ME 149: Midterm Exam

Optimal Control for Robotics

March 15 — Start: 6:00pm — End: 7:15pm

No calculators, notes, books, or computers allowed. Total time: 75 minutes.

Student Name: _____

How to optimize your score?

- Be neat and well organized
- For longer problems, show intermediate steps and box your answer
- If you need extra space... use the back of the page, the final (blank) page of the exam, or ask for more paper. Clearly indicate where the extra work is.
- Define all variables that you use and state any assumptions that you make.

Score: _____ / _____

1. _____ / _____
2. _____ / _____
3. _____ / _____
4. _____ / _____
5. _____ / _____
6. _____ / _____
7. _____ / _____
8. _____ / _____
9. _____ / _____

1 Newton's Method

A. Suppose that you want to solve the scalar nonlinear equation $f(x) = 0$.
The current estimate of the root is given by x_k and the next estimate of the root is given by x_{k+1}. Derive the Newton-Rhapson update that computes x_{k+1} given x_k.

B. What is the difference between the Newton–Rhapson and the secant methods for scalar root finding? In what situation would you prefer one to the other?

C. Draw a figure that clearly demonstrates a situation in which Newton's method will fail to converge to a root of the function. Why does the Newton–Rhapson method fail in this situation?

2 Midpoint Method Implementation

Implement the function `simStepMidpoint()` on the following page. This function computes a single simulation step using the midpoint method. You code should be clear, correct, and follow the best practices that have been discussed throughout the course. Use the space below for planning your solution. Write your Matlab code inside of the function template on the following page.

```
function [zNext, nEval] = simStepMidpoint(dynFun, tPrev, tNext, zPrev)
% [zNext, nEval] = simStepMidpoint(dynFun, tPrev, tNext, zPrev)
%
% Compute a single integration step using the midpoint method.
%
% INPUTS:
%    dynFun = a function handle:  dz = dynFun(t, z)
%        IN:  t = [1, nTime] = row vector of time
%        IN:  z = [nState, nTime] = matrix of states at times t
%        OUT: dz = [nState, nTime] = time-derivative of z
%    tPrev = scalar = previous time grid point
%    tNext = scalar = next time grid point
%    zPrev = [nDim, 1] = state at tPrev
%
% OUTPUTS:
%    zNext = [nDim, 1] = state at tNext
%    nEval = scalar = numer of internal calls to dynamics function
%









end
```

3 Bisection Search

Use a bisection search to iteratively reduce the interval that is known to bracket the root of the function shown in Figure 1. **Populate the table below**, showing the bracket for the first five iterations and the new point that will be evaluated on that iteration.

- `Iter 0:  bracket:  [-1.000,       1.000]    xNew =`
- `Iter 1:  bracket:  [                  ]    xNew =`
- `Iter 2:  bracket:  [                  ]    xNew =`
- `Iter 3:  bracket:  [                  ]    xNew =`
- `Iter 4:  bracket:  [                  ]    xNew =`

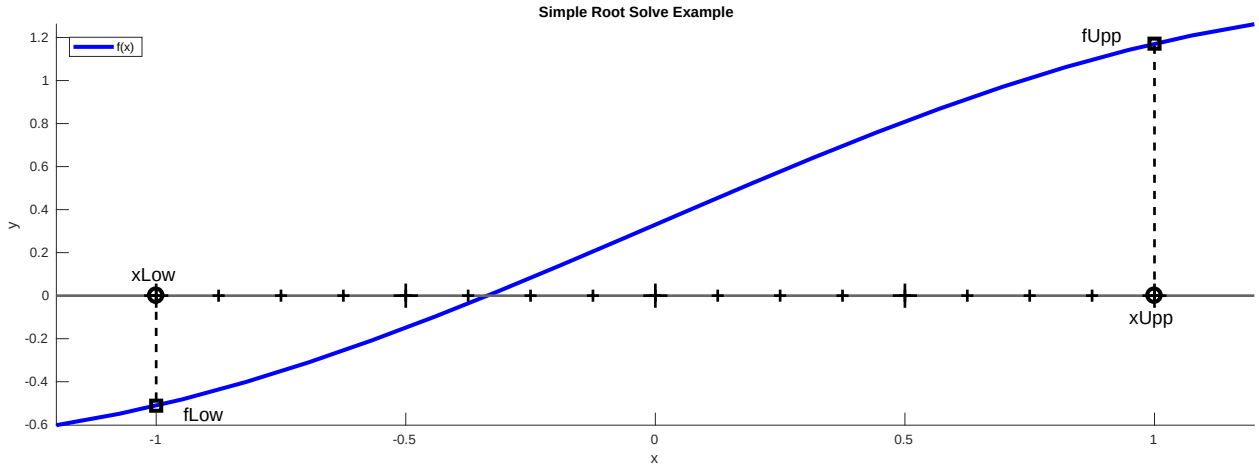

Figure 1: Bisection Search Example Figure

4 Scalar Taylor Series

Write Taylor series approximation of $f(t)$ to second-order around the point t_0.

