Modellierung und Simulation **Übung 03**
WS 2016/17 **Abgabefrist: Siehe elearning bzw. exchange**

Name: _____ **Aufwand (h):** _____

Punkte: _____

Aufgabe 1 (3 + 3 + 2 = 8 Pkt): Modellierung einer Mondlandung

(a) Modellieren Sie in SIMULINK die Landung einer Landefähre, wie sie in den Unterlagen zur Vorlesung (idealisiert und vereinfacht) formuliert ist.
Gehen Sie von einer anfänglichen Sinkgeschwindigkeit von 10 m/s und einer Ausgangshöhe von 1000 m aus; die Bremskraftkoeffizienten der beiden verfügbaren Bremsstufen seien 42,000 und 34,000, und die Masse der Landefähre 20,000 kg.
Ermitteln Sie durch simulationsgesteuerte Optimierung optimale Start- und End-Zeitpunkte für die Bremsphasen; dokumentieren Sie Ihre optimalen Ergebnisse in Bezug auf Landezeitpunkt und Geschwindigkeit beim Aufsetzen.

(b) Wirken sich Änderungen am Solver bei der Simulation Ihres Modells signifikant aus? Dokumentieren Sie Veränderungen durch die Verwendung von anderen Integrationsmethoden und veränderten Schrittweiteneinstellungen. Wie erklären Sie sich dieses Ergebnis?

(c) Nehmen Sie an, daß durch die Bremsung die Masse der Landefähre verringert wird (mit Faktor 0.001). Adaptieren Sie Ihr Modell entsprechend und dokumentieren Sie Auswirkungen in Bezug auf die Landung Ihrer Landefähre.
Ermitteln Sie erneut optimale Bremsparameter, und dokumentieren Sie Ihre neue Landezeit und Aufsetzgeschwindigkeit.

Aufgabe 2 (8 + 4 + 4 = 16 Pkt): Optimierung von Simulationsmodellen mit ES

(a) Implementieren Sie in MATLAB einen Optimierungsalgorithmus basierend auf einer Evolutionsstrategie, der die Simulation der Mondlandung aus Aufgabe 1 optimiert.

(b) Dokumentieren Sie Ihre Implementierung sowie Ihre Überlegungen zu den folgenden Fragen:
- Was muß *optimiert* werden, was ist eine geeignete Fitnessfunktion?
- Welche Parameter können *modifiziert* werden?
- Warum gerade soll gerade eine Evolutionsstrategie verwendet werden und nicht z.B. ein genetischer Algorithmus, Tabu Suche, oder Ant Colony Optimization?

(c) Welche algorithmischen Parametersetzungen führen zu eher guten, welche zu eher schlechten Ergebnissen? Wie bewerten Sie die Performance eines solchen Optimierungslaufes?
Wirkt sich die Anpassung der Mutationsweite aus?
Dokumentieren Sie Ihre Ergebnisse mit unterschiedlichen Parameter-Settings, zeigen Sie Statistiken (Graphiken, Tabellen, …).

Hinweise: Geben Sie Ihre Ausarbeitung gedruckt auf Papier ab.
Abgegebene Beispiele müssen in der Übungsstunde präsentiert werden können.

1 Modellierung einer Mondlandung

1.1 Simulink Modell der Mondlandung

Die Abbildung 1 zeigt das implementierte Modell der Mondlandung. -0.008801562 -7.17180995

