K Method Tutorial for Sound Synthesis

Martin Maunsbach

Contents

1 State Space Modelling

Discretization is a necessary step when working with continuous-time physical equations that has to be used in discrete time systems, as is the case of any sound synthesis. The state space system is useful for modelling higher order linear and non-linear systems and makes discretizing the system easier. A continuous multiple input multiple output (MIMO) system can be expressed in state-variable form as

$$\dot{w} = Aw + Bu \tag{1.1}$$

where w are the states transformed by the A matrix and u is the input transformed by B. The vector $\dot{w}$ is the derivative of the states.

Several methods can be used to discretize the system. In this section we will introduce the Euler backwards, the Euler forwards and the trapezoidal rule.

1.1 Euler Differencing

The first method is discretizing by the Euler backward differencing:

$$\begin{aligned} s &= \alpha(1 - z^{-1}) \\ \alpha &= Sr \end{aligned} \tag{1.2}$$

where s is the Laplace complex and α is the sampling rate Sr. The variable z^{-1} will for digital signals account for a delay by one sample.

To discretize by Euler backward differencing to a linear state-space system, $\dot{w}$ becomes w multiplied by s on the left-hand side and the equation is therefore expressed in the z-transform. In the z-transform we express the input u and output w as U and W. As described in Definition 1.1, the discrete-time equation takes the form

$$\begin{aligned} w[n] &= H(\alpha w[n-1] + Bu[n]) \\ H &= (\alpha I - A)^{-1} \end{aligned} \tag{1.3}$$

Definition 1.1: Backwards Euler on a State Space System

Changes are underlined for clarity.

State Space as a function of the z-complex:

$$W[z]s = AW[z] + BU[z]$$

s is replaced by $\alpha(1 - z^{-1})$:

$$W[z]\underline{\alpha(1 - z^{-1})} = AW[z] + BU[z]$$

$\alpha W[z]$ is multiplied into the parenthesis:

$$\underline{\alpha W[z] - \alpha z^{-1} W[z]} = AW[z] + BU[z]$$

Restructuring of $AW[z]$ and $z^{-1}W[z]$:

$$\alpha W[z]\underline{-AW[z]} = \underline{\alpha z^{-1} W[z]} + BU[z]$$

$W[z]$ is moved outside a the parenthesis. An identity matrix I is multiplied onto α to allow it to be subtracted by matrix A:

$$\underline{(\alpha I - A)W[z]} = \alpha z^{-1} W[z] + BU[z]$$

The matrix is moved to the other side by multiplying both sides with its inverse. This can only be done if the inverse matrix exists (see Helpbox 1.1):

$$W[z] = \underline{(\alpha I - A)^{-1}}(\alpha z^{-1} W[z] + BU[z])$$

$(\alpha I - A)^{-1}$ is rewritten as H:

$$W[z] = \underline{H}(\alpha z^{-1} W[z] + BU[z])$$

The equation is rewritten in discrete-time variable n where $[z]$ becomes the current sample $[n]$ and $z^{-1}[z]$ becomes a delay of one sample $[n-1]$:

$$\underline{w[n] = H(\alpha w[n-1] + Bu[n])}$$

The forward Euler is very similar:

Helpbox 1.1: The H matrix

The determinant $(ad - bc)$ of 2×2 matrix is not allowed to be zero when calculating the inverse matrix:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} \tag{1.4}$$

$$\begin{aligned} s &= \alpha(1 + z^{-1}) \\ \alpha &= Sr \end{aligned} \tag{1.5}$$

with the only difference being the positive sign. The discrete-time equation then takes the form

$$\begin{aligned} w[n] &= H(Bu[n] - \alpha w[n-1]) \\ H &= (\alpha I - A)^{-1} \end{aligned} \tag{1.6}$$

1.2 Bilinear transform

A more accurate discretization is the bilinear transformation with

$$\begin{aligned} s &= \alpha \frac{1 - z^{-1}}{1 + z^{-1}} \\ \alpha &= 2 \cdot Sr \end{aligned} \tag{1.7}$$

where Sr is the sampling rate. Two times the sampling rate is used for the bilinear transform compared to one times the sampling rate for the backwards and forwards Euler. As derived in Definition 1.2, the linear state-space system the takes on a different form

$$\begin{aligned} w[n] &= H(\alpha I + A)w[n-1] + HB(u[n] + u[n-1]) \\ H &= [\alpha I - A]^{-1} \end{aligned} \tag{1.8}$$

Definition 1.2: Bilinear Transform on a State Space System

Changes are underlined for clarity.

