Lecture 1, Sep 8, 2023

Lecture 2, Sep 13, 2023

System Modeling (Continuous and Discrete Time)

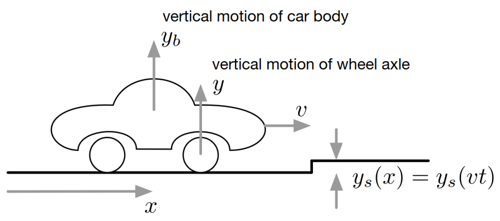


Figure 1: Example 1: Modeling the wheel motion of a car.

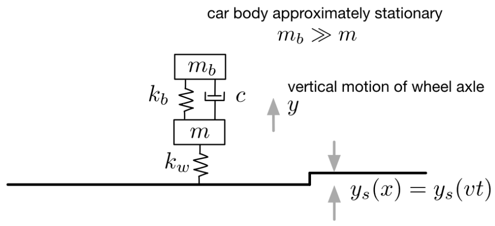

car body approximately stationary
$m_b \gg m$
m_b
k_b
c
vertical motion of wheel axle
y
m
k_w
$y_s(x) = y_s(vt)$

Figure 2: Modeling the example as a simple mechanical system.

- Example 1: modeling the wheel motion of a car when it encounters a bump in the road
  - Assumptions: constant velocity, 2D system model, variable step height $y_s(x)$
  - Begin with a simple mechanical system:
    * k_w is a spring representing the wheel; m is the mass of the wheel axle
    * The spring-damper system k_b and c model the shock absorber in the car
    * In addition, we assume $m_b \gg m$, so that the car itself is approximately stationary and only the wheel axle moves; we also assume the suspension is 1D and that the car is at rest in the vertical direction before we hit the bump
  - Now we can use Newton's second law to form a mathematical model:
    * $m\ddot{y} = \sum f = -c\dot{y} - k_b y - k_w(y - y_s)$
    * $m\ddot{y} + c\dot{y} + (k_b + k_w)y = m\ddot{y} + c\dot{y} + ky = k_w y(s) = u(t)$
      - This is a second order, linear, nonhomogeneous, time-invariant system
      - The initial conditions are $y(0) = \dot{y}(0) = 0$ and we wish to find $y, \dot{y}, \ddot{y}$ for $t \geq 0$
    * Now we need to represent it in a standard form
      - $\begin{bmatrix} \dot{y} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\dfrac{k}{m} & -\dfrac{c}{m} \end{bmatrix} \begin{bmatrix} y \\ \dot{y} \end{bmatrix} + \begin{bmatrix} 0 \\ \dfrac{1}{m} \end{bmatrix} u$
      - This is now in standard form: $\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}u, \boldsymbol{x}(0) = \begin{bmatrix} y(0) \\ \dot{y}(0 \end{bmatrix}$
      - This form corresponds to the following simulation block diagram:

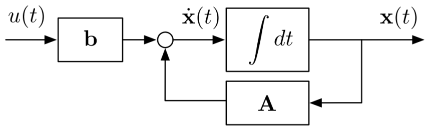


Figure 3: Simulation block diagram of a standard linear system.

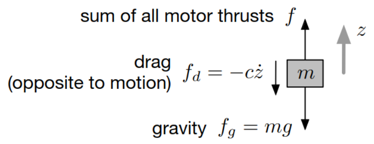

sum of all motor thrusts f
drag (opposite to motion) $f_d = -c\dot{z}$
m
z
gravity $f_g = mg$

Figure 4: Simple mechanical system for Example 2.

- Example 2: modelling how a drone reacts to given motor inputs
  - Assumptions: horizontal motion is stabilized
  - Applying Newton's laws:
    - $m\ddot{z} = -mg - c\dot{z} + f$
    - We again have a second order, linear, nonhomogeneous, time-invariant system
  - In standard form: $\begin{bmatrix}\dot{z}\\ \ddot{z}\end{bmatrix} = \begin{bmatrix}0 & 1\\ 0 & -\dfrac{c}{m}\end{bmatrix}\begin{bmatrix}z\\ \dot{z}\end{bmatrix} + \begin{bmatrix}0\\ \dfrac{1}{m}\end{bmatrix}u$
  - But how do we actually perform the simulation?

Summary

In general for any linear system we have: a set of inputs $\boldsymbol{u}(t)$ (which we partially control); a set of outputs $\boldsymbol{y}(t)$ which we can measure; and states $\boldsymbol{x}(t)$ that are internal to the system which we cannot directly manipulate or measure.
To model a linear system in continuous time:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t), \boldsymbol{x}(0) = \boldsymbol{x}_0$$
$$\boldsymbol{y}(t) = \boldsymbol{C}\boldsymbol{x}(t) + \boldsymbol{D}\boldsymbol{u}(t)$$

where $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}$ are *state matrices*.
To model a nonlinear system in continuous time:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t)), \boldsymbol{x}(0) = \boldsymbol{x}_0$$
$$\boldsymbol{y}(t) = \boldsymbol{h}(\boldsymbol{x}(t), \boldsymbol{u}(t))$$

- If we have n states, m inputs and p outputs, then $\boldsymbol{A}$ is $n \times n$, $\boldsymbol{B}$ is $n \times m$, $\boldsymbol{C}$ is $p \times n$ and $\boldsymbol{D}$ is $p \times m$
- In practice, we often need to represent things in discrete time, since the computers running simulations are discrete
- To represent things in discrete time, we replace continuous signals with a sequence of regular samples at $t_k = kh$
  - $t_k = kh$ is the *sampling time*
  - $f_s = \dfrac{1}{h}$ is the *sampling frequency*
- So how do we convert our continuous model to a discrete one?
  - Recall that the solution for $\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}$ is $\boldsymbol{x} = e^{\boldsymbol{A}t}\boldsymbol{x}_0$
  - Over a short time interval $t_k \leq t \leq t_{k+1}$ the solution evolves as:
    - $\begin{aligned}\boldsymbol{x}(t) &= e^{\boldsymbol{A}(t-t_k)}\boldsymbol{x}(t_k) + \int_{t_k}^{t} e^{\boldsymbol{A}(t-\tau)}\boldsymbol{B}\boldsymbol{u}(\tau)\,\mathrm{d}\tau\\ &= e^{\boldsymbol{A}(t-t_k)}\boldsymbol{x}(t_k) + \int_{t_k}^{t} e^{\boldsymbol{A}(t-\tau)}\boldsymbol{B}\,\mathrm{d}\tau\boldsymbol{u}(t_k)\\ &= \boldsymbol{\Phi}(t, t_k)\boldsymbol{x}(t_k) + \Gamma(t, t_k)\boldsymbol{u}(t_k)\end{aligned}$
      - Note that we have assumed $\boldsymbol{u}(\tau) = \boldsymbol{u}(t_k)$ (i.e. $\boldsymbol{u}$ stays constant over the timestep), which is referred to as a *zero-order hold*
    - $\boldsymbol{x}(t_{k+1}) = \boldsymbol{\Phi}(t_{k+1}, t_k)\boldsymbol{x}(t_k) + \Gamma(\boldsymbol{t}_{k+1}, t_k)\boldsymbol{u}(t_k) = \boldsymbol{A}_d\boldsymbol{x}(t_k) + \boldsymbol{B}_d\boldsymbol{u}(t_k)$
      - We have discretized the system

- We now have difference equations for the system (h is the sampling period):
  - $\boldsymbol{x}_{k+1} = \boldsymbol{A}_d\boldsymbol{x}_k + \boldsymbol{B}_d\boldsymbol{u}_k$
  - $\boldsymbol{y}_k = \boldsymbol{C}\boldsymbol{x}_k + \boldsymbol{D}\boldsymbol{u}_k$
  - $\boldsymbol{A}_d = \boldsymbol{\Phi}(t_{k+1}, t_k) = e^{\boldsymbol{A}h}$
  - $\boldsymbol{B}_d = \Gamma(t_{k+1}, t_k) = \int_0^h e^{\boldsymbol{A}\tau'}\,\mathrm{d}\tau'\boldsymbol{B}$
- To solve for $\boldsymbol{A}_d$ and $\boldsymbol{B}_d$:
  - Note that $\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{\Phi}(t) = \boldsymbol{\Phi}(t)\boldsymbol{A}$ and $\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{\Gamma}(t) = \boldsymbol{\Phi}(t)\boldsymbol{B}$
  - Using this we have: $\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}\boldsymbol{\Phi}(t) & \boldsymbol{\Gamma}(t) \\ \boldsymbol{0} & \boldsymbol{I}\end{bmatrix} = \begin{bmatrix}\boldsymbol{\Phi}(t) & \boldsymbol{\Gamma}(t) \\ \boldsymbol{0} & \boldsymbol{I}\end{bmatrix}\begin{bmatrix}\boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{0} & \boldsymbol{0}\end{bmatrix}$
    - Now we can use another matrix exponential to solve this
  - $\begin{bmatrix}\boldsymbol{A}_d & \boldsymbol{B}_d \\ \boldsymbol{0} & \boldsymbol{I}\end{bmatrix} = \exp\left(\begin{bmatrix}\boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{0} & \boldsymbol{0}\end{bmatrix}h\right)$

Note

The matrix exponential can be calculated with the Matlab function `expm()` or scipy function `scipy.linalg.expm()`. We can also manually do a series expansion of the matrix exponential function. Alternatively, `c2d()` in Matlab or `control.matlab.c2d()` can be used to do the same conversion (which computes the matrix exponentials internally).

- We can now recursively apply the difference equations to propagate the state:
  - $\boldsymbol{x}_1 = \boldsymbol{A}_d\boldsymbol{x}_0 + \boldsymbol{B}_d\boldsymbol{u}_0$
  - $\boldsymbol{x}_2 = \boldsymbol{A}_d(\boldsymbol{A}_d\boldsymbol{x}_0 + \boldsymbol{B}_d\boldsymbol{u}_0) + \boldsymbol{B}_d\boldsymbol{u}_1$
  - $\boldsymbol{x}_3 = \boldsymbol{A}_d(\boldsymbol{A}_d^2\boldsymbol{x}_0 + \boldsymbol{A}_d\boldsymbol{B}_d\boldsymbol{u}_0 + \boldsymbol{B}_d\boldsymbol{u}_1) + \boldsymbol{B}_d\boldsymbol{u}_2$ and so on
  - We can stack all these together: $\begin{bmatrix}\boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_N\end{bmatrix} = \boldsymbol{F}\begin{bmatrix}\boldsymbol{u}_0 \\ \boldsymbol{u}_1 \\ \vdots \\ \boldsymbol{u}_{N-1}\end{bmatrix} + \boldsymbol{F}_0\boldsymbol{x}_0$

Definition

A system is *reachable* or *controllable* if and only if

$$\operatorname{rank}\begin{bmatrix}\boldsymbol{A}_d^{N-1}\boldsymbol{B}_d & \boldsymbol{A}_d^{N-2}\boldsymbol{B}_d & \cdots & \boldsymbol{B}_d\end{bmatrix} = N$$

where N is the dimension of the state $\boldsymbol{x}_k$. Physically this means that there is a given sequence of control inputs to reach any state.

Definition

A system is *observable* if and only if

$$\operatorname{rank}\begin{bmatrix}\boldsymbol{C} \\ \boldsymbol{C}\boldsymbol{A}_d \\ \vdots \\ \boldsymbol{C}\boldsymbol{A}_d^{N-1}\end{bmatrix} = N$$

Physically this means that given some sequence of outputs, we can derive the system state.

- Example: consider the system: $\dot{x}(t) = u(t), y(t) = x(t), x(0) = x_0$; find the difference equations assuming a zero-order hold at the input and sampled output, with sampling period h
  - Note that we have $\boldsymbol{A} = 0$ and $\boldsymbol{B} = 1$

  - For $t_k \leq t \leq t_{k+1}$:
    * $x(t) = x(t_k) + \int_{t_k}^{t} u(\tau)\,\mathrm{d}\tau$
      $= x(t_k) + \int_{t_k}^{t} 1\,\mathrm{d}\tau u(t_k)$
      $= x(t_k) + (t - t_k)u(t_k)$
    * Applying this at $t = t_{k+1} = t_k + h$ we get $x_{k+1} = x_k + hu_k$
- Example: $\ddot{x} = u$
  - State: $\boldsymbol{z} = \begin{bmatrix} x \\ \dot{x} \end{bmatrix}$
  - Continuous time: $\dot{\boldsymbol{z}} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \boldsymbol{z} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u = \boldsymbol{A}\boldsymbol{z} + \boldsymbol{B}u$
  - $\begin{bmatrix} \boldsymbol{A}_d & \boldsymbol{B}_d \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} = \exp\left(\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} h \right) = \exp\left(\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} h \right) = \exp(\boldsymbol{E}h)$
  - Notice that $\boldsymbol{E}^2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ and $\boldsymbol{E}^3 = \boldsymbol{0}$ so $\boldsymbol{E}$ is nilpotent
  - Using the series expansion: $\exp(\boldsymbol{E}h) = \boldsymbol{I} + \boldsymbol{E}h + \frac{1}{2}\boldsymbol{E}^2h^2 = \begin{bmatrix} 1 & h & \frac{1}{2}h^2 \\ 0 & 1 & h \\ 0 & 0 & 1 \end{bmatrix}$
  - Therefore $\boldsymbol{A}_d = \begin{bmatrix} 1 & h \\ 0 & 1 \end{bmatrix}, \boldsymbol{B}_d = \begin{bmatrix} \frac{1}{2}h^2 \\ h \end{bmatrix}$
  - $\boldsymbol{z}_{k+1} = \begin{bmatrix} 1 & h \\ 0 & 1 \end{bmatrix} \boldsymbol{z}_k + \begin{bmatrix} \frac{1}{2}h^2 \\ h \end{bmatrix} u_k$
    * Notice that if we substitute the definition of $\boldsymbol{z}$, we get the simple kinematic equations $x_{k+1} = x_k + hv_k + \frac{1}{2}h^2u_k, v_{k+1} = v_k + hu_k$
    * Assuming a zero-order hold, we can see that this is exact

Summary

Given some linear continuous system given by

$$\begin{aligned} \dot{\boldsymbol{x}}(t) &= \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t), \boldsymbol{x}(0) = \boldsymbol{x}_0 \\ \boldsymbol{y}(t) &= \boldsymbol{C}\boldsymbol{x}(t) + \boldsymbol{D}\boldsymbol{u}(t) \end{aligned}$$

we can discretize it to find discrete state matrices $\boldsymbol{A}_d$ and $\boldsymbol{B}_d$ so that the system can be equivalently modelled discretely by:

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{A}_d\boldsymbol{x}_k + \boldsymbol{B}_d\boldsymbol{u}_k \\ \boldsymbol{y}_k &= \boldsymbol{C}\boldsymbol{x}_k + \boldsymbol{D}\boldsymbol{u}_k \end{aligned}$$

which is an exact model with the assumption of a zero-order hold on the input. The matrices $\boldsymbol{A}_d, \boldsymbol{B}_d$ can be found by the matrix exponential

$$\begin{bmatrix} \boldsymbol{A}_d & \boldsymbol{B}_d \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} = \exp\left(\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} h \right)$$

where h is the size of each discrete time step.

Lecture 3, Sep 15, 2023

Numerical Methods and Stability

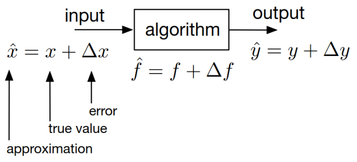


Figure 5: Model of errors in a numerical algorithm.

- Any numerical system will accumulate errors in a variety of ways; how can we quantify and evaluate these errors?
- Some common sources of error are:
  - Rounding/truncation errors, due to finite precision
  - Approximate numerical algorithms, in which we simplify complex models for efficiency
  - Input error, where the inputs to our algorithm are the outputs of an upstream algorithm which itself has errors
  - Modelling errors, where the model itself is a simplified representation of the real world (e.g. discretization)
- These errors can be categorized into two general sources: input error (Δx) and algorithm/approximation error (Δf) which combine to result in an output error (Δy)
- *Absolute* errors are simply the absolute value of the error, $|\Delta x|$; *relative* errors are the absolute errors divided by the parameter, $\delta\dfrac{|\Delta x|}{x}$
  - If the true values of x, f, y are not known, we cannot compute the relative error and might have to settle for an upper bound instead

Well- and Ill-Conditioned Problems

- First we will consider what happens when we have an ideal algorithm with some input error
- Intuitively, a problem is *well-conditioned* if, assuming an ideal algorithm ($\Delta f = 0$), the input error does not grow when propagated through the algorithm, i.e. $\Delta\bar{y} < \Delta x$
  - Consider the Taylor expansion: $f(x + \Delta x) = f(x) + \dfrac{\mathrm{d}f}{\mathrm{d}x}\Delta x + O(\Delta x^2) = y + \Delta\bar{y}$
  - $\dfrac{\Delta\bar{y}}{y} = \dfrac{1}{y}\left(f(x) + \dfrac{\mathrm{d}f}{\mathrm{d}x}\cdot x\dfrac{\Delta x}{x} + O(\Delta x^2)\right) = \dfrac{\mathrm{d}f}{\mathrm{d}x}\dfrac{x}{f(x)}\cdot\dfrac{\Delta x}{x} + O(\Delta x^2) = K_x\delta x + O(\Delta x^2)$
  - If $|\delta x| \leq \varepsilon$ then $|\delta y| \approx |K_x\delta x| \leq |K_x|\varepsilon$, so given a bound on δx we can find a bound on δy

Definition

The *absolute condition number* is defined as

$$\text{Cond}_x = \left|\frac{\Delta\bar{y}}{\Delta x}\right| \approx \left|\frac{\mathrm{d}f}{\mathrm{d}x}\right|$$

The *relative condition number* is defined as

$$\text{cond}_x = \left|\frac{\delta\bar{y}}{\delta x}\right| \approx |K_x| = \left|\frac{\mathrm{d}f}{\mathrm{d}x}\cdot\frac{x}{f(x)}\right|$$

(note, multiplication not differentiation)

A problem is *well-conditioned*:
1. If and only if Cond_x is small (using absolute error)
2. If and only if $\text{cond}_x \leq 1$ (using relative error)

- We typically want the absolute condition number to be small (but how small depends on the problem), and we want the relative condition number to be less than 1; so most of the time the relative condition number is used since it is easier to interpret
- Conditioning is a property of the *problem*, not a particular algorithm (since we assumed a perfect algorithm to begin with)
- Example: linear function: $y = ax$
  - $\frac{\mathrm{d}f}{\mathrm{d}x} = a \implies K_x = \frac{\mathrm{d}x}{\mathrm{d}f} = a\frac{x}{ax} = 1$
  - The condition number is 1, so the relative error stays the same and the problem is well-conditioned
  - The absolute error is smaller than the absolute input error if $\left|\frac{\mathrm{d}f}{\mathrm{d}x}\right| = |a| < 1$
    - Intuitively, for a steeper function the error will get bigger, but for a smaller slope the error is smaller
- Example: linear equation: find y such that $ay + b = 0$ where b is the input and a is fixed
  - $y = -\frac{b}{a} = f(b)$
  - $\frac{\mathrm{d}f}{\mathrm{d}b} = -\frac{1}{a} \implies K_b = \frac{\mathrm{d}f}{\mathrm{d}b}\cdot\frac{1}{f(b)}b = -\frac{1}{a}\cdot -\frac{a}{b}\cdot b = 1$
  - This is another well-conditioned problem
  - $\left|\frac{\mathrm{d}f}{\mathrm{d}b}\right| = \left|\frac{1}{a}\right|$ so for small a, the absolute condition number is large
- Example: differential equation: find y such that $\dot{y} = (\lambda + \Delta\lambda)y, y(0) = y_0$ where λ is the input
  - $\hat{y}(t) = y_0e^{(\lambda+\Delta\lambda)t} = y_0e^{\lambda t}e^{\Delta\lambda t} \approx y_0e^{\lambda t} + y_0te^{\lambda t}\cdot\Delta\lambda = y + \Delta\bar{y}$
    - Note we used $e^x \approx 1 + x$
  - The absolute condition number is $\frac{\Delta\bar{y}}{\Delta\lambda} = y_0te^{\lambda t}$
  - $\lim_{t\to\infty}\left|\frac{\Delta\bar{y}}{\Delta\lambda}\right| = \begin{cases} 0 & \lambda < 0 \\ \infty & \lambda \geq 0 \end{cases}$
  - This shows that asymptotically stable differential equations (DEs that approach some fixed value) are well-conditioned
    - Exercise: what we think of the error as being on the initial condition? $\dot{y} = \lambda y, y(0) = y_0 + \Delta y_0$
- Example: root finding: find y such that $g(y) = 0$
  - y is the output, but the input is hard to define since it is a function
  - We can think of it as $g(y) = 0, g(y + \Delta\bar{y}) + \varepsilon = 0$ so ε is an “input” representing an additive error in g (see diagram above); if we shift g up or down, the location of the root y will also change by an amount

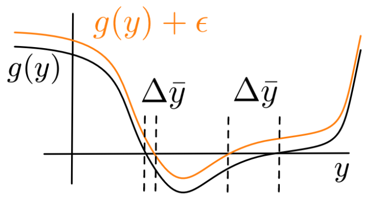


Figure 6: Representation of error in root finding.

  - Taylor series expand: $g(y)+g'(y)\Delta\bar{y}+\varepsilon = g'(y)\Delta\bar{y}+\varepsilon \approx 0 \implies \left|\frac{\Delta\bar{y}}{\varepsilon}\right| \approx \frac{1}{|g'(y)|}$
  - We find that the absolute condition number is inversely proportional to slope; notice that for the same shift, the left root with a larger slope is shifted a lot less

Order and Consistency

- We shall now consider what happens when we have an algorithmic error, while the input is ideal

Definition

Let $\varphi(x,\Delta)$ represent some approximate algorithm that models $f(x)$, where Δ are the parameters of the algorithm; φ's accuracy is of *order p* (alternatively, φ is $O(\Delta^p)$) if

$$\tilde{y} = \varphi(x,\Delta) - f(x) \propto \Delta^p$$

φ is *consistent* if

$$\lim_{\Delta\to 0} \varphi(x,\Delta) = f(x)$$

i.e. the approximate algorithm φ approaches the real model as the parameter approaches zero.

- The parameters can be e.g. a step size for numerical ODE solving; note that we assume that Δ is typically smaller than 1
- Example: numerical differentiation: $y = f'(x), \varphi(x,\Delta) = \dfrac{f(x+\Delta)-f(x)}{\Delta}$
  - Taylor expansion: $\varphi(x,\Delta) = \dfrac{f(x)+f'(x)\Delta+f''(x)\frac{\Delta^2}{2}+\cdots-f(x)}{\Delta} = f'(x)+f''(x)\dfrac{\Delta}{2}+\cdots$
  - Therefore $\varphi(x,\Delta)-f'(x) = f''(x)\dfrac{\Delta}{2}+f'''(x)\dfrac{\Delta^2}{6}+\cdots$ so φ is $O(\Delta)$, i.e. order 1

Stability and Convergence

- If we have both an input error and an approximate algorithm, the errors compound
- $\Delta y = \varphi(x+\Delta x,\Delta) - f(x) = \underbrace{(\varphi(x+\Delta x,\Delta)-\varphi(x,\Delta))}_{\Delta\tilde{y}} + \underbrace{(\varphi(x,\Delta)-f(x))}_{\tilde{y}} = \Delta\tilde{y}+\tilde{y}$
  - $\Delta\tilde{y}$ is the result of our input error, and $\tilde{y}$ is the result of our approximate algorithm
- The propagated error is $\Delta\tilde{y} = \varphi(x+\Delta x,\Delta)-\varphi(x,\Delta) \approx \dfrac{\mathrm{d}\varphi}{\mathrm{d}x}\cdot\Delta x$

Definition

φ is *numerically stable* if the ratio

$$\left|\frac{\Delta\tilde{y}}{\Delta x}\right| = \left|\frac{\mathrm{d}\varphi}{\mathrm{d}x}\right| < 1$$

If this ratio is greater than 1, then φ is *unstable*; if the ratio is exactly 1, then φ is *marginally stable*.

