Comment: You are suggested to read the following portions of the text: Section 7.11, pp 262-264; Section 7.12, pg. 264 and the first paragraph at the top of pg 265; Section 7.13, pp. 275-277; Section 7.14, pp. 277-278; Appendix B, 675-683. Also, you can take entire courses on numerical methods for solving differential equations, e.g., Math 481, 561, and 581.

Numerical Methods

1.0 Problem set-up

We previously showed that we may write the swing equation for a single machine as a pair of first-order differential equations, according to:

$$\dot{\omega} = \frac{\omega_{\mathrm{Re}}}{2H}\left[P_m - P_e(\delta(t))\right] \tag{1}$$

$$\dot{\delta} = \omega \tag{2}$$

Here, $P_e(\delta(t))$ is given by the power flow equations:

$$P_{ei} = E_i^2 G_{ii} + \sum_{\substack{j=1 \\ j\neq i}} E_i E_j Y_{ij} \cos(\theta_{ij} - \delta_i + \delta_j) \tag{3}$$

where E_i, E_j are the internal bus voltage magnitudes having angles of δ_i and δ_j, respectively, and $Y_{ij}=G_{ij}+jB_{ij}=Y_{ij}/\_\theta_{ij}$ is the element in the ith row, jth column of the appropriate Y-bus. Depending on the particular time interval we are integrating, the "appropriate Y-bus" will be Y_1, Y_2, or Y_3, corresponding to pre-fault, fault-on, and post-fault conditions, respectively, as described in the notes called "Multimachine."

The solution to (1) and (2) is found by integrating both sides of each equation, resulting in

$$\omega(t) = \int_0^t \dot{\omega}(\tau) d\tau = \int_0^t \frac{\omega_{\mathrm{Re}}}{2H} \left[P_m - P_e(\delta(\tau)) \right] d\tau \tag{4}$$

$$\delta(t) - \delta(0) = \int_0^t \dot{\delta}(\tau) d\tau = \int_0^t \omega(\tau) d\tau \tag{5}$$

Note that δ appears in the integrand of (4),

and that ω appears in the integrand of (5).

But these functions are what we are trying to compute. And so we observe that in general, we will not be able to obtain any closed form expression for δ and ω; we cannot solve for them in closed form. Thus, our only choice is to resort to numerical integration.

More compactly, we may define x_1=ω and x_2=δ, so that (4) and (5) may be written as

$$\left. \begin{aligned} \dot{x}_1 &= f_1(x_1, x_2) \\ \dot{x}_2 &= f_2(x_1, x_2) \end{aligned} \right\} \Rightarrow \underline{\dot{x}} = \underline{f}(\underline{x}) \tag{6a}$$

At this point, we need to recall that,

if we want to represent additional nodes beyond the internal machine nodes, i.e., if we want to represent any load buses within the solution procedure so that related information is available to us from the solution procedure,

➔the differential-algebraic system (DAS) of the power system "electromechanical positive-sequence time-domain simulation problem" has another set of equations to deal with;

➔the algebraic set, which is a function of the algebraic variables and the states.

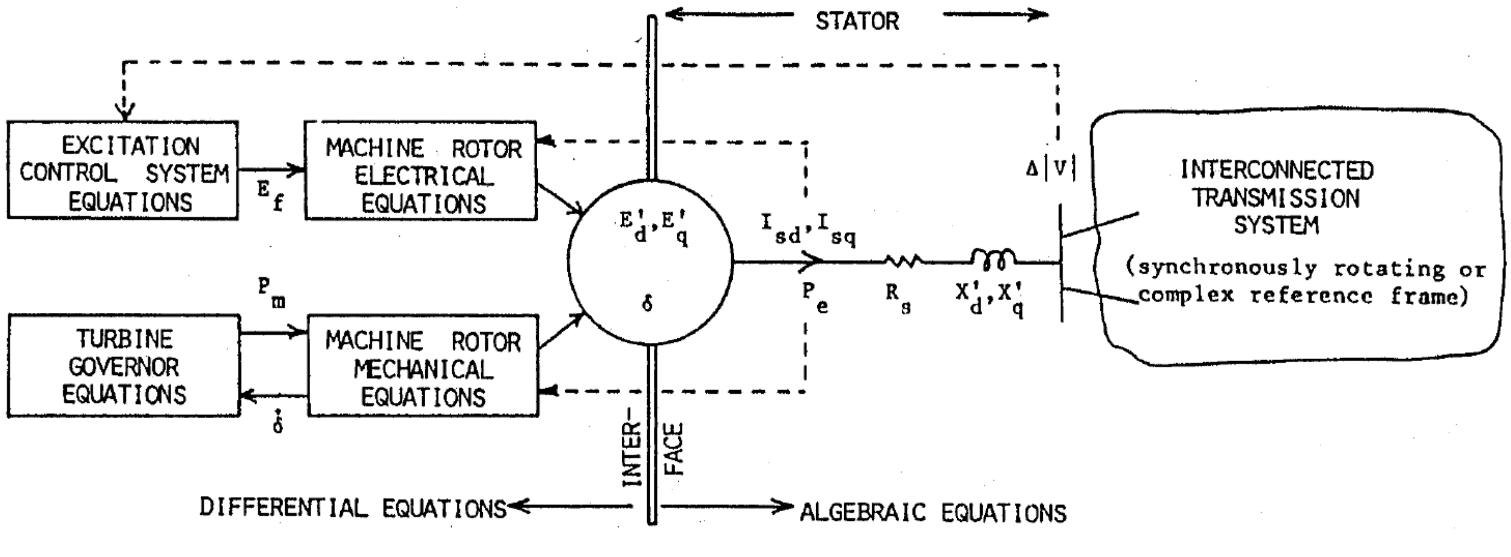
Thus, our real problem is stated as (6a) plus the algebraic set, i.e.,

$$\begin{aligned}\underline{\dot{x}} &= \underline{f}\left(\underline{x}, \underline{y}\right)\\ \underline{0} &= \underline{g}(\underline{x}, \underline{y})\end{aligned} \tag{6b}$$

where $\underline{x}$ represent the state variables, $\underline{y}$ represents the algebraic variables, and $\underline{0}=\underline{g}(\underline{x}, \underline{y})$ represents the algebraic equations.

One advantage we do have in seeking to solve (6b) is that we do know $\underline{x}(0)$, i.e., we know the initial angles and speeds δ(0) and ω(0), so it is an initial value problem (δ(0) comes from the solution of the power flow equations, and ω(0)=0).

Figure 0a, taken from a very famous paper on this subject [1], shows the so-called "interface problem" between the differential and algebraic equations of the DAS. This problem refers to the need to solve both the ODEs of the system (on the left of Fig. 0a) as well as the algebraic equations of the system (on the right of Fig. 0a).



STATOR
EXCITATION CONTROL SYSTEM EQUATIONS
E_f
MACHINE ROTOR ELECTRICAL EQUATIONS
E'_d, E'_q
δ
I_{sd}, I_{sq}
P_e
R_s
X'_d, X'_q
$\Delta|V|$
INTERCONNECTED TRANSMISSION SYSTEM
(synchronously rotating or complex reference frame)
P_m
TURBINE GOVERNOR EQUATIONS
MACHINE ROTOR MECHANICAL EQUATIONS
$\dot{\delta}$
INTER- FACE
DIFFERENTIAL EQUATIONS
ALGEBRAIC EQUATIONS

Fig. 0a

In general, there are two classes of methods for addressing the interface problem and thus solving the ODEs and the algebraic equations:

- Alternating (or partitioned) approach with initially, t=0:
  1. solve the differential equations at t=Δt using values of the algebraic variables from t=0 power flow solution;
  2. solve the algebraic equations at t+ Δt using angles from the differential equations;
  3. let t ← t+Δt; solve the differential equations at t+Δt using values of algebraic variables from step 2; return to step 2.

  The alternating method applies numerical integration to only the discretized ODEs (and not to the original algebraic equations). For this reason, the alternating method may use either explicit integration (which integrates only and so cannot include the algebraic equations) or implicit integration (which integrates by solving nonlinear algebraic equations and so can include the algebraic equations).

- Direct (or simultaneous) solution method: Here, we combine the ODEs (discretized, and therefore algebraic equations) and the original algebraic equations into a single set of algebraic equations. This single set of algebraic equations are nonlinear, and so a nonlinear solver such as Newton-Raphson is used to solve them. Because the direct solution method solves together the discretized ODEs and the algebraic equations, it cannot use explicit integration; rather, it *must* use implicit integration.

Figure 0b (7.13 in your text) illustrates the various design decisions associated with building a time-domain simulation software application. The top attribute (hardware) is whether one wants to parallelize the computations or not. The second layer is whether one uses alternating or direct solution methods. The third layer is whether one uses explicit or implicit integration methods (we will discuss this further in these notes). The fourth layer depicts the type of nonlinear solver chosen. The fifth layer shows that a linear equation solver is also needed, because most nonlinear solvers require one.

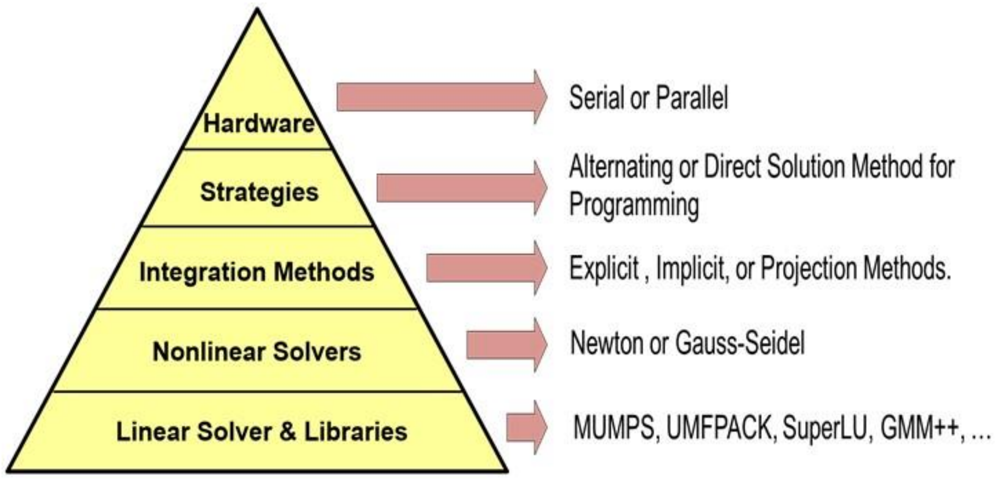

Hardware
Strategies
Integration Methods
Nonlinear Solvers
Linear Solver & Libraries
Serial or Parallel
Alternating or Direct Solution Method for Programming
Explicit , Implicit, or Projection Methods.
Newton or Gauss-Seidel
MUMPS, UMFPACK, SuperLU, GMM++, ...