State Space as a function of the z-complex:

$$W[z]s = AW[z] + BU[z]$$

s is replaced by the bilinear transform:

$$W[z]\underline{\alpha \frac{1 - z^{-1}}{1 + z^{-1}}} = AW[z] + BU[z]$$

Both sides are multiplied by $1 + z^{-1}$ eliminating it on the left hand side. α is multiplied onto the remaining nominator of the fraction:

$$W[z]\underline{(\alpha - \alpha z^{-1})} = AW[z]\underline{(1 + z^{-1})} + BU[z]\underline{(1 + z^{-1})}$$

The parenthesis with α is expanded:

$$\underline{\alpha W[z] - \alpha z^{-1} W[z]} = AW[z](1 + z^{-1}) + BU[z](1 + z^{-1})$$

$AW[z]$ and $U[z]$ is multiplied into parenthesis:

$$\alpha W[z] - \alpha z^{-1} W[z] = \underline{AW[z] + Az^{-1}W[z]} + B\underline{(U[z] + z^{-1}U[z])}$$

Instances of $W[z]$ without z^{-1} is isolated on the left side of the equation and instances with z^{-1} is moved to the right side:

$$\alpha W[z]\underline{- AW[z]} = \underline{\alpha z^{-1} W[z]} + Az^{-1}W[z] + B(U[z] + z^{-1}U[z])$$

Variables are combined to $(\alpha - A)$ and $(\alpha + A)$ to have single instances of $W[z]$ and $z^{-1}W[z]$:

$$\underline{(\alpha I - A)W[z]} = \underline{(\alpha I + A)z^{-1}W[z]} + B(U[z] + z^{-1}U[z])$$

The matrix is moved to the other side by multiplying both sides with its inverse. This can only be done if the inverse matrix exists (see Helpbox 1.1):

$$W[z] = \underline{(\alpha I - A)^{-1}}(\alpha I + A)z^{-1}W[z] + \underline{(\alpha I - A)^{-1}}B(U[z] + z^{-1}U[z])$$

The inverse matrix is rewritten as H:

$$W[z] = \underline{H}(\alpha I + A)z^{-1}W[z] + \underline{H}B(U[z] + z^{-1}U[z])$$

The equation is rewritten in discrete-time variable n where $[z]$ becomes the current sample $[n]$ and $z^{-1}[z]$ becomes a delay of one sample $[n-1]$:

$$\underline{w[n] = H(\alpha I + A)w[n-1] + HB(u[n] + u[n-1])}$$

2 K Method - Elimination of Delay-Free Loops

While the previous equations are valid with inputs and outputs that are not mutually dependent, more steps must be taken to accommodate for mutual dependencies.

An example of such a mutual dependency where the output depends on the input and the input depends on the output can be a contact interaction. Say we want to find the displacement of an object in contact with a force. The displacement (output) might require thevalue of the force (input) applied to it, but to calculate the force the value of displacement might be required, leading to the mutual dependency. They both require non-delayed version of the other value. The easiest trick to use is to use the delayed input instead of the current, but this leads to inaccuracies.

The K-Method, described by Borin and De Poli [2], adds an additional matrix and state vector y for the mutual dependency input to equation 1.1

$$\dot{w} = Aw + Bu + Cy \tag{2.1}$$

with

$$y = f(Dw + Eu + Fy) \tag{2.2}$$

where D, E and F are transformations of the state vectors for the function used to find y.

2.1 Euler Differencing

Discretizing equation 2.1 is similar to equation 1.1 with Euler backward differencing

$$\begin{aligned} s &= \alpha(1 - z^{-1}) \\ \alpha &= Sr \end{aligned} \tag{2.3}$$

As described in Definition 2.1, the discrete-time equation takes the form

$$w[n] = H(\alpha w[n-1] + Bu[n] + Cy[n]) \tag{2.4}$$

Definition 2.1: Backwards Euler on a K method State Space

Changes are underlined for clarity.

K method as a function of the z-complex:

$$W[z]s = AW[z] + BU[z] + CY[Z]$$

s is replaced by $\alpha(1 - z^{-1})$:

$$W[z]\underline{\alpha(1 - z^{-1})} = AW[z] + BU[z] + CY[Z]$$

$\alpha W[z]$ is multiplied into the parenthesis:

$$\underline{\alpha W[z] - \alpha z^{-1} W[z]} = AW[z] + BU[z] + CY[Z]$$

Restructuring of $AW[z]$ and $z^{-1}W[z]$:

$$\alpha W[z] \underline{- AW[z]} = \underline{\alpha z^{-1} W[z]} + BU[z] + CY[Z]$$

