**MTH 654-2024 Homework C2.** **Noah Prentice**
Date: January 3, 2025
Prof. M. Peszynska

---

# 1 Introduction.

In this project, we consider optimal control problems under PDE constraints. In particular, we focus on boundary and distributed control problems constrained by the Poisson equation. The work was guided by [1] and [2].

# 2 An uncoupled boundary control problem.

The first part of the project involved solving the following problem:

**Problem 2.1** – *1D uncoupled boundary control problem*

Suppose we have a 1-dimensional rod represented by the interval $[0, 1]$. We will control the heat on the left side of the rod by $u \in \mathbb{R}$ and fix the rod's temperature at the right endpoint to be 0. We would like the steady-state temperature $v$ of each point $x$ of the rod to be as close to 1 as possible. From physics, we know $v$ must satisfy $-\Delta v = 0$ in $\Omega = (0, 1)$, which motivates solving the following constrained optimization problem:

$$\underset{u,v}{\text{minimize}}\ J(v, u) := \int_\Omega (v(x) - 1)^2\, dx + \frac{\lambda}{2}u^2 \tag{1a}$$

$$\text{subject to } -\Delta v = 0 \text{ on } \Omega, \tag{1b}$$

$$v(0) = u,\ v(1) = 0 \tag{1c}$$

$$u \in \mathbb{R}. \tag{1d}$$

Let's introduce some typical terminology for constrained optimization problems of this form: $u$ is the *control*, $v$ is the *state*, the function $J$ being minimized in (1a) is the *objective*, $0 \leq \lambda \ll 1$ is a fixed *regularization parameter*, (1b)-(1d) are called the *constraints*, and $\mathbb{R}$ in (1d) defines the set of *admissible controls*.

We will solve this problem two ways. First, we solve the problem analytically using the fact that the control $u$ comes from a finite-dimensional vector space. Then, we solve the problem numerically using a method that is applied to higher-dimensional problems, where the control is generally taken to be a function $u \in L^2(\partial\Omega)$.

## 2.1 Analytic solution.

As posed in (1), the objective is a function of two variables, the control $u$ and the state $v$. From the theory of differential equations, though, we know that each $u \in \mathbb{R}$ gives rise to a unique solution $v$ that satisfies the constraints of the problem. So, we begin by defining

the *solution operator*, $S : \mathbb{R} \to L^2(\Omega)$, such that $u \mapsto v$. This means we may re-frame the problem as

$$\underset{u}{\text{minimize}}\ j(u) := \int_\Omega (S(u)(x) - 1)^2\, dx + \frac{\lambda}{2}u^2 \tag{2a}$$

$$\text{subject to } u \in \mathbb{R}. \tag{2b}$$

To solve this problem, we compute a formula for the solution operator. From the theory of differential equations, if $v$ solves (1b), then $v(x) = ax + b$ for some $a, b \in \mathbb{R}$. Inputting the boundary conditions in (1c) therefore yields the formula $v(x) = u(1-x)$, so that $S(u)(x) = u(1-x)$ is the solution operator we wanted. Thus the modified objective becomes

$$\begin{aligned} j(u) &:= \int_\Omega (S(u)(x) - 1)^2\, dx + \frac{\lambda}{2}u^2 \\ &= \int_0^1 (u(1-x) - 1)^2\, dx + \frac{\lambda}{2}u^2 \\ &= 1 - u + \frac{1}{3}u^2 + \frac{\lambda}{2}u^2. \end{aligned}$$

We therefore solve $j'(u) = 0$ and take $\lambda \to 0$ to get $\boxed{u = 3/2.}$

## 2.2 Numerical solution using `scipy.optimize`.

To solve the problem numerically, we follow an approach that applies even when $u$ comes from an infinite-dimensional function space as is the case for higher dimensions. We begin by discretizing everything in the problem: given $M \in \mathbb{N}$, we partition $\Omega$ into a grid with boundaries $0 = x_{1/2} < x_{3/2} < \cdots < x_{M+1/2} = 1$, centers $x_k := 1/2 \cdot (x_{k-1/2} + x_{k+1/2})$, and widths $h_k := x_{k+1/2} - x_{k-1/2}$. Then define $\omega_k := [x_{k-1/2}, x_{k+1/2}]$, and approximate $v$ by $v_h = \sum_{k=1}^M v_k \mathbb{1}_{\omega_k}$. Under this discretization, we reformulate the objective and constraints to get

$$\underset{u, v_h}{\text{minimize}}\ J_h(v_h, u) := \frac{1}{2}\int_\Omega (v_h(x) - 1)^2\, dx + \frac{\lambda}{2}u^2 \tag{3a}$$

