Project 2 + 3: Modeling and optimization of a slithering snake

ME498: Computational modeling and optimization

Mattia Gazzola

Issue Date : March 10, 2019
Teaching Assistant : Tejaswin Parthasarathy, `tp5@illinois.edu`
Submission Date/Time: Wednesday 11:59 PM, 01 May, 2019
Submission Instructions:

- You need to submit both your presentation (as `.pdf`, `.ppt[x]` or `.key`) and code (as `.py` or `.ipynb`) on Compass, in separate archived files (see below)
- Submit your code file(s) packaged as a `.zip` or `.tar.gz` files (please do not use `.7z` or other formats) and name the compressed file as `<net_id>_code_02`
- Submit your presentation file(s) packaged as a `.zip` or `.tar.gz` files (please do not use `.7z` or other formats) and name the compressed file as `<net_id>_slides_02`

1 Guidelines

General advice that pertains to solving the dynamics of an idealized slithering snake:

1. Validate your code (i.e. test that it captures the physics) before doing any optimization on your snake model. Some sensible cases for validation are discussed later on.

2. You will need to couple the physical simulations that you develop in this project with the optimizer—hence develop code keeping this requirement in mind.

3. Use the CMAes class/function that you developed as part of the previous project to run the optimization campaign.

4. We will evaluate your work based on your understanding and findings, and not on the code quality. However since you are learning `Python`, it is always a good idea to code efficiently. Use canned algorithms in `numpy` or `scipy` wherever possible.

5. You are free to consult any material you want (including but not limited to resources in section 4). However you are expected to follow the Student Code on academic integrity—plagiarism will not be tolerated!

2 A soft snake

2.1 Definition

In `Python`, implement a code for computing the mechanics of soft filaments using Cosserat rod theory, mirroring the in-class discussion. Construct a model of a slithering snake using this implementation and furthermore optimize the gait of the snake for achieving maximal forward speed.

BONUS : Visualize any/all of your results using the PovRay raytracing library or its `Python` interface—Vapory.

2.2 Details

2.2.1 Governing equations

Cosserat rod model The following are the equations that you need to implement in your code. Note that these equations are derived **after** appro-

priate simplification as detailed in the next subsection.

$$\frac{\partial \mathbf{r}}{\partial t} = \mathbf{v} \tag{1}$$

$$\frac{\partial \mathbf{d}_j}{\partial t} = (\mathbf{Q}^T \boldsymbol{\omega}_{\mathcal{L}}) \times \mathbf{d}_j, \qquad j = 1, 2, 3 \tag{2}$$

$$dm \cdot \frac{\partial^2 \mathbf{r}}{\partial t^2} = \underbrace{\frac{\partial}{\partial \hat{s}} \left(\frac{\mathbf{Q}^T \hat{\mathbf{S}} \boldsymbol{\sigma}_{\mathcal{L}}}{1} \right) d\hat{s}}_{\text{shear/stretch internal force}} + \underbrace{\mathbf{F}}_{\text{ext. force}} \tag{3}$$

$$\frac{d\hat{\mathbf{J}}}{e} \cdot \frac{\partial \boldsymbol{\omega}_{\mathcal{L}}}{\partial t} = \underbrace{\frac{\partial}{\partial \hat{s}} \left(\frac{\hat{\mathbf{B}} \hat{\boldsymbol{\kappa}}_{\mathcal{L}}}{1^3} \right) d\hat{s} + \frac{\hat{\boldsymbol{\kappa}}_{\mathcal{L}} \times \hat{\mathbf{B}} \hat{\boldsymbol{\kappa}}_{\mathcal{L}}}{1^3} d\hat{s}}_{\text{bend/twist internal couple}} + \underbrace{\left(\mathbf{Qt} \times \hat{\mathbf{S}} \boldsymbol{\sigma}_{\mathcal{L}} \right) d\hat{s}}_{\text{shear/stretch internal couple}} + \underbrace{\mathbf{C}_{\mathcal{L}}}_{\text{ext. couple}}, \tag{4}$$

where $dm = \rho \hat{A} d\hat{s} = \rho A ds$ is the infinitesimal mass element, and $d\hat{\mathbf{J}} = \rho \hat{\mathbf{I}} d\hat{s}$ is the infinitesimal mass second moment of inertia. Note that both initial and boundary conditions are problem dependent.

