12 Appendices

12.1 Normalizing - the important step towards preparation for programming

So far, the algorithms with their control variables and parameters are given in the originally derived physical form. An implementation or a programming in this form is mainly impossible. Before this practical implementation can start, all algorithms have to be normalized and scaled if necessary, e.g. for fixed point and partly also for floating point processors. The purpose of the *normalization* consists of transferring these variables and parameters into a unity-less form and thus to prepare them for programming. The *scaling* is primarily necessary to increase the numerical accuracy which is of great importance for the use of fixed point processors.

This important step towards preparation for programming is demonstrated on two examples.

a) Example 1:

The first equation of (3.55), which can be written in detail as follows, serves as the first example:

$$\begin{cases} i_{sd}(k+1) = \Phi_{11}\, i_{sd}(k) + \Phi_{12}\, i_{sq}(k) + \Phi_{13}\, \psi_{rd}^{/}(k) + h_{11}\, u_{sd}(k) \\ i_{sq}(k+1) = -\Phi_{12}\, i_{sd}(k) + \Phi_{11}\, i_{sq}(k) - \Phi_{14}\, \psi_{rd}^{/}(k) + h_{11}\, u_{sq}(k) \end{cases} \tag{12.1}$$

From (12.1) the following can be noticed:

- The variables like currents $i_{sd}, i_{sq}, \psi_{rd}^{/}$ and voltages u_{sd}, u_{sq} have to be normalized.
- From the parameters only Φ_{11}, Φ_{13} are already unity-less. All others have to be normalized.

For the normalization of the currents, the maximum inverter current $I_{\max}$ is often chosen. For the normalization of the voltage, the maximum value, which is $2U_{DC}$[1]/3, is chosen. The quantity U_{DC} is itself variable and, with respect to the hardware, has to be normalized by $U_{\max}$ while measuring. The equation (12.1) is totally identical with the following:

$$\begin{cases} \dfrac{i_{sd}(k+1)}{I_{\max}} = \Phi_{11}\dfrac{i_{sd}(k)}{I_{\max}} + \Phi_{12}\dfrac{i_{sq}(k)}{I_{\max}} + \Phi_{13}\dfrac{\psi'_{rd}(k)}{I_{\max}} \\ \qquad + h_{11}\dfrac{2U_{\max}}{3I_{\max}}\left(\dfrac{U_{DC}(k)}{U_{\max}}\right)\left(\dfrac{u_{sd}(k)}{\frac{2}{3}U_{DC}}\right) \\ \dfrac{i_{sq}(k+1)}{I_{\max}} = -\Phi_{12}\dfrac{i_{sd}(k)}{I_{\max}} + \Phi_{11}\dfrac{i_{sq}(k)}{I_{\max}} - \Phi_{14}\dfrac{\psi'_{rd}(k)}{I_{\max}} \\ \qquad + h_{11}\dfrac{2U_{\max}}{3I_{\max}}\left(\dfrac{U_{DC}(k)}{U_{\max}}\right)\left(\dfrac{u_{sq}(k)}{\frac{2}{3}U_{DC}}\right) \end{cases} \tag{12.2}$$

In the equation (12.2) new symbols are introduced and replaced:

$$\begin{aligned} & i^N_{sd,sq} = \frac{i_{sd,sq}}{I_{\max}};\ \psi'^N_{rd} = \frac{\psi'_{rd}}{I_{\max}};\ u^N_{sd,sq} = \frac{u_{sd,sq}}{2U_{DC}/3} \\ & h^N_{11} = k_u U^N_{DC};\ k_u = h_{11}\frac{2U_{\max}}{3I_{\max}};\ U^N_{DC} = \frac{U_{DC}}{U_{\max}} \end{aligned} \tag{12.3}$$

Superscripts *N*: normalized quantities

The parameters Φ_{12}, Φ_{14} are frequency dependent. The value $f_{\max}$ is used for the normalization of the frequencies. Using (3.54) it can be written then:

$$\begin{aligned} \Phi_{12} &= \omega_s T = 2\pi f_s T = (2\pi f_{\max} T)\left(\frac{f_s}{f_{\max}}\right) = k_{f1} f^N_s \\ \Phi_{14} &= \frac{1-\sigma}{\sigma}\omega T = \frac{1-\sigma}{\sigma}2\pi f T = \left(\frac{1-\sigma}{\sigma}2\pi f_{\max} T\right)\left(\frac{f}{f_{\max}}\right) \\ &= k_{f2} f^N \end{aligned} \tag{12.4}$$

