

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 Schrittweitereinstellungen. 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.

## Übung 1

# 1 Modellierung einer Mondlandung

## 1.1 Simulink Modell der Mondlandung

Die Abbildung 1 zeigt das implementierte Modell der Mondlandung.

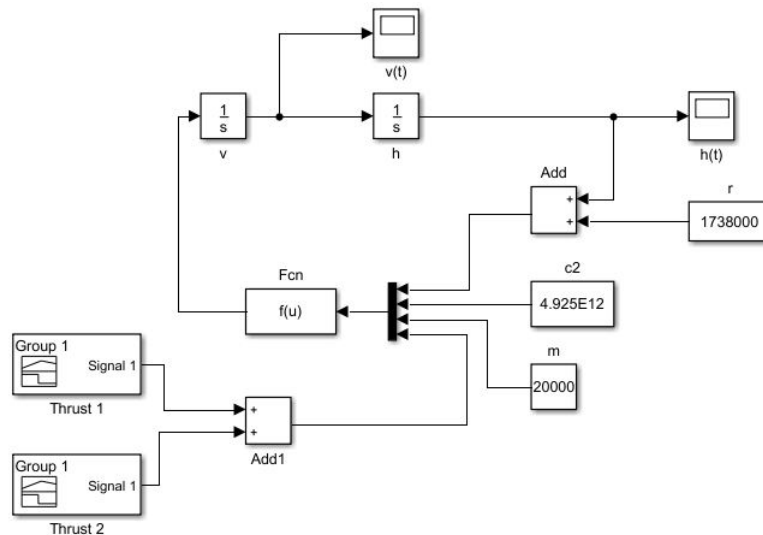


Abbildung 1: Simulink Modell der Mondlandung

### 1.1.1 Test 1

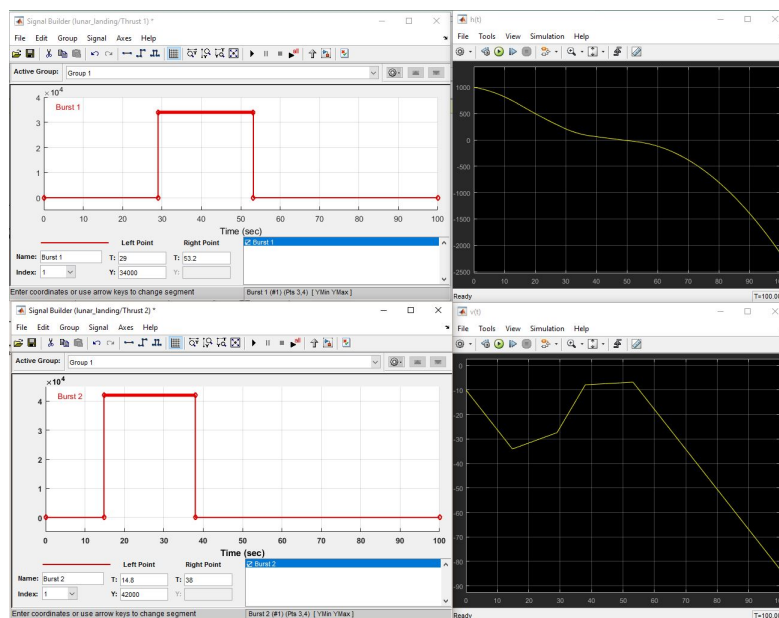


Abbildung 2:  $h(t) = -0.0088 \text{ m}$ ,  $v(t) = -7.17 \text{ m/s}$

## Übung 1

### 1.1.2 Test 2

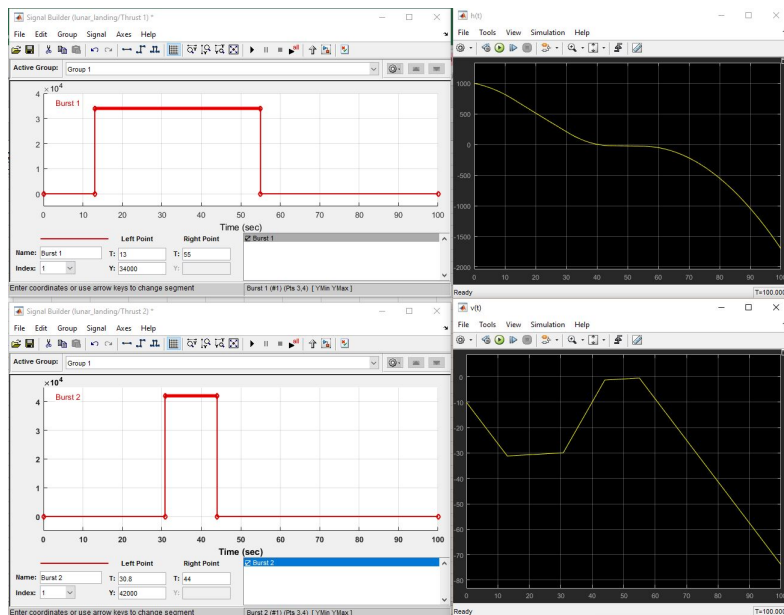


Abbildung 3:  $h(t) = -0.0256 \text{ m}$ ,  $v(t) = -8.72 \text{ m/s}$

### 1.1.3 Test 3

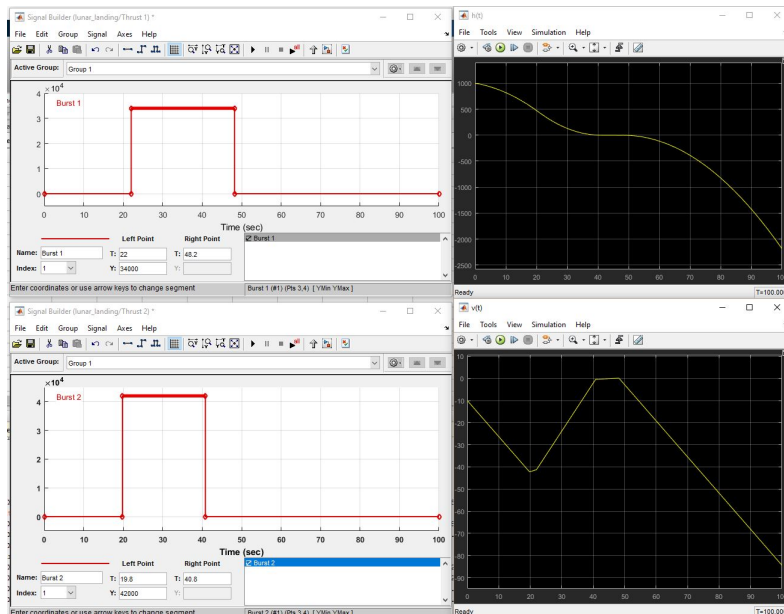


Abbildung 4:  $h(t) = -0.0018 \text{ m}$ ,  $v(t) = -1.44 \text{ m/s}$

## Übung 1

### 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.

Listing 1: Testprogramm für die Simulation mit modifiziertem Solver

```

1 % Prepare model parameters
2 fileName = 'parametrized_lunar_landing_modified';
3 modelParams = [1 22 48.2 19.8 40.8];
4 simParams = simget(fileName);
5
6 % -----
7 % test euler
8 % -----
9 simParams.Solver = 'euler';
10 [T,X] = sim(fileName,100,simParams,modelParams);
11
12 h = X.signals(1).values;
13 minHEuler = min(h);
14 impactIdx = find(h < 0);
15 qualityEuler = X.signals(2).values(impactIdx(1));
16
17 % -----
18
19 % -----
20 % Test fix step
21 % -----
22 simParams.Solver = 'FixedStepAuto';
23 [T,X] = sim(fileName,100,simParams,modelParams);
24
25 h = X.signals(1).values;
26 minHAuto = min(h);
27 impactIdx = find(h < 0);
28 qualityAuto = X.signals(2).values(impactIdx(1));
29 % -----
30
31 % -----
32 % Test fix step
33 % -----
34 simParams.Solver = 'VariableStepAuto';
35 [T,X] = sim(fileName,100,simParams,modelParams);
36
37 h = X.signals(1).values;
38 minHVAuto = min(h);
39 impactIdx = find(h < 0);
40 qualityVAuto = X.signals(2).values(impactIdx(1));
41 % -----