$$\text{subject to } \frac{1}{h_k^2}(-v_{k-1} + 2v_k - v_{k+1}) = 0 \text{ for } k = 2, \ldots, M-1, \tag{3b}$$

$$v_1 = u,\ v_m = 0 \tag{3c}$$

$$u \in \mathbb{R}. \tag{1d}$$

where, in general, the differential equation constraint in (3b) comes from a discrete version of the Laplacian operator $\Delta$. To solve this problem using `scipy.optimize`, we simplify the objective to get

$$J_h(v_h, u) = \frac{1}{2}\sum_{k=1}^M h_k \cdot (v_k - 1)^2 + \frac{\lambda}{2}v_1^2$$

and we rewrite the constraints in matrix-vector form as

$$\begin{bmatrix} 0 & 0 & 0 & 0 & \cdots & 0 \\ -1 & 2 & -1 & 0 & \cdots & 0 \\ 0 & -1 & 2 & -1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_k \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Then we use `scipy.minimize` on a uniform grid.

```
M = 100
h = 1 / M
regularization_lambda = 0.01


def finite_dimensional_objective(y: np.ndarray) -> float:
    average_as_vector = np.ones(y.shape)
    objective = 0.5 * h * np.sum(
        np.square(y - average_as_vector)
    ) + 0.5 * regularization_lambda * math.pow(y[0], 2)
    return objective


def build_constraint_matrix() -> sparse.lil_array:
    constraint_matrix = sparse.lil_array((M, M))
    constraint_matrix[-1, -1] = 1
    for i in range(1, M - 1):
        constraint_matrix[i, i - 1] = -1
        constraint_matrix[i, i] = 2
        constraint_matrix[i, i + 1] = -1
    return sparse.csr_matrix(constraint_matrix)


def build_constraint_vector() -> np.ndarray:
    return np.zeros(M)


constraint = opt.LinearConstraint(
    build_constraint_matrix(),
    build_constraint_vector(),
    build_constraint_vector(),
)

result = opt.minimize(
    finite_dimensional_objective,
    np.ones(M),
    method="trust-constr",
    constraints=constraint,
)
```

This results in the numerical solution for $M = 100$ plotted in figure 1, as well as the error for finer and finer grids plotted in figure 2.

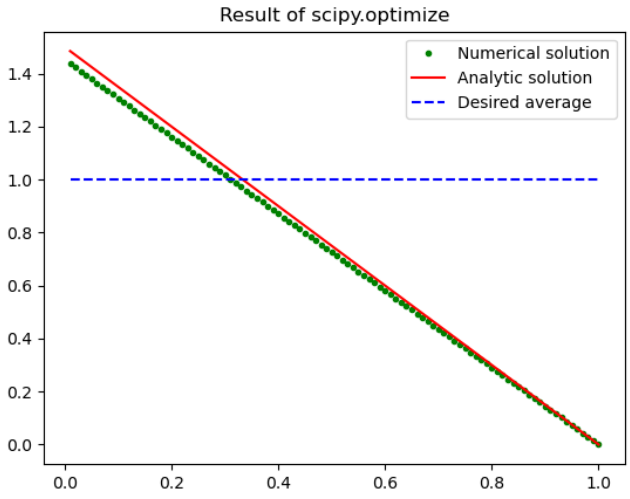


Figure 1: Numerical approximation of $v = S(u)$ from `scipy.optimize`

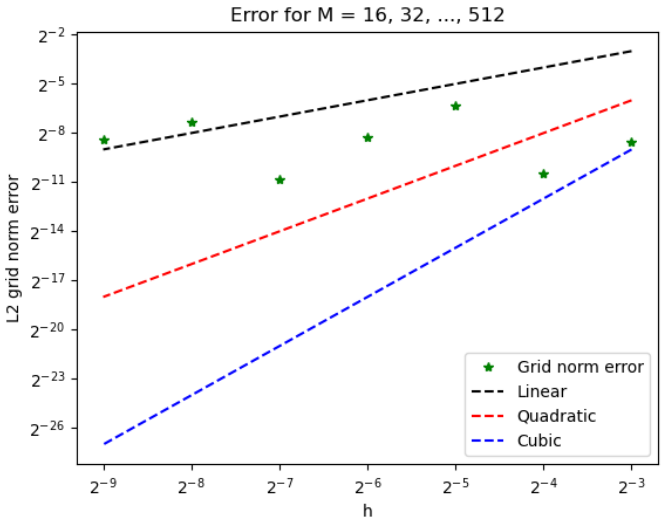


Figure 2: $L_2$ grid-norm error for `scipy.optimize`

Interestingly, we do not see that the error $\to 0$ as $h \to 0$ as we would expect.

