
Versuch I, Modellbildung und Simulation eines Doppelpendel-Systems

Andreas Jentsch, Ali Kerem Sacakli

Praktikumsbericht – Praktikum Matlab/Simulink II



TECHNISCHE
UNIVERSITÄT
DARMSTADT

REGELUNGSTECHNIK
UND MECHATRONIK

rtm

Vorbereitung

(Text, Bild des Modells)

Code:

Listing 1: Code des Doppelpendel-Modells

```
1 clear all
  clc
  %% Variablen
  % Parameter
  syms m1 l1 J1 m2 l2 J2 Rp1 Rp2 g t positive;

6
  % Koordinaten
  syms phi1 dphi1 ddphi1 phi2 dphi2 ddphi2 real;

  % Momente
11 syms M real;

  %% Zu ersetzende Größen

16 % Reibungen
  MR1 = Rp1 * dphi1;
  MR2 = Rp2 * (dphi2 - dphi1);

  % Trägheitsmomente
21 J1 = 1 / 12 * m1 * l1^2;
  J2 = 1 / 12 * m2 * l2^2;

  % Pendelkoordinaten
  x1 = l1 / 2 * sin(phi1);
26 dx1 = l1 / 2 * dphi1 * cos(phi1);

  y1 = -l1 / 2 * cos(phi1);
  dy1 = l1 / 2 * dphi1 * sin(phi1);

31 x2 = 2*x1 + l2 / 2 * sin(phi2);
  dx2 = 2*dx1 + l2 / 2 * dphi2 * cos(phi2);

  y2 = 2*y1 - l2 / 2 * cos(phi2);
```

```

dy2 = 2*dy1 + l2 / 2 * dphi2 * sin(phi2);

36
%% Mechanik

T1 = 1/2 * m1 * (dx1^2 + dy1^2) + 1/2 * J1 * dphi1^2;
T2 = 1/2 * m2 * (dx2^2 + dy2^2) + 1/2 * J2 * dphi2^2;

41
T = T1 + T2;

U = g * (m1 * y1 + m2 * y2);

46 %Generalisierte Kräfte

Qphi1 = M - MR1;
Qphi2 = -MR2;

51 %Lagrangegleichung

L = T-U;

%% Herleitung der Ableitung nach generalisierter Koordinate

56 % dL/dphi1
L_phi1 = jacobian(L, phi1);

% dL/ddphi1
61 L_dphi1 = jacobian(L, dphi1);

% dL/dhphi2
L_phi2 = jacobian(L, phi2);

66 % dL/ddphi2
L_dphi2 = jacobian(L, dphi2);

% d(L_dphi1)/dt und d(L_dphi2)/dt

71 % Variablen ohne t durch Variablen mit t ersetzen
L_dphi1_t = subs(L_dphi1, {phi1, dphi1, phi2, dphi2}, ...
    {'phi1(t)', 'dphi1(t)', 'phi2(t)', 'dphi2(t)'});

```

```

L_dphi2_t = subs(L_dphi2,{phi1,dphi1,phi2,dphi2},...
76     {'phi1(t)', 'dphi1(t)', 'phi2(t)', 'dphi2(t)'});

%Berechnung der Zeitableitung
dL_dphi1_t = diff(L_dphi1_t, t);
dL_dphi2_t = diff(L_dphi2_t, t);

81
%Variablen mit t
Var_t = {'phi1(t)', 'dphi1(t)', 'diff(phi1(t),t)', 'diff(dphi1(t),t)'>
    '<',...
        'phi2(t)', 'dphi2(t)', 'diff(phi2(t),t)', 'diff(dphi2(t),t)'>
    '<'};

86 %Variablen ohne t
Var_ot = {phi1, dphi1, dphi1, ddphi1, phi2, dphi2, dphi2, ddphi2};

dL_dphi1_t = subs(dL_dphi1_t, Var_t, Var_ot);
dL_dphi2_t = subs(dL_dphi2_t, Var_t, Var_ot);

91

%% Berechnung der LAGRANGEschen Gleichungen

Sol = solve([dL_dphi1_t - L_phi1 == Qphi1, dL_dphi2_t - L_phi2 == >
    '<Qphi2],...
96     [ddphi1, ddphi2]);
Sol.ddphi1=simplify(Sol.ddphi1);
Sol.ddphi2=simplify(Sol.ddphi2);