$W[z]$ is moved outside a the parenthesis. An identity matrix I is multiplied onto α to allow it to be subtracted by matrix A:

$$\underline{(\alpha I - A)W[z]} = \alpha z^{-1} W[z] + BU[z] + CY[Z]$$

The matrix is moved to the other side by multiplying both sides with its inverse. This can only be done if the inverse matrix exists (see Helpbox 1.1):

$$W[z] = \underline{(\alpha I - A)^{-1}}(\alpha z^{-1} W[z] + BU[z + CY[Z]])$$

$(\alpha I - A)^{-1}$ is rewritten as H:

$$W[z] = \underline{H}(\alpha z^{-1} W[z] + BU[z] + CY[Z])$$

The equation is rewritten in discrete-time variable n where $[z]$ becomes the current sample $[n]$ and $z^{-1}[z]$ becomes a delay of one sample $[n - 1]$:

$$\underline{w[n] = H(\alpha w[n - 1] + Bu[n] + Cy[n])}$$

The remainder of the system from Equation 2.2 must also be taken into account. Now that $w[n]$ is found, the inside of the function f can be decomposed as described in Definitions 2.2 as all the equations become

$$\begin{aligned} w[n] &= H(\alpha w[n - 1] + Bu[n] + Cy[n]) \\ y &= f(p(n) + Ky) \\ p(n) &= (DHB + E)u[n] + \alpha DHw[n - 1] \\ K &= (DHC + F) \end{aligned} \tag{2.5}$$

Definition 2.2: System decomposition of Backwards Euler on the K method

Changes are underlined for clarity.

The inside of the function in the mutually dependent component y:

$$Dw[n] + Eu[n] + Fy[n]$$

The discretized $w[n]$ found in Definition 2.1 is inserted:

$$\underline{D(HCy[n] + H(\alpha w[n - 1] + Bu[n]))} + Eu + Fy$$

The D matrix is multiplied into the parenthesis:

$$\underline{DHCy[n]} + \alpha DHw[n-1] + \underline{DHBu[n]} + Eu + Fy$$

The equation is rewritten to have a single instance of input $u[n]$, mutual dependent input $y[n]$ and delayed output $w[n-1]$:

$$\underline{(DHB+E)u[n]} + \alpha DHw[n-1] + \underline{(DHC+F)y[n]}$$

This can then be split in to $p(n)$ as a linear combination of current and previous known samples:

$$p(n) = (DHB+E)u[n] + \alpha DHw[n-1]$$

A weighting matrix for the mutual dependent value K is written as:

$$K = (DHC+F)$$

The combination of $p(n)$ and K is then used as the inside of the function:

$$y = f(p(n) + Ky)$$

Using forward Euler results in a slight difference due to the positive sign:

$$\begin{aligned} w[n] &= H(Bu[n] + Cy[n] - \alpha w[n-1]) \\ y &= f(p(n) + Ky) \\ p(n) &= (DHB+E)u[n] - \alpha DHw[n-1] \\ K &= (DHC+F) \end{aligned} \tag{2.6}$$

2.2 Bilinear Transform

As previously stated, the bilinear transform is given by

$$\begin{aligned} s &= \alpha \frac{1-z^{-1}}{1+z^{-1}} \\ \alpha &= 2 \cdot Sr \end{aligned} \tag{2.7}$$

Discretizing the K method using the bilinear transform is very similar to Definition 1.2 and can be seen in Definition 2.3 to take the form

$$w[n] = HC(y[n] + y[n-1]) + H(\alpha I + A)w[n-1] + HB(u[n] + u[n-1]) \tag{2.8}$$

Definition 2.3: Bilinear Transform using the K method

Changes are underlined for clarity.

State Space as a function of the z-complex:

$$W[z]s = AW[z] + BU[z] + CY[z]$$

s is replaced by the bilinear transform:

$$W[z]\underline{\alpha \frac{1-z^{-1}}{1+z^{-1}}} = AW[z] + BU[z] + CY[z]$$

Both sides are multiplied by $1 + z^{-1}$ eliminating it on the left hand side. α is multiplied onto the remaining nominator of the fraction:

$$W[z](\alpha - \alpha z^{-1}) = AW[z](1 + z^{-1}) + BU[z](1 + z^{-1}) + CY[z](1 + z^{-1})$$

The parenthesis with α is expanded:

$$\alpha W[z] - \alpha z^{-1} W[z] = AW[z](1 + z^{-1}) + BU[z](1 + z^{-1}) + CY[z](1 + z^{-1})$$

$AW[z]$, $U[z]$ and $Y[z]$ is multiplied into parenthesis:

$$\begin{aligned}\alpha W[z] - \alpha z^{-1} W[z] = AW[z] + Az^{-1}W[z] + B(U[z] + z^{-1}U[z]) \\ + C(Y[z] + z^{-1}Y[z])\end{aligned}$$

Instances of $W[z]$ without z^{-1} is isolated on the left side of the equation and instances with z^{-1} is moved to the right side:

$$\begin{aligned}\alpha W[z] - AW[z] = \alpha z^{-1} W[z] + Az^{-1}W[z] + B(U[z] + z^{-1}U[z]) \\ + C(Y[z] + z^{-1}Y[z])\end{aligned}$$

