

Tutorial: Limit cycle analysis

Daniël Karssen (j.g.d.karssen@tudelft.nl) & Martijn Wisse

In this tutorial we are going to investigate the behavior of a dynamic walker. The goal of this tutorial is to learn about limit cycles and the stability of these limit cycles. In this tutorial we will use a simulation of a simple walking robot. For this simulation you need to have a version of Matlab (version 6.5 or higher will certainly work and lower versions will probably work). Figure 1 shows the model that we are going to simulate. The model consists of two identical rigid legs which are connected at the hip with a rotational joint. The model is passive and gains energy by walking down a slope. We describe the state of the model with the absolute angles and angular velocities of the two legs with respect to the normal of the ground.

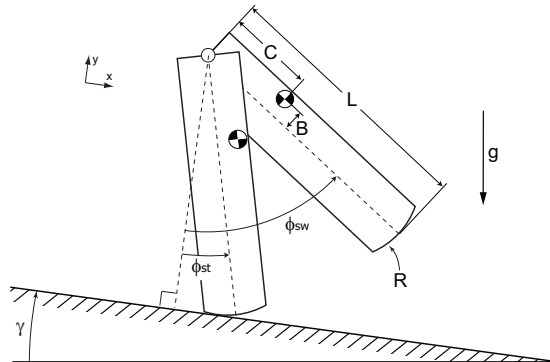


Figure 1: Simple dynamic walking model

A part of the simulation is provided¹. The file **Step.m** is a Matlab function that simulates one step of the model. The simulation integrates the equations of motion until the swing foot hits the ground. After this event is detected, the integration is stopped and an impact calculation is done to determine the state of the model after the impact. Also the legs are swapped so that the swing leg becomes the new stance leg and visa versa. The equations of motion and impact equations are determined with the TMT-method². How this is done for this model can be found in the file **DeriveEOM.m**.

¹The simulation files can be downloaded from: <http://www.dynamicwalking.org>

²For more details about the TMT-method see the third chapter of <http://audiophile.tam.cornell.edu/~als93/wb1413spring2008/MultibodyDynamicsB.pdf>

Exercise 1 [15 min] Create an m-file that simulates one step of the prototype starting from the initial condition: $[\phi_{st}; \phi_{sw}; \dot{\phi}_{st}; \dot{\phi}_{sw}] = [0.2; -0.2; -2.5; -2.0]$ and plots the state as a function of time. The parameters of the prototype are given in table 1. For help on the Step function, type 'help Step' in the command window of Matlab (make sure that you have the correct current directory) or open the file **Step.m**. Note that you can make an animation of the simulation with the file **Animation.m**.

Table 1: Parameter values of the prototype.

Leg length	L	0.132	$[m]$
Foot radius	R	0.06	$[m]$
Leg mass	m	0.07	$[kg]$
Leg inertia	I	0.00018	$[kgm^2]$
Hor. pos. CoM	B	0.00	$[m]$
Ver. pos. CoM	C	0.061	$[m]$
Slope	γ	0.005	$[rad]$
Gravity constant	g	9.81	$[m/s^2]$

Exercise 2 [5 min] Simulate a walk of multiple steps by calling the function **Walk**. This function loops the Step function. The end state of one step is used as initial state of the next step.

The Step function can mathematically be seen as $\mathbf{s}_{n+1} = F(\mathbf{s}_n)$ with $\mathbf{s}$ as the state of the system $[\phi_{st}; \phi_{sw}; \dot{\phi}_{st}; \dot{\phi}_{sw}]$. This function can have a fixed point, meaning that the output of the function is the same as the input, $\mathbf{s}^* = F(\mathbf{s}^*)$. For dynamic walking robots a fixed point is called a limit cycle and is a walking cycle that keeps on repeating. To find a limit cycle we minimize the difference between the input and output state, $\min_{\mathbf{s}} |F(\mathbf{s}) - \mathbf{s}|$.

Exercise 3 [10 min] Find a limit cycle for the prototype using Matlab's minimize function **lsqnonlin**. This function can be called with the following command³:

```
lsqnonlin(@(s) Step(s,0,par)-s,s0,[],[],options)
```

in which **s0** is the initial guess. Before calling the **lsqnonlin** command you have to set the options. For this problem the following options work out well:

```
options = optimset('TolFun',1e-10,'TolX',1e-10,'LargeScale','off')
```

Exercise 4 [5 min] Simulate a walk starting from the found fixed point. Is this a stable cycle?

A limit cycle can be stable or unstable, in which stable means that small perturbations shrink and unstable that small perturbations grow. To see if a system is stable we have to look at what happens if a small perturbation with respect

³Note that you need for this command at least Matlab 7 and the Optimization toolbox. You can use the following command instead: **fminsearch**(@StepError,s0,options,par)

to the fixed point is applied. In the Step function a small perturbation can be seen as

$$\mathbf{s}^* + \Delta \mathbf{s}_{n+1} = F(\mathbf{s}^* + \Delta \mathbf{s}_n). \quad (1)$$

The function $F(\mathbf{s}^* + \Delta \mathbf{s}_n)$ can be linearized around $\mathbf{s}^*$ to get $F(\mathbf{s}^*) + \frac{\partial F}{\partial \mathbf{s}} \Big|_{\mathbf{s}^*} \Delta \mathbf{s}_n$. With the linearization the $\mathbf{s}^*$ -terms are canceled out and we get:

$$\Delta \mathbf{s}_{n+1} = \frac{\partial F}{\partial \mathbf{s}} \Big|_{\mathbf{s}^*} \Delta \mathbf{s}_n. \quad (2)$$

For systems with more than one state variable the partial derivative $\frac{\partial F}{\partial \mathbf{s}}$ is a matrix. This matrix is called the Jacobian J . The eigenvalues of the Jacobian tell what happens to small perturbations. If the magnitude of an eigenvalue is larger than 1 the perturbation will grow and the system is unstable. So for a stable system all the eigenvalues have to be smaller than 1.

Exercise 5 [15 min] Calculate the Jacobian (matrix of partial derivatives) of the limit cycle found in exercise 3. The partial derivatives can numerical be found by perturbing the state variables one after the other. Use a perturbation size of 10^{-4} . Hint: equation 2 can be rewritten as $\frac{\partial F}{\partial \mathbf{s}} \Big|_{\mathbf{s}^*} = \frac{\mathbf{s}_{n+1} - \mathbf{s}^*}{\mathbf{s}_n - \mathbf{s}^*}$.

Exercise 6 [5 min] Use the Matlab command `eig` to calculate the eigenvalues of the Jacobian found in exercise 5. Is the limit cycle stable?

On the prototype we have the option to add masses to the legs. In the simulation we can add masses by changing the mass properties of the legs. This can be done in Matlab as follows:

```
% mass properties additional mass
par.mm = ; % mass of the additional mass [kg]
par.mB = ; % horizontal position of the additional mass [m]
par.mC = ; % vertical position of the additional mass [m]

% update mass properties with additional mass
par.I = par.I + ((par.B-par.mB)^2+(par.C-par.mC)^2)*par.m*par.mm/(par.m+par.mm);
par.B = (par.m*par.B+par.mm*par.mB) / (par.m+par.mm);
par.C = (par.m*par.C+par.mm*par.mC) / (par.m+par.mm);
par.m = par.m+par.mm;
```

Exercise 7 [15 min] Using the code above, add a mass of 0.08kg to each leg. To prevent typing errors you can copy the code from `AddMass.txt`. Choose the position of the mass so that the model has a stable limit cycle.

Exercise 8 [5 min] One of the four eigenvalues is (very close to) zero. Give an explanation for this.