Numerical discretization Numerically discretize the above equations according to the discussion in class, using nodal masses and elemental frames. Implement frame rotations/ transformations using the Rodrigues rotation formula, using code from the hands-on session on rotations. Utilize the energy-preserving second-order (in position and velocity) position-Verlet scheme, following the details on the hands-on session on timestepping schemes.

2.2.2 Simplifications to the governing equations

- The first simplification performed above involves a crude approximation—we simply remove the Lagrangian transport and unsteady dilatation terms from the angular momentum evolution equationsection 2.2.1.
- The second simplification comes from explicitly constraining $e = 1$. This means you **DO NOT** need quantities in the reference and current configuration—they are always identical. This means that you do not need hatted ($\hat{\cdot}$) and non-hatted matrices/quantities. Implementation-wise, you will define constant matrices and reuse them throughout all the elements. We note that this approximation is only valid in the specific example of slithering snake that we are interested in.

- As we will later see, rolling/lateral friction is neglected.

2.2.3 Validation

Once you are confident that all your operators (`Python` functions) are performing the right task, it is prudent to validate your code against physical cases with known analytical (or numerical/experimental) solutions. One such example is the deformation of a cantilever beam under an applied load (and you are free to choose other cases too, if you so wish).

In this problem, you clamp one end of the beam $\hat{s} = 0$, while applying a load F to the free end of the beam $\hat{s} = \hat{L}$. This is shown in fig. 1 below, taken from the first link in section 4. In this case, analytical solutions dervied from the Timoshenko (and Euler-Bernoulli) theory exists, which relies on the assumption of small deflections, so that the horizontal coordinate x along the direction $\mathbf{k}$ can be approximated by the arc-length $\hat{s}$.

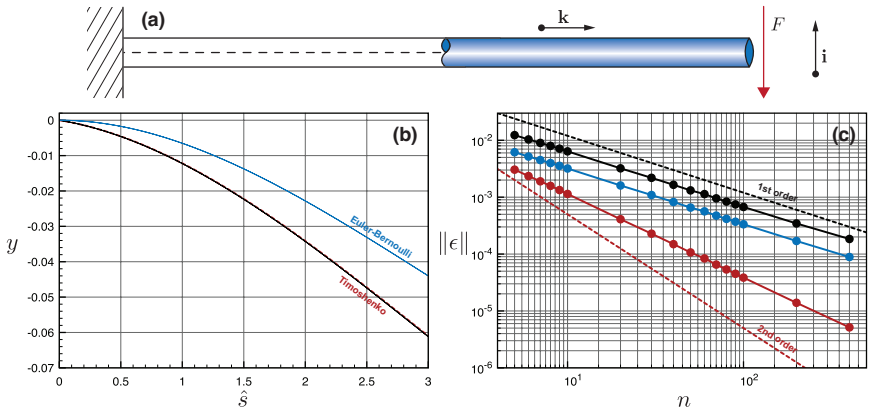

Figure 1: Validation—Deformation of a cantilever beam

Considering the rod has length $\hat{L}$, constant cross sectional area $\hat{A}$, area second moment of inertia about the axis $\mathbf{j} = \mathbf{k} \times \mathbf{i}$ to be $\hat{I}_1$, Young's and shear moduli E and G, the analytical solution is

$$y = -\frac{3F}{4\hat{A}G}\hat{s} - \frac{F\hat{L}}{2E\hat{I}_1}\hat{s}^2 + \frac{F}{6E\hat{I}_1}\hat{s}^3$$

If the shear modulus G approaches infinity or if the ratio $3E\hat{I}_1/(4\hat{L}^2\hat{A}G) \ll 1$, then the Timoshenko model in the static case reduces to the Euler-

Bernoulli approximation, yielding

$$y = -\frac{F\hat{L}}{2E\hat{I}_1}\hat{s}^2 + \frac{F}{6E\hat{I}_1}\hat{s}^3$$

You can then compare your numerical model with these results by carrying out simulations of the cantilever beam shown in fig. 1, with generous number of elements and an appropriate dt. You should recover the results obtained from Timoshenko theory, shown in fig. 1 for these parameters:

Table 1: Parameters for the Timoshenko beam validation

Parameters	Value
Rod density ρ	$5 \times 10^3\,\mathrm{kgm}^{-3}$
Young's modulus E	$1 \times 10^6\,\mathrm{Pa}$
Shear modulus G	$1 \times 10^4\,\mathrm{Pa}$
Downward force F	$15\,\mathrm{N}$
Rod Length L	$3\,\mathrm{m}$
Rod radius r	$0.25\,\mathrm{m}$
Dissipation constant γ	$0.1\,\mathrm{kg\,m}^{-1}\,\mathrm{s}^{-1}$
Simulation time T	$5 \times 10^3\,\mathrm{s}$
Number of discretization points n	100
Time step dt	$3 \times 10^{-4}\,\mathrm{s}$

Notice that if you change one of these parameters such that $3E\hat{I}_1/(4\hat{L}^2\hat{A}G) \ll 1$, you should also recover the results of the Euler-Bernoulli theory (Say by setting $E = G = 1 \times 10^5\,\mathrm{Pa}$).