Variables are combined to $(\alpha - A)$ and $(\alpha + A)$ to have single instances of $W[z]$ and $z^{-1}W[z]$:

$$\begin{aligned}(\alpha I - A)W[z] = (\alpha I + A)z^{-1}W[z] + B(U[z] + z^{-1}U[z]) \\ + C(Y[z] + z^{-1}Y[z])\end{aligned}$$

The matrix is moved to the other side by multiplying both sides with its inverse. This can only be done if the inverse matrix exists (see Helpbox 1.1):

$$\begin{aligned}W[z] = (\alpha I - A)^{-1}((\alpha I + A)z^{-1}W[z] + B(U[z] + z^{-1}U[z]) \\ + C(Y[z] + z^{-1}Y[z]))\end{aligned}$$

The inverse matrix is rewritten as H:

$$\begin{aligned}W[z] = H(\alpha I + A)z^{-1}W[z] + HB(U[z] + z^{-1}U[z]) \\ + HC(Y[z] + z^{-1}Y[z])\end{aligned}$$

The equation is rewritten in discrete-time variable n where $[z]$ becomes the current sample $[n]$ and $z^{-1}[z]$ becomes a delay of one sample $[n-1]$:

$$\begin{aligned}w[n] = H(\alpha I + A)w[n-1] + HB(u[n] + u[n-1]) \\ + HC(y[n] + y[n-1])\end{aligned}$$

The extra delayed values that comes from the bilinear transform makes the system composition of the values inside function f in Equation 2.2 slightly longer as seen in Definition 2.4 and takes the form

$$\begin{aligned}w[n] = HC(y[n] + y[n-1]) + H(\alpha I + A)w[n-1] + HB(u[n] + u[n-1]) \\ y = f(p(n) + Ky) \\ p(n) = DH(\alpha I + A)w[n-1] + (DHB + E)u + DHBu[n-1] + DHCy[n-1] \\ K = (DHC + F)\end{aligned} \tag{2.9}$$

Definition 2.4: System decomposition of Bilinear Transform on the K method

Changes are underlined for clarity.

The inside of the function in the mutually dependent component y:

$$Dw[n] + Eu[n] + Fy[n]$$

The discretized $w[n]$ found in Definition 2.3 is inserted:

$$D(\underline{H(Cy[n] + y[n-1]) + H((\alpha I + A)w[n-1] + B(u[n] + u[n-1]))}) \\ + Eu[n] + Fy[n]$$

The D matrix is multiplied into the parenthesis:

$$\underline{DHCy[n] + DHCy[n-1] + DH(\alpha I + A)w[n-1] + DHBu[n] + DHBu[n-1]} \\ + Eu[n] + Fy[n]$$

The equation is rewritten to have a single instance of input $u[n]$, delayed input $u[n-1]$, mutual dependent input $y[n]$, its delayed version $y[n-1]$ and delayed output $w[n-1]$:

$$DH(\alpha I + A)w[n-1] + \underline{(DHB + E)u[n]} + DHBu[n-1] + DHCy[n-1] \\ \underline{+(DHC + F)y[n])}$$

This can then be split in to $p(n)$ as a linear combination of current and previous known samples:

$$p(n) = DH(\alpha I + A)w[n-1] + (DHB + E)u[n] + DHBu[n-1] + DHCy[n-1]$$

A weighting matrix for the mutual dependent value K is written as:

$$K = (DHC + F)$$

The combination of $p(n)$ and K is then used as the inside of the function:

$$y = f(p(n) + Ky)$$

3 Examples: Spring-Mass-Damper

To explain what this means, lets look at the simple mass-spring-damper model using the equation

$$m\ddot{x} + b\dot{x} + kx = -f \tag{3.1}$$

A graphical representation of the system in Fig. 1 explains the equation. The mass is the mass of the object m, the spring has the spring constant k and the damping is denoted by b. All these values are usually constants. The system is impacted by a force F that moves the mass to its displacement x. The displacement x describes the vibration of the system and therefore the sound that comes from it. Its derivative, $\dot{x}$, is the velocity of the mass while $\ddot{x}$ is the acceleration of the mass (see Helpbox 3.1).

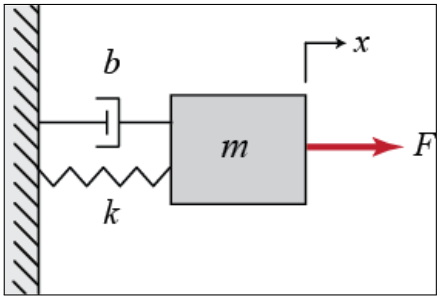

Figure 1: Mass-spring-damper model.