```

$$\frac{l1 * l2 * m2^2 * \sin(2 * u(3) - u(4)) + 6 * Rp1 * u(2) * l2 * m2 * \cos(u(3) - u(4)) + 2 * u(2)^2 * l1^2 * l2 * m1 * m2 * \sin(u(3) - u(4)) - (g * l1 * l2 * m1 * m2 * \sin(u(4))) / 2 + (3 * g * l1 * l2 * m1 * m2 * \sin(2 * u(3) - u(4))) / 2)}{(l1 * l2^2 * m2 * (8 * m1 + 15 * m2 - 9 * m2 * \cos(2 * u(3) - 2 * u(4)))}$$

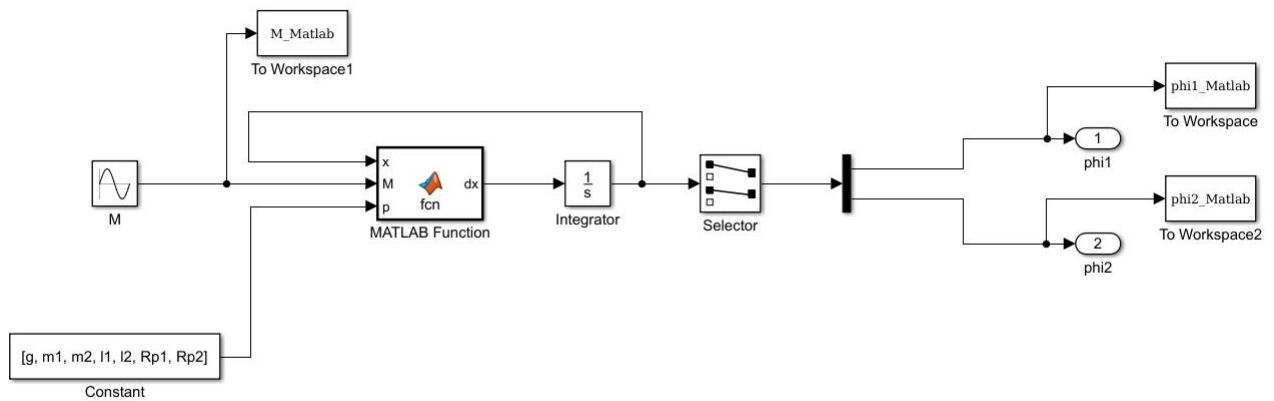


Figure 0.2: Implementierung mit einem Matlab-Function-Block

Matlab-Function-Block

Der Code des Matlab-Fcn-Blocks ist wie folgt:

Listing 2: Code für das Matlab-Function-Block

```
function dx = fcn(x,M,p)
2  %#codegen

% Zustände definieren
phi1 = x(1);
dphi1 = x(2);
7 phi2 = x(3);
dphi2 = x(4);

% Parameter definieren
g = p(1);
12 m1 = p(2);
m2 = p(3);
l1 = p(4);
l2 = p(5);
Rp1 = p(6);
17 Rp2 = p(7);

% Bewegungsgleichungen des Modells
ddphi1 = -(6*(4*Rp1*dphi1*l2 - 4*M*l2 + 6*Rp2*dphi1*l1*cos(phi1 - →
    ← phi2) - 6*Rp2*dphi2*l1*cos(phi1 - phi2) + (3*g*l1*l2*m2*sin(phi1 →
    ← - 2*phi2))/2 + (3*dphi1^2*l1^2*l2*m2*sin(2*phi1 - 2*phi2))/2 + →
```

```

    2*dphi2^2*l1*l2^2*m2*sin(phi1 - phi2) + 2*g*l1*l2*m1*sin(phi1) +
    (5*g*l1*l2*m2*sin(phi1))/2))/(l1^2*l2*(8*m1 + 15*m2 - 9*m2*cos
    (2*phi1 - 2*phi2)));

