ソフトウェアテスト
[8] テストの評価と信頼性

Software Testing
[8] Evaluation of testing and reliability

あまん　ひろひさ
阿萬 裕久（AMAN Hirohisa）
aman@ehime-u.ac.jp

評価（ひょうか）: evaluation
信頼性（しんらいせい）: reliability

テストの評価

- テストはソフトウェアの品質保証を行う活動

テストが完璧
⇒ 欠陥(フォールト)は存在せず，障害は発生しない．
いわゆる「バグ」は無い．

- 「テストという活動」の**評価**が必要
  - **網羅率**に基づく評価
  - **バグ解決率**に基づく評価
  - **運用性**に基づく評価

品質保証(ひんしつほしょう): quality assurance
活動(かつどう): activity
完璧(かんぺき): perfect
欠陥(けっかん): defect
フォールト: fault
障害(しょうがい): failure
網羅率(もうらりつ): coverage
バグ解決率(バグかいけつりつ): bug fix rate
運用性(うんようせい): operability

テストの評価(1)
網羅率に基づく評価

- テストケースで「**どれだけ網羅できたのか**」という視点での評価
  - **状態遷移**に着目
  - **仕様に従った動作・入出力**に着目
    ※ブラックボックステスト
  - **プログラムでの実行の流れ**に着目
    ※ホワイトボックステスト

どれだけ網羅(もうら)できたのか: how many cases(paths) could you cover?

状態遷移(じょうたいせんい): state transition

仕様(しよう)に従った(したがった)動作(どうさ)・入出力(にゅうしゅつりょく): behavior or input/output according to the specification

プログラムでの実行(じっこう)の流れ(ながれ): execution flows in the program

テストの評価(2)
バグ解決率に基づく評価

- テストを実施することでバグが見つかり，それが修正されていく
- もしも**バグの総数**が分かっていれば，その中での解決数からテスト活動の評価ができる

$$\text{バグの解決率} = \frac{\text{解決したバグ数}}{\text{バグの総数}}$$

※バグという用語は正確でないことに注意；
正しく動作しない現象(「障害，不具合」)がテストで見つかり，その源である「欠陥」(フォールト)あるいはそれを作り出した「エラー」を解決することが「解決」にあたる．

バグの総数(バグのそうすう): total number of bugs
解決したバグ数(かいけつしたバグすう): number of resolved (fixed) bugs
用語(ようご): term
正確でない(せいかくでない): inaccurate
現象(げんしょう): phenomena
障害(しょうがい)，不具合(ふぐあい): failure
源(みなもと): source
欠陥(けっかん): defect
フォールト: fault
解決(かいけつ): resolution

どうやって「バグ総数」を知るのか

- 結論からいえば，**真の個数は分からない**
  - 人間をシステムに例えるならば，一生のうち**何回病気になるのか**を知ろうとするのに似ている
  - 「不具合の発生≒病気の症状が出る」と考えると，フォールトはその源となった細菌やウイルスであり，エラーはそれに感染するに至った行為ともとれる．
- （統計的に）**バグ総数を推定**するしかない

どうやってバグ総数（そうすう）を知るのか：how can we get the total bug count?
結論（けつろん）：conclusion
真（しん）の個数（こすう）：true count
病気になる（びょうきになる）:getting sick
症状が出る（しょうじょうがでる）：display symptoms
細菌（さいきん）：bacteria
ウイルス：virus
感染する（かんせんする）：catch infection
統計的（とうけいてき）：statistically
推定（すいてい）：estimation

バグ数推定法① 捕獲・再捕獲法(1／4)

【例】

琵琶湖にブラックバスが何匹いるのかを調べたい.

※琵琶湖にいるすべての魚を捕獲できればよいが, それは現実的に不可能である.

捕獲・再捕獲法(ほかく・さいほかくほう): capture-recapture sampling method
琵琶湖(びわこ): Lake Biwa (= the largest lake in Japan)
ブラックバス : black bass
捕獲(ほかく): capture

バグ数推定法① 捕獲･再捕獲法（2／4）

1. いったん適当な数のブラックバスを捕獲する.
2. そして，そのブラックバスにマーキングをして，逃がす.
3. しばらくした後，再びブラックバスを捕獲する. その中にマーキングされたものが何匹含まれているかで総数を推定する.

適当な数（てきとうなかず）: some

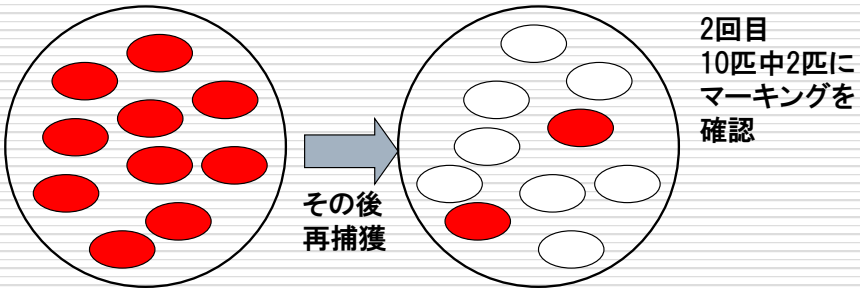
マーキング : marking

逃がす（にがす）: release

バグ数推定法① 捕獲･再捕獲法（3／4）

- このようなイメージ

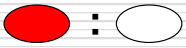
1回目
10匹に
マーキング



その後
再捕獲

2回目
10匹中2匹に
マーキングを
確認

全体における割合の推定



＝ 2 : 8

このイメージ図だと
10 : ？ = 2 : 8
なので残りは 40 匹

バグ数推定法① 捕獲・再捕獲法（4／4）

- これをソフトウェアに適用すると
  1. 人工的なバグを n 個用意しておく（マーキング）
  2. その後，テストで人工的なバグが x 個，そうでない本物のバグが y 個見つかったとする
  3. 本物のバグの総数は次式で推定される．

$n : ? = x : y$ より $\dfrac{n\,y}{x}$

人工的なバグ（じんこうてきなバグ）: artificial bugs
本物のバグ（ほんもののバグ）: real bugs
推定（すいてい）: estimation

捕獲・再捕獲法の欠点

- リアルな人工バグを作るのは難しい
  - どうしてもわざとらしいバグになってしまい，すぐに見つかってしまう
  - エンジニアの心理として「バグを作る」ことには抵抗がある （現場の声）
- いわば「捕まりやすい」魚を放つことになって，推定数が実際よりも小さくなってしまう

$$\frac{n\,y}{x}$$

※本来よりも y が小さく，x が大きくなる

欠点（けってん）: shortcoming
リアルな人工バグ（リアルなじんこうバグ）: realistic artificial bugs
わざとらしい: seem intentional, not look natural
心理（しんり）: mind
抵抗（ていこう）がある: uncomfortable
捕まりやすい（つかまりやすい）魚（さかな）: fish that we can easily capture