To get an understanding of when to use simple state-space, when to use K method and when to use K method

combined with an approximation algorithm (Newton-Rhapson), three examples are produced. The difference lies in how the force is calculated. The three examples are:

1. Initial force set to 2, the rest is 0 (simple state-space)
2. Using the hammer felt compression kx^{α} where *alpha* is 2 (K method)
3. Using the hammer felt compression kx^{α} where *alpha* is not 2 or 3 (K method and Newton-Rhapson)

3.1 Spring-Mass-Damper with Initial Force

Equation 3.1 must be modelled to fit the MIMO system in Equation 2.1. The first step is to find the state vector w. Since we are looking for the displacement x in Equation 3.1, it will be the first value of the state vector w. On the left side of the equal sign we need the derivative of the state vector, $\dot{w}$, meaning the first value there is $\dot{x}$. This leaves us with only one way to model the second element of the state vector, as we only have one more derivative of a variable, namely $\ddot{x}$:

$$\dot{w} = Aw + Bu \quad \Rightarrow \quad \begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = A \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + Bu \tag{3.2}$$

Beside the model parameters of k, m and b, the remaining changing input is the force f.

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = A \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + Bf \tag{3.3}$$

The transformation matrices A and B are found by isolating the variables in $\dot{w}$, $\dot{x}$ and $\ddot{x}$. For $\dot{x}$ it is quite simple, as it of course is equal to itself. The other variable $\ddot{x}$ is found by rewriting Equation 3.1:

$$\begin{aligned} m\ddot{x} + b\dot{x} + kx &= -f \\ m\ddot{x} &= -kx - b\dot{x} - f \\ \ddot{x} &= \frac{k}{m}x - \frac{b}{m}\dot{x} - \frac{1}{m}f \end{aligned} \tag{3.4}$$

Now that the equation for both variables of the derivative of the state vector is found, the values can be inserted as the transformation matrices A and B:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{b}{m} \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} f \tag{3.5}$$

Using Equation 1.8, the discretized form using the bilinear transform is:

$$w[n] = H(\alpha I + A)w[n-1] + HB(u[n] + u[n-1]) \tag{3.6}$$

where

$$\begin{aligned} A = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{b}{m} \end{bmatrix}; \quad B = \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix}; \quad w = \begin{bmatrix} x \\ \dot{x} \end{bmatrix}; \quad u = f; \\ H = [\alpha I - A]^{-1} = \frac{m}{\alpha^2 m + \alpha b + k} \begin{bmatrix} \alpha + \frac{b}{m} & 1 \\ -\frac{k}{m} & \alpha \end{bmatrix} \end{aligned} \tag{3.7}$$

Helpbox 3.1: Velocity v and Acceleration a

$v = \frac{dx}{dt} = \dot{x}$	$a = \frac{dv}{dt} = \frac{d^2x}{d^2t} = \ddot{x}$

An unoptimized version using MatLab can be seen below with an initial delayed force set to 2 and 0 after that. The output values for the displacement x can be found in the vector `w(1,:)`.

```
% spring constant K, damping b, mass m, samplerate sr, alpha a
freq = 440; k = (2*pi*freq)^2; b =10; m = 1; sr = 48000; a = 2*sr;
% transformation matrices A, B and H
A = [0 1; -k/m -b/m]; B = [0; 1/m]; H = inv(a*eye(2)-A);
% length 3 seconds, state values, prev values and force
N = 3*sr; w = zeros(2,N); wPrev = [0; 0]; f = 0; fPrev = 2;

for i = 1:N
    w(:,i) = H*(a*eye(2)+A)*wPrev + H*B*(f + fPrev);
    wPrev = w(:,i); fPrev = f;
end
```

3.2 Spring-Mass-Damper with K Method

Using a single value for the initial force and none for the remainder is of course not optimal. One way to calculate the contact force on the hammer felt compression x is using the equation [4]:

$$f(x[n]) = \begin{cases} kx[n]^{\alpha} & x > 0 \\ 0 & x \leq 0 \end{cases} \tag{3.8}$$

where α describes the local geometry and not a multiple of the sampling rate. In cases where α is 2 or 3, it is possibly to explicitly solve f when combined with the K method. This section will set the value to 2, while the next section will approximate it using Newton-Rhapson when it is not 2 or 3. The stiffness k is the stiffness of what can be described as a hammer. The displacement (or compression) is the same x as in Equation 1, which leads to the mutual dependency and the need for the K method. Modelling the system using the K method is very similar to the state-space, as the B matrix becomes the C matrix because the previously known input now is the mutually dependent input:

$$w[n] = H(\alpha I + A)w[n-1] + HB(u[n] + u[n-1]) \tag{3.9}$$

where

$$\begin{gathered} A = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{b}{m} \end{bmatrix}; \quad B = \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix}; \quad w = \begin{bmatrix} x \\ \dot{x} \end{bmatrix}; \quad u = f; \\ H = [\alpha I - A]^{-1} = \frac{m}{\alpha^2 m + \alpha b + k} \begin{bmatrix} \alpha + \frac{b}{m} & 1 \\ -\frac{k}{m} & \alpha \end{bmatrix} \end{gathered} \tag{3.10}$$