# 3 An uncoupled distributed control problem.

The next problem involved a similar problem where the control is *distributed* instead of on the boundary:

Problem 3.1 – *1D uncoupled distributed control problem.*

As before, we have a 1-dimensional rod represented by the interval $[0,1]$, but we fix its temperature to be 0 at both ends and instead control the heat sources in the interior of the rod. Thus, where before the control $u$ was the temperature of the rod on the left endpoint, now $u \in L^2(\Omega)$ defines the heat sources. We still want the temperature of the rod to be as close to 1, pointwise, as possible. This leads the following constrained optimization problem:

$$\underset{u,v}{\text{minimize}}\ J(v,u) := \frac{1}{2}\int_\Omega (v(x)-1)^2\,dx + \frac{\lambda}{2}\int_\Omega u(x)^2\,dx \tag{4a}$$

$$\text{subject to}\quad -\Delta v = u \text{ on } \Omega, \tag{4b}$$

$$v(0) = v(1) = 0 \tag{4c}$$

$$u \in L^2(\Omega). \tag{4d}$$

$$a \le u \le b \text{ a.e. in } \Omega \tag{4e}$$

Note that the additional constraint on $u$ in (4e) is a typical one and called a *box* constraint.

Here we will not solve the problem analytically, but rather numerically via an implementation of the active set algorithm described in [2].

As a precursor to the algorithm, we start by discretizing the control, $u$. As above, we partition the domain into subdomains $\omega_k$ and approximate $u$ by a piecewise constant function $u_h = \sum_{k=1}^M u_k \mathbb{1}_{\omega_k}$, $u_1, \dots, u_M \in \mathbb{R}$. We then re-define the solution operator $S : L^2(\Omega) \to L^2(\Omega)$ that maps the control $u$ to the unique solution $v$ to the constraints in (4b)-(4c).

The main difficulty in implementing the algorithm involves finding a discretized version $S_h : \mathbb{R}^M \to L^2(\omega)$ of the solution operator $S : L^2(\Omega) \to L^2(\Omega)$. We do this by computing $v_k := S(\mathbb{1}_{\omega_k})$ for each $k = 1, \dots, M$, that is, we compute the unique solution $v_k$ to the boundary value problem

$$-\Delta v_k = \mathbb{1}_{\omega_k} \text{ on } \Omega, \tag{5a}$$

$$v_k(0) = v_k(1) = 0. \tag{4c}$$

We could very well numerically approximate the solution to this problem using the `ELLIPTIC1d` code used in part (a) of the course, but we can also find an analytic solution without too much difficulty. In 1D, (5a) is equivalent to $-v'' = \mathbb{1}_{\omega_k}$, so we begin by finding a second antiderivative for $\mathbb{1}_{\omega_k}$. Define $f : [0,1] \to \mathbb{R}$ by

$$f(x) = \int_0^x \mathbb{1}_{\omega_k}(t)\,dt + C, \quad C \in \mathbb{R}$$

$$= \begin{cases} C & \text{if } x < x_{k-1/2} \\ C + x - x_{k-1/2} & \text{if } x \in \omega_k \\ C + h_k & \text{if } x > x_{k+1/2} \end{cases}$$

so that $f' = \mathbb{1}_{\omega_k}$ by the fundamental theorem of calculus. Then define $g : [0,1] \to \mathbb{R}$ by

$$g(x) = \int_0^x f(t)\,dt + D, \quad D \in \mathbb{R}$$

$$= \begin{cases} Cx + D & \text{if } x < x_{k-1/2} \\ Cx + D + x^2/2 + x_{k-1/2}^2/2 - x \cdot x_{k-1/2} & \text{if } x \in \omega_k \\ Cx + D + h_k(x - x_k) & \text{if } x > x_{k+1/2} \end{cases}$$

so that $g' = f$ by the fundamental theorem of calculus. Thus $g'' = \mathbb{1}_{\omega_k}$, so $v = -g$ satisfies (5a). Plugging in boundary conditions yields $D = 0$, $C = -h_k(1 - x_k)$.

With this formula for $v_k$, we can define the discretized version of the solution operator $S_h : \mathbb{R}^M \to L^2(\Omega)$ by $S_h(\vec{u}) = \sum_{k=1}^M u_k v_k$. We also need the adjoint $S_h^* : L^2(\Omega) \to \mathbb{R}^n$ of this operator, which, as described on p. 100 in [2], is given by

$$\left(S_h^*(y)\right)_k = \int_\Omega y \cdot v_k\,dx.$$

We will use the `quad` integration function from `scipy.integrate` to compute this integral as necessary.

