```
################################################################################
# Radial Action example code for three-bladed turbine # Tested on Python 3.8.7 # with the following library versions: # matplotlib==3.4.3 # numpy==1.21.2 # scipy==1.7.1 # To run, pass the script name to python like this: # python radial_action_example.py
################################################################################
import matplotlib import matplotlib.pyplot as plt from matplotlib import cm import numpy as np import scipy.integrate import scipy.optimize
################################################################################
# Constants
################################################################################
deg_to_rad = 2.0 * np.pi / 360.0 rad_to_deg = 360.0 / (2 * np.pi) radsec_to_revmin = 60 / (2 * np.pi) rho = 1.225
################################################################################
# Integrals
################################################################################
def tw_integrand(r, v, c, theta): return (rho * c * np.sin(theta) * np.sin(2*theta) * np.power(v, 2) * r) def tb_integrand(r, omega, c, theta): return (np.power(rho, 2/3) * c * np.cos(theta) * np.power(r, 2) * omega)
################################################################################
# Torque functions
################################################################################
def torque_windward(v, c_fn, theta_fn, R0, RL, theta0): integrand = lambda r: tw_integrand(r, v, c_fn(r), theta_fn(theta0, r)) res, _ = scipy.integrate.quad(integrand, R0, RL) return res def torque_backward(omega, c_fn, theta_fn, R0, RL, theta0): integrand = lambda r: tb_integrand(r, omega, c_fn(r), theta_fn(theta0, r)) res, _ = scipy.integrate.quad(integrand, R0, RL) return res
################################################################################
# Power and Optimisation
################################################################################
def power(v, omega, c_fn, theta_fn, R0, RL, theta0): tw = torque_windward(v, c_fn, theta_fn, R0, RL, theta0) tb = torque_backward(omega, c_fn, theta_fn, R0, RL, theta0) return (tw - tb) * omega def maximise_power(v, omega, c_fn, theta_fn, R0, RL): ''' Maximises power output over theta0. Returns the maximum power output of the blade and the theta0 that produces it ''' # scipy implements only minimisation # the max of a function's values occurs at the min of the negative of the function's values minimisation_objective = lambda theta0: -power(v, omega, c_fn, theta_fn, R0, RL, theta0) res = scipy.optimize.minimize_scalar(minimisation_objective) max_power = -res.fun # negate the minimum to recover the maximum value max_theta = res.x return max_power, max_theta
################################################################################
# Rectangular blade definition, example turbine configuration
################################################################################
# blade r0 = 0.00 rL = 38.75 c_fn = lambda r: 1.0 theta_fn = lambda theta0, r: theta0 # turbine L = rL λ = 8.0 v = 15.0 Ω = v*λ/L print(f'Ω = {Ω} rad/s') print(f'Ω = {Ω*radsec_to_revmin} rev/min')
################################################################################
# create vectorised forms
################################################################################
power_vec = np.vectorize(power) torque_windward_vec = np.vectorize(torque_windward) torque_backward_vec = np.vectorize(torque_backward)
################################################################################
# Example line plot: power, windward and backward torque
################################################################################
# produce an array of input values for the plots # vary theta0 over [5, 90] degrees, 100 evenly spaced points N = 100 θ = np.linspace(5, 90, N) * deg_to_rad # using an array in the theta0 argument, so that the function produces an array result p = power_vec(v, Ω, c_fn, theta_fn, r0, rL, theta0=θ) tw = torque_windward_vec(v, c_fn, theta_fn, r0, rL, theta0=θ) tb = torque_backward_vec(Ω, c_fn, theta_fn, r0, rL, theta0=θ) # max power, theta0 for the given configuration max_p, θ_max_p = maximise_power(v, Ω, c_fn, theta_fn, r0, rL) print(f'θ_max_p: {θ_max_p*rad_to_deg} deg') print(f'max_p: {3*max_p*1e-6} MW') # plot the power and torques fig, ax = plt.subplots() ax.plot(θ*rad_to_deg, 3*p*1e-6, linewidth=1.0) ax.plot(θ*rad_to_deg, 3*tw*1e-6,
```

```
linewidth=1.0) ax.plot(θ*rad_to_deg, 3*tb*1e-6, linewidth=1.0) ax.axvline(θ_max_p*rad_to_deg, linewidth=0.5, color='k') ax.axhline(3*max_p*1e-6, linewidth=0.5, color='k') ax.set_xlabel(r'Angle of incidence, $\theta$ (deg)') ax.set_ylabel(r'Power, $p$ (MW)') ax.grid() plt.show()
################################################################################
# Example surface plot: power as a function of velocity and angle of incidence
################################################################################
N = 100 λ = 8 idx_v = np.linspace(5, 20, N) idx_θ = np.linspace(5, 90, N)*deg_to_rad grid_v, grid_θ = np.meshgrid(idx_v, idx_θ) grid_Ω = grid_v*λ/L lambda8 = power_vec(grid_v, grid_Ω, c_fn, theta_fn, r0, rL, grid_θ) fig, ax = plt.subplots(subplot_kw={"projection": "3d"}) surf = ax.plot_surface(grid_v, grid_θ*rad_to_deg, 3*lambda8*1e-6, cmap=cm.coolwarm, alpha=0.8, linewidth=0, antialiased=True) ax.view_init(15, 220) ax.set_xlabel(r'Wind speed, $v$ (m/s)') ax.set_ylabel(r'Angle of incidence, $\theta$ (deg)') ax.set_zlabel(r'Power, $p$ (MW)') ax.grid() plt.show()
```