16-299 Lecture 5: Linear Systems

In these notes we are only considering time invariant systems, meaning the dynamics functions in continuous time $\mathbf{f}()$ and discrete time $\mathbf{F}()$ do not change over time.

What is a linear system?

A continuous time linear system $\mathbf{f}()$ has the property that for any $\dot{\mathbf{x}}_1 = \mathbf{f}(\mathbf{x}_1, \mathbf{u}_1)$ and any $\dot{\mathbf{x}}_2 = \mathbf{f}(\mathbf{x}_2, \mathbf{u}_2)$:

$$c_1\dot{\mathbf{x}}_1 + c_2\dot{\mathbf{x}}_2 = \mathbf{f}(c_1\mathbf{x}_1 + c_2\mathbf{x}_2, c_1\mathbf{u}_1 + c_2\mathbf{u}_2) \tag{1}$$

and can be written as

$$\dot{\mathbf{x}} = \mathbf{A}_c\mathbf{x} + \mathbf{B}_c\mathbf{u} \tag{2}$$

with $\mathbf{A}_c$ and $\mathbf{B}_c$ being constant matrices.

A discrete time linear system $\mathbf{F}()$ has the property that for any $\mathbf{x}_{\text{next}1} = \mathbf{F}(\mathbf{x}_1, \mathbf{u}_1)$ and any $\mathbf{x}_{\text{next}2} = \mathbf{F}(\mathbf{x}_2, \mathbf{u}_2)$:

$$c_1\mathbf{x}_{\text{next}1} + c_2\mathbf{x}_{\text{next}2} = \mathbf{F}(c_1\mathbf{x}_1 + c_2\mathbf{x}_2, c_1\mathbf{u}_1 + c_2\mathbf{u}_2) \tag{3}$$

and can be written as

$$\mathbf{x}_{\text{next}} = \mathbf{A}_d\mathbf{x} + \mathbf{B}_d\mathbf{u} \tag{4}$$

with $\mathbf{A}_d$ and $\mathbf{B}_d$ being constant matrices.

What are some examples of nonlinear systems?

Linearizing non-linear systems

Continuous time: Given a general continuous time non-linear system $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$, and an equilibrium point $\mathbf{0} = \mathbf{f}(\mathbf{x}_e, \mathbf{u}_e)$, we can ask what happens for small deviations from the equilibrium point $\Delta_{\mathbf{x}}$ and $\Delta_{\mathbf{u}}$? Taking derivatives:

$$\Delta_{\dot{\mathbf{x}}} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \Delta_{\mathbf{x}} + \frac{\partial \mathbf{f}}{\partial \mathbf{u}} \Delta_{\mathbf{u}} \tag{5}$$

Note that for an already linear system, $\frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \mathbf{A}_c$ and $\frac{\partial \mathbf{f}}{\partial \mathbf{u}} = \mathbf{B}_c$. The system

$$\Delta_{\dot{\mathbf{x}}} = \mathbf{A}_c \Delta_{\mathbf{x}} + \mathbf{B}_c \Delta_{\mathbf{u}} \tag{6}$$

locally approximates the behavior of the original nonlinear system about the equilibrium point.

Discrete time: Given a general discrete time non-linear system $\mathbf{x}_{\text{next}} = \mathbf{F}(\mathbf{x}, \mathbf{u})$, and an equilibrium point $\mathbf{x}_e = \mathbf{F}(\mathbf{x}_e, \mathbf{u}_e)$, we can ask what happens for small deviations from the equilibrium point $\Delta_{\mathbf{x}}$ and $\Delta_{\mathbf{u}}$? Taking derivatives:

$$\Delta_{\mathbf{x}_{\text{next}}} = \frac{\partial \mathbf{F}}{\partial \mathbf{x}} \Delta_{\mathbf{x}} + \frac{\partial \mathbf{F}}{\partial \mathbf{u}} \Delta_{\mathbf{u}} \tag{7}$$

Note that for an already linear system, $\frac{\partial \mathbf{F}}{\partial \mathbf{x}} = \mathbf{A}_d$ and $\frac{\partial \mathbf{F}}{\partial \mathbf{u}} = \mathbf{B}_d$. The system

$$\Delta_{\mathbf{x}_{\text{next}}} = \mathbf{A}_d \Delta_{\mathbf{x}} + \mathbf{B}_d \Delta_{\mathbf{u}} \tag{8}$$

locally approximates the behavior of the original nonlinear system about the equilibrium point.

When does this fail?