In (12.4) the symbols mean:

[1] U_{DC}: DC link voltage of the inverter

$$f_s^N = \frac{f_s}{f_{\max}};\ f^N = \frac{f}{f_{\max}};\ k_{f1} = 2\pi f_{\max} T;\ k_{f2} = \frac{1-\sigma}{\sigma} 2\pi f_{\max} T \tag{12.5}$$

The equation (12.1) can now be rewritten in the normalized form, in which the constants k_u, k_{f1} and k_{f2} as well as the constant parameters Φ_{11}, Φ_{13} have to be calculated only at the beginning, i.e. at the initialization of the system.

$$\begin{cases} i_{sd}^N(k+1) = \ \ \Phi_{11} i_{sd}^N(k) + \Phi_{12} i_{sq}^N(k) + \Phi_{13} \psi_{rd}^{\prime N}(k) \\ \qquad\qquad + h_{11}^N u_{sd}^N(k) \\ i_{sq}^N(k+1) = -\Phi_{12} i_{sd}^N(k) + \Phi_{11} i_{sq}^N(k) - \Phi_{14} \psi_{rd}^{\prime N}(k) \\ \qquad\qquad + h_{11}^N u_{sq}^N(k) \end{cases} \tag{12.6}$$

The original equation (12.1) exists now in programmable form without loss of its physical meaning. *The voltages represent the degree of modulation in this normalized form* in which the variable DC link voltage U_{DC} is considered by the parameter h_{11}^N, which has to be updated on-line.

To achieve the most possible numerical accuracy with fixed point or integer arithmetic, the normalized quantities (represented in hexadecimal form) are shifted to the left (multiplied with 2) as much as possible without producing overflow. For normalized currents, voltages and frequencies which accept only values smaller than one, the multiplication factor can be e.g. 2^{15} for 16 bit fixed point processors. This process is commonly described as the *scaling*. The multiplication factor of 2^{15} is the *scaling factor* which at the same time means the number of digits behind the comma. For parameters, which by their nature are already greater than one, the scaling has to be carried out in a way that on the one hand the maximum word length is used, but on the other hand overflow is avoided simultaneously. In principle, this problem does not exist any more with the use of floating point processors and only appears for conversions between data types again, e.g. between integer numbers and signed floating point numbers.

b) Example 2:

A further typical example is shown by normalizing and scaling of the quantities within the equation (12.6) for the calculation of the angular velocity ω.

$$\begin{aligned}\vartheta(k+1) &= \vartheta(k) + \omega(k)T \\ &= \vartheta(k) + 2\pi f(k)T\end{aligned} \qquad (12.7)$$

If the values 2π and $f_{\max}$ are chosen as normalizing quantities for the angle and frequency, and it is considered that:

- the frequency has to be signed (e.g. positive for right and negative for left rotation), and
- the angle has to be unsigned (i.e. only forwards counting 0, π, 2π, 3π, 4π...)

then e.g. 2^{15} is possible to be used as scaling factor for the frequency at 16 bit word length, and 2^{16} for the angle. From (12.7) it will be obtained:

$$\left[2^{16}\frac{\vartheta(k+1)}{2\pi}\right] = \left[2^{16}\frac{\vartheta(k)}{2\pi}\right] + \left[2^{15}\frac{f}{f_{\max}}\right]\left(2 f_{\max}T\right) \qquad (12.8)\text{a}$$

or

$$\left[2^{16}\vartheta^N(k+1)\right] = \left[2^{16}\vartheta^N(k)\right] + \left[2^{15} f^N(k)\right]k_{f3} \qquad (12.8)\text{b}$$

In the equation (12.8)b it means:

- $2^{16}\times\vartheta^N$ the unsigned integer calculation quantity for the angle,
- $2^{15}\times f^N$ the signed calculation quantity for the frequency

