SC42130: Fault Diagnosis and Fault Tolerant Control Lecture Notes (Q4, 24/25)

Gerb

Compilation Date: May 2, 2025

Contents

Lecture 01: Fundamentals 1
- Motivation . . . 1
- Definitions and Models . . . 2
  - Types of Faults . . . 2
- Fault Diagnosis . . . 2

Lecture 02: Structural Analysis 3
- Graphs . . . 3
- Structural Bi-partite Graphs . . . 4
- Components and Service Models . . . 4
  - Boolean Algebra . . . 5
- Fault Tree Analysis . . . 5
- Failure Mode and Effect Analysis . . . 6

Lecture 03: Change Detection Algorithms 7
- Introduction to Change Detection . . . 7
- Deterministic Tests: Limit Check . . . 8
- Basic Probabilistic Tests . . . 8

Lecture 01: Fundamentals

Motivation

Resilience and robustness of control systems. Especially interesting for smart buildings and aerospace applications.

Content of the course (roughly)

- Analyse structure of a system and carry out Fault Tree (FT) and Fault Modes and Effects Analysis (FMEA)
- Analyse anomalies that occur
- Design an algorithm for detecting, isolating and identifying faults
- Design policies for reconfiguration in order to accommodate faults

The goal is to detect anomalies and apply suitable interventions to guarantee safety of the system.

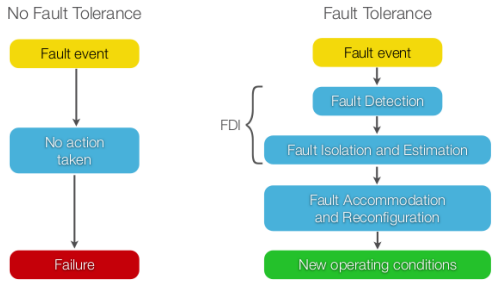


Figure 1: *Without and with fault tolerance*

Control systems can be subdivided into the following components which can be considered attack vectors:

- Controller, subject to cyber-physical attacks
- Field bus/WAN for communication between sensors, actuators and controllers
- Actuators
- Sensor
- The physical plant, usually not subject to cyber attacks, but can be attacked physically

Definitions and Models

Definition (Hardware Redundancy)**.** *Have redundancy of components at the hardware level to ensure robustness to sensor or actuator faults. In case of sensors this allows for* ***majority voting****. Hardware redundancy can be applies to sensors, actuators and controllers.*

Definition (Physical Redundancy)**.** *A static, passive tolerance approach. Over-design components such that they cannot fail for the used application. E.g. over-spec motors or expensive reliable sensors. Design approach is robust in the sense the system will work even if components fail. Think of spokes in a bike wheel, threads in a rope, cables on a bridge.*

Definition (Analytical Redundancy)**.** *A dynamic switching approach where sensors are made redundant by mathematical models. E.g. sensor fails, this is detected by a FDA (Fault Detection Algorithm) and this sensor is switched to a virtual sensor (Oberserver).*

Table 1: *An overview of different redundancy types and their pros and cons*

	Hardware	**Physical**	**Analytical**
Pros	No perf. loss	Best if in budget	most efficient
Cons	Costly	Cost	perf. loss

Definition (Faults and Failures)**.** *Faults reduce functionality or performance while failures render the system inoperable. Formally: A* ***fault*** *in a dynamical system is an un-permitted deviation from the nominal situation of at least one characteristic property of the system. A* ***Failure*** *is a permanent interruption of a systems ability to perform a required function.*

Definition (Reliability)**.** *Reliability of a system in the ability to perform a required function under stated conditions within a given scope during a given period of time.*

As in some sense the inverse of faults and failures we have reliability. Mathematically we can define the Mean Time to Failure (MTTF):

$$\text{MTTF} = \frac{1}{\lambda}$$

Where λ is the average number of failures per unit of time. The availability also follows from this. This can be viewed as the probability that a system or equipment will operate satisfactorily at any period of time:

$$A = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$$

Where MTTR is the Mean Time To Repair. Note that lower MTTR (short repair time) means more availability and larger MTTF means higher availability.