Eigenvalues and characteristic equations

Eigenvalues are an important way to characterize a system described by a matrix $\mathbf{A}$:

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x} \tag{9}$$

where λ is an eigenvalue. This means

$$(\mathbf{A} - \lambda\mathbf{1})\mathbf{x} = \mathbf{0} \tag{10}$$

and

$$\det(\mathbf{A} - \lambda\mathbf{1}) = 0; \tag{11}$$

This equation is known as the characteristic equation of $\mathbf{A}$, and the left hand side is known as the characteristic polynomial.

For a 2x2 matrix

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \tag{12}$$

the determinant of

$$\begin{pmatrix} a_{11} - \lambda & a_{12} \\ a_{21} & a_{22} - \lambda \end{pmatrix} \tag{13}$$

is

$$\lambda^2 - \lambda(a_{11} + a_{22}) + (a_{11}a_{22} - a_{12}a_{21}) \tag{14}$$

Solutions to this equation are given by the quadratic formula:

$$\lambda = \frac{1}{2}\left[(a_{11} + a_{22}) \pm \sqrt{4a_{12}a_{21} - (a_{11} - a_{22})^2}\right] \tag{15}$$

The role of eigenvalues of $\mathbf{A}_c$

In a previous lecture we noticed that if the discrete time dynamics matrix $\mathbf{A}_d$ had eigenvalues with magnitudes all less than one, the system was stable. What is the comparable test for a continuous time system?

For a linear continuous time dynamics with no input $\mathbf{u}$

$$\dot{\mathbf{x}} = \mathbf{A}_c\mathbf{x} \tag{16}$$

what is the solution given an initial state $\mathbf{x}_0$? In the scalar case the solution to $\dot{x} = ax$ is $x(t) = e^{at}x_0$. By analogy (and the explanation in the textbook Chapter 6), the solution to the matrix equation 16 above is $\mathbf{x}(t) = e^{\mathbf{A}_c t}\mathbf{x}_0$. What does the notation $e^{\mathbf{A}_c t}$ mean? We can do an eigenvalue decomposition: $\mathbf{A}_c = \mathbf{TDT}^{-1}$, which allows us to operate in the state space $\mathbf{z} = \mathbf{Tx}$ with the dynamics $\dot{\mathbf{z}} = \mathbf{Dz}$ and a mapping back to the original state space $\mathbf{x} = \mathbf{T}^{-1}\mathbf{z}$. With a diagonal matrix $\mathbf{D}$ for the dynamics, conceptually we can decompose the problem into independent systems. Some subsystems or modes have first order dynamics $z(t) = e^{\lambda_i t}z_0$ where λ_i is a real eigenvalue. Other subsystems or modes have pairs of complex eigenvalues $\lambda = \sigma \pm i\omega$, with solutions of the form $z_i(t) = e^{\sigma t}\sin(\omega t + \psi)$ where ψ is a phase shift.

Notation: Some fields of engineering (such as electrical engineering) use j rather than i for complex numbers.

By looking at the eigenvalues of $\mathbf{A}_c$ we can assess the stability of the system, and see what frequency the system will oscillate at. The real parts of all the eigenvalues have to be negative for the modes of $\mathbf{A}$ to decay rather than grow, and the system to be stable. The magnitude of the real part of the eigenvalues indicate how "fast" the system is. The complex part of an eigenvalue tells us the frequency that mode oscillates at.

Because the system is linear, the modes can just be added to find the total output.

What can full state feedback do?

With feedback $\mathbf{u} = -\mathbf{K}\mathbf{x}$, we can generate any eigenvalues we want, both in continuous and discrete time dynamics. Let's test this with a continuous time second order system:

$$\mathbf{A}_c = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \tag{17}$$

$$\mathbf{B}_c = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{18}$$

and

$$\mathbf{K} = \begin{pmatrix} k_1 \\ k_2 \end{pmatrix} \tag{19}$$

What is $\mathbf{A}_c - \mathbf{B}_c\mathbf{K}$?

$$\begin{pmatrix} 0 & 1 \\ -k_1 & -k_2 \end{pmatrix} \tag{20}$$

The characteristic equation is

$$\lambda^2 + \lambda k_2 + k_1 \tag{21}$$

which can have any roots (eigenvalues) we want:

$$\lambda = \frac{1}{2}\left[-k2 \pm \sqrt{k_2^2 - 4k_1}\right] \tag{22}$$

Full state feedback can place eigenvalues (which are called "poles") anywhere (in theory). This is called pole placement.