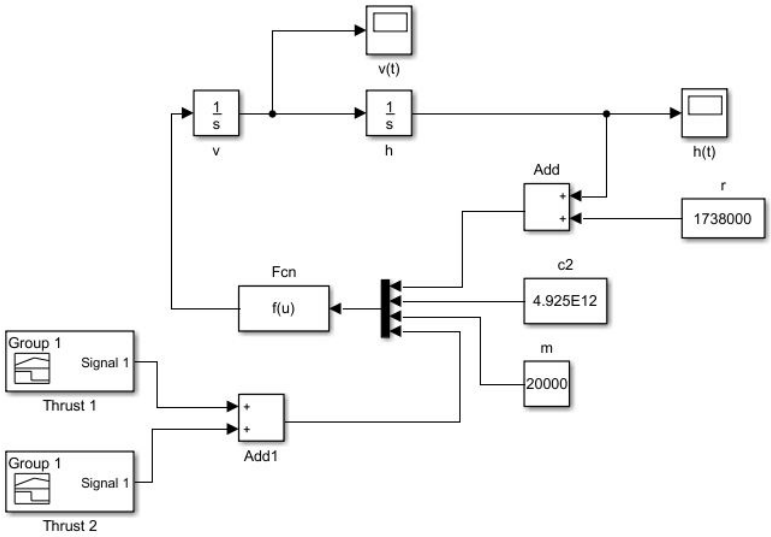

Abbildung 1: Simulink Modell der Mondlandung

1.1.1 Test 1

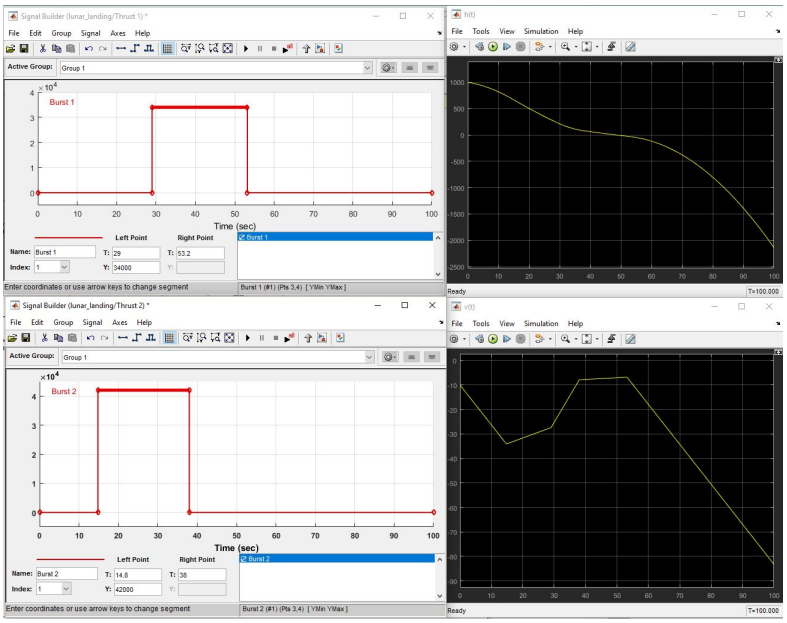

Abbildung 2: $h_{(t)}$= -0.0088 m, $v_{(t)}$= -7.17 m/s

1.1.2 Test 2

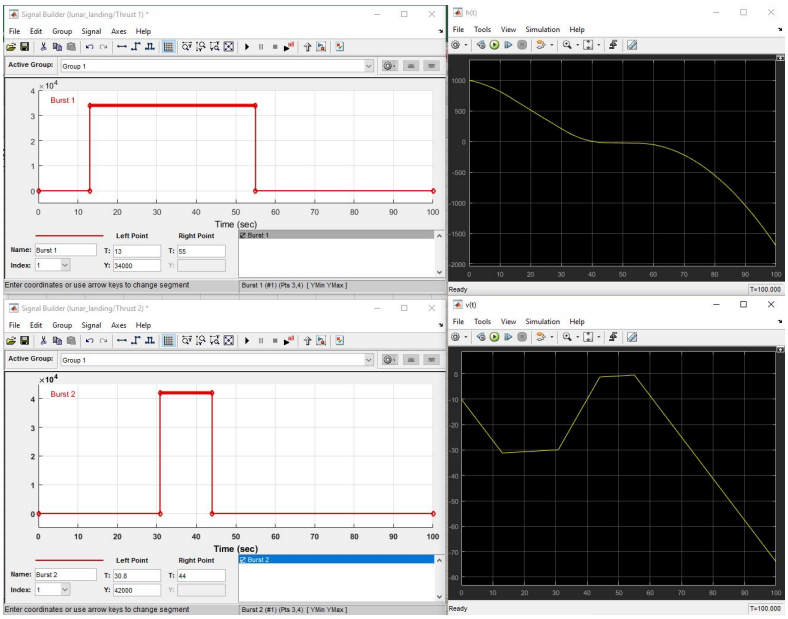

Abbildung 3: $h_{(t)}$= -0.0256 m, $v_{(t)}$= -8.72 m/s

1.1.3 Test 3

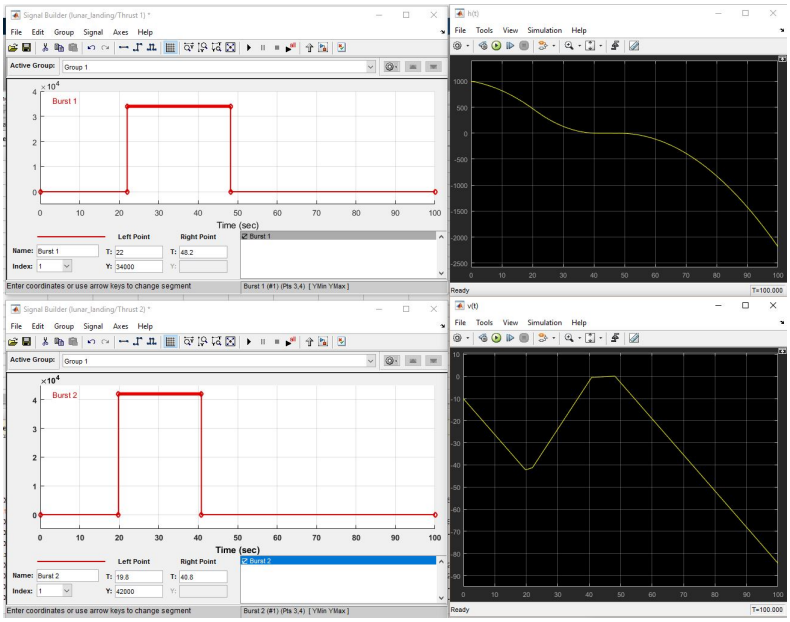

Abbildung 4: $h_{(t)}$= -0.0018 m, $v_{(t)}$= -1.44 m/s

1.1.4 Testergebnisse

Nr.	Start Thrust 1	End Thrust 1	Start Thrust 2	End Thrust 2	h(t)	v(t)
1	29	53.2	14.8	38	-0.008801562	-7.17180995
2	13	55	30.8	44	-0.02586503	-8.71786497
3	22	48.8	19.8	40.8	0.001799328	-1.441021505

Abbildung 5: Tabelle mit den Gesamtergebnissen

1.2 Solver Modifikationen

Als Beispiel wird der Lösungskandidat aus Abschnitt 1.1.4 herangezogen, mit dem gezeigt wird, wie sich die Änderungen an den Einstellungen auswirken.

1.2.1 Test Schrittweite 1 und 0.01

Nr.	Start Thrust 1	End Thrust 1	Start Thrust 2	End Thrust 2	Änderung	h(t)	v(t)
1	22	48.8	19.8	40.8	step=0.01	0.001799328	-1.441021505
					step=0.1	-0.005784244	-0.313609826
					step=1	-1.922308429	-9.593092329