Initial conditions: Notice that in this case, the initial condition constrains the rod to be straight, with its axis (and hence all elemental frames) pointing in the **i** direction. Setting the spatial location of the equispaced nodes/frames initially then, is pretty straightforward. Additionally at the start, all nodes have translational and angular velocities set to **0**, in the appropriate units.

Boundary conditions: Notice that in this case, the boundary condition constrains the elemental frame (and its angular velocity) at $\hat{s} = 0.0$ to retain its initial configuration. Furthermore at this location, the node is time invariant—hence its location is fixed, and its velocity always **0**.

2.2.4 Towards a slithering snake

Muscular activity To model muscular activity, we express it as torques acting along the body. The magnitude A_m of this torque is a traveling wave propagating head to tail along the filament

$$A_m = \beta_m(\hat{s}) \cdot \sin\left(\frac{2\pi}{T_m}t + \frac{2\pi}{\lambda_m}\hat{s}\right)$$

where t is time, T_m and λ_m are, respectively, the activation period and wavelength. In the equation above, the amplitude of the traveling wave is represented by the cubic B-spline $\beta(\hat{s})$ characterized by N_m control points $(\hat{S}_i, \beta_i)$ with $i = 0, \ldots, N_m - 1$, as illustrated in fig. 2. The first and last control points are fixed so that $(\hat{s}_0, \beta_0) = (0, 0)$ and $(\hat{s}_{N_m-1}, \beta_{N_m-1}) = (\hat{L}, 0)$, therefore assuming the ends of the deforming body to be free.

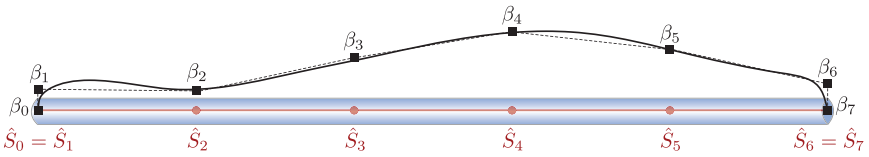

Figure 2: B-spline parametrization for modeling muscular activity using torques. We exhibit the case with $N_m = 8$ here.

We then prescribe this muscular activity as an internal torque activation of the form

$$\boldsymbol{\tau}_{\mathcal{L}}^{m} = \mathbf{Q}(A_m \mathbf{d}_1)$$

assuming $\mathbf{d}_2$ and $\mathbf{d}_3$ to be coplanar to the ground. This contribution is directly added to the internal torque $\boldsymbol{\tau}_{\mathcal{L}}$ resulting from solving the Cosserat equations.

The cubic B-spline function with the appropriate boundary conditions has been implemented for you and is available as a function from the script-file `b_spline.py`. A typical use-case is shown in the code listing 1 below, which produces the spline shown in fig. 3.

Contact with the wall: The wall (or ground) contact is modeled as an external response force experienced by the rod $\mathbf{F}_{\perp}^{w}$ that balances the sum of all forces $\mathbf{F}_{\perp}$ that push the rod against the wall, and is complemented by other two components which help prevent possible interpenetration due to numerical errors. The interpenetration distance ϵ triggers a normal elastic response proportional to the stiffness of the wall k_w, while a dissipative term

```python
import numpy as np
from matplotlib import pyplot as plt
from bspline import snake_bspline

# Length of centerline
l_centre = 1.0
# Non-zero coefficients of spline, set by you
t_coeff = np.array([0.1, 0.3, 0.15, 0.22, 0.25, 0.1])
# See function documentation for more details
my_spline, ctr_pts, ctr_coeffs = snake_bspline(t_coeff, keep_pts=True)

s = np.linspace(0.0, l_centre, 200)

# Figure beautification
fig, ax = plt.subplots(figsize=(8,2))
ax.set_aspect('equal')
ax.set_xlim(0.0, 1.0)
ax.set_ylim(0.0, 0.4)
ax.set_xlabel(r'$\hat{s}$')
ax.set_ylabel(r'$\beta_m(\hat{s})$')

# Plot the spline along as function of centerline
ax.plot(s, my_spline(s))

# Plot the control points of the spline too
ax.plot(ctr_pts, ctr_coeffs, 'kx')

# Export
FILE_NAME = 'images/snake_spline.pdf'
fig.savefig(FILE_NAME, bbox_inches='tight')
FILE_NAME
```