Visualizing eigenvalues (poles): The complex plain

This discussion applies to continuous time. It is useful to visualize eigenvalues on the complex plain. In thinking about the modes, first order modes have real eigenvalues that are on the real axis (horizontal). Second order modes have eigenvalues

$$\lambda = \frac{1}{2}\left[(a_{11} + a_{22}) \pm \sqrt{4a_{12}a_{21} - (a_{11} - a_{22})^2}\right] \qquad (23)$$

which can either be a pair of different real eigenvalues on the real axis, or a pair of complex eigenvalues symmetrically placed above and below the real axis. A purely oscillatory system (no damping) would have its eigenvalues on the imaginary axis.

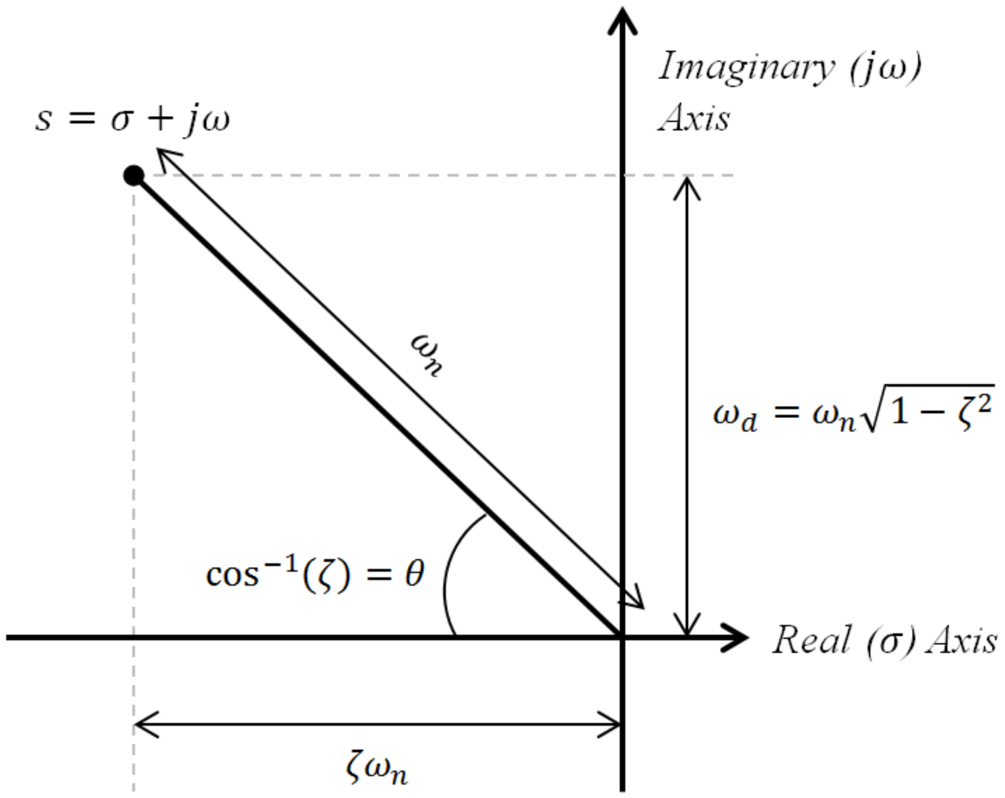
For a characteristic equation with complex roots

$$\lambda^2 + 2\gamma\omega_n\lambda + \omega_n^2 = 0 \tag{24}$$

the eigenvalues are

$$\lambda = -\gamma\omega_n + i\omega_n\sqrt{1-\gamma^2} \tag{25}$$

and γ is the damping ratio (ζ in the picture) and ω_n is the natural frequency, which serve as polar coordinates in locating the eigenvalue. ω_d is the actual frequency the system oscillates at, and $\gamma\omega_n$ is how fast the system decays or grows.


Imaginary ($j\omega$) Axis
$s = \sigma + j\omega$
ω_n
$\omega_d = \omega_n\sqrt{1-\zeta^2}$
$\cos^{-1}(\zeta) = \theta$
Real (σ) Axis
$\zeta\omega_n$

Talk about root locus plots.