$f(t) \approx$

5 Vector Taylor Series

Write the Taylor series approximation of $f(t, \boldsymbol{x}, \boldsymbol{u})$ to first-order about the point: t_0, $\boldsymbol{x}_0$, and $\boldsymbol{u}_0$.

$f(t, \boldsymbol{x}, \boldsymbol{u}) \approx$

6 Function Handle Gymnastics

What will the following script print to the command prompt?
For each part A, B, C, show your work and then put a box around the text that Matlab will print to the command prompt.

```
%% Set up the workspace
a = 1; b = 2; c = 3; d = 5; e = 7;
trigFun = @(t)( b + sin(a + t) );
funTimes = @(x, y)( d * x + b );
addFun = @(x, z)( [x + c; z - b] );
chainFun = @(y)( sum(addFun(y, e)) );
printFun = @(a, b)( fprintf('%s: %d\n', a, b) );
%% PART A:
A = trigFun(-a);    printFun('A', A);
%% PART B:
B = funTimes(b, d);    printFun('B', B);
%% PART C:
C = chainFun(c);    printFun('C', C);
```

Part A:

Part B:

Part C:

7 Matlab Programming Style

The program below simulates a simple pendulum. It works, but is written poorly.
Clearly identify at least 5 distinct issues with the code.

```
function runSloppySimulation()
tLow = -0.15;  % start time
tUpp = 2.9234;  % stop time
dynFun = @(t, z)( [z(2); -sin(z(1))] );
zInit = [-0.2; 0.6];
[tGrid, zGrid] = runSimBadly(dynFun, tLow, tUpp, zInit);
plot(tGrid, zGrid);
end

function [tGrid, zGrid] = runSimBadly(dynFun, tLow, tUpp, zInit)
tGrid = tLow:0.1:tUpp;
zGrid = zeros(2)
zGrid = zInit;
nDim = length(zInit);
for i = 2:31
    dz = dynFun(tGrid(i), zGrid(:, i-1))
    zGrid(:, i) = zGrid(:, i-1) + 0.1 * dz;
end
end
```

8 Linearized Dynamical System

Given the non-linear system dynamics described below,

$$\dot{\boldsymbol{z}} = \begin{bmatrix} \dot{\alpha} \\ \dot{\gamma} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \beta * \gamma + \theta^2 \\ \alpha\,\sigma - \theta \\ \sin(\sigma) + \beta\,\gamma \end{bmatrix} = \boldsymbol{f}(\boldsymbol{z}, \boldsymbol{u}) \qquad \boldsymbol{z} = \begin{bmatrix} \alpha \\ \gamma \\ \theta \end{bmatrix} \qquad \boldsymbol{u} = \begin{bmatrix} \beta \\ \sigma \end{bmatrix}$$

A. **List the state variables:**

B. **List the control variables:**

C. **What is the difference between a state and a control variable?**

D. **Compute:** $\dfrac{\delta \boldsymbol{f}}{\delta \boldsymbol{z}} =$

E. **Compute:** $\dfrac{\delta \boldsymbol{f}}{\delta \boldsymbol{u}} =$

9 Trajectory Optimization

Given the following continuous-time trajectory optimization problem.

$$\begin{aligned} &\text{minimize:} && \int_0^T \boldsymbol{g}(t,\, \boldsymbol{x},\, \boldsymbol{u})\, dt \\ &\text{subject to:} && \mathbf{0} = \boldsymbol{h}(\boldsymbol{x}(0),\, \boldsymbol{x}(T)) \\ &\text{dynamics:} && \dot{\boldsymbol{x}} = \boldsymbol{f}(t,\, \boldsymbol{x},\, \boldsymbol{u}) \end{aligned}$$

Suppose that you plan to solve the optimization using direct multiple shooting with Euler's method on a uniform grid of N segments with one integration step per segment. Write out the decision variables, objective function, and constraints that will form the resulting non-linear program. The objective function and constraints should be written in terms of the decision variables, known parameters (T), and known functions ($\boldsymbol{g}$, $\boldsymbol{h}$, $\boldsymbol{f}$).

A. **Decision Variables**

B. **Objective Function**

C. **Boundary Constraints**

D. **System Dynamics Constraints**

Blank page for additional work space for any problem.