Listing 1: B-spline code snippet

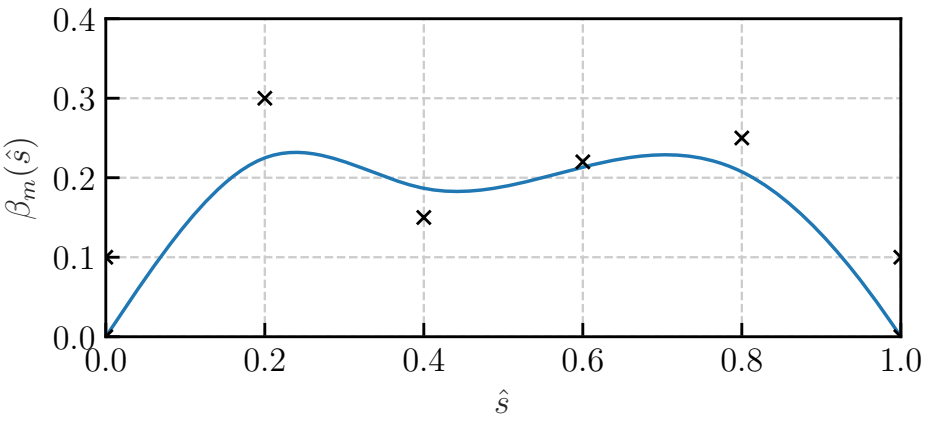

Figure 3: The spline generated by the script `b_spline.py`

related to the normal velocity component of the filament with respect to the wall accounts for a damping force proportional to γ_w, so that the overall wall response is

$$\mathbf{F}_{\perp}^{w} = H(\epsilon) \cdot (-\mathbf{F}_{\perp} + k_w \epsilon - \gamma_w \mathbf{v} \cdot \mathbf{u}_{\perp}^{w}) \mathbf{u}_{\perp}^{w}$$

where $H(\epsilon)$ denotes the Heaviside function and ensures that a wall force is produced only in case of contact ($\epsilon \geq 0$). Here $\mathbf{u}_{\perp}^{w}$ is the boundary outward normal (evaluated at the contact point, that is the contact location for which the normal passes through the center of mass of the element), and k_w and γ_w are, respectively, the wall stiffness and dissipation coefficients.

Once wall contact is modeled, you can run some test cases to see whether it works. As the response is linear, when $\epsilon > 0$, consider running the following three cases while recording the force on the cylinder:

- A rod with nodal mass dm resting horizontally on the ground (which is also at rest), when uniform gravity $g = 9.81\,\mathrm{m/s^2}$ acts in the vertical direction (i.e, in the wall normal direction). In this case, the wall force should equal the force due to gravity for static equilibrium, i.e. $\mathbf{F}_{\perp}^{w} =$
- Now turn gravity off in the scenario above, but initialize the rod such that it has some interpenetration ϵ with the ground (once again, in the wall normal direction). If the wall stiffness is k_w, then the instantaneous wall response should record in your solver as $k_w \epsilon$.
- To check the damping force, we envision two cases shown below. In both cases gravity is turned off:
  - The rod lies on the ground similar to the first case, but it now has a uniform velocity v in the ground coplanar direction (say horizontal). In this case, the wall response should record zero (or values close to zero).
  - If however, the uniform velocity v now acts in the wall normal direction and tries to penetrate the rod into the ground, then the instantaneous wall normal response should read $\gamma_w v$ (or values close to the same, accounting for interpenetration ϵ).

Anisotropic friction: The modeling of friction should closely follow the in-class discussions. Once the isotropic friction law is setup using the Amonton–Coloumb law, anisotropy can be included by changing the friction coefficients based on the direction of locomotion. For this project, you can **neglect** lateral/rolling friction.