Discrete time linear quadratic regulator

The linear quadratic regulator (LQR) is a design method for controllers based on optimal control. Let's consider a **linear** discrete time dynamic system

$$\mathbf{x}_{\text{next}} = \mathbf{A}_d\mathbf{x} + \mathbf{B}_d\mathbf{u} \tag{26}$$

We need to say what we want. Let's set a goal of $\mathbf{x} = 0$ and penalize deviations $\mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x}$ from the goal (a **regulator**). Let's also penalize effort $\mathbf{u}^{\mathrm{T}}\mathbf{R}\mathbf{u}$, so we can trade off performance (staying near the control) and effort (things like fuel or battery use). We want to minimize the sum of **quadratic** costs

$$L(\mathbf{x}, \mathbf{u}) = \mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} + \mathbf{u}^{\mathrm{T}}\mathbf{R}\mathbf{u} \tag{27}$$

at each time step.

To find the optimal gain matrix $\mathbf{K}$ we are going to introduce the concept of a value function $V_k(\mathbf{x})$. The value function tells us the total cost of the rest of the system's "life", starting in state $\mathbf{x}$ at time step k. This is also known as the "cost-to-go". Systems can run forever, which means the cost has to asymptote at zero or the value function is always infinite, and not useful. Systems can also run for a finite time, with a terminal penalty on the last state. We will consider the infinite time case. For the above cost function (equation 27), if the system goes to the goal, the cost goes to zero (we are assuming perfect knowledge and no disturbances).

Our strategy is to hypothesize a value function, and then show that the value function satisfies an iterative (recursive) equation. Let's hypothesize that the value function is a pure quadratic

$$V_k(\mathbf{x}) = \mathbf{x}^{\mathrm{T}}\mathbf{P}_k\mathbf{x} \tag{28}$$

This is a reasonable guess because a constant term can be ignored since it won't affect our choice of action (cost is the same for each action) and a term linear in $\mathbf{x}$ (or any odd polynomial in $\mathbf{x}$) has to be zero around the goal by symmetry. There could be additional higher order even terms, but I happen to know there won't be.

At time step $k + 1$ the value function is $V_{k+1}(\mathbf{x})$. At time step k the cost for choosing action $\mathbf{u}_k$ is the sum of the one step cost function and the value function for the next state:

$$C(\mathbf{u}_k) = L(\mathbf{x}_k, \mathbf{u}_k) + V_{k+1}(\mathbf{x}_{k+1}) \tag{29}$$

This expands out to:

$$C(\mathbf{u}_k) = \mathbf{x}_k^{\mathrm{T}}\mathbf{Q}\mathbf{x}_k + \mathbf{u}_k^{\mathrm{T}}\mathbf{R}\mathbf{u}_k + (\mathbf{A}_d\mathbf{x}_k + \mathbf{B}_d\mathbf{u}_k)^{\mathrm{T}}\mathbf{P}_{k+1}(\mathbf{A}_d\mathbf{x}_k + \mathbf{B}_d\mathbf{u}_k) \tag{30}$$

To find the optimal $\mathbf{u}_k$, we can solve

$$\frac{\partial C(\mathbf{u}_k)}{\partial \mathbf{u}_k} = 0 \tag{31}$$

because at optimal points the derivative of the cost with respect to the variables being optimized is zero (insert caveats and weasel words about smoothness and no constraints).

Notation: Some fields of engineering (such as electrical engineering) use J rather than C for total costs.

Being somewhat sloppy

$$\frac{\partial C(\mathbf{u}_k)}{\partial \mathbf{u}_k} = 2\mathbf{R}\mathbf{u}_k + 2\mathbf{B}_d^{\mathrm{T}}\mathbf{P}_{k+1}\mathbf{A}_d\mathbf{x}_k + 2\mathbf{B}_d^{\mathrm{T}}\mathbf{P}_{k+1}\mathbf{B}_d\mathbf{u}_k \tag{32}$$

so

$$\begin{aligned}\mathbf{u}_k^* = &- (\mathbf{R} + \mathbf{B}_d^{\mathrm{T}}\mathbf{P}_{k+1}\mathbf{B}_d)^{-1}\mathbf{B}_d^{\mathrm{T}}\mathbf{P}_{k+1}\mathbf{A}_d\mathbf{x}_k \\ &- \mathbf{K}_k^*\mathbf{x}_k\end{aligned} \tag{33}$$

(* means optimal).