```

## Übung 1

### 1.3 Modifikation der Mondlandungsmodell

Die Abbildung 8 zeigt das modifizierte Modell der Mondlandung.

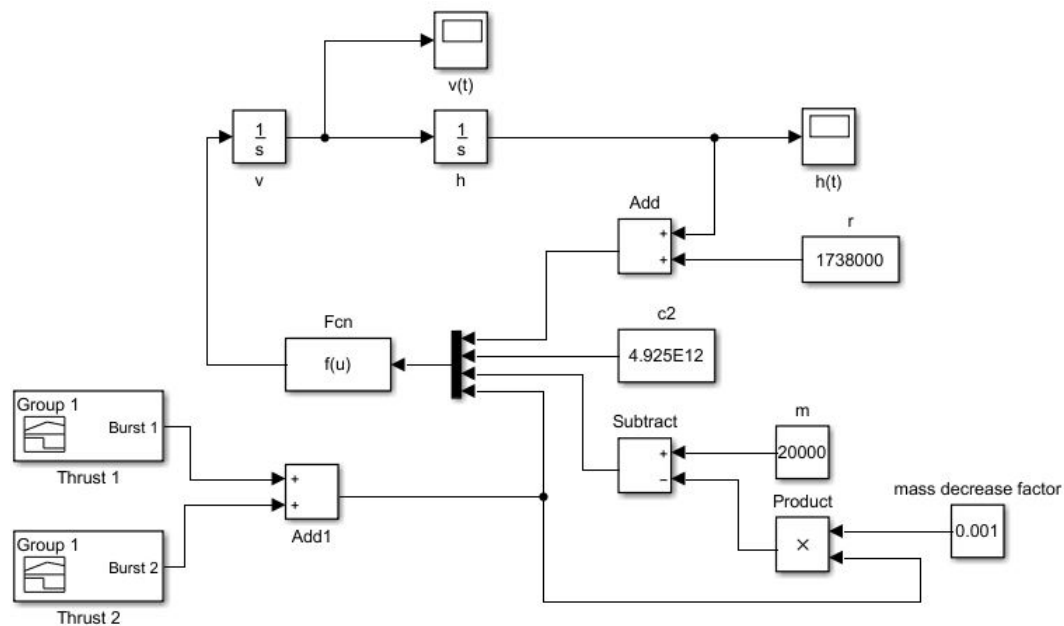


Abbildung 8: Mondlandungsmodell mit Massenreduktion beim Bremsen

#### 1.3.1 Test 1

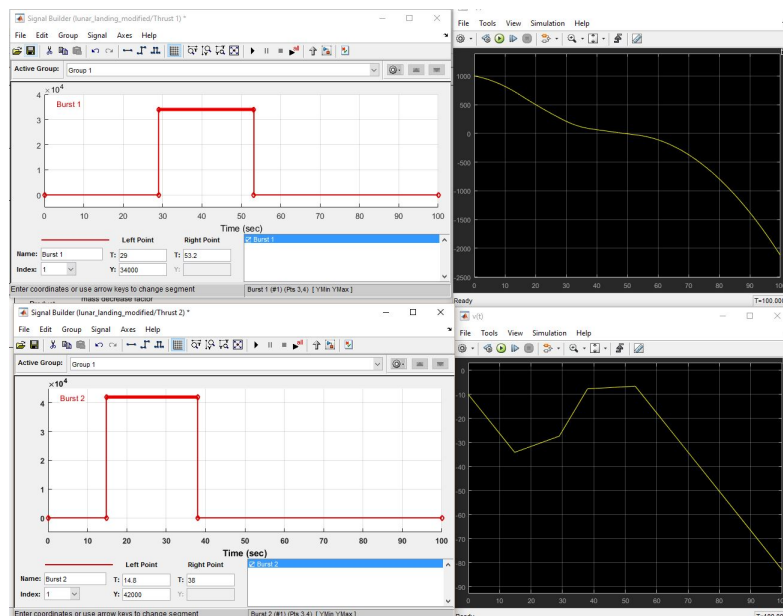


Abbildung 9:  $h(t) = -0.0685 \text{ m}$ ,  $v(t) = -6.91 \text{ m/s}$

## Übung 1

### 1.3.2 Test 2

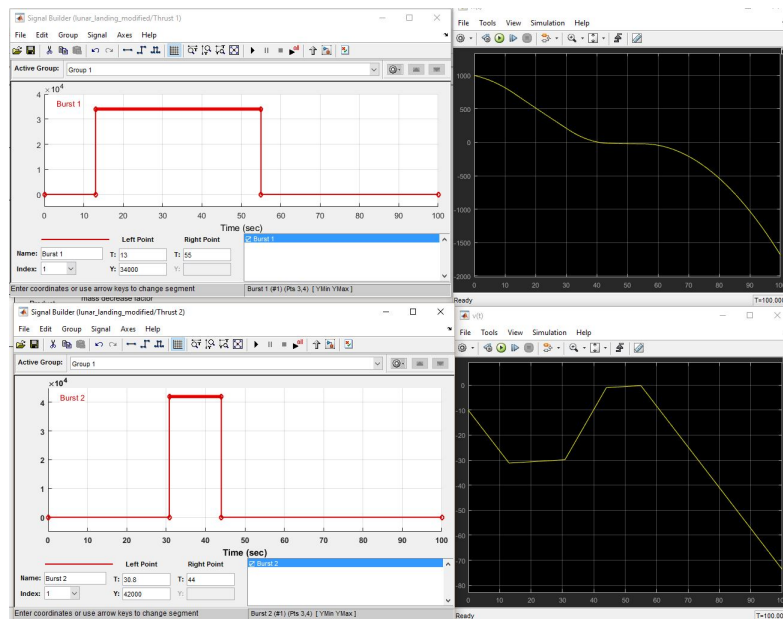


Abbildung 10:  $h(t) = -0.033 \text{ m}$ ,  $v(t) = -8.088 \text{ m/s}$

### 1.3.3 Test 3

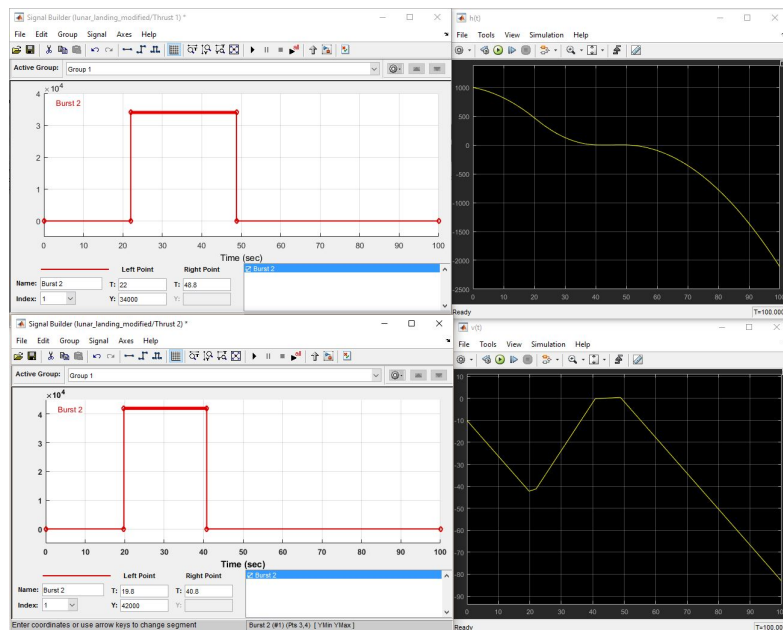


Abbildung 11:  $h(t) = -0.0088 \text{ m}$ ,  $v(t) = -3.4123 \text{ m/s}$

## Übung 1

## 1.3.4 Testergebnisse

Nr.	Start Thrust 1	End Thrust 1	Start Thrust 2	End Thrust 2	h(t)	v(t)	h(t)	v(t)	h(t) div	v(t) div
1	29	53.2	14.8	38	-0.06852981	-6.908718221	-0.008801562	-7.17180995	0.059728247	-0.2630917
2	13	55	30.8	44	-0.032871106	-8.08786718	-0.02586503	-8.71786497	0.007006076	-0.6299978
3	22	48.8	19.8	40.8	-0.008805211	-3.412267071	0.001799328	-1.441021505	0.010604539	1.97124557

Abbildung 12: Tabelle mit den Gesamtergebnissen

Es wurden dieselben Parametervektoren wie für das originale Modell verwendet, wobei sich die Resultate für die ersten beiden Testfälle verbessert und für den dritten Testfall akzeptabel verschlechtert haben.