12.2 Example for the model discretization in the section 3.1.2

A system of second order following (3.5) is given with:

$$\mathbf{A} = \begin{bmatrix} a & \omega \\ -\omega & a \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} b & 0 \\ 0 & b \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad (12.9)$$

a) Method 1: Series expansion with truncation after the linear term (Euler). The use of (3.14) provides:

$$\mathbf{\Phi} = \begin{bmatrix} 1+aT & \omega T \\ -\omega T & 1+aT \end{bmatrix}; \quad \mathbf{H} = T\begin{bmatrix} b & 0 \\ 0 & b \end{bmatrix} \qquad (12.10)$$

b) Method 2: Series expansion with truncation after the quadratic term. The use of (3.14) provides again:

$$\mathbf{\Phi} = \begin{bmatrix} 1+aT+\left[(aT)^2-(\omega T)^2\right]/2 & \omega T(1+aT) \\ -\omega T(1+aT) & 1+aT+\left[(aT)^2-(\omega T)^2\right]/2 \end{bmatrix}$$

(12.11)a

$$\mathbf{H} = bT\begin{bmatrix} 1+aT/2 & \omega T/2 \\ -\omega T/2 & 1+aT/2 \end{bmatrix} \tag{12.11)b}$$

c) Method 3: Euler discretization in a suitable coordinate system.

The method is applicable if the system matrix **A** owns the symmetry properties of the special block diagonal structure (12.9) which is often the case at the modeling of three-phase machines. In this case, state and input quantities can be understood as complex variables.

$$\begin{aligned} &\mathbf{x} = x_x + j\,x_y \\ &\overset{\bullet}{\mathbf{x}}(t) = (a - j\omega)\,\mathbf{x}(t) + b\,\mathbf{u}(t) \end{aligned} \tag{12.12}$$

At first, the continuous system is viewed in a coordinate system which circulates with the frequency -ω with respect to the target coordinate system, i.e. $\mathbf{x} = \mathbf{x}^{\omega} e^{-j\omega t}$ (to the topic “transformation of coordinates” cf. chapter 1). Considering the product rule, the following is obtained for the time derivative:

$$\overset{\bullet}{\mathbf{x}}^{\omega}(t) = a\,\mathbf{x}^{\omega}(t) + b\,\mathbf{u}^{\omega}(t) \tag{12.13}$$

The discretization using Euler method leads to the following discrete state equation:

$$\mathbf{x}^{\omega}(k+1) = (1+aT)\mathbf{x}^{\omega}(k) + bT\,\mathbf{u}^{\omega}(k) \tag{12.14}$$

For the inverse transformation into the target coordinate system, the equation (12.14) has to be subjected to the counter-rotation, i.e.

$$\mathbf{x}^{\omega}(k) = \mathbf{x}(k)e^{j\vartheta(k)},\ \mathbf{x}^{\omega}(k+1) = \mathbf{x}(k+1)e^{j\vartheta(k+1)} \tag{12.15}$$

Thereat, the discrete transformation angle ϑ results by Euler discretization as follows:

$$\vartheta(k+1) = \vartheta(k) + \omega T \tag{12.16}$$

The state equation is now in the target coordinate system:

$$\mathbf{x}(k+1) = e^{-j\omega T}\left[(1+aT)\mathbf{x}(k) + bT\,\mathbf{u}(k)\right] \tag{12.17}$$

or resolved with discrete transfer matrices:

$$\mathbf{\Phi} = (1+aT)\begin{bmatrix} \cos\omega T & \sin\omega T \\ -\sin\omega T & \cos\omega T \end{bmatrix};\ \mathbf{H} = bT\begin{bmatrix} \cos\omega T & \sin\omega T \\ -\sin\omega T & \cos\omega T \end{bmatrix} \tag{12.18}$$

d) Method 4: Substitute function using the Sylvester-Lagrange substitute polynomials.