Definition (Fault Detection and Diagnosis)**.** *Fault detection is the determining of the presence of a fault in a given system at a given time.* ***Fault diagnosis*** *is more broad and consists of determining the presence, type, size and location of a fault in a given system at a given time, assuming knowledge of the possible faults.*

Definition (Fault Tolerance)**.** *Fault tolerance is the possibility of achieving (control) objectives in the presence of a given (set of) fault(s). It can be interpreted as robustness to faults in the sense that we contain the consequences of faults and failures thus that the components remain functional.*

Definition (Nominal Behaviour)**.** *All faults and failures refer to nominal conditions. Nominal behaviour is defined as the expected or desired behaviour w.r.t. control objectives.*

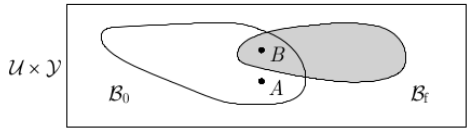


Figure 2: *In some cases it can be ambiguous whether the system is faulty or nominal. For example, the light is off, so it is ambigious whether it is broken or off until we test and observe*

For dynamical systems, we generally use the following models:

$$\begin{cases} \dot{x} = g(x, u, w, f) \\ y = h(x, u, v) \end{cases}, \quad \begin{cases} x_{k+1} = g(x_k, u_k, w_k, f_k) \\ y_k = h(x_k, u_k, v_k) \end{cases}$$

Where $x \in \mathbb{R}^n$ is the system state $u \in \mathbb{R}^m$ the input, $w \in \mathbb{R}^p$ the exogeneous inputs and disturbances, f the (set of) faults and $v \in \mathbb{R}^q$ the sensor noise.

Types of Faults

- Abrupt, the fault is not there until it abruptly is and stays
- Incipient, the faulty behaviour slowly grows over time
- Intermittent, the fault abruptly happens and then disappears again intermittently

Faults can strike at many locations:

$$\begin{aligned} \tilde{u} &= u \cdot (1 + f) \quad \text{(Actuator)} \\ \dot{x} &= \tilde{g}(\cdot) \quad \text{(Plant)} \\ \tilde{y} &= y + f \quad \text{(Sensor)} \end{aligned}$$

Faults can be additive ($\tilde{y} = y + f$), multiplicative ($\tilde{y} = y \cdot (1 + f)$) or general $\tilde{y} = f(y)$.

Fault Diagnosis

Diagnosis can be signal based or model based. Examples:

- Signal Based
  - Raw signal
  - Mean or variance over moving window
  - Peak values
  - Fourier transform, STFT, Wavelet, etc.
  - Machine Learning model based on nominal behaviour
- Model Based
  - (Kalman) Observer, estimation errors
  - Parity Relations for error
  - Model parameter estimates and check whether these match nominal

A **symptom** is the difference between actual value of features extracted from observations and nominal ones. Examples are for example unexpected or unexplainable spikes in the power of a specific frequency. In the model based case we may observe an estimate of a spring constant over time and see how this matches the nominal behaviour.

Physiological changes shall be ignored but pathological changes are evaluated in the next step. A measure of a symptom is so compared against a known pathological threshold. In general this is a change detection problem.

1. **Detection**, testing the null hypothesis $\mathcal{H}_0$: "the sytsem is behaving in a nominal way"
2. **Isolation**, testing the N faulty hypotheses: $\mathcal{H}_i$: "the system is behaving as if the i-th fault is present"
3. **Identfication/Estimation**, if $\mathcal{H}_0$ and every but one $\mathcal{H}_i$ are falsified, then estimate parameters of the i-th fault. Note that if $\mathcal{H}_0$ and every $\mathcal{H}_i$ are falsified, we identify a model of a new fault

Lecture 02: Structural Analysis

Graphs

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denote a graph with set of vertices (nodes) $\mathcal{V}$ and set of edges $\mathcal{E}$. For example:

$$\mathcal{V} = \{1, 2, 3, 4\}, \quad \mathcal{E} = \{e_{12}, e_{23}, e_{34}\}$$

We can also represent this in terms of an adjacency matrix A. Note that $|\mathcal{V}| = n$ and $A \in \mathbb{R}^{n \times n}_{\text{sym}}$ (or asymmetric when the graph is directed) with the property $\text{tr}(A) = 0$. The adjacency matrix has 0 for nodes not connected by edges (with self-connection being 0). Then for the example above we write

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

There are many graph structures, implicitly we assume that all of them are strongly connected and undirected. For example:

- Random, all nodes connected at random with one another
- Regular, all nodes have at most degree 2
- Complete, every node is connected to every other node
- Disconnected, NOT strongly connected
- Tree, has a root and trunk and branches to this trunk, no redundant edges so only 1 path from each node to every other node
- Bipartite, nodes are divided in different sets, no interconnection between nodes in the same set BUT there are connection to nodes in other sets

Edge weights can encode properties such as cost, time, distance, etc. We can then define for example linear programs on these graphs or apply algorithms such as Dijkstra, A*[1] to find the shortest path through this graph.

