin{bmatrix} I_1 + I_2 + m_1 d_1^2 + m_2\left(L_1^2 + d_2^2 + 2d_2 L_1 \cos q_2\right) & I_2 + m_2 d_2\left(d_2 + L_1 \cos q_2\right) \\ I_2 + m_2 d_2\left(d_2 + L_1 \cos q_2\right) & I_2 + m_2 d_2^2 \end{bmatrix}$$

$$\mathbf{C(q,\dot{q})} = \begin{bmatrix} -m_2 d_2 L_1 \sin q_2 \dot{q}_2 & -m_2 d_2 L_1 \sin q_2 \left(\dot{q}_1 + \dot{q}_2\right) \\ m_2 d_2 L_1 \sin q_2 \dot{q}_1 & 0 \end{bmatrix}$$

$$\mathbf{g(q)} = \begin{bmatrix} m_1 g d_1 \cos q_1 + m_2 g\left(L_1 \cos q_1 + d_2 \cos\left(q_1 + q_2\right)\right) \\ m_2 g d_2 \cos\left(q_1 + q_2\right) \end{bmatrix}$$

State space, first order dynamics

$$\mathbf{M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = u}$$

State $\mathbf{x} = \begin{pmatrix} q_1 \\ q_2 \\ \dot{q}_1 \\ \dot{q}_2 \end{pmatrix} = \begin{pmatrix} \mathbf{q} \\ \dot{\mathbf{q}} \end{pmatrix}$ Dynamics : $\dot{\mathbf{x}} = \begin{pmatrix} \dot{\mathbf{q}} \\ \ddot{\mathbf{q}} \end{pmatrix} = \begin{pmatrix} \dot{\mathbf{q}} \\ \mathbf{M}^{-1}(\mathbf{u} - \mathbf{C(q,\dot{q})\dot{q}} - \mathbf{g(q)}) \end{pmatrix}$

$= \mathbf{f(x,u)}$

Matlab:

```
q = x(1:2);
qd =x(3:4);
M = ....expressions with q
C = ...expressions with q,qd
g = ...expressions with q
qdd = M\(u - C*qd - g);          inv(M)*(u - C*qd - g);
xd = [qd ; qdd];
```

much faster than:

Matlab

First order dynamics: tlp_dyn.m
To draw the model: tlp_draw.m

For simulating standing and stance phase of gait:

- Foot "glued" to ground
- Link 1 = shank, Link 2 = thigh
- Rest of body mass in a point at the top of the thigh

Simulation code: tlpsim.m

Standing: stability analysis

First order dynamics: $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}(\mathbf{x}, t)) = \mathbf{f}(\mathbf{x}, t)$

When x = (pi/2,0,0,0) and u = 0,0 → f(x,u) is zero

So this is a steady state. How stable is it?

Define **y**: deviation from reference state $\mathbf{x}_r$

First order Taylor expansion (linearization):

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{x}_r, \mathbf{u}_r) + \left[\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right]_{\mathbf{x}_r} \mathbf{y}$$

$$\dot{\mathbf{y}} = \left[\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right]_{\mathbf{x}_r} \mathbf{y}$$

$$\dot{\mathbf{y}} = \mathbf{A}\mathbf{y}, \quad with: \quad \mathbf{A} = \left[\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right]_{\mathbf{x}_r} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_N}{\partial x_1} & \cdots & \frac{\partial f_N}{\partial x_N} \end{pmatrix}$$

a 4x4 Jacobian matrix

Solution: $\mathbf{y}(t) = \sum \mathbf{p}_i e^{\lambda_i t}$ λ_i : eigenvalues of A

System is stable if all λ_i are in the left half of the complex plane

```
% find the eigenvalues for q1=pi/2 and q2=0
   df_dx = zeros(4,4);         % initialize Jacobian
   x = [pi/2; 0; 0; 0];        % the state we are interested in
   f = tlp_dyn(0, x);          % the function f at this state
   hh = 1e-7;                  % a small finite difference
   for i=1:4
      xsave = x;
      x(i) = x(i) + hh;                %  add hh to state variable i
      df_dx(:,i) = (tlp_dyn(0,x) - f)/hh;  % column i of the Jacobian
      x = xsave;
   end
   [eigenvectors,eigenvalues] = eig(df_dx)
```

```
eigenvectors =
   -0.0179    0.0179    0.3021    0.3021
    0.0360   -0.0360    0.0148    0.0148
    0.4453    0.4453   -0.9520    0.9520
   -0.8945   -0.8945   -0.0466    0.0466
eigenvalues =
  -24.8779         0         0         0
         0   24.8779         0         0
         0         0   -3.1508         0
         0         0         0    3.1508
```