Abbildung 6: Testergebnisse mit den drei Schrittweiten 0.01, 0.1 und 1

Die gravierenden Unterschiede entstehen, da mit einer zu großen Schrittweite schnelle Veränderungen im System nicht erkannt werden können. Bei schnellen Veränderungen im System ist eine kleine Schrittweite von Vorteil, bei langsamen Veränderungen eine große Schrittweite.

1.2.2 Test verschiedener Integrationsmethoden

Nr.	Start Thrust 1	End Thrust 1	Start Thrust 2	End Thrust 2	Änderung	h(t)	v(t)
1	22	48.8	19.8	40.8	ode5	0.001799328	-1.441021505
					euler	-0.0021843	-1.6195
					fixed auto	-0.0021854	-1.5762
					variable auto	-0.0021856	-0.4263

Abbildung 7: Testergebnisse mit den vier Integrationsmethoden

Die verschiedenen Integrationsmethoden *ode5*, *euler* unterscheiden sich durch ihren lokalen und globalen Error. Diese Error Indikatoren werden durch verschiedene Mechanismen die bei der Integration angewandt werden, wie gewichtetes Mittel, Miteinbeziehung von vorherigen und/oder geschätzten Nachfolgern und der Verwendung empirischer Faktoren, beeinflusst.

1.3 Modifikation der Mondlandungsmodell

Die Abbildung zeigt das modifizierte Modell der Mondlandung.