Edges can also be conditional (flowcharts) or probabilistic (Markov Chains). Important for dynamical systems are finite automaton which model state transition graphs.

For the state transition graph, we model:

- Nodes $\rightarrow$ states
- Edges $\rightarrow$ transitions
  - Typically modelled as boolean functions of an input, so for example *Open* : $\mathbb{U} \rightarrow \mathbb{B}$, where $\mathbb{B} = \{0, 1\}$ and $\mathbb{U}$ is the set of inputs (e.g. $\mathbb{U} = \{\text{Open Door}, \neg\text{Open Door}\}$
- As an extra, some states may have an output function per each state (i.e. specific outputs may be given upon output of a state).

[1] A is Dijkstra with an extra heuristic function to guide the search.

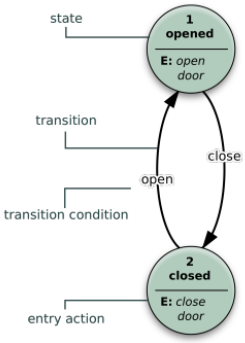


Figure 3: *An example of a state transition graph used to model the dynamcis of a system*

Structural Bi-partite Graphs

Definition (Bipartite Graphs)**.** *Let $\mathcal{G} = ((\mathcal{C}, \mathcal{Z}), \mathcal{E})$. Here $\mathcal{C}$ are the constraints, $\mathcal{Z}$ are the variables and $\mathcal{E}$ is the set of connecting edges. The constraints typically represents our equations. We can further split*

$$\mathcal{Z} = \{\mathcal{K}, \mathcal{X}\}$$

Where $\mathcal{K}$ are known variables and $\mathcal{X}$ unknown variables. We can split

$$\mathcal{C} = \{\mathcal{C}_\mathcal{K}, \mathcal{C}_\mathcal{X}\}$$

Where $\mathcal{C}_\mathcal{K}$ are known and $\mathcal{C}_\mathcal{X}$ are unknown.

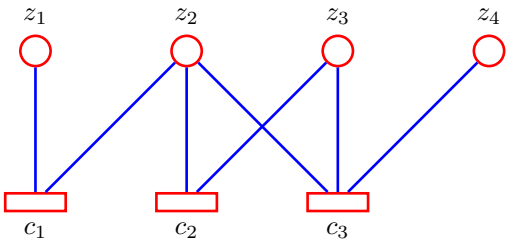


Figure 4: *Example of a bi-partite graph with states z_i and constraints c_i*

An example of a bi-partite graph can be nodes z_i and constraints c_i where

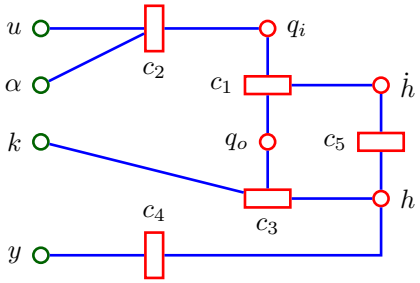
$$\begin{aligned} c_1: &\quad z_1 = 2z_2 \\ c_2: &\quad z_2 = 3z_3 \\ c_3: &\quad z_1 + z_2 + z_3 = 1 \end{aligned}$$

Now presume that z_1 and z_2 are known and z_3 and z_4 are not known. Based on the graph we can graphically inspect (without looking at constraint equations) and find that c_2 depends on z_2 and z_3, so we can use this to compute z_3. c_3 depends on z_2 (known), z_3 unknown but computable from c_2 and z_4, so we can also compute z_4.