Figure 0b

Towards the end of these notes (Sec 5.2, pp. 38-40), we return to the topic of solving ODEs vs. algebraic equations, while explaining how to treat the network when retaining nodes beyond those representing internal machine nodes.

Regardless of whether we use the alternating approach (with either explicit or implicit integration) or the direct solution approach (with implicit integration), we must solve ODEs of the problem posed in (6b) (repeated here for convenience):

$$\begin{aligned} \underline{\dot{x}} &= \underline{f}\left(\underline{x}, \underline{y}\right) \\ \underline{0} &= \underline{g}(\underline{x}, \underline{y}) \end{aligned} \tag{6b}$$

And so let’s look at a simpler initial value ODE problem in one-dimension (i.e., a single state variable):

$$\dot{x} = f(x(t)), \quad x(0) = x_0 \tag{7}$$

So we want to solve

$$x(t) = \int_0^t \dot{x}(\tau) d\tau = \int_0^t f(x(\tau)) d\tau \tag{8}$$

If we are going to try a numerical solution, i.e., one using computers, then we must deal with discrete time. In discrete time, (8) becomes:

$$\begin{aligned} x(kT) &= \int_0^{kT} f(x(k\tau))d\tau \\ &= \underbrace{\int_0^{kT-T} f(x(k\tau))d\tau}_{x(kT-T)} + \int_{kT-T}^{kT} f(x(k\tau))d\tau \end{aligned} \tag{9}$$

From (9), we can write:

$$x(kT) = x(kT-T) + \int_{kT-T}^{kT} f(x(k\tau))d\tau \tag{10}$$

Equation (10) says the following:

If we want to know x at the “next” step, kT, we need to know x at the “last” step, kT-T, and we need to know the integral in (10). This integral is giving us the change in x from the last step to the next, i.e.,

$$\Delta x = \int_{kT-T}^{kT} f(x(k\tau))d\tau \tag{11}$$

Since we do know x(0), we can say that we initially know x at the “last” step. Thus, solving our problem requires only the ability to compute the integral in (11).

There are many methods that will allow us to compute the integral in (11). We will review only a few of them. You should read Appendix B in VMAF as background material for this topic.

2.0 Euler method

Consider plotting our function f(x(t)) as a function of t. What we want to do, based on (11), is to obtain the area under the curve of f(x(t)) from t=kT-T to t=kT, as illustrated in Fig. 1.

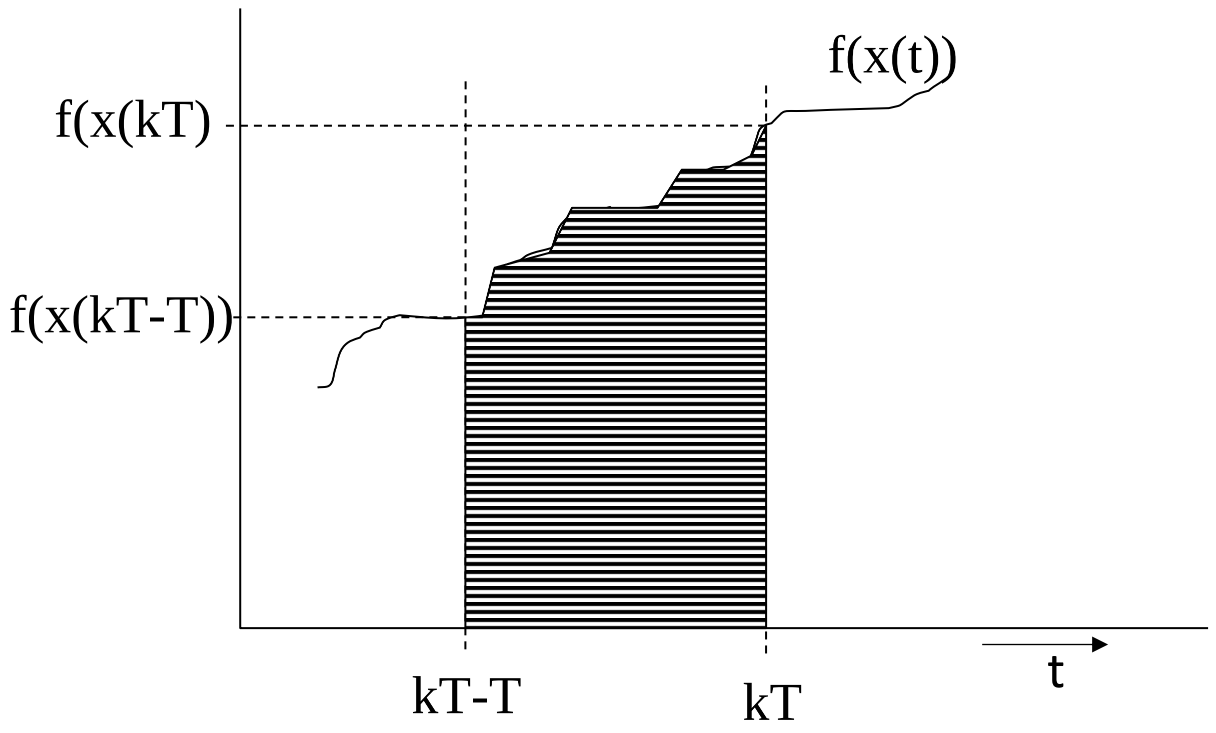


Fig. 1

Here is a proposed approach: Assume that f is constant throughout the interval at f(kT-T). This assumption is clearly not good if kT is large, but it might be reasonable if kT is made small enough.

This approach approximates the integral as the area shaded by the vertical lines in Fig. 2. One observes that it misses the area shaded by the horizontal lines in Fig. 2.

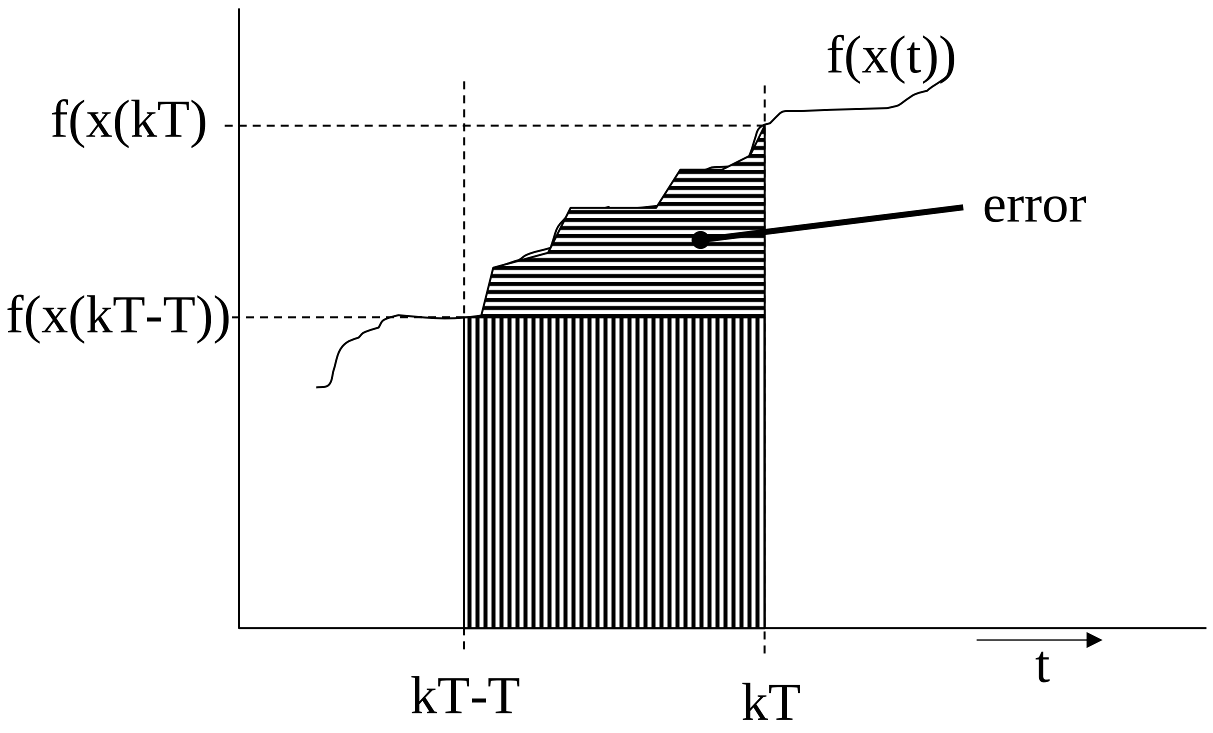


Fig. 2

$$x(kT) = x(kT - T) + \int_{kT-T}^{kT} f(x(k\tau))d\tau \qquad (10)$$

Analytically, (10) becomes

$$x(kT) = x(kT - T) + Tf(x(kT - T)) \qquad (12)$$

where the last term is Δx. This approach is also called the “forward rule” because we assume a value for f and hold it constant as we move forward in time.

2.1 Alternative development of forward rule

We may also develop the forward rule in another way….

Assume that we know x(kT-T) and that T is chosen small enough so that x(kT) is close to x(kT-T). Then by Taylor series,

$$\begin{aligned} x(kT) &= x(kT-T) + T\dot{x}\big|_{t=kT-T} + \frac{T^2}{2}\ddot{x}\big|_{t=kT-T} + \frac{T^2}{3!}\dddot{x}\big|_{t=kT-T} + \ldots. \\ &= x(kT-T) + T\dot{x}\big|_{t=kT-T} + \mathrm{O}(T^2) \end{aligned} \tag{13}$$

where O(T^2) is the remainder of the Taylor series, and its argument T^2 indicates that the lowest power of T present in the remainder is T^2.

If T is small enough, O(T^2) is negligible and

$$x(kT) = x(kT-T) + T\dot{x}\big|_{t=kT-T} \tag{14}$$

where the last term is Δx. This is illustrated in Fig. 3.