バグ数推定法②
2段階エディット法(1／3)

- 人工バグを使わないかたちに捕獲・再捕獲法を改良

1. 2つのテストグループ A, B を用意
2. グループ A がテストを行い，見つかったバグを記録(B には秘密にする)
3. グループ B がテストを行い，見つかったバグの中で何個が A と同じだったのかによって残っているバグ数を推定

2段階エディット法(にだんかいエディットほう): two-stage edit procedure

改良(かいりょう): improve

秘密(ひみつ)にする : hide

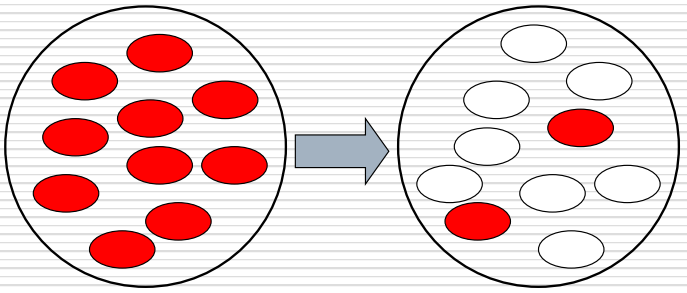
何個(なんこ)が A と同じだったのか : how many bugs were common to the ones detected by A

残っているバグ数(のこっているバグ数): number of remained bugs, number of bugs that have not been detected yet

バグ数推定法②
2段階エディット法(2／3)

- 捕獲・再捕獲法と同様に考える

グループA
が 10 件
見つけた



グループB
で見つけた
10 件中 2 件
に A で見つけ
たものが含ま
れていた

全体における割合の推定

＝ 2：8

10：？ ＝ 2：8
なのでAで見つかって
いないバグは 40 件

バグ数推定法②
2段階エディット法(3／3)

- グループ A が見つけたバグ数 ＝ a
- グループ B が見つけたバグ数 ＝ b
- 両方で見つけたバグ数 ＝ c

$a : ? = c : (b - c)$ より
A が見つけていないバグ数 = $\frac{a(b-c)}{c}$

(見つけたものも含めた)総バグ数は

$$\frac{a(b-c)}{c} + a = \frac{ab}{c} - \frac{ac}{c} + a = \boxed{\frac{ab}{c}}$$

2段階エディット法の特徴

- **人工バグを用意する必要がない**
  （本物のバグだけで推定ができる）
- 2チームが別々にテストする必要があるので，
  それだけ**時間やコストがかかる**
- **実際よりも推定値が小さい**傾向にある

2段階エディット法の特徴（とくちょう）: characteristics of two-stage edit procedure

人工（じんこう）**バグを用意**（よう い）**する必要**（ひつよう）**がない**: you don’t need to prepare artificial bugs

2チームが別々（べつべつ）にテストする: two teams test it independently

推定値（すいていち）: estimation

小さい傾向（けいこう）: tends to be a smaller value

【演習1】

① バグ総数の推定値を求めよ

グループ A が 10 個のバグを見つけ，
グループ B が 50 個のバグを見つけた．
ただし，5 個は両方のグループで見つかった．

② （まだ見つかっていない）残存バグ数の推定値を求めよ

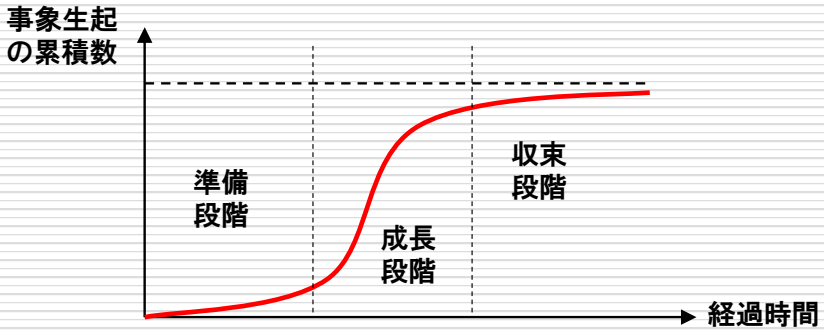
グループ A が 100 個のバグを見つけ，
グループ B が 80 個のバグを見つけた．
ただし，20 個は両方のグループで見つかった．

まだ見つかっていない：have not been detected yet

バグ数推定法③
ゴンペルツ(Gompertz)曲線モデル(1／4)

□ ゴンペルツ曲線モデルとは

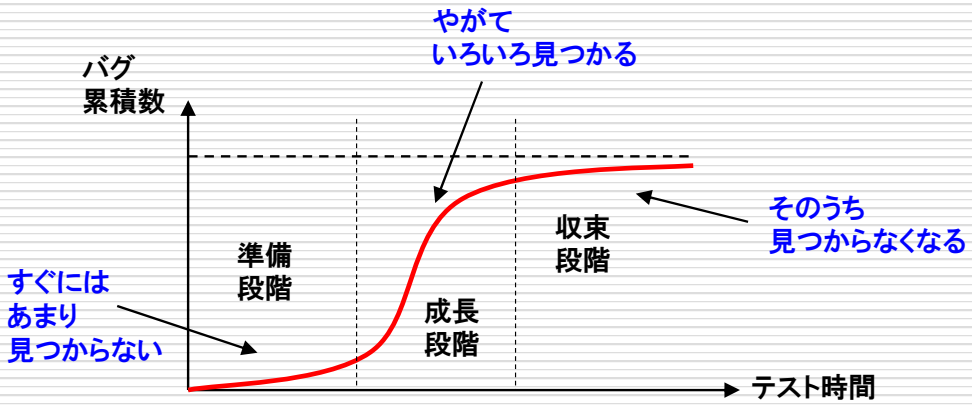
成長曲線モデルの一種で，さまざまな事象の始まりから成長，収束へ至る過程を表す



成長曲線(せいちょうきょくせん): growth curve
事象(じしょう): event
収束(しゅうそく): convergence
事象生起(じしょうせいき): event occurrence
累積数(るいせきすう): accumulated count
経過時間(けいかじかん): elapsed time

バグ数推定法③
ゴンペルツ(Gompertz)曲線モデル(2／4)