Standing: proportional-derivative control

```
function u = controller_pd(t,x)
    % controller, generates torques u as a func
    q = x(1:2);            % angles
    qd = x(3:4);           % angular velocities
    qr = [pi/2 ; 0];       % desired posture
    Kp = 1000;             % proportional gain
    Kd = 100;              % derivative gain
    u = -Kp*(q-qr) - Kd*qd;
end
```

$$\mathbf{u} = \begin{pmatrix} -K_p(q_1 - q_{1r}) - K_d\dot{q}_1 \\ -K_p(q_2 - q_{2r}) - K_d\dot{q}_2 \end{pmatrix}$$

$K_p = 1000$ Nm/rad

$K_d = 100$ Nms/rad

$q_{1r} = \pi / 2$

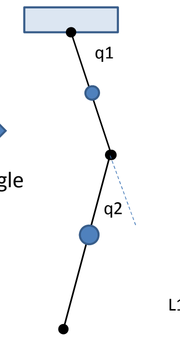
$q_{2r} = 0$

```
eigenvectors =
   0.0011 + 0.0000i   0.0499 + 0.0000i   0.6813 + 0.0000i   0.6813 + 0.0000i
  -0.0022 + 0.0000i  -0.1026 + 0.0000i   0.4101 - 0.0048i   0.4101 + 0.0048i
  -0.4467 + 0.0000i  -0.4346 + 0.0000i  -0.3723 + 0.3621i  -0.3723 - 0.3621i
   0.8947 + 0.0000i   0.8934 + 0.0000i  -0.2216 + 0.2206i  -0.2216 - 0.2206i
eigenvalues =
   1.0e+02 *
  -4.1332 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  -0.0871 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i  -0.0055 + 0.0053i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i  -0.0055 - 0.0053i
```

Gait: stance phase and swing phase

Stance phase
- same model as standing
- probably different control

Swing phase
- same model equations, but
  - q1, q2 are thigh angle and knee extension angle
  - rest of body mass not included in model
- hip pivots on ground
- Link 1 = thigh
- Link 2 = shank
- assumption: pelvis constant speed



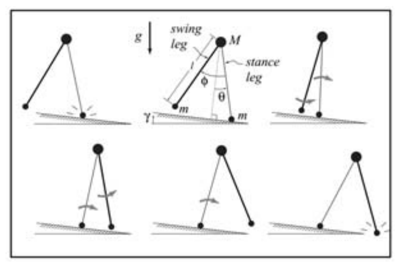
In this posture, q1≈ –60°, q2≈ –30°

Homework:

Modify tlpsim.m
- Simulate the swing phase
  - You could use invdyn2d (requires data file) to determine **x** at start of swing
- How good is the swing with zero torques?
- Improve the swing with a PD controller
  - get the foot on the ground at the right time and the right place
- Compare to human motion from data file

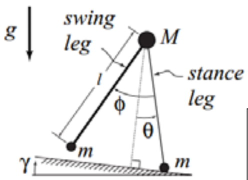
The simplest walking model

Garcia et al. (1998) The Simplest Walking Model: Stability, Complexity, and Scaling. *ASME J Biomech Eng* 120, 281-288. (PDF)



http://ruina.tam.cornell.edu/research/

Equations of motion



in the limit $m \to 0$

and time scaled by $\sqrt{l/g}$

$$\ddot{\theta}(t) - \sin(\theta(t) - \gamma) = 0$$

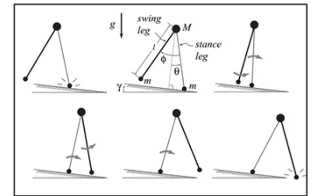
$$\ddot{\theta}(t) - \ddot{\phi}(t) + \dot{\theta}(t)^2 \sin\phi(t) - \cos(\theta(t) - \gamma)\sin\phi(t) = 0$$

One model parameter: γ

State $x = \begin{pmatrix} \theta \\ \dot{\theta} \\ \phi \\ \dot{\phi} \end{pmatrix}$ Dynamics : $\dot{x} = \begin{pmatrix} \dot{\theta} \\ \sin(\theta - \gamma) \\ \dot{\phi} \\ \sin(\theta - \gamma) + (\dot{\theta}^2 - \cos(\theta - \gamma))\sin\phi \end{pmatrix}$

No control, "passive dynamics". pdw_sim.m.

