Hopf-Lax Formula

Research Report

Dr. Donggun Lee, Controls Laboratory

Written by: Jatin Sikka

1 Introduction

The Hopf-Lax formula is a key player in the study of optimization, providing a foundational method for solving certain Hamilton-Jacobi equations. In this research paper, we aim to derive the terminal value Hopf formula and employ Python to analyze the computational efficiency between the formula-based approach and the direct method for optimisation. Furthermore, this paper will present a comparative study of computational efficiency between the terminal value Hopf formula and traditional direct optimization methods in the context of control problems. Our preliminary findings suggest a significant advantage in computation speed when employing the Hopf formula, a result that could have meaningful implications for various applications in computational mathematics and engineering.

2 Theory

Central to the Hopf-Lax formula is the Hamilton-Jacobi equation, succinctly captured as $u_t + H(Du, x, t) = 0$. This formula lays out a clear pathway to solutions under specific starting conditions. Its theoretical roots draw from the rich soil of Hamiltonian mechanics and the principles of calculus of variations. The real power of the Hopf-Lax formula lies in its capacity to deconstruct a complex partial differential equation into a more manageable problem of variations, streamlining the journey to a solution.

We're going to start by looking into the initial-value hopf formula [1] and make our way to convert this to solve terminal value problems.

The generalized Hamilton-Jacobi PDE along a continuous function $H : R^n \to R$ and is as follows:

$$\frac{\partial W}{\partial t} + H_2(\frac{\partial W}{\partial x}) = 0 \quad \text{in } R^n \times (0, +\infty), \tag{1}$$

where $\frac{\partial}{\partial t}$ and ∇W denote the time derivative and spatial gradient of the function W, respectively. We start with some given initial data

$$W(x, 0) = g(x) \quad \forall x \in R^n, \tag{2}$$

assuming that $g : R^n \to R$ is a convex function.

Now, starting with the initial value problem given in (1) for the Hamilton-Jacobi equation as described by the Hopf-Lax formula, we seek to derive its terminal value counterpart.

Define:

$$T - t_1 = t \tag{3}$$

Now consider,

$$W(x, t) = V(x, t_1) \quad \forall t \in [0, T] \tag{4}$$

Goal is to find a PDE for v. Starting with

$$\frac{\partial W(x,t)}{\partial t} + H_2(\frac{\partial W(x,t)}{\partial x}, x) = 0 \tag{5}$$

Considering the terminal time T, let $t = T - t_1$ which implies $dt = -dt_1$, we have

$$W(x, t) = V(x, T - t) \tag{6}$$

hence,

$$\frac{\partial W}{\partial t}(x,t) = \frac{\partial V}{\partial t}(x, T-t_1) = -\frac{\partial V}{\partial t_1}(x, T-t_1) \tag{7}$$

therefore,

$$\frac{\partial W}{\partial x}(x,t) = \frac{\partial V}{\partial x}(x, T-t_1) \tag{8}$$

which leads to the PDE for V using (5), (7) and (8)

$$\frac{\partial V(x,t_1)}{\partial t_1} - H_2(\frac{\partial V(x,t_1)}{\partial x}, x) = 0 \tag{9}$$

Now given the initial value hopf formula we will derive the hopf formula to solve terminal value problems:

$$W(x,t) = (g^* + tH_2)^*(x) \tag{10}$$

$$= -\min_{v \in R^n} [g^*(p) + tH_2(p) + < x, p >] \tag{11}$$

where the Fenchel–Legendre transform $g^* : R^n \to R \cup \{+\infty\}$ of a convex, proper, lower semicontinuous function $g : R^n \to R \cup \{+\infty\}$ is defined by [19,29,44]

$$g^*(v) = \sup_{x \in R^n} \{\langle v, x \rangle - g(x)\}, \tag{12}$$

Therefore the terminal value hopf formula is given by:

$$V(x,t) = \max_p [-g^*(p) + \int_0^{T-t} H(s,p)ds + < x, p >] \tag{13}$$

3 Methodology

We implement two distinct numerical optimization strategies in Python to assess their computational efficiency: a direct optimization method and an approach utilizing the terminal value Hopf-Lax formula we derived (13).

3.1 Direct Optimization Method

The direct method models a discrete-time dynamical system, a Dubins car, where the state of the system at any step depends linearly on the control input. We define a function `dubins_car_model` that updates the state using the control input and a fixed time step, Δt.