The eigen values of the continuous system matrix **A** will be:

$$\lambda_{1,2} = a \pm j\omega$$

It follows then for equations (3.19) to (3.22):

$$\begin{aligned}
&M(\lambda)=(\lambda-\lambda_1)(\lambda-\lambda_2)\\
&M_1=(\lambda-\lambda_1),\ M_2=(\lambda-\lambda_2)\\
&m_1=(\lambda_1-\lambda_2),\ m_2=(\lambda_2-\lambda_1)\\
&R(\lambda)=\frac{\lambda_1 e^{\lambda_2 T}-\lambda_2 e^{\lambda_1 T}}{\lambda_1-\lambda_2}+\lambda\frac{e^{\lambda_1 T}-e^{\lambda_2 T}}{\lambda_1-\lambda_2}
\end{aligned} \tag{12.19}$$

Therewith the substitute function □(□) can be given as:

$$\mathbf{R}(\mathbf{A})=\mathbf{\Phi}=\frac{\lambda_1 e^{\lambda_2 T}-\lambda_2 e^{\lambda_1 T}}{\lambda_1-\lambda_2}\mathbf{I}+\frac{e^{\lambda_1 T}-e^{\lambda_2 T}}{\lambda_1-\lambda_2}\mathbf{A} \tag{12.20}$$

and finally the system matrix **Φ**:

$$\mathbf{\Phi}=e^{aT}\begin{bmatrix}\cos\omega T & \sin\omega T\\ -\sin\omega T & \cos\omega T\end{bmatrix} \tag{12.21}$$

The input matrix □ is calculated by direct integration of **Φ** according to (3.12):

$$\mathbf{H}=\frac{b}{a^2+\omega^2}\begin{bmatrix} e^{aT}(a\cos\omega T+\omega\sin\omega T)-a & e^{aT}(a\sin\omega T-\omega\cos\omega T)+\omega\\ -e^{aT}(a\sin\omega T-\omega\cos\omega T)+\omega & e^{aT}(a\cos\omega T+\omega\sin\omega T)-a\end{bmatrix} \tag{12.22}$$

12.3 Application of the method of the least squares regression

The method of the least squares regression is often used for the optimization of control loops or the identification of the system parameters. The goal is normally to find an approximate function $y(x)$ in the form of a polynomial of nth order

$$y(x)=a_0+a_1 x+a_2 x^2+\ldots+a_n x^n \tag{12.23}$$

from a set of m experimental measurement pairs $[y_i, x_i]$, (i = 1,2,3,...,m) and by the prerequisite, that the loss function (cf. [Rojiani 1996]):

$$S=\sum_{i=1}^{m}\left[y_i-y(x_i)\right]^2 \tag{12.24}$$

is minimized. A typical application example is the off-line identification of the main inductance L_s (cf. section 6.4.4, figure 6.18) in dependence on the magnetizing current i_μ. As an approach for $L(i)$[1] a polynomial of 4th order, thus n = 4, is very suitable.

$$L(i) = a_0 + a_1 i + a_2 i^2 + a_3 i^3 + a_4 i^4 \tag{12.25}$$

The task is now to determine the coefficients a_0, a_1, a_2, a_3 and a_4 from m pairs [L_i, i_i] with (i=1,2,3,...,m). To minimize the loss function, at first (12.25) has to be inserted into (12.24), and then the partial derivations

$$\frac{\partial S}{\partial a_0};\frac{\partial S}{\partial a_1};\frac{\partial S}{\partial a_2};\frac{\partial S}{\partial a_3};\frac{\partial S}{\partial a_4} \tag{12.26}$$

have to be set to zero. Thereby a system with (n+1)=5 linear equations results (cf. [Rojiani 1996]):

$$\begin{bmatrix} m & \sum_{i=1}^m i_i & \sum_{i=1}^m i_i^2 & \sum_{i=1}^m i_i^3 & \sum_{i=1}^m i_i^4 \\ \sum_{i=1}^m i_i & \sum_{i=1}^m i_i^2 & \sum_{i=1}^m i_i^3 & \sum_{i=1}^m i_i^4 & \sum_{i=1}^m i_i^5 \\ \sum_{i=1}^m i_i^2 & \sum_{i=1}^m i_i^3 & \sum_{i=1}^m i_i^4 & \sum_{i=1}^m i_i^5 & \sum_{i=1}^m i_i^6 \\ \sum_{i=1}^m i_i^3 & \sum_{i=1}^m i_i^4 & \sum_{i=1}^m i_i^5 & \sum_{i=1}^m i_i^6 & \sum_{i=1}^m i_i^7 \\ \sum_{i=1}^m i_i^4 & \sum_{i=1}^m i_i^5 & \sum_{i=1}^m i_i^6 & \sum_{i=1}^m i_i^7 & \sum_{i=1}^m i_i^8 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^m L_i \\ \sum_{i=1}^m i_i L_i \\ \sum_{i=1}^m i_i^2 L_i \\ \sum_{i=1}^m i_i^3 L_i \\ \sum_{i=1}^m i_i^4 L_i \end{bmatrix} \tag{12.27}$$

