Practice Exercises for Installation and Python Tips

Sean Matthew Nolan

February 23, 2021

1 Install Beluga

Problem

On your local machine, install beluga per the instructions given in the lecture. Use a virtual environment. Navigate to the folder that contains beluga, and to verify installation, run:

```
> python .\examples\Classic\MoonLander\MoonLander.py
```

2 Morse Code Translator

Problem

Write a program in Python that automatically converts text to Morse code and vice versa.

3 Catapult Launch Problem

Problem

Using Python, numerically maximize the range of a projectile launched by a catapult by finding the optimal release angle γ_0. After the projectile is released, it travels with dynamics:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{v} \\ \dot{\gamma} \end{bmatrix} = \begin{bmatrix} v\cos\gamma \\ v\sin\gamma \\ -\frac{D(v)}{m} - g\sin\gamma \\ -\frac{g}{v}\cos\gamma \end{bmatrix} \tag{3.1}$$

Using fixed initial velocity $v_0 = 100\,\mathrm{ft/s}$, mass $m = 10\,\mathrm{slugs}$, gravity $g = 32.17\,\mathrm{ft/s}$, drag equation

$$D(v) = \tfrac{1}{2}\rho v^2 C_D A\,, \tag{3.2}$$

density $\rho = 0.00235\,\mathrm{slugs/ft^3}$, drag coefficient $C_D = 0.5$, and reference area $A = 15\,\mathrm{ft}^2$

Assume that projectile lands at the same altitude from which it was launched.

Hints

The cost function should numerically propagate the trajectory of the projectile to obtain the range. Use SciPy's `solve_ivp()` with a terminal event (see: `https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html`)

The optimization problem can be solved with SciPy's `minimize` solver (see: `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html`)

4 Tarjan's Algorithm

Problem

Tarjan's strongly connected components algorithm finds groups of strongly connected components in a directed graph. Strongly connected refer to groups in which every node is reachable from every node.

For algorithm details, refer to `https://en.wikipedia.org/wiki/Tarjan%27s_strongly_connected_components_algorithm`

Implement this algorithm using a custom "node" class in Python. It is suggested that you make a function to construct a graph from an adjancy matrix. Test the implementation with the adjacency matrix (where a non-zero element A_{ij} indicates a edge pointing from i to j):

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}$$

References

[1] A. E. Bryson and Y.-C. Ho, *Applied optimal control: optimization, estimation, and control.* Washington : New York: Hemisphere Pub. Corp. ; distributed by Halsted Press, rev. printing ed., 1975.