For the K method we also need the D, E and F matrices. These are found by modelling the force in Equation 3.8 using the same state vectors. As $\dot{x}$ is not in the contact force equation, we only have to focus on the state of x:

$$\begin{aligned} w_Y &= Dw + Eu + Fy \\ x &= \begin{bmatrix} 1 & 0 \end{bmatrix} x + [0]u + [0]f \end{aligned} \tag{3.11}$$

where the w_Y is a vector containing the variables of force, in this case the single variable x, and the transformation matrices E and F as well as the input u are here empty.

To excite the mass-spring-damper, it is excited by a hammer that is hitting it at an initial velocity. The hammer is given by:

$$f = -m_h a \tag{3.12}$$

The right side is negative because of Newton's third law stating that for every action, there is an equal and opposite reaction. m_h is the hammer mass as a lumped object. Similarly to the spring-mass-damper, the state-space system can be derived

$$\begin{aligned} \dot{x} &= \dot{x} \\ a = \ddot{x} &= -\frac{1}{m_h} f \end{aligned} \tag{3.13}$$

Using the same state vectors as the mass-spring-damper and D matrix, the matrices $A^{(h)}$ and $C^{(h)}$ (with (h) denoting the hammer) are then given by:

$$A^{(h)} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}; \quad C^{(h)} = \begin{bmatrix} 0 \\ -\frac{1}{m_h} \end{bmatrix}; \quad H^{(h)} = \begin{bmatrix} \frac{1}{\alpha} & \frac{1}{\alpha^2} \\ 0 & \frac{1}{\alpha} \end{bmatrix}; \quad D^h = \begin{bmatrix} 1 & 0 \end{bmatrix}. \tag{3.14}$$

The following process is then used to calculate an output $w[n]$ of the system using the mass-spring-damper and hammer:

Variable		Computation
$p[n]^r$	=	$D^r H^r(\alpha I + A^r)w[n-1]^r + D^r H^r C^r y[n-1]^r$[1]
$p[n]^h$	=	$D^h H^h(\alpha I + A^h)w[n-1]^h + D^h H^h C^h y[n-1]^h$
$p[n]$	=	$p[n]^h - p[n]^r$
$y[n]$	=	$k(p[n] + Ky[n])^2$
$w[n]^r$	=	$H^r(\alpha I + A^r)w[n-1]^r + H^r C^r(y[n]^r + y[n-1]^r)$;
$w[n]^h$	=	$H^h(\alpha I + A^h)w[n-1]^h + H^h C^h(y[n]^h + y[n-1]^h)$

Table 1: Real-time simulation steps with the resonator denoted with r and the hammer with h. K is computed offline as the hammer's K subtracted by the resonator's K. The output at each step is found as the first value in the vector $w[n]^r$. The exponent α is set to 2.

The equation for the contact force is now the equation

$$f = y[n] = k(p[n] + Ky[n])^2 \tag{3.15}$$

To find the value of $y[n]$, the above equation is expanded to a quadratic equation:

$$\begin{aligned} y[n] &= k(p[n] + Ky[n])^2 \\ 0 &= k(p[n]^2 + k^2 y[n]^2 + 2p[n]Ky[n]) - y[n] \\ 0 &= kp[n]^2 + kK^2 y[n]^2 + 2kp[n]Ky[n] - y[n] \\ 0 &= kK^2 y[n]^2 + (2kp[n]K - 1)y[n] + kp[n]^2 \end{aligned}$$

Of course, we can solve a quadratic equation, which yields 2 real results with a positive discriminant (see Helpbox 3.2). It is easy to see which of the two solutions are correct, as the value of the other is far beyond what is possible.

This example can be seen in the MatLab code below where the initial velocity of the hammer is set to 1.

[1] Simplified from $DH(\alpha I + A)w[n-1] + (DHB + E)u + DHBu[n-1] + DHCy[n-1]$ due to the empty E, B and u values.