The system (12.27) can be merged into the following form:

$$\mathbf{A}[5,5] * \mathbf{a}[5] = \mathbf{b}[5] \tag{12.28}$$

thereat $\mathbf{A}[5,5]$ and $\mathbf{b}[5]$ are given by the measurement pairs, following (12.27). If C is used as programming language, then the calculation can be realized by the following program section as an example, where $\mathbf{A}[5,5]$ and $\mathbf{b}[5]$ are summarized in a matrix $\mathbf{A}[5,6]$ with $\mathbf{b}[5]$ as the 6th column. In this example it is assumed, that „MeasNum“ is the number of the measurement pairs and „PolyOrder“ the order of the approached polynomial. Here it holds:

MeasNum = 10; PolyOrder = 4

During measuring the current is increased by 0,1×ImNominal step by step from 0,1×ImNominal to ImNominal (e.g. nominal magnetizing

[1] For simplification L_s is replaced by L, and i_μ by i

current). The 10 measured inductance values are saved in the field *L*[0]...L[9].

```
/* Calculation of the sums or the elements of the matrix A[ ][ ] */
s[0] = (float)MeasNum;
for (i = 1; i <= 2*PolyOrder; i++)
  {  s[i] = 0.0;
     for (j = 1; j <= MeasNum; j++)
        s[i] += pow(ImNominal * (float)j/10.0, i);
  }
/* The sums are assigned to the elements of the matrix A[ ][ ] */
for (i = 0; i <= PolyOrder; i++)
   for (j = 0; j <= PolyOrder; j++)
      A[i][j] = s[i][j];
/* Calculation of the vector b[ ] as the 6th column of the matrix A[ ][ ] */
A[0][PolyOrder+1] = 0.0;
for (i = 0; i < MeasNum; i++)
   A[0][PolyOrder+1] += L[i];
for (i = 1; i <= PolyOrder; i++)
  {  A[i][PolyOrder+1] = 0.0;
     for (j = 0; j < MeasNum; j++)
        A[i][PolyOrder+1] += L[j] * pow(ImNominal * (float)(j+1)/10.0, i);
  }
```

After calculation of $\mathbf{A}[5,5]$ and $\mathbf{b}[5]$ the linear equation system (12.27) or (12.28) can be relatively simply solved by using the *Gauss elimination method*. The first step of the method is the *forward elimination*, which can be summarized (cf. [Sedgewick 1992]) as follows:

The first variable in all equations, with exception of the first one, has to be eliminated by addition of suitable multiples of the first equation to each of the other equations. Then the second variable in all equations, with exception of the first two, has to be eliminated by addition of suitable multiples of the second equation to each of the equations from the third up to the last one (now named as the *N*-th). Then the third variable in all equations, with the exception of the first three, has to be eliminated etc. To eliminate the *i*-th variable in the *j*-th equation (for *j* between *i*+1 and *N*), the *i*-th equation must be multiplied with a_{ji}/a_{ii} and subtracted from the *j*-th equation.