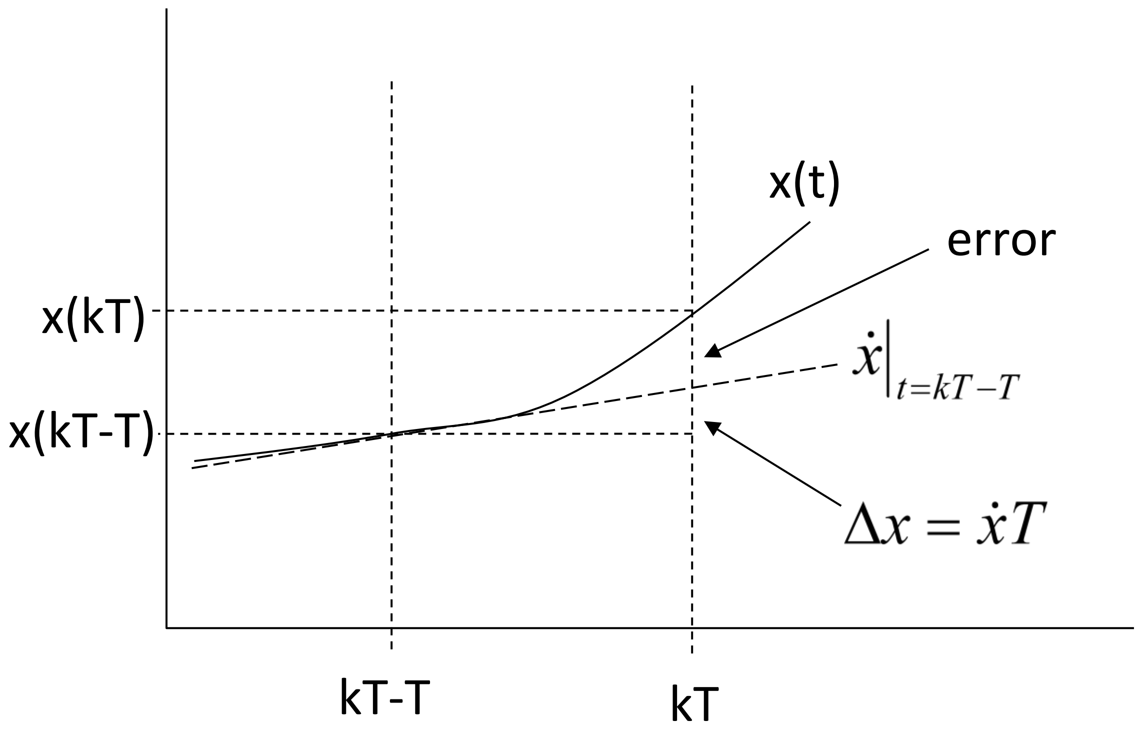


Fig. 3

Recalling that $\dot{x} = f(x)$, (14) becomes

$$x(kT) = x(kT-T) + Tf(x(kT-T)) \qquad (15)$$

which is the same as (12), our forward rule.

From both Figs. 2 and 3, we can observe that the forward method will incur some error, and this error will increase with larger values of T.

One major problem with this method is that if T is too large, the error will *propagate* from one time step to another.

This means that error incurred at one time step kT will add to the error incurred at later time steps. Solution methods where this can happen are said to be numerically unstable.

However, occurrence of this phenomenon does depend on our choice of T. I will provide you with a way to consider this issue.

2.2 Numerical stability of forward rule

The following development may be hard to follow if you have not had a course in discrete-time systems and control. ISU offers such a course as EE 576. The text by Franklin & Powell is a well-known text in this area.

I will state some “Facts,” without proof, that we need in order to consider numerical stability of integration schemes. This will give you a framework to consider this issue. If you want more depth, take EE 576, or read the Franklin & Powell book, or read a similar one.

Fact 1: A transfer function in Laplace, H(s) corresponding to continuous time impulse response h(t), may be obtained for a dynamical system.

Fact 2: The eigenvalues of the system are the poles of the transfer function, i.e., the roots of the denominator of H(s).

Fact 3: The system is stable if all poles are in the left-hand-plane.

Fact 4: We may discretize H(s) in a number of different ways, and the particular way of discretization will specify the mapping between the Laplace variable "s" (used in the transform H(s) of the continuous-time function h(t)) to the z-domain variable z (used in the transform H(z) of the discrete-time function h(kT)). Specifically, use of the forward integration rule corresponds to finding a z-domain transfer function

$$H(z) = H(s)\Big|_{s=\frac{z-1}{T}} \tag{16}$$

where H(z) is the z-transform of the system's discrete time impulse response (see Franklin & Powell, pp. 54-56)

Fact 5: Whereas the continuous-time system is stable if all poles of H(s) *are in the left-half-plane*, the discrete-time system is stable if all poles of H(z) *are within the unit circle*. This is because z/(z-γ) corresponds to a discrete-time pole at γ, which has a z-transform of:

$$\frac{z}{z-\gamma} \leftrightarrow |\gamma|^{kT} u(kT) \tag{17a}$$

where u(kT) is the unit step-function such that

$$u(kT) = \begin{cases} 0, & k < 0 \\ 1, & k \geq 0 \end{cases} \tag{17b}$$

From (17a), we see that if | γ|>1, the time-domain term will go to infinity as time increases. This would indicate unstable behavior.

Fact 6: From Fact 4, a pole of H(s), call it s_p, maps to the z-plane according to:

$$s_p = \frac{z_p - 1}{T} \Rightarrow z_p = 1 + Ts_p \tag{18}$$

Conclusion: The stability of the discrete-time system (as indicated by whether all z_p are within unit circle) depends on

- The stability of the continuous time system through the location of continuous time system eigenvalues, s_p, and
- The time step T.

Let's look at how an eigenvalue s_p maps to the z-plane through the function (18).

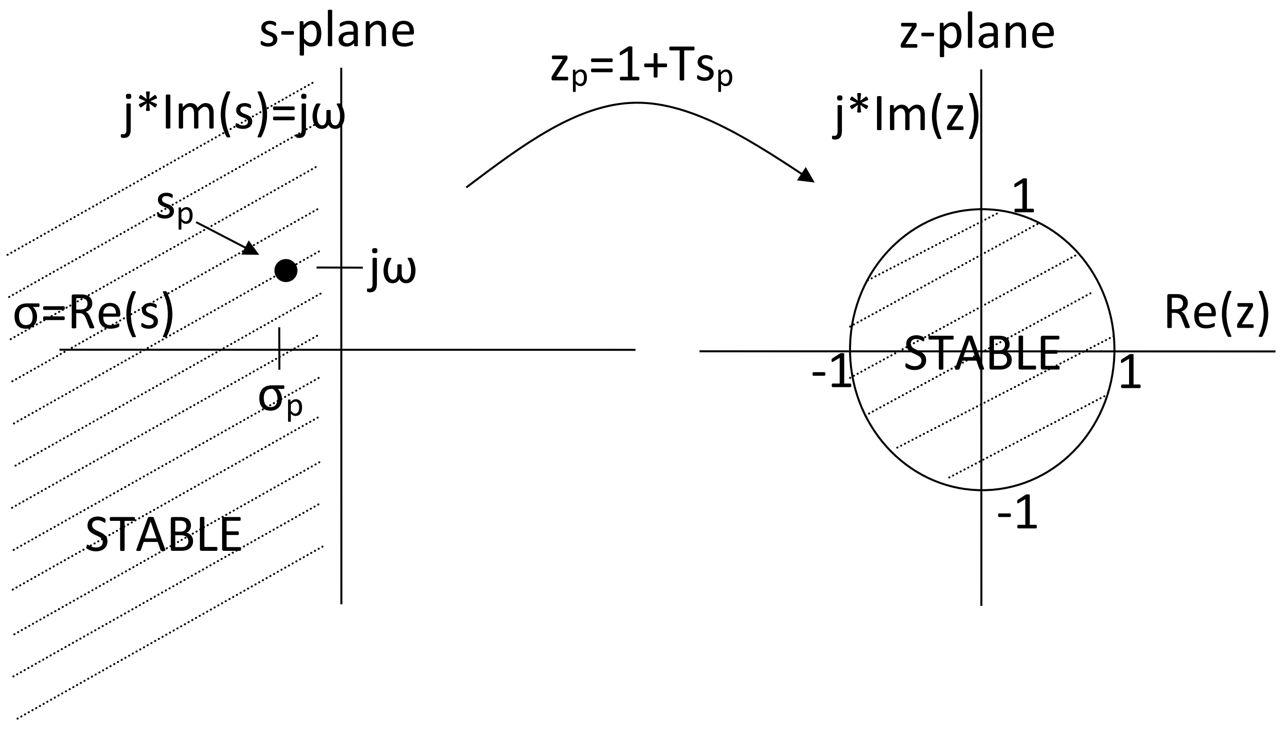


Fig. 4

From Fact 5, we recall that if all poles z_p are within the unit circle, H(z) is stable. So this implies that for stability, the poles must map to the z-plane so that

$$|z_p| < 1 \tag{19}$$

But the forward rule corresponds to a mapping of

$$z_p = 1 + Ts_p \tag{20}$$

Substitution of (20) into (19) yields:

$$|1 + Ts_p| < 1 \tag{21}$$

But

$$s_p = \sigma_p + j\omega_p \tag{22}$$

Substitution of (22) into (21) results in

$$|1 + T(\sigma_p + j\omega_p)| < 1 \tag{23}$$

Or

$$|1 + T\sigma_p + jT\omega_p| < 1 \tag{24}$$

Taking the magnitude of the complex number within (24)

$$\sqrt{(1 + T\sigma_p)^2 + (T\omega_p)^2} < 1 \tag{25}$$

Squaring both sides results in

$$(1+T\sigma_p)^2+(T\omega_p)^2<1 \tag{26}$$

Observe, in (26), that if $\sigma_p>0$ (a right-half-plane pole!), then (25) will be violated. Since (25) must be satisfied for the discrete-time-system to be stable, we see that an unstable continuous time system necessarily implies an unstable discrete time system. This is a good thing, because we would not be happy if our integration method could stabilize an unstable simulation.

And so we are no longer interested in unstable continuous-time systems since we know our integration method will also show them to be unstable, as desired. What we are interested in are the stable continuous time systems. Is it possible for our integration method to cause them to be unstable?

And so we will assume now that $\sigma_p<0$ (implying a stable continuous time system).

Expanding (26), we get

$$1+2T\sigma_p+T^2\sigma_p^2+T^2\omega_p^2<1 \tag{27}$$

Subtracting off 1 from both sides, and factoring out a T,

$$T\left(2\sigma_p + T\sigma_p^2 + T\omega_p^2\right) < 0 \tag{28}$$

Now T will always be positive (it is step size). Therefore for (28) to be true, it must be the case that

$$2\sigma_p + T\sigma_p^2 + T\omega_p^2 < 0 \tag{29}$$

Solving for T results in

$$T < \frac{-2\sigma_p}{\sigma_p^2 + \omega_p^2} \tag{30}$$

Equation (30) must be satisfied in order to be sure that the forward integration method does not create numerical instability.