```

```

22 ddphi2 = (6*(4*Rp2*dphi1*l1*m1 - 6*M*l2*m2*cos(phi1 - phi2) + 12*Rp2*
    dphi1*l1*m2 - 4*Rp2*dphi2*l1*m1 - 12*Rp2*dphi2*l1*m2 + 6*dphi1*
    l1^2*l2^2*m2^2*sin(phi1 - phi2) - 3*g*l1*l2*m2^2*sin(phi2) +
    (3*dphi2^2*l1*l2^2*m2^2*sin(2*phi1 - 2*phi2))/2 + 3*g*l1*l2*m2*
    l1^2*sin(2*phi1 - phi2) + 6*Rp1*dphi1*l2*m2*cos(phi1 - phi2) + 2*
    dphi1^2*l1^2*l2*m1*m2*sin(phi1 - phi2) - (g*l1*l2*m1*m2*sin(phi2
    ))/2 + (3*g*l1*l2*m1*m2*sin(2*phi1 - phi2))/2))/(l1*l2^2*m2*(8*
    m1 + 15*m2 - 9*m2*cos(2*phi1 - 2*phi2)));

```

```

% Zustandsraummodell

```

```

dx = [dphi1; ddphi1; dphi2; ddphi2];

```

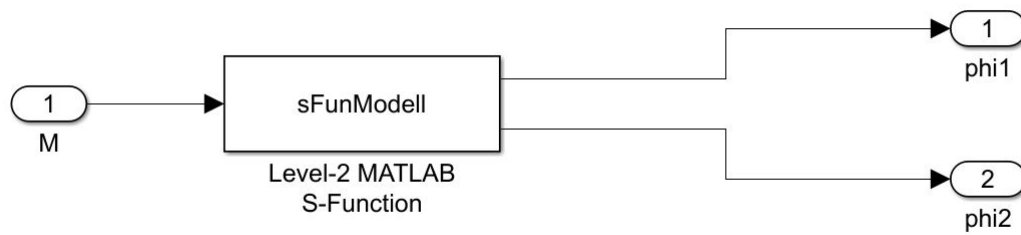


Figure 0.3: Implementierung mit einem Matlab-Function-Block

Level 2 M-File-S-Function

Die Implementierung des Codes für die S-Function ist wie folgt:

Listing 3: Code für das Matlab-Function-Block

```

function sFunModell(block)

    setup(block);

5  end

% →
% ← ***** →
% ←
% Initialisierung
10 % →
% ← ***** →
% ←

function setup(block)

    % Anzahl der Ein-/Ausgänge
    block.NumInputPorts = 1;
    15 block.NumOutputPorts = 2;

    % Eigenschaften des Eingangs
    block.InputPort(1).Dimensions = 1;
    block.InputPort(1).DatatypeID = 0; % double
    20 block.InputPort(1).Complexity = 'Real';
    block.InputPort(1).DirectFeedthrough = false;
    block.InputPort(1).SamplingMode = 'Sample';
  
```

```

25      % Eigenschaften des 1. Ausgangs
      block.OutputPort(1).Dimensions = 1;
      block.OutputPort(1).DatatypeID = 0; % double
      block.OutputPort(1).Complexity = 'Real';
      block.OutputPort(1).SamplingMode = 'Sample';

30      % Eigenschaften des 2. Ausgangs
      block.OutputPort(2).Dimensions = 1;
      block.OutputPort(2).DatatypeID = 0; % double
      block.OutputPort(2).Complexity = 'Real';
      block.OutputPort(2).SamplingMode = 'Sample';

35      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

      % Anzahl der Zustände
40      block.NumContStates = 4;

      % Anzahl der Parameter
      block.NumDialogPrms = 7;

45      % Abtastzeit definieren -> zeitkontinuierlich
      block.SampleTimes = [0 0];

50      % weitere Methoden registrieren
      block.RegBlockMethod('InitializeConditions', →
          @InitializeConditions);
      block.RegBlockMethod('Outputs', @Outputs);
      block.RegBlockMethod('Derivatives', @Derivatives);
55      block.RegBlockMethod('Terminate', @Terminate);

end

```

```

60 % →
    ← ***** →
    ←
    % Anfangsbedingungen setzen
    % →
    ← ***** →
    ←
    function InitializeConditions(block)

65     block.ContStates.Data = [0, 0, 0, 0];

    end

70 % →
    ← ***** →
    ←
    % Ausgänge berechnen
    % →
    ← ***** →
    ←
    function Outputs(block)

75     % Zustände auslesen
    x      = block.ContStates.Data;

    block.OutputPort(1).Data = x(1);
    block.OutputPort(2).Data = x(3);

80
    end

    % →
    ← ***** →
    ←
85 % Ableitungen berechnen
    % →
    ← ***** →
    ←
    function Derivatives(block)