- バグ累積数の成長モデルに用いられる



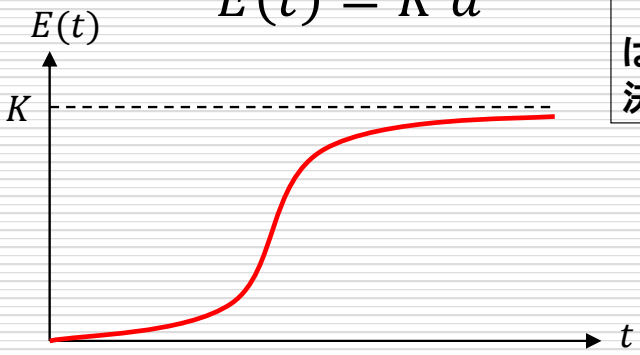
バグ累積数(バグるいせきすう): accumulated number of detected bugs

バグ数推定法③
ゴンペルツ(Gompertz)曲線モデル(3／4)

□ 数学的には

$$E(t) = K\,a^{b^t}$$

$a, b\,(<1), K$
は曲線の形状を決めるパラメータ

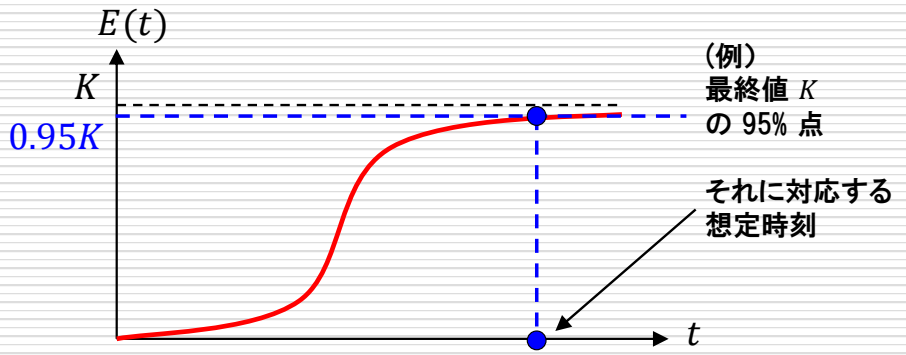


※この曲線は，経験的な傾向を示すものである

経験的(けいけんてき)な傾向(けいこう): empirical tendency

バグ数推定法③
ゴンペルツ(Gompertz)曲線モデル(4／4)

- 特定時点での信頼度や目標の信頼度を得るまでにかかるテスト時間を推定(予測)できる



信頼度(しんらいど): reliability
予測(よそく)する: predict
想定時刻(そうていじこく): expected time

バグ数推定法③
ソフトウェア信頼度成長モデル

バグ検出を確率的に表現した数理モデル

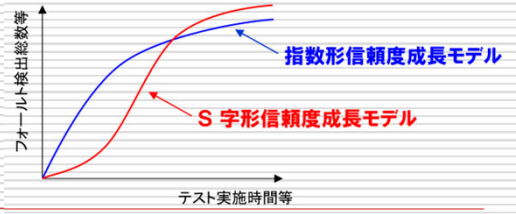
後で詳しく説明する

- 指数形ソフトウェア信頼度成長モデル

$$E(t) = a(1 - e^{-bt})$$

- 遅延S字形ソフトウェア信頼度成長モデル

$$E(t) = a\{1 - (1 + bt)e^{-bt}\}$$



ソフトウェア信頼度成長モデル（ソフトウェアしんらいどせいちょうモデル）: software reliability growth model (SRGM)

確率的（かくりつてき）: stochastic

指数形（しすうがた）: exponential

遅延 S 字形（ちえん S じがた）: delayed S-shaped

バグ数推定法④
過去の統計データによる方法

- **過去のプロジェクトでの実績値**を用いる方法
- 例えば過去のバグ密度が 3.8 [件/KLOC] とし，今回の開発規模が 100 [KLOC] ならば，単純に 3.8 × 100 = 380 件のバグがあると推定

※ LOC = Lines Of Code：コード行数(コメント，空行を除く)
KLOC = 1000 LOC

- 単純な方法であるが，予測精度は悪くない

過去の(かこの): past
統計データ(とうけいデータ): statistic data
実績値(じっせきち): actual value
バグ密度(バグみつど): bug density
規模(きぼ): size
空行(くうぎょう): blank line

【演習2】残存バグ数を予測せよ

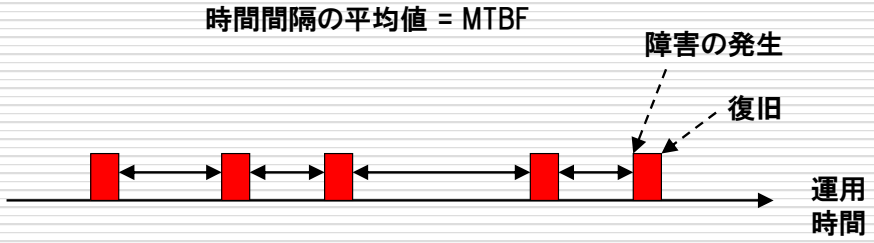
- あるプロジェクトの成果物に対してテストを実施したところ，**全部で 38 件のバグ**が見つかった．
- 過去の同様のプロジェクトでの**バグ密度は 2.5 [件/KLOC]** であった．
- 今回の**開発規模は 20 [KLOC]** である．

あと**どれだけのバグが未検出で残っている**と予測されるか？

成果物（せいかぶつ）: product

テストの評価(3)
運用性に基づく評価

- システムテスト(運用を模倣)において
  障害(不具合)が起こる時間間隔の平均値
  MTBF (Mean Time Between Failure)
  を利用する



運用性(うんようせい): operability
模倣(もほう): simulate
障害(しょうがい), 不具合(ふぐあい): failure
時間間隔(じかんかんかく): time interval
平均値(へいきんち): mean, average
復旧(ふっきゅう): recover

MTBF による評価

- MTBF の長さが信頼性の高さを意味する

- ただし，システムテストの環境が実運用環境に近くなければ意味は無い

  適切な確率モデルに従ったランダムテストが重要である

  （言うまでもなく人間によるテストも重要）

実運用環境（じつうんようかんきょう）: real operational environment
適切な（てきせつな）: proper, appropriate
確率モデル（かくりつモデル）: stochastic model

