Final Project

Henry Freed

June 26, 2023

Problem 1.

Solution. The code used to implement *lmbvpoptim* is below:

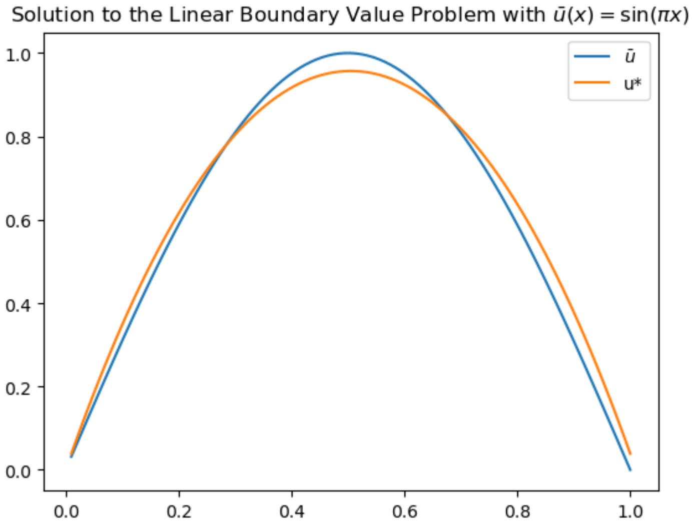
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import fsolve


def cost(u, u_bar):
    return 0.5 * np.linalg.norm(u - u_bar)**2

def A(n):
    h = 1 / (n + 1)
    A = np.eye(n) * 2  # create an identity matrix and multiply it by 2
    # set the off-diagonal elements to -1
    A = A + np.diag([-1] * (n - 1), k=1) + np.diag([-1] * (n - 1), k=-1)
    return (1 / h**2) * A

def G(u):
    return u**3

def jacG(u,n):
    return np.eye(n)*(3 * u**2) #jacG is a diagonal matrix

def lmbvpoptim(u_bar, n=99):
    # initial guess for fsolve
    x0 = np.zeros((2 * n) + 1)  # I think parentheses make it more readable
```

```python
    def func(x):  # x is our initial guess
        u, lambdas, alpha = x[:n], x[n:2*n], x[-1]
        # add partial derivatives to the vector
        # dLdu
        dLdu = u - u_bar - (A(n) + jacG(u,n)) @ lambdas
        # dLdlam
        dLdlam = A(n) @ u + G(u) - alpha * np.ones(n)
        # dLdalpha
        dLdalpha = np.ones(n).T @ lambdas
        return np.concatenate([dLdu, dLdlam, [dLdalpha]])

    # shove the vector into fsolve
    solution = fsolve(func, x0)

    alpha_star = solution[-1]
    u_star = solution[:n]
    lambda_star = solution[n:2 * n]
    f_star = cost(u_star, u_bar)
    return alpha_star, u_star, lambda_star, f_star

#define the inputs
n = 99
x = np.linspace(0, 1, n + 1)[1:] #this defines our increment
a = np.sin(np.pi * x)
b = np.sin(2 * np.pi * x) ** 2