Heelstrike collision



Swing foot hits the ground when:

$$\phi(t) - 2\theta(t) = 0$$

- Impulsive GRF decelerates swing foot velocity to zero
- Angular velocities of links will change instantaneously, based on conservation of angular momentum
- Swing foot becomes the stance foot

All of this is accomplished by:

$$\underbrace{\begin{bmatrix} \theta \\ \dot{\theta} \\ \phi \\ \dot{\phi} \end{bmatrix}^{+}}_{\text{state after impact}} = \underbrace{\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & \cos 2\theta & 0 & 0 \\ -2 & 0 & 0 & 0 \\ 0 & \cos 2\theta(1 - \cos 2\theta) & 0 & 0 \end{bmatrix}}_{\mathbf{h}} \underbrace{\begin{bmatrix} \theta \\ \dot{\theta} \\ \phi \\ \dot{\phi} \end{bmatrix}^{-}}_{\text{state before impact}}$$

"jump matrix"

Limit cycle

- Is possible at certain values of gamma
- Find an initial state, such that the system returns to the same state
  - this means the cycle will repeat for ever!
- Limit cycles are found by root finding

  $\mathbf{x}_i$: a state just after heelstrike (both feet at ground)

  Simulate the system until the next heelstrike

  New state $\mathbf{x}_{i+1}$ is a function of $\mathbf{x}_i$:

  $\mathbf{x}_{i+1} = \mathbf{F}(\mathbf{x}_i)$

  "Poincare map"
  "walk map" (Garcia)
  "stride function" (McGeer)

  Limit cycle starting point $\mathbf{x}$ is "fixed point" of $\mathbf{F}$:

  $\mathbf{F}(\mathbf{x}) - \mathbf{x} = 0$

  solved by Newton-Raphson method
  Matlab: fsolve

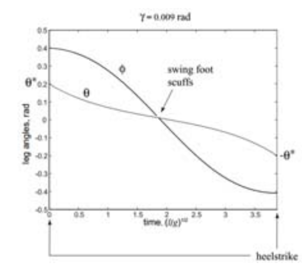
Example of a limit cycle

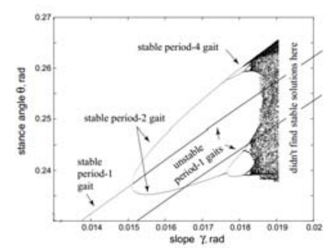
Limit cycle can be stable or unstable.

- By linearization we can show: limit cycle is stable when all (complex) eigenvalues of Jacobian d$\mathbf{F}$/d$\mathbf{x}$ are inside the unit circle.

- Stable period-1 limit cycles (gaits) exist for $\gamma < 0.0151$ rad.

- At larger slope we see period-2 gaits (limping / galloping), then period-4 gaits. etc. , then chaos.





Control of an unstable limit cycle

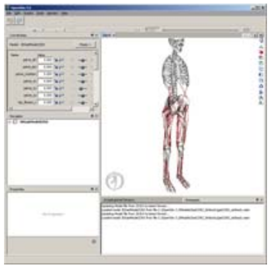
- Continuous time control
  - Generate a torque that counteracts a deviation from the planned state trajectory
  - For example: PD controller, like we did in standing
  - You lose the nice passive behavior
- Discrete time