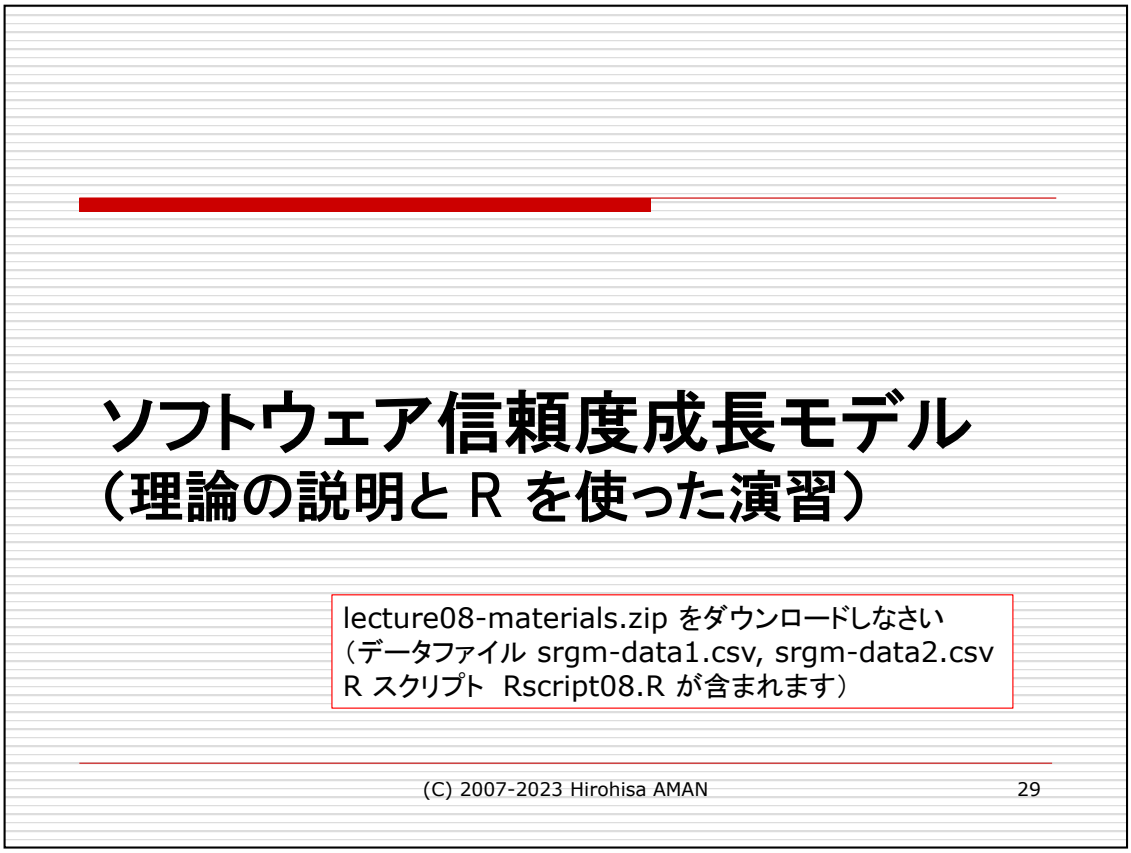
(ここまでの)まとめ

- **テストの評価**: 網羅性, **バグ除去率**
  バグ数推定→**2段階エディット法**, **成長曲線**, **バグ密度**

- 運用性の評価: MTBF

10-minutes rest break

10分休憩



ソフトウェア信頼度成長モデル（ソフトウェアしんらいどせいちょうモデル）: software reliability growth model (SRGM)

理論の説明（りろんのせつめい）: explanation of the theory

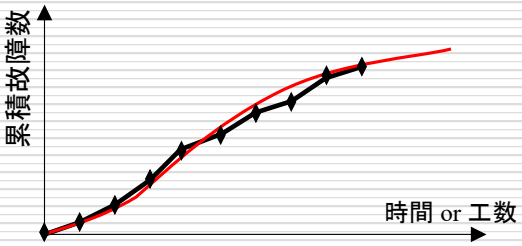
Download lecture08-materials.zip.

It contains data files srgm-data1.csv and srgm-data2.csv, and R script Rscrip08.R.

成長曲線(growth curve)

- 時間の経過や工数の投入に伴って**対象データが変化していく様子をモデル化した曲線**
  - **累積値** (cumulative number) を用い，**その成長**をとらえる
  - モデルを用いた**見積り** (estimation) も行われる

故障検出数をモデル化した**ソフトウェア信頼度成長モデル**(software reliability growth model)が有名

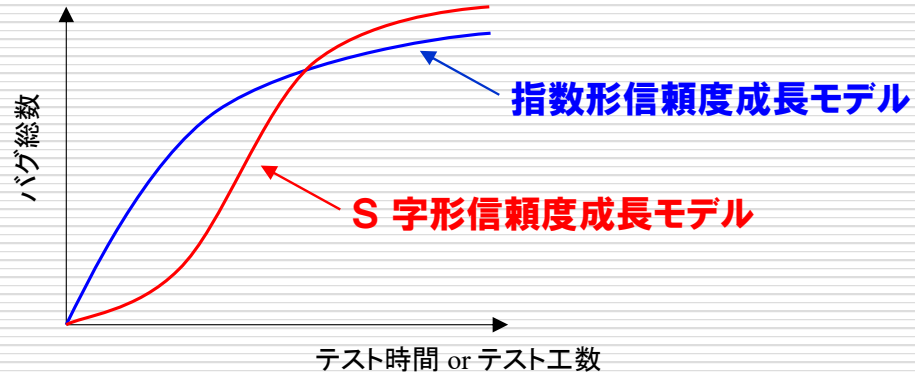


時間の経過(じかんのけいか): lapse of time
工数の投入(こうすうのとうにゅう): take man-hours
故障検出数(こしょうけんしゅつすう): number of detected failures

ソフトウェア信頼度成長モデル
(Software Reliability Growth Model: SRGM)

- **テスト時間**(または**テスト工数**)と**バグ**(**故障**)**総数**の関係をモデル化したもの

※ここでは「フォールト」を「バグ」と呼んでいる

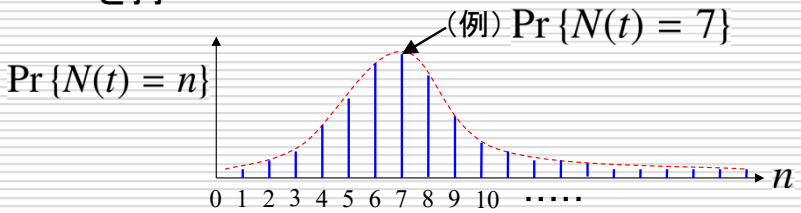


テスト時間(じかん): testing time
テスト工数(こうすう): testing effort
バグ総数(そうすう): total number of bugs
故障総数(こしょうそうすう): total number of failures
指数形(しすうがた): exponential
遅延 S 字形(ちえん S じがた): delayed S-shaped

バグ数の確率過程モデル

- ここではテスト時間とバグ総数の関係をモデル化する
- 時刻 t での**バグ総数**を**確率変数** $N(t)$で表す
  （時間に沿った確率変数の系列を考える）

  時刻 t を固定したとき，$N(t)$ は**何らかの確率分布**を持つ



確率過程（かくりつかてい）: stochastic process
検出済みバグ総数（けんしゅつずみバグそうすう）: total number of detected bugs
確率変数（かくりつへんすう）: random variable