- The idea is that if you iteratively apply the algorithm, each time the error will be multiplied by this ratio; therefore a numerically stable algorithm will decrease in error, but an unstable algorithm will increase

Theorem

The numerical solution $\hat{y} = \varphi(\hat{x}, \Delta)$ with input $\hat{x} = x + \Delta x$ *converges* for $\Delta \to 0$ towards the exact solution $y = f(x)$ if

1. $\varphi(x, \Delta)$ is consistent, i.e. $\lim\limits_{\Delta \to 0} \varphi(x, \Delta) = f(x)$
2. $\varphi(x, \Delta)$ is at least marginally stable for $\Delta \to 0$, i.e. $\lim\limits_{\Delta \to 0} \left| \dfrac{\mathrm{d}\varphi}{\mathrm{d}x} \right| \leq 1$

Lecture 4, Sep 20, 2023

Numerical Root Finding

- Root finding involves finding x^* given f such that $f(x^*) = 0$
  - One application is optimization problems where we find the roots of the derivative
- Regularizing assumptions are used to make the problem well-posed; some common assumptions are, in increasing order of restrictiveness:
  - Continuity: f is continuous if $f(x) \to f(z)$ as $x \to z$ for all z
  - Lipschitz continuity: f is Lipschitz continuous if there exists a constant c such that $|f(x) - f(z)| \leq c|x - z|$ for all $x, z \in \mathbb{R}$
    - This is like saying that the derivative is bounded
    - e.g. $\dfrac{1}{x}$ is not Lipschitz continuous
  - Differentiability: f is differentiable if $f'(x)$ exists for all x; $f \in C^m$ if m derivatives exist and are continuous; $f \in C^\infty$ if all derivatives exist and are continuous
- Generally, the more assumptions we can make, the faster our algorithm becomes

Definition

The *convergence rate* of an algorithm is the rate at which the error decreases; formally, the convergence rate (aka order) of an algorithm is r if

$$\lim_{k \to \infty} \frac{|E_{k+1}|}{|E_k|^r} = C$$

where C is a constant.

- The first algorithm we will look at is *bisection*, which relies on the intermediate value theorem; if we have a continuous f and two points, where f is positive at one test point and negative at the other, we know there must be a root between the test points
  - The algorithm:
    1. Initialization: start with bracketing guesses l_0, r_0 such that $f(l_0)f(r_0) < 0$ (that is, $\operatorname{sgn} f(l_0) \neq \operatorname{sgn} f(r_0)$)
       - These points are typically picked based on some prior knowledge of the function
    2. Iterative update: bisect the current interval by computing $c = \dfrac{l_k + r_k}{2}$
       - If $f(c) \cdot f(l_k) < 0$, then $r_{k+1} = c, l_{k+1} = l_k$
       - If $f(c) \cdot f(r_k) < 0$, then $r_{k+1} = r_k, l_{k+1} = c$
       - In other words, change the bound that has the same sign as c to be c
    3. Iterate until $|r_k - l_k| < \epsilon$ and return c
  - For bisection $E_k = |r_k - l_k|$, which is decreased by a factor of $\dfrac{1}{2}$ for each iteration, therefore

$\frac{|E_{k+1}|}{|E_k|} = \frac{1}{2}$ and so bisection has linear convergence (order 1)

  - In summary, bisection is easy to implement and guaranteed to converge on any continuous f, but finding the initial bracket l_0, r_0 can be challenging
- With more assumptions, we can use fixed-point iteration
  - Assume that there exists some update algorithm g such that $g(x^*) = x^*$, so $f(x^*) = g(x^*) - x^* = 0$; this update function will bring us closer to the root if it converges
    * This update is problem-dependent, but has to be Lipschitz continuous for convergence
    * Example: for $f(x) = e^{\frac{1}{2}x} - x - 2$
      - $g_1(x) = e^{\frac{1}{2}x} - 2$
      - $e^{\frac{1}{2}x} = x + 2 \implies \frac{1}{2}x = \ln(x+2) \implies -x + 2\ln(x+2) = 0$
      - $g_2(x) = 2\ln(x+2)$
  - The algorithm:
    1. Start with an initial guess x_0
    2. Update $x_{k+1} = g(x_k)$
    3. Iterate until $|x_{k+1} - x_k| < \epsilon$, then return x_{k+1}
  - This is called a "fixed-point" algorithm because if $x_k = x^*$ then all $x_{k+n} = x^*$
  - $$\begin{aligned} E_{k+1} &= |x_{k+1} - x^*| \\ &= |g(x_k) - g(x^*)| \leq c|x_k - x^*| \\ &= cE_k \\ &= c^k E_0 \end{aligned}$$
    * Convergence requires that $c < 1$ when x is close to x^* (i.e. g is not too steep near the root), otherwise it diverges
    * Convergence is linear but can be quadratic if $g'(x) \approx 0$ (e.g. Newton's method)
  - In summary, fixed-point iteration requires choosing a problem-dependent $g(x)$, which must be Lipschitz continuous with $c < 1$ close to x^*
- Newton's method is a form of fixed-point iteration which assumes $f \in C^1$ (continuously differentiable up to the first derivative), which has quadratic convergence
  - We can do a Taylor expansion around x_k to get $f(x) \approx f(x_k) + f'(x_k)(x - x_k)$, then $f(x) = 0 \implies f(x_k) + f'(x_k)(x - x_k) = 0 \implies x = x_k - \frac{f(x_k)}{f'(x_k)}$
  - Therefore Newton's method is equivalent to a fixed-point iteration with $g(x) = x - \frac{f(x)}{f'(x)}$ (i.e. the update step is $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$)
  - Newton's method is exact for a linear function; the more linear a function is, the faster the convergence; if the function is highly nonlinear, Newton's method converges slowly or not at all
  - Assuming f is differentiable, then we can show $E_{k+1} = \left|g'(x^*)E_k + \frac{1}{2}g''(x^*)E_k^2\right| + O(E_k^3)$
    * For Newton updates, $g'(x) = \frac{f(x)f''(x)}{f'(x)^2}$
    * Assuming x^* is a single root, then $f'(x) \neq 0$ close to x^* and so $g'(x) \approx 0$ when close to x^*; this means the $g'(x^*)$ term disappears and the convergence rate becomes quadratic
  - In summary, Newton's method assumes continuous differentiability, in return for quadratic convergence for single roots with $f'(x^*) \neq 0$; however it requires computing the derivative at each step which could be expensive
- If we don't know the actual derivative, we can use the secant method
  - Instead of the derivative we use * $f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$
    * $f(x_{k-1})$ can be recycled from the previous iteration to minimize cost
  - Convergence rate is difficult to analyze, but it is between linear and quadratic
- Hybrid methods that combine one or more methods also exist

- Note the overall speed of a root finding algorithm depends both on the rate of convergence and the cost per iteration; they must be balanced to achieve a fast algorithm
  - The difference between linear and quadratic convergence can be huge in robotics applications, but rate of convergence is not everything

Lecture 5, Sep 22, 2023

Numerical Integration and Differentiation

- Note, numerical integration can refer to two things: computing an integral or solving an ODE
  - Solving an integral is an open-loop process, since the derivative of the function does not depend on its current value
  - Integrating an ODE requires a feedback process, since the derivative is dependent on the current state
    - The existence of feedback means numerical stability must be studied – otherwise errors can accumulate and lead to divergence
  - We will start with the former

Numerical Integration

- Numerical integration is sometimes referred to as quadrature or cubature in higher dimensions
- We want to approximate $\int_a^b f(x)\,\mathrm{d}x$ with a finite number of evaluations of f
  - There is often a tradeoff between accuracy and speed

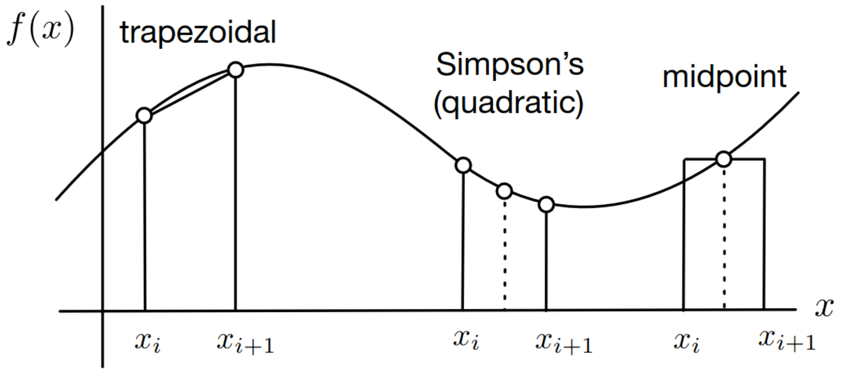


Figure 7: Illustration of 3 common numerical integration rules.

- Common numerical integration rules include:
  - Midpoint rule (1 point of evaluation): $\int_{x_i}^{x_{i+1}} f(x)\,\mathrm{d}x \approx (x_{i+1}-x_i)\cdot f\left(\frac{x_{i+1}+x_i}{2}\right)$
    - The function is approximated as constant between the two bounds
    - $O(\Delta^2)$
  - Trapezoidal rule (2 points of evaluation): $\int_{x_i}^{x_{i+1}} f(x)\,\mathrm{d}x \approx (x_{i+1}-x_i)\cdot \frac{f(x_{i+1})+f(x_i)}{2}$
    - The function is approximated as linear between the two points
    - $O(\Delta^2)$
  - Simpson's rule (3 points of evaluation): $\int_{x_i}^{x_{i+1}} f(x)\,\mathrm{d}x \approx (x_{i+1}-x_i)\cdot \frac{f(x_{i+1})+4f\left(\frac{x_{i+1}+x_i}{2}\right)+f(x_i)}{6}$
    - The function is approximated as quadratic between the two points
    - $O(\Delta^4)$

- The accuracy of integration depends on:
  - The step size $\Delta_i = x_{i+1} - x_i$ – smaller step sizes are more accurate but take more time
  - The type of approximation rule used (midpoint, trapezoidal, Simpson's, etc)
  - The evolution of $f(x)$ (i.e. the nature of the function) – functions that are rougher are inherently harder to integrate
- Techniques exist to adapt the step size dynamically based on where the function is changing the fastest

Numerical Differentiation

- Derivatives can be approximated by a finite difference:
  - Forward difference: $f'(x) \approx \dfrac{f(x+\Delta) - f(\Delta)}{\Delta}$
    - $O(\Delta)$
  - Backward difference: $f'(x) \approx \dfrac{f(x) - f(x-\Delta)}{\Delta}$
    - $O(\Delta)$
  - Centered difference: $f'(x) \approx \dfrac{f(x+\Delta) - f(x-\Delta)}{2\Delta}$
    - $O(\Delta^2)$
- Choosing Δ involves a tradeoff between the approximation accuracy and resilience to numerical errors and noise
  - Choosing Δ makes the algorithm prone to noise and numerical issues because $f(x) \approx f(x+\Delta)$ as $\Delta \to 0$
- Example: order of accuracy for the central difference
  - $\tilde{f}'(t) = \dfrac{f(x+\Delta) - f(x-\Delta)}{2\Delta}$
  - We wish to find $\Delta y = \tilde{f}'(x) - f'(x)$ and its relationship to Δ
  - $$\tilde{f}'(x) = \frac{\begin{array}{c}(f(x) + \Delta f'(x) + \frac{1}{2}\Delta^2 f''(x) + \frac{1}{6}\Delta^3 f'''(x) + O(\Delta^4)) \\ -(f(x) - \Delta f'(x) + \frac{1}{2}\Delta^2 f''(x) - \frac{1}{6}\Delta^3 f'''(x) + O(\Delta^4))\end{array}}{2\Delta}$$
    $$= f'(x) + \frac{1}{6}\Delta^2 + O(\Delta^3)$$
  - $\Delta y = \tilde{f}'(t) - f'(t) = \dfrac{1}{6}\Delta^2 + O(\Delta^3)$
  - Therefore Δy is of order Δ^2

Numerical ODE Solving

- Even for asymptotically stable and well-conditioned ODEs, if we choose the wrong solver or step size, the solution can diverge
- If the function that defines the derivative is continuous and Lipschitz, then the ODE has exactly one solution for all $t \geq 0$, for each initial condition
- If we have a time-varying ODE $\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), t)$, we can convert this into a time-invariant ODE using an *augmented state*, $\boldsymbol{x}'(t) = \begin{bmatrix}\boldsymbol{x}(t) \\ g(t)\end{bmatrix}$ where $g(t) = t$, so then we have $\dfrac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}\boldsymbol{x}(t) \\ g(t)\end{bmatrix} = \begin{bmatrix}f(\boldsymbol{x}(t), g(t)) \\ 1\end{bmatrix}$
  - This allows us to change any time-varying ODE into a time-invariant ODE, but it will become nonlinear

Lecture 6, Sep 27, 2023

Numerical ODE Solving

- We will consider a linear ODE $\dot{x}(t) = f(t) = \lambda x(t), x(0) = x_0$ as our test problem
- We want the test problem to be well-conditioned so we can separate the error resulting from the algorithm from the error resulting from the input/problem

  - This allows us to analyze the algorithm, not the problem itself
- The exact solution to our test problem is $x(t) = x_0 e^{\lambda t}$ so with error if we simulate until time T, $x(T) = (x_0 + \Delta x_0)e^{\lambda T} \implies \Delta\bar{x} = \Delta x_0 e^{\lambda T}$
  - The absolute conditioning number is $\left|\dfrac{\Delta\bar{x}}{\Delta x_0}\right| = e^{\lambda T}$
  - For $\lambda < 0$ this will shrink, so our problem is well-conditioned for $\lambda < 0$
- For any ODE solver, we can analyze two types of error: the *local truncation error* (error per iteration) or the *global truncation error* (error accumulated over time)
  - The global truncation error is typically an order lower than the local truncation error
  - When we say an algorithm is of a certain order, we refer to the global truncation error
- Note that all error discussion assumes numerical stability; if the algorithm is unstable the algorithm will diverge

Forward Euler's Method

- Approximate the derivative as $\dot{x}_k = f_k = \dfrac{x_{k+1} - x_k}{h} + O(h) \implies x_{k+1} = x_k + hf_k$
- This is an *explicit* method because x_{k+1} depends on past values x_k, f_k
- Local truncation error: let $\hat{x}$ be the exact solution, then for each step:
  - $\begin{aligned}\hat{x}(t_{k+1}) &= x(t_k) + h\dot{x}(t_k) + \frac{h^2}{2}\ddot{x}(t_k) + O(h^3) \\ &= x_k + hf_k + O(h^2) \\ &= x_{k+1} + O(h^2)\end{aligned}$
  - This makes the local truncation error second-order
- Global truncation error:
  - To get the state at T_{sim} we have to go through $\dfrac{T_{sim}}{h} = O\left(\dfrac{1}{h}\right)$ times
  - This gives a global truncation error of $O\left(\dfrac{1}{h}\right)O(h^2) = O(h)$
  - Therefore forward Euler is a first-order method

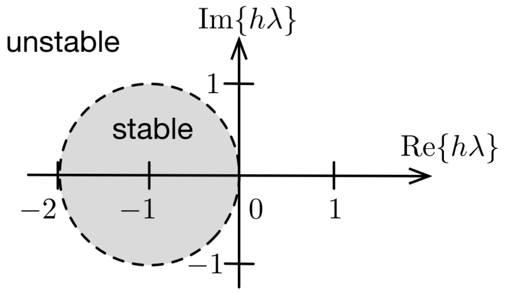


Figure 8: Stability condition for forward Euler's method.

- Stability:
  - Consider the linear test equation: $\hat{x}_{k+1} = x_k + h\lambda x_k$
  - With an initial error of Δx_0: $\begin{aligned}\hat{x}_1 &= (x_0 + \Delta x_0) + h\lambda(x_0 + \Delta x_0) \\ &= \underbrace{(1 + h\lambda)x_0}_{x_1 + \tilde{x}_1} + \underbrace{(1 + h\lambda)\Delta x_0}_{\Delta\tilde{x}_1}\end{aligned}$
  - $\Delta\tilde{x}_1 = (1 + h\lambda)\Delta x_0 \implies \left|\dfrac{\Delta\tilde{x}_{k+1}}{\Delta x_k}\right| = |1 + h\lambda| < 1$ for stability
  - This makes forward Euler *conditionally stable* (even when the problem is well-conditioned, we still need additional conditions for stability)
  - With a larger λ the function is changing faster, so it makes sense that we require a smaller timestep
- If we have a *stiff* system (i.e. ratio of fastest to slowest eigenvalue is large), we need to make h small to

accommodate the fast mode which wastes resources on the slow mode

Backward Euler's Method

- The backward Euler's method instead uses $x_{k+1} = x_k + hf_{k+1}$
  - The derivative at the next timestep is used instead
- This is an *implicit* time-stepping scheme because x_{k+1} no longer depends only on past variables
- To actually implement this we have a number of options, including inverting f, using a numerical root finding algorithm, or making further approximations
  - For some cases, including the test equation, it is still straightforward to solve for x_{k+1}
  - For the test equation $x_{k+1} = x_k + h\lambda x_{k+1} \implies x_{k+1} = \dfrac{x_k}{1 - h\lambda}$
- The stability conditions are $|1 - h\lambda| > 1$, so this time we're stable everywhere except a circle around 1
  - Since the system is always stable for $\operatorname{Re}\lambda < 0$, it is *unconditionally stable*
- Backward Euler can be harder to implement but has much better stability; this gives us more freedom to choose h, which helps with stiff systems in particular (where forward Euler struggles)

Lecture 7, Sep 29, 2023

Numerical ODE Solving Continued

- Recall that we were able to convert a continuous system to a discrete system exactly for a linear system with a zero-order hold; however in practice our systems are nonlinear, so these numerical ODE solvers are used
- Example: $\dot{x} = -\dfrac{1}{\tau}x, x(0) = x_0, \tau > 0$
  - $\tau = -\dfrac{1}{\lambda}$ is known as the time constant of the system; it is the time taken for the output to decay to approximately 37% of the initial response
  - The analytical solution is $x(t) = x_0 e^{-\frac{1}{\tau}t}$
  - Using forward Euler we require $|1 + h\lambda| = \left|1 - \dfrac{h}{\tau}\right| < 1 \implies 0 < h < 2\tau$
    - Smaller τ requires smaller step sizes to avoid divergence
- Example: simple harmonic oscillator $\ddot{x} + \omega^2 x = 0, x(0) = 0, \dot{x}(0) = v_0$
  - The analytical solution is $x(t) = \dfrac{v_0}{\omega}\sin(\omega t)$
  - Put into first-order form: $\dot{\boldsymbol{x}} = \begin{bmatrix} x \\ v \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\omega^2 & 0 \end{bmatrix}\begin{bmatrix} x \\ v \end{bmatrix}, \boldsymbol{x}(0) = \begin{bmatrix} 0 \\ v_0 \end{bmatrix}$
    - The eigenvalues are clearly $\lambda = \pm j\omega$
  - If we diagonalize the system using these eigenvalues we get $\dfrac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$ which gives us a new decoupled system whose eigenvalues are the same as the original system
  - Each equation's stability can now be checked independently
  - $|1 + h\lambda_i| < 1 \implies |1 \pm j\omega h| < 1$ but $|1 + j\omega h| = \sqrt{1 + (h\omega)^2} \geq 1$, so regardless of ω or h forward Euler is unstable
- We can generalize the consistency of time-stepping methods such as forward and backward Euler
  - All time-stepping methods are in some way trying to approximate the average slope $s(x_k, x_{k+1}, h)$ in the interval (t_k, t_{k+1}), so $x_{k+1} = x_k + hs(x_k, x_{k+1}, h)$
  - Therefore for consistency we require $\lim\limits_{h\to 0} s(x_k, x_{k+1}, h) = \dot{x}(t_k) = f(x(t_k))$
  - For example with forward Euler $\lim\limits_{h\to 0} s(x_k, x_{k+1}, h) = \lim\limits_{h\to 0} f_k = f_k$
- Instead of using derivative approximations, what if we try to approximate the integral instead?
  - $x_{k+1} = x_k + \displaystyle\int_{t_k}^{t_{k+1}} \dot{x}(t)\,\mathrm{d}t = x_k + \int_{t_k}^{t_{k+1}} f(x(t))\,\mathrm{d}t$
  - There are many ways to approximate this derivative, each one leads to a different numerical solving scheme

Trapezoidal Method

- This approximates the integral using the trapezoidal rule
- $x_{k+1} = x_k + \frac{h}{2}(f_{k+1} + f_k) + O(h^3)$
- This is an implicit method and it is second-order
- The stability condition is $\left|\frac{1 + \frac{1}{2}h\lambda}{1 - \frac{1}{2}h\lambda}\right| < 1$
  - This is unconditionally stable for $\lambda < 0$, but it requires that we can find f_{k+1}, so it may still be unstable if we don't have a good way of doing so
- However, to implement this we must turn it into an explicit scheme by approximating f_{k+1} with order $O(h^2)$ or higher
  - e.g. we can simply use a forward Euler scheme so $f(x_{k+1}) = f(x_k + hf(x_k)) + O(h^2)$
  - We need at least a second-order approximation scheme so that after multiplication by the timestep h, we still get a third-order error, so the order of the trapezoidal method remains intact
  - This leads to a family of time-stepping schemes known as the *Runge-Kutta methods*

Second-Order Runge-Kutta (Heun's Method)

- *Heun's method*, also known as RK2 (second-order Runge-Kutta) is what we get when we apply the forward Euler's method to approximate f_{k+1} and then use the trapezoidal rule
- $x_{k+1} = x_k + \frac{h}{2}(f(x_k + hf(x_k)) + f_k) + O(h^3)$
- This is an explicit, second-order accurate method that is conditionally stable
- For $\dot{x} = \lambda x$ we have $$\begin{aligned} x_{k+1} &= x_k + \frac{h}{2}(\lambda(x_k + h\lambda x_k) + \lambda x_k) \\ &= \left(1 + \lambda h + \frac{1}{2}h^2\lambda^2\right)x_k \\ &= Mx_k \end{aligned}$$
  - Stability requires that $\left|\frac{\Delta\tilde{x}_{k+1}}{\Delta x_k}\right| = |M| < 1 \implies -4 < 2h\lambda + h^2\lambda^2 < 0$
  - Therefore stability is conditional
  - The region of stability is slightly larger than that of Euler's method but it is not a significant improvement when it comes to stability

Fourth-Order Runge-Kutta

- RK4 uses Simpson's rule for approximating the integral
- $x_{k+1} = x_k + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$ where:
  - $k_1 = f(x_k)$
  - $k_2 = f\left(x_k + \frac{1}{2}hk_1\right)$
  - $k_3 = f\left(x_k + \frac{1}{2}hk_2\right)$
  - $k_4 = f(x_k + hk_3)$
- This is an explicit, fourth-order accurate method that is conditionally stable
- We can see that the method is consistent pretty easily by showing $\lim_{h \to 0} \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) = f(x_k)$
- Stability is cumbersome to derive but the standard approach can be used
- The region of stability is larger but still conditional, however it now includes part of the imaginary axis, which is very good because this allows us to simulate simple harmonic oscillators as their poles lie on the imaginary axis
  - This applies for both SHOs with no damping and very lightly damped SHOs, which is important for structural analysis applications

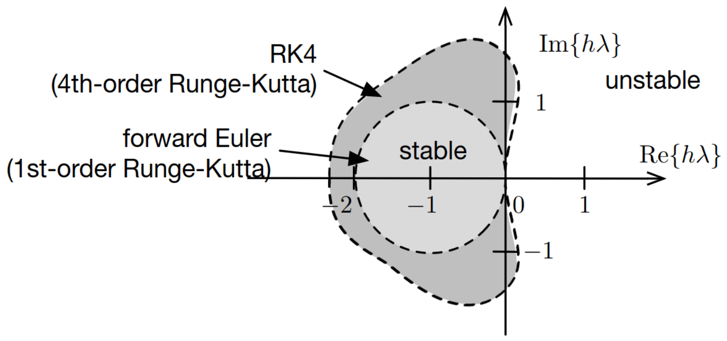


Figure 9: Region of stability for RK4.