Substituting this policy into the cost equation 30 above and dropping the subscripts k and d and the superscript $*$ for brevity results in

$$C(\mathbf{u}) = \mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} + (-\mathbf{K}\mathbf{x})^{\mathrm{T}}\mathbf{R}(-\mathbf{K}\mathbf{x}) + ((\mathbf{A} - \mathbf{B}\mathbf{K})\mathbf{x})^{\mathrm{T}}\mathbf{P}_{k+1}(\mathbf{A} - \mathbf{B}\mathbf{K})\mathbf{x} \tag{34}$$

All of this is quadratic in $\mathbf{x}$:

$$C(\mathbf{u}) = \mathbf{x}^{\mathrm{T}}(\mathbf{Q} + \mathbf{K}^{\mathrm{T}}\mathbf{R}\mathbf{K} + (\mathbf{A} - \mathbf{B}\mathbf{K})^{\mathrm{T}}\mathbf{P}_{k+1}(\mathbf{A} - \mathbf{B}\mathbf{K}))\mathbf{x} \tag{35}$$

Since $C(\mathbf{u}^*)$ is $\mathbf{x}^{\mathrm{T}}\mathbf{P}\mathbf{x}$,

$$\mathbf{P}_k = \mathbf{Q} + \mathbf{K}^{\mathrm{T}}\mathbf{R}\mathbf{K} + (\mathbf{A} - \mathbf{B}\mathbf{K})^{\mathrm{T}}\mathbf{P}_{k+1}(\mathbf{A} - \mathbf{B}\mathbf{K}) \tag{36}$$

So we have an iterative equation where on each step k we first compute the optimal $\mathbf{K}$ using equation 33 and then compute the value function for the current step using equation 36. Eventually the value function and optimal gain stop changing and the converged values are known as steady state values $\mathbf{P}_{ss}$ and $\mathbf{K}_{ss}$. This convergence still happens whatever symmetric positive definite $\mathbf{P}$ one starts with, although the time to convergence varies. $\mathbf{P}$ at all time steps is symmetric and positive definite. For finite horizon problems (N steps), $\mathbf{P}_N$ is initialized as the terminal penalty on $\mathbf{x}_N$. The finite horizon problem is typically not time invariant, so one must keep track of $\mathbf{P}_k$ and $\mathbf{K}_k$ for each time step, rather than just use the same steady state values on all time steps.

By substituting for $\mathbf{K}$ in equation 36 and algebraically manipulating the result, one can get other equivalent iterative equations:

$$\mathbf{P}_k = \mathbf{Q} + \mathbf{A}^{\mathrm{T}}\mathbf{P}_{k+1}\mathbf{A} - \mathbf{A}^{\mathrm{T}}\mathbf{P}_{k+1}\mathbf{B}\mathbf{K} \tag{37}$$

This equation tells us that the value function at time step k is incremented by three things:

1. The performance penalty $\mathbf{Q}$ acting on the current state.
2. The plant dynamics acting on the next value function $\mathbf{A}^{\mathrm{T}}\mathbf{P}_{k+1}\mathbf{A}$. The smaller the eigenvalues of $\mathbf{A}$, the quadratically smaller the contribution of the future value function is.
3. A term $-\mathbf{A}^{\mathrm{T}}\mathbf{P}_{k+1}\mathbf{B}\mathbf{K}$ that takes into account the effect of the feedback control.

Writing the above equation without using $\mathbf{K}$ and dropping all subscripts gives us the Discrete time Algebraic Ricatti Equation (DARE) `https://en.wikipedia.org/wiki/Algebraic_Riccati_equation`:

$$\mathbf{P} = \mathbf{Q} + \mathbf{A}^{\mathrm{T}}\mathbf{P}\mathbf{A} - \mathbf{A}^{\mathrm{T}}\mathbf{P}\mathbf{B}(\mathbf{R} + \mathbf{B}^{\mathrm{T}}\mathbf{P}\mathbf{B})^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{P}\mathbf{A} \tag{38}$$

This iteration is a form of dynamic programming. Dynamic programming is an efficient way to optimize a policy/control law by stepping backwards in time, and representing "future" costs in a value function. It is not necessary to take into account future states, actions, or policies in optimizing the current step, as that information is represented by the value function. The iterative equation is typically expressed as:

$$\begin{aligned} V_k(\mathbf{x}_k) &= \min_{\mathbf{u}_k} \left[L(\mathbf{x}_k, \mathbf{u}_k) + V_{k+1}(\mathbf{x}_{k+1})\right] \\ &= \min_{\mathbf{u}_k} \left[L(\mathbf{x}_k, \mathbf{u}_k) + V_{k+1}(\mathbf{F}(\mathbf{x}_k, \mathbf{u}_k))\right] \end{aligned} \tag{39}$$

and this equation is known as the Bellman or Hamilton-Jacobi equation. To optimize the cost at the current step, one must optimize the sum of the cost of the current step and the future cost (cost-to-go) of the resulting next state.