## 2 Optimierung mit einem Evolutionsalgorithmus

### 2.1 Der Evolutionsalgorithmus

Folgende Quelltexte sind das implementierte Hauptprogramm und die implementierten Funktionen des Evolutionsalgorithmus.

Listing 2: Hauptprogramm

```

1  cfg                      = [];
2  cfg.timeSpan             = 100;
3  cfg.mu                   = 10;
4  cfg.lambda               = 100;
5  cfg.delta                = 5;
6  cfg.deltaStep            = 0.1;
7  cfg.simFile              = 'parametrized_lunar_landing';
8  cfg.simParams            = simget(cfg.simFile);
9  cfg.maxValue              = 100;
10 cfg.maxUnsuccessCount    = 100;
11 cfg.curUnsuccessCount    = 100;
12 cfg.maxGenerations       = 20;
13
14 runCount                 = 1;
15 run                      = 1;
16 generationCount          = 1;
17 curUnsuccessCount        = 1;
18 curBestQuality            = 1000;
19 bestQualities            = [];
20 population               = [];
21 initSolution              = [];
22
23 % [14.4130]   [13.6334]   [17.8422]   [ 99.6011]   [9.7887e-04]   []   []
24 % initial candiadte
25 initSolution.start1 = 14;
26 initSolution.end1   = 13;
27 initSolution.start2 = 17;
28 initSolution.end2   = 99;
29 population{1}       = initSolution;
30
31 while (run == 1)
32     result          = [];
33     newGeneration    = [];
34     qualities        = [];
35
36     % 1. Select
37     for i=1:cfg.mu
38         idxSize      = size(population);           % get size
39         % break if no further mutant available
40         if idxSize(:,2) == 0
41             break;
42         end
43
44         % random selection
45         randMatrix    = randperm(idxSize(:,2));
46         idx           = randMatrix(1:1);
47         solution       = population{idx};
48         population(:,idx) = [];
49
50         % 2. Bread, 3. Mutate
51         result         = bread(solution, cfg);
52         newGeneration  = [newGeneration result.population];
53         qualities      = [qualities; result.qualities];
54     end

```

# Übung 1

```

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 quality
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);

```

Listing 3: Funktion die einen Lösungskandidaten initialisiert

```

1 function solution = initialize(solution)
2 % Initializes the solution object
3
4 % First Thrust
5 solution.start1 = rand * 100;
6 solution.end1 = rand * 100;
7 % Ensure that end is after start
8 while solution.end1 <= solution.start1
9     solution.end1 = rand * 100;
10 end
11 % Second Thrust
12 solution.start2 = rand * 100;
13 solution.end2 = rand * 100;
14 % Ensure that end is after start
15 while solution.end2 <= solution.start2
16     solution.end2 = rand * 100;
17 end