Now, as in [2], the active set algorithm for this problem is as follows:

**D0** Choose arbitrary initial vectors $\vec{u}_0, \vec{p}_0 \in \mathbb{R}^M$.

**D1** Given $\vec{u}_n$ and $\vec{p}_n$, determine the new *finite active sets* $A_{n+1}^+$ and $A_{n+1}^-$ as well as the finite inactive set $I_{n+1}$:

$$A_{n+1}^+ := \left\{k \in \{1,\dots,M\} : -\frac{p_{n,k}}{\lambda} > b\right\}$$
$$A_{n+1}^- := \left\{k \in \{1,\dots,M\} : -\frac{p_{n,k}}{\lambda} < a\right\}$$
$$I_{n+1} := \{1,\dots,M\} \setminus \left(A_{n+1}^+ \cup A_{n+1}^-\right).$$

**D2** Compute $\vec{p}_{n+1} := S_h^*\left(S_h\vec{u}_n - \mathbb{1}_\Omega\right)$ and

$$u_{n+1,k} := \begin{cases} a & \text{if } k \in A_{n+1}^- \\ -\frac{p_{n+1,k}}{\lambda} & \text{if } k \in I_{n+1} \\ b_k & \text{if } k \in A_{n+1}^+. \end{cases}$$

Proceed to **D1** for the next iteration step.

The iteration is halted once $A_{n+1}^+ = A_n^+$ and $A_{n+1}^- = A_n^-$. This algorithm is implemented as follows:

```
def analytic_v_k(x: float, k: int = 0) -> float:
    C = -cell_widths[k] * (1 - cell_centers[k])
    if x <= cell_boundaries[k]:
        return -C * x
    elif cell_boundaries[k] <= x <= cell_boundaries[k + 1]:
        return -(
            0.5 * math.pow(x, 2)
            - x * cell_boundaries[k]
            + 0.5 * math.pow(cell_boundaries[k], 2)
            + C * x
        )
    return -(cell_widths[k] * (x - cell_centers[k]) + C * x)


def integral_of_v_k(k: int) -> float:
    return quad(analytic_v_k, 0, 1, args=(k))[0]


integrals_of_v_k = np.array([integral_of_v_k(i) for i in range(M)])

initial_u = np.ones(M)
initial_p = np.ones(M)
p, u = initial_p, initial_u

(
    previous_positive_active_set,
    previous_negative_active_set,
    previous_inactive_set,
    positive_active_set,
    negative_active_set,
    inactive_set,
) = (set(), set(), set(), set(), set(), set())
n = 0
while n < 1000:
    n += 1
    print("iteration " + str(n))

    (
        previous_positive_active_set,
        previous_negative_active_set,
        previous_inactive_set,
    ) = (positive_active_set, negative_active_set, inactive_set)
    positive_active_set, negative_active_set, inactive_set = set(), set()
    , set()

    for k in range(M):
        if -p[k] / regularization_lambda > b[k]:
            positive_active_set.add(k)
        elif -p[k] / regularization_lambda < a[k]:
            negative_active_set.add(k)
```

```
        else:
            inactive_set.add(k)

    p = np.zeros(M)
    for k in range(M):
        for j in range(M):
            p[k] += u[j] * integrals_of_v_k[j]
        p[k] -= integrals_of_v_k[k]

    for k in positive_active_set:
        u[k] = a[k]
    for k in negative_active_set:
        u[k] = b[k]
    for k in inactive_set:
        u[k] = -p[k] / regularization_lambda

    if (
        positive_active_set == previous_positive_active_set
        and negative_active_set == previous_negative_active_set
    ):
        print("Found u: " + str(u))
        break
```

For $a \equiv 0$ and $b \equiv 10$, this produced the result of $u_k = 10$ for all $k = 1, \ldots, M$, and thus $v(x) = -5x(x-1)$.

# 4 Future work.

For C3, I would like to work on a coupled problem. For instance, if $\Omega_1 = [0, 1/2]$, $\Omega_2 = [1/2, 1]$, and $v_1 \in L^2(\Omega_1)$, $v_2 \in L^2(\Omega_2)$, we can try to control $u \in \mathbb{R}$ such that $v_2$ is as close to 1, pointwise, as possible, and such that

$$-\Delta v_i = 0, \quad v_1(0) = u, \quad v_1(1/2) = v_2(1/2), \quad v_1'(1/2) = v_2'(1/2), \quad v_2(1) = 0.$$

Other future directions which I may not have time for would (1) include implementing my own algorithm for the boundary control problems or (2) solving a 2D distributed control problem.

# References

[1] M. Ulbrich, *Semismooth Newton methods for variational inequalities and constrained optimization problems in function spaces*, ser. MOS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA; Mathematical Optimization Society, Philadelphia, PA, 2011, vol. 11. [Online]. Available: https://doi.org/10.1137/1.9781611970692

[2] P. Philip, "Optimal control of partial differential equations," 2013. [Online]. Available: https://www.math.lmu.de/~philip/publications/lectureNotes/philipPeter_OptimalControlOfPDE.pdf