Multi-Step Methods

- These methods try to interpolate $f(x(t)) = \dot{x}(t)$ over multiple intervals
  - Information about previous values of $f(t)$ can be used to fit a polynomial, which is used to extrapolate the behaviour
  - This is then fed back to the integration rule
- e.g. we can use a second-order polynomial $\xi(\tau) = a_0 + a_1\tau + a_2\tau^2, -h \le \tau \le h$ so $x_{k+1} = \xi(h)$
- Typically additional approximations on f are used to get different methods
- Example multi-step methods:
  - Midpoint scheme: $x_{k+1} = x_{k-1} + 2hf_k$
    * This uses $x_k = \xi(0) = a_0, f_k = a_1, x_{k-1} = a_0 - a_1h + a_2h^2$
    * This is equivalent to assuming that f is constant over the interval t_{k-1}, t_k, t_{k+1}
      - If we then integrate f then the area under the curve between these two points is $2hf_k$
  - Adams-Bashforth scheme: $x_{k+1} = x_{k-1} + \dfrac{h}{2}(3f_k - f_{k-1})$
    * This uses $x_k = \xi(0) = a_0, f_k = a_1, f_{k-1} = a_1 - 2a_2h$
    * This corresponds to assuming that f is constant over the interval
- Multi-step methods can be more efficient since they attempt to make use of previous information

Lecture 8, Oct 4, 2023

Rules of Thumb for ODEs

- Consider the ODE: $\dot{x} = -\dfrac{1}{\tau}x, x(0) = x_0$
  - The solution is $x_0e^{-\frac{1}{\tau}t}$
  - τ is the *time constant* of the system
  - Numerical stability for forward Euler requires $\left|1 - \dfrac{h}{\tau}\right| < 1 \implies 0 \le h \le 2\tau$
  - So for the system to be stable, h depends on τ
  - In general, how can we determine the simulation parameters such as simulation step size h and simulation time T for an arbitrary ODE?
- Consider $\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x}$ with no input
  - Depending on the eigenvalues the solution can behave differently shown in the diagram below
- We can assign a time constant based on the eigenvalues
  - Purely real eigenvalue: $\tau_i = \dfrac{1}{|\lambda_i|}$
    * In this case τ describes the rate of decay of the system
  - Purely imaginary: $\tau_i = \dfrac{2\pi}{|\lambda_i|}$

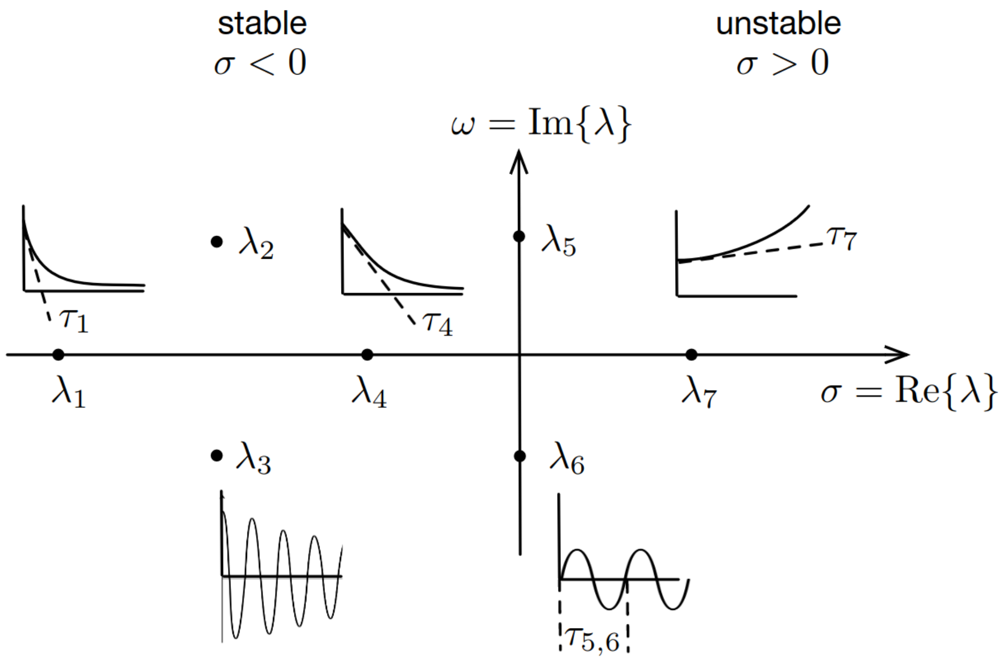


Figure 10: Behaviour of the ODE is determined by eigenvalues.

    * In this case τ is the period of the oscillation; faster oscillations means smaller τ
  - For a complex eigenvalue, we will consider both: $\tau_i = \left\{ \frac{1}{\mathrm{Re}\{\lambda_i\}}, \frac{2\pi}{\mathrm{Im}\{\lambda_i\}} \right\}$
- We define the *stiffness* of the system as $\gamma = \frac{\tau_{max}}{\tau_{min}}$ as the ratio between the maximum and minimum time constants; the ODE is considered *stiff* when $\tau > 10^3$
  - We only care about the max and min time constants since if we can compute the fastest and slowest behaviours of our system, we can compute anything in-between
- General rule of thumb:
  - Take the simulation time to be $T = 5\tau_{max}$ if the system is stable
    * If the system is unstable, stop when a component of x exceeds a threshold
  - Take the step size to be $h = \min\left\{ \frac{\tau_{min}}{10}, \frac{T}{200} \right\}$
    * This gives the number of steps as $k = \frac{T}{h} = \max\{50\gamma, 200\}$
  - For plotting, take the step size to be $H = \frac{T}{200} = \frac{\tau_{max}}{40}$, since we do not need to plot every step
  - For stiff ODEs, fixed-step solvers are generally very expensive, so variable step-size solvers with an initial step size of $\frac{\tau_{min}}{10}$ should be used
    * In MATLAB, solvers ending with "s" are good for stiff systems, e.g. `ode23s`, `ode15s`

Lecture 9, Oct 6, 2023

Optimization

Definition

An *optimization* problem in general seeks to minimize $f(\boldsymbol{x})$, subject to $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0}$ (equality constraints) and $\boldsymbol{h}(\boldsymbol{x}) \geq \boldsymbol{0}$ (inequality constraints), where $f : \mathbb{R}^n \mapsto \mathbb{R}, \boldsymbol{g} : \mathbb{R}^n \mapsto \mathbb{R}^m, \boldsymbol{h} : \mathbb{R}^n \mapsto \mathbb{R}^p$

- Note that argmin $f(\boldsymbol{x}) = \operatorname{argmax}(-f(\boldsymbol{x}))$, so maximisation and minimization are interchangeable
- Some applications in robotics:
  - Motion planning: finding the most efficient path through a cluttered environment
  - Control: finding the sequence of inputs that cause a system to stay as close to a desired trajectory as possible
  - Machine learning: find a hypothesis/model that best explains the input data
  - Computer vision: matching an image to an existing map for localization
  - Reinforcement learning: optimizing a robot's performance over many training runs
- Example: steering a unicycle robot to follow the x axis by controlling ω_k with a constant v_k
  - Dynamics: $\underbrace{\begin{bmatrix} y_{k+1} \\ \theta_{k+1} \end{bmatrix}}_{\boldsymbol{x}_{k+1}} = \underbrace{\begin{bmatrix} y_k \\ \theta_k \end{bmatrix}}_{\boldsymbol{x}_k} + h \begin{bmatrix} v_k \sin\theta_k \\ \omega_k \end{bmatrix}$
  - Problem: given an initial condition (y_0, θ_0) find a sequence $\{\, \omega_0, \dots, \omega_K \,\}$ such that the cost function, $f = \displaystyle\sum_{k=0}^{K} (\boldsymbol{x}_k^T \boldsymbol{Q} \boldsymbol{x}_k + r\omega_k^2)$, is minimized
  - $\boldsymbol{Q}, r$ are the *optimization parameters*
    - In this problem, if we're following our desired trajectory, then the optimal $\boldsymbol{x}_k = \boldsymbol{x}^*$ for all k should be zero, therefore $\boldsymbol{x}_k$ is the error
    - $\boldsymbol{Q}$ is a matrix that weights each component of the error
    - The term $r\omega_k^2$ encourages the algorithm to make less aggressive turns
  - The decision variable is $\boldsymbol{\omega} = \begin{bmatrix} \omega_0 & \omega_1 & \cdots & \omega_K \end{bmatrix}^T$
  - The vehicle dynamics are the equality constraints
  - Inequality constraints can be e.g. $|\omega_k| \le \omega_{\max}$
  - This is an example of *model predictive control*: finding the optimal set of control inputs to execute a trajectory for the next time period

Unconstrained Optimization

- *Unconstrained optimization* is optimization without equality or inequality constraints, i.e. minimizing only $\boldsymbol{f}(\boldsymbol{x})$

Definition

A point $\boldsymbol{x}^* \in \mathbb{R}^n$ is a *global minimum* of $f : \mathbb{R}^n \mapsto \mathbb{R}$ if

$$\forall \boldsymbol{x} \in \mathbb{R}^n, f(\boldsymbol{x}^*) \le f(\boldsymbol{x})$$

$\boldsymbol{x}^*$ is a *local minimum* if

$$\exists \varepsilon > 0 \text{ s.t. } \forall \boldsymbol{x} \in \mathbb{R}^n, \|\boldsymbol{x} - \boldsymbol{x}^*\|_2 < \varepsilon \implies f(\boldsymbol{x}^*) \le f(\boldsymbol{x})$$

- Assuming $f(\boldsymbol{x})$ is differentiable, then $\vec{\nabla} f = \begin{bmatrix} \dfrac{\partial f}{\partial x_1} & \dfrac{\partial f}{\partial x_2} & \cdots & \dfrac{\partial f}{\partial x_n} \end{bmatrix}^T \in \mathbb{R}^{1 \times n}$ describes the direction of steepest ascent; so $-\vec{\nabla} f$ is the direction of steepest descent
- If $\boldsymbol{x}_o$ is a local minimum, then $\vec{\nabla} f(\boldsymbol{x}_o) = \boldsymbol{0}$, which is a necessary but not sufficient condition for minimization

Definition

A *stationary point* of f is a point $\boldsymbol{x} \in \mathbb{R}^n$ satisfying $\vec{\nabla} f(\boldsymbol{x}) = \boldsymbol{0}$.

Theorem

First-Order Optimality condition: If $\boldsymbol{x}$ is a local minimum of f, then $\boldsymbol{x}$ is a stationary point. Note that being a stationary point does not imply that $\boldsymbol{x}$ is a local minimum.

- A common strategy is to then find all stationary points $\boldsymbol{x}_i$, compute $f(\boldsymbol{x}_i)$ for all the points, and find the point with the lowest $f(\boldsymbol{x}_i)$
  - This is guaranteed to work, but it can be very hard to find all the local minima
  - Note we also need to check the boundaries with $\boldsymbol{x} \to \pm\infty$, e.g. if the function asymptotically converges

Definition

The *Hessian* matrix for a twice-differentiable f is the symmetric matrix

$$\boldsymbol{H}_f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{bmatrix}$$

If (and only if) $\boldsymbol{x}^*$ is a stationary point, and $\boldsymbol{H}_f(\boldsymbol{x}^*)$ is positive-definite (i.e. $\forall \boldsymbol{v}, \boldsymbol{v}^T \boldsymbol{H}_f \boldsymbol{v} > 0$, or that all eigenvalues are real and positive), then $\boldsymbol{x}^*$ is a local minimum. This is the *Second-Order Optimality Condition.*

- Note the Hessian reduces to a second derivative in the single-dimensional case
- This condition works because at a stationary point $f(\boldsymbol{x}) \approx f(\boldsymbol{x}^*) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^*)^T \boldsymbol{H}_f(\boldsymbol{x}^*)(\boldsymbol{x} - \boldsymbol{x}^*)$ since the gradient disappears, so if $\boldsymbol{H}_f(\boldsymbol{x}^*)$ is positive-definite, this expression is guaranteed to be greater than $f(\boldsymbol{x}^*)$
- Note also:
  - If $\boldsymbol{H}_f(\boldsymbol{x}^*)$ is negative definite, then $\boldsymbol{x}^*$ is a local maximum
  - If $\boldsymbol{H}_f(\boldsymbol{x}^*)$ is indefinite (positive and negative eigenvalues), then $\boldsymbol{H}_f(\boldsymbol{x}^*)$ is a saddle point
  - If $\boldsymbol{H}_f(\boldsymbol{f}^*)$ is noninvertible (at least one zero eigenvalue), then odd things can happen, e.g. multiple local minima next to each other (flat function)

Definition

A function $f : \mathbb{R}^n \mapsto \mathbb{R}$ is *convex* if

$$\forall \boldsymbol{x}_1 \neq \boldsymbol{x}_2, \forall \alpha \in (0,1), f((1-\alpha)\boldsymbol{x}_1 + \alpha \boldsymbol{x}_2) \leq (1-\alpha)f(\boldsymbol{x}_1) + \alpha f(\boldsymbol{x}_2)$$

f is said to be *strictly convex* if the $\leq$ is replaced with a $<$ in the above expression.

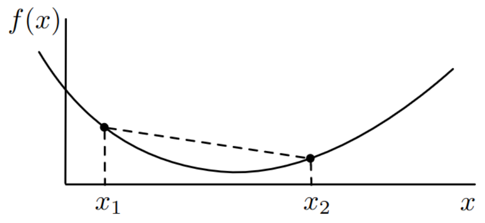


Figure 11: Illustration of convexity.

- Intuitively, this says that if we took two values $\boldsymbol{x}_1, \boldsymbol{x}_2$, then all the values of f in between the points will lie below the line connecting the two points

Theorem

A local minimum of a convex function $f : \mathbb{R}^n \mapsto \mathbb{R}$ is necessarily a global maximum.

- This can be proven by contradiction
- f is convex if $\boldsymbol{H}_f(\boldsymbol{x})$ is positive definite for all $\boldsymbol{x}$
- Convex functions are much easier to work with, but in the real world, few functions are actually convex
  - Sometimes we can reformulate or relax specific parameters to make the function convex

Lecture 10, Oct 11, 2023

Numerical Methods for Unconstrained Optimization

- Finding a global minimum is hard to do analytically because we need to find all stationary points, and then compute the value of the function at all stationary points and then the boundaries, which could be expensive
  - Often we will have to use a numerical root finding method to solve for where the gradient is zero
  - In this case it might be easier to solve for the minimum numerically to begin with

One-Dimensional Methods

- We will focus on finding the stationary points $\vec{\nabla} f(x^*) = 0$
- Newton's method can be used if the function is differentiable:
  - Approximate $f(x)$ locally by a quadratic: $f(x) \approx f(x_k) + f'(x_k)(x - x_k) + \dfrac{1}{2} f''(x_k)(x - x_k)^2$
  - $f'(x) \approx f'(x_k) + f''(x_k)(x - x_k) \implies x_{k+1} = x_k - \dfrac{f'(x_k)}{f''(x_k)}$
    - Keep iterating until the gradient is below some tolerance
    - Once we find a stationary point, check if it is a minimum using the Hessian/second derivative
  - This is equivalent to applying Newton's method for root finding to solve for $f'(x) = 0$
  - This method has quadratic convergence (pretty fast!) provided we start close enough to the minimum
  - If derivatives are expensive to compute, we can fit a quadratic to three points to make the approximation, e.g. $f(x_{k-2}), f(x_{k-1}), f(x_k)$

Definition

A function $f : [a, b] \mapsto \mathbb{R}$ is *unimodular* if there exists $x^* \in [a, b]$ such that f is decreasing/nonincreasing for $x \in [a, x^*]$ and increasing/nondecreasing for $x \in [x^*, b]$, i.e. $f'(x) \leq 0$ for $x < x^*$ and $f'(x) \geq 0$ for $x > x^*$.

- What if our function is non-differentiable? We can exploit other properties of the function such as unimodularity
  - Suppose we evaluate f at x_0 and x_1; then if $f(x_0) > f(x_1)$ we know x_0 is not a minimum, so discard everything to the left; or if $f(x_0) < f(x_1)$ we know x_1 is not a minimum, so we discard everything to the right
  - Note that we can't discard both sides because that could potentially discard the section that the minimum is in
- This algorithm is called *golden section search*:
  - Steps:
    1. Without loss of generality, assume $a = 0, b = 1$ (rescale the function)

2. Choose $x_0 = \alpha, x_1 = 1 - \alpha$ for $\alpha \in \left(0, \frac{1}{2}\right)$
3. Discard one side of the interval based on $f(x_0)$ and $f(x_1)$, then rescale the interval and repeat

- This algorithm has unconditional linear convergence as long as f is unimodular
- We can further optimize the algorithm by choosing x_1 at the next iteration to be x_0 from the previous iteration to save one function evaluation, so $\alpha = (1-\alpha)^2 \implies 1 - \alpha = \frac{1}{2}(\sqrt{5} - 1)$
  * This gives us the golden ratio, hence the name

- If f is neither unimodular nor differentiable, we need to exploit some other structure of f

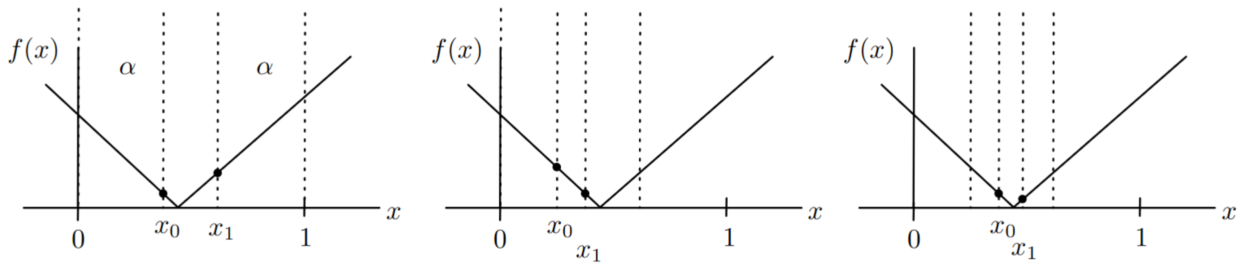


Figure 12: Illustration of golden section search.

Multidimensional Methods

- Suppose we want to minimize $f(\boldsymbol{x})$ where $f : \mathbb{R}^n \mapsto \mathbb{R}$; assume $f(\boldsymbol{x})$ is twice-differentiable
- Gradient descent requires only the first derivative, but is relatively slow (linear)
  - The idea is to iteratively take small steps in the local direction of steepest descent, which is in the opposite of direction as the gradient
  - For a sufficiently small α, $f(\boldsymbol{x} - \alpha\vec{\nabla}f(\boldsymbol{x})^T) \leq f(\boldsymbol{x})$
  - Steps:
    1. Let $g(\alpha) = f(\boldsymbol{x}_k - \alpha\vec{\nabla}f(\boldsymbol{x}_k)^T)$
    2. Find $\alpha^* = \min_{\alpha} g(\alpha)$ through a one-dimensional line search on $g(\alpha)$ for $\alpha \geq 0$
    3. Update the estimate as $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha^*\vec{\nabla}f(\boldsymbol{x}_k)^T$
    4. Repeat until $\boldsymbol{x}_k$ changes sufficiently slowly
  - Note that this allows us to convert a multidimensional optimization problem down to a single-variable optimization
  - The linear search for α^* is so that we can get to the minimum faster and don't get stuck/jump around the minimum
  - In practice the line search is expensive, so suboptimal techniques such as fixed α and backtracking are used, but these have no convergence guarantees
  - Gradient descent is impacted by poor conditioning of $f(\boldsymbol{x})$; i.e. if $f(\boldsymbol{x})$ is changing rapidly in one dimension and slowly in another, it takes many iterations as the faster dimension is prioritized

- Newton's method can be adapted for multiple dimensions by replacing the derivative by the gradient and the second derivative by the Hessian
  - $f(\boldsymbol{x}) \approx f(\boldsymbol{x}_k) + \vec{\nabla}f(\boldsymbol{x}_k)(\boldsymbol{x} - \boldsymbol{x}_k) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_k)^T\boldsymbol{H}_f(\boldsymbol{x}_k)(\boldsymbol{x} - \boldsymbol{x}_k)$
  - Solving for $\vec{\nabla}f(\boldsymbol{x}) = \boldsymbol{0}$ for critical points gives $\boldsymbol{H}_f(\boldsymbol{x}_k)(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k) = -\vec{\nabla}f(\boldsymbol{x}_k)$
  - Therefore we can iterate as $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{H}_f^{-1}(\boldsymbol{x}_k)\vec{\nabla}f(\boldsymbol{x}_k)$ and repeat until the change in $\boldsymbol{x}$ becomes small
  - This helps with poor conditioning because applying $\boldsymbol{H}_f^{-1}$ effectively unwarps the problem
  - The tradeoff is that Newton's method requires a Hessian to be inverted, which makes it slower and less applicable; there can also be issues with singularity of the Hessian
  - This has quadratic convergence
- Common variants of Newton's method:

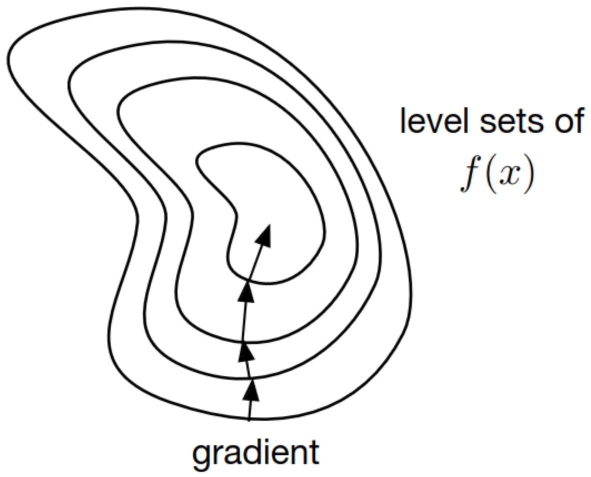


Figure 13: Illustration of gradient descent.

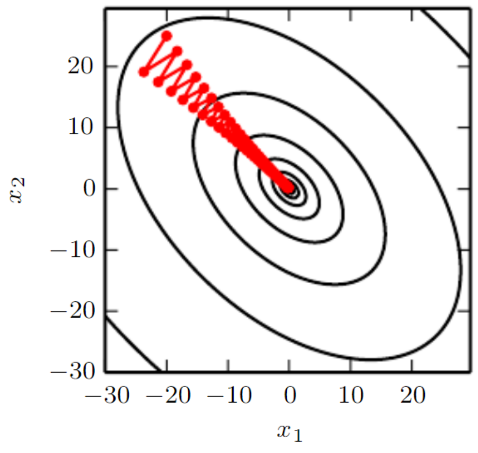


Figure 14: Gradient descent with poor conditioning.

  - The Gauss-Newton method approximates the Hessian using first derivatives instead, which is a common compromise between gradient descent and Newton's method
  - Levenberg-Marquardt provides adaptive regularization to the Hessian when it is close to singular (effectively downgrading to gradient descent)

Note

In this course gradients are assumed to be row vectors, $\vec{\nabla} f(\boldsymbol{x}) \in \mathbb{R}^{1\times n}$ for $x \in \mathbb{R}^n$.