Helpbox 3.2: Quadratic equation

$0 = ax^2 + bx + c$	$x = \frac{-b \pm \sqrt{\Delta}}{2a}; \quad \Delta = b^2 - 4ac$

```
% system values samplerate, alpha and length 3 seconds in samples
sr = 48000; alpha = 2*sr*eye(2); N = 1*sr;

% RESONATOR %
% spring constant K, damping b, mass m, samplerate sr, alpha a
freq = 440; k_r = (2*pi*freq)^2; b_r =10; m_r = 1;
% transformation matrices A, C, D and H
A_r = [0 1; -k_r/m_r -b_r/m_r]; C_r = [0; 1/m_r];
H_r = inv(alpha-A_r); D_r = [1 0]; K_r = D_r*H_r*C_r;

% HAMMER %
m_h = 1; A_h = [0 1;0 0]; C_h = [0; -1/m_h]; D_h = [1 0];
H_h = inv(alpha-A_h); K_h = D_h*H_h*C_h;

% hammer felt compression stiffness
k = 1.5*10^11;

% state values, prev values and force
w_r = zeros(2,N); w_h = zeros(2,N); wPrev_r = [0; 0];
wPrev_h = [0; 1]; f = zeros(N,1); fPrev = 0; delta = zeros(N,1);

% computed offline
K = K_h-K_r;

for i = 1:N
    % RESONATOR and HAMMER p
    p_r = D_r*H_r*(alpha+A_r)*wPrev_r+D_r*H_r*C_r*fPrev;
    p_h = D_h*H_h*(alpha+A_h)*wPrev_h+D_h*H_h*C_h*fPrev;
    % combined p
    p = p_h-p_r;

    % solve quadratic equation
    a = k*K^2; b_r = 2*k*p*K-1; c = k*p^2;
    delta(i) = b_r^2-4*a*c;
    if delta(i) > 0
        %f(i) = (-b + sqrt(delta(i)))/(2*a); % Not possible
        f(i) = (-b_r - sqrt(delta(i)))/(2*a);
    else
        f(i) = 0;
    end

    % no negative compression
    if p+K*f(i) < 0
        f(i) = 0;
    end

    % update states for both RESONATOR and HAMMER
    w_r(:,i) = H_r*(alpha*eye(2)+A_r)*wPrev_r + H_r*C_r*(f(i)+fPrev
       );
    w_h(:,i) = H_h*(alpha*eye(2)+A_h)*wPrev_h + H_h*C_h*(f(i)+fPrev
       );

    % update delayed values
    fPrev = f(i);wPrev_r = w_r(:,i); wPrev_h = w_h(:,i);
end
```

3.3 Spring-Mass-Damper with K Method and Newton-Rhapson

When α in the contact force equation

$$f(x[n]) = \begin{cases} kx[n]^{\alpha} & x > 0 \\ 0 & x \leq 0 \end{cases} \tag{3.16}$$

is not 2 or 3 (or 1), it is not possible to explicitly solve the non-linearity. To approximate a value, the Newton-Rhapson algorithm can be used. Definition 3.1 below explains Newton-Rhapson in connection with the K method.

Definition 3.1: Newton-Rhapson Algorithm with the K Method

The root-finding algorithm Newton-Rhapson can be used to approximate values from an equation with non-linearity. The approach here is derived from Rocchesso and Fontana [3]. The approximated value h can be found by repeating the process

$$h_{k+1} = h_k - \frac{g(h_k)}{g'(h_k)} \tag{3.17}$$

where k is a counter that increases until the function converges with a satisfying error margin. h_0 starts out as an estimated guess.
In connection with the K-Method, $g(h)$ is stated by searching for a local zero of the function

$$g(h_k) = f(p[n] + Kh_k) - h_k \tag{3.18}$$

A maximum error value is determined and the Newton-Rhapson algorithm searched until the error is below. The error is calculated by the absolute value of the difference between the current and previous h.

The approach is the same as the previous section, except Newton-Rhapson is used instead of solving a quadratic equation. Using Equation 3.16 in Equation 3.18 we have

$$g(h_k) = k\tilde{x}(h_k)^{\alpha} - h_k \tag{3.19}$$

where

$$\tilde{x}(h_k) = p[n] + Kh_k \tag{3.20}$$

Its derivative is found using the composite rule as $\tilde{x}(h_k)$ is also a function of h_k and that $-h_k = -1$:

$$\begin{aligned} g'(h) &= \frac{\partial g}{\partial h} = \frac{\partial f}{\partial \tilde{x}} \cdot \frac{\partial \tilde{x}}{\partial h} - 1 \\ g'(h) &= \alpha k \tilde{x}^{\alpha-1} K - 1 \end{aligned} \tag{3.21}$$

as the derivatives of the individual parts are found to be

$$\frac{\partial f}{\partial \tilde{x}} = \alpha k \tilde{x}^{\alpha-1}; \quad \frac{\partial \tilde{x}}{\partial h} = K. \tag{3.22}$$

The original $g(h_k)$ and its derivative is then used to search for the root as seen in Equation 3.17.

$$h_{k+1} = h_k - \frac{k\tilde{x}(h_k)^{\alpha} - h_k}{\alpha k \tilde{x}(h_k)^{\alpha-1} K - 1} \tag{3.23}$$

The steps taken in real-time are described in the table below, where the Newton-Rhapson part is repeated while the error is too high. For this example the error can be set to as low as 10e-13 and the algorithm will still converge within 4 iterations.