But there are many eigenvalues s_p! Which one to choose?

The answer is to choose the one with the smallest ratio corresponding to the right-hand-side of (30), as that one will most restrict what T can be. This means we want the eigenvalues with small $|\sigma_p|$ and large ω_p. For these kind of eigenvalues, it will be the case that $|\sigma_p| << \omega_p$. Therefore, (30) collapses to

$$T < \frac{-2\sigma_p}{\omega_p^2} \tag{31}$$

These eigenvalues are typically the “fast” modes, usually associated with the excitation system. These eigenvalues are closely related to the time constants of the control systems.

You can test out the above theory with a stability program that uses the forward integration rule by decreasing the time constant of the excitation system for one of your generators, keeping the time step fixed. At some point, you will see instability. Then begin decreasing your time step, and then you will see it stabilize!

The “cost” of decreasing the time step is that it increases the computation time.

There are other ways of improving numerical performance of an integration method. We will look at two of these.

1. Reduce the error: we will look at two methods of doing this.
2. Use an implicit integration method: we will look at two methods of doing this.

3.0 Reducing the error

Two algorithms that improve on the forward (Euler) method are predictor-corrector and Runge-Kutta. We will look at both briefly.

<u>3.1 Predictor-corrector method</u>

This method is called the modified Euler in your text. The idea here is that we will take a step to compute x(kT) (a predictor) and then we will use that calculation to recalculate that same step (the corrector).

Step 1: Predict x(kT) using Euler to get x(kT):

In (32), the expression under f is "x-dot of kT-T".

$$x^p(kT) = x(kT-T) + T\,\underbrace{f(x(kT-T))}_{\dot{x}(kT-T)} \qquad (32)$$

Step 2: Use $x^p(kT)$ to obtain a corrected value $x^c(kT)$:

a. Get a corrected derivative f^c as the average of the derivatives at x(kT-T) and $x^p(kT)$:

$$f^c = \frac{1}{2}\left[f(x(kT-T)) + f(x^p(kT))\right] \qquad (33)$$

b. Then apply the forward rule:

$$
\begin{aligned}
x^c(kT) &= x(kT-T) + Tf^c \\
&= x(kT-T) + \frac{T}{2}\left[f(x(kT-T)) + f(x^p(kT))\right] \qquad (34)
\end{aligned}
$$

This method is illustrated in Fig. 5.

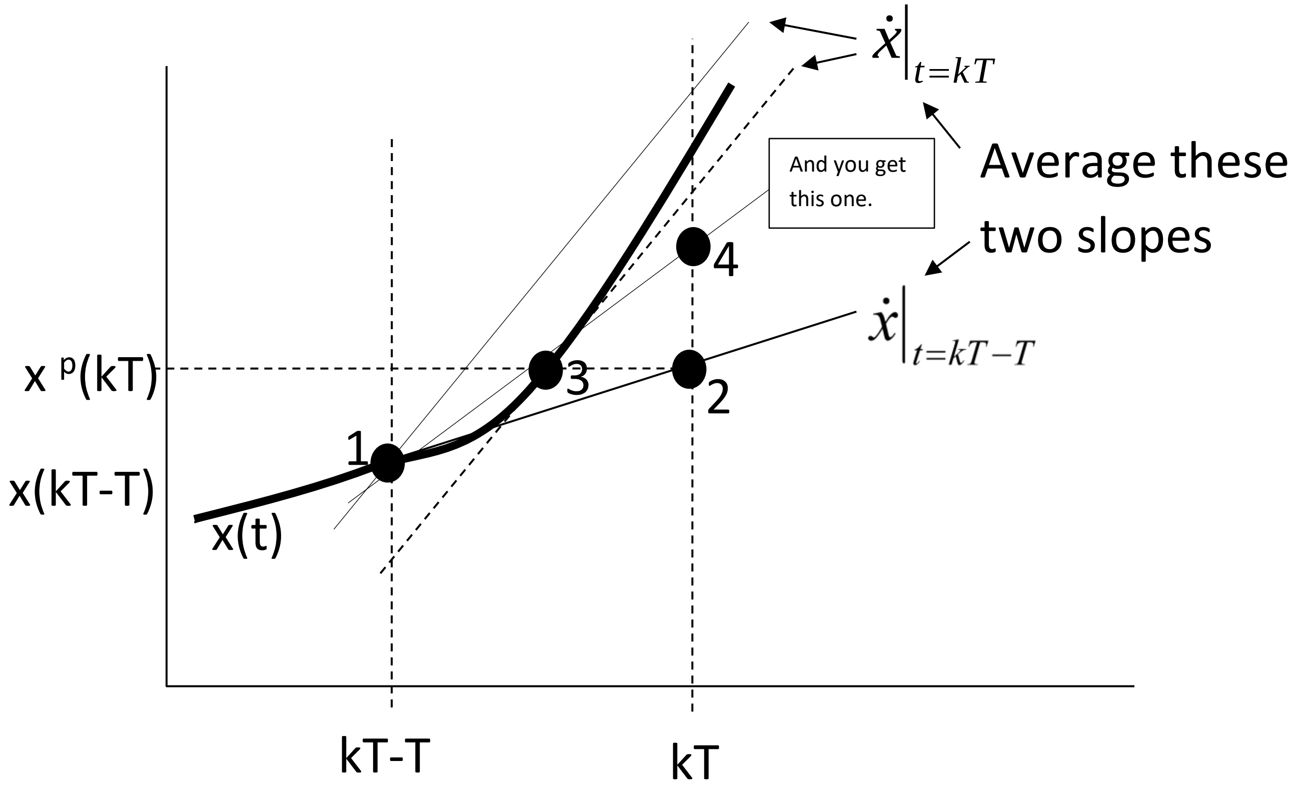


Fig. 5

Understanding the method is facilitated by observing the sequence of points in Fig. 5. The derivative f(x(kT-T)) is obtained at point 1. The predicted point $x^p(kT)$ is obtained at point 2. The derivative of the predicted point $f(x^p(kT))$ is obtained at point 3. The final point, point 4, is obtained from averaging the two obtained derivatives f(x(kT-T)) and f(x^p(kT)).

One can observe intuitively from Fig. 5 that the error will be reduced. This method can be shown to be equivalent to considering up to the second derivative term in the Taylor series, therefore the error is of O(T^3).

This is a significant improvement over the Euler method, but it is still a numerically unstable algorithm. In other words, for the predictor-corrector method, for a given minimum eigenvalue, T can be larger than it can be in the Euler method, but it is still true that the algorithm may be unstable if T is too large.

3.2 Runge-Kutta method

(pronounced Run-gah Kut-tah)

This algorithm was developed in 1895, and it also applies the idea of averaging, similar to predictor-corrector, but in a slightly different way.

There are different R-K algorithms of different order. We will only study one of them, the 4th order R-K.

The 4th order R-K method requires, at each successive time step, computing 4 different increments Δx_j, as follows:

Increment Δx_j	Derivative used
$\Delta x_1 = K_1 = Tf\,(x(kT-T))$	Start-point derivative only
$\Delta x_2 = K_2 = Tf\left(x(kT-T) + \frac{K_1}{2} \right)$	First interior derivative
$\Delta x_3 = K_3 = Tf\left(x(kT-T) + \frac{K_2}{2} \right)$	Second interior derivative
$\Delta x_4 = K_4 = Tf\left(x(kT-T) + K_3\right)$	Approx. end-point derivative

Note the following about the K_i's.

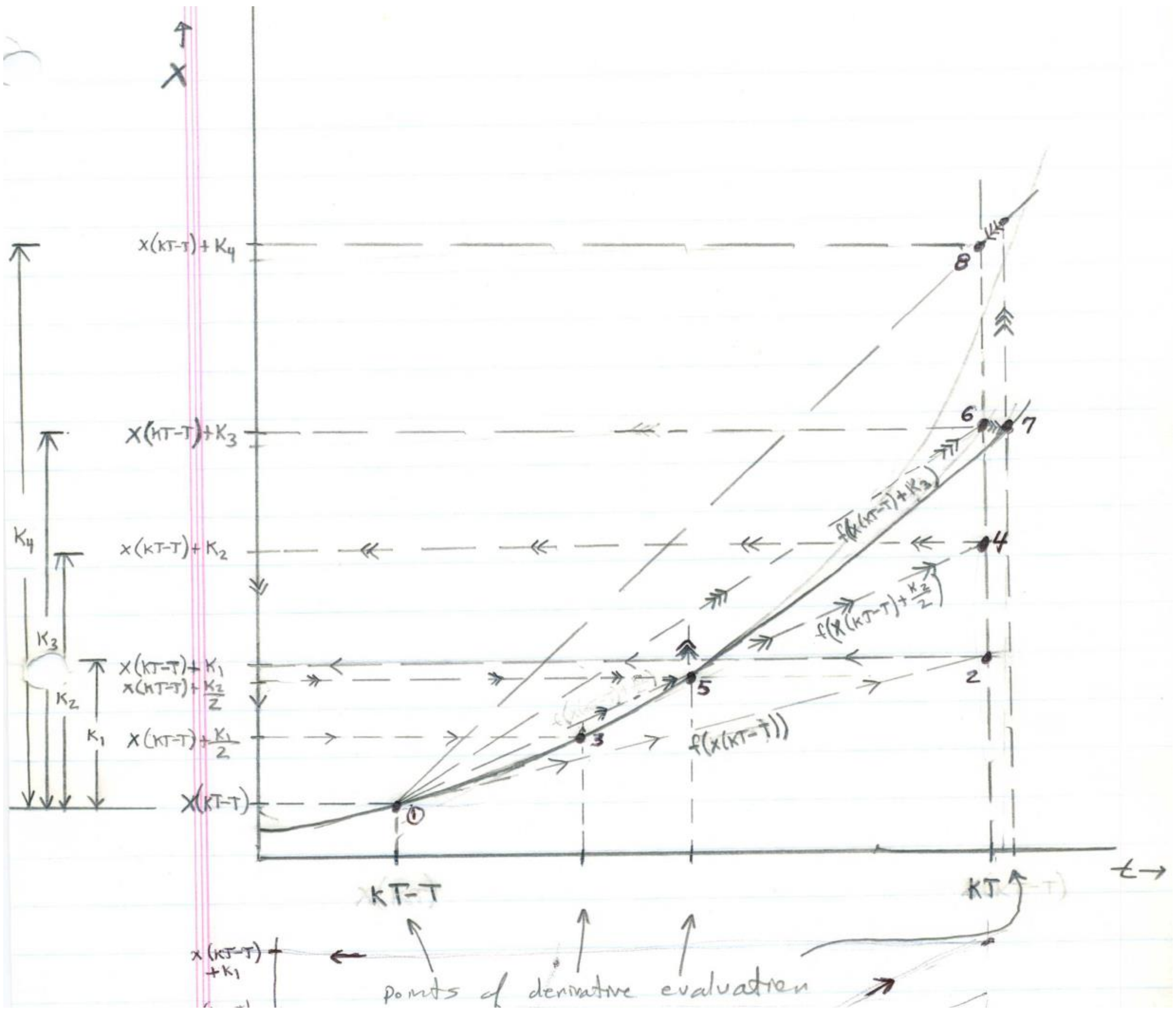
1. K_i is always used to compute K_{i+1}.
2. Each K_i is *not* a derivative but rather an increment in the integration variable, i.e.,

$$\Delta x_i = K_i = x_i(kT) - x(kT-T) \qquad (35)$$

Any of the K_i's *could* be used to obtain the new value x(kT).