Example: Nonlinear Least Squares

- Let $e_i(\boldsymbol{\theta}) = y_i - g(x_i, \boldsymbol{\theta})$; we want to find $\boldsymbol{\theta}^*$ such that $e_i(\boldsymbol{\theta}^*)$ is minimized
  - This is applicable in e.g. neural network training (where θ are the weights), or system identification (where θ are the system parameters)
- The nonlinear least squares problem is $\min\limits_{\boldsymbol{\theta}} \sum\limits_i \|e_i(\boldsymbol{\theta})\|^2$, or equivalently $\min\limits_{\boldsymbol{\theta}} \boldsymbol{e}(\boldsymbol{\theta})^T\boldsymbol{e}(\boldsymbol{\theta})$
- Gradient: $\vec{\nabla} f(\boldsymbol{\theta}_k) = \dfrac{\partial f}{\partial \boldsymbol{e}}\dfrac{\partial \boldsymbol{e}}{\partial \boldsymbol{\theta}} = 2\boldsymbol{e}(\theta)^T\dfrac{\partial \boldsymbol{e}(\theta)}{\partial \boldsymbol{\theta}}$ where $\dfrac{\partial \boldsymbol{e}(\theta)}{\partial \boldsymbol{\theta}} = \boldsymbol{J}$ is the Jacobian
- Hessian: $\boldsymbol{H}_f = \vec{\nabla}^2 f(\boldsymbol{\theta}_k) = 2\left(\dfrac{\partial \boldsymbol{e}(t)}{\partial \boldsymbol{\theta}}^T\dfrac{\partial \boldsymbol{e}(t)}{\partial \boldsymbol{\theta}} + \boldsymbol{e}(\boldsymbol{\theta})^T\vec{\nabla}^2_{\boldsymbol{\theta}}\boldsymbol{e}(\boldsymbol{\theta})\right)$
  - Note we get $\dfrac{\partial \boldsymbol{e}(t)}{\partial \boldsymbol{\theta}}^T\dfrac{\partial \boldsymbol{e}(t)}{\partial \boldsymbol{\theta}} = \boldsymbol{J}^T\boldsymbol{J}$ for free from the gradient, but the second term $\boldsymbol{e}(\boldsymbol{\theta})^T\vec{\nabla}^2_{\boldsymbol{\theta}}\boldsymbol{e}(\boldsymbol{\theta})$ is expensive
  - Since the second term involves the error, we can approximate it as zero assuming that we are close to the optimum
- Using Newton's method, $\boldsymbol{H}_f\Delta\boldsymbol{\theta} = 2\boldsymbol{e}(\boldsymbol{\theta}_k)^T\boldsymbol{J}(\boldsymbol{\theta}_k) \implies \Delta\boldsymbol{\theta} = (\boldsymbol{J}\boldsymbol{J}^T)^{-1}\boldsymbol{J}^T\boldsymbol{e}(\boldsymbol{\theta}_k)$
- Making this Hessian approximation is known as the Gauss-Newton method
- In practice we use techniques to make sure $\boldsymbol{J}\boldsymbol{J}^T$ remains sparse, so it can be inverted quickly
- Most SLAM algorithms use some form of Gauss-Newton

Lecture 11, Oct 13, 2023

Equality-Constrained Optimization

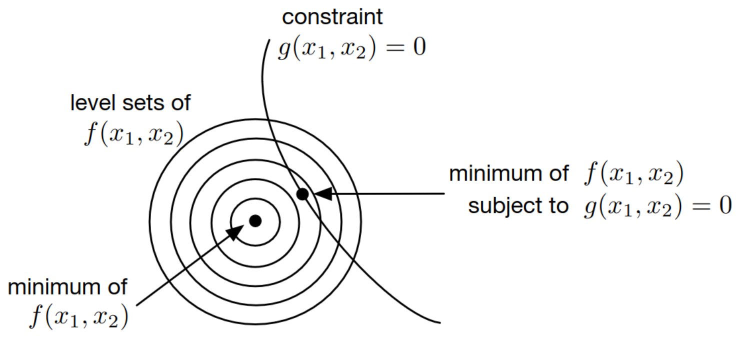


Figure 15: Illustration of equality-constrained optimization.

Definition

A *feasible point* of a constrained optimization problem (minimize $f(\boldsymbol{x})$ subject to $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0}$, $\boldsymbol{h}(\boldsymbol{x}) \geq \boldsymbol{0}$) is any point $\boldsymbol{x}$ satisfying the constraints $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0}$ and $\boldsymbol{h}(\boldsymbol{x}) \geq \boldsymbol{0}$.

The *feasible set* is the set of all feasible points.

Definition

A *critical point* of constrained optimization is a local maximum, minimum, or saddle point of f within the feasible set.

- Consider a simpler problem of minimizing $f(\boldsymbol{x})$ subject to a single equality constraint $g(\boldsymbol{x}) = 0$
  - The feasible set is $S_0 = \{\, \boldsymbol{x} \mid g(\boldsymbol{x}) = 0 \,\}$
  - Consider taking a small step $\Delta\boldsymbol{x}$ from a feasible point; to stay on the level curve $g(\boldsymbol{x}) = 0$, this step must be taken in a directional orthogonal to $\vec{\nabla} g(\boldsymbol{x})$, i.e. $\vec{\nabla} g(\boldsymbol{x})\Delta\boldsymbol{x} = 0$
  - Now consider taking a small step Δx from a minimum; then we must have $f(\boldsymbol{x}^* + \Delta\boldsymbol{x}) \geq f(\boldsymbol{x}^*)$, if the step is taken such that $g(\boldsymbol{x}) = 0$ is satisfied as above
    * This means $\vec{\nabla} f(\boldsymbol{x}^*)\Delta\boldsymbol{x} \geq 0$ by Taylor expansion
    * But we also have $\vec{\nabla} g(\boldsymbol{x})(-\Delta\boldsymbol{x}) = 0$, so we know that $\vec{\nabla} f(\boldsymbol{x}^*)(-\Delta\boldsymbol{x}) \geq 0$, which means $\vec{\nabla} f(\boldsymbol{x}^*)\Delta\boldsymbol{x} = 0$ is the only way this can be true
- Since $\vec{\nabla} f(x^*)\Delta\boldsymbol{x} = 0 = \vec{\nabla} g(\boldsymbol{x}^*)\Delta\boldsymbol{x}$ we have $\vec{\nabla} f(\boldsymbol{x}^*) = \lambda\vec{\nabla} g(\boldsymbol{x}^*)$ for some λ at a minimum (that is, the gradient of the function and the constraint are parallel)
  - λ is known as the *Lagrange multiplier* (nothing to do with eigenvalues)
- Let the *Lagrangian* be $\Lambda(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) - \lambda g(\boldsymbol{x})$, then the unconstrained stationary points of $\Lambda(\boldsymbol{x}, \lambda)$ with respect to both λ and $\boldsymbol{x}$ are critical points of the optimization problem, since they satisfy:
  - $\dfrac{\partial \Lambda}{\partial \lambda} = -g(\boldsymbol{x}) = 0$
  - $\dfrac{\partial \Lambda}{\partial \boldsymbol{x}} = \vec{\nabla} f(\boldsymbol{x}) - \lambda\vec{\nabla} g(\boldsymbol{x}) = 0$
  - This allows us to convert a constrained optimization problem to an unconstrained one, so that any unconstrained solver we saw earlier can be used
  - This can be extended to multiple equality constraints

Theorem

Method of Lagrange Multipliers: The critical points of the equality-constrained optimization problem of minimizing $f(\boldsymbol{x})$ subject to $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0}$ are the unconstrained critical points of the *Lagrangian*:

$$\min_{\boldsymbol{x}, \boldsymbol{\lambda}} \Lambda(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) - \boldsymbol{\lambda}^T \boldsymbol{g}(\boldsymbol{x})$$

Note that the Lagrangian is minimized with respect to both $\boldsymbol{x}$ and $\boldsymbol{\lambda}$.

- The Lagrange multiplier gives information about how the objective function changes if the constraints change by a small amount
- Example: maximizing a rectangle's area subject to a fixed perimeter of length 1
  - We wish to minimize $f(\boldsymbol{x}) = f(w, l) = -wl$ subject to the constraint that $g(\boldsymbol{x}) = g(w, l) = 2w + 2l - 1 = 0$
  - Use the Lagrangian: $\Lambda(w, l, \lambda) = -wl - \lambda(2w + 2l - 1)$, which we can optimize without constraints
  - $\dfrac{\partial \Lambda}{\partial w} = -l^* - 2\lambda^*$, $\dfrac{\partial \Lambda}{\partial l} = -w^* - 2\lambda^*$, $\dfrac{\partial \Lambda}{\partial \lambda} = 1 - 2w^* - 2l^*$
  - In matrix form: $\begin{bmatrix} 0 & -1 & -2 \\ -1 & 0 & -2 \\ 2 & 2 & 0 \end{bmatrix} \begin{bmatrix} w^* \\ l^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

- We get the solution $w^* = l^* = \frac{1}{4}, \lambda^* = -\frac{1}{8}$, which is a square as expected

- Example: minimize $f(\boldsymbol{x}) = x_1^2 + x_2^2 = \boldsymbol{x}^T\boldsymbol{x}$ subject to $g(\boldsymbol{x}) = x_1 + x_2 - 1 = \begin{bmatrix}1 & 1\end{bmatrix}\boldsymbol{x} - 1 = \boldsymbol{a}^T\boldsymbol{x} - 1 = 0$
  - The level curves are circles centered at the origin, and the constraint is a line passing through $(0,1)$ and $(1,0)$
  - $\Lambda = f(\boldsymbol{x}) - \lambda g(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{x} - \lambda(\boldsymbol{a}^T\boldsymbol{x} - 1)$
  - $\frac{\partial\Lambda}{\partial\lambda} = \boldsymbol{a}\boldsymbol{x} - 1 = 0 = g(x)$ (we can always just take the constraint; no need to take this derivative)
  - $\frac{\partial\Lambda}{\partial x} = 2\boldsymbol{x}^T - \lambda\boldsymbol{a}^T = 0 \implies x = \frac{\lambda}{2}\boldsymbol{a}$
  - $\boldsymbol{a}^T\boldsymbol{a}\frac{\lambda}{2} = 1 \implies 2\frac{\lambda}{2} = 1 \implies \lambda = 1, \boldsymbol{x} = \begin{bmatrix}\frac{1}{2} \\ \frac{1}{2}\end{bmatrix}$

- Example: $f(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}, \boldsymbol{A} = \begin{bmatrix}3 & \sqrt{2} \\ \sqrt{2} & 2\end{bmatrix}, g(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{x} - 1 = 0$
  - The level curves are skewed ellipses around the origin; the constraint is a circle of radius 1
  - $\Lambda = \boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} - \lambda(\boldsymbol{x}^T\boldsymbol{x} - 1)$
  - $\frac{\partial\Lambda}{\partial\boldsymbol{x}} = 2\boldsymbol{x}^T\boldsymbol{A} - 2\lambda\boldsymbol{x}^T = 0 \implies \boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x}$ which is an eigenvalue equation
    - Substituting this back in: $\Lambda = \boldsymbol{x}^T\boldsymbol{x}\lambda - \lambda(\boldsymbol{x}^T\boldsymbol{x} - 1) = \lambda$
  - The eigenvalues are $\lambda = 1, 4$, so we choose $\lambda = 1$ and its eigenvector normalized to unit length (for constraint g)
  - Note that there are actually two solutions here ($\pm$ on the unit eigenvector)

Lecture 12, Oct 18, 2023

Inequality-Constrained Optimization

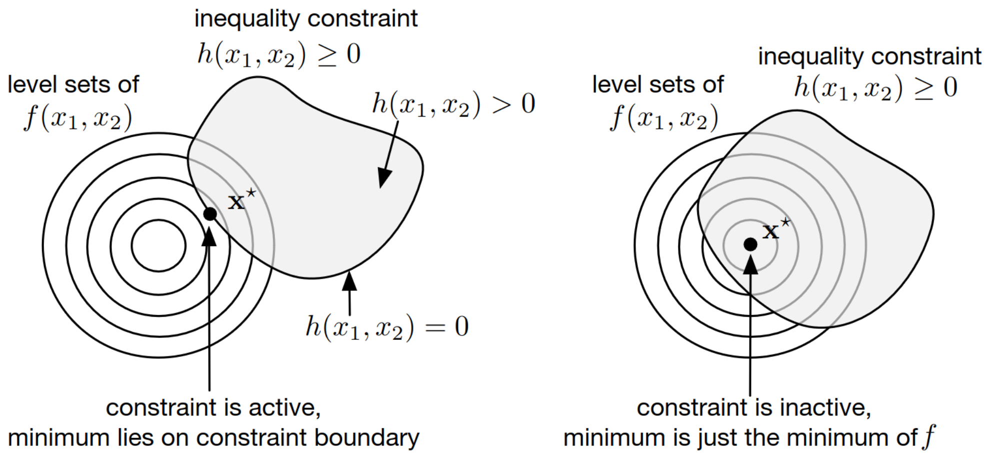


Figure 16: Active vs. inactive constraints.

- We will now consider the full optimization problem of minimizing $f(\boldsymbol{x})$ subject to $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0}$ and $\boldsymbol{h}(\boldsymbol{x}) \geq \boldsymbol{0}$
- There are 2 possible cases for each inequality constraint h_i; consider a minimum $\boldsymbol{x}$, then either $h_i(\boldsymbol{x}) = 0$ this point (active), or $h_i(\boldsymbol{x}) > 0$ (inactive)
  - In the first case, the inequality constraint is the same as an equality constraint and the optimum will be on the boundary of the inequality region
  - In the second case, the inequality constraint has effectively no impact on where the optimum is, since it lies fully within the inequality region

- If there are multiple active inequality constraints, then the solution lies on the intersection of their boundaries

- If we assume that all inequality constraints are active, we can then treat them as equality constraints and use the Lagrange multiplier approach
  - $\Lambda(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) - \boldsymbol{\lambda}^T \boldsymbol{g}(\boldsymbol{x}) - \boldsymbol{\mu}^T \boldsymbol{h}(\boldsymbol{x})$
  - Then for a critical point $\vec{\nabla} f(\boldsymbol{x}) - \sum_i \lambda_i \vec{\nabla} g_i(\boldsymbol{x}) - \sum_j \mu_j \vec{\nabla} h_j(\boldsymbol{x}) = \boldsymbol{0}$ and $g_i(\boldsymbol{x}) = h_j(\boldsymbol{x}) = \boldsymbol{0}, \forall i, j$
- If we let $\mu_j = 0$ whenever the inequality constraint h_j is inactive (i.e. $h_j(\boldsymbol{x}) \neq 0$), then the inactive constraints drop out and the above condition holds for the general case
  - Let $\forall j, \mu_j h_j(\boldsymbol{x}) = \boldsymbol{0}$, then the condition above holds in general
  - This condition is known as *complementary slackness*

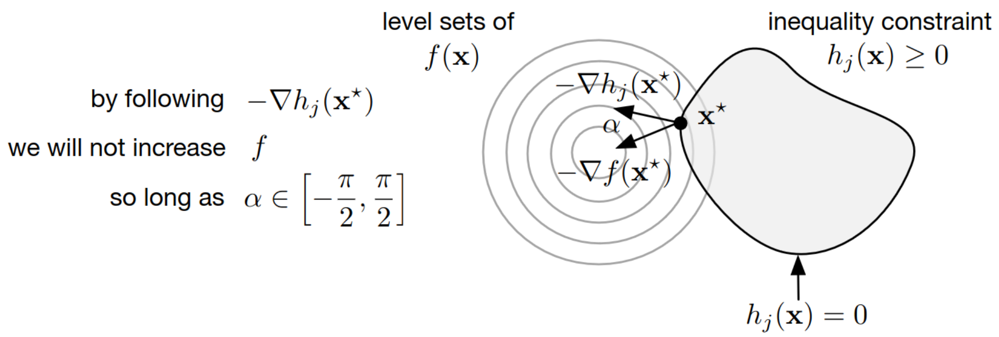

level sets of
$f(\mathbf{x})$
inequality constraint
$h_j(\mathbf{x}) \geq 0$
by following $-\nabla h_j(\mathbf{x}^\star)$
we will not increase f
so long as $\alpha \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$
$-\nabla h_j(\mathbf{x}^\star)$
α
$\mathbf{x}^\star$
$-\nabla f(\mathbf{x}^\star)$
$h_j(\mathbf{x}) = 0$

Figure 17: Constraints on Lagrange multipliers for inequality constraints.

- Note that in the equality constraint case, our Lagrange multipliers can be positive or negative; but for inequality constraints, this is not the case
  - Consider the diagram above where $\boldsymbol{x}^*$ is a critical point and $\Delta \boldsymbol{x}$ is some perturbation; let constraint $h_j(\boldsymbol{x})$ be active
  - Note that $\vec{\nabla} h_i(\boldsymbol{x})$ is pointing inwards of the feasible set
  - Intuitively, if $\vec{\nabla} f(\boldsymbol{x})$ and $\vec{\nabla} h_i(\boldsymbol{x})$ have a negative dot product (i.e. they point at least somewhat in the opposite direction), then we can move into our feasible set while decreasing $f(\boldsymbol{x})$
    * But if $\vec{\nabla} f(\boldsymbol{x})$ and $\vec{\nabla} h_i(\boldsymbol{x})$ have a positive dot product, then we cannot both decrease $f(\boldsymbol{x})$ while keeping the inequality constraint satisfied
  - Let $\Delta \boldsymbol{x} = \vec{\nabla} h_i(\boldsymbol{x}^*)^T$, then
    * $h_i(\boldsymbol{x}^* + \Delta \boldsymbol{x}) \approx h_i(\boldsymbol{x}^*) + \vec{\nabla} h_i(\boldsymbol{x}^*) \Delta \boldsymbol{x} = h_i(\boldsymbol{x}^*) + \vec{\nabla} h(\boldsymbol{x}^*) \vec{\nabla} h(\boldsymbol{x}^*)$
    * Since both terms on the right are positive, we know that $h_i(\boldsymbol{x}^* + \Delta \boldsymbol{x}) \geq 0$, i.e. this perturbation is still feasible
    * $f(\boldsymbol{x}^* + \Delta x) \approx f(\boldsymbol{x}^*) + \vec{\nabla} f(\boldsymbol{x}^*) \Delta \boldsymbol{x} = f(\boldsymbol{x}^*) + \vec{\nabla} f(\boldsymbol{x}^*) \vec{\nabla} h_i(\boldsymbol{x}^*)^T$
    * If the second term is negative, then we know that the perturbation decreases f, which means $\boldsymbol{x}^*$ is no longer a critical point – contradiction
  - Therefore we must have that $\vec{\nabla} f(\boldsymbol{x}^*) \vec{\nabla} h_i(\boldsymbol{x}^*)^T \geq 0$ at a local minimum, hence $\vec{\nabla} f(\boldsymbol{x}^*) = \mu_i \vec{\nabla} h_i(\boldsymbol{x}^*)$ where $\mu_i \geq 0$ (strictly greater if h_i must be active)
  - This condition is known as *dual feasibility*
- Note that we can always take an equality constraint and rewrite it in terms of 2 inequality constraints, i.e. $g_i(\boldsymbol{x}) = 0 \iff g_i(\boldsymbol{x}) \geq 0, -g_i(\boldsymbol{x}) \geq 0$
  - This lets us write the critical point condition in terms of only inequality constraints

Theorem

Karush-Kuhn-Tucker (KTT) Conditions: The vector $\boldsymbol{x}^* \in \mathbb{R}^n$ is a critical point for minimizing $f(\boldsymbol{x})$ subject to $\boldsymbol{g}(\boldsymbol{x}) = 0$, $\boldsymbol{h}(\boldsymbol{x}) \geq \boldsymbol{0}$ when there exists $\boldsymbol{\lambda} \in \mathbb{R}^m$ and $\boldsymbol{\mu} \in \mathbb{R}^p$ such that:

1. Stationarity: $\vec{\nabla} f(\boldsymbol{x}^*) - \sum_i \lambda_i \vec{\nabla} g_i(\boldsymbol{x}^*) - \sum_j \mu_j \vec{\nabla} h_j(\boldsymbol{x}^*) = \boldsymbol{0}$
2. Primal feasibility: $\boldsymbol{g}(\boldsymbol{x}^*) = \boldsymbol{0}$ and $\boldsymbol{h}(\boldsymbol{x}^*) \geq \boldsymbol{0}$
3. Complementary slackness: $\forall j, \mu_j h_j(\boldsymbol{x}^*) = 0$
4. Dual feasibility: $\forall j, \mu_j \geq 0$

- To find whether $\boldsymbol{x}^*$ is a local minimum (instead of a maximum or saddle point), we need to check the Hessian constrained to the subspace of $\mathbb{R}^n$ in which $\boldsymbol{x}$ can move without violating the constraints
- Example: optimal rectangle: same rectangle optimization constraint from last time, but we enforce $w \leq \bar{w}$
  - The inequality constraint can be expressed as $\bar{w} - w \geq 0$
  - The Lagrangian becomes $\Lambda(w, l, \lambda, \mu) = -wl - \lambda(2w + 2l - 1) - \mu(\bar{w} - w)$
  - Using the KTT optimality conditions:
    * Stationarity:
      - $\dfrac{\partial \Lambda}{\partial w} = -l^* - 2\lambda^* + \mu^* = 0$
      - $\dfrac{\partial \Lambda}{\partial l} = -w^* - 2\lambda^* = 0$
    * Primal feasibility:
      - $2w^* + 2l^* - 1 = 0$
      - $\bar{w} - \bar{w}^* \geq 0$
    * Complementary slackness: $\mu^*(\bar{w} - \bar{w}^*) = 0$
    * Dual feasibility: $\mu^* \geq 0$
  - We now need to consider 2 cases:
    * $\mu^* = 0$, where the inequality constraint is inactive, so we have $w^* = l^* = \dfrac{1}{4}$ as before
      - We can see that all our equations reduce to the same thing we had previously
    * $\mu^* \neq 0$, where the inequality constraint is active
      - We have $w^* = \bar{w}$ from complementary slackness
      - $\lambda^* = -\dfrac{\bar{w}}{2}, l^* = \dfrac{1}{2} - \bar{w}$
      - Substituting into the first stationary equation, $\mu^* = \dfrac{1}{2} - 2\bar{w}$
      - Since $\mu^* \geq 0$, this only holds if $\bar{w} \leq \dfrac{1}{4}$ – i.e. the constraint would only be active if $\bar{w} \leq \dfrac{1}{4}$ (above this value we just get the normal solution)
  - In summary: when $\bar{w} \leq \dfrac{1}{4}$, the constraint is active and we have $w^* = \bar{w}, l^* = \dfrac{1}{2} - \bar{w}$; when $\bar{w} \geq \dfrac{1}{4}$, then the inequality constraint is active and $w^* = l^* = \dfrac{1}{4}$
  - In general inequality constraints complicate the problem since now we need to consider multiple cases

Numerical Optimization Methods

- We will discuss two broad categories of algorithms:
  - *Sequential quadratic programming (SQP)*: iteratively solve a set of simpler, less constrained problems that approximate the fully constrained problem
  - *Barrier methods*: replace the constraints with penalties in the objective function, and then optimize the new function using unconstrained methods
- SQP approximates $f, \boldsymbol{g}, \boldsymbol{h}$ by simpler functions
  - f is replaced by a quadratic objective while the constraints are replaced by linear constraints

- This is similar to Newton's method – taking a quadratic approximation, solving for the minimum, and then approximate again
- The active set method checks which constraints are active
  * Given an active set, all constraints are treated as equality constraints and Lagrange multipliers are used to minimize
- SQP only converges if we start with an initial guess in the neighbourhood of the true minimum, similar to Newton's method
- Most model predictive controllers (MPC controllers) use SQP methods

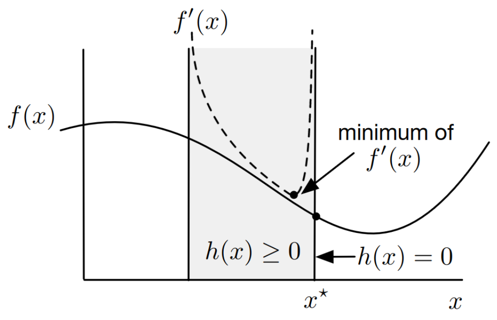
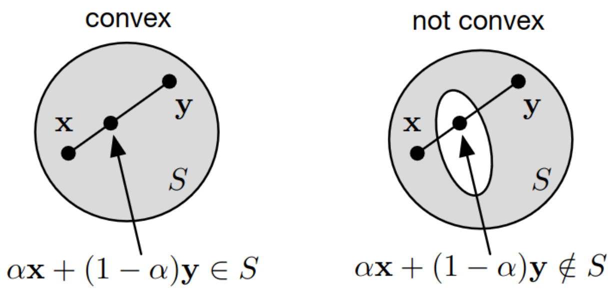


Figure 18: Illustration of barrier methods.