Variable		Computation
$p[n]^r$	=	$D^r H^r(\alpha I + A^r)w[n-1]^r + D^r H^r C^r y[n-1]^{r2}$
$p[n]^h$	=	$D^h H^h(\alpha I + A^h)w[n-1]^h + D^h H^h C^h y[n-1]^h$
$p[n]$	=	$p[n]^h - p[n]^r$
$\tilde{x}$	=	$p[n] + Kh_k$
$y[n]$	=	$k(p[n] + Kh_k)^\alpha$ Repeat while NR error is too high
$w[n]^r$	=	$H^r(\alpha I + A^r)w[n-1]^r + H^r C^r(y[n]^r + y[n-1]^r)$;
$w[n]^h$	=	$H^h(\alpha I + A^h)w[n-1]^h + H^h C^h(y[n]^h + y[n-1]^h)$

Table 2: Real-time simulation steps. K is computed offline as the hammer's K subtracted by the resonator's K. The output at each step is found as the first value in the vector $w[n]^r$. The exponent α makes it non-linear and the approximation algorithm is used when it is not a quadratic or cubic equation (ie. 2 or 3).

The MatLab implementation is listed below. The Newton-Rhapson part is encapsulated in the while loop, where a maximum iteration count also is used in case the algorithm does not converge. If α (alpha in the MatLab script) is set to 2, the same result as the previous section is seen.

```
% system values samplerate, alpha and length 3 seconds in samples
sr = 48000; alpha = 2*sr*eye(2); N = 1*sr;

% RESONATOR %
% spring constant K, damping b, mass m, samplerate sr, alpha a
freq = 440; k_r = (2*pi*freq)^2; b_r =10; m_r = 1;
% transformation matrices A, C, D and H
A_r = [0 1; -k_r/m_r -b_r/m_r]; C_r = [0; 1/m_r]; H_r = inv(alpha-
    A_r);D_r = [1 0];
K_r = D_r*H_r*C_r;

% HAMMER %
m_h = 1; k = 1.5*10^11;
%  transformation matrices
A_h = [0 1;0 0]; C_h = [0; -1/m_h]; D_h = [1 0];
H_h = inv(alpha-A_h); K_h = D_h*H_h*C_h;

% state values, prev values and force
w_r = zeros(2,N); w_h = zeros(2,N); wPrev_r = [0; 0]; wPrev_h = [0;
     1];
f = zeros(N,1); fPrev = 0;

K = K_h-K_r; h0 = 0; errMax = 10^(-13); maxIt = 0; a= 2.8;

for i = 1:N
    % RESONATOR and HAMMER p
    p_r = D_r*H_r*(alpha+A_r)*wPrev_r+D_r*H_r*C_r*fPrev;
    p_h = D_h*H_h*(alpha+A_h)*wPrev_h+D_h*H_h*C_h*fPrev;
    % combined p
    p = p_h-p_r;

    % hammer felt compression
    count = 1;
    err = 99;
    while err > errMax && count < 20
        xTi = p+K*h0;

        if (xTi < 0)   % Negative penetration => no contact force
```

```
            h0 = 0;
            err = 0;
        else
        h1 = h0 - (k*xTi^a-h0)/(a*k*xTi^(a-1)*K-1);

        count = count+1;
        err = abs(h1 - h0);
        h0 = h1;
        end
    end

    if (p+K*h0 < 0)    % Negative penetration => no contact force
        h0 = 0;
    end

    f(i) = h0;
    hs(i) = count; % save number of iterations in the NR loop

    % update states for both RESONATOR and HAMMER
    w_r(:,i) = H_r*(alpha*eye(2)+A_r)*wPrev_r + H_r*C_r*(f(i)+fPrev
        );
    w_h(:,i) = H_h*(alpha*eye(2)+A_h)*wPrev_h + H_h*C_h*(f(i)+fPrev
        );

    % update delayed values
    fPrev = f(i);wPrev_r = w_r(:,i); wPrev_h = w_h(:,i);
end
```

4 References

References

[1] Nikolaj Andersson & Stefania Serafin (2018): "The ultimate guide to mass-spring systems", Aalborg University Copenhagen.

[2] G. Borin, G. De Poli and D. Rocchesso, "Elimination of delay-free loops in discrete-time models of nonlinear acoustic systems," in IEEE Transactions on Speech and Audio Processing, vol. 8, no. 5, pp. 597-605, Sep 2000. doi: 10.1109/89.861380

[3] Davide Rocchesso and Federico Fontana (2003): "The Sounding Object" IST Project. ISBN 88-901126-0-3.

[4] Hall, Donald E. (1992): "Piano string excitation. VI: Nonlinear modeling", The Journal of the Acoustical Society of America, Volume 92, Issue 1, July 1992, pp.95-105.