3. Use of K_1 to obtain the new value x(kT) is equivalent to the Euler method.
4. The derivatives are computed at four different locations in the interval:
   - The beginning of the time step x(kT-T)
   - The first interior point x(kT-T)+K_1/2
   - The second interior point x(kT-T)+K_2/2
   - The approximate end-point point x(kT-T)+K_3

Figure 6 below illustrates the sequence of calculations, which can be understood by following the single arrows from point 1 to point 2 to point 3, and then the double arrows from point 3 to point 4 to point 5, and then the triple arrows from point 5 to point 6 to point 7 to point 8.



X
x(KT-T)+K4
X(KT-T)+K3
x(KT-T)+K2
X(KT-T)+K1
X(KT-T)+K2/2
X(KT-T)+K1/2
X(KT-T)
K4
K3
K2
K1
8
6
7
4
5
3
2
1
f(X(KT-T)+K3)
f(X(KT-T)+K2/2)
f(x(KT-T))
KT-T
KT
t→
x(KT-T)+K1
points of derivative evaluation

Fig. 6

Once each of the increments K_1-K_4 are computed, then the final increment is obtained by taking a weighted average of the four increments, where the middle increments are weighted heaviest, according to (36).

$$\Delta x = \frac{1}{6}\left(K_1 + 2K_2 + 2K_3 + K_4\right) \tag{36}$$

The middle increments (K_2 and K_3) are weighted heaviest as they are computed based on slopes that will be more representative of the slope in the interval than the beginning (K_1) and final (K_4) increments.

The R-K method can be shown to be equivalent to considering up to the fourth derivative term in the Taylor series, therefore the error is $O(T^5)$.

Although this is a significant improvement over the Euler or the P/C method, R-K is also a numerically unstable algorithm therefore the stability domain, although enlarged relative to the unit circle of the Euler, is bounded, as shown in the appendix.

4.0 Two general types of integration methods

We have so-far explored the Euler method, the P/C method, and the R-K method of numerical integration. These are in a class of integration methods called *explicit methods*. They are so-called because they evaluate x(kT) explicitly as a function of values at ***previous*** steps and their derivatives.

All explicit methods are numerically unstable, i.e., they have a bounded stability domain.

Another class of integration methods that does not have this problem is called *implicit* methods. We will study two of these: Backwards rule and Trapezoidal rule.

Implicit methods require a value of the function x(kT) at a future time step. To get this, we need to perform an extra procedural step.

Implicit methods are good for “stiff” problems, where explicit methods must utilize small step-sizes to work. Stiff problems are often characterized by large differences between the real parts of system eigenvalues.

5.0 Backwards rule

Recall that our integration problem is characterized by (10), repeated here for convenience.

$$x(kT) = x(kT - T) + \int_{kT-T}^{kT} f(x(k\tau))d\tau \qquad (10)$$

where we have the 1rst term (because we know the initial value) and need to evaluate the 2nd term, which corresponds to the area under the curve of f vs. time, as illustrated in Fig. 1, repeated here for convenience.

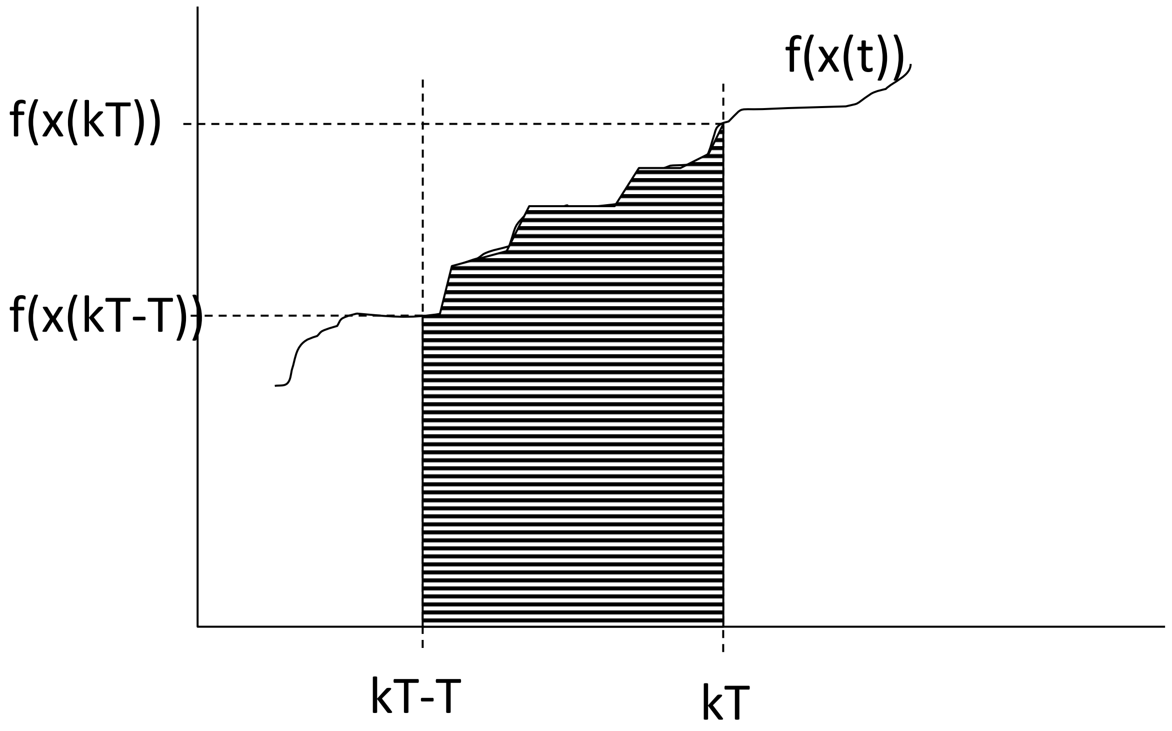


Fig. 1

In the forward (Euler) method, we assumed f is constant throughout the interval at f(kT-T). This was simple, and convenient, since we already knew f(kT-T).

Now we will again assume f is constant throughout the interval. This time, however, we will assume that it is constant at f(kT) instead of f(kT-T), as illustrated in Fig. 8.

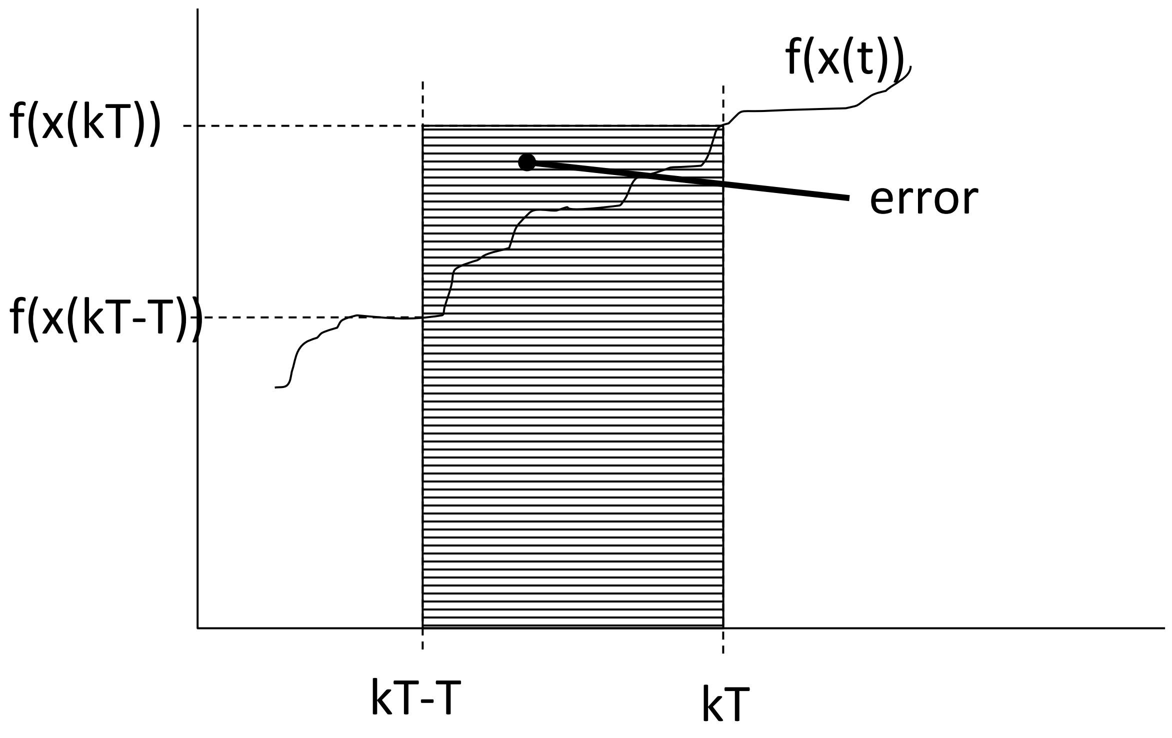


Fig. 8

Then, (10) becomes

$$x(kT) = x(kT - T) + Tf\,(x(kT)) \qquad (37a)$$

Of course, (37a) is still an approximation, but this time it includes area as indicated in Fig. 8 (as opposed to the error of the Euler method as indicated by Fig. 2).

But note a problem with this method:

➔ we do not know x(kT) !!!!

If we do not know x(kT), how do we evaluate f(x(kT))?