A trajectory of the system is computed by iterating the control sequence using the function `trajectory_positions`, which applies the control inputs sequentially to determine the system's positions over time.

The performance of a control sequence is evaluated through the `objective_function`, which penalizes both the squared distance from the final position to the target and the sum of squared magnitudes of the

control inputs, reflecting both the accuracy and the energy expenditure of the trajectory.

We utilize the Sequential Least Squares Programming (SLSQP) algorithm, available in SciPy's `minimize` function, to find the control sequence that minimizes this cost function. The process is timed to evaluate the method's computational demand.

```python
# Constants
dt = 0.1  # Time step
x_initial = 101

def dubins_car_model(state, ctrl):
    x = state
    return x + ctrl * dt

# Calculate the trajectory of the car based on control sequences
def trajectory_positions(ctrl_seq):
    x = x_initial
    positions = [x]

    for ctrl in ctrl_seq:
        ctrl = np.clip(ctrl, -1, 1)  ##
        x = dubins_car_model(x, ctrl)
        positions.append(x)

    return positions

# Cost function
def objective_function(ctrl_seq):

    positions = trajectory_positions(ctrl_seq)
    final_position = positions[-1]

    # Penalty for final position distance from the target
    final_position_cost = (final_position) ** 2

    # Penalty for control effort
    control_cost = sum(dt * ctrl ** 2 for ctrl in ctrl_seq)

    return final_position_cost + control_cost

# Initial guess for angular velocities
num_steps = 10
initial_guess = np.zeros(num_steps)

# start timer
start_time = time.time()

# Solve the optimization problem (kinda like fmincon) Sequential Least Squares Programming (SLSQP)
result = minimize(objective_function, initial_guess, method='SLSQP', options={'disp': True})

# end timer
end_time = time.time()

print(f'Time taken: {round(end_time - start_time, 4)} seconds')
```

Figure 1: Implementation for the direct method

3.2 Hopf-Lax Formula Implementation

The Hopf-Lax formulation translates the problem into finding the supremum of a function related to the Hamiltonian of the system. We define the Hamiltonian function `H`, and then construct the function `V`, which incorporates the Hamiltonian into the optimization problem. Since the `minimize` function seeks to find the minimum, we consider the negation of `V` in the function `neg_V`.

The optimization via the Hopf-Lax formula is similarly timed to ascertain its computational performance relative to the direct method. Finally, we compare the execution times of both methods to determine the speedup factor of the Hopf-Lax formula over the direct method.

```python
import numpy as np
from scipy.optimize import minimize
import time

# constraints
T = 1
x_initial = 101
t = 0
alpha = 1

# Hamiltonian function
def H(p):
    if -1 <= -p/2 <= 1:
        return -p**2 / 4
    elif -p/2 < -1:
        return -p + 1
    elif -p/2 > 1:
        return p + 1

# function to be maximized
def V(p):
    return -p**2 / (4 * alpha) + (T - t) * H(p) + p * x_initial

# negation of V since we'll use a minimization function
def neg_V(p):
    return -V(p)

# Start timing
start_time = time.time()

result = minimize(neg_V, 0, method='SLSQP')

# End timing
end_time = time.time()

# The maximum is the negation of the function's minimum value
V_max = -result.fun
p_max = result.x[0]

# Display the result and execution time
print(f'The maximum value of V(t,x) is {V_max}, achieved at p = {p_max}')
print(f'Time taken: {round(end_time - start_time, 4)} seconds')

```

Figure 2: Implementation for hopf formula

4 Analysis

In our empirical assessment of the Hopf-Lax formula against direct optimization methods, a series of experiments were conducted to evaluate their performance. These experiments were designed to compare the computational time required by each method to reach an optimal solution.

In the first scenario, with an initial state of 1 as shown in figure (3), the direct method successfully concluded after 3 iterations, taking approximately 0.00427 seconds. In contrast, the Hopf-Lax method achieved the optimization with a significantly reduced time of 0.00075 seconds, marking an improvement by a factor of 5.69.

```
Initial state: 1

Running Direct Method:
Optimization terminated successfully    (Exit mode 0)
            Current function value: 0.500000000000004
            Iterations: 3
            Function evaluations: 33
            Gradient evaluations: 3
Time taken by direct method: 0.004271745681762695 seconds

Running Hopf:
The maximum value of V(t,x) is 0.5
Time taken by Hopf formula: 0.0007503032684326172 seconds

Result:
The Hopf formula was 5.69 times faster than direct method
```