※田栗ほか：やさしい統計入門，講談社（2007）

（参考）二項分布とポアソン分布（1／3）

□ 二項分布：ある事象が確率 p で起こるとして，n 回の試行の中で実際に k 回起こる確率は

$$\binom{n}{k} p^k (1-p)^{n-k}$$

例）ある街で1分間に交通事故が起こる確率 $p = \frac{1}{1440}$

（※1日＝1440 分；1日に1回と想定）

これから 24 時間以内に交通事故が**起こらない**確率

$$\binom{1440}{0} p^0 (1-p)^{1440-0} = (1-p)^{1440} = \left(\frac{1439}{1440}\right)^{1440}$$

二項分布（にこうぶんぷ）: binominal distribution
ポアソン分布（ポアソンぶんぷ）: Poisson distribution
事象（じしょう）: event
起こる（おこる）: occur
試行（しこう）: trial
交通事故（こうつうじこ）: traffic accident

（参考）二項分布とポアソン分布(2／3)

- 実際に計算すると

$$\left(\frac{1439}{1440}\right)^{1440} = 0.3677517$$

- n が大きく p が小さいときは，ポアソン分布で近似できる：

$$P(X=k)\frac{(np)^k e^{-np}}{k!} \quad \left(n=1440, p=\frac{1}{1440}, k=0\right)$$

$$= e^{-1} = 0.3678794$$

近似（きんじ）: approximate

（参考）二項分布とポアソン分布(3／3)

- 近似式は（$\lambda = np$ として）

$$P(X = k) = \frac{(np)^k e^{-np}}{k!} = \frac{\lambda^k e^{-\lambda}}{k!}$$

- これが一般に**ポアソン分布**として知られている
- 期待値（平均），分散ともに $\lambda = np$

期待値（きたいち）: expected value
平均（へいきん）: mean
分散（ぶんさん）: variance

ポアソン過程(Poisson Process)

- 単位時間あたり λ 個の割合でバグが見つかるとすると $(\lambda > 0)$

  時刻 t での総バグ数の**期待値**は λt で，

  その確率分布はポアソン分布に従う

$$\Pr\{ N(t) = n \} = \frac{(\lambda t)^n}{n!} e^{-\lambda t} \quad (n = 0,1,2, \ldots)$$

つまり，このような(ポアソン分布に従った)確率変数の系列が形成される → ポアソン過程

単位時間(たんいじかん): unit time

期待値(きたいち): expectation

確率分布(かくりつぶんぷ): probabilistic distribution

確率変数の系列(かくりつへんすうのけいれつ): series of random variables

バグ数の期待値について(1/3)

- 実際，単位時間あたりに見つかるバグの数は一定ではない(仮にそうだとするとバグは無限に存在することになる: $t \to \infty$ のとき，期待値 $\lambda t \to \infty$)
- そこで，時刻 t での期待値を $H(t)$ とおいて

$$\Pr\{ N(t) = n \} = \frac{H(t)^n}{n!} e^{-H(t)}$$

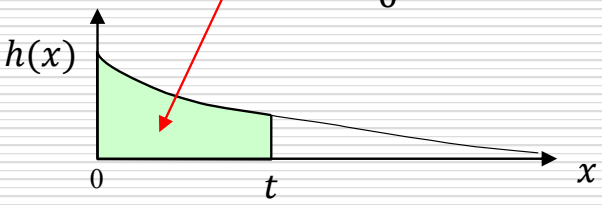
$$(n = 0,1,2, \ldots)$$

一定(いってい)ではない: not constant
無限(むげん)に: infinitely
期待値(きたいち): expected value

バグ数の期待値について(2/3)

- 時刻 t におけるバグ数の期待値 $H(t)$ は，それまでの各時刻におけるバグ検出率 $h(t)$ から得られる

$$H(t) = \int_0^t h(x)dx$$



例えば，$h(3)$ は時刻 $t = 3$ におけるバグ検出率であり，その時点では単位時間あたり $h(3)$個のバグを検出する見込み

検出率(けんしゅつりつ): detection rate

バグ数の期待値について(3/3)

- 当然ながら，バグ検出率を常に一定とするならば $h(t) = \lambda$（定数）

$$H(t) = \int_0^t h(x)dx = \int_0^t \lambda\, dx = \lambda t$$

$$\Pr\{ N(t) = n \} = \frac{H(t)^n}{n!} e^{-H(t)}$$

$$= \frac{(\lambda t)^n}{n!} e^{-\lambda t}$$

定数（ていすう）: constant

代表的なバグ検出数モデル：非同次ポアソン過程（NHPP）モデル

- バグ検出率を時間関数 $h(t)$ に一般化
- 期待値を λt ではなく $H(t)$ として考えたポアソン過程を**非同次ポアソン過程**（NonHomogeneous Poison process: **NHPP**）という

$$\Pr\{N(t) = n\} = \frac{H(t)^n}{n!} e^{-H(t)} \quad (n = 0,1,2,\ldots)$$

$$H(t) = \int_0^t h(x)dx$$

一般化（いっぱんか）: generalization

(用語)NHPPの平均値関数と強度関数

$$\Pr\{N(t)=n\}=\frac{H(t)^n}{n!}e^{-H(t)} \qquad (n=0,1,2,\ldots)$$

$$H(t)=\int_0^t h(x)dx$$

$H(t)$: **平均値関数**
時刻 t での**総バグ数** $N(t)$ の**期待値**

$h(t)$: **強度関数**
時刻 t での**バグ検出率**　※確率ではない
単位時間でどれだけのフォールトが見つかりそうか，その時点での見込み

平均値関数(へいきんちかんすう) : mean value function
強度関数(きょうどかんすう) : intensity function

(参考)同次ポアソン過程
(homogeneous Poisson process)