The answer to this question lies in observing that (37a) is not a differential equation, but rather it is an algebraic equation, and there is only one unknown, x(kT). Therefore we may solve it!

Of course, we will need to account for the fact that there is no reason to assume f is linear and in fact, most of the time, f is nonlinear. Therefore we will need to use a nonlinear algebraic solver to solve it.

The Newton-Raphson is one such solver with which we are familiar. To apply it, let's rewrite (37a) as

$$x(kT) - x(kT - T) - Tf\big(x(kT)\big) = 0 \qquad \text{(37b)}$$

and then define:

$$F(x(kT)) = x(kT) - x(kT - T) - Tf\,(x(kT)) = 0 \qquad \text{(38)}$$

Observe that, in the multi-dimensional case, (38) is just a set of nonlinear algebraic equations. Thus, we may also include the network equations into this set which enables us to solve the differential equations and the algebraic equations of the DAE simultaneously! This is one of the beauties of implicit integration methods (the other one being that they are numerically stable). See section 5.2 in these notes for additional perspective on this issue.

Apply Taylor series expansion up to the first derivative, yielding

$$\begin{aligned} &F(x(kT)+\Delta x(kT)) \\ &= F(x(kT)) + F'(x(kT))\Delta x(kT) = 0 \end{aligned} \tag{39}$$

The derivative F'(x(kT)) is the derivative of F with respect to x, not t.

Solving (39) for Δx(kT) results in

$$\Delta x(kT) = -\left[F'(x(kT))\right]^{-1} F(x(kT)) \tag{40}$$

In Newton-Raphson, we must guess an initial solution, and then we use (40) to iterate.

In the multi-dimensional case, (40) becomes

$$\underline{\Delta x}(kT) = -\left[\underline{J}(\underline{x}(kT))\right]^{-1} \underline{F}(\underline{x}(kT)) \tag{41}$$

where J is the Jacobian given by[1]

[1] See Kundur, pp. 862-864 for more on this.

$$\underline{J} = \begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \cdots & \frac{\partial F_1}{\partial x_n} \\ \vdots & \cdots & \vdots \\ \frac{\partial F_n}{\partial x_1} & \cdots & \frac{\partial F_n x}{\partial x_n} \end{bmatrix}_{x=x(kT)} \tag{42}$$

Of course, (41) is solved using a linear solver such as LU-decomposition, based on

$$\left[\underline{J}(\underline{x}(kT))\right]\underline{\Delta x}(kT) = -\underline{F}(\underline{x}(kT)) \tag{43}$$

We address three issues regarding this method: linear solvers, network solutions, and numerical instability.

<u>5.1 Linear solvers [2]</u>

An important observation about implicit methods is that (43) is just a problem in the form of Ax=b and therefore simply requires a linear solver to obtain the answer. Although solution to simultaneous linear equations is a very old topic, because implicit methods devote over 90% of their computation to this problem, it is still a very important topic for problems that require high computational speed.

Researchers have put much effort into writing computational libraries for performing solution of linear equations. The people who write these libraries know much more about solving linear equations than you do. ➔**<u>Never</u>** invert the matrix;

➔**<u>Always</u>** use the libraries;

➔**<u>Never</u>** write your own method.

There are a number of standard, portable solver libraries available, including:

- BLAS (Basic linear algebra subprograms): Many linear algebra packages including LAPACK, ScaLAPACK and PETSc, are built on top of BLAS. Most supercomputer vendors have versions of BLAS that are highly tuned for their platforms.
- ATLAS (Automatically Tuned Linear Algebra Software package) is a version of BLAS that, upon installation, tests and times a variety of approaches to each routine and selects the version that runs fastest. ATLAS is substantially faster than the generic version of BLAS.

- LAPACK (Linear Algebra PACKage) solves dense or special case sparse systems of equations depending on matrix properties such as:
    - Precision: single, double
    - Data type: real, complex
    - Shape: diagonal, bidiagonal, tridiagonal, banded, triangular, trapezoidal, Hesenberg, general dense
    - Properties: orthogonal, positive definite, Hermetian (complex), symmetric, general.

  LAPACK is built on top of BLAS, which means it can benefit from ATLAS. LAPACK is a library that you can download for free from www.netlib.org.

- ScaLAPACK is the distributed parallel (MPI) version of LAPACK. It contains only a subset of the LAPACK routines. ScaLAPACK is also available from www.netlib.org.

- PETSc (Portable, Extensible Toolkit for Scientific Computation) is a solver library for sparse matrices that uses distributed parallelism (MPI). It is designed for general sparse matrices with no special properties, but it also works well for sparse matrices with simple properties like banding & symmetry. It has a simpler Application Programming Interface than ScaLAPACK.

When choosing a solver, pick a version that’s tuned for the platform you’re running on, and use the information that you have about your system to select the one that will be most efficient. You will have to do some research and discuss with people to gain a level of knowledge to enable you to most effectively make this selection.
Figures 9 and 10 illustrate the relation between implicit integration methods, nonlinear solvers, & linear solvers.

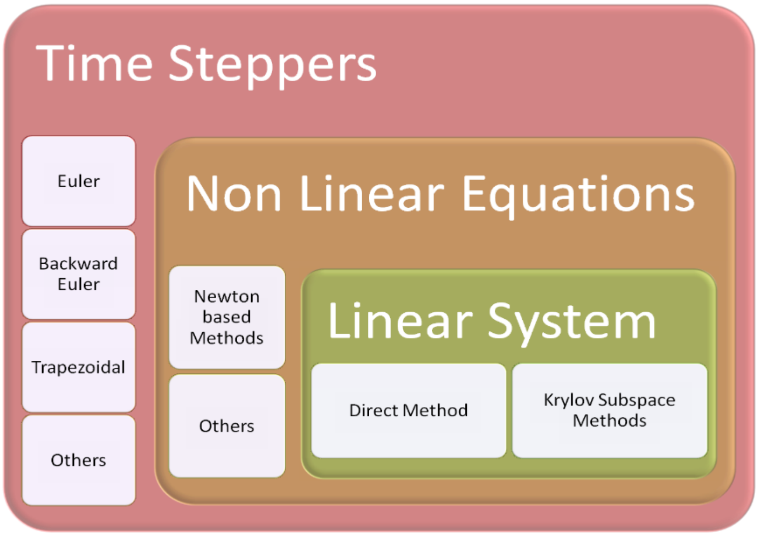


Fig. 9

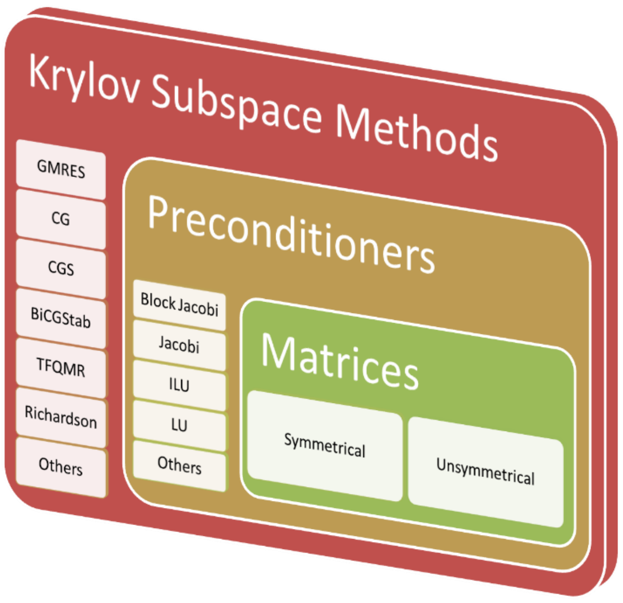


Fig. 10

5.2 Network solutions

This section recaps information given on pp. 3-5 of these notes.

In our problem formulation, because we eliminate all nodes except machine internal nodes, we are able to know all node voltage magnitudes and thus write the entire problem (swing dynamics and network equations) as a single set of state equations where the only unknowns are the states (angles and speed).

However, if we represent load as something other than constant impedance, then we may want to retain identity of these buses. But there are no swing dynamics at these buses, and so there will be no state equation for them, and so if we only solve the system differential equations, we will have no way to obtain the bus voltage magnitudes or the angles of these buses.

We will see later that the solution to this problem is to write a set of algebraic equations for the network. These equations, essentially the Y-bus relation, are then solved together with the state equations of the swing dynamics.

In the explicit integration methods, the only way to do this is through what is called a partitioned approach where the state equations are first solved and then the network equations are solved separately. The reason why this is the only way to do this is because the two sets of equations (ODEs and algebraic equations) must use different methods for solutions.

One problem encountered by the partitioned approach is referred to as the interface error, where the solution to the state equation is not quite consistent with the solution to the network equations.

The partitioned approach may be used with implicit integration methods also. In this case, implicit methods will also encounter the problem of interface error.

In addition, implicit integration methods enable a second approach called the direct solution (or simultaneous) approach, where the network equations are embedded in (43) and solved at the same time as the state equations. This very convenient method eliminates interface error.

5.3 Numerical stability for backwards rule

The following development is very similar to that given in Section 2.2. Again, additional reading may be found in the the text by Franklin & Powell.

I will again state "facts," and in fact Facts 1-3 and 5 are exactly as before. This analysis differs only in Facts 4 and 6.

Fact 1: A transfer function in Laplace, H(s) corresponding to continuous time impulse response h(t), may be obtained for a dynamical system.

Fact 2: The eigenvalues of the system are the poles of the transfer function, i.e., the roots of the denominator of H(s).

Fact 3: The system is stable if all poles are in the left-hand-plane.

Fact 4: We may discretize H(s) in a number of different ways. Whereas use of the forward integration rule corresponds to finding a z-domain transfer function

$$H(z) = H(s)\Big|_{s=\frac{z-1}{T}} \tag{16}$$

where H(z) is the z-transform of the system's discrete time impulse response (see Franklin & Powell, pp. 54-56), use of the backwards integration rule corresponds to finding a z-domain transfer function

$$H(z) = H(s)\Big|_{s=\frac{z-1}{Tz}} \tag{44}$$

Fact 5: The discrete-time system is stable if all poles of H(z) are within the unit circle. This is because z/(z-γ) corresponds to a discrete-time pole, which has a z-transform of:

$$\frac{z}{z-\gamma} \leftrightarrow |\gamma|^{kT} u(kT) \tag{17}$$

From (17), we see that if | γ|>1, the time-domain term will go to infinity as time increases.