Example (Bipartite Graph Model for a single tank system)**.** Consider a simplified 1 tank model. By modelling the dynamical system we find the following constraint equations:

$$\begin{aligned} \text{Tank } c_1: &\quad \dot{h}(t) = q_i(t) - q_o(t) \\ \text{Input Valve } c_2: &\quad q_i(t) = \alpha u(t) \\ \text{Output Valve } c_3: &\quad q_o(t) = k\sqrt{h(t)} \\ \text{Level Sensor } c_4: &\quad y(t) = h(t) \\ \text{Integration } c_5: &\quad h(t) = \int_{t_0}^{t} \dot{h}(s)\, ds \end{aligned}$$

We consider the input measurement and system parameters α and k to be known. We consider h, $\dot{h}$, q_i and q_o to be unknowns. The bi-partite graph below shows a graphical representation of this system. This reveals and **analytical redundancy** of the output y! We can use a virtual sensor (Luenberger Observer/Kalman Filter) to estimate y in the case that the sensor fails, which easily follows from graphical analysis of the graph below.



Note that this example is relatively simple. In the case that we have a more complex system (e.g. a building HVAC system), we can subdivide this system into components. We then build bi-partite graphs for each of these components and interconnect these with one another. To then determine analytical redundancies (i.e. detectability of states) we can traverse the graph and try to find a path from known variables to unknown variables. If such a path exists, then there is analytical redundancy.

Components and Service Models

Components and service models are a way to organise knowledge of the system (nominal behaviour) in a hierarchical way. This allows for fault propagation and effects as well as root causes analysis.

To do this, we divide this system into subsystems. All of these subsystems consist of components, which have lower level nodes called services. These services are e.g. operating modes of the component. Finally, the services can be described by a set of (non-linear) differential equations ($\dot{x} = f(t, x, u)$). From this structure, a tree graph naturally arises. This tree clearly shows how faults propagate through the system.

As an example, consider a measurement sub-system with

sensors that each have 2 services, normal operation and self-calibration. These sensors are not redundant versions of the same sensor but instead different kind of sensors (e.g. Temperature, pressure, etc.) In the case 1 of these sensors fails, the sensor sub-system also fails. If the sensor system fails and we do not have analytical redundancy to (temporarily) mitigate this, the entire system fails as we now do not know what it is doing and hence cannot control it.

Definition (Formal Definition of a service)**.** *A service s_i is described by a 6-tuple given as*

$$s = (\mathit{cons}, \mathit{prod}, \mathit{proc}, \mathit{rqst}, \mathit{enable}, \mathit{res},)$$

Where cons are the consumed variables, proc are the produced variables, proc is the process (dynamic equations modelling the relation between consumed and produced variables). The rqst are the requests as some components only provides services on request. enable shows whether the component is operating, which can be conditional on e.g. other components failing. Finally, res are the resources which are the (physical) parts of the component.

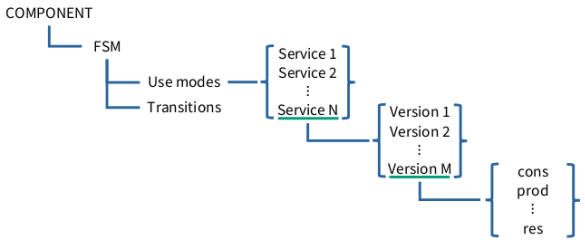


Figure 5: *A breakdown of a general component model where FSM is a finite state machine. A FSM consists of modes and transitions. The modes consist of services, services have versions and each has a service 6-tuple as formally defined above*

Definition (General Component Model)**.** *Component k is defined by a state transition graph $\mathcal{G}(M(k), \tau(k), m^0(k)$, where*

- *$M(k)$ is the set of use modes*
- *$\tau(k)$ is the set of mode transitions*
- *$m^0(k)$ are the initial use modes*
- *use-mode $m_i(k)$ is a set of services $s_i(k) \subseteq s(k)$*
- *service $s_\ell(k)$ are the pre-ordered versions*
- *version $s_\ell^j(k)$ is version j of service ℓ which has a 6-tuple as defined formally in the service definition $\tau_{ij}(k)$ is a 3-tuple $(c_{ij}(k), m_i(k), m_j(k))$ defining the transition from mode $m_i(k)$ (origin) to $m_j(k)$ (destination) in the case that condition $c_{ij}(k)$ is met.*

Boolean Algebra

Boolean algebra is algebra on boolean variable $\delta \in \mathbb{B} = \{0, 1\}$. We have the following operations define on $\mathbb{B}$:

$$A \wedge B = \begin{cases} 1, & \text{if } A = 1,\ B = 1 \\ 0, & \text{otherwise} \end{cases}$$

$$A \vee B = \begin{cases} 1, & \text{if } A = 1 \text{ or } B = 1 \text{ or } A = B = 1 \\ 0, & \text{otherwise} \end{cases}$$

$$\neg A = \begin{cases} 1, & \text{if } A = 0 \\ 0, & \text{otherwise} \end{cases}$$

$$\langle A_1, \cdots, A_N \rangle_k = \begin{cases} 1, & \text{if at least } k\ A_i = 1 \\ 0, & \text{otherwise} \end{cases}$$

From top to bottom we have: "and", "or", "not" and "majority vote".