alphaa, u_stara, lambda_stara, f_stara = lmbvpoptim(a)
alphab, u_starb, lambda_starb, f_starb = lmbvpoptim(b)
```

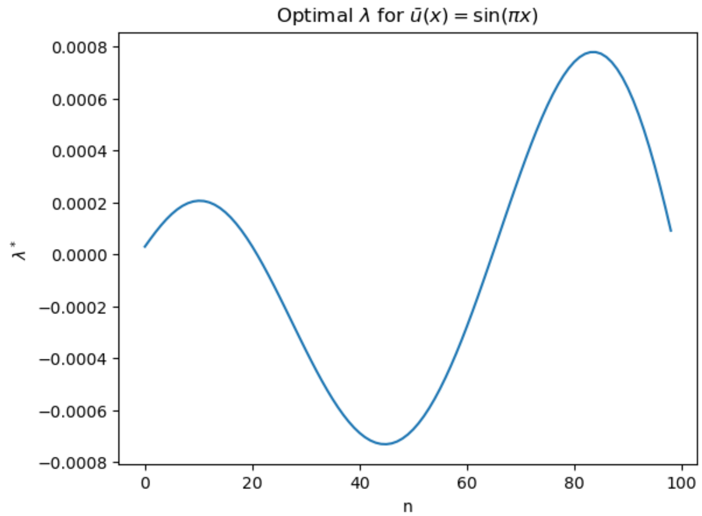
Problem 2.

Solution. For the $\bar{u}(x) = \sin(\pi x), \alpha \approx 8.2435$, and the function evaluated function is ≈ 0.07263. For the $\bar{u}(x) = \sin^2(2\pi x)$, $\alpha \approx 5.5798, f \approx 6.5250$. So, our optimization here works well for the first problem, but, as we can see with the graphs, not well at all for the second $\bar{u}$. The Lagrange multipliers ($\boldsymbol{\lambda}^*$) measure sensitivity of u^* in relation to the constraints. This means that the $\boldsymbol{\lambda}^*s$ measure sensitivity of the system to changes in α. When lambda is higher, the system is more sensitive to changes in alpha, and it is less sensitive when lambda is lower. [1]. The graphs are below:

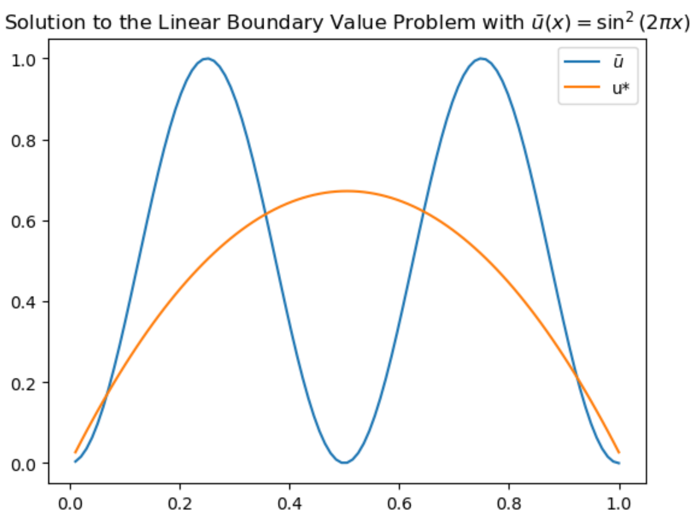
[1] I referred to the notes "SensitivityLagrMult.pdf" for clarification on this



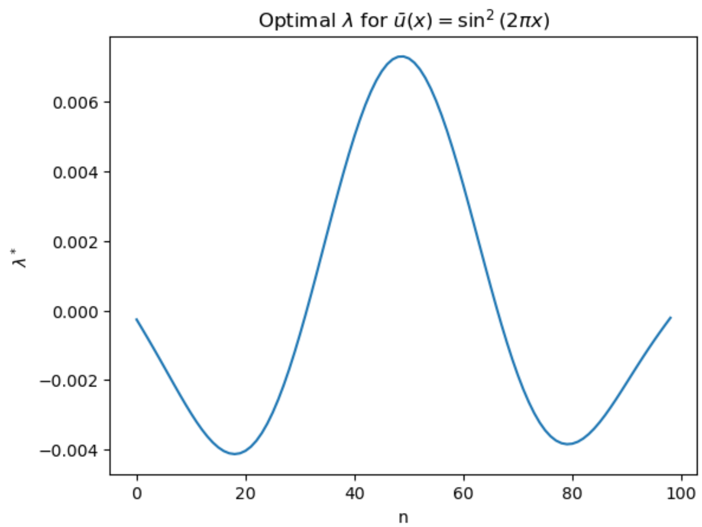
(a) Comparison of $\bar{\boldsymbol{u}}, \boldsymbol{u}^*$ for $\bar{\boldsymbol{u}}(x) = \sin(\pi x)$



(b) $\boldsymbol{\lambda}^*$ for $\bar{\boldsymbol{u}}(x) = \sin(\pi x)$



(c) Comparison of $\bar{\boldsymbol{u}}, \boldsymbol{u}^*$ for $\bar{\boldsymbol{u}}(x) = \sin^2(2\pi x)$



(d) $\boldsymbol{\lambda}^*$ for $\bar{\boldsymbol{u}}(x) = \sin^2(2\pi x)$

Figure 1: Graphs of $\bar{\boldsymbol{u}}, \boldsymbol{u}^*, \boldsymbol{\lambda}^*$