<u>Fact 6</u>: From Fact 4, whereas a pole of H(s), call it s_p, maps to the z-plane in the forward rule according to:

$$s_p = \frac{z_p - 1}{T} \Rightarrow z_p = 1 + Ts_p \tag{18}$$

in the backwards rule, a pole of H(s), s_p, maps to the z-plane according to:

$$s_p = \frac{z_p - 1}{Tz} \Rightarrow z_p = \frac{1}{1 - Ts_p} \tag{45}$$

<u>Conclusion</u>: For the forward rule, the stability of the discrete-time system depends on

- The stability of the continuous time system through the continuous time system eigenvalues, s_p, and
- The time step T.

For the forward rule, we concluded that the dependence on T was that T had to satisfy

$$T < \frac{-2\sigma_p}{\sigma_p^2 + \omega_p^2} \tag{30}$$

The question we must answer now is: For the backwards rule, (a) does stability of the discrete-time also depend on T, and (b) if so, in what way?

Let's take the same approach by looking at how an eigenvalue s_p maps to the z-plane through the function (45). Our real question here is: what are the conditions on T to force all left-hand-plane eigenvalues of H(S) to map into the unit circle of the z-plane?

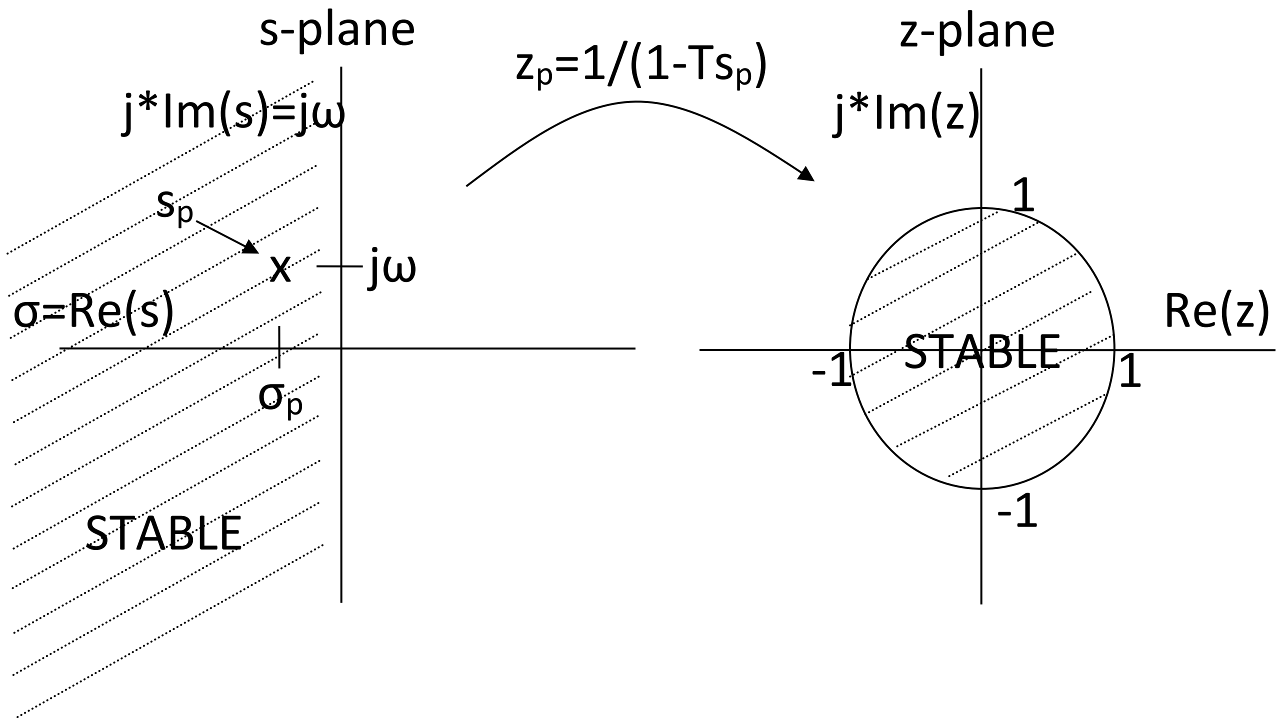


Fig. 11

From Fact 5, we recall that if all poles z_p are within the unit circle, H(z) is stable. So this implies that for stability, the poles must map to the z-plane so that

$$|z_p| < 1 \tag{19}$$

But the backwards rule corresponds to a mapping of

$$z_p = \frac{1}{1 - Ts_p} \tag{46}$$

Trick: Add and subtract ½ to the right-hand-side:

$$z_p = \frac{1}{2} + \left[\frac{1}{1 - Ts_p} - \frac{1}{2} \right] \qquad (47)$$

Getting common denominator for the term in brackets:

$$z_p = \frac{1}{2} + \left[\frac{2 - 1 + Ts_p}{2(1 - Ts_p)} \right] = \frac{1}{2} + \frac{1}{2}\left[\frac{1 + Ts_p}{1 - Ts_p} \right] \qquad (48)$$

Now substitute $s_p = \sigma_p + j\omega_p$ to obtain

$$z_p = \frac{1}{2} + \frac{1}{2}\left[\frac{1 + T\left(\sigma_p + j\omega_p\right)}{1 - T\left(\sigma_p + j\omega_p\right)} \right] \qquad (49)$$

Now collect real and imaginary terms to get

$$z_p = \frac{1}{2} + \frac{1}{2}\left[\frac{(1 + T\sigma_p) + Tj\omega_p}{(1 - T\sigma_p) - Tj\omega_p} \right] \qquad (50)$$

Now we will use this rule:

If for three vectors a, b, and c, a=b+c, then $|a| \leq |b| + |c|$.

Application of this rule to (5) results in

$$\left|z_p\right| \le \frac{1}{2} + \frac{1}{2}\frac{\sqrt{(1+T\sigma_p)^2 + (T\omega_p)^2}}{\sqrt{(1-T\sigma_p)^2 + (T\omega_p)^2}} \qquad (51)$$

Define A as the square-root term in the numerator of (51); B as the square-root term in the denominator, i.e.,

$$\begin{aligned} A &= \sqrt{(1+T\sigma_p)^2 + (T\omega_p)^2} \\ B &= \sqrt{(1-T\sigma_p)^2 + (T\omega_p)^2} \end{aligned} \qquad (52)$$

Then (51) becomes:

$$\left|z_p\right| \le \frac{1}{2} + \frac{1}{2}\frac{A}{B} \qquad (53)$$

Now consider one eigenvalue having $\sigma_P<0$, corresponding to a stable pole of the continuous time system.

Then (52) indicates that B>A>0.

Then 0<A/B<1.

Then the right-hand-side of (53) must be in (0.5,1), i.e.,

$$\left|z_p\right| \le \frac{1}{2} + \frac{1}{2}\frac{A}{B} < 1 \qquad (54)$$

And so the discrete-time-system must be stable. The implication is that numerical integration using the backwards rule of a stable continuous-time-system will result in a stable discrete-time-system, independent of choice of T.

Likewise, we can consider the case of having one eigenvalue having $\sigma_P>0$, corresponding to an unstable pole of the continuous time system. But in this case, it will not be good enough to simply show that $|z_p|$ *can be* greater than 1; we will need to show that $|z_p|$ *must be* > 1 so that we can be certain an unstable continuous time system results in an unstable discrete time system. Doing so results in

I think I started from (46) to derive (55), but need to check this…

$$\left|z_p\right| = \frac{\sqrt{1+T^2\omega_p^2}}{\sqrt{(1-T\sigma_p)^2+T^2\omega_p^2}} \qquad (55)$$

Here we can see that, for $\sigma_P>0$, if $T\sigma_P<1$, then the numerator is greater than the denominator, and $|z_p|>1$. The condition $T\sigma_P<1$ is easy to satisfy in practice, since unstable eigenvalues typically have very small real parts.

Equation (55) may also be used to verify our earlier conclusion: for $\sigma_P<0$, the denominator is always greater than the numerator, and $|z_p|<1$.

6.0 Trapezoidal rule

The trapezoidal rule is an implicit method that approximates the area under the curve of the second term below

$$x(kT) = x(kT-T) + \int_{kT-T}^{kT} f(x(k\tau))d\tau \qquad (10)$$

using the formula for the area of a trapezoid, as:

$$A = \frac{1}{2}(a+b)T \qquad (56)$$

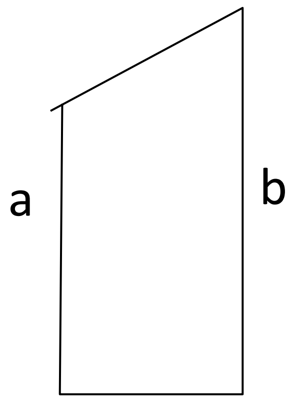


Fig. 12

Figure 13 illustrates the application of the trapezoidal method.

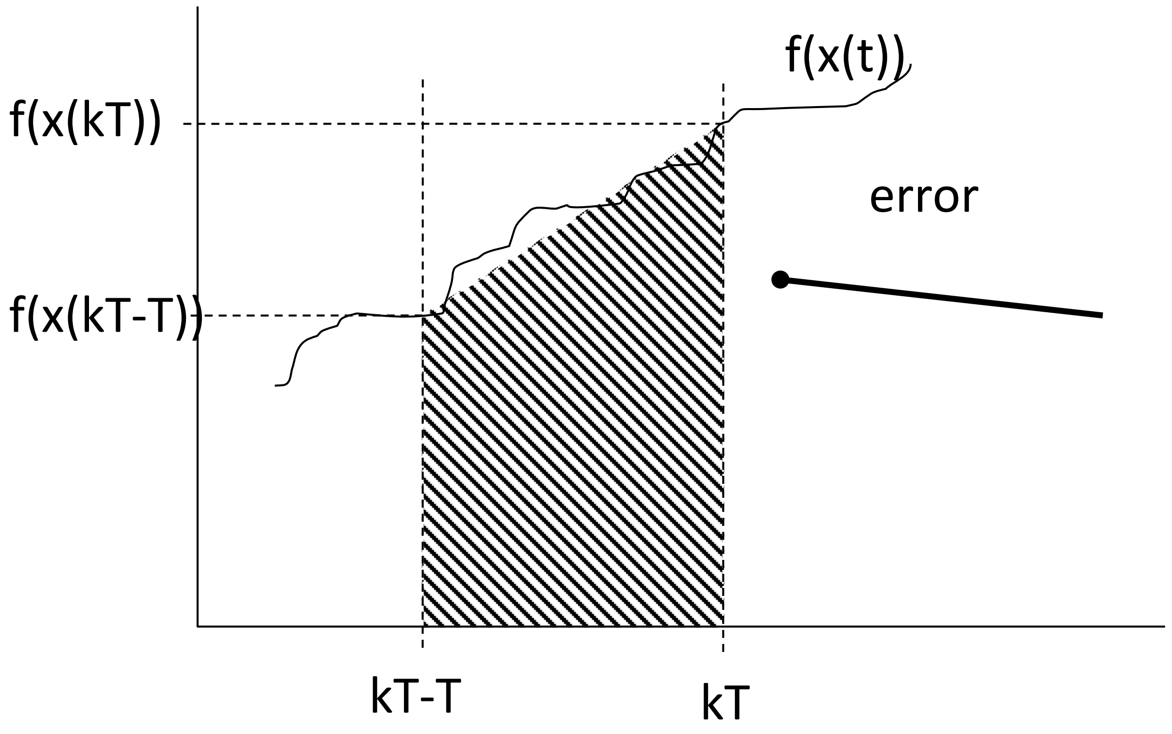


Fig. 13

Therefore (10) becomes

$$x(kT) = x(kT-T) + \frac{T}{2}\left(f\,(x(kT-T) + f\,(x(kT))\right) \tag{57}$$

Discretization using Trapezoidal integration corresponds to finding a z-domain transfer function of