- 強度関数が時間に関係なく一定の場合:

$$h(t) = \lambda \qquad (\lambda > 0)$$

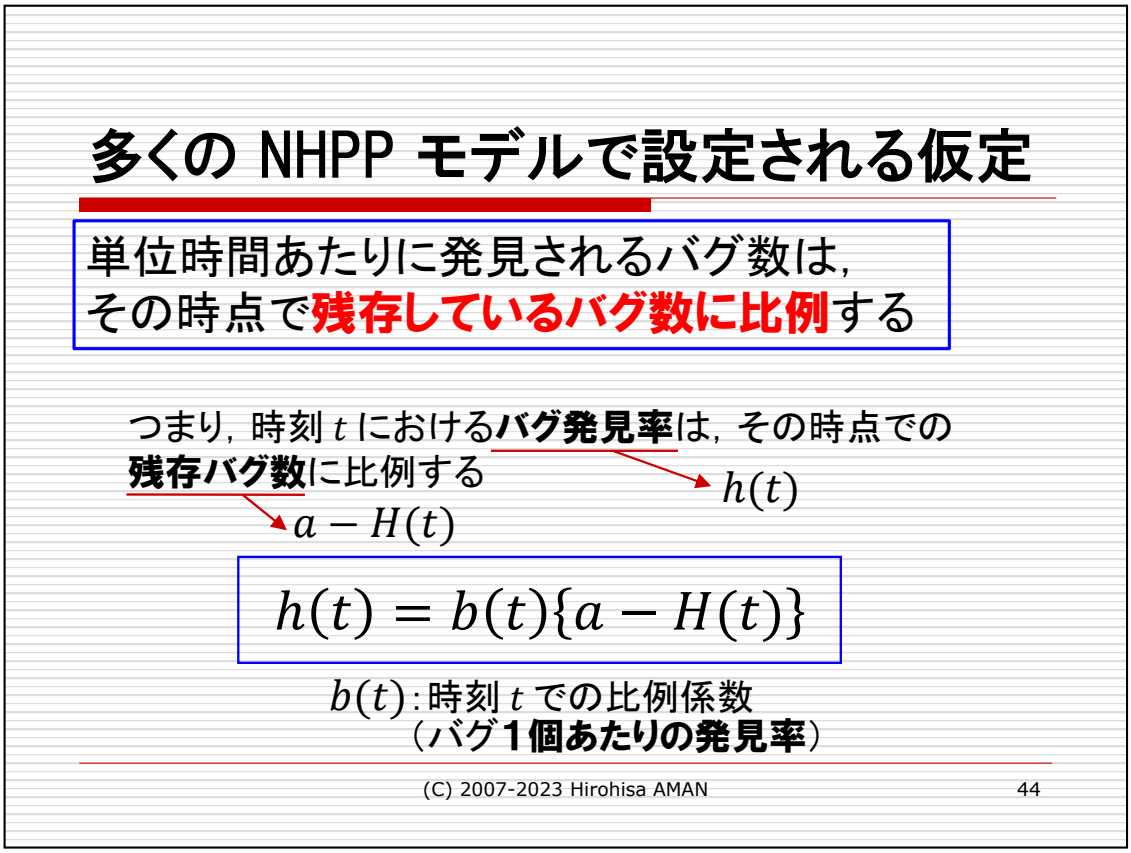
- このとき，確率変数 $N(t)$ は同次ポアソン過程(HPP)に従うという

代表的な NHPP モデル

- 実際に信頼度の評価・予測を行う場合，**何らかの仮定の下**で**平均値関数を特定**する必要がある
- 代表例

モデル名	平均値関数 $H(t)$
指数形	$a(1-e^{-bt})$
遅延S字形	$a\{1-(1+bt)e^{-bt}\}$

何らかの仮定（なんらかのかてい）の下で: under an assumption



多くの　NHPP モデルで設定（せってい）されている仮定（かてい）: the assumption on which many NHPP models base

単位時間（たんいじかん）あたりに発見（はっけん）されるバグ数: the number of bugs detected per a unit time

残存しているバグ数（ざんぞんしているバグすう）に比例する（ひれいする）: be proportional to the number of remained (undetected) bugs

バグ発見率（はっけんりつ）: bug detection rate

比例係数（ひれいけいすう）: proportionality coefficient

多くの NHPP モデルで設定される仮定

- 改めて，平均値関数 $H(t)$ を使って書くと

$$\frac{dH(t)}{dt} = b(t)\{a - H(t)\}$$

$$H(t) = \int_0^t h(x)dx$$

- この方程式を初期条件 $H(0) = 0$ で解くと

$$H(t) = a(1 - e^{-\int_0^t b(x)\,dx})$$

方程式（ほうていしき）: equation
初期条件（しょきじょうけん）: initial condition

①指数形ソフトウェア信頼度成長モデル (Exponential SRGM)

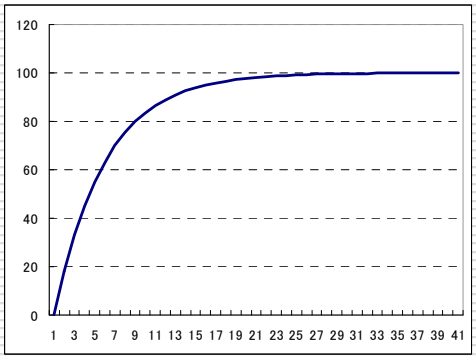
- 比例係数に用いる $b(t)$ が時刻によらず一定であると仮定

$$H(t) = a(1 - e^{-\int_0^t b(x)\,dx})$$

$$b(t) = b$$

$$H(t) = a(1 - e^{-bt})$$

（例）$a = 100, b = 0.2$



実際に利用する場合

- 当然ながら，曲線パラメータ a, b は不明であるため，何らかの方法で推定する必要がある
- 現実には（その時点で入手できている）**実データに曲線が当てはまるようパラメータを推定**する（**最尤法**や**最小二乗法**が用いられる）

安定した推定値が得られるのは，
テストが50～60%程度完了した後といわれている

実データ（じつデータ）: actual data
当てはまる（あてはまる）: fit
最尤法（さいゆうほう）: maximum likelihood method
最小二乗法（さいしょうにじょうほう）: least-square method

【R演習】数値例

- リアルタイム指令制御プログラム（217 KLOC）のフォールト検出データ [Goel(1985)]
  - srgm-data1.csv

- これをデータフレーム d1 として読み込む

```
d1 = read.csv( file.choose( ) )
```

[Goel(1985)] Goel, A.L. : Software reliability models: Assumptions, limitations, and applicability, IEEE Trans. Software Eng., vol.SE-11, no.12, pp.1411-1423, Dec. 1985.

Open Rscript08.R on your PC.

Run “d1 = read.csv(file.choose( ))” to read srgm-data1.csv.