- Barrier methods add penalties to the objective for violating constraints, then optimizes the new objective without constraints
  - e.g. define a new objective as $f'(x) = f(x) + \rho \dfrac{1}{h(x)}$ with ρ being some weight
  - If constraints are not satisfied closely enough, increase ρ to penalize the cost more
  - ρ is decreased iteratively if we are close enough to the constraints, which brings us closer to the true minimum

Convex Optimization

- We want to develop methods for identifying convex problems



convex
not convex
$\mathbf{x}$
$\mathbf{y}$
S
$\alpha\mathbf{x} + (1-\alpha)\mathbf{y} \in S$
$\mathbf{x}$
$\mathbf{y}$
S
$\alpha\mathbf{x} + (1-\alpha)\mathbf{y} \notin S$

Figure 19: Convex and non-convex sets.

Definition

A set $S \subseteq \mathbb{R}^n$ is *convex* if
$$\forall \boldsymbol{x}, \boldsymbol{y} \in S, \alpha \in [0,1], \alpha\boldsymbol{x} + (1-\alpha)\boldsymbol{y} \in S$$

- Additional useful properties:
  - The intersection of convex sets is also a convex set
    * This means if we combine two convex constraints, we also get a convex constraint back

  - The sum and maximum of convex functions is also a convex function
  - If f is convex, then $\{\, \boldsymbol{x} \mid f(\boldsymbol{x}) \leq c \,\}$ is a convex set for a fixed $c \in \mathbb{R}$
    * We can "chop off" a function below some line to get a convex set

Theorem

If the objective function f and feasible set of an optimization problem are both convex, then the optimization problem possesses a unique minimum.

- An optimization problem is convex if its objective function is convex and its feasibility region is also a convex set
  - When a problem is convex, we can make strong convergence guarantees
  - Intuition: if both $\boldsymbol{x}$ and $\boldsymbol{y}$ are in the feasible set, then any potential minimum that lies between them is also in the feasible set
- Example: consider the constrained least-squares problem of minimizing $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2$ subject to $\boldsymbol{x} \in \boldsymbol{x} \in \mathbb{R}^n | \boldsymbol{x} \geq \boldsymbol{0}$
  - The objective function is convex since it expands to $\boldsymbol{x}^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} - 2b^T\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}^T\boldsymbol{b}$, which has a Hessian of $\boldsymbol{A}^T\boldsymbol{A}$, which is positive semi-definite for any $\boldsymbol{A}$
  - We can also show that the feasible set is convex since the positive weighted sum of any two positive numbers is also positive
  - Therefore this problem is convex
- Example: optimizing the 1-norm, $\|\boldsymbol{x}\|_1 = |x_1| + |x_2| + \cdots + |x_n|$
  - We can rewrite it as minimizing $\displaystyle\sum_i y_i$ with respect to $\boldsymbol{x}, \boldsymbol{y}$, subject to $\boldsymbol{x}, \boldsymbol{y} \in \{\, \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n \mid y_i \geq x_i, y_i \geq -x_i \,\}$
  - We're forcing $|x_i| \geq y_i$ and trying to minimize the sum of y_i
  - This is now a convex problem: the objective function is convex since all y_i are convex, so their sum is convex; the feasible set is convex since $y_i - x_i \geq 0$ and $y_i + x_i \geq 0$ are both convex functions, so if we let $h_{i_1}(y, x) = y_i - x_i$, then $h_{i_1} \leq 0$ is also convex; the intersection of both convex constraints gives a convex feasible set
- Note that our optimal rectangle problem is not convex but we still found the global minimum
- Minimizing $\boldsymbol{c}^T\boldsymbol{x}$ subject to $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ is referred to as a *linear program*, which is a special convex problem that can be solved very quickly
- Second-order cone programs are $\boldsymbol{c}^T\boldsymbol{x}$ subject to $\|\boldsymbol{A}_i\boldsymbol{x} - \boldsymbol{b}_i\|_2 \leq d_i + \boldsymbol{c}^T\boldsymbol{x}$ which are also convex
- What about nonconvex problems?
  - We can try to approximate $f(\boldsymbol{x})$ with an easier (convex) problem
  - We can sample the space of feasible $\boldsymbol{x}$ as "seeds" for starting points of a local optimization, optimize all of them, and then picking the minimum
  - Randomized algorithms also exist (e.g. simulated annealing)
- Sometimes we want to do *online optimization*, where the objective function changes over time
  - A simple strategy is to use the old optimum $\boldsymbol{x}^*$ as the initial guess for the new problem

Lecture 13, Oct 20, 2023

Additional Linear Algebra Topics

Positive Definiteness

Definition

A matrix $\boldsymbol{B} \in \mathbb{R}^{n \times n}$ is *positive semidefinite* if

$$\forall \boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{x}^T \boldsymbol{B} \boldsymbol{x} \geq 0$$

$\boldsymbol{B}$ is *positive definite* if

$$\forall \boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{x} \neq \boldsymbol{0} \implies \boldsymbol{x}^T \boldsymbol{B} \boldsymbol{x} > 0$$

- $\boldsymbol{x}^T \boldsymbol{B} \boldsymbol{x}$ is referred to as the *quadratic form*, which is the matrix version of x^2
- For any $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, $\boldsymbol{A}^T \boldsymbol{A}$ is positive semi-definite; $\boldsymbol{A}^T \boldsymbol{A}$ is positive definite if and only if $\boldsymbol{A}$ is full rank
  - $\boldsymbol{x}^T(\boldsymbol{A}^T \boldsymbol{A})\boldsymbol{x} = (\boldsymbol{A}\boldsymbol{x})^T(\boldsymbol{A}\boldsymbol{x}) = |\boldsymbol{A}\boldsymbol{x}|_2^2 \geq 0$
  - If $\boldsymbol{A}$ has linearly independent columns, then $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{0} \implies \boldsymbol{x} = \boldsymbol{0}$, so $\boldsymbol{x}^T(\boldsymbol{A}^T \boldsymbol{A})\boldsymbol{x} = (\boldsymbol{A}\boldsymbol{x})^T(\boldsymbol{A}\boldsymbol{x}) = \boldsymbol{0}$ only when $\boldsymbol{x} = \boldsymbol{0}$; this goes both ways
- The eigenvalues of a positive semi-definite matrix are always greater than or equal to zero; for positive definite matrices all eigenvalues are strictly positive
- Positive definite matrices come up often as inertia matrices or covariance matrices

Orthogonality

Definition

A set of vectors $\{\, \boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_n \,\}$ is *orthonormal* if and only if

$$\boldsymbol{v}_i \cdot \boldsymbol{v}_j = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

That is, each vector has norm 1 and is orthogonal to every other vector.

A square matrix whose columns are orthonormal is called an *orthogonal* matrix.

- Let $\boldsymbol{Q}$ be orthogonal, then:
  - $\boldsymbol{Q}^T \boldsymbol{Q} = \boldsymbol{1}$, and so $\boldsymbol{Q}^T = \boldsymbol{Q}^{-1}$
  - Applying $\boldsymbol{Q}$ has a linear transformation will not affect the length of a vector or the angle between two vectors; this means $\boldsymbol{Q}$ is an *isometry*
    - $\|\boldsymbol{Q}\boldsymbol{x}\|_2^2 = \boldsymbol{x}^T \boldsymbol{Q}^T \boldsymbol{Q} \boldsymbol{x} = \boldsymbol{x}^T \boldsymbol{x} = \|\boldsymbol{x}\|_2^2$
    - $(\boldsymbol{Q}\boldsymbol{x}) \cdot (\boldsymbol{Q}\boldsymbol{y}) = \boldsymbol{x}^T \boldsymbol{Q}^T \boldsymbol{Q} \boldsymbol{y} = \boldsymbol{x}^T \boldsymbol{y} = \boldsymbol{x} \cdot \boldsymbol{y}$
- An orthogonal matrix can rotate vectors but not scale them; all rotation matrices are orthogonal

Solving Linear Systems

- Solving systems in the form of $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ is a common problem
- However solving by $\boldsymbol{A}^{-1}\boldsymbol{b}$ is almost never a good idea since $\boldsymbol{A}^{-1}$ can be expensive to compute, reduces solution accuracy, and is less efficient since a sparse $\boldsymbol{A}$ will have a dense $\boldsymbol{A}^{-1}$
- Gaussian elimination works for any $\boldsymbol{A}$ and $\boldsymbol{b}$, but we can only achieve $O(n^3)$ for $\boldsymbol{A} \in \mathbb{R}^{n \times n}$; to get better performance, we can exploit the structure of a matrix (e.g. sparse/dense, triangular, Hermitian, etc)
  - Simplest case: $\boldsymbol{A}$ diagonal, which we can solve in $O(n)$

    - If $\boldsymbol{A}$ is upper or lower triangular, we can solve in $O(n^2)$; we can use each row to solve for exactly a single element of $\boldsymbol{x}$
    - For a sparse matrix, if we can split it up into blocks, we can solve for each block individually
- Gaussian elimination is equivalent to first factorizing $\boldsymbol{Ax} = \boldsymbol{LUx} = \boldsymbol{b}$ and then solving $\boldsymbol{Ly} = \boldsymbol{b}, \boldsymbol{Ux} = \boldsymbol{y}$, where $\boldsymbol{L}$ is lower triangular and $\boldsymbol{U}$ is upper triangular
    - If we want to reuse $\boldsymbol{A}$ with different values of $\boldsymbol{b}$, we can prefactorize $\boldsymbol{A}$ and reuse the factors to save time
- In practice, use `x = np.linalg.solve(A, b)` in Python or `x = A \ b` in MATLAB
- Example: solve $\begin{bmatrix}1 & 2\\2 & 5\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix} = \begin{bmatrix}1\\1\end{bmatrix}$
    - We could do this by Gaussian elimination:
        * $\begin{bmatrix}1 & 2 & 1\\2 & 5 & 1\end{bmatrix} \to \begin{bmatrix}1 & 2 & 1\\0 & 1 & -1\end{bmatrix} \to \begin{bmatrix}1 & 2\\0 & 1\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix} = \begin{bmatrix}1\\-1\end{bmatrix}$
        * $x_2 = -1 \implies x_1 = 3$
    - Note the first row operation was $\left(I - 2\begin{bmatrix}0 & 0\\1 & 0\end{bmatrix}\right)\boldsymbol{A} = \begin{bmatrix}1 & 2\\0 & 1\end{bmatrix}$
        * $\begin{bmatrix}1 & 0\\-2 & 1\end{bmatrix}\begin{bmatrix}1 & 2\\2 & 5\end{bmatrix} = \begin{bmatrix}1 & 2\\0 & 1\end{bmatrix}$
        * Notice the matrix multiplying $\boldsymbol{A}$ is lower triangular and the result is upper triangular; now we can invert the first matrix to get a form of $\boldsymbol{A} = \boldsymbol{LU}$, since the inverse of a lower triangular matrix is also lower triangular
    - LU factorization:
        * $\begin{bmatrix}1 & 0\\2 & 1\end{bmatrix}\begin{bmatrix}1 & 2\\0 & 1\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix} = \begin{bmatrix}b_1\\b_2\end{bmatrix}$
        * Solve first $\begin{bmatrix}1\\0\\2\\1\end{bmatrix}\begin{bmatrix}y_1\\y_2\end{bmatrix} = \begin{bmatrix}1 & 1\\ \implies & y\end{bmatrix}_1 = 1, y_2 = -1$
        * Now we can solve $\begin{bmatrix}1 & 2\\0 & 1\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix} = \boldsymbol{y} = \begin{bmatrix}1\\-1\end{bmatrix}$, which gives us the same result

Matrix and Vector Norms

Definition

A general *vector norm* is any function $\|\cdot\| : \mathbb{R}^n \mapsto [0, \infty)$ which satisfies the following conditions:

1. $\|\boldsymbol{x}\| = 0 \iff \boldsymbol{x} = \boldsymbol{0}$
2. $\forall c \in \mathbb{R}, \boldsymbol{x} \in \mathbb{R}^n, \|c\boldsymbol{x}\| = |c|\|\boldsymbol{x}\|$
3. $\forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n, \|\boldsymbol{x} + \boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\|$

- $\|\boldsymbol{x}\| \geq 0$ follows from these conditions
- As with the Euclidean norm, norms encode some notion of "length"
- Typical vector norms:
    - The p-norm for $p \geq 1$ is defined as $\|\boldsymbol{x}\|_p = (|x_1|^p + |x_2|^p + \cdots + |x_n|^p)^{\frac{1}{p}}$
        * The 2-norm, or Euclidean norm, is an example of a p-norm
        * If we constrain $\|\boldsymbol{x}\|_p = 1$, we get boxes of various shapes; e.g. a 1-norm is a rotated square in 2D, 2-norm is a circle in 2D, and infinity norm is a square in 2D; all other norms are somewhere in between
    - The ∞-norm (infinity norm) is defined as $\|\boldsymbol{x}\|_\infty = \max(|x_1|, |x_2|, \ldots, |x_n|)$
- All p-norms for $p \geq 1$ (including the ∞-norm) are convex

Definition

The *matrix norm* on $\mathbb{R}^{m\times n}$ induced by a vector norm $\|\cdot\|$ is given by

$$\|\boldsymbol{A}\| = \max\left\{\, \|\boldsymbol{A}\boldsymbol{x}\| \mid \|\boldsymbol{x}\| = 1 \,\right\}$$

Or equivalently

$$\|\boldsymbol{A}\| = \max_{\boldsymbol{x}\in\mathbb{R}^n, \boldsymbol{x}\neq 0} \frac{\|\boldsymbol{A}\boldsymbol{x}\|}{\|\boldsymbol{x}\|}$$

- The induced vector norm is essentially the maximum norm of a unit vector after multiplying by $\boldsymbol{A}$
- This makes the property that $\|\boldsymbol{A}\boldsymbol{x}\| \leq \|\boldsymbol{A}\|\|\boldsymbol{x}\|$
- Typical matrix norms:
  - 1-norm: $\|\boldsymbol{A}\|_1 = \max\limits_{i\leq j\leq n} \sum\limits_{i=1}^{m} |a_{ij}|$
    - This is equivalent to the maximum column sum
  - 2-norm: $\|\boldsymbol{A}\|_2 = \max\left\{ \sqrt{\lambda} \;\middle|\; \exists \boldsymbol{x} \in \mathbb{R}^n \text{ s.t. } \boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x} \right\}$
    - This is the square root of the largest eigenvalue of $\boldsymbol{A}^T\boldsymbol{A}$ (intuitively we can interpret this as the largest eigenvalue of $\boldsymbol{A}$)
    - Sometimes called the *spectral radius* of $\boldsymbol{A}$
  - Frobenius norm: $\|\boldsymbol{A}\|_F = \sqrt{\sum\limits_{i=1}^{m}\sum\limits_{j=1}^{n} |a_{ij}|^2} = \sqrt{\operatorname{tr} \boldsymbol{A}^T\boldsymbol{A}}$
    - $\|\boldsymbol{A}\|_2 \leq \|\boldsymbol{A}\|_F$ always holds
  - ∞-norm (infinity norm): $\|\boldsymbol{A}\|_\infty = \max\limits_{1\leq i\leq m} \sum\limits_{j=1}^{n} |a_{ij}|$
    - This is the maximum row sum
    - Proof:
      - $\|\boldsymbol{A}\|_\infty = \max\left\{\, \|\boldsymbol{A}\boldsymbol{x}\|_\infty \mid \|\boldsymbol{x}\|_\infty = 1 \,\right\}$
      - Note $[\boldsymbol{A}\boldsymbol{x}]_i = \sum\limits_j a_{ij}x_j$, so $\|\boldsymbol{A}\boldsymbol{x}\|_\infty = \max\limits_i \left|\sum\limits_j a_{ij}x_j\right| \leq \max\limits_i \sum\limits_j |a_{ij}||x_j| \leq \max\limits_i \sum\limits_j |a_{ij}|x_{max} = \max\limits_i \sum\limits_j |a_{ij}|\|\boldsymbol{x}\|_\infty = \max\limits_i \sum\limits_j |a_{ij}|$
      - We can show the bound the other way by selecting x_j in a special way (see posted notes)

Condition Number

Definition

The *condition number* of $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ with respect to a given norm $\|\cdot\|$ is

$$\operatorname{cond} \boldsymbol{A} = \|\boldsymbol{A}\|\|\boldsymbol{A}^{-1}\|$$

if $\boldsymbol{A}$ is non-invertible, $\operatorname{cond} \boldsymbol{A} = \infty$ by definition.

- Recall that conditioning describes how a small error in the input propagates to an error in the output
- For a matrix, we ask the question: for finding $\boldsymbol{x}$ such that $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, how does the solution $\boldsymbol{x}$ change if we make a small change to the matrices $\boldsymbol{A}$ and $\boldsymbol{b}$?
  - We can derive the relative condition number to be $|\varepsilon|\|\boldsymbol{A}\|\|\boldsymbol{A}^{-1}\|$, where ε is some input error
- e.g. if we used the 2-norm, we would essentially get the ratio between the largest eigenvalue and the smallest eigenvalue; intuitively if these two eigenvalues are different, we will see more error since the system is stiff

Lecture 14, Oct 25, 2023

Eigendecomposition

Definition

An *eigenvector* $\boldsymbol{x} \neq \boldsymbol{0}$ of a square matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is any vector satisfying

$$\boldsymbol{A}\boldsymbol{x} = \lambda \boldsymbol{x}$$

for some *eigenvalue* $\lambda \in \mathbb{C}$.

Definition

The *spectrum* of $\boldsymbol{A}$ is the set of all eigenvalues of $\boldsymbol{A}$.

The *spectral radius* of $\boldsymbol{A}$ is $\rho(\boldsymbol{A}) = \max|\lambda|$ overall eigenvalues λ of $\boldsymbol{A}$.

- Eigenvalues are connected to optimization; consider the optimization: $\min_{\boldsymbol{x}} \boldsymbol{x}^T \boldsymbol{A}\boldsymbol{x}$ subject to $\|\boldsymbol{x}\|_2 = 1$
  - This is a quadratically constrained quadratic program (QCQP)
  - The Lagrangian is $\Lambda(\boldsymbol{x}, \lambda) = \boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} - \lambda(\boldsymbol{x}^T\boldsymbol{x} - 1)$
  - From $\dfrac{\partial \Lambda}{\partial \boldsymbol{x}^T} = 0$ we get that $2\boldsymbol{A}\boldsymbol{x} - 2\lambda\boldsymbol{x} = 0 \implies \boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x}$
  - Therefore the critical points are the unit eigenvectors of $\boldsymbol{A}$ (unit due to the constraint)
- Important properties:
  - Eigenvectors corresponding to different eigenvalues are linearly independent
  - $\boldsymbol{A}$ is diagonalizable if its eigenvectors span $\mathbb{R}^n$, in which case $\boldsymbol{A} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{-1} \iff \boldsymbol{\Lambda} = \boldsymbol{V}^{-1}\boldsymbol{A}\boldsymbol{V}$, which is called a *similarity transformation*
    - If $\boldsymbol{A}$ is not diagonalizable, it is *degenerate* or has *degenerate eigenvalues*
    - If $\boldsymbol{V}$ is orthogonal, we can interpret this as decomposing $\boldsymbol{A}$ into a pure rotation, a pure scaling, and then the inverse of the pure rotation
  - *Spectral theorem*: Symmetric matrices have n orthonormal eigenvectors with (possibly repeated) real eigenvalues
  - Positive definite matrices have all positive eigenvalues; positive semi-definite matrices have all nonnegative eigenvalues

Singular Value Decomposition

Definition

The *singular value decomposition* (SVD) of $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ is

$$\boldsymbol{U}^T\boldsymbol{A}\boldsymbol{V} = \boldsymbol{\Sigma} \iff \boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$$

where $\boldsymbol{V} \in \mathbb{R}^{n \times n}, \boldsymbol{U} \in \mathbb{R}^{m \times m}$ are orthogonal matrices, and $\boldsymbol{\Sigma} \in \mathbb{R}^{m \times n}$ is a rectangular diagonal matrix which contains the singular values of $\boldsymbol{A}$.

Note that $\boldsymbol{V}$ contains the eigenvectors of $\boldsymbol{A}^T\boldsymbol{A}$ and $\boldsymbol{U}$ contains the eigenvectors of $\boldsymbol{A}\boldsymbol{A}^T$, while the $\sigma_i \in \mathbb{R}$ singular values on the diagonal of $\boldsymbol{\Sigma}$ are the square roots of the eigenvalues of $\boldsymbol{A}^T\boldsymbol{A}$.