$$H(z) = H(s)\big|_{s=\frac{2}{T}\frac{z-1}{z+1}} \tag{58}$$

(Tustin's method, or the bilinear transformation). This corresponds to the mapping

$$z_p = \frac{1+Ts_p/2}{1-Ts_p/2} \tag{59}$$

as indicated in Fig. 14.

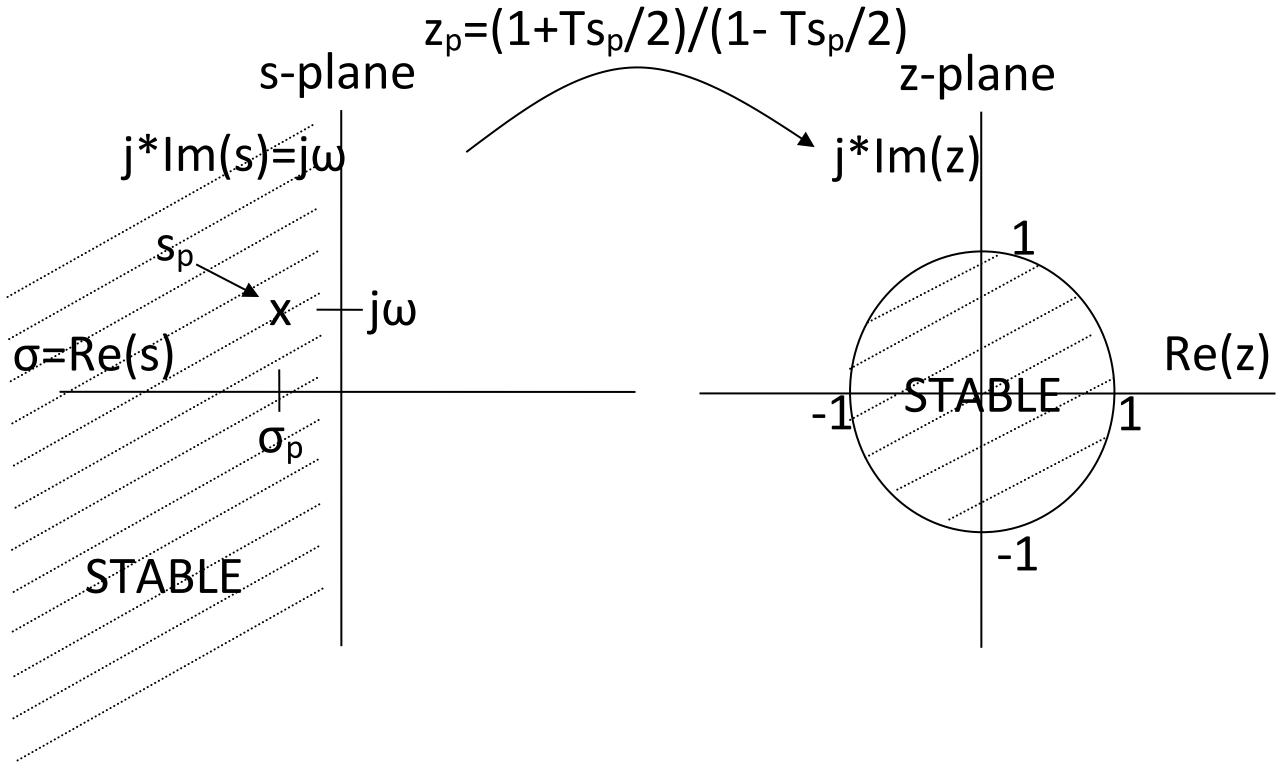


Fig. 14

We may again show that the trapezoidal integration approach is numerically stable.

Appendix: Stability Domain Analysis of Integration Methods

Assume that the general form of a differential equation is $\dot{x} = f(t,x)$. We linearize f in its neighborhood as follows.

$$\dot{x} = f(t_n, x_n) + (t - t_n)\left.\frac{\partial f}{\partial t}\right|_{(t_n, x_n)} + (x - x_n)\left.\frac{\partial f}{\partial x}\right|_{(t_n, x_n)} + \cdots \qquad (1)$$

The high order items can be omitted. Let $\lambda = \left.\frac{\partial f}{\partial x}\right|_{(t_n, x_n)}$, and we can get

$$\dot{x} = \lambda x + f(t_n, x_n) + (t - t_n)\left.\frac{\partial f}{\partial t}\right|_{(t_n, x_n)} - \lambda x_n \qquad (2)$$

If $\lambda \neq 0$, we can do transformation $\overline{x} = x + \frac{1}{\lambda}\left[f(t_n, x_n) + (t - t_n + \frac{1}{\lambda})\left.\frac{\partial f}{\partial t}\right|_{(t_n, x_n)} - \lambda x_n \right]$,

Thus (1) can be transformed to

$$\dot{\overline{x}} = \lambda \overline{x} \qquad (3)$$

Expression (3) is usually called test equation (see the definition as follows). For a set of differential equations, λ_i are the engenvalues of Jacobian matrix.

Now we apply a numerical method to expression (3), for example Forward Euler method.

$$x_{n+1} = x_n + h\lambda x_n = (1 + h\lambda)x_n = R(h\lambda)x_n = R(z)x_n \qquad (4)$$

where $z = h\lambda$.

Assume that there is disturbance δ_n on x_n, and the resulting disturbance on x_{n+1} is δ_{n+1}.

Then, $\delta_{n+1} = R(z)\delta_n$

If we want $\left|\delta_{n+1}\right| \le \left|\delta_n\right|$, we just need $\left|R(z)\right| \le 1$.

Definition[3]: The function $R(z)$ is called the ***stability function*** of the method. It can be interpreted as the numerical solution after one step for

$$\dot{x} = \lambda x, \text{ with } x_0 = 1, \qquad z = h\lambda,$$

the famous ***Dahlquist test equation***. The set

$$S = \left\{ z \in \mathbf{C}; \quad \left|R(z)\right| \le 1 \right\}$$

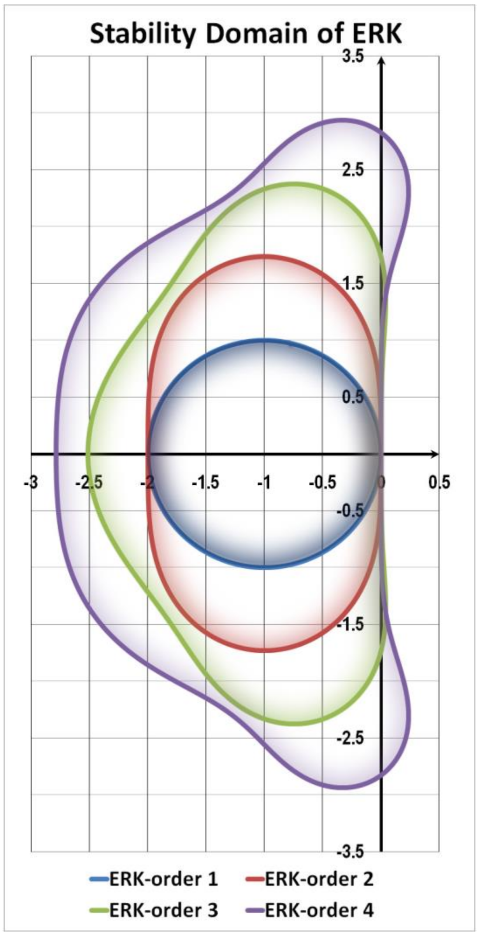
is called the ***stability domain*** of the method.

In order to analyze the numerical stability of *p*-order Explicit Runge-Kutta(ERK) method, we just need to calculate their stability domain.

Conclusion[3] : If the Explicit Runge-Kutta is of order *p*, then

$$R(z) = 1 + z + \frac{z^2}{2!} + \cdots + \frac{z^p}{p!} + O(z^{p+1})$$

The stability domain of first to fourth order Explicit Runge-Kutta methods is shown in the figure below.



Stability domain of *p*-order Explicit Runge-Kutta

Similarly, we can find stability domain of some implicit methods, as generalized in below table.

Numerical Method	Formula Expression	Stability Function
Backward Euler	$x_{n+1} = x_n + hf(x_{n+1})$	$R(z) = \dfrac{1}{1+z}$
Trapezoidal rule	$x_{n+1} = x_n + \dfrac{h}{2}\left[f(x_n) + f(x_{n+1})\right]$	$R(z) = \dfrac{1+z/2}{1-z/2}$
Implicit Midpoint rule	$x_{n+1} = x_n + hf\left[\dfrac{1}{2}(x_n + x_{n+1})\right]$	$R(z) = \dfrac{1+z/2}{1-z/2}$

For power system domain simulation, Trapezoidal rule is usually used. It can be seen from its stability function that the stability domain of Trapezoidal rule is the whole left complex domain.

---

[1] B. Stott, "Power system dynamic response calculations," Proceedings of the IEEE, Vol. 67, No. 2, Feb., 1979.

[2] 2007 presentation slides from Paul Gray, University of Northern Iowa, Henry Neeman, University of Oklahoma, Charlie Peck, Earlham College.

[3] E. Hairer, G. Wanner, Solving ordinary differential equations II: stiff and differential-algebraic problems, Springer-Verlag, 1996.