Theorem (De Morgan's Laws)**.** *Let $A, B \in \mathbb{B}$. Then*

$$\neg(A \wedge B) \iff \neg A \vee \neg B$$
$$\neg(A \vee B) \iff \neg A \wedge \neg B$$

Theorem (Disjunctive Normal Form)**.** *The disjunctive normal form is a canonical form of semantic (logic) functions and is given by the disjunction (or) of several conjunctions (and). We write*

$$f(A_1, \cdots, A_N) = \bigvee \left(\bigwedge_j A_j \right)$$

for example

$$f(A, B, C, D) = (A \wedge B) \vee (C \wedge D) \vee (A \wedge D)$$

Definition (Boolean Matrix-Vector Product)**.** *Given $M \in \mathbb{B}^{m \times n}$ and $f \in \mathbb{B}^n$. We denote $e = M \otimes f$ which is represented as*

$$e_{(i)} = (M_{(i,1)} \wedge f_{(1)}) \vee \cdots \vee (M_{(i,n)} \wedge f_{(n)})$$

Definition (Inverse Inference Operator)**.** *Given $M \in \mathbb{B}^{m \times n}$. $f = M^T \odot e$ represent the logical function*

$$f_{(i)} = (M_{(1,i)} \equiv f_{(1)}) \vee \cdots \vee (M_{(n,i)} \equiv e_{(m)})$$

Fault Tree Analysis

Fault trees are an important tool (diagnosis method) for determining the effect of component failure on system dependability.

Fault Tree Objective. *Determine whether the failure of a (subset of) component(s) can lead to the failure of the whole system*

It is enough to have qualitative knowledge on the system. We break down the system by analysing what **components** make up the system, what **services** these components offer and how these components are **interconnected**.

The process of identification of a fault tree is usually the following:

- Identification of components
- Represent how the components compose subsystems and how subsystems compose the system.

A Fault tree is a **directed, acyclic graph** (not necessarily a tree graph), where the leaves represent component failures (basic events) which are interconnected through logic gates. The root node is the top event, representing complete system failure. Note that fault trees represent propagation of failure (i.e., when component failure leads to system failure).

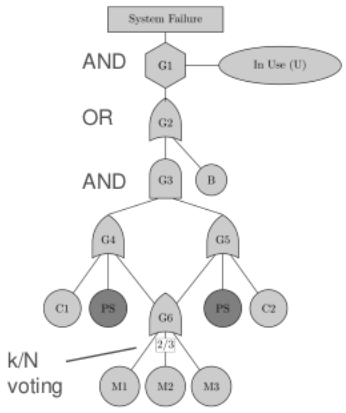


Figure 6: *An example of a fault tree that can be used for failure analysis*

Definition (Fault Tree)**.** *A Fault tree is a tuple*

$$F = (\mathcal{E}_B, \mathcal{G}, T, I)$$

Where $\mathcal{E}_B$ is a set of basic events $e \in \mathcal{E}_B$ and $e \in \mathbb{B}$ (with 0 denoting healthy and 1 faulty). $\mathcal{G}$ is a set of gates, $T : \mathcal{G} \to \{\wedge, \vee, \neg, \langle\cdot\rangle_k\}$ us a function which maps hates to types. Finally $I : \mathcal{G} \to \mathcal{P}(\mathcal{E})$ is a function which maps gates to their input with $\mathcal{P}(\cdot)$ denoting the powerset and $\mathcal{E} = \mathcal{E}_b \cup \mathcal{G}$.

Each fault tree has a semantic function associate to it:

$$\pi_F : \mathcal{P}(\mathcal{E}_B) \times \mathcal{E} \to \mathbb{B}$$

Where $\pi_F(S, e_i) = 1$. If $e_i \in \mathcal{E}$ is a failure ($e_i = 1$) when all elements of $\mathcal{S} \subset \mathcal{P}(\mathcal{E}_B)$ are failures. We use shorthand notation $\pi_F(S)$ if e_i is the top event. $\pi_F(S)$ represents the effect of component failure on system health.