To apply dynamic programming, costs need to be separable across time steps. What does this mean? We can have a one step cost function which can be time variant $L_k(\mathbf{x}_k, \mathbf{u}_k)$, but we can't have an arbitrary multi-step cost function

$$L(\mathbf{x}_k, \mathbf{x}_{k+1}, \cdots, \mathbf{x}_{k+n}, \mathbf{u}_k, \mathbf{u}_{k+1}, \cdots, \mathbf{u}_{k+n}) \tag{40}$$

because then the future paths and actions actually taken would need to be considered on each time step, not just the optimal cost encoded in the value function.

LQR can easily handle time varying everything ($\mathbf{A}_k$, $\mathbf{B}_k$, $\mathbf{Q}_k$, and $\mathbf{R}_k$) resulting in time varying value functions $\mathbf{P}_k$ and control laws $\mathbf{K}_k$. In this case there is usually no steady state solution. Periodic systems can result in repeating the same trajectory of value functions and control laws on each cycle.

LQR can easily handle varying goals on each time step:

$$L_k(\mathbf{x}_k, \mathbf{u}_k) = (\mathbf{x}_k - \mathbf{x}_{dk})^{\mathrm{T}}\mathbf{Q}_k(\mathbf{x}_k - \mathbf{x}_{dk}) + \mathbf{u}_k^{\mathrm{T}}\mathbf{R}_k\mathbf{u}_k \tag{41}$$

and can be used to do trajectory optimization, finding optimal sequences $\mathbf{x}_k$ and $\mathbf{K}_k$. A fast but crude way (iLQR) to do trajectory optimization for nonlinear systems is to

1. Given the current sequence of policies, simulate the trajectory forward in time. This takes into account the full nonlinear dynamics.

2. Linearize about the current trajectory, creating a sequence of $\mathbf{A}_k$ and $\mathbf{B}_k$

3. Use LQR to do trajectory optimization, finding a new set of optimal policies on each step.

4. Loop

A more accurate but more expensive approach (Differential Dynamic Programming or DDP) is to try to model the nonlinear dynamics with local 2nd order models:

$$\begin{aligned}\mathbf{F}(\mathbf{x} + \Delta_{\mathbf{x}}, \mathbf{u} + \Delta_{\mathbf{u}}) \approx & \mathbf{F}_0 + \frac{\partial \mathbf{F}}{\partial \mathbf{x}}\Delta_{\mathbf{x}} + \frac{\partial \mathbf{F}}{\partial \mathbf{u}} + \Delta_{\mathbf{x}}^{\mathrm{T}}\frac{\partial^2 \mathbf{F}}{\partial^2 \mathbf{x}}\Delta_{\mathbf{x}} + \Delta_{\mathbf{x}}^{\mathrm{T}}\frac{\partial^2 \mathbf{F}}{\partial \mathbf{x}\partial \mathbf{u}} + \Delta_{\mathbf{u}}^{\mathrm{T}}\frac{\partial^2 \mathbf{F}}{\partial^2 \mathbf{u}}\Delta_{\mathbf{u}} \\ & \mathbf{F}_0 + \mathbf{A} + \mathbf{B} + \frac{\partial \mathbf{A}}{\partial \mathbf{x}} + \frac{\partial \mathbf{A}}{\partial \mathbf{u}} + \frac{\partial \mathbf{B}}{\partial \mathbf{x}} + \frac{\partial \mathbf{B}}{\partial \mathbf{u}}\end{aligned}$$

(42)

After all, the whole point of acknowledging the nonlinear dynamics is that a linear approximation wasn't adequate. The modified iterative algorithm is

1. Given the current sequence of policies, simulate the trajectory forward in time. This takes into account the full nonlinear dynamics. (Same as before).
2. Create local 2nd order models along the trajectory, creating a sequence of $\mathbf{A}_k$, $\mathbf{B}_k$, $\partial\mathbf{A}_k/\partial\mathbf{x}$, $\partial\mathbf{A}_k/\partial\mathbf{u}$, $\partial\mathbf{B}_k/\partial\mathbf{x}$, and $\partial\mathbf{B}_k/\partial\mathbf{u}$.
3. Use DDP to do trajectory optimization, finding a new set of optimal policies on each step. Google "Differential Dynamic Programming" to get the details.
4. Loop

Continuous time LQR

Impulse responses and convolution

Frequency response

s and z plain

LQG