- Note $\boldsymbol{\Sigma}$ has the same dimension as $\boldsymbol{A}$
- $\boldsymbol{A}^T\boldsymbol{A}$, $\boldsymbol{A}\boldsymbol{A}^T$ are symmetric, so by the spectral theorem their eigenvalues are orthonormal; furthermore, $\boldsymbol{A}^T\boldsymbol{A}$ is positive semi-definite so eigenvalues are all nonnegative, thus the singular values are real

- $\boldsymbol{\Sigma}$ has the singular values $\sigma_1, \dots, \sigma_m$ (assuming $m \leq n$) on its diagonal; note that it can be rectangular
  - We can split up $\boldsymbol{\Sigma}$ into a square diagonal matrix and a block of all zeroes
  - In general $\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1 & & & \mathbf{0} \\ & \ddots & & \vdots \\ & & \sigma_k & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \end{bmatrix}$
  - By convention the singular values are sorted such that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k \geq \sigma_{k+1} = \sigma_{k+2} = \cdots = 0$
  - The first $k = \operatorname{rank} \boldsymbol{A}$ singular values are nonzero (note k is also the number of strictly positive eigenvalues of $\boldsymbol{A}^T\boldsymbol{A}$)
- Note that $\boldsymbol{AV} = \boldsymbol{U\Sigma} \implies \boldsymbol{A}\boldsymbol{v}_j = \sigma_j \boldsymbol{u}_j$; therefore we see that the SVD is a generalization of eigendecomposition for non-square $\boldsymbol{A}$
  - $\boldsymbol{V} = \begin{bmatrix} \boldsymbol{v}_1 & \boldsymbol{v}_2 & \cdots & \boldsymbol{v}_k & \boldsymbol{v}_{k+1} & \boldsymbol{v}_{k+2} & \cdots & \boldsymbol{v}_n \end{bmatrix}$
    * $\boldsymbol{v}_1$ to $\boldsymbol{v}_k$ are the normalized eigenvectors of $\boldsymbol{A}^T\boldsymbol{A}$
    * $\boldsymbol{v}_{k+1}$ to $\boldsymbol{v}_n$ are taken from the null space of $\boldsymbol{A}^T\boldsymbol{A}$
    * All $\boldsymbol{v}_j$ are chosen such that $\boldsymbol{V}$ is orthogonal
  - $\boldsymbol{U} = \begin{bmatrix} \boldsymbol{u}_1 & \boldsymbol{u}_2 & \cdots & \boldsymbol{u}_k & \boldsymbol{u}_{k+1} & \boldsymbol{u}_{k+2} & \cdots & \boldsymbol{u}_n \end{bmatrix}$
    * $\boldsymbol{u}_1$ to $\boldsymbol{u}_k$ are the normalized eigenvectors of $\boldsymbol{AA}^T$
    * $\boldsymbol{u}_{k+1}$ to $\boldsymbol{u}_n$ are taken from the null space of $\boldsymbol{AA}^T$
    * All $\boldsymbol{u}_j$ are chosen such that $\boldsymbol{U}$ is orthogonal
    * This is because $\boldsymbol{AA}^T = \boldsymbol{U\Sigma V}^T(\boldsymbol{V\Sigma}^T\boldsymbol{U}^T) = \boldsymbol{U\Sigma\Sigma}^T\boldsymbol{U}^T = \boldsymbol{U\Lambda U}^T$ which is just a diagonalization; ditto for $\boldsymbol{V}$
- Example: find the SVD of $\begin{bmatrix} 1 & 0 \end{bmatrix}$
  - $\boldsymbol{A}^T\boldsymbol{A} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$
    * This has rank 1, giving eigenvalues $\lambda_1 = 1, \lambda_2 = 0$
    * We get the nonzero singular value $\sigma_1 = \sqrt{\lambda_1} = 1$
    * The unit eigenvector corresponding to λ_1 is $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$
    * To get the second vector in $\boldsymbol{V}$ we take $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$, which is orthogonal to the first and is a zero eigenvector
  - To find $\boldsymbol{u}_j$ we can use $\boldsymbol{A}\boldsymbol{v}_j = \sigma_j\boldsymbol{u}_j$, giving $u_1 = 1$
  - Therefore $\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \end{bmatrix}, \boldsymbol{V} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \boldsymbol{U} = \begin{bmatrix} 1 \end{bmatrix}$
- Example: find the SVD of $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$
  - $\boldsymbol{A}^T\boldsymbol{A} = 1$ giving $\lambda_1 = 1 \implies \sigma_1 = 1$ and an eigenvector of 1
  - $\boldsymbol{A}\boldsymbol{v}_j = \sigma_j\boldsymbol{u}_j \implies \boldsymbol{u}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} 1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$
    * Choose $\boldsymbol{u}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ so that it is orthogonal to $\boldsymbol{u}_1$ and gives $\boldsymbol{AA}^T\boldsymbol{u}_2 = \mathbf{0}$
  - Therefore $\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \boldsymbol{V} = \begin{bmatrix} 1 \end{bmatrix}, \boldsymbol{U} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
  - We could have found this using the previous example's result, since the matrix was just transposed
- If $\boldsymbol{A}_1 = \boldsymbol{U}_1\boldsymbol{\Sigma}_1\boldsymbol{V}_1^T$ and $\boldsymbol{A}_1^T = \boldsymbol{A}_2 = \boldsymbol{U}_2\boldsymbol{\Sigma}_2\boldsymbol{V}_2^T$, then $\boldsymbol{V}_1 = \boldsymbol{U}_2, \boldsymbol{U}_1 = \boldsymbol{V}_2, \boldsymbol{\Sigma}_1 = \boldsymbol{\Sigma}_2^T$
- Consider the effect of $\boldsymbol{A}$ on length: examine the critical points of $\|\boldsymbol{Ax}\|_2^2 = \boldsymbol{x}^T\boldsymbol{A}^T\boldsymbol{Ax}$ (the *Rayleigh quotient*) subject to $\|\boldsymbol{x}\|_2 = 1$
  - From the example in the previous section we know that critical points are the unit eigenvectors of $\boldsymbol{A}^T\boldsymbol{A}$: $\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{v}_i = \lambda_i\boldsymbol{v}_i$, where all eigenvalues are nonnegative and eigenvectors are orthonormal
  - Let $\tilde{\boldsymbol{u}}_i = \boldsymbol{A}\boldsymbol{v}_i$, then $\lambda_i\tilde{\boldsymbol{u}}_i = \lambda_i\boldsymbol{A}\boldsymbol{v}_i = \boldsymbol{A}(\lambda_i\boldsymbol{v}_i) = \boldsymbol{AA}^T\boldsymbol{A}\boldsymbol{v}_i = \boldsymbol{AA}^T\tilde{\boldsymbol{u}}_i$ so $\tilde{\boldsymbol{u}}_i$ is an eigenvector of $\boldsymbol{AA}^T$ with the same eigenvalue λ_i
    * $\|\tilde{\boldsymbol{u}}_i\|_2 = \sqrt{\boldsymbol{v}_i^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{v}_i} = \sqrt{\lambda_i\boldsymbol{v}_i^T\boldsymbol{v}_i} = \sqrt{\lambda}\|\boldsymbol{v}_i\|_2 = \sqrt{\lambda}$

    - * We can normalize to get $\boldsymbol{u}_i = \dfrac{\tilde{\boldsymbol{u}}_i}{\lambda_i}$
  - So we get two eigenproblems $\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{v}_i = \lambda_i\boldsymbol{v}_i, \boldsymbol{A}\boldsymbol{A}^T\boldsymbol{u}_i = \lambda_i\boldsymbol{u}_i$ with all $\boldsymbol{v}_i, \boldsymbol{u}_1$ being orthonormal
  - Now define $\boldsymbol{U}_1 = \begin{bmatrix}\boldsymbol{u}_1 & \cdots & \boldsymbol{u}_k\end{bmatrix}, \boldsymbol{V}_1 = \begin{bmatrix}\boldsymbol{v}_1 & \cdots & \boldsymbol{v}_k\end{bmatrix}$ and we can show that this leads us to the SVD
- $\boldsymbol{A} = \begin{bmatrix}\boldsymbol{U}_1 & \boldsymbol{U}_2\end{bmatrix}\begin{bmatrix}\boldsymbol{\Sigma}_{11} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0}\end{bmatrix}\begin{bmatrix}\boldsymbol{V}_1^T \\ \boldsymbol{V}_2^T\end{bmatrix}$ where $\boldsymbol{U}_2$ and $\boldsymbol{V}_2$ are chosen in the null spaces of $\boldsymbol{A}\boldsymbol{A}^T$, $\boldsymbol{A}^T\boldsymbol{A}$ to give orthogonality
  - Essentially, $\boldsymbol{A}$ takes everything from span $\{\boldsymbol{V}_1\}$ and maps it to span $\{\boldsymbol{U}_1\}$; everything in span $\{\boldsymbol{V}_2\}$ is mapped to zero, so everything in span $\{\boldsymbol{U}_2\}$ are all the points that $\boldsymbol{A}$ cannot reach
  - $\boldsymbol{V}_1, \boldsymbol{U}_1$ are unique, but $\boldsymbol{V}_2, \boldsymbol{U}_2$ can be chosen arbitrarily as long as orthogonality is maintained
  - In practice, we rarely need to compute $\boldsymbol{V}_2, \boldsymbol{U}_2$
- The SVD allows us to write $\boldsymbol{A}$ using its modes: $\boldsymbol{A} = \sum_{i=1}^{k}\sigma_i\boldsymbol{u}_i\boldsymbol{v}_i^T$
  - Since the σ_i are sorted, the earlier terms in the summation are more "important"
  - We can cut of this summation to get a version of $\boldsymbol{A}$ with some of the unimportant bits removed

Lecture 15, Oct 27, 2023

Interpretations of the SVD

- If $\boldsymbol{A}$ is a linear mapping, we can interpret the SVD as breaking up $\boldsymbol{A}$ into a rotation/projection $\boldsymbol{V}^T$, a scaling $\boldsymbol{\Sigma}$ and reconstructing in the basis $\boldsymbol{U}$
  - The input vector $\boldsymbol{x}$ is projected into the orthonormal basis $\boldsymbol{V}^T$
  - Each component gets scaled by a singular value
  - The rescaled components are reconstructed into an output vector using the basis $\boldsymbol{U}$
  - Note that since the singular values are ordered, $\boldsymbol{v}_1$ is the "highest gain" input vector direction and $\boldsymbol{u}_2$ is the "highest gain" output vector direction
- We can approximate $\boldsymbol{A}$ with a lower rank matrix $\tilde{\boldsymbol{A}}$
  - $\tilde{\boldsymbol{A}}_l = \sum_{i=1}^{l}\sigma_i\boldsymbol{u}_1\boldsymbol{v}_i^T$
  - Since the singular values are in descending order, we're basically keeping the "more important" parts of the matrix
  - This is linked to dimensionality reduction techniques

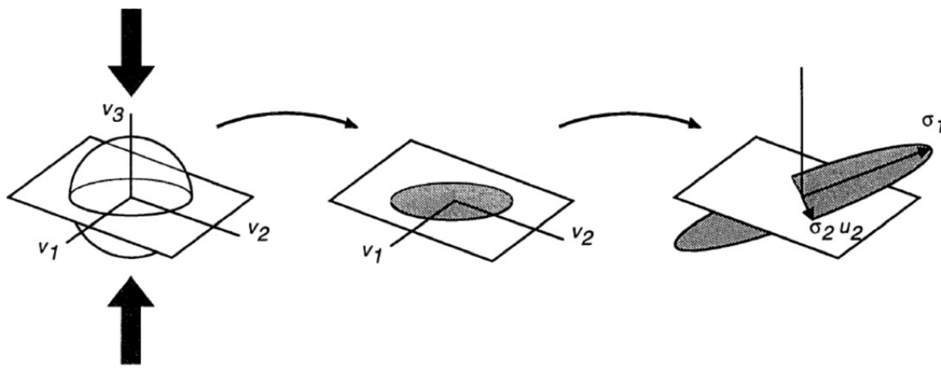


Figure 20: Geometric illustration of the SVD for $n = m = 3$, $k = 2$.

- Geometrically, consider how the SVD transforms a vector on the unit sphere:
  - Let $\boldsymbol{z} = z_1\boldsymbol{v}_1 + z_2\boldsymbol{v}_2 + \cdots + z_n\boldsymbol{v}_n$ for $\sum_i z_i^2 = \boldsymbol{z}^T\boldsymbol{z} = 1$

- $\boldsymbol{A}\boldsymbol{z} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T\boldsymbol{z}$
  $= \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T(\boldsymbol{z} = z_1\boldsymbol{v}_1 + z_2\boldsymbol{v}_2 + \cdots + z_n\boldsymbol{v}_n)$
  $= \sigma_1 z_1 \boldsymbol{u}_1 + \cdots + \sigma_k z_k \boldsymbol{u}_k$
  $= w_1\boldsymbol{u}_1 + w_2\boldsymbol{u}_2 + \cdots + w_k\boldsymbol{u}_k$
- Since $\sum_{i=1}^{k} \frac{w_i^2}{\sigma_i}^2 = \sum_{i=1}^{k} z_i^2 \leq \sum_{i=1}^{n} z_i^2 = 1$ we have an ellipsoid in k dimensions
  * If $k = n$ (full rank), then we have an equality, so we get the surface of the ellipsoid (no collapse)
  * If $k < n$, we have the inequality so we get the solid interior of the ellipsoid (some dimensions are collapsed)
- We can interpret the SVD as first collapsing the unit sphere by $n - k$ dimensions, then stretching the remaining k dimensions and then embedding the result in $\mathbb{R}^m$

Applications of the SVD

- SVD has many applications, the most common of which are dimensionality reduction techniques – we can throw away parts of the matrix that are less important

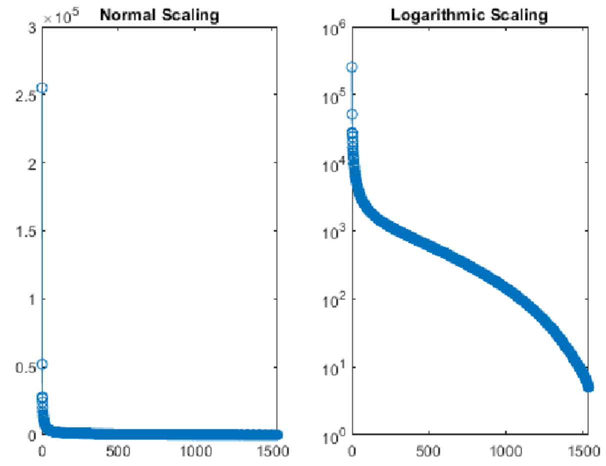


Figure 21: Typical singular value spectrum of an image.

- Image compression: treat the image as a big matrix, take the SVD, and truncate the singular values below a certain threshold
  - In general with data matrices we don't really have a clear "rank", so we will see a "continuous" spectrum of singular values – they decrease relatively smoothly instead of having a cutoff
  - We can now store the singular values and singular vectors above a certain threshold instead of the full image matrix
  - SVD compression is good at picking up patterns that align with the axes of the image (since these correspond to lower rank patterns)
    * e.g. a checkerboard would be very efficient when compressed since it has an effective rank of almost 1; but if we warp this checkerboard so that it is no longer axis-aligned, it becomes worse
- Principal component analysis (PCA): given $\boldsymbol{A}$ as the covariance matrix for a large number of high dimensional data points, we can use an SVD to get a lower dimensional subset that gives us more insight
  - The axes of the SVD are the axes of the error ellipsoid and the singular values are how large the ellipsoid is along each axis

- Dynamic mode decomposition (DMD): finding the best linear operator that represents the nonlinear dynamics of a system: $\dot{\boldsymbol{z}} = \boldsymbol{f}(\boldsymbol{z}) \leftrightarrow \dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x}$
  - The system dynamics are often lifted into higher dimensional space where things become more linear
  - We get *emergent dynamics* from the system, e.g. the behaviour of vortices shed by an object
  - DMD uses a large number of samples of the time series evolution of the dynamics in higher dimensional space: $\{ \boldsymbol{x}(t_0), \boldsymbol{x}(t_1), \dots, \boldsymbol{x}(t_N) \} = \{ \boldsymbol{x}_0, \boldsymbol{x}_1, \dots, \boldsymbol{x}_N \}$
    - Each $\boldsymbol{x}$ is a stacked vector of all the data (state) of the system at a particular time sample
  - Assume there is some matrix $\boldsymbol{A}$ such that $\boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k$
    - $\boldsymbol{X'} = \boldsymbol{A}\boldsymbol{X}$ where $\boldsymbol{X'} = \begin{bmatrix}\boldsymbol{x}_1 & \cdots & \boldsymbol{x}_N\end{bmatrix}, \boldsymbol{X} = \begin{bmatrix}\boldsymbol{x}_0 & \cdots & \boldsymbol{x}_{N-1}\end{bmatrix}$
  - Now we need to find $\boldsymbol{A}$, but it can be very large, so we can approximate it with a smaller matrix $\boldsymbol{A}_r$ using the SVD of the data matrix
    - $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T \implies \boldsymbol{X'} = \boldsymbol{A}\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$
    - Truncate the SVD to get the dominant modes: $\boldsymbol{X'} \approx \boldsymbol{A}\boldsymbol{U}_r\boldsymbol{\Sigma}_r\boldsymbol{V}^T$
    - Therefore $\boldsymbol{U}_r^T\boldsymbol{X'}\boldsymbol{V}_r\boldsymbol{\Sigma}_r^{-1} = \boldsymbol{U}_r^T\boldsymbol{A}\boldsymbol{U}_r = \boldsymbol{A}_r$
  - Now we have the transition matrix we can perform an eigendecomposition $\boldsymbol{A}_r\boldsymbol{W} = \boldsymbol{W}\boldsymbol{\Lambda}$ to look at its modes
    - Map the eigenspace back to the original space: $\boldsymbol{\Phi} = \boldsymbol{X'}\boldsymbol{V}_r\boldsymbol{\Sigma}_r^{-1}\boldsymbol{W}$
    - The eigenvalues corresponding these modes allow us to see how they evolve – whether they grow or shrink with time, oscillations, etc, just like a linear system
  - This can also be used to predict the future steps of the dynamics
- Frame-to-frame visual odometry: finding the coordinate transformation that maps one point cloud to another
  - This can be performed using an SVD of the point cloud data

Lecture 16, Nov 1, 2023

Probability Review

- Example: a man has M pairs of pants and L shirts; over time we observe that out of N observations, $n_{ps}(i, j)$ is the number of times he wore pants i with shirt j, $_p$ is the number of pants he wore pants i, and $n_s(j)$ is the number of times he wore shirt j
  - Let $f_{ps}(i, j) = \dfrac{n_{ps}(i, j)}{N}$ be the likelihood of wearing pants i with shirt j
  - Let $f_p(i) = \dfrac{n_p(i)}{N}$, $f_s(i) = \dfrac{n_s(j)}{N}$
  - Note that all the frequencies/likelihoods are nonnegative, and summing over all i or j gets us 1
  - $n_p(i) = \displaystyle\sum_{j=1}^{L} n_{ps}(i, j), n_s(j) = \sum_{i=1}^{M} n_{ps}(i, j)$
  - Therefore $f_p(i) = \displaystyle\sum_{j=1}^{L} f_{ps}(i, j), f_s(j) = \sum_{i=1}^{M} f_{ps}(i, j)$
    - This is known as *marginalization* or the *sum rule*
- There are two main ways of thinking about probability: *frequentist*, which examines the relative frequency of events occurring in a large number of trials; or *Bayesian*, which thinks about probability in terms of belief and uncertainty
- Let $\mathcal{X}$ be the set of all possible outcomes; let $f_x(\cdot)$ be the *probability density function* (PDF), a real-valued function that is nonnegative and sums to 1
  - $f_x(\cdot)$ and $\mathcal{X}$ define a discrete random variable (DRV) x
  - The PDF defines the notion of probability: $\Pr(x = \bar{x}) = f_x(\bar{x})$
  - We typically use x to denote the variable and a value it can take, e.g. $\Pr(x) = f_x(x)$; we will often also drop the subscript
- We can form a joint PDF $f_{xy}(x, y)$ over two variables
  - $f_{xy}(x, y) \geq 0$ as usual

- $\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} f_{xy}(x,y) = 1$
- *Marginalization* (*sum rule*): $f_x(x) = \sum_{y \in \mathcal{Y}} f_{xy}(x,y)$
  - f_x is fully defined by f_{xy}; consider this as a definition
- *Conditioning* (*product rule*): $f_{x|y}(x,y) = f(x|y) = \frac{f_{xy}(x,y)}{f_y(y)}$
  - This is like taking a slice of the joint probability distribution, and normalizing so it becomes a proper distribution
  - "Probability of y given x"
  - $f(x,y) = f(x|y)f(y) = f(y|x)f(x)$ is known as *Bayes' rule*

Theorem

Total probability theorem:

$$f_x(x) = \sum_{y \in \mathcal{Y}} f_{x|y}(x|y) f_y(y)$$

- Continuous random variables have a continuous range of values and must be integrated instead of summed
  - Note that we will assume $f_x(x)$ is uniformly bounded and piecewise continuous; this excludes things like delta functions and distributions that go to infinity
  - For a CRV it makes no sense to talk about the probability of x being exactly some value
  - Instead we use intervals $\Pr(x \in [a,b]) = \int_a^b f_x(x)\,\mathrm{d}x$
  - All other properties for discrete random variables apply, with integrals instead of sums
- We can mix continuous and discrete random variables in a joint probability distribution

Lecture 17, Nov 3, 2023

Properties of Conditional PDFs

- Conditional PDFs can be treated just like regular PDFs; the condition parameterizes the PDF
  - Marginalization: $f_{x|z}(x|z) = \sum_{y \in \mathcal{Y}} f_{xy|z}(x,y|z)$
  - Conditioning: $f_{x|yz}(x|y,z) = \frac{f_{xy|z}(x,y|z)}{f_{y|z}(y|z)}$
  - Notice that if we remove z we just get the regular marginalization and conditioning laws
    - The parameter z can be e.g. a mean or standard deviation for a normal distribution, etc
- Note that everything to the left of the bar is a random variable and everything to the right is a fixed conditioning variable (the bar has lower "precedence" than the comma)
- $f_{xyz}(x,y,z) = f_{xy|z}(x,y|z)f_z(z) \implies f_{xz}(x,z) = \sum_{y \in \mathcal{Y}} f_{xyz}(x,y,z) = \sum_{y \in \mathcal{Y}} f_{x,y|z}(x,y|z)f_z(z) \implies f_{x|z}(x|z) = \frac{f_{xz}(x,z)}{f_z(z)} = \sum_{y \in \mathcal{Y}} f_{x,y|z}(x,y|z)$
- $f_{x|yz}(x|y,z) = \frac{f_{xyz}(x,y,z)}{f_{yz}(y,z)} = \frac{f_{xy|z}(x,y|z)f_z(z)}{f_{y|z}(y|z)f_z(z)} = \frac{f_{xy|z}(x,y|z)}{f_{y|z}(y|z)} \implies f_{xy|z}(x,y|z) = f_{x|yz}(x|y,z)f_{y|z}(y|z)$
  - $f_{x|yz}(x|y,z)$ is conditioned on both y and z; so we can think of the above as multiplying by a distribution of y moves y to the left of the bar
  - Since both distributions are conditioned on z, the resulting distribution will also be conditioned on z
- Random variables x and y are *independent* if $f(x|y) = f(x)$, i.e. knowing y does not give us additional information about x

- $f(x, y) = f(x|y)f(y) = f(x)f(y)$ is an equivalent definition
  - This also implies that $f(y|x) = f(y)$
- Random variables x and y are *conditionally independent* on z if $f(x|y, z) = f(x|z)$, i.e. knowing z makes x and y independent (y has no information on x if we already know z)
  - This is equivalent to $f(x, y|z) = f(x|z)f(y|z)$ similar to above
- Independence greatly simplifies algorithms and allows us to decouple information and processes
  - Suppose $y_i = g_i(x, w_i)$ where y_i are measurements, x is the state and w_i are noise
  - $f(y_1, \cdots, y_N|x) = \prod_{i=1}^{N} f(y_i|x)$ if we have independence, which greatly simplifies the problem

Bayes' Theorem

- $f(x|y)f(y) = f(y|x)f(x) \implies f(x|y) = \dfrac{f(y|x)f(x)}{f(y)}$ is *Bayes' theorem* (or rule)
- This applies to both continuous and discrete and mixed PDFs
- Example: disease diagnosis
  - Doctors were asked to estimate the probability that a woman with no symptoms between 40 and 50 years old with a positive mammogram actually has breast cancer
  - Given: 7% of mammograms give false positives, 10% of mammograms gives false negatives, actual incidence of breast cancer in this age group is 0.8%
  - Let x be whether the patient has cancer (0) or not (1), and let y be whether the test is negative (1) or positive (0)
  - We want $f_{x|y}(0, 0)$, i.e. the patient has cancer given a positive test
  - The 7% false positives is $f_{y|x}(0|1)$, the 10% false negatives is $f_{y|x}(1|0)$, the 0.8% is $f_x(0)$
  - $f_{x|y}(0|0) = \dfrac{f_{y|x}(0|0)f_x(0)}{f_y(0)}$
    * $f_{y|x}(1|0) = 0.10 \implies 1 - f_{y|x}(1|0) = f_{y|x}(0|0) = 0.90$
    * $f_x(0) = 0.008 \implies 1 - f_x(0) = f_x(1) = 0.992$
    * $f_{y|x}(0|1) = 0.007 \implies f_{y|x}(0|1)f_x(1) + f_{y|x}(0|0)f_x(0) = f_y(0) = 0.07664$
  - Using these numbers we get $f_{x|y}(0, 0) = 9.4\%$
  - Since the actual probability of cancer is very low, very few positives will be due to true positives
- Example: girls named Lulu
  - A family has two children; given that one is a girl, what is the probability that both are girls?
    * Let x be 1 if there are no boys in the family and 0 if there are boys in the family
    * Let y be 1 if there are no girls in the family and 0 if there are girls in the family
    * $f_y(0) = \dfrac{3}{4}, f_y(1) = \dfrac{1}{4}$ because there are 4 cases, 3 of which have at least 1 girl
    * $f_x(1) = \dfrac{1}{4}, f_x(0) = \dfrac{3}{4}$ similarly
    * $f_{y|x}(0|1) = 1$ since if there are no boys, there must be girls
    * We want to know $f_{x|y}(1|0) = \dfrac{f_{y|x}(0|1)f_x(1)}{f_y(0)} = \dfrac{1}{3}$
  - If we are given that one of them is a girl named Lulu (given that this is an uncommon name), what is the probability that both are girls?
    * Let x be 1 if there are no boys in the family and 0 if there are boys in the family
    * Let y be 1 if there are no girls named Lulu in the family and 0 if there are
    * $f_{x|y}(1|0) = \dfrac{f_{y|x}(0|1)f_x(1)}{f_y(0)}$
      - $f_{y|x}(0|1)$ is the probability that given 2 girls, at least one is named Lulu
        - If $p \ll 1$ is the probability of a girl being named Lulu
        - We can enumerate all possible outcomes:
          * Both children not named Lulu: $(1 - p)(1 - p)$
          * First child not Lulu, second child Lulu: $(1 - p)p$
          * First child Lulu, second child not Lulu: p

        * Both children named Lulu: 0
    - Therefore $f_{y|x}(0|1) = 2p - p^2 \approx 2p$
- $f_y(0) = f_{y|x}(0|0)f_x(0) + f_{y|x}(0|1)f_x(1) = p - \frac{1}{4}p^2$
    - $f_{y|x}(0|0)$ has at least one boy in the family, so the probability of having a girl is $\frac{2}{3}$ so having a girl named Lulu is $\frac{2}{3}p$
    - $f_x(0) = \frac{3}{4}, f_x(1) = \frac{1}{4}$
        * This gives us $f_{x|y}(1|0) \approx \frac{1}{2}$

Lecture 18, Nov 17, 2023

Mean, Variance, and Change of Variables

Definition

The *expected value* of $\boldsymbol{x}$ over a distribution $f_x(\boldsymbol{x})$ is defined as:

$$E[\boldsymbol{x}] = \sum_{\boldsymbol{x} \in \mathcal{X}} x f_x(\boldsymbol{x})$$

The sum is replaced by an integral for a continuous distribution.