The described procedure is too simple to be completely right: a_{ii} (now named as *pivot element*) can become zero, so that a division by zero could arise. This can be avoided because any arbitrary row (from the (*i*+1)-th to the *N*-th) can be swapped with the i-th row, so that a_{ii} in the outer loop is different from zero. For swapping it is the best to use the row for which the value in the *i*-th column is the greatest with respect to the absolute amount. The reason is, that in the calculation considerable errors can arise if the pivot value, which is used to multiply a row by a factor, is very small. If a_{ii} is very small, a_{ji}/a_{ii} can become very big. This process, called the partial pivoting, is realized in the example of the L_s identification by the following program section.

```
/* Forward elimination: partial pivoting */
for (i = 0; i <= PolyOrder; i++)
   {  max = i;
      for (j = i+1; j <= PolyOrder; j++)
         if (fabs(A[j][i]) > fabs(A[max][i])) max = j;
      for (k = i; k <= PolyOrder+1; k++)
         {  Temp = A[i][k];                  /* Temp: temporary variable */
                     A[i][k] = A[max][k];
                              A[max][k] = Temp;   }
      for (j = i+1; j <= PolyOrder; j++)
         for (k = PolyOrder+1; k >= i; k– –)
            A[j][k] –= A[i][k] * A[j][i] / A[i][i]
   }
```

After the step of the forward elimination is completed, the field below the diagonal of the modified matrix □[5][5] contains only zeros. The step of the *backwards insertion* can be executed now to calculate the coefficients a_0, a_1, a_2, a_3 and a_4.

```
/* Backwards insertion: Calculation of a0, a1, a2, a3, a4,
   then storage in a[0]...[4] */
for (j = PolyOrder; j >= 0; j– –)
   {  Temp = 0.0;
      for (k = j+1; k <= PolyOrder; k++) Temp += A[j][k] * a[k];
      a[j] = (A[j][PolyOrder+1] – Temp) / A[j][j];
   }
```

With the calculated coefficients a_0, a_1, a_2, a_3 and a_4, the magnetizing curve $L(i)$ in form of a polynomial (cf. equation (12.25)) is now available.

12.4 Definition and calculation of Lie derivation

An unexcited system (input vector $\mathbf{u} = \mathbf{0}$) is defined (cf. [Wey 2001, appendix B], [Phuoc 2006, section 5.1.2]) as follows:

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}) \tag{12.29}$$

A scalar function $v(\mathbf{x})$ is given. The derivation of this scalar function along the freely moving state trajectory (along the vector field $\mathbf{f}(\mathbf{x}) \in \mathbb{R}^n$) of the unexcited system (12.29):

$$\mathbf{x}(t) = \mathbf{\Phi}_t^f(\mathbf{x}_0) \tag{12.30}$$

can be given as follows:

$$L_f v(\mathbf{x}) = \sum_{i=1}^{n} \left[\frac{\partial v(\mathbf{x})}{\partial x_i} f_i(x_1, \cdots, x_n) \right] \tag{12.31}$$

with:

$$\frac{\partial v(\mathbf{x})}{\partial \mathbf{x}} = \left[\frac{\partial v}{\partial x_1}, \quad \frac{\partial v}{\partial x_2}, \quad \cdots, \quad \frac{\partial v}{\partial x_n} \right] \tag{12.32}$$

Using (12.32), the Lie derivation $L_f v(\mathbf{x})$ can also be formulated as a scalar product (a scalar function):

$$L_f v = \frac{\partial v}{\partial \mathbf{x}} \mathbf{f} \quad \Rightarrow \quad L_f v(\mathbf{x}) = \frac{\partial v(\mathbf{x})}{\partial \mathbf{x}} \mathbf{f}(\mathbf{x}) \tag{12.33}$$

The function $L_f v(\mathbf{x})$ returns the quantitative change of $v(\mathbf{x})$ along the trajectory (12.30). The figure 12.1 illustrates this fact.

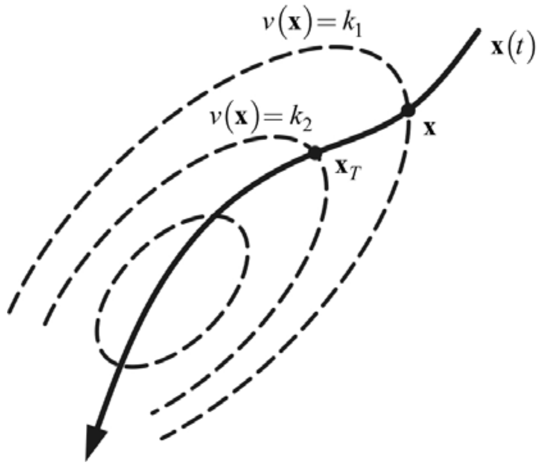

Fig. 12.1 Derivation of the scalar function $v(\mathbf{x})$ along the state trajectory $\mathbf{x}$(t)