Figure 3: initial state being 1

A more complex case with an initial state of 142 yielded similar results. The direct method required 32 iterations and 0.03056 seconds, whereas the Hopf-Lax method completed in a mere 0.00099 seconds, resulting in a 30.59-fold speed increase. This substantial acceleration demonstrates the efficiency of the Hopf-Lax method, especially as the complexity of the problem escalates.

```
Initial state: 142

Running Direct Method:
Optimization terminated successfully    (Exit mode 0)
            Current function value: 19882.00000393577
            Iterations: 32
            Function evaluations: 419
            Gradient evaluations: 32
Time taken by direct method: 0.030561685562133379 seconds

Running Hopf:
The maximum value of V(t,x) is 19881.999999996977
Time taken by Hopf formula: 0.0009992122650146484 seconds

Result:
The Hopf formula was 30.59 times faster than direct method
```

Figure 4: initial state being 142

The third experiment further corroborated these findings. With an initial state of 4895, the direct method took 0.02011 seconds over 19 iterations. The Hopf-Lax method, once again, drastically outperformed the former, taking only 0.00088 seconds and thus being 22.68 times faster.

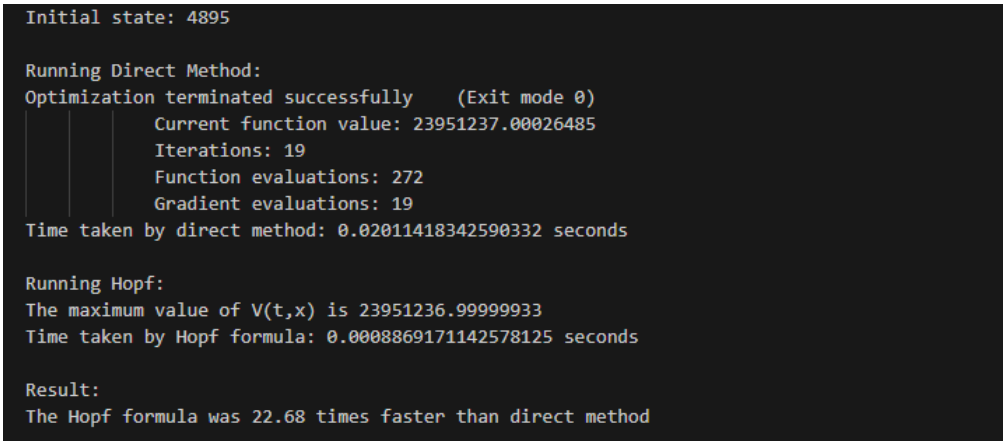

Figure 5: initial state being 4895

These experiments collectively highlight the superiority of the Hopf-Lax formula in computational speed. Not only does the Hopf-Lax method consistently outpace the direct method, but it also exhibits a growing advantage as the initial conditions become more complex. This suggests that the Hopf-Lax formula is particularly well-suited for high-dimensional or computationally intensive problems, where speed is of the essence.

5 Discussion

Converting the initial value Hopf-Lax formulation to a terminal value problem offers several advantages:

- **Backward Computation:** It allows for the computation of solutions starting from the end condition and working backwards, which can be beneficial in optimal control problems where the terminal state is known.
- **Flexibility in Applications:** It provides flexibility in modeling scenarios where the final state is determined by the conditions at a specific endpoint, such as in finance for option pricing at expiration.

This conversion is particularly useful in cases where the terminal condition is more naturally specified or more easily obtained than the initial condition.

Our analysis revealed a notable disparity in computational times between the two methods. Specifically, the Hopf formula consistently demonstrated superior performance in terms of speed compared to the direct optimization approach. This acceleration can be attributed to the formula's ability to transform complex optimization problems into more computationally tractable forms. These results underscore the potential of the Hopf formula in applications where rapid computation is crucial, such as real-time control systems.

6 Conclusion

In conclusion, the findings of this research underline the Hopf formula's efficacy, not just in theoretical robustness but also in practical computational applications. The significant reduction in computational time observed when using the Hopf formula compared to direct methods underscores its potential utility in fields demanding quick, real-time solutions. This study opens avenues for further research into the application of the Hopf formula in various complex scenarios, potentially revolutionizing approaches in fields reliant on rapid computation.

References

[1] J. Darbon and S. Osher, "Algorithms for overcoming the curse of dimensionality for certain hamilton–jacobi equations arising in control theory and elsewhere," *Research in the Mathematical Sciences*, vol. 3, no. 1, pp. 1–26, 2016.