The *variance* of $\boldsymbol{x}$ is defined as

$$\operatorname{Var}[\boldsymbol{x}] = E[(\boldsymbol{x} - E[\boldsymbol{x}])(\boldsymbol{x} - E[\boldsymbol{x}])^T]$$

- $E[\boldsymbol{x}]$ is also known as the *mean* and is generally a vector
- $\operatorname{Var}[\boldsymbol{x}]$ is generally a matrix; called a *covariance* when a matrix
    - Diagonal entries are variances in each entry of $\boldsymbol{x}$ while the off-diagonal entries describe the correlation in the variances
    - If we look at a Gaussian, the spread looks like an ellipse; the eigenvalues describe the length of the axes of the ellipse, while the eigenvectors describe how it's aligned/skewed
- Mean and variance are first and second-order *moments* of $\boldsymbol{x}$; we can also have higher order moments
- If $\mathcal{Y} = \{\, y \mid y = g(x), x \in \mathcal{X} \,\}$, then $E[y] = E[g(x)]$; i.e. to find the mean of y we don't need to find its PDF, we just need to apply g to every element of $\mathcal{X}$
    - This is known as the *Law of the Unconscious Statistician*
- Let $f_y(y)$ be a discrete PDF; consider some $x = g(y)$, then what is $f_x(x)$?
    - We assume that multiple y values can map to the same x (but the same y can't map to multiple x)
    - Let $\mathcal{X} = g(\mathcal{Y})$; for each $x_j \in \mathcal{X}$, let $\mathcal{Y}_j = \{\, y_{j,i} \,\}$ be the set of all $y \in \mathcal{Y}$ such that $g(y_{j,i}) = x_j$ (i.e. $\mathcal{Y}_j$ contains all elements in $\mathcal{Y}$ that map to x_j)
    - Claim: $f_x(x_j) = \sum_{y_{j,i}} f_y(y_{j,i})$, that is, to find the probability of x_j we just sum the probabilities of all $y_{j,i}$ that map to it
        * $f_x(x_j) = \Pr(x = x_j) = \Pr(y \in \mathcal{Y}_j) = \sum_{y_{j,i} \in \mathcal{Y}_j} f_y(y_{j,i})$
        * Assume $\mathcal{Y}_j \cap \mathcal{Y}_k = \varnothing$ when $j \neq k$, and $\mathcal{Y}_1 \cup \mathcal{Y}_2 \cup \cdots \cup \mathcal{Y}_n = \mathcal{Y}$ (this is true because the same $\mathcal{Y}$ can't map to multiple $\mathcal{X}$)
        * $\sum_{x_j \in \mathcal{X}} f_x(x_j) = \sum_{x_j \in \mathcal{X}} \sum_{y_{j,i} \in \mathcal{Y}_j} f_y(y_{j,i}) = \sum_{j=1}^{m} \sum_{y_{j,i} \in \mathcal{Y}_j} f_y(y_{j,i}) = \sum_{y \in \mathcal{Y}} f_y(y) = 1$
- For a continuous probability distribution, we assume $g(y)$ is continuously differentiable and strictly

monotonic (i.e. strictly increasing or decreasing) and that $f_y(y)$ is continuous

- Claim: $f_x(x) = \frac{f_y(y)}{\frac{\mathrm{d}g(y)}{\mathrm{d}y}}$
  * $\Pr(y \in [\bar{y}, \bar{y} + \Delta y]) = \int_{\bar{y}}^{\bar{y}+\Delta y} f_y(y)\,\mathrm{d}y \approx f_y(\bar{y})\Delta y$
  * Let $\bar{x} = g(\bar{y})$ and $g(\bar{y} + \Delta y) = g(\bar{y}) + \frac{\mathrm{d}g(\bar{y})}{\mathrm{d}y}\Delta y = \bar{x} + \Delta x$
  * $\Pr(x \in [\bar{x}, \bar{x} + \Delta x]) = \int_{\bar{x}}^{\bar{x}+\Delta x} f_x(x)\,\mathrm{d}x \approx f(\bar{x})\Delta x$
  * But we also have $\Pr(x \in [\bar{x}, \bar{x}+\Delta x]) = \Pr(y \in [\bar{y}, \bar{y}+\Delta y])$ because these are the same intervals
  * $f_x(\bar{x})\Delta x = f_y(\bar{x})\Delta y \implies f_x(\bar{x})\frac{\mathrm{d}g(\bar{y})}{\mathrm{d}y}\Delta y = f_y(\bar{y})\Delta y \implies f_x(x) = \frac{f_y(y)}{\frac{\mathrm{d}g(y)}{\mathrm{d}y}}$
- We can also think about this as a change of variables in the integral; we need $\int_{\mathcal{Y}} f_y(y)\,\mathrm{d}y = 1$, and since $x = g(y)$, $\frac{\mathrm{d}g(y)}{\mathrm{d}y}\,\mathrm{d}y \implies \mathrm{d}y = \frac{1}{\frac{\mathrm{d}g(y)}{\mathrm{d}y}}\,\mathrm{d}x$, then $\int_{\mathcal{X}} f_y(y)\frac{1}{\frac{\mathrm{d}g(y)}{\mathrm{d}y}}\,\mathrm{d}x = 1$, so the expression inside the integral must be the PDF of x

Bayes' Theorem in Practice

- We will use Bayes' Theorem to create a recursive filter; given a prior belief distribution and some measurement info, we use Bayes' Theorem to construct a new posterior belief/distribution
  - The measurement info itself is probabilistic since there may be errors
  - This can be done in a variety of ways, e.g. particle filters, Kalman filter
- $f(x|y) = \frac{f(y|x)f(x)}{f(y)}$
  - x is some unknown quantity of interest, e.g. the system state
  - y is some observation related to the state, e.g. sensor measurements
  - $f(x)$ is some prior belief
  - $f(y|x)$ is the observation model; for each state x, what is the likelihood of observing y?
  - $f(x|y)$ is the posterior belief, which takes the new observation into account
  - $f(y) = \sum_x f(y|x)f(x)$ is the probability of observing y (independent of x); this can be seen as a normalization constant since it's a constant multiplier as far as x is involved
- Given $f(x|y)$, we can then find the "most likely" state; this is often defined as the mode or mean
- We can generalize to N observations $y_1, \dots, y_N$, each of which may be vector-valued; assume conditional independence so that $f(y_1, \dots, y_N|x) = f(y_1|x) \cdots f(y_N|x)$
  - The conditional independence means that the noise corrupting each state x is independent
  - $y_i = g_i(x, w_i)$ where w_i are noise; then we assume $f(w_1, \dots, w_N) = f(w_1) \cdots f(w_N)$
- Then $f(x|y_1, \dots, y_N) = \frac{f(x)\prod_i f(y_i|x)}{f(y_i, \dots, y_N)} = \frac{f(x)\prod_i f(y_i|x)}{\sum_{x\in\mathcal{X}} f(x)\prod_i f(y_i|x)}$
- Example: Let $x \in \{0, 1\}$ represent the truthful answer to a question (0 – no, 1 – yes); the response (i.e. observation) from person i modelled as $y_i = x + w_i$, where w_i is some independent noise (0 – truth, 1 – lie)
  - Note the + operator works like an XOR here
  - We ask 2 people the same question, and estimate what the truth is
  - The prior is $f(x) = \frac{1}{2}$ for both $x = 0, 1$ (i.e. we have no information and all states are equally likely)
  - We model the truthfulness as $f_{w_i}(0) = p_i, f_{w_i}(1) = 1 - p_i$, i.e. person i tells the truth with probability p_i
  - By Bayes' theorem $f(x|y_1, y_2) = \frac{f(x)f(y_1|x)f(y_2|x)}{f(y_1, y_2)}$

  - We build tables for the numerator and denominator to find the probabilities (see figure below)
    * Note terms in the table on the right are obtained by summing over all possible values of x; e.g. for $y_1 = 0, y_2 = 0$ we are taking the sum of $x = 0, y_1 = 0, y_2 = 0$ and $x = 1, y_1 = 0, y_2 = 0$ in the left table
  - Note that all the information in the tables could have been obtained from just $f(x|y_1), f(x|y_2)$

x	y_1	y_2	$f(x)\,f(y_1\|x)\,f(y_2\|x)$
0	0	0	$0.5p_1p_2$
0	0	1	$0.5p_1(1-p_2)$
0	1	0	$0.5(1-p_1)p_2$
0	1	1	$0.5(1-p_1)(1-p_2)$
1	0	0	$0.5(1-p_1)(1-p_2)$
1	0	1	$0.5(1-p_1)p_2$
1	1	0	$0.5p_1(1-p_2)$
1	1	1	$0.5p_1p_2$

y_1	y_2	$f(y_1, y_2)$
0	0	$0.5\left(p_1p_2 + (1-p_1)(1-p_2)\right)$
0	1	$0.5\left(p_1(1-p_2) + (1-p_1)p_2\right)$
1	0	$0.5\left((1-p_1)p_2 + p_1(1-p_2)\right)$
1	1	$0.5\left((1-p_1)(1-p_2) + p_1p_2\right)$

Figure 22: Probability tables for the example problem.

Lecture 19, Nov 22, 2023

Bayesian Tracking

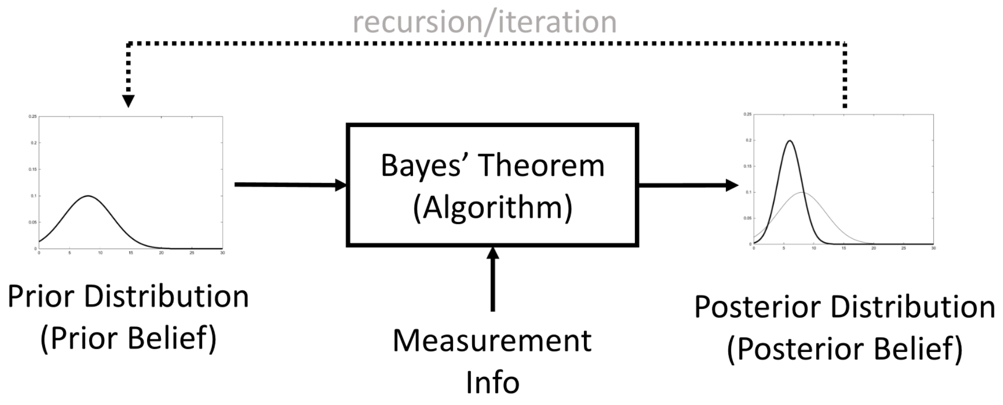

recursion/iteration
Bayes' Theorem (Algorithm)
Prior Distribution (Prior Belief)
Measurement Info
Posterior Distribution (Posterior Belief)

Figure 23: High-level overview of Bayesian localization.

- We wish to derive a recursive state estimation algorithm (i.e. iterating at each timestep) for a system with a finite state space, consisting of two main steps:
  1. The *prior update*, where the state estimate is predicted forward using the process model
  2. The *measurement update*, where the prior is combined with observation and measurements to correct it
- Let $\boldsymbol{x}_k \in \mathcal{X}$ be the vector-valued state at time k (assumed discrete, i.e. $\mathcal{X}$ is finite); let $\boldsymbol{y}_k$ be a vector-valued measurement that we can observe (continuous or discrete)
- We have a motion model $\boldsymbol{x}_k = \boldsymbol{f}_{k-1}(\boldsymbol{x}_{k-1}, \boldsymbol{v}_{k-1})$ and the observation model $\boldsymbol{y}_k = \boldsymbol{h}_k(\boldsymbol{x}_k, \boldsymbol{w}_k)$, where $\boldsymbol{v}_k, \boldsymbol{w}_k$ are independent noise terms with known PDFs; we also assume noise is independent of the initial condition $\boldsymbol{x}_0$
  - Note $\boldsymbol{u}_{k-1}$ is not explicitly included, but we can incorporate it by absorbing it into $\boldsymbol{f}_{k-1}$ and $\boldsymbol{h}_k$
- Let $\boldsymbol{y}_{1:k} = \{\, \boldsymbol{y}_1, \dots, \boldsymbol{y}_k \,\}$; we want to calculate $f(\boldsymbol{x}_k|\boldsymbol{y}_{1:k})$, i.e. the probability distribution of the state at time k, given all our measurements
- Assuming the *Markov property* (i.e. each state only depends on the prior state, and not the state history), we can formulate the problem as computing $f(\boldsymbol{x}_k|\boldsymbol{y}_{1:k})$ from $f(\boldsymbol{x}_{k-1}|\boldsymbol{y}_{1:k-1})$

- Prior update: compute $f(\boldsymbol{x}_k|\boldsymbol{y}_{1:k-1})$ in terms of $f(\boldsymbol{x}_{k-1}|\boldsymbol{y}_{1:k-1})$
  - By total probability, $f(\boldsymbol{x}_k|y_{1:k-1}) = \displaystyle\sum_{x_{k-1}\in\mathcal{X}} f(\boldsymbol{x}_k|\boldsymbol{x}_{k-1},\boldsymbol{y}_{1:k-1})f(\boldsymbol{x}_{k-1}|\boldsymbol{y}_{1:k-1})$
    * i.e. we introduce $\boldsymbol{x}_{k-1}$ and marginalize across it
  - $\boldsymbol{x}_k$ and $\boldsymbol{y}_{1:k-1}$ are conditionally independent given $\boldsymbol{x}_{k-1}$, because the distribution of $\boldsymbol{x}_{k-1}$ already incorporates the information from all previous measurements
    * $\boldsymbol{x}_k$ is a function of $\boldsymbol{v}_{k-1}$ only (because $\boldsymbol{x}_{k-1}$ is known)
    * $\boldsymbol{y}_{k-1}$ is a function of $\boldsymbol{w}_{k-1}$
    * $\boldsymbol{y}_{k-2}$ is a function of $\boldsymbol{x}_{k-2}$ and $\boldsymbol{w}_{k-2}$, but $\boldsymbol{x}_{k-2}$ is a function of $\boldsymbol{x}_{k-3}$ and $\boldsymbol{v}_{k-3}$, and so on
    * Therefore $\boldsymbol{y}_{1:k-1}$ is a function of $\boldsymbol{x}_{k-1}, \boldsymbol{v}_{1:k-3}, \boldsymbol{w}_{1:k-1}, \boldsymbol{x}_0$
    * $\boldsymbol{x}_k$, and $y_{1:k-1}$ depend only on random variables that are independent, so these two variables must be independent
  - Therefore the prior update is $f(\boldsymbol{x}_k|\boldsymbol{y}_{1:k-1}) = \displaystyle\sum_{x_{k-1}\in\mathcal{X}} f(\boldsymbol{x}_k|\boldsymbol{x}_{k-1})f(\boldsymbol{x}_{k-1}|\boldsymbol{y}_{1:k-1})$
    * The distribution $f(\boldsymbol{x}_k|\boldsymbol{x}_{k-1})$ can be calculated exactly from our process model and noise distribution using change of variables
- Measurement update: compute $f(\boldsymbol{x}_k|y_{1:k-1})$, given $\boldsymbol{y}_k$ and $f(\boldsymbol{x}_k|\boldsymbol{y}_{1:k-1})$
  - Using Bayes' rule, $f(\boldsymbol{x}_k|\boldsymbol{y}_{1:k}) = f(\boldsymbol{x}_k|\boldsymbol{y}_k, \boldsymbol{y}_{1:k-1})$
    $$= \frac{f(\boldsymbol{y}_k|\boldsymbol{x}_k,\boldsymbol{y}_{1:k-1})f(\boldsymbol{x}_k|\boldsymbol{y}_{1:k-1})}{f(\boldsymbol{y}_k|\boldsymbol{y}_{1:k-1})}$$
  - Once again, $\boldsymbol{y}_k$ and $\boldsymbol{y}_{1:k-1}$ are conditionally independent, given $\boldsymbol{x}_k$
    * $\boldsymbol{y}_k$ is a function of only $\boldsymbol{w}_k$, if given $\boldsymbol{x}_k$
    * Using a similar procedure we can show $\boldsymbol{y}_{1:k-1}$ is a function of $\boldsymbol{v}_{0:k-2}, \boldsymbol{w}_{1:k-1}, \boldsymbol{x}_0$, all of which are independent of $\boldsymbol{w}_k$
    * Therefore $\boldsymbol{y}_k, \boldsymbol{y}_{1:k-1}$ are conditionally independent on $\boldsymbol{x}_k$
    * $f(\boldsymbol{y}_k|x_k, \boldsymbol{y}_{1:k-1}) = f(\boldsymbol{y}_k|\boldsymbol{x}_k)$, and can be computed from our measurement model
  - The term in the denominator is simply a normalization constant
    * We can compute it as $f(\boldsymbol{y}_k|\boldsymbol{y}_{1:k-1}) = \displaystyle\sum_{\boldsymbol{x}_k\in\mathcal{X}} f(\boldsymbol{y}_k|\boldsymbol{x}_k)f(\boldsymbol{x}_k|\boldsymbol{y}_{1:k-1})$ by total probability
  - Therefore the measurement update is $f(\boldsymbol{x}_k|\boldsymbol{y}_{1:k}) = \dfrac{f(\boldsymbol{y}_k|\boldsymbol{x}_k)f(\boldsymbol{x}_k|\boldsymbol{y}_{1:k-1})}{\sum_{\boldsymbol{x}_k\in\mathcal{X}} f(\boldsymbol{y}_k|\boldsymbol{x}_k)f(\boldsymbol{x}_k|\boldsymbol{y}_{1:k-1})}$

Implementation

- Enumerate the state as $\mathcal{X} = \{1, 2, \dots, N\}$
- Define $\boldsymbol{a}^i_{k|k} = \Pr(\boldsymbol{x}_k = i|\boldsymbol{y}_{1:k-1}), i = 1, \dots, N$ as an array of N elements in which we store the posterior
  - Initialize $\boldsymbol{a}^i_{0|0} = \Pr(\boldsymbol{x}_0 = i)$
- Define $\boldsymbol{a}^i_{k|k-1} = \Pr(\boldsymbol{x}_k = i|\boldsymbol{x}_{1:k-1}), i = 1, \dots, N$ to store the prior
- Recursive update:
  - $\boldsymbol{a}^i_{k|k-1} = \displaystyle\sum_{j=1}^{N} \Pr(\boldsymbol{x}_k = i|\boldsymbol{x}_{k-1} = j)\boldsymbol{a}^j_{k-1|k-1}$
    * $\Pr(\boldsymbol{x}_k = i|\boldsymbol{x}_{k-1} = j)$ can be calculated from $\boldsymbol{x}_k = \boldsymbol{f}_{k-1}(\boldsymbol{x}_{k-1}, \boldsymbol{v}_{k-1})$ and the distribution of $\boldsymbol{v}_k$
  - $\boldsymbol{a}^i_{k|k} = \dfrac{f(\boldsymbol{y}_k|\boldsymbol{x}_k = i)\boldsymbol{a}^i_{k|k-1}}{\sum_{j=1}^{N} f(\boldsymbol{y}_k|\boldsymbol{x}_k = j)\boldsymbol{a}^j_{k|k-1}}$
    * $f(\boldsymbol{y}_k|\boldsymbol{x}_k = i)$ can be calculated from $\boldsymbol{y}_k = \boldsymbol{h}_k(\boldsymbol{x}_k, \boldsymbol{w}_k)$ and the distribution of $\boldsymbol{w}_k$
- Example: consider an object moving randomly on a circle, in discrete steps; our measurement is the distance to the object from a distance sensor located at $(L, 0)$
  - Let x_k be the object's location on the circle, then $\theta_k = \dfrac{2\pi x_k}{N}$
  - Set up the models:
    * The process model is $x_k = f(x_{k-1}, v_{k-1}) = (x_{k-1} + v_{k-1}) \bmod N$

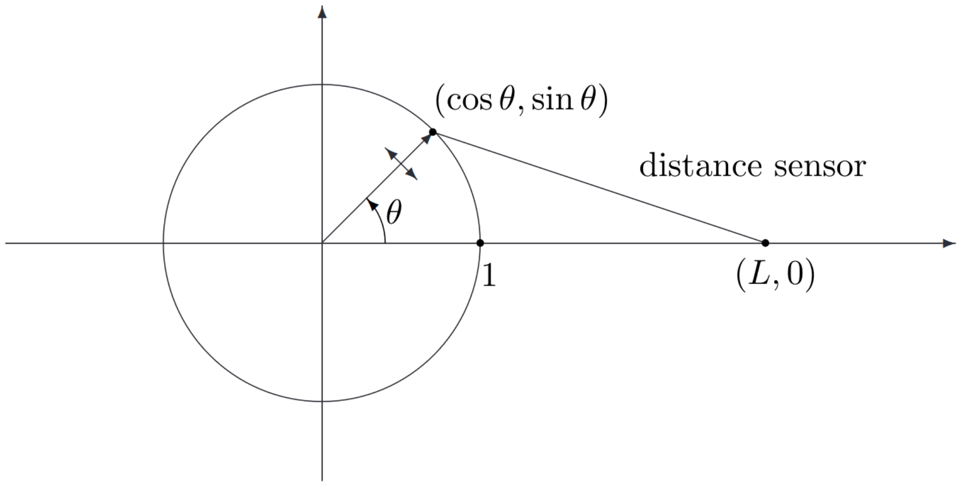


Figure 24: Setup for the example problem.

    * The process noise is 1 with probability p, and -1 with probability $1 - p$
    * The measurement model is $y_k = h(x_k, w_k) = \sqrt{(L - \cos\theta_k)^2 + \sin^2\theta_k} + w_k$
    * The measurement noise is uniformly distributed over $[-e, e]$
  - Using a change of variables we can now compute the PDFs of the process and sensor models
    * $f(x_k|x_{k-1}) = \begin{cases} p & x_k = (x_{k-1} + 1) \bmod N \\ 1 - p & x_k = (x_{k-1} - 1) \bmod N \\ 0 & \text{otherwise} \end{cases}$
    * $f(y_k|x_k) = \begin{cases} \frac{1}{2e} & \left|y_k - \sqrt{(L - \cos\theta_k)^2 + \sin\theta^2\theta_k}\right| \leq e \\ 0 & \text{otherwise} \end{cases}$
  - Initialize as $f(x_0) = \frac{1}{N} \forall x_0 \in \{ 0, 1, \dots, N - 1 \}$ which assumes a state of maximum ignorance

Lecture 20, Nov 24, 2023

Sampling Distributions in Practice

- Most math libraries have functions that generate uniformly distributed random real numbers in the range $(0, 1)$
  - e.g. `rand()` in MATLAB, `np.random.rand()` in Python
  - This interval will sometimes be closed or half-open, but practically we don't care
  - $f_u(u) = \begin{cases} 1 & u \in (0, 1) \\ 0 & \text{otherwise} \end{cases}$
- Repeated calls to the RNG are independent
- The generator can usually be seeded, e.g. with `np.random.seed()`; this gives the same sequence of random numbers for the same seed
- How do we draw samples from arbitrary, non-uniform PDFs?