The dashed curves in the figure 12.1 represent the sets of points inside $\mathbb{R}^n$ at which the function $v(\mathbf{x})$ has the same values. The dashed curve with the point $\mathbf{x}$ contains the set of points which fulfills $v(\mathbf{x}) = k_1$, and the curve with $\mathbf{x}_T$ the set of points fulfilling $v(\mathbf{x}_T) = k_2$. In this case, the speed of the quantitative change of $v(\mathbf{x})$ along $\mathbf{x}(t)$, from point $\mathbf{x}$ to the point $\mathbf{x}_T$, will be:

$$\begin{aligned} L_f v(\mathbf{x}) &= \lim_{T\to 0} \frac{k_2 - k_1}{T} = \lim_{T\to 0} \frac{v(\mathbf{x}_T) - v(\mathbf{x})}{T} = \lim_{T\to 0} \frac{v\left(\mathbf{\Phi}_T^f(\mathbf{x})\right) - v(\mathbf{x})}{T} \\ &= \frac{\partial v}{\partial \mathbf{x}} \lim_{T\to 0} \frac{\mathbf{\Phi}_T^f(\mathbf{x}) - \mathbf{x}}{T} = \frac{\partial v(\mathbf{x})}{\partial \mathbf{x}} \frac{d\mathbf{x}}{dt} = \frac{\partial v(\mathbf{x})}{\partial \mathbf{x}} \mathbf{f}(\mathbf{x}) \end{aligned} \tag{12.34}$$

The Lie derivation has the following properties:

- The multiple Lie derivation of $v(\mathbf{x})$, at first along the vector field $\mathbf{f}(\mathbf{x})$ and then along $\mathbf{g}(\mathbf{x})$, can be written as follows:

$$L_g L_f v(\mathbf{x}) = L_g\left[L_f v(\mathbf{x})\right] = \frac{\partial\left[L_f v(\mathbf{x})\right]}{\partial \mathbf{x}} \mathbf{g}(\mathbf{x}) = \frac{\partial}{\partial \mathbf{x}}\left[\frac{\partial v(\mathbf{x})}{\partial \mathbf{x}} \mathbf{f}(\mathbf{x})\right] \mathbf{g}(\mathbf{x}) \tag{12.35}$$

- Let $w(\mathbf{x})$ be an additional scalar function, then the following relation is valid:

$$L_{wf} v(\mathbf{x}) = \left[L_f v(\mathbf{x})\right] w$$

$$\text{because} \quad L_{wf} v(\mathbf{x}) = \frac{\partial v(\mathbf{x})}{\partial \mathbf{x}} (w\mathbf{f}) = \underbrace{\left[\frac{\partial v(\mathbf{x})}{\partial \mathbf{x}} \mathbf{f}(\mathbf{x})\right]}_{L_f v(\mathbf{x})} w(\mathbf{x}) \tag{12.36}$$

- Let k be an integer number, then the k-fold Lie derivation of $v(\mathbf{x})$ along $\mathbf{f}(\mathbf{x})$ can be recursively calculated as follows:

$$L_f^k v(\mathbf{x}) = \frac{\partial\left[L_f^{k-1} v(\mathbf{x})\right]}{\partial \mathbf{x}} \mathbf{f}(\mathbf{x}) \quad \text{with} \quad L_f^0 v(\mathbf{x}) = v(\mathbf{x}) \tag{12.37}$$

12.5 References to chapter 12

Rojiani KB (1996) Programming in C with numerical methods for engineers. Prentice-Hall International, Inc.

Sedgewick R (1992) Algorithmen in C. Addison-Wesley

Phuoc ND, Minh PX, Trung HT (2006) Nonlinear control theory. Publishing House of Science and Technique, Hanoi (in Vietnamese)

Wey T (2001) Nichtlineare Regelungssysteme: Ein differentialalgebraischer Ansatz. B.G. Teubner Stuttgart - Leipzig - Wiesbaden.