```

```

% Parameter auslesen
90  g = block.DialogPrm(1).Data;
    m1 = block.DialogPrm(2).Data;
    m2 = block.DialogPrm(3).Data;
    l1 = block.DialogPrm(4).Data;
    l2 = block.DialogPrm(5).Data;
95  Rp1 = block.DialogPrm(6).Data;
    Rp2 = block.DialogPrm(7).Data;

% Zustände auslesen
    x      = block.ContStates.Data;
100  phi1   = x(1);
    dphi1  = x(2);
    phi2   = x(3);
    dphi2  = x(4);

105  % Eingang auslesen
    M = block.InputPort(1).Data(1);

110  % Ableitungen berechnen
    ddphi1 = -(6*(4*Rp1*dphi1*l2 - 4*M*l2 + 6*Rp2*dphi1*l1*cos(phi1 -
        phi2) - 6*Rp2*dphi2*l1*cos(phi1 - phi2) + (3*g*l1*l2*m2*
        sin(phi1 - 2*phi2))/2 + (3*dphi1^2*l1^2*l2*m2*sin(2*phi1 -
        2*phi2))/2 + 2*dphi2^2*l1*l2^2*m2*sin(phi1 - phi2) + 2*g*l1*
        l2*m1*sin(phi1) + (5*g*l1*l2*m2*sin(phi1))/2))/(l1^2*l2*(8*
        m1 + 15*m2 - 9*m2*cos(2*phi1 - 2*phi2)));

115  ddphi2 = (6*(4*Rp2*dphi1*l1*m1 - 6*M*l2*m2*cos(phi1 - phi2) +
        12*Rp2*dphi1*l1*m2 - 4*Rp2*dphi2*l1*m1 - 12*Rp2*dphi2*l1*m2 +
        6*dphi1^2*l1^2*l2*m2^2*sin(phi1 - phi2) - 3*g*l1*l2*m2^2*
        sin(phi2) + (3*dphi2^2*l1*l2^2*m2^2*sin(2*phi1 - 2*phi2))/2 +
        3*g*l1*l2*m2^2*sin(2*phi1 - phi2) + 6*Rp1*dphi1*l2*m2*cos(
        phi1 - phi2) + 2*dphi1^2*l1^2*l2*m1*m2*sin(phi1 - phi2) - (g*
        l1*l2*m1*m2*sin(phi2))/2 + (3*g*l1*l2*m1*m2*sin(2*phi1 -

```

```
←phi2))/2)))/(l1*l2^2*m2*(8*m1 + 15*m2 - 9*m2*cos(2*phi1 - 2*→  
←phi2)));
```

```
% Ableitungen zuweisen
```

```
120 block.Derivatives.Data = [dphi1; ddphi1; dphi2; ddphi2];
```

```
end
```

```
125 % →
```

```
←*****→
```

```
←
```

```
% Aufräumen (wenn nötig)
```

```
% →
```

```
←*****→
```

```
←
```

```
function Terminate(block)
```

```
end
```

Vergleich der Modellierungsmöglichkeiten

Die in den vorangegangenen Abschnitten vorgestellten Modelle werden mit einem Anregungssignal

$$M(t) = 0.1 \sin(t)$$

und der Anfangsbedingung ($\phi_1(t=0) = \phi_2(t=0) = 0 \text{ rad}$) simuliert. Dabei werden die Winkel ϕ_1 und ϕ_2 in Abhängigkeit vom Anregungssignal zum einen für eine Simulationsdauer von 10s und 2s dargestellt.