```

# Übung 1

```

18
19 solution.quality          = [];
20 solution.heightProgress  = [];
21 solution.velocityProgress = [];
22
23 end

```

Listing 4: Funktion, welche die Mutanten aus einem Elter erzeugt

```

1 function result = bread(solution, cfg)
2 % Breads the mutants for the given candidate
3
4 result      = [];
5 population  = [];
6 qualities   = [];
7
8 % generate lambda mutants for the given candidate
9 for i=1:cfg.lambda
10     % mutate from parent
11     solution      = mutate(solution, cfg);
12     % evaluate new mutant
13     solution      = evaluate(solution, cfg);
14
15     % set new candidate and map quality to candidate index
16     population{i} = solution;
17     qualities(i, 1) = solution.quality;
18     qualities(i, 2) = i;
19 end
20
21 % build result
22 result.population = population;
23 result.qualities  = qualities;
24
25 end

```

Listing 5: Funktion, welche einen Elter zu einem Mutanten konvertiert

```

1 function solution = mutate(solution, cfg)
2 % Mutates the given candidate
3
4 % Mutate first thrust
5 tmpStart1 = mod(solution.start1 + (rand * cfg.delta), 100);
6 if tmpStart1 > cfg.maxValue
7     solution.start1 = rand * cfg.delta;
8 else
9     solution.start1 = tmpStart1;
10 end
11 tmpEnd1    = mod(solution.end1 + (rand * cfg.delta), 100);
12 if tmpEnd1 > cfg.maxValue
13     solution.end1 = rand * cfg.delta;
14 else
15     solution.end1 = tmpEnd1;
16 end
17 while solution.end1 <= solution.start1
18     solution.end1 = solution.end1 + (rand * cfg.delta);
19 end
20
21 % Mutate second thrust
22 tmpStart2 = mod(solution.start2 + (rand * cfg.delta), 100);
23 if tmpStart2 > cfg.maxValue

```

## Übung 1

```

24     solution.start2 = rand * cfg.delta;
25 else
26     solution.start2 = tmpStart2;
27 end
28 tmpEnd2 = mod(solution.end2 + (rand * cfg.delta), 100);
29 if tmpEnd2 > cfg.maxValue
30     solution.end2 = rand * cfg.delta;
31 else
32     solution.end2 = tmpEnd2;
33 end
34 while solution.end2 <= solution.start2
35     solution.end2 = solution.end2 + (rand * cfg.delta);
36 end
37
38 % Reset members
39 solution.quality = [];
40 solution.heightProgress = [];
41 solution.velocityProgress = [];
42
43 end

```

Listing 6: Funktion, welche die Simulation für einen Lösungskandidaten durchführt

```

1 function solution = evaluate(solution, cfg)
2 % Evaluates the simulation for the given candidate
3
4 % Prepare model parameters
5 modelParams = [1, solution.start1, solution.end1, solution.start2, solution.end2];
6 % run simulation
7 [T,X] = sim(cfg.simFile,100, cfg.simParams,modelParams);
8
9 % get height
10 h = X.signals(1).values;
11 % get min height
12 minH = min(h);
13
14 % we haven't fully landed
15 if minH > 0
16     solution.quality = minH/10;
17 % We have landed already
18 else
19     impactIdx = find(h < 0);
20     solution.quality = -X.signals(2).values(impactIdx(1));
21 end
22
23 end

```

## **2.2 Auswertung der Testfälle**

### **2.2.1 Test 1**

### **2.2.2 Test 2**

### **2.2.3 Test 3**

### **2.2.4 Test 4**

### **2.2.5 Test 5**

### **2.2.6 Gegenüberstellung der Tests**

## **2.3 Optimierungen des Evolutionsalgorithmus**

**2.3.1 Was muss optimiert werden ?**

**2.3.2 Was ist eine geeignete Fitnessfunktion ?**

**2.3.3 Welche Parameter können modifiziert werden ?**

**2.3.4 Warum ein Evolutionsalgorithmus ?**