1. For the Motor Design problem introduced in class, compute the probably of meeting a requirement that weight $<$ 22 kgs using Monte Carlo. Use the following uncertain parameters:

D N(7.5,0.5)
L N(9.5,0.5)
ρ_{cu} $N(8.94e3, 100)$
ρ_{fe} $N(7.98e3, 100)$

Using 100,000 samples I got a probability of 0.99016 (see attached code).

2. Repeat problem 1 using Monte Carlo but use the following distributions for uncertainty

D Uniform(6.5, 8.5)
L Uniform (8.5, 10.5)
ρ_{cu} $Uniform(8840, 9040)$
ρ_{fe} $Uniform(7880, 8080)$

Using 100,000 samples I got a probability of 0.99023 (see attached code).

3. Repeat problem 1 but use the Mean-value First-order second moment (MVFOSM) method. Again the uncertainties are (the variables are independent, i.e. 0 correlation):

D N(7.5,0.5)
L N(9.5,0.5)
ρ_{cu} $N(8.94e3, 100)$
ρ_{fe} $N(7.98e3, 100)$

Using 100,000 samples I got a probability of 0.6065 (see attached code).

4. Repeat problem 3 but use the following correlation matrix:

$$\begin{bmatrix} 1 & .2 & .3 & .7 \\ .2 & 1 & .5 & .6 \\ .3 & .5 & 1 & .2 \\ .7 & .6 & .2 & 1 \end{bmatrix}$$

How does the correlation change the solution?

Using 100,000 samples I got a probability of 0.6053. The correlation between these variables created a 0.0012 decrease in probability of meeting the design constraint.

Homework 2

April 27, 2020

Motor Design Problem / Heather Miller / Started: 4/22/20

[3]:
```python
import numpy as np
from scipy.stats import norm
```

Motor Design problem : weight of motor must be <22kgs

[23]:
```python
# answers for each part will be stored here
success_probabilities = []

# number of samples to run
samples = 100000

#limit in kgs
limit = 22
```

Given Constraints

[24]:
```python
# Motor Design Variables
Lwa = 14.12  # armature wire length (m)
Lwf = 309.45  # field wire length (m)
ds = 0.00612  # slot depth (m)

# Coupling Variables b, shared with control problem
n = 122  # rotational speed (rev/s)
v = 40  # design voltage (V)
pmin = 3.94  # minimum required power (kW)
ymin = 5.12e-3  # minimum required torque (kNm)

# Parameter Vector a (constants)
fi = 0.7  # pole arc to pole pitch ratio
p = 2  # number of poles
s = 27  # number of slots (teeth on rotor)
rho = 1.8e-8  # resistivity (ohm-m) of copper at 25C

# Derived parameters and constants
mu0 = 4 * np.pi * 1e-7  # magnetic constant
ap = p  # parallel circuit paths (equals poles)
eff = 0.85  # efficiency
```

```
bfc = 40e-3  # pole depth (m)
fcf = 0.55  # field coil spacing factor
Awa = 2.0074e-006  # cross sectional area of armature winding (m^2)
Awf = 0.2749e-6  # cross sectional area of field coil winding (m^2)
```

Functions used in this code

[25]:
```
def calculate_weight(diameter, length, rho_cu, rho_fe):
    # calculate the weight of the motor
    weight = rho_cu * (Awa*Lwa + Awf*Lwf) + rho_fe * length * np.pi *␣
 ↪pow(diameter + ds, 2)
    return weight

def prob_success_mc(weights, limit):
    # determine the probability of success that the weight of engine will be␣
 ↪less than a limit
    # add 1 to limit_sum every time weight is under limit weight
    x = len(weights)
    limit_sum = 0
    for i in weights:
        if i < limit:
            limit_sum += 1
    return limit_sum/x


def first_derivative(variable_dict, variable_of_interest):
    # calculate the first derivatives of each of the variables
    h = 0.1 * variable_dict[variable_of_interest][1]
    # inputs [diameter, length, rho_cu, rho_fe]
    inputs = []
    # this loop will put two values into input for each value, if the variable␣
 ↪selected matches the key it will
    # modify those values with h otherwise both inputs will be the same
    for key in variable_dict:
        if key == variable_of_interest:
            inputs.append([variable_dict[key][0] + h, variable_dict[key][0] - h])
        else:
            inputs.append([variable_dict[key][0], variable_dict[key][0]])
    # calculate the first derivative with the values in the input
    first_d = (calculate_weight(inputs[0][0], inputs[1][0], inputs[2][0],␣
 ↪inputs[3][0]) -
                calculate_weight(inputs[0][1], inputs[1][1], inputs[2][1],␣
 ↪inputs[3][1]))/2*h
    return first_d

def calculate_sigma(variable_sigmas, variable_derivatives, correlation_matrix):
    # calculate the sigma of the function
```

```
    sigma_squared = 0
    for i in range(len(variable_derivatives)):
        for j in range(len(variable_derivatives)):
            sigma_squared += variable_derivatives[i] * variable_derivatives[j] *␣
↪correlation_matrix[i][j] \
                                  * variable_sigmas[i] * variable_sigmas[j]
    return np.sqrt(sigma_squared)
```