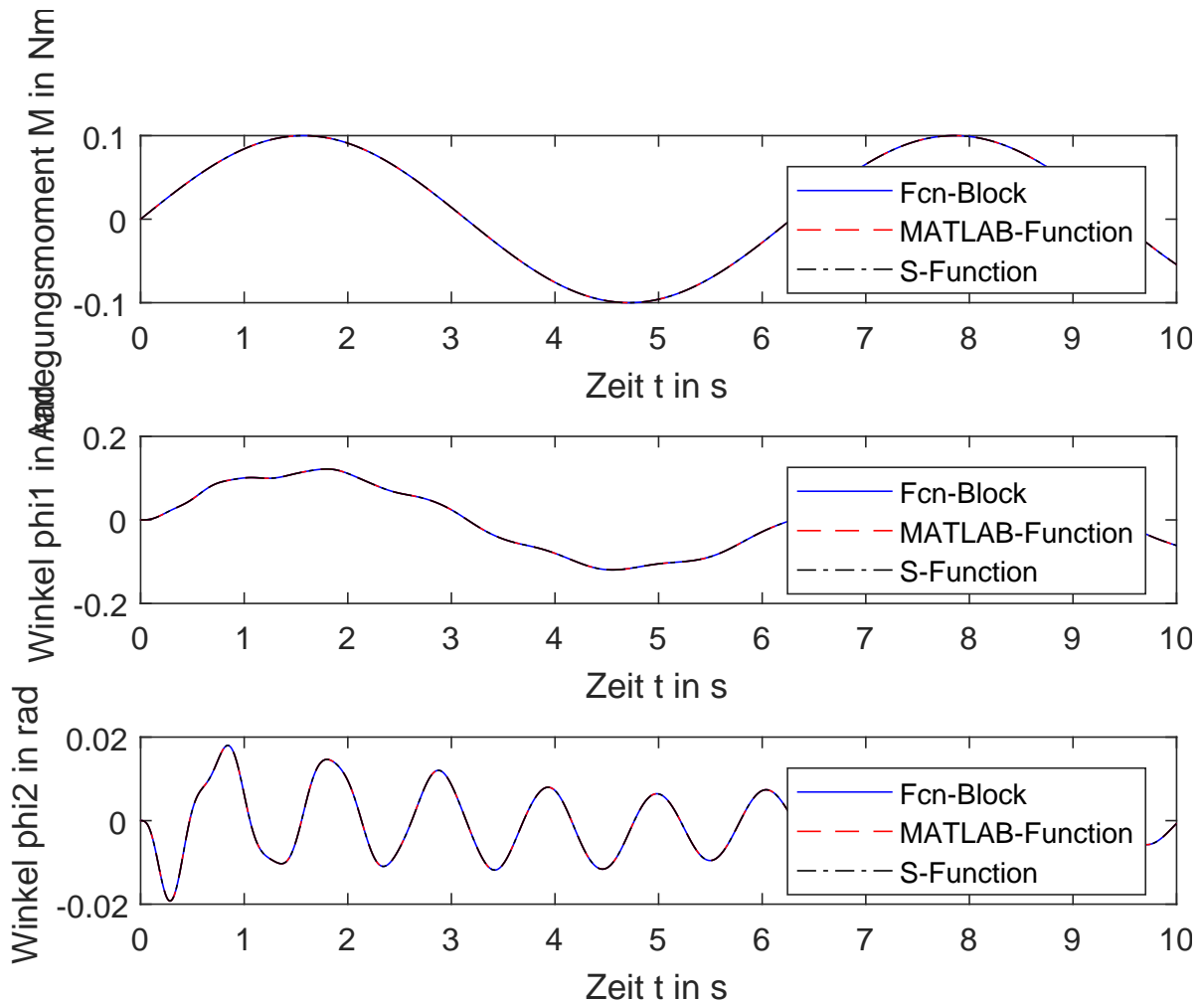


Figure 0.4: Vergleich der Modellierungsmöglichkeiten bei einer Simulationsdauer von 10s

Alle drei behandelten Modellierungsmöglichkeiten sind für die nichtlineare Modellierung geeignet. Bei gleicher Simulationseinstellung sind daher identische Ergebnisse zu erwarten. Wie die Abbildungen zeigen erfüllen die Ergebnisse diese Erwartung.