2.2.5 The slithering snake

With all the components in place, we can assemble them together to model a snake. For this case, the muscular activity is modeled as an internal torque, calculated as a parametrized B-spline, as mentioned before. We first discuss initial and boundary conditions:

Initial conditions The rod representing the snake is initialized coplanar to the ground, with equispaced nodes along the forward direction. At the start, $\mathbf{d}_1$ is assumed to point in the wall-normal direction and so $\mathbf{d}_2, \mathbf{d}_3$ point in the coplanar direction. We also remind you that $\mathbf{d}_3$ is set to points along the body centerline coordinate, at the start. All nodal translational velocities and elemental angular velocities are initialized as zero.

Boundary conditions With the torque profile imposed by the B-spline, we need not specify boundary conditions on the snake (aka Free boundary conditions).

Muscle activity We consider a **six** parameter B-spline, with $\beta_{i=0,5} = 0$ to model the muscle activity.

Additional validation If you are not confident with your snake model, you can refer to the first link in section 4 for more validation cases or alternatively ask the TA.

2.2.6 Gait optimization for maximal forward velocity

We are now (almost) ready to tackle the optimization problem of finding the maximal forward velocity for a model snake. The code setup, initial and boundary conditions follow from the previous section, including the six coefficient spline parameterization.

The rod parameters for this case are given in table 2.

Coupling with CMAes : With these parameters, you can now run an optimization campaign using CMAes, to find an optimal gait that maximizes the forward velocity $v_{\text{max}}^{\text{fwd}}$ over one activation cycle. That is, you are required to find the spline coefficients and wavelength:

$$\beta_i \quad i = 1, 2, 3, 4 \quad \text{and} \quad \lambda_m$$

with $\beta_{i=0} = \beta_{i=5} = 0\,\text{N}\,\text{m}$ identically. Think about the fitness function for this problem, and any bounds that you would like to place on the parameters

Table 2: Parameters for the snake to be optimized for maximal forward velocity

Parameters	Value
Rod density ρ	$5 \times 10^3\,\mathrm{kgm}^{-3}$
Young's modulus E	$1 \times 10^7\,\mathrm{Pa}$
Shear modulus G	$2E/3\,\mathrm{Pa}$
Shear/Stretch matrix $\mathbf{S}$	$\mathrm{diag}(4GA/3, 4GA/3, 2GA/3)\mathrm{N/m}^2$
Bend/Twist matrix $\mathbf{B}$	$\mathrm{diag}(EI_1, EI_2, GI_3)\mathrm{N/m}^2$
Rod length L	$1\,\mathrm{m}$
Rod radius r	$0.025\,\mathrm{m}$
Muscular activation period T_m	$1\,\mathrm{s}$
Dissipation constant γ	$5\,\mathrm{kg\,m}^{-1}\,\mathrm{s}^{-1}$
Acceleration due to gravity normal to ground g	$9.81\,\mathrm{m/s}^2$
Forward kinetic friction coefficientμ_k^f	1.019368
Backward kinetic friction coefficient μ_k^b	$1.5 \cdot \mu_k^f$
Forward static friction coefficientμ_s^f	$2 \cdot \mu_k^f$
Backward static friction coefficient μ_s^b	$1.5 \cdot \mu_s^f$
Friction threshold velocity v_ϵ	$1 \times 10^{-8}\,\mathrm{m\,s}^{-1}$
Ground stiffness k_w	$1\,\mathrm{kg/s}^2$
Ground viscous dissipation	$1 \times 10^{-6}\,\mathrm{kg\,s}^{-1}$
Number of discretization points n	50
Time step dt	$2.5 \times 10^{-5}\,\mathrm{s}$

to be optimized (for example, we ran it with $|\beta|^{\max}_{i=0,\ldots,5} = 50\,\mathrm{N\,m}$). The optimized parameters in our case (including lateral friction) were

$$\beta_{i=0,\ldots,5} = \{0, 17.4, 48.5, 5.4, 14.7, 0\} \quad \text{and} \quad \lambda_m = 0.97\,\mathrm{m}$$

which gives an average forward velocity of $v^{\mathrm{fwd}}_{\max} \simeq 0.6\,\mathrm{m\,s^{-1}}$, which compares well to real-life snakes*. This is shown in fig. 4, and should (ideally) not be far from the final velocity that your implementation gives as well (considering that you are making a lot of assumptions). Once again, to encourage comparison with our results, the forward and lateral velocities of the optimal snake is attached in the file `optimized_snake.dat`, and can be read into your `Python` environment, using `numpy`'s `loadtxt` function, as shown in listing 2.

```python
import numpy as np

# my_data is a (x, 3) time series data
# The first columnd my_data[:,0] contains the time
# The second columnd my_data[:,1] contains the forward snake velocity
# The third columnd my_data[:,2] contains the lateral snake velocity
my_data = np.loadtxt('optimized_snake.dat')
```

Listing 2: Importing the snake dataset

2.3 Expected submission

We expect you to submit your code and presentation. In your presentation, please try and include all your validation cases, information on your optimization campaign (CMAes fitness function, bounds on parameters, time interval for optimization etc), performance of CMA on this problem and finally the results obtained. Other information such as timing data from your code (i.e. time for a single function evaluation) etc. can also be included.