Definition ((Minimal) Cut Sets)**.** *A set $\mathcal{C} \subseteq \mathcal{E}_B$ is a cut set of F if $\pi_F(\mathcal{C}) = 1$. The cut set is a minimal cut set if no subset of the cut set is itself a cutset. That is*

$$\pi_F(\mathcal{C}) = 1 \wedge \pi_F(\mathcal{C}') = 0 \quad \forall \mathcal{C}' \subset \mathcal{C}$$

An important method for quantitative analysis of fault trees is the structure function

$$f : \mathbb{B}^N \to \mathbb{B}$$

where $N = |\mathcal{E}_B|$ are the amount of leaves of the fault tree. This function tells us when the values e_i result in a failure at the top event. $f(\cdot)$ can be represented in canonical form (disjoint normal form) where each conjunction in f is a minimal cut set!

Note that there are many more flavours of fault tree not covered: probabilistic fault trees for reliability, propagating probability of failure, analysis of mean-time to failure, dynamic fault trees, repairable fault trees.

Failure Mode and Effect Analysis

FMEA is meant to analyse the effect of component failure on system dependability.

Failure Mode and Effect Analysis Objective. *Determine how the failure of a (subset of) component(s) can lead to the failure of the whole systems.*

Note (Difference between FTA and FMEA)**.** FTA is only concerned with **whether or not component failure can result in system failure**. The main differnce with FMEA is that FMEA actually analyses how this happens.

For FMEA we need fault/failure modes of each component, effect of each mode at the component level and how the mode of one component affects the connected components. We can again represent FMAE using a graph. Each node in the graph is associate with a component. Then local faults are denoted f_i^{loc}, local effects denoted e_i and inherited effects are e_i^{in} (inherited effects are local effects of other nodes so $e_i^{\text{in}} = e_j$).

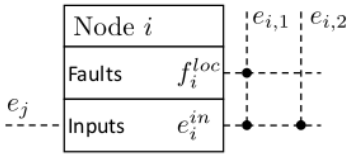


Figure 7: *Graph representation of Failure mode and effect analysis*

Note that for FMEA, boolean matrix-vector products are particularly useful as the fault propagation can be represented

by the boolean mapping

$$e = M \otimes f$$

Where f are faults and e are the effects. Note that this puts faults (causes) in relation to effects. For Fault diagnosis purposes we are interested in the inverse relation: "Given these effects, what faults cause these?". This is the inverse inference problem and is closely related to the problem of isolation. Note that this mapping in general is surjective (not bijective). For any given effect there may be multiple faults that cause this effect.

Lecture 03: Change Detection Algorithms

Introduction to Change Detection

Probabilistic detection algorithms are based on hypothesis testing. To do fault detection and diagnosis with hypothesis testing we have the following 3 step process:

1. **Detection**. Testing the *null hypothesis*:

   $\mathcal{H}_0$: "The system is behaving in a nominal way"

2. **Isolation**. Testing the N *faulty hypotheses*:

   $\mathcal{H}_i$: "The system is behaving as if i-th fault present"

3. **Identification and Estimation**. If everyone by $\mathcal{H}_i$ falsified, estimate the parameters of the i-th fault. If every $\mathcal{H}_i$ is falsified, identify a model of a new fault.

It is important to classify different kinds of results from hypothesis testing to distinguish between True positive, False positive etc.

We use the following definitions for detection errors and detection performance

Definition (Detection Errors). *We define the*
- ***True Positive***: $TP := \#\neg\mathcal{H}_0(F)$
- ***True Negative***: $TN := \#\mathcal{H}_0(H)$
- ***False Positive***: $FP := \#\neg\mathcal{H}_0(H)$
- ***False Negative***: $FN := \#\mathcal{H}_0(F)$

Definition (Detection Performance). *We define*
- ***False Positive Rate***: $FPR := \frac{FP}{FP+TN}$
- ***True Negative Rate***: $TNR = 1 - FPR$
- ***True Positive Rate***: $TPR = \frac{TP}{TP+FN}$
- ***False Negative Rate***: $FNR := 1 - TPR$
- ***Accuracy***: $ACC = \frac{TP+TN}{TP+FP+TN+FN}$
- ***Recall***: $RE = \frac{TP}{TP+FN} = TPR$