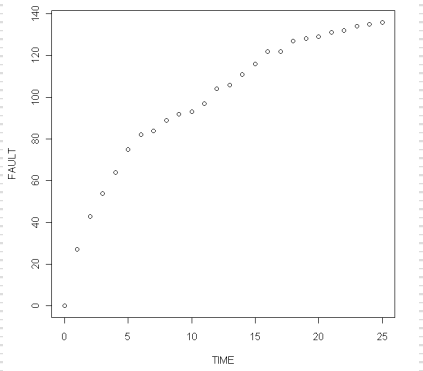
【R演習】数値例:CSV データの内容

□ 経過時間と累積バグ数

```
> d1
   TIME  FAULT
1     0      0
2     1     27
3     2     43
・・・・・・・・・・・・・・・
```

時間はテスト実施に費やされた CPU 時間(単位:hour)

```
plot(FAULT ~ TIME, data=d1)
```



経過時間(けいかじかん): elapsed time

累積バグ数(るいせきバグすう):accumulated number of bugs

成長曲線当てはめの例

□ 例えば**指数形成長モデル**(exponential growth model)を使った場合

$$m(t) = a(1 - e^{-bt})$$

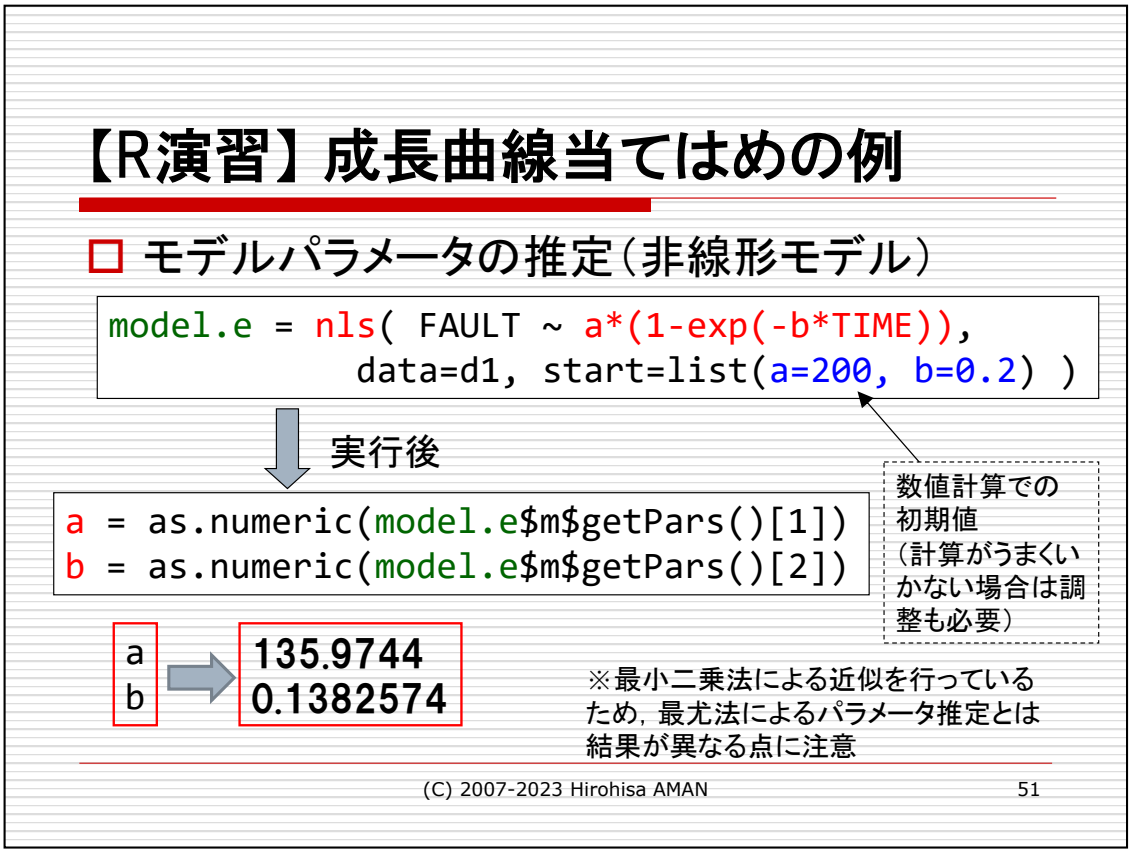
$m(t)$:平均値関数
(時刻 t における検出バグ数の期待値)

a :バグ総数の期待値($t \to \infty$)

b :バグ検出率

※これは,一定の発見率(b)でもって
バグが検出されていくというモデル

当てはめ(あてはめ): fitting
期待値(きたいち): expectation
検出率(けんしゅつりつ): detection rate



推定（すいてい）: estimation
非線形（ひせんけい）: nonlinear
数値計算（すうちけいさん）: numerical computation
初期値（しょきち）: initial value
最小二乗法（さいしょうにじょうほう）: least square method
最尤法（さいゆうほう）: maximum likelihood method

【R演習】
成長曲線を関数として用意しておく

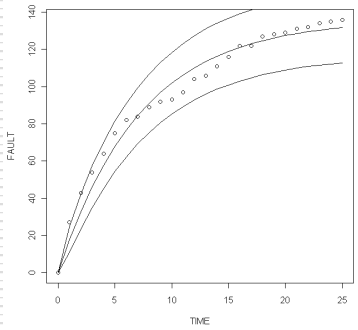
- 指数形成長モデルのパラメータを用意し，
  平均値関数を関数 m として用意する

```
a.e = as.numeric(model.e$m$getPars()[1])
b.e = as.numeric(model.e$m$getPars()[2])
m = function(t){
  a.e*(1-exp(-b.e*t))
}
```

$$m(t) = a(1 - e^{-bt})$$

【R演習】成長曲線当てはめの例（90%信頼区間）

```
plot(FAULT ~ TIME, data=d1)
lines( m(TIME) ~ TIME, data=d1, col="red")
lines( m(TIME)+1.645*sqrt(m(TIME)) ~ TIME, data=d1)
lines( m(TIME)-1.645*sqrt(m(TIME)) ~ TIME, data=d1)
```




```
a.e+1.645*sqrt(a.e)
a.e-1.645*sqrt(a.e)
```

→ **155.1564**
116.7923

バグ総数は 117 ～ 155 個と予測される

$t = 25$ で 136 個であるため
19個ほどがまだ見つかっていない
可能性がある（信頼度90%）

信頼区間（しんらいくかん）: confidence interval

②遅延S字形ソフトウェア信頼度成長モデル(Delayed S-shaped SRGM)(1/4)

- 複雑なソフトウェアの場合，バグの検出は

不具合の発見
→ 原因の調査
→ バグ(フォールト)の検出

というステップになる

- ゆえに，時刻 t で**期待される不具合数とバグ数の間には差**があり，**その差がバグ検出率に影響する**という解釈ができる

期待される不具合数(きたいされるふぐあいすう): expected number of failures
期待されるバグ数(きたいされるバグ数): expected number of bugs (faults)
差(さ): difference
影響する(えいきょうする): influence

②遅延S字形ソフトウェア信頼度成長モデル (2/4)