$$\mathbf{x}_{i+1} = \mathbf{F}(\mathbf{x}_i)$$

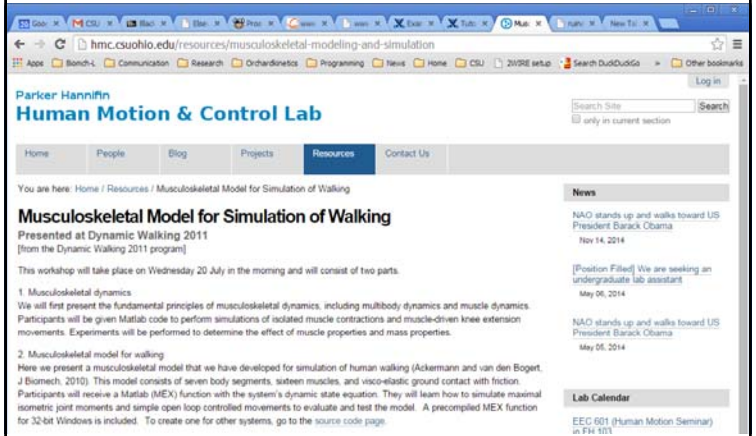
  - Once in each cycle, determine deviation from the fixed point of the cycle
  - Perform an action that pushes system towards fixed point
  - This can be done, for example in mid-swing

More complex models

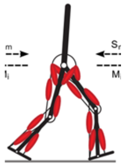
- Make your own equations
  - Matlab symbolic toolbox
  - www.motiongenesis.com
  - www.pydy.org
- Opensim (www.simtk.org)
  - software and models
  - e.g. Dynamic Walking challenge
  - multibody dynamics and muscle dynamics
  - mostly 3D models
  - "gait2392"
  - C++ and Matlab API



Or check out this one



States and controls



- Model has 9 DOF and 16 muscles
  - 50 state variables, 16 control inputs
  - musculoskeletal dynamics with ground contact

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x},\mathbf{u}) \qquad \mathbf{x} = \begin{pmatrix} \mathbf{q}_{1..9} \\ \dot{\mathbf{q}}_{1..9} \\ \mathbf{L}_{\mathbf{CE}1..16} \\ \mathbf{a}_{1..16} \end{pmatrix} \qquad \mathbf{u} = \begin{pmatrix} u_1 \\ \vdots \\ u_M \end{pmatrix}$$

- C-MEX function, Matlab API, vectorized

```
% function [xdot, grf, stick, forces, moments] = gait2de(x,u,M);
%
% Full documentation is in the file gait2de_reference.pdf
%
% Explicit differential equation for 2D musculoskeletal model : xdot = f(x,u,M)
% with additional optional outputs.
%
% Input:
%   x          State of the model (50 x N)
%   u          Neural excitations for the muscles (16 x N)
%   M          (optional) Additional actuation (9 x N)
%
% Output:
%   xdot       State derivatives (50 x N)
%   grf        (optional) 6 x N matrix, ground reactions (Fx, Fy, Mz) on right and left foot
%   stick      (optional) 20 x N matrix, x and y coordinates for 10 stick figure points
%   forces     (optional) 16 x N matrix with muscle forces
%   moments    (optional) 6 x N matrix with joint moments
```

Matlab

- Our model vs. using Opensim
  - built-in model, 2D, hard to change (C code)
  - probably OK for research on prosthetic legs
  - easier to use if you're a Matlab programmer
  - no fancy graphics (but you can do that elsewhere)
  - **f(x,u)** is twice differentiable, suitable for linearization and gradient-based solvers
- Matlab
  - Exploring the API
  - GUI for real-time simulation

How to model a prosthetic leg

- Remove the amputated muscles
  - or simply set their control input (u) to zero
- Use extra actuation inputs to apply joint torques
  - in the HMC lab model: q5 = right knee flexion
  - To add a knee torque T: M = [0 0 0 0 T 0 0 0 0]';
  - in Opensim, use function calls (C++ or Matlab)
- Alter the mass properties
  - in Opensim: edit the XML file (gait2392.osim)
  - in the HMC lab model: edit C code & recompile

```
// multibody model parameters, from Winter book for 75 kg body mass and 1.8 m
param.TrunkMass     = 50.8500;
param.ThighMass     = 7.5000;
param.ShankMass     = 3.4875;
param.FootMass      = 1.0875;

param.TrunkInertia  = 3.1777;
param.ThighInertia  = 0.1522;
param.ShankInertia  = 0.0624;
param.FootInertia   = 0.0184;

param.TrunkCMy      = 0.3155;
param.ThighCMy      = -0.1910;
param.ShankCMy      = -0.1917;
param.FootCMx       = 0.0768;
param.FootCMy       = -0.0351;
param.ThighLen      = 0.4410;
param.ShankLen      = 0.4428;

// contact model parameters
param.ContactHeelX  = -0.06;          // X coordinate of heel contact point
param.ContactToeX   = 0.15;           // X coordinate of toe contact point
```