1 Using Monte Carlo and Normal Distribution

- D~N(7.5, 0.5) %m rotor diameter
- L~N(9.5, 0.5) %m rotor axial length
- dcu~N(8.94e3, 100) %copper density density at 25C (kg/m^3)
- dfe~N(7.98e3, 100) %iron density density at 25C (kg/m^3)

[26]:
```
d_n = np.random.normal(0.075, 0.005, samples)  # rotor diameter (cm)
l_n = np.random.normal(0.095, 0.005, samples)  # rotor axial length (cm)
rho_cu_n = np.random.normal(8.94e3, 100, samples)  # copper density density at␣
↪25C (kg/m^3)
rho_fe_n = np.random.normal(7.98e3, 100, samples)  # iron density density at 25C␣
↪(kg/m^3)
weight_p1 = [calculate_weight(d_n[i], l_n[i], rho_cu_n[i], rho_fe_n[i]) for i in␣
↪range(samples)]
success_probabilities.append(prob_success_mc(weight_p1, limit))
```

2 Using Monte Carlo and Uniform Distribution

- D~Uniform(6.5, 8.5) %m rotor diameter
- L~Uniform (8.5, 10.5) %m rotor axial length
- dcu~Uniform (8840, 9040) %copper density density at 25C (kg/m^3)
- dfe~Uniform (7880, 8080) %iron density density at 25C (kg/m^3)

[27]:
```
d_u = np.random.uniform(0.065, 0.085, samples)  # rotor diameter (cm)
l_u = np.random.uniform(0.085, 0.105, samples)  # rotor axial length (cm)
rho_cu_u = np.random.uniform(8840, 9040, samples)  # copper density density at␣
↪25C (kg/m^3)
rho_fe_u = np.random.uniform(7880, 8080, samples)  # iron density density at 25C␣
↪(kg/m^3)
weight_p2 = [calculate_weight(d_u[i], l_u[i], rho_cu_u[i], rho_fe_u[i]) for i in␣
↪range(samples)]
success_probabilities.append(prob_success_mc(weight_p2, limit))
```

3 Using MVFOSM method

[28]:
```
# no correlation
correlation_matrix_3 = [[1, 0, 0 , 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]]

variables = {"diameter": (0.075, 0.005),
             "length": (0.095, 0.005),
             "rho_cu": (8.94e3, 100),
             "rho_fe": (7.98e3, 100)}

# determine mean of function using variable means
function_mu = calculate_weight(variables["diameter"][0],
                               variables["length"][0], variables["rho_cu"][0],␣
 ↪variables["rho_fe"][0])

variable_sigmas = [variables["diameter"][1], variables["length"][1],
                   variables["rho_cu"][1], variables["rho_fe"][1]]

diameter_output = first_derivative(variables, "diameter")
length_output = first_derivative(variables, "length")
cu_output = first_derivative(variables,"rho_cu")
fe_output = first_derivative(variables, "rho_fe")

variable_derivatives = [diameter_output, length_output, cu_output, fe_output]

sigma = calculate_sigma(variable_sigmas, variable_derivatives,␣
 ↪correlation_matrix_3)

success_probabilities.append(norm.cdf(22, function_mu, sigma))
```

4 Variables with Correlation.

How does the correlation change the solution?

[29]:
```
correlation_matrix_4 = [[1, .2, .3, .7], [.2, 1, .5, .6], [.3, .5, 1, .2], [.7, .
 ↪6, .2, 1]]

#same varibles as previous part
sigma = calculate_sigma(variable_sigmas, variable_derivatives,␣
 ↪correlation_matrix_4)

success_probabilities.append(norm.cdf(22, function_mu, sigma))
```

Answers

[30]:

```
for i, answer in enumerate(success_probabilities):
    print("The probability of a motor being less than 22kg with the parameters␣
 ↪in #", i+1, " is", answer)

print('The correlation matrix in #4 created a',␣
 ↪(success_probabilities[2]-success_probabilities[3]),
      " decrease in probability of meeting design parameters.")
```

```
The probability of a motor being less than 22kg with the parameters in # 1  is
0.98862
The probability of a motor being less than 22kg with the parameters in # 2  is
0.99015
The probability of a motor being less than 22kg with the parameters in # 3  is
0.6064653429109432
The probability of a motor being less than 22kg with the parameters in # 4  is
0.6052891090555454
The correlation matrix in #4 created a 0.0011762338553977791  decrease in
probability of meeting design parameters.
```

[ ]:

```
```