Der Verlauf der Winkel in Abhängigkeit des Anregungssignals ist plausibel. Für den Winkel ϕ_1 ist ein ähnlicher Verlauf wie das Anregungsmoment, nur mit einer kleiner Verzögerung durch die Trägheitsmomente, zu erwarten, da das Anregungsmoment direkt auf die Drehachse in ϕ_1 wirkt und somit einen ähnlichen Verlauf erzwingt. Der Winkel ϕ_2 ist dagegen frei gelagert und ist nur durch ein Reibmoment

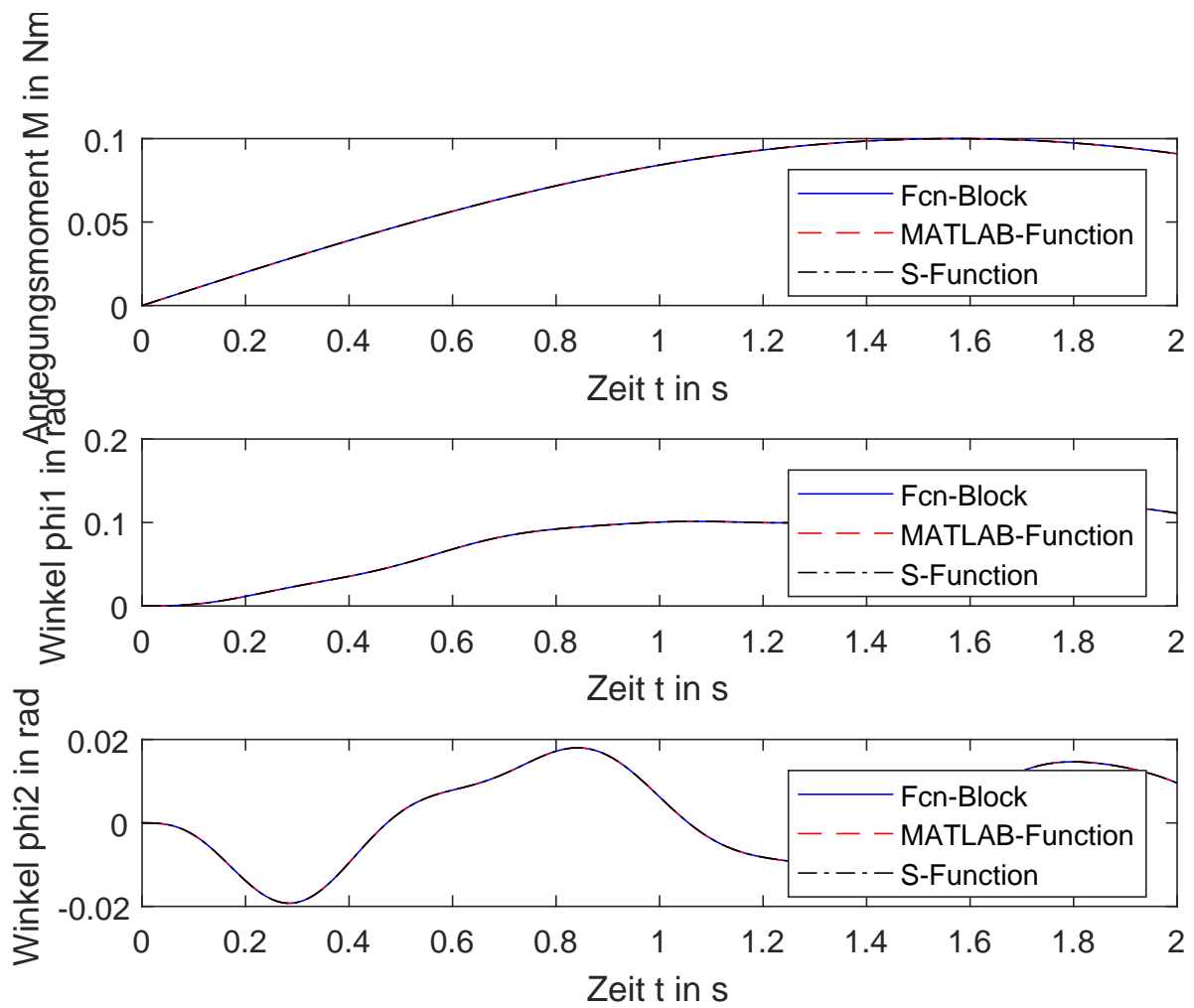


Figure 0.5: Vergleich der Modellierungsmöglichkeiten bei einer Simulationsdauer von 2s