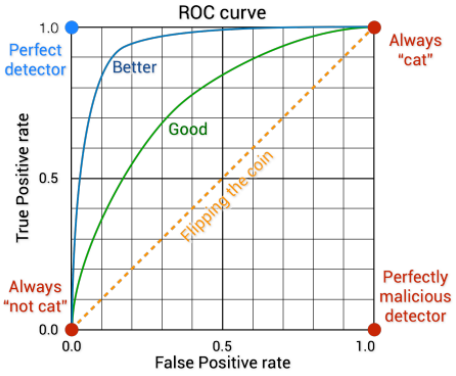


Figure 8: *Example of detection algorithm false positive rates. Note that Yellow is effectively random geussing, anything below is worse, anything above is good.*

Problem (Chnage Detection Problem)**.** *The behaviour of the system is described by a parameter θ. This parameter changes from nominal value θ_0 to fault θ_1 at time instant k_0. We measure the symptom $z(k)$. The goal is to estimate k_0 and θ_1 by observing sympton $z(k)$ and system behaviour.*

Deterministic Tests: Limit Check

The most basic deterministic test is limit checking. We are given the minimum and maximum values, which are used to determine whether $z(k)$ is in a known range. In the scalar case:

$$z(k) \notin [z_{\min}, z_{\max}] \implies \text{Alarm}$$

Some pros of this approach are that it is simple to define, check and implement. There are no false positives, however the approach is overly conservative and only works in steady state. In the case that $z(k) \in \mathbb{R}^n$ is vector valued, we define a scalar valued evaluation function of $z(k)$ to reduce the problem to scalar limit checking

$$Z = f_{\text{ev}}(z(k)),\ f_{\text{ev}} : \mathbb{R}^n \to \mathbb{R}$$

There are many such examples, such as p-norms (squircles), quadratic forms (circles/ellipsoids), (transformed) rectangles (scaled infinity norm), general polynomials or convex hulls. The defined region need not be convex.

Given a sequence of vector valued symptoms $\{z(k)\}_{k=k_1}^{k_2}$, we again reduce this to a scalar problem by transforming the sequence to a scalar. For example

1. First over all components, then over time. Examples:

$$Z = \|z(k)\|_p, [k_1, k_2] = \sum_{k=k_1}^{k_2} \|z(k)\|_p \quad \text{Sum } L^p \text{ norms}$$

$$Z = \|\{z(k)\|_p\}_{k=k_1}^{k_2}\|_{\ell_q} \quad \ell^q \text{ norm of } L^p norms$$

2. First over time, then over all components. Examples:

$$Z = \left\| \sum_{k=k_1}^{k_2} \frac{z(k)}{k_2 - k_1 + 1} \right\|_p \quad L^p \text{ norm of averages}$$

$$Z = \left\| \text{FFT}(\{z(k)\}_{k=k_1}^{k_2}) \right\|_p \quad L^p \text{ norm of FFT}$$

Possible ways to improve this are time-varying limits based on operating conditions of the system, or do model based limit checking. In this case, these are also valid for non-steady state conditions.

Basic Probabilistic Tests

By construction deterministic tests have zero FPR, however they can have very poor TPR (no-free-lunch principle). This is evident when z might rarely take large values. Additionally, when data is generated by a random process with infinite support, which is practically speaking always the case, we cannot (easily) define $z_{\min}$ and $z_{\max}$.

The intuition behind statistical hypothesis testing is that we are checking whether our underlying PDF has changed from what we originally believe it to be.

1. Assume that under $\mathcal{H}_0$ data is generate by a random process with PDF $p_0(z)$
2. Collect actual data
3. Is the data likely to have been drawn from $p_0(z)$ or from another PDF?