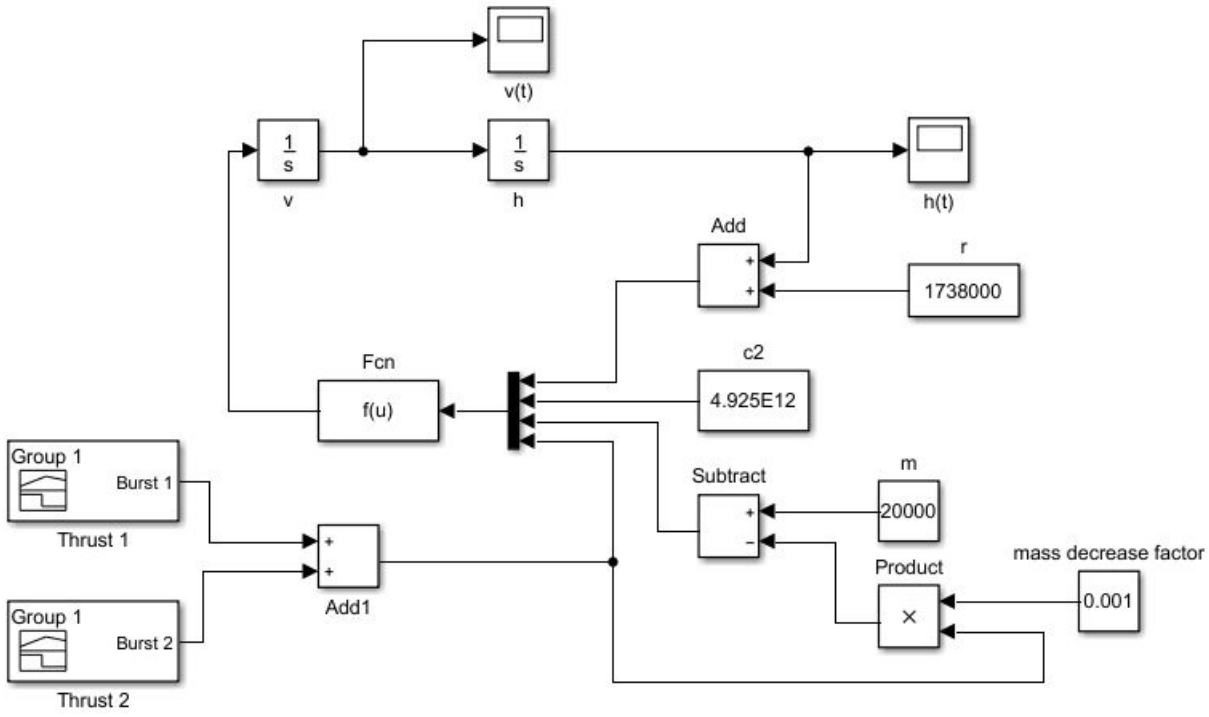

Abbildung 8: Mondladungsmodell mit Massenreduktion beim Bremsen

1.3.1 Test 1

1.3.2 Test 2

1.3.3 Test 3

1.3.4 Test 4

1.3.5 Test 5

2 Optimierung mit einen Evolutionsalgorithmus

2.1 Der Evolutionsalgorithmus

Listing 1: Funktion die einen Lösungskandidaten initialisiert

```
function solution = initialize(solution)
% Initializes the solution object

% First Thrust
solution.start1 = rand * 100;
solution.end1 = rand * 100;
% Ensure that end is after start
while solution.end1 <= solution.start1
    solution.end1 = rand * 100;
end
% Second Thrust
solution.start2 = rand * 100;
solution.end2 = rand * 100;
% Ensure that end is after start
while solution.end2 <= solution.start2
    solution.end2 = rand * 100;
end

solution.quality          = [];
solution.heightPRogress   = [];
solution.velocityProgress = [];

end
```

Listing 2: Funktion, welche die Mutanten aus einem Elter erzeugt

```
function result = bread(solution, cfg)
% Breads the mutants for the given candidate

result      = [];
population  = [];
qualities   = [];

% generate lambda mutants for the given candidate
for i=1:cfg.lambda
    % mutate from parent
    solution          = mutate(solution, cfg);
    % evaluate new mutant
    solution          = evaluate(solution, cfg);

    % set new candidate and map quality to candidate index
    population{i}     = solution;
    qualities(i, 1)   = solution.quality;
    qualities(i, 2)   = i;
end

% build result
result.population = population;
result.qualities  = qualities;

end
```

Listing 3: Funktion, welche einen Elter zu einem Mutanten konvertiert

```
function solution = mutate(solution, cfg)
% Mutates the given candidate

% Mutate first thrust
tmpStart1 = mod(solution.start1 + (rand * cfg.delta), 100);
if tmpStart1 > cfg.maxValue
    solution.start1 = rand * cfg.delta;
else
    solution.start1 = tmpStart1;
end
tmpEnd1    = mod(solution.end1 + (rand * cfg.delta), 100);
if tmpEnd1 > cfg.maxValue
    solution.end1 = rand * cfg.delta;
else
    solution.end1 = tmpEnd1;
end
while solution.end1 <= solution.start1
    solution.end1 = solution.end1 + (rand * cfg.delta);
end

% Mutate second thrust
tmpStart2 = mod(solution.start2 + (rand * cfg.delta), 100);
if tmpStart2 > cfg.maxValue
    solution.start2 = rand * cfg.delta;
else
    solution.start2 = tmpStart2;
end
tmpEnd2   = mod(solution.end2 + (rand * cfg.delta), 100);
if tmpEnd2 > cfg.maxValue
    solution.end2 = rand * cfg.delta;
else
    solution.end2 = tmpEnd2;
end
while solution.end2 <= solution.start2
    solution.end2 = solution.end2 + (rand * cfg.delta);
end

% Reset members
solution.quality          = [];
solution.heightPRogress   = [];
solution.velocityProgress = [];

end
```