One Variable, Discrete

- Given a desired PDF $\hat{f}_x(x)$ for a DRV x, we want to come up with a procedure to generate x from u
- WLOG let $\mathcal{X} = \mathbb{Z}$ (note we can remap any DRV to be over the integers)
- The *cumulative distribution function* (CDF) of $\hat{f}_x$ is $\hat{F}_x(x) = \sum_{\bar{x}=-\infty}^{x} \hat{f}_x(\bar{x})$

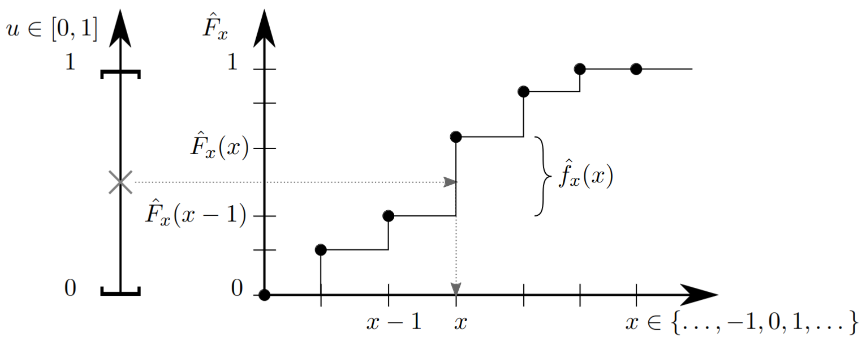


Figure 25: Algorithm to select one discrete random variable.

  - Note $\hat{F}_x(-\infty) = 0$ and $\hat{F}_x(\infty) = 1$, and $\hat{F}_x$ is a non-decreasing function
  - We will make use of the fact that both $\hat{F}_x$ and the output of the RNG range from 0 to 1
- Let u be generated from $f_u(u)$; solve for x such that $\hat{F}_x(x-1) < u, \hat{F}_x(x) \geq u$; we claim that x will have PDF $\hat{f}_x(x)$
  - Intuition: we chop up the interval $[0,1]$, so that each x gets a portion of the interval that is proportional to $\hat{f}_x(x)$
    * To see this note $\hat{F}_x(x) - \hat{F}_x(x-1) = \hat{f}_x(x)$
- Note that we can always solve for such an x given u, since $\hat{F}_x$ ranges from 0 to 1
  - There may be issues with $u = 0$ and $u = 1$, but this probability is technically 0 u is a CRV
  - In practice we can explicitly check for these cases and re-sample if we obtain them
- For a fixed x, to have $\hat{F}_x(x-1) < u \leq \hat{F}_x(x)$ we need $\hat{F}_x(x-1) < u \leq \hat{F}_x(x-1) + \hat{f}_x(x)$
  - Therefore $\displaystyle\int_{\hat{F}_x(x-1)}^{\hat{F}_x(x)} f_u(u)\,\mathrm{d}u = \int_{\hat{F}_x(x-1)}^{\hat{F}_x(x-1)+\hat{f}_x(x)} 1\,\mathrm{d}u = \hat{f}_x(x)$
  - So the probability that we get x is $\hat{f}_x(x)$

Multiple Variables, Discrete

- If we want $\hat{f}_{xy}(x,y)$
- If $\mathcal{X}$ and $\mathcal{Y}$ are finite, with N_x and N_y elements respectively, let $\mathcal{Z} = \{\,1, 2, \dots, N_xN_y\,\}$, and define a one-to-one mapping between elements of $\mathcal{Z}$ and (x,y), and sample with the one-variable algorithm
- Otherwise, decompose $\hat{f}_{xy}(x,y) = \hat{f}_{x|y}(x|y)\hat{f}_y(y)$
  - Apply the one-variable algorithm to sample y first from $\hat{f}_y(y)$ (obtained by marginalizing the joint PDF)
  - Then apply the same algorithm again to get a value for x from $\hat{f}_{x|y}(x|y)$
- These algorithms both apply to any number of DRVs

One Variable, Continuous

- Let $\hat{F}_x(x) = \displaystyle\int_{-\infty}^{x} \hat{f}_x(\bar{x})\mathrm{d}\bar{x}$
- Let u be generated from $f_u(u)$; then have $x = \hat{F}_x^{-1}(u)$, and x will have PDF $f_x(x) = \hat{f}_x(x)$
  - x is any value that satisfies $u = \hat{F}_x(x)$; this will still work even if $\hat{f}_x$ is zero sometimes
- Assume that $\hat{F}_x$ is strictly increasing, then we can solve for a unique x given any u
  - For some arbitrary a, $F_x(a) = \Pr(x \leq a) = \Pr(\hat{F}_x^{-1}(u) \leq a)$
  - Applying $\hat{F}_x$ to both sides, this becomes $\Pr(u \leq \hat{F}_x(a))$
  - Since u is uniform, $\Pr(u \leq \hat{F}_x(a)) = \hat{F}_x(a)$
  - Therefore $F_x(a) = \hat{F}_x(a)$, so we must have $f_x(x) = \hat{f}_x(x)$

Multiple Variables, Continuous

- For multiple CRVs, again decompose $\hat{f}_{xy}(x,y) = \hat{f}_{x|y}(x|y)\hat{f}_y(y)$, and apply the one-variable algorithm to get values for y first from $\hat{f}_y(y)$, and then x from $\hat{f}_{x|y}(x|y)$

Lecture 21, Nov 29, 2023

Extracting Estimates from PDFs

Maximum Likelihood (ML)

- This method is often used when $\boldsymbol{x}$ is an unknown constant parameter without a known probabilistic description, i.e. we have no prior information about $\boldsymbol{x}$
  - e.g. in Bayesian estimation, we had a prior (prediction) for $\boldsymbol{x}$, but here we are assuming no knowledge of that
- For a given observation $\boldsymbol{y}$ and observation model $f(\boldsymbol{y}|\boldsymbol{x})$, the method seeks a value of $\boldsymbol{x}$ that maximizes the likelihood of observing $\boldsymbol{y}$, i.e. $\hat{\boldsymbol{x}}^{ML} = \underset{\boldsymbol{x}\in\mathcal{X}}{\operatorname{argmax}} f(\boldsymbol{y}|\boldsymbol{x})$
  - $f(\boldsymbol{y}|\boldsymbol{x})$ as a function of $\boldsymbol{x}$ is the *likelihood function*
  - $\boldsymbol{x}$ is a parameter of the observation model; e.g. the model can be a Gaussian, and $\boldsymbol{x}$ may denote its mean or variance, etc
- Example: Consider two measurements of a scalar quantity $x \in \mathbb{R}$: $y_1 = x + w_1, y_2 = x + w_2$ where $w_1, w_2 \sim \mathcal{N}(0,1)$
  - Note $\mathcal{N}(\mu,\sigma)$ denotes a Gaussian with mean μ and variance σ
  - $f(w_i) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{w_i^2}{2}\right)$
  - We can consider w_1, w_2 as additive noise parameters; this essentially makes $y_i \sim \mathcal{N}(x,1)$
    - Note formally we would use a change of variables: $y_i = x + w_i \implies w_i = y_i - x$
    - Now we can just substitute w_i into the Gaussian equation since we have a linear relationship
  - y_1, y_2 are conditionally independent on x, so $f(y_1,y_2|x) = f(y_1|x)f(y_2|x) = \frac{1}{2\pi}e^{-\frac{1}{2}\left((y_1-x)^2+(y_2-x)^2\right)}$
  - This is now an unconstrained optimization problem; we can differentiate with respect to x and set this to 0
  - We get $(y_1 - \hat{x}) + (y_2 - \hat{x}) = 0 \implies \hat{x} = \frac{y_1+y_2}{2}$, which is just the average
- Suppose we generalize the last example to a collection of measurements $\boldsymbol{z} = \boldsymbol{H}\boldsymbol{x} + \boldsymbol{w}$ where $\boldsymbol{z}, \boldsymbol{w} \in \mathbb{R}^m, \boldsymbol{x} \in \mathbb{R}^n$ and $m > n$; as above $w_i \sim (0,1)$ are independent
  - Let $\boldsymbol{H} = \begin{bmatrix} \boldsymbol{h}_1^T \\ \vdots \\ \boldsymbol{h}_m^T \end{bmatrix}$ where $\boldsymbol{h}_i^T = \begin{bmatrix} h_{i_1} & \cdots & h_{in} \end{bmatrix}$
  - Then $z_i = \boldsymbol{h}_i^T\boldsymbol{x} + \boldsymbol{w}_i$
  - As before $f(\boldsymbol{z}|\boldsymbol{x}) \propto \exp\left(-\frac{1}{2}\left((z_1 - \boldsymbol{h}_1^T\boldsymbol{x})^2 + \cdots + (z_m - \boldsymbol{h}_m^T\boldsymbol{x})^2\right)\right)$
  - Differentiating with respect to each x_j we have $(z_1 - \boldsymbol{h}_1^T\hat{\boldsymbol{x}})h_{1j} + \cdots + (z_m - \boldsymbol{h}_m^T\hat{\boldsymbol{x}})h_{mj} = \begin{bmatrix} h_{1j} & \cdots & h_{mj} \end{bmatrix}(\boldsymbol{z} - \boldsymbol{H}\hat{\boldsymbol{x}}) = 0$
  - With all the rows, we get $\boldsymbol{H}^T(\boldsymbol{z} - \boldsymbol{H}\hat{\boldsymbol{x}}) = \boldsymbol{0} \implies \hat{\boldsymbol{x}} = (\boldsymbol{H}^T\boldsymbol{H})^{-1}\boldsymbol{H}^T\boldsymbol{z}$, which is the least squares solution
    - Note we can write $\boldsymbol{w}(\boldsymbol{x}) = \boldsymbol{z} - \boldsymbol{H}\boldsymbol{x}$, so $\boldsymbol{w}$ is some error term; then $\hat{\boldsymbol{x}} = \underset{\boldsymbol{x}}{\operatorname{argmin}}\, \boldsymbol{w}^T\boldsymbol{w}$
    - If not all the errors have the same variance, then we have weighted least squares
- Limitations of ML:
  - In general ML is more sensitive to outliers and modelling error
  - The maximum of the distribution may not always be what we want – we may lose robustness
    - In the example above, ML will give x_1 if there are measurements on it, which is very sensitive to changes in the data or model – small variations in the model might cause x_1 to have a

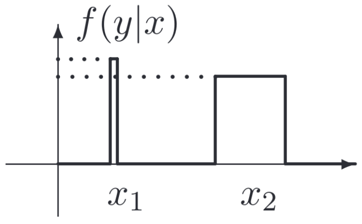


Figure 26: Undesirable case of maximum likelihood.

likelihood of zero instead
    * Choosing x_2 is more robust; since the distribution is wider, we're less sensitive to changes in the data or model
    * Outliers that happen to line up with a peak can give us an incorrect estimate
  - We might also have prior knowledge about x (i.e. its PDF), which ML cannot incorporate

Maximum *a Posteriori* (MAP)

- If we have a PDF for $\boldsymbol{x}$, we can use MAP
- From Bayes's theorem: $f(\boldsymbol{x}|\boldsymbol{y}) = \dfrac{\boldsymbol{f}(\boldsymbol{y}|\boldsymbol{x})\boldsymbol{f}(\boldsymbol{x})}{\boldsymbol{f}(\boldsymbol{y})}$
- With MAP, we have $\hat{\boldsymbol{x}}^{MAP} = \operatorname*{argmax}_{\boldsymbol{x}} f(\boldsymbol{y}|\boldsymbol{x})f(\boldsymbol{x})$
  - We want to maximize the choice of the parameter that makes both the observations and the parameter itself most likely
- If $f(\boldsymbol{x})$ is constant, then $\hat{\boldsymbol{x}}^{MAP} = \hat{\boldsymbol{x}}^{ML}$
- As with ML, we are still maximizing a function over $\boldsymbol{x}$, so the same sensitivity to outliers and modelling error still applies
- Example: consider the scalar observation $y = x + w$, where $w \sim \mathcal{N}(0,1), x \sim \mathcal{N}(\bar{x}, \sigma_x^2)$ and x, w independent
  - $f(x) \propto \exp\left(-\dfrac{1}{2}\dfrac{(x-\bar{x})^2}{\sigma_x^2}\right)$
  - $f(y|x) \propto \exp\left(-\dfrac{1}{2}(y-x)^2\right)$
  - $f(y|x)f(x) \propto \exp\left(-\dfrac{1}{2}\left(\dfrac{(x-\bar{x})^2}{\sigma_x^2} + (y-x)^2\right)\right)$
  - Differentiating with respect to x and setting to zero gives the following solution:
  - $\hat{x}^{MAP} = \dfrac{1}{1+\sigma_x^2}\bar{x} + \dfrac{\sigma_x^2}{1+\sigma_x^2}y$
    * Notice that this is a weighted sum between the mean of the prior distribution and the new measurement
  - Consider the extreme cases:
    * $\sigma_x^2 = 0 \implies \hat{x}^{MAP} = \bar{x}$ (if we're certain about x before any measurements, we just get the max of the prior)
    * $\sigma_x^2 \to \infty \implies \hat{x}^{MAP} = y$ (if we're uncertain about x, we just get the new measurement; note this is the same as maximum likelihood)
- This is most often used in state estimation

Minimum Mean Squared Error (MMSE)

- The MMSE is the a posteriori estimate that minimizes the mean squared error
- $\hat{\boldsymbol{x}}^{MMSE} = \operatorname*{argmin}_{\hat{\boldsymbol{x}}} E_{x|y}\left[(\boldsymbol{x}-\hat{\boldsymbol{x}})^T(\boldsymbol{x}-\hat{\boldsymbol{x}})|y\right]$
  - Expand and differentiate with respect to $\hat{\boldsymbol{x}}$: $2\hat{\boldsymbol{x}} - 2E[x|y] = 0 \implies \hat{\boldsymbol{x}} = E[x|y]$
  - The MMSE estimate is the expected value of $\boldsymbol{x}$ conditioned on $\boldsymbol{y}$

- While MAP is the maximum of the posterior, MMSE is the mean of the posterior
- Note we did not constrain $\hat{\boldsymbol{x}}$ in our minimization, but for some applications we might want to introduce constraints
  - e.g. for a discrete random variable with sample space $\mathcal{X}$, we need to constrain the minimization to $\hat{\boldsymbol{x}} \in \mathcal{X}$

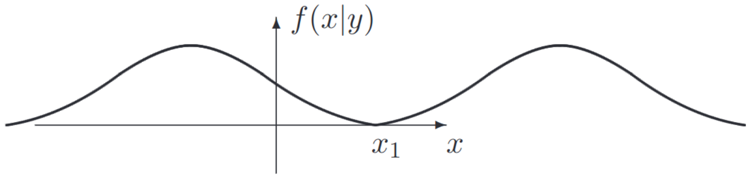


Figure 27: Undesirable case of MMSE.

- MMSE always takes the whole probability mass into consideration, whereas MAP and ML just pick the maximum probability – in some cases, this is desirable, while in other cases it is not
  - Consider the bimodal distribution of $f(x|y)$ above; MMSE would give x_1, but the probability of having x actually being near x_1 is zero
    - * MAP would have picked one of the two peaks
  - On the other hand, the MMSE is typically more robust to modelling errors and outliers, since it is not as sensitive to sharp peaks

Lecture 22, Dec 1, 2023

Gaussian Probability Distributions

- A 1-dimensional Gaussian PDF is given by $f(x|\mu, \sigma^2) = \dfrac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\dfrac{1}{2}\dfrac{(x-\mu)^2}{\sigma^2}\right)$
- Why are Gaussians common?
  - Central limit theorem: averages of independently drawn random variables become normally distributed when the number of random variables is sufficiently large
    - * If we don't actually know the distributions, we can usually approximate it as a Gaussian
    - * Let $y_1, \dots, y_n$ be a sequence of n independent random variables drawn from a distribution with finite mean and variance and let $\bar{y} = y_1 + \dots + y_n$, then for some $a, b \in \mathbb{R}$, $\lim_{n\to\infty} \Pr\left(a < \dfrac{\bar{y} - n\mu}{\sqrt{n}\sigma} < b\right) = \int_a^b \dfrac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}y^2}\,\mathrm{d}y$, i.e. $\bar{y}$ is Gaussian distributed with mean $n\mu$ and variance $n\sigma^2$
    - * This still holds if the y_i come from different distributions, provided the mean and variance are finite for each PDF
  - They are easy to handle mathematically – you only need the mean and variance
  - They remain Gaussian under many operations (summation, marginalization, conditioning, etc)
- In multiple variables, $f(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \dfrac{1}{\sqrt{(2\pi)^M \det \boldsymbol{\Sigma}}} \exp\left(-\dfrac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right)$
  - $\boldsymbol{\mu} \in \mathbb{R}^M$ is the mean and $\boldsymbol{\Sigma} \in \mathbb{R}^{M\times M}$ is the symmetric, positive definite covariance matrix
  - This can be visualized as an ellipse around the mean
- The above table shows the probability of a sample lying within a certain number of standard deviations of the mean
  - Note that this is for 1 dimension only; in higher dimensional space we need to look at probability ellipses
- Given some multivariate Gaussian $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, we can partition it as $\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix}, \boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix}, \boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix}$, then we can write $f(\boldsymbol{x}) = f(\boldsymbol{x}_1, \boldsymbol{x}_2)$

n	$\mathsf{Pr}(-n\sigma \le x \le n\sigma)$
0.67	≈ 0.5
1	≈ 0.68
2	≈ 0.95
3	≈ 0.997
4	≈ 0.99994

Figure 28: Probability of samples lying in different intervals around the mean.

- We can marginalize to find $f(\boldsymbol{x}_1) = \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_{11}) = \int_{-\infty}^{\infty} f(\boldsymbol{x}_1, \boldsymbol{x}_2 | \boldsymbol{\mu}, \boldsymbol{\Sigma})\,\mathrm{d}\boldsymbol{x}_2$ and likewise for $\boldsymbol{x}_2$
  * Marginalization picks out the relevant subblocks of the partitioned Gaussian, and gives Gaussian marginals
- For conditioning, $f(\boldsymbol{x}_1|\boldsymbol{x}_2) = \mathcal{N}(\boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2), \boldsymbol{\sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21})$
  * The factors are Gaussian
  * This can be derived using the Schur complement
    • Using an LDL decomposition we can find the inverse of $\boldsymbol{\Sigma}$, and then substitute into $(\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu})$

• The sum of two independent Gaussians is Gaussian: $\boldsymbol{y}_1 \sim \mathcal{N}(\boldsymbol{a}_1, \boldsymbol{B}_1), \boldsymbol{y}_2 \sim \mathcal{N}(\boldsymbol{a}_2, \boldsymbol{B}_2) \implies c_1\boldsymbol{y}_1 + c_2\boldsymbol{y}_2 \sim \mathcal{N}(c_1\boldsymbol{a}_1 + c_2\boldsymbol{a}_2, c_1^2\boldsymbol{B}_1 + c_2^2\boldsymbol{B}_2)$
  - This extends to matrix coefficients, since the transformation is linear
  - In general passing a Gaussian through a linear transformation preserves the Gaussian property
  - $\boldsymbol{C}_1\boldsymbol{y}_1 + \boldsymbol{C}_2\boldsymbol{y}_2 \sim \mathcal{N}(\boldsymbol{C}_1\boldsymbol{a}_1 + \boldsymbol{C}_2\boldsymbol{a}_2, \boldsymbol{C}_1^T\boldsymbol{B}_1\boldsymbol{C}_1, \boldsymbol{C}_2^T\boldsymbol{B}_2\boldsymbol{C}_2)$

• However, Gaussians don't remain Gaussian after passing through a nonlinear mapping
  - We can approximate using a linear function around the mean, so that if $y = g(x)$, then $x \sim \mathcal{N}(\mu_x, \sigma_x^2) \implies y \sim \mathcal{N}(\mu_y, a^2\sigma_x^2)$
    * $\delta y = y - \mu_y \approx \left.\dfrac{\mathrm{d}g(x)}{\mathrm{d}x}\right|_{x=\mu_x} (x - \mu x) = a\delta x$
    * $\sigma_y^2 = E[\delta y^2] = a^2 E[\delta x^2] = a^2\sigma_x^2$
  - If the nonlinear function can be approximated as linear in $[-3\sigma, 3\sigma]$, then the resulting Gaussian is a good approximation
  - In multiple variables: $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x), \boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x}) \implies \boldsymbol{y} \sim (\boldsymbol{g}(\boldsymbol{\mu}_x), \boldsymbol{A}\boldsymbol{\Sigma}_x\boldsymbol{A}^T)$, where $\boldsymbol{A} = \left.\dfrac{\partial \boldsymbol{g}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}=\boldsymbol{\mu}_x}$ is the Jacobian of $\boldsymbol{g}(\boldsymbol{x})$, $A_{ij} = \dfrac{\partial g_i}{\partial x_j}$
    * To see this, note $\Delta\boldsymbol{y} = \left.\dfrac{\partial \boldsymbol{g}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}=\boldsymbol{\mu}_x} \Delta\boldsymbol{x}$, so $\boldsymbol{\Sigma}_y = E[\Delta\boldsymbol{y}\Delta\boldsymbol{y}^T] = \boldsymbol{A}E[\Delta\boldsymbol{x}\Delta\boldsymbol{x}^T]\boldsymbol{A}^T = \boldsymbol{A}\boldsymbol{\Sigma}_x\boldsymbol{A}^T$

• We can fuse two Gaussians by multiplying them together and then renormalizing; this gives another Gaussian
  - This comes up when we want to combine multiple sources of information with different uncertainties
  - For two Gaussians with means μ_1, μ_2 and variances σ_1^2, σ_2^2, then $\dfrac{1}{\sigma^2} = \dfrac{1}{\sigma_1^2} + \dfrac{1}{\sigma_2^2}, \dfrac{\mu}{\sigma^2} = \dfrac{\mu_1}{\sigma_1^2} + \dfrac{\mu_2}{\sigma_2^2}$
    * Notice the means are weighted by the inverse of their variances, since a lower variance means more certainty
    * The inverse variance is sometimes referred to as the *precision* of the Gaussian
  - The direct product of two Gaussians has the exponent $\dfrac{(x-\mu)^2}{\sigma^2} = \dfrac{(x_1-\mu_1)^2}{\sigma_1^2} + \dfrac{(x_2-\mu_2)^2}{\sigma_2^2}$
    * $\dfrac{x^2 - 2\mu x + \mu^2}{\sigma^2} = \dfrac{(\sigma_1^2 + \sigma_2^2)x^2 - 2(\sigma_2^2\mu_1 + \sigma_1^2\mu_2)x + (\sigma_2^2\mu_1^2 + \sigma_2^2\mu_2^2)}{\sigma_1^2\sigma_2^2}$
    * Comparing the x^2 terms gives $\dfrac{1}{\sigma^2} = \dfrac{\sigma_1^2 + \sigma_2^2}{\sigma_1^2\sigma_2^2} = \dfrac{1}{\sigma_1^2} + \dfrac{1}{\sigma_2^2}$
    * Comparing the x terms gives $\dfrac{\mu}{\sigma^2} = \dfrac{\sigma_2^2\mu_1 + \sigma_1^2\mu_2}{\sigma_1^2\sigma_2^2} = \dfrac{\mu_1}{\sigma_1^2} + \dfrac{\mu_2}{\sigma_2^2}$

        * Note the constant terms are not equal, but this is fixed by normalization
    - Note the product of Gaussians is not normalized, so we need to find the normalization constant so that the distribution integrates to 1
        * In practice however we almost never have to compute this, since we usually only keep track of the mean and variance
    - In the multivariate case, $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \beta \prod_{n=1}^{N} \mathcal{N}(\boldsymbol{\mu}_n, \boldsymbol{\Sigma}_n)$
        * $\boldsymbol{\Sigma}^{-1} = \sum_{n=1}^{n} \boldsymbol{\Sigma}_n^{-1}$
        * $\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} = \sum_{n=1}^{N} \boldsymbol{\Sigma}_n^{-1}\boldsymbol{\mu}_n$
- Since Gaussians remain Gaussian under many different useful operations, it often suffices to keep track of only the sum and (co)variance of the distributions
- Under the assumption that random variables are Gaussian, analytic results for state estimation and other applications are available (e.g. a Kalman filter); without this assumption PDFs often have to be propagated through sampling (e.g. Monte Carlo methods)