The following quantities are of interest:

- μ_0, True mean (known)
- $\hat{\mu}_0$, empirical mean ($\mathbb{E}[\hat{\mu}_0] = \mu_0$)
- $p_{\hat{\mu}_0}$, PDF of $\hat{\mu}_0$ (not PDF of data)
- $\mu_{\frac{\alpha}{2}}$, $\int_{-\infty}^{\mu_{\frac{\alpha}{2}}} p_{\hat{\mu}_0}(\mu)\, d\mu = \frac{\alpha}{2}$
- $\mu_{1-\frac{\alpha}{2}}$, $\int_{\mu_{1-\frac{\alpha}{2}}}^{\infty} p_{\hat{\mu}_0}(\mu)\, d\mu = \frac{\alpha}{2}$

Definition (p-values)**.** *The p-value of a given realisation x^* of a random process X is the probability that X can take values more extreme than x^*.*

Note that p-values are easy to abuse to care needs to be taken. It tells us how likely the data is given $\mathcal{H}_0$, NOT the other way around. The p-value does not give meaningful info (i.e. nonsense) if the hypothesis on the PDF or data was wrong in the first place.

Definition (Empirical Moments)**.** *Given observed data sequence over horizon N $\{z(i)\}_{i=k-N+1}^{k}$, we compute the empirical mean over a sliding window as*

$$\hat{\mu}_{[k-N+1,k]} = \frac{1}{N} \sum_{i=k-N+1}^{k} z(i)$$

and the empirical variance over a sliding window as

$$\hat{\sigma}^2_{[k-N+1,k]} = \frac{1}{N-1} \sum_{i=k-N+1}^{k} (z(i) - \mu_0)^2$$

In the case that μ_0 is not known, we use $\hat{\mu}_{[k-N+1,k]}$ in the computation of the variance. For compactness we will often denote $\hat{\mu}_{[k-N+1,k]} = \hat{\mu}(k)$.

Perhaps one of the most widely used and important statistical tests is the student t-test. This determines if the mean of a normal random variable changed relative to the null hypothesis. We assume nominal mean μ_0 known, nominal variance unknown but constant. Empirical moments $\hat{\mu}(k)$ and $\hat{\sigma}^2(k)$ are estimated from the last N samples.

Theorem (Student t-test)**.** *Assume normally distributed data. Let $\hat{\mu}(k)$, $\hat{\sigma}(k)$ be the sliding window empirical mean and variance, let nominal mean μ_0 be known. We compute*

$$t(k) = \frac{\hat{\mu}(k) - \mu_0}{\hat{\sigma}(k)/\sqrt{N}}$$

If $|t| > t_{\alpha,N-1}$ reject the null hypothesis $\mathcal{H}_0$.

Theorem (Student t-test with unknown Mean)**.** *Estimate true mean before and after change as $\hat{\mu}_0(k) = \hat{\mu}_{[k-N_0+1,k]}$ and $\hat{\mu}_1(k) = \hat{\mu}_{[k,k+N_1-1]}$. Via the same procedure estimate $\hat{\sigma}_0^2$ and $\hat{\sigma}_1^2$. Compute the random variable*

$$t(k) = \frac{\hat{\mu}_0(k) - \hat{\mu}_1(k)}{\sqrt{(N_0-1)\hat{\sigma}_0^2(k) + (N_1-1)\hat{\sigma}_1^2(k)}} \times \sqrt{\frac{N_0 N_1 (N_0+N_1-2)}{N_0+N_1}}$$

We can also test whether the variance of a normal random variable is changed relative from a known value σ_0^2.

Theorem (χ^2-test)**.** *Let $\hat{\sigma}_0^2$ be the known nominal variance. Compute the empirical variance $\hat{\sigma}^2(k)$ from previous N samples. The following random variable is χ^2 with $N-1$ degrees of freedom:*

$$\chi^2(k) = \frac{(N-1)\hat{\sigma}^2}{\sigma_0^2}$$

If $\chi^2(k) > \chi^2_{N-1,1-\alpha}$, then reject the null hypothesis $\mathcal{H}_0$ with significance α.

Theorem (F-test)**.** *The F-test is the same as the χ^2-test but the nominal variance is not known. We compute empirical variance $\hat{\sigma}_0^2$ from previous N_0 samples, and we compute $\hat{\sigma}_1(k)$ from the next N_1 samples. The random variable with (N_0-1, N_1-1) degrees of freedom. We compute the F-statistic as*

$$F(k) = \frac{\hat{\sigma}_0^2}{\hat{\sigma}_1^2}$$

If $F > F_{N_0-1,N_1-1,1-\alpha}$ then reject null hypothesis $\mathcal{H}_0$

Note (Limitations of Hypothesis Testing)**.** To falsify the null hypothesis generally requires quite a large amount of samples both before and after the hypothetical change. This delays the detection and generally likelihood based algorithms such as CUSUM are better. This will be the topic of the next lecture.