Listing 4: Funktion, welche die Simulation für einen Lösungskandidaten durchführt

```
function solution = evaluate(solution, cfg)
% Evaluates the simulation for the given candidate

% Prepare model parameters
modelParams = [1, solution.start1, solution.end1,solution.start2, solution.end2];
% run simulation
[T,X] = sim(cfg.simFile,100,cfg.simParams,modelParams);

% get height
h    = X.signals(1).values;
% get min height
minH = min(h);

% we haven't fully landed
if minH > 0
```

```
    solution.quality = minH/10;
% We have landed already
else
    impactIdx        = find(h < 0);
    solution.quality = -X.signals(2).values(impactIdx(1));
end

end
```

Listing 5: Funktion, welche die Simulation für einen Lösungskandidaten durchführt

```
cfg                   = [];
cfg.timeSpan          = 100;
cfg.mu                = 10;
cfg.lambda            = 100;
cfg.delta             = 5;
cfg.deltaStep         = 0.1;
cfg.simFile           = 'parametrized_lunar_landing';
cfg.simParams         = simget(cfg.simFile);
cfg.maxValue          = 100;
cfg.maxUnsuccessCount = 100;
cfg.curUnsuccessCount = 100;
cfg.maxGenerations    = 20;

runCount          = 1;
run               = 1;
generationCount     = 1;
curUnsuccessCount = 1;
curBestQuality    = 1000;
bestQualities     = [];
population        = [];
initSolution      = [];

% [14.4130]    [13.6334]    [17.8422]    [ 99.6011]    [9.7887e-04]    []    []
% initial candiadte
initSolution.start1 = 14;
initSolution.end1   = 13;
initSolution.start2 = 17;
initSolution.end2   = 99;
population{1}       = initSolution;

while (run == 1)
    result        = [];
    newGeneration = [];
    qualities     = [];

    % 1. Select
    for i=1:cfg.mu
        idxSize          = size(population);              % get size
        % break if no further mutant available
        if idxSize(:,2) == 0
            break;
        end

        % random selection
        randMatrix       = randperm(idxSize(:,2));
        idx              = randMatrix(1:1);
        solution         = population{idx};
        population(:,idx) = [];

        % 2. Bread, 3. Mutate
        result        = bread(solution, cfg);
```

```
52          newGeneration = [newGeneration result.population];
53          qualities     = [qualities; result.qualities];
54      end
55
56      % Selektion of the mu best
57      qualities = sortrows(qualities);
58      population     = [];
59      for i=1:1:cfg.mu
60        population{i} = newGeneration{qualities(i,2)};
61      end
62
63      % Modify delta depending on qualtity
64      bestQuality                = qualities(1,1);
65      sizeQualities              = size(qualities);
66      bestQualities(runCount,1) = bestQuality;
67      betterCounter              = 0;
68      for i=1:1:sizeQualities(1,2)
69          if curBestQuality < bestQuality
70              break;
71          end
72          betterCounter = betterCounter + 1;
73      end
74      % new best quality found
75      if curBestQuality > bestQuality
76          curBestQuality  = bestQuality;
77      else
78          curUnsuccessCount = curUnsuccessCount + 1;
79      end
80      % decrease step because quality is getting better
81      if betterCounter >= (sizeQualities(1,1) / 5)
82          cfg.delta = cfg.delta - cfg.deltaStep;
83      % increase step because quality is getting worse
84      else
85          cfg.delta = cfg.delta + cfg.deltaStep;
86      end
87
88      oldBestQuality = curBestQuality;
89      % Determine if continue
90      run             = ((generationCount < cfg.maxGenerations) && (curUnsuccessCount <
    ↪  cfg.curUnsuccessCount));
91      generationCount = generationCount + 1;
92      runCount        = runCount + 1;
93  end
94
95  plot(bestQualities);
```

2.2 Optimierungen des Evolutionsalgorithmus

2.2.1 Was ist eine geeignete Fitnessfunktion ?

2.2.2 Was muss optimiert werden ?

2.2.3 Welche Parameter können modifiziert werden ?

2.2.4 Warum ein Evolutionsalgorithmus ?

2.3 Auswertung der Testfälle

2.3.1 Test 1

2.3.2 Test 2

2.3.3 Test 3

2.3.4 Test 4

2.3.5 Test 5

2.3.6 Gegenüberstellung der Tests