- いま，**不具合の発見は指数形モデルで表現されると仮定**し，その平均値関数を以下とする：

$$m(t) = a(1 - e^{-bt})$$

- 時刻 t におけるバグ検出率 $h(t)$ は，不具合発見数 $m(t)$ とバグ検出数 $H(t)$ の**差に比例**すると考える：

$$h(t) = b\{ m(t) - H(t) \}$$

差に比例する（さにひれいする）: be proportional to the difference

②遅延S字形ソフトウェア信頼度成長モデル（3/4）

- つまり，

$$\begin{cases} h(t) = \dfrac{dH(t)}{dt} = b\{ m(t) - H(t) \} \\ m(t) = a(1 - e^{-bt}) \end{cases}$$

- この微分方程式を解くと

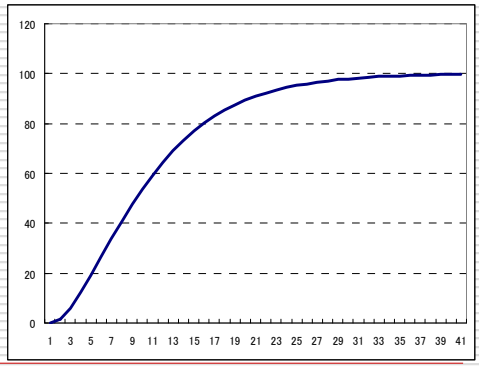
$$H(t) = a\{ 1 - (1 + bt)e^{-bt} \}$$

微分方程式（びぶんほうていしき）: differential equation

②遅延S字形ソフトウェア信頼度成長モデル (4/4)

$$H(t) = a\{1 - (1 + bt)e^{-bt}\}$$

（例）$a = 100, b = 0.2$



- 直感的には，指数形モデルが2つ入ることで成長に「遅れ」が生じる

【R演習】数値例

- オンラインデータ入力ソフトウェアパッケージ(40 KLOC)のバグ検出データ [Ohba(1984)]
  - srgm-data2.csv

- これをデータフレーム d2 として読み込む

```
d2 = read.csv( file.choose( ) )
```

[Ohba(1984)] M. Ohba : Software reliability analysis models,
IBM J. Research and Development, vol.28, no.4, pp.428-443, July 1984.

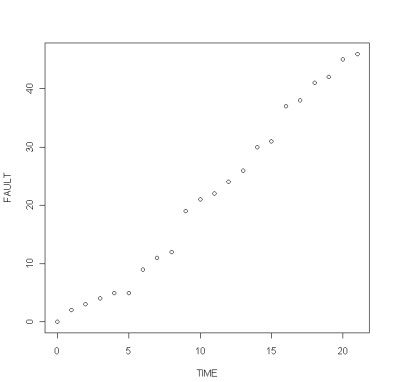
【R演習】数値例：CSV データの内容

□ 観測時間と累積バグ数

```
> d2
   TIME  FAULT
1     0      0
2     1      2
3     2      3
・・・・・・・・・・・・・・・
```

時間はテストに費やされた日数

```
plot (FAULT ~ TIME, data=d2)
```



【R演習】遅延S字形曲線の当てはめ

□ モデルパラメータの推定（非線形モデル）

```
model.d = nls(FAULT ~ a*(1-(1+b*TIME)*exp(-b*TIME)),
              data=d2, start=list(a=200,b=0.2) )
```

数値計算での初期値（計算がうまくいかない場合は調整も必要）

実行後

```
a = as.numeric(model.d$m$getPars()[1])
b = as.numeric(model.d$m$getPars()[2])
```

a
b

78.69114
0.09493751

【R演習】
成長曲線を関数として用意しておく

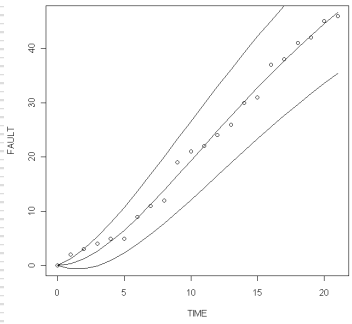
- 遅延S字形長曲線のパタメータを用意し，
  平均値関数を H として用意する

```
a.d = as.numeric(model.d$m$getPars()[1])
b.d = as.numeric(model.d$m$getPars()[2])
H = function(t){
  a.d*(1-(1+b.d*t)*exp(-b.d*t))
}
```

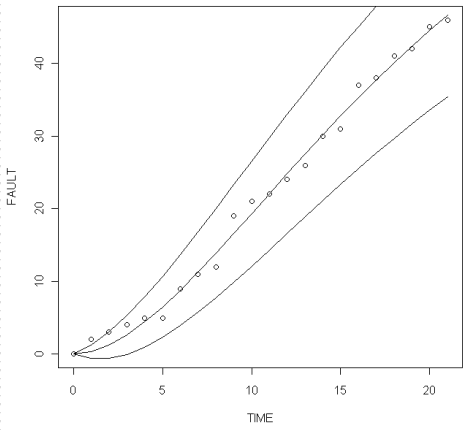
$$H(t) = a\{1 - (1+bt)e^{-bt}\}$$

【R演習】成長曲線当てはめの例（90%信頼区間）（1／2）

```
plot(FAULT ~ TIME, data=d2)
lines( H(TIME) ~ TIME, data=d2, col="red")
lines( H(TIME)+1.645*sqrt(H(TIME)) ~ TIME, data=d2)
lines( H(TIME)-1.645*sqrt(H(TIME)) ~ TIME, data=d2)
```



【R演習】成長曲線当てはめの例（90%信頼区間）（2／2）




```
a.d + 1.645*sqrt(a.d)
a.d - 1.645*sqrt(a.d)
```

93.28361
64.09867

バグ総数は 64 ～ 93 個と予測される

$t = 20$ で 46 個であるためまだ18～47個が見つかっていない可能性がある（信頼度90%）

宿題(homework)

“[08] quiz”に答えなさい (明日の 13 時まで)

Answer “[08] quiz”
by tomorrow 13:00 (1pm)

注意:quiz のスコアは final project の成績の一部となります
(Note: Your quiz score will be a part of your final project evaluation)

【演習1】解答

① $a = 10,\ b = 50,\ c = 5$ より

バグ総数 $= a \cdot b/c = 10 \times 50/5 =$ **100**

② $a = 100,\ b = 80,\ c = 20$ より

バグ総数 $= a \cdot b/c = 100 \times 80/20 = 400$

したがって,

残存バグ数 $= 400 - a - b + c$

$= 400 - 100 - 80 + 20 =$ **240**

両方で見つかったものが二重に引かれるため

【演習2】解答

- まず，過去の実績と今回の開発規模から

  $2.5 \times 20 = 50$

  より，**全部で 50 件のバグ**があると予測.

- 現時点で 38 件（76%=38/50）のバグを見つけているので，**残りは 12 件**（24%）である.