3 A soft X

This section is compulsary for students taking the course for four credit hours and is optional for those taking the course for three credit hours.

3.1 Solve for X

Think of an physical problem/application in which the modeling capabilities that you learnt in this course can come in handy, and design, using Cosserat

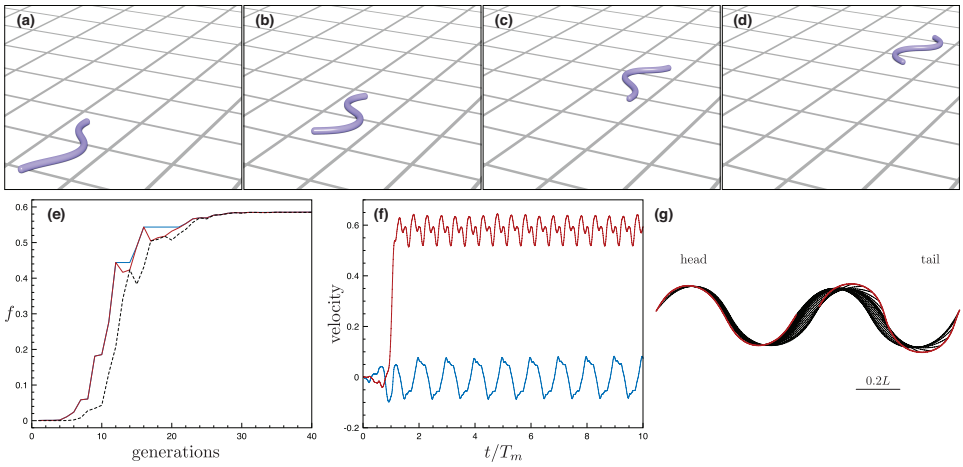

Figure 4: Optimal lateral undulation gait. (a, b, c, d) Instances at different times of a snake characterized by the identified optimal gait. (e) Evolution of the fitness function $f = v_{\max}^{\text{fwd}}$ as function of the number of generations produced by CMA-ES. Solid blue, solid red and dashed black lines represent, respectively, the evolution of f corresponding to the best solution, the best solution within the current generation, and the mean generation value. (f) Scaled forward (red) and lateral (blue) center of mass velocities versus normalized time. (g) Gait envelope over one oscillation period T_m. Red lines represent head and tail displacement in time.

rod(s), a setup that can be used to study/solve the problem you have in mind. This problem/application can be inspired from your research (or) even something you are really curious about (a few examples will be given in the class, but we will brainstorm with you about potential project ideas and their feasibility).

3.2 Optimize for X

Once you have a preliminary design that partially/fully solves your problem, you need to see whether you can do *better*. You are then expected to setup an inverse design problem, define what's *better* in your case and use the optimization techniques that you have learnt thus far, to *evolve* new designs.

3.3 Understand X

After you have arrived at a *good* design, try and understand what makes it *good*. While this may not be straightforward in all cases, tracking the designs that CMAes evolves can give you some intution as to why your design may be optimal.

4 The following resouces may prove useful:

- Paper describing the governing equations, numerical algorithm and optimization of a slithering snake, found in this link.
- The CMA-ES tutorial @ Arxiv, found here
- More information on timestepping schemes found at this link
- This link on a short but gentle introduction to symplectic time integration schemes accompanied by this link that compares many other schemes to the same.

5 Compass Instructions

Project 2 In this project, you will implement a computational soft mechanics code (in Python) and construct a near-realistic model of a slithering snake. You will also hook it up to a stochastic optimization algorithm (CMAes) to find a gait that maximizes the forward speed, under some given conditions.

Project 3 Students taking the course for four credit hours additionally need to think of a model problem/application which they can solve using the implemented code. You also need to develop an inverse design study to find a better (optimal?) solution that tackles the same problem, and try and determine why the solution is